ALIBABA CLOUD

Alibaba Cloud

E-MapReduce

Best Practices

Document Version: 20220712

Alibaba Cloud

Legal disclaimer

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

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

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

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

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

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

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

Document conventions

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

Table of Contents

1.Data analysis

1.1. Adaptive execution framework of Spark SQL

Spark SQL in Alibaba Cloud E-MapReduce (EMR) V3.13.0 and later provides an adaptive execution framework. This framework can be used to dynamically adjust the number of reduce tasks, handle data skew, and optimize execution plans.

Limits

In this topic, the parameters used in the adaptive execution framework of Spark SQL are suitable only for Spark 2.X. If you use Spark 3.X, you can view the related parameters in Adaptive Query Execution.

Capabilities

The adaptive execution framework of Spark SQL has the following capabilities:

- Dynamically adjusts the number of reduce tasks.

  The number of tasks in a reduce stage in Spark SQL depends on the value of the spark.sql.shuffle.partition parameter. The default value of this parameter is 200. If you configure this parameter for a job, the number of tasks remains the same in all reduce stages of the job.

  However, for different jobs or for different reduce stages of the same job, the data to be processed may significantly vary in size. The spark.sql.shuffle.partition parameter may have a negative impact on processing efficiency. For example, if you want to process only 10 MB of data, one reduce task is sufficient. If the spark.sql.shuffle.partition parameter is set to the default value 200, the 10 MB of data is split into 200 pieces and is processed by 200 tasks. This significantly increases scheduling overheads and reduces processing efficiency.

  The adaptive execution framework of Spark SQL allows you to configure the upper and lower limits of the number of shuffle partitions. The framework can dynamically adjust the number of tasks in the reduce stages of different jobs within the specified range.

  This way, you no longer need to set the spark.sql.shuffle.partition parameter, and the optimization costs are significantly reduced. The framework can also dynamically adjust the number of tasks in different reduce stages of the same job.

  The following table describes the related parameters.

Parameter	Default value	Description
spark.sql.adaptive.enabled	false	Specifies whether to enable the adaptive execution framework of Spark SQL.
spark.sql.adaptive.minNumPostShufflePartitions	1	The minimum number of reduce tasks.
spark.sql.adaptive.maxNumPostShufflePartitions	500	The maximum number of reduce tasks.

Parameter	Default value	Description
spark.sql.adaptive.shuffle.targetPostShuffleInputSize	67108864	The minimum size of data that each reduce task must process. Unit: bytes. For example, if you use the default value 67108864, each reduce task processes a minimum of 64 MB of data. The system dynamically adjusts the number of reduce tasks based on this parameter.
spark.sql.adaptive.shuffle.targetPostShuffleRowCount	20000000	The minimum number of records that each reduce task must process. For example, if you use the default value 20000000, each reduce task processes a minimum of 20,000,000 records. The system dynamically adjusts the number of reduce tasks based on this parameter.

- Automatically handles data skew.

  Data skew is a common issue in JOIN operations. When data skew occurs, some tasks need to process excessively large amounts of data, which leads to long tails. Apache Spark SQL cannot handle data skew.

  In EMR V3.13.0 and later, when a job is running, the adaptive execution framework of Spark SQL can automatically detect skewed data and handle the issue based on the settings of the related parameters.

  The adaptive execution framework of Spark SQL splits the data that is in the skewed partition, processes the data by using multiple tasks, and then combines the results by performing SQL UNION operations.

  The following table describes how the adaptive execution framework of Spark SQL handles data skew in different JOIN operations.

JOIN operation	Description
Inner	Skewed data can be handled in both tables.
Cross	Skewed data can be handled in both tables.
LeftSemi	Skewed data can be handled only in the left table.
LeftAnti	Skewed data can be handled only in the left table.
LeftOuter	Skewed data can be handled only in the left table.
RightOuter	Skewed data can be handled only in the right table.

  The following table describes the related parameters.

Parameter	Default value	Description
spark.sql.adaptive.enabled	false	Specifies whether to enable the adaptive execution framework of Spark SQL.
spark.sql.adaptive.skewedJoin.enabled	false	Specifies whether to enable data skew handling.
spark.sql.adaptive.skewedPartitionFactor	10	The median size of all partitions or the median number of records in all partitions. If you use this parameter to specify the median size of all partitions, a partition is identified as a skewed partition only if the size of the partition is greater than both the value of this parameter and the value of the spark.sql.adaptive.skewedPartitionSizeThreshold parameter. If you use this parameter to specify the median number of records in all partitions, a partition is identified as a skewed partition only if the number of records in the partition is greater than both the value of this parameter and the value of the spark.sql.adaptive.skewedPartitionRowCountThreshold parameter.
spark.sql.adaptive.skewedPartitionSizeThreshold	67108864	The size threshold of a partition. This parameter is used to determine whether a partition is skewed.
spark.sql.adaptive.skewedPartitionRowCountThreshold	10000000	The threshold of the number of records in a partition. This parameter is used to determine whether a partition is skewed.
spark.shuffle.statistics.verbose	false	If you set this parameter to true, MapStatus collects information about the number of records in each partition. This parameter is used to handle data skew.

- Dynamically optimizes execution plans.

The Catalyst optimizer of Spark SQL converts SQL statements into physical execution plans and runs the physical execution plans. Due to the lack or inaccuracy of statistics, a physical execution plan that is produced by Catalyst may not be optimal. For example, when a broadcast join is optimal, Spark SQL may use a sort-merge join based on conversion results.

When Spark SQL runs a physical execution plan, the adaptive execution framework of Spark SQL dynamically determines whether to use a broadcast join instead of a sort-merge join based on the data size of the shuffle write in the shuffle stage. This way, the query efficiency is improved.

The following table describes the related parameters.

Parameter	Default value	Description
spark.sql.adaptive.enabled	false	Specifies whether to enable the adaptive execution framework of Spark SQL.
spark.sql.adaptive.join.enabled	true	Specifies whether to enable the dynamic optimization of execution plans.
spark.sql.adaptiveBroadcastJoinThreshold	Value of spark.sql.autoBroadcastJoinThreshold	A condition that is used to determine whether to use a broadcast join.

2.Data migration and synchronization

2.1. Use Presto to query table data in an ApsaraDB RDS for MySQL database

This topic describes how to use the Presto service of an E-MapReduce (EMR) cluster to query table data in an ApsaraDB RDS for MySQL database. To use this feature, make sure that the EMR cluster and ApsaraDB RDS for MySQL database are created in the same virtual private cloud (VPC).

Prerequisites

- An EMR cluster is created, and Presto is selected from the optional services when you create the cluster. For more information, see Create a cluster.
- An ApsaraDB RDS for MySQL instance is purchased. For more information, see Create an ApsaraDB RDS for MySQL instance.

> **Note** We recommend that you select 5.7 from the **MySQL** drop-down list for **Database Engine** and set **Edition** to **High-availability** when you create an ApsaraDB RDS for MySQL instance.

Context

For more information about Presto terms, see Overview.

Step 1: Configure a connector

1. Go to the Presto service page.
   i. Log on to the Alibaba Cloud EMR console.
   ii. In the top navigation bar, select the region where your cluster resides and select a resource group based on your business requirements.
   iii. Click the **Cluster Management** tab.
   iv. On the **Cluster Management** page, find your cluster and click **Details** in the Actions column.
   v. In the left-side navigation pane, choose **Cluster Service > Presto**.
2. Configure connector-related parameters.
   i. On the Presto service page, click the **Configure** tab.
   ii. In the **Service Configuration** section, click the **connector1.properties** tab.

   To connect to multiple ApsaraDB RDS for MySQL databases, you can also configure parameters on the connector2.properties and connector3.properties tabs.
   iii. Set **connector.name** to **mysql**.
   iv. In the upper-right corner of the Service Configuration section, click **Custom Configuration**.

v. In the **Add Configuration Item** dialog box, add the parameters that are described in the following table.

Parameter	Description
connection-user	The username that is used to access the database. In this example, the username is hiveuser.
connection-password	The password that is used to access the database.
connection-url	The URL of the database. For more information, see View and change the internal and public endpoints and port numbers of an ApsaraDB RDS for MySQL instance. Example: *jdbc:mysql://rm-2ze5ipacsu8265qxxxxxxxx.mysql.rds.aliyuncs.com:3306*.

3. Save the configurations.
   i. In the upper-right corner of the Service Configuration section, click **Save**.
   ii. In the **Confirm Changes** dialog box, configure **Description** and turn on **Auto-update Configuration**.
   iii. Click **OK**.
4. Restart the service for the configurations to take effect.
   i. In the upper-right corner of the Presto service page, choose **Actions > Restart All Components**.
   ii. In the **Cluster Activities** dialog box, configure **Description**.
   iii. Click **OK**.
   iv. In the **Confirm** message, click **OK**.

Step 2: View information about the ApsaraDB RDS for MySQL database

1. Log on to the master node of the EMR cluster in SSH mode.

   For more information, see Log on to a cluster.
2. Run the following command to connect to the Presto client:

   ```
   presto --server emr-header-1:9090 --catalog hive --schema default --user hadoop
   ```

   If the following information is returned, the Presto client is connected:

   ```
   presto:default>
   ```
3. Run the following command to view the schema from **connector1.properties**:

   ```
   show schemas from connector1;
   ```

> **Note** Replace `connector1` with the property that you configure in Step 1: Configure a connector.

Information similar to the following example is returned:

```
      Schema
--------------------
 emruser
 information_schema
 performance_schema
 sys
(4 rows)
```

Step 3: Query table data

> **Note** In this example, `hive.default.tbl_department` is a Hive table, and `connector1.emruser.tbl_employee` is a MySQL table.

- Query data in the *emruser.tbl_employee* table.

```
select * from connector1.emruser.tbl_employee;
```

Data similar to the following example is returned:

```
 id |    name     | dept_id | salary
----+-------------+---------+---------
  1 | Ming Li     |       1 | 10000.0
  2 | Eric Cai    |       1 | 11000.0
  3 | Bonnie Liu  |       2 | 11000.0
(3 rows)
```

- Query data in the *tbl_department* table.

```
select * from hive.default.tbl_department;
```

Data similar to the following example is returned:

```
dept_id | dept_name
--------+----------
      1 | IT
      2 | Finance
(2 rows)
```

- Perform a cross query on data in the tables.

  Run the following command to perform a cross query on data in the *hive.default.tbl_department* and *connector1.emruser.tbl_employee* tables.

```
select * from hive.default.tbl_department a, connector1.emruser.tbl_employee b where a.de
pt_id = b.dept_id;
```

The following information is returned:

```
dept_id |dept_name | id |    name    | dept_id | salary
--------+----------+----+------------+---------+---------
      2 | Finance  |  3 | Bonnie Liu |       2 | 11000.0
      1 | IT       |  2 | Eric Cai   |       1 | 11000.0
      1 | IT       |  1 | Ming Li    |       1 | 10000.0
(3 rows)
```

2.2. E-MapReduce data migration solution

This topic describes how to migrate data from a self-built cluster to an E-MapReduce (EMR) cluster.

Applicable migration scenarios include:

- Migrating data from an offline Hadoop cluster to E-MapReduce.
- Migrating data from a self-built Hadoop cluster on ECS to E-MapReduce.

Supported data sources include:

- HDFS incremental upstream data sources such as RDS incremental data and Flume.

Network interconnection between new and old clusters

- Self-built Hadoop cluster on an offline IDC

  A self-built Hadoop cluster can be migrated to E-MapReduce through OSS, or by using Alibaba Cloud Express Connect, to establish a connection between your offline IDC and the VPC in which your E-MapReduce cluster is located.

