

Alibaba Cloud E-MapReduce

Best Practices

Issue: 20190820

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 due to product version upgrades, adjustments, or other reasons. Alibaba Cloud reserves the right to modify the content of this document without notice and the updated versions of this document will be occasionally released through Alibaba Cloud-authorized channels. You shall 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 the document in the context that Alibaba Cloud products and services are provided on an "as is", "with all faults" and "as available" basis. 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 bear any liability for any errors or financial losses incurred by any organizations, companies, or individuals arising from their download, use, or trust in this document. Alibaba Cloud shall not, under any circumstances, bear responsibility for any indirect, consequential, exemplary, incidental, 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 content of the Alibaba Cloud website, including but not limited to works, products, images, archives, information, materials, website architecture, website graphic layout, and webpage design, 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 the Alibaba Cloud website, product programs, or content 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 contact Alibaba Cloud directly if you discover any errors in this document.

Generic conventions

Table -1: Style conventions

| Style | Description | Example |
|---|--|--|
|  | This warning information 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. |
|  | This warning information 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 restore business. |
|  | This indicates warning information, supplementary instructions, and other content that the user must understand. |  Notice: Take the necessary precautions to save exported data containing sensitive information. |
| | This indicates supplemental instructions, best practices, tips, and other content that is good to know for the user. |  Note: You can use Ctrl + A to select all files. |
| > | Multi-level menu cascade. | Settings > Network > Set network type |
| Bold | It is used for buttons, menus, page names, and other UI elements. | Click OK. |
| Courier font | It is used for commands. | Run the <code>cd / d C :/ windows</code> command to enter the Windows system folder. |
| <i>Italics</i> | It is used for parameters and variables. | <code>bae log list --instanceid Instance_ID</code> |
| [] or [a b] | It indicates that it is an optional value, and only one item can be selected. | <code>ipconfig [-all -t]</code> |

| Style | Description | Example |
|---------------------------------------|--|-------------------------------------|
| <code>{}</code> or <code>{a b}</code> | It indicates that it is a required value, and only one item can be selected. | <code>switch {stand slave}</code> |

Contents

| | |
|---|----|
| Legal disclaimer..... | I |
| Generic conventions..... | I |
| 1 Use E-MapReduce to collect metrics from a Kafka client..... | 1 |
| 2 Use E-MapReduce to process offline jobs..... | 6 |
| 3 Submit Storm topologies to process data in Kafka on E-MapReduce..... | 10 |
| 4 Use ES-Hadoop on E-MapReduce..... | 17 |
| 5 Use Mongo-Hadoop on E-MapReduce..... | 24 |
| 6 Deep learning with Analytics Zoo on E-MapReduce..... | 30 |
| 7 Adaptive execution of Spark SQL..... | 36 |
| 8 E-MapReduce data migration solution..... | 42 |
| 9 Use EMR Data Science clusters for deep learning..... | 51 |
| 10 Use Flink jobs to process OSS data..... | 60 |
| 11 Submit a Flink job to process OSS data using EMR..... | 67 |
| 12 Connect to ApsaraDB for HBase using E-MapReduce Hive... | 71 |
| 13 Use EMR for real-time MySQL binlog transmission..... | 77 |
| 14 Run Flume on a Gateway node to synchronize data..... | 83 |
| 15 OSS ACL..... | 87 |
| 16 Configure a network connection for using Sqoop to transfer data from a database to an EMR cluster..... | 90 |
| 17 Use E-MapReduce to submit a Spark Streaming job for consuming Kafka data..... | 93 |
| 18 Use Kafka Connect to migrate data..... | 99 |

1 Use E-MapReduce to collect metrics from a Kafka client

This section describes how to use E-MapReduce to collect metrics from a Kafka client to conduct effective performance monitoring.

Background

Kafka provides a collection of metrics that are used to measure the performance of Broker, Consumer, Producer, Stream, and Connect. E-MapReduce collects metrics for Kafka Broker by using Ganglia to monitor the running status of this Kafka Broker. A Kafka system consists of two roles: a Kafka Broker and multiple Kafka clients. When an issue of read/write performance occurs, you must perform an analysis on the both Kafka Broker and clients. Metrics from Kafka clients are important for performing the analysis.

Principle

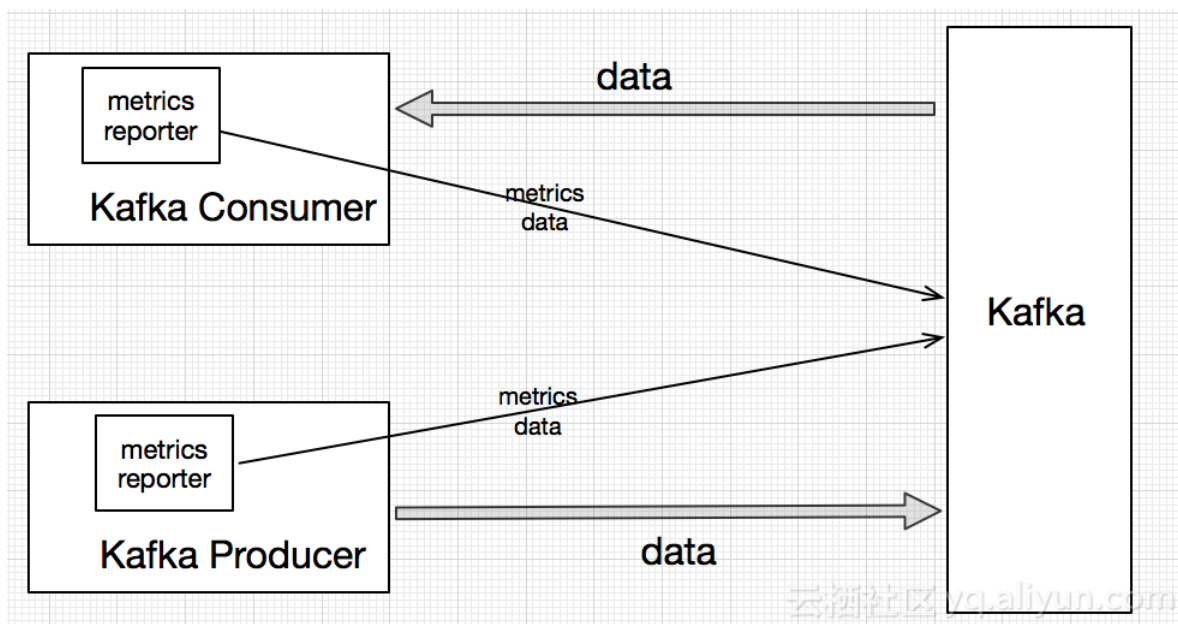
- Collect Metrics for Kafka performance

Kafka supports multiple external Metrics Reporters. JMX Reporter is built in to Kafka by default. You can use the JMX tool to view metrics of Kafka. You can implement your own Metrics Reporter such as `org.apache.kafka.common.metrics.MetricsReporter` to collect custom metrics.

- Store Metrics

You can customize Kafka metrics. In addition, you need a data store to keep these metrics for later use and analysis. You can store metrics to Kafka without using a third-party data store as Kafka itself is a data store. In addition, Kafka can be

easily integrated with other services. You can collect metrics from a client as the following figure shows:



E-MapReduce provides a sample `emr-kafka-client-metrics`. You can download the source code from the link: [source code](#).





Prerequisites

- Restrictions
 - Support for only Java applications;
 - Support for only clients of Kafka 0.10 or later;
- Without compiling code by yourself, E-MapReduce has published the jar package in Maven. You can download the latest version from the [download link](#).

- In this section, we use E-MapReduce to automatically create a Kafka cluster. For more information, see [Create a cluster](#).

We use the following versions of E-MapReduce and Kafka:

- EMR Version: EMR-3.12.1
- Cluster Type: Kafka
- Software: Kafka-Manager (1.3.3.16), Kafka (2.11-1.0.1), ZooKeeper (3.4.12), and Ganglia (3.7.2)
- The network type of this Kafka cluster is VPC in the China (Hangzhou) region. The master instance group is configured with a public IP and an internal network IP. The following figure shows the details.

| Cluster | | | | | | | | | | | | | |
|---|--|--|--|---------------------|--------|-----------|-------------|------------|------------------------|--|--------------|--------------|---------------------|
| Name: dtplus_docs ID: C-035238279077DFF3 Region: cn-hangzhou Start Time: 2018-11-13 15:50:55 | Software Configuration: I/O Optimization: Yes High Availability: No Security Mode: Standard | Billing Method: Pay-As-You-Go Current Status: Idle Runtime: 9 Minutes12 Seconds | Bootstrap Operation/Software Configuration: EMR-3.14.0 ECS Role: AliyunEmrEcsDefaultRole | | | | | | | | | | |
| Software | | Network | | | | | | | | | | | |
| EMR Version: EMR-3.14.0 Cluster Type: KAFKA Software: Ganglia3.7.2 / Zookeeper3.4.13 / Kafka1.0.1 / Kafka-Manager1.3.3.16 | | Region ID: cn-hangzhou-g Network Type: vpc Security Group ID: sg-bp10qjy9802g8aslojbz VPC/VSwitch: vpc-bp122agmllskkby207co / vsw-bp11gkxodl87h0hcoy0j | | | | | | | | | | | |
| Host | | Master Instance Group  | | | | | | | | | | | |
| Master Instance Group(MASTER) Pay-As-You-Go Hosts: 1 CPU: 4 Cores Memory: 8GB Data Disk Type: SSD Disk80GB*4 Disks | | <table><tr><th>ECS ID</th><th>组件部署状态</th><th>Public IP</th><th>Intranet IP</th><th>Created At</th></tr><tr><td>i-bp10qjy9802g8aslojbz</td><td> Normal</td><td>47.110.76.42</td><td>192.168.0.92</td><td>2018-11-13 15:51:03</td></tr></table> | | ECS ID | 组件部署状态 | Public IP | Intranet IP | Created At | i-bp10qjy9802g8aslojbz |  Normal | 47.110.76.42 | 192.168.0.92 | 2018-11-13 15:51:03 |
| ECS ID | 组件部署状态 | Public IP | Intranet IP | Created At | | | | | | | | | |
| i-bp10qjy9802g8aslojbz |  Normal | 47.110.76.42 | 192.168.0.92 | 2018-11-13 15:51:03 | | | | | | | | | |
| Core Instance Group(CORE) Pay-As-You-Go Hosts: 2 CPU: 4 Cores Memory: 8GB Data Disk Type: SSD Disk80GB*4 Disks | | | | | | | | | | | | | |

- Configure metrics

| Metric | Description |
|---|--|
| <code>metric . reporters</code> | The sample Metrics Reporter: <code>org . apache . kafka . clients . reporter . EMRClientMetricsReporter</code> |
| <code>emr . metrics . reporter . bootstrap . servers</code> | The metrics that stores bootstrap servers of a Kafka cluster. |
| <code>emr . metrics . reporter . zookeeper . connect</code> | The metrics that stores Zookeeper addresses of a Kafka cluster. |

- Load metrics
 - Place the `emr-kafka-client-metrics` jar package on a client. Add the path of the jar package to the classpath of a client-side application.
 - Install the `emr-kafka-client-metrics` dependency on the jar package of a client-side application.

Procedures

1. Download the latest `emr-kafka-client-metrics` package.

```
wget http://central.maven.org/maven2/com/aliyun/emr/emr-kafka-client-metrics/1.4.3/emr-kafka-client-metrics-1.4.3.jar
```

2. Create a test topic.

```
kafka-topics.sh --zookeeper emr-header-1:2181/kafka-1.0.1 --partitions 10 --replication-factor 2 --topic test-metrics --create
```

3. Copy the `emr-kafka-client-metrics` package to the lib directory of a Kafka client.

```
cp emr-kafka-client-metrics-1.4.3.jar /usr/lib/kafka-current/libs/
```

4. Write data to a test topic. You can write the configurations of a Kafka Producer to the local `client.conf` file.

```
## client.conf :
metric.reporters = org.apache.kafka.clients.reporter.EMRClientMetricsReporter
emr.metrics.reporter.bootstrap.servers = emr-worker-1:9092
emr.metrics.reporter.zookeeper.connect = emr-header-1:2181/kafka-1.0.1
bootstrap.servers = emr-worker-1:9092
## Command :
kafka-producer-perf-test.sh --topic test-metrics --throughput 1000 --num-records 100000 --record-size 1024 --producer.config client.conf
```

5. View the current metrics from a client. The default metrics topic is `_emr-client-metrics`.

```
Kafka-console-consumer.sh --Topic _emr-client-metrics --Bootstrap-server emr-worker-1:9092
```

```
-- from - beginning
```

The returned message is shown as follows.

```
{ prefix = kafka . producer , client . ip = 192 . 168 . xxx . xxx  
  , client . process = 25536 @ emr - header - 1 . cluster - xxxx ,  
  attribute = request - rate , value = 894 . 4685104965 012 ,  
  timestamp = 1533805225 045 , group = producer - metrics ,  
  tag . client - id = producer - 1 }
```

**Note:**

| Field name | Description: |
|----------------|--|
| client.ip | The IP address of a client host. |
| client.process | The process ID of a client-side application. |
| attribute | The attribute name of a metric. |
| value | The value of a metric. |
| timestamp | The timestamp when you collect a metric. |
| tag.xxx | Other tag information of a metric. |

2 Use E-MapReduce to process offline jobs

This section describes how to use E-MapReduce to read data from OSS, and a set of offline data processing operations, such as data collection and data clean-up.

Overview

E-MapReduce clusters can be used in various scenarios. E-MapReduce supports all the scenarios that the Hadoop ecosystem and Spark support. E-MapReduce is based on Hadoop and Spark clusters. You can use Alibaba Cloud ECS instances hosted by E-MapReduce clusters in the same way as you would on your physical machines.

Two popular kinds of big data processing that we use today are offline and online data processing.

- **Offline data processing:** You only want to obtain the analytical results of data without a major concern about the time it takes. For example, in a batch data processing scenario, you receive data from OSS and output processing results to OSS, using MapReduce, Hive, Pig, and Spark.
- **Online data processing:** You want to obtain the analytical results of data with a strict requirement on the time it takes, such as real-time streaming data processing. Deeply integrated with Spark MLlib, GrapX, and SQL, Spark Streaming can be used to process streaming messages.

This section describes how to run an offline job called word count in E-MapReduce.

Process

OSS -> EMR -> Hadoop MapReduce

This process includes two steps:

1. Store data to OSS.
2. Read data from OSS and analyze the data by using E-MapReduce.

Prerequisites

- The following steps are performed in a Windows system. You need to ensure that Maven and Java have been installed and configured properly into your system.

- You can use E-MapReduce to automatically create a Hadoop cluster. For more information, see [Create a cluster](#).
- **EMR Version:** EMR-3.12.1
- **Cluster Type:** HADOOP
- **Software:** HDFS2.7.2, YARN2.7.2, Hive2.3.3, Ganglia3.7.2, Spark2.3.1, HUE4.1.0, Zeppelin0.8.0, Tez0.9.1, Sqoop1.4.7, Pig0.14.0, ApacheDS2.0.0, and Knox0.13.0
- **The network type of this Hadoop cluster is VPC in the China (Hangzhou) region.** The master instance group is configured with a public IP and an internal network IP. The high availability mode is set to No (a non-HA mode). The following figure shows the details.

| Cluster | | | | | | | | | | | | | |
|--|--|--|---|---------------------|--------|-----------|-------------|------------|------------------------|----------|--------------|--------------|---------------------|
| Name: dtplus_docs ID: C-DC57F7C835A178CD Region: cn-hangzhou Start Time: 2018-11-13 10:28:29 | Software Configuration: I/O Optimization: Yes High Availability: No Security Mode: Standard | Billing Method: Pay-As-You-Go Current Status: Idle Runtime: 1 Hours1 Minutes46 Seconds | Bootstrap Operation/Software Configuration: EMR-3.14.0 ECS Role: AliyunEmrEcsDefaultRole | | | | | | | | | | |
| Software | | Network | | | | | | | | | | | |
| EMR Version: EMR-3.14.0 Cluster Type: HADOOP Software: HDFS2.7.2 / YARN2.7.2 / Hive2.3.3 / Ganglia3.7.2 / Spark2.3.1 / HUE4.1.0 / Tez0.9.1 / Sqoop1.4.7 / Pig0.14.0 / ApacheDS2.0.0 / Knox0.13.0 | | Region ID: cn-hangzhou-f Network Type: vpc Security Group ID: sg-bp1344g9n4g1n4p4u0l0 VPC/VSwitch: vpc-bp1344g9n4g1n4p4u0l0 / vsw-bp1344g9n4g1n4p4u0l0 | | | | | | | | | | | |
| Host | | Master Instance Group | | | | | | | | | | | |
| Master Instance Group(MASTER) Pay-As-You-Go Hosts: 1 CPU: 4 Cores Memory: 8GB Data Disk Type: SSD Disk80GB*1 Disks | | <table><tr><th>ECS ID</th><th>组件部署状态</th><th>Public IP</th><th>Intranet IP</th><th>Created At</th></tr><tr><td>i-bp19ibpdre8wylu27ogp</td><td>● Normal</td><td>47.110.64.34</td><td>192.168.1.20</td><td>2018-11-13 10:28:35</td></tr></table> | | ECS ID | 组件部署状态 | Public IP | Intranet IP | Created At | i-bp19ibpdre8wylu27ogp | ● Normal | 47.110.64.34 | 192.168.1.20 | 2018-11-13 10:28:35 |
| ECS ID | 组件部署状态 | Public IP | Intranet IP | Created At | | | | | | | | | |
| i-bp19ibpdre8wylu27ogp | ● Normal | 47.110.64.34 | 192.168.1.20 | 2018-11-13 10:28:35 | | | | | | | | | |
| Core Instance Group(CORE) Pay-As-You-Go Hosts: 2 CPU: 4 Cores Memory: 8GB Data Disk Type: Ultra Disk80GB*4 Disks | | | | | | | | | | | | | |

Procedures

1. Download sample code to your local disk.

Open git bash in your system and execute the clone command as follows.

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

Execute the `mvn install` command to compile the code.

2. For more information about how to create a bucket, see [Create a bucket](#).



Note:

You must create a bucket and an E-MapReduce cluster in the same region.

3. Upload jar packages and resource files

- a. Log on to the [OSS console](#) and click the Files tab.
- b. Click Upload to upload resources files in the `aliyun - emapreduce - demo / resources` directory and jar packages in the `aliyun - emapreduce - demo / target` directory.

4. Create a workflow project

For more information, see [Workflow project management](#).

5. Create a job

For more information, see [Edit jobs](#). Take a MapReduce job as an example.

New Job



| | |
|----------------|--|
| * Project: | <input type="text" value="es_test_pro03"/> |
| * Folder: | <input type="text" value="JOB/"/> |
| * Name: | <input type="text"/> |
| * Description: | <div></div> |
| * Type | <div>Shell</div> |

OK

Cancel

6. After you configure a job, click Run. The following figure shows the details.

- For more information about how to use OSS, see [#unique_7](#).
- For more information about how to configure jobs, see the *job* section of the E-MapReduce User Guide.



Note:

- If the OSS output URI already exists, an error occurs when you execute a job.
- When you click the Insert an OSS UNI button and select OSSREF as a File Prefix , E-MapReduce downloads OSS files to your cluster and add these files to a specified classpath.
- Currently, only OSS Standard storage is supported for all operations.

View logs

For more information about how to view logs of an execution plan, see [Connect to a cluster using SSH](#).

3 Submit Storm topologies to process data in Kafka on E-MapReduce

This topic describes how to deploy Storm clusters and Kafka clusters on E-MapReduce and run Storm topologies to consume data in Kafka.

Prepare the environment

The test is performed using EMR that is deployed in the China East 1 (Hangzhou) region. The version of EMR is 3.8.0. The component versions required for this test are as follows.

- Kafka: 2.11_1.0.0
- Storm: 1.0.1

In this topic, we use Alibaba Cloud E-MapReduce to create a Kafka cluster automatically. For more information, see [Create a cluster](#).

- Create a Hadoop cluster

Version Configuration

EMR Version: EMR-3.8.0

Cluster Type: ☒ Hadoop ☐ Kafka

Required Services:

ApacheDS (2.0.0) Knox (0.13.0) Hadoop YARN (2.7.2) Hadoop HDFS (2.7.2)

Ganglia (3.7.2) Zeppelin (0.7.1) HUE (3.12.0) Sqoop (1.4.6) Tez (0.8.4)

Pig (0.14.0) Spark (2.2.1) Hive (2.3.2)

Optional Services:

Flink (1.4.0) Impala (2.10.0) HAS (1.1.0) Phoenix (4.10.0)

Zookeeper (3.4.11) Oozie (4.2.0) Storm (1.0.1) Presto (0.188) HBase (1.1.1)

Click to Choose

High Security Mode: ☐

Enable Custom Setting: ☐

Next

- Create a Kafka cluster

Version Configuration

EMR Version:

Cluster Type: ☐ Hadoop ☒ Kafka

Required Services:

Optional Services:

Click to Choose

High Security Mode: ☐ ☒

Enable Custom Setting: ☐ ☒

[Next](#)

**Note:**

- If you choose classic network as the network type, put the Hadoop cluster and the Kafka cluster in the same security group to save time for configuring connections between instances.
- If you choose VPC as the network type, put the Hadoop cluster and the Kafka cluster in the same VPC and the same security group to save time for configuring a VPC peering connection.
- If you are familiar with networking and security groups for ECS, you can create configurations as needed.

- Configure the environment for Storm

Consuming Kafka data fails if you run Storm topologies in the initial environment. To avoid such failures, you need to install the following dependencies for the Storm environment:

- [curator-client](#)
- [curator-framework](#)
- [curator-recipes](#)
- [json-simple](#)
- [metrics-core](#)
- [scala-library](#)
- [zookeeper](#)
- [commons-cli](#)
- [commons-collections](#)
- [commons-configuration](#)
- [htrace-core](#)
- [jcl-over-slf4j](#)
- [protobuf-java](#)
- [guava](#)
- [hadoop-common](#)
- [kafka-clients](#)
- [kafka](#)
- [storm-hdfs](#)
- [storm-kafka](#)

These dependencies have been tested. If you need additional dependencies, perform the following operations to add them to the lib folder of Storm.

```
[hadoop@emr-header-1 ~]$ ll
total 8524
-rw-rw-r-- 1 hadoop hadoop 52988 Jun 14 2015 commons-cli-1.3.1.jar
-rw-rw-r-- 1 hadoop hadoop 588337 Nov 13 2015 commons-collections-3.2.2.jar
-rw-rw-r-- 1 hadoop hadoop 298829 Feb 5 2009 commons-configuration-1.6.jar
-rw-r--r-- 1 root root 73448 Feb 9 14:01 curator-client-2.10.0.jar
-rw-r--r-- 1 root root 195437 Feb 9 14:01 curator-framework-2.10.0.jar
-rw-r--r-- 1 root root 281476 Feb 9 14:01 curator-recipes-2.10.0.jar
-rw-rw-r-- 1 hadoop hadoop 31212 Apr 19 2014 htrace-core-3.0.4.jar
-rw-rw-r-- 1 hadoop hadoop 17289 Jun 11 2012 jcl-over-slf4j-1.6.6.jar
-rw-rw-r-- 1 hadoop hadoop 16046 Aug 13 2009 json-simple-1.1.jar
-rw-rw-r-- 1 hadoop hadoop 82123 Nov 27 2012 metrics-core-2.2.0.jar
-rw-rw-r-- 1 hadoop hadoop 533455 Mar 8 2013 protobuf-java-2.5.0.jar
-rw-r--r-- 1 root root 5745606 Feb 9 14:01 scala-library-2.11.7.jar
-rw-rw-r-- 1 hadoop hadoop 792964 Feb 24 2014 zookeeper-3.4.6.jar
[hadoop@emr-header-1 ~]$ pwd
/home/hadoop
[hadoop@emr-header-1 ~]$ sudo cp ./usr/lib/storm-current/lib/
```

You need to perform the preceding operations on each node in the Hadoop cluster. After the operations are complete, restart Storm in the E-MapReduce console as shown in the following figure.

