

Elastic MapReduce

EMR Development Guide

Product Documentation



Copyright Notice

©2013-2022 Tencent Cloud. All rights reserved.

Copyright in this document is exclusively owned by Tencent Cloud. You must not reproduce, modify, copy or distribute in any way, in whole or in part, the contents of this document without Tencent Cloud's the prior written consent.

Trademark Notice



All trademarks associated with Tencent Cloud and its services are owned by Tencent Cloud Computing (Beijing) Company Limited and its affiliated companies. Trademarks of third parties referred to in this document are owned by their respective proprietors.

Service Statement

This document is intended to provide users with general information about Tencent Cloud's products and services only and does not form part of Tencent Cloud's terms and conditions. Tencent Cloud's products or services are subject to change. Specific products and services and the standards applicable to them are exclusively provided for in Tencent Cloud's applicable terms and conditions.

Contents

EMR Development Guide

Hadoop Development Guide

HDFS Common Operations

HDFS Federation Management Development Guide

HDFS Federation Management

Submitting MapReduce Tasks

Automatically Adding Task Nodes Without Assigning ApplicationMasters

YARN Task Queue Management

Practices on YARN Label Scheduling

Hadoop Best Practices

Using API to Analyze Data in HDFS and COS

Dumping YARN Job Logs to COS

Spark Development Guide

Spark Environment Info

Using Spark to Analyze Data in COS

Using Spark Python to Analyze Data in COS

SparkSQL Tutorial

Integrating Spark Streaming with Ckafka

Practices on Dynamic Scheduling of Spark Resources

Spark Integration with Kafka

Spark Dependencies in Each EMR Version

Hbase Development Guide

Using HBase Through API

Using Hbase with Thrift

Spark on Hbase

MapReduce on Hbase

Phoenix on Hbase Development Guide

Phoenix Client Usage

Phoenix JDBC Usage

Phoenix Best Practices

Hive Development Guide

Hive's Support for LLAP

Basic Hive Operations

Connecting to Hive Through Java

Connecting to Hive Through Python

Mapping Hbase Tables

Hive Best Practices

Creating Databases Based on COS

Practices on Loading JSON Data to Hive

Hive Metadata Management

Presto Development Guide

Presto Web UI

Connector

Analyzing Data in COS

Sqoop Development Guide

Import/Export of Relational Database and HDFS

Incremental Data Import into HDFS

Importing and Exporting Data Between Hive and TencentDB for MySQL

Hue Development Guide

Hue Overview

Hue Best Practices

Oozie Development Guide

Flume Development Guide

Flume Overview

Storing Kafka Data in Hive Through Flume

Storing Kafka Data in HDFS or COS Through Flume

Storing Kafka Data in Hive Through Flume

Kerberos Development Guide

Kerberos HA Support

Kerberos Overview

Kerberos Use Instructions

Accessing Hadoop Secure Cluster

Sample Connection from Hadoop to Kerberos

Knox Development Guide

Knox Development Guide

Integrating Knox with Tez

Integrating Knox with Tez v0.8.5

Integrating Knox with Tez v0.10.0

Alluxio Development Guide

Alluxio Development Documentation

Common Alluxio Commands

Mounting File System to Unified Alluxio File System

Using Alluxio in Tencent Cloud

Support for COS Transparent-URI

Support for Authentication

Kylin Development Guide

Kylin Overview

Livy Development Guide

Livy Overview

Kyuubi Development Guide

Kyuubi Overview

Kyuubi Best Practices

Zeppelin Development Guide

Zeppelin Overview

Zeppelin Interpreter Configuration

Hudi Development Guide

Hudi Overview

Superset Development Guide

Superset Overview

Impala Development Guide

Impala Overview

Impala OPS Manual

Analyzing Data on COS/CHDFS

ClickHouse Development Guide

ClickHouse Overview

ClickHouse Usage

ClickHouse SQL Syntax

Client Overview

Getting Started

ClickHouse Data Import

COS Data Import

HDFS Data Import

Kafka Data Import

MySQL Data Import

ClickHouse OPS

Monitoring

Configuration Description

Log Description

Data Backup

System Table Description

Access Permission Control

ClickHouse Data Migration Guide

Druid Development Guide

Druid Overview

Druid Usage

Ingesting Data from Hadoop in Batches

Ingesting Data from Kafka in Real Time

TensorFlow Development Guide

TensorFlow Overview

TensorFlowOnSpark Overview

Jupyter Development Guide

Jupyter Notebook Overview

Kudu Development Guide

Kudu Overview

Ranger Development Guide

Ranger Overview

Ranger User Guide

Integrating HDFS with Ranger

Integrating YARN with Ranger

Integrating HBase with Ranger

Integrating Presto with Ranger

Doris Development Guide

Doris Overview

User Guide

Creating a User

Creating a Data Table and Importing Data

Querying Data

Kafka Development Guide

Kafka Overview

Use Cases

Kafka Usage

Iceberg Development Guide

StarRocks Development Guide

StarRocks Overview

User Guide

EMR Development Guide

Hadoop Development Guide

HDFS Common Operations

Last updated : 2020-12-21 17:21:45

Tencent Cloud Elastic MapReduce (EMR) Hadoop service has integrated COS. If you enable COS when purchasing an EMR cluster, you can manipulate the data stored in COS using common Hadoop commands as shown below:

```
#cat data
hadoop fs -cat /usr/hive/warehouse/hivewithhdfs.db/record/data.txt
#Modify the directory or file permission
hadoop fs -chmod -R 777 /usr
#Change the file or directory owner
hadoop fs -chown -R root:root /usr
#Create a folder
hadoop fs -mkdir <paths>
#Send a local file to HDFS
hadoop fs -put <localsrc> ... <dst>
#Copy a local file to HDFS
hadoop fs -copyFromLocal <localsrc> URI
#View the storage usage of a file or directory
hadoop fs -du URI [URI ...]
#Delete a file
hadoop fs -rm URI [URI ...]
#Set the number of copies of a directory or file
hadoop fs-setrep [-R] [-w] REP PATH [PATH ...]
#Check for bad blocks of a cluster file
hadoop fsck <path> [-move | -delete | -openforwrite] [-files [-blocks [-locations
| -racks]]]
```

For more HDFS commands, see the [community documentation](#). If your cluster is a high-availability cluster (dual-namenode), you can see which namenode is active by running following commands:

```
#nn1 is the ID of the namenode; usually nn1 and nn2.
hdfs haadmin -getServiceState nn1
#View the current cluster report
hdfs dfsadmin -report
#namenode exits safe mode
hdfs dfsadmin -safemode leave
```

HDFS Federation Management Development Guide

Last updated : 2022-05-16 12:52:25

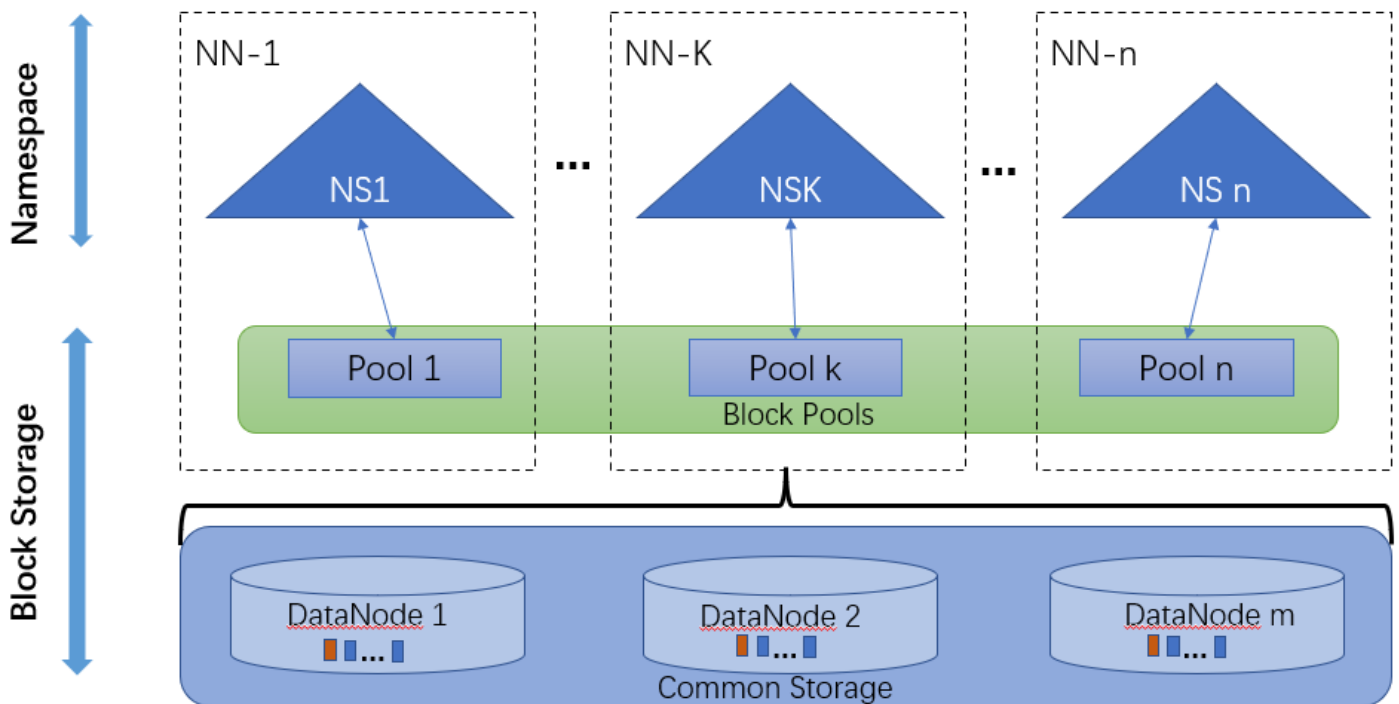
Type Selection for HDFS Federation Management

Note :

The HDFS federation management feature is currently made available through an allowlist. To use it, [submit a ticket](#) for application.

HDFS federation management architecture

In order to scale the name service horizontally, HDFS federation uses multiple independent NameNodes/nameSpaces. The DataNodes are used as common storage for blocks by all the NameNodes. Each DataNode registers with all the NameNodes in the cluster. DataNodes send periodic heartbeats and block reports. They also handle commands from the NameNodes.



For more information, see [HDFS Federation](#).

How ViewFs federation works

To make it easier to manage multiple namespaces, HDFS ViewFs federation uses the classic client-side mount table (a feature of open-source ViewFs). Different paths of applications are mapped to specific NameServices on the client, so as to achieve storage separation or performance separation. This mechanism distributes files and loads but requires more human intervention (clear planning) to implement ideal load balancing.

How router-based federation works

Router-based federation provides a software layer to manage multiple namespaces. Compared with ViewFs which maintains the mount table information on the client, it is truly transparent to the client, because the mapping information will be additionally saved and persisted.

HDFS Federation Management Configuration

You need to consider two factors: the required number of NameServices and the mount method of the business data directory.

Planning principles for the number of NameServices

1. One NameService can store up to 100 million files securely. After this limit is exceeded, its access speed and read/write throughput will be greatly reduced. Therefore, if you expect to store more than 100 million files, you need to add an extra NameService.
2. For an application that reads/writes HDFS heavily, it is necessary to allocate a separate NameService to process its requests. This ensures that the data in the application can exclusively use all the processing capabilities of the NameService and avoids the impact on other applications.
3. For an application that demands high reliability, you can allocate an independent NameService to it, because it may be prevented from accessing HDFS and thus rendered unstable if the reads/writes of other applications are too frequent.

Planning principles for the mount method of business data directories

1. Mount the data directories of the services associated with the business data to the same NameService, because cross-NameService file reads/writes are slow and may reduce the application's file storage performance.
2. For a service that involves a large amount of data but is not associated with other services, you can directly use only one NameService.
3. For a small business, we recommend you mount its directories directly to the default NameService (HDFS\${clusterid}), so as to eliminate the need of data migration and simplify the federation configuration.
4. We recommend you map only first-level directories to NameServices in order to reduce the configuration complexity.

Scheme Comparison

ViewFs federation configuration method

1. Directly use ViewFs

Strengths: Unified views are provided, and different applications can be used in the same way.

Shortcomings: Changes to ViewFs mount tables require all applications using the cluster to read the latest mount point synchronously.

2. Specify the NameService

Strengths: You don't need to modify the configurations of all applications.

Shortcomings: You need to specify directories for different components and applications, which makes the coupling of component paths more complicated.

Router-based federation configuration method

1. Directly use router-based federation

Strengths: Unified views are provided, and different applications can be used in the same way. Changes to router-based federation mount tables take effect directly, so applications using the cluster don't need to update the mount tables synchronously.

Shortcomings: The DFSRouter forwarding layer is added, which slightly affects the performance.

2. Specify the NameService

Strengths: You don't need to modify the configurations of all applications.

Shortcomings: You need to specify directories for different components and applications, which makes the coupling of component paths more complicated.

HDFS Federation Management

Last updated : 2022-05-16 12:52:25

Overview

HDFS federation management is an HDFS federated cluster deployment and management feature, including NameService management and mount table management. Federation management is supported for Hadoop-type clusters in HA mode. There are two federation types to choose from: ViewFs federation and router-based federation, and the federation type cannot be changed once selected. A router node will be used to deploy an added NameNode. This router node does not support termination and role start/stop at the node level.

Note :

1. The HDFS federation management feature is currently made available through an allowlist. To use it, [submit a ticket](#) for application.
2. All EMR versions support ViewFs federation. As only HDFS v2.9.0 and later support router-based federation, only EMR v3.x.x and later support router-based federation.
3. Suspending the NameNode role process on a federated node on the **Role Management** page may affect the cluster scaling, so you need to resume the process first before scaling the cluster.
4. After NameNodes are federated, the `fs.defaultFS` configurations on different NameNodes will differ. When delivering the configuration of the HDFS `core-site.xml` file, do not select the cluster level; otherwise, the `fs.defaultFS` values on NameNodes will be overwritten. Other configuration files will not be affected.

Directions

1. Log in to the [EMR console](#) and click a **Cluster ID/Name** on the **Cluster List** page to enter the **cluster details** page.
2. On the cluster details page, click **Cluster Service** and select **Operation > Federation Management** in the top-right corner of the HDFS component block to enter the **Federation Management** page.

Cluster Service / HDFS ▾ Help Documentation 📄

Service Status Role Management Configuration Management Configuration Record **Federation Management**

📘 1. After you add a NameService, it cannot be deleted and its name cannot be modified. For more information, see [Federation Management](#).

2. You are advised to add Router nodes first as federated nodes.

NameService Management

[Add NameService](#) Enter an IP or Name 🔍 ↻

NameService	Role Name	Resource ID/Node ID	Node IP
HDFS1000365	NameNode(主)	emr-vm-8vxxwqotr/ins-8xgubvca	10.0.0.144
	NameNode(备)	emr-vm-gt97uq27/ins-l64lsthm	10.0.0.143
	zkfc(主)	emr-vm-8vxxwqotr/ins-8xgubvca	10.0.0.144
	zkfc(备)	emr-vm-gt97uq27/ins-l64lsthm	10.0.0.143

Mount Table Management

[Add Mount Table](#)

Global Path	Target NameService	Target Path	Operation
No data yet			

Total items: 0 10 / page 1 / 1 page

3. Click **Add NameService** to create an HDFS federation. You need to enter the NameService name and select the federation type, NameNode, and DFSRouter (for router-based federation only).

Add NameService ✕

📘 How federation works: ViewFs federation maps different paths of applications to the specific federation via the client side to achieve storage or performance separation. Router-based federation uses proxies to implement federation. It provides a software layer to manage multiple namespaces and separate the configuration and implementation of the mount table from the client, making it fully transparent to the client.

1. After you add a NameService, it cannot be deleted and its name cannot be modified. For more information, see [Federation Management](#).

2. If there are not enough nodes for you to select from, add Router nodes in Resource Management before creating a federation.

Add NameService ⓘ *

Federation Type ⓘ ViewFs Federation

Nodes to Deploy NameNode ⓘ

Resource Name/...	Node IP	Configuration	Processes deployed	Creation time
No data yet				

Selected nodes: 0

4. Select **Add Federated Node**.

Federated nodes adopt router nodes in the cluster. Therefore, you need to add a router node on the **Resource Management** page first and then set it to a federated node. The NameNode process requires two nodes, on each of which the NameNode and ZKFC processes will be deployed.

When creating a router-based federation for the first time, you need to select at least two nodes to deploy the DFSRouter process. When creating another federation, you can reuse the DFSRouter nodes, and the number of the nodes can be greater than or equal to zero.

Add NameService

How federation works: ViewFs federation maps different paths of applications to the specific federation via the client side to achieve storage or performance separation. Router-based federation uses proxies to implement federation. It provides a software layer to manage multiple namespaces and separate the configuration and implementation of the mount table from the client, making it fully transparent to the client.

- After you add a NameService, it cannot be deleted and its name cannot be modified. For more information, see [Federation Management](#).
- If there are not enough nodes for you to select from, add Router nodes in Resource Management before creating a federation.

Used to deploy NameNode and ZKFailoverController processes. You need to select 2 nodes.

Required

ViewFs Federation Router-based Federation

Nodes to Deploy NameNode

Enter a node IP/resource name/resource ID

Resource Name/...	Node IP	Configuration	Processes deployed	Creation time
No data yet				

Selected nodes: 0

Nodes to Deploy DFSRouter

Enter a node IP/resource name/resource ID

<input type="checkbox"/> Resource Name/...	Node IP	Configuration	Processes deployed	Creation time
No data yet				

Selected nodes: 0

Confirm **Cancel**

Note :

- For HDFS versions earlier than 3.3.0, when you successfully add a NameService to a router-based federation (**not for the first time**), you need to restart the old DFSRouter process on the **Role Management** page (preferably during off-peak hours). For HDFS v3.3.0 and later which support hot loading the configuration, this operation is not required.
- After adding a federated NameService to a cluster with Kerberos enabled, you need to restart the YARN ResourceManager first (preferably during off-peak hours) before you can use the files on the new NameService for jobs submitted to YARN.

- The NameService name cannot be modified or deleted once set and cannot be system keywords such as "nsfed", "haclusterX", and "ClusterX".

5. Add a mount table.

You can add a mount table only after successfully adding a NameService. To reduce the configuration complexity, we recommend you map only first-level directories such as "/tmp", "/user", and "/srv" to NameServices. You can add multiple mount paths at a time.

- Path: Path name of the unified ViewFs or router-based federation namespace, also known as the mount point.
- Target NameService: The NameService corresponding to the real path to which the mount point maps.
- Target path: The real path on the corresponding NameService, whose name can be different from that of the global path.

Add Mount Table ✕

Global Path*	Target NameService*	Target Path*	Operation
Enter a global path	Please select ▼	Enter a target path	Delete
+ Add			
Confirm Cancel			

Note :

- Path direction:
 - i. Log in to the NameNode and run `hdfs dfs -ls /` to point to the path under the namespace managed by the NameNode. For ViewFs federation, you need to use `hdfs dfs -ls ViewFs://ClusterX/` to point to the global path; for router-based federation, use `hdfs dfs -ls hdfs://nsfed/` instead.
 - ii. Log in to another node, such as the router node serving as a client. `hdfs dfs -ls /` points to the global path.
- The data of all business components needs to be placed in first-level directories but not the root directory for access, as the root directory cannot be mounted.
- The default NameService has the `/emr` directory, which needs to be mounted.

Submitting MapReduce Tasks

Last updated : 2020-12-15 15:24:37

This operation guide describes: 1. How to perform basic MapReduce job operations in command-line interfaces. 2. How to allow MapReduce jobs to access to the data stored in COS. For more information, please see the [community documentation](#).

- The job submitted is a wordcount job. To count the words in a file, you need to upload the specified file in advance.
- The path of relevant software such as Hadoop is `/usr/local/service/`.
- The relevant logs are stored in `/data/emr`.

1. Preparations for Development

- You need to [create a bucket](#) in COS for this job.
- Confirm that you have activated Tencent Cloud and created an EMR cluster. When creating the EMR cluster, select **Enable COS** on the basic configuration page and then enter your SecretId and SecretKey. You can find your SecretId and SecretKey at [API Key Management](#). If you don't have a key, click **Create Key** to create one.

2. Logging in to an EMR Server

You need to log in to any server in the EMR cluster first before performing the relevant operations. A master node is recommended for this step. EMR is built on CVM instances running on Linux; therefore, using EMR in command line mode requires logging in to an CVM instance.

After creating the EMR cluster, select Elastic MapReduce in the console. In **Cluster Resource > Resource Management**, click **Master Node** to select the resource ID of the master node. Then, you can enter the CVM Console and find the instance of the EMR cluster.

For more information on how to log in to a CVM instance, please see [Logging in to a Linux Instance](#). Here, you can choose to log in with WebShell. Click **Login** on the right of the desired CVM instance to enter the login page. The default username is root, and the password is the one you set when creating the EMR cluster.

Once your credentials have been validated, you can access the EMR command-line interface. All Hadoop operations are under the Hadoop user. The root user is logged in by default when you log in to the EMR server, so you need to switch to the Hadoop user. Run the following command to switch users and go to the Hadoop folder:

```
[root@172 ~]# su hadoop
[hadoop@172 root]$ cd /usr/local/service/hadoop
```

```
[hadoop@172 hadoop]$
```

3. Data Preparations

You need to prepare a text file for counting. There are two ways to do so: **storing data in an HDFS cluster** and **storing data in COS**.

The first step is to upload a local file to the CVM instance of the EMR cluster with the scp or sftp service. Run the following command on the local command line:

```
scp $localfile root@public IP address:$remotefolder
```

Here, \$localfile is the path and the name of your local file; root is the CVM instance username. You can look up the public IP address in the node information in the EMR or CVM Console; and \$remotefolder is the path where you want to store the file in the CVM instance.

After the upload succeeds, you can check whether the file is in the corresponding folder on the command line of the EMR cluster.

```
[hadoop@172 hadoop]$ ls -l
```

Storing data in HDFS

After uploading the data to the CVM instance, you can copy it to the HDFS cluster. The README.txt file in the `/usr/local/service/hadoop` directory is used here as an example. Copy the file to the Hadoop cluster by running the following command:

```
[hadoop@172 hadoop]$ hadoop fs -put README.txt /user/hadoop/
```

After the copy is completed, run the following command to view the copied file:

```
[hadoop@172 hadoop]$ hadoop fs -ls /user/hadoop
Output:
-rw-r--r-- 3 hadoop supergroup 1366 2018-06-28 11:39 /user/hadoop/README.txt
```

If there is no `/user/hadoop` folder in Hadoop, you can create it on your own by running the following command:

```
[hadoop@172 hadoop]$ hadoop fs -mkdir /user/hadoop
```

For more Hadoop commands, please see [Common HDFS Operations](#)

Storing data in COS

There are two ways to store data in COS: **uploading through the COS Console from the local file system** and **uploading through Hadoop command from the EMR cluster**.

- When [uploading through the COS Console from the local file system](#), if the data file is already in COS, you can view it by running the following command:

```
[hadoop@10 hadoop]$ hadoop fs -ls cosn://$bucketname/README.txt
-rw-rw-rw- 1 hadoop hadoop 1366 2017-03-15 19:09 cosn://$bucketname /README.txt
```

Replace \$bucketname with the name and path of your bucket.

- To upload through Hadoop command from the EMR cluster, run the following command:

```
[hadoop@10 hadoop]$ hadoop fs -put README.txt cosn:// $bucketname /
[hadoop@10 hadoop]$ bin/hadoop fs -ls cosn:// $bucketname /README.txt
-rw-rw-rw- 1 hadoop hadoop 1366 2017-03-15 19:09 cosn://$bucketname /README.txt
```

4. Submitting a Job Through MapReduce

The job submitted this time is the wordcount routine that comes with the Hadoop cluster, which has already been compressed into a .jar package and uploaded to the created Hadoop cluster for direct call and use.

Counting a text file in HDFS

Go to the `/usr/local/service/hadoop` directory as described in data preparations, and submit the job by running the following command:

```
[hadoop@10 hadoop]$ bin/yarn jar ./share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.3.jar wordcount
/user/hadoop/README.txt /user/hadoop/output
```

Note :

In the complete command above, `/user/hadoop/README.txt` is the input file to be processed, and `/user/hadoop/output` is the output folder. You should make sure that there is no output folder before submitting the command; otherwise, the submission will fail.

After the execution is completed, view the output file by running the following command:

```
[hadoop@10 hadoop]$ bin/hadoop fs -ls /user/hadoop/output
Found 2 items
-rw-r--r-- 3 hadoop supergroup 0 2017-03-15 19:52 /user/hadoop/output/_SUCCESS
-rw-r--r-- 3 hadoop supergroup 1306 2017-03-15 19:52 /user/hadoop/output/part-r-0000
```

View the statistics in part-r-00000 by running the following command:

```
[hadoop@10 hadoop]$ bin/hadoop fs -cat /user/hadoop/output/part-r-00000
(BIS), 1
(ECCN) 1
(TSU) 1
(see 1
5D002.C.1, 1
740.13) 1
<http://www.wassenaar.org/> 1
.....
```

Counting a text file in COS

Go to the `/usr/local/service/hadoop` directory and submit the job by running the following command:

```
[hadoop@10 hadoop]$ bin/yarn jar ./share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.3.jar wordcount
cosn://$bucketname/README.txt /user/hadoop/output
```

The input file for the command is changed to `cosn:// $bucketname /README.txt`, which indicates to process the file in COS, where `$bucketname` is your bucket name and path. The result is still outputted to the HDFS cluster, but you can also choose to output to COS. The way to view the output is the same as above.

Viewing job logs

```
# View job status
bin/mapred job -status jobid
# View job logs
yarn logs -applicationId id
```

Automatically Adding Task Nodes Without Assigning ApplicationMasters

Last updated : 2022-05-16 12:52:25

Overview

In the automatic scaling scenario, when a scale-in rule is triggered, if an ApplicationMaster (AM) is running on the task node to be terminated, the running AM will also be terminated, which will cause the current job to fail.

No AMs will be assigned to task nodes automatically added through scale-out by default. This ensures that when the nodes are removed, running AMs will not be terminated, so that jobs can run properly.

Feature

No AMs will be assigned to task nodes automatically added through the transformation of YARN source code and the addition of configuration items to the automatic scaling process. Instead, all AMs will be assigned to those that are not automatically added. Automatically added task nodes are responsible for compute tasks only. Therefore, an automatic scale-in action will only terminate the nodes processing compute tasks, while the AMs remain active and will try to take over the compute tasks on other nodes to keep the current job running.

Application Scope

This only applies to task nodes automatically added.

YARN Task Queue Management

Last updated : 2021-10-08 15:16:21

You can log in to the YARNwebUI, which is the web user interface of YARN, through the shortcut entry provided in EMR. For more information, please see the [Software WebUI Entry](#). After logging in, you will see the following:

The screenshot displays the Hadoop YARN web interface. At the top, it shows the Hadoop logo and the title "NEW,NEW_SAVING,SUBMITTED,ACCEPTED,RUNNING Applications". The interface is divided into several sections:

- Cluster Metrics:** A table showing overall cluster statistics.

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved	Active Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes
67	9	7	51	15	30 GB	36.00 GB	10 GB	15	20	5	5	0	0	0	0
- User Metrics for hadoop:** A table showing metrics for the 'hadoop' user.

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Containers Pending	Containers Reserved	Memory Used	Memory Pending	Memory Reserved	VCores Used	VCores Pending	VCores Reserved
66	9	7	50	15	15	5	30 GB	30 GB	10 GB	15	15	5
- Scheduler Metrics:** Information about the Fair Scheduler, including its type and scheduling resource type.

Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation
Fair Scheduler	[MEMORY, CPU]	<memory:16, vCores:1>	<memory:7372, vCores:4>
- Application Queues:** A section showing the status of application queues. It includes a legend for "Used" (green) and "Used (over fair share)" (orange). A bar chart shows the usage of resources for different queues:
 - root: 83.3% used
 - root.default: 0.0% used
 - root.root: 0.0% used
 - root.hadoop: 83.3% used
- 'root.hadoop' Queue Status:** Detailed resource usage for the 'root.hadoop' queue.

Used Resources:	<memory:30720, vCores:15>
Num Active Applications:	7
Num Pending Applications:	9
Min Resources:	<memory:0, vCores:0>
Max Resources:	<memory:36860, vCores:20>
Steady Fair Share:	<memory:12287, vCores:0>
Instantaneous Fair Share:	<memory:36860, vCores:0>
- Table of Application Entries:** A table at the bottom showing details for a specific application.

ID	User	Name	Application Type	Queue	Fair Share	StartTime	FinishTime	State	FinalStatus	Progress	Tracking UI
			SPARK	root.hadoop	2304	Wed Oct 25 10:35:06	N/A	ACCEPTED	UNDEFINED		UNASSIGNED

You can see some monitoring information of the entire cluster here:

- Apps: 67 submitted apps, including 9 pending ones, 7 running ones, and 51 completed ones, where 15 containers are running.
- Memory: 36 GB in total with 30 GB used and 10 GB reserved.
- Virtual cores: 20 cores in total with 15 used ones and 5 reserved ones.
- Nodes: 5 available nodes, 0 decommissioned ones, 0 lost ones, 0 unhealthy ones, and 0 rebooted ones.
- Scheduler: Fair Scheduler, including maximum and minimum memory size and CPU allocation information.

The Application Queues section contains job queuing information of the cluster. Take the root.hadoop queue as an example:

- Used resources: 30,720 MB (30 GB) memory, 15 virtual cores.
- Active applications: 7.
- Pending applications: 9.
- Minimum memory occupied: 0 MB, 0 cores.
- Maximum memory occupied: 36,860 MB, 20 cores.

- Steady fair share.
- Instantaneous fair share.
- The proportion of memory occupied by this queue is 83.3%.

Practices on YARN Label Scheduling

Last updated : 2022-04-21 11:00:37

Feature Overview

Spark on YARN uses YARN as the resource scheduler. In Hadoop version above 2.7.2, YARN is equipped with Label Scheduler in addition to Capacity Scheduler.

Capacity Scheduler is a multi-tenant resource scheduler. Its core principle is that the available resources in a Hadoop cluster are shared by multiple organizations, which collectively provide computing power to the cluster based on their own computing needs. Capacity Scheduler has various features such as hierarchical queuing, capacity guarantee, security, elastic resource, multi-tenancy, resource-based scheduling, and mapping. All resources of the cluster are allocated to multiple queues. All applications submitted to a queue can use the resources allocated to the queue. Idle resources of other organizations can be non-preemptively allocated to applications running in queues where resources are not fully utilized, ensuring that the resource requirements of different applications can be met flexibly while applications can get the minimum resources required.

Capacity Scheduler roughly allocates cluster resources to different queues and cannot specify the running locations of applications in queues. Based on Capacity Scheduler, Label Scheduler adds different node labels to each node in the cluster for more fine-grained resource division, so that running locations of applications can be specified. Node labels have the following characteristics:

1. One node has only one node label (i.e., belonging to only one node partition). A cluster can be divided into multiple disjoint subclusters based on node partitioning.
2. There are two types of node partitions based on the match policy: exclusive and non-exclusive. An exclusive node partition will allocate containers to nodes that exactly match the node partition, while a non-exclusive node partition shares idle resources to containers requesting the default partition.
3. You can set accessible node labels for each queue to specify the node partitions for running applications.
4. You can set the resource ratio of different queues in each node partition.
5. If the required node label is specified in a resource request, the request will only be allocated to nodes with the same label. If the node label is not specified (you can use a configuration item to rename the default label of a queue), the request will only be allocated to nodes in the default partition.

Configuration Description

1. Enable Capacity Scheduler in ResourceManager

Label Scheduler cannot be used alone; instead, it must be used together with Capacity Scheduler, which is the default scheduler of YARN. If you are using another scheduler, please enable Capacity Scheduler first in

```
${HADOOP_HOME}/etc/hadoop/yarn-site.xml :
```

```
<property>
<name>yarn.resourcemanager.scheduler.class</name>
<value>org.apache.hadoop.yarn.server.resourcemanager.scheduler.capacity.CapacityS
cheduler</value>
</property>
```

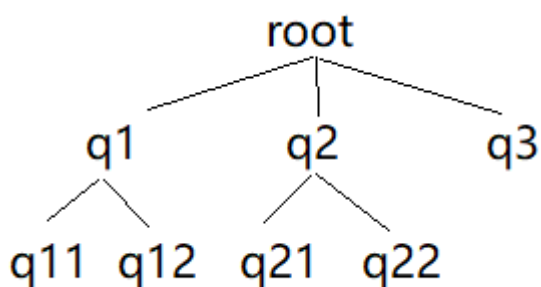
2. Configure Capacity Scheduler parameters

In `${HADOOP_HOME}/etc/hadoop/capacity-scheduler.xml` , set

`yarn.scheduler.capacity.root` as the predefined root queue of Capacity Scheduler. All other queues are subqueues of the root queue. All queues are organized in a tree structure. `yarn.scheduler.capacity.`

`<queue-path>.queues` is used to set the subqueues under the `queue-path` path, with subqueues separated by commas.

Example:



The structure shown above is configured as follows:

```
<property>
<name>yarn.scheduler.capacity.root.queues</name>
<value>q1,q2,q3</value>
</property>
<property>
<name>yarn.scheduler.capacity.root.q1.queues</name>
<value>q11,q12</value>
</property>
<property>
<name>yarn.scheduler.capacity.root.q2.queues</name>
<value>q21,q22</value>
</property>
```

For other Capacity Scheduler configuration items, please see the relevant documentation.

3. Enable node label in ResourceManager

Set it in `conf/yarn-site.xml` .

```
<property>
<name>yarn.node-labels.fs-store.root-dir</name>
<value>hdfs://namenode:port/path-to-store/node-labels/</value>
</property>
<property>
<name>yarn.node-labels.enabled</name>
<value>>true</value>
</property>
<property>
<name>yarn.node-labels.configuration-type</name>
<value>centralized or delegated-centralized or distributed</value>
</property>
```

Note :

1. Make sure that `yarn.node-labels.fs-store.root-dir` has been created and ResourceManager has the permission to access it.
2. You can store node labels in the local file system of ResourceManager at a path such as `file://home/yarn/node-label` ; however, in order to ensure high availability of the >cluster and avoid loss of label information due to ResourceManager failures, you are recommended to store the label information in HDFS.
3. In Hadoop 2.8.2, you need to configure the `yarn.node-labels.configuration-type` configuration item.

4. Configure node label

Set it in `etc/hadoop/capacity-scheduler.xml` .

Configuration Item	Description
<code>yarn.scheduler.capacity.<queue-path>.capacity</code>	It specifies the percentage of nodes in the default partition that a queue can access. The total capacity of the default partition for all subqueues under the parent queue must be 100.

Configuration Item	Description
<code>yarn.scheduler.capacity.<queue-path>.accessible-node-labels</code>	It specifies the list of labels that a queue can access, and labels should be separated by commas. For example, "HBASE,STORM" means that the queue can access the labels "HBASE" and "STORM". All queues can access nodes with no label. If this field is not set, it will inherit the value of its parent field. If you want a queue to access only nodes with no label, leave this field empty.
<code>yarn.scheduler.capacity.<queue-path>.accessible-node-labels.<label>.capacity</code>	It specifies the percentage of nodes in the <code><label></code> partition that a queue can access. Note that the total capacity of the <code><label></code> partition for all subqueues under the parent queue must be 100. The default value of this configuration item is 0.
<code>yarn.scheduler.capacity.<queue-path>.accessible-node-labels.<label>.maximum-capacity</code>	Similar to the <code>yarn.scheduler.capacity.<queue-path>.maximum-capacity</code> configuration item in Capacity Scheduler, it specifies the maximum capacity of <code><queue-path></code> in the <code><label></code> partition. The default value of this configuration item is 100.
<code>yarn.scheduler.capacity.<queue-path>.default-node-label-expression</code>	If a resource request does not have a node label specified, the application will be submitted to the corresponding partition specified by this configuration item. By default, this value is empty, i.e., applications will be allocated to containers in nodes with no label.

Use Cases

Preparations

1. Prepare a cluster

Confirm that you have activated Tencent Cloud and created an EMR cluster.

2. Check configuration of the YARN component

On the "Component Management" page, select the YARN component to enter its component management page.

On the role management tab, confirm the IP of the node where the ResourceManager service resides. Then, switch to the configuration management tab to modify relevant parameters in `yarn-site.xml`, save the changes, and restart all YARN components.

3. Enter the YARN component management page.

The screenshot shows the 'Components' management page for an Elastic MapReduce cluster. The 'YARN' component is highlighted with a red box. The table below lists the components and their details:

Component name	Status	Version	Native WebUI Access Address	Access Info	Operation
ZOOKEEPER	Running	3.4.9		QuorumPeerMain IPC: [redacted]	Configuration More
HDFS	Running	2.7.3	[redacted]	NameNode IPC: 172.16.16.43:4007 zkfc IPC: JournalNode IPC: DataNode IPC: [redacted]	Configuration More
YARN	Running	2.7.3	[redacted]	ResourceManager IPC: 172.16.16.43:5004 NodeManager IPC: [redacted] JobHistoryServer [redacted]	Configuration More
				HbaseThrift IPC: [redacted] HMaster [redacted]	Configuration

4. Confirm the IP address of ResourceManager.

The screenshot shows the 'YARN' configuration management page. The 'ResourceManager' service is highlighted with a red box. The table below lists the services and their details:

Service Type	Service Status	Configuration Gro...	Maintenance Status	Node IP	Last Restarted
JobHistoryServer	Running	yarn-defaultGroup	Normal mode	172.16.16.43	
NodeManager	Running	yarn-defaultGroup	Normal mode	172.16.16.98	
NodeManager	Running	yarn-defaultGroup	Normal mode	172.16.16.45	
ResourceManager	Running	yarn-defaultGroup	Normal mode	172.16.16.43	

5. Modify the `yarn.resourcemanager.scheduler.class` parameter in `yarn-site.xml` of the node where ResourceManager resides.

Configuration Management

Level: Sever level 172.16.16.43(Master) Reselect

Note: if the values contain special characters such as <> & when you modify configuration items on the console, the console will not escape these characters. To ensure that the special characters are processed properly, please follow the XML standard to set configurations.

Configuration File	File	YARN : yarn-site.xml	Restart Component	Modify Configuration
capacity-scheduler.xml	yarn.resourcemanager.resource-tracker.address	172.16.16.43:3002		
fair-scheduler.xml	yarn.resourcemanager.scheduler.address	172.16.16.43:5001		
	yarn.resourcemanager.scheduler.class	org.apache.hadoop.yarn.server.resourcemanager.scheduler.fair.FairScheduler		
	yarn.resourcemanager.store.class	org.apache.hadoop.yarn.server.resourcemanager.recovery.FileSystemRMStateStore		
mapred-env.sh	yarn.resourcemanager.webapp.address	172.16.16.43:5004		
mapred-site.xml	yarn.resourcemanager.webapp.https.address	172.16.16.43:5005		
	yarn.resourcemanager.zk-address			
yarn-env.sh	yarn.resourcemanager.zk-timeout-ms	60000		
yarn-site.xml	yarn.resourcemanager.zk.state-store.address			

Configuring the mappings and ratios of node labels and queues in Capacity-Scheduler.xml

1. Create an HDFS directory for storing node labels.

```
[root@172 ~]# cd /usr/local/service/hadoop/
[root@172 hadoop]# su hadoop
[hadoop@172 hadoop]$ hadoop fs -mkdir -p /yarn/node-labels
```

2. Get the IP and port of NN in `core-site.xml` .

```
<property>
  <name>fs.defaultFS</name>
  <value>hdfs://172.16.32.47:4007</value>
</property>
```

3. In `yarn-site.xml` of the master node, create a configuration item. Then, restart ResourceManager.

<code>yarn.node-labels.fs-store.root-dir</code>	<code>hdfs://172.16.32.47:4007/syst</code>
<code>yarn.node-labels.enabled</code>	<code>true</code>
<code>yarn.node-labels.configuration-type</code>	<code>centralized</code>

4. Run the `yarn rmdadmin -addToClusterNodeLabels` command to add labels.

```
[hadoop@172 hadoop]$ yarn cluster --list-node-labels
Node Labels:
[hadoop@172 hadoop]$ yarn rmdadmin -addToClusterNodeLabels "normal,cpu"
[hadoop@172 hadoop]$ yarn cluster --list-node-labels
Node Labels: <normal:exclusivity=true>,<cpu:exclusivity=true>
```

Enter the WebUI of the YARN component. You can view all labels of the cluster in the "NodeLabels" panel.

Cluster

- About
- Nodes
- Node Labels**
- Applications
 - NEW
 - NEW_SAVING
 - SUBMITTED
 - ACCEPTED
 - RUNNING
 - FINISHED
 - FAILED
 - KILLED
- Scheduler

Tools

Label Name	Label Type	Num Of Active NMs	Total Resource
<DEFAULT_PARTITION>	Exclusive Partition	2	<memory:14744, vCores:8>
cpu	Exclusive Partition	0	<memory:0, vCores:0>
normal	Exclusive Partition	0	<memory:0, vCores:0>

Showing 1 to 3 of 3 entries

5. Run the `yarn rmdadmin -replaceLabelsOnNode` command to add labels to nodes.

```
[hadoop@172 hadoop]$ yarn rmdadmin -replaceLabelsOnNode "172.16.32.43=normal"
[hadoop@172 hadoop]$ yarn rmdadmin -replaceLabelsOnNode "172.16.32.25=cpu"
```

As can be seen in the "NodeLabels" panel, the number of nodes in the "normal" and "cpu" partitions has become 1 from 0.

As can be seen in the "Scheduler" panel, the labels of the two nodes in the testing system have changed.

6. Edit configuration items in `Capacity-Scheduler.xml` to configure the cluster queues, resource ratio of queues, and accessible labels of queues as shown in the following sample:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration><property>
<name>yarn.scheduler.capacity.maximum-am-resource-percent</name>
<value>0.8</value>
</property>
<property>
<name>yarn.scheduler.capacity.maximum-applications</name>
```

```
<value>1000</value>
</property>
<property>
<name>yarn.scheduler.capacity.root.queues</name>
<value>default,dev,product</value>
</property>
<property>
<name>yarn.scheduler.capacity.root.default.capacity</name>
<value>20</value>
</property>
<property>
<name>yarn.scheduler.capacity.root.dev.capacity</name>
<value>40</value>
</property>
<property>
<name>yarn.scheduler.capacity.root.product.capacity</name>
<value>40</value>
</property>
<property>
<name>yarn.scheduler.capacity.root.accessible-node-labels.cpu.capacity</name>
<value>100</value>
</property>
<property>
<name>yarn.scheduler.capacity.root.accessible-node-labels.normal.capacity</name>
<value>100</value>
</property>
<property>
<name>yarn.scheduler.capacity.root.accessible-node-labels</name>
<value>*</value>
</property>
<property>
<name>yarn.scheduler.capacity.root.dev.accessible-node-labels.normal.capacity</name>
<value>100</value>
</property>
<property>
<name>yarn.scheduler.capacity.root.product.accessible-node-labels.cpu.capacity</name>
<value>100</value>
</property>
<property>
<name>yarn.scheduler.capacity.root.dev.accessible-node-labels</name>
<value>normal</value>
</property>
<property>
<name>yarn.scheduler.capacity.root.dev.default-node-label-expression</name>
```

```
<value>normal</value>
</property>
<property>
<name>yarn.scheduler.capacity.root.product.accessible-node-labels</name>
<value>cpu</value>
</property>
<property>
<name>yarn.scheduler.capacity.root.product.default-node-label-expression</name>
<value>cpu</value>
</property>
<property>
<name>yarn.scheduler.capacity.normal.sharable-partitions</name>
<value>cpu</value>
</property>
<property>
<name>yarn.scheduler.capacity.normal.require-other-partition-resource</name>
<value>>true</value>
</property>
<property>
<name>yarn.scheduler.capacity.cpu.sharable-partitions</name>
<value></value>
</property>
<property>
<name>yarn.scheduler.capacity.cpu.require-other-partition-resource</name>
<value>>true</value>
</property>
</configuration>
```

The "Scheduler" panel displays the 3 partitions of the testing cluster, resource allocation of partitions, and information of included queues. In the "Application Queues" panel, there are 3 partitions: default, normal, and cpu, among which the "default" partition is the default one, the "normal" partition consists of nodes with the "normal" label, and the "cpu" partition consists of nodes with the "cpu" label. In the testing environment, there are two nodes which are labeled as

"normal" and "cpu", respectively. You can click "+" on the left of a partition to expand all queues in it.

NEW_SAVING
SUBMITTED
ACCEPTED
RUNNING
FINISHED
FAILED
KILLED

Scheduler

Tools

Active Nodes	Decommissioning Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted
2	0	0	0	0	0

Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation	Maxim
Capacity Scheduler	[MEMORY]	<memory:16, vCores:1>	<memory:7372, vCores:4>	0

Dump scheduler logs 1 min

Application Queues

Legend: Capacity Used Used (over capacity) Max Capacity Users Requesting Resources

- Partition: <DEFAULT_PARTITION> <memory:0, vCores:0>
 - Queue: root
 - Queue: default
 - Queue: dev
 - Queue: product
- Partition: cpu <memory:7372, vCores:4>
 - Queue: root
 - Queue: default
 - Queue: product
- Partition: normal <memory:7372, vCores:4>
 - Queue: root
 - Queue: default
 - Queue: dev
 - Queue: product

Testing Label Scheduling

- **Test 1. Submit a task to the "product" queue**

```
[hadoop@172 hadoop]$ cd /usr/local/service/hadoop
[hadoop@172 hadoop]$ yarn jar ./share/hadoop/mapreduce/hadoop-mapreduce-client-
jobclient-2.8.4-tests.jar sleep -Dmapreduce.job.queueName=product -m 32 -mt 100
0
```

After the task is submitted, the resource usage of queues in each partition is as shown below:

Application Queues

Legend: Capacity Used Used (over capacity) Max Capacity Users Requesting Resources

- Partition: <DEFAULT_PARTITION> <memory:0, vCores:0>
 - Queue: root
- Partition: cpu <memory:7372, vCores:4>
 - Queue: root
 - Queue: default
 - Queue: product

Used Capacity:	90.3%
Configured Capacity:	100.0%
Configured Max Capacity:	100.0%
Absolute Used Capacity:	90.3%
Absolute Configured Capacity:	100.0%
Absolute Configured Max Capacity:	100.0%
Used Resources:	<memory:6656, vCores:6>
Configured Max Application Master Limit:	80.0
Max Application Master Resources:	<memory:5904, vCores:1>
Used Application Master Resources:	<memory:1536, vCores:1>
Max Application Master Resources Per User:	<memory:5904, vCores:1>

Conclusion: there is a mapping between the "product" queue and the "cpu" label, the default label is "cpu", and the task submitted to the "product" queue runs on the node with the "cpu" label.

• **Test 2. Submit a task to the "dev" queue**

```
[hadoop@172 hadoop]$ cd /usr/local/service/hadoop
[hadoop@172 hadoop]$ yarn jar ./share/hadoop/mapreduce/hadoop-mapreduce-client-jobclient-2.8.4-tests.jar sleep -Dmapreduce.job.queueName=dev -m 32 -mt 1000
```

After the task is submitted, the resource usage of queues in each partition is as shown below:

Application Queues

Legend: Capacity Used Used (over capacity) Max Capacity Users Requesting Resources

- Partition: <DEFAULT_PARTITION> <memory:0, vCores:0>
 - Queue: root
- Partition: cpu <memory:7372, vCores:4>
 - Queue: root
- Partition: normal <memory:7372, vCores:4>
 - Queue: root
 - Queue: default
 - Queue: dev

Used Capacity:	41.7%
Configured Capacity:	50.0%
Configured Max Capacity:	100.0%
Absolute Used Capacity:	20.8%
Absolute Configured Capacity:	50.0%
Absolute Configured Max Capacity:	100.0%
Used Resources:	<memory:1536, vCores:1>
Configured Max Application Master Limit:	80.0
Max Application Master Resources:	<memory:2960, vCores:1>
Used Application Master Resources:	<memory:1536, vCores:1>
Max Application Master Resources Per User:	<memory:2960, vCores:1>

Conclusion: there is a mapping between the "dev" queue and the "normal" label, the default label is "normal", and the task submitted to the "dev" queue runs on the node with the "normal" label.

Hadoop Best Practices

Last updated : 2021-07-13 14:41:40

In Hadoop, distributed file system HDFS, resource scheduling framework YARN, and iterative computing framework MR. Tencent Cloud's Hadoop version that have integrated with COS, allowing you to access to COS by running the `hadoop fs` command line so as to separate compute and storage apart. Below are some best practices:

- HDFS

For both high-availability (HA) cluster and non-HA cluster, **do not format the namenode**; otherwise, your data will be lost permanently. Tencent Cloud shall not be responsible under any circumstance for any loss of data caused by formatting the namenode.

- YARN

The fair scheduler is enabled by default, and you can change the scheduler based on your actual needs.

Using API to Analyze Data in HDFS and COS

Last updated : 2021-07-08 10:43:44

The WordCount application is a great example that gives you a hands-on experience in developing your first Hadoop MapReduce application. In this tutorial, you will learn how to implement WordCount example code in MapReduce to count the number of occurrences of a given word in the input file stored in HDFS or in COS. The program is the same as the one shown in the Hadoop community.

1. Development Preparations

- This task requires access to COS, so you need to [create a bucket](#) in COS first.
- Create an EMR cluster. When creating the EMR cluster, you need to select a cluster type that includes HDFS and enable access to COS on the basic configuration page.

2. Logging in to an EMR Server

Log in to any node (preferably a master node) in the EMR cluster before performing relevant operations. EMR is built on CVM instances running on Linux; therefore, using EMR in command line mode requires logging in to an CVM instance.

After creating the EMR cluster, select **Elastic MapReduce** in the console, click the ID/name of the cluster you just created in the cluster list, click **Cluster Resource > Resource Management > Master**, and click the resource ID of an active master node to enter the CVM console and find the CVM instance of the EMR cluster.

For information about how to log in to a CVM instance, see [Logging in to Linux Instance Using Standard Login Method](#). Here, you can use WebShell to log in. Click **Login** on the right of the desired CVM instance to go to the login page. The default username is `root`, and the password is the one you set when creating the EMR cluster.

Once your credentials are validated, you can enter the EMR command line interface. All Hadoop operations should be performed under the `Hadoop` user. The `root` user is logged in by default when you log in to the EMR node, so you need to switch to the `Hadoop` user. Run the following command to switch users and go to the `Hadoop` folder:

```
[root@172 ~]# su hadoop
[hadoop@172 root]$ cd /usr/local/service/hadoop
[hadoop@172 hadoop]$
```


3. Data Preparations

Prepare an input text file. You can either **store data in an HDFS cluster** or **store data in COS**.

First, create a .txt file named `test.txt` locally and add the following sentences to the file:

```
Hello World.  
this is a message.  
this is another message.  
Hello world, how are you?
```

Use scp or sftp service to upload a local file to the CVM instance in your EMR cluster. Run the following command in your local shell:

```
scp $localfile root@public IP address:$remotefolder
```

Here, \$localfile is the path and the name of your local file; root is the CVM instance username. You can look up the public IP address in the node information in the EMR or CVM Console; and \$remotefolder is the path where you want to store the file in the CVM instance.

After the upload is completed, you can check whether the file is in the corresponding folder on the command line of the EMR cluster. The file is uploaded to the `/usr/local/service/hadoop` path in the EMR cluster in this example.

```
[hadoop@172 hadoop]$ ls -l
```

Storing Data in HDFS

After uploading the data to the CVM instance, you can copy the data file to the Hadoop cluster by running the following command:

```
[hadoop@172 hadoop]$ hadoop fs -put /usr/local/service/hadoop/test.txt /user/hadoop/
```

After the copy is completed, run the following command to view the copied file:

```
[hadoop@172 hadoop]$ hadoop fs -ls /user/hadoop  
Output:  
-rw-r--r-- 3 hadoop supergroup 85 2018-07-06 11:18 /user/hadoop/test.txt
```

If there is no `/user/hadoop` folder in Hadoop, you can create it on your own by running the following command:

```
[hadoop@172 hadoop]$ hadoop fs -mkdir /user/hadoop
```

For more Hadoop commands, see [HDFS Common Operations](#).

Storing Data in COS

There are two ways to store data in COS: **uploading via the COS console from the local storage** and **uploading via a Hadoop command**.

- When [uploading via the COS console from the local storage](#), you can view the uploaded data file by running the following command:

```
[hadoop@10 hadoop]$ hadoop fs -ls cosn://$bucketname/ test.txt
```

- ```
rw-rw-rw- 1 hadoop hadoop 1366 2017-03-15 19:09 cosn://$bucketname/test.txt
```

Replace `$bucketname` with the name and path of your bucket.

- To upload via Hadoop command, run the following command:

```
[hadoop@10 hadoop]$ hadoop fs -put test.txt cosn://$bucketname /
[hadoop@10 hadoop]$ hadoop fs -ls cosn:// $bucketname / test.txt
```

- ```
rw-rw-rw- 1 hadoop hadoop 1366 2017-03-15 19:09 cosn://$bucketname / test.txt
```

4. Creating a Project with Maven

Maven is recommended for project management. It can help you manage project dependencies with ease.

Specifically, it can get .jar packages through the configuration of the `pom.xml` file, eliminating the need to add them manually.

Download and install Maven first and then configure its environment variables. If you are using the IDE, please set the Maven-related configuration items in the IDE.

Creating a Maven Project

Enter the directory of the Maven project, such as `D://mavenWorkplace`, and create the project using the following commands:

```
mvn archetype:generate -DgroupId=$yourgroupID -DartifactId=$yourartifactID  
-DarchetypeArtifactId=maven-archetype-quickstart
```

Here, `$yourgroupId` is your package name, `$yourartifactID` is your project name, and `maven-archetype-quickstart` indicates to create a Maven Java project. Some files need to be downloaded during the project creation, so please keep the network connected.

After successfully creating the project, you will see a folder named `$yourartifactID` in the `D://mavenWorkplace` directory. Files in the folder have the following structure:

```
simple
  ---pom.xml           Core configuration, under the project root directory
  ---src
    ---main
      ---java          Java source code directory
      ---resources     Java configuration file directory
    ---test
      ---java          Test source code directory
      ---resources     Test configuration directory
```

We need to pay attention to the `pom.xml` file and the Java folder under the main directory, as the former is primarily used for dependency and packaging configuration, and the latter for source code storage.

First, add the Maven dependencies to `pom.xml`:

```
<dependencies>
<dependency>
<groupId>org.apache.hadoop</groupId>
<artifactId>hadoop-common</artifactId>
<version>2.7.3</version>
</dependency>
<dependency>
<groupId>org.apache.hadoop</groupId>
<artifactId>hadoop-mapreduce-client-core</artifactId>
<version>2.7.3</version>
</dependency>
</dependencies>
```

Then, add the packaging and compiling plugins to `pom.xml`:

```
<build>
<plugins>
<plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-compiler-plugin</artifactId>
<configuration>
<source>1.8</source>
<target>1.8</target>
```

```
<encoding>utf-8</encoding>
</configuration>
</plugin>
<plugin>
<artifactId>maven-assembly-plugin</artifactId>
<configuration>
<descriptorRefs>
<descriptorRef>jar-with-dependencies</descriptorRef>
</descriptorRefs>
</configuration>
<executions>
<execution>
<id>make-assembly</id>
<phase>package</phase>
<goals>
<goal>single</goal>
</goals>
</execution>
</executions>
</plugin>
</plugins>
</build>
```

Right-click in `src>main>java` and create a Java Class. Enter the Class name (e.g., `WordCount` here) and add the sample code to the Class:

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;
import java.io.IOException;
import java.util.StringTokenizer;
/**
 * Created by tencent on 2018/7/6.
 */
public class WordCount {
    public static class TokenizerMapper
    extends Mapper<Object, Text, Text, IntWritable>
    {
        private static final IntWritable one = new IntWritable(1);
```

```
private Text word = new Text();
public void map(Object key, Text value, Mapper<Object, Text, Text, IntWritable>.Context context)
throws IOException, InterruptedException
{
    StringTokenizer itr = new StringTokenizer(value.toString());
    while (itr.hasMoreTokens())
    {
        this.word.set(itr.nextToken());
        context.write(this.word, one);
    }
}

public static class IntSumReducer
extends Reducer<Text, IntWritable, Text, IntWritable>
{
    private IntWritable result = new IntWritable();
    public void reduce(Text key, Iterable<IntWritable> values, Reducer<Text, IntWritable, Text, IntWritable>.Context context)
    throws IOException, InterruptedException
    {
        int sum = 0;
        for (IntWritable val : values) {
            sum += val.get();
        }
        this.result.set(sum);
        context.write(key, this.result);
    }
}

public static void main(String[] args)
throws Exception
{
    Configuration conf = new Configuration();
    String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
    if (otherArgs.length < 2)
    {
        System.err.println("Usage: wordcount <in> [<in>...] <out>");
        System.exit(2);
    }
    Job job = Job.getInstance(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    for (int i = 0; i < otherArgs.length - 1; i++) {
```

```
FileInputFormat.addInputPath(job, new Path(otherArgs[i]));
}
FileOutputFormat.setOutputPath(job, new Path(otherArgs[(otherArgs.length - 1)]));
System.exit(job.waitForCompletion(true) ? 0 : 1);
}
}
```

As you can see, there is a Map function and a Reduce function.