- Self-built Hadoop cluster on ECS instances

  Because VPC networks are logically isoloated, we recommend that you use the VPC-Connected E-MapReduce service to establish an interconnection. Depending on the specific network types involved for interconnection, you need to perform the following actions:

  - Interconnection between classic networks and VPC networks

    For a Hadoop cluster built on ECS instances, you need to interconnect the classic network and VPC network using the ECS ClassicLink method. For more information, see Build a ClassicLink connection.

  - Interconnection between VPC networks

    To ensure optimal connectivity between VPC networks, we recommned that you create the new cluster in the same region and zone as the old cluster.

HDFS data migration

- Synchronize data with DistCp

  For more information, see Hadoop DistCp.

  You can migrate full and incremental HDFS data using the DistCp tool. To alleviate pressures on your current cluster resources, we recommend that you execute the **distcp** command after the new and old cluster networks are interconnected.

- Full data synchronization

```
hadoop distcp -pbugpcax -m 1000 -bandwidth 30 hdfs://oldclusterip:8020/user/hive/warehouse /user/hive/warehouse
```

- Incremental data synchronization

```
hadoop distcp -pbugpcax -m 1000 -bandwidth 30  -update -delete hdfs://oldclusterip:8020 /user/hive/warehouse /user/hive/warehouse
```

Parameter descriptions:

- *hdfs://oldclusterip:8020* indicates the namenode IP of the old cluster. If there are multiple namenodes, input the namenode that is in the active status.
- By default, the number of replicas is 3. If you want to keep the original number of replicas, add r after -p, such as -prbugpcax. If the permissions and ACL do not need to be synchronized, remove p and a after -p.
- -m indicates the amount of maps and the size of the cluster, which corresponds to the data volume. For example, if a cluster has a 2000-core cpu, you can specify 2000 maps.
- -bandwidth indicates an estimated value of the synchronized speed of a single map, which is implemented by controlling the copy speed of replicas.
- -update, verifies the checksum and file size of the source and target files. If the file sizes compared are different, the source file updates the target cluster data. If there are data writes during the synchronization of the old and new clusters, -update can be used for incremental data synchronization.
- -delete, if data in the old cluster no longer exists, the data in the new cluster will be deleted.

> **Note**
>
> - The overall speed of migration is affected by cluster badwidth and size. The larger the number of files, the longer the checksum takes to process. If you have a large amount of data to migrate, try to synchronize several directories to evaluate the overall time. If synchronization is performed within the specified time period, you can split the directory into several small directories and synchronize them one at a time.
> - A short service stop is required for the full data synchronization to enable double write and double counting, and to directly switch the service to the new cluster.

- HDFS permission configuration

  HDFS provides permission settings. Before migrating HDFS data, you need to ensure whether the old cluster has an ACL rule and if the rule is to be synchronized, and check if dfs.permissions.enabled and dfs.namenode.acls.enabled were configured the same in the old and new clusters. The configurations will take effect immediately.

  If there is an ACL rule to be synchronized, the distcp parameter must be added to -p to synchronize the permission parameter. If the distcp operation displays that the cluster does not support the ACL, this means that you did not set the ACL rule for the corresponding cluster. If the new cluster is not configured with the ACL rule, you can add it and restart NM. If the old cluster does not support an ACL rule, you do not need to set or synchronize an ACL rule.

Hive metadata synchronization

- Overview

Hive metadata is generally stored in MySQL. When compared with MySQL data synchronization, note that:

- The location must be changed
- Hive version alignment is required

E-MapReduce supports Hive metabases, including

- Unified metabase, whereby EMR manages RDS and each user has a schema
- Self-built RDS
- Self-built MySQL on ECS instances

To ensure data is consistent after migration between the old and new clusters, we recommend that you stop the metastore service during the migration, open the metastore service on the old cluster after the migration, and then submit jobs on the new cluster.

- Procedure:
  i. Delete the metabase of the new cluster and input `drop database xxx`.
  ii. Run the `mysqldump` command to export the table structure and data of the old cluster's metabase.
  iii. Replace the location. Tables and partitions in the Hive metadata all have location information witinh the dfs nameservices prefix, such as *hdfs://mycluster:8020/*. However, the nameservices prefix of an E-MapReduce cluster is emr-cluster, which means you need to replace the location information.

  To replace the location information, export the data as follows.

  ```
  mysqldump --databases hivemeta --single-transaction -u root -p > hive_databases.sql
  ```

  Use sed to replace *hdfs://oldcluster:8020/* with *hdfs://emr-cluster/* and then import data into the new database.

  ```
  mysql hivemeta -p < hive_databases.sql
  ```

  iv. n the interface of the new cluster, stop the hivemetastore service.
  v. Log on to the new metabase and create a database.
  vi. In the new metabase, import all data exported from the old metabase after the location field is replaced.
  vii. Currently, E-MapReduce Hive version is the latest stable version. However, if the Hive version of your self-built cluster is earlier, any imported data may not be directly usable. To resolve this issue, you need to execute the upgraded Hive script (ignore table and field problems). For more information, see Hive upgrade scripts. For example, to upgrade Hive 1.2 to 2.3.0, execute **upgrade-1.2.0-to-2.0.0.mysql.sql**, **upgrade-2.0.0-to-2.1.0.mysql.sql**, **upgrade-2.1.0-to-2.2.0.mysql.sql**, and **upgrade-2.2.0-to-2.3.0.mysql.sql** in sequence. This script is mainly used to build the table, add the field, and change the content.
  viii. Exceptions that the table and the field already exist can be ignored. After all metadata are revised, restart MetaServer, input the **hive** command in the command line, query the database and table, and verify the information is correct.

Flume data migration

- Flume simultaneous write in two clusters configuration

Enable the Flume service in the new cluster and write the data to the new cluster in accordance with the rules that are identical to the old cluster.

- Write the Flume partition table

  When executing the Flume data double-write, you must control the start timing to make sure that the new cluster is synchronized when Flume starts a new time partition. If Flume synchronizes all the tables every hour, you need to enable the Flume synchronization service before the next synchronization. This ensures that the data written by Flume in the new hour is properly duplicated. Incomplete old data is then synchronized when full data synchronization with DistCp is performed. The new data generated after the simultaneous write time is enabled is not synchronized.

  > **Note** When you partition data, do NOT put the new written data into the data synchronization directory.

Job synchronization

If the verion upgrades of Hadoop, Hive, Spark, and MapReduce are large, you may rebuild your jobs on demand.

Common issues and corresponding solutions are as follows:

- Gateway OOM

  Change */etc/ecm/hive-conf/hive-env.sh*.

  export HADOOP_HEAPSIZE=512 is changed to 1024.

- Insufficient job execution memory

  mapreduce.map.java.opts adjusts the startup parameters passed to the virtual machine when the JVM virtual machine is started. The default value -Xmx200m indicates the maximum amount of heap memory used by this Java program. When the amount is exceeded, the JVM displays the Out of Memory exception

  ```
  and terminates the set mapreduce.map.java.opts=-Xmx3072m process.
  ```

  mapreduce.map.memory.mb sets the memory limit of the Container, which is read and controlled by NodeManager. When the memory size of the Container exceeds this parameter value, NodeManager will kill the Container.

  ```
  set mapreduce.map.memory.mb=3840
  ```

Data verification

Data is verified through a customer's self-generated reports.

Presto cluster migration

If a Presto cluster is used for data queries, the Hive configuration files need to be modified. For more information, see Presto documentation.

The Hive properties that need to be modified are as follows:

- connector.name=hive-hadoop2
- hive.metastore.uri=thrift://emr-header-1.cluster-500148414:9083
- hive.config.resources=/etc/ecm/hadoop-conf/core-site.xml, /etc/ecm/hadoop-conf/hdfs-site.xml
- hive.allow-drop-table=true

- hive.allow-rename-table=true
- hive.recursive-directories=true

Appendix

Alignment example of upgrading Hive version 1.2 to 2.3:

```
source /usr/lib/hive-current/scripts/metastore/upgrade/mysql/upgrade-1.2.0-to-2.0.0.mysql.s
ql
CREATE TABLE COMPACTION_QUEUE (
  CQ_ID bigint PRIMARY KEY,
  CQ_DATABASE varchar(128) NOT NULL,
  CQ_TABLE varchar(128) NOT NULL,
  CQ_PARTITION varchar(767),
  CQ_STATE char(1) NOT NULL,
  CQ_TYPE char(1) NOT NULL,
  CQ_WORKER_ID varchar(128),
  Cq_start bigint,
  CQ_RUN_AS varchar(128),
  CQ_HIGHEST_TXN_ID bigint,
  CQ_META_INFO varbinary(2048),
  CQ_HADOOP_JOB_ID varchar(32)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
CREATE TABLE TXNS (
  TXN_ID bigint PRIMARY KEY,
  TXN_STATE char(1) NOT NULL,
  TXN_STARTED bigint NOT NULL,
  TXN_LAST_HEARTBEAT bigint NOT NULL,
  TXN_USER varchar(128) NOT NULL,
  TXN_HOST varchar(128) NOT NULL,
  TXN_AGENT_INFO varchar(128),
  TXN_META_INFO varchar(128),
  TXN_HEARTBEAT_COUNT int
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
CREATE TABLE HIVE_LOCKS (
  HL_LOCK_EXT_ID bigint NOT NULL,
  HL_LOCK_INT_ID bigint NOT NULL,
  HL_TXNID bigint,
  HL_DB varchar(128) NOT NULL,
  HL_TABLE varchar(128),
  HL_PARTITION varchar(767),
  HL_LOCK_STATE char(1) not null,
  HL_LOCK_TYPE char(1) not null,
  HL_LAST_HEARTBEAT bigint NOT NULL,
  HL_ACQUIRED_AT bigint,
  HL_USER varchar(128) NOT NULL,
  HL_HOST varchar(128) NOT NULL,
  HL_HEARTBEAT_COUNT int,
  HL_AGENT_INFO varchar(128),
  HL_BLOCKEDBY_EXT_ID bigint,
  HL_BLOCKEDBY_INT_ID bigint,
  PRIMARY KEY(HL_LOCK_EXT_ID, HL_LOCK_INT_ID),
  KEY HIVE_LOCK_TXNID_INDEX (HL_TXNID)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
CREATE INDEX HL_TXNID_IDX ON HIVE_LOCKS (HL_TXNID);
```

```
CREATE INDEX HL_TXNID_IDX ON HIVE_LOCKS (HL_TXNID);
source /usr/lib/hive-current/scripts/metastore/upgrade/mysql/upgrade-1.2.0-to-2.0.0.mysql.sql
source /usr/lib/hive-current/scripts/metastore/upgrade/mysql/upgrade-2.0.0-to-2.1.0.mysql.sql
CREATE TABLE TXN_COMPONENTS (
  TC_TXNID bigint,
  TC_DATABASE varchar(128) NOT NULL,
  TC_TABLE varchar(128),
  TC_PARTITION varchar(767),
  FOREIGN KEY (TC_TXNID) REFERENCES TXNS (TXN_ID)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
source /usr/lib/hive-current/scripts/metastore/upgrade/mysql/upgrade-2.0.0-to-2.1.0.mysql.sql
source /usr/lib/hive-current/scripts/metastore/upgrade/mysql/upgrade-2.1.0-to-2.2.0.mysql.sql
CREATE TABLE IF NOT EXISTS `NOTIFICATION_LOG`
(
    `NL_ID` BIGINT(20) NOT NULL,
    `EVENT_ID` BIGINT(20) NOT NULL,
    `EVENT_TIME` INT(11) NOT NULL,
    `EVENT_TYPE` varchar(32) NOT NULL,
    `DB_NAME` varchar(128),
    `TBL_NAME` varchar(128),
    `MESSAGE` mediumtext,
    PRIMARY KEY (`NL_ID`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
CREATE TABLE IF NOT EXISTS `PARTITION_EVENTS` (
  `PART_NAME_ID` bigint(20) NOT NULL,
  `DB_NAME` varchar(128) CHARACTER SET latin1 COLLATE latin1_bin DEFAULT NULL,
  `EVENT_TIME` bigint(20) NOT NULL,
  `EVENT_TYPE` int(11) NOT NULL,
  `PARTITION_NAME` varchar(767) CHARACTER SET latin1 COLLATE latin1_bin DEFAULT NULL,
  `TBL_NAME` varchar(128) CHARACTER SET latin1 COLLATE latin1_bin DEFAULT NULL,
  PRIMARY KEY (`PART_NAME_ID`),
  KEY `PARTITIONEVENTINDEX` (`PARTITION_NAME`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
CREATE TABLE COMPLETED_TXN_COMPONENTS (
  CTC_TXNID bigint NOT NULL,
  CTC_DATABASE varchar(128) NOT NULL,
  CTC_TABLE varchar(128),
  CTC_PARTITION varchar(767)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
 source /usr/lib/hive-current/scripts/metastore/upgrade/mysql/upgrade-2.1.0-to-2.2.0.mysql.sql
 source /usr/lib/hive-current/scripts/metastore/upgrade/mysql/upgrade-2.2.0-to-2.3.0.mysql.sql
  CREATE TABLE NEXT_TXN_ID (
  NTXN_NEXT bigint NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
INSERT INTO NEXT_TXN_ID VALUES(1);
CREATE TABLE NEXT_LOCK_ID (
  NL_NEXT bigint NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
INSERT INTO NEXT_LOCK_ID VALUES(1);
```

```
INSERT INTO NEXT_BLOCK_ID VALUES(1);
```