Status **Health Check**

| Services | Monitoring Data |
|---|--------------------|
| <div>Normal</div> <div>HDFS</div> <div>①</div> <div>Actions ▼</div> | cpu_idle(%) |
| <div>Normal</div> <div>YARN</div> <div></div> <div>Actions ▼</div> | |
| <div>Normal</div> <div>Hive</div> <div></div> <div>Actions ▼</div> | |
| <div>Normal</div> <div>Ganglia</div> <div></div> <div>Actions ▼</div> | |
| <div>Normal</div> <div>ZooKeeper</div> <div></div> <div>Actions ▼</div> | |
| <div>Normal</div> <div>Spark</div> <div></div> <div>Actions ▼</div> | ty_max_used(%) |
| <div>Normal</div> <div>Hue</div> <div></div> <div>Actions ▼</div> | |
| <div>Normal</div> <div>Tez</div> <div></div> <div>Actions ▼</div> | |
| <div>Normal</div> <div>Sqoop</div> <div></div> <div>Actions ▼</div> | |
| <div>Normal</div> <div>Pig</div> <div></div> <div>Actions ▼</div> | |
| <div>Normal</div> <div>Storm</div> <div>→</div> <div>Actions ▼</div> | |
| <div>Normal</div> <div>HAProxy</div> <div></div> <div>Actions ▼</div> | |
| <div>Normal</div> <div>ApacheDS</div> <div></div> <div>Actions ▼</div> | |

CONFIGURE All Components
 START All Components
 STOP All Components
 RESTART All Components
 RESTART Logviewer
 RESTART Nimbus
 RESTART Supervisor
 RESTART UI

You can view operation logs to check the status of Storm:

| Operation Logs | | | | | | | Refresh |
|----------------|------------------|---------------------|--------------|------------|--------------|---------|---------|
| ID | Operation | Start Time | Duration (s) | Status | Progress (%) | Remarks | Manage |
| 23726 | START STORM L... | 2018-11-13 16:10:21 | 88 | ✓ Succe... | 100 | ok | |
| 23725 | RESTART STOR... | 2018-11-13 16:09:57 | 102 | ✓ Succe... | 100 | ok | |

Create Storm topologies and Kafka topics

- E-MapReduce provides sample code that you can use directly. The links are as follows:

- [e-mapreduce-demo](#)
- [e-mapreduce-sdk](#)

- Write data to topics

1. Log on to the Kafka cluster.
2. Create a test topic with 10 partitions and 2 replicas.

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

3. Write 100 records of data to the test topic.

```
/usr/lib/kafka-current/bin/kafka-producer-perf
-test.sh --num-records 100 --throughput 10000 --
record-size 1024 --producer-props bootstrap.servers =
emr-worker-1: 9092 --topic test
```



Note:

The preceding command is run on the `emr-header-1` node in the Kafka cluster. You can also run the command on client nodes.

- Run a Storm topology.

Log on to the Hadoop cluster, compile the project and copy the `examples-1-1-shaded.jar` file that under `target/shaded` directory to the `emr-header-1` node. In this example, the file is stored in the HDFS root directory. Run the following command to submit the topology:

```
/usr/lib/storm-current/bin/storm jar examples-1-1-
-shaded.jar com.aliyun.emr.example.storm.StormKafka
```

```
Sample    test    aaa . bbb . ccc . ddd    hdfs :// emr - header - 1 :
9000      sample
```

- View the running status of a topology
- View the running status of Storm

You can use the Web UI to view the services on a cluster in the following ways:

- With Knox. For more information, see [Knox instructions](#).
- Use SSH. For more information, see [Use SSH to log on to a cluster](#).

In this topic, we use SSH to access the Web UI. The endpoint is `http :// localhost : 9999 / index . html`. You can see the topology that we have submitted. Click the topology to view the running logs:

Topology actions

Topology stats

| Window | Emitted | Transferred | Complete latency (ms) | Acked | Failed |
|-------------|---------|-------------|-----------------------|-------|--------|
| 10m 0s | 40 | 0 | 0 | 0 | |
| 3h 0m 0s | 640 | 400 | 22.200 | 100 | |
| 1d 0h 0m 0s | 640 | 400 | 22.200 | 100 | |
| All time | 640 | 400 | 22.200 | 100 | |

Spouts (All time)

| Search: <input type="text"/> | | | | | | | | | | | |
|------------------------------|-----------|-------|---------|-------------|-----------------------|-------|--------|------------|------------|------------|------------|
| ID | Executors | Tasks | Emitted | Transferred | Complete latency (ms) | Acked | Failed | Error Host | Error Port | Last error | Error Time |
| spout | 1 | 1 | 280 | 220 | 22.200 | 100 | 0 | | | | |

Showing 1 to 1 of 1 entries

Bolts (All time)

Search:

| Id | Executors | Tasks | Emitted | Transferred | Capacity (last 10m) | Execute latency (ms) | Executed | Process latency (ms) | Acked | Failed | Error Host | Error Port | Last error | Error Time |
|---------|-----------|-------|---------|-------------|---------------------|----------------------|----------|----------------------|-------|--------|------------|------------|------------|------------|
| __acker | 1 | 1 | 180 | 80 | 0.000 | 0.000 | 200 | 0.000 | 200 | 0 | | | | |
| bolt | 1 | 1 | 180 | 100 | 0.000 | 0.400 | 100 | 0.200 | 100 | 0 | | | | |

Showing 1 to 2 of 2 entries

- View the output files in HDFS

■ View the output files in HDFS.

```
[ root @ emr - header - 1 ~]# hadoop fs - ls / foo /
-rw-r--r-- 3 root hadoop 615000 2018-02-11 13:37 / foo / bolt - 2 - 0 - 1518327393 692 . txt
-rw-r--r-- 3 root hadoop 205000 2018-02-11 13:37 / foo / bolt - 2 - 0 - 1518327441 777 . txt
[ root @ emr - header - 1 ~]# hadoop fs - cat / foo / bolt - 2 - 0 - 1518327441 777 . txt | wc -l
200
```

■ Write 120 records of data to the test topic in Kafka.

```
[ root @ emr - header - 1 ~]# / usr / lib / kafka - current / bin / kafka - producer - perf - test . sh -- num - records 120 -- throughput 10000 -- record - size 1024 -- producer - props bootstrap . servers = emr - worker - 1 : 9092 -- topic test
```

```
120 records sent , 816 . 326531 records / sec ( 0 . 80
MB / sec ), 35 . 37 ms avg latency , 134 . 00 ms
max latency , 35 ms 50th , 39 ms 95th , 41 ms
99th , 134 ms 99 . 9th .
```

■ Output the line number of the HDFS file.

```
[ root @ emr - header - 1 ~]# hadoop fs - cat / foo /
bolt - 2 - 0 - 1518327441 777 . txt | wc - l
320
```

Summary

We have successfully deployed a Storm cluster and a Kafka cluster on E-MapReduce , run a Storm topology and consumed Kafka data. E-MapReduce also supports the Spark streaming and the Flink components, which can run in Hadoop clusters and process Kafka data.



Note:

E-MapReduce does not provide the Storm cluster option. Therefore, we have created a Hadoop cluster and have installed the Storm components. If you do not need to use other components, you can easily disable them in the E-MapReduce console. Then a Hadoop cluster is equivalent to a Storm cluster.

4 Use ES-Hadoop on E-MapReduce

ES-Hadoop is a tool used to connect the Hadoop ecosystem provided by Elasticsearch (ES). It enables users to use tools such as MapReduce (MR), Spark, and Hive to process data in ES (ES-Hadoop also supports taking a snapshot of ES indices and storing it in HDFS, which is not discussed in this topic).

Background

We know that the advantage of the Hadoop ecosystem is processing large data sets. But the disadvantage is also obvious: interactive analysis can be delayed. ES is adept at many types of queries, especially ad-hoc queries. Subsecond response time has been reached. ES-Hadoop has combined both advantages. With ES-Hadoop, users only need to make small changes to the code for quickly processing data stored in ES. ES also provides acceleration.

ES-Hadoop uses ES as the data source of data processing engines, such as MR, Spark, and Hive. ES plays the role of storage in architectures where compute and storage are separated. This is the same for other data sources of MR, Spark, and Hive. But ES has faster data filtering ability compared with other data sources. This ability is one of the most critical abilities of an analytics engine.

EMR has already integrated with ES-Hadoop. Users can use ES-Hadoop directly without any configurations. The following examples introduce ES-Hadoop on EMR.

Preparation

ES can automatically create indices and identify data types based on input data. In some cases, this feature is helpful, by avoiding many actions by users. However, it also cause problems. The biggest problem is that sometimes the data types identified by ES are not correct. For example, we define a field called *age*. The data type of this column is INT but it may be identified as LONG in the ES index. Users need to convert data types when performing some specified actions. We recommend that you create indices manually to avoid such problems.

In the following examples, we use the *company* index and the *employees* ' type (you can consider an ES index as a database and a type as a table in the database). This type defines four fields (field types are defined by ES).

```
{
```

```

    " id ": long ,
    " name ": text ,
    " age ": integer ,
    " birth ": date
  }

```

Run the following commands to create an index in Kibana (you can also use cURL commands):

```

PUT    company
{
  " mappings ": {
    " employees ": {
      " properties ": {
        " id ": {
          " type ": " long "
        },
        " name ": {
          " type ": " text ",
          " fields ": {
            " keyword ": {
              " type ": " keyword ",
              " ignore_above ": 256
            }
          }
        },
        " birth ": {
          " type ": " date "
        },
        " addr ": {
          " type ": " text "
        }
      }
    }
  },
  " settings ": {
    " index ": {
      " number_of_shards ": " 5 ",
      " number_of_replicas ": " 1 "
    }
  }
}

```



Note:

Specify the index parameters in settings as needed. This step is optional.

Prepare a file where each row is a JSON object as follows:

```

{" id ": 1 , " name ": " zhangsan ", " birth ": " 1990 - 01 - 01 ", "
  addr ": " No . 969 , wenyixi Rd , yuhang , hangzhou "}
{" id ": 2 , " name ": " lisi ", " birth ": " 1991 - 01 - 01 ", "
  addr ": " No . 556 , xixi Rd , xihu , hangzhou "}
{" id ": 3 , " name ": " wangwu ", " birth ": " 1992 - 01 - 01 ", "
  addr ": " No . 699 wangshang Rd , binjiang , hangzhou "}

```

Save the file to the specified directory in HDFS (for example, `/ es - hadoop / employees . txt`).

Mapreduce

In the following example, we read the JSON files in the `/es - hadoop` directory in HDFS and write each row in the JSON files into ES as a document. Writing is finished in the map stage through `EsOutputFormat`.

Use the following options to set ES.

- `es.nodes`: ES nodes. The format is `host:port`. For ES hosted on Alibaba Cloud, set the value to the endpoint of ES provided by Alibaba Cloud.
- `es.net.http.auth.user`: Username.
- `es.net.http.auth.pass`: Password.
- `es.nodes.wan.only`: For ES hosted on Alibaba Cloud, set the value to `true`.
- `es.resource`: The indices and types of ES.
- `es.input.json`: If the input file is in JSON format, set the value to `true`. Otherwise, you need to parse the input data using the `map()` function and output the corresponding Writable class.



Notice:

Disable speculative execution for map tasks and reduce tasks

```
package    com . aliyun . emr ;

import    java . io . IOException ;
import    org . apache . hadoop . conf . Configuration ;
import    org . apache . hadoop . fs . Path ;
import    org . apache . hadoop . io . NullWritable ;
import    org . apache . hadoop . io . Text ;
import    org . apache . hadoop . mapreduce . Job ;
import    org . apache . hadoop . mapreduce . Mapper ;
import    org . apache . hadoop . mapreduce . lib . input .
FileInputFormat ;
import    org . apache . hadoop . mapreduce . lib . input .
TextInputFormat ;
import    org . apache . hadoop . util . GenericOptionsParser ;
import    org . apache . hadoop . util . Tool ;
import    org . apache . hadoop . util . ToolRunner ;
import    org . elasticsearch . hadoop . mr . EsOutputFormat ;

public class Test implements Tool {

    private Configuration conf ;

    @Override
    public int run ( String [] args ) throws Exception {

        String [] otherArgs = new GenericOptionsParser ( conf ,
args ). getRemainingArgs () ;

        conf . setBoolean ( " mapreduce . map . speculative " , false ) ;
```

```

        conf . setBoolean ( " mapreduce . reduce . speculativ e " , false
    );
    conf . set ( " es . nodes " , "< your_es_ho st >: 9200 " );
    conf . set ( " es . net . http . auth . user " , "< your_usern ame
>");
    conf . set ( " es . net . http . auth . pass " , "< your_passw ord
>");
    conf . set ( " es . nodes . wan . only " , " true " );
    conf . set ( " es . resource " , " company / employees " );
    conf . set ( " es . input . json " , " yes " );

    Job job = Job . getInstanc e ( conf );
    job . setInputFo rmatClass ( TextInputF ormat . class );
    job . setOutputF ormatClass ( EsOutputFo rmat . class );
    job . setMapOutp utKeyClass ( NullWritab le . class );
    job . setMapOutp utValueCla ss ( Text . class );
    job . setJarByCl ass ( Test . class );
    job . setMapperC lass ( EsMapper . class );

    FileInputF ormat . setInputPa ths ( job , new Path (
otherArgs [ 0 ]));

    return job . waitForCom pletion ( true ) ? 0 : 1 ;
}

@Override
public void setConf ( Configurati on conf ) {
    this . conf = conf ;
}

@Override
public Configurati on getConf () {
    return conf ;
}

public static class EsMapper extends Mapper < Object ,
Text , NullWritab le , Text > {
    private Text doc = new Text ();

    @Override
    protected void map ( Object key , Text value , Context
context ) throws IOExceptio n , Interrupte dException {
        if ( value . getLength () > 0 ) {
            doc . set ( value );
            context . write ( NullWritab le . get () , doc );
        }
    }
}

public static void main ( String [] args ) throws
Exception {
    int ret = ToolRunner . run ( new Test () , args );
    System . exit ( ret );
}
}

```

Compile and package the code into a JAR file called *mr - test . jar* . Submit it to an instance that has installed an EMR client program (such as a gateway, or any node in an EMR cluster).

Run the following commands on any node that has installed an EMR client to run the MapReduce program:

```
hadoop jar mr - test . jar com . aliyun . emr . Test -
Dmapreduce . job . reduces = 0 - libjars mr - test . jar / es -
hadoop
```

At this point, writing data to ES has finished. You can query the written data through Kibana (or by using the cURL commands).

```
GET
{
  " query ": {
    " match_all ": {}
  }
}
```

Spark

In this example, we write data to an index in ES using Spark instead of MapReduce. Spark persists a resilient distributed dataset (RDD) to ES using the JavaEsSpark class. Users also need to use the options mentioned above in the MapReduce section to set ES.

```
package com . aliyun . emr ;

import java . util . Map ;
import java . util . concurrent . atomic . AtomicInte ger ;
import org . apache . spark . SparkConf ;
import org . apache . spark . SparkConte xt ;
import org . apache . spark . api . java . JavaRDD ;
import org . apache . spark . api . java . function . Function ;
import org . apache . spark . sql . Row ;
import org . apache . spark . sql . SparkSessi on ;
import org . elasticsea rch . spark . rdd . api . java .
JavaEsSpark ;
import org . spark_proj ect . guava . collect . ImmutableM ap ;

public class Test {

    public static void main ( String [] args ) {
        SparkConf conf = new SparkConf () ;
        conf . setAppName ( " Es - test " ) ;
        conf . set ( " es . nodes " , "< your_es_host >: 9200 " ) ;
        conf . set ( " es . net . http . auth . user " , "< your_username
>");
        conf . set ( " es . net . http . auth . pass " , "< your_password
>");
        conf . set ( " es . nodes . wan . only " , " true " ) ;

        SparkSessi on ss = new SparkSessi on ( new SparkConte
xt ( conf ) ) ;
        final AtomicInte ger employeesN o = new AtomicInte ger
( 0 ) ;
        JavaRDD < Map < Object , ? >> javaRDD = ss . read () . text ( "
hdfs :// emr - header - 1 : 9000 / es - hadoop / employees . txt " )
```

```

    . javaRDD (). map (( Function < Row , Map < Object , ?
>>) row -> ImmutableMap . of (" employees " + employeesN o .
getAndAdd ( 1 ), row . mkString ());

    JavaEsSpark . saveToEs ( javaRDD , " company / employees ");
}
}

```

Package the code in a JAR file called `spark-test.jar`. Run the following command to write data:

```

spark - submit -- master yarn -- class com . aliyun . emr .
Test spark - test . jar

```

After the task has finished, you can query the results through Kibana or the `cURL` commands.

In addition to `Spark RDD`, ES-Hadoop also provides a Spark SQL component to read and write ES data. For more information, see the [official website](#) of ES-Hadoop.

Hive

This example introduces SQL statements to read and write ES data through Hive.

First, run the `hive` command to enter CLI and create a table:

```

CREATE DATABASE IF NOT EXISTS company ;

```

Then create an external table that is stored in ES. Specify the option using `TBLPROPERTIES`.

```

CREATE EXTERNAL table IF NOT EXISTS employees (
  id BIGINT ,
  name STRING ,
  birth TIMESTAMP ,
  addr STRING
)
STORED BY ' org . elasticsearch . hadoop . hive . EsStorageH
andler '
TBLPROPERTIES (
  ' es . resource ' = ' tpcds / ss ',
  ' es . nodes ' = ' < your_es_host > ',
  ' es . net . http . auth . user ' = ' < your_username > ',
  ' es . net . http . auth . pass ' = ' < your_password > ',
  ' es . nodes . wan . only ' = ' true ',
  ' es . resource ' = ' company / employees '
);

```



Note:

We set the data type of the birth columns to `TIMESTAMP` in the Hive table. In ES, we set it to `DATE`. This is because Hive and EC handle data types differently. Parsing of converted date data can fail when Hive writes data to ES. In contrast, parsing of

returned data can also fail when Hive reads ES data. For more information, click [here](#).

Insert some data into the table:

```
INSERT INTO TABLE employees VALUES ( 1 , " zhangsan ", " 1990 - 01 - 01 ", " No . 969 , wenyixi Rd , yuhang , hangzhou " );
INSERT INTO TABLE employees VALUES ( 2 , " lisi ", " 1991 - 01 - 01 ", " No . 556 , xixi Rd , xihu , hangzhou " );
INSERT INTO TABLE employees VALUES ( 3 , " wangwu ", " 1992 - 01 - 01 ", " No . 699 wangshang Rd , binjiang , hangzhou " );
```

Execute queries to view the results:

```
SELECT * FROM employees LIMIT 100 ;
OK
1      zhangsan      1990 - 01 - 01      No . 969 , wenyixi Rd ,
yuhang , hangzhou
2      lisi          1991 - 01 - 01      No . 556 , xixi Rd , xihu
, hangzhou
3      wangwu        1992 - 01 - 01      No . 699 wangshang Rd ,
binjiang , hangzhou
```

5 Use Mongo-Hadoop on E-MapReduce

Mongo-Hadoop is a component provided by MongoDB for Hadoop components to connect to MongoDB. Using Mongo-Hadoop is similar to using ES-Hadoop which is described in the previous topic. EMR has already integrated with Mongo-Hadoop. Users can directly use Mongo-Hadoop without any deployment configuration. This topic describes how to use Mongo-Hadoop using some examples.

Preparation

We use the same data model for the following examples:

```
{
  " id ": long ,
  " name ": text ,
  " age ": integer ,
  " birth ": date
}
```

We write data into the specified collection (similar to a table in a database) in a MongoDB database. Therefore, we need to first ensure that the collection exists in the MongoDB database. First, run the MongoDB client program on a client node that can access the MongoDB database. You may need to download the client program from the MongoDB website and install it. Take the connection to ApsaraDB for MongoDB as an example:

```
mongo --host dds-xxxxxxxxx.xxxxxxxxxx.x.mongodb.rds.aliyuncs.com:3717 --authenticationDatabase admin -u root -p 123456
```

The hostname of the MongoDB database is dds-xxxxxxxxxxxxxxxxxxxxxxxxx.mongodb.rds.aliyuncs.com. The port number is 3717. The actual port number depends on the MongoDB cluster. For an external MongoDB cluster that you have deployed on your own, the default port number is 27017. In this example, the password is set to 123456 using the -p option. Run the following commands in CLI to create a collection named employees in the company database:

```
> use company ;
> db.createCollection("employees")
```

Prepare a file where each row is a JSON object as follows.

```
{" id ": 1 , " name ": " zhangsan " , " birth ": " 1990 - 01 - 01 " , " addr ": " No . 969 , wenyixi Rd , yuhang , hangzhou " }
```



```
{ "id": 2, "name": "lisi", "birth": "1991-01-01", "addr": "No. 556, Xixi Rd, Xihu, Hangzhou" }
{ "id": 3, "name": "wangwu", "birth": "1992-01-01", "addr": "No. 699 Wangshang Rd, Binjiang, Hangzhou" }
```

Save the file to the specified directory on HDFS (for example, the file path can be `/mongo-hadoop/employees.txt`).