If your Maven is configured correctly and its dependencies are successfully imported, the project can be compiled directly. Enter the project directory in the local shell, and run the following command to package the entire project:

```
mvn package
```

Some files may need to be downloaded during the running process. "Build success" indicates that package is successfully created. You can see the generated .jar package in the target folder under the project directory.

Upload the packaged project file to the CVM instance of the EMR cluster using the scp or sftp service. Run the following command in the local shell:

```
scp $jarpackage root@public IP address: /usr/local/service/hadoop
```

Here, `$jarpackage` is the path plus name of your local .jar package; `root` is the CVM instance username; and the public IP address can be viewed in the node information in the EMR console or the CVM console. The file is uploaded to the `/usr/local/service/hadoop` folder of the EMR cluster.

Counting a Text File in HDFS

Go to the `/usr/local/service/hadoop` directory as described in data preparations, and submit the task by running the following command:

```
[hadoop@10 hadoop]$ bin/hadoop jar
/usr/local/service/hadoop/WordCount-1.0-SNAPSHOT-jar-with-dependencies.jar
WordCount /user/hadoop/test.txt /user/hadoop/WordCount_output
```

Note :

Above is a complete command, where `/user/hadoop/ test.txt` is the input file and `/user/hadoop/ WordCount_output` is the output folder. You should not create the `WordCount_output` folder before the command is submitted; otherwise, the submission will fail.

After the execution is completed, view the output file by running the following command:

```
[hadoop@172 hadoop]$ hadoop fs -ls /user/hadoop/WordCount_output
Found 2 items
-rw-r--r-- 3 hadoop supergroup 0 2018-07-06 11:35 /user/hadoop/MEWordCount_output/_SUCCESS
-rw-r--r-- 3 hadoop supergroup 82 2018-07-06 11:35 /user/hadoop/MEWordCount_output/part-r-00000
```

View the statistics in part-r-00000 by running the following command:

```
[hadoop@172 hadoop]$ hadoop fs -cat /user/hadoop/MEWordCount_output/part-r-00000
Hello 2
World. 1
a 1
another 1
are 1
how 1
is 2
message. 2
this 2
world, 1
you? 1.....
```

Counting a Text File in COS

Go to the `/usr/local/service/hadoop` directory and submit the task by running the following command:

```
[hadoop@10 hadoop]$ hadoop jar
/usr/local/service/hadoop/WordCount-1.0-SNAPSHOT-jar-with-dependencies.jar
WordCount cosn://$bucketname/test.txt cosn://$bucketname /WordCount_output
```

The input file for the command is changed to `cosn:// $bucketname/ test.txt`, where `$bucketname` is your bucket name and path. The result will go to COS as well. Run the following command to view the output file:

```
[hadoop@10 hadoop]$ hadoop fs -ls cosn:// $bucketname /WordCount_output
Found 2 items
-rw-rw-rw- 1 hadoop Hadoop 0 2018-07-06 10:34 cosn://$bucketname /WordCount_output/_SUCCESS
-rw-rw-rw- 1 hadoop Hadoop 1306 2018-07-06 10:34 cosn://$bucketname /WordCount_output/part-r-00000
```

View the final output result:

```
[hadoop@10 hadoop]$ hadoop fs -cat cosn:// $bucketname /WordCount_output1/part-r-00000
```

```
Hello 2  
World. 1  
a 1  
another 1  
are 1  
how 1  
is 2  
message. 2  
this 2  
world, 1  
you? 1
```


Dumping YARN Job Logs to COS

Last updated : 2021-07-01 15:34:12

By default, Hadoop stores YARN job logs in HDFS. Tencent Cloud EMR also provides the ability to store YARN job logs in external storage (COS).

Prerequisites

The EMR cluster needs to support COS. For more information, please see [Analyzing Data in HDFS/COS with API](#).

Directions

1. Modify the configuration in `yarn-site.xml` and deliver the configuration to all nodes.

```
yarn.nodemanager.remote-app-log-dir=cosn://[bucket_name]/[logs_dirs]
```

2. Add a new configuration item to `core-site.xml` and deliver it to all nodes.

```
fs.AbstractFileSystem.cosn.impl=org.apache.hadoop.fs.cosnative.COS
```

3. Restart all the `nodemanager/datanode` services in the cluster.
4. Run the `hive/spark` job to view the job logs stored in COS.

```
hdfs dfs -ls cosn://[bucket_name]/[logs_dirs]
```

Spark Development Guide

Spark Environment Info

Last updated : 2022-11-25 16:06:47

EMR supports Spark 3.x and 2.x. The software environment information is as follows:

- By default, Spark is installed on a master node.
- After logging in, run the `su hadoop` command to switch to the `hadoop` user.
- The Spark software path is `/usr/local/service/spark` .
- The relevant logs are stored in `/data/emr` .

The above is mainly about how to access COS by using Spark. For more information, see the [community documentation](#).

Using Spark to Analyze Data in COS

Last updated : 2021-07-08 10:43:44

Apache Spark is an open-source project for fast, general-purpose, large-scale data processing. It is similar to Hadoop's MapReduce but faster and more efficient for batch processing. It utilizes in-memory caching and optimized execution for fast performance, and it supports reading/writing Hadoop data in any format. Now Spark has become a unified big data processing platform with a lightning-fast analysis engine for real-time streaming processing, machine learning, and interactive queries.

Spark is an in-memory parallel computing framework for big data processing. Its in-memory computing feature improves the real-time performance of data processing in a big data environment, while ensuring high fault tolerance and scalability. Spark can be deployed on a large number of inexpensive hardware devices to create clusters.

The task submitted in this tutorial is a wordcount task, i.e., counting the number of words. You need to upload the file for counting to the cluster in advance.

1. Development Preparations

- This task requires access to COS, so you need to [create a bucket](#) in COS first.
- Create an EMR cluster. When creating the EMR cluster, you need to select the Spark component on the software configuration page and enable access to COS on the basic configuration page.

2. Creating a Project with Maven

Here, the demo that comes with the system is not used; instead, you need to create a project and compile, compress, and upload it to the EMR cluster on your own for execution. Maven is recommended for project management, as it can help you manage project dependencies with ease. Specifically, it can get .jar packages through the configuration of the `pom.xml` file, eliminating the need to add them manually.

Download and install Maven, then configure its environment variables. If you are using the IDE, please set the Maven-related configuration in the IDE.

Creating a Maven Project

In the local shell environment, enter the directory where you want to create the Maven project, such as

```
D://mavenWorkplace , and enter the following command to create it:
```

```
mvn archetype:generate -DgroupId=$yourgroupID -DartifactId=$yourartifactID -DarchetypeArtifactId=maven-archetype-quickstart
```

Here, `$yourgroupId` is your package name, `$yourartifactId` is your project name, and `maven-archetype-quickstart` indicates to create a Maven Java project. Some files need to be downloaded during the project creation, so please keep the network connected.

After successfully creating the project, you will see a folder named `$yourartifactId` in the `D://mavenWorkplace` directory. Files in the folder have the following structure:

```
simple
---pom.xml      Core configuration, under the project root directory
---src
---main
---java        Java source code directory
---resources   Java configuration file directory
---test
---java        Test source code directory
---resources   Test configuration directory
```

Among the files above, pay extra attention to the `pom.xml` file and the Java folder under the main directory. The `pom.xml` file is primarily used to create dependencies and package configurations; the Java folder is used to store your source codes.

First, add the Maven dependencies to `pom.xml`:

```
<dependencies>
<dependency>
<groupId>org.apache.spark</groupId>
<artifactId>spark-core_2.11</artifactId>
<version>2.0.2</version>
</dependency>
</dependencies>
```

Then, add the packaging and compiling plugins to `pom.xml`:

```
<build>
<plugins>
<plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-compiler-plugin</artifactId>
<configuration>
<source>1.8</source>
<target>1.8</target>
<encoding>utf-8</encoding>
</configuration>
</plugin>
```

```
<plugin>
<artifactId>maven-assembly-plugin</artifactId>
<configuration>
<descriptorRefs>
<descriptorRef>jar-with-dependencies</descriptorRef>
</descriptorRefs>
</configuration>
<executions>
<execution>
<id>make-assembly</id>
<phase>package</phase>
<goals>
<goal>single</goal>
</goals>
</execution>
</executions>
</plugin>
</plugins>
</build>
```

Right-click in `src>main>Java` and create a Java Class. Enter the Class name (e.g., `WordCountOnCos` here) and add the sample code to the Class:

```
import java.util.Arrays;
import org.apache.spark.SparkConf;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.api.java.JavaSparkContext;
import scala.Tuple2;
/**
 * Created by tencent on 2018/6/28.
 */
public class WordCountOnCos {
public static void main(String[] args){
SparkConf sc = new SparkConf().setAppName("spark on cos");
JavaSparkContext context = new JavaSparkContext(sc);
JavaRDD<String> lines = context.textFile(args[0]);
lines.flatMap(x -> Arrays.asList(x.split(" ")).iterator())
.mapToPair(x -> new Tuple2<String, Integer>(x, 1))
.reduceByKey((x, y) -> x+y)
.saveAsTextFile(args[1]);
}
}
```

If your Maven is configured correctly and its dependencies are successfully imported, the project will be compiled directly. Enter the project directory in the local shell, and run the following command to package the entire project:

```
mvn package
```

Some files may need to be downloaded during the running process. "Build success" indicates that package is successfully created. You can see the generated .jar package in the target folder under the project directory.

Data Preparations

First, you need to upload the compressed .jar package to the EMR cluster using the scp or sftp tool by running the following command in local command line mode:

```
scp $localfile root@public IP address:$remotefolder
```

Here, `$localfile` is the path plus name of your local file; `root` is the CVM instance username. You can look up the public IP address in the node information in the EMR console or the CVM console. `$remotefolder` is the path where you want to store the file in the CVM instance. After the upload is completed, you can check whether the file is in the corresponding folder on the EMR command line.

You need to upload the to-be-processed file to COS in advance. If the file is in your local storage, you can upload it directly via the [COS console](#); if it is in the EMR cluster, you can upload it by running the following Hadoop command:

```
[hadoop@10 hadoop]$ hadoop fs -put $testfile cosn://$bucketname/
```

Here, `$testfile` is the full path plus name of the file for counting, and `$bucketname` is your bucket name. After the upload is completed, you can check whether the file is present in COS in the COS console.

Running the Demo

First, log in to any node (preferably a master one) in the EMR cluster. For information about how to log in to EMR, see [Logging in to Linux Instance Using Standard Login Method](#). Here, you can use WebShell to log in. Click **Login** on the right of the desired CVM instance to go to the login page. The default username is `root`, and the password is the one you set when creating the EMR cluster. Once your credentials are validated, you can enter the command line interface.

Run the following command in EMR command-line interface to switch to the Hadoop user:

```
[root@172 ~]# su hadoop
```

Then, go to the folder where the .jar package is stored and run the following command:

```
[hadoop@10spark]$ spark-submit --class $WordCountOnCOS --master
yarn-cluster $packagename.jar cosn:// $bucketname /$testfile cosn:// $bucketname
/output
```

Here, `$WordCountOnCOS` is your Java Class name, `$packagename` is the name of the .jar package generated in the new Maven project you created, `$bucketname` is your bucket name plus path, and `$testfile` is the name of the file for counting. The output file is stored in the output folder, **which cannot be created beforehand; otherwise, the execution will fail.**

After successful execution, you can see the result of the wordcount task in the specified bucket and folder.

```
[hadoop@172 /]$ hadoop fs -ls cosn:// $bucketname /output
Found 3 items
-rw-rw-rw- 1 hadoop Hadoop 0 2018-06-28 19:20 cosn:// $bucketname /output/_SUCCESS
-rw-rw-rw- 1 hadoop Hadoop 681 2018-06-28 19:20 cosn:// $bucketname /output/part-00000
-rw-rw-rw- 1 hadoop Hadoop 893 2018-06-28 19:20 cosn:// $bucketname /output/part-00001
[hadoop@172 demo]$ hadoop fs -cat cosn://$bucketname/output/part-00000
18/07/05 17:35:01 INFO cosnative.NativeCosFileSystem: Opening 'cosn:// $bucketname/output/part-00000' for reading
(under,1)
(this,3)
(distribution,2)
(Technology,1)
(country,1)
(is,1)
(Jetty,1)
(currently,1)
(permitted.,1)
(Security,1)
(have,1)
(check,1)
```

Using Spark Python to Analyze Data in COS

Last updated : 2021-07-08 10:43:44

This section describes running a Spark wordcount application in Python.

Development Preparations

- This task requires access to COS, so you need to [create a bucket](#) in COS first.
- Create an EMR cluster. When creating the EMR cluster, you need to select the Spark component on the software configuration page and enable access to COS on the basic configuration page.

Data Preparations

Upload the to-be-processed file to COS first. If the file is in your local storage, upload it directly via the [COS console](#); if it is in the EMR cluster, upload it by running the following Hadoop command:

```
[hadoop@10 hadoop]$ hadoop fs -put $testfile cosn:// $bucketname/
```

Here, \$testfile is the full path with file name and \$bucketname is your bucket name. After the upload is completed, you can check whether the file is available in COS.

Running the Demo

First, log in to any node (preferably a master one) in the EMR cluster. For information about how to log in to EMR, see [Logging in to Linux Instance Using Standard Login Method](#). Here, you can use WebShell to log in. Click **Login** on the right of the desired CVM instance to go to the login page. The default username is `root`, and the password is the one you set when creating the EMR cluster. Once your credentials are validated, you can enter the command line interface.

Run the following command on the EMR command-line interface to switch to the Hadoop user and go to the Spark installation directory `/usr/local/service/spark` :

```
[root@172 ~]# su hadoop
[hadoop@172 root]$ cd /usr/local/service/spark
```

Create a Python file named wordcount.py and add the following code:


```
from __future__ import print_function
import sys
from operator import add
from pyspark.sql import SparkSession
if __name__ == "__main__":
if len(sys.argv) != 3:
print("Usage: wordcount <file>", file=sys.stderr)
exit(-1)
spark = SparkSession\
.builder\
.appName("PythonWordCount")\
.getOrCreate()
sc = spark.sparkContext
lines = spark.read.text(sys.argv[1]).rdd.map(lambda r: r[0])
counts = lines.flatMap(lambda x: x.split(' ')) \
.map(lambda x: (x, 1)) \
.reduceByKey(add)
output = counts.collect()
counts.saveAsTextFile(sys.argv[2])
spark.stop()
```

Submit the task by running the following command:

```
[hadoop@10 spark]$ ./bin/spark-submit --master yarn ./wordcount.py
cosn://$bucketname/$yourtestfile cosn:// $bucketname/$output
```

Here, `$bucketname` is your COS bucket name, `$yourtestfile` is the full path with test file name in the bucket, and `$output` is your output folder. **If the `$output` folder already exists before the command is executed, the program will fail.**

After the program is running automatically, you can find the output file in the destination bucket:

```
[hadoop@172 spark]$ hadoop fs -ls cosn:// $bucketname/$output
Found 2 items
-rw-rw-rw- 1 hadoop Hadoop 0 2018-06-29 15:35 cosn:// $bucketname/$output /_SUCCESS
-rw-rw-rw- 1 hadoop Hadoop 2102 2018-06-29 15:34 cosn:// $bucketname/$output /part-00000
```

You can also look up the the result by running the following command:

```
[hadoop@172 spark]$ hadoop fs -cat cosn:// $bucketname/$output /part-00000
(u'', 27)
(u'code', 1)
(u'both', 1)
```

```
(u'Hadoop', 1)
(u'Bureau', 1)
(u'Department', 1)
```

You can also output the result to HDFS by changing the output location in the command as follows:

```
[hadoop@10spark]$ ./bin/spark-submit ./wordcount.py
cosn://$bucketname/$yourtestfile /user/hadoop/$output
```

Here, `/user/hadoop/` is the path in HDFS. If this path does not exist, you can create one.

After the task is completed, you can view the Spark execution log by running the following command:

```
[hadoop@10 spark]$ /usr/local/service/hadoop/bin/yarn logs -applicationId $yourId
```

Here, `$yourId` should be replaced with your task ID, which can be viewed in Yarn's WebUI.

SparkSQL Tutorial

Last updated : 2020-11-20 12:59:00

Spark SQL is Apache Spark's module for structured data processing. It provides a DataFrame abstraction in a variety of languages to simplify working with structured datasets and lets you query the data with distributed SQL.

1. Preparations for Development

Confirm that you have activated Tencent Cloud and created an EMR cluster. When creating the EMR cluster, select the Spark component on the software configuration page.

2. Using the Interactive SparkSQL Console

First, log in to a master node of the EMR cluster before using SparkSQL. For more information on how to log in to EMR, please see [Logging in to a Linux Instance](#). Here, you can use WebShell to log in. Click *Login* button on the right of the desired CVM instance and then enter the login page. The default username is root, and the password is the one you set when creating the EMR cluster. Once your credentials have been validated, you can access to the EMR command-line interface.

Run the following command in EMR command-line interface to switch to the Hadoop user and go to the directory

```
/usr/local/service/spark :
```

```
[root@172 ~]# su hadoop
[hadoop@172 root]$ cd /usr/local/service/spark
```

You can access the interactive SparkSQL Console by running the following command:

```
[hadoop@10spark]$ bin/spark-sql --master yarn --num-executors 64 --executor-memory 2g
```

Here, --master indicates your master URL, --num-executors the number of executors, and --executor-memory the storage capacity of the executors. You can modify these parameters based on your actual conditions and start/stop a SparkSQLthriftserver through `sbin/start-thriftserver.sh` / `sbin/stop-thriftserver.sh` .

Below are some basic operations in SparkSQL :

- Create and view a database:

```
spark-sql> create database sparksql;  
Time taken: 0.907 seconds
```

```
spark-sql> show databases;  
default  
sparksql  
test  
Time taken: 0.131 seconds, Fetched 5 row(s)
```

- **Create a new table in the database** you just created **and view the table:**

```
spark-sql> use sparksql;  
Time taken: 0.076 seconds  
  
spark-sql> create table sparksql_test(a int,b string);  
Time taken: 0.374 seconds  
  
spark-sql> show tables;  
sparksql_test false  
Time taken: 0.12 seconds, Fetched 1 row(s)
```

- **Insert two rows of data into the table and view them:**

```
spark-sql> insert into sparksql_test values (42,'hello'),(48,'world');  
Time taken: 2.641 seconds  
  
spark-sql> select * from sparksql_test;  
42 hello  
48 world  
Time taken: 0.503 seconds, Fetched 2 row(s)
```

For more information on **Spark command line parameters**, please see the [community documentation] (<http://spark.apache.org/docs/latest/sql-programming-guide.html>).

3. Creating a Project with Maven

Download **and** install Maven **first and then** configure its environment variables. If you are **using the IDE**, please **set the** Maven-related configuration **items in the IDE**.

Creating a Maven project

Enter **the directory of the** Maven project, such as ``D://mavenWorkplace``, **and create the project by running the** following commands:

```
mvn archetype:generate -DgroupId=$yourgroupId -DartifactId=$yourartifactID
-DarchetypeArtifactId=maven-archetype-quickstart
```

Here, `$yourgroupId` is your package name; `$yourartifactID` is your project name; `maven-archetype-quickstart` indicates **to create a** Maven Java project. Some **files need to be** downloaded during **the** project is being created, so please stay connected **to the** Internet.

After successfully creating **the** project, you will see **a folder** named `$yourartifactID` **in the** ``D://mavenWorkplace`` **directory**. The **files included in the folder** have **the** following structure:

simple

```
---pom.xml          Core configuration, under the project root directory
---src
  ---main
    ---java          Java source code directory
  ---resources      Java configuration file directory
  ---test
    ---java          Test source code directory
    ---resources      Test configuration directory
```

Among **the files** above, pay extra attention **to the** `pom.xml` **file and the** Java **folder** under **the** `main` **directory**. The `pom.xml` **file** is primarily used **to create** dependencies **and** package configurations; **the** Java **folder** is used **to** store your source code.

Adding Hadoop dependencies and sample code

First, **add the** Maven dependencies **to the** `pom.xml` **file**:

```
org.apache.spark    spark-core_2.11    2.0.2
org.apache.spark    spark-sql_2.11    2.0.2
```

Then, add the packaging and compiling plugins to the `pom.xml` file: `org.apache.maven.plugins maven-compiler-plugin 1.8 1.8 utf-8 maven-assembly-plugin jar-with-dependencies make-assembly package single` Below is an example of a complete `pom.xml` file: `4.0.0`

```
<groupId>$yourgroupId </groupId>
<artifactId>$yourartifactID </artifactId>
<version>1.0-SNAPSHOT</version>

<dependencies>
<dependency>
<groupId>org.apache.spark</groupId>
```

```
<artifactId>spark-core_2.11</artifactId>
<version>2.0.2</version>
</dependency>
<!--spark sql-->
<dependency>
<groupId>org.apache.spark</groupId>
<artifactId>spark-sql_2.11</artifactId>
<version>2.0.2</version>
</dependency>
</dependencies>

<build>
<plugins>
<plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-compiler-plugin</artifactId>
<configuration>
<source>1.8</source>
<target>1.8</target>
<encoding>utf-8</encoding>
</configuration>
</plugin>
<plugin>
<artifactId>maven-assembly-plugin</artifactId>
<configuration>
<descriptorRefs>
<descriptorRef>jar-with-dependencies</descriptorRef>
</descriptorRefs>
</configuration>
<executions>
<execution>
<id>make-assembly</id>
<phase>package</phase>
<goals>
<goal>single</goal>
</goals>
</execution>
</executions>
</plugin>
</plugins>
</build>
```

"" >!Replace \$yourgroupId and \$yourartifactID with your real information.

Then, create a Java Class named Demo.java in the main>Java folder and add the following code to it:

```
import org.apache.spark.rdd.RDD;
import org.apache.spark.sql.Dataset;
import org.apache.spark.sql.Row;
import org.apache.spark.sql.Session;

/**
 * Created by tencent on 2018/6/28.
 */
public class Demo {
    public static void main(String[] args){
        SparkSession spark = SparkSession
            .builder()
            .appName("Java Spark Hive Example")
            .enableHiveSupport()
            .getOrCreate();

        Dataset<Row> df = spark.read().json(args[0]);

        RDD<Row> test = df.rdd();

        test.saveAsTextFile(args[1]);
    }
}
```

Compiling code and packaging it for upload

Use the local command prompt to enter the project directory and run the following command to compile and package the project:

```
mvn package
```

"Build success" indicates that package is successfully created. You can see the generated .jar package in the target folder under the project directory.

Use the scp or sftp tool to upload the compressed .jar package to the EMR cluster. Run the following command in your local shell:

```
scp $localfile root@public IP address:$remotefolder
```

Here, \$localfile is the path and the name of your local file; root is the CVM instance username. You can look up the public IP address in the node information in the EMR or CVM Console. \$remotefolder is the path where you want to store the file in the CVM instance. After the upload is completed, you can check whether the file is in the corresponding folder on the EMR command line.

4. Preparing the Data and Running the Demo

You can use SparkSQL to process data stored in HDFS. First, upload the data to HDFS. The built-in file `people.json` stored in `/usr/local/service/spark/examples/src/main/resources/` is used as an example here. Run the following command to upload it to HDFS:

```
[hadoop@10 hadoop]$ hadoop fs -put /usr/local/service/spark/examples/src/main/resources/people.json /user/hadoop
```

You can choose a different test file. Here, `/user/hadoop/` is a folder under HDFS, which you can create if it does not exist.

Run the demo. First, log in to a master node of the EMR cluster and switch to the Hadoop user, as shown in the interactive SparkSQL Console. Run the following command:

```
[hadoop@10spark]$ bin/spark-submit --class Demo --master yarn-client $yourjarpackage /user/hadoop/people.json /user/hadoop/$output
```

Here, `--class` is the executed entry class. In this example, `Demo` is the class, which is also the name of the Java Class you created when adding Hadoop dependencies and sample code. `--master` is the master URL of the cluster, `$yourjarpackage` is the package name, and `$output` is the output folder (**if the `$output` folder already exists before the command is executed, the program will fail**).

After the program is successfully executed, you can see the result in `/user/hadoop/$output` :

```
[hadoop@172 spark]$ hadoop fs -cat /user/hadoop/$output/part-00000
[null,Michael]
[30,Andy]
[19,Justin]
```

For more `spark-submit` parameters, run the following commands, or see the [official documentation](#).

```
[hadoop@10spark]$ spark-submit -h
```


Integrating Spark Streaming with Ckafka

Last updated : 2020-10-10 17:58:21

Tencent Cloud Elastic MapReduce (EMR) allows you to realize the following streaming applications with CKafka:

- Log information stream processing
- User behavior record stream processing
- Alarm information collection and processing
- Messaging

1. Preparations for Development

- This job is required to access to CKafka, so you need to create a CKafka instance first. For more information, please see [CKafka](#).
- Confirm that you have activated Tencent Cloud and created an EMR cluster. When creating the EMR cluster, select the Spark component on the software configuration page.

2. Using Kafka Toolkit in EMR Cluster

First, you need to check the private IP and port number of CKafka. Log in to the CKafka Console, select the CKafka instance you want to use, and view its private IP as \$kafkaIP in the basic information section, and the port number is generally defaulted to 9092. Create a topic named spark_streaming_test on the topic management page.

Log in to any node (preferably a master one) in the EMR cluster. For more information on how to log in to EMR, please see [Logging in to Linux Instances](#). Here, you can choose to log in with WebShell. Click "Log in" on the right of the desired CVM instance to enter the login page. The default username is `root`, and the password is the one you set when creating the EMR cluster. Once the correct credentials are entered, you can enter the command line interface.

Run the following command in EMR command-line interface to switch to the Hadoop user and go to the directory

```
/usr/local/service/spark :
```

```
[root@172 ~]# su hadoop
[root@172 root]$ cd /usr/local/service/spark
```

Download the installation package [Kafka's official website](#). A Kafka client is recommended as it is most compatible with Tencent Cloud CKafka. Then, Decompress the package and move the extracted folder to the `/opt` directory:

```
[hadoop@172 data]$ tar -xzvf kafka_2.10-0.10.2.0.tgz
[hadoop@172 data]$ mv kafka_2.10-0.10.2.0 /opt/
```

Once the package is decompressed, you can use Kafka. Run the `telnet` command to see whether the EMR cluster is connected to the CKafka instance:

```
[hadoop@172 kafka_2.10-0.10.2.0]$ telnet $kafkaIP 9092
Trying $kafkaIP...
Connected to $kafkaIP.
```

`$kafkaIP` is the private IP address of the CKafka instance you created.

The following example describes how to test the Kafka toolkit. Log in to the EMR cluster in two WebShell terminals, switch to the Hadoop user, and go to the Kafka installation path:

```
[root@172 ~]# su hadoop
[hadoop@172 root]$ cd /opt/kafka_2.10-0.10.2.0/
```

Connect to CKafka on the first terminal and send the following message:

```
[hadoop@172 kafka_2.10-0.10.2.0]$ bin/kafka-console-producer.sh --broker-list $kafkaIP:9092
--topic spark_streaming_test
hello world
this is a message
```

Connect to CKafka on the other terminal. Now, as a consumer, you are able to access or consume records from a Kafka cluster:

```
[hadoop@172 kafka_2.10-0.10.2.0]$ bin/kafka-console-consumer.sh --bootstrap-server $kafkaIP:9092 --from-beginning --new-consumer --topic spark_streaming_test
hello world
this is a message
```

3. Connecting Spark Streaming to CKafka

On the consumer side, Spark Streaming is used to continuously pull data from CKafka for word frequency counting, i.e. performing the WordCount job on the streaming data. On the producer side, a program is used to constantly generate data which is continuously delivered to CKafka.

[Download and install Maven](#) first and then configure its environment variables. If you are using IDE, please configure Maven-related items in your IDE.

Creating a Spark Streaming consumer project

Enter the directory for your Maven project, such as `D://mavenWorkplace` , by running the following commands:

```
mvn archetype:generate -DgroupId=$yourgroupId -DartifactId=$yourartifactID
-DarchetypeArtifactId=maven-archetype-quickstart
```

Here, \$yourgroupId is your package name, \$yourartifactID is your project name, and maven-archetype-quickstart indicates to create a Maven Java project. Some files need to be downloaded during the process, so please keep the Internet connected.

After successfully creating the project, you will see a folder named \$yourartifactID in the `D://mavenWorkplace` directory. The files included in the folder have the following structure:

```
simple
  ---pom.xml           Core configuration, under the project root directory
  ---src
    ---main
      ---java          Java source code directory
    ---resources      Java configuration file directory
  ---test
    ---java           Test source code directory
    ---resources      Test configuration directory
```

Among the files above, pay extra attention to the pom.xml file and the Java folder under the main directory. The pom.xml file is primarily used to create dependencies and package configurations; the Java folder is used to store your source code.

First, add the Maven dependencies to the pom.xml file:

```
<dependencies>
<dependency>
<groupId>org.apache.spark</groupId>
<artifactId>spark-core_2.11</artifactId>
<version>2.0.2</version>
</dependency>
<dependency>
<groupId>org.apache.spark</groupId>
<artifactId>spark-streaming_2.11</artifactId>
<version>2.0.2</version>
</dependency>
<dependency>
<groupId>org.apache.spark</groupId>
```

```
<artifactId>spark-streaming-kafka-0-10_2.11</artifactId>
<version>2.0.2</version>
</dependency>
</dependencies>
```

Then, add the packaging and compiling plugins to the pom.xml file:

```
<build>
<plugins>
<plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-compiler-plugin</artifactId>
<configuration>
<source>1.8</source>
<target>1.8</target>
<encoding>utf-8</encoding>
</configuration>
</plugin>
<plugin>
<artifactId>maven-assembly-plugin</artifactId>
<configuration>
<descriptorRefs>
<descriptorRef>jar-with-dependencies</descriptorRef>
</descriptorRefs>
</configuration>
<executions>
<execution>
<id>make-assembly</id>
<phase>package</phase>
<goals>
<goal>single</goal>
</goals>
</execution>
</executions>
</plugin>
</plugins>
</build>
```

Note :

Replace \$yourgroupId and \$yourartifactId with your real information.

Then, add the sample code by creating a Java Class named KafkaTest.java in the main>Java folder and adding the following code to it:

```
import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.apache.spark.SparkConf;
import org.apache.spark.api.java.JavaSparkContext;
import org.apache.spark.streaming.Durations;
import org.apache.spark.streaming.api.java.JavaInputDStream;
import org.apache.spark.streaming.api.java.JavaPairDStream;
import org.apache.spark.streaming.api.java.JavaStreamingContext;
import org.apache.spark.streaming.kafka010.ConsumerStrategies;
import org.apache.spark.streaming.kafka010.KafkaUtils;
import org.apache.spark.streaming.kafka010.LocationStrategies;
import scala.Tuple2;

import java.util.*;
import java.util.concurrent.TimeUnit;

/**
 * Created by tencent on 2018/7/3.
 */
public class KafkaTest {
    public static void main(String[] args) throws InterruptedException {
        String brokers = "$kafkaIP:9092";
        String topics = "spark_streaming_test1"; // Subscribed topics; multiple topics should be separated by ','
        int durationSeconds = 60; // Interval
        SparkConf conf = new SparkConf().setAppName("spark streaming word count");
        JavaSparkContext sc = new JavaSparkContext(conf);
        JavaStreamingContext ssc = new JavaStreamingContext(sc, Durations.seconds(durationSeconds));
        Collection<String> topicsSet = new HashSet<>(Arrays.asList(topics.split(",")));
        // Kafka-related parameter
        Map<String, Object> kafkaParams = new HashMap<>();
        kafkaParams.put("metadata.broker.list", brokers);
        kafkaParams.put("bootstrap.servers", brokers);
        kafkaParams.put("group.id", "group1");
        kafkaParams.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");
        kafkaParams.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
        kafkaParams.put("value.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
        // Create a connection
        JavaInputDStream<ConsumerRecord<Object, Object>> lines = KafkaUtils.createDirectStream(
            ssc,
            LocationStrategies.PreferConsistent(),
            ConsumerStrategies.Subscribe(topicsSet, kafkaParams)
        );
    }
}
```

```
);  
// wordcount logic  
JavaPairDStream<String, Integer> counts = lines  
.flatMap(x -> Arrays.asList(x.value().toString().split(" ")).iterator())  
.mapToPair(x -> new Tuple2<String, Integer>(x, 1))  
.reduceByKey((x, y) -> x + y);  
// Save the result  
counts.dstream().saveAsTextFiles("$hdfsPath", "result");  
//  
ssc.start();  
ssc.awaitTermination();  
ssc.close();  
}  
}
```

Pay attention to the following settings in the code:

- The brokers variable should be set to the private IP of the CKafka instance found in step 2.
- The topics variable should be set to the name of the topic you created, e.g., spark_streaming_test1 here.
- durationSeconds is the interval for the program to consume the data in CKafka, e.g., 60 seconds here.
- \$hdfsPath is the path in HDFS to which the result will be output.

Use the local command prompt to enter the project directory and run the following command to compile and package the project:

```
mvn package
```

"Build success" indicates that package is successfully created. You can see the generated .jar package in the target folder under the project directory.

Upload the package file to the EMR cluster with the scp or sftp tool. Be sure to include the dependencies in the .jar package to be uploaded:

```
scp $localfile root@public IP address:$remotefolder
```

Here, \$localfile is the path and the name of your local file; root is the CVM instance username. You can look up the public IP address in the node information in the EMR or CVM Console. \$remotefolder is the path where you want to store the file in the CVM instance. After the upload is completed, you can check whether the file is in the corresponding folder on the EMR command line.

Creating a Spark Streaming producer project

Enter the directory for your Maven project, such as `D://mavenWorkplace`, by running the following commands:

```
mvn archetype:generate -DgroupId=$yourgroupId -DartifactId=$yourartifactID
-DarchetypeArtifactId=maven-archetype-quickstart
```

First, add the Maven dependencies to the pom.xml file:

```
<dependencies>
<dependency>
<groupId>org.apache.kafka</groupId>
<artifactId>kafka_2.11</artifactId>
<version>0.10.1.0</version>
</dependency>
<dependency>
<groupId>org.apache.kafka</groupId>
<artifactId>kafka-clients</artifactId>
<version>0.10.1.0</version>
</dependency>
<dependency>
<groupId>org.apache.kafka</groupId>
<artifactId>kafka-streams</artifactId>
<version>0.10.1.0</version>
</dependency>
</dependencies>
```

Then, add the packaging and compiling plugins to the pom.xml file:

```
<build>
<plugins>
<plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-compiler-plugin</artifactId>
<configuration>
<source>1.8</source>
<target>1.8</target>
<encoding>utf-8</encoding>
</configuration>
</plugin>
<plugin>
<artifactId>maven-assembly-plugin</artifactId>
<configuration>
<descriptorRefs>
<descriptorRef>jar-with-dependencies</descriptorRef>
</descriptorRefs>
</configuration>
<executions>
<execution>
```

```
<id>make-assembly</id>
<phase>package</phase>
<goals>
<goal>single</goal>
</goals>
</execution>
</executions>
</plugin>
</plugins>
</build>
```

⚠ Note :

Replace \$yourgroupId and \$yourartifactId with your real information.

Then, add the sample code by creating a Java Class named SendData.java in the main>Java folder and adding the following code to it:

```
import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.ProducerRecord;
import java.util.Properties;

/**
 * Created by tencent on 2018/7/4.
 */
public class SendData {
public static void main(String[] args) {

Properties props = new Properties();
props.put("bootstrap.servers", "$kafkaIP:9092");
props.put("acks", "all");
props.put("retries", 0);
props.put("batch.size", 16384);
props.put("linger.ms", 1);
props.put("buffer.memory", 33554432);
props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializ
er");
props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerial
izer");
// The producer sends a message
String topic = "spark_streaming_test1";
org.apache.kafka.clients.producer.Producer<String, String> procuder = new KafkaPr
oducer<String, String>(props);
```



```
while (true) {
    int num = (int) ((Math.random()) * 10);
    for (int i = 0; i <= 10; i++) {
        int tmp = (num+i)%10;
        String value = "value_" + tmp;
        ProducerRecord<String, String> msg = new ProducerRecord<String, String>(topic, value);
        producer.send(msg);
    }

    try {Thread.sleep(1000*10);}
    catch (InterruptedException e) {}
}
}
```

Replace \$kafkaIP with the private IP address of your CKafka instance.

This program sends 10 messages from value_0 to value_9 to CKafka every 10 seconds, starting at a random order.

For more information on the parameters in the program, please see the consumer program.

Use the local command prompt to enter the project directory and run the following command to compile and package the project:

```
mvn package
```

"Build success" indicates that package is successfully created. You can see the generated .jar package in the target folder under the project directory.

Upload the package file to the EMR cluster with the scp or sftp tool. Be sure to include the dependencies in the .jar package to be uploaded:

```
scp $localfile root@public IP address:$remotefolder
```

Using a program to consume CKafka data

Use two interfaces to log in to the WebShell of the EMR cluster.

In the first interface: log in to a master node of the EMR cluster and switch to the Hadoop user, as shown in section 2. Run the following command to run the demo:

```
[hadoop@172 ~]$ bin/spark-submit --class KafkaTest --master yarn-cluster $consume
rpackage
```

The parameters are as follows:

- --class indicates the entry class to be executed, e.g., KafkaTest in this example

- `--master` is the master URL of the cluster.
- `$consumerpackage` is the package name of the packaged consumer program.

After the program is started, it will run continuously in the Yarn cluster. Run the following command to view the status of the program running:

```
[hadoop@172 ~]$ yarn application -list
```

In the second interface: log in to the WebShell of EMR and run the producer program, so that Spark Streaming can retrieve the data for consumption.

```
[hadoop@172 spark]$ bin/spark-submit --class SendData $producerpackage
```

Here, `$producerpackage` is the package name of the packaged producer program. The result of the wordcount job will be output to the specified HDFS folder in a while. You can view in HDFS the result of Spark Streaming's consumption of the CKafka data:

```
[hadoop@172 root]$ hdfs dfs -ls /user
Found 9 items
drwxr-xr-x - hadoop supergroup 0 2018-07-03 16:37 /user/hadoop
drwxr-xr-x - hadoop supergroup 0 2018-06-19 10:10 /user/hive
-rw-r--r-- 3 hadoop supergroup 0 2018-06-29 10:19 /user/pythontest.txt
drwxr-xr-x - hadoop supergroup 0 2018-07-05 20:25 /user/sparkstreamingtest-153079
3500000.result

[hadoop@172 root]$ hdfs dfs -cat /user/sparkstreamingtest-1530793500000.result/*
(value_6,16)
(value_7,22)
(value_8,18)
(value_0,18)
(value_9,17)
(value_1,18)
(value_2,17)
(value_3,17)
(value_4,16)
(value_5,17)
```

Finally, exit the KafkaTest program in the Yarn cluster:

```
[hadoop@172 ~]$ yarn application -kill $Application-Id
```

Here, `$Application-Id` is the ID found by running the `yarn application -list` command.

For more information on Kafka, please see the [official documentation](#).

Practices on Dynamic Scheduling of Spark Resources

Last updated : 2021-10-08 11:23:06

Preparations for Development

Confirm that you have activated Tencent Cloud and created an EMR cluster. When creating the EMR cluster, you need to select the spark_hadoop component on the software configuration page.

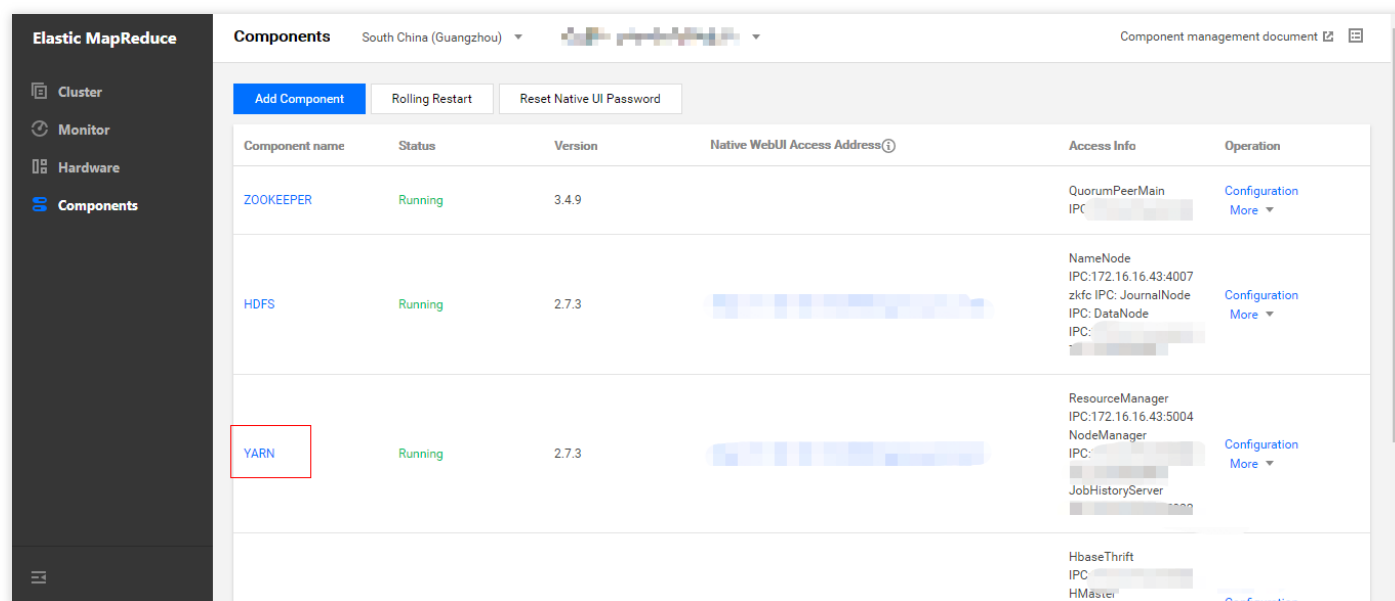
Spark is installed in the `/usr/local/service/` path (`/usr/local/service/spark`) in the CVM instance for the EMR cluster.

Copying JAR Package

You need to copy `spark-<version>-yarn-shuffle.jar` to the `/usr/local/service/hadoop/share/hadoop/yarn/lib` directory of all nodes in the cluster.

Method 1. Use the SSH Console

1. In **Cluster Service** > **YARN**, select **Operation** > **Role Management** and confirm the IP of the node where NodeManager resides.



The screenshot shows the Elastic MapReduce console interface. The left sidebar contains navigation options: Cluster, Monitor, Hardware, and Components. The main content area is titled 'Components' and shows a table of installed components. The YARN component is highlighted with a red box. The table columns are Component name, Status, Version, Native WebUI Access Address, Access Info, and Operation.

Component name	Status	Version	Native WebUI Access Address	Access Info	Operation
ZOOKEEPER	Running	3.4.9		QuorumPeerMain IPC: [redacted]	Configuration More
HDFS	Running	2.7.3	[redacted]	NameNode IPC: 172.16.16.43:4007 zkfc IPC: JournalNode IPC: DataNode IPC: [redacted]	Configuration More
YARN	Running	2.7.3	[redacted]	ResourceManager IPC: 172.16.16.43:5004 NodeManager IPC: [redacted] JobHistoryServer [redacted]	Configuration More
				HbaseThrift IPC: [redacted] HMaster	Configuration

2. Log in to the nodes where NodeManager resides one by one.

- You need to log in to any node (preferably a master one) in the EMR cluster. For more information on how to log in to EMR, please see [Logging in to Linux Instance](#). Here, you can log in by using XShell.
- Use SSH to log in to other nodes where NodeManager resides. The used command is `ssh $user@$ip`, where `$user` is the login username, and `$ip` is the remote server IP (i.e., IP address confirmed in step 1).

```
[root@172 ~]# ssh root@172.16.1.1
The authenticity of host '172.16.1.1 (172.16.1.1)' can't be established.
ECDSA key fingerprint is b9:16:16:16:16:16:16:16:16:16:16:16:16:16:16:16:2d.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '172.16.1.1' (ECDSA) to the list of known hosts.
root@172.16.1.1's password: █
```

- Verify that the switch is successful.

```
[root@172 ~]# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
    inet 172.16.1.1 netmask 255.255.240.0  broadcast 172.16.1.255
    ether 52:54:00:0d:ad:e3  txqueuelen 1000  (Ethernet)
    RX packets 212406  bytes 110895121 (105.7 MiB)
    RX errors 0  dropped 0  overruns 0  frame 0
    TX packets 178536  bytes 26768271 (25.5 MiB)
    TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    loop txqueuelen 1  (Local Loopback)
    RX packets 378979  bytes 167242376 (159.4 MiB)
    RX errors 0  dropped 0  overruns 0  frame 0
    TX packets 378979  bytes 167242376 (159.4 MiB)
    TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

3. Search for the path of the `spark-<version>-yarn-shuffle.jar` file.

```
[root@172 ~]# find / -name *shuffle.jar
/usr/local/service/hadoop/share/hadoop/yarn/spark-2.3.2-yarn-shuffle.jar
/usr/local/service/spark/yarn/spark-2.3.2-yarn-shuffle.jar
/usr/local/service/hive/spark/yarn/spark-2.0.2-yarn-shuffle.jar
/usr/local/service/apps/kylin-2.5.2/spark/yarn/spark-2.1.2-yarn-shuffle.jar
```

4. Copy `spark-<version>-yarn-shuffle.jar` to `/usr/local/service/hadoop/share/hadoop/yarn/lib`.

```
[root@172 ~]# cd /usr/local/service/hadoop/share/hadoop/yarn/lib/
[root@172 lib]# cp /usr/local/service/spark/yarn/spark-2.3.2-yarn-shuffle.jar spark-2.3.2-yarn-shuffle.jar
[root@172 lib]# ls
activation-1.1.jar          hadoop-lzo-0.4.20.jar      jetty-util-6.1.26.jar
aopalliance-1.0.jar        jackson-core-asl-1.9.13.jar json-io-2.5.1.jar
asm-3.2.jar                jackson-jaxrs-1.9.13.jar  jsr305-3.0.0.jar
commons-cli-1.2.jar        jackson-mapper-asl-1.9.13.jar leveldbjni-all-1.8.jar
commons-codec-1.4.jar      jackson-xc-1.9.13.jar    log4j-1.2.17.jar
commons-collections-3.2.2.jar javassist-3.18.1-GA.jar  netty-3.6.2.Final.jar
commons-compress-1.4.1.jar java-util-1.9.0.jar      protobuf-java-2.5.0.jar
commons-io-2.4.jar         javax.inject-1.jar       ranger-plugin-classloader-0.7.1.jar
commons-lang-2.6.jar       jaxb-api-2.2.2.jar      ranger-yarn-plugin-impl
commons-logging-1.1.3.jar  jaxb-impl-2.2.3-1.jar   ranger-yarn-plugin-shim-0.7.1.jar
commons-math-2.2.jar       jersey-client-1.9.jar    servlet-api-2.5.jar
curator-client-2.7.1.jar   jersey-core-1.9.jar      spark-2.3.2-yarn-shuffle.jar
curator-test-2.7.1.jar    jersey-guice-1.9.jar     stax-api-1.0-2.jar
fst-2.50.jar              jersey-json-1.9.jar      xz-1.0.jar
guava-11.0.2.jar          jersey-server-1.9.jar   zookeeper-3.4.6.jar
guice-3.0.jar             jettison-1.1.jar        zookeeper-3.4.6-tests.jar
guice-servlet-3.0.jar     jetty-6.1.26.jar
[root@172 lib]# ls | grep spark-*
spark-2.3.2-yarn-shuffle.jar
```

5. Log out and switch to other nodes.

```
[root@172 lib]# exit
logout
Connection to [REDACTED] closed.
```

Method 2. Use batch deployment script

You need to log in to any node (preferably a master one) in the EMR cluster. For more information on how to log in to EMR, please see [Logging in to Linux Instance](#). Here, you can log in by using XShell.

Write the following Shell script for batch file transfer. When there are many nodes in a cluster, to avoid entering the password for multiple times, you can use `sshpass` for file transfer. `sshpass` provides password-free transfer to eliminate your need to enter the password repeatedly; however, the password plaintext is prone to disclosure and can be found with the `history` command.

1. Install `sshpass` for password-free transfer.

```
[root@172 ~]# yum install sshpass
```

Write the following script:

```
#!/bin/bash
nodes=(ip1 ip2 ... ipn) # List of IPs of all nodes in the cluster separated by spaces
len=${#nodes[@]}
password=<your password>
```

```
file=" spark-2.3.2-yarn-shuffle.jar "  
source_dir="/usr/local/service/spark/yarn"  
target_dir="/usr/local/service/hadoop/share/hadoop/yarn/lib"  
echo $len  
for node in ${nodes[*]}  
do  
echo $node;  
sshpass -p $password scp "$source_dir/$file"root@$node:"$target_dir";  
done
```

2. Transfer files in a non-password-free manner.

Write the following script:

```
#!/bin/bash  
nodes=(ip1 ip2 ... ipn) # List of IPs of all nodes in the cluster separated by spaces  
len=${#nodes[@]}  
password=<your password>  
file=" spark-2.3.2-yarn-shuffle.jar "  
source_dir="/usr/local/service/spark/yarn"  
target_dir="/usr/local/service/hadoop/share/hadoop/yarn/lib"  
echo $len  
for node in ${nodes[*]}  
do  
echo $node;  
scp "$source_dir/$file" root@$node:"$target_dir";  
done
```

Modifying YARN Configuration

1. In **Cluster Service** > **YARN**, select **Operation** > **Configuration Management**. Select the configuration file `yarn-site.xml` and select "cluster level" as the **level** (modifications of configuration items at the cluster level

will be applied to all nodes in the cluster).

The screenshot shows the Elastic MapReduce console interface for YARN configuration management. The 'Level' dropdown is set to 'Cluster level'. A note indicates that special characters like '<->' should be escaped in XML. The configuration table lists various files and their properties. The 'yarn-site.xml' file is selected, and the 'Modify Configuration' button is highlighted with a red box.

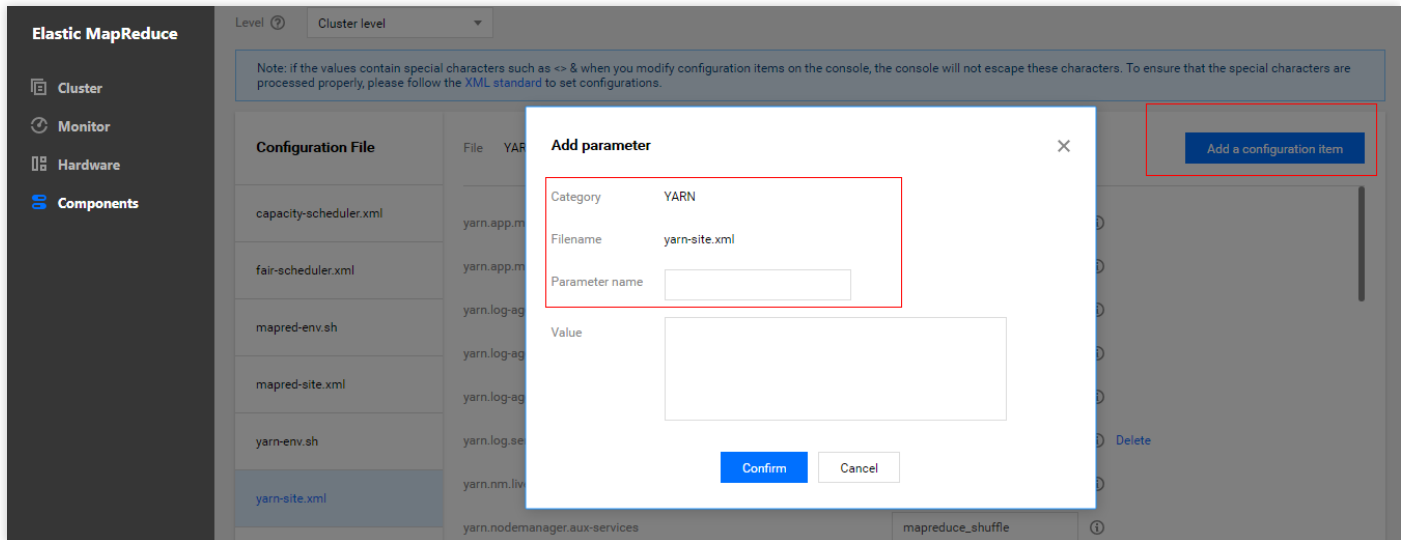
Configuration File	File	Value
capacity-scheduler.xml	yarn.app.mapreduce.am.scheduler.connection.wait.interval.ms	5000
fair-scheduler.xml	yarn.app.mapreduce.am.staging.dir	/emr/hadoop-yarn/staging
mapred-env.sh	yarn.log-aggregation.enable	true
mapred-site.xml	yarn.log-aggregation.retain-check-interval-seconds	604800
yarn-env.sh	yarn.log-aggregation.retain-seconds	604800
yarn-site.xml	yarn.log.server.url	http://...
	yarn.nm.liveness-monitor.expiry-interval.ms	100000
	yarn.nodemanager.aux-services	mapreduce_shuffle
	yarn.nodemanager.aux-services.mapreduce.shuffle.class	org.apache.hadoop.mapred.ShuffleHandler
	yarn.nodemanager.container-executor.class	org.apache.hadoop.yarn.server.nodemanager.DefaultContainerExecutor
	yarn.nodemanager.linux-container-executor.cgroups.hierarchy	/hadoop-yarn
	yarn.nodemanager.linux-container-executor.cgroups.mount	true
	yarn.nodemanager.linux-container-executor.cgroups.mount-path	/sys/fs/cgroup

2. Modify the `yarn.nodemanager.aux-services` configuration item and add `spark_shuffle`.

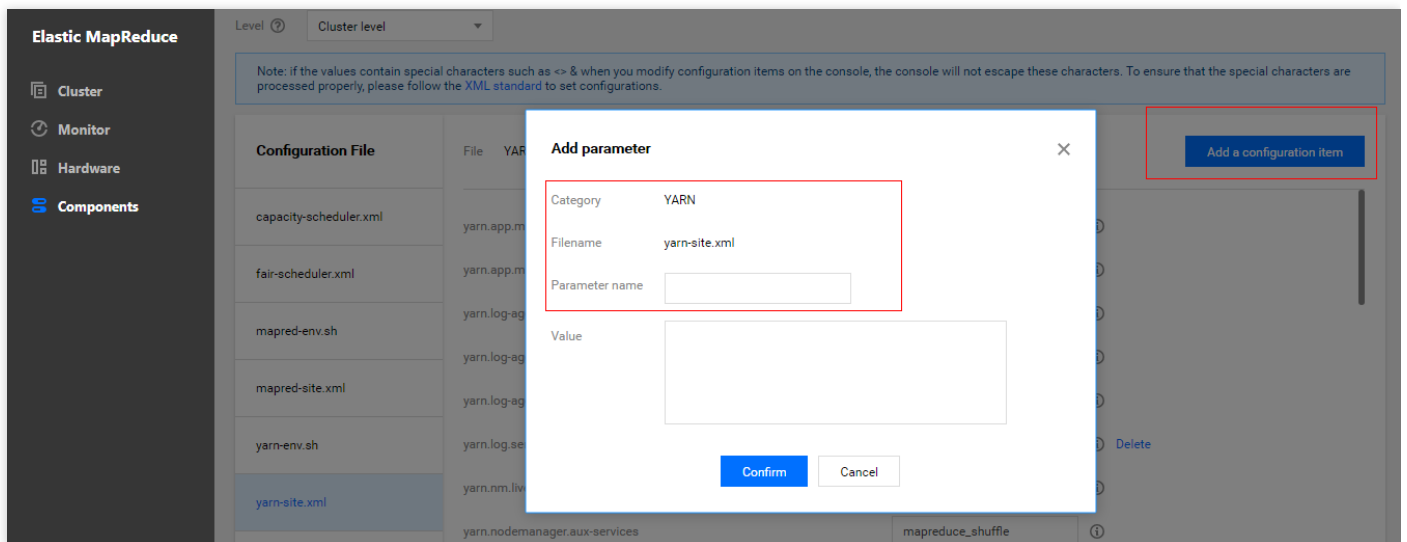
`yarn.nodemanager.aux-services`

`mapreduce_shuffle,spark_shuffle`

3. Add the configuration item `yarn.nodemanager.aux-services.spark_shuffle.class` and set it to `org.apache.spark.network.yarn.YarnShuffleService`.



4. Add the configuration item `spark.yarn.shuffle.stopOnFailure` and set it to `false` .



5. Save and distribute the settings. Restart the YARN component for the configuration to take effect.

Modifying Spark Configuration

1. In **Cluster Service** > **SPARK**, select **Operation** > **Configuration Management**.
2. Select the configuration file **spark-defaults.conf**, click **Modify Configuration**, and create configuration items as shown below:

spark.shuffle.service.enabled	true
spark.dynamicAllocation.enabled	true
spark.dynamicAllocation.minExecutors	1
spark.dynamicAllocation.maxExecutors	30
spark.dynamicAllocation.initialExecutors	1
spark.dynamicAllocation.schedulerBacklogTimeout	1s
spark.dynamicAllocation.sustainedSchedulerBacklogTimeout	5s
spark.dynamicAllocation.executorIdleTimeout	60s

Configuration Item	Value	Remarks
spark.shuffle.service.enabled	true	It starts the shuffle service.
spark.dynamicAllocation.enabled	true	It starts dynamic resource allocation.
spark.dynamicAllocation.minExecutors	1	It specifies the minimum number of executors allocated for each applicat
spark.dynamicAllocation.maxExecutors	30	It specifies the maximum number of executors allocated for each applicat
spark.dynamicAllocation.initialExecutors	1	Generally, its value is the same as th of `spark.dynamicAllocation.minExecut
spark.dynamicAllocation.schedulerBacklogTimeout	1s	If there are pending jobs backlogged more than this duration, new executo will be requested.
spark.dynamicAllocation.sustainedSchedulerBacklogTimeout	5s	If the queue of pending jobs still exist will be triggered again once every several seconds. The number of executors requested per round grows exponentially compared to the previo round.

spark.dynamicAllocation.executorIdleTimeout	60s	If an executor has been idle for more than this duration, it will be deleted by the application.
---	-----	--

3. Save and distribute the configuration and restart the component.

Testing Dynamic Scheduling of Spark Resources

1. Resource configuration description of the testing environment

In the testing environment, there are two nodes where NodeManager is deployed, and each node has a 4 CPU cores and 8 GB memory. The total resources of the cluster are 8 CPU cores and 16 GB memory.

2. Testing job description

Test 1

- In the EMR Console, enter the `/usr/local/service/spark` directory, switch to the "hadoop" user, and run `spark-submit` to submit a job. The data needs to be stored in HDFS.