2.3. Use Flink jobs to process OSS data

This topic describes how to run Flink jobs in a Dataflow cluster to process Object Storage Service (OSS) data.

Background information

In the example of this topic, the data development feature provided by E-MapReduce (EMR) is used. The data development feature stopped updating at 21:00 on February 21, 2022. We recommend that you do not use the feature.

Prerequisites

- EMR and OSS are activated.
- The Alibaba Cloud account or RAM user that you want to use is granted the required permissions. For more information, see Assign roles.

Procedure

1. Step 1: Prepare the environment
2. Step 2: Prepare test data
3. Step 3: Create and run a Flink job
4. Step 4: View a job log and job details (Optional)

Step 1: Prepare the environment

Create a Dataflow cluster in Flink mode in the EMR console. For more information, see Create a cluster.

> **Note** A Dataflow cluster of EMR V3.39.1 is used in this topic.

Step 2: Prepare test data

Before you create a Flink job, you must upload test data to an OSS bucket. In the example of this topic, a file named *test.txt* that contains `Nothing is impossible for a willing heart. While there is a life, there is a hope~` is uploaded.

1. Log on to the OSS console.
2. Create an OSS bucket and upload the test data file to the bucket. For information about how to create an OSS bucket, see Create buckets. For information about how to upload a file, see Upload objects.

   In this example, the path of the uploaded file is *oss://vvr-test/test.txt*. Keep the path for later use.

Step 3: Create and run a Flink job

1. Go to the Data Platform tab.
   i. Log on to the Alibaba Cloud EMR console by using your Alibaba Cloud account.

ii. In the top navigation bar, select the region where your cluster resides and select a resource group based on your business requirements.

iii. Click the **Data Platform** tab.

2. On the **Data Platform** page, create a project. For more information, see Manage projects.
3. Create a Flink job.
   i. In the Edit Job pane on the left, right-click the folder on which you want to perform operations and select **Create Job**.
   ii. In the **Create Job** dialog box, specify **Name** and **Description**, and select **Flink** from the **Job Type** drop-down list.
   iii. Click **OK**.
4. Edit job content.

   Example:

   ```
   run -m yarn-cluster -yjm 1024 -ytm 1024 -ynm flink-oss-sample /usr/lib/flink-current/examples/batch/WordCount.jar --input oss://vvr-test/test.txt
   ```

   Descriptions of key parameters in the preceding code:

   - */usr/lib/flink-current/examples/batch/WordCount.jar*: the built-in Flink WordCount job in the Dataflow cluster. For more information about the code, see the official code repository.
   - *oss://vvr-test/test.txt*: the path of the uploaded test data file.
5. Click **Run** in the upper-right corner.

   In the **Run Job** dialog box, select the created Dataflow cluster for **Target Cluster**.
6. Click **OK**.

   The Flink job is run in the EMR cluster to process OSS data. The following information is printed in the log:

   ```
   (a,3)
   (for,1)
   (heart,1)
   (hope,1)
   (impossible,1)
   (is,3)
   (life,1)
   (nothing,1)
   (there,2)
   (while,1)
   (willing,1)
   ================JOB OUTPUT END================
   ```

Step 4: View a job log and job details (Optional)

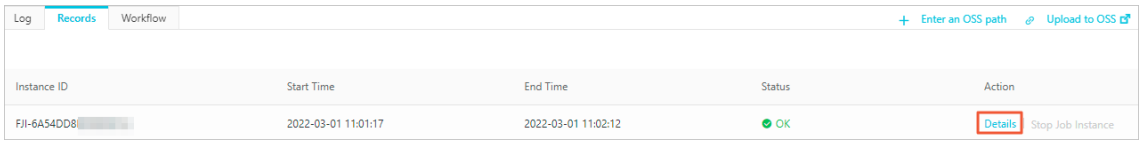
You can click the Log tab of a job to identify the cause of a job failure. You can click the Records tab of a job to view details about the job.

1. View a job log.

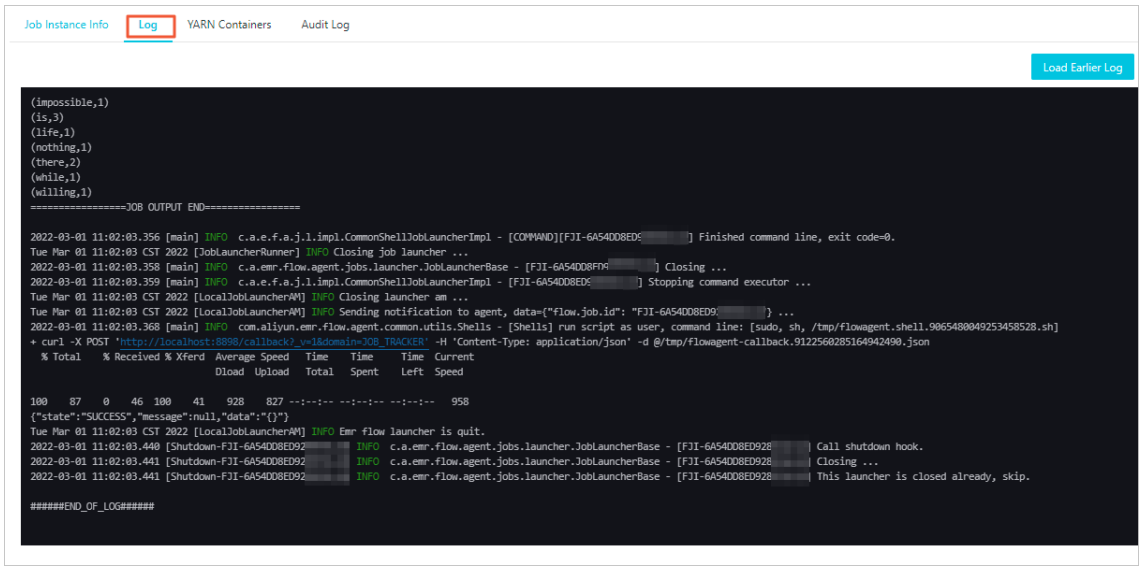
   You can view a job log in the EMR console or on an SSH client.

   - View a job log in the EMR console: After you submit a job, click the **Records** tab, find your job,

and then click **Details** in the Action column.



Click the **Log** tab to view the job log.



- View a job log on an SSH client: Connect to the master node of the cluster in SSH mode and view the log of the job you submitted.

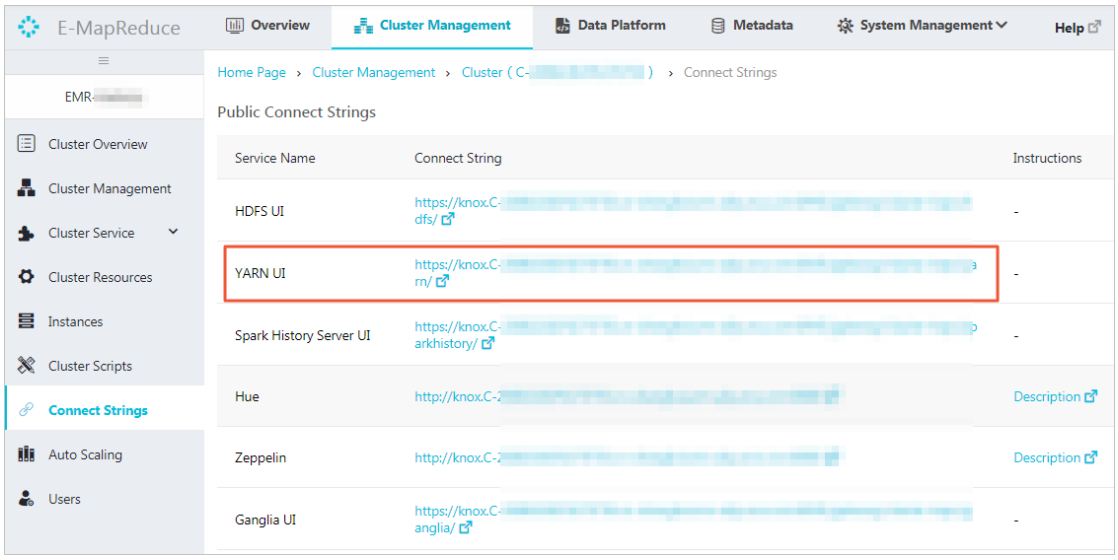
  By default, the log data of a Flink job is stored in */mnt/disk1/log/flink/flink-{user}-client-{hostname}.log* based on the configurations in **log4j**. For more information about the configurations in log4j, see log4j-yarn-session.properties in the */etc/ecm/flink-conf/* directory.

  user indicates the account that you use to submit a Flink job. hostname indicates the name of the node on which a job is submitted. For example, you submit a Flink job on the **emr-header-1** node as the root user. The log path is */mnt/disk1/log/flink/flink-flink-historyserver-0-emr-header-1.cluster-126601.log*.