Mapreduce

In the following example, we read JSON files in the `/mongo-hadoop` directory on HDFS and write each row in the JSON files as a document to the MongoDB database.

```
package com.aliyun.emr;

import com.mongodb.BasicDBObject;
import com.mongodb.hadoop.MongoOutputFormat;
import com.mongodb.hadoop.io.BSONWritable;
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.util.GenericOptionsParser;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

public class Test implements Tool {

    private Configuration conf;

    @Override
    public int run(String[] args) throws Exception {

        String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();

        conf.set("mongo.output.uri", "mongodb://<your_username>:<your_password>@dds-xxxxxxxxx.mongodb.rds.aliyuncs.com:3717/company.employees?authSource=admin");

        Job job = Job.getInstance(conf);
        job.setInputFormatClass(TextInputFormat.class);
        job.setOutputFormatClass(MongoOutputFormat.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClasses(BSONWritable.class);

        job.setJarByClass(Test.class);
        job.setMapperClass(MongoMapper.class);

        FileInputFormat.setInputPaths(job, new Path(otherArgs[0]));

        return job.waitForCompletion(true) ? 0 : 1;
    }
}
```

```

    }

    @Override
    public Configuration getConf () {
        return conf ;
    }

    @Override
    public void setConf ( Configuration conf ) {
        this . conf = conf ;
    }

    public static class MongoMapper extends Mapper < Object
    , Text , Text , BSONWritable > {

        private BSONWritable doc = new BSONWritable ();
        private int employeeNo = 1 ;
        private Text id ;

        @Override
        protected void map ( Object key , Text value , Context
        context ) throws IOException , InterruptedException {
            if ( value . getLength () > 0 ) {
                doc . setDoc ( BasicDBObject . parse ( value . toString
                ())) ;
                id = new Text (" employee " + employeeNo ++);
                context . write ( id , doc );
            }
        }
    }

    public static void main ( String [] args ) throws
    Exception {
        int ret = ToolRunner . run ( new Test (), args );
        System . exit ( ret );
    }
}

```

Compile and package the code into a JAR file called `mr - test . jar` . Run the following command:

```

hadoop jar mr - test . jar com . aliyun . emr . Test -
Dmapreduce . job . reduces = 0 - libjars mr - test . jar / mongo
- hadoop

```

After the execution is complete, you can view the results using the MongoDB client program:

```

> db . employees . find ();
{ "_id " : " employee1 " , " id " : 1 , " name " : " zhangsan " , "
  birth " : " 1990 - 01 - 01 " , " addr " : " No . 969 , wenyixi Rd
  , yuhang , hangzhou " }
{ "_id " : " employee2 " , " id " : 2 , " name " : " lisi " , " birth
  " : " 1991 - 01 - 01 " , " addr " : " No . 556 , xixi Rd , xihu
  , hangzhou " }

```

```
{ "_id" : " employee3 ", " id " : 3 , " name " : " wangwu ", "
  birth " : " 1992 - 01 - 01 ", " addr " : " No . 699   wangshang   Rd
,   binjiang ,   hangzhou " }
```

Spark

In this example, we write data to a MongoDB database using Spark instead of MapReduce.

```
package    com . aliyun . emr ;

import    com . mongodb . BasicDBObject ;
import    com . mongodb . hadoop . MongoOutputFormat ;
import    java . util . concurrent . atomic . AtomicInteger ;
import    org . apache . hadoop . conf . Configuration ;
import    org . apache . spark . SparkContext ;
import    org . apache . spark . api . java . JavaPairRDD ;
import    org . apache . spark . api . java . JavaRDD ;
import    org . apache . spark . api . java . function . Function ;
import    org . apache . spark . sql . Row ;
import    org . apache . spark . sql . SparkSession ;
import    org . bson . BSONObject ;
import    scala . Tuple2 ;

public    class    Test {

    public    static    void    main ( String [] args ) {

        SparkSession    ss = new    SparkSession ( new    SparkContext
xt ());

        final    AtomicInteger    employeeNo    = new    AtomicInteger (
0 );
        JavaRDD < Tuple2 < Object , BSONObject >> javaRDD =
            ss . read () . text ( " hdfs :// emr - header - 1 : 9000 /
mongo - hadoop / employees . txt " )
            . javaRDD () . map (( Function < Row , Tuple2 < Object ,
BSONObject >>) row -> {
                BSONObject    bson = BasicDBObject . parse ( row .
mkString ());
                return    new    Tuple2 <>(" employee " + employeeNo .
getAndAdd ( 1 ), bson );
            });

        JavaPairRDD < Object , BSONObject > documents = JavaPairRD
D . fromJavaRDD ( javaRDD );

        Configuration    outputConfig = new    Configuration ();
        outputConfig . set ( " mongo . output . uri ", " mongodb ://<
your_username >:< your_password >@dds - xxxxxxxxxxx xxxxxxxxxxx
x . mongodb . rds . aliyuncs . com : 3717 / company . employees ?
authSource = admin " );

        // It is saved as a " Hadoop file ." Actually , the
        data is written into the MongoDB database through
        the MongoOutputFormat class .
        documents . saveAsNewAPIHadoopFile (
            " file :/// this - is - completely - unused ",
            Object . class ,
            BSONObject . class ,
            MongoOutputFormat . class ,
            outputConfig
```

```
    );
  }
}
```

Package the code into a JAR file named `spark - test . jar` . Run the following command to write data.

```
spark - submit -- master    yarn -- class    com . aliyun . emr .
Test    spark - test . jar
```

After the writing has finished, you can use the MongoDB client to view the results.

Hive

This example describes how to use Hive to read and write data in MongoDB databases through SQL statements.

First, run the `hive` command to enter CLI mode and create a table:

```
CREATE DATABASE IF NOT EXISTS company ;
```

You need to create an external table that is stored in a MongoDB database. Before you do that, create a MongoDB collection named `employees` as described in the Preparation section.

Go back to CLI mode, execute the following SQL statements to create an external table. Connection to MongoDB is set through the `TBLPROPERTIES` clause.

```
CREATE EXTERNAL TABLE IF NOT EXISTS employees (
  id BIGINT ,
  name STRING ,
  birth STRING ,
  addr STRING
)
STORED BY ' com . mongodb . hadoop . hive . MongoStorageHandler
WITH SERDEPROPERTIES (' mongo . columns . mapping '=' {" id ":"
_id "}")
TBLPROPERTIES (' mongo . uri '=' mongodb ://< your_username >:<
your_password >@dds - xxxxxxxxxxxx xxxxxxxxxxxx x . mongodb . rds .
aliyuncs . com : 3717 / company . employees ? authSource = admin ');
```



Notice:

Values of the `id` column in Hive are mapped to values of the `_id` column in MongoDB through `SERDEPROPERTIES`. You can map column values as needed. Note that the data type of the `birth` column is set to `STRING`. The reason is that Hive and MongoDB handle `DATE` format differently. After Hive sends data in `DATE` format to MongoDB, `NULL` may be returned when the data is queried in Hive.

Insert some data into the table:

```
INSERT INTO TABLE employees VALUES ( 1 , " zhangsan ", " 1990 - 01 - 01 ", " No . 969 , wenyixi Rd , yuhang , hangzhou ");
INSERT INTO TABLE employees VALUES ( 2 , " lisi ", " 1991 - 01 - 01 ", " No . 556 , xixi Rd , xihu , hangzhou ");
INSERT INTO TABLE employees VALUES ( 3 , " wangwu ", " 1992 - 01 - 01 ", " No . 699 wangshang Rd , binjiang , hangzhou ");
```

Execute the following statement to see the results:

```
SELECT * FROM employees LIMIT 100 ;
OK
1      zhangsan      1990 - 01 - 01      No . 969 , wenyixi Rd ,
yuhang , hangzhou
2      lisi          1991 - 01 - 01      No . 556 , xixi Rd , xihu
, hangzhou
3      wangwu        1992 - 01 - 01      No . 699 wangshang Rd ,
binjiang , hangzhou
```

6 Deep learning with Analytics Zoo on E-MapReduce

This topic describes how to use Analytics Zoo to develop deep learning applications on Alibaba Cloud E-MapReduce.

Introduction

Analytics Zoo is an analytics and AI platform that unites Apache Spark and Intel BigDL into an integrated pipeline. It helps users develop deep learning applications based on big data and end-to-end pipelines.

System requirements

- JDK 8
- Spark cluster (Spark 2.x supported by EMR is recommended)
- Python 2.7(also Python 3.5 or Python 3.6), pip

Installation of Analytics Zoo

- The latest release of Analytics Zoo is 0.2.0.
- Installation for Scala users
 - Download the pre-build version.

You can download the [Pre-build version](#) from the Analytics Zoo page on GitHub.

- Build Analytics Zoo using the make-dist.sh script.

Install Apache Maven and set the environment variable MAVEN_OPTS as follows:

```
export MAVEN_OPTS="-Xmx2g -XX:ReservedCodeCacheSize=512m"
```

If you use ECS instances to compile code, we recommend that you modify the mirror of the Maven repository.

```
< mirror >
  < id > nexus - aliyun </ id >
  < mirrorOf > central </ mirrorOf >
  < name > Nexus aliyun </ name >
  < url > http://maven.aliyun.com/nexus/content/groups/public </ url >
```

```
</ mirror >
```

Download an [Analytics Zoo release](#). Extract the file, move to the corresponding directory, and run the following command:

```
bash make - dist . sh
```

After building Analytics Zoo, you can find a dist directory, which contains all the needed files to run an Analytics Zoo program. Use the following command to copy the files in the dist directory to the directory of the EMR software stack:

```
cp -r dist /usr/lib/analytics_zoo
```

- **Installation for Python users**

Analytics Zoo can be installed either with pip or without pip. When you install Analytics Zoo with pip, PySpark and BigDL are installed. This may cause a software conflict because PySpark has already been installed on the EMR cluster. To avoid such conflicts, install Analytics Zoo without pip.

- **Installation without pip**

First, you need to run the following command:

```
bash make - dist . sh
```

Change to the pyzoo directory and install Analytics Zoo:

```
python setup . py install
```

- **Setting environment variables**

After building Analytics Zoo, copy the dist directory to the directory of the EMR software stack and set the environment variable. Add the following lines to the `/etc/profile.d/analytics_zoo.sh` file.

```
export ANALYTICS_ZOO_HOME=/usr/lib/analytics_zoo
export PATH=$ANALYTICS_ZOO_HOME/bin:$PATH
```

You do not need to set SPARK_HOME because it has already been set on EMR.

Using Analytics Zoo

- Use Spark to train and test deep learning models.
- Use Analytics Zoo to do text classification. You can find the code and description on [GitHub](#). Download the required data as required. Submit the following commands:

```
spark - submit -- master yarn \
-- deploy - mode cluster -- driver - memory 8g \
-- executor - memory 20g -- class com . intel . analytics .
zoo . examples . textclassi fication . TextClassi fication \
/ usr / lib / analytics_ zoo / lib / analytics - zoo - bigdl_0 .
6 . 0 - spark_2 . 1 . 0 - 0 . 2 . 0 - jar - with - dependenci es
. jar -- baseDir / news
```

- You can log on to the instance of the Spark cluster through [ssh proxy](#) to view the status of the jobs.

Stages for All Jobs

Active Stages: 1
Pending Stages: 1
Completed Stages: 698
Skipped Stages: 293

Active Stages (1)

| Stage Id | Description | Submitted | Duration | Tasks: Succeeded/Total | Input | Output | Shuffle Read | Shuffle Write |
|----------|---|---|----------|------------------------|-------|--------|--------------|---------------|
| 1392 | reduce at DistriOptimizer.scala:320 | +details (kill) 2018/09/12 12:21:47 | Unknown | 0/2 | | | | |

Pending Stages (1)

| Stage Id | Description | Submitted | Duration | Tasks: Succeeded/Total | Input | Output | Shuffle Read | Shuffle Write |
|----------|---|----------------------------------|----------|------------------------|-------|--------|--------------|---------------|
| 1391 | coalesce at DataSet.scala:361 | +details Unknown | Unknown | 0/4 | | | | |

Completed Stages (698)

Page: [1](#) [2](#) [3](#) [4](#) [5](#) [6](#) [7](#) [>](#)

7 Pages. Jump to . Show items in a page. [Go](#)

| Stage Id | Description | Submitted | Duration | Tasks: Succeeded/Total | Input | Output | Shuffle Read | Shuffle Write |
|----------|---|--|----------|------------------------|--------|--------|--------------|---------------|
| 1390 | count at DistriOptimizer.scala:369 | +details 2018/09/12 12:21:47 | 12 ms | 2/2 | 4.5 MB | | | |
| 1388 | reduce at DistriOptimizer.scala:320 | +details 2018/09/12 12:21:46 | 0.9 s | 2/2 | 5.6 GB | | | |
| 1386 | count at DistriOptimizer.scala:369 | +details 2018/09/12 12:21:46 | 12 ms | 2/2 | 4.5 MB | | | |
| 1384 | reduce at DistriOptimizer.scala:320 | +details 2018/09/12 12:21:45 | 1.0 s | 2/2 | 5.6 GB | | | |
| 1382 | count at DistriOptimizer.scala:369 | +details 2018/09/12 12:21:45 | 11 ms | 2/2 | 4.5 MB | | | |
| 1380 | reduce at DistriOptimizer.scala:320 | +details 2018/09/12 12:21:44 | 0.9 s | 2/2 | 5.6 GB | | | |
| 1378 | count at DistriOptimizer.scala:369 | +details 2018/09/12 12:21:44 | 11 ms | 2/2 | 4.5 MB | | | |
| 1376 | reduce at DistriOptimizer.scala:320 | +details 2018/09/12 12:21:43 | 1.0 s | 2/2 | 5.6 GB | | | |
| 1374 | count at DistriOptimizer.scala:369 | +details 2018/09/12 12:21:43 | 11 ms | 2/2 | 4.5 MB | | | |

You can also view the accuracy of each epoch through logs.

```
INFO optim . DistriOpti mizer $: [ Epoch 2 9600 / 15107 ]
[ Iteration 194 ][ Wall Clock 193 . 266637037s ] Trained
128 records in 0 . 958591653 seconds . Throughput is
133 . 52922 records / second . Loss is 0 . 74216986 .
INFO optim . DistriOpti mizer $: [ Epoch 2 9728 / 15107 ]
[ Iteration 195 ][ Wall Clock 194 . 224064816s ] Trained
128 records in 0 . 957427779 seconds . Throughput is
133 . 69154 records / second . Loss is 0 . 51025534 .
INFO optim . DistriOpti mizer $: [ Epoch 2 9856 / 15107 ]
[ Iteration 196 ][ Wall Clock 195 . 189488678s ] Trained
128 records in 0 . 965423862 seconds . Throughput is
132 . 58424 records / second . Loss is 0 . 553785 .
INFO optim . DistriOpti mizer $: [ Epoch 2 9984 / 15107 ]
[ Iteration 197 ][ Wall Clock 196 . 164318688s ] Trained
```



```
128 records in 0.97483001 seconds. Throughput is
131.30495 records / second. Loss is 0.5517549.
```

- Use PySpark and Jupyter to train deep learning models on Analytics Zoo.

- Install Jupyter.

```
pip install jupyter
```

- Run the following command to start Jupyter.

```
jupyter - with - zoo . sh
```

- We recommend that you use the pre-defined Wide And Deep Learning models provided by Analytics Zoo, for more information please refer to [GitHub](#).

1. Import data.

localhost:8889/notebooks/Untitled1.ipynb?kernel_name=python2

The screenshot shows a Jupyter Notebook titled 'Untitled1' with a toolbar and a code editor. The code in the editor is as follows:

```
In [2]: from zoo.models.recommendation import *
        from zoo.models.recommendation.utils import *
        from zoo.common.nncontext import init_nncontext
        import os
        import sys
        import datetime as dt
        from bigdl.dataset.transformer import *
        from bigdl.dataset.base import *
        from bigdl.nn.criterion import *
        from bigdl.optim.optimizer import *
        from bigdl.util.common import *
        import matplotlib

        matplotlib.use('agg')
        import matplotlib.pyplot as plt
        %pylab inline

        Populating the interactive namespace from numpy and matplotlib

In [3]: sc = init_nncontext("WideAndDeep Example")

In [5]: from bigdl.dataset import movielens
        movielens_data = movielens.get_id_ratings("/tmp/movielens/")
        min_user_id = np.min(movielens_data[:,0])
        max_user_id = np.max(movielens_data[:,0])
        min_movie_id = np.min(movielens_data[:,1])
        max_movie_id = np.max(movielens_data[:,1])
        rating_labels = np.unique(movielens_data[:,2])

        print(movielens_data.shape)
        print(min_user_id, max_user_id, min_movie_id, max_movie_id, rating_labels)
```

2. Build a model and create an optimizer.

```

In [10]: wide_n_deep = WideAndDeep(5, column_info, "wide_n_deep")

         creating: createZooWideAndDeep

In [11]: # Create an Optimizer
         batch_size = 8000

         optimizer = Optimizer(
             model=wide_n_deep,
             training_rdd=train_data,
             criterion=ClassNLLCriterion(),
             optim_method=Adam(learningrate = 0.001, learningrate_decay=0.00005),
             end_trigger=MaxEpoch(10),
             batch_size=batch_size)

         # Set the validation logic
         optimizer.set_validation(
             batch_size=batch_size,
             val_rdd=test_data,
             trigger=EveryEpoch(),
             val_method=[Top1Accuracy(), Loss(ClassNLLCriterion())])
         )
         log_dir='/tmp/bigdl_summaries/'
         app_name='wide_n_deep-'+dt.datetime.now().strftime("%Y%m%d-%H%M%S")
         train_summary = TrainSummary(log_dir=log_dir,
                                     app_name=app_name)
         val_summary = ValidationSummary(log_dir=log_dir,
                                       app_name=app_name)
         optimizer.set_train_summary(train_summary)
         optimizer.set_val_summary(val_summary)
         print("saving logs to %s" % (log_dir + app_name))

```

3. Start the training process.

```

In [12]: %%time
         # Boot training process
         optimizer.optimize()
         print("Optimization Done.")

Optimization Done.
CPU times: user 85.9 ms, sys: 16.7 ms, total: 103 ms
Wall time: 2min 52s

```

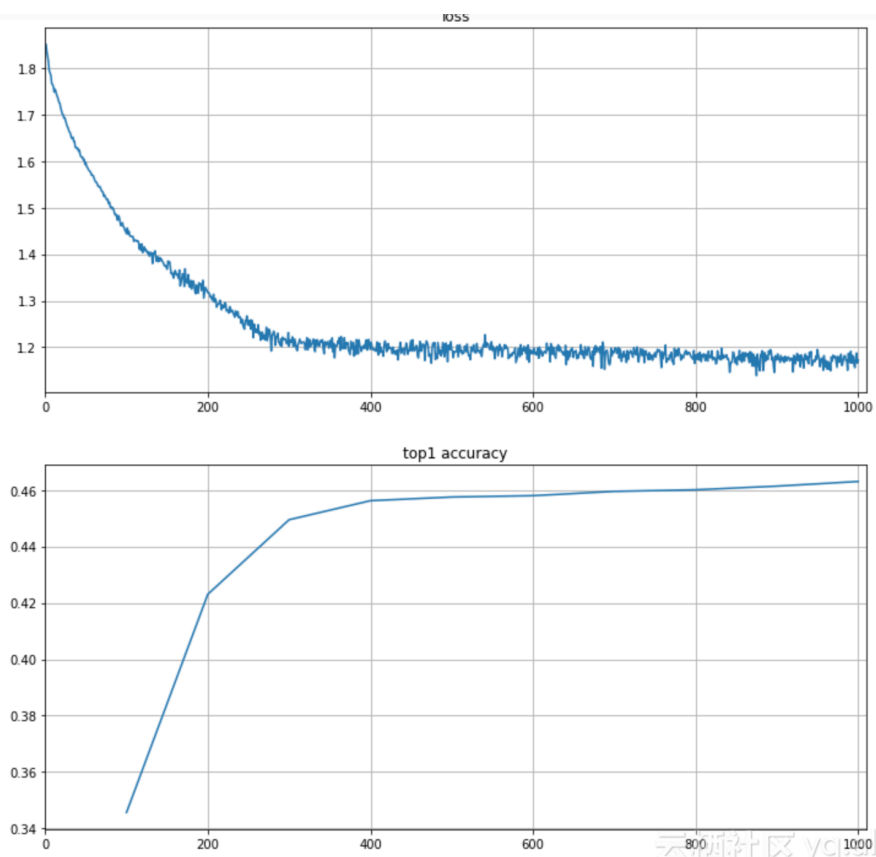
4. View training results.

```

In [16]: loss = np.array(train_summary.read_scalar("Loss"))
         topl = np.array(val_summary.read_scalar("Top1Accuracy"))

         plt.figure(figsize = (12,12))
         plt.subplot(2,1,1)
         plt.plot(loss[:,0],loss[:,1],label='loss')
         plt.xlim(0,loss.shape[0]+10)
         plt.grid(True)
         plt.title("loss")
         plt.subplot(2,1,2)
         plt.plot(topl[:,0],topl[:,1],label='topl')
         plt.xlim(0,loss.shape[0]+10)
         plt.title("topl accuracy")
         plt.grid(True)

```



7 Adaptive execution of Spark SQL

Spark SQL of Alibaba Cloud Elastic MapReduce (E-MapReduce) 3.13.0 supports adaptive execution. It is used to set the number of reduce tasks automatically, solve data skew, and dynamically optimize execution plans.

Solved problems

Adaptive execution of Spark SQL solves the following problems:

- The number of shuffle partitions

Currently, the number of tasks in the reduce stage in Spark SQL depends on the value of the `spark.sql.shuffle.partition` parameter (the default value is 200). Once this parameter has been specified for a job, the number of reduce tasks in all stages is the same value when the job is running.

For different jobs, and for different reduce stages of one job, the actual data size can be quite different. For example, data to be processed in the reduce stage may have a size of 10 MB or 100 GB. If the parameter is specified using the same value, it has a significant impact on the actual processing efficiency. For example, 10 MB of data can be processed using only one task. If the value of the `spark.sql.shuffle.partition` parameter is set to the default value of 200, then 10 MB of data is partitioned to be processed by 200 tasks. This increases scheduling overheads and lowers processing efficiency.

By setting the range of the shuffle partition number, the adaptive execution framework of Spark SQL can dynamically adjust the number of reduce tasks in the range for different stages of different jobs.

This significantly reduces the costs for optimization (no need to find a fixed value). Additionally, the numbers of reduce tasks in different stages of one job can be dynamically adjusted.

Parameter:

| Attribute | Default value | Description |
|---|---------------|---|
| <code>spark.sql.adaptive.enabled</code> | false | Enables or disables adaptive execution. |

| Attribute | Default value | Description |
|---|---------------|--|
| spark.sql.adaptive.minNumPostShufflePartitions | 1 | The minimum number of reduce tasks. |
| spark.sql.adaptive.maxNumPostShufflePartitions | 500 | The maximum number of the reduce tasks. |
| spark.sql.adaptive.shuffle.targetPostShuffleInputSize | 67108864 | Dynamically adjusts the number of reduce tasks based on the partition size. For example, if the value is set to 64 MB, then each task in the reduce stage processes more than 64 MB data. |
| spark.sql.adaptive.shuffle.targetPostShuffleRowCount | 20000000 | Dynamically adjusts the number of reduce tasks based on the row number in the partition. For example, if the value is set to 20000000, then each task in the reduce stage processes more than 20,000,000 rows of data. |

- Data skew

Data skew is a common issue in SQL join operations. It refers to the scenario where certain tasks involve too much data in the processing, which leads to long tails. Currently, Spark SQL does not perform optimization for skewed data.

The Adaptive Execution framework of Spark SQL can automatically detect skewed data and perform optimization for it at runtime.

SparkSQL optimizes skewed data as follows: it splits the data that is in the skewed partition, processes the data through multiple tasks, and then combines the results through SQL union operations.

Supported join types:

| Type | Description |
|-------|--|
| Inner | Skewed data can be handled in both tables. |

| Type | Description |
|------------|---|
| Cross | Skewed data can be handled in both tables. |
| LeftSemi | Skewed data can only be handled in the left table. |
| LeftAnti | Skewed data can only be handled in the left table. |
| LeftOuter | Skewed data can only be handled in the left table. |
| RightOuter | Skewed data can only be handled in the right table. |

Parameter:

| Attribute | Default value | Description |
|--|---------------|--|
| spark.sql.adaptive.enabled | false | Enables or disables the adaptive execution framework. |
| spark.sql.adaptive.skewedJoin.enabled | false | Enables or disables the handling of skewed data. |
| spark.sql.adaptive.skewedPartitionFactor | 10 | A partition is identified as a skewed partition only when the following scenarios occur. First, the size of a partition is greater than this value (median size of all partitions) and the value of the spark.sql.adaptive.skewedPartitionSizeThreshold parameter. Second, the rows in a partition are greater than this value (median rows in all partitions) and the value of the spark.sql.adaptive.skewedPartitionSizeThreshold parameter. |

| Attribute | Default value | Description |
|---|---------------|--|
| spark.sql.adaptive.skewedPartitionSizeThreshold | 67108864 | The size threshold for a skewed partition. |
| spark.sql.adaptive.skewedPartitionRowCountThreshold | 10000000 | The row number threshold for a skewed partition. |
| spark.shuffle.statistics.verbose | false | When the value of this parameter is true , MapStatus collects information about the number of rows in each partition for handling skewed data. |

- Execution plan optimization at runtime

Catalyst optimizer of Spark SQL converts logical plans that are converted from SQL statements into physical execution plans and executes those physical execution plans. However, the physical execution plan produced by Catalyst may not be optimal because of lack or inaccuracy of statistics. For example, Spark SQL may choose SortMergeJoinExec instead of BroadcastJoin, while BroadcastJoin is the optimal option in the scenario.