```
[root@172 ~]# cd /usr/local/service/spark/
[root@172 spark]# su hadoop
[hadoop@172 spark]$ hadoop fs -put ./README.md /
[hadoop@172 spark]$ spark-submit --class org.apache.spark.examples.JavaWordCount --master yarn-client --num-executors 10 --driver-memory 4g --executor-memory 4g --executor-cores 2 ./examples/jars/spark-examples_2.11-2.3.2.jar /README.md /output
```

- In the "Application" panel of the WebUI of the YARN component, you can view the container and CPU allocation before and after the configuration.

ID	User	Name	Application Type	Queue	Application Priority	StartTime	FinishTime	State	FinalStatus	Running Containers	Allocated CPU VCores	Allocated Memory MB	% of Queue	% of Cluster	Progress	Tracking UI	Blacklist Nodes
application_1562242321020_0001	hadoop	spark word count	SPARK	default	0	Thu Jul 4 20:17:09 +0800 2019	N/A	RUNNING	UNDEFINED	3	3	9920	168.2	67.3	<div style="width: 100%;"></div>	ApplicationMaster	0

- Before dynamic resource scheduling is configured, at most 3 CPUs can be allocated.

ID	User	Name	Application Type	Queue	Application Priority	StartTime	FinishTime	State	FinalStatus	Running Containers	Allocated CPU VCores	Allocated Memory MB	% of Queue	% of Cluster	Progress	Tracking UI	Blacklist Nodes
application_1562243747104_0001	hadoop	spark word count	SPARK	root.default	0	Thu Jul 4 20:40:57 +0800 2019	N/A	RUNNING	UNDEFINED	3	5	11264	76.4	76.4	<div style="width: 100%;"></div>	ApplicationMaster	0

- After dynamic resource scheduling is configured, up to 5 CPUs can be allocated.

Conclusion: after dynamic resource scheduling is configured, the scheduler will allocate more resources based on the real-time needs of applications.

Test 2

- In the EMR Console, enter the `/usr/local/service/spark` directory, switch to the "hadoop" user, and run `spark-sql` to start the interactive SparkSQL Console, which is set to use most of the resources in the testing cluster. Configure dynamic resource scheduling and check resource allocation before and after the configuration.

```
[root@172 ~]# cd /usr/local/service/spark/
[root@172 spark]# su hadoop
[hadoop@172 spark]$ spark-sql --master yarn-client --num-executors 5 --driver-memory 4g --executor-memory 2g --executor-cores 1
```

- Use the example for calculating the pi that comes with Spark 2.3.0 as the testing job. When submitting the job, set the number of executors to 5, the driver memory to 4 GB, the executor memory to 4 GB, and the number of executor cores to 2.

```
[root@172 ~]# cd /usr/local/service/spark/
[root@172 spark]# su hadoop
[hadoop@172 spark]$ spark-submit --class org.apache.spark.examples.SparkPi --master yarn-client --num-executors 5 --driver-memory 4g --executor-memory 4g --executor-cores 2 examples/jars/spark-examples_2.11-2.3.2.jar 500
```

ID	User	Name	Application Type	Queue	Application Priority	StartTime	FinishTime	State	FinalStatus	Running Containers	Allocated CPU V-Cores	Allocated Memory MB	% of Queue	% of Cluster	Progress	Tracking UI
application_1562243747104_0005	hadoop	SparkSQL:172.16.32.47	SPARK	root.default	0	Thu Jul 4 21:03:15 +0800 2019	N/A	RUNNING	UNDEFINED	5	5	13312	90.3	90.3	<div style="width: 90.3%;"></div>	ApplicationMaster

- The resource utilization when only the SparkSQL job is running is 90.3%.

ID	User	Name	Application Type	Queue	Application Priority	StartTime	FinishTime	State	FinalStatus	Running Containers	Allocated CPU V-Cores	Allocated Memory MB	% of Queue	% of Cluster	Progress	Tracking UI
application_1562243747104_0006	hadoop	Spark Pi	SPARK	root.default	0	Thu Jul 4 21:04:22 +0800 2019	N/A	ACCEPTED	UNDEFINED	1	1	1024	6.9	6.9	<div style="width: 6.9%;"></div>	ApplicationMaster
application_1562243747104_0005	hadoop	SparkSQL:172.16.32.47	SPARK	root.default	0	Thu Jul 4 21:03:15 +0800 2019	N/A	RUNNING	UNDEFINED	2	2	4096	27.8	27.8	<div style="width: 27.8%;"></div>	ApplicationMaster

- After the SparkPi job is submitted, the resource utilization of SparkSQL becomes 27.8%.

Conclusion: although the SparkSQL job applies for a large amount of resources during submission, no analysis jobs are executed; therefore, there are a lot of idle resources actually. When the idle duration exceeds the limit set by `spark.dynamicAllocation.executorIdleTimeout`, idle executors will be released, and other jobs will get resources. In this test, the cluster resource utilization of the SparkSQL job decreases from 90% to 28%, and idle resources are allocated to the pi calculation job; therefore, automatic scheduling is effective.

Note :

The value of the configuration item `spark.dynamicAllocation.executorIdleTimeout` affects the speed of dynamic resource scheduling. In the test, it is found that the resource scheduling duration is basically the same as this value. You are recommended to adjust this value based on your actual needs for optimal performance.

Spark Integration with Kafka

Last updated : 2022-11-25 16:15:56

Dependencies

Starting from v2.3, Spark no longer supports Kafka 0.8.2. As Spark 2.4.3 and later are integrated into EMR in the production environment, you need to integrate Kafka 0.10.0 and later.

How to view

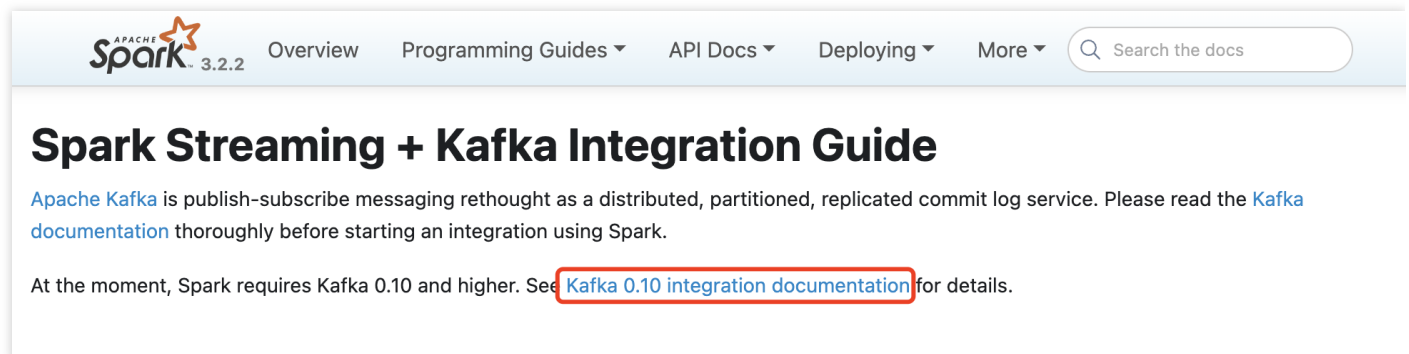
1. Enter the version number in the following URL and open it:

```
https://spark.apache.org/docs/{spark.version}/streaming-kafka-integration.html
```

Replace `{spark.version}` with the actual Spark version. For example, to view the dependencies of v3.2.2, the URL should be as follows:

```
https://spark.apache.org/docs/3.2.2/streaming-kafka-integration.html
```

2. Click the appropriate link to view detailed integration instructions.



The screenshot shows the Apache Spark documentation page for Kafka integration. The page title is "Spark Streaming + Kafka Integration Guide". The navigation bar includes "Overview", "Programming Guides", "API Docs", "Deploying", and "More". A search bar is present with the text "Search the docs". The main content area contains the following text:

Spark Streaming + Kafka Integration Guide

Apache Kafka is publish-subscribe messaging rethought as a distributed, partitioned, replicated commit log service. Please read the [Kafka documentation](#) thoroughly before starting an integration using Spark.

At the moment, Spark requires Kafka 0.10 and higher. See [Kafka 0.10 integration documentation](#) for details.

Spark Dependencies in Each EMR Version

Last updated : 2022-11-25 16:06:47

Dependencies

Spark	Scala	Python	R	Java
2.4.3	2.12.x	2.7+/3.4+	3.1+	8+
3.0.0	2.12.x	2.7+/3.4+	3.1+	8/11
3.0.2	2.12.x	2.7+/3.4+	3.5+	8/11
3.2.1	2.12.x/2.13.x	3.6+	3.5+	8/11
3.2.2	2.12.x/2.13.x	3.6+	3.5+	8/11

How to view

1. Enter the version number in the following URL and open it:

```
https://spark.apache.org/docs/{spark.version}/index.html
```

Replace `{spark.version}` with the actual Spark version. For example, to view the dependencies of v3.2.2, the URL should be as follows:

```
https://spark.apache.org/docs/3.2.2/index.html
```

2. View the dependencies.

Note: Kafka 0.8 support is deprecated as of Spark 2.3.0.

	spark-streaming-kafka-0-8	spark-streaming-kafka-0-10
Broker Version	0.8.2.1 or higher	0.10.0 or higher
API Maturity	Deprecated	Stable
Language Support	Scala, Java, Python	Scala, Java
Receiver DStream	Yes	No
Direct DStream	Yes	Yes
SSL / TLS Support	No	Yes
Offset Commit API	No	Yes
Dynamic Topic Subscription	No	Yes

Hbase Development Guide

Using HBase Through API

Last updated : 2022-11-07 16:26:04

HBase is an open-source, high-reliability, high-performance, column-oriented, scalable distributed storage system developed based on Google BigTable. It uses Hadoop file system (HDFS) as the file storage system, Hadoop MapReduce for processing massive amounts of data in HBase, and ZooKeeper for collaboration.

- HBase consists of ZooKeeper, HMaster, and HRegionServer. ZooKeeper prevents single points of failure on HMaster, and its master election mechanism guarantees that there is always a master node available in the cluster.
- HMaster manages the CRUD operations on the tables and the load balancing of the HRegionServers. In addition, when an HRegionServer exits, it moves the HRegion of that HRegionServer to another.
- HRegionServer is the core module in HBase and responsible for reading and writing data from and to HDFS based on user's I/O requests. HRegionServer internally manages a series of HRegion objects, each of which corresponds to a Region and consists of multiple Stores. Each Store corresponds to the storage in the Column Family.

This development guide describes how to use an EMR cluster for development from a technical perspective. For data security reasons, only VPC access is currently supported for EMR.

1. Development Preparations

- Confirm that you have activated Tencent Cloud and created an EMR cluster. When creating the EMR cluster, select the HBase and ZooKeeper components on the software configuration page.

2. Using HBase Shell

Log in to a master node of the EMR cluster first before using HBase Shell. For more information on how to log in to EMR, please see [Logging in to Linux Instance Using Standard Login Method](#). You can choose to log in with WebShell. Click "Log in" on the right of the desired CVM instance to enter the login page. The default username is `root`, and the password is the one you set when creating the EMR cluster. Once the correct information is entered, you can enter the EMR command line interface.

Run the following command on the EMR command line interface to switch to the Hadoop user and go to the directory

```
/usr/local/service/hbase :
```



```
[root@172 ~]# su hadoop
[hadoop@10root]$ cd /usr/local/service/hbase
```

You can enter HBase Shell by running the following command:

```
[hadoop@10hbase]$ bin/hbase shell
```

Enter `help` in HBase Shell to see basic usage information and command examples. Next, create a table by running the following command:

```
hbase(main):001:0> create 'test', 'cf'
```

Once the table is created, you can run the `list` command to see whether it exists.

```
hbase(main):002:0> list 'test'
TABLE
test
1 row(s) in 0.0030 seconds
=> ["test"]
```

Run the `put` command to add elements to the table you created:

```
hbase(main):003:0> put 'test', 'row1', 'cf:a', 'value1'
0 row(s) in 0.0850 seconds
hbase(main):004:0> put 'test', 'row2', 'cf:b', 'value2'
0 row(s) in 0.0110 seconds
hbase(main):005:0> put 'test', 'row3', 'cf:c', 'value3'
0 row(s) in 0.0100 seconds
```

Three values are added to the created table. The first is "value1" inserted into row "row1" column "cf:a", and so on.

Run the `scan` command to traverse the entire table:

```
hbase(main):006:0> scan 'test'
ROW COLUMN+CELL
row1 column=cf:a, timestamp=1530276759697, value=value1
row2 column=cf:b, timestamp=1530276777806, value=value2
row3 column=cf:c, timestamp=1530276792839, value=value3
3 row(s) in 0.2110 seconds
```

Run the `get` command to get the value of the specified row in the table:

```
hbase(main):007:0> get 'test', 'row1'
COLUMN CELL
```

```
cf:a timestamp=1530276759697, value=value
1 row(s) in 0.0790 seconds
```

Run the `drop` command to delete a table, which should be disabled first before deletion by running the `disable` command:

```
hbase(main):010:0> disable 'test'
hbase(main):011:0> drop 'test'
```

Finally, run the `quit` command to close HBase Shell.

For more HBase Shell commands, please see the [official documentation](#).

3. Using HBase with APIs

[Download and install Maven](#) first and then configure its environment variables. If you are using IDE, please configure Maven-related items in your IDE.

Creating Maven project

Enter the directory of the Maven project, such as `D://mavenWorkplace`, and create the project by running the following commands:

```
mvn archetype:generate -DgroupId=$yourgroupId -DartifactId=$yourartifactID
-DarchetypeArtifactId=maven-archetype-quickstart
```

Here, `$yourgroupId` is your package name, `$yourartifactID` is your project name, and `maven-archetype-quickstart` indicates to create a Maven Java project. Some files need to be downloaded during the project creation, so please keep the network connected.

After successfully creating the project, you will see a folder named `$yourartifactID` in the `D://mavenWorkplace` directory. The files included in the folder have the following structure:

```
simple
  ---pom.xml           Core configuration, under the project root directory
  ---src
    ---main
      ---java          Java source code directory
    ---resources       Java configuration file directory
  ---test
    ---java            Test source code directory
    ---resources       Test configuration directory
```

Among the files above, pay extra attention to the `pom.xml` file and the Java folder under the main directory. The `pom.xml` file is primarily used to create dependencies and package configurations; the Java folder is used to store your source code.

Adding Hadoop dependencies and sample code

First, add the Maven dependencies to the `pom.xml` file:

```
<dependencies>
  <dependency>
    <groupId>org.apache.hbase</groupId>
    <artifactId>hbase-client</artifactId>
    <version>1.2.4</version>
  </dependency>
</dependencies>
```

Then, add the packaging and compiling plugins to the `pom.xml` file:

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <configuration>
        <source>1.8</source>
        <target>1.8</target>
        <encoding>utf-8</encoding>
      </configuration>
    </plugin>
    <plugin>
      <artifactId>maven-assembly-plugin</artifactId>
      <configuration>
        <descriptorRefs>
          <descriptorRef>jar-with-dependencies</descriptorRef>
        </descriptorRefs>
      </configuration>
      <executions>
        <execution>
          <id>make-assembly</id>
          <phase>package</phase>
          <goals>
            <goal>single</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
```

```
</plugins>
</build>
```

Before adding the sample code, you need to get the ZooKeeper address of the HBase cluster. Log in to any master or core node in EMR, go to the `/usr/local/service/hbase/conf` directory, and view the `hbase.zookeeper.quorum` configuration in the `hbase-site.xml` file for ZooKeeper's IP address `$quorum` and the `hbase.zookeeper.property.clientPort` configuration for the port number `$clientPort`.

Then, add the sample code by creating a Java Class named `PutExample.java` in the `main>java` folder and adding the following code to it:

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.*;
import org.apache.hadoop.hbase.client.*;
import org.apache.hadoop.hbase.util.Bytes;
import org.apache.hadoop.hbase.io.compress.Compression.Algorithm;
import java.io.IOException;
/**
 * Created by tencent on 2018/6/30.
 */
public class PutExample {
    public static void main(String[] args) throws IOException {
        Configuration conf = HBaseConfiguration.create();
        conf.set("hbase.zookeeper.quorum", "$quorum");
        conf.set("hbase.zookeeper.property.clientPort", "$clientPort");
        conf.set("zookeeper.znode.parent", "$znodePath");
        Connection connection = ConnectionFactory.createConnection(conf);
        Admin admin = connection.getAdmin();
        HTableDescriptor table = new HTableDescriptor(TableName.valueOf("test1"));
        table.addFamily(new HColumnDescriptor("cf").setCompressionType(Algorithm.NONE));
        System.out.print("Creating table. ");
        if (admin.tableExists(table.getTableName())) {
            admin.disableTable(table.getTableName());
            admin.deleteTable(table.getTableName());
        }
        admin.createTable(table);
        Table table1 = connection.getTable(TableName.valueOf("test1"));
        Put put1 = new Put(Bytes.toBytes("row1"));
        put1.addColumn(Bytes.toBytes("cf"), Bytes.toBytes("a"),
            Bytes.toBytes("value1"));
        table1.put(put1);
        Put put2 = new Put(Bytes.toBytes("row2"));
        put2.addColumn(Bytes.toBytes("cf"), Bytes.toBytes("b"),
            Bytes.toBytes("value2"));
        table1.put(put2);
        Put put3 = new Put(Bytes.toBytes("row3"));
```

```
put3.addColumn(Bytes.toBytes("cf"), Bytes.toBytes("c"),
Bytes.toBytes("value3"));
table1.put(put3);
System.out.println(" Done.");
}
}
```

Compiling code and packaging it for upload

Use the local command prompt to enter the project directory and run the following command to compile and package the project:

```
mvn package
```

"Build success" indicates that package is successfully created. You can see the generated .jar package in the target folder under the project directory.

Upload the package file to the EMR cluster with the scp or sftp tool. **Be sure to include the dependencies in the .jar package to be uploaded.** Run the following command in local command line mode:

```
scp $localfile root@public IP address:$remotefolder
```

Here, `$localfile` is the path and the name of your local file; `root` is the CVM instance username. You can look up the public IP address in the node information in the EMR or CVM console. `$remotefolder` is the path where you want to store the file in the CVM instance. After the upload is completed, you can check whether the file is in the corresponding folder on the EMR command line.

4. Running Demo

Log in to a master node of the EMR cluster and switch to the Hadoop user. Run the following command to run the demo:

```
[hadoop@10 hadoop]$ java -jar $package.jar
```

If the console outputs "Done", all operations are completed. You can switch to HBase Shell and run the `list` command to see whether the HBase table is successfully created with the API, and if yes, you can run the `scan` command to see the detailed content of the table.

```
[hadoop@10hbase]$ bin/hbase shell
hbase(main):002:0> list 'test1'
```

```
TABLE
Test1
1 row(s) in 0.0030 seconds
=> ["test1"]
hbase(main):006:0> scan 'test1'
ROW COLUMN+CELL
row1 column=cf:a, timestamp=1530276759697, value=value1
row2 column=cf:b, timestamp=1530276777806, value=value2
row3 column=cf:c, timestamp=1530276792839, value=value3
3 row(s) in 0.2110 seconds
```

For more information on API usage, please see the [official documentation](#).

Using Hbase with Thrift

Last updated : 2021-08-12 10:25:16

Apache Thrift is a software framework used for scalable cross-language services development. It allows you to define data types and service interfaces in a Thrift File through interface definition language (IDL). The Thrift compiler generates your Thrift File into source code which is to be used to build different clients and servers that communicate seamlessly across programming languages.

The Apache Thrift software framework, for scalable cross-language services development, combines a software stack with a code generation engine to build services that work efficiently and seamlessly between C++, Java, Go, Python, PHP, Ruby, Erlang, Perl, Haskell, C#, Cocoa, JavaScript, Node.js, Smalltalk, and OCaml languages.

Thrift server is a Hive-compatible interface for HBase used to support multi-language APIs. The HBase Thrift interface allows other languages to access HBase over Thrift by connecting to a Thrift server that interfaces with the Java client. This section will describe how to connect HBase with Python and Thrift.

1. Prerequisites

- Confirm that you have activated Tencent Cloud and created an EMR cluster. When creating the EMR cluster, select the HBase component on the software configuration page.

2. Using HBase with Python API

HBase on EMR is integrated with Thrift by default, and the Thrift server is started on the Master1 node (the node with a public IP).

Log in to any node (preferably a master one) in the EMR cluster. For information on how to log in to EMR, please see [Logging in to Linux Instance Using Standard Login Method](#). Here, you can use WebShell to log in. Click **Login** on the right of the desired CVM instance to go to the login page. The default username is `root`, and the password is the one you set when creating the EMR cluster. Once your credentials are validated, you can enter the command line interface.

Run the following commands to switch to the Hadoop user and go to the Hbase installation folder:

```
[root@172 ~]# su hadoop
[hadoop@172 root]$ cd /usr/local/service/hbase/
[hadoop@172 hbase]$
```

View the IP address and port number of Thrift in HBase's configuration file:

```
[hadoop@172 hbase]$ vim conf/hbase-site.xml
<property>
<name>hbase.master.hostname</name>
<value>$thriftIP</value>
</property>
<property>
<name>hbase.regionserver.thrift.port</name>
<value>$port</value>
</property>
```

Here, `$port` is the port number of the Thrift server.

By default, HBase is connected with Thrift for EMR clusters. So you don't need to install and configure Thrift. Run the following command to check whether the Thrift server has been started:

```
[hadoop@172 hbase]$ jps
4711 ThriftServer
```

The message above indicates that the Thrift server is already running in the background. At this time, you can operate HBase directly with Python.

Load balancing

An HA cluster has two master nodes, and both nodes start Thrift server by default. If load balancing is required, the client code needs a custom policy to distribute requests to the two Thrift servers which are completely independent of each other with no communication.

Preparing data

Use HBase Shell to create an HBase table. If you have already created one through HBase on EMR, skip this step:

```
[hadoop@172 hbase]$ hbase shell
hbase(main):001:0> create 'thrift_test', 'cf'
hbase(main):005:0> list
thrift_test
1 row(s) in 0.2270 seconds
hbase(main):001:0> quit
```

Viewing table in HBase with Python

First, you need to install the Python dependencies. Switch to the root user with the password that is same as the one for EMR cluster, install the python-pip tool first and then dependencies:


```
[hadoop@172 hbase]$ su
Password: *****
[root@172 hbase]# yum install python-pip
[root@172 hbase]# pip install hbase-thrift
```

Then, switch back to the Hadoop user, create a Python file `Hbase_client.py`, and add the following code to it:

```
#!/usr/bin/env python
#coding=utf-8
from thrift.transport import TSocket,TTransport
from thrift.protocol import TBinaryProtocol
from hbase import Hbase
socket = TSocket.TSocket('$thriftIP ', $port)
socket.setTimeout(5000)
transport = TTransport.TBufferedTransport(socket)
protocol = TBinaryProtocol.TBinaryProtocol(transport)
client = Hbase.Client(protocol)
transport.open()
print client.getTableNames()
```

Note :

Here, `$thriftIP` is the IP address of the master node on the private network, and `$port` is the port number of ThriftService.

Save and run the file, and the table in HBase will be shown in the console:

```
[hadoop@172 hbase]$ python Hbase_client.py
['thrift_test']
```

Creating HBase table with Python

Create a Python file `Create_table.py` and add the following code to it:

```
#!/usr/bin/env python
#coding=utf-8
from thrift import Thrift
from thrift.transport import TSocket,TTransport
from thrift.protocol import TBinaryProtocol
from hbase import Hbase
from hbase.ttypes import ColumnDescriptor,Mutation,BatchMutation,TRegionInfo
from hbase.ttypes import IOError,AlreadyExists
```

```
socket = TSocket.TSocket('$thriftIP ', $port)
socket.setTimeout(5000)
transport = TTransport.TBufferedTransport(socket)
protocol = TBinaryProtocol.TBinaryProtocol(transport)
client = Hbase.Client(protocol)
transport.open()
new_table = ColumnDescriptor(name = 'cf:', maxVersions = 1)
client.createTable('thrift_test_1', [new_table])
tables = client.getTableNames()
socket.close()
print tables
```

The program will add a new table `thrift_test_1` in HBase and output all existing tables:

```
[hadoop@172 hbase]$ python Create_table.py
['thrift_test', 'thrift_test_1']
```

Inserting data into HBase table with Python

Create a Python file `Insert.py` and add the following code to it:

```
#!/usr/bin/env python
#coding=utf-8
from thrift import Thrift
from thrift.transport import TSocket, TTransport
from thrift.protocol import TBinaryProtocol
from hbase import Hbase
from hbase.ttypes import ColumnDescriptor, Mutation, BatchMutation, TRegionInfo
from hbase.ttypes import IOError, AlreadyExists
socket = TSocket.TSocket('$thriftIP ', $port)
socket.setTimeout(5000)
transport = TTransport.TBufferedTransport(socket)
protocol = TBinaryProtocol.TBinaryProtocol(transport)
client = Hbase.Client(protocol)
transport.open()
mutation1 = [Mutation(column = "cf:a", value = "value1")]
client.mutateRow('thrift_test_1', "row1", mutation1)
mutation2 = [Mutation(column = "cf:b", value = "value2")]
client.mutateRow('thrift_test_1', "row1", mutation2)
mutation1 = [Mutation(column = "cf:a", value = "value3")]
client.mutateRow('thrift_test_1', "row2", mutation1)
mutation2 = [Mutation(column = "cf:b", value = "value4")]
client.mutateRow('thrift_test_1', "row2", mutation2)
socket.close()
```

The program will add two rows of data to the `thrift_test_1` table in HBase, each with two data entries, which can be viewed in HBase Shell:

```
hbase(main):005:0> scan 'thrift_test_1'
ROW COLUMN+CELL
row1 column=cf:a, timestamp=1530697238581, value=value1
row1 column=cf:b, timestamp=1530697238587, value=value2
row2 column=cf:a, timestamp=1530704886969, value=value3
row2 column=cf:b, timestamp=1530704886975, value=value4
2 row(s) in 0.0190 seconds
```

Viewing data in HBase table with Python

You can view the data by row or scan the entire dataset. Create a Python file `Scan_table.py` and add the following code to it:

```
#!/usr/bin/env python
#coding=utf-8
from thrift import Thrift
from thrift.transport import TSocket, TTransport
from thrift.protocol import TBinaryProtocol
from hbase import Hbase
from hbase.ttypes import ColumnDescriptor, Mutation, BatchMutation, TRegionInfo
from hbase.ttypes import IOError, AlreadyExists
socket = TSocket.TSocket('$thriftIP ', $port)
socket.setTimeout(5000)
transport = TTransport.TBufferedTransport(socket)
protocol = TBinaryProtocol.TBinaryProtocol(transport)
client = Hbase.Client(protocol)
transport.open()
result1 = client.getRow("thrift_test_1", "row1")
print result1
for r in result1:
    print 'the rowname is ', r.row
    print 'the frist value is ', r.columns.get('cf:a').value
    print 'the second value is ', r.columns.get('cf:b').value
scanId = client.scannerOpen('thrift_test_1', "", ["cf"])
result2 = client.scannerGetList(scanId, 10)
print result2
client.scannerClose(scanId)
socket.close()
```

Use `GetRow` to get the data of one row, or use `scannerGetList` to get all the data in the table. The output of the program is as follows:

```
[hadoop@172 hbase]$ python Scan_table.py
[TRowResult(columns={'cf:a': TCell(timestamp=1530697238581, value='value1'), 'cf:
b': TCell(timestamp=1530697238587, value='value2')}, row='row1')]
the rowname is row1
the frist value is value1
the second value is value2
[TRowResult(columns={'cf:a': TCell(timestamp=1530697238581, value='value1'), 'cf:
b': TCell(timestamp=1530697238587, value='value2')}, row='row1'), TRowResult(colu
mns={'cf:a': TCell(timestamp=1530704886969, value='value3'), 'cf:b': TCell(timest
amp=1530704886975, value='value4')}, row='row2')]
```

As you can see, the data of the first row and the data of the entire table are outputted separately.

Deleting data from HBase with Python

Create a Python file `Delete_row.py` and add the following code to it:

```
#!/usr/bin/env python
#coding=utf-8
from thrift import Thrift
from thrift.transport import TSocket, TTransport
from thrift.protocol import TBinaryProtocol
from hbase import Hbase
from hbase.ttypes import *
socket = TSocket.TSocket('$thriftIP ', $port)
socket.setTimeout(5000)
transport = TTransport.TBufferedTransport(socket)
protocol = TBinaryProtocol.TBinaryProtocol(transport)
client = Hbase.Client(protocol)
transport.open()
client.deleteAllRow("thrift_test_1", "row2")
socket.close()
```

The program will delete the second row of data in the test table. After the program is executed, you can view the table in HBase Shell:

```
[hadoop@172 hbase]$ python Delete_row.py
[hadoop@172 hbase]$ hbase shell
hbase(main):004:0> scan 'thrift_test_1'
ROW COLUMN+CELL
row1 column=cf:a, timestamp=1530697238581, value=value1
row1 column=cf:b, timestamp=1530697238587, value=value2
1 row(s) in 0.2050 seconds
```

At this time, the table contains only the data in the first row

For more information on Thrift operations, please see [How to Use Thrift](#).

Spark on Hbase

Last updated : 2020-03-25 10:55:24

For more information on Spark on HBase operations, please see [Spark-HBase Connector](#) on GitHub.

MapReduce on Hbase

Last updated : 2020-10-09 11:00:49

For more information on MapReduce on Hbase operations (e.g., read and write), please see [Use Cases](#).

Phoenix on Hbase Development Guide

Phoenix Client Usage

Last updated : 2022-11-25 16:06:47

Apache Phoenix is a massively parallel relational database engine supporting OLTP for Hadoop with Apache HBase as its backing store. Phoenix compiles simple queries in just milliseconds and queries millions of rows of data in seconds. By default, the Phoenix client is integrated in HBase-enabled EMR clusters.

1. Start the client.

Switch to the `hadoop` user, go to the `/usr/local/service/hbase/phoenix-client/bin` directory, and use Phoenix's Python command line tool:

```
./sqlline.py
```

If the execution succeeds, the following result will be displayed:

```
[hadoop@i72 /usr/local/service/hbase/phoenix-client/bin]$ ./sqlline.py
Setting property: [incremental, false]
Setting property: [isolation, TRANSACTION_READ_COMMITTED]
issuing: !connect -p driver org.apache.phoenix.jdbc.PhoenixDriver -p user "none" -p password "none" "jdbc:phoenix:"
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/local/service/hbase/phoenix-client/phoenix-client-hbase-2.4-5.1.2.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/local/service/hadoop/share/hadoop/common/lib/slf4j-log4j12-1.7.25.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]
Connecting to jdbc:phoenix:
Connected to: Phoenix (version 5.1)
Driver: PhoenixEmbeddedDriver (version 5.1)
Autocommit status: true
Transaction isolation: TRANSACTION_READ_COMMITTED
sqlline version 1.9.0
0: jdbc:phoenix:> █
```

2. Phoenix supports SQL queries. The following are some common operations:

- Create a table

```
0: jdbc:phoenix:> CREATE TABLE IF NOT EXISTS TEST (
host char(50) not null,
txn_count bigint
CONSTRAINT pk PRIMARY KEY (host)
);
```

- Insert data

```
0: jdbc:phoenix:> UPSERT INTO TEST (host, txn_count) VALUES ('192.168.1.1', 1);
0: jdbc:phoenix:> UPSERT INTO TEST (host, txn_count) VALUES ('192.168.1.2', 2);
```


- Query data

```
0: jdbc:phoenix:>SELECT * FROM TEST;
```

- Delete a data table

```
0: jdbc:phoenix:>DROP TABLE IF EXISTS TEST;
```

For more operations and instructions, see [Grammar](#).

Phoenix JDBC Usage

Last updated : 2022-11-25 16:06:47

Adding Maven dependencies

```
<dependency>
<groupId>org.apache.phoenix</groupId>
<artifactId>phoenix-core</artifactId>
<version>${phoenix.version}</version>
</dependency>
```

Here, `phoenix.version` should be consistent with the Phoenix version in the cluster.

Creating a JDBC object

```
Class.forName("org.apache.phoenix.jdbc.PhoenixDriver");
// Connect to the database
connection = DriverManager.getConnection("jdbc:phoenix:10.0.0.3:2181,10.0.0.5:2181,10.0.0.8:2181");
```

Running a query

```
private static void instertPhoenix(Connection connection) throws Exception{
String sql="upsert into album_subscribe_log(id,album_id,user_id,op_time,sub_flag,
is_optimize,type_parent_id,type_id,host_id,is_pay,user_type,identtity_typ) "
+" values(?,?,?,?,?,?,?,?,?,?,?,?,?)";
PreparedStatement ps=connection.prepareStatement(sql);
ps.setLong(0,1);
ps.setLong(1,3);
ps.setLong(2,1);
ps.setString(3,"2017-09-05 14:00:00");
ps.setInt(4,1);
ps.setString(5,"1");
ps.setInt(6,3);
ps.setInt(7,5);
ps.setInt(8,6);
ps.setInt(9,7);
```

```
ps.setString(10, "1");  
ps.setString(11, "1");  
ps.setString(12, "1");  
ps.executeUpdate();  
ps.close();  
connection.commit();  
}
```

Phoenix Best Practices

Last updated : 2021-08-12 10:25:16

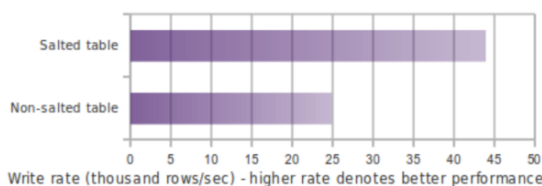
Using Phoenix Salted Table

HBase sequential write may suffer from region server hotspotting if your row key is monotonically increasing. Phoenix provides a way to transparently salt the row key with a salting byte for a particular table. You need to specify this in table creation time by specifying a table property "SALT_BUCKETS" with a value from 1 to 256. Like this:

```
0: jdbc:phoenix:>CREATE TABLE table (a_key VARCHAR PRIMARY KEY, a_col VARCHAR) SALT_BUCKETS = 20;
```

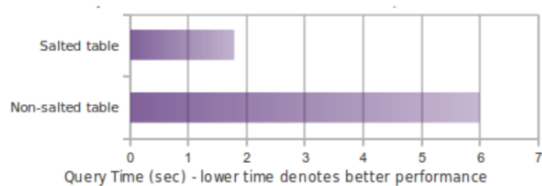
Salting the row key provides a way to mitigate the problem caused by HBase sequential write. With salted tables, you don't need to be knowledgeable about the row key design in HBase. Below is Phoenix's official performance comparison of read and write performance between salted and non-salted tables:

Following chart shows write performance with and without the use of Salting which splits table in 4 regions running on 4 region server cluster (Note: For optimal performance, number of salt buckets should match number of region servers).



Following chart shows in-memory query performance for 10M row table where host='NA' filter matches 3.3M rows

```
select count(1) from t where host='NA'
```



For more information on salting performance or directions, please see the [community documentation for salted tables in Phoenix](#).

Phoenix Secondary Indexing

Secondary indexes are an orthogonal way to access data from its primary access path. In HBase, you have a single index that is lexicographically sorted on the primary row key. Access to records in any way other than through the primary row requires scanning over potentially all the rows in the table to test them against your filter. With secondary

indexing, the columns or expressions you index form an alternate row key to allow point lookups and range scans along this new axis.

Configuring Secondary Indexing in Phoenix

Phoenix on EMR supports Phoenix's secondary indexing. If you need to use non-transactional, variable indexing, just follow the steps below to configure it. Go to the component management page in the EMR console, click **HBase**, select **Configuration > Configuration Management**, and add three configuration items in hbase-site.xml:

```
hbase.regionserver.wal.codec , hbase.region.server.rpc.scheduler.factory.class , and  
hbase.rpc.controllerfactory.class . The detailed configuration is as follows:
```

```
<property>  
<name>hbase.regionserver.wal.codec</name>  
<value>org.apache.hadoop.hbase.regionserver.wal.IndexedWALEditCodec</value>  
</property>  
<property>  
<name>hbase.region.server.rpc.scheduler.factory.class</name>  
<value>org.apache.hadoop.hbase.ipc.PhoenixRpcSchedulerFactory</value>  
<description>Factory to create the Phoenix RPC Scheduler that uses separate queue  
s for index and metadata updates</description>  
</property>  
<property>  
<name>hbase.rpc.controllerfactory.class</name>  
<value>org.apache.hadoop.hbase.ipc.controller.ServerRpcControllerFactory</value>  
<description>Factory to create the Phoenix RPC Scheduler that uses separate queue  
s for index and metadata updates</description>  
</property>
```

Using Secondary Indexing in Phoenix

To create a secondary index, run the following command:

```
0: jdbc:phoenix:>CREATE INDEX my_index ON my_table (v1) INCLUDE (v2);
```

For more information on secondary indexing, please see the [community documentation of Phoenix secondary indexing](#).

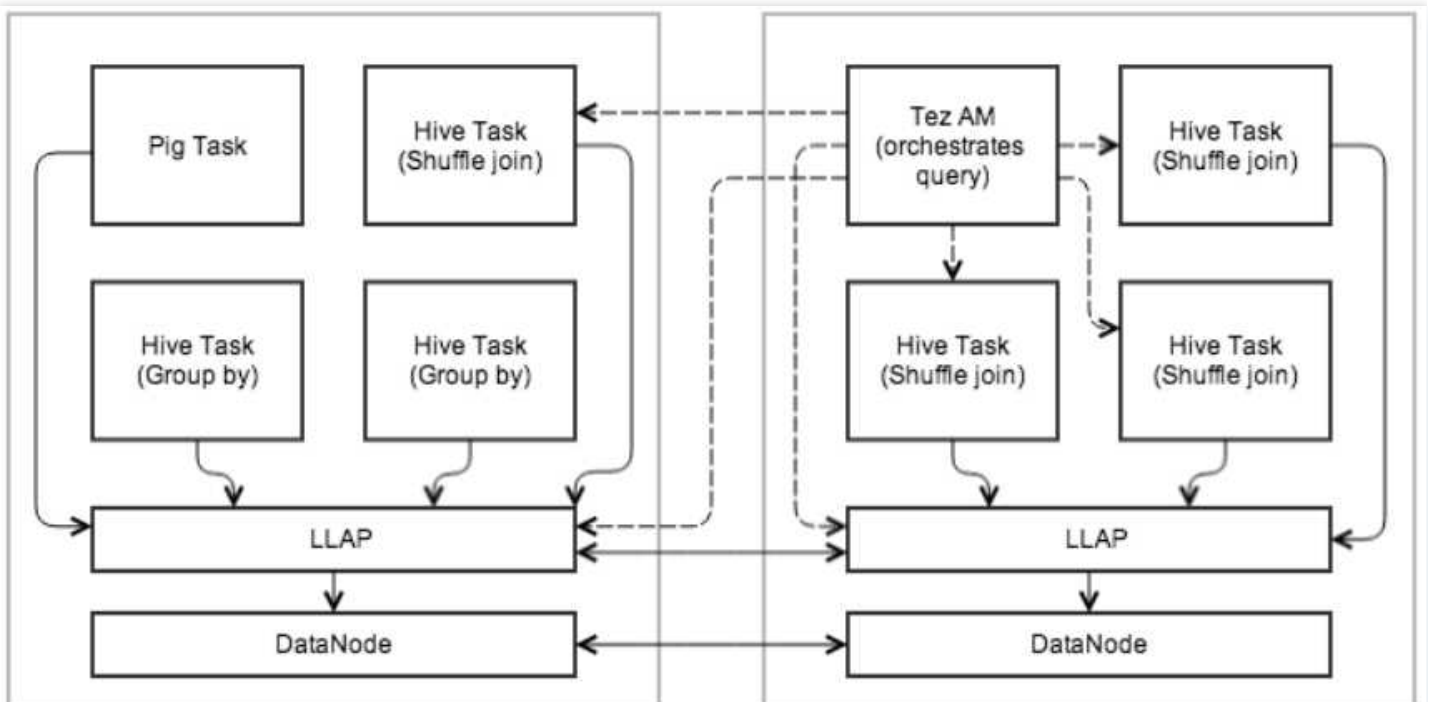
Hive Development Guide

Hive's Support for LLAP

Last updated : 2022-04-18 15:38:03

Apache Hive introduced Live Long and Process (LLAP) on v2.0 and made great optimization on v2.1, which shows that Hive has evolved to memory computing. Currently, as tested by Hortonworks, LLAP working together with Tez is 25 times faster than Hive v1.x. LLAP provides a hybrid execution model consisting of a long-lived daemon which replaces direct interactions with the HDFS DataNode, and a tightly integrated DAG-based framework. For example, features such as caching, pre-fetching, certain query processing, and access control are moved into the daemon. Small/Short queries are largely processed by this daemon directly, while any heavy lifting will be performed in standard YARN containers.

Hive LLAP Structure Diagram



- Persistent daemon

To facilitate caching and JIT optimization, and to eliminate most of the startup costs, a daemon runs on the worker nodes in the cluster. It handles I/O, caching, and query fragment execution.

- Execution engine

LLAP works within existing, process-based Hive execution to preserve the scalability and versatility of Hive. It does

not replace the existing execution model but rather enhances it.

- Query fragment execution

LLAP nodes execute "query fragments" such as filters, projections, data transformations, partial aggregates, sorting, bucketing, and hash `joins/semi-joins`.

- I/O

The daemon off-loads I/O and transformation from compressed format to separate threads. The data is passed in to the execution as it becomes ready, so the previous batches can be processed while the next ones are being prepared. The data is passed in to the execution in a simple RLE-encoded columnar format that is ready for vectorized processing; this is also the caching format, with the intent to minimize replication between I/O, cache, and execution.

- Caching

The daemon caches metadata for input files, as well as the data. The metadata and index information can be cached even for data that is not currently cached.

Installing Hive LLAP

1. Enter the EMR [purchase page](#).
2. Select `EMR-V2.3.0` as the product version.
3. Select **TEZ 0.9.2** in the **Optional Component** list, and Hive LLAP will be automatically installed in the `/usr/local/service/slider` directory.

Using Hive LLAP

- Modify `/usr/local/service/slider/conf/slider-client.xml` by adding the following configuration items:

```
<property>
<name>hadoop.registry.zk.quorum</name>
<value>zk_ip:zk_port</value>
</property>
```

The IP in the `value` is the master node IP for a non-high-availability cluster. For a high-availability cluster, run the following command to view the ZooKeeper component IP:

```
cat /usr/local/service/hadoop/etc/hadoop/hdfs-site.xml | grep 2181
```

- Modify `hive-site.xml` (delivered in the console).

```
<property>
<name>hive.execution.engine</name>
<value>tez</value>
</property>
<property>
<name>hive.llap.execution.mode</name>
<value>all</value>
</property>
<property>
<name>hive.execution.mode</name>
<value>llap</value>
</property>
<property>
<name>hive.llap.daemon.service.hosts</name>
<value>@llap_service</value>
</property>
<property>
<name>hive.zookeeper.quorum</name>
<value>zk_ip:zk_port</value>
</property>
<property>
<name>hive.llap.daemon.memory.per.instance.mb</name>
<value>4000</value>
</property>
<property>
<name>hive.llap.daemon.num.executors</name>
<value>2</value>
</property>
<property>
<name>hive.server2.tez.default.queues</name>
<value>root.default</value>
</property>
<property>
<name>hive.server2.tez.initialize.default.sessions</name>
<value>true</value>
</property>
<property>
<name>hive.server2.tez.sessions.per.default.queue</name>
<value>2</value>
</property>
```

Note :

Here, you need to enter the actual ZooKeeper address and port in the `hive.zookeeper.quorum` configuration item.

- Restart all Hive services.
- Generate the LLAP start file and command.

```
hive --service llap --name llap_service --instances 2 --size 2g --loglevel INFO
--cache 1g --executors 2 --iothreads 5 --slider-am-container-mb 1024 --args " -
XX:+UseG1GC -XX:+ResizeTLAB -XX:+UseNUMA -XX:-ResizePLAB"
```

Here, LLAP execution and configuration files will be generated in the current directory. Run the `run.sh` script as prompted as shown below:

```
[hadoop@10 ~]$ hive --service llap --name llap_service --instances 4 --size 2g --loglevel INFO --cache 1g --executors 10 --iothread
: 10 --slider-am-container-mb 1024 --args " -XX:+UseG1GC -XX:+ResizeTLAB -XX:+UseNUMA -XX:-ResizePLAB"
which: no hbase in (/usr/local/jdk/bin:/usr/local/service/hadoop/bin:/usr/local/service/hive/bin:/usr/local/service/hbase/bin:/usr/l
ocal/service/spark/bin:/usr/local/service/storm/bin:/usr/local/service/sqoop/bin:/usr/local/service/kylin/bin:/usr/local/service/all
uxio/bin:/usr/local/service/Flink/bin:/data/Impala/bin:/usr/local/service/oozie/bin:/usr/local/service/presto/bin:/usr/local/service
/slider/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/root/bin)
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/local/service/hive/lib/log4j-slf4j-impl-2.6.2.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/local/service/hive/spark/jars/slf4j-log4j12-1.7.16.jar!/org/slf4j/impl/StaticLoggerBinde
r.class]
SLF4J: Found binding in [jar:file:/usr/local/service/tez/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/local/service/hadoop/share/hadoop/common/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/Static
LoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]
INFO cli.LlapServiceDriver: LLAP service driver invoked with arguments=-hiveconf
INFO conf.HiveConf: Found configuration file file:/usr/local/service/hive/conf/hive-site.xml
WARN conf.HiveConf: HiveConf of name hive.metastore.db.encoding does not exist
WARN conf.HiveConf: HiveConf of name hive.hwi.listen.host does not exist
WARN conf.HiveConf: HiveConf of name hive.hwi.listen.port does not exist
WARN cli.LlapServiceDriver: Ignoring unknown llap server parameter: [hive.aux.jars.path]
INFO cli.LlapServiceDriver: Memory settings: container memory: 2.00GB executor memory: -1B cache memory: 1.00GB
WARN cli.LlapServiceDriver: Java versions might not match : JAVA_HOME=[/usr/local/jdk],process jre=[/usr/local/jdk/jre]
INFO cli.LlapServiceDriver: Using [/usr/local/jdk] for JAVA_HOME
INFO lzo.GPLNativeCodeLoader: Loaded native gpl library from the embedded binaries
INFO lzo.LzoCodec: Successfully loaded & initialized native-lzo library [hadoop-lzo rev 5dbddb8c9fb544e58b4e0b9664b9d1b66657faf5]
INFO cli.LlapServiceDriver: Error getting an optional config ssl-server.xml; ignoring: null
WARN cli.LlapServiceDriver: hadoop-metrics2-llapdaemon.properties cannot be found. Looking for hadoop-metrics2.properties
INFO cli.LlapServiceDriver: copied hadoop metrics2 properties file from file:/usr/local/service/hadoop/etc/hadoop/hadoop-metrics2.pr
operties
INFO cli.LlapServiceDriver: Adding the following aux jars from the environment and configs: []
WARN conf.HiveConf: HiveConf of name hive.metastore.db.encoding does not exist
WARN conf.HiveConf: HiveConf of name hive.hwi.listen.host does not exist
WARN conf.HiveConf: HiveConf of name hive.hwi.listen.port does not exist
WARN conf.HiveConf: HiveConf of name hive.metastore.db.encoding does not exist
WARN conf.HiveConf: HiveConf of name hive.hwi.listen.host does not exist
WARN conf.HiveConf: HiveConf of name hive.hwi.listen.port does not exist
INFO metastore.HiveMetaStore: 0: Opening raw store with implementation class:org.apache.hadoop.hive.metastore.ObjectStore
INFO metastore.ObjectStore: ObjectStore, initialize called
WARN conf.HiveConf: HiveConf of name hive.metastore.db.encoding does not exist
WARN conf.HiveConf: HiveConf of name hive.hwi.listen.host does not exist
WARN conf.HiveConf: HiveConf of name hive.hwi.listen.port does not exist
INFO metastore.ObjectStore: Setting MetaStore object pin classes with hive.metastore.cache.pinobjtypes="Table,StorageDescriptor,SeRD
eInfo,Partition,Database,Type,Fieldschema,Order"
INFO metastore.MetaStoreDirectSql: Using direct SQL, underlying DB is MYSQL
INFO metastore.ObjectStore: Initialized ObjectStore
INFO metastore.HiveMetaStore: Added admin role in metastore
INFO metastore.HiveMetaStore: Added public role in metastore
INFO metastore.HiveMetaStore: No user is added in admin role, since config is empty
INFO metastore.HiveMetaStore: 0: get_all_functions
INFO HiveMetaStore.audit: ugi=hadoop ip=unknown-ip-addr cmd=get_all_functions
INFO cli.LlapServiceDriver: LLAP service driver finished
02:31:20 Running after LlapServiceDriver
02:31:20 Prepared the files
02:31:26 Packaged the files
02:31:27 Prepared llap-slider-27May2020/run.sh for running LLAP on slider
[hadoop@10 ~]$
```

```
llap-slider-27May2020/run.sh
```

Run the `run.sh` file above, and a resident application will be submitted in YARN. Wait for several minutes, and you can see the application in `yarn-ui`.

Note :

LLAP initialization will take several minutes. Wait until it ends before performing data operations.

- Verify LLAP.

Enter Hive CLI:

```
create table t1(id int, name string);
insert into t1 values ('1','test1'),('2', 'test2');
select id, count(1) from t1 group by id;
```

Expected result:

```
hive> insert into t1 values('1','test1'),('2', 'test2');
Query ID = hadoop_20200527114442_1d4a9647-b10f-435f-9b78-5fbdacd78f1f
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1590545548684_0014)

-----
VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
-----
Map 1 .....  llap      SUCCEEDED    1         1           0         0         0         0
-----
VERTICES: 01/01 [=====>>>] 100% ELAPSED TIME: 1.16 s
-----
Loading data to table default.t1
OK
Time taken: 2.732 seconds
hive> select id,count(1) from t1 group by id;
Query ID = hadoop_20200527115110_847ad7c3-634b-44b4-af5f-f1c0e9e8b43b
Total jobs = 1
Launching Job 1 out of 1
Tez session was closed. Reopening...
Session re-established.
Status: Running (Executing on YARN cluster with App id application_1590545548684_0015)

-----
VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
-----
Map 1 .....  llap      SUCCEEDED    2         2           0         0         0         0
Reducer 2 ..... llap      SUCCEEDED    1         1           0         0         0         0
-----
VERTICES: 02/02 [=====>>>] 100% ELAPSED TIME: 0.77 s
-----
OK
1      5
2      5
Time taken: 6.925 seconds, Fetched: 2 row(s)
hive> █
```

Basic Hive Operations

Last updated : 2021-01-22 16:17:15

Hive is a data warehouse architecture built on the Hadoop file system (HDFS). It provides many features for data warehouse management, including data ETL (extraction, transformation, and loading), data storage management, and query and analysis for large data sets. In addition, it also defines the SQL-like language Hive-SQL, which allows users to perform operations similar to those in SQL. Hive-SQL maps structured data files to a single database table and provides simple SQL query capabilities. It also enables developers to easily use Mapper and Reducer operations to convert SQL statements into MapReduce jobs. This is a powerful support for the MapReduce framework, as the learning cost is low, and MapReduce statistics can be quickly collected with ease through SQL-like statements, eliminating the need to develop dedicated MapReduce applications. All these advantages make Hive very suitable for statistical analysis of data warehouses.

Hive uses Hadoop's HDFS as its file storage system, making it easy to expand the storage capacity and increase the computing power. It can achieve the same horizontal stability as that of Hadoop, so that a cluster of thousands of servers can be built conveniently. In general, Hive is designed for mining massive amounts of data, but its real-time performance is relatively poor.

Hive's internal and external tables:

- **Internal table:** an internal table actually maps a file in HDFS to a table, and the data warehouse of Hive generates the corresponding directory for it. The default warehouse path in EMR is `usr/hive/warehouse/${tablename}`. **This path is in HDFS**, where `${tablename}` is the name of the table you created. Simply load the files matching the table definition into this directory, and they can be queried with Hive-SQL.
- **External table:** an external table in Hive is very similar to a common table, except that the data is stored somewhere else but not in the directories of the table. The advantage of this mechanism is that when you delete the external table, the data it points to will not be deleted; instead, only the metadata corresponding to it will be deleted. By contrast, if you delete an internal table, all the data in it, including the metadata, will be deleted.

This document demonstrates how to create tables in an EMR cluster and query them through Hive.

1. Development Preparations

- You need to create a bucket in COS for this job. For more information, please see [Creating Buckets](#).
- Confirm that you have activated Tencent Cloud and created an EMR cluster. When creating the EMR cluster, select the Hive component on the software configuration page and "Enable COS" on the basic configuration page and

enter your own `SecretId` and `SecretKey` below, which can be viewed on the [API Key Management](#) page. If there is no key yet, click **Create Key** to create one.

- Hive and its dependencies are installed in the EMR cluster directory `/usr/local/service/`.

2. Data Preparations

Log in to any node (preferably a master one) in the EMR cluster first. For more information on how to log in to EMR, please see [Logging in to Linux Instance Using Standard Login Method](#). You can choose to log in with WebShell. Click "Log in" on the right of the desired CVM instance to enter the login page. The default username is `root`, and the password is the one you set when creating the EMR cluster. Once the correct information is entered, you can enter the command line interface.

Run the following command on the EMR command line interface to switch to the Hadoop user, then go to the Hive folder:

```
[root@172 ~]# su hadoop
[hadoop@172 ~]# cd /usr/local/service/hive
```

Create a bash script file named `gen_data.sh` and add the following code to it:

```
#!/bin/bash
MAXROW=1000000 # Specify the number of data rows to be generated
for((i = 0; i < $MAXROW; i++))
do
echo $RANDOM, \"$RANDOM\"
done
```

And run it as follows:

```
[hadoop@172 hive]$ chmod +x script name
[hadoop@172 hive]$ ./gen_data.sh > hive_test.data
```

This script file will generate 1,000,000 random number pairs and save them to the `hive_test.data` file.

- Run the following command to upload the generated test data to HDFS first. Here, `$hdfspath` is the path of your file on HDFS.