2. View job details.

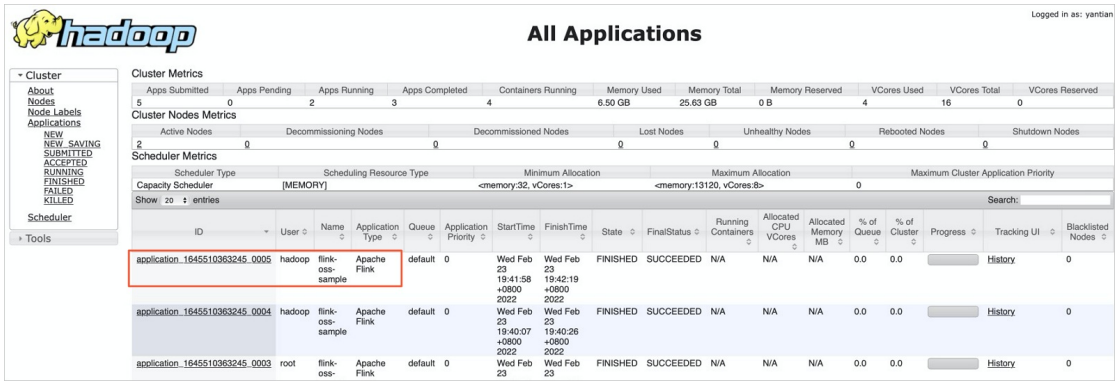
   You can view the details of a Flink job on the web UI of **YARN**. You can use an SSH tunnel or Knox to access the web UI of **YARN**. For information about how to use an SSH tunnel to access the web UI of YARN, see Create an SSH tunnel to access web UIs of open source components. For information about how to use Knox to access the web UI of YARN, see Knox and Access the web UIs of open source components. Knox is used in this example to describe how to view the details of a job.

i. On the **Public Connect Strings** page, click the URL for **YARN UI**.

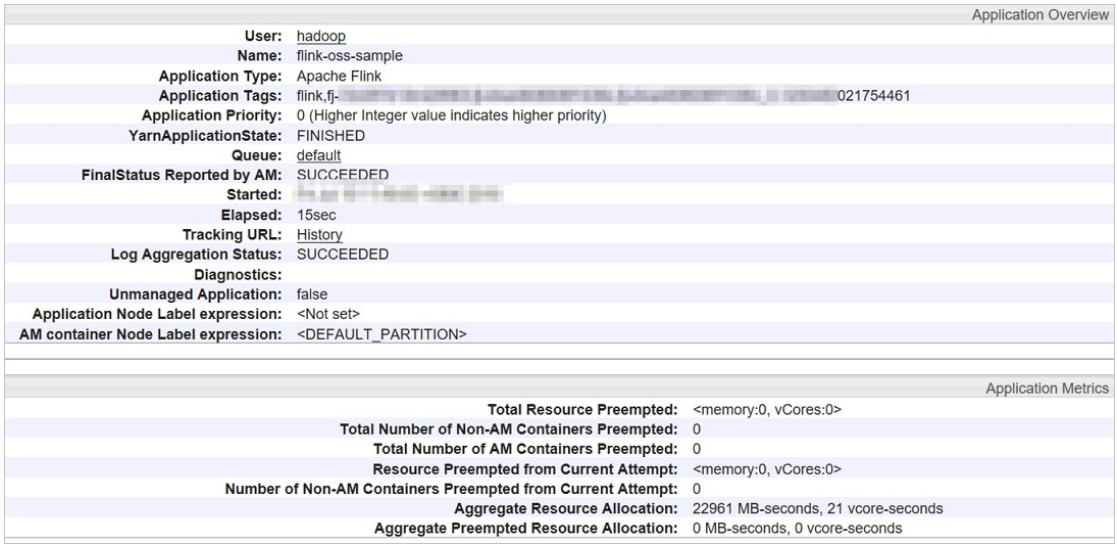


ii. In the Hadoop console, click the ID of the job.

View the details of a specific job.



The following figure shows the details.



iii. If you want to view the Flink jobs that are running, click the link next to **Tracking URL** on the job details page. The **Flink Dashboard** page displays the list of running Flink jobs.

iv. After the Flink job is completed, you can view the list and logs of all completed Flink jobs.

For more information, see How do I access Flink HistoryServer in a Dataflow cluster? and How do I view the status of Flink jobs?.

2.4. Use EMR to transmit MySQL binary logs in near real time

This topic describes how to use plug-ins in Alibaba Cloud Log Service and an E-MapReduce (EMR) cluster to transmit MySQL binary logs in near real time.

Prerequisites

- An EMR Hadoop cluster is created. For more information, see Create a cluster.
- A MySQL database, such as an ApsaraDB RDS for MySQL database or a DRDS database, is created. The binary logging feature must be enabled for the MySQL database, and the binary log format must be set to ROW.

  An ApsaraDB RDS for MySQL instance is used as an example in this topic. For information about how to create an ApsaraDB RDS for MySQL instance, see Create an ApsaraDB RDS for MySQL instance.

  > **Note** By default, the binary logging feature is enabled for the ApsaraDB RDS for MySQL instance.

Procedure

1. Connect to the ApsaraDB RDS for MySQL instance and add user permissions.
   i. Run a command to connect to the ApsaraDB RDS for MySQL instance. For more information, see Use a database client or the CLI to connect to an ApsaraDB RDS for MySQL instance.
   ii. Run the following commands to add user permissions:

   ```
   CREATE USER canal IDENTIFIED BY 'canal';
   GRANT SELECT, REPLICATION SLAVE, REPLICATION CLIENT ON *.* TO 'canal'@'%';
   FLUSH PRIVILEGES;
   ```

2. Add a configuration file for Log Service. For more information, see Collect MySQL binary logs.

   > **Note** A project named canaltest and a Logstore named canal are created.

   Check whether log data is uploaded in the Log Service console. If log data fails to be uploaded, use the log collection feature of Log Service to troubleshoot issues.

3. Compile a JAR package and upload it to an Object Storage Service (OSS) bucket.
   i. Open Git Bash on your system and run the following command to copy sample code:

   ```
   git clone https://github.com/aliyun/aliyun-emapreduce-demo.git
   ```

ii. Modify sample code.

LoghubSample is contained in sample code. This class is used to collect data from Log Service and display the data. Sample code after the modification:

```
package com.aliyun.emr.example
import org.apache.spark.SparkConf
import org.apache.spark.storage.StorageLevel
import org.apache.spark.streaming.aliyun.logservice.LoghubUtils
import org.apache.spark.streaming.{Milliseconds, StreamingContext}
object LoghubSample {
def main(args: Array[String]): Unit = {
if (args.length < 7) {
 System.err.println(
   """Usage: bin/spark-submit --class LoghubSample examples-1.0-SNAPSHOT-shaded.jar
     |
     |
   """.stripMargin)
 System.exit(1)
}
val loghubProject = args(0)
val logStore = args(1)
val loghubGroupName = args(2)
val endpoint = args(3)
val accessKeyId = args(4)
val accessKeySecret = args(5)
val batchInterval = Milliseconds(args(6).toInt * 1000)
val conf = new SparkConf().setAppName("Mysql Sync")
//    conf.setMaster("local[4]");
val ssc = new StreamingContext(conf, batchInterval)
val loghubStream = LoghubUtils.createStream(
 ssc,
 loghubProject,
 logStore,
 loghubGroupName,
 endpoint,
 1,
 accessKeyId,
 accessKeySecret,
 StorageLevel.MEMORY_AND_DISK)
loghubStream.foreachRDD(rdd =>
   rdd.saveAsTextFile("/mysqlbinlog")
)
ssc.start()
ssc.awaitTermination()
}
}
```

In the preceding sample code, `loghubStream.foreachRDD(rdd => rdd.saveAsObjectFile("/mysqlbinlog") )` is changed to loghubStream.foreachRDD(rdd => rdd.saveAsTextFile("/mysqlbinlog"). This way, data from Spark Streaming jobs that are run in EMR can be stored to HDFS of EMR.

iii. Debug code in your system and run the following command to package the code:

```
mvn clean install
```

iv. Upload the JAR package to an OSS bucket.

Create an OSS bucket and upload the JAR package to the OSS bucket. For more information, see Create buckets and Upload objects.

> **Note** In this example, an OSS bucket named EMR-test is created. The *examples-1.1-shaded.jar* package is uploaded to the *EMR-test/jar* directory.

4. Create a Spark job.

i. Go to the job editing page.

a. Log on to the Alibaba Cloud EMR console by using your Alibaba Cloud account.

b. In the top navigation bar, select the region where your cluster resides and select a resource group based on your business requirements.

c. Click the **Data Platform** tab.

d. In the **Projects** section of the page that appears, find the project that you want to edit and click **Edit Job** in the Actions column.

ii. In the **Edit Job** pane on the left, right-click the folder on which you want to perform operations and select **Create Job**.

iii. In the Create Job dialog box, specify **Name** and **Description** and select **Spark** from the **Job Type** drop-down list.

iv. Click **OK**.

v. Specify the command line parameters required to submit the job in the **Content** field.

```
--master yarn --deploy-mode client --driver-memory 4g --executor-memory 2g --executor-cores 2 --class com.aliyun.EMR.example.LoghubSample ossref://EMR-test/jar/examples-1.1-shaded.jar canaltest canal sparkstreaming <SLS_endpoint> <SLS_access_id> <SLS_secret_key> 1
```

Parameter	Description
SLS_endpoint	The endpoint of Log Service.
SLS_access_id	The AccessKey ID of your Alibaba Cloud account.
SLS_secret_key	The AccessKey secret of your Alibaba Cloud account.

> **Note** The digit 1 at the end of the sample code indicates the batchInterval parameter that specifies the batch processing interval of Spark jobs. For more information about other parameters, see Configure spark-submit parameters.

vi. Click **Save** in the upper-right corner.

5. Run the job.

i. In the upper-right corner of the job editing page, click **Run**.

ii. In the **Run Job** dialog box, select the created Hadoop cluster from the **Target Cluster** drop-down list.

iii. Click **OK**.

6. View files in the mysqlbinlog directory.

i. Log on to the master node of the Hadoop cluster in SSH mode. For more information, see Log on to a cluster.

ii. Run the following command to view files in the mysqlbinlog directory:

```
hadoop fs -ls /mysqlbinlog
```

You can also run the `hadoop fs -cat /mysqlbinlog/part-00000` command to view the file content.

2.5. Use Spark Streaming jobs to process Kafka data

This topic describes how to run Spark Streaming jobs to process Kafka data in the E-MapReduce (EMR) console.

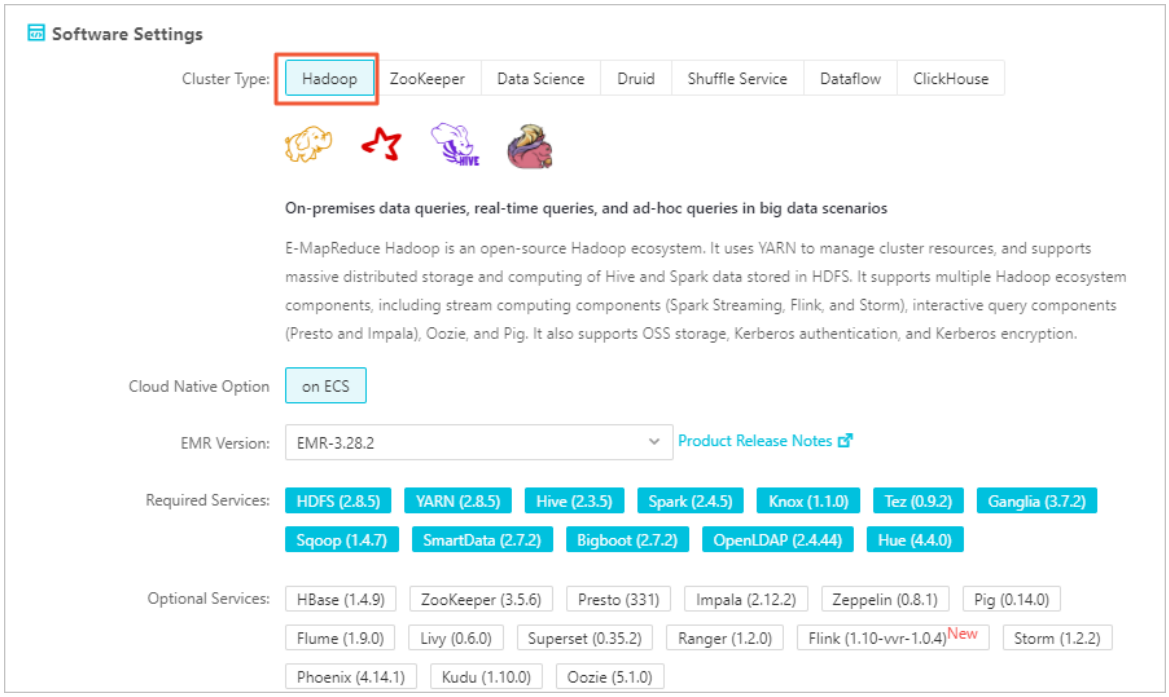
Prerequisites

- EMR is activated.
- The Alibaba Cloud account is authorized. For more information, see Assign roles.
- PuTTY and SSH Secure File Transfer Client are installed on your on-premises machine.