The Adaptive Execution framework of Spark SQL determines whether to use BroadcastJoin instead of SortMergeJoin to improve query performance based on the size of the shuffle write in the shuffle stage.

Parameter:

| Attribute | Default value | Description |
|--|---|--|
| spark.sql.adaptive.enabled | false | Enables or disables the adaptive execution framework. |
| spark.sql.adaptive.join.enabled | true | Whether to determine a better join strategy at runtime. |
| spark.sql.adaptiveBroadcastJoinThreshold | Equals to spark.sql.autoBroadcastJoinThreshold. | Determines whether to use broadcast join to optimize join queries. |

Test

Take some TPC-DS queries as test samples.

- Shuffle partition number

- Query 30

Native Spark:

Completed Stages: 15

Completed Stages (15)

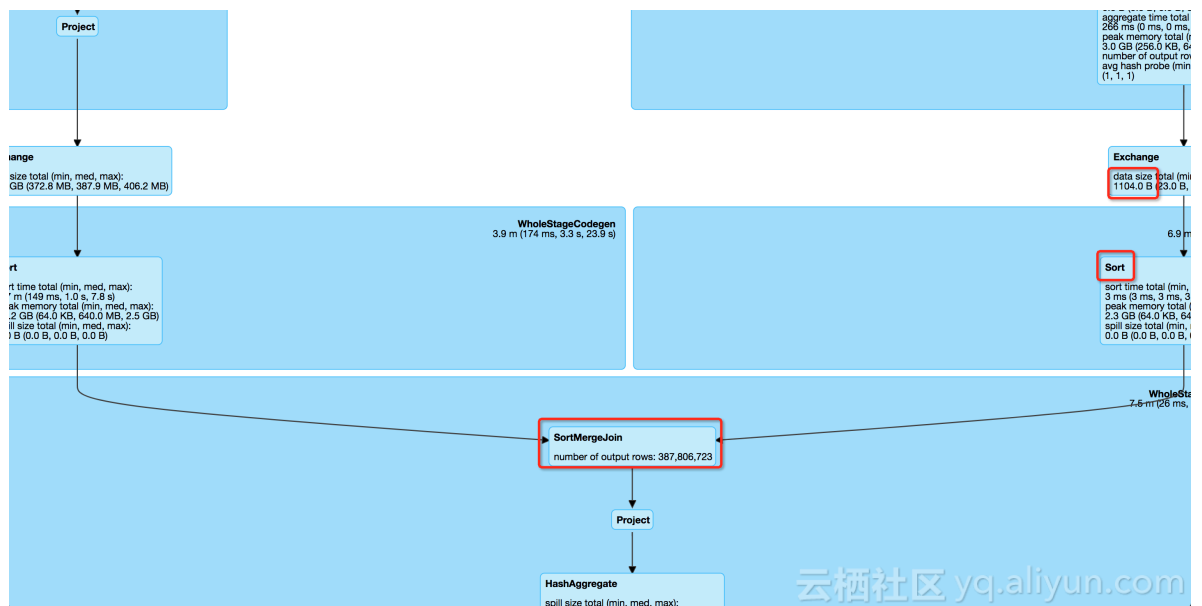
| Stage Id | Description | Submitted | Duration | Tasks: Succeeded/Total | Input | Output | Shuffle Read | Shuffle Write |
|----------|--|---------------------|----------|------------------------|-------|---------|--------------|---------------|
| 14 | Execution: q30-v2.4, iteration: 1, StandardRun=true save at Benchmark.scala:436 | 2018/05/20 13:37:48 | 0.4 s | 1/1 | | 34.0 KB | | |
| 13 | benchmark q30-v2.4 collect at Query.scala:124 | 2018/05/20 13:37:39 | 8 s | 10976/10976 | | | 11.2 GB | |
| 12 | benchmark q30-v2.4 collect at Query.scala:124 | 2018/05/20 13:37:22 | 16 s | 10376/10376 | | 3.6 GB | | 791.3 MB |

- Adjusts the number of reduce tasks adaptively.

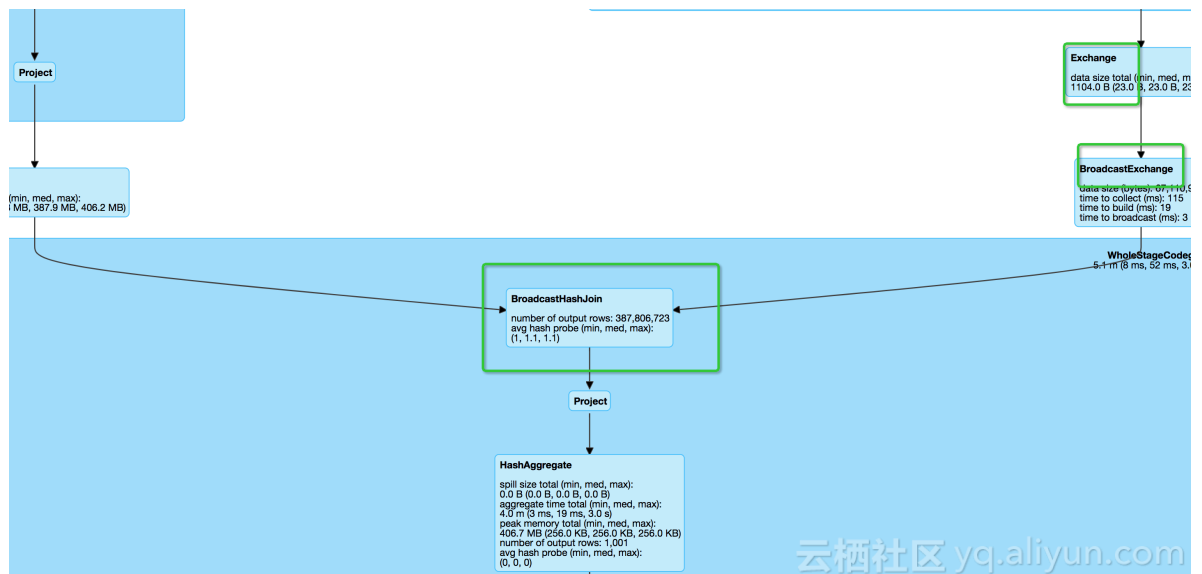
Completed Stages (16)

| Stage Id | Description | Submitted | Duration | Tasks: Succeeded/Total | Input | Output | Shuffle Read | Shuffle Write |
|----------|--|---------------------|----------|------------------------|-------|---------|--------------|---------------|
| 41 | Execution: q30-v2.4, iteration: 1, StandardRun=true save at Benchmark.scala:436 | 2018/05/20 13:44:32 | 0.5 s | 1/1 | | 35.1 KB | | |
| 40 | benchmark q30-v2.4 collect at Query.scala:124 | 2018/05/20 13:44:27 | 4 s | 1027/1027 | | | 12.5 GB | |
| 33 | benchmark q30-v2.4 run at ThreadPoolExecutor.java:1149 | 2018/05/20 13:44:18 | 3 s | 1091/1091 | | 3.5 GB | | 2000.0 MB |

· Execution plan optimization at runtime (SortMergeJoin to BroadcastJoin).



Uses BroadcastJoin adaptively.



8 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 isolated, 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 recommend 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  
:// oldcluster ip : 8020 / user / hive / warehouse / user /  
hive / warehouse
```

- Incremental data synchronization

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

Parameter descriptions:

- `hdfs :// oldcluster ip : 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 bandwidth 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 . permission s . enabled` and `dfs . namenode . ac ls . 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:

1. Delete the metabase of the new cluster and input `drop database xxx`.
2. Run the `mysqldump` command to export the table structure and data of the old cluster's metabase.
3. Replace the location. Tables and partitions in the Hive metadata all have location information within 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
```

4. In the interface of the new cluster, stop the `hivemetastore` service.
5. Log on to the new metabase and create a database.
6. In the new metabase, import all data exported from the old metabase after the location field is replaced.
7. 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.
8. 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 version 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 - directorie s = 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.sql
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_START_ID 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_EX T_ID bigint NOT NULL ,
HL_LOCK_IN T_ID bigint NOT NULL ,
HL_TXNID bigint ,
HL_DB varchar ( 128 ) NOT NULL ,
HL_TABLE varchar ( 128 ) ,
HL_PARTITI ON varchar ( 767 ) ,
HL_LOCK_ST ATE char ( 1 ) not null ,
HL_LOCK_TY PE char ( 1 ) not null ,
HL_LAST_HE ARTBEAT bigint NOT NULL ,
HL_ACQUIRE D_AT bigint ,
HL_USER varchar ( 128 ) NOT NULL ,
HL_HOST varchar ( 128 ) NOT NULL ,
HL_HEARTBE AT_COUNT int ,
HL_AGENT_I NFO varchar ( 128 ) ,
HL_BLOCKED BY_EXT_ID bigint ,
HL_BLOCKED BY_INT_ID bigint ,
PRIMARY KEY ( HL_LOCK_EX T_ID , HL_LOCK_IN T_ID ) ,
KEY HIVE_LOCK_ TXNID_INDE X ( HL_TXNID )
) ENGINE = InnoDB DEFAULT CHARSET = latin1 ;
CREATE INDEX HL_TXNID_I DX 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_COMPON ENTS (
TC_TXNID bigint ,
TC_DATABAS E varchar ( 128 ) NOT NULL ,
TC_TABLE varchar ( 128 ) ,
TC_PARTITI ON 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 ` NOTIFICATI ON_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 ` PARTITIONE VENTINDEX ` ( ` PARTITION_ NAME ` )
) ENGINE = InnoDB DEFAULT CHARSET = latin1 ;

CREATE TABLE COMPLETED_ TXN_COMPON ENTS (
CTC_TXNID bigint NOT NULL ,
CTC_DATABA SE varchar ( 128 ) NOT NULL ,

```

```
CTC_TABLE varchar ( 128 ),
CTC_PARTIT ION 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_I D (
    NTXN_NEXT bigint NOT NULL
) ENGINE = InnoDB DEFAULT CHARSET = latin1 ;
INSERT INTO NEXT_TXN_I D VALUES ( 1 );
CREATE TABLE NEXT_LOCK_ ID (
    NL_NEXT bigint NOT NULL
) ENGINE = InnoDB DEFAULT CHARSET = latin1 ;
INSERT INTO NEXT_LOCK_ ID VALUES ( 1 );
```

9 Use EMR Data Science clusters for deep learning

Data Science cluster is a new model available in E-MapReduce (EMR) 3.13.0 and later versions for machine learning and deep learning. You can use GPU or CPU models to perform data training through Data Science clusters. Training data can be stored on HDFS and OSS. EMR supports TensorFlow for distributed training on large amounts of data.

Create cluster

- Prerequisites for creating an EMR Data Science cluster:
 - EMR 3.13.0 or later.
 - Data Science as the cluster type.

• Create a cluster

Create Cluster

The screenshot shows the 'Software Configuration' tab of the 'Create Cluster' wizard. It includes a progress bar with four steps: Software Configuration (active), Hardware Configuration, Basic Configuration, and OK. The 'Version Configuration' section shows 'EMR Version' set to 'EMR-3.16.0'. Under 'Cluster Type', 'Data Science' is selected. The 'Required Services' section lists Jupyter (4.4.0), Analytics Zoo (0.2.0), ApacheDS (2.0.0), Tensorflow on YARN (1.0.0), Zeppelin (0.8.0), Spark (2.3.2), YARN (2.7.2), HDFS (2.7.2), ZooKeeper (3.4.13), and Ganglia (3.7.2). The 'Optional Services' section shows TensorFlow (1.8.0), Hue (4.1.0), and Hive (2.3.3). There is a 'Click to Choose' link and an 'Enable Custom Setting' toggle. A 'Next' button is at the bottom right.

Select a CPU model for Master Instance Type, and select a CPU or GPU model for Core Instance Type.

The screenshot shows the 'Hardware Configuration' tab of the 'Create Cluster' wizard. It is divided into two sections: 'Master Instance Type' and 'Core Instance Type'.
In the 'Master Instance Type' section, '4 vCPU 16GB' is selected, with 'ecs.g5.xlarge' shown to the right. Below this, 'System Disk Type' has 'SSD Disk' selected, 'System Disk Size' is '120' GB, 'Data Disk Type' has 'Ultra Disk' selected, 'Data Disk Size' is '80' GB, and 'Master Instances' is '1'.
In the 'Core Instance Type' section, a dropdown menu is open for '4 vCPU 16GB', showing options like 'General Purpose ecs.g5' and 'Compute Optimized ecs.c5'. The 'Core Instance Type' is currently set to '4 vCPU 16GB'.

If you select a GPU model, EMR provides Nvidia GPU drivers and the corresponding Cudnn installation.

After the cluster is created, the corresponding service, driver, and Cudnn are installed. The docker service is installed on all core nodes.

Run TensorFlow on a Data Science cluster

- TensorFlow

TensorFlow is an open-source machine learning framework for deep learning of machine learning tasks and training neural models. For more information about TensorFlow, see [TensorFlow](#).

- TensorFlow on YARN

TensorFlow on YARN developed by the EMR kernel team is a distributed TensorFlow framework based on YARN scheduling. It supports running TensorFlow jobs on YARN and using GPU for training. For information about how to use TensorFlow on YARN, see [TensorFlow instructions](#).

- Use TensorFlow on YARN to perform deep learning

TensorFlow on YARN can use high-level APIs for training with more concise codes. This topic takes the Wide and Deep model as an example for training. For information about more models, see [github](#). Click [here](#) to download training data. Data for training is adult.data and adult.test. This example writes training steps in Python according to the stand-alone version.

1. Define training data, and then upload training data and validation data to HDFS.

Put the training data to the /ml/ directory of hdfs:

```
hdfs dfs -put adut . data adult . test / ml /
```

2. Specify the training data path in the training code:

```
TRAIN_FILE S = [' hdfs :// emr - header - 1 . cluster -  
500157403 : 9000 / ml / adult . data ']  
EVAL_FILES = [' hdfs :// emr - header - 1 . cluster - 500157403  
: 9000 / ml / adult . test ']
```

HDFS schema is set according to your cluster. If the cluster is not a high availability cluster (HA cluster), you only need to check the `fs . defaultFS`

property in `core-site.xml`. If the cluster is an HA cluster, by default, the HDFS schema is `emr-cluster`.

3. Define feature columns.

Define the corresponding features according to the wide and deep model:

```
""" Build a wide and deep model for predicting
income category .
"""
( gender , race , education , marital_status , relationship ,
workclass , occupation , native_country , age ,
education_num , capital_gain , capital_loss ,
hours_per_week ) = INPUT_COLUMNS
# Continuous columns can be converted to categorical
via bucketization
age_buckets = tf.feature_column.bucketized_column (
age , boundaries =[ 18 , 25 , 30 , 35 , 40 , 45 , 50 , 55
, 60 , 65 ])
# Wide columns and deep columns .
wide_columns = [
# Interactions between different categorical features
can also
# be added as new virtual features .
tf.feature_column.crossed_column (
[ ' education ' , ' occupation ' ], hash_bucket_size = int ( 1e4
)),
tf.feature_column.crossed_column (
[ age_buckets , race , ' occupation ' ], hash_bucket_size =
int ( 1e6 )),
tf.feature_column.crossed_column (
[ ' native_country ' , ' occupation ' ], hash_bucket_size = int
( 1e4 )),
gender ,
native_country ,
education ,
occupation ,
workclass ,
marital_status ,
relationship ,
age_buckets ,
]
deep_columns = [
# Use indicator columns for low dimensional
vocabularies
tf.feature_column.indicator_column ( workclass ),
tf.feature_column.indicator_column ( education ),
tf.feature_column.indicator_column ( marital_status ),
tf.feature_column.indicator_column ( gender ),
tf.feature_column.indicator_column ( relationship ),
tf.feature_column.indicator_column ( race ),
# Use embedding columns for high dimensional
vocabularies
tf.feature_column.embedding_column (
native_country , dimension = embedding_size ),
tf.feature_column.embedding_column ( occupation ,
dimension = embedding_size ),
age ,
education_num ,
capital_gain ,
capital_loss ,
```

```
hours_per_ week ,
]
```

4. Define input_fn.

You can use `input_fn` to obtain training data:

```
def input_fn ( filenames ,
num_epochs = None ,
shuffle = True ,
skip_heade r_lines = 0 ,
batch_size = 200 ):
    """ Generates features and labels for training or
    evaluation .
    """
    dataset = tf . data . TextLineDa taset ( filenames ). skip (
skip_heade r_lines ). map ( parse_csv )
    if shuffle :
        dataset = dataset . shuffle ( buffer_siz e = batch_size * 10
        )
    dataset = dataset . repeat ( num_epochs )
    dataset = dataset . batch ( batch_size )
    iterator = dataset . make_one_s hot_iterat or ()
    features = iterator . get_next ()
    return features , parse_labe l_column ( features . pop (
LABEL_COLU MN ))
    train_inpu t = lambda : input_fn (
    TRAIN_FILE S ,
    batch_size = 40
    )
    # Don ' t shuffle evaluation data
    eval_input = lambda : input_fn (
    EVAL_FILES ,
    batch_size = 40 ,
    shuffle = False
    )
```

5. Initiate Estimator.

The following example uses the pre-defined TensorFlow's wide and deep model to build the Estimator:

```
tf . estimator . DNNLinearC ombinedCla ssifier (
config = config ,
linear_fea ture_colum ns = wide_colum ns ,
dnn_featur e_columns = deep_colum ns ,
dnn_hidden _units = hidden_uni ts or [ 100 , 70 , 50 , 25
]
)
```

6. Train the models:

```
train_spec = tf . estimator . TrainSpec ( train_inpu t ,
max_steps = 1000
)
exporter = tf . estimator . FinalExpor ter ( ' census ' ,
json_servi ng_input_f n )
eval_spec = tf . estimator . EvalSpec ( eval_input ,
steps = 100 ,
exporters =[ exporter ],
name = ' census - eval ' )
```

```
)  
tf.estimator.train_and_evaluate(estimator, train_spec,  
                                eval_spec)
```

After the codes are complete, you can submit a task to your Data Science cluster. We recommend that you send the task to the cluster to perform standalone training using a standalone model. After verifying the task without code errors, you can submit a distributed task and specify worker and ps resources for training. An example command for submitting a task is as follows:

```
el_submit -t tensorflow -ps -a wide_and_deep -m  
local -x True -f ./ -pn 1 -pc 1 -pm 2000 -wn
```



```
1 - wc 1 - wg 1 - wm 2000 - c python census_sin
gle .
```

The running status of the submitted task can be viewed in the YARN page.

| Apps Submitted | Apps Pending | Apps Running | Apps Completed | Containers Running | Memory Used | Memory Total | Memory Reserved | VCores Used | VCores Total | VCores Reserved | GCores Used | GCores Total | GCores Reserved | Active Nodes | Decommissioned Nodes | Lost Nodes | Unhealthy Nodes |
|----------------|--------------|--------------|----------------|--------------------|-------------|--------------|-----------------|-------------|--------------|-----------------|-------------|--------------|-----------------|--------------|----------------------|------------|-----------------|
| 1 | 0 | 1 | 0 | 3 | 5.25 GB | 26.50 GB | 0 B | 3 | 16 | 0 | 1 | 2 | 0 | 2 | 0 | 0 | 0 |

| Scheduler Metrics | | Scheduling Resource Type | | Minimum Allocation | | Maximum Allocation | |
|--------------------|--|--------------------------|--|---------------------------------|--|------------------------------------|--|
| Capacity Scheduler | | [MEMORY, CPU, GPU] | | <memory:32, vCores:1, gCores:0> | | <memory:13568, vCores:8, gCores:1> | |

| Show 20 entries | | | | | | | | | | Search: | |
|--------------------------------|------|---------------|------------------|---------|--------------------------------|------------|---------|-------------|----------|-------------------|-------|
| ID | User | Name | Application Type | Queue | StartTime | FinishTime | State | FinalStatus | Progress | Tracking UI | Black |
| application_1539156175145_0001 | root | wide_and_deep | tensorflow-ps | default | Wed Oct 10 18:17:05 +0800 2018 | N/A | RUNNING | UNDEFINED | | ApplicationMaster | 0 |

Showing 1 to 1 of 1 entries

Click the ApplicationMaster link to view the running status and details of the task.

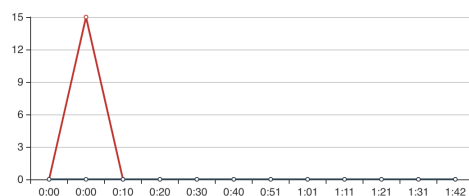
App Info:

| Key | Value |
|---------------------------|--------------------------------|
| App ID | application_1539156175145_0015 |
| App Type | tensorflow-ps |
| App Command | python census_single.py |
| App Mode | local |
| App Container Number | 2 |
| App Ps Number | 1 |
| App Worker Number | 1 |
| App Worker Number Per GPU | 1 |
| App Resource Cpu | 2 |
| App Resource Gpu | 1 |
| App Resource Mem | 4,000MB |

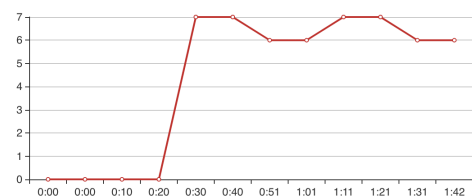
All Containers:

| Container ID | Container CPU | Container GPU | Container MEM | Container Role | Container Status | Allocate Time | Start Time | Finish Time | Container Log |
|--|---------------|---------------|---------------|----------------|------------------|---------------------|---------------------|-------------|---------------|
| container_1539156175145_0015_01_000002 | 1 | 0 | 2,000MB | ps | RUNNING | 2018-10-11 11:45:38 | 2018-10-11 11:45:38 | N/A | log |
| container_1539156175145_0015_01_000003 | 1 | 1 | 2,000MB | worker | RUNNING | 2018-10-11 11:45:38 | 2018-10-11 11:45:38 | N/A | log |


CPU



GPU



Click log to go to ps or worker to view training information.



Showing 4096 bytes. Click [here](#) for full log

questsDependencyWarning: urllib3 (1.22) or chardet (2.2.1) doesn't match a supported version!

RequestsDependencyWarning

INFO:tensorflow:Using config: {'_save_checkpoint_secs': 600, '_session_config': None, '_keep_checkpoint_max': 5, '_task_type': 'u'chief', '_train_distrib

INFO:tensorflow:Start Tensorflow server.

2018-10-11 10:30:49.184647: I tensorflow/core/platform/cpu_feature_guard.cc:140] Your CPU supports instructions that this TensorFlow binary was not compil

2018-10-11 10:30:49.339606: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:898] successful NUMA node read from SysFS had negative value (-1), but '

2018-10-11 10:30:49.339989: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1356] Found device 0 with properties:

name: Tesla P4 major: 6 minor: 1 memoryClockRate(GHz): 1.1135

pciBusID: 0000:00:0b:0

totalMemory: 7.43GiB freeMemory: 7.31GiB

2018-10-11 10:30:49.340025: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1435] Adding visible gpu devices: 0

2018-10-11 10:30:49.659716: I tensorflow/core/common_runtime/gpu/gpu_device.cc:923] Device interconnect StreamExecutor with strength 1 edge matrix:

2018-10-11 10:30:49.659776: I tensorflow/core/common_runtime/gpu/gpu_device.cc:929] 0

2018-10-11 10:30:49.659794: I tensorflow/core/common_runtime/gpu/gpu_device.cc:942] 0: N

2018-10-11 10:30:49.660116: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1053] Created TensorFlow device (/job:chief/replica:0/task:0/device:GPU:0 w

2018-10-11 10:30:49.792150: I tensorflow/core/distributed_runtime/rpc/grpc_channel.cc:215] Initialize GrpcChannelCache for job chief -> {0 -> localhost:38

2018-10-11 10:30:49.792192: I tensorflow/core/distributed_runtime/rpc/grpc_channel.cc:215] Initialize GrpcChannelCache for job ps -> {0 -> 192.168.0.49:44

2018-10-11 10:30:49.792750: I tensorflow/core/distributed_runtime/rpc/grpc_server_lib.cc:332] Started server with target: grpc://localhost:38934

Picked up _JAVA_OPTIONS: -XX:ErrorFile=/tmp/hs_err.log

SLF4J: Class path contains multiple SLF4J bindings.

SLF4J: Found binding in [jar:file:/mnt/disk3/yarn/filecache/10/el-on-yarn-1.0.0.jar!/org/slf4j/impl/StaticLoggerBinder.class]

SLF4J: Found binding in [jar:file:/opt/apps/emr/service/hadoop/2.7.2-1.2.10-gpu/package/hadoop-2.7.2-1.2.10-gpu/share/hadoop/common/lib/slf4j-log4j12-1.7.

SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.

SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]

INFO:tensorflow:Calling model_fn.

INFO:tensorflow:Done calling model_fn.

INFO:tensorflow:Create CheckpointSaverHook.

INFO:tensorflow:Graph was finalized.

2018-10-11 10:30:55.070582: I tensorflow/core/distributed_runtime/master_session.cc:1136] Start master session 989a426030234d3b with config: allow_soft_pl

INFO:tensorflow:Running local_init_op.

INFO:tensorflow:Done running local_init_op.

INFO:tensorflow:Saving checkpoints for 1 into hdf://emr-header-1.cluster-500159381:9000/census/model.ckpt.

INFO:tensorflow:loss = 20.34863, step = 0

INFO:tensorflow:global_step/sec: 32.336

INFO:tensorflow:loss = 17.772442, step = 100 (3.093 sec)

INFO:tensorflow:global_step/sec: 45.4349

INFO:tensorflow:loss = 16.894323, step = 200 (2.201 sec)

INFO:tensorflow:global_step/sec: 45.7951

INFO:tensorflow:loss = 17.784704, step = 300 (2.184 sec)

In this example, after the training ends, you can find models in the HDFS path / *census* where models are generated.