```
[hadoop@172 hive]$ hdfs dfs -put ./hive_test.data /$hdfspath
```

- You can also use the data stored in COS. First, upload the data to COS. If the data is in the local file system, use the COS console. If it is in your EMR cluster, run the following command. Here, `$bucketname` is the name of the COS bucket you created.

```
[hadoop@172 hive]$ hdfs dfs -put ./hive_test.data cosn://$bucketname/
```

3. Basic Hive Operations

Connecting to Hive

Log in to a master node of the EMR cluster, switch to the Hadoop user, go to the Hive directory, and connect to Hive:

```
[hadoop@172 hive]$ su hadoop
[hadoop@172 hive]$ cd /usr/local/service/hive/bin
[hadoop@172 bin]$ hive
```

You can use the `-h` parameter to get basic information on Hive commands. You can also use the Beeline mode to connect to a database. To do so, you also need to log in to a master node in EMR, switch to the Hadoop user, and go to the Hive directory. In the `conf/hive-site.xml` configuration file, get the connection port `$port` and host address `$host` of Hive server 2:

```
<property>
<name>hive.server2.thrift.bind.host</name>
<value>$host</value>
</property>
<property>
<name>hive.server2.thrift.port</name>
<value>$port</value>
</property>
```

In the bin directory, run the following statement to connect to Hive:

```
[hadoop@172 hive]$ cd bin
[hadoop@172 bin]$ ./beeline -u "jdbc:hive2:// $host: $port " -n hadoop -p hadoop
```

Creating Hive table

You run the same Hive-SQL statements in Hive mode and the Beeline mode. The following example shows how to run Hive-SQL statements in Hive mode. Run the following command in Hive mode to view the database:

```
hive> show databases;
OK
default
Time taken: 0.26 seconds, Fetched: 1 row(s)
```

Run the `create` command to create a database:

```
hive> create database test; # Create a database named test
OK
Time taken: 0.176 seconds
```

Run the `use` command to go to the test database you just created:

```
hive> use test;
OK
Time taken: 0.176 seconds
```

Run the `create` command to create an internal table named `hive_test` in the test database:

```
hive> create table hive_test (a int, b string)
hive> ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';
# Create a data table named hive_test and specify the column separator as ','
OK
Time taken: 0.204 seconds
```

There is only one command. If you do not enter the semicolon ";", Hive-SQL can put one command in multiple lines for input. Finally, you can run the following command to see whether the table has been created successfully:

```
hive> show tables;
OK
hive_test
Time taken: 0.176 seconds, Fetched: 1 row(s)
```

Importing data into table

For data stored in HDFS, run the following command to import it into the table:

```
hive> load data inpath "$hdfspath/hive_test.data" into table hive_test;
```

Here, `$hdfspath` is the path of your file in HDFS. After the import is completed, the source data file in the import path in HDFS will be deleted.

For data stored in COS, run the following command to import it into the table:

```
hive> load data inpath "cosn://$bucketname/hive_test.data" into table hive_test;
```

Here, `$bucketname` is your bucket name plus the path of the data in the bucket.

Also, after the import is completed, the source data file in the import path in COS will be deleted. You can also import data stored locally in the EMR cluster into Hive by running the following command:

```
hive>load data local inpath "$localpath/hive_test.data" into table hive_test;
```

Here, `$localpath` is the path of the data locally stored in the EMR cluster. After the import is completed, the source data will be deleted.

Running query

Run the `select` command to perform a query and count the number of data rows in a table:

```
hive> select count(*) from hive_test;
Query ID = hadoop_20170316142922_967b5f0e-1f89-4464-bfa3-b6ed53273fc2
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
set hive.exec.reducers.bytes.per.reducer=
In order to limit the maximum number of reducers:
set hive.exec.reducers.max=
In order to set a constant number of reducers:
set mapreduce.job.reduces=
Starting Job = job_1489458311206_9869, Tracking URL =
http://10.0.1.125:5004/proxy/application_1489458311206_9869/
Kill Command = /usr/local/service/hadoop/bin/hadoop job -kill job_1489458311206_9869
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2017-03-16 14:29:29,023 Stage-1 map = 0%, reduce = 0%
2017-03-16 14:29:34,208 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 3.87 sec
2017-03-16 14:29:40,404 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 5.79 sec
MapReduce Total cumulative CPU time: 5 seconds 790 msec
Ended Job = job_1489458311206_9869
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1 Reduce: 1 Cumulative CPU: 5.79 sec
HDFS Read: 40974623 HDFS Write: 107 SUCCESS
Total MapReduce CPU Time Spent: 5 seconds 790 msec
OK
1000000
Time taken: 18.504 seconds, Fetched: 1 row(s)
```

The final output is 1000000.

Run the `select` command to query the first 10 elements in the table:

```
hive> select * from hive_test limit 10;
OK
30847 "31583"
14887 "32053"
19741 "16590"
8104 "20321"
29030 "32724"
27274 "5231"
10028 "22594"
924 "32569"
10603 "27927"
4018 "30518"
Time taken: 2.133 seconds, Fetched: 10 row(s)
```

Deleting Hive table

Run the `drop` command to delete a Hive table:

```
hive> drop table hive_test;
Moved: 'hdfs://HDFS/usr/hive/warehouse/hive_test' to trash at: hdfs://HDFS/user/h
adoop/.Trash/Current
OK
Time taken: 2.327 seconds
```

For more information on Hive operations, please see the [official documentation](#).

Connecting to Hive Through Java

Last updated : 2020-10-10 14:57:18

Apache Hadoop Hive is integrated with the Thrift service. Thrift, created by Facebook, is an interface definition language and binary communication protocol used for defining and creating services for numerous languages. HiveServer2 is based on Thrift, allowing many languages such as Java and Python to call Hive's APIs. Additionally, JDBC Driver for Apache Hadoop Hive enables Java application to interact with the Hive database.

This section describes how to connect to HiveServer2 through Java code.

1. Prerequisites

- Confirm that you have activated Tencent Cloud and created an EMR cluster. When creating the EMR cluster, select the Hive component on the software configuration page.
- Hive and its dependencies are installed in the EMR cluster directory `/usr/local/service/`.

2. Using Maven to Create a Project

Viewing parameters

First, log in to any node (preferably a master one) in the EMR cluster. For more information about how to log in to EMR, please see [Logging in to a Linux Instance](#). Here, you can use WebShell to log in. Click *Login* on the right of the desired CVM instance and then enter the login page. The default username is root, and the password is the one you set when you created the EMR cluster. Once your credentials have been validated, you can access the command-line interface.

Run the following command in EMR command-line interface to switch to the Hadoop user, then go to the Hive installation folder:

```
[root@172 ~]# su hadoop
[hadoop@172 root]$ cd /usr/local/service/hive/
[hadoop@172 hive]$
```

View the parameters that are required for the program:

```
[hadoop@172 hive]$ vim conf/hive-site.xml

<property>
<name>hive.server2.thrift.bind.host</name>
```

```
<value>$hs2host</value>
</property>
<property>
<name>hive.server2.thrift.port</name>
<value>$hs2port</value>
</property>
```

Here, `$hs2host` is the hostID of your HiveServer2, and `$hs2port` is the port number of your HiveServer2.

Creating a Maven project

Maven is recommended for project management, as it can help you manage project dependencies with ease.

Specifically, it can get .jar packages through the configuration of the pom.xml file, eliminating your need to add them manually.

Download and install Maven in your local system first and then configure its environment variables. If you are using the IDE, please set the Maven-related configuration items in the IDE.

In the local shell environment, enter the directory where you want to create the Maven project, such as

```
D://mavenWorkplace , and enter the following command to create it:
```

```
mvn archetype:generate -DgroupId=$yourgroupId -DartifactId=$yourartifactID -DarchetypeArtifactId=maven-archetype-quickstart
```

Here, `$yourgroupId` is your package name, `$yourartifactID` is your project name, and `maven-archetype-quickstart` indicates to create a Maven Java project. Some files need to be downloaded for creating the project, so please keep the network connected.

After successfully creating the project, you will see a folder named `$yourartifactID` in the `D://mavenWorkplace` directory. The files included in the folder have the following structure:

```
simple
---pom.xml      Core configuration, under the project root directory
---src
---main
---java        Java source code directory
---resources   Java configuration file directory
---test
---java        Test source code directory
---resources   Test configuration directory
```

Among the files above, pay extra attention to the pom.xml file and the Java folder under the main directory. The pom.xml file is primarily used to create dependencies and package configurations; the Java folder is used to store your source codes.

First, add the Maven dependencies to pom.xml:

```
<dependencies>
<dependency>
<groupId>org.apache.hive</groupId>
<artifactId>hive-jdbc</artifactId>
<version>2.1.1</version>
</dependency>
<dependency>
<groupId>org.apache.hadoop</groupId>
<artifactId>hadoop-common</artifactId>
<version>2.7.3</version>
</dependency>
</dependencies>
```

Then, add the packaging and compiling plugins to pom.xml:

```
<build>
<plugins>
<plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-compiler-plugin</artifactId>
<configuration>
<source>1.8</source>
<target>1.8</target>
<encoding>utf-8</encoding>
</configuration>
</plugin>
<plugin>
<artifactId>maven-assembly-plugin</artifactId>
<configuration>
<descriptorRefs>
<descriptorRef>jar-with-dependencies</descriptorRef>
</descriptorRefs>
</configuration>
<executions>
<execution>
<id>make-assembly</id>
<phase>package</phase>
<goals>
<goal>single</goal>
</goals>
</execution>
</executions>
</plugin>
```

```
</plugins>
</build>
```

Right-click in `src>main>Java` and create a Java Class. Enter the Class name (e.g., `HiveTest.java` here) and add the sample code to the Class:

```
import java.sql.*;

/**
 * Created by tencent on 2018/7/6.
 */
public class HiveTest {
    private static String driverName =
        "org.apache.hive.jdbc.HiveDriver";

    public static void main(String[] args)
        throws SQLException {
        try{
            Class.forName(driverName);
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
            System.exit(1);
        }

        Connection con = DriverManager.getConnection(
            "jdbc:hive2://$hs2host:$hs2port/default", "hadoop", "");
        Statement stmt = con.createStatement();
        String tableName = "HiveTestByJava";
        stmt.execute("drop table if exists " + tableName);
        stmt.execute("create table " + tableName +
            " (key int, value string)");
        System.out.println("Create table success!");
        // show tables
        String sql = "show tables '" + tableName + "'";
        System.out.println("Running: " + sql);
        ResultSet res = stmt.executeQuery(sql);
        if (res.next()) {
            System.out.println(res.getString(1));
        }

        // describe table
        sql = "describe " + tableName;
        System.out.println("Running: " + sql);
        res = stmt.executeQuery(sql);
        while (res.next()) {
            System.out.println(res.getString(1) + "\t" + res.getString(2));
        }
    }
}
```

```
}

sql = "insert into " + tableName + " values (42,\"hello\"), (48,\"world\")";
stmt.execute(sql);

sql = "select * from " + tableName;
System.out.println("Running: " + sql);
res = stmt.executeQuery(sql);
while (res.next()) {
System.out.println(String.valueOf(res.getInt(1)) + "\t"
+ res.getString(2));
}

sql = "select count(1) from " + tableName;
System.out.println("Running: " + sql);
res = stmt.executeQuery(sql);
while (res.next()) {
System.out.println(res.getString(1));
}
}
}
```

Note :

The parameters `$hs2host` and `$hs2port` in the program should be replaced with the values of the hostID and port number of the HiveServer2 you queried.

The entire program will connect to the HiveServer2 before a table called `HiveTestByJava` is created in the default database. Then, insert two elements into the table and output the content of the entire table.

If your Maven is configured correctly and its dependencies are successfully imported, the project will be compiled directly. Enter the project directory in the local shell, and run the following command to package the entire project:

```
mvn package
```

Some files may need to be downloaded during the running process. "Build success" indicates that package is successfully created. You can see the generated `.jar` package in the target folder under the project directory.

3. Uploading and Running the Program

First, use the `scp` or `sftp` tool to upload the compressed `.jar` package to the EMR cluster. Run the following command in your local shell:

```
scp $localfile root@public IP address:/usr/local/service/hive
```

Here, \$localfile is the path and the name of your local file; root is the CVM instance username. You can look up the public IP address in the node information in the EMR or CVM Console. Upload the compressed .jar package to the /usr/local/service/hive directory in the EMR cluster. After the upload is completed, you can check whether the file is in the corresponding folder by using EMR command lines. **Be sure to upload a .jar package that contains the dependencies.**

Log in to the EMR cluster, switch to the Hadoop user, and go to the /usr/local/service/hive directory. Then you can run the program:

```
[hadoop@172 hive]$ yarn jar $package.jar HiveTest
```

Here, \$package.jar is the path and the name of your .jar package; HiveTest is the name of the previously created Java Class. The result is as follows:

```
Create table success!
Running: show tables 'HiveTestByJava'
hivetestbyjava
Running: describe HiveTestByJava
key int
value string
Running: select * from HiveTestByJava
42 hello
48 world
Running: select count(1) from HiveTestByJava
2
```

Connecting to Hive Through Python

Last updated : 2020-10-10 15:01:07

Apache Hadoop Hive is integrated with the Thrift service. Thrift, created by Facebook, is an interface definition language and binary communication protocol used for defining and creating services for numerous languages. HiveServer2 is based on Thrift, allowing many languages such as Java and Python to call Hive's APIs.

This section describes how to connect to HiveServer2 through Python.

1. Preparations for Development

- Confirm that you have activated Tencent Cloud and created an EMR cluster. When creating the EMR cluster, select the Hive component on the software configuration page.
- Hive and its dependencies are installed in the EMR cluster directory `/usr/local/service/`.

2. Viewing Parameters

First, log in to any node (preferably a master one) in the EMR cluster. For more information about how to log in to EMR, please see [Logging in to a Linux Instance](#). Here, you can use WebShell to log in. Click *Login* on the right of the desired CVM instance and then enter the login page. The default username is root, and the password is the one you set when you created the EMR cluster. Once your credentials have been validated, you can access the command-line interface.

Run the following command in EMR command-line interface to switch to the Hadoop user, then go to the Hive installation folder:

```
[root@172 ~]# su hadoop
[hadoop@172 root]$ cd /usr/local/service/hive/
[hadoop@172 hive]$
```

View the parameters that are required for the program:

```
[hadoop@172 hive]$ vim conf/hive-site.xml

<property>
<name>hive.server2.thrift.bind.host</name>
<value>$hs2host</value>
</property>
<property>
```

```
<name>hive.server2.thrift.port</name>
<value>${hs2port}</value>
</property>
```

Here, `hs2host` is the hostID of your HiveServer2, and `hs2port` is the port number of your HiveServer2.

3. Using Python to Manipulate Hive

To manipulate Hive with a Python program, you need to install pip:

```
[hadoop@172 hive]$ su
[root@172 hive]# pip install pyhs2
```

Switch back to the Hadoop user after the installation is completed. Create a Python file named `hivetest.py` in the

`/usr/local/service/hive/` directory and add the following code:

```
#!/usr/bin/env python

import pyhs2
import sys

default_encoding = 'utf-8'

conn = pyhs2.connect(host='${hs2host}',
port=${hs2port},
authMechanism='PLAIN',
user='hadoop',
password='',
database='default',)

tablename = 'HiveByPython'
cur = conn.cursor()
print 'show the databases: '
print cur.getDatabases()

print "\n"
print 'show the tables in default: '
cur.execute('show tables')
for i in cur.fetch():
print i

cur.execute('drop table if exists ' + tablename)
cur.execute('create table ' + tablename + ' (key int,value string)')
```



```
print "\n"
print 'show the new table: '
cur.execute('show tables ' + "'" + tablename + "'")
for i in cur.fetch():
    print i

print "\n"
print "contents from " + tablename + ":";
cur.execute('insert into ' + tablename + ' values (42,"hello"),(48,"world")')
cur.execute('select * from ' + tablename)
for i in cur.fetch():
    print i
```

⚠ Note :

The parameters \$hs2host and \$hs2port in the program should be replaced with the values of the hostID and port number of the HiveServer2 you queried.

After connecting to HiveServer2, this program outputs all the databases first and then displays the tables in the "default" database. Create a table named "hivebypython", insert two data entries into it, and output them. Run the program:

```
[hadoop@172 hive]$ python hivetest.py
show the databases:
[['default', ''], ['hue_test', ''], ['test', '']]

show the tables in default:
['dd']
['ext_table']
['hive_test']
['hivebypython']

show the new table:
['hivebypython']

contents from HiveByPython:
[42, 'hello']
[48, 'world']
```

Mapping Hbase Tables

Last updated : 2021-07-09 10:45:31

You can use Hive to map HBase tables. By doing so, you can read data in HBase with Hive and run Hive-SQL statements to perform operations such as query and insertion on HBase tables.

Preparations for Development

- Confirm that you have activated Tencent Cloud and created an EMR cluster. When creating the EMR cluster, select the Hive and HBase components on the software configuration page.
- Hive and its dependencies are installed under the EMR cluster directory `/usr/local/service/`

Creating an HBase Table

First, you need to log in to any node (preferably a master one) in the EMR cluster. For more information on how to log in to EMR, please see [Logging in to Linux Instances](#). Here, you can choose to log in with WebShell. Click "Log in" on the right of the desired CVM instance to enter the login page. The default username is `root`, and the password is the one you set when creating the EMR cluster. Once the correct credentials are entered, you can enter the command line interface.

Run the following command in EMR command-line interface to switch to the Hadoop user and go to the HBase folder to enter HBase Shell:

```
[root@172 ~]# su hadoop
[hadoop@172 ~]# cd /usr/local/service/hbase
[hadoop@10hbase]$ bin/hbase shell
```

Create a table in HBase as shown below:

```
hbase (main):001:0> create 'test', 'cf'
hbase (main):003:0> put 'test', 'row1', 'cf:a', 'value1'
hbase (main):004:0> put 'test', 'row1', 'cf:b', 'value2'
hbase (main):005:0> put 'test', 'row1', 'cf:c', 'value3'
```

For more information on HBase operations, please see the Hbase Operation Guide or [official documentation](#).

After the creation is completed, you can use the `list` and `scan` operations to view the newly created table.

```
hbase(main):001:0> list 'test'
TABLE
test
1 row(s) in 0.0030 seconds
=> ["test"]

hbase(main):002:0> scan 'test'
ROW COLUMN+CELL
row1 column=cf:a, timestamp=1530276759697, value=value1
row2 column=cf:b, timestamp=1530276777806, value=value2
row3 column=cf:c, timestamp=1530276792839, value=value3
3 row(s) in 0.2110 seconds
```

Mapping a Hive Table

Switch to the Hive folder and connect to Hive:

```
[hadoop@172 hive]$ cd /usr/local/service/hive/
[hadoop@172 hive]$ bin/hive
```

Next, create a Hive external table and map it to the HBase table created in step 2:

```
hive> CREATE EXTERNAL TABLE hive_test (
> rowkey string,
> a string,
> b string,
> c string
> ) STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler' WITH
> SERDEPROPERTIES("hbase.columns.mapping" = ":key,cf:a,cf:b,cf:c")
> TBLPROPERTIES("hbase.table.name" = "test");
OK
Time taken: 2.086 seconds
```

Now, a mapping from the Hive table to the HBase is created. You can run the following command to view the elements in the Hive table:

```
hive> select * from hive_test;
OK
row1 value1 value2 value3
Time taken: 0.305 seconds, Fetched: 1 row(s)
```

Hive Best Practices

Last updated : 2020-10-10 16:55:21

Execution Engine

Hive on Tencent Cloud Elastic MapReduce (EMR) currently supports three execution engines:

- MR
- TEZ
- Spark

Select TEZ, if needed, when you purchase an EMR cluster. Normally, TEZ is the recommended execution engine due to higher computing efficiency.

Storage

Currently, Tencent Cloud supports the following storage media: local data disk, HDD cloud disk, SSD cloud disk, and COS (most cost-efficient).

Data format

Tencent Cloud supports multiple compression algorithms such as Snappy and LZO. You are recommended to format your files in ORC or Parquet for greater space utilization and computing efficiency in Hive.

How to choose the best query engine?

Currently, EMR supports SQL-based query engine Presto, SparkSQL, and Hive. Presto allows you to query a variety of data sources, while SparkSQL is good for applications that require low-latency. For querying general data warehouse, use Hive + TEZ.

Data security

If you use COS as underlying storage, external tables are recommended to prevent the data from being accidentally deleted; if your data is stored in HDFS, you are recommended to enable the HDFS Recycle Bin to recover the deleted data.

Creating Databases Based on COS

Last updated : 2021-08-12 10:25:16

There are two ways to create a Cloud Object Storage (COS) data warehouse:

Method 1: Creating Whole Database in COS

To do so, run the following statement:

```
create database hivewithcos location 'cosn://huadong/hive';
```

Here, `huadong` is the bucket name, and `hive` is the path, which can be set as needed. After creating the database, you can create a data table. Then, load data into the table, which can be used in exactly the same way as HDFS.

```
create table record(id int, name string) row format delimited fields terminated b  
y ',' stored as textfile;
```

Method 2: Moving Specified Table into COS

First, select or create a database in Hive. Then move the specified table into COS by running the following statement:

```
create table record(id int, name string) row format delimited fields terminated b  
y ',' stored as textfile location 'cosn://huadong/hive/cos';
```

Load data into the table. At this time, the table is stored in COS.

Practices on Loading JSON Data to Hive

Last updated : 2020-12-24 10:32:22

1. Connect to Hive

Log in to a master node of the EMR cluster, switch to the "hadoop" user, go to the Hive directory, and connect to Hive by running the following command:

```
[root@10 ~]# su hadoop
[hadoop@10 root]$ cd /usr/local/service/hive
```

2. Prepare data

Create a data file in JSON format. Compile the following code and save:

```
vim test.data
{"name":"Mary","age":12,"course":[{"name":"math","location":"b208"}, {"name":"english","location":"b702"}],"grade":[99,98,95]}
{"name":"Bob","age":20,"course":[{"name":"music","location":"b108"}, {"name":"history","location":"b711"}],"grade":[91,92,93]}
```

Store the data file in HDFS:

```
hadoop fs -put ./test.data /
```

3. Create a table

Connect to Hive:

```
[hadoop@10 hive]$ hive
```

Create a table based on the mapping:

```
hive> CREATE TABLE test (name string, age int, course array<map<string,string>>,
grade array<int>) ROW FORMAT SERDE 'org.apache.hive.hcatalog.data.JsonSerDe' STORED AS TEXTFILE;
```

4. Import data

```
hive>LOAD DATA INPATH '/test.data' into table test;
```

5. Check whether data import is successful

Query all data:

```
hive> select * from test;
OK
Mary 12 [{"name":"math","location":"b208"}, {"name":"english","location":"b702"}]
[99,98,95]
Bob 20 [{"name":"music","location":"b108"}, {"name":"history","location":"b711"}]
[91,92,93]
Time taken: 0.153 seconds, Fetched: 2 row(s)
```

Query the first score of each record:

```
hive> select grade[0] from test;
OK
99
91
Time taken: 0.374 seconds, Fetched: 2 row(s)
```

Query the name and location of the first course of each record:

```
hive> select course[0]['name'], course[0]['location'] from test;
OK
math b208
music b108
Time taken: 0.162 seconds, Fetched: 2 row(s)
```

Hive Metadata Management

Last updated : 2022-08-04 10:19:39

When you choose to deploy the Hive component, there are two storage methods for Hive metadata: you can store the metadata in a MetaDB instance separately purchased for the cluster (default method) or associate the metadata with EMR-MetaDB or a self-built MySQL database. In the latter case, metadata will be stored in the associated database and will not be deleted when the cluster is terminated.

In the former case, a MetaDB instance is automatically purchased to store Hive metadata, alongside metadata of other components. The MetaDB instance will be terminated too when the cluster is terminated. If you need to retain the metadata, please go to the TencentDB console and save it manually in advance.

Note :

1. Hive metadata is stored together with the metadata of Druid, Superset, Hue, Ranger, Oozie, and Presto.
2. A MetaDB instance needs to be purchased separately for the cluster as the metadatabase.
3. The MetaDB instance will be terminated when the cluster is terminated, i.e., the metadata stored in it will be deleted.

Associating with EMR-MetaDB for Hive Metadata Sharing

When a cluster is created, the system will pull an MetaDB instance available in the cloud as the Hive metadatabase, so you don't need to purchase a MetaDB instance separately, which helps reduce your storage costs. Moreover, Hive metadata will not be deleted when the cluster is terminated.

Note :

1. The ID of the available MetaDB instance is the ID of an existing MetaDB instance in the EMR cluster under the same account.
2. If one or more components among Hue, Ranger, Oozie, Druid and Superset are selected, a MetaDB instance will be automatically purchased to store the metadata of such components other than Hive.
3. To terminate the associated EMR-MetaDB instance, you need to go to the TencentDB console to do so. Hive metadatabases cannot be recovered once terminated.
4. Make sure that the associated EMR-MetaDB instance is in the same network as the newly created cluster.

Add Component



The current cluster does not have a metadatabase. To add the Hue, Ranger, Oozie, Druid, and Superset components, you need to purchase a TencentDB instance to store metadata.

Product Version **EMR-V2.5.0**

- Optional Components
- | | | | |
|---|--|--|--|
| <input checked="" type="checkbox"/> zookeeper-3.6.1 | <input checked="" type="checkbox"/> hadoop-2.8.5 | <input checked="" type="checkbox"/> Knox-1.2.0 | <input type="checkbox"/> zeppelin-0.8.2 |
| <input type="checkbox"/> livy-0.7.0 | <input type="checkbox"/> hbase-1.4.9 | <input checked="" type="checkbox"/> hive-2.3.7 | <input type="checkbox"/> hue-4.6.0 |
| <input type="checkbox"/> oozie-5.1.0 | <input type="checkbox"/> prestosql-332 | <input type="checkbox"/> ranger-1.2.0 | <input type="checkbox"/> spark_hadoop2.8-3.0.... |
| <input type="checkbox"/> sqoop-1.4.7 | <input type="checkbox"/> storm-1.2.3 | <input type="checkbox"/> tez-0.9.2 | <input type="checkbox"/> flume-1.9.0 |
| <input type="checkbox"/> ganglia-3.7.2 | <input type="checkbox"/> impala-2.10.0 | <input type="checkbox"/> alluxio-2.3.0 | <input type="checkbox"/> flink-1.10.0 |
| <input type="checkbox"/> kylin-2.5.2 | <input type="checkbox"/> superset-0.35.2 | <input type="checkbox"/> tensorflowspark-1.4.... | |

Hive Metadatabase: Default Associate EMR-MetaDB Associate self-built MYSQL

Instance ID: Select data ▼

Confirm Cancel

Elastic MapReduce

1. Availability Zone and Software Configuration 2. Hardware Configuration 3. Basic Configuration

Billing Mode: Pay-as-you-go

Region: South China East China North China South China Western US Asia Pacific South Asia

AZ: Guangzhou Shanghai Beijing Singapore Silicon Valley Seoul Mumbai

AZ: Guangzhou Zone 3 Guangzhou Zone 4 Guangzhou Zone 6

Cloud products in different regions are not interconnected over private networks. The region and availability zone cannot be changed after purchase. You are advised to select a region and availability zone close to your business data to reduce access delay and increase download speed.

Cluster Type: HADOOP DRUID CLICKHOUSE

Product Version: EMR-V2.5.0 [Product Version Introduction](#)

Required Components: zookeeper 3.6.1 knox 1.2.0 hadoop 2.8.5

Optional Components: prestosql 332 ranger 1.2.0 spark_hadoop2.8 3.0.0 sqoop 1.4.7 storm 1.2.3 superset 0.35.2
tensorflowspark 1.4.4 tez 0.9.2 zeppelin 0.8.2 oozie 5.1.0 livy 0.7.0 kylin 2.5.2 flink 1.10.0 flume 1.9.0



Associating with Self-built MySQL Database for Hive Metadata Sharing

You can associate with your local self-built MySQL database to store Hive metadata, so you don't need to purchase a MetaDB instance separately, which helps reduce your storage costs. Please enter the address beginning with "jdbc:mysql://", username, and login password of the database correctly and make sure that the database can be connected with the cluster.

Note :

1. Make sure that the self-built database is in the same network as the EMR cluster.
2. Enter the username and password of the database correctly.
3. If one or more components among Hue, Ranger, Oozie, Druid and Superset are selected, a MetaDB instance will be automatically purchased to store the metadata of such components other than Hive.
4. Make sure that the Hive metadata version in the custom database is not below the Hive version in the new cluster.

Cluster Type HADOOP DRUID CLICKHOUSE

Product Version EMR-V2.5.0 [Product Version Introduction](#)

Required Components zookeeper 3.6.1 knox 1.2.0 hadoop 2.8.5

Optional Components prestoql 332 ranger 1.2.0 spark_hadoop2.8 3.0.0 sqoop 1.4.7 storm 1.2.3 superset 0.35.2
tensorflowspark 1.4.4 tez 0.9.2 zeppelin 0.8.2 oozie 5.1.0 livy 0.7.0 kylin 2.5.2 flink 1.10.0 flume 1.9.0
ganglia 3.7.2 hbase 1.4.9 hive 2.3.7 hue 4.6.0 impala 2.10.0 kudu 1.12.0 alluxio 2.3.0

Hive Metadatabase Default Associate EMR-MetaDB Associate self-built MYSQL

Please ensure that the self-built database contains Hive metadatabase tables and is on the same network as the EMR cluster; correctly enter the database username and password. Otherwise, the cluster cannot be created successfully.

Database Link
 Please enter the jdbc link of the database, e.g., : jdbc:mysql://10.10.10.3306/dbname

Database Username

Database Password

Advanced Settings

Next Step: Hardware Configuration

Add Component



The current cluster does not have a metadatabase. To add the Hue, Ranger, Oozie, Druid, and Superset components, you need to purchase a TencentDB instance to store metadata.

Product Version EMR-V2.5.0

Optional Components zookeeper-3.6.1 hadoop-2.8.5 Knox-1.2.0 zeppelin-0.8.2
 livy-0.7.0 hbase-1.4.9 hive-2.3.7 hue-4.6.0
 oozie-5.1.0 prestoql-332 ranger-1.2.0 spark_hadoop2.8-3.0....
 sqoop-1.4.7 storm-1.2.3 tez-0.9.2 flume-1.9.0
 ganglia-3.7.2 impala-2.10.0 alluxio-2.3.0 flink-1.10.0
 kylin-2.5.2 superset-0.35.2 tensorflowspark-1.4....

Hive Metadatabase Default Associate EMR-MetaDB Associate self-built MYSQL

Please ensure that the self-built database contains Hive metadatabase tables and is on the same network as the EMR

Please ensure that the self-built database contains five metadatabase tables and is on the same network as the EMR cluster; correctly enter the database username and password. Otherwise, the cluster cannot be created successfully.

Database Link

Please enter the jdbc link of the database, e.g., : jdbc:mysql://10.10.10.10:3306/dbname

Database Username

Database Password

Confirm

Cancel

Presto Development Guide

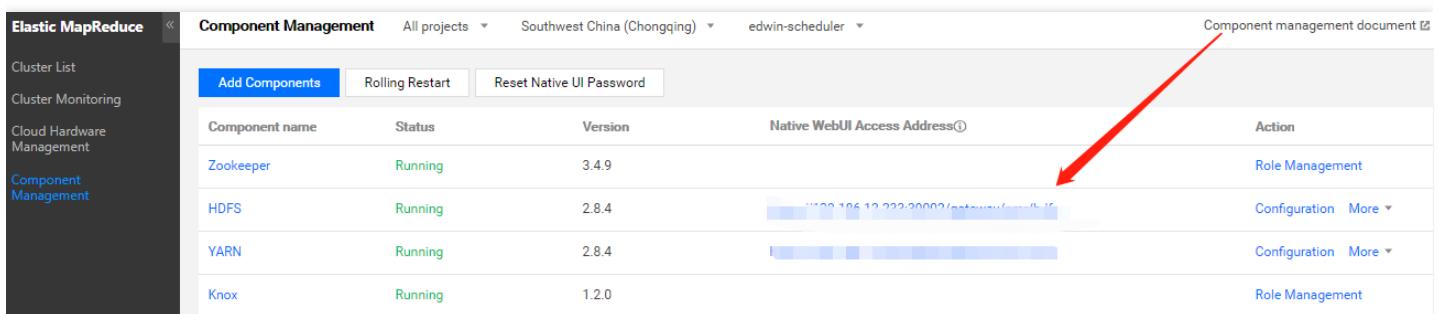
Presto Web UI

Last updated : 2020-07-10 17:28:47

Presto is an open source distributed SQL query engine for running interactive analytic queries against data sources of all sizes ranging from gigabytes to petabytes. Presto was designed and written from the ground up for interactive analysis and approaches the speed of commercial data warehouses.

Presto allows querying data where it lives, including Hive, Cassandra, relational databases or even proprietary data stores. A single Presto query can combine data from multiple sources, allowing for analysis across your entire organization.

EMR integrates the native web UI of Presto, which you can see in the EMR Console. Log in to the [EMR Console](#), click a cluster instance ID to enter the instance management page. Click **Cluster Service** on the left sidebar to view the entry to the **WebUI Address** page and then click the entry to Presto. The username for login is `root`, and the password is the one set when the cluster was created as shown below:



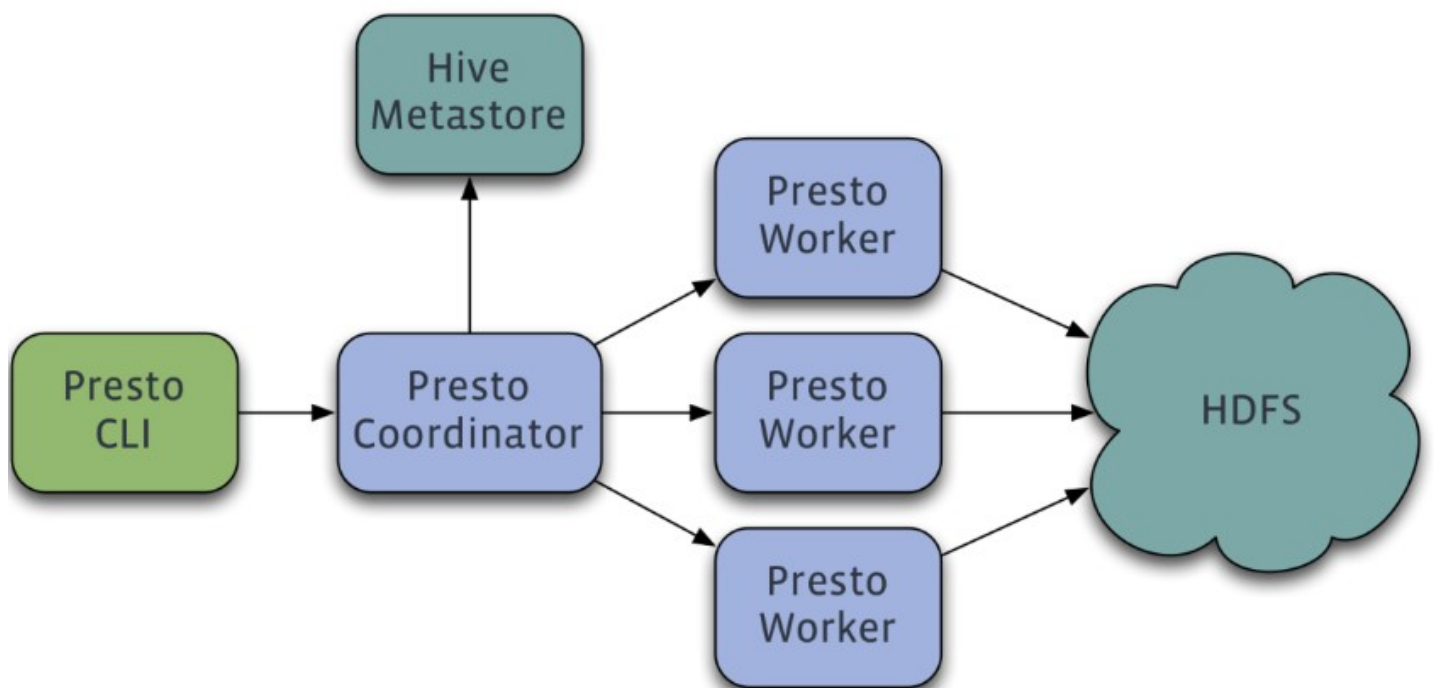
Component name	Status	Version	Native WebUI Access Address①	Action
Zookeeper	Running	3.4.9		Role Management
HDFS	Running	2.8.4	100.106.12.222:20002/presto/	Configuration More ▾
YARN	Running	2.8.4		Configuration More ▾
Knox	Running	1.2.0		Role Management

It is required to authenticate the accesses. The username is `root`, and the default password is the one entered when the cluster was created. To change the password, click **Reset Native UI Password**.

Connector

Last updated : 2021-07-09 11:05:00

Presto is a distributed SQL query engine developed by Facebook that is designed to perform high-speed, real-time data analysis for interactive analytical queries of gigabytes to petabytes of data. It supports standard ANSI SQL, including complex queries, aggregations, joins, and window functions. Implemented by Java, it supports various data sources such as Hive, HBase, relational databases, and even proprietary data stores. Below is its architecture diagram:



Presto adopts a distributed system running on multiple servers in a master-slave architecture, which consists of one master node (coordinator) and multiple slave nodes (worker). The client (Presto CLI) is responsible for submitting a query to the Coordinator node which is responsible for parsing the SQL statement, generating the query execution plan, and managing the worker nodes. A worker node is responsible for actually executing the query job.

Presto on EMR is pre-configured with connectors such as Hive, MySQL, and Kafka. In this section, the Hive connector is used as an example to illustrate how Presto reads the data in a Hive table for query. The EMR cluster is configured with environment variables related to presto-client, so you can switch directly to the Hadoop user and use Presto CLI.

1. Preparations for Development

- Confirm that you have activated Tencent Cloud and created an EMR cluster. When creating the EMR cluster, select the Presto component on the software configuration page.
- Relevant software programs such as Presto are installed in the `/usr/local/service/` directory of the CVM instance for the EMR cluster.

2. Using a Connector to Manipulate Hive

First, you need to log in to any node (preferably a master one) in the EMR cluster. For more information on how to log in to EMR, please see [Logging in to Linux Instances](#). Here, you can choose to log in with WebShell. Click "Log in" on the right of the desired CVM instance to enter the login page. The default username is `root`, and the password is the one you set when creating the EMR cluster. Once the correct credentials are entered, you can enter the command line interface.

Run the following command in EMR command-line interface to switch to the Hadoop user and go to the Presto folder:

```
[root@172 ~]# su hadoop
[hadoop@172 ~]# cd /usr/local/service/presto
```

View the value of uri in the `etc/config.properties` configuration file:

```
[hadoop@172 presto]$ vim etc/config.properties
http-server.http.port=$port
discovery.uri=http://$host:$port
```

Here, `$host` is your host address, and `$port` is your port number. Then, switch to the presto-client folder and connect to Hive through Presto:

```
[hadoop@172 presto]# cd /usr/local/service/presto-client
[hadoop@172 presto-client]$ ./presto --server $host:$port --catalog hive --schema
default
```

Here, `--catalog` indicates the type of database to be manipulated, and `--schema` the database name (the default database "default" is entered here). For more information on the parameters, run the `presto -h` command or see the [official documentation](#).

After successful execution, you can enter the Presto interface and go directly to the specified database. You can use Hive-SQL to view the table in the Hive database:

```
presto:default> show tables;
Table
-----
hive_from_cos
```



```
test
(2 rows)
Query 20180702_140619_00006_c4qzg, FINISHED, 2 nodes
Splits: 2 total, 2 done (100.00%)
0:00 [3 rows, 86B] [17 rows/s, 508B/s]
```

The table `hive_from_cos` is the one created in the Hive Development Guide.

For more information on Presto usage, please see the [official documentation](#).

Analyzing Data in COS

Last updated : 2021-07-08 10:43:44

This document describes more options on using COS-Hive connectors, where the data is from direct-insert, COS, and LZO.

1. Development Preparations

- This task requires access to COS, so you need to [create a bucket](#) in COS first.
- Create an EMR cluster. When creating the EMR cluster, you need to select the Presto component on the software configuration page and enable access to COS on the basic configuration page.
- Relevant software programs such as Presto are installed in the `/usr/local/service/` directory of the CVM instance for the EMR cluster.

2. Data Preparations

First, log in to any node (preferably a master one) in the EMR cluster. For information about how to log in to EMR, see [Logging in to Linux Instance Using Standard Login Method](#). Here, you can use WebShell to log in. Click **Login** on the right of the desired CVM instance to go to the login page. The default username is `root`, and the password is the one you set when creating the EMR cluster. Once your credentials are validated, you can enter the command line interface.

Run the following commands to switch to the Hadoop user and go to the Hive installation folder:

```
[root@172 ~]# su hadoop
[hadoop@172 ~]# cd /usr/local/service/hive
```

Create a file named `cos.txt` and add the following data to it:

```
5,cos_patrick
6,cos_stone
```

Upload the file to COS by running the following HDFS command. Here, `$bucketname` is the name plus path of the bucket you created.

```
[hadoop@172 hive]# hdfs dfs -put cosn://$bucketname/
```

Create a file named `lzo.txt` and add the following data to it:

```
10,lzo_pop
11,lzo_tim
```

Compress it into an .lzo file:

```
[hadoop@172 hive]$ lzop -v lzo.txt
compressing hive_test.data into lzo.txt.lzo
```

Note :

To create an .lzo compressed file, you need to install lzo and lzop first by running this command: `yum -y install lzo lzop`.

3. Creating a Hive Table and Using Presto for Query

A script file is used here to create a Hive database and table. Create a script file named `presto_on_cos_test.sql` and add the following program to it:

```
create database if not exists test;
use test;
create external table if not exists presto_on_cos (id int,name string) ROW FORMAT
DELIMITED FIELDS TERMINATED BY ',';
insert into presto_on_cos values (12,'hello'),(13,'world');
load data inpath "cosn://$bucketname/cos.txt" into table presto_on_cos;
load data local inpath "$yourpath/lzo.txt.lzo" into table presto_on_cos;
```

Here, `$bucketname` is the name plus path of your COS bucket, and `$yourpath` is the path where the `lzo.txt.lzo` file is stored.

The script file creates a database named "test" first and then a table named "presto_on_cos" in the created database. Here, the data is loaded in three steps: first, insert the data directly; then, insert the data stored in COS; and finally, insert the data stored in the .lzo package.

It is recommended to use an external table for Hive testing as shown in this example so as to avoid deleting important data by mistake. Run this script with Hive CLI:

```
[hadoop@172 hive]$ hive -f "presto_on_cos_test.sql"
```

Once the execution is completed, you can enter Presto to view the data in the table. Use the same method as described in the previous section to enter Presto, but you should modify the `schema` parameter.

```
[hadoop@172 presto-client]$ ./presto --server $host:$port --catalog hive --schema test
```

Query the Hive table you just created:

```
presto:test> select * from presto_on_cos ;
id | name
----+-----
 5 | cos_patrick
 6 | cos_stone
10 | lzo_pop
11 | lzo_tim
12 | hello
13 | world
(6 rows)
Query 20180702_150000_00011_c4qzg, FINISHED, 3 nodes
Splits: 4 total, 4 done (100.00%)
0:03 [6 rows, 127B] [1 rows/s, 37B/s]
```

For more information about Presto usage, see the [official documentation](#).

Sqoop Development Guide

Import/Export of Relational Database and HDFS

Last updated : 2020-11-23 17:13:14

Sqoop is an open-source tool for transferring data between Hadoop and traditional databases such as MySQL and PostgreSQL. It can import data in a relational database (e.g., MySQL, Oracle, and Postgres) into Hadoop's HDFS, and vice versa. One of the highlights of Sqoop is its ability to import data in a relational database into HDFS through Hadoop MapReduce.

This document describes how to use Sqoop on EMR to import and export data between MySQL and HDFS.

1. Prerequisites

- You have signed up for a Tencent Cloud account and created an EMR cluster. When creating the EMR cluster, select the Sqoop component on the software configuration page.
- Relevant software programs such as Sqoop are installed in the `/usr/local/service/` directory of the CVM instance for the EMR cluster.

2. Creating a MySQL Table

Connect to the created MySQL database first. Enter the EMR Console and copy the instance ID of the destination cluster, i.e., the cluster name. Then, enter the TencentDB for MySQL Console, use Ctrl+F to find the MySQL database corresponding to the cluster, and view the private IP address of the database (`$mysqlIP`).

Log in to any node (preferably a master one) in the EMR cluster. For more information on how to log in to EMR, please see [Logging in to Linux Instances](#). Here, you can choose to log in with WebShell. Click "Log in" on the right of the desired CVM instance to enter the login page. The default username is `root`, and the password is the one you set when creating the EMR cluster. Once the correct credentials are entered, you can enter the command line interface.

Run the following command in EMR command-line interface to switch to the Hadoop user and go to the Sqoop folder:

```
[root@172 ~]# su hadoop
[hadoop@172 ~]# cd /usr/local/service/sqoop
```

Connect to the MySQL database:

```
[hadoop@172 sqoop]$ mysql -h $mysqlIP -p
Enter password:
```

The password is the one you set when you created the EMR cluster.

After connecting to the MySQL database, enter the test database and create a table. You can also choose the destination database:

```
mysql> use test;
Database changed

mysql> create table sqoop_test(id int not null primary key auto_increment, title
varchar(64), time timestamp, content varchar(255));
Query ok , 0 rows affected(0.00 sec)
```

This command creates a MySQL table with a primary key of ID and three columns of title, time, and content. Insert data entries into the table as follows:

```
mysql> insert into sqoop_test values(null, 'first', now(), 'hdfs');
Query ok, 1 row affected(0.00 sec)

mysql> insert into sqoop_test values(null, 'second', now(), 'mr');
Query ok, 1 row affected (0.00 sec)

mysql> insert into sqoop_test values(null, 'third', now(), 'yarn');
Query ok, 1 row affected(0.00 sec)
```

Run the following command to view the data in the table:

```
Mysql> select * from sqoop_test;
+----+-----+-----+-----+
| id | title | time | content |
+----+-----+-----+-----+
| 1 | first | 2018-07-03 15:29:37 | hdfs |
| 2 | second | 2018-07-03 15:30:57 | mr |
| 3 | third | 2018-07-03 15:31:07 | yarn |
+----+-----+-----+-----+
3 rows in set (0.00 sec)
```

Exit the MySQL database:

```
Mysql> exit;
```

3. Importing MySQL Data into HDFS

Run the `sqoop-import` command to import the data from the `sqoop_test` table created in the previous step into HDFS:

```
[hadoop@172 sqoop]$ bin/sqoop-import --connect jdbc:mysql://$mysqlIP/test --username root
-P --table sqoop_test --target-dir /sqoop
```

Here, `--connect` is used to connect to the MySQL database, `test` can be replaced with your database name, `-P` indicates that a password is required, `--table` is the name of the database you want to export, `--target-dir` is the path in HDFS to import into. **Make sure that the `/sqoop` folder does not exist before the command is run; otherwise, a failure will occur.**

After you press Enter, you will be asked for a password, which is the one set when you created the EMR cluster.

After successful execution, you can view the imported data in the corresponding path in HDFS:

```
[hadoop@172 sqoop]$ hadoop fs -cat /sqoop/*
1, first, 2018-07-03 15:29:37.0,hdfs
2, second, 2018-07-03 15:30:57.0,mr
3, third, 2018-07-03 15:31:07.0,yarn
```

4. Importing HDFS Data into MySQL

You need to create a table in MySQL first to store the data from HDFS:

```
[hadoop@172 sqoop]$ mysql -h $mysqlIP -p
Enter password:
mysql> use test;
Database changed

mysql> create table sqoop_test_back(id int not null primary key auto_increment, title varchar(64), time timestamp, content varchar(255));
Query ok , 0 rows affected(0.00 sec)
```

Check whether the table is created successfully and then exit MySQL:

```
mysql> show tables;
+-----+
| Tables_in_test |
+-----+
| sqoop_test |
```

```
| sqoop_test_back |
+-----+
2 rows in set (0.00 sec)

mysql> exit;
```

Run the `sqoop-export` command to export the data imported into HDFS in the previous step into MySQL:

```
[hadoop@172 sqoop]$ bin/sqoop-export --connect jdbc:mysql://$mysqlIP/test --usern
ame
root -P --table sqoop_test_back --export-dir /sqoop
```

The parameters are similar to those for `sqoop-import`, except `--export-dir`, which is the path of the data in HDFS. You also need to enter the password after pressing Enter.

After successful execution, you can verify the data in the database `sqoop_test_back`:

```
[hadoop@172 sqoop]$ mysql -h $mysqlIP -p
Enter password:
mysql> use test;
Database changed

mysql> select * from sqoop_test_back;
+----+-----+-----+-----+
| id | title | time | content |
+----+-----+-----+-----+
| 1 | first | 2018-07-03 15:29:37 | hdfs |
| 2 | second | 2018-07-03 15:30:57 | mr |
| 3 | third | 2018-07-03 15:31:07 | yarn |
+----+-----+-----+-----+
3 rows in set (0.00 sec)
```

For more information about Sqoop usage, please see the [official documentation](#).

Incremental Data Import into HDFS

Last updated : 2020-10-13 16:10:54

Sqoop is an open-source tool for transferring data between Hadoop and traditional databases such as MySQL and PostgreSQL. It can import data in a relational database (e.g., MySQL, Oracle, and Postgres) into Hadoop's HDFS, and vice versa. One of the highlights of Sqoop is its ability to import data in a relational database into HDFS through Hadoop MapReduce.

This document describes the incremental import operation of Sqoop, i.e., importing only new or updated data in the database into HDFS. This can be done in either append or lastmodified mode. The former can be used in the case where there is only new but not modified data in the database, while the latter can be used in either case.

1. Preparations for Development

- Confirm that you have activated Tencent Cloud and created an EMR cluster. When creating the EMR cluster, select the Sqoop component on the software configuration page.
- Relevant software programs such as Sqoop are installed in the `/usr/local/service/` directory of the CVM instance for the EMR cluster.

2. Using the append Mode

This section will continue to use the use case from the previous section.

Enter the EMR Console and copy the instance ID of the target cluster, i.e., the cluster name. Then, enter the TencentDB for MySQL Console, use Ctrl+F to find the MySQL database corresponding to the cluster, and view the private IP address of the database (`$mysqlIP`).

Log in to any node (preferably a master one) in the EMR cluster. For more information on how to log in to EMR, please see [Logging in to Linux Instances](#). Here, you can choose to log in with WebShell. Click "Log in" on the right of the desired CVM instance to enter the login page. The default username is `root`, and the password is the one you set when creating the EMR cluster. Once the correct credentials are entered, you can enter the command line interface.

Run the following command in EMR command-line interface to switch to the Hadoop user and go to the Sqoop folder:

```
[root@172 ~]# su hadoop
[hadoop@172 ~]# cd /usr/local/service/sqoop
```

Connect to the MySQL database:

```
[hadoop@172 sqoop]$ mysql -h $mysqlIP -p
Enter password:
```

The password is the one you set when you created the EMR cluster.

After connecting to the MySQL database, add a new data entry to the table `sqoop_test`, as shown below:

```
mysql> use test;
Database changed

mysql> insert into sqoop_test values(null, 'forth', now(), 'hbase');
Query ok, 1 row affected(0.00 sec)
```

View the data in the table:

```
Mysql> select * from sqoop_test;
+----+-----+-----+-----+
| id | title | time | content |
+----+-----+-----+-----+
| 1 | first | 2018-07-03 15:29:37 | hdfs |
| 2 | second | 2018-07-03 15:30:57 | mr |
| 3 | third | 2018-07-03 15:31:07 | yarn |
| 4 | forth | 2018-07-03 15:39:38 | hbase |
+----+-----+-----+-----+
4 rows in set (0.00 sec)
```

Sync the new data entry in append mode to the HDFS path where the data in the previous section is stored:

```
[hadoop@172 sqoop]$ bin/sqoop-import --connect jdbc:mysql://$mysqlIP/test --usern
ame
root -P --table sqoop_test --check-column id --incremental append --last-value 3
--target-dir
/sqoop
```

Here, `$mysqlIP` is the private IP address of your MySQL database.

To run the command, you need to enter the password for the database, which is defaulted to the password you set when you created the EMR cluster. There are more parameters than the normal `sqoop-import` command, where `--check-column` is the reference data during import, `--incremental` is the import mode (i.e., append in this example), and `--last-value` is the value of the reference data. All data entries that are newer than this value are imported into HDFS.

After successful execution, you can view the updated data in the corresponding directory of HDFS:

```
[hadoop@172 sqoop]$ hadoop fs -cat /sqoop/*
1, first, 2018-07-03 15:29:37.0,hdfs
2, second, 2018-07-03 15:30:57.0,mr
```

```
3, third, 2018-07-03 15:31:07.0, yarn
4, forth, 2018-07-03 15:39:38.0, hbase
```

Using a Sqoop job

To sync data to HDFS in append mode, you need to manually enter `--last-value` each time, but you can also use the Sqoop job method, where Sqoop automatically saves the value of `last-value` in the last successful import. To use it, you need to start the `sqoop-metastore` process in the following steps:

Start the `sqoop-metastore` process in `conf/sqoop-site.xml` first:

```
<property>
<name>sqoop.metastore.client.enable.autoconnect</name>
<value>>true</value>
</property>
```

Then, start the `sqoop-metastore` service in the `bin` directory:

```
./sqoop-metastore &
```

Create a Sqoop job by running the following command:

Note :

This command is applicable to Sqoop 1.4.6.