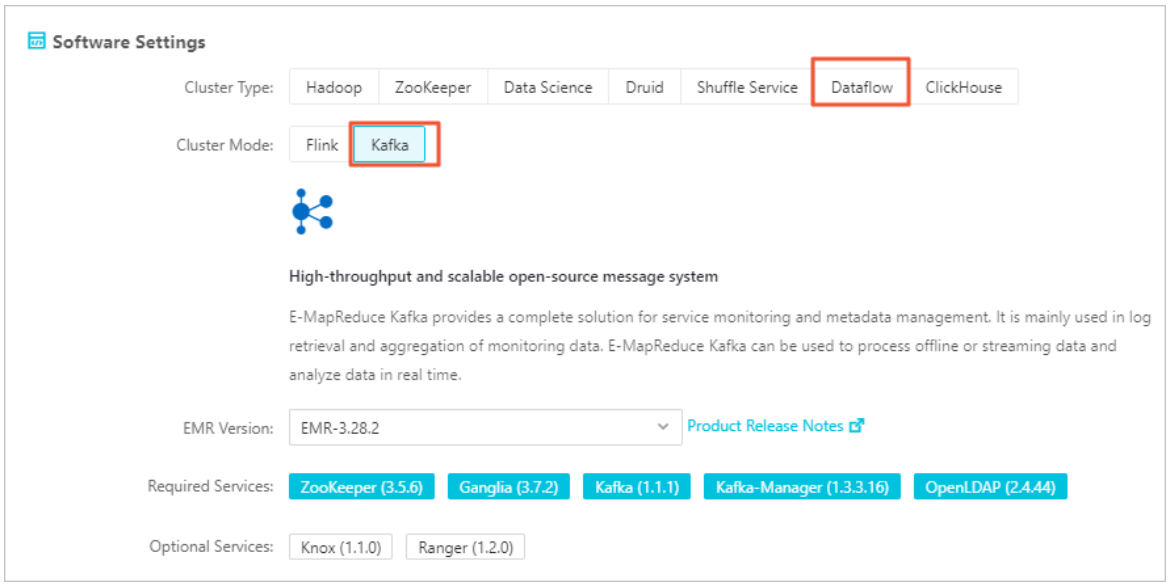
Step 1: Create a Hadoop cluster and a Kafka cluster

Create a Hadoop cluster and a Kafka cluster that belong to the same security group. For more information, see Create a cluster.

1. Log on to the Alibaba Cloud EMR console.
2. Create a Hadoop cluster.

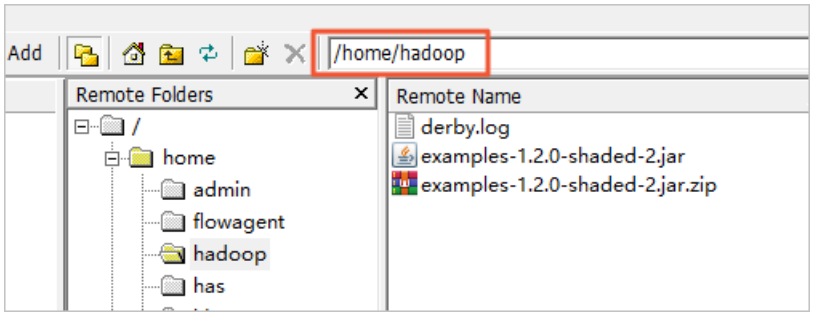


3. Create a Kafka cluster.



Step 2: Obtain the required JAR package and upload it to the Hadoop cluster

1. Obtain the JAR package examples-1.2.0-shaded-2.jar.zip.
2. Use SSH Secure File Transfer Client to upload the JAR package to the */home/hadoop* path of the master node in the Hadoop cluster.



Step 3: Create a topic on the Kafka cluster

In this example, a topic named test is created. The topic has 10 partitions and 2 replicas.

1. Log on to the master node of the Kafka cluster. For more information, see Log on to a cluster.
2. Run the following command to create a topic:

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

> **Note** After you create the topic, keep the logon window open for later use.

Step 4: Run a Spark Streaming job

In this example, a WordCount job is run for streaming data.

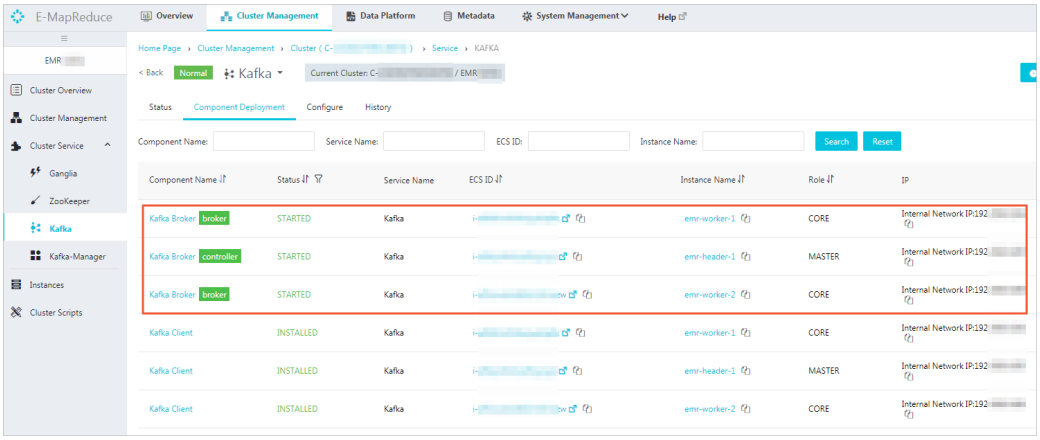
1. Log on to the master node of the Hadoop cluster. For more information, see Log on to a cluster.
2. Run the following command to submit a WordCount job for streaming data:

```
spark-submit --class com.aliyun.emr.example.spark.streaming.KafkaSample  /home/hadoop/e
xamples-1.2.0-shaded-2.jar 192.168.xxx.xxx:9092 test 5
```

The following table describes the parameters.

Parameter	Description
`192.168.xxx.xxx`	The internal IP address of a **Kafka broker** component in the Kafka cluster. For more information, see List of components in the Kafka cluster.
`test`	The name of the topic.
`5`	The time interval.

List of components in the Kafka cluster



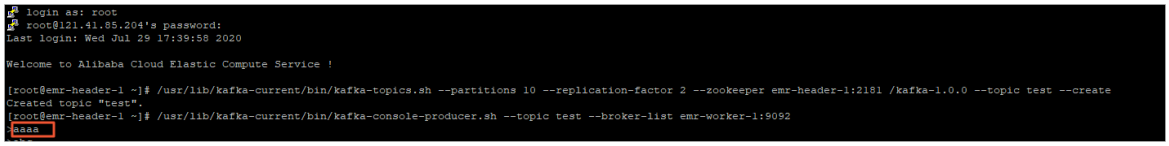
Step 5: Use Kafka to publish messages

1. In the command-line interface (CLI) of the Kafka cluster, run the following command to start the Kafka producer:

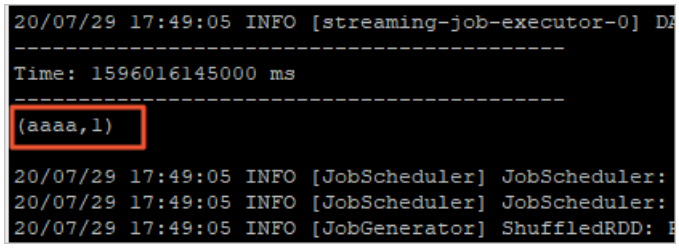
   ```
   /usr/lib/kafka-current/bin/kafka-console-producer.sh --topic test --broker-list emr-worker-1:9092
   ```

2. Enter text information in the logon window of the Kafka cluster. Text statistics are displayed in the logon window of the Hadoop cluster in real time.

   For example, enter the information shown in the following figure to the logon window of the Kafka cluster.

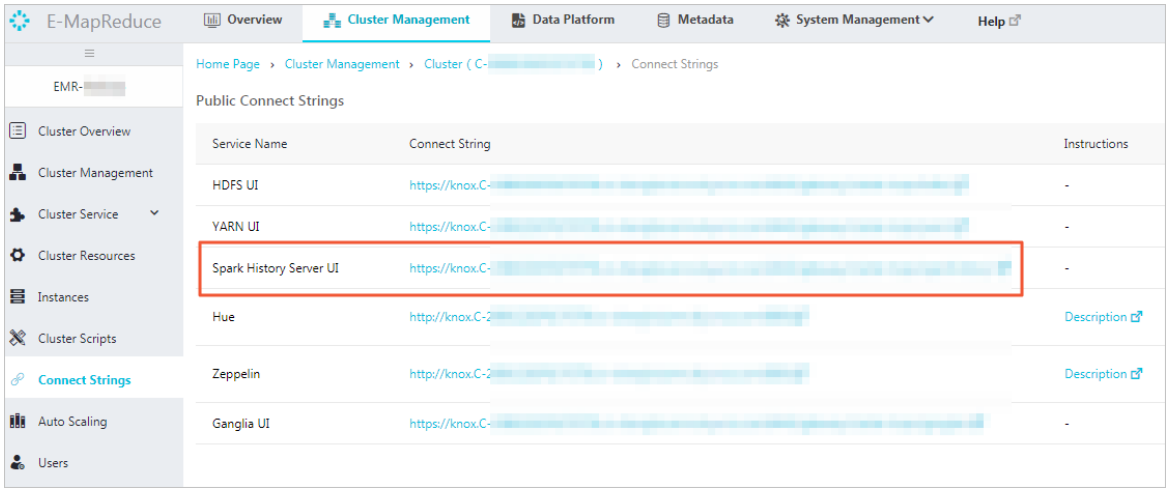
   

   The information shown in the following figure is displayed in the logon window of the Hadoop cluster.

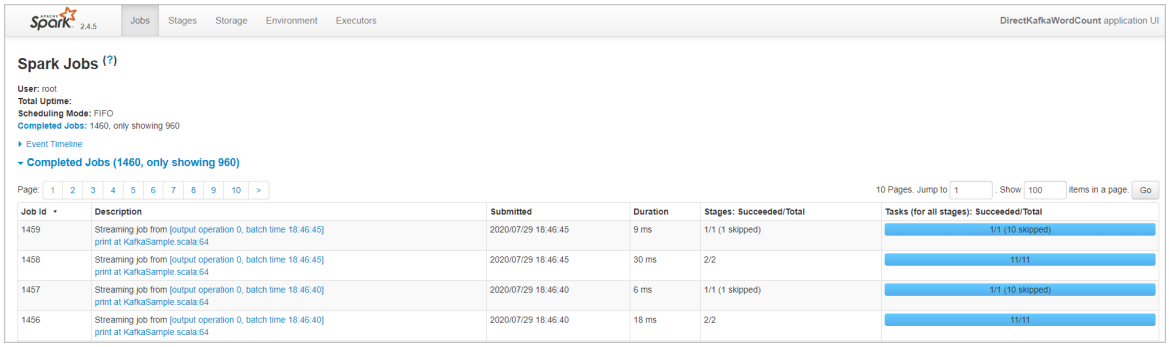
Step 6: View the status of the Spark Streaming job

1. Click the **Cluster Management** tab in the EMR console.
2. On the Cluster Management page, find the Hadoop cluster you created and click **Details** in the Actions column.
3. In the left-side navigation pane of the Cluster Overview page, click **Connect Strings**.
4. Click the link of **Spark History Server UI**.