```
[root@emr-header-1 ~]# hdfs dfs -ls /census
Found 9 items
-rw-r----- 2 hadoop hadoop      132 2018-10-11 10:34 /census/checkpoint
-rw-r----- 2 hadoop hadoop       40 2018-10-11 10:30 /census/events.out.tfevents.1539225054.emr-worker-1.cluster-500159381
-rw-r----- 2 hadoop hadoop    1839028 2018-10-11 10:30 /census/graph.pbtxt
-rw-r----- 2 hadoop hadoop 12406864 2018-10-11 10:31 /census/model.ckpt-1.data-00000-of-00001
-rw-r----- 2 hadoop hadoop     2463 2018-10-11 10:31 /census/model.ckpt-1.index
-rw-r----- 2 hadoop hadoop     870015 2018-10-11 10:31 /census/model.ckpt-1.meta
-rw-r----- 2 hadoop hadoop 12406864 2018-10-11 10:34 /census/model.ckpt-10000.data-00000-of-00001
-rw-r----- 2 hadoop hadoop     2463 2018-10-11 10:34 /census/model.ckpt-10000.index
-rw-r----- 2 hadoop hadoop     870015 2018-10-11 10:34 /census/model.ckpt-10000.meta
```

Question descriptions

If the following errors display, check if there are empty lines in *adult.data* and *adult.test*.

```
tensorflow.python.framework.errors_impl.InvalidArgumentError: Expect 15 fields but have 0 in record
0
[[ Node : DecodeCSV = DecodeCSV [ OUT_TYPE =[ DT_INT32 , DT_STRING
, DT_INT32 , DT_STRING , DT_INT32 , ..., DT_INT32 , DT_INT32 ,
DT_INT32 , DT_STRI
NG , DT_STRING ], field_delim = ",", na_value = "", use_quote_
delim = true ]( ExpandDims , DecodeCSV / record_def aults_0 ,
DecodeCSV / record_def aults_1 , D
ecodeCSV / record_def aults_2 , DecodeCSV / record_def aults_3 ,
DecodeCSV / record_def aults_4 , DecodeCSV / record_def aults_5 ,
DecodeCSV / record_def ault
s_6 , DecodeCSV / record_def aults_7 , DecodeCSV / record_def
aults_8 , DecodeCSV / record_def aults_9 , DecodeCSV / record_def
aults_10 , DecodeCSV / record_
defaults_1 1 , DecodeCSV / record_def aults_12 , DecodeCSV /
record_def aults_13 , DecodeCSV / record_def aults_14 )]]
```

```

[[ Node : IteratorGe tNext = IteratorGe tNext [ output_sha pes
=[?, 1 ], [?, 1 ], [?, 1 ], [?, 1 ], [?, 1 ], [?, 1 ], [?, 1 ], [?,
1 ], [?, 1 ], [?, 1 ], [?, 1 ],
[?, 1 ], [?, 1 ], [?, 1 ]], output_typ es =[ DT_INT32 , DT_INT32
, DT_INT32 , DT_STRING , DT_INT32 , DT_STRING , DT_INT32 ,
DT_STRING , DT_STRING , DT_STRING
, DT_STRING , DT_STRING , DT_STRING , DT_STRING ], _device
="/ job : chief / replica : 0 / task : 0 / device : CPU : 0 "](
OneShotIte rator )]]
[[ Node : global_ste p / cond / pred_id_S6 15 = _HostRecv [
client_ter minated = false , recv_devic e ="/ job : ps / replica :
0 / task : 0 / device : CPU : 0 ", send_d
evice ="/ job : chief / replica : 0 / task : 0 / device : GPU : 0 ",
send_devic e_incarnat ion = 6104642431 418663740 , tensor_nam e
=" edge_602_g lobal_step / cond / pred_
id ", tensor_typ e = DT_BOOL , _device ="/ job : ps / replica : 0
/ task : 0 / device : CPU : 0 "]()]
2

```

10 Use Flink jobs to process OSS data

This topic describes how to create a Hadoop cluster by using E-MapReduce (EMR) and use Flink jobs to process Object Storage Service (OSS) data in the cluster.

Prerequisites

- You have registered an Alibaba Cloud account. For more information, see [Create an Alibaba Cloud account](#).
- You have activated EMR and OSS.
- You have authorized the Alibaba Cloud account. For more information, see [#unique_19](#).

Context

During practical applications, you always need to consume data stored in OSS. In EMR, you can run a Flink job to consume data stored in OSS buckets. You can perform the following steps to create a Flink job in EMR and run the Flink job on a Hadoop cluster to obtain and output the specified content of a file stored in OSS.

Step 1: Prepare the environment

Before creating a Flink job, you must prepare the Maven and Java environment on your local host and create a Hadoop cluster in EMR. If you are using Maven 3.0 or later, we recommend that you use Java 2.0 or earlier to ensure compatibility.

1. Install Maven and Java on your local host.
2. Log on to the [EMR console](#) and create a Hadoop cluster. You must select Flink in the Optional Services field. For more information, see [#unique_5](#).

Step 2: Prepare testing data

Before creating a Flink job, you must upload testing data to OSS. The following is an example of uploading a file named `test.txt`. The file content is: Nothing is impossible for a willing heart. While there is a life, there is a hope.

1. Log on to the [OSS console](#).

2. Create a bucket and upload the file to the bucket. For more information, see [#unique_20](#) and [#unique_21](#).

The sample path of the uploaded file is `oss :// emr - logs2 / hengwu / test . txt` . Keep the path for later use.



Note:

After uploading the file, keep the OSS logon window open for later use.

Step 3: Build a JAR file and upload it to OSS or the Hadoop cluster

Download EMR sample code [aliyun-emapreduce-demo](#) and compile the code to create a JAR file. You can upload the JAR file to the header node of the Hadoop cluster or an OSS bucket. The following is an example of uploading the JAR file to an OSS bucket.

1. Download EMR sample code [aliyun-emapreduce-demo](#) to your local disk.
2. Run the `mvn clean package -DskipTests` command to create the JAR file.

The new JAR file is installed in the `../ target /` directory, for example, `target / examples - 1 . 2 . 0 . jar` .

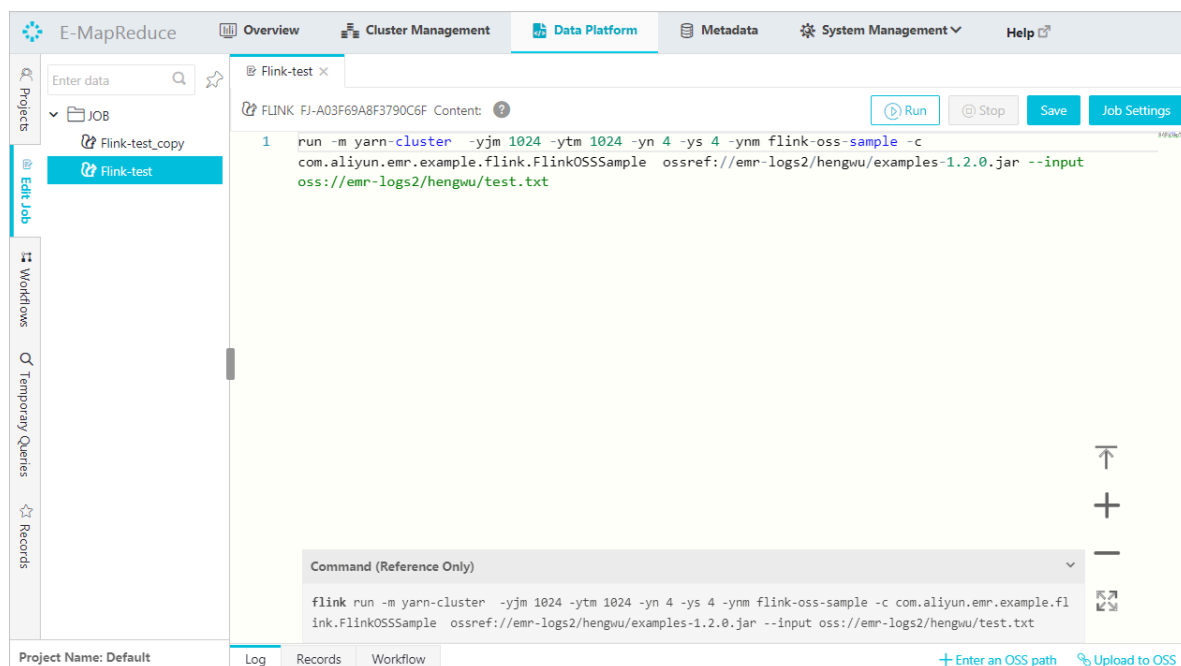
3. Go to the [OSS console](#) and upload the JAR file to an OSS directory.

The sample path of the JAR file is `oss :// emr - logs2 / hengwu / examples - 1 . 2 . 0 . jar` . Keep the path for later use.

Step 4: Create and run a Flink job

1. Log on to the [EMR console](#).
2. On the Data Platform tab, create a project. For more information, see [#unique_22](#).
3. Open the new project, select the Edit Job tab, and create a job with the type of Flink.

4. After the new Flink job is created, specify the content of the job.



The job content is a snippet of code. An example is shown as follows.

```
run -m yarn-cluster -yjm 1024 -ytm 1024 -yn 4
-ys 4 -ynm flink-oss-sample -c com.aliyun.emr
.example.flink.FlinkOSSSample ossref://emr-logs2/
hengwu/examples-1.2.0.jar --input oss://emr-logs2/
hengwu/test.txt
```

Key parameters in the preceding code are described as follows:

- `ossref://emr-logs2/hengwu/examples-1.2.0.jar` : indicates the path of the uploaded JAR file.
- `oss://emr-logs2/hengwu/test.txt` : indicates the path of the testing data.



Note:

Replace the value of each parameter with values based on the configurations in the [Step 1: Prepare the environment](#) and [Step 3: Build a JAR file and upload it to OSS or the Hadoop cluster](#) topics.

5. After the job configuration is complete, click Run in the upper-right corner, and select the name of the new Hadoop cluster in the Target Cluster field.

6. Click OK to run the Flink job.

When the job is running, the Log window appears. After the job is complete, the file content is obtained from an OSS bucket and output to logs. At this point, the Flink job that runs on an EMR cluster to consume OSS data is complete.

| Log | Records | Workflow |
|--|---------|----------|
| <pre>2019-07-19 11:49:00,582 INFO org.apache.flink.yarn.AbstractYarnClusterD 2019-07-19 11:49:00,599 INFO org.apache.hadoop.yarn.client.api.impl.Yar 2019-07-19 11:49:00,600 INFO org.apache.flink.yarn.AbstractYarnClusterD 2019-07-19 11:49:00,601 INFO org.apache.flink.yarn.AbstractYarnClusterD 2019-07-19 11:49:04,382 INFO org.apache.flink.yarn.AbstractYarnClusterD Starting execution of program Nothing is impossible for a willing heart While there is life, there is hope~ Program execution finished Job with JobID 7fccacdb6f870a4d85949a969374023 has finished. Job Runtime: 8292 ms Accumulator Results: - 56193209a112f1ed8c611176ab2597f4 (java.util.ArrayList) [2 elements]</pre> | | |

Step 6: View the logs and details of the job

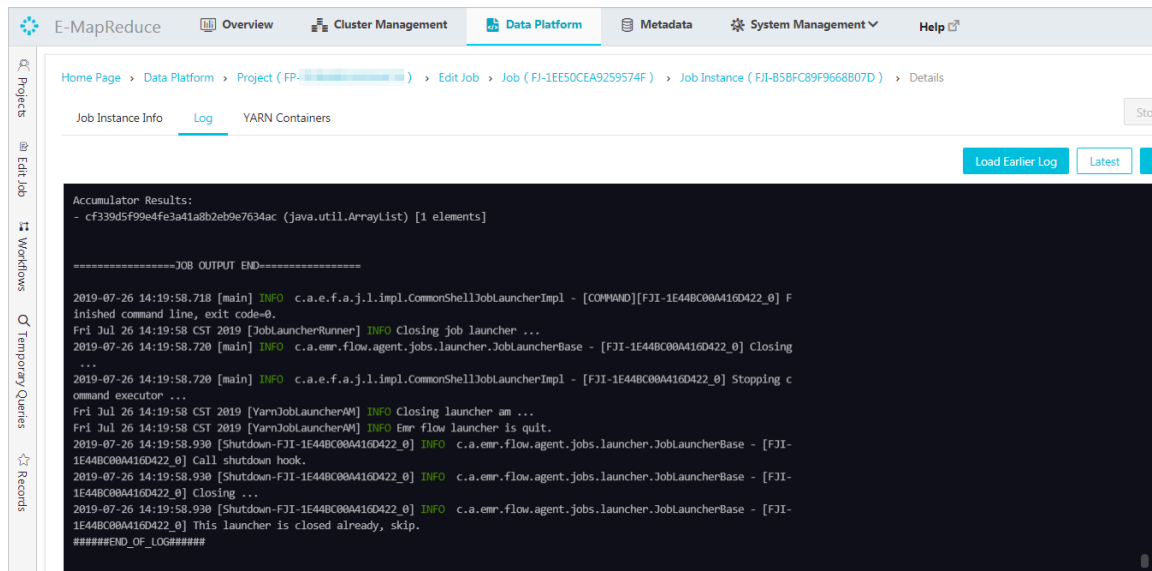
You can view the logs and details of the job to identify the cause of a job failure and details of the job.

1. View the logs of the job.

You can view logs in the EMR console or on an SSH client.

- Log on to the [EMR console](#) to view logs.

After submitting a job in the console, you can open the Details page of a job listed on the Records tab. On the Details page, you can view the results of the job.



- You can log on to the header node of a Hadoop cluster by using SSH to view logs.

By default, the logs of a Flink job are stored in the `/mnt/disk1/log/flink/flink -< user >- client -< hostname >.log` file based on the configurations in the `log4j` file. For more information about detailed configurations, see the `/etc/ecm/flink-conf/log4j-yarn-session.properties` file.

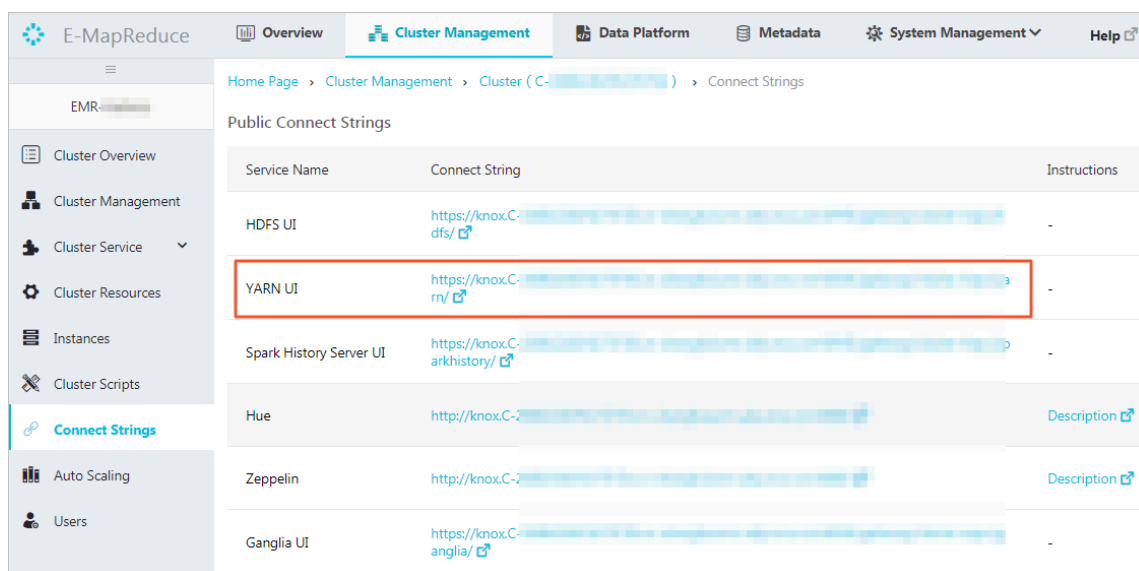
The `user` field indicates the account with which you submit the Flink job. The `hostname` field indicates the name of the instance to which you submit the job. Assume that you log on as a root user and submit a Flink job on the `emr-header-1` instance. In this case, the log path is `/mnt/disk1/log/flink/flink - flink - historyser ver - 0 - emr - header - 1 . cluster - 126601 . log`.

2. View the details of the job.

You can use Yarn UI. You can access Yarn UI by using SSH and Knox. For more information about SSH, see [#unique_23](#). For more information about Knox, see

#unique_24 and Access links and ports. The following takes Knox as an example to describe how to view the details of a job.

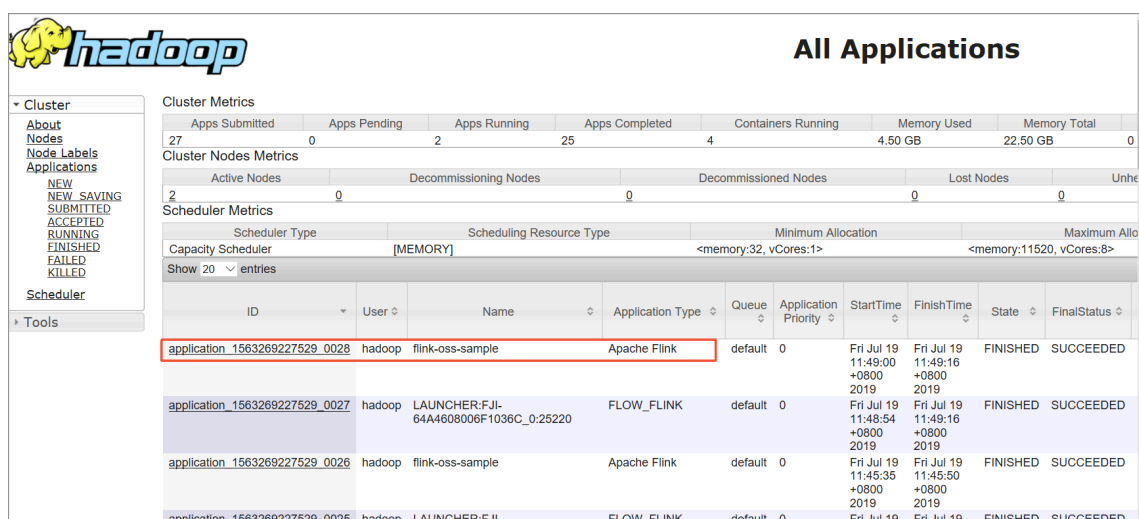
- a) On the Connect Strings tab of the Hadoop cluster, click the link next to Yarn UI to open the Hadoop console.



The screenshot shows the E-MapReduce console interface. The 'Cluster Management' tab is selected, and the 'Connect Strings' sub-tab is active. A table lists various services and their connect strings. The 'YARN UI' row is highlighted with a red box, and a link is visible next to it.

| Service Name | Connect String | Instructions |
|-------------------------|---|-----------------------------|
| HDFS UI | https://knox.C-...dfs/ | - |
| YARN UI | https://knox.C-...rn/ | - |
| Spark History Server UI | https://knox.C-...arkhistory/ | - |
| Hue | http://knox.C-... | Description |
| Zeppelin | http://knox.C-... | Description |
| Ganglia UI | https://knox.C-...anglia/ | - |

- b) In the Hadoop console, click the ID of the job to view the details of the job.



The screenshot shows the Hadoop console interface. The 'All Applications' page is displayed, showing a table of applications. The first application row is highlighted with a red box, showing its ID and details.

| ID | User | Name | Application Type | Queue | Application Priority | StartTime | FinishTime | State | FinalStatus |
|--------------------------------|--------|---------------------------------------|------------------|---------|----------------------|--------------------------------|--------------------------------|----------|-------------|
| application_1563269227529_0028 | hadoop | flink-oss-sample | Apache Flink | default | 0 | Fri Jul 19 11:49:00 +0800 2019 | Fri Jul 19 11:49:16 +0800 2019 | FINISHED | SUCCEEDED |
| application_1563269227529_0027 | hadoop | LAUNCHER:FJI-64A4608006F1036C_0:25220 | FLOW_FLINK | default | 0 | Fri Jul 19 11:48:54 +0800 2019 | Fri Jul 19 11:49:16 +0800 2019 | FINISHED | SUCCEEDED |
| application_1563269227529_0026 | hadoop | flink-oss-sample | Apache Flink | default | 0 | Fri Jul 19 11:45:35 +0800 2019 | Fri Jul 19 11:45:50 +0800 2019 | FINISHED | SUCCEEDED |
| application_1563269227529_0025 | hadoop | LAUNCHER:FJI- | FLOW_FLINK | default | 0 | Fri Jul 19 | Fri Jul 19 | FINISHED | SUCCEEDED |

Application Overview

User: [hadoop](#)

Name: [flink-oss-sample](#)

Application Type: [Apache Flink](#)

Application Tags: [flink,fj-72c097d13e428963,fji-64a4608006f1036c,fji-64a4608006f1036c_0,1250460021754461](#)

Application Priority: [0](#) (Higher Integer value indicates higher priority)

YarnApplicationState: [FINISHED](#)

Queue: [default](#)

FinalStatus Reported by AM: [SUCCEEDED](#)

Started: [Fri Jul 19 11:49:00 +0800 2019](#)

Elapsed: [15sec](#)

Tracking URL: [History](#)

Log Aggregation Status: [SUCCEEDED](#)

Diagnostics:

Unmanaged Application: [false](#)

Application Node Label expression: [<Not set>](#)

AM container Node Label expression: [<DEFAULT_PARTITION>](#)

Application Metrics

Total Resource Preempted: [<memory:0, vCores:0>](#)

Total Number of Non-AM Containers Preempted: [0](#)

Total Number of AM Containers Preempted: [0](#)

Resource Preempted from Current Attempt: [<memory:0, vCores:0>](#)

Number of Non-AM Containers Preempted from Current Attempt: [0](#)

Aggregate Resource Allocation: [22961 MB-seconds, 21 vcore-seconds](#)

Aggregate Preempted Resource Allocation: [0 MB-seconds, 0 vcore-seconds](#)

Show 20 entries

Search:

| Attempt ID | Started | Node | Logs | Nodes blacklisted by the app | Nodes blacklisted by the system |
|--------------------------------------|---------------------|--|----------------------|------------------------------|---------------------------------|
| appattempt_1563269227529_0028_000001 | Fri Jul 19 11:49:00 | http://emr-worker-2.cluster- | Logs | 0 | 0 |

- c) If you need to view a list of running Flink jobs, you can click the link next to Tracking URL on the Details page. The Flink Dashboard page that appears displays the list of running Flink jobs.

You can also access <http://emr-header-1:8082> to view a list of completed jobs.

11 Submit a Flink job to process OSS data using EMR

This topic describes how to create a Hadoop cluster using EMR and run a Flink job to consume OSS data.

Context

Consuming data stored in Alibaba Cloud OSS is a common scenario in the development process. This topic describes how to create a Hadoop cluster using EMR and run a Flink job to consume OSS data.

Procedure

1. Prepare the environment

Select Flink from the optional services when you create a Hadoop cluster.

2. Develop a Flink job

a) Run the `mvn clean package -DskipTests` command.

EMR provides the sample code for you to immediately use. You can download the sample code from [e-mapreduce-demo](#). The path of the JAR file is `target/examples - 1 . 1 . jar .`

b) Prepare OSS test data

Create an OSS path and prepare test data.

c) Run a Flink job.

Upload `examples - 1 . 1 . jar` to the `emr-header-1` node of the Hadoop cluster. Connect to the Hadoop cluster and submit a job. Run the following command.