```
[hadoop@172 sqoop]$ bin/sqoop job --create job1 -- import --connect
jdbc:mysql://$mysqlIP/test --username root -P --table sqoop_test --check-column i
d
--incremental append --last-value 4 --target-dir /sqoop
```

Here, `$mysqlIP` is the private IP address of your MySQL database. With this command, a Sqoop job is successfully created, and each execution will automatically use the last modified value of `last-value`.

Add a new data entry to the table `sqoop_test` in the MySQL database:

```
mysql> insert into sqoop_test values(null, 'fifth', now(), 'hive');
Query ok, 1 row affected(0.00 sec)
```

```
Mysql> select * from sqoop_test;
+----+-----+-----+-----+
| id | title | time | content |
+----+-----+-----+-----+
| 1 | first | 2018-07-03 15:29:37 | hdfs |
```

```
| 2 | second | 2018-07-03 15:30:57 | mr |
| 3 | third | 2018-07-03 15:31:07 | yarn |
| 4 | forth | 2018-07-03 15:39:38 | hbase |
| 5 | fifth | 2018-07-03 16:02:29 | hive |
+----+-----+-----+-----+-----+
5 rows in set (0.00 sec)
```

Then, execute the Sqoop job:

```
[hadoop@172 sqoop]$ bin/sqoop job --exec job1
```

To run this command, you need to enter the password for your MySQL database. After successful execution, you can view the updated data in the corresponding directory of HDFS:

```
[hadoop@172 sqoop]$ hadoop fs -cat /sqoop/*
1, first, 2018-07-03 15:29:37.0,hdfs
2, second, 2018-07-03 15:30:57.0,mr
3, third, 2018-07-03 15:31:07.0,yarn
4,forth,2018-07-03 15:39:38.0,hbase
5,fifth,2018-07-03 16:02:29.0,hive
```

3. Using the lastmodified Mode

Create a Sqoop job in lastmodified mode of sqoop-import directly to first query the last modified time in sqoop_test:

```
mysql> select max(time) from sqoop_test;
```

Create a Sqoop job:

```
[hadoop@172 sqoop]$ bin/sqoop job --create job2 -- import --connect jdbc:mysql://
$mysqlIP/test --username root -P --table sqoop_test --check-column time --increme
ntal lastmodified --merge-key id --last-value '2018-07-03 16:02:29' --target-dir
/sqoop
```

Parameter description

- `$mysqlIP` : refers to the private IP address of your MySQL database
- `--check-column` : must use a timestamp
- `--incremental` : selects the lastmodified mode
- `--merge-key` : selects the ID
- `--last-value` : refers to the last modified time in the table that is queried. All modifications made after this time will be synced to HDFS, and the Sqoop job will automatically save and update the value each time.

Add data to the table `sqoop_test` in the MySQL database and make changes:

```
mysql> insert into sqoop_test values(null, 'sixth', now(), 'sqoop');
Query ok, 1 row affected(0.00 sec)

mysql> update sqoop_test set time=now(), content='spark' where id = 1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 changed: 1 warnings: 0

Mysql> select * from sqoop_test;
+----+-----+-----+-----+
| id | title | time | content |
+----+-----+-----+-----+
| 1 | first | 2018-07-03 16:07:46 | spark |
| 2 | second | 2018-07-03 15:30:57 | mr |
| 3 | third | 2018-07-03 15:31:07 | yarn |
| 4 | forth | 2018-07-03 15:39:38 | hbase |
| 5 | fifth | 2018-07-03 16:02:29 | hive |
| 6 | fifth | 2018-07-03 16:09:58 | sqoop |
+----+-----+-----+-----+
6 rows in set (0.00 sec)
```

Execute the Sqoop job:

```
[hadoop@172 sqoop]$ bin/sqoop job --exec job2
```

To run this command, you need to enter the password for your MySQL database. After successful execution, you can view the updated data in the corresponding directory of HDFS:

```
[hadoop@172 sqoop]$ hdfs dfs -cat /sqoop/*
1,first,2018-07-03 16:07:46.0,spark
2,second,2018-07-03 15:30:57.0,mr
3,third,2018-07-03 15:31:07.0,yarn
4,forth,2018-07-03 15:39:38.0,hbase
5,fifth,2018-07-03 16:02:29.0,hive
6,sixth,2018-07-03 16:09:58.0,sqoop
```

For more information on Sqoop usage, please see the [official documentation](#).

Importing and Exporting Data Between Hive and TencentDB for MySQL

Last updated : 2021-10-29 10:09:45

This document describes how to use Sqoop on EMR to import and export data between MySQL and Hive.

1. Prerequisites

- Confirm that you have activated Tencent Cloud and created an EMR cluster. When creating the EMR cluster, you need to select the Sqoop and Hive components on the software configuration page.
- Relevant software programs such as Sqoop are installed in the `/usr/local/service/` directory of the CVM instance for the EMR cluster.

2. Importing Data from TencentDB for MySQL into Hive

This section will continue to use the use case from the previous section.

Enter the [Elastic MapReduce Console](#) and copy the instance ID of the target cluster, i.e., the cluster name. Then, enter the TencentDB for MySQL Console, use Ctrl+F to find the MySQL database corresponding to the cluster, and view the private IP address of the database (`$mysqlIP`).

Log in to any node (preferably a master one) in the EMR cluster. For more information on how to log in to EMR, please see [Logging in to Linux Instances](#). Here, you can choose to log in with WebShell. Click "Log in" on the right of the desired CVM instance to enter the login page. The default username is `root`, and the password is the one you set when creating the EMR cluster. Once the correct credentials are entered, you can enter the command line interface.

Run the following command on the EMR command line to switch to the Hadoop user and go to the Hive folder:

```
[root@172 ~]# su hadoop
[hadoop@172 ~]# cd /usr/local/service/hive
```

Create a Hive database:

```
[hadoop@172 hive]$ hive
hive> create database hive_from_sqoop;
OK
Time taken: 0.167 seconds
```

Import the MySQL database created in the previous section into Hive by running the `sqoop-import` command:

```
[hadoop@172 hive]# cd /usr/local/service/sqoop
[hadoop@172 sqoop]$ bin/sqoop-import --connect jdbc:mysql://$mysqlIP/test --username
ame
root -P --table sqoop_test_back --hive-database hive_from_sqoop --hive-import --h
ive-table hive_from_sqoop
```

- `$mysqlIP`: private IP of the TencentDB instance.
- `test`: MySQL database name.
- `--table`: name of the MySQL table to be exported.
- `--hive-database`: Hive database name.
- `--hive-table`: name of the Hive table to be imported.

Running this command requires the password of your MySQL database, which is the one you set when you created the EMR cluster. After successful execution, you can view the imported database in Hive:

```
hive> select * from hive_from_sqoop;
OK
1 first 2018-07-03 16:07:46.0 spark
2 second 2018-07-03 15:30:57.0 mr
3 third 2018-07-03 15:31:07.0 yarn
4 forth 2018-07-03 15:39:38.0 hbase
5 fifth 2018-07-03 16:02:29.0 hive
6 sixth 2018-07-03 16:09:58.0 sqoop
Time taken: 1.245 seconds, Fetched: 6 row(s)
```

3. Importing Data from Hive into TencentDB for MySQL

Sqoop supports importing data from Hive tables into TencentDB for MySQL. To do so, create a table in Hive first and then import data.

Log in to any node (preferably a master one) in the EMR cluster. Run the following command on the EMR command line to switch to the Hadoop user and go to the Hive folder:

```
[root@172 ~]# su hadoop
[hadoop@172 ~]# cd /usr/local/service/hive
```

Create a bash script file named `gen_data.sh` and add the following code to it:

```
#!/bin/bash
MAXROW=1000000 # Specify the number of data rows to be generated
for((i = 0; i < $MAXROW; i++))
do
    echo $RANDOM, \"$RANDOM\"
done
```

Run it as follows:

```
[hadoop@172 hive]$ ./gen_data.sh > hive_test.data
```

This script file will generate 1,000,000 random number pairs and save them to the `hive_test.data` file.

Run the following command to upload the generated test data to HDFS first:

```
[hadoop@172 hive]$ hdfs dfs -put ./hive_test.data /$hdfspath
```

Here, `$hdfspath` is the path to your file on HDFS.

Connect to Hive and create a test table:

```
[hadoop@172 hive]$ bin/hive
hive> create database hive_to_sqoop; # Create the database `hive_to_sqoop`
OK
Time taken: 0.176 seconds
hive> use hive_to_sqoop; # Switch databases
OK
Time taken: 0.176 seconds
hive> create table hive_test (a int, b string)
hive> ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';
# Create a data table named `hive_test` and specify the
column separator as ``,`
OK
Time taken: 0.204 seconds
hive> load data inpath "$hdfspath/hive_test.data" into table hive_test; # Import
the data
```

`$hdfspath` is the path to your file stored in HDFS.

After success, you can run the `quit` command to exit the Hive data warehouse. Then, connect to TencentDB for MySQL and create a corresponding table:

```
[hadoop@172 hive]$ mysql -h $mysqlIP -p
Enter password:
```


Here, `$mysqlIP` is the private IP address of this database, and the password is the one you set when creating the cluster.

Create a table named `test` in MySQL. **Note that the field names in the MySQL table must be the same as those in the Hive table:**

```
mysql> create table table_from_hive (a int,b varchar(255));
```

You can exit MySQL after successfully creating the table.

There are two ways to import data from a Hive data warehouse into a TencentDB for MySQL database by using Sqoop: using the Hive data stored in HDFS directly or using HCatalog to import the data.

Using Hive data stored in HDFS

Switch to the Sqoop folder and export the data from the Hive database to the TencentDB for MySQL database by running the following command:

```
[hadoop@172 hive]$ cd ../sqoop/bin
[hadoop@172 bin]$ ./sqoop-export --connect jdbc:mysql://$mysqlIP/test --username
root -P
--table table_from_hive --export-dir /usr/hive/warehouse/hive_to_sqoop.db/hive_te
st
```

Here, `$mysqlIP` is the private IP address of your TencentDB for MySQL instance, `test` is the name of the MySQL database, `--table` is followed by the name of the table in the MySQL database, and `--export-dir` is followed by the location of the Hive table data stored in HDFS.

Importing data by using HCatalog

Switch to the Sqoop folder and export the data from the Hive database to the TencentDB for MySQL database by running the following command:

```
[hadoop@172 hive]$ cd ../sqoop/bin
[hadoop@172 bin]$ ./sqoop-export --connect jdbc:mysql://$mysqlIP/test --username
root -P
--table table_from_hive --hcatalog-database hive_to_sqoop --hcatalog-table hive_t
est
```

Here, `$mysqlIP` is the private IP address of your TencentDB for MySQL instance, `test` is the name of the MySQL database, `--table` is followed by the name of the table in the MySQL database, `-hcatalog-database` is followed by the name of the database where the Hive table to be exported is stored, and `--hcatalog-table` is followed by the name of the Hive table to be exported.

After the operation is completed, you can enter the MySQL database to see whether the import is successful:

```
[hadoop@172 hive]$ mysql -h $mysqlIP -p # Connect to MySQL
Enter password:
mysql> use test;
Database changed
mysql> select count(*) from table_from_hive; # Now there are 1,000,000 data entries in the table
+-----+
| count(*) |
+-----+
| 1000000 |
+-----+
1 row in set (0.03 sec)
mysql> select * from table_from_hive limit 10; # View the first 10 entries in the table
+-----+-----+
| a | b |
+-----+-----+
| 28523 | "3394" |
| 31065 | "24583" |
| 399 | "23629" |
| 18779 | "8377" |
| 25376 | "30798" |
| 20234 | "22048" |
| 30744 | "32753" |
| 21423 | "6117" |
| 26867 | "16787" |
| 18526 | "5856" |
+-----+-----+
10 rows in set (0.00 sec)
```

You can view more parameters about the `sqoop-export` command by running the following command:

```
[hadoop@172 bin]$ ./sqoop-export --help
```

4. Importing a Hive Table in ORC Format into TencentDB for MySQL

ORC is a columnar storage format that can greatly improve the performance of Hive. This section describes how to create an ORC table, load data into it, and then use the Sqoop on EMR to export the data stored in ORC format in Hive to TencentDB for MySQL.

Note :

After importing a Hive table in ORC format to TencentDB for MySQL, you cannot use the data stored in HDFS directly; instead, you have to use HCatalog.

This section will continue to use the use case from the previous section.

Log in to a master node of the EMR cluster and run the following command on the EMR command line to switch to the Hadoop user and go to the Hive folder:

```
[root@172 ~]# su hadoop
[hadoop@172 ~]# cd /usr/local/service/hive
```

Create a table in the database `hive_from_sqoop` created in the previous section:

```
[hadoop@172 hive]$ hive
hive> use hive_to_sqoop;
OK
Time taken: 0.013 seconds
hive> create table if not exists orc_test(a int,b string) ROW FORMAT DELIMITED FI
ELDS TERMINATED BY ',' stored as orc;
```

You can view the storage format of the data in the table by running the following command:

```
hive> show create table orc_test;
OK
CREATE TABLE `orc_test` (
  `a` int,
  `b` string)
ROW FORMAT SERDE
'org.apache.hadoop.hive.ql.io.orc.OrcSerde'
WITH SERDEPROPERTIES (
  'field.delim'=',',
  'serialization.format'=',')
STORED AS INPUTFORMAT
'org.apache.hadoop.hive.ql.io.orc.OrcInputFormat'
OUTPUTFORMAT
'org.apache.hadoop.hive.ql.io.orc.OrcOutputFormat'
LOCATION
'hdfs://HDFS2789/usr/hive/warehouse/hive_to_sqoop.db/orc_test'
TBLPROPERTIES (
  'COLUMN_STATS_ACCURATE'='{\"BASIC_STATS\": \"true\"}',
  'numFiles'='0',
  'numRows'='0',
```

```
'rawDataSize'='0',
'totalSize'='0',
'transient_lastDdlTime'='1533563293')
Time taken: 0.041 seconds, Fetched: 21 row(s)
```

It can be seen from the returned data that the data storage format in the table is ORC.

There are several ways to import data into a Hive table in ORC format, but only one of them is described below, i.e., importing data into an ORC table by creating a temporary Hive table in text storage format. The table `hive_test` created in the previous section is used here as the temporary table, and the following command is used to import the data:

```
hive> insert into table orc_test select * from hive_test;
```

After successful import, you can view the data in the table by running the `select` command.

Then, use Sqoop to export the Hive table in ORC format to MySQL. Connect to TencentDB for MySQL and create a corresponding table. The specific way to connect is as described above.

```
[hadoop@172 hive]$ mysql -h $mysqlIP -p
Enter password:
```

Here, `$mysqlIP` is the private IP address of this database, and the password is the one you set when creating the cluster.

Create a table named `test` in MySQL. **Note that the field names in the MySQL table must be the same as those in the Hive table:**

```
mysql> create table table_from_orc (a int,b varchar(255));
```

You can exit MySQL after successfully creating the table.

Switch to the Sqoop folder and export the data in ORC format from the Hive database to the TencentDB for MySQL database by running the following command:

```
[hadoop@172 hive]$ cd ../sqoop/bin
[hadoop@172 bin]$ ./sqoop-export --connect jdbc:mysql://$mysqlIP/test --username
root -P
--table table_from_orc --hcatalog-database hive_to_sqoop --hcatalog-table orc_tes
t
```

Here, `$mysqlIP` is the private IP address of your TencentDB for MySQL instance, `test` is the name of the MySQL database, `--table` is followed by the name of the table in the MySQL database, `-hcatalog-`

`database` is followed by the name of the database where the Hive table to be exported is stored, and `--`
`hcatalog-table` is followed by the name of the Hive table to be exported.

After successful import, you can view the data in the corresponding table in the MySQL database:

```
mysql> select count(*) from table_from_orc;
+-----+
| count(*) |
+-----+
| 1000000 |
+-----+
1 row in set (0.24 sec)
mysql> select * from table_from_orc limit 10;
+-----+-----+
| a | b |
+-----+-----+
| 28523 | "3394" |
| 31065 | "24583" |
| 399 | "23629" |
| 18779 | "8377" |
| 25376 | "30798" |
| 20234 | "22048" |
| 30744 | "32753" |
| 21423 | "6117" |
| 26867 | "16787" |
| 18526 | "5856" |
+-----+-----+
10 rows in set (0.00 sec)
```

For more information on Sqoop usage, please see the [official documentation](#).

Hue Development Guide

Hue Overview

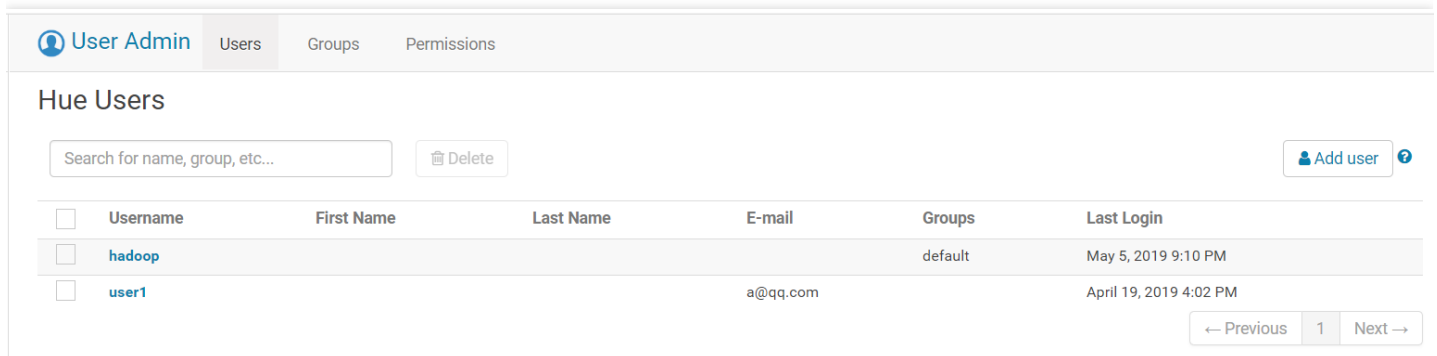
Last updated : 2022-07-04 11:23:37

Hue is an open-source Apache Hadoop UI system that evolved from Cloudera Desktop, which Cloudera contributed to the Hadoop project of the Apache Software Foundation. Hue is implemented on the basis of Django, a Python web framework. By using Hue, you can interact with Hadoop clusters in the web-based console on a browser, such as manipulating HDFS data, running MapReduce jobs, executing Hive SQL statements, and browsing HBase databases.

Accessing Hue WebUI

To use the Hue component to manage workflows, log in to the Hue console first as shown below:

1. Log in to the [EMR console](#), click the **ID/Name** of the target cluster to enter the cluster details page, and click **Cluster Service**.
2. Find the Hue component on the list page and click **WebUI Access Address** to enter the Hue page.
3. When logging in to the Hue console for the first time, use the root account and the password set when you created the cluster.



The screenshot shows the 'User Admin' interface in Hue. At the top, there are tabs for 'User Admin', 'Users', 'Groups', and 'Permissions'. Below the tabs, there is a search bar labeled 'Search for name, group, etc...' and a 'Delete' button. To the right, there is an 'Add user' button. Below these elements is a table with the following columns: Username, First Name, Last Name, E-mail, Groups, and Last Login. The table contains two rows of user data:

<input type="checkbox"/>	Username	First Name	Last Name	E-mail	Groups	Last Login
<input type="checkbox"/>	hadoop				default	May 5, 2019 9:10 PM
<input type="checkbox"/>	user1			a@qq.com		April 19, 2019 4:02 PM

At the bottom right of the table, there are navigation buttons: '← Previous', '1', and 'Next →'.

Note :

As the default component account upon start in EMR is Hadoop, you need to create a Hadoop account after logging in to the Hue console with the root account for the first time. All subsequent jobs should be submitted by using the Hadoop account.

User Permission Management

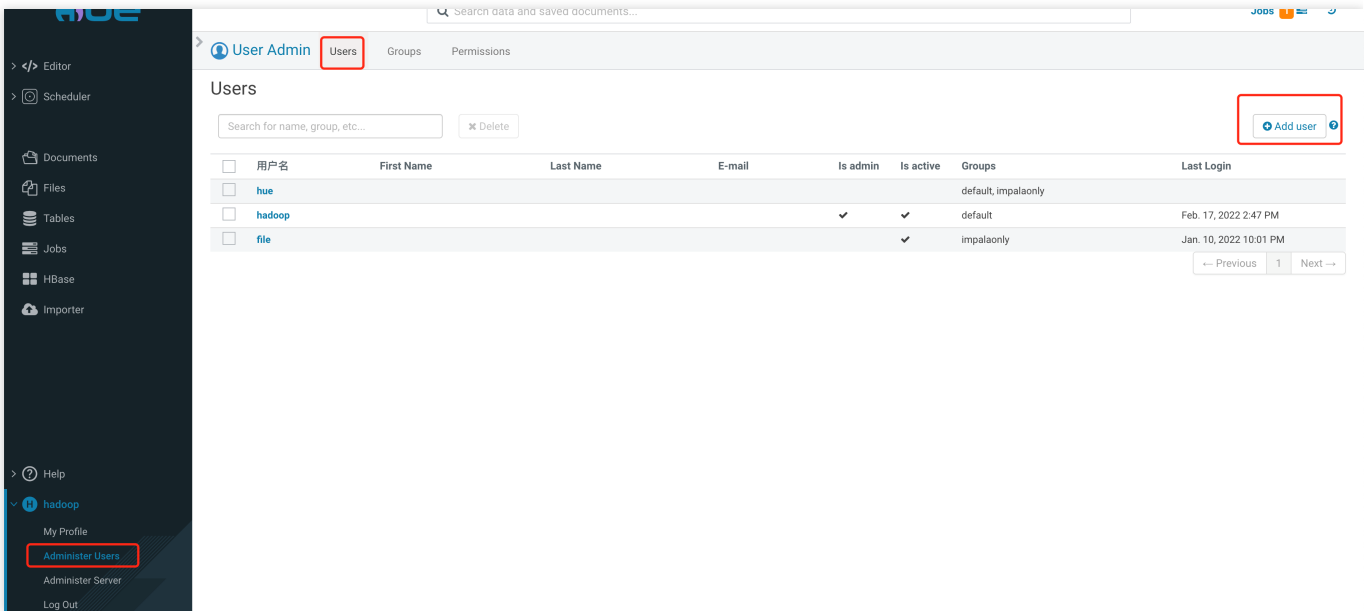
Log in to Hue with the admin account first.

Note :

The following takes Hue with OpenLDAP not used for user management as an example to describe the directions. In EMR v2.6.0 and later and EMR v3.3.0 and later, Hue uses OpenLDAP users by default, and Hue users can be directly managed through [user management](#).

1. Add a user.

i. Click **Add user** on the right.



ii. Enter the user information.

The screenshot shows the 'User Admin' interface with tabs for 'Users', 'Groups', and 'Permissions'. The 'Create user' page is displayed, with 'Step 1: Credentials (required)' selected. The form contains the following elements:

- Username:
- New Password:
- Password confirmation:
- Create home directory:

iii. Enter the user group and other information.

The screenshot shows the 'Step 2: Profile and Groups' section of the 'Create user' form. The form contains the following elements:

- First name:
- Last name:
- Email address:
- Groups: A list of groups with checkboxes. 'default' is checked, and 'impalaonly' is highlighted with a red box.

iv. Click **Add user**.

Create user

Step 1: Credentials (required) Step 2: Profile and Groups Step 3: Advanced

Active ?

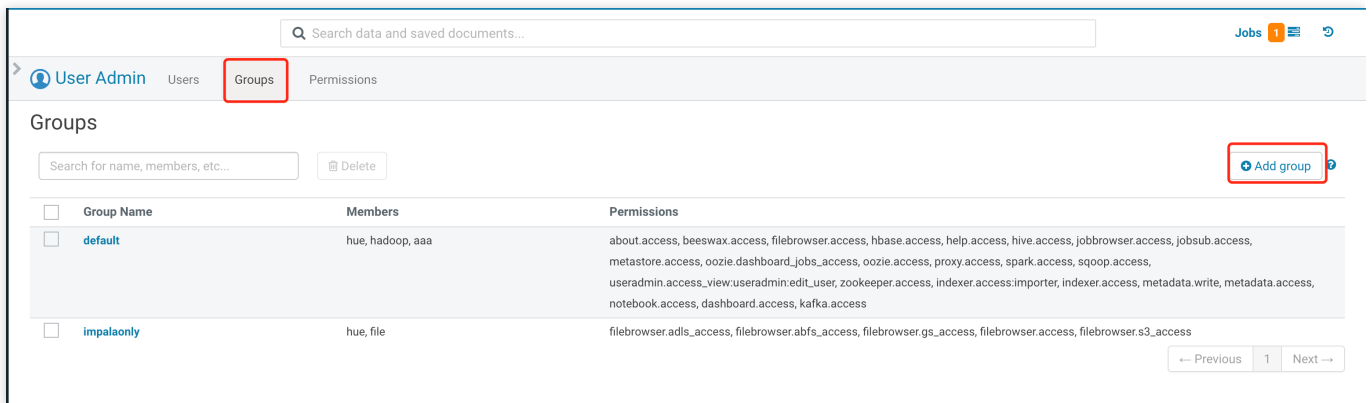
Superuser status ?

[Back](#) [Next](#) **Add user**

2. Perform permission control.

You can assign different permissions to groups through Hue and add users to groups to get specific permissions.

i. Click **Groups** at the top of the user management page and then click **Add group** on the right.



ii. Enter the user group information, select the users to be added to the group, specify the permissions for the group, and click **Add group**.

User Admin Users **Groups** Permissions

Create group

Name

members Select all

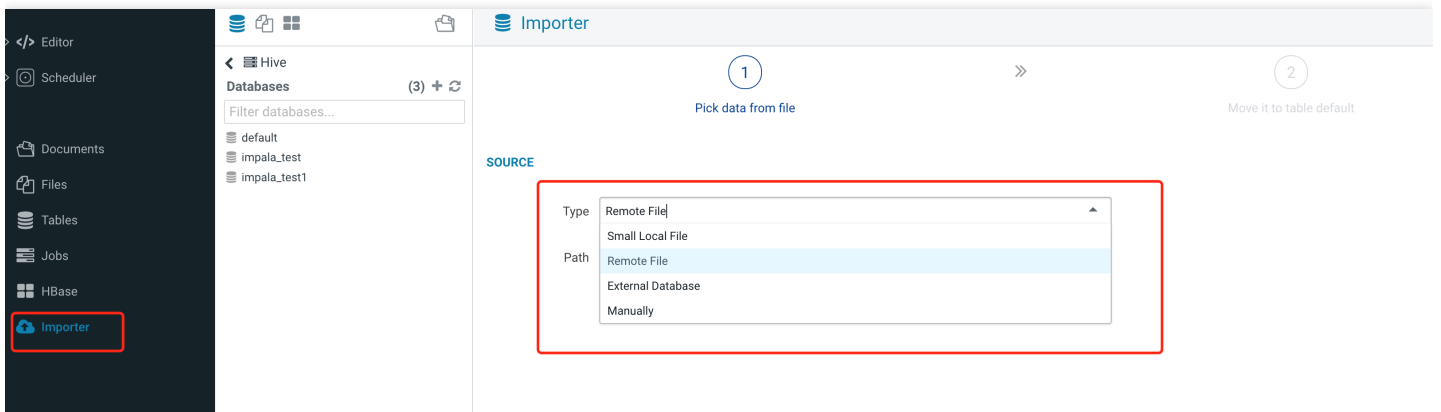
- A**
 - aaa
- F**
 - file
- H**
 - hadoop
 - hue

permissions Select all

- K**
 - kafka.access:Launch this application(32)
- M**
 - metadata.write:Allow edition of metadata like tags.(28)
 - metadata.access:Launch this application(29)
 - metastore.write:Allow DDL operations. Need the app access too.(14)
 - metastore.access:Launch this application(15)
- N**
 - notebook.access:Launch this application(30)

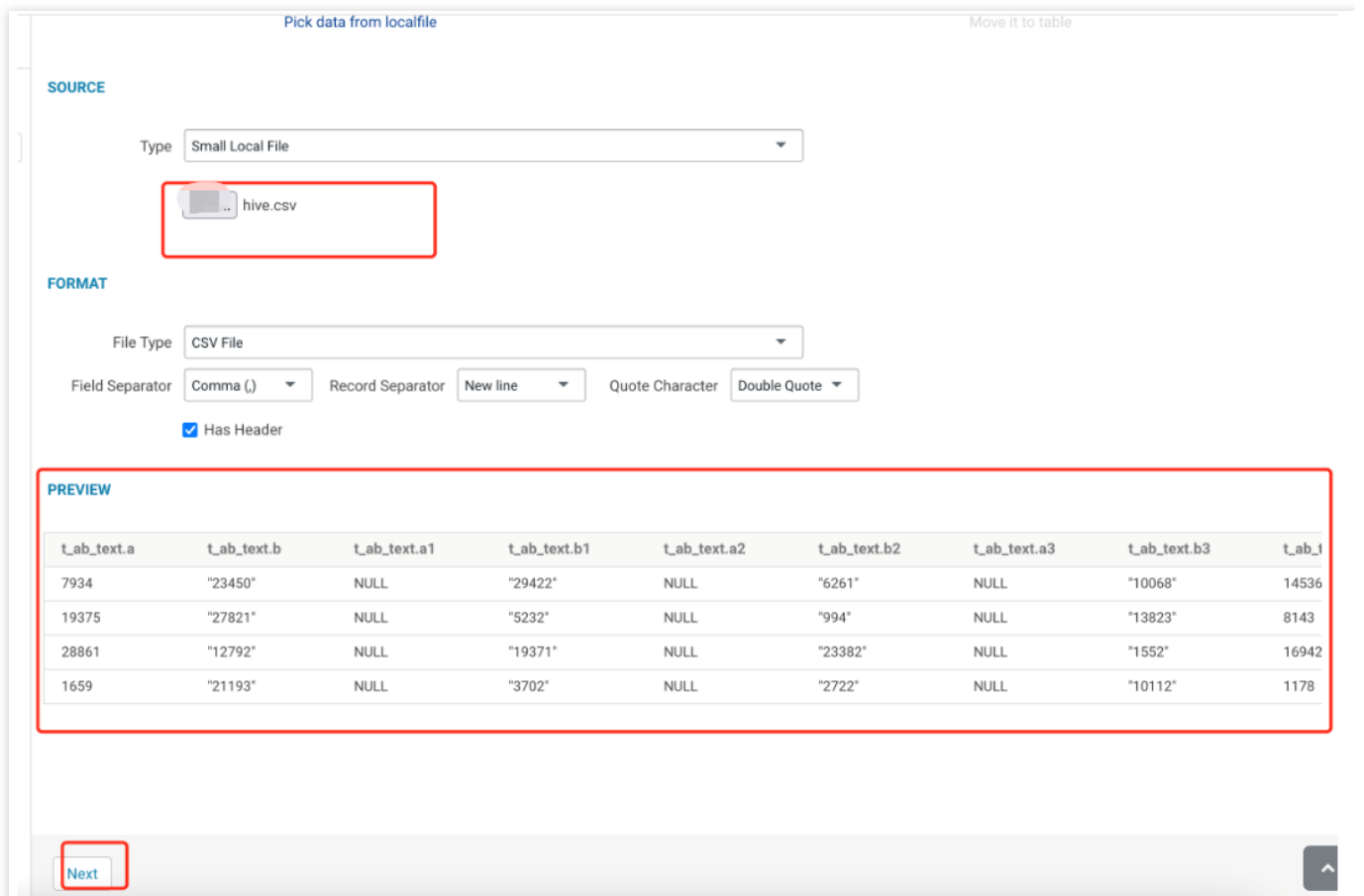
Data Import

Hue allows you to import data from a local file, HDFS file, external database, or manually.



1. Import a local file.

- i. Click **Choose file** and select a CSV file. Hue will automatically recognize the delimiter and generate a preview. Click **Next** to import the file to a table.



ii. Enter the table information and click **Save**.

Dialect Hive

Name impala_test.test_load1

PROPERTIES

Format Text

Extras

Store in Default location

Transactional table Insert only

Import data

Description Description

Custom char delimiters

Partitions + Add partition

FIELDS

Name	t_ab_text.a	Type	bigint		7934	19375
Name	t_ab_text.b	Type	string		"23450"	"27821"
Name	t_ab_text.a1	Type	string		NULL	NULL

Back

2. Import an HDFS file.

i. Select a CSV file from HDFS.

The screenshot shows the 'Importer' interface with the following sections and elements:

- Progress:** Step 1 (Pick data from file /tmp/aaa.csv) is active, and Step 2 (Move it to table default.aaa) is next.
- SOURCE:** A dropdown menu for 'Type' is set to 'Remote File'. The 'Path' field contains '/tmp/aaa.csv'.
- FORMAT:** 'File Type' is set to 'CSV File'. 'Field Separator' is 'Comma (,)', 'Record Separator' is 'New line', and 'Quote Character' is 'Double Quote'. The 'Has Header' checkbox is checked.
- PREVIEW:** A table with two columns: 'note_hvie.id' and 'note_hvie.name'. The data rows are (2, tom) and (1, jack).
- Navigation:** A 'Next' button is located at the bottom left.

ii. Enter the table information and click **Save**.

Importer

Pick data from file /tmp/aaa.csv

Move it to table default.aaa1

DESTINATION

Type Table

Name default.aaa1

PROPERTIES

Format Text

Extras

Partitions + Add partition

FIELDS

Name	id	Type	bigint	2	1
Name	name	Type	string	tom	jack

Back

3. Import an external database.

i. Enter the external database information, click **Test Connection** to get the database information, select the database and table, and click **Next**.

1 » 2
Pick data from rdbms Move it to table auth_group

SOURCE

Type External Database

Mode Custom Configured

Driver MySQL

Hostname [Redacted]

Port 3306

[Redacted] root

[Redacted]

[Test Connection](#)

Database Name hue

All Tables

Table Name auth_group

PREVIEW

[Next](#)

ii. Enter the information of the target table, click **Libs**, select the MySQL driver, and click **Save**.

The screenshot shows the 'Importer' interface with the following configuration:

- DESTINATION**: Type is 'Table', Name is 'auth_group'.
- PROPERTIES**: Libs is '/tmp/mysql-connector-java-5.1.47.jar' (highlighted with a red box).
- FIELDS**:
 - Name: 'id', Type: 'INTEGER(11)', 1
 - Name: 'name', Type: 'VARCHAR(80)', default, impalaonly

Buttons: Back, Save

Job Management

Click the **Jobs** tab on the right to enter the job management page. Then, click a job type tab at the top to view and manage jobs.

Search data and saved documents...

Jobs 1

Job Browser Jobs Impala Workflows Schedules Bundles SLAs Livy

user:hadoop Succeeded Running Failed in the last 7 days

Kill

Running

Name	用户	Type	Status	Progress	Group	Started	Duration	Id
<input type="checkbox"/> Zeppelin Flink Session	hadoop	Apache Flink	RUNNING	100%	root.default	2022年2月16日晚上8点28分	19h, 8m, 14s	application_1645011680999_0002

Completed

Name	用户	Type	Status	Progress	Group	Started	Duration	Id
<input type="checkbox"/> select * from t1 order by id asc (Stage-1)	hadoop	MAPREDUCE	SUCCEEDED	100%	root.default	2022年2月17日下午2点46分	15.57s	application_1645011680999_0094
<input type="checkbox"/> insert into t1 values (1, 'jack') (Stage-1)	hadoop	MAPREDUCE	SUCCEEDED	100%	root.default	2022年2月17日下午2点46分	11.92s	application_1645011680999_0093
<input type="checkbox"/> insert into t1 values (2, 'tom') (Stage-1)	hadoop	MAPREDUCE	SUCCEEDED	100%	root.default	2022年2月17日下午2点46分	10.72s	application_1645011680999_0092
<input type="checkbox"/> oozie-launcher:T-hive2:W=workflow_hive_20220214154805:A=hive-f317:ID=0000893-220216183138078-oozie-hado-W	hadoop	Oozie Launcher	SUCCEEDED	100%	root.default	2022年2月17日下午2点46分	49.33s	application_1645011680999_0091
<input type="checkbox"/> select * from t1 order by id asc (Stage-1)	hadoop	MAPREDUCE	SUCCEEDED	100%	root.default	2022年2月17日下午2点43分	17.43s	application_1645011680999_0090
<input type="checkbox"/> insert into t1 values (1, 'jack') (Stage-1)	hadoop	MAPREDUCE	SUCCEEDED	100%	root.default	2022年2月17日下午2点43分	10.94s	application_1645011680999_0089

Search data and saved documents...

Jobs 1

Job Browser Jobs Impala **Workflows** Schedules Bundles SLAs Livy

user:hadoop Succeeded Running Failed in the last 7 days

Resume Suspend Kill

Running

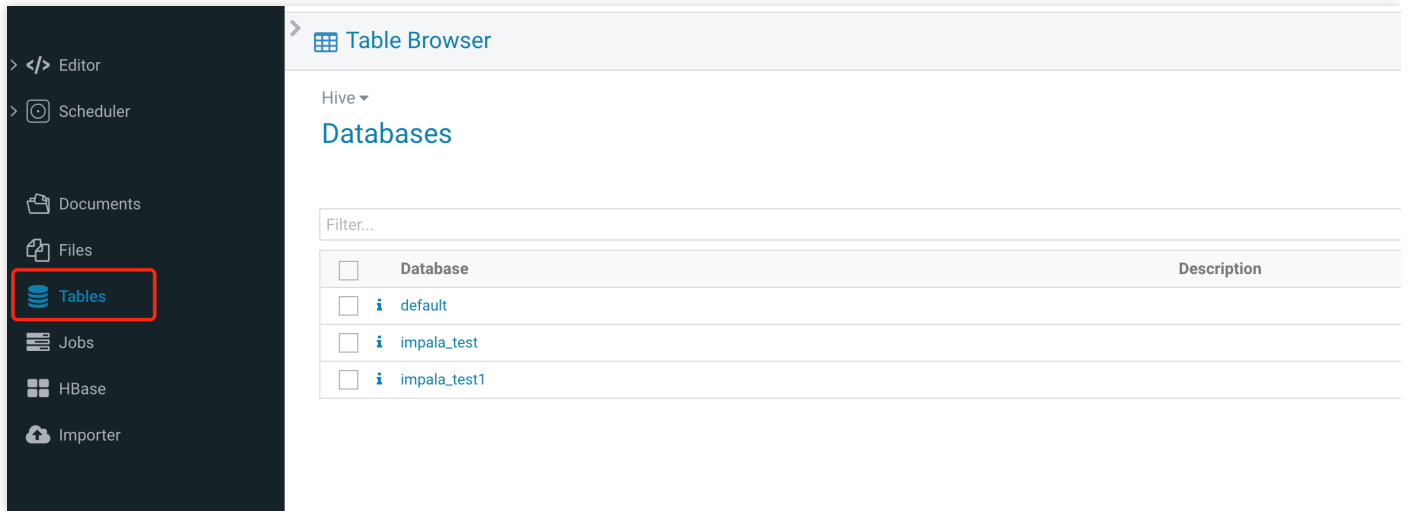
Name	用户	Type	Status	Progress	Group	Started	Duration	Id
------	----	------	--------	----------	-------	---------	----------	----

Completed

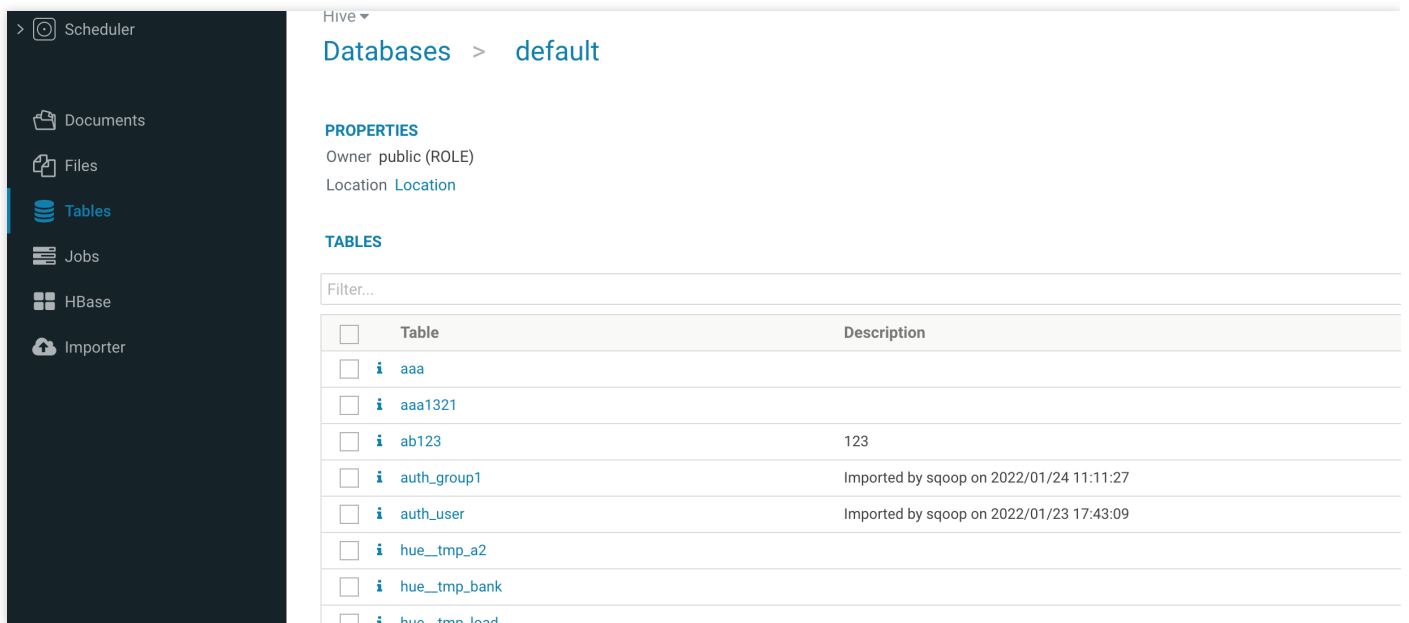
Name	用户	Type	Status	Progress	Group	Started	Duration	Id
<input type="checkbox"/> workflow_hive_20220214154805	hadoop	workflow	SUCCEEDED	100%		2022年2月17日下午2点46分	50s	0000893-220216183138078-oozie-hado-W
<input type="checkbox"/> workflow_hive_20220217144313	hadoop	workflow	SUCCEEDED	100%		2022年2月17日下午2点43分	50s	0000891-220216183138078-oozie-hado-W
<input type="checkbox"/> workflow_hive_20220214154805	hadoop	workflow	SUCCEEDED	100%		2022年2月17日中午11点49分	49s	0000890-220216183138078-oozie-hado-W
<input type="checkbox"/> workflow_hive_20220217114630	hadoop	workflow	SUCCEEDED	100%		2022年2月17日中午11点46分	50s	0000888-220216183138078-oozie-hado-W
<input type="checkbox"/> Batch for My Notebook	hadoop	workflow	SUCCEEDED	100%		2022年2月17日中午11点43分	2s	0000887-220216183138078-oozie-hado-W
<input type="checkbox"/> Batch for My Notebook	hadoop	workflow	SUCCEEDED	100%		2022年2月17日中午11点43分	3s	0000886-220216183138078-oozie-hado-W

Table Management

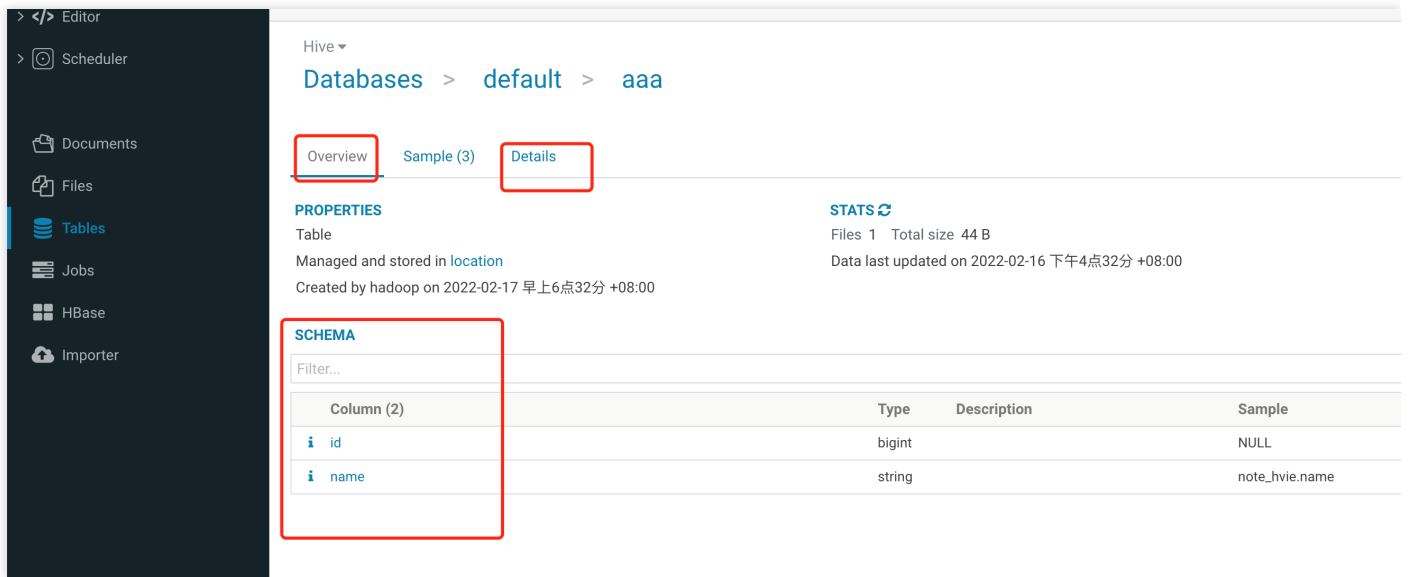
1. Click **Tables** on the right to enter the table management page and view the basic database information.



2. Click a database to view the information of its tables.



3. Click a table to view its details.



Hive ▾
Databases > default > aaa

Overview Sample (3) Details

PROPERTIES
Table
Managed and stored in [location](#)
Created by hadoop on 2022-02-17 早上6点32分 +08:00

STATS ↻
Files 1 Total size 44 B
Data last updated on 2022-02-16 下午4点32分 +08:00

SCHEMA
Filter...

Column (2)	Type	Description	Sample
i id	bigint		NULL
i name	string		note_hvie.name

Hue Best Practices

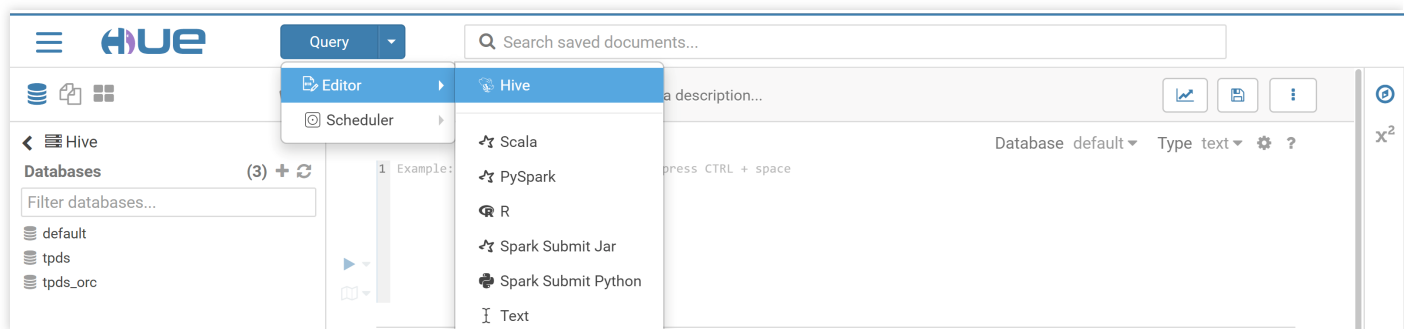
Last updated : 2022-05-16 12:52:25

This document describes how to use Hue.

Hive SQL Query

Hue's Beeswax app provides user-friendly and convenient Hive query capabilities, enabling you to select different Hive databases, write HQL statements, submit query tasks, and view results with ease.

1. At the top of the Hue console, select **Query > Editor > Hive**.



2. Enter the statement to be executed in the statement input box and click **Run** to run it.

The screenshot shows the Hive console interface. At the top, there is a header with the Hive logo, a refresh icon, and fields for 'Add a name...' and 'Add a description...'. On the right, there are icons for a chart, a document, and a menu. Below the header, the query input area contains the SQL statement: `1|select * from tpd.date_dim limit 5;`. To the left of the input area is a 'Run' button with a play icon. Below the input area, the execution progress shows '0.99s Database tpd' and 'Type text'. The results section is titled 'Results (5)' and contains a table with 5 rows and 6 columns. The columns are: `date_dim.d_date_sk`, `date_dim.d_date_id`, `date_dim.d_date`, `date_dim.d_month_seq`, and `date_dim.d_week_seq`. The rows contain data for dates from 1900-01-02 to 1900-01-06. Red arrows point from the text 'Run' and 'Search Results' to the respective elements in the interface.

	<code>date_dim.d_date_sk</code>	<code>date_dim.d_date_id</code>	<code>date_dim.d_date</code>	<code>date_dim.d_month_seq</code>	<code>date_dim.d_week_seq</code>
1	2415022	AAAAAAAAOKJNECAA	1900-01-02	0	1
2	2415023	AAAAAAAAPKJNECAA	1900-01-03	0	1
3	2415024	AAAAAAAALJNECAA	1900-01-04	0	1
4	2415025	AAAAAAAABLJNECAA	1900-01-05	0	1
5	2415026	AAAAAAAACLJNECAA	1900-01-06	0	1

HBase Data Query, Modification, and Display

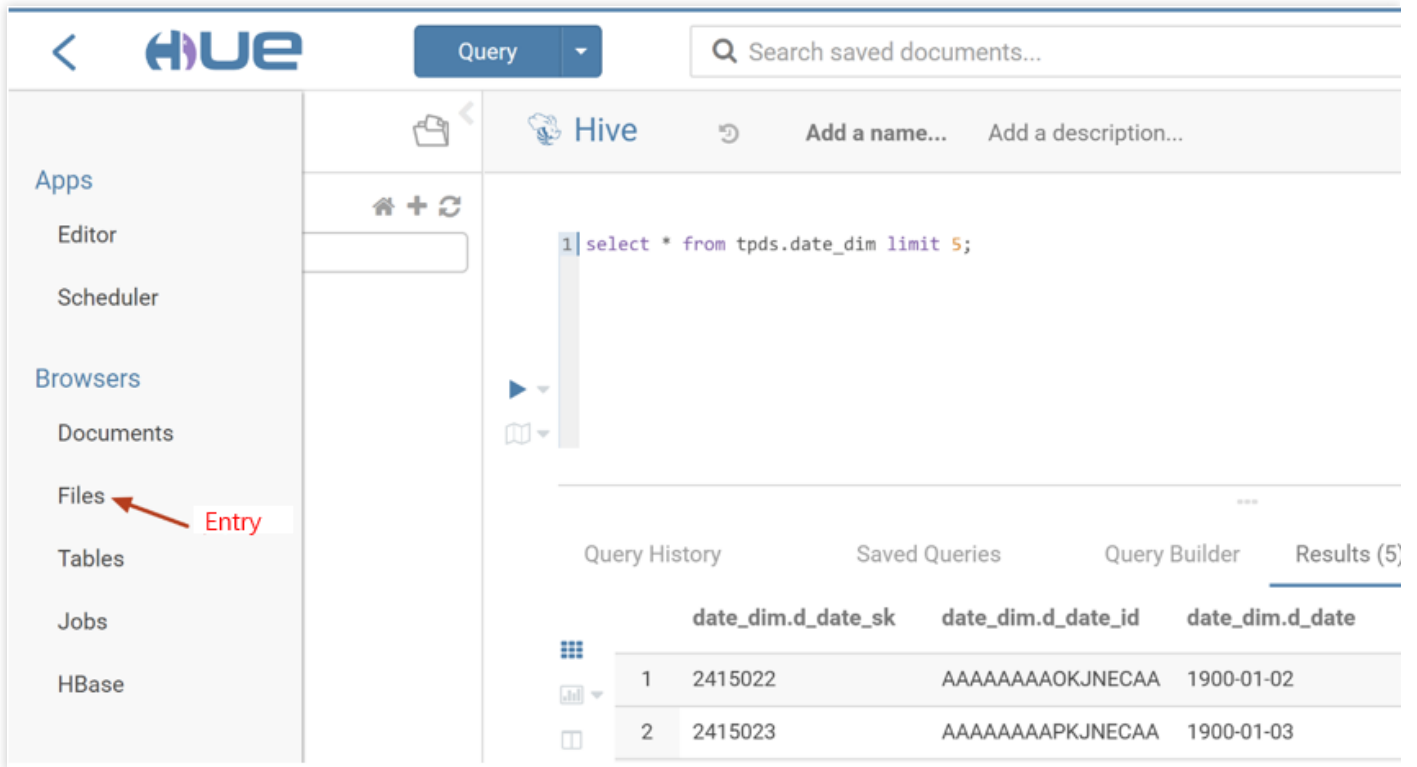
You can use HBase Browser to query, modify, and display data from tables in an HBase cluster.

The screenshot displays the HBase Browser interface. On the left sidebar, the 'HbaseCluster' menu is expanded, showing 'hbase_test' as the selected 'Hbase Table'. The main area shows the 'Home - HbaseCluster / hbase_test' view. A search bar at the top is labeled 'Search Input Box'. Below it, a table displays data with columns 'id' and 'name'. The first row has '1' and 'chris'. The second row has '2' and a timestamp. A 'Full Editor' button is visible next to the second row, labeled 'Edit a Column'. A 'Filter Columns/Families' button is labeled 'Column Filter'. A trash icon is labeled 'Delete a Row'. At the bottom right, a 'New Row' button is labeled 'Add a Row'. The status bar at the bottom indicates 'Fetched 10 entries starting from null in 0.258 seconds.' and includes buttons for 'Drop Rows', 'Bulk Upload', and 'New Row'.

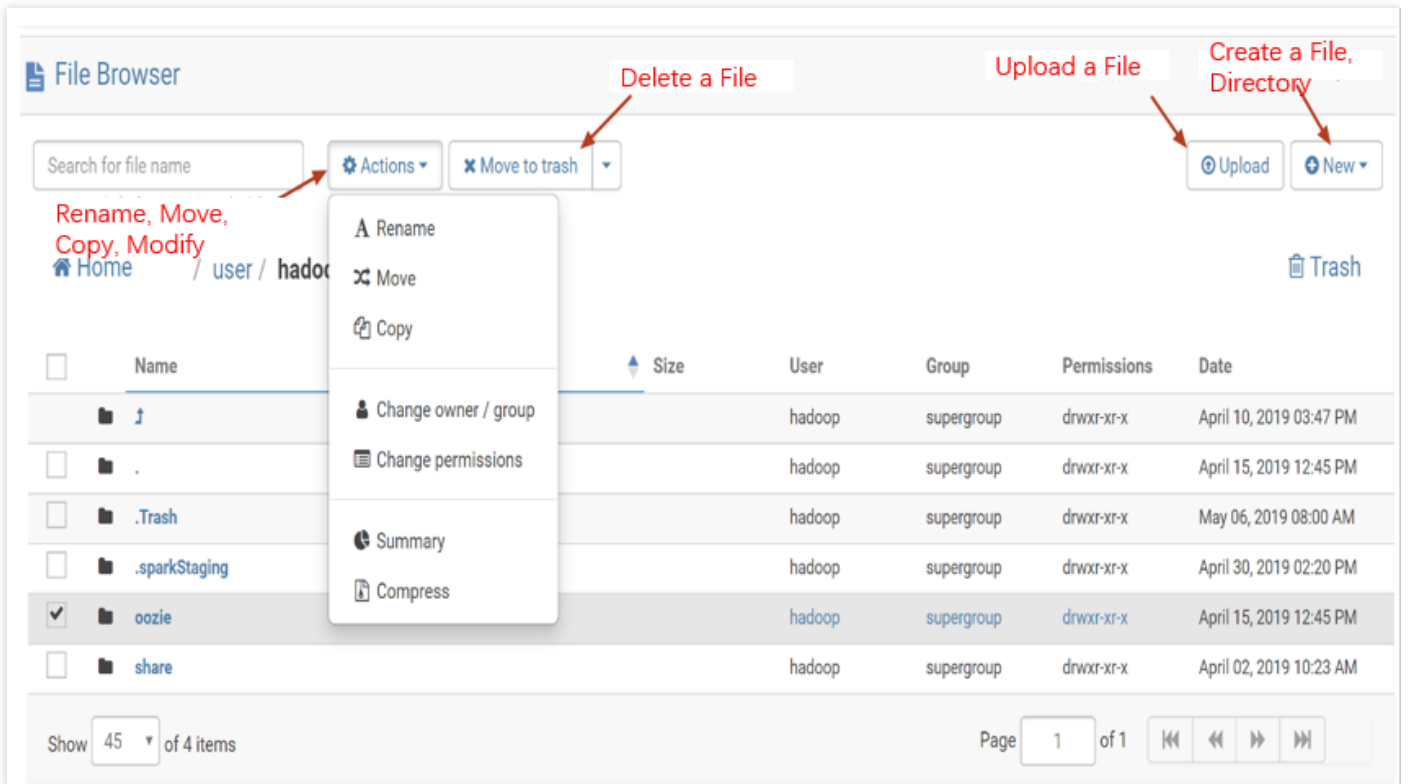
HDFS Access and File Browsing

Hue's web UI makes it easy to view files and folders in HDFS and perform operations such as creation, download, upload, copy, modification, and deletion.

1. On the left sidebar in the Hue console, select **Browsers** > **Files** to browse HDFS files.



2. Perform various operations.



Oozie Job Development

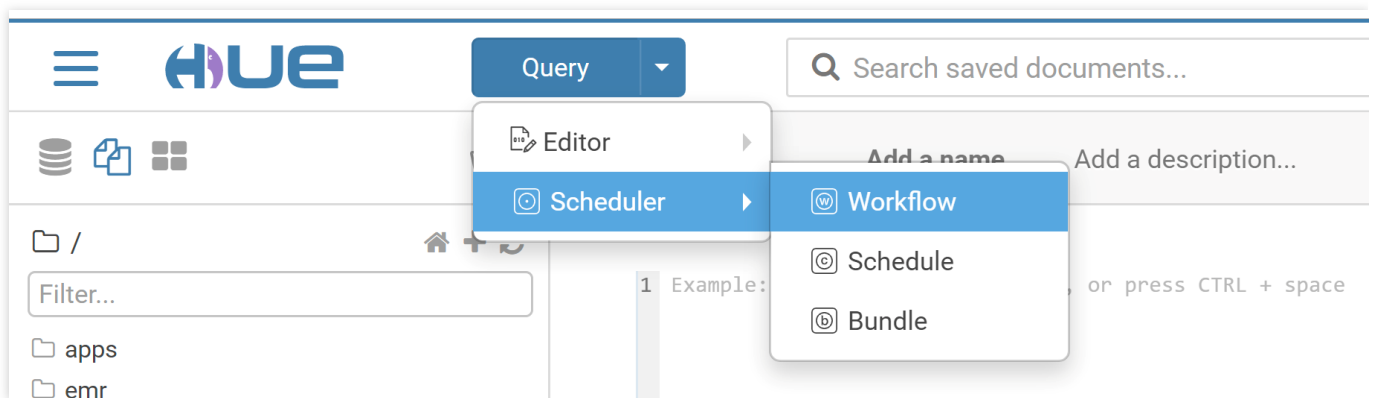
1. Prepare workflow data: Hue's job scheduling is based on workflows. First, create a workflow containing a Hive script with the following content:

```
create database if not exists hive_sample;
show databases;
use hive_sample;
show tables;
create table if not exists hive_sample (a int, b string);
show tables;
insert into hive_sample select 1, "a";
select * from hive_sample;
```

Save the above content as a file named `hive_sample.sql`. The Hive workflow also requires a `hive-site.xml` configuration file, which can be found on the cluster node where the Hive component is installed. The specific path is `/usr/local/service/hive/conf/hive-site.xml`. Copy the `hive-site.xml` file and then upload the Hive script file and `hive-site.xml` to a directory in HDFS, such as `/user/hadoop`.

2. Create a workflow.

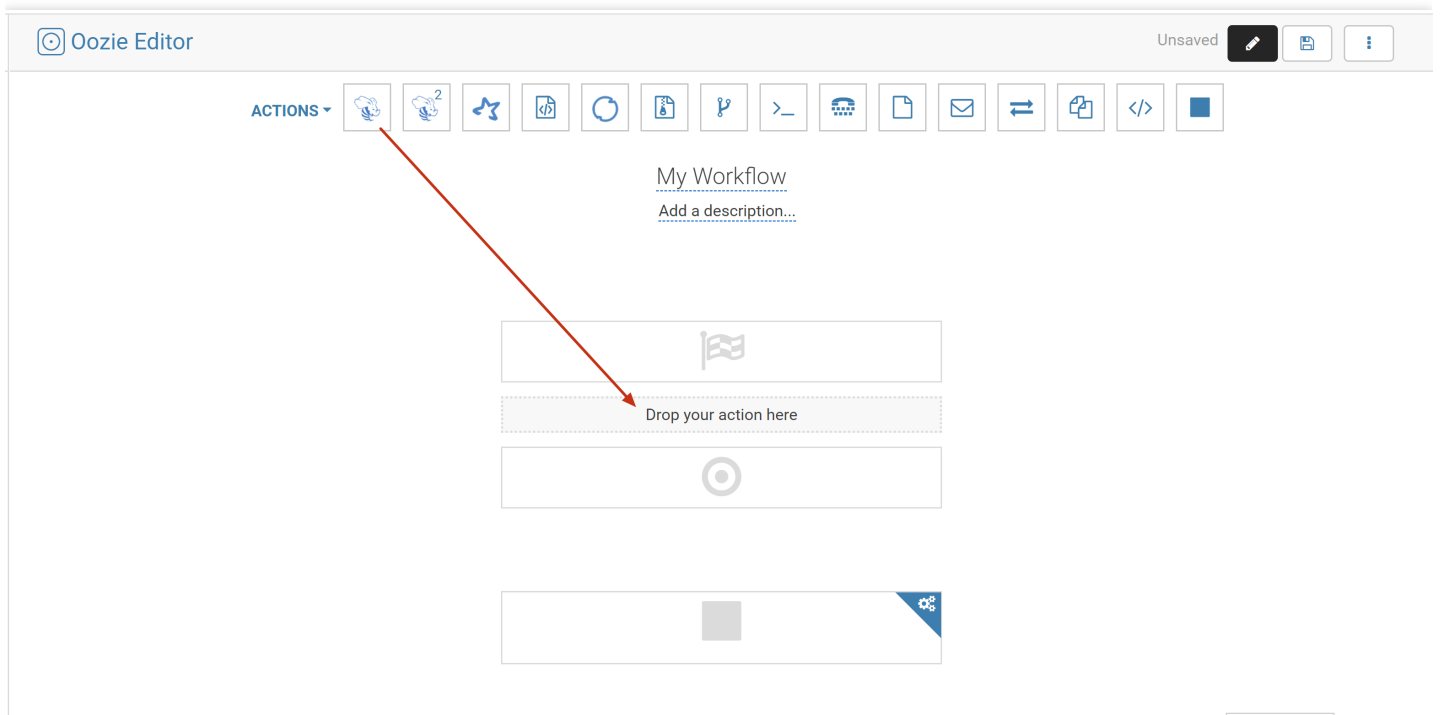
- i. Switch to the `hadoop` user. At the top of the Hue console, select **Query > Scheduler > Workflow**.



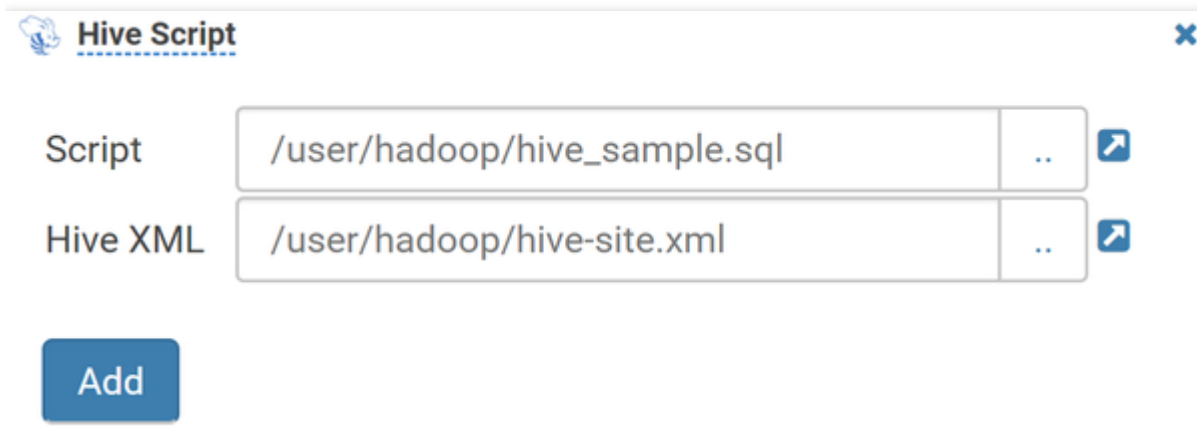
- ii. Drag a Hive script into the workflow editing page.

Note :

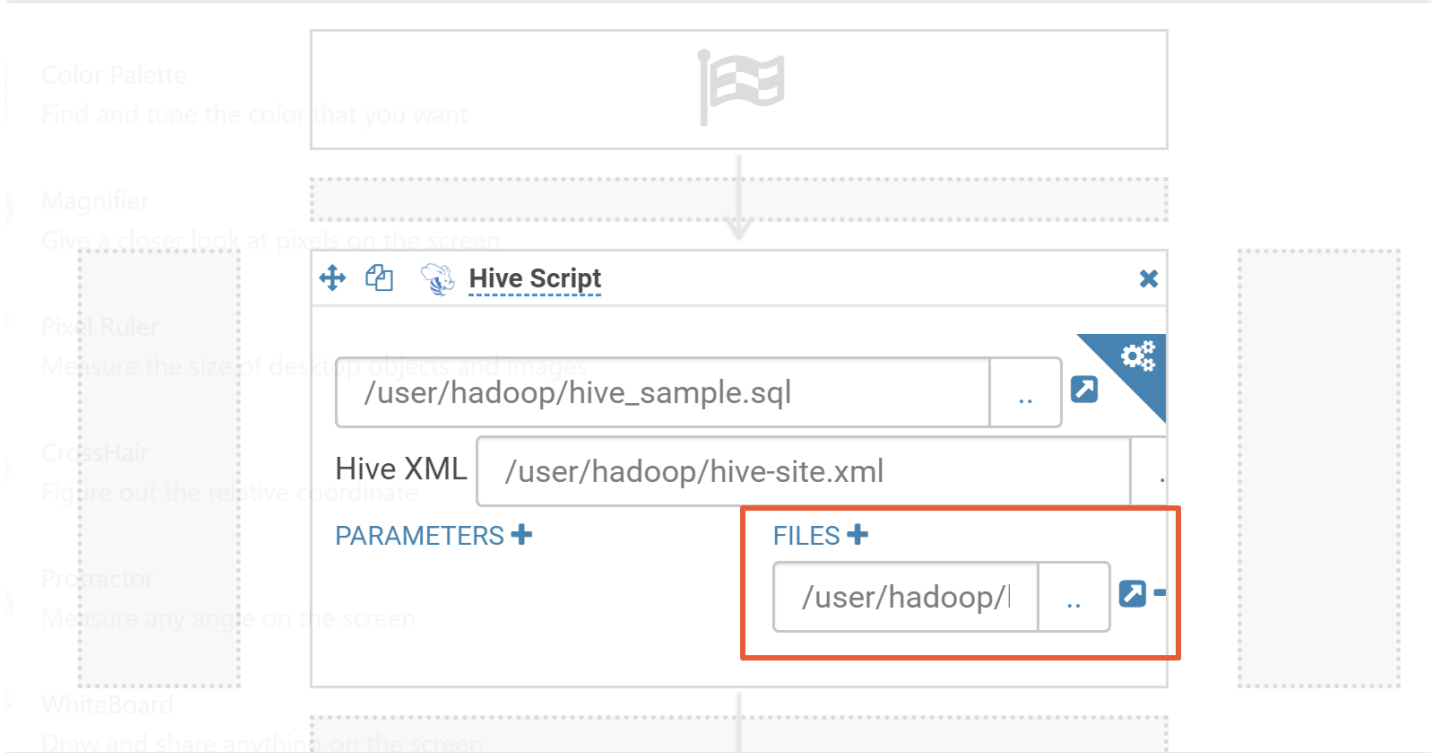
This document uses the installation of Hive v1 as an example, and the configuration parameter is `HiveServer1`. If it is deployed with other Hive versions (i.e., configuring configuration parameters of other versions), an error will be reported.



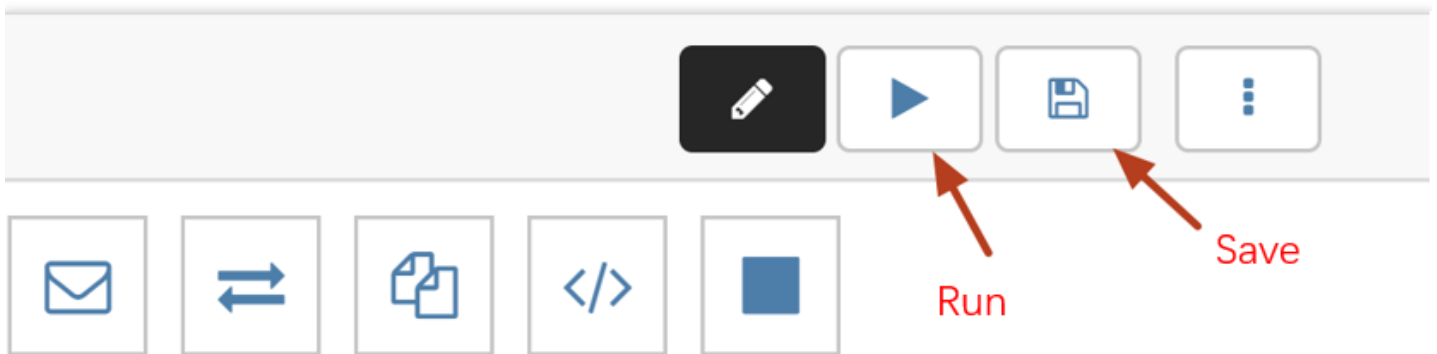
3. Select the Hive script and `hive-site.xml` files you just uploaded.



4. Click **Add** and specify the Hive script file in `FILES` .



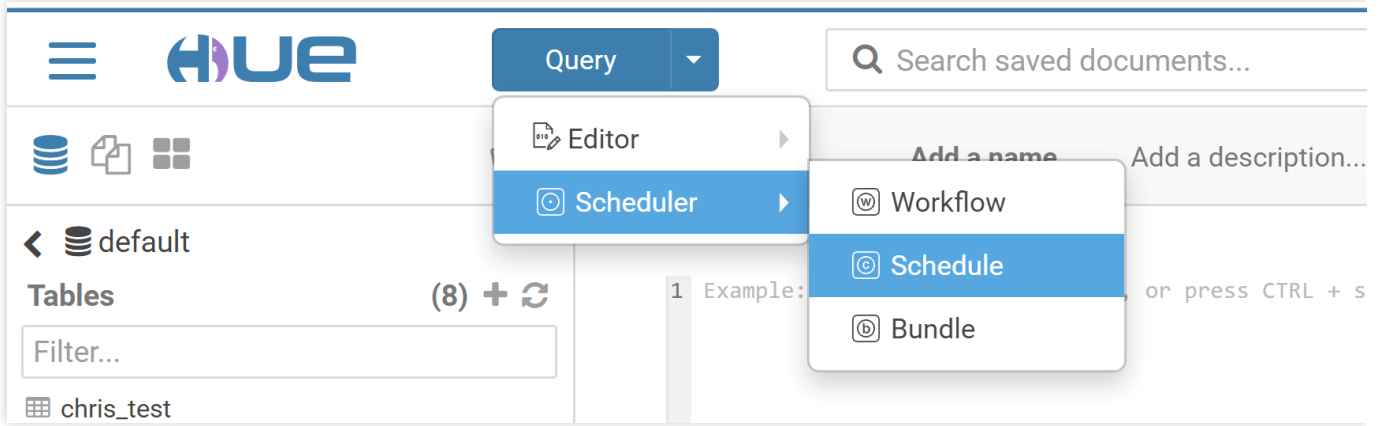
5. Click **Save** in the top-right corner and then click **Run** to run the workflow.



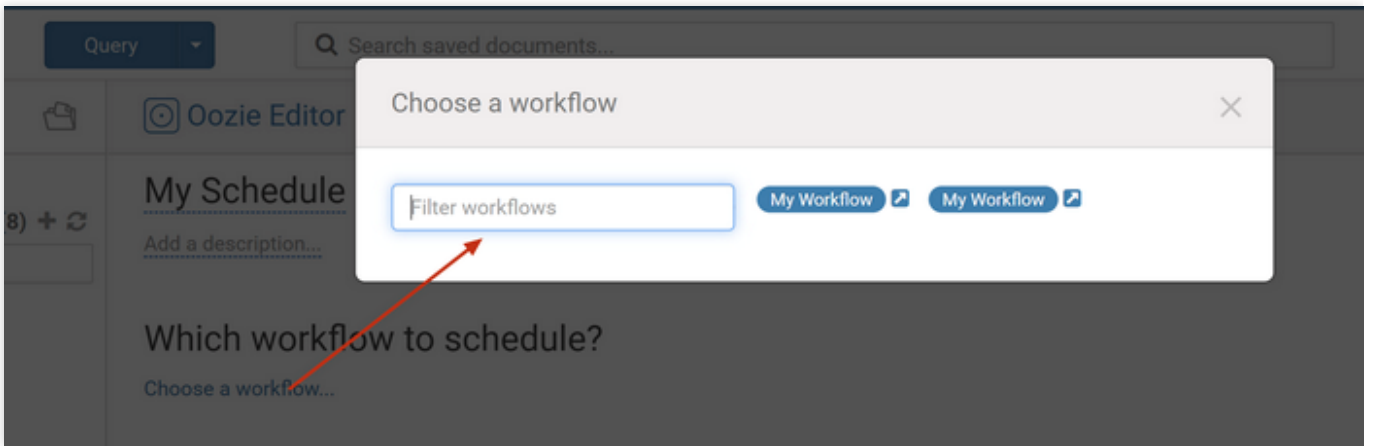
3. Create a scheduled job.

The scheduled job in Hive is "schedule", which is similar to the crontab in Linux. The supported scheduling granularity can be down to the minute level.

i. Select **Query** > **Scheduler** > **Schedule** to create a schedule.



ii. Click **Choose a workflow** to select a created workflow.



iii. Select the execution time, frequency, time zone, start time, and end time of the schedule and click **Save**.

Which workflow to schedule?

My Workflow

How often?

Every day at 17 : 35

Hide

Advanced syntax

Timezone Asia/Shanghai

From 2019-05-06 17:27

To 2019-05-13 17:27

Parameters

+ Add parameter

Save

Interval and Run Time of the Scheduling Task

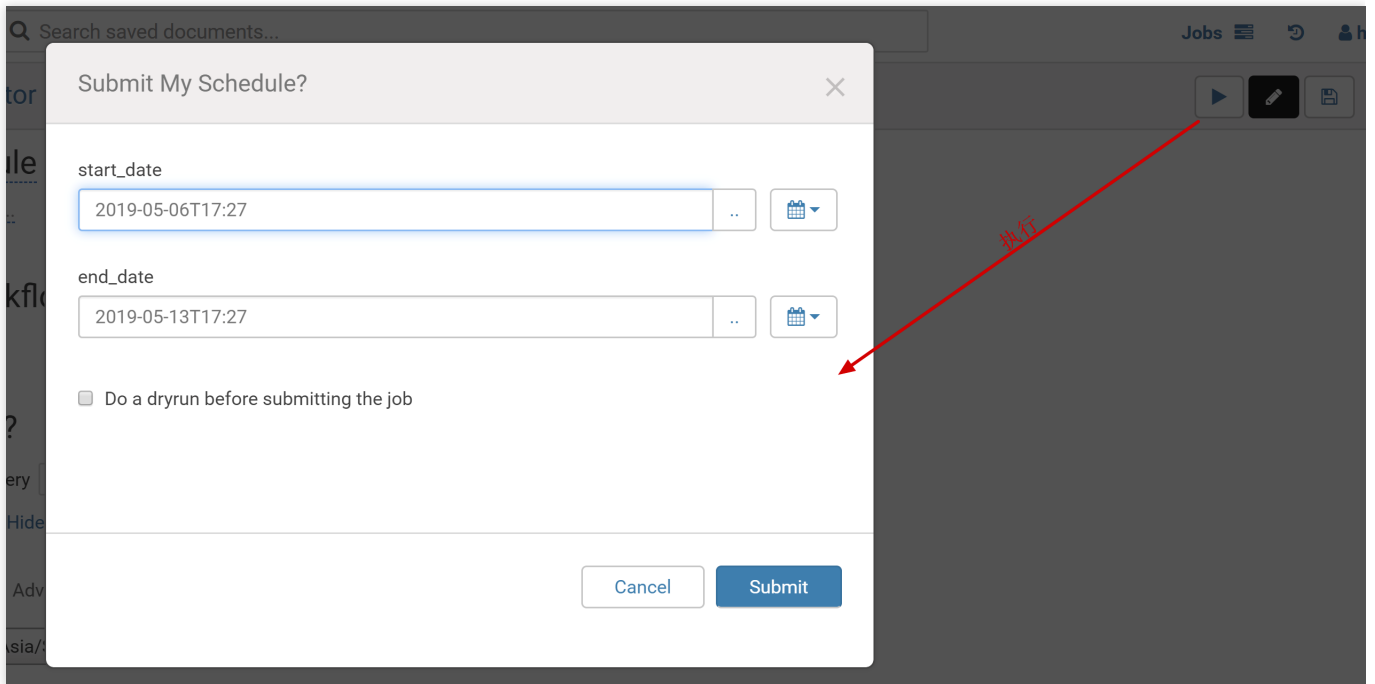
Time Zone

Run Time Range of the Scheduling Task

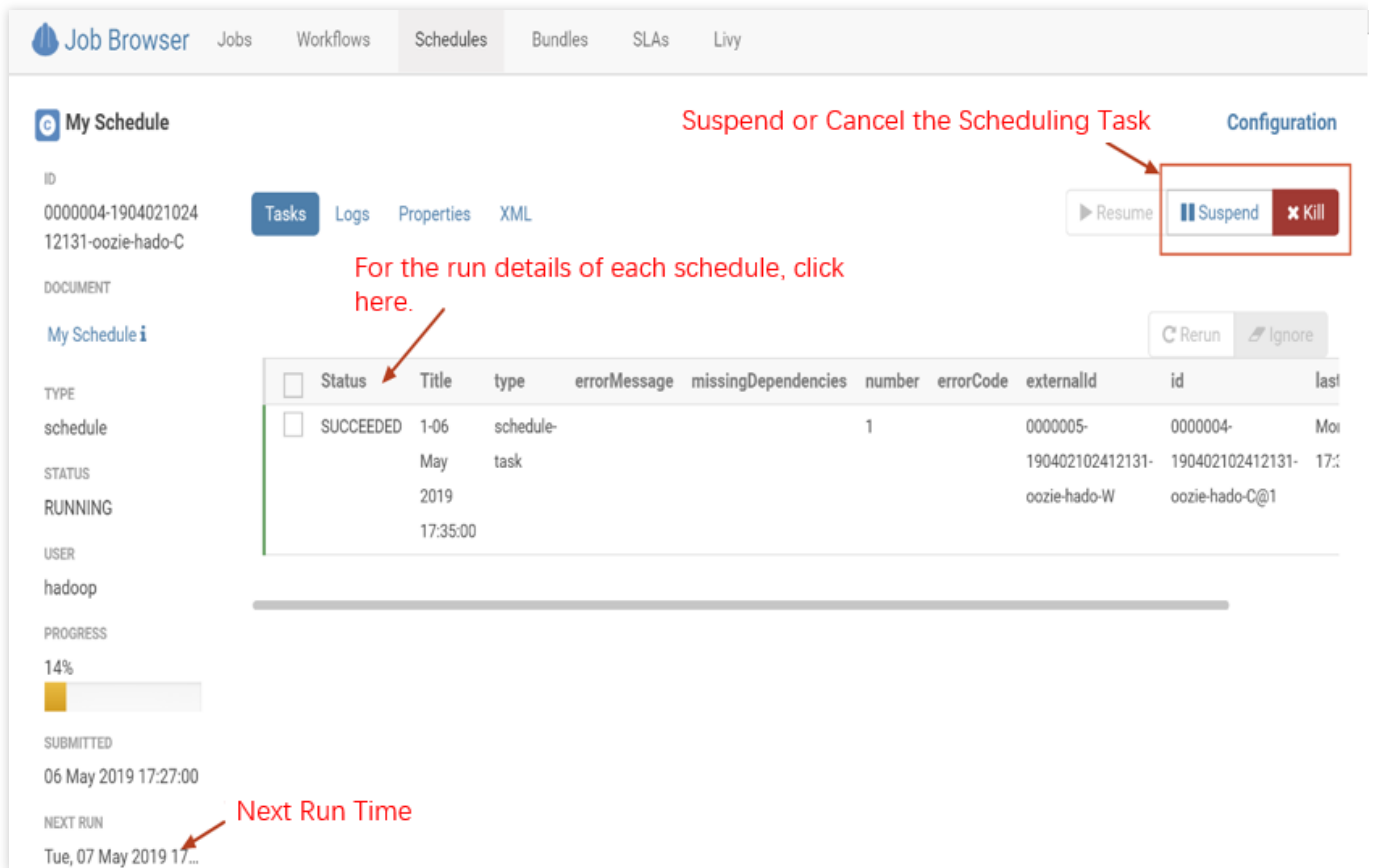
Save

4. Create a scheduled job.

i. Click **Submit** in the top-right corner to submit the schedule.



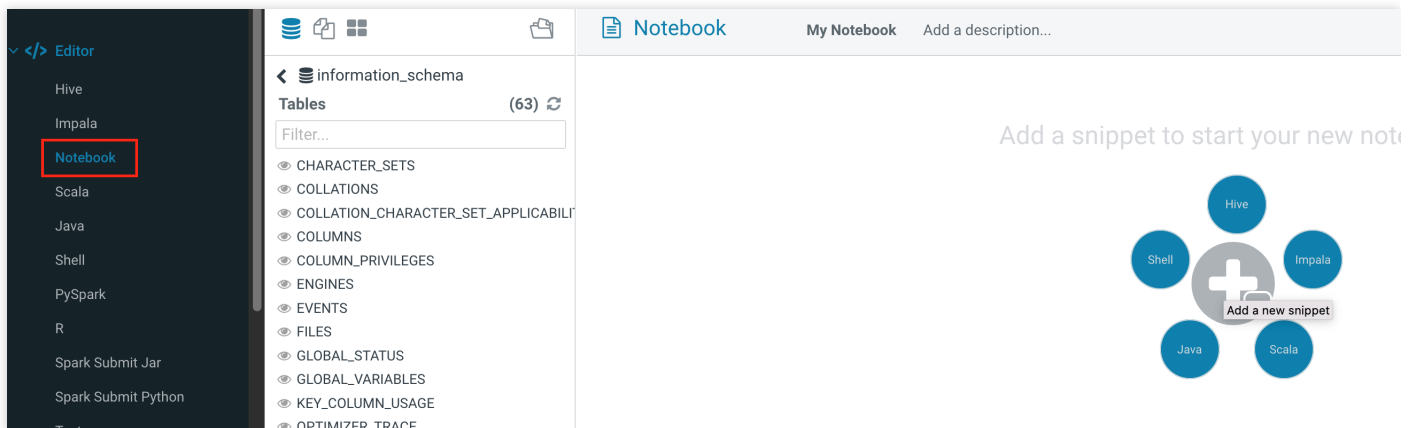
ii. You can view the scheduling status on the monitoring page of the schedulers.



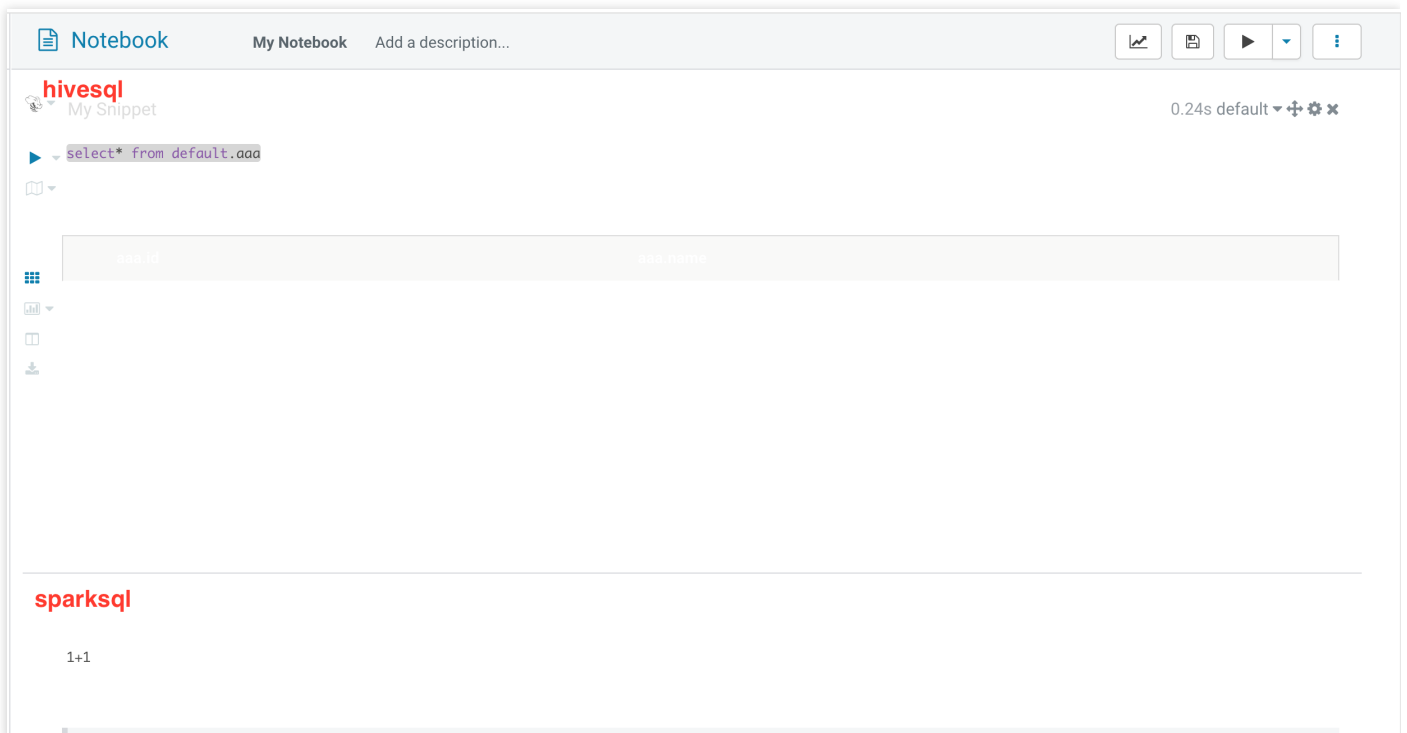
Notebook Query and Comparative Analysis

Notebooks can quickly build access requests and queries and put the query results together for comparative analysis. It supports five types: Hive, Impala, Spark, Java, and Shell.

1. Click **Editor**, **Notebook**, and **+** to add the required query.



2. Click **Save** to save the added notebook and click **Run** to run the entire notebook.



Oozie Development Guide

Last updated : 2022-05-16 12:52:25

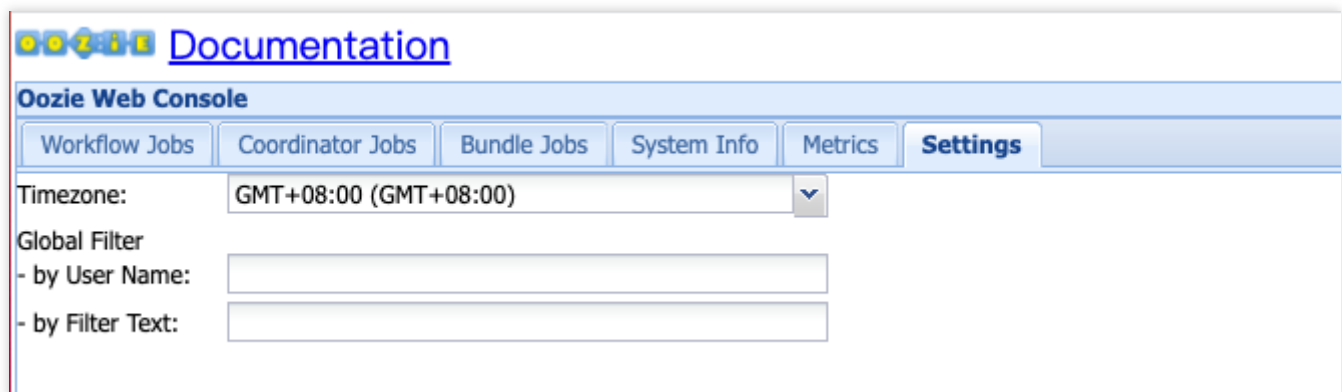
Apache Oozie is an open-source workflow engine. It is designed to orchestrate the tasks of Hadoop ecosystem components into workflows and then schedule, execute, and monitor them. This document briefly describes how to use Oozie in EMR. For detailed directions, visit the website. Here, we recommend you use Oozie through Hue's GUI as instructed in the Hue development documentation.

Prerequisites

You have created an EMR Hadoop cluster and selected the Oozie service. For more information, see [Creating EMR Cluster](#).

Accessing Oozie WebUI

- If you have enabled public network access for cluster nodes during cluster purchase, you can click the WebUI link in the EMR console for access.
- If you are in the Chinese mainland, we recommend you set the WebUI time zone to GMT+08:00.



Updating ShareLib

As the EMR cluster is preinstalled with ShareLib, you no longer need to install it when using Oozie to submit a workflow job. Of course, you can edit and update ShareLib as instructed below:

```
cd /usr/local/service/oozie
Add `tar -xf oozie-sharelib.tar.gz` to `bin/oozie-setup.sh sharelib create -fs hd
fs://active-namenode-ip:4007 -locallib shareoozie admin --oozie http://oozie-serv
```


`er-ip:12000/oozie -sharelibupdate` in the directory of the action to be supported in the `share` directory generated by decompressing the JAR package.`

Submitting Workflow in Non-Kerberos Environment

Decompress the `oozie-examples.tar.gz` file in the Oozie installation directory

`/usr/local/service/oozie` , which provides the sample workflows of the components supported by Oozie:

```
tar -xf oozie-examples.tar.gz
```

Take `action hive2` as an example:

- `su hadoop`.
- `cd examples/apps/hive2/`.
- Modify `job.properties` :
 - Set the value of `namenode` to the value of `fs.defaultFS` in `core-site.xml` .
 - Set the value of **resourceManager** to the value of `yarn.resourcemanager.ha.rm-ids` in `yarn-site.xml` in HA mode, or to the value of `yarn.resourcemanager.address` in non-HA mode.
 - The value of **jdbcURL** is `jdbc:hive2://hive2-server:7001/default` .
- `hadoop fs -put examples`.
- `oozie job -debug -oozie http://oozie-server-ip:12000/oozie -config examples/apps/hive2/job.properties -run`.
- `oozie job -info` the job ID returned in the previous step (or viewed on the WebUI).

Submitting Workflow in Kerberos Environment

Take `action hive2` as an example again. Check the README file in the `hive2` directory for other notes.

- `kinit -kt /var/krb5kdc/emr.keytab hadoop's principal && su hadoop`.
- `cd examples/apps/hive2/`.
- `mv job.properties.security job.properties && mv workflow.xml.security workflow.xml`.
- Modify `job.properties` :
 - Set the value of `namenode` to the value of `fs.defaultFS` in `core-site.xml` .
 - Set the value of **resourceManager** to the value of `yarn.resourcemanager.ha.rm-ids` in `yarn-site.xml` in HA mode, or to the value of `yarn.resourcemanager.address` in non-HA mode.
 - The value of `jdbcURL` is `jdbc:hive2://hive2-server:7001/default` .
 - The value of `jdbcPrincipal` is the value of `hive.server2.authentication.kerberos.principal` .

- `hadoop fs -put examples.`
- `oozie job -debug -oozie http://oozie-server-ip:12000/oozie -config examples/apps/hive2/job.properties -run.`
- `oozie job -info` the job ID returned in the previous step (or viewed on the WebUI).

Flume Development Guide

Flume Overview

Last updated : 2021-07-12 17:22:35

Flume Overview

Apache Flume is a highly available, distributed, and configurable tool/service that collects and aggregates large amounts of data such as logs and events from different sources. It is designed to collect data flows (e.g., log data) from various web servers and store them in centralized data storage system like HDFS and HBase.

Flume Architecture

A Flume event is defined as a unit of data flow. A Flume agent is a JVM process that contains the components required for completing a task. Among them, Source, Channel, and Sink are the core ones.



- **Source**

It consumes an event passed to it by an external source (e.g., web servers or other sources) and save it to one or more channels.

- **Channel**

Located between a source and a sink, a channel is used to cache incoming events. After the sink successfully sends the events to the channel at the next hop or the final destination, the events are removed from the channel.

- **Sink**

A sink is responsible for transferring the events to the next hop or final destination and removing them from the channel upon transfer completion.

Instructions

Preparations

- Create an EMR cluster. When [creating the EMR cluster](#), you need to select the Flume component on the software configuration page.
- Install Flume in the `/usr/local/service/flume` path on the core and task nodes (CVM instances) of the EMR cluster. The installation path for master nodes is `/usr/local/service/apps/`.

Configuring Flume

Go to the `/usr/local/service/flume` folder and create a file named `example.conf`.

```
[hadoop@10 /usr/local/service/flume]$ cd /usr/local/service/flume/  
[hadoop@10 /usr/local/service/flume]$ vim example.conf  
[hadoop@10 /usr/local/service/flume]$ |
```

```
# example.conf: A single-node Flume configuration  
  
# Name the components on this agent  
a1.sources = r1  
a1.sinks = k1  
a1.channels = c1  
  
# Describe/configure the source  
a1.sources.r1.type = netcat  
a1.sources.r1.bind = localhost  
a1.sources.r1.port = 44444  
  
# Describe the sink  
a1.sinks.k1.type = logger  
  
# Use a channel which buffers events in memory  
a1.channels.c1.type = memory  
a1.channels.c1.capacity = 1000  
a1.channels.c1.transactionCapacity = 100  
  
# Bind the source and sink to the channel  
a1.sources.r1.channels = c1  
a1.sinks.k1.channel = c1
```

Starting Flume

```
bin/flume-ng agent --conf conf --conf-file example.conf --name a1 -Dflume.root.logger=INFO,console
```

Configuring a test sample

After successful configuration, you will see the Flume agent started previously printing to the terminal.

```
telnet localhost 44444
Trying 127.0.0.1...
Connected to localhost.localdomain (127.0.0.1).
Escape character is '^]'.
Hello world! <ENTER>
OK
```

Storing Kafka Data in Hive Through Flume

Last updated : 2021-07-26 17:28:35

Scenario Description

Data in Kafka can be collected through Flume and stored in Hive.

Preparations for Development

- As this job requires access to CKafka, you need to create a CKafka instance first. For more information, please see [CKafka](#).
- Confirm that you have activated Tencent Cloud and created an EMR cluster. When creating the EMR cluster, you need to select the Spark component on the software configuration page.

Using the Kafka Toolkit in the EMR Cluster

First, you need to check the private IP and port number of CKafka. Log in to the CKafka Console, select the CKafka instance you want to use, and view its private IP as `$kafkaIP` in the basic information section, and the port number is generally 9092 by default. Create a topic named `kafka_test` on the topic management page.

Configuring Flume

1. Create the Flume configuration file `hive_kafka.properties`.

```
vim hive_kafka.properties
agent.sources = kafka_source
agent.channels = mem_channel
agent.sinks = hive_sink
# The following code is used to configure the source
agent.sources.kafka_source.type = org.apache.flume.source.kafka.KafkaSource
agent.sources.kafka_source.channels = mem_channel
agent.sources.kafka_source.batchSize = 5000
agent.sources.kafka_source.kafka.bootstrap.servers = $kafkaIP:9092
agent.sources.kafka_source.kafka.topics = kafka_test
# The following code is used to configure the sink
agent.sinks.hive_sink.channel = mem_channel
agent.sinks.hive_sink.type = hive
```

```
agent.sinks.hive_sink.hive.metastore = thrift://172.16.32.51:7004
agent.sinks.hive_sink.hive.database = default
agent.sinks.hive_sink.hive.table = weblog
agent.sinks.hive_sink.hive.partition = asia,india,%y-%m-%d-%H-%M
agent.sinks.hive_sink.useLocalTimeStamp = true
agent.sinks.hive_sink.round = true
agent.sinks.hive_sink.roundValue = 10
agent.sinks.hive_sink.roundUnit = minute
agent.sinks.hive_sink.serializer = DELIMITED
agent.sinks.hive_sink.serializer.delimiter = ","
agent.sinks.hive_sink.serializer.serdeSeparator = ','
agent.sinks.hive_sink.serializer.fieldnames =id,msg
# The following code is used to configure the channel
agent.channels.mem_channel.type = memory
agent.channels.mem_channel.capacity = 100000
agent.channels.mem_channel.transactionCapacity = 100000
```

You can confirm Hive Metastore in the following way:

```
grep "hive.metastore.uris" -C 2 /usr/local/service/hive/conf/hive-site.xml

<property>
<name>hive.metastore.uris</name>
<value>thrift://172.16.32.51:7004</value>
</property>
```

2. Create a Hive table.

```
create table weblog ( id int , msg string )
partitioned by (continent string, country string, time string)
clustered by (id) into 5 buckets
stored as orc TBLPROPERTIES ('transactional'='true');
```

Note :

All the following conditions must be met: it must be a table with partitions and buckets, the storage format is ORC, and `TBLPROPERTIES ('transactional'='true')` is set.

3. Enable the Hive transaction.

In the console, add the following configuration items to `hive-site.xml` .

```
<property>
<name>hive.support.concurrency</name>
<value>>true</value>
```

```
</property>
<property>
<name>hive.exec.dynamic.partition.mode</name>
<value>nonstrict</value>
</property>
<property>
<name>hive.txn.manager</name>
<value>org.apache.hadoop.hive.ql.lockmgr.DbTxnManager</value>
</property>
<property>
<name>hive.compactor.initiator.on</name>
<value>>true</value>
</property>
<property>
<name>hive.compactor.worker.threads</name>
<value>1</value>
</property>
<property>
<name>hive.enforce.bucketing</name>
<value>>true</value>
</property>
```

Note :

After the configuration is distributed and Hive is restarted, the `hadoop-hive` log will prompt that the Metastore cannot be connected to. Please ignore this error. Because of the startup order of the processes, Metastore will be started before HiveServer2.

4. Copy `hive-hcatalog-streaming-xxx.jar` of Hive to the `lib` directory of Flume.

```
cp -ra /usr/local/service/hive/hcatalog/share/hcatalog/hive-hcatalog-streaming-2.3.3.jar /usr/local/service/flume/lib/
```

5. Run Flume.

```
./bin/flume-ng agent --conf ./conf/ -f hive_kafka.properties -n agent -Dflume.root.logger=INFO,console
```

6. Run the Kafka producer.

```
[hadoop@172 kafka]$ ./bin/kafka-console-producer.sh --broker-list $kafkaIP:9092 --topic kafka_test
1,hello
2,hi
```


Test

- Enter information on the client of the Kafka producer and press Enter.
- Check whether there is corresponding data in the Hive table.

Reference Documentation

- [Hive Sink Configuration Description](#)
- [Hive Log Configuration Description](#)

Storing Kafka Data in HDFS or COS Through Flume

Last updated : 2021-07-08 10:43:44

Scenario Description

Collecting and saving the data in Kafka to HDFS or COS via Flume

Development Preparations

- This task requires access to CKafka, so you need to create a CKafka instance first. For more information, see [Message Queue CKafka](#).
- Create an EMR cluster. When creating the EMR cluster, you need to select the Spark component on the software configuration page and enable access to COS on the basic configuration page.

Using the Kafka Toolkit in the EMR Cluster

First, you need to check the private IP and port number of CKafka. Log in to the CKafka console, select the CKafka instance you want to use, and find the private IP (`$kafkaIP`) and port number (generally `9092`) in the basic information section. Create a topic named `kafka_test` on the topic management page.

Configuring Flume

1. Create a Flume configuration file `kafka.properties` .

```
vim kafka.properties
agent.sources = kafka_source
agent.channels = mem_channel
agent.sinks = hdfs_sink
# The following is for source configuration.
agent.sources.kafka_source.type = org.apache.flume.source.kafka.KafkaSource
agent.sources.kafka_source.channels = mem_channel
agent.sources.kafka_source.batchSize = 5000
agent.sources.kafka_source.kafka.bootstrap.servers = $kafkaIP:9092
agent.sources.kafka_source.kafka.topics = kafka_test
```

```
# The following is for sink configuration.
agent.sinks.hdfs_sink.type = hdfs
agent.sinks.hdfs_sink.channel = mem_channel
agent.sinks.hdfs_sink.hdfs.path = /data/flume/kafka/%Y%m%d (or cosn://bucket/xx
x)
agent.sinks.hdfs_sink.hdfs.rollSize = 0
agent.sinks.hdfs_sink.hdfs.rollCount = 0
agent.sinks.hdfs_sink.hdfs.rollInterval = 3600
agent.sinks.hdfs_sink.hdfs.threadsPoolSize = 30
agent.sinks.hdfs_sink.hdfs.fileType=DataStream
agent.sinks.hdfs_sink.hdfs.useLocalTimeStamp=true
agent.sinks.hdfs_sink.hdfs.writeFormat=Text
# The following is for channel configuration.
agent.channels.mem_channel.type = memory
agent.channels.mem_channel.capacity = 100000
agent.channels.mem_channel.transactionCapacity = 10000
```

2. Run Flume.

```
./bin/flume-ng agent --conf ./conf/ -f kafka.properties -n agent -Dflume.root.l
ogger=INFO,console
```

3. Run Kafka producer.

```
[hadoop@172 kafka]$ ./bin/kafka-console-producer.sh --broker-list $kafkaIP:9092
--topic kafka_test
test
hello
```

Testing

- Enter information on the Kafka producer client and press Enter.
- Check whether the corresponding directory and file `hadoop fs -ls /data/flume/kafka/` have been generated in HDFS.

References

[Kafka Source Configuration Description](#)

Storing Kafka Data in Hive Through Flume

Last updated : 2020-02-28 20:02:33

Scenario Description

Data in Kafka can be collected through Flume and stored in HBase.

Preparations for Development

- As this job requires access to CKafka, you need to create a CKafka instance first. For more information, please see [CKafka](#).
- Confirm that you have activated Tencent Cloud and created an EMR cluster. When creating the EMR cluster, you need to select the Spark component on the software configuration page.

Using the Kafka Toolkit in the EMR Cluster

First, you need to check the private IP and port number of CKafka. Log in to the CKafka Console, select the CKafka instance you want to use, and view its private IP as `$kafkaIP` in the basic information section, and the port number is generally 9092 by default. Create a topic named `kafka_test` on the topic management page.

Configuring Flume

1. Create the Flume configuration file `hbase_kafka.properties` .

```
vim hbase_kafka.properties
agent.sources = kafka_source
agent.channels = mem_channel
agent.sinks = hbase_sink
# The following code is used to configure the source
agent.sources.kafka_source.type = org.apache.flume.source.kafka.KafkaSource
agent.sources.kafka_source.channels = mem_channel
agent.sources.kafka_source.batchSize = 5000
agent.sources.kafka_source.kafka.bootstrap.servers = $kafkaIP:9092
agent.sources.kafka_source.kafka.topics = kafka_test
# The following code is used to configure the sink
agent.sinks.hbase_sink.channel = mem_channel
agent.sinks.hbase_sink.table = foo_table
```

```
agent.sinks.hbase_sink.columnFamily = cf
agent.sinks.hbase_sink.serializer = org.apache.flume.sink.hbase.RegexHbaseEvent
Serializer
# The following code is used to configure the channel
agent.channels.mem_channel.type = memory
agent.channels.mem_channel.capacity = 100000
agent.channels.mem_channel.transactionCapacity = 10000
```

2. Create an HBase table.

```
hbase shell
create 'foo_table','cf'
```

3. Run Flume.

```
./bin/flume-ng agent --conf ./conf/ -f hbase_kafka.properties -n agent -Dflume.
root.logger=INFO,console
```

4. Run the Kafka producer.

```
[hadoop@172 kafka]$ ./bin/kafka-console-producer.sh --broker-list $kafkaIP:9092
--topic kafka_test
hello
hbase_test
```

Test

- Enter information on the client of the Kafka producer and press Enter.
- Check whether there is corresponding data in the HBase table.

Reference Documentation

[HBaseSink Configuration Description](#)

Kerberos Development Guide

Kerberos HA Support

Last updated : 2020-06-01 17:04:46

For a formal production system with Kerberos enabled, you should also consider the high availability (HA) of KDC. The Kerberos service can be configured as master/slave mode where data is synced from the master node to the slave node through the kprop service. After an EMR HA cluster is purchased, Kerberos is highly available by default with no additional configuration required.

This document describes how to configure and use HA for the Kerberos service.

Prerequisites

- You have purchased an EMR HA cluster.
- You enabled Kerberos authentication when purchasing the cluster.

KDC HA Configuration Overview

Configure the `/etc/krb5.conf` file as follows:

Two KDC addresses are configured in the example: `active kdc` and `backup kdc`, so as to ensure that when either KDC service is exceptional, normal KDC service can still be provided for the cluster.

```
[libdefaults]
dns_lookup_realm = false
ticket_lifetime = 24h
renew_lifetime = 7d
forwardable = true
rdns = false
default_realm = REALM
default_tgs_enctypes = des3-cbc-sha1
default_tkt_enctypes = des3-cbc-sha1
permitted_enctypes = des3-cbc-sha1

[realms]
REALM = {
```

```
kdc = active_kdc:88
admin_server = active_kdc
kdc = backup_kdc:88
admin_server = backup_kdc
}

[domain_realm]
# .example.com = EXAMPLE.COM
```

KDC Data Sync

- kprop configuration

There is a `/var/kerberos/krb5kdc/kpropd.acl` configuration file on both the KDC servers by default.

```
host/active_kdc@REALM
host/backup_kdc@REALM
```

2. Execute the `/var/kerberos/krb5kdc/kpropd.acl` file.

After executing the default configuration file `/var/kerberos/krb5kdc/kpropd.acl`, a `/etc/krb5.keytab` file will be generated on both the KDC servers. At the same time, the two KDC servers will also start the kprop service.

```
[root@10 krb5kdc]# service kprop status
Redirecting to /bin/systemctl status kprop.service
kprop.service - Kerberos 5 Propagation
Loaded: loaded (/usr/lib/systemd/system/kprop.service; disabled; vendor preset: disabled)
Active: active (running) since Thu 2020-05-07 15:33:35 CST; 1h 9min ago
Process: 3752 ExecStart=/usr/sbin/_kpropd $KPROPD_ARGS (code=exited, status=0/SUCCESS)
Main PID: 3753 (kpropd)
CGroup: /system.slice/kprop.service
└─3753 /usr/sbin/kpropd
```

3. Run the regular KDC data sync command

The EMR agent will regularly (once every five minutes by default) sync the `principals` of `active kdc` to `backup kdc` to ensure that the data on the two KDC servers are in sync. The following sync command will be executed regularly on `active kdc`:

```
kdb5_util dump /var/kerberos/krb5kdc/master.dump & kprop -f /var/kerberos/krb5kdc/master.dump -d -P 754 backup_kdc
```

Result Verification

1. Run the following command on `active kdc` .

```
kadmin.local "-q addprinc -randkey test"
```

2. You can see the newly added `principal` on `backup kdc` , indicating that the data sync between the two KDC servers has been completed. This operation takes a while (five minutes by default).

```
kadmin.local "-q listprincs"|grep test
```


Kerberos Overview

Last updated : 2020-08-03 11:37:20

Currently, only EMR v2.1.0 supports the creation of secure clusters, i.e., the open-source components in the cluster are launched in Kerberos secure mode. In this security environment, only authenticated clients can access the services (such as HDFS) of the cluster.

Below lists the components in EMR v2.1.0 that currently supports Kerberos.

Component Name	Version
Hadoop	2.8.4
Hbase	1.3.1
Hive	2.3.3
Hue	4.4.0
Ooize	4.3.1
Presto	0.7.1
Zookeeper	3.4.9

Important Concepts

- **KDC**

Full name: Key distribution center

Role: Generating and managing tickets in the entire authentication process, including two services: AS and TGS.

- **AS**

Full name: Authentication service

Role: Generating TGTs for a client.

- **TGS**

Name: Ticket granting service

Role: Granting a client tickets for a specified service.

- **AD**

Name: Account database

Role: Storing the allowlist of all clients, and only clients in the allowlist can successfully apply for TGTs.

- **TGT**

Name: Ticket-granting ticket

Role: A ticket used to obtaining tickets.

- **client**

A client that wants to access a server.

- **server**

A server that provide a service for a certain business.

Other Concepts

- **principal**

A subject of authentication, i.e., the username.

- **realm**

Realm is a little bit like a namespace in programming languages. In programming languages, a variable name only makes sense in a namespace. Similarly, a principal only makes sense in a realm. Generally, a realm can be seen as a "container" or "space" of a principal.

Correspondingly, the naming rule for principals is `what_name_you_like@realm`.

In Kerberos, it is customary to use uppercase letters to name a realm, such as `EXAMPLE.COM`.

- **password**

A password of a user, corresponding to a `master_key` in Kerberos. It can be stored in a keytab file; therefore, in all scenarios that require passwords in Kerberos, a keytab can be used as the input.

- **credential**

A credential is a proof that "proves that someone is truly of the claimed identity or that something can really happen". The specific meaning of the credential varies slightly by usage scenario:

- For an individual principal, a credential is a password.
- In the Kerberos authentication process, a credential means a variety of tickets.

Authentication Process

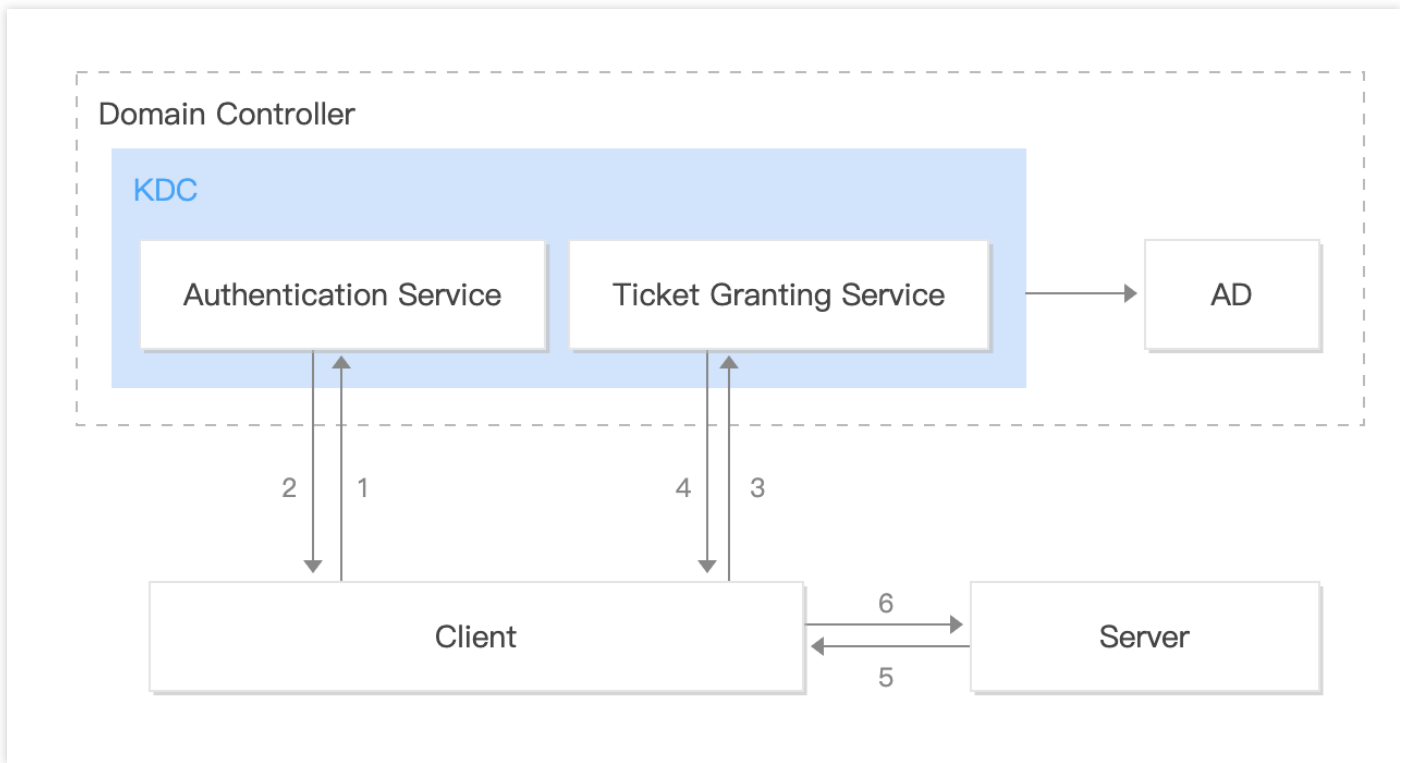
During the process of accessing a server by a client, to ensure that both the client and the server are reliable, it is necessary to introduce a third-party authentication platform. Therefore, two services, AS and TGS, are designed. They are usually in the same service process and provided by a KDC for the implementation of MIT Kerberos.

The authentication process is divided into the following three steps:

1. A client requests access to a server from the Kerberos service. Kerberos first determines whether the client is trustworthy by checking whether it is in the blocklist and allowlist stored in the AD. After success, the AS returns a

TGT to the client.

2. After receiving the TGT, the client continues to request access to the server from Kerberos. Kerberos uses the TGT in the message submitted by the client to determine that the client has the permissions, and then grants the server access ticket to the client.
3. After receiving the ticket, the client can access the server, but the ticket is only for the specified server. To access other servers, the client needs to apply to the TGS again.



Kerberos Use Instructions

Last updated : 2021-02-19 17:24:30

This document uses MIT's Kerberos as the KDC service and assumes that KDC has been properly installed and started. To use Kerberos, create a realm, add the principals of relevant roles (including server and client), and generate a keytab file.

Creating a Database

Run the `kdb5_util` command to create a database for storing information about the principals.