5. On the **History Server** page, click the **App ID** of the Spark Streaming job that you want to view.

   You can view the status of the Spark Streaming job.



2.6. Use Kafka Connect to migrate data

When streaming data is processed, data synchronization between Kafka and other systems or data migration between Kafka clusters is often required. This topic describes how to use Kafka Connect in E-MapReduce (EMR) to migrate data between Kafka clusters.

Prerequisites

- An Alibaba Cloud account is created.
- EMR is activated.
- The Alibaba Cloud account is authorized. For more information, see Assign roles.

Context

Kafka Connect is a scalable and reliable tool used to synchronize data between Kafka and other systems and to transmit streaming data between Kafka clusters. For example, you can use Kafka Connect to obtain binlog data from a database and migrate the data of the database to a Kafka cluster. This also indirectly connects the database to the downstream streaming data processing system of the Kafka cluster. Kafka Connect also provides a RESTful API to help you create and manage Kafka Connect connectors.

Kafka Connect can run in standalone or distributed mode. In standalone mode, all workers run in the same process. The distributed mode is more scalable and fault-tolerant than the standalone mode. The distributed mode is the most commonly used mode and the recommended mode for the production environment.

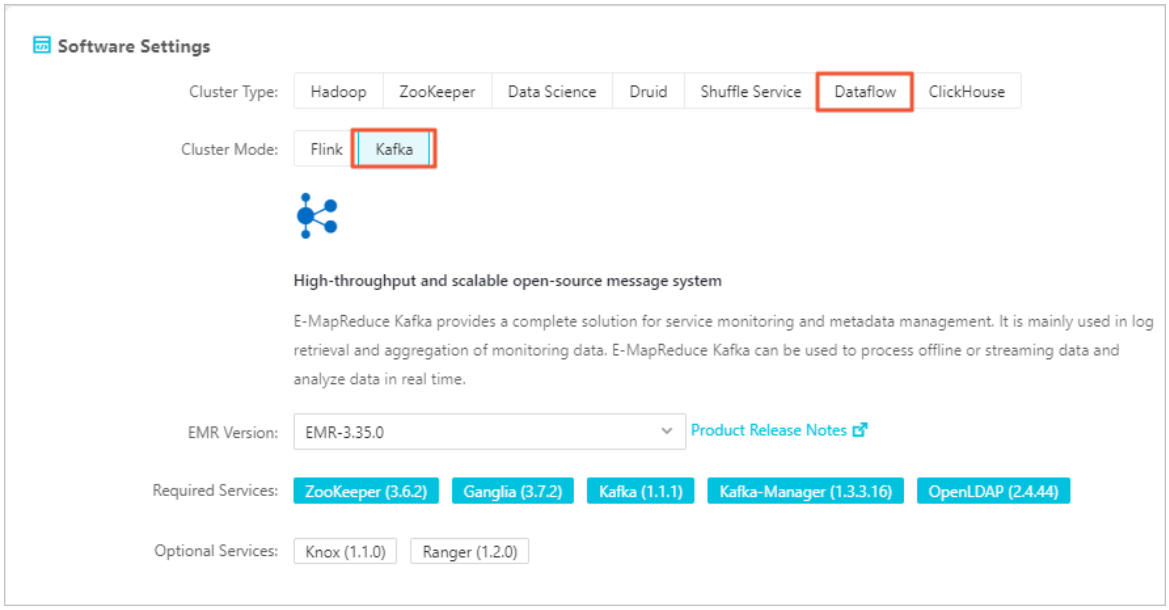
This topic describes how to call the RESTful API of Kafka Connect to migrate data between Kafka clusters, where Kafka Connect runs in distributed mode.

Step 1: Create Kafka clusters

Create a source Kafka cluster and a destination Kafka cluster in the EMR console. Kafka Connect is installed on a task node. To install Kafka Connect, you must create a task node in the destination Kafka cluster. Kafka Connect is started on the task node after the destination cluster is created. The port number is 8083.

1.
2. Create a source Kafka cluster and a destination Kafka cluster. For more information, see Create a cluster.

> **Notice**
>
> - We recommend that you add the source and destination Kafka clusters to the same security group.
>
> If the two clusters belong to different security groups, they are not accessible to each other. You must modify the settings of the security groups to allow mutual access between the two clusters.
> - After the destination Kafka cluster is created, you must add a task node.



Step 2: Create a topic used to store the data you want to migrate

Create a topic named **connect** in the source Kafka cluster.

1. Log on to the master node of the source Kafka cluster by using SSH. In this example, the master node is **emr-header-1**.

2. Run the following command as the root user to create a topic named **connect**:

```
kafka-topics.sh --create --zookeeper emr-header-1:2181 --replication-factor 2 --partitions 10 --topic connect
```

```
[root@emr-header-1 ~]# kafka-topics.sh --create --zookeeper emr-header-1:2181 --replication-factor 2 --partitions 10 --topic connect
Created topic "connect".
[root@emr-header-1 ~]#
```

> **Note** After you complete the preceding operations, do not close the logon window. The logon window is required in a later step.

Step 3: Create a Kafka Connect connector

On the task node of the destination Kafka cluster, run the `curl` command to create a Kafka Connect connector by using JSON data.

1. Customize Kafka Connect configurations.

   Go to the **Configure** tab of the **Kafka** service under the destination Kafka cluster in the Alibaba Cloud EMR console. Specify the **offset.storage.topic**, **config.storage.topic**, and **status.storage.topic** parameters on the **connect-distributed.properties** subtab.

   Kafka Connect saves the offsets, configurations, and task status in the topics specified by the **offset.storage.topic**, **config.storage.topic**, and **status.storage.topic** parameters, respectively. Kafka Connect automatically creates these topics by using the default settings of partitions and replication-factor that are saved in */etc/ecm/kafka-conf/connect-distributed.properties*.

2. Log on to the master node of the destination Kafka cluster. In this example, the master node is **emr-header-1**.

3. Switch to the task node named **emr-worker-3** in this example.

```
[root@emr-header-1 ~]# ssh emr-worker-3
root@emr-worker-3's password:
Last login: Fri Sep 18 11:23:39 2020 from 192.

Welcome to Alibaba Cloud Elastic Compute Service !

[root@emr-worker-3 ~]#
```

4. Run the following command as the root user to create a Kafka Connect connector:

```
curl -X POST -H "Content-Type: application/json" --data '{"name": "connect-test", "config": { "connector.class": "EMRReplicatorSourceConnector", "key.converter": "org.apache.kafka.connect.converters.ByteArrayConverter", "value.converter": "org.apache.kafka.connect.converters.ByteArrayConverter", "src.kafka.bootstrap.servers": "${src-kafka-ip}:9092", "src.zookeeper.connect": "${src-kafka-curator-ip}:2181", "dest.zookeeper.connect": "${dest-kafka-curator-ip}:2181", "topic.whitelist": "${source-topic}", "topic.rename.format": "${dest-topic}", "src.kafka.max.poll.records": "300" } }' http://emr-worker-3:8083/connectors
```

   In the JSON data, the name field specifies the name of the Kafka Connect connector that you want to create. In this example, the name is *connect-test*. Configure the config field based on your needs. The following table describes the key variables of the config field.

Variable	Description
${source-topic}	The topics that store the data to be migrated in the source Kafka cluster, for example, **connect**. Separate multiple topics with commas (,).
${dest-topic}	The topics to which data is migrated in the destination Kafka cluster, for example, **connect.replica**.
${src-kafka-curator-hostname}	The internal IP address of the node where the ZooKeeper service is deployed in the source Kafka cluster.
${dest-kafka-curator-hostname}	The internal IP address of the node where the ZooKeeper service is deployed in the destination Kafka cluster.

> **Note** After you complete the preceding operations, do not close the logon window. The logon window is required in a later step.

Step 4: View the status of the Kafka Connect connector and task node

View the status of the Kafka Connect connector and task node and make sure that they are normal.

1. Return to the logon window of the task node of the destination Kafka cluster. In this example, the task node is **emr-worker-3**.
2. Run the following command as the root user to view all Kafka Connect connectors:

   ```
   curl emr-worker-3:8083/connectors
   ```

   ```
   [root@emr-worker-3 ~]# curl emr-worker-3:8083/connectors
   ["connect-test"][root@emr-worker-3 ~]#
   ```

3. Run the following command as the root user to view the status of the Kafka Connect connector created in this example, that is, **connect-test**:

   ```
   curl emr-worker-3:8083/connectors/connect-test/status
   ```

   ```
   [root@emr-worker-3 ~]# curl emr-worker-3:8083/connectors/connect-test/status
   {"name":"connect-test","connector":{"state":"RUNNING","worker_id":"192.168.  :8083"},"tasks":[{"state":"RUNNING","id":0,"worker_id":"192.168.  :8083"}],"type":"source"}[r
   oot@emr-worker-3 ~]#
   ```

   Make sure that the Kafka Connect connector, **connect-test** in this example, is in the **RUNNING** state.

4. Run the following command as the root user to view the details of the task node:

   ```
   curl emr-worker-3:8083/connectors/connect-test/tasks
   ```

   ```
   [root@emr-worker-3 ~]# curl emr-worker-3:8083/connectors/connect-test/status
   {"name":"connect-test","connector":{"state":"RUNNING","worker_id":"192.168.  :8083"},"tasks":[{"state":"RUNNING","id":0,"worker_id":"192.168.  :8083"}],"type":"source"}[r
   oot@emr-worker-3 ~]# curl emr-worker-3:8083/connectors/connect-test/tasks
   [{"id":{"connector":"connect-test","task":0},"config":{"connector.class":"EMRReplicatorSourceConnector","src.zookeeper.connect":"emr-header-1.cluster-127390:2181","topic.rename.f
   ormat":"connect.replica","dest.zookeeper.connect":"emr-header-1.cluster-127499:2181","task.class":"org.apache.kafka.connect.replicator.EMRReplicatorSourceTask","src.kafka.max.pol
   l.records":"300","name":"connect-test","task.id":"connect-test-0","value.converter":"org.apache.kafka.connect.converters.ByteArrayConverter","key.converter":"org.apache.kafka.con
   nect.converters.ByteArrayConverter","src.kafka.bootstrap.servers":"192.168.  :9092","topic.whitelist":"connect","partition.assignment":"AAAAAAABAAdjb25uZWN0AAAACgAAAAAAAAABAA
   AAAgAAAAMAAAAEAAAABQAAAAYAAAAHAAAACAAAAAkAAAAA"}}][root@emr-worker-3 ~]#
   ```

   Make sure that no error message about the task node is returned.

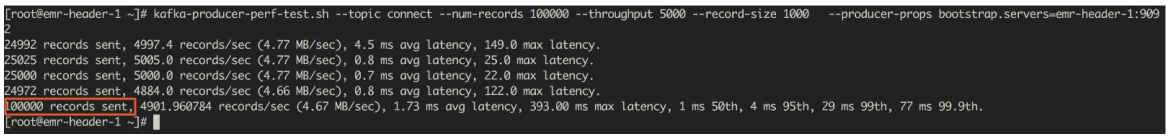
Step 5: Generate the data to be migrated

Send the data to be migrated to the **connect** topic in the source Kafka cluster.