```
flink run -m yarn - cluster - yjm 1024 - ytm 1024  
- yn 4 - ys 4 - ynm flink - oss - sample - c com .
```

```
aliyun . emr . example . flink . FlinkOSSSample . examples - 1
. 1 . jar -- input oss :// bucket / path
```

oss :// bucket / path is the OSS path set in the previous step.

d) View job submission

By default, logs of job submission are stored in the following path according to the Log4j configurations (see */ etc / ecm / flink - conf / log4j - yarn - session . properties*).

```
/ mnt / disk1 / log / flink / flink -{ user }- client -{ hostname }.
log
```

Replace "user" with the username of the user that submits the Flink job. Replace "hostname" with the hostname of the host to which the job is submitted. For example, if the root user submits a Flink job to the emr-header-1 node, the log path is:

```
/ mnt / disk1 / log / flink / flink - root - client - emr - header -
1 . cluster - 500162572 . log
```

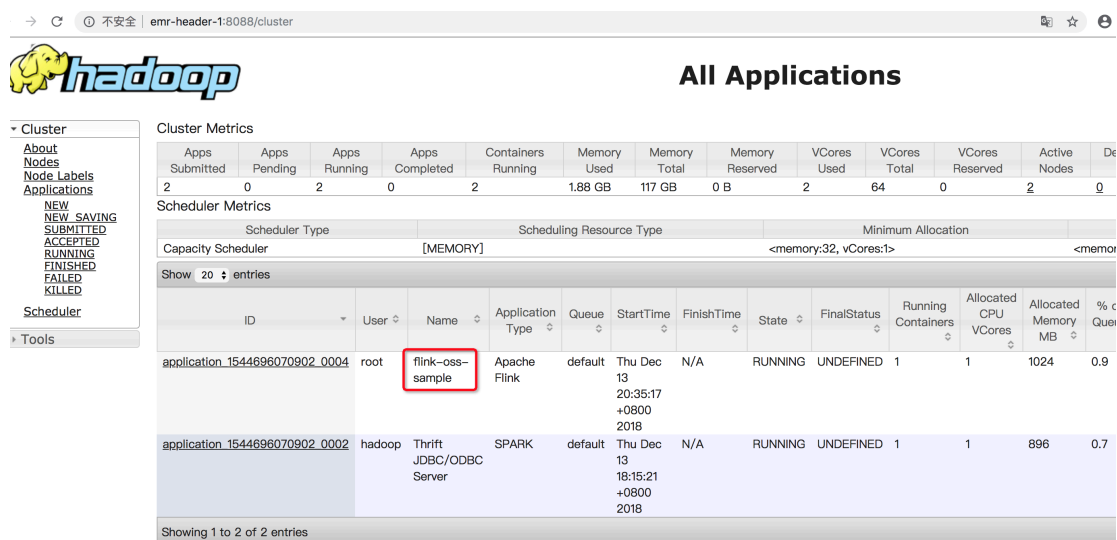
e) View the information of a Flink job

You can view the information of a Flink job using the YARN Web UI. You can access the YARN Web UI using the following methods: Using SSH. For

more information, see [Connect to a cluster using SSH](#). Using Knox. For more information, see [How to use Knox](#).

- View the information of a running job

The following example uses SSH tunneling. The endpoint is `http://emr-header-1:8088/index.html`. On the YARN Web UI, you can see the submitted job as the following figure.



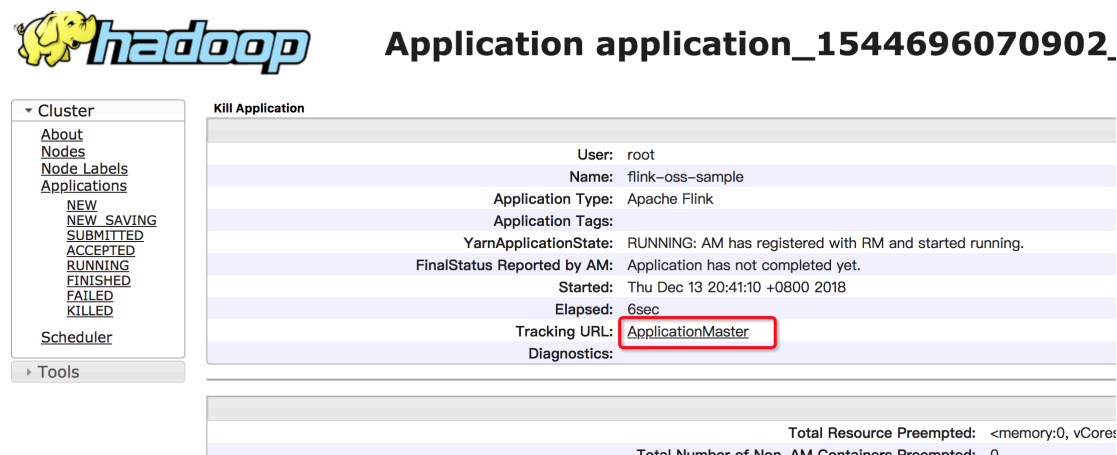
All Applications

| Cluster Metrics | | | | | | | | | | | | | |
|-----------------|--------------|--------------|----------------|--------------------|-------------|--------------|-----------------|-------------|--------------|-----------------|--------------|----|--|
| Apps Submitted | Apps Pending | Apps Running | Apps Completed | Containers Running | Memory Used | Memory Total | Memory Reserved | VCores Used | VCores Total | VCores Reserved | Active Nodes | De | |
| 2 | 0 | 2 | 0 | 2 | 1.88 GB | 117 GB | 0 B | 2 | 64 | 0 | 2 | 0 | |

| Scheduler Metrics | | | | | | | | | | | | | |
|--------------------------------|--------|-------------------------|------------------|--------------------------|--------------------------------|------------|---------|-----------------------|--------------------|---------------------|---------------------|-----|-----|
| Scheduler Type | | | | Scheduling Resource Type | | | | Minimum Allocation | | | | | |
| Capacity Scheduler | | | | [MEMORY] | | | | <memory:32, vCores:1> | | | | | |
| Showing 1 to 2 of 2 entries | | | | | | | | | | | | | |
| ID | User | Name | Application Type | Queue | StartTime | FinishTime | State | FinalStatus | Running Containers | Allocated CPU VCPUs | Allocated Memory MB | % c | Que |
| application_1544696070902_0004 | root | flink-oss-sample | Apache Flink | default | Thu Dec 13 20:35:17 +0800 2018 | N/A | RUNNING | UNDEFINED | 1 | 1 | 1024 | 0.9 | |
| application_1544696070902_0002 | hadoop | Thrift JDBC/ODBC Server | SPARK | default | Thu Dec 13 18:15:21 +0800 2018 | N/A | RUNNING | UNDEFINED | 1 | 1 | 896 | 0.7 | |

Showing 1 to 2 of 2 entries

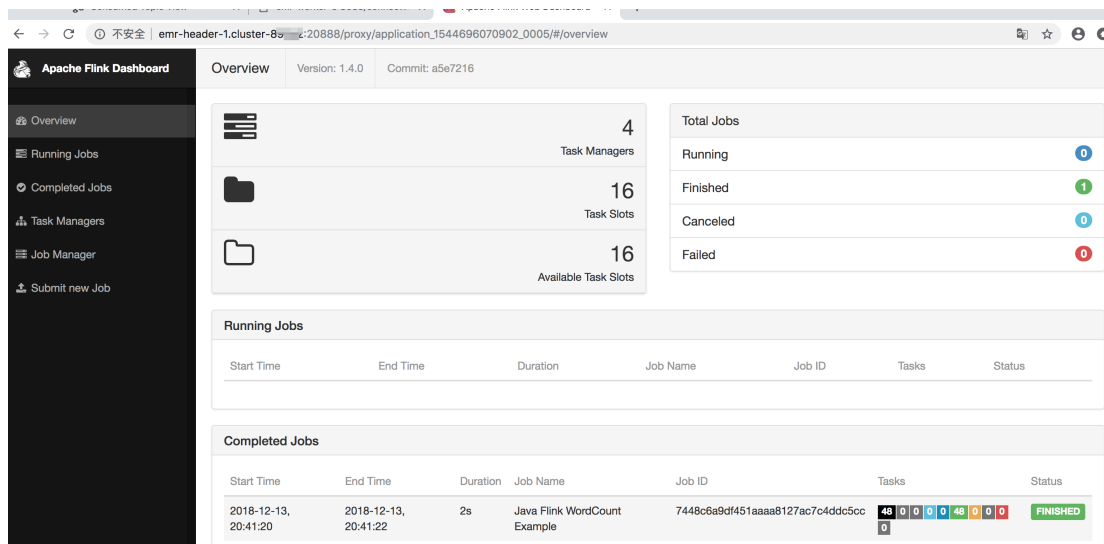
Click the tracking URL of the job to jump to Flink Dashboard for viewing the running job.



Application application_1544696070902_0004

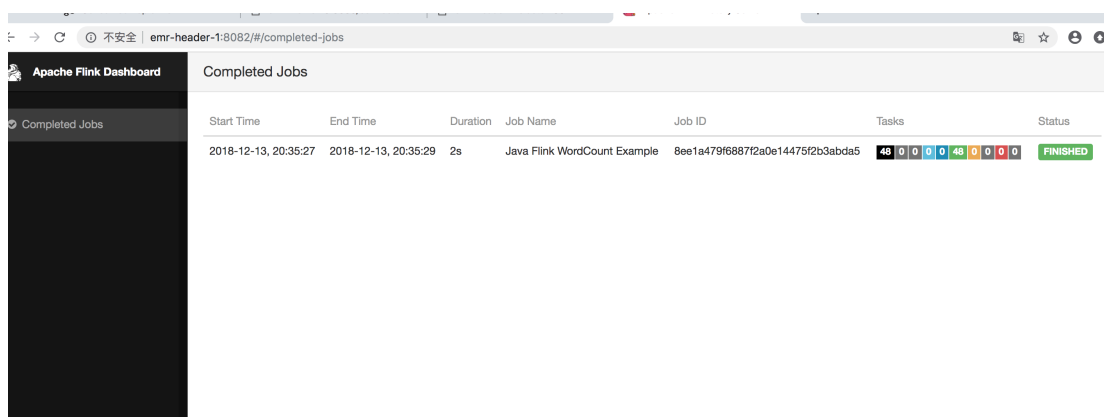
| Kill Application | |
|-----------------------------|---|
| User: | root |
| Name: | flink-oss-sample |
| Application Type: | Apache Flink |
| Application Tags: | |
| YarnApplicationState: | RUNNING: AM has registered with RM and started running. |
| FinalStatus Reported by AM: | Application has not completed yet. |
| Started: | Thu Dec 13 20:41:10 +0800 2018 |
| Elapsed: | 6sec |
| Tracking URL: | ApplicationMaster |
| Diagnostics: | |

Total Resource Preempted: <memory:0, vCores:0>
Total Number of Non-AM Containers Preempted: 0



- View history of jobs

You can view a list of all completed jobs by accessing `http://emr-header-1:8082`.



Result

We have completed consuming OSS data by running a Flink job on an EMR cluster.

12 Connect to ApsaraDB for HBase using E-MapReduce Hive

This topic describes how to connect E-MapReduce Hive and ApsaraDB for HBase. The analysis of HBase tables is based on the connection between Hive and ApsaraDB for HBase.



Note:

ApsaraDB for HBase will be integrated into Spark. We recommend that you use Spark to analyze HBase data at that time.

Preparations

- Purchase a Pay-As-You-Go EMR cluster and create configurations based on the actual scenarios. Note: Make sure ApsaraDB for HBase and the EMR cluster are in the same VPC. We recommend that you do not enable High Availability for the cluster.
- Add the IP addresses of all nodes in the EMR cluster to the whitelist of ApsaraDB for HBase.
- You can view the endpoint of ZooKeeper that is built in Hive in the ApsaraDB for HBase console.
- You need to contact the Alibaba Cloud team to open the HDFS ports of an ApsaraDB for HBase for you.

Procedures

1. Modify Hive configurations

- Go to the Hive configuration directory `/etc/ecm/hive-conf/`.
- Modify the `hbase-site.xml` file by setting the value of the `hbase.zookeeper.quorum` property to the endpoint of ZooKeeper that is built in HBase.

```
< property >
    < name > hbase . zookeeper . quorum </ name >
    < value > hb - bp1mhyea77 54bpigt - 001 . hbase . rds
    . aliyuncs . com , hb - bp1mhyea77 54bpigt - 002 . hbase . rds
    . aliyuncs . com , hb - bp1mhyea77 54bpigt - 003 . hbase . rds .
    aliyuncs . com </ value >
```

```
</ property >
```

2. Connect to an HBase table using a Hive table

Create a table in Hive using the HBase handler. By doing this, the same table is created in ApsaraDB for HBase as well.

a. Start the Hive command-line interface (CLI).

```
[root@emr-header-2 hive-conf]# hive
Logging initialized using configuration in file:/etc/emr/hive-conf-2.3.3-1.0.1/hive-log4j2.properties Async: true
Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution engine (i.e., spark, tez) or using Hive 1.X releases.
hive>
```

b. Use the following statement to create a table in Hive.

```
CREATE TABLE hive_hbase_table ( key int , value string )
```

```
STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
WITH SERDEPROPERTIES ("hbase.columns.mapping" = ":key,cf1:val")
```



```
TBLPROPERTIES (" hbase . table . name " = " hive_hbase _table ", " hbase . mapped . output . outputtable " = " hive_hbase _table ");
```

c. Insert data to the HBase table in Hive.

```
hive> insert into hive_hbase_table values(212,'bab');
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution engine (i.e. spark, tez) or using Hive 1.X releases.
Query ID = root_20181014173030_a0e99198-9aa5-4d29-b011-dc7b36365a20
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_1536221485395_0084, Tracking URL = http://emr-header-1.cluster-74778:20888/proxy/application_1536221485395_0084/
Kill Command = /usr/lib/hadoop-current/bin/hadoop job -kill job_1536221485395_0084
Hadoop job information for Stage-3: number of mappers: 1; number of reducers: 0
2018-10-14 17:30:40,833 Stage-3 map = 0%, reduce = 0%
2018-10-14 17:30:47,252 Stage-3 map = 100%, reduce = 0%, Cumulative CPU 3.66 sec
MapReduce Total cumulative CPU time: 3 seconds 660 msec
Ended Job = job_1536221485395_0084
MapReduce Jobs Launched:
Stage-Stage-3: Mapi: 1 Cumulative CPU: 3.66 sec HDFS Read: 11867 HDFS Write: 0 SUCCESS
Total MapReduce CPU Time Spent: 3 seconds 660 msec
OK
Time taken: 17.385 seconds
```

d. Verify that the HBase table has been created and the data has been inserted to the table.

```
[root@izbp16ku9i9clejitib6dz ~]# hbase shell
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/apps/t-apsara-hbase-1.4.6.3/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/apps/t-emr-hadoop-2.7.2.2/share/hadoop/common/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 1.4.6.3, r89ac288a5add370c07548ec3ce25f6e1f3210d23, Fri Jul 6 14:13:46 CST 2018

hbase(main):001:0> list
TABLE
A_TABLE
BAKE
BASE_TABLE
B_IDX
B_IDX1
B_IDX2
DEFAULT.TEST_IDX
MY_TABLE
PROD_METRICS
SYSTEM_MUTEX
SYSTEM.CATALOG
SYSTEM.FUNCTION
SYSTEM.SEQUENCE
SYSTEM.STATS
hive_hbase_table
tv
18 row(s) in 0.2110 seconds

hbase(main):004:0> scan 'hive_hbase_table'
COLUMN=CELL
ROW
212 column=cf1:val,timestamp=1539509446271,value=bab
1 row(s) in 0.0950 seconds
```

e. Write data to the HBase table using the put command.

```
hbase(main):005:0> put 'hive_hbase_table','132','cf1:val','acb'
0 row(s) in 0.0430 seconds
```

Select all data from the table in Hive.

```
hive> select * from hive_hbase_table;
OK
132      acb
212      bab
Time taken: 0.273 seconds, Fetched: 2 row(s)
```

f. Delete the table in Hive using the drop command. The table in HBase is deleted as well, which is to be verified in the subsequent step.

```
hive>  
>  
>  
> drop table hive_hbase_table;  
OK  
Time taken: 6.307 seconds 云栖社区 yq.aliyun.com
```

View the contents on the table in HBase using the scan command. An error message appears showing the table does not exist.

```
hbase(main):008:0* scan 'hive_hbase_table'  
ROW                                COLUMN+CELL  
ERROR: Unknown table hive_hbase_table! 云栖社区 yq.aliyun.com
```



Note:

Existing HBase tables can be connected using the Hive external tables. Deleting a Hive external table does not cause the deletion of the corresponding HBase table.

- g. Create a table in ApsaraDB for HBase and write test data to the table using the put command.

```
hbase(main):020:0> create 'hbase_table','f'
0 row(s) in 1.3010 seconds

=> Hbase::Table - hbase_table
hbase(main):021:0> put 'hbase_table','1122','f:col1','hello'
0 row(s) in 0.0190 seconds

hbase(main):022:0> put 'hbase_table','1122','f:col2','hbase'
0 row(s) in 0.0110 seconds
```

- h. Create a Hive external table to connect to an HBase table and select all data from the HBase table.

```
hive> create external table hbase_table(key int, col1 string, col2 string)
> STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
> WITH SERDEPROPERTIES ("hbase.columns.mapping" = "f:col1,f:col2")
> TBLPROPERTIES("hbase.table.name" = "hbase_table", "hbase.mapred.output.puttable" = "hbase_table");
OK
Time taken: 0.129 seconds
hive> select * from hbase_table;
OK
1122    hello    hbase
Time taken: 0.181 seconds, Fetched: 1 row(s)
hive> 
```

- i. Verify that deleting the Hive external table does not cause the deletion of the corresponding HBase table.

```
hive> drop table hbase_table;
OK
Time taken: 0.102 seconds
hive> 
```

```
hbase(main):023:0> scan 'hbase_table'
ROW                                COLUMN+CELL
1122                                column=f:col1, timestamp=1539510170256, value=hello
1122                                column=f:col2, timestamp=1539510181752, value=hbase
1 row(s) in 0.0160 seconds
```

Summary

For more operations on HBase using Hive, see [HBase Integration](#). The operations in this topic are based on Hive installed on an Alibaba Cloud EMR cluster. Operations based on Hive installed on a custom MapReduce cluster of ECS instances are similar. Note: Configuration items in the configuration file `hbase-site.xml` of Hive may be different from those of ApsaraDB for HBase. You only need to configure the

hbase . zookeeper . quorum **property for connecting to ApsaraDB for HBase using Hive.**

References

Alibaba Cloud Community: [Use E-MapReduce Hive to integrate with ApsaraDB for HBase](#)

13 Use EMR for real-time MySQL binlog transmission

This section describes how to use the SLS plug-in function of Alibaba Cloud and the E-MapReduce cluster to implement quasi-real-time transmission of MySQL binlog.

Basic architecture

RDS -> SLS -> Spark Streaming -> Spark HDFS

The preceding links contain three processes:

1. How to collect RDS binlog to SLS.
2. How to read and analyze the logs in SLS through Spark Streaming.
3. How to save the logs read and processed in the second link to Spark HDFS.

Prepare the environment

1. Install a MySQL database (using MySQL protocol, such as RDS and DRDS), and enable the log-bin function. Configure the binlog type to ROW mode. (RDS is enabled by default.)
2. Enable the SLS service.

Procedure

1. Check the MySQL database environment.
 - a. View whether the log-bin function is enabled.

```
mysql > show variables like " log_bin ";
+-----+-----+
| Variable_name | Value |
+-----+-----+
| log_bin       | ON    |
+-----+-----+
1 row in set (0.02 sec)
```

- b. View the binlog type.

```
mysql > show variables like " binlog_for mat ";
+-----+-----+
| Variable_name | Value |
+-----+-----+
| binlog_for mat | ROW   |
+-----+-----+
```

```
1 row in set (0.03 sec)
```

2. Add user permissions. You can also add user permissions directly from the RDS console.

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

3. Add the corresponding configuration file for the SLS service, and check if the data is collected properly.

- a. Add the corresponding project and logstore in the SLS console. For example, create a project named canaltest and a logstore named canal.
- b. Configure SLS: create a file named user_local_config.json under the directory of /etc/ilogtail.