```
kdb5_util -r EXAMPLE.COM create -s
Initializing database '/var/krb5/principal' for realm 'EXAMPLE.COM'
master key name 'K/M@EXAMPLE.COM'
You will be prompted for the database Master Password.
It is important that you NOT FORGET this password.
Enter KDC database master key: <Type the key>
Re-enter KDC database master key to verify: <Type it again>
```

Adding a Principal

```
kadmin.local
kadmin.local: add_principal -pw testpassword test/host@EXAMPLE.COM

WARNING: no policy specified fortest/host@EXAMPLE.COM; defaulting to no policy
Principal "test/host@EXAMPLE.COM" created.
```

Generating a Keytab File

```
kadmin.local
kadmin.local: ktadd -k /var/krb5kdc/test.keytab test/host@EXAMPLE.COM

Entry for principal test/host@EXAMPLE.COM with kvno 2, encryption type des3-cbc-sha1 added to keytab WRFILE:/var/krb5kdc/test.keytab.
```

Here, we created a user `test/host@EXAMPLE.COM` and put the key of this user into the file `/var/krb5kdc/test.keytab`.

Starting KDC

```
service krb5-kdc start
* Starting Kerberos KDC krb5kdc
```

Performing kinit Authentication

```
kinit -k -t /etc/krb5.keytab test-client/host@EXAMPLE.COM
```

`kinit` is used to obtain a TGT from KDC. It sends a request to the KDC server specified in `/etc/krb5.conf`. If the TGT is successfully obtained, you can see it by using `klist`.

```
klist
Ticket cache: FILE:/tmp/krb5cc_1000
Default principal: test-client/host@EXAMPLE.COM

Valid starting Expires Service principal
2019-01-15T17:50:25 2019-01-16T17:50:25 krbtgt/EXAMPLE.COM@EXAMPLE.COM
renew until 2019-01-16T00:00:25
```

Using in a Project

After the `kinit` authentication succeeds, you can copy the keytab file to the server and client you need to use and configure the corresponding principals to use them.

Accessing Hadoop Secure Cluster

Last updated : 2020-10-13 15:45:56

Hadoop Commands

Ticket not obtained

If Kerberos is enabled, you need to obtain a ticket before running a Hadoop command.

Otherwise, the following error message will appear:

```
hadoop fs -ls /
19/04/19 19:59:03 WARN ipc.Client: Exception encountered while connecting to the
server : javax.security.sasl.SaslException: GSS initiate failed [Caused by GSSExc
eption: No valid credentials provided (Mechanism level: Failed to find any Kerber
os tgt)]
ls: Failed on local exception: java.io.IOException: javax.security.sasl.SaslExcep
tion: GSS initiate failed [Caused by GSSExcption: No valid credentials provided
(Mechanism level: Failed to find any Kerberos tgt)]; Host Details : local host is
: "172.30.0.27/172.30.0.27"; destination host is: "172.30.0.27":4007;
```

Obtaining a ticket

Prerequisites: The principal of hadoop@EMR has been added.

```
kinit -kt /var/krb5kdc/emr.keytab hadoop@EMR
```

Run the command to access normally.

```
hadoop fs -ls /

Found 8 items
-rw-r--r-- 3 hadoop supergroup 3809 2019-03-06 11:10 /README.md
drwxr-xr-x - hadoop supergroup 0 2019-01-14 21:43 /apps
drwxrwx--- - hadoop supergroup 0 2019-01-17 19:46 /emr
drwxr-xr-x - hadoop supergroup 0 2019-01-23 20:02 /hbase
drwxr-xr-x - hadoop supergroup 0 2019-03-07 11:10 /spark-history
drwx-wx-wx - hadoop supergroup 0 2019-03-06 20:23 /tmp
drwxr-xr-x - hadoop supergroup 0 2019-01-17 14:43 /user
drwxr-xr-x - hadoop supergroup 0 2019-01-17 19:43 /usr
```

Accessing HDFS Through Java Code

Using a local ticket

⚠ Note :

You need to run kinit in advance to obtain a ticket. After the ticket expires, the program cannot be accessed.

```
public static void main(String[] args) throws IOException {
    Configuration conf = new Configuration();
    conf.addResource(new Path("/usr/local/service/hadoop/etc/hadoop/hdfs-site.xml"));
    conf.addResource(new Path("/usr/local/service/hadoop/etc/hadoop/core-site.xml"));
    UserGroupInformation.setConfiguration(conf);
    UserGroupInformation.loginUserFromSubject(null);
    FileSystem fileSystem = FileSystem.get(conf);
    FileStatus[] fileStatus = fileSystem.listStatus(new Path("/"));
    for (int i = 0; i < fileStatus.length; i++) {
        System.out.println(fileStatus[i].getPath());
    }
}
```

Using a keytab file

```
public static void main(String[] args) throws IOException {
    Configuration conf = new Configuration();
    conf.addResource(new Path("/usr/local/service/hadoop/etc/hadoop/hdfs-site.xml"));
    conf.addResource(new Path("/usr/local/service/hadoop/etc/hadoop/core-site.xml"));
    UserGroupInformation.setConfiguration(conf);
    UserGroupInformation.loginUserFromKeytab("hadoop@EMR", "/var/krb5kdc/emr.keytab");
    FileSystem fileSystem = FileSystem.get(conf);
    FileStatus[] fileStatus = fileSystem.listStatus(new Path("/"));
    for (int i = 0; i < fileStatus.length; i++) {
        System.out.println(fileStatus[i].getPath());
    }
}
```

Sample Connection from Hadoop to Kerberos

Last updated : 2020-10-10 15:57:13

This document describes how to modify the Hadoop configuration to enable it to access Kerberos. For secure clusters purchased through EMR, the required settings are already automatically configured by the system.

Prerequisites

- The KDC service has been set up successfully.
- The Hadoop-related principals have been created.
- The keytab file has been distributed to each server (assuming that its path is `/var/krb5kdc/emr.keytab`).

Accessing Kerberos by Hadoop

Hadoop mainly contains HDFS and Yarn services. You need to modify their configurations separately and restart the service processes.

HDFS access

Modifying core-site.xml

```
hadoop.security.authentication= kerberos
hadoop.security.authorization= true
```

Modifying hdfs-site.xml

```
dfs.namenode.kerberos.principal= hadoop/_HOST@EMR
dfs.namenode.keytab.file= /var/krb5kdc/emr.keytab
dfs.namenode.kerberos.internal.spnego.principal= HTTP/_HOST@EMR
dfs.secondary.namenode.kerberos.principal= hadoop/_HOST@EMR
dfs.secondary.namenode.keytab.file= /var/krb5kdc/emr.keytab
dfs.secondary.namenode.kerberos.internal.spnego.principal= HTTP/_HOST@EMR
dfs.journalnode.kerberos.principal= hadoop/_HOST@EMR
dfs.journalnode.keytab.file= /var/krb5kdc/emr.keytab
dfs.journalnode.kerberos.internal.spnego.principal= HTTP/_HOST@EMR
dfs.datanode.kerberos.principal= hadoop/_HOST@EMR
dfs.datanode.keytab.file= /var/krb5kdc/emr.keytab
dfs.datanode.data.dir.perm= 700
dfs.web.authentication.kerberos.keytab= /var/krb5kdc/emr.keytab
```



```
dfs.web.authentication.kerberos.principal= HTTP/_HOST@EMR
ignore.secure.ports.for.testing= true
```

⚠ Note :

The `ignore.secure.ports.for.testing` option must be set to true; otherwise, the sasl mode has to be configured, and webhdfs has to have HTTPS enabled.

Modifying httpfs-site.xml (If httpfs Is Enabled)

```
httpfs.authentication.type= kerberos
httpfs.hadoop.authentication.type= kerberos
httpfs.authentication.kerberos.principal= HTTP/_HOST@EMR
httpfs.hadoop.authentication.kerberos.principal= hadoop/_HOST@EMR
httpfs.authentication.kerberos.keytab= /var/krb5kdc/emr.keytab
httpfs.hadoop.authentication.kerberos.keytab= /var/krb5kdc/emr.keytab
```

Yarn access

Modifying yarn-site.xml

```
yarn.resourcemanager.keytab= /var/krb5kdc/emr.keytab
yarn.resourcemanager.principal= hadoop/_HOST@EMR
yarn.nodemanager.keytab= /var/krb5kdc/emr.keytab
yarn.nodemanager.principal= hadoop/_HOST@EMR
```

Modifying mapred-site.xml

```
mapreduce.jobhistory.keytab= /var/krb5kdc/emr.keytab
mapreduce.jobhistory.principal= hadoop/_HOST@EMR
```

Knox Development Guide

Knox Development Guide

Last updated : 2020-10-13 15:48:05

Currently, EMR v1.3.1 and v2.0.1 support [Apache Knox](#). After completing the following preparations, you can access the web UIs of services such as Yarn and HDFS on the internet.

Preparations

- You have signed up for a Tencent Cloud account and created an EMR cluster.
- In EMR version 1.3.1 and 2.0.1, Knox is a required component by default when a cluster is created. If you use a legacy version, please [contact our customer service](#) to help you install Knox.

Accessing Knox

Access by using the public IP address of the cluster. You are recommended to modify the CVM security group rule of this IP to limit the accessing IP address on the TCP:30002 port to your own IP address.

1. View the public IP address in the cluster details.
2. Access the URL of the corresponding service in a browser.
 - HDFS UI: `https://{public IP address of the cluster}:30002/gateway/emr/hdfs`
 - Yarn UI: `https://{public IP address of the cluster}:30002/gateway/emr/yarn`
 - Hive UI: `https://{public IP address of the cluster}:30002/gateway/emr/hive`
 - HBase UI: `https://{public IP address of the cluster}:30002/gateway/emr/hbase/webui`
 - Hue UI: `https://{public IP address of the cluster}:30002/gateway/emr/hue`
 - Storm UI: `https://{public IP address of the cluster}:30002/gateway/emr/stormui`
 - Ganglia UI: `https://{public IP address of the cluster}:30002/gateway/emr/ganglia/`
 - Presto UI: `https://{public IP address of the cluster}:30002/gateway/emr/presto/`
 - Oozie UI: `https://{public IP address of the cluster}:30002/gateway/emr/oozie/`
3. The browser may display an error message saying that **your connection is not private**. This is because that the Knox service uses a self-signed certificate. Please confirm again that you are accessing the public IP address of your own cluster and the port is 30002. Then, select **Advanced > Proceed**.
4. In the login box that pops up, the username is root, and the default password is the one entered when the cluster was created. **It is recommended to change the password by clicking ****Reset Native UI Password** on the page.****

Integrating Knox with Tez

Last updated : 2022-05-16 12:52:25

As earlier EMR versions don't come with the Timeline Server and Tomcat packages necessary for Tez UI integration, you need to complete the steps below.

Note :

As starting Tez UI requires running Timeline Server, which uses a lot of resources, you need to assess the impact on your business and proceed with caution.

This document describes how to integrate Knox with Tez, such as installing Tomcat and Tez UI, creating roles, configuring Timeline Server, configuring Tez, and starting services. `172.**.**.9` is the private IP of the master node, `159.**.**.70` is the public IP of the master node, and the Tez version is v0.9.2.

Installing Tomcat and Tez UI

```
cd /usr/local/service
wget https://jaydihu-package-1258469122.cos.ap-guangzhou.myqcloud.com/apache-tomcat-9.0.46.tar.gz
tar -zxvf apache-tomcat-9.0.46.tar.gz
mv /usr/local/service/apache-tomcat-9.0.46 /usr/local/service/tomcat
```

Modify Tomcat port numbers:

```
vim /usr/local/service/tomcat/conf/server.xml
```

1. Change 8005 to 2020 .

```
the License. You may obtain a copy of the License at
    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
-->
<!-- Note: A "Server" is not itself a "Container", so you may not
define subcomponents such as "Valves" at this level.
Documentation at /docs/config/server.html
-->
<Server port="2020" shutdown="SHUTDOWN">
  <Listener className="org.apache.catalina.startup.VersionLoggerListener" />
  <!-- Security listener. Documentation at /docs/config/listeners.html
  <Listener className="org.apache.catalina.security.SecurityListener" />
  -->
  <!--APR library loader. Documentation at /docs/apr.html -->
  <Listener className="org.apache.catalina.core.AprLifecycleListener" SSLEngine="on" />
  <!-- Prevent memory leaks due to use of particular java/javax APIs-->
  <Listener className="org.apache.catalina.core.JreMemoryLeakPreventionListener" />
  <Listener className="org.apache.catalina.mbeans.GlobalResourcesLifecycleListener" />
  <Listener className="org.apache.catalina.core.ThreadLocalLeakPreventionListener" />

  <!-- Global JNDI resources
  Documentation at /docs/jndi-resources-howto.html
  -->
  <GlobalNamingResources>
    <!-- Editable user database that can also be used by
    UserDatabaseRealm to authenticate users
```

2. Change 8080 to 2000 .

```
<!-- A "Connector" represents an endpoint by which requests are received
and responses are returned. Documentation at :
Java HTTP Connector: /docs/config/http.html
Java AJP  Connector: /docs/config/ajp.html
APR (HTTP/AJP) Connector: /docs/apr.html
Define a non-SSL/TLS HTTP/1.1 Connector on port 8080
-->
<Connector port="2000" protocol="HTTP/1.1"
connectionTimeout="20000"
redirectPort="8443" />
<!-- A "Connector" using the shared thread pool-->
```

```
mkdir -p /usr/local/service/tomcat/webapps/tez-ui
cp /usr/local/service/tez/tez-ui-0.9.2.war /usr/local/service/tomcat/webapps/tez-
ui/
cd /usr/local/service/tomcat/webapps/tez-ui
```

```
unzip tez-ui-0.9.2.war
vim ./config/configs.env
```

Change `localhost` to the private IP of the current server.

```
* Licensed to the Apache Software Foundation (ASF) under one
* or more contributor license agreements. See the NOTICE file
* distributed with this work for additional information
* regarding copyright ownership. The ASF licenses this file
* to you under the Apache License, Version 2.0 (the
* "License"); you may not use this file except in compliance
* with the License. You may obtain a copy of the License at
*
* http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/
ENV = {
  hosts: {
    /*
    * Timeline Server Address:
    * By default TEZ UI looks for timeline server at http://localhost:8188, uncomment and change
    * the following value for pointing to a different address.
    */
    timeline: "http://172.17.0.9:8188",
    /*
    * Resource Manager Address:
    * By default RM REST APIs are expected to be at http://localhost:8088, uncomment and change
    * the following value to point to a different address.
    */
    rm: "http://172.17.0.9:8088",
```

Creating Roles

```
vim /usr/local/service/knox/conf/topologies/emr.xml
```

Modify the `emr.xml` file:

1. Add the following content:

```
80      <!-- HA Provider -->
81      <provider>
82          <role>ha</role>
83          <name>HaProvider</name>
84          <enabled>>true</enabled>
85          <param>
86              <name>HDFSUI</name>
87              <value>maxFailoverAttempts=3; failoverSleep=1000; enabled=true</value>
88          </param>
89          <param>
90              <name>TEZUI</name>
91              <value>maxFailoverAttempts=3; failoverSleep=1000; enabled=true</value>
92          </param>
93          <param>
94              <name>APPLICATIONHISTORY</name>
95              <value>maxFailoverAttempts=3; failoverSleep=1000; enabled=true</value>
96          </param>
97          <param>
98              <name>WEBHDFS</name>
99              <value>maxFailoverAttempts=3; failoverSleep=1000; enabled=true</value>
100         </param>
101         <param>
102             <name>YARNUI</name>
103             <value>maxFailoverAttempts=3; failoverSleep=1000; enabled=true</value>
104         </param>
105         <param>
106             <name>GANGLIAUI</name>
107             <value>maxFailoverAttempts=3; failoverSleep=1000; enabled=true</value>
108         </param>
```

```
<param>
<name>TEZUI</name>
<value>maxFailoverAttempts=3; failoverSleep=1000; enabled=true</value>
</param>
<param>
<name>APPLICATIONHISTORY</name>
<value>maxFailoverAttempts=3; failoverSleep=1000; enabled=true</value>
</param>
```

2. Add the following content:

```

<!--HA情况下，需要区分默认active和standby，对于active，url1是activeurl，url2是standbyurl，对于standby则相反-->
<service>
  <role>HDFSUI</role>
  <url>http://172.***.***.9:4008</url>
  <version>2.7.0</version>
</service>

<service>
  <role>TEZUI</role>
  <url>http://172.***.***.9:2000/tez-ui</url>
  <version>0.9.2</version>
</service>

<service>
  <role>APPLICATIONHISTORY</role>
  <url>http://172.***.***.9:8188</url>
  <version>2.7.3</version>
</service>

```

```

<service>
<role>TEZUI</role>
<url>http://172.**.**.9:2000/tez-ui</url>
<version>0.9.2</version>
</service>
<service>
<role>APPLICATIONHISTORY</role>
<url>http://172.**.**.9:8188</url>
<version>2.7.3</version>
</service>

```

Configuring YARN Timeline Server

Modify the `yarn-site.xml` file in configuration management, save the changes, and restart the components whose configuration has been changed.

Parameter	Value
<code>yarn.timeline-service.enabled</code>	true
<code>yarn.timeline-service.hostname</code>	<code>172.**.**.9</code> (replace it with your IP)
<code>yarn.timeline-service.http-cross-origin.enabled</code>	true
<code>yarn.resourcemanager.system-metrics-publisher.enabled</code>	true
<code>yarn.timeline-service.address</code>	<code>172.**.**.9:10201</code> (replace it with your IP)

Parameter	Value
yarn.timeline-service.webapp.address	172.**.**.9:8188 (replace it with your IP.)
yarn.timeline-service.webapp.https.address	172.**.**.9:2191 (replace it with your IP)
yarn.timeline-service.generic-application-history.enabled	true
yarn.timeline-service.handler-thread-count	24

Modifying Tez Configuration

In the `tez-site.xml` file, add the following parameters, save the changes, and restart the components whose configuration has been changed.

Parameter	Value
tez.tez-ui.history-url.base	http://172.**.**.9:2000/tez-ui/ (replace it with your IP)
tez.history.logging.service.class	org.apache.tez.dag.history.logging.ats.ATSHistoryLoggingService

Starting Services

1. Start Timeline Server.

```
/usr/local/service/hadoop/sbin/yarn-daemon.sh start timelinedserver
```

2. Start Tomcat.

```
/usr/local/service/tomcat/bin/startup.sh
```

3. Restart Tez.

```
su hadoop
rm -rf /usr/local/service/knox/data/deployments/*
/usr/local/service/knox/bin/ldap.sh stop
/usr/local/service/knox/bin/ldap.sh start
/usr/local/service/knox/bin/gateway.sh stop
/usr/local/service/knox/bin/gateway.sh start
```


4. Access Tez UI at the following address.

Note :

The account and password are the same as the server login account and password.

```
https://{public IP of the cluster}:30002/gateway/emr/tez/
```

Stopping Services

If you find that the Timeline Server significantly affects your business after running it for a period of time, you can stop relevant services as follows:

1. Stop Tomcat.

```
/usr/local/service/tomcat/bin/shutdown.sh
```

2. Stop Timeline Server.

```
/usr/local/service/hadoop/sbin/yarn-daemon.sh stop timelineserver
```

Integrating Knox with Tez v0.8.5

Last updated : 2021-07-05 10:03:56

1. The path of the `tez-ui` file in Tez v0.8.5 is `/usr/local/service/tomcat/webapps/tez-ui/scripts/configs.js`, corresponding to `configs.env` in v0.9.2.
2. Configure the `yarn-site.xml` file.

Create a `mkdir -p /usr/local/service/hadoop/filesystem/yarn/timeline` directory.

Parameter	Value
<code>yarn.timeline-service.enabled</code>	<code>true</code>
<code>yarn.timeline-service.hostname</code>	<code>172.**.**.9</code>
<code>yarn.timeline-service.http-cross-origin.enabled</code>	<code>true</code>
<code>yarn.resourcemanager.system-metrics-publisher.enabled</code>	<code>true</code>
<code>yarn.timeline-service.address</code>	<code>172.**.**.9:10201</code>
<code>yarn.timeline-service.webapp.address</code>	<code>172.**.**.9:8188</code>
<code>yarn.timeline-service.webapp.https.address</code>	<code>172.**.**.9:2191</code>
<code>arn.timeline-service.generic-application-history.enabled</code>	<code>true</code>
<code>yarn.timeline-service.leveldb-timeline-store.path</code>	<code>/usr/local/service/hadoop/filesystem/yarn/timeline</code>
<code>yarn.timeline-service.handler-thread-count</code>	<code>24</code>

3. In the `tez-site.xml` file, configure the following parameters:

Parameter	Value
<code>tez.allow.disabled.timeline-domains</code>	<code>true</code>
<code>tez.history.logging.service.class</code>	<code>org.apache.tez.dag.history.logging.ats.ATSHistoryLoggingService</code>

Parameter	Value
tez.tez-ui.history-url.base	<code>http://172.**.**.9:2000/tez-ui/</code>

Integrating Knox with Tez v0.10.0

Last updated : 2022-06-15 15:12:41

1. Download and compile the [Tez v0.10.0 source code package](#).

```
tar -zxvf apache-tez-0.10.1-src.tar.gz
chmod -R 777 apache-tez-0.10.1-src
cd apache-tez-0.10.1-src
mvn -X clean package -DskipTests=true -Dmaven.javadoc.skip=true
```

2. Decompress the compiled WAR package.

3. Configure the `yarn-site.xml` file.

Create a `mkdir -p /usr/local/service/hadoop/filesystem/yarn/timeline` directory.

Parameter	Value
<code>yarn.timeline-service.enabled</code>	<code>true</code>
<code>yarn.timeline-service.hostname</code>	<code>172.**.**.9</code>
<code>yarn.timeline-service.http-cross-origin.enabled</code>	<code>true</code>
<code>yarn.resourcemanager.system-metrics-publisher.enabled</code>	<code>true</code>
<code>yarn.timeline-service.address</code>	<code>172.**.**.9:10201</code>
<code>yarn.timeline-service.webapp.address</code>	<code>172.**.**.9:8188</code>
<code>yarn.timeline-service.webapp.https.address</code>	<code>172.**.**.9:2191</code>
<code>yarn.timeline-service.generic-application-history.enabled</code>	<code>true</code>
<code>yarn.timeline-service.handler-thread-count</code>	<code>24</code>

4. Configure the `tez.xml` file.

Parameter	Value
<code>tez.tez-ui.history-url.base</code>	<code>http://172.**.**.9:2000/tez-ui/</code>
<code>tez.history.logging.service.class</code>	<code>org.apache.tez.dag.history.logging.ats.ATSHistoryLoggingService</code>

Alluxio Development Guide

Alluxio Development Documentation

Last updated : 2021-04-28 15:51:06

Background

Alluxio provides access to data through a filesystem interface. Files in Alluxio offer write-once semantics: they become immutable after they have been written in their entirety and cannot be read before being completed. Alluxio provides two different Filesystem APIs, the Alluxio API and a Hadoop compatible API. The Alluxio API provides additional functionality, while the Hadoop compatible API gives you the flexibility of leveraging Alluxio without having to modify existing code written through Hadoop's API.

All resources with the Alluxio Java API are specified through an AlluxioURI which represents the path to the resource.

Getting a Filesystem Client

To obtain an Alluxio filesystem client in Java code, use:

```
FileSystem fs = FileSystem.Factory.get();
```

Creating a File

All metadata operations as well as opening a file for reading or creating a file for writing are executed through the Filesystem object. Since Alluxio files are immutable once written, the idiomatic way to create files is to use

`FileSystem#createFile(AlluxioURI)`, which returns a stream object that can be used to write the file. For example:

```
FileSystem fs = FileSystem.Factory.get();
AlluxioURI path = new AlluxioURI("/myFile");
// Create a file and get its output stream
FileOutputStream out = fs.createFile(path);
// Write data
out.write(...);
// Close and complete file
out.close();
```

Specifying Operation Options

For all FileSystem operations, an additional options field may be specified, which allows you to specify non-default settings for the operation. For example:

```
FileSystem fs = FileSystem.Factory.get();
AlluxioURI path = new AlluxioURI("/myFile");
// Generate options to set a custom blocksize of 128 MB
CreateFileOptions options = CreateFileOptions.defaults().setBlockSize(128 * Constants.MB);
FileOutputStream out = fs.createFile(path, options);
```

IO Options

Alluxio uses two different storage types: Alluxio managed storage and under storage. Alluxio managed storage is the memory, SSD, and/or HDD allocated to Alluxio workers. Under storage is the storage resource managed by the underlying storage system, such as S3, Swift, or HDFS. You can specify the interaction with Alluxio managed storage through ReadType and WriteType. ReadType specifies the data read behavior when reading a file. WriteType specifies the data write behavior when writing a new file, e.g., whether the data should be written in Alluxio storage.

Below is a table of the expected behaviors of ReadType. Reads will always prefer Alluxio storage over the under storage system.

Read Type	Behavior
CACHE_PROMOTE	<ul style="list-style-type: none"> Data is moved to the highest tier in the worker where the data was read. If the data was not in the Alluxio storage of the local worker, a replica will be added to the local Alluxio worker. If <code>alluxio.user.file.cache.partially.read.block</code> is set to true, data blocks that are not completely read will also be entirely stored in Alluxio. In contrast, a data block can be cached only when it is completely read.
CACHE	<ul style="list-style-type: none"> If the data was not in the Alluxio storage of the local worker, a replica will be added to the local Alluxio worker. If <code>alluxio.user.file.cache.partially.read.block</code> is set to true, data blocks that are not completely read will also be entirely stored in Alluxio. In contrast, a data block can be cached only when it is completely read.
NO_CACHE	Data is read without storing a replica in Alluxio.

Below is a table of the expected behaviors of WriteType.

Write Type	Behavior
CACHE_THROUGH	Data is written synchronously to an Alluxio worker and the under storage system.
MUST_CACHE	Data is written synchronously to an Alluxio worker. No data will be written to the under storage. This is the default write type.
THROUGH	Data is written synchronously to the under storage. No data will be written to Alluxio.
ASYNC_THROUGH	Data is written synchronously to an Alluxio worker and asynchronously to the under storage system. This is experimental.

Location Policy

Alluxio provides location policy to choose which workers to store the blocks of a file.

Using Alluxio's Java API, you can set the policy in `CreateFileOptions` for writing files into Alluxio and `OpenFileOptions` for reading files from Alluxio.

You can simply override the default policy class in the configuration file at property `alluxio.user.file.write.location.policy.class`. The built-in policies include:

- `LocalFirstPolicy` (`alluxio.client.file.policy.LocalFirstPolicy`)
It returns the local node first, and if the local worker doesn't have enough capacity of a block, it randomly selects a worker from the active workers list. This is the default policy.
- `MostAvailableFirstPolicy` (`alluxio.client.file.policy.MostAvailableFirstPolicy`)
It returns the worker with the most available bytes.
- `RoundRobinPolicy` (`alluxio.client.file.policy.RoundRobinPolicy`)
It chooses the worker for the next block in a round-robin manner and skips workers that do not have enough capacity.
- `SpecificHostPolicy` (`alluxio.client.file.policy.SpecificHostPolicy`)
It returns a worker with the specified node name. This policy cannot be set as default policy.

Alluxio supports custom policies, so you can also develop your own policy appropriate for your workload by implementing interface `alluxio.client.file.policy.FileWriteLocationPolicy`.

Note :

The default policy must have an empty constructor. To use the `ASYNC_THROUGH` write type, all the blocks of a file must be written to the same worker.

Alluxio allows a client to select a tier preference when writing blocks to a local worker. Currently, this policy preference exists only for local workers but not remote workers; remote workers will write blocks to the highest tier.

By default, data is written to the top tier. You can modify the default setting through the

```
alluxio.user.file.write.tier.default
```

 configuration property or override it through an option to the `FileSystem#createFile(AlluxioURI)` API call.

All operations on existing files or directories require you to specify the AlluxioURI. With the AlluxioURI, you may use any of the methods of `FileSystem` to access the resource.

An AlluxioURI can be used to perform Alluxio `FileSystem` operations, such as modifying the file metadata, ttl, or pin state, or getting an input stream to read the file. For example, to read a file:

```
FileSystem fs = FileSystem.Factory.get();
AlluxioURI path = new AlluxioURI("/myFile");
// Open the file for reading
FileInputStream in = fs.openFile(path);
// Read data
in.read(...);
// Close file relinquishing the lock
in.close();
```


Common Alluxio Commands

Last updated : 2021-04-28 15:20:26

Operation	Syntax	Description
cat	cat "path"	Print the content of a file in Alluxio to the console.
checkConsistency	checkConsistency "path"	Check the metadata consistency between Alluxio and the under storage.
checksum	checksum "path"	Calculate the md5 checksum for a file.
chgrp	chgrp "group" "path"	Change the group of a file or directory in Alluxio.
chmod	chmod "permission" "path"	Change the permission of the directory or file in Alluxio.
chown	chown "owner" "path"	Change the owner of a file or directory in Alluxio.
copyFromLocal	copyFromLocal "source path" "remote path"	Copy the file specified by "source path" to the path in Alluxio specified by "remote path". This command will fail if "remote path" already exists.
copyToLocal	copyToLocal "remote path" "local path"	Copy the file specified by "remote path" in Alluxio to a local destination.
count	count "path"	Display the number of folders and files matching the specified prefix in "path".
cp	cp "src" "dst"	Copy a file or directory within the Alluxio file system.
du	du "path"	Display the size of the file or directory specified by the input path.
fileInfo	fileInfo "path"	Print the information of the blocks of a specified file.
free	free "path"	Free a file or all files under a directory from Alluxio. If the file/directory is also in under storage, it will still be available there.
getCapacityBytes	getCapacityBytes	Get the capacity of the Alluxio file system.
getUsedBytes	getUsedBytes	Get the number of bytes used in the Alluxio file system.

help	help "cmd"	Print help information for the given command. If no command is given, print help information for all supported commands.
leader	leader	Print the current Alluxio leader master node name.
load	load "path"	Load the data of a file or a directory from under storage into Alluxio.
loadMetadata	loadMetadata "path"	Load the metadata of a file or directory from under storage into Alluxio.
location	location "path"	Output a list of node that have the specified file data.
ls	ls "path"	List all the files and directories directly under the given path with information such as size.
masterInfo	masterInfo	Print information regarding Alluxio master fault tolerance such as leader address, list of master addresses, and configured Zookeeper address.
mkdir	mkdir "path1" ... "pathn"	Create one or more directories under the given paths, along with any necessary parent directories. Multiple paths are separated by spaces or tabs. This command will fail if any of the given paths already exist.
mount	mount "path" "uri"	Mount the underlying file system path "uri" into the Alluxio namespace as "path". The "path" is assumed not to exist and is created by the operation. No data or metadata is loaded from under storage into Alluxio. After a path is mounted, operations on objects under the mounted path are mirror to the mounted under storage.
mv	mv "source" "destination"	Move a file or directory specified by "source" to a new location "destination". This command will fail if "destination" already exists.
persist	persist "path1" ... "pathn"	Persist files or directories currently stored only in Alluxio to the underlying file system.
pin	pin "path"	Pin the given file to avoid evicting it from memory. If the given path is a directory, it recursively pins all the files contained and any new files created within this directory.
report	report "path"	Report to the master that a file is lost.
rm	rm "path"	Remove a file. This command will fail if the given path is a directory rather than a file.
setTtl	setTtl "path" "time"	Set the TTL (time to live) in milliseconds for a file.

stat	stat "path"	Display information of the specified file or directory.
tail	tail "path"	Print the last 1 KB of the specified file to the console.
test	test "path"	Test a property of a path, returning 0 if the property is true, or 1 otherwise.
touch	touch "path"	Create a 0-byte file at the specified location.
unmount	unmount "path"	Unmount the underlying file system path mounted in the Alluxio namespace as "path". Alluxio objects under "path" are removed from Alluxio, but they still exist in the previously mounted under storage.
unpin	unpin "path"	Unpin the given file to allow Alluxio to evict this file again. If the given path is a directory, it recursively unpins all files contained and any new files created within this directory.
unsetTtl	unsetTtl "path"	Remove the TTL (time to live) setting from a file.

cat

Background

The cat command prints the entire contents of a file in Alluxio to the console. This can be useful for verifying the file. Use the copyToLocal command to copy the file to your local file system.

Operation example

When trying out a new computation job, cat can be used as a quick way to check the output:

```
$ ./bin/alluxio fs cat /output/part-00000
```

checkConsistency

Background

The checkConsistency command compares Alluxio and under storage metadata for a given path. If the path is a directory, the entire subtree will be compared. The command returns a message listing each inconsistent file or directory. The system administrator should reconcile the differences of these files at their discretion. To avoid metadata inconsistencies between Alluxio and under storages, design your systems to modify files and directories through the Alluxio and avoid directly modifying state in the underlying storage.

If the `-r` option is used, the `checkConsistency` command will repair all inconsistent files and directories under the given path. If an inconsistent file or directory exists only in under storage, its metadata will be added to Alluxio. If an inconsistent file exists in Alluxio and its data is fully present in Alluxio, its metadata will be loaded to Alluxio again.

Note :

This command requires a read lock on the subtree being checked, meaning writes and updates to files or directories in the subtree cannot be completed until this command completes.

Operation example

`checkConsistency` can be used to periodically validate the integrity of the namespace.

List each inconsistent file or directory:

```
$ ./bin/alluxio fs checkConsistency /
```

Repair the inconsistent files or directories:

```
$ ./bin/alluxio fs checkConsistency -r /
```

checksum

Background

The `checksum` command outputs the md5 value of a file in Alluxio.

Operation example

`checksum` can be used to verify the content of a file stored in Alluxio matches the content stored in an under file system or local file system.

```
$ ./bin/alluxio fs checksum /LICENSE
```

chgrp

Background

The `chgrp` command changes the group of the file or directory in Alluxio. Alluxio supports file authorization with Posix file permission. Group is an authorizable entity in Posix file permission model. The file owner or super-user can

execute this command to change the group of the file or directory.

Adding -R option also changes the group of child file and child directory recursively.

Operation example

chgrp can be used as a quick way to change the group of file:

```
$ ./bin/alluxio fs chgrp alluxio-group-new /input/file1
```

chmod

Background

The chmod command changes the permission of file or directory in Alluxio. Currently, octal mode is supported: the numerical format accepts three octal digits which refer to permissions for the file owner, the group and other users. Here is the number-permission mapping table:

Number	Permission	rwX
7	read, write and execute	rwX
6	read and write	rw-
5	read and execute	r-X
4	read only	r--
3	write and execute	-WX
2	write only	-W-
1	execute only	--X
0	none	---

Adding -R option also changes the permission of child file and child directory recursively.

Operation example

chmod can be used as a quick way to change the permission of file:

```
$ ./bin/alluxio fs chmod 755 /input/file1
```

chown

Background

The `chown` command changes the owner of the file or directory in Alluxio. For obvious security reasons, the ownership of a file can only be altered by a super-user.

Adding `-R` option also changes the owner of child file and child directory recursively.

Sample

`chown` can be used as a quick way to change the owner of file:

```
$ ./bin/alluxio fs chown alluxio-user /input/file1
```

copyFromLocal

Background

The `copyFromLocal` command copies the contents of a file in your local file system into Alluxio. If the node you run the command from has an Alluxio worker, the data will be available on that worker. Otherwise, the data will be copied to a random remote node running an Alluxio worker. If a directory is specified, the directory and all its contents will be copied recursively.

Operation example

`copyFromLocal` can be used as a quick way to inject data into the system for processing:

```
$ ./bin/alluxio fs copyFromLocal /local/data /input
```

copyToLocal

Background

The `copyToLocal` command copies the contents of a file in Alluxio to a file in your local file system. If a directory is specified, the directory and all its contents will be downloaded recursively.

Operation example

`copyToLocal` can be used as a quick way to download output data for additional investigation or debugging.

```
$ ./bin/alluxio fs copyToLocal /output/part-00000 part-00000
```

count

Background

The count command outputs the number of files and folders matching a prefix as well as the total size of the files. count works recursively and accounts for any nested directories and files. count is best utilized when you have some predefined naming conventions for their files.

Operation example

If data files are stored by their date, count can be used to determine the number of data files and their total size for any date, month, or year.

```
$ ./bin/alluxio fs count /data/2014
```

cp

Background

The cp command copies a file or directory in the Alluxio file system or between local file system and Alluxio file system.

Scheme file indicates the local file system and scheme alluxio or no scheme indicates the Alluxio file system.

If the -R option is used and the source designates a directory, cp copies the entire subtree at source to the destination.

Operation example

cp can be used to copy files between Under file systems.

```
$ ./bin/alluxio fs cp /hdfs/file1 /s3/
```

du

Background

The du command outputs the size of a file. If a directory is specified, it will output the aggregate size of all files in the directory and its children directories.

Operation example

If the Alluxio space is unexpectedly over utilized, du can be used to detect which folders are taking up the most space.

```
$ ./bin/alluxio fs du /\**
```

fileInfo

Background

The fileInfo command is deprecated since Alluxio version 1.5. Please use the stat command instead.

The fileInfo command dumps the FileInfo representation of a file to the console. It is primarily intended to assist users in debugging their system. Generally, viewing the file information in the UI will be much easier to understand.

Operation example

fileInfo can be used to debug the block locations of a file. This is useful when trying to achieve locality for compute workloads.

```
$ ./bin/alluxio fs fileInfo /data/2015/logs-1.txt
```

free

Background

The free command sends a request to the master to evict all blocks of a file from the Alluxio workers. If the argument to free is a directory, it will recursively free all files. This request is not guaranteed to take effect immediately, as readers may be currently by using the blocks of the file. free will return immediately after the request is acknowledged by the master. Note that, files must be persisted already in under storage before being freed, or the free command will fail; also any pinned files cannot be freed unless -f option is specified. The free command does not delete any data from the under storage system, but only removing the blocks of those files in Alluxio space to reclaim space. In addition, metadata will not be affected by this operation, meaning the freed file will still show up if an ls command is run.

Operation example

free can be used to manually manage Alluxio's data caching.

```
$ ./bin/alluxio fs free /unused/data
```

getCapacityBytes

Background

The `getCapacityBytes` command returns the maximum number of bytes Alluxio is configured to store.

Operation example

`getCapacityBytes` can be used to verify if your cluster is set up as expected.

```
$ ./bin/alluxio fs getCapacityBytes
```

getUsedBytes

Background

The `getUsedBytes` command returns the number of used bytes in Alluxio.

Operation example

`getUsedBytes` can be used to monitor the health of your cluster.

```
$ ./bin/alluxio fs getUsedBytes
```

leader

Background

The `leader` command prints the current Alluxio leader master node name.

Operation example

```
$ ./bin/alluxio fs leader
```

load

Background

The `load` command moves data from the under storage system into Alluxio storage. If there is an Alluxio worker on the machine this command is run from, the data will be loaded to that worker. Otherwise, a random worker will be selected to serve the data. If the data is already loaded into Alluxio, `load` is a no-op. If `load` is run on a directory, files in the directory will be recursively loaded.

Operation example

load can be used to prefetch data for analytics jobs.

```
$ ./bin/alluxio fs load /data/today
```

loadMetadata

Background

The loadMetadata command queries the under storage system for any file or directory matching the given path and then creates a mirror of the file in Alluxio backed by that file. Only the metadata, such as the file name and size, are loaded this way and no data transfer occurs.

Operation example

loadMetadata can be used when other systems output to the under storage directly (bypassing Alluxio), and the application running on Alluxio needs to use the output of those systems.

```
$ ./bin/alluxio fs loadMetadata /hdfs/data/2015/logs-1.txt
```

location

Background

The location command returns the addresses of all the Alluxio workers which contain blocks belonging to the given file.

Operation example

location can be used to debug data locality when running jobs by using a compute framework.

```
$ ./bin/alluxio fs location /data/2015/logs-1.txt
```

ls

Background

The ls command lists all the immediate children in a directory and displays the file size, last modification time, and in memory status of the files. Using ls on a file will only display the information for that specific file. The ls command will

also load the metadata for any file or immediate children of a directory from the under storage system to Alluxio namespace, if it does not exist in Alluxio yet. `ls` queries the under storage system for any file or directory matching the given path and then creates a mirror of the file in Alluxio backed by that file. Only the metadata, such as the file name and size are loaded this way and no data transfer occurs.

Options:

`-d` option lists the directories as plain files. For example, `ls -d /` shows the attributes of root directory.

`f` option forces loading metadata for immediate children in a directory. By default, it loads metadata only at the first time at which a directory is listed.

`-h` option displays file sizes in human-readable formats.

`-p` option lists all pinned files.

`-R` option also recursively lists child directories, displaying the entire subtree starting from the input path.

Operation example

`ls` can be used to browse the file system.

```
$ ./bin/alluxio fs mount /cos/data cosn://data-bucket/
```

Verify:

```
$ ./bin/alluxio fs ls /s3/data/
```

masterInfo

Background

The `masterInfo` command prints information regarding master fault tolerance such as leader address, list of master addresses, and the configured Zookeeper address. If Alluxio is running in single master mode, `masterInfo` will print the master address. If Alluxio is running in fault tolerance mode, the leader address, list of master addresses and the configured Zookeeper address will be printed.

Operation example

`masterInfo` can be used to print information regarding master fault tolerance.

```
$ ./bin/alluxio fs masterInfo
```

mkdir

Background

The `mkdir` command creates a new directory in Alluxio space. It is recursive and will create any nonexistent parent directories. Note that the created directory will not be created in the under storage system until a file in the directory is persisted to the underlying storage. Using `mkdir` on an invalid or already existing path will fail.

Operation example

`mkdir` can be used by an admin to set up the basic folder structures.

```
$ ./bin/alluxio fs mkdir /users
$ ./bin/alluxio fs mkdir /users/Alice
$ ./bin/alluxio fs mkdir /users/Bob
```

mount

Background

The `mount` command links an under storage path to an Alluxio path, and files and folders created in Alluxio space under the path will be backed by a corresponding file or folder in the under storage path. For more details, please see [Unified Namespace](#).

Options:

`--readonly` option sets the mount point to be readonly in Alluxio

`--option <key>=<val>` option passes a property to this mount point (e.g., S3 credential)

Operation example

`mount` can be used to make data in another storage system available in Alluxio.

```
$ ./bin/alluxio fs mount /mnt/hdfs hdfs://host1:9000/data/
```

mv

Background

The `mv` command moves a file or directory to another path in Alluxio. The destination path must not exist or be a directory. If it is a directory, the file or directory will be placed as a child of the directory. `mv` is purely a metadata operation and does not affect the data blocks of the file. `mv` cannot be done between mount points of different under storage systems.

Operation example

mv can be used to move older data into a non working directory.

```
$ ./bin/alluxio fs mv /data/2014 /data/archives/2014
```

persist

Background

The persist command persists data in Alluxio storage into the under storage system. This is a data operation and will take time depending on how large the file is. After persist is complete, the file in Alluxio will be backed by the file in the under storage, make it still valid if the Alluxio blocks are evicted or otherwise lost.

Operation example

persist can be used after filtering a series of temporary files for the ones containing useful data.

```
$ ./bin/alluxio fs persist /tmp/experimental-logs-2.txt
```

pin

Background

The pin command marks a file or folder as pinned in Alluxio. This is a metadata operation and will not cause any data to be loaded into Alluxio. If a file is pinned, any blocks belonging to the file will never be evicted from an Alluxio worker. If there are too many pinned files, Alluxio workers may run low on storage space preventing other files from being cached.

Operation example

pin can be used to manually ensure performance if the administrator understands the workloads well.

```
$ ./bin/alluxio fs pin /data/today
```

report

Background

The report command marks a file as lost to the Alluxio master. This command should only be used with files created by using the Lineage API. Marking a file as lost will cause the master to schedule a recomputation job to regenerate the file.

Operation example

report can be used to force recomputation of a file.

```
$ ./bin/alluxio fs report /tmp/lineage-file
```

rm

Background

The rm command removes a file from Alluxio space and the under storage system. The file will be unavailable immediately after this command returns, but the actual data may be deleted a while later.

Add -R option will delete all contents of the directory and then the directory itself. Add -U option to not check whether the UFS contents being deleted are in-sync with Alluxio before attempting to delete persisted directories.

Operation example

rm can be used to remove temporary files which are no longer needed.

```
$ ./bin/alluxio fs rm /tmp/unused-file
```

setTtl

Background

The setTtl command sets the time-to-live of a file or a directory, in milliseconds. If set ttl to a directory, all the children inside that directory will set too. So a directory's TTL expires, all the children inside that directory will also expire.

Action parameter will indicate the action to perform once the current time is greater than the TTL + creation time of the file. Action delete (default) will delete file or directory from both Alluxio and the under storage system, whereas action free will just free the file from Alluxio even they are pinned.

Operation example

setTtl with action delete can be used to clean up files the administrator knows are unnecessary after a period of time, or with action free just remove the contents from Alluxio to make room for more space in Alluxio.

```
$ ./bin/alluxio fs setTtl -action free /data/good-for-one-day 86400000
```

stat

Background

The stat command dumps the FileInfo representation of a file or a directory to the console. It is primarily intended to assist users in debugging their system. Generally, viewing the file information in the UI will be much easier to understand.

You can specify -f to display information in a given format:

- "%N": name of the file
- "%z": size of the file in bytes
- "%u": owner
- "%g": group name of the owner
- "%y" or "%Y": modification time, %y shows 'yyyy-MM-dd HH:mm:ss' (the UTC date), %Y shows milliseconds since January 1, 1970 UTC
- "%b": number of blocks allocated for the file

Operation example

stat can be used to debug the block locations of a file. This is useful when trying to achieve locality for compute workloads.

```
$ ./bin/alluxio fs stat /data/2015/logs-1.txt
$ ./bin/alluxio fs stat /data/2015
$ ./bin/alluxio fs stat -f %z /data/2015/logs-1.txt
```

tail

Background

The tail command outputs the last 1 KB of data in a file to the console.

Operation example

tail can be used to verify the output of a job is in the expected format or contains expected values.

```
$ ./bin/alluxio fs tail /output/part-00000
```

test

Background

The test command tests a property of a path, returning 0 if the property is true, or 1 otherwise.

Options:

-d option tests whether the path is a directory.

-e option tests whether the path exists.

-f option tests whether the path is a file.

-s option tests whether the directory is empty.

-z option tests whether the file is zero length.

Operation example

```
$ ./bin/alluxio fs test -d /someDir
```

touch

Background

The touch command creates a 0-byte file. Files created with touch cannot be overwritten and are mostly useful as flags.

Operation example

touch can be used to create a file signifying the completion of analysis on a directory.

```
$ ./bin/alluxio fs touch /data/yesterday/_DONE_
```

unmount

Background

The unmount command disassociates an Alluxio path with an under storage directory. Alluxio metadata for the mount point will be removed along with any data blocks, but the under storage system will retain all metadata and data. See Unified Namespace for more details.

Operation example

unmount can be used to remove an under storage system when you no longer need data from that system.

```
$ ./bin/alluxio fs unmount /s3/data
```


unpin

Background

The unpin command unmarks a file or directory in Alluxio as pinned. This is a metadata operation and will not evict or delete any data blocks. Once a file is unpinned, its data blocks can be evicted from various Alluxio workers containing the block.

Operation example

unpin can be used when the administrator knows there is a change in the data access pattern.

```
$ ./bin/alluxio fs unpin /data/yesterday/join-table
```

unsetTtl

Background

The unsetTtl command will remove the TTL of a file in Alluxio. This is a metadata operation and will not evict or store blocks in Alluxio. The TTL of a file can later be reset with setTtl.

Operation example

unsetTtl can be used if a regularly managed file requires manual management due to some special case.

```
$ ./bin/alluxio fs unsetTtl /data/yesterday/data-not-yet-analyzed
```

Mounting File System to Unified Alluxio File System

Last updated : 2022-08-08 11:04:32

Background

Alluxio provides a unified namespace mechanism that allows other file systems to be mounted to the file system of Alluxio. In addition, it allows upper-layer applications to use the unified namespace to access data scattered across different systems.

Mounting COS

Sample: Mounting a COS bucket to an Alluxio directory

```
bin/alluxio fs mount --option fs.cos.access.key=<COS_SECRET_ID> \
--option fs.cos.secret.key=<COS_SECRET_KEY> \
--option fs.cos.region=<COS_REGION> \
--option fs.cos.app.id=<COS_APP_ID> \
/cos cos://<COS_BUCKET>/
```

Configure the COS information in each `--option` .

Configuration Item	Description
fs.cos.access.key	The COS secret ID
fs.cos.secret.key	The COS secret key
fs.cos.region	The COS region name, such as <code>ap-beijing</code>
fs.cos.app.id	Your <code>AppID</code>
COS_BUCKET	The COS bucket name without the <code>AppID</code> suffix

This command mounts the COS directory specified by `cos://bucket/xxx` to the `/cos` directory in Alluxio.

Mounting HDFS

Sample: Mounting an HDFS directory to an Alluxio directory

```
`bin/alluxio fs mount /hdfs hdfs://data`
```

This command mounts the `/data` directory of HDFS to the `/hdfs` subdirectory of Alluxio.

After the mount is successful, the mounted content can be viewed by running the `alluxio fs ls` command.

Mounting CHDFS

Sample: Mounting CHDFS to Alluxio through `mount`

Note :

This is supported only for EMR 2.5.0 or later + Alluxio 2.3.0 or later.

```
alluxio fs mount \  
--option alluxio.underfs.hdfs.configuration=/usr/local/service/hadoop/etc/hadoop/  
core-site.xml \  
/chdfs ofs://f4modr7kmvw-wMqw.chdfs.ap-chongqing.myqcloud.com
```

Using Alluxio in Tencent Cloud

Last updated : 2022-08-19 11:31:23

Overview

Tencent Cloud EMR comes with the ready-to-use Alluxio service, helping you accelerate distributed memory-level caching and simplify data management. You can also use the configuration delivery feature to configure multi-level caching and manage metadata via the EMR console or APIs. In addition, EMR offers one-stop monitoring and alarming.

Preparations

Tencent Cloud EMR Hadoop Standard v2.1.0 or above.

For specific Alluxio versions supported in EMR, see [Component Version](#).

Creating an Alluxio-based EMR Cluster

This section describes how to create a ready-to-use Alluxio-based EMR cluster. You can create an EMR cluster via the purchase page or API.

Creating a cluster via the purchase page

Go to the EMR [purchase page](#), choose an Alluxio-supported version, and select the Alluxio component in **Optional Components**.

Elastic MapReduce

1.Availability Zone and Software Configuration

2.Hardware Configuration

3.Basic Configuration

Billing Mode:

Region:

AZ:

Cluster Type:

Product Version: step 1

Required Components:

Optional Components:

step 2

▶ [Advanced Settings](#)

[Next Step: Hardware Configuration](#)

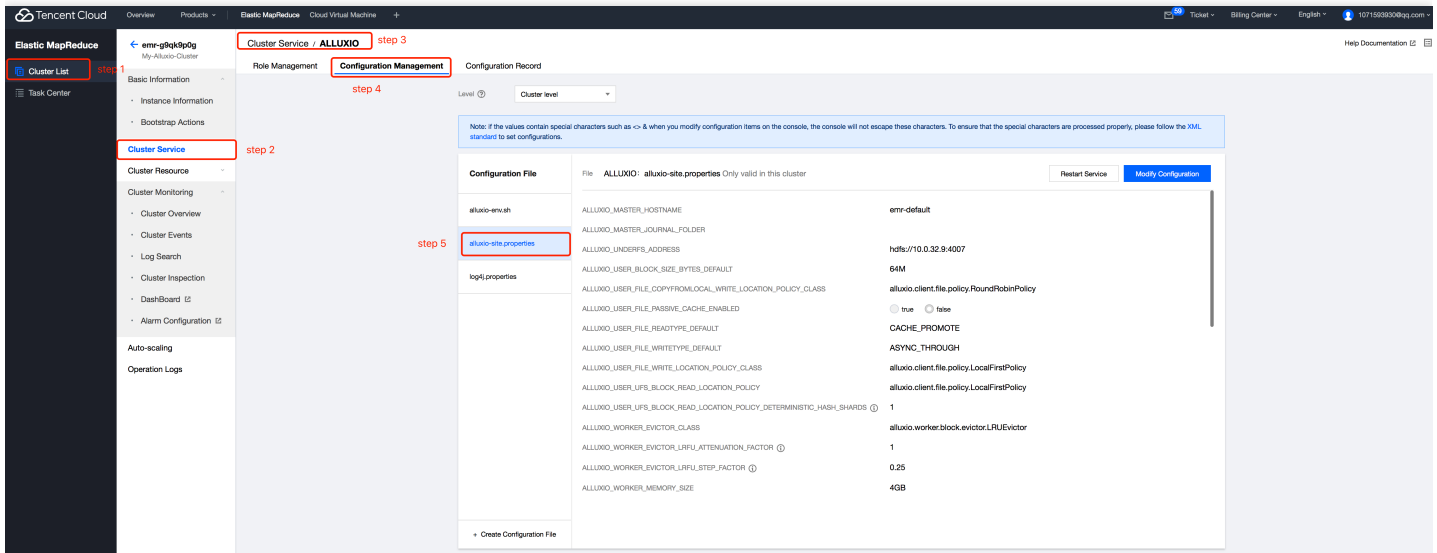
Select other options as needed to meet your business needs. For reference, see [Creating EMR Cluster](#).

Creating a cluster via API

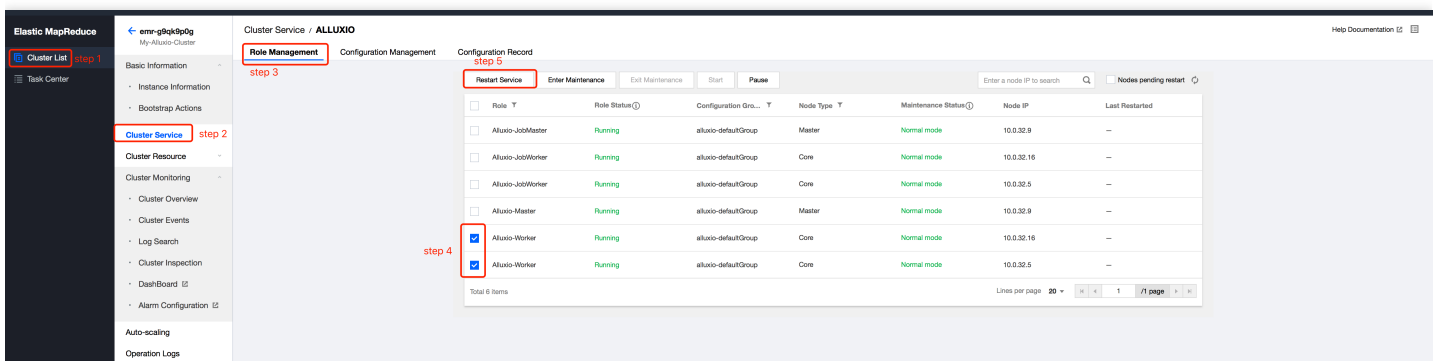
Tencent Cloud EMR also allows you to build a big data cluster based on Alluxio. For details, see [DescribeClusterNodes](#).

Basic Configurations

When you create an EMR cluster containing the Alluxio component, HDFS will be mounted to Alluxio and memory will be used for single-level (level 0) storage by default. You can use the configuration delivery feature to change the storage mode to multi-level storage or make other optimizations.



After delivering configurations, you need to restart the Alluxio service for some configurations to take effect.



For more details on configuration delivery and restarting policies, see [Configuration Management](#) and [Restarting Services](#).

Storage and compute separation based on Alluxio acceleration

Tencent Cloud EMR provides the compute and storage separation capability based on Tencent Cloud COS. By default, when directly accessing the data in COS, applications do not have node-level data locality or cross-application caching. Alluxio acceleration helps alleviate these issues.

COS is deployed on Tencent Cloud EMR clusters by default and serves as the dependent JAR package of UFS. You only need to grant EMR clusters the permission to access COS and mount COS to Alluxio.

Authorization

If COS is not enabled for the current cluster, you can go to [CAM console > Roles](#) to grant permission. After authorization, EMR nodes can access the data in COS using temporary keys.

Instance Information

Basic Information

Instance ID: emr-g6j4p0g
Region Info: Beijing(Beijing Zone 2)
High Availability Cluster: Disabled

Network Info: xue-vpc-kue-sub
Security Group: sg-6c390z6
Cloud disk encryption: Disabled

Creation Date: 2020-12-28 15:21:41
Master Node Public IP: 211.158.162.24
Billing Mode: Pay-as-you-go

COS: Not authorized. **Step 3**

Custom Service Role: Not set. **Set**

Software Info

Cluster Type: HADOOP
Product Version: EMR-V2.5.0
Component Info: zookeeper-3.6.1,hadoop-2.8.5,alluxio-2.3.0,java-1.8.0
MetaDB: None
Hive Metastore: None
Kerberos mode: Disabled

Hardware Info

Core Node: 2
Master Node: 1

Role Management

Service Authorization

After you agree to grant permissions to EMR, a preset role will be created and relevant permissions will be granted to EMR.

Role Name: EMR_LGCSRole
Role Type: Service Role
Description: Current role is a EMR service role, which will access your other cloud service resources within the permissions of the associated policies.
Authorized Policies: Preset policy QcloudAccessForEMRRoleApplicationDataAccess

Grant

Mounting

Log in to any machine of EMR and mount COS to Alluxio.