1. Return to the logon window of the master node of the source Kafka cluster. In this example, the master node is **emr-header-1**.
2. Run the following command as the root user to send data to the **connect** topic:

   ```
   kafka-producer-perf-test.sh --topic connect --num-records 100000 --throughput 5000 --record-size 1000 --producer-props bootstrap.servers=emr-header-1:9092
   ```

   When the information shown in the following figure appears, the data to be migrated is generated.

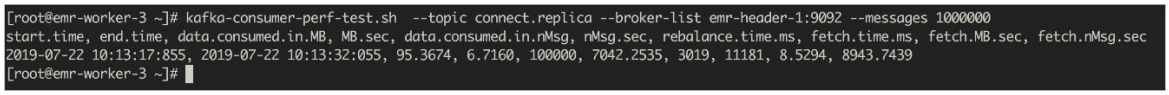


Step 6: View the data migration results

After the data to be migrated is generated, Kafka Connect automatically migrates the data to the corresponding topic in the destination Kafka cluster. In this example, the topic is **connect.replica**.

1. Return to the logon window of the task node of the destination Kafka cluster. In this example, the task node is **emr-worker-3**.
2. Run the following command as the root user to check whether the data is migrated:

   ```
   kafka-consumer-perf-test.sh --topic connect.replica --broker-list emr-header-1:9092 --messages 100000
   ```



   Based on the command output in the preceding figure, the 100,000 messages sent to the source Kafka cluster are migrated to the destination Kafka cluster.

Summary

This topic describes how to use Kafka Connect to migrate data between Kafka clusters. For more information about how to use Kafka Connect, visit the Kafka official website and see RESTful API.

2.7. Use PyFlink jobs to process Kafka data

This topic describes how to run PyFlink jobs in a Hadoop cluster and a Kafka cluster created in the E-MapReduce (EMR) console to process Kafka data.

Prerequisites

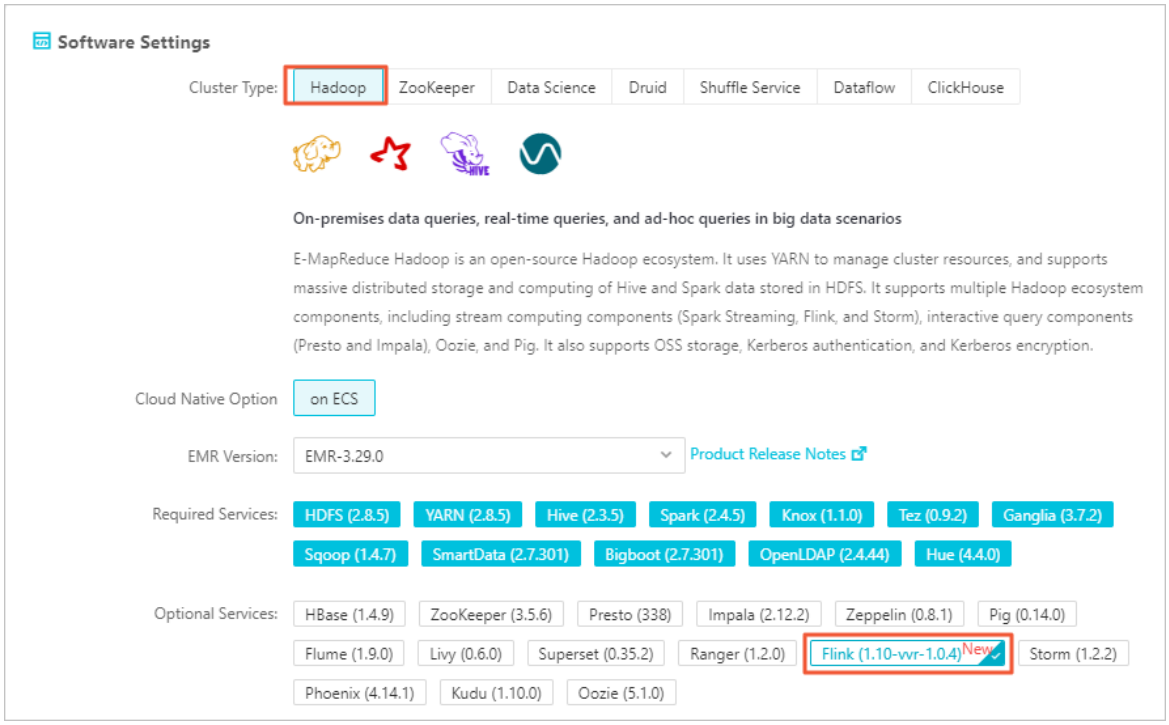
- EMR is activated.
- The Alibaba Cloud account is authorized. For more information, see Assign roles.
- A project is created. For more information, see Manage projects.
- PuTTY and SSH Secure File Transfer Client are installed on your on-premises machine.

Step 1: Create a Hadoop cluster and a Kafka cluster

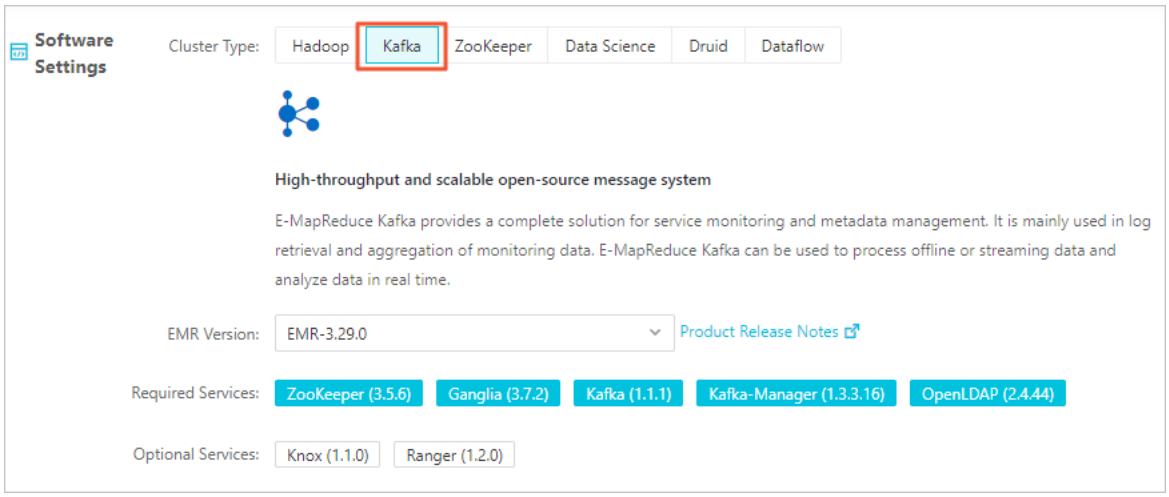
Create a Hadoop cluster and a Kafka cluster that belong to the same security group. For more information, see Create a cluster.

> **Note** EMR V3.29.0 is used as an example in this topic.

1. Log on to the Alibaba Cloud EMR console.
2. Create a Hadoop cluster and select Flink from Optional Services.



3. Create a Kafka cluster.



Step 2: Create a topic in the Kafka cluster

In this example, two topics named payment_msg and results are created. Each topic has 10 partitions and 2 replicas.

1. Log on to the master node of the Kafka cluster. For more information, see Log on to a cluster.
2. Run the following command to create a topic named payment_msg:

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

3. Run the following command to create a topic named results:

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

> **Note** After you create the topic, keep the logon window open for later use.

Step 3: Prepare test data

In the command-line interface (CLI) of the Kafka cluster that is created in Step 2, run the following commands to continuously generate test data:

```
python3 -m pip install kafka
rm -rf produce_data.py
cat>produce_data.py<<EOF
import random
import time, calendar
from random import randint
from kafka import KafkaProducer
from json import dumps
from time import sleep
def write_data():
    data_cnt = 20000
    order_id = calendar.timegm(time.gmtime())
    max_price = 100000
    topic = "payment_msg"
    producer = KafkaProducer(bootstrap_servers=['emr-worker-1:9092'],
                             value_serializer=lambda x: dumps(x).encode('utf-8'))
    for i in range(data_cnt):
        ts = time.strftime("%Y-%m-%d %H:%M:%S", time.localtime())
        rd = random.random()
        order_id += 1
        pay_amount = max_price * rd
        pay_platform = 0 if random.random() < 0.9 else 1
        province_id = randint(0, 6)
        cur_data = {"createTime": ts, "orderId": order_id, "payAmount": pay_amount, "payPla
tform": pay_platform, "provinceId": province_id}
        producer.send(topic, value=cur_data)
        sleep(0.5)
if __name__ == '__main__':
    write_data()
EOF
python3 produce_data.py
```

Step 4: Create and run a PyFlink job

1. Log on to the master node of the Hadoop cluster. For more information, see Log on to a cluster.
2. Run the following commands to generate the *lib.jar* and *job.py* files:

```
rm -rf job.py
```