```
{
  "metrics ": {
    "## 1.0 ## canaltest $ plugin - local ": {
      "aliuid ": "****",
      "enable ": true,
      "category ": "canal",
      "defaultEnd point ": "*****",
      "project_name ": "canaltest",
      "region ": "cn - hangzhou",
      "version ": 2
      "log_type ": "plugin",
      "plugin ": {
        "inputs ": [
          {
            "type ": "service_canal",
            "detail ": {
              "Host ": "****",
              "Password ": "****",
              "ServerID ": ****,
              "User ": "****",
              "DataBases ": [
                "yourdb"
              ],
              "IgnoreTables ": [
                "\\ S + _inner"
              ],
              "TextToString ": true
            }
          }
        ],
        "flushers ": [
          {
            "type ": "flusher_sls",
            "detail ": {}
          }
        ]
      }
    }
  }
}
```

```
}

```

The information such as host and password in detail is MySQL database information, and the user information is the user name authorized previously. AliUid, defaultEndpoint, project_name, and category are information related with users and SLS. Fill in the information according to your actual situation.

- c. Wait about 2 minutes to see if the log data has been uploaded successfully in the SLS console.

If the log data acquisition is not successful, view the acquisition log of SLS based on its prompt for troubleshooting.

4. Prepare and compile the code to jar package, and upload it to OSS.

- a. Copy the example code of EMR using Git and modify the code. The command is as follows: `git clone https://github.com/aliyun/aliyun-emapreduce-demo.git`. The example code includes the LoghubSample class, which is primarily used to capture and print data from SLS. The modified code is as below:

```
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 ,
```

```

loghubGroup pName ,
qendpoint ,
1 ,
accessKeyId ,
accessKeySecret ,
StorageLevel . MEMORY_AND _DISK )
loghubStream . foreachRDD ( rdd =>
    rdd . saveAsTextFile ("/ mysqlbinlog ")
)
ssc . start ()
ssc . awaitTermination ()
}
}

```

The main change is as follows: `loghubStream . foreachRDD (rdd => rdd . saveAsObjectFile ("/ mysqlbinlog "))`. When the example code is run in the EMR cluster, the data that flows out of Spark Streaming will be saved in HDFS of EMR.



Note:

- To run the example code locally, create a Hadoop cluster in the local environment in advance.
- Because the Spark SDK of EMR is updated, its example code is old and cannot directly transfer the AccessKey ID and AccessKey Secret of OSS in the parameter. You need to set the Spark SDK with the SparkConf constructor, as shown in the following figure:

```

trait RunLocally {
    val conf = new SparkConf (). setAppName ( getAppName ) .
    setMaster (" local [ 4 ]")
    conf . set (" spark . hadoop . fs . oss . impl ", " com .
    aliyun . fs . oss . nat . NativeOssFileSystem ")
    conf . set (" spark . hadoop . mapreduce . job . run - local
    ", " true ")
    conf . set (" spark . hadoop . fs . oss . endpoint ", "
    YourEndpoint ")
    conf . set (" spark . hadoop . fs . oss . accessKeyId ", "
    YourId ")
    conf . set (" spark . hadoop . fs . oss . accessKeySecret ",
    " YourSecret ")
    conf . set (" spark . hadoop . job . runlocal ", " true ")
    conf . set (" spark . hadoop . fs . oss . impl ", " com .
    aliyun . fs . oss . nat . NativeOssFileSystem ")
    conf . set (" spark . hadoop . fs . oss . buffer . dirs ", "/"
    mnt / disk1 ")
    val sc = new SparkContext ( conf )
    def getAppName : String
}

```



```
}
```

- During local debugging, you need to change `/mysqlbinlogloghubStream.foreachRDD(rdd => in rdd.saveAsObjectFile("/mysqlbinlog"))` to the local HDFS address.

b. Compile code.

After local debugging is complete, you can run the following command to package and compile the code:

```
mvn clean install
```

c. Upload the jar package.

Create a directory on an OSS instance where the bucket is `qiaozhou-EMR/jar`, and upload `examples-1.1-shaded.jar` under the directory of `/target/shaded` to the OSS directory through the OSS console or the SDK of OSS. The uploaded jar package address is `oss://qiaozhou-EMR/jar/examples-1.1-shaded.jar`. This address will be used later.

5. Create an EMR cluster and tasks, and run the execution plans.

- Create an EMR cluster in the EMR console, which takes about 10 minutes.
- Create a job of the Spark type.

Replace `SLS_endpoint` `$SLS_access_id` `$SLS_secret_key` with your actual values. Make sure that the order of the parameters is correct.

Otherwise, errors may be reported.

```
-- 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
```

- c. After the execution plan is created, bind jobs to the EMR cluster. Start to run the jobs.
- d. Search for the IP address of the master node.

After you login through SSH, run the following command:

```
hadoop fs -ls /
```

You can see the directory at the beginning of mysqlbinlog, and view the mysqlbinlog file with the following command:

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

```
[root@emr-header-1 ~]# hadoop dfs -ls /
DEPRECATED: Use of this script to execute hdfs command is deprecated.
Instead use the hdfs command for it.

SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/apps/ecn/service/hadoop/2.7.2-1.2.12/package/hadoop-2.7.2-1.2.12/share/hadoop/common/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/apps/ecn/service/tez/0.8.4/package/tez-0.8.4/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Found 5 items
drwxr-xr-x 2 hadoop hadoop 0 2018-01-03 23:42 /apps
drwxr-xr-x 2 hadoop hadoop 0 2018-01-03 23:44 /mysqlbinlog
drwxr-xr-x 2 hadoop hadoop 0 2018-01-03 23:44 /spark-history
drwxr-xr-x 2 root hadoop 0 2018-01-03 23:44 /tmp
drwxr-xr-x 2 hadoop hadoop 0 2018-01-03 23:43 /user
[root@emr-header-1 ~]# hadoop dfs -ls /mysqlbinlog
DEPRECATED: Use of this script to execute hdfs command is deprecated.
Instead use the hdfs command for it.

SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/apps/ecn/service/hadoop/2.7.2-1.2.12/package/hadoop-2.7.2-1.2.12/share/hadoop/common/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/apps/ecn/service/tez/0.8.4/package/tez-0.8.4/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Found 7 items
-rw-r--r-- 2 hadoop hadoop 0 2018-01-03 23:45 /mysqlbinlog/_SUCCESS
-rw-r--r-- 2 hadoop hadoop 2845 2018-01-03 23:44 /mysqlbinlog/part-00000
-rw-r--r-- 2 hadoop hadoop 15763 2018-01-03 23:44 /mysqlbinlog/part-00001
-rw-r--r-- 2 hadoop hadoop 346041 2018-01-03 23:44 /mysqlbinlog/part-00002
-rw-r--r-- 2 hadoop hadoop 311749 2018-01-03 23:44 /mysqlbinlog/part-00003
-rw-r--r-- 2 hadoop hadoop 292142 2018-01-03 23:44 /mysqlbinlog/part-00004
-rw-r--r-- 2 hadoop hadoop 139044 2018-01-03 23:44 /mysqlbinlog/part-00005
```

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

6. Troubleshoot.

If you don't see the normal results, you can troubleshoot problems in the running records of EMR.

14 Run Flume on a Gateway node to synchronize data

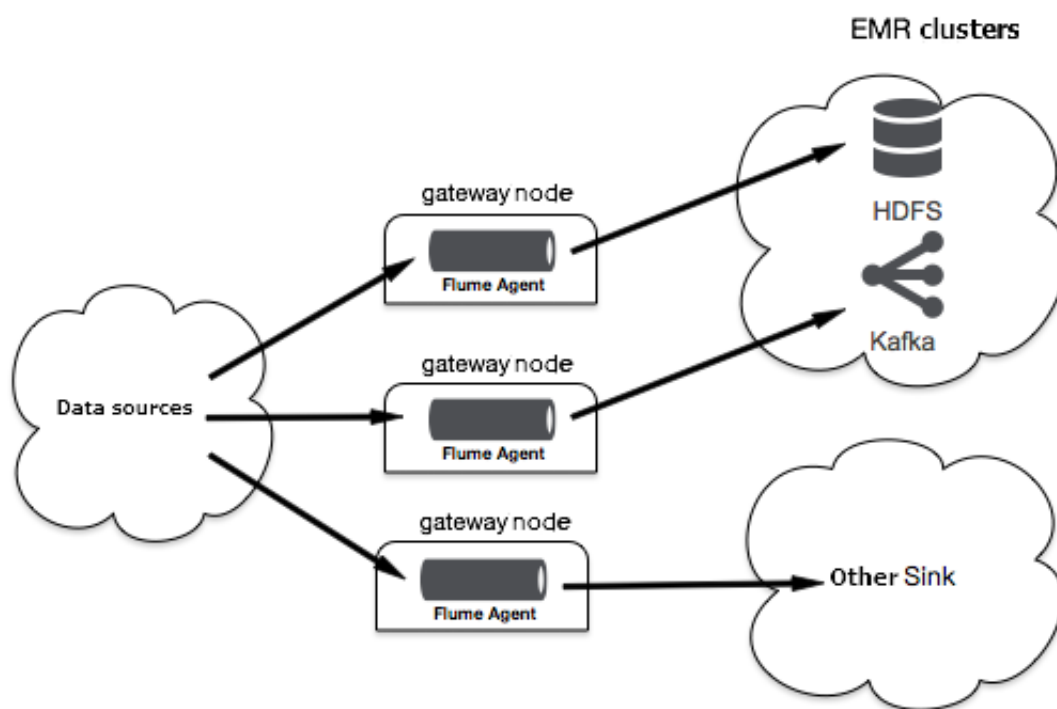
This topic describes how to run Flume on a Gateway node to synchronize data based on Alibaba Cloud E-MapReduce (EMR) V3.17.0 and later versions.

Background

EMR has supported Apache Flume since V3.16.0 and has supported default monitoring since V3.17.0.

- Basic data flows

Running Flume on Gateway nodes avoids the impact on EMR Hadoop clusters. Basic data flows that are streamed through Flume agents installed on Gateway nodes are shown in the following figure.



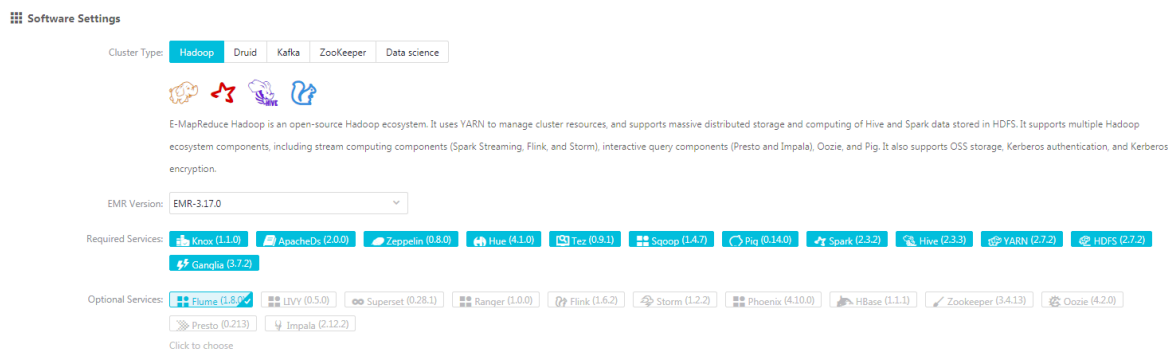
Prepare the environment

The test is performed using EMR that is deployed in the China East 1 (Hangzhou) region. The version of EMR is V3.17.0. The components required for this test are as follows.

- Flume: 1.8.0

You can use EMR to automatically create a Hadoop cluster. For more information, see [Create a cluster](#).

- Click **Create Cluster**, click **Flume** for the cluster type, and select **Flume** from **Optional Services**.



- Create a Gateway node and associate it to the Hadoop cluster created in the previous step.

Procedure

- **Run Flume**
 - The default path of Flume configuration files is `/etc/ecm/flume-conf`. See [Use Flume](#) for modifying the configuration file `flume.properties` for Flume agents. After the modification, use the following command to run a Flume agent.

```
nohup flume -ng agent -n a1 -f flume.properties &
```

- You can use the `-c` flag or the `--conf` flag to replace the default configuration file with a custom file. For example:

```
nohup flume -ng agent -n a1 -f flume.properties -c path-to-flume-conf &
```



Notice:

For more information, see [Use Flume](#). You need to add the `zookeeperQuorum` configuration item to the `flume.properties` configuration file when the Flume agents installed on Gateways use sinks to write data to HBase. For example:

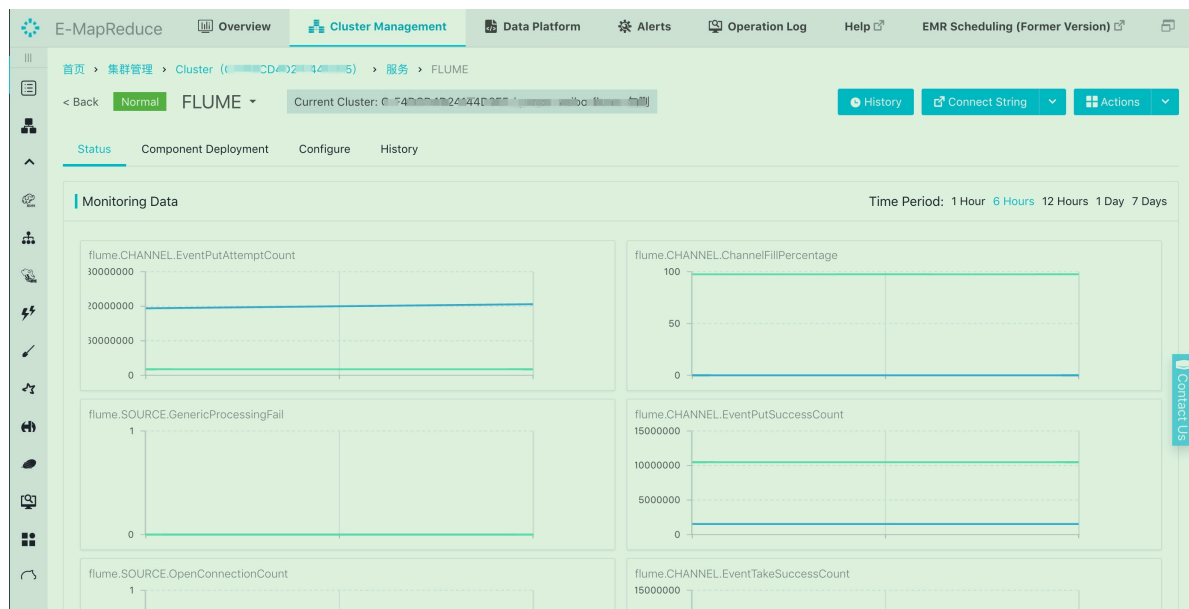
```
a1.sinks.k1.zookeeperQuorum = emr-header-1.cluster-46349:2181
```

The hostname of the ZooKeeper cluster `emr-header-1.cluster-46349` is the value of the `hbase.zookeeper.quorum` configuration item in the `/etc/ecm/hbase-conf/hbase-site.xml` file.

- View monitoring information

Monitoring data of Flume agents is displayed in the cluster console by default.

On the Clusters and Services page, click FLUME to jump to the cluster console as shown in the following figure.



Notice:

Monitoring data is classified by the components (sources, channels, or sinks) of Flume agents. For example, `CHANNEL.channel1` represents monitoring data of the `channel1` channel component. Note: Avoid using the same component name when configuring different agents.

Refer to the official Flume website for viewing monitoring data of Flume agents using Ganglia by creating proper configurations. After doing this, monitoring data of Flume agents will not be displayed in the console.

- **View logs**

By default, the log path of a Flume agent is `/mnt / disk1 / log / flume / ${ flume - agent - name } / flume . log` . You can modify the `/ etc / ecm / flume - conf / log4j . properties` configuration file to change the log path. However, we recommend that you do not change the default log path.

**Notice:**

A log path contains a Flume agent name. Provide a unique name for each agent to avoid logs of different agents to be stored in the same directory.

15 OSS ACL

This article introduces how to use RAM Access control to isolate the data of different sub-account.

Procedure

E-MapReduce supports using RAM to isolate the data of different sub-accounts. To do so, complete the following steps:

1. Log on to the [Alibaba Cloud RAM console](#).
2. Create a sub-account in RAM. For more information, see [Create a RAM user](#).
3. In the navigation pane on the left, click Policies to enter the policy management page.
4. Click Custom Policy.
5. In the upper-right corner, click Create Authorization Policy. Follow the steps there to create a policy. You can create as many policies as the number of authorization control sets that you need.

In this example, it is assumed that you need the following two sets of data control policies:

- Bucket name of the testing environment: test-bucket. The corresponding policy is as follows:

```
{
  "Version": "1",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "oss:ListBucket"
      ],
      "Resource": [
        "acs:oss:*:*:*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "oss:ListObject",
        "oss:GetObject",
        "oss:PutObject",
        "oss:DeleteObject"
      ],
      "Resource": [
        "acs:oss:*:*:* test - bucket ",
        "acs:oss:*:*:* test - bucket /*"
      ]
    }
  ]
}
```

```
}
]
}
```

- **Bucketname of the production environment: prod-bucket. The corresponding policy is as follows:**

```
{
  "Version ": " 1 ",
  "Statement ": [
    {
      "Effect ": " Allow ",
      "Action ": [
        " oss : ListBucket  s "
      ],
      "Resource ": [
        " acs : oss :*:~:~:"
      ]
    },
    {
      "Effect ": " Allow ",
      "Action ": [
        " oss : Listobject  s ",
        " oss : GetObject ",
        " oss : PutObject "
      ],
      "Resource ": [
        " acs : oss :*:~:~: prod - bucket ",
        " acs : oss :*:~:~: prod - bucket /*"
      ]
    }
  ]
}
```

6. Click Users.
7. Find the sub-account item which the policy is given to and click Manage.
8. In the navigation pane on the left, click User Authorization Policy.
9. In the upper-right corner, click Edit Authorization Policy.
10. Select and add authorization policies.
11. Click OK to complete the policy authorization of the sub-account.
12. In the upper-right corner, click User Details to enter the user details page of the sub-account.
13. Click Start Console Logon in the console logon management bar to initialize the authorization of the sub-account logon console.

Complete and use

Once you have completed all of the preceding steps, you can use the sub-account to log on to E-MapReduce. Note the following limitations:

- All buckets are displayed in the OSS interface for clusters, operations, and plan executions. However, you can only enter the authorized bucket.
- Only the content under the authorized bucket is displayed.
- The authorized bucket can only be read and written. Otherwise, an error is reported.


16 Configure a network connection for using Sqoop to transfer data from a database to an EMR cluster

When you need to transfer data from external databases to your EMR cluster, make sure that the networks are connected. This topic describes how to configure network connections for accessing ApsaraDB for RDS instances, user-created databases hosted on ECS, and on-premises databases.

ApsaraDB for RDS

- Classic network

We recommend that you use an EMR cluster deployed in the classic network to access a classic network RDS instance. You can set internal and public IP addresses for classic network RDS instances. Sqoop synchronizes data by running map tasks on the master node and worker nodes. However, only the master node of a classic network EMR cluster can access the public network. You need to access the internal IP address of the RDS instance for Sqoop. Make sure that the internal IP address of the EMR cluster is in the whitelist of the RDS instance.

| Basic Information | | Set Whitelist | Migrate Zone | ⌵ |
|--|--|---------------|--------------|---|
| Instance ID: <code>pgm-bp144871125y1</code> | Name: <code>pgm-bp144871125y1</code>  | | | |
| Instance Region and Zone: China (Hangzhou)ZoneG | Instance Type & Series Standard (High-availability) | | | |
| Intranet Address: <code>pgm-bp144871125y1.mysql.rds.aliyuncs.com</code> | Intranet Port: 3433 | | | |
| Apply for Internet Address Apply for Internet Address | | | | |
| Storage Type Local SSD Disk | | | | |
| Note: Use the connection string above to connect to the instance. You need to change the VIP in the connection string to the one used in your environment. | | | | |

For more information about how to create a classic network EMR cluster, see [Create a cluster](#).

- VPC

If the RDS is in a VPC network, you must specify a VPC network for the EMR cluster. We recommend that you use the same VPC for your EMR cluster and the

RDS instance to save time for configuring a network connection. Otherwise, use **Express Connect** to configure a network connection.

The screenshot displays the AWS EMR console configuration page. Under the 'Network Settings' tab, the 'Zone' is set to 'China (Hangzhou) Zone F', 'Network Type' is 'VPC', 'VPC' is 'vpc-12345678', 'VSwitch' is 'vsw-12345678', and 'Security Group Name' is 'sg-12345678'. The 'High Availability' toggle is turned off. Under the 'Instance' tab, the 'Master Instance' is selected, and the 'General Purpose' instance type is chosen. The instance configuration table shows various options, with 'General Purpose ecs.sn2ne ecs.sn2ne.xlarge vCore: 4 vCPU vMem: 16 GB Band Width: 1.536 Gbps' selected. The 'Current Master Node Type' is 'ecs.sn2.large'. The 'System Disk Type' is 'SSD' with a size of '120' GB. The 'Data Disk Type' is 'SSD' with a size of '80' GB. The 'Master Nodes' are set to '1'.

User-created database hosted on ECS

- Classic network

The process for accessing a classic network user-created database and a classic network RDS instance is similar. Use the classic network for the EMR cluster to access the internal IP address of the user-created database. Make sure that the ECS instance on which the database is deployed and the instances of the EMR cluster are in the same security group. In the ECS console, choose Security Groups > Manage Instances > Add Instance.

- VPC

The process for accessing a VPC user-created database and a VPC RDS instance is similar. Use a VPC for your EMR cluster. Make sure that the ECS instance where the database is deployed and the EMR cluster are in the same security group.

On-premises database

You can assign an EIP address to your EMR cluster and access the public IP address of the database. Or, you can use Express Connect to connect the VPCs to access the database.

- Associate an EIP

We recommend that you use a VPC EMR cluster if the on-premises database can be accessed over the public network. Create an EMR cluster in a VPC. Then, in the

ECS console, choose **Manage > Configuration Information > More > Bind EIP** and associate an EIP with each ECS instance. After the configurations, your cluster can access the public IP address of the on-premises database.

- **Express Connect**

If the on-premises database is not allowed to be accessed over the public network, create an EMR cluster in a VPC and use Express Connect to connect the on-premises IDC and the VPC. For more information about Express Connect, see [Express Connect](#).

17 Use E-MapReduce to submit a Spark Streaming job for consuming Kafka data

This topic describes how to create a Hadoop cluster and Kafka cluster by using E-MapReduce (EMR) and run a Spark Streaming job to consume Kafka data.

Prerequisites

- You have registered an Alibaba Cloud account. For more information, see [Create an Alibaba Cloud account](#).
- You have activated EMR.
- You have authorized the Alibaba Cloud account. For more information, see [#unique_19](#).

Context

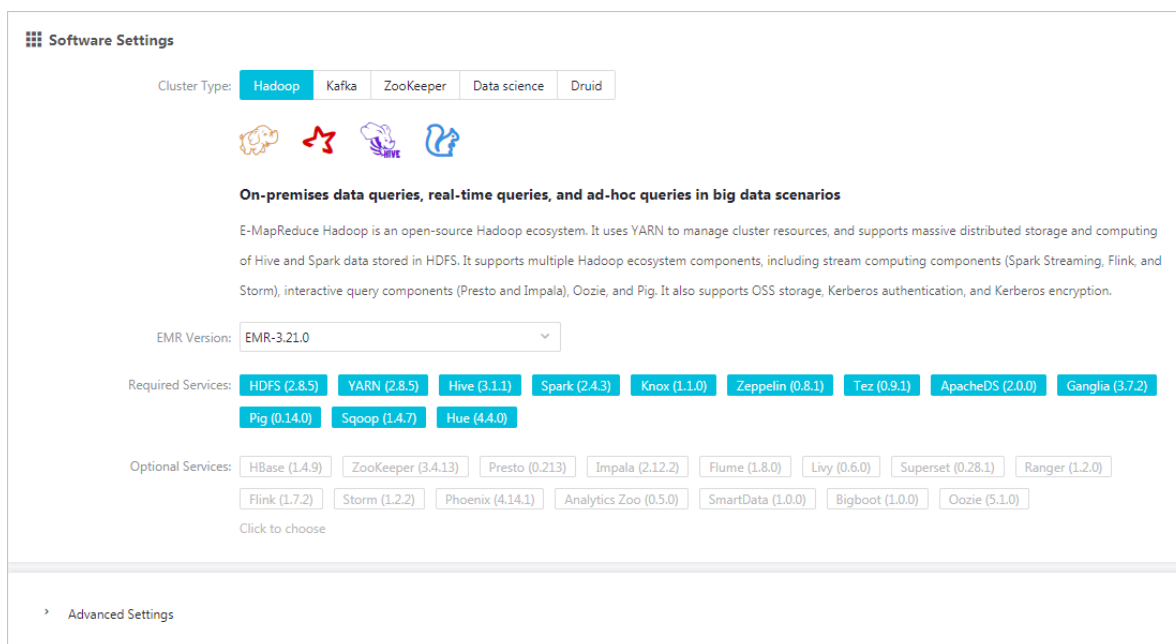
You always consume Kafka data in practical applications. In EMR, you can run a Spark Streaming job to consume Kafka data.

Step 1: Create a Hadoop cluster and Kafka cluster

We recommend that you specify the same security group for the Hadoop cluster as that of the Kafka cluster when creating the two clusters. If the clusters are linked to different security groups, the two clusters are not accessible by each other. You must modify the required settings of the security groups to allow mutual access.





1. Log on to the [Alibaba Cloud EMR console](#).

2. Create a Hadoop cluster. For more information, see [Create a cluster](#).



Software Settings

Cluster Type: **Hadoop** Kafka ZooKeeper Data science Druid

On-premises data queries, real-time queries, and ad-hoc queries in big data scenarios

E-MapReduce Hadoop is an open-source Hadoop ecosystem. It uses YARN to manage cluster resources, and supports massive distributed storage and computing of Hive and Spark data stored in HDFS. It supports multiple Hadoop ecosystem components, including stream computing components (Spark Streaming, Flink, and Storm), interactive query components (Presto and Impala), Oozie, and Pig. It also supports OSS storage, Kerberos authentication, and Kerberos encryption.

EMR Version: EMR-3.21.0

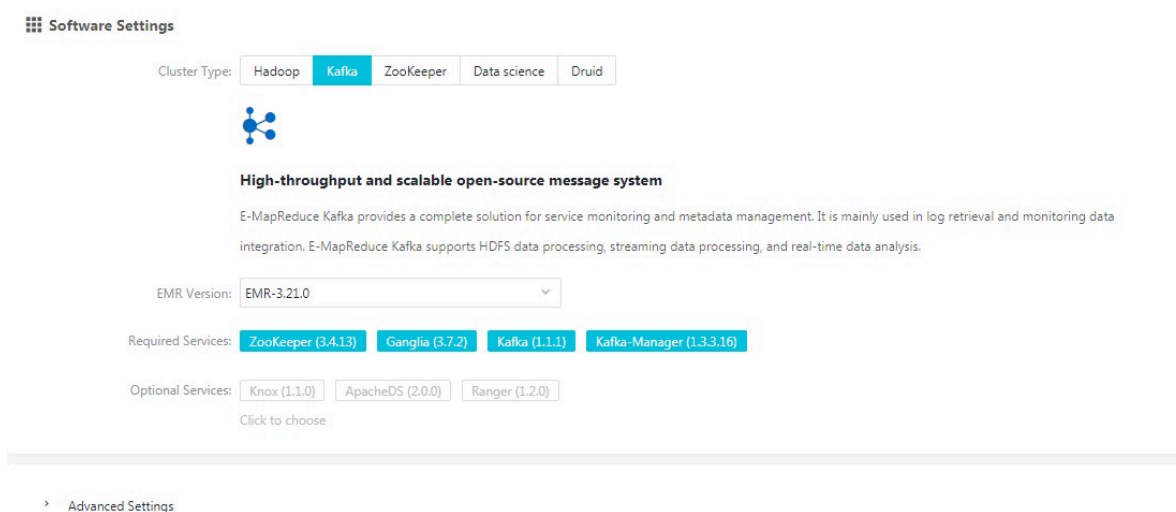
Required Services: **HDFS (2.8.5)** **YARN (2.8.5)** **Hive (3.1.1)** **Spark (2.4.3)** **Knox (1.1.0)** **Zeppelin (0.8.1)** **Tez (0.9.1)** **ApacheDS (2.0.0)** **Ganglia (3.7.2)**
Pig (0.14.0) **Sqoop (1.4.7)** **Hue (4.4.0)**

Optional Services: **HBase (1.4.9)** **ZooKeeper (3.4.13)** **Presto (0.213)** **Impala (2.12.2)** **Flume (1.8.0)** **Livy (0.6.0)** **Superset (0.28.1)** **Ranger (1.2.0)**
Flink (1.7.2) **Storm (1.2.2)** **Phoenix (4.14.1)** **Analytics Zoo (0.5.0)** **SmartData (1.0.0)** **Bigboot (1.0.0)** **Oozie (5.1.0)**

Click to choose


Advanced Settings

3. Create a Kafka cluster. For more information, see [Create a cluster](#).



Software Settings

Cluster Type: Hadoop **Kafka** ZooKeeper Data science Druid



High-throughput and scalable open-source message system

E-MapReduce Kafka provides a complete solution for service monitoring and metadata management. It is mainly used in log retrieval and monitoring data integration. E-MapReduce Kafka supports HDFS data processing, streaming data processing, and real-time data analysis.

EMR Version: EMR-3.21.0

Required Services: **ZooKeeper (3.4.13)** **Ganglia (3.7.2)** **Kafka (1.1.1)** **Kafka-Manager (1.3.3.16)**

Optional Services: **Knox (1.1.0)** **ApacheDS (2.0.0)** **Ranger (1.2.0)**

Click to choose

Advanced Settings

Step 2: Download a JAR file and upload it to the Hadoop cluster

In this example, the [Demo](#) project is customized and compiled to create a new JAR file. You need to upload the JAR file to the emr-header-1 instance of the Hadoop cluster.

1. Download the JAR file from [this link](#).
2. Log on to the [Alibaba Cloud EMR console](#).
3. On the Cluster Management tab, click the Cluster ID of the target cluster to enter the Hadoop cluster.

4. In the left-side navigation pane, select Instances and view the IP address of the `emr-header-1` instance in the Hadoop cluster.
5. Log on to the `emr-header-1` instance by using SSH.
6. Upload the JAR file to a directory of the `emr-header-1` instance.

**Note:**

The `/ home / hadoop` directory is specified as a repository for data storage in this case. After you upload the JAR file, we recommend that you keep the logon window open for later use.

Step 3: Create a topic on the Kafka cluster

You can create a topic in the EMR console. For more information, see [#unique_36](#). You can also log on to the `emr-header-1` instance and create a topic by using the CLI. In this example, you can create a topic named `test` with 10 partitions, 2 replicas.

1. Go to the [Alibaba Cloud EMR console](#).
2. On the Cluster Management tab, click the Cluster ID of the target Kafka cluster to open the Details page of the cluster.
3. In the left-side navigation pane, select Instances and view the IP address of the `emr-header-1` instance in the Kafka cluster.
4. Open a new shell in the SSH client and log on to the `emr-header-1` instance in the new shell.
5. Use the following command to create a topic.

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

**Note:**

After you create the topic, we recommend that you keep this logon window open for later use.

Step 4: Run a Spark Streaming job

After performing the preceding steps, you can run a Spark Streaming job on the Hadoop cluster. The following is an example of running a job to count the number of words for a data stream.

1. Go to the logon window of the emr-header-1 instance in the Hadoop cluster.

If you close the window, you need to log on again. For more information about the logon procedure, see [Step 2: Download a JAR file and upload it to the Hadoop cluster](#).

2. Use the following command to submit a job to the Kafka cluster for counting.

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

In the preceding command, the parameters after the name of the JAR file are described as follows:

- 192.168.xxx.xxx: indicates the internal or public IP address of a broker in the Kafka cluster. [Figure 17-1: List of components in the Kafka cluster](#) shows an example.
- test: indicates the name of the topic.
- 5: indicates the time interval.

Figure 17-1: List of components in the Kafka cluster

| Component Name | Status | Service Name | ECS ID | Instance Name | Role | IP |
|--------------------------------|-----------|--------------|-----------|---------------|--------|---------------------------------|
| Kafka Broker broker | STARTED | Kafka | i-xxxxxxx | emr-worker-1 | CORE | Internal Network IP:192.168.1.1 |
| Kafka Broker controller | STARTED | Kafka | i-xxxxxxx | emr-header-1 | MASTER | Internal Network IP:192.168.1.2 |
| Kafka Broker broker | STARTED | Kafka | i-xxxxxxx | emr-worker-2 | CORE | Internal Network IP:192.168.1.3 |
| Kafka Client | INSTALLED | Kafka | i-xxxxxxx | emr-worker-1 | CORE | Internal Network IP:192.168.1.1 |
| Kafka Client | INSTALLED | Kafka | i-xxxxxxx | emr-header-1 | MASTER | Internal Network IP:192.168.1.2 |
| Kafka Client | INSTALLED | Kafka | i-xxxxxxx | emr-worker-2 | CORE | Internal Network IP:192.168.1.3 |

Step 5: Use Kafka to publish messages

When you perform this step, ensure that the Spark Streaming job is running. After you start a Kafka producer, the number of words is displayed in a shell on a client instance of the Hadoop cluster. The value is updated in real time when you enter words into a shell on a client instance of the Kafka cluster.

1. Go to the logon window of the emr-header-1 instance.

If you close the window, you need to log on again. For more information about the logon procedure, see [Step 3: Create a topic on the Kafka cluster](#).

2. In the logon window of the client instance in the Kafka cluster, use the following command to start a producer.

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

3. When you enter words in the Kafka logon window, the number of words is displayed and updated in the Hadoop logon window in real time.

```
2. root@emr-header-1:/bin (ssh)
[root@emr-header-1 bin]# kafka-console-producer.s
h --topic test --broker-list emr-worker-1:9092
>abd abd abd ww
>badf d f fd
> ffhjkdf fhjhfd
>ff ff ff ff ff ff
>fffff ggg gggg gg ggg
>adsafb fdjksf fdfd
>dfdfg dfdfd
>ff ff ff ff ff
>fff fff fff fff
>ggg ggg ggg
>gg gg ggg ggg ggg
>jjj jj j
>ggg ggg ggg ggg
>aaa aa aa aa aa
>aa
>aa aa aa
>bb bb bb bb
>
```

```
1. root@emr-header-1:~ (ssh)
19/07/19 14:44:45 INFO TaskSetManager: Finished task 2.0 in stage 17.0 (TID 66)
19/07/19 14:44:45 INFO TaskSetManager: Finished task 0.0 in stage 17.0 (TID 64)
19/07/19 14:44:45 INFO TaskSchedulerImpl: Removed TaskSet 17.0, whose tasks have
19/07/19 14:44:45 INFO DAGScheduler: ResultStage 17 (print at KafkaSample.scala:
19/07/19 14:44:45 INFO DAGScheduler: Job 8 finished: print at KafkaSample.scala:
-----
Time: 1563518685000 ms
-----
(aa,3)
19/07/19 14:44:45 INFO JobScheduler: Finished job streaming job 1563518685000 ms
19/07/19 14:44:45 INFO JobScheduler: Total delay: 0.117 s for time 1563518685000
19/07/19 14:44:45 INFO ShuffledRDD: Removing RDD 19 from persistence list
19/07/19 14:44:45 INFO MapPartitionsRDD: Removing RDD 18 from persistence list
19/07/19 14:44:45 INFO BlockManager: Removing RDD 19
19/07/19 14:44:45 INFO BlockManager: Removing RDD 18
19/07/19 14:44:45 INFO MapPartitionsRDD: Removing RDD 17 from persistence list
19/07/19 14:44:45 INFO BlockManager: Removing RDD 17
19/07/19 14:44:45 INFO MapPartitionsRDD: Removing RDD 16 from persistence list
19/07/19 14:44:45 INFO KafkaRDD: Removing RDD 15 from persistence list
19/07/19 14:44:45 INFO ReceivedBlockTracker: Deleting batches:
```

Step 6: View the progress of the Spark Streaming job

After you run a Spark Streaming job, you can view the status of the job in the EMR console.

1. Go to the [EMR console](#).

2. On the Connect Strings page, click the link next to the Spark History Server UI service name to view the status of the Spark Streaming job. For more information, see [Access links and ports](#).

The screenshot shows the E-MapReduce console interface. The top navigation bar includes links for Overview, Cluster Management, Data Platform, Metadata, System Management, and Help. The left sidebar contains a menu with options like Cluster Overview, Cluster Management, Cluster Service, Cluster Resources, Instances, Cluster Scripts, Connect Strings, Auto Scaling, and Users. The main content area displays the 'Public Connect Strings' table.

| Service Name | Connect String | Instructions |
|-------------------------|--------------------|--------------|
| HDFS UI | https://knox.C-... | - |
| YARN UI | https://knox.C-... | - |
| Spark History Server UI | https://knox.C-... | - |
| Hue | http://knox.C-... | Description |
| Zeppelin | http://knox.C-... | Description |
| Ganglia UI | https://knox.C-... | - |

Below the table is the 'Event Timeline' section, showing 'Completed Jobs (1772, only showing 972)'. It includes a pagination bar and a table of job details.

| Job Id | Description | Submitted | Duration | Stages: Succeeded/Total | Tasks (for all stages): Succeeded/Total |
|--------|--|---------------------|----------|-------------------------|---|
| 1771 | Streaming job from [output operation 0, batch time 11:18:35] print at KafkaSample.scala:64 | 2019/07/18 11:18:35 | 6 ms | 1/1 (1 skipped) | 3/3 (10 skipped) |
| 1770 | Streaming job from [output operation 0, batch time 11:18:35] print at KafkaSample.scala:64 | 2019/07/18 11:18:35 | 30 ms | 2/2 | 11/11 |
| 1769 | Streaming job from [output operation 0, batch time 11:18:34] print at KafkaSample.scala:64 | 2019/07/18 11:18:34 | 3 ms | 1/1 (1 skipped) | 3/3 (10 skipped) |
| 1768 | Streaming job from [output operation 0, batch time 11:18:34] print at KafkaSample.scala:64 | 2019/07/18 11:18:34 | 10 ms | 2/2 | 11/11 |
| 1767 | Streaming job from [output operation 0, batch time 11:18:33] print at KafkaSample.scala:64 | 2019/07/18 11:18:33 | 4 ms | 1/1 (1 skipped) | 3/3 (10 skipped) |
| 1766 | Streaming job from [output operation 0, batch time 11:18:33] print at KafkaSample.scala:64 | 2019/07/18 11:18:33 | 23 ms | 2/2 | 11/11 |
| 1765 | Streaming job from [output operation 0, batch time 11:18:32] print at KafkaSample.scala:64 | 2019/07/18 11:18:32 | 4 ms | 1/1 (1 skipped) | 3/3 (10 skipped) |
| 1764 | Streaming job from [output operation 0, batch time 11:18:32] print at KafkaSample.scala:64 | 2019/07/18 11:18:32 | 12 ms | 2/2 | 11/11 |
| 1763 | Streaming job from [output operation 0, batch time 11:18:31] print at KafkaSample.scala:64 | 2019/07/18 11:18:31 | 6 ms | 1/1 (1 skipped) | 3/3 (10 skipped) |
| 1762 | Streaming job from [output operation 0, batch time 11:18:31] print at KafkaSample.scala:64 | 2019/07/18 11:18:31 | 24 ms | 2/2 | 11/11 |

18 Use Kafka Connect to migrate data

During streaming data processing, data synchronization between Kafka and other systems or data migration between Kafka clusters is often required. This topic describes how to use Kafka Connect to migrate data between Kafka clusters.

Prerequisites

- You have registered an Alibaba Cloud account. For more information, see [Create an Alibaba Cloud account](#).
- You have activated E-MapReduce.
- You have authorized the Alibaba Cloud account. For more information, see [Role authorization](#).

Context

Kafka Connect is a scalable and reliable tool for fast transmitting streaming data between Kafka and other systems. 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. In this way, you can migrate the data of the database and indirectly connect the database to a downstream streaming data processing system. Kafka Connect also provides a Representational State Transfer (REST) application programming interface (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. Compared with the standalone mode, the distributed mode is more scalable and fault-tolerant. It is the most commonly used mode and the recommended mode for the production environment.

This topic describes how to call the REST 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 target Kafka cluster in E-MapReduce. Kafka Connect is installed on the task node. Therefore, a task node must be created in the target Kafka cluster. Kafka Connect is started on the task node by default after the cluster is created. The port number is 8083.

We recommend that you add the source Kafka cluster and the target Kafka cluster to the same security group. If the source Kafka cluster and the target Kafka cluster belong to different security groups, the two clusters are not accessible to each other by default. You must modify the required settings of the security groups to allow mutual access.


1. Log on to the [Alibaba Cloud E-MapReduce console](#).
2. Create the source Kafka cluster and the target Kafka cluster. For more information, see [#unique_34](#).

**Note:**

When creating the target Kafka cluster, you must configure a task instance, that is, a task node.

Software Settings

Cluster Type: Hadoop **Kafka** ZooKeeper Data science Druid



High-throughput and scalable open-source message system

E-MapReduce Kafka provides a complete solution for service monitoring and metadata management. It is mainly used in log retrieval and monitoring data integration. E-MapReduce Kafka supports HDFS data processing, streaming data processing, and real-time data analysis.

EMR Version: EMR-3.21.0

Required Services: **ZooKeeper (3.4.13)** **Ganglia (3.7.2)** **Kafka (1.1.1)** **Kafka-Manager (1.3.3.16)**

Optional Services: Knox (1.1.0) ApacheDS (2.0.0) Ranger (1.2.0)

[Click to choose](#)

[Advanced Settings](#)

Step 2: Create a topic for storing the data to be migrated

Create a topic named connect in the source Kafka cluster.

1. Use Secure Shell (SSH) to log on to the header node of the source Kafka cluster. In this example, the header 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 -- replicatio n - 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 performing the preceding operations, keep the logon window for later use.

Step 3: Create a Kafka Connect connector

On the task node of the target Kafka cluster, run the `curl` command to create a Kafka Connect connector by using JavaScript Object Notation (JSON) data.

1. Use SSH to log on to the task node of the target Kafka cluster. In this example, the task node is `emr-worker-3`.
2. Optional: Customize Kafka Connect configuration.

Go to the Configuration page of the Kafka service under the target Kafka cluster. Customize the `offset.storage.topic`, `config.storage.topic`, and `status.storage.topic` parameters in `connect-distributed.properties`. For more information, see [#unique_38](#).

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 partition and replication factor that are saved in `/etc/ecm/kafka-conf/connect-distributed.properties`.

3. 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.ByteArrayC
onverter", "value.converter": "org.apache.kafka.connect
.converters.ByteArrayC onverter", "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 indicates the name of the Kafka Connect connector to create, which is `connect - test` in this example. The `config` field needs to be configured based on your actual requirements. The following table describes the key variables of the `config` field.

| Variable | Description |
|--|---|
| <code>\${source-topic}</code> | The topics for storing the data to be migrated in the source Kafka cluster. For example, <code>connect</code> . Separate multiple topics with commas (,). |
| <code>\${dest-topic}</code> | The topics to which the data is migrated in the target Kafka cluster. For example, <code>connect.replica</code> . |
| <code>\${src-kafka-curator-hostname}</code> | The internal IP address of the node where the ZooKeeper service is installed in the source Kafka cluster. |
| <code>\${dest-kafka-curator-hostname}</code> | The internal IP address of the node where the ZooKeeper service is installed in the target Kafka cluster. |



Note:

After performing the preceding operations, keep the logon window for later use.

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 in normal status.

1. Return to the logon window on the task node of the target Kafka cluster. In the 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":"192.168.1.100:8083"},"tasks":[{"state":"RUNNING","id":0,"worker_id":"192.168.1.100:8083"},"type":"source"}][root@emr-worker-3 ~]#
```

Make sure that the Kafka Connect connector (connect-test in this example) is in the RUNNING status.

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/tasks
{"name":"connect-test","connector":{"state":"RUNNING","worker_id":"192.168.1.100:8083"},"tasks":[{"state":"RUNNING","id":0,"worker_id":"192.168.1.100:8083"},"type":"source"}][root@emr-worker-3 ~]# curl emr-worker-3:8083/connectors/connect-test/tasks
{"id":0,"connector":"connect-test","task_id":0,"config":{"connector.class":"EMRReplicatorSourceConnector","src.zookeeper.connect":"emr-header-1.cluster-127390:2181","topic.rename.format":"connect.replica","dest.zookeeper.connect":"emr-header-1.cluster-127499:2181","task.class":"org.apache.kafka.connect.replicator.EMRReplicatorSourceTask","src.kafka.max.poll.records":"300","name":"connect-test","task_id":"connect-test-0","value.converter":"org.apache.kafka.connect.converters.ByteArrayConverter","key.converter":"org.apache.kafka.connect.converters.ByteArrayConverter","src.kafka.bootstrap.servers":"192.168.1.100:9092","topic.whitelist":"connect","partition.assignment.strategy":"org.apache.kafka.connect.replicator.EMRReplicatorSourceTask"},"type":"source"}][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 on the header node of the source Kafka cluster. In this example, the header 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
```

```
[root@emr-header-1 ~]# kafka-producer-perf-test.sh --topic connect --num-records 100000 --throughput 5000 --record-size 1000 --producer-props bootstrap.servers=emr-header-1:9092
24992 records sent, 4997.4 records/sec (4.77 MB/sec), 4.5 ms avg latency, 149.0 max latency.
25025 records sent, 5005.0 records/sec (4.77 MB/sec), 0.8 ms avg latency, 25.0 max latency.
25000 records sent, 5000.0 records/sec (4.77 MB/sec), 0.7 ms avg latency, 22.0 max latency.
24972 records sent, 4884.0 records/sec (4.66 MB/sec), 0.8 ms avg latency, 122.0 max latency.
100000 records sent, 4901.960784 records/sec (4.67 MB/sec), 1.73 ms avg latency, 393.00 ms max latency, 1 ms 50th, 4 ms 95th, 29 ms 99th, 77 ms 99.9th.
[root@emr-header-1 ~]#
```

Step 6: View the result of data migration

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

1. Return to the logon window on the task node of the target 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
```

```
[root@emr-worker-3 ~]# kafka-consumer-perf-test.sh --topic connect.replica --broker-list emr-header-1:9092 --messages 1000000  
start.time, end.time, data.consumed.in.MB, MB.sec, data.consumed.in.nMsg, nMsg.sec, rebalance.time.ms, fetch.time.ms, fetch.MB.sec, fetch.nMsg.sec  
2019-07-22 10:13:17:855, 2019-07-22 10:13:32:055, 95.3674, 6.7160, 1000000, 7042.2535, 3019, 11181, 8.5294, 8943.7439  
[root@emr-worker-3 ~]#
```

According to the command output in the preceding figure, the 100,000 messages sent to the source Kafka cluster are migrated to the target Kafka cluster.

Summary

This topic describes and demonstrates how to use Kafka Connect to migrate data between Kafka clusters. For more information about how to use Kafka Connect, see [Kafka official website](#) and [REST API](#).