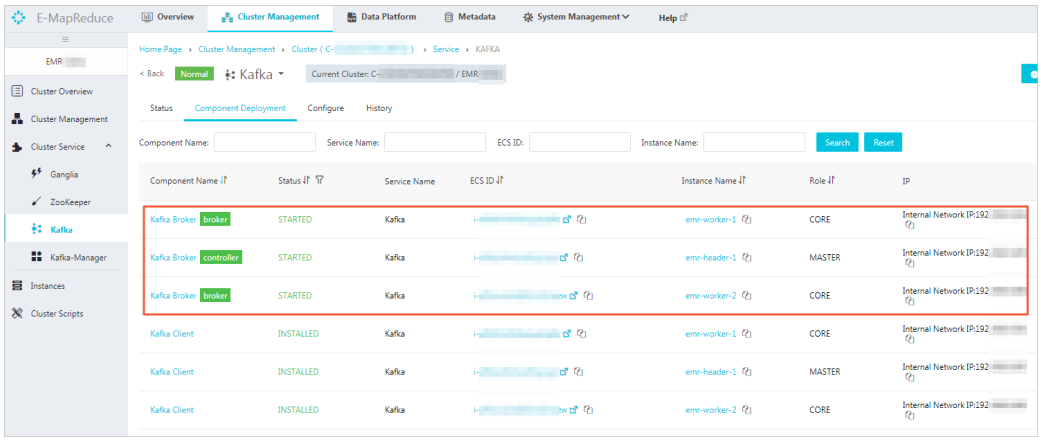
```
cat>job.py<<EOF
import os
from pyflink.datastream import StreamExecutionEnvironment, TimeCharacteristic
from pyflink.table import StreamTableEnvironment, DataTypes, EnvironmentSettings
from pyflink.table.udf import udf
provinces = ("beijing", "shanghai", "hangzhou", "shenzhen", "jiangxi", "chongqing", "xi
zang")
@udf(input_types=[DataTypes.INT()], result_type=DataTypes.STRING())
def province_id_to_name(id):
    return provinces[id]
# Enter the following information based on the created Kafka cluster:
def log_processing():
    kafka_servers = "xx.xx.xx.xx:9092,xx.xx.xx.xx:9092,xx.xx.xx.xx:9092"
    kafka_zookeeper_servers = "xx.xx.xx.xx:2181,xx.xx.xx.xx:2181,xx.xx.xx.xx:2181"
    source_topic = "payment_msg"
    sink_topic = "results"
    kafka_consumer_group_id = "test_3"
    env = StreamExecutionEnvironment.get_execution_environment()
    env.set_stream_time_characteristic(TimeCharacteristic.EventTime)
    env_settings = EnvironmentSettings.Builder().use_blink_planner().build()
    t_env = StreamTableEnvironment.create(stream_execution_environment=env, environment
_settings=env_settings)
    t_env.get_config().get_configuration().set_boolean("python.fn-execution.memory.mana
ged", True)
    source_ddl = f"""
            CREATE TABLE payment_msg(
                createTime VARCHAR,
                rt as TO_TIMESTAMP(createTime),
                orderId BIGINT,
                payAmount DOUBLE,
                payPlatform INT,
                provinceId INT,
                WATERMARK FOR rt as rt - INTERVAL '2' SECOND
            ) WITH (
              'connector.type' = 'kafka',
              'connector.version' = 'universal',
              'connector.topic' = '{source_topic}',
              'connector.properties.bootstrap.servers' = '{kafka_servers}',
              'connector.properties.zookeeper.connect' = '{kafka_zookeeper_servers}',
              'connector.properties.group.id' = '{kafka_consumer_group_id}',
              'connector.startup-mode' = 'latest-offset',
              'format.type' = 'json'
            )
            """
    es_sink_ddl = f"""
            CREATE TABLE es_sink (
            province VARCHAR,
            pay_amount DOUBLE,
            rowtime TIMESTAMP(3)
            ) with (
              'connector.type' = 'kafka',
              'connector.version' = 'universal',
              'connector.topic' = '{sink_topic}',
              'connector.properties.bootstrap.servers' = '{kafka_servers}',
```

```
                'connector.properties.zookeeper.connect' = '{kafka_zookeeper_servers}',
                'connector.properties.group.id' = '{kafka_consumer_group_id}',
                'connector.startup-mode' = 'latest-offset',
                'format.type' = 'json'
            )
    """
    t_env.sql_update(source_ddl)
    t_env.sql_update(es_sink_ddl)
    t_env.register_function('province_id_to_name', province_id_to_name)
    query = """
    select province_id_to_name(provinceId) as province, sum(payAmount) as pay_amount, t
umble_start(rt, interval '5' second) as rowtime
    from payment_msg
    group by tumble(rt, interval '5' second), provinceId
    """
    t_env.sql_query(query).insert_into("es_sink")
    t_env.execute("payment_demo")
if __name__ == '__main__':
    log_processing()
EOF
rm -rf lib
mkdir lib
cd lib
wget https://maven.aliyun.com/nexus/content/groups/public/org/apache/flink/flink-sql-co
nnector-kafka_2.11/1.10.1/flink-sql-connector-kafka_2.11-1.10.1.jar
wget https://maven.aliyun.com/nexus/content/groups/public/org/apache/flink/flink-json/1
.10.1/flink-json-1.10.1-sql-jar.jar
cd ../
zip -r lib.jar lib/*
```

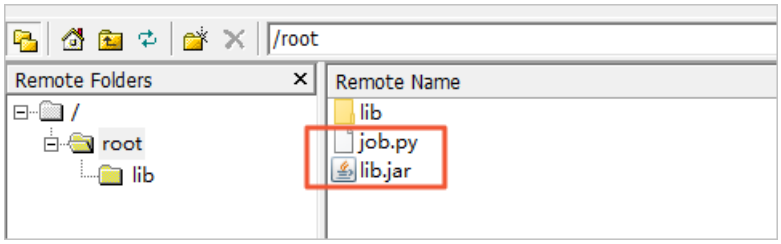
Specify the following parameters in *job.py* based on the actual situation of the cluster.

Parameter	Description
`kafka_servers`	The list of IP addresses for Kafka brokers in the Kafka cluster. All the IP addresses are the internal IP address of the Kafka cluster. The default port number is 9092. For more information about the IP addresses, see List of components in the Kafka cluster.
`kafka_zookeeper_servers`	The list of IP addresses for ZooKeeper components in the Kafka cluster. All the IP addresses are the internal IP address of the Kafka cluster. The default port number is 2181. For more information about the IP addresses, see List of components in the Kafka cluster.
`source_topic`	The Kafka topic of the source table. In this example, the topic is payment_msg.
`sink_topic`	The Kafka topic of the result table. In this example, the topic is results.

List of components in the Kafka cluster



The following figure provides an example of *lib.jar* and *job.py*.



3. Use SSH Secure File Transfer Client to connect to the master node of the Hadoop cluster, and then download and save *lib.jar* and *job.py* to your on-premises machine that runs a Windows operating system.
4. Upload *lib.jar* and *job.py* to the OSS console.
   i. Log on to the OSS console.
   ii. Create an OSS bucket and upload the two files to the bucket. For information about how to create an OSS bucket, see Create buckets. For information about how to upload a file, see Upload objects.

   In this example, the upload paths are *oss://emr-logs2/test/lib.jar* and *oss://emr-logs2/test/job.py*.
5. Create a PyFlink job.
   i.
   ii. In the top navigation bar, select the region where your cluster resides and select a resource group based on your business requirements.
   iii. Click the **Data Platform** tab.
   iv. In the **Projects** section of the page that appears, find the project you want to edit and click **Edit Job** in the Actions column.
   v. In the **Edit Job** pane on the left, right-click the folder on which you want to perform operations and select **Create Job**.
   vi. In the Create Job dialog box, specify Name and Description and select **Flink** from the **Job Type** drop-down list.
   vii. Configure the job content. Example:

```
run -m yarn-cluster -py ossref://emr-logs2/test/job.py -j ossref://emr-logs2/test/lib.jar
```

6. Run the PyFlink job.
   i. Click **Save** in the upper-right corner.
   ii. Click **Run** in the upper-right corner.
   iii. In the **Run Job** dialog box, select the created Hadoop cluster from the **Target Cluster** drop-down list.
   iv. Click **OK**.

Step 5: View job details

1. You can view the details of a PyFlink job on the web UI of YARN.

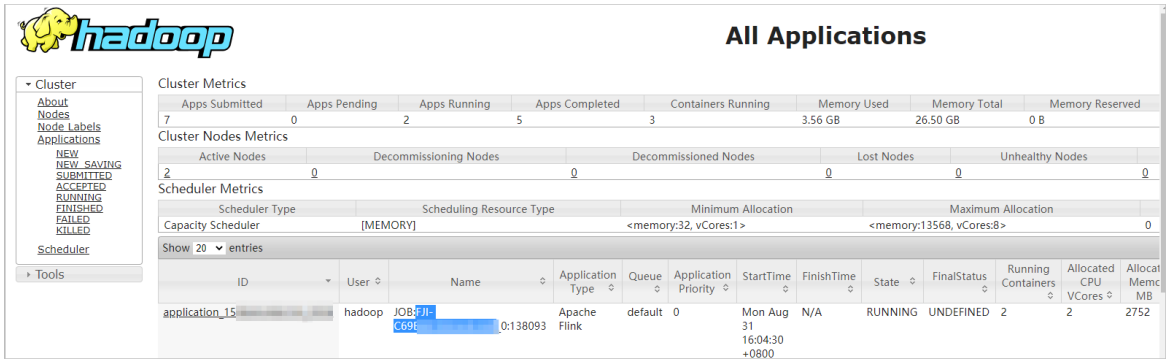
   You can use one of the following methods to access the web UI of YARN:

   - Use an SSH tunnel: For more information, see Create an SSH tunnel to access web UIs of open source components.
   - Use Knox: For more information, see Access the web UIs of open source components.

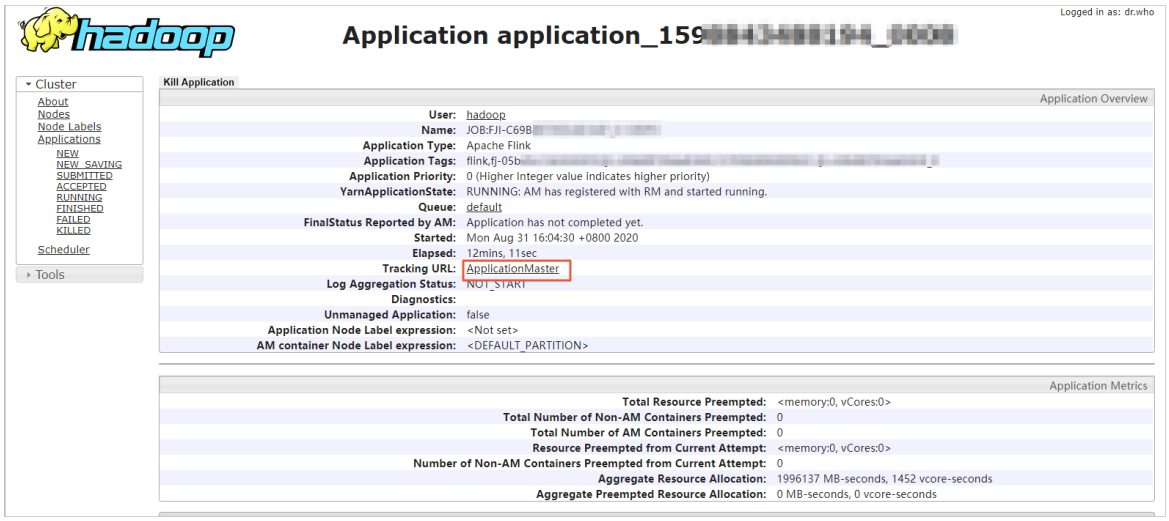
   Use Knox as an example to view the details of a PyFlink job.

2. In the Hadoop console, click the ID of the job.

   You can view the running details of the job.



   The following figure shows the running details.



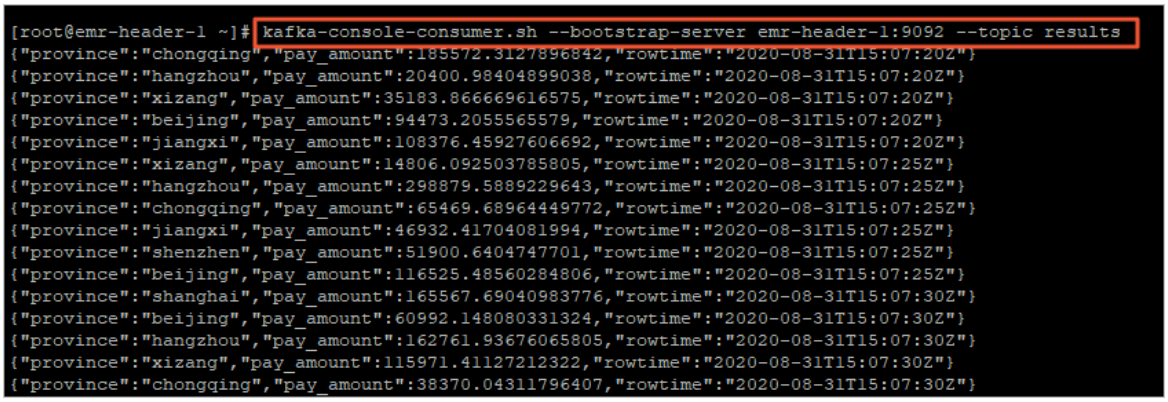
3. (Optional)Click the link next to **Tracking URL** to go to the **Apache Flink Dashboard** page and view detailed job information.

Step 6: View the output

1. Log on to the master node of the Kafka cluster. For more information, see Log on to a cluster.
2. Run the following command to view the data in the results topic:

```
kafka-console-consumer.sh --bootstrap-server emr-header-1:9092 --topic results
```

The information shown in the following figure is returned.



After you view the information, you can click **Stop** in the upper-right corner of the job page on the **Data Platform** tab to stop the running job.