```
bin/alluxio fs mount <alluxio-path> <source-path>
//TODO,
```

For more information on using Alluxio in Tencent Cloud EMR, see [Alluxio Development Documentation](#).

Support for COS Transparent-URI

Last updated : 2022-08-12 14:59:19

Alluxio users often have existing applications accessing their existing storage systems (Under-FileSystem). Alluxio can be added to existing ecosystems, but one thing that must always be changed is the URI used by the application. The Transparent-URI feature allows users to access their existing storage systems without changing URIs at the application level.

Supported Versions and URI Configuration

1. Supported service component version: Alluxio 2.8.0.
2. Product version: Hadoop 3.x Standard EMR 3.4.0.
3. In order to use Alluxio Transparent-URI, a new Hadoop compatible file system client implementation should be configured. This new ShimFileSystem will replace existing FileSystem whenever the client is configured for receiving foreign URIs. The APIs of Hadoop FileSystem (a Hadoop compatible compute framework) define the mapping from FileSystem scheme to FileSystem implementation. To configure ShimFileSystem, make sure that the following items are configured in the `core-site.xml` file:

Configuration Item	Value
<code>fs.cosn.impl</code>	<code>alluxio.hadoop.ShimFileSystem</code>

4. In order to be compatible with the Transparent-URI schema, Alluxio needs to perform schema conversion. Therefore, make sure that the following items are configured in the `alluxio-site.properties` file:

Configuration Item	Value
<code>alluxio.master.uri.translator.impl</code>	<code>alluxio.master.file.uritranslator.AutoMountUriTranslator</code>
<code>alluxio.user.shimfs.bypass.ufs.impl.list</code>	<code>fs.cosn.impl:org.apache.hadoop.fs.cosnative.NativeCosFileSystem</code>

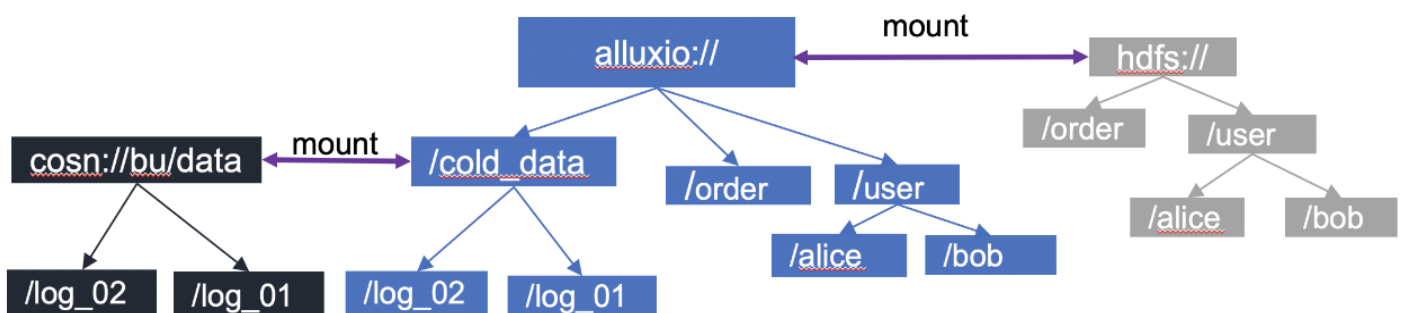
Note :

- You need to restart the Alluxio service after modifying the configuration in `alluxio-site.properties`.

- Once ShimFileSystem is configured, master will need to route URIs that are native to external storage system, to Alluxio namespace. This requires COSN to have been mounted to Alluxio namespace.
- To disable the Transparent-URI feature, roll back the `fs.cosn.impl` configuration item in `core-site.xml`.

Mounting

The `mount` command is one of Alluxio's most distinctive commands. It is similar to Linux's `mount` command, through which users can load disks, SSDs and other storage devices to the local file system of Linux. In Alluxio, the concept of mounting is further extended to the distributed system level, that is, users can mount one or more other storage systems/cloud storage services (such as HDFS and COS) to the Alluxio distributed file system using the `mount` command. So they can run distributed applications on Alluxio, such as Spark, Presto, and MapReduce, without any adaptation or even knowledge of the specific data access protocol and path. Users only need to know the path corresponding to the data in the Alluxio file system. It greatly facilitates application development and maintenance.



EMR-Alluxio uses HDFS as the root directory mount point by default

Beginning from EMR-Alluxio v2.5.1, Alluxio's UFS supports the COSN protocol. COS UFS has relatively poor read/write performance and is unstable. In order to solve this issue, the community provides the underlying COSN UFS. Both COS UFS and COSN UFS are used to access Tencent Cloud COS. COSN UFS is a great improvement compared with COS UFS. Its read/write performance are twice as good as that of COS UFS, and it has better stability. Therefore, COSN UFS is strongly recommended. The maintenance for COS UFS will stop after EMR-Alluxio v2.6.0.

Example of mounting COSN:

```
alluxio fs mount --option fs.cosn.userinfo.secretId=xx \  
--option fs.cosn.userinfo.secretKey=xx \  
--option fs.cosn.bucket.region=ap-xx \  

```

```
--option fs.cosn.impl=org.apache.hadoop.fs.cosnative.NativeCosFileSystem \
--option fs.AbstractFileSystem.cosn.impl=org.apache.hadoop.fs.CosN \
--option fs.cosn.userinfo.appid=xx \
/cosn cosn://COS_BUCKET/path
```

Configure the COS information in each `--option` .

Configuration Item	Description
fs.cosn.userinfo.secretId	COS secret ID
fs.cosn.userinfo.secretKey	COS secret key
fs.cosn.impl	Fixed value: <code>org.apache.hadoop.fs.CosFileSystem</code> .
fs.AbstractFileSystem.cosn.impl	Fixed value: <code>org.apache.hadoop.fs.CosN</code> .
fs.cosn.bucket.region cos region	Region name such as <code>ap-beijing</code>
fs.cosn.userinfo.appid	The AppID of root account
COS_BUCKET COS BUCKET	Bucket name without the AppID suffix

Support for Authentication

Last updated : 2022-08-12 14:59:19

Authentication is not required when Alluxio users access data from COS, HDFS, or CHDFS in the existing unified namespace or access the data cached in Alluxio through Transparent-URI; that is, any user can get the data as long as they get the URI. In view of this, EMR-Alluxio improves authentication based on Ranger and COSRanger.

Note :

To configure the authentication feature, make sure that the cluster is integrated with the following components:

- If only HDFS is mounted to Alluxio, you need to integrate the Ranger component.
- If COS and CHDFS are mounted to Alluxio, you need to integrate the COSRanger component.

Supported Versions

- Supported service component version: Alluxio v2.8.0.
- Product version: Hadoop 3.x Standard EMR v3.4.0.

Configuring Authentication

Prerequisite configuration

```
# Add the `ranger-hive-security.xml` configuration item in the Hive component
ranger.plugin.hive.urlauth.filesystem.schemes==hdfs:,file:,wasb:,adl:,alluxio:
# Add the `hive.properties` configuration item in the Presto component
hive.hdfs.authentication.type=NONE
hive.metastore.authentication.type=NONE
hive.hdfs.impersonation.enabled=true
hive.metastore.thrift.impersonation.enabled=true
```

Note :

The above prerequisite configuration items need to be configured based on the existing components in your cluster.

HDFS authentication

Create a soft link to the Ranger configuration file as follows:

```
[hadoop@172 conf]$ pwd
/usr/local/service/alluxio/conf
[hadoop@172 conf]$ ln -s /usr/local/service/hadoop/etc/hadoop/ranger-hdfs-audit.xml
ranger-hdfs-audit.xml
[hadoop@172 conf]$ ln -s /usr/local/service/hadoop/etc/hadoop/ranger-hdfs-security.xml
ranger-hdfs-security.xml
```

Configure `alluxio-site.properties`

We recommend you deliver the cluster configuration in the EMR console.

```
# Authentication switch (`false` by default)
alluxio.security.authorization.plugins.enabled=true
alluxio.security.authorization.plugin.name=ranger
alluxio.security.authorization.plugin.paths=/usr/local/service/alluxio/conf
alluxio.underfs.security.authorization.plugin.name=ranger
alluxio.underfs.security.authorization.plugin.paths=/usr/local/service/alluxio/conf
alluxio.master.security.impersonation.hadoop.users=*
alluxio.security.login.impersonation.username=_HDFS_USER_
```

Note :

You need to restart the Alluxio service after the delivery is completed.

COS and CHDFS authentication

```
# Add the `core-site.xml` configuration item
fs ofs.ranger.enable.flag=true
```

Configure `alluxio-site.properties`

We recommend you deliver the cluster configuration in the EMR console.

```
# Authentication switch (`false` by default)
# Authentication switch (`false` by default)
alluxio.security.authorization.plugins.enabled=true
alluxio.security.authorization.plugin.name=ranger
alluxio.security.authorization.plugin.paths=/usr/local/service/alluxio/conf
alluxio.underfs.security.authorization.plugin.name=ranger
```

```
alluxio.underfs.security.authorization.plugin.paths=/usr/local/service/alluxio/co
nf
alluxio.cos.qcloud.object.storage.ranger.service.config.dir=/usr/local/service/co
sranger/conf
alluxio.master.security.impersonation.hadoop.users=*
alluxio.security.login.impersonation.username=_HDFS_USER_
# The number of retries is 5 by default.
alluxio.cos.qcloud.object.storage.permission.check.max.retry=5
```

Note :

You need to restart the Alluxio service after the delivery is completed.

Kylin Development Guide

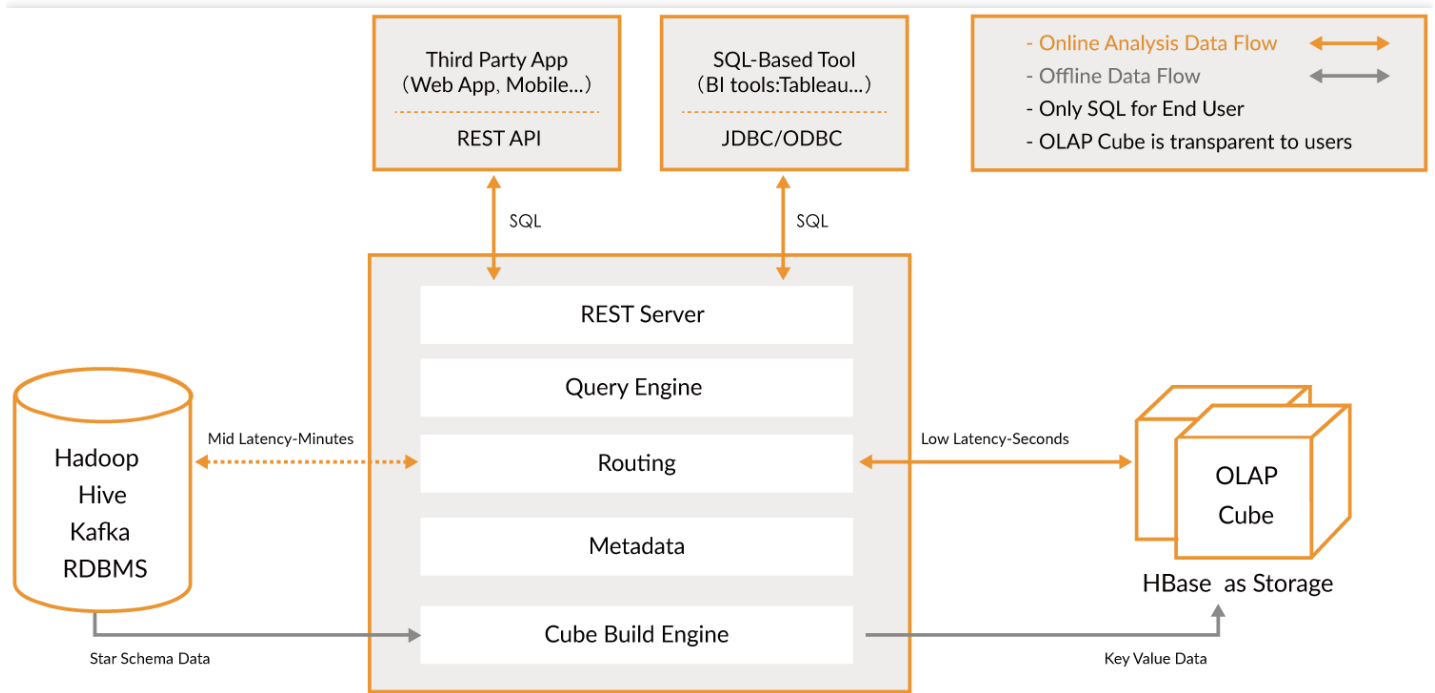
Kylin Overview

Last updated : 2020-10-12 15:38:05

Originally developed by eBay and then contributed to the open source community, Apache Kylin™ is an open-source and distributed analytical data warehouse designed to provide SQL interface and multi-dimensional analysis (OLAP) for Hadoop and Spark. It supports extremely large-scale datasets and can query huge tables in sub-seconds.

Overview of Kylin Framework

The key that enables Kylin to provide a sub-second latency is pre-calculation, which involves pre-calculating the measures of a data cube with a star topology in a combination of dimensions, saving the results in HBase, and then providing query APIs such as JDBC, ODBC, and RESTful APIs to implement real-time queries.



Core concepts of Kylin

- **Table:** it is defined in Hive and is the data source of a data cube which must be synced into Kylin before a cube is built.
- **Model:** it describes a star schema data structure and defines the connection and filtering relationship between a fact table and a number of lookup tables.

- **Cube descriptor:** it describes the definition and configuration of a cube instance, including which data model to use, what dimensions and measures to have, how to partition data, how to handle auto-merge, etc.
 - **Cube Instance:** it is built from a cube descriptor and consists of one or more cube segments.
- **Partition:** you can use a DATE/STRING column in a cube descriptor to separate a cube into several segments by date.
- **Cube segment:** it is the carrier of cube data. A segment maps to a table in HBase. After a cube instance is built, a new segment will be created. In case of any changes in the raw data of a built cube, you simply need to refresh the associated segments.
- **Aggregation group:** each aggregation group is a subset of dimensions and builds cuboid with combinations inside.
- **Job:** after a request to build a cube instance is initiated, a job will be generated, which records information about each step in building the cube instance. The status of the job reflects the result of cube instance creation. For example, RUNNING indicates that the cube instance is being built; FINISHED indicates that the cube instance has been successfully built; and ERROR indicates that cube instance creation failed.
- **Dimension and measure types**
 - **Mandatory:** this is a dimension that all cuboids must have.
 - **Hierarchy:** there is a hierarchical relationship between dimensions. Only hierarchical cuboids need to be retained.
 - **Derived:** in lookup tables, some dimensions can be derived from their primary key, so they will not be involved in cuboid creation.
 - **Count Distinct(HyperLogLog):** immediate COUNT DISTINCT is hard to calculate, so an approximate algorithm (HyperLogLog) is introduced to keep error rate at a lower level.
 - **Count Distinct(Precise):** precise COUNT DISTINCT will be pre-calculated based on RoaringBitmap. Currently, only `int` and `bigint` are supported.
- **Cube action types**
 - **BUILD:** this action builds a new cube segment at the time interval specified by a partition.
 - **REFRESH:** this action rebuilds a cube segment over some intervals.
 - **MERGE:** this action merges multiple cube segments. It can be set to be automated when building a cube.
 - **PURGE:** this action clears segments under a cube instance. However, it will not delete tables from HBase.
- **Job status**
 - **NEW:** this indicates that the job has been created.
 - **PENDING:** this indicates that the job has been submitted by the job scheduler and is waiting for execution resources.
 - **RUNNING:** this indicates that the job is running.
 - **FINISHED:** this indicates that the job has been successfully finished.
 - **ERROR:** this indicates that the job has exited due to an error.
 - **DISCARDED:** this indicates that the job has been canceled by the user.
- **Job execution**

- RESUME: this action will continue to execute a failed job from the last successful checkpoint.
- DISCARD: regardless of the status of the job, the user can end it and release the resources.

Getting Started with Cubes

Run the script to restart the Kylin server to purge the cache.

```
/usr/local/service/kylin/bin/sample.sh
```

Log in at the Kylin website with the default username and password (ADMIN/KYLIN), select the `learn_kylin` project from the project drop-down list in the top-left corner, select the sample cube named `kylin_sales_cube`, click **Actions** > **Build**, and select a date after January 1, 2014 (overwriting all 10000 sample records).

The screenshot shows the Kylin web interface. The top navigation bar includes 'Kylin', a project dropdown menu set to 'learn_kylin', and tabs for 'Insight', 'Model', 'Monitor', and 'System'. The main content area is divided into 'Models' and 'Data Source' sections. The 'Cubes' table is visible, with the following data:

Name	Status	Cube Size	Source Records	Last Build Time	Owner	Create Time	Actions	Admins
kylin_sales_cube	READY	93.40 MB	9,988	2017-10-24 17:18:56 GMT+8			Action	Action

Summary statistics: Total: 1, Storage: 93.40 MB.

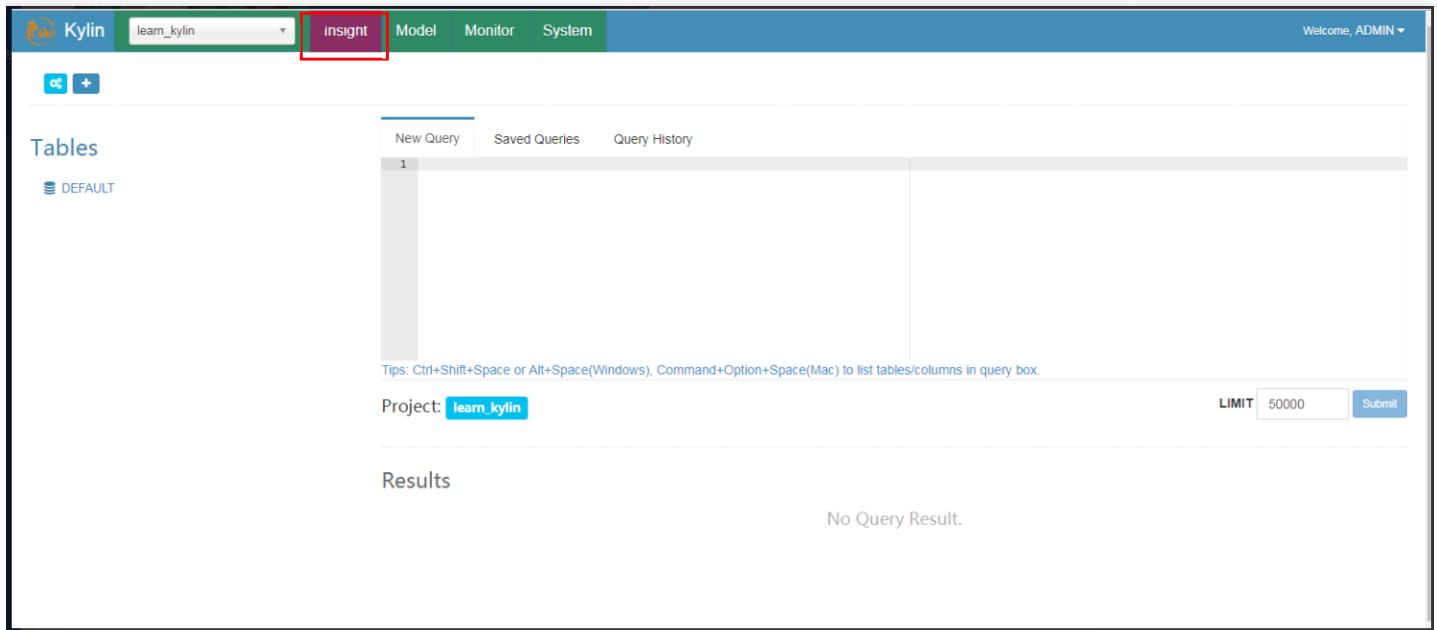
Click the **Monitor** to view the building progress until 100%.

The screenshot shows the 'Monitor' view in the Kylin web interface. The top navigation bar is the same as in the previous screenshot, but the 'Monitor' tab is selected. The main content area shows a 'Jobs' section with a search bar and a filter for 'Cube Name'. The 'Jobs in:' section includes filters for 'LAST ONE WEEK', 'NEW', 'PENDING', 'RUNNING', 'STOPPED', 'FINISHED', 'ERROR', and 'DISCARDED'. The 'Jobs' table is displayed with the following data:

Job Name	Cube	Progress	Last Modified Time	Duration	Actions
BUILD CUBE - kylin_sales_cube - 20120101000000_20140101000000 - GMT+08:00 2017-10-24 14:58:35	kylin_sales_cube	100%	2017-10-24 17:19:08 GMT+8	11.50 mins	Action

Summary statistics: Total: 1.

Click the **Insight** to execute SQLs; for example:



```
select part_dt, sum(price) as total_sold, count(distinct seller_id) as sellers from kylin_sales group by part_dt order by part_dt
```

Building Cube with Spark

1. Set the `kylin.env.hadoop-conf-dir` property in `kylin.properties`.

```
kylin.env.hadoop-conf-dir=/usr/local/service/hadoop/etc/hadoop
```

2. Check the Spark configuration.

Kylin embeds a Spark binary (v2.1.2) in `$KYLIN_HOME/spark`, and all Spark properties prefixed with `kylin.engine.spark-conf.` can be managed in `$KYLIN_HOME/conf/kylin.properties`. These properties will be extracted and applied when a submitted Spark job is executed; for example, if you configure `kylin.engine.spark-conf.spark.executor.memory=4G`, Kylin will use `-conf spark.executor.memory=4G` as a parameter when executing `spark-submit`.

Before you run Spark cubing, you are recommended to take a look at these configurations and customize them based on your cluster. Below is the recommended configuration with Spark dynamic resource allocation enabled:

```
kylin.engine.spark-conf.spark.master=yarn
kylin.engine.spark-conf.spark.submit.deployMode=cluster
kylin.engine.spark-conf.spark.dynamicAllocation.enabled=true
kylin.engine.spark-conf.spark.dynamicAllocation.minExecutors=1
kylin.engine.spark-conf.spark.dynamicAllocation.maxExecutors=1000
kylin.engine.spark-conf.spark.dynamicAllocation.executorIdleTimeout=300
kylin.engine.spark-conf.spark.yarn.queue=default
kylin.engine.spark-conf.spark.driver.memory=2G
kylin.engine.spark-conf.spark.executor.memory=4G
kylin.engine.spark-conf.spark.yarn.executor.memoryOverhead=1024
kylin.engine.spark-conf.spark.executor.cores=1
kylin.engine.spark-conf.spark.network.timeout=600
kylin.engine.spark-conf.spark.shuffle.service.enabled=true
#kylin.engine.spark-conf.spark.executor.instances=1
kylin.engine.spark-conf.spark.eventLog.enabled=true
kylin.engine.spark-conf.spark.hadoop.dfs.replication=2
kylin.engine.spark-conf.spark.hadoop.mapreduce.output.fileoutputformat.compress
=true
kylin.engine.spark-conf.spark.hadoop.mapreduce.output.fileoutputformat.compres
s.codec=org.apache.hadoop.io.compress.DefaultCodec
kylin.engine.spark-conf.spark.io.compression.codec=org.apache.spark.io.SnappyCo
mpressionCodec
kylin.engine.spark-conf.spark.eventLog.dir=hdfs:\/\/kylin/spark-history
kylin.engine.spark-conf.spark.history.fs.logDirectory=hdfs:\/\/kylin/spark-hist
ory
## Uncommenting for HDP
#kylin.engine.spark-conf.spark.driver.extraJavaOptions=-Dhdp.version=current
#kylin.engine.spark-conf.spark.yarn.am.extraJavaOptions=-Dhdp.version=current
#kylin.engine.spark-conf.spark.executor.extraJavaOptions=-Dhdp.version=current
```

For running on the Hortonworks platform, you need to specify `hdp.version` as the Java option for Yarn container; therefore, you should uncomment the last three lines in `kylin.properties` .

Besides, in order to avoid repeatedly uploading Spark jars to Yarn, you can manually upload them once and then configure the jar's HDFS path. **The HDFS path must be a full path name.**

```
jar cv0f spark-libs.jar -C $KYLIN_HOME/spark/jars/ .
hadoop fs -mkdir -p /kylin/spark/
hadoop fs -put spark-libs.jar /kylin/spark/
```

Then, configure `kylin.properties` as follows:

```
kylin.engine.spark-conf.spark.yarn.archive=hdfs://sandbox.hortonworks.com:8020/kylin/spark/spark-libs.jar
```

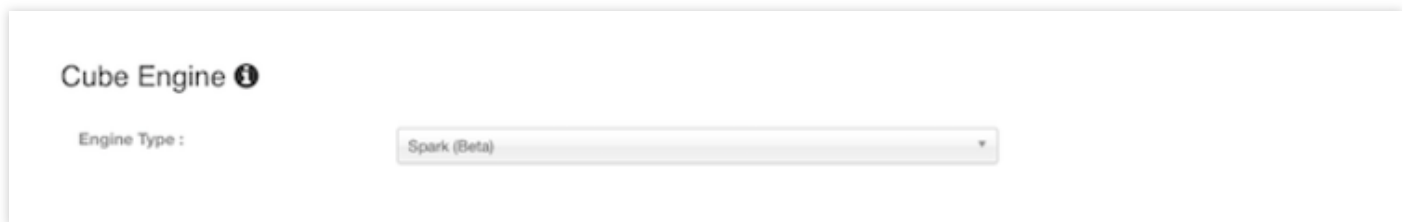
All the `kylin.engine.spark-conf.*` parameters can be overwritten at the cube or project level, which gives you more flexibility.

3. Create and modify a sample cube.

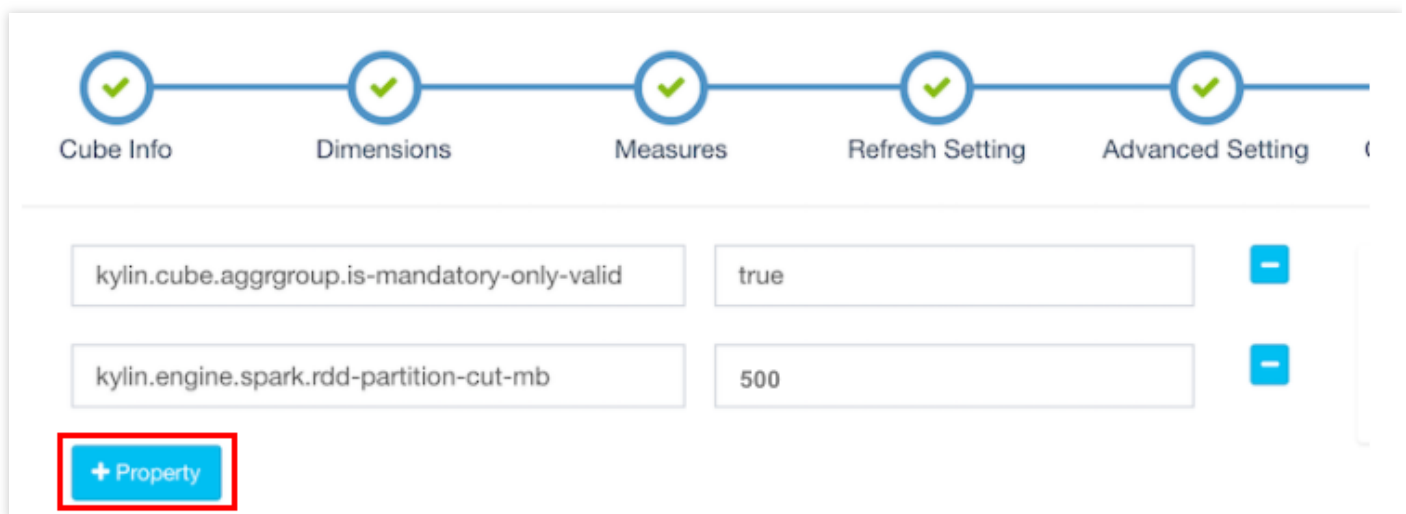
Run `sample.sh` to create a sample cube and then start the Kylin server:

```
/usr/local/service/kylin/bin/sample.sh  
/usr/local/service/kylin/bin/kylin.sh start
```

After Kylin is started, access the Kylin website and edit the `kylin_sales` cube on the "Advanced Setting" page by changing **Cube Engine** from **MapReduce** to **Spark (Beta)**:



Click **Next** to enter the "Configuration Overwrites" page, and click **+Property** to add the `kylin.engine.spark.rdd-partition-cut-mb` property with a value of 500.



The sample cube has two memory hungry measures: `COUNT DISTINCT` and `TOPN(100)`. When the source

data is small, their estimated size will be much larger than their actual size, thus causing more RDD partitions to be split and slowing down the building process. 500 is a reasonable number. Click **Next** and **Save** to save the cube.

For cubes without `COUNT DISTINCT` and `TOPN`, please keep the default configuration.

4. Build a cube with Spark.

Click **Build** and select the current date as the end date. Kylin will generate a building job on the "Monitor" page, in which the 7th step is Spark cubing. The job engine will start to execute the steps in sequence.

The screenshot shows the Kylin Monitor interface. At the top, there is a search bar for 'Cube Name' and a filter for 'Jobs in: LAST ONE WEEK'. Below this, there are checkboxes for job statuses: NEW, PENDING, RUNNING, STOPPED, FINISHED, ERROR, and DISCARDED. A table lists the jobs with columns for Job Name, Cube, Progress, Last Modified Time, Duration, and Actions. One job is highlighted in blue:

Job Name	Cube	Progress	Last Modified Time	Duration	Actions
kylin_sales_cube - 20120101000000_20170301000000 - BUILD - GMT+08:00 2017-03-06 21:58:26	kylin_sales_cube	<div style="width: 100%;"></div>	2017-03-06 21:58:27 GMT+8	0.00 mins	Action

Below the table, a detailed view of the selected job is shown. It indicates that the current step is the 7th step, named 'Build Cube with Spark', with a duration of 0 seconds and a waiting time of 0 seconds.

When Kylin executes this step, you can monitor the status in the Yarn resource manager. Click the "Application Master" link to open the web UI of Spark, which will display the progress and details of each stage.

			type	
<u>application_1488805575687_0009</u>	root	Cubing for:kylin_sales_cube segment b2ba11a5- 3299-4193-b7c0- 38b553a77067	SPARK	default

Showing 1 to 1 of 1 entries

Spark Jobs ^(?)

Total Uptime: 5.0 min
Scheduling Mode: FIFO
Active Jobs: 1
Completed Jobs: 6
[Event Timeline](#)

Active Jobs (1)

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
6	saveAsNewAPIHadoopFile at SparkCubingByLayer.java:287	2017/03/06 14:13:57	47 s	1/8	502/1603

Completed Jobs (6)

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
5	saveAsNewAPIHadoopFile at SparkCubingByLayer.java:287	2017/03/06 14:12:32	1.4 min	2/2 (5 skipped)	816/816 (345 skipped)
4	saveAsNewAPIHadoopFile at SparkCubingByLayer.java:287	2017/03/06 14:11:23	1.1 min	2/2 (4 skipped)	602/602 (116 skipped)
3	saveAsNewAPIHadoopFile at SparkCubingByLayer.java:287	2017/03/06 14:10:40	41 s	2/2 (3 skipped)	315/315 (30 skipped)
2	saveAsNewAPIHadoopFile at SparkCubingByLayer.java:287	2017/03/06 14:10:28	12 s	2/2 (2 skipped)	100/100 (16 skipped)
1	saveAsNewAPIHadoopFile at SparkCubingByLayer.java:287	2017/03/06 14:10:22	5 s	2/2 (1 skipped)	28/28 (2 skipped)
0	saveAsNewAPIHadoopFile at SparkCubingByLayer.java:287	2017/03/06 14:10:08	14 s	2/2	16/16

After all the steps are successfully performed, the cube will become "Ready", and you can perform queries.

Livy Development Guide

Livy Overview

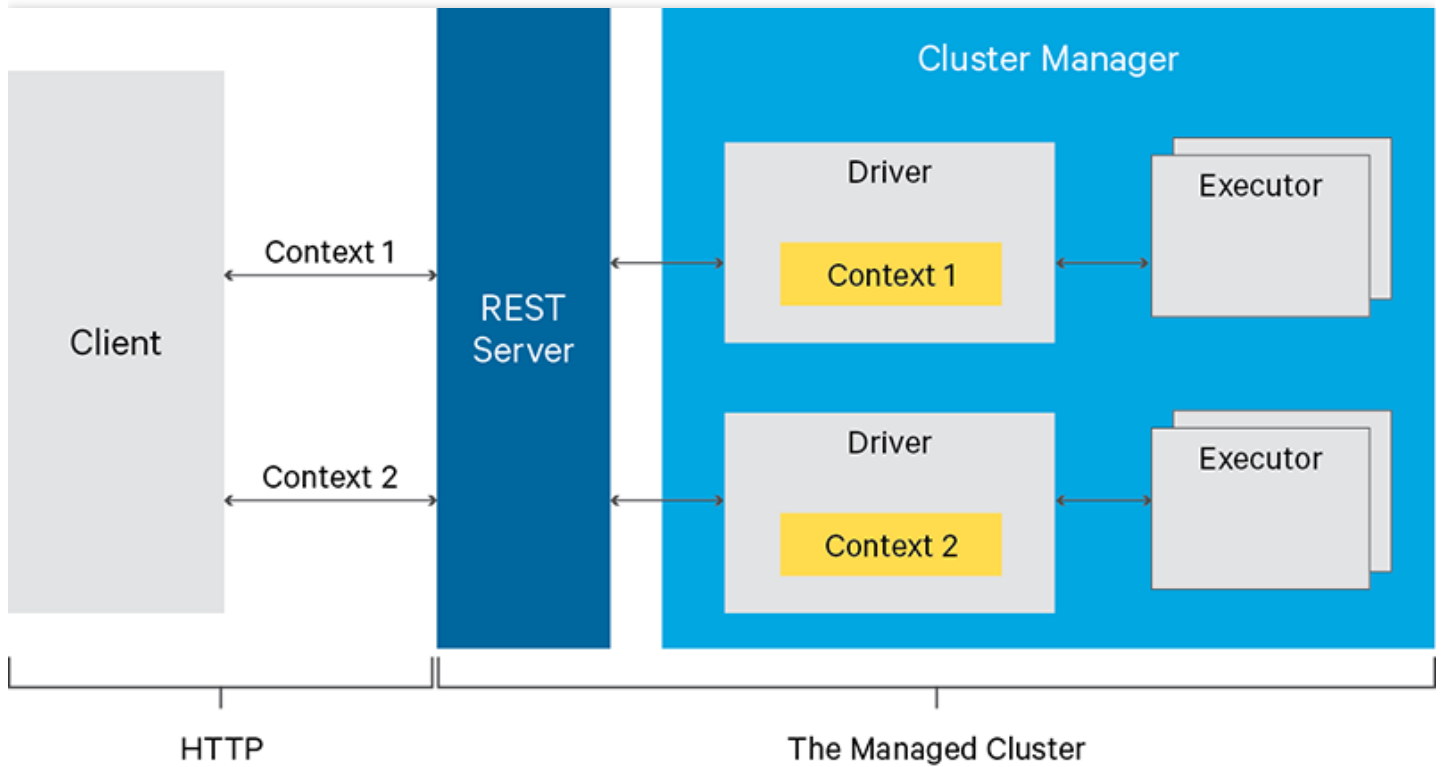
Last updated : 2022-05-24 09:31:34

Apache Livy is a service that enables easy interaction with a Spark cluster over a REST interface. It enables easy submission of Spark jobs or snippets of Spark code, synchronous or asynchronous result retrieval, as well as Spark Context management, all via a simple REST interface or an RPC client library. Apache Livy also simplifies the interaction between Spark and application servers, thus enabling the use of Spark for interactive web/mobile applications.

Livy Features

Additional features include:

- Have long running Spark Contexts that can be used for multiple Spark jobs, by multiple clients.
- Share cached RDDs or Dataframes across multiple jobs and clients.
- Multiple Spark Contexts can be managed simultaneously, and the Spark Contexts run on the cluster (YARN/Mesos) instead of the Livy Server, for good fault tolerance and concurrency.
- Jobs can be submitted as precompiled jars, snippets of code or via java/scala client API
- Ensure security via secure authenticated communication.



Using Livy

1. Access `http://IP:8998/ui` to enter the UI of Livy (**this IP is the public IP. You need to apply for a public IP for the server where Livy is installed and set the security group policy to open the port for access**).
2. Create an interactive session.

```
curl -X POST --data '{"kind":"spark"}' -H "Content-Type:application/json" IP:8998/sessions
```

3. View the alive sessions on Livy.

```
curl IP:8998/sessions
```

4. Execute the code snippet to perform a simple addition operation (here, specify session 0, or specify another session if there are multiple sessions).

```
curl -X POST IP:8998/sessions/0/statements -H "Content-Type:application/json" -d '{"code":"1+1"}'
```

5. Calculate the pi (execute the JAR package).

Step 1. Upload the JAR package to HDFS, such as to `/usr/local/spark-examples_2.11-2.4.3.jar` .

Step 2. Run the following command:

```
curl -H "Content-Type: application/json" -X POST -d '{ "file":"/usr/local/spark-examples_2.11-2.4.3.jar", "className":"org.apache.spark.examples.SparkPi" }' IP:8998/batches
```

6. Check whether the code snippet has been successfully executed. You can also view it on the UI at

`http://IP:8998/ui/session/0` .

```
curl IP:8998/sessions/0/statements/0
```

7. Delete the session.

```
curl -X DELETE IP:8998/sessions/0
```

Notes

Open configuration files

Currently, open configuration files include `livy.conf` and `livy-env.sh` , both of which can be used to modify the configuration through configuration distribution. Check the EMR console for the actual open configuration files.

Changing the port used by Livy

By default, Livy runs on port 8998, which can be changed with the `livy.server.port` configuration option in the `livy.conf` configuration file.

If Hue is installed in the cluster, due to the connectivity between Hue and Livy, the port for Hue also needs to be changed with the `livy_server_port=8998` configuration option in the `pseudo-distributed.ini` configuration file at `/usr/local/service/hue/desktop/conf` . The service needs to be restarted after the change.

Unless necessary, we don't recommend you modify the port used by Livy. Instead, you can control the access by using a security group. Any change may cause other potential problems.

Livy deployment

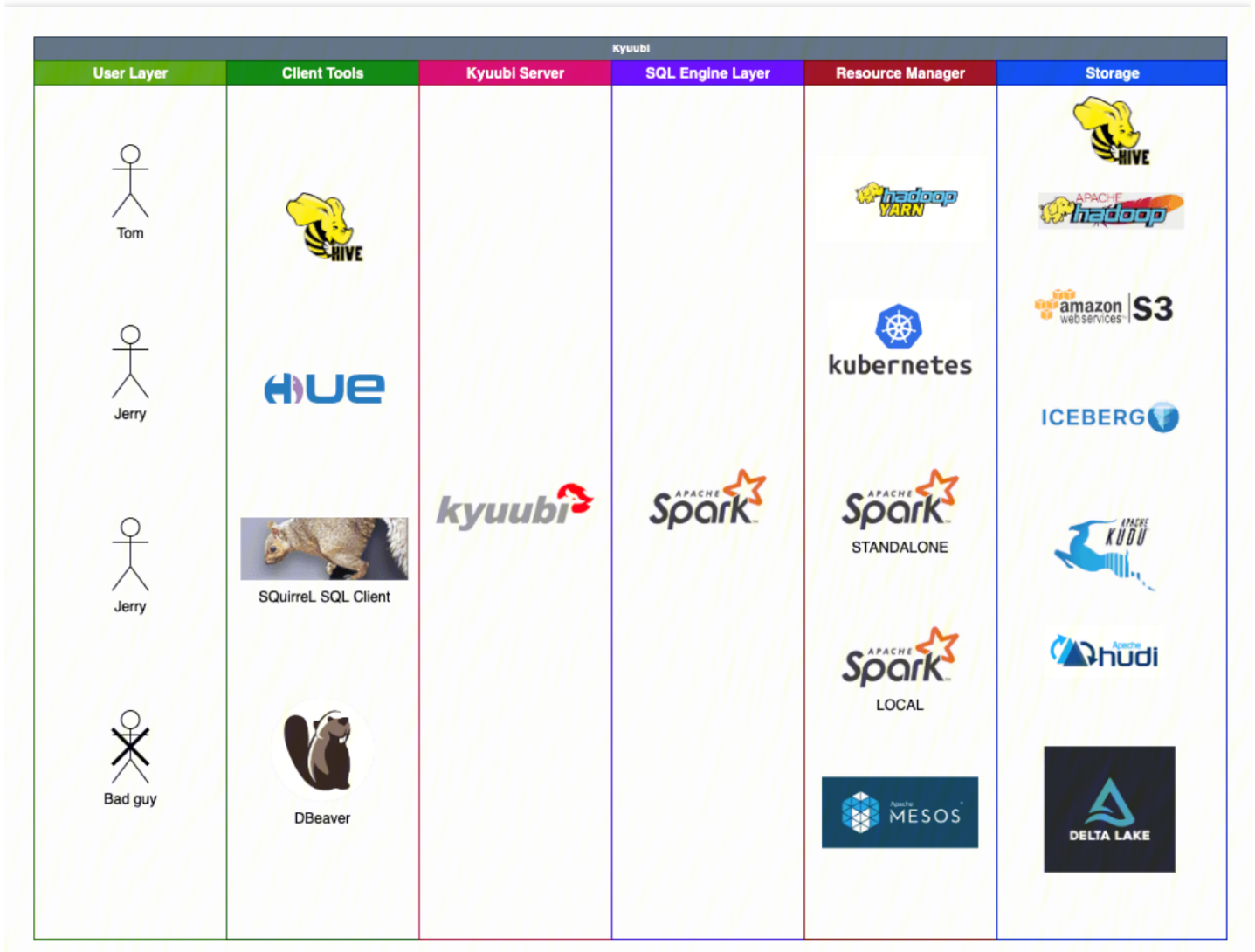
Currently, Livy is deployed on all master nodes by default. You can also deploy it on router nodes by scaling out router nodes.

Kyuubi Development Guide

Kyuubi Overview

Last updated : 2022-05-16 12:52:25

Apache Kyuubi (Incubating) is a distributed multi-tenant Thrift JDBC/ODBC service. Currently, it has been connected to Apache Spark and is being connected to Apache Flink and Trino. It can be used for diverse big data scenarios within enterprises, such as ETL and BI reporting.



Use Cases

- You can replace HiveServer2 to increase the performance by 10–100 times.
 - Kyuubi is highly compatible with HiveServer2 APIs and supports seamless migration.

- The layered architecture of Kyuubi eliminates client compatibility issues and supports imperceptible upgrade.
- Kyuubi supports full-linkage Spark SQL optimization and enhancement for an excellent performance.
- Kyuubi offers various enterprise-grade features, such as high availability, multi-tenancy, and fine-grained authentication.
- You can build a Serverless Spark platform.
 - The goal of serverless Spark is definitely not to let you call Spark APIs or continue to write Spark jobs.
 - The built-in engine module of Kyuubi makes Spark easier to use, and you don't even need to understand the logic.
 - You can focus on your business development by manipulating data through JDBC and SQL. The resources are elastically scalable and Ops-free.
 - Resource managers such as Kubernetes and YARN as well as engine lifecycle are supported. Spark allocates resources dynamically and elastically at three different granularities.
 - You can schedule resources with multiple resource managers such as YARN and Kubernetes simultaneously. This guarantees the secure migration of historical jobs to the cloud.
 - Spark Adaptive Query Execution (AQE) and Kyuubi AQE Plus robustly empower data operations.
- You can build a unified data lake insight, analysis, and management platform (with Kyuubi 1.5 or later).
 - All Spark and third-party data sources are supported.
 - Metadata can be managed through Spark DSv2 to visually build and manage data lakes.
 - All popular data lake frameworks are supported, including Apache Iceberg, Apache Hudi, and Delta Lake.
 - With one copy of data for one API and one engine, Kyuubi provides a unified query, analysis, data ingestion, and data lake management platform.
 - Kyuubi integrates batch processing and stream processing and supports stream operations (upcoming).

Kyuubi Best Practices

Last updated : 2022-05-16 12:52:25

Connecting Beeline to Kyuubi

Log in to a master node in the EMR cluster, switch to the Hadoop user, and go to the Kyuubi directory:

```
[root@172 ~]# su hadoop
[hadoop@172 root]$ cd /usr/local/service/kyuubi
```

Connect to Kyuubi:

```
[hadoop@10kyuubi]$ bin/beeline -u "jdbc:hive2://${zkserverip1}:${zkport},${zkserverip2}:${zkport},${zkserverip3}:${zkport}/default;serviceDiscoveryMode=zooKeeper;zooKeeperNamespace=kyuubi" -n hadoop
```

Or

```
[hadoop@10kyuubi]$ bin/beeline -u "jdbc:hive2://${kyuubiserverip}:${kyuubiserverport}" -n hadoop
```

For more information on `${zkserverip}:${zkport}` , see the `kyuubi.ha.zookeeper.quorum` configuration item of `kyuubi-defaults.conf` .

For more information on `${kyuubiserverport}` , see the `kyuubi.frontend.bind.port` configuration item of `kyuubi-defaults.conf` .

Creating and viewing database

Create a new table in the database you just created and view the table:

```
0: jdbc:hive2://ip:port> create database sparksql;
+-----+
| Result |
+-----+
+-----+
No rows selected (0.326 seconds)
```

Insert two rows of data into the table and view them:

```
0: jdbc:hive2://ip:port> use sparksql;
+-----+
```

```

| Result |
+-----+
+-----+
No rows selected (0.077 seconds)
0: jdbc:hive2://ip:port> create table sparksql_test (a int,b string);
+-----+
| Result |
+-----+
+-----+
No rows selected (0.402 seconds)
0: jdbc:hive2://ip:port> show tables;
+-----+-----+-----+
| database | tableName | isTemporary |
+-----+-----+-----+
| sparksql | sparksql_test | false |
+-----+-----+-----+
1 row selected (0.108 seconds)

```

Connecting Hue to Kyuubi

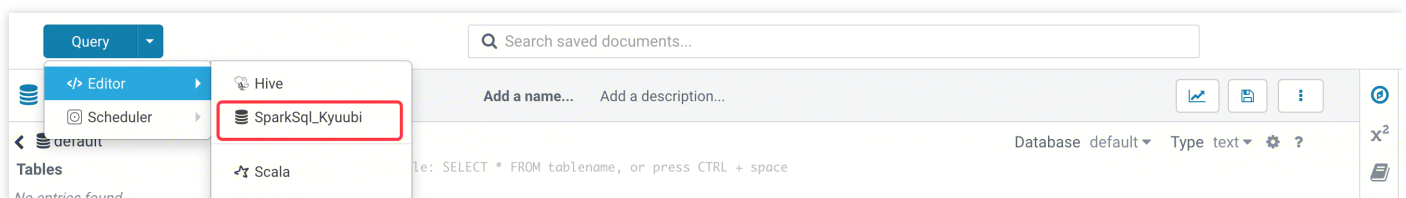
Prerequisites

If Kyuubi is installed in the existing cluster, you need to perform the following operations before you can use Kyuubi on Hue:

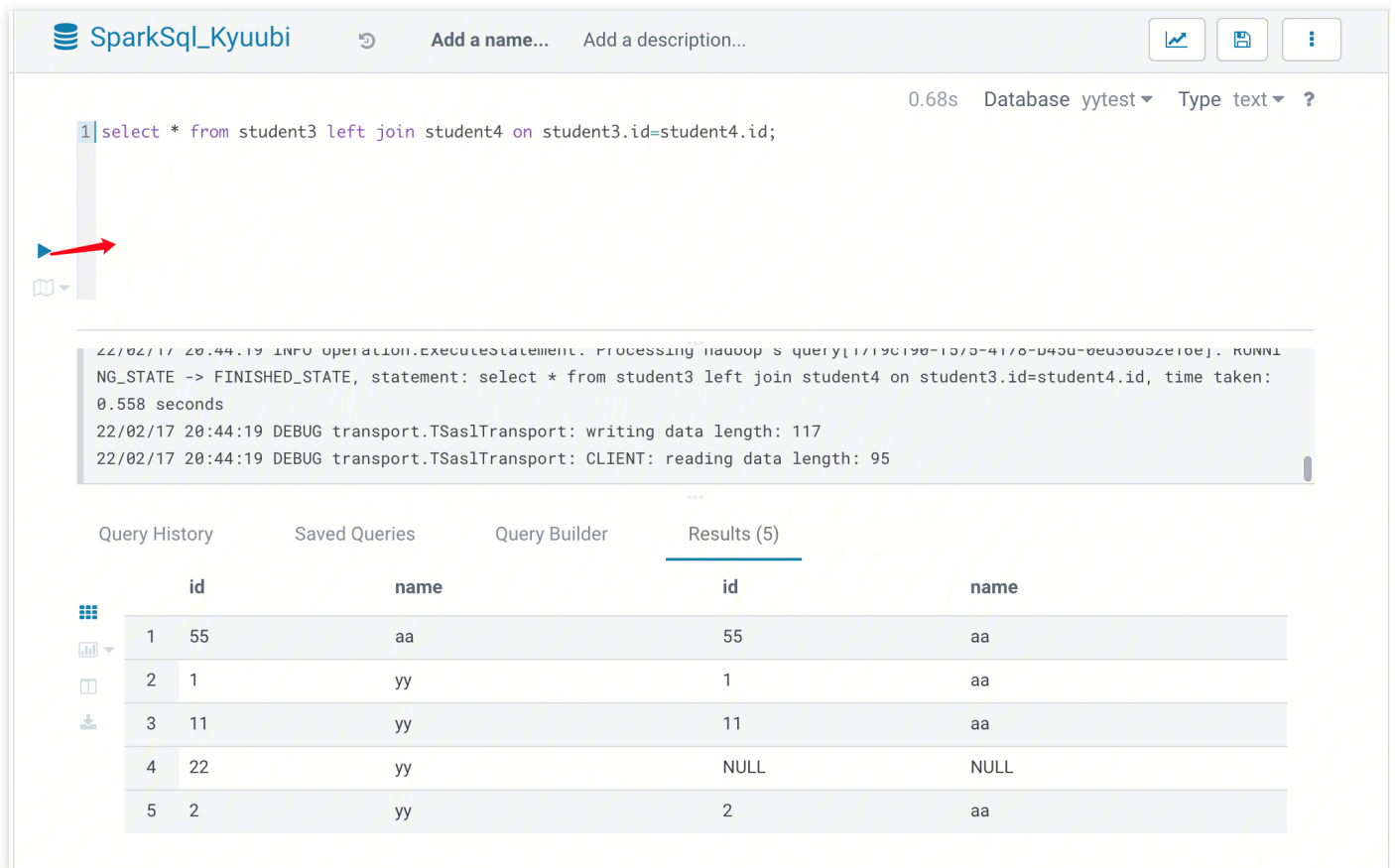
1. Go to the **Configuration Management** page of HDFS, add the `hadoop.proxyuser.hue.groups` and `hadoop.proxyuser.hue.hosts` configuration items to `core-site.xml`, and set their values to `*`.
2. Restart Kyuubi and Hue.
3. Access the Hue console. For more information, see [Hue Development Guide](#).

Kyuubi query

1. At the top of the Hue console, select **Query > Editor > SparkSql_Kyuubi**.



2. Enter the statement to be executed in the statement input box and click **Run** to run it.



The screenshot shows the SparkSQL_Kyuubi interface. At the top, there's a header with the logo and 'Add a name...' and 'Add a description...' fields. Below that, a SQL query is entered in a text box: `select * from student3 left join student4 on student3.id=student4.id;`. A red arrow points to the 'Run' button. Below the query, there's a log output showing the execution details, including the time taken (0.558 seconds) and some debug messages. At the bottom, there's a 'Results (5)' section with a table showing the output of the query.

	id	name	id	name
1	55	aa	55	aa
2	1	yy	1	aa
3	11	yy	11	aa
4	22	yy	NULL	NULL
5	2	yy	2	aa

Connecting to Kyuubi Through Java

KyuubiServer is integrated with the Thrift service. Thrift, created by Facebook, is an interface definition language and binary communication protocol used for defining and creating services for numerous languages. Apache Kyuubi is based on Thrift, allowing many languages such as Java and Python to call Kyuubi's APIs. Additionally, the Hive JDBC driver for Kyuubi enables Java applications to interact with Kyuubi. This section describes how to connect to Kyuubi through Java code.

1. Prepare for development.

- Confirm that you have activated EMR and created an EMR cluster. When creating the EMR cluster, select the Kyuubi and Spark components on the software configuration page.
- Kyuubi and its dependencies are installed in the EMR cluster directory `/usr/local/service/`.

2. Use Maven to create a project.

Maven is recommended for project management, as it can help you manage project dependencies with ease.

Specifically, it can get JAR packages through the configuration of the `pom.xml` file, eliminating your need to add them manually. Download and install Maven locally first and then configure its environment variables. If you are using the IDE, set the Maven-related configuration items in the IDE.

In the local shell environment, enter the directory where you want to create the Maven project, such as

`D://mavenWorkplace` , and enter the following command to create it:

```
mvn archetype:generate -DgroupId=$yourgroupId -DartifactId=$yourartifactId -DarchetypeArtifactId=maven-archetype-quickstart
```

Here, `$yourgroupId` is your package name, `$yourartifactId` is your project name, and `maven-archetype-quickstart` indicates to create a Maven Java project. Some files need to be downloaded for creating the project, so stay connected to the internet. After successfully creating the project, you will see a folder named `$yourartifactId` in the `D://mavenWorkplace` directory. The files included in the folder have the following structure:

```
simple
---pom.xml      Core configuration, under the project root directory
---src
---main
---java        Java source code directory
---resources   Java configuration file directory
---test
---java        Test source code directory
---resources   Test configuration directory
```

Among the above files, pay extra attention to the `pom.xml` file and the Java folder under the main directory. The `pom.xml` file is primarily used to create dependencies and package configurations; the Java folder is used to store your source code. First, add the Maven dependencies to `pom.xml` :

```
<dependencies>
<dependency>
<groupId>org.apache.kyuubi</groupId>
<artifactId>kyuubi-hive-jdbc-shaded</artifactId>
<version>1.4.1-incubating</version>
</dependency>
<dependency>
<groupId>org.apache.hadoop</groupId>
<artifactId>hadoop-common</artifactId>
<!-- keep consistent with the build hadoop version -->
<version>2.8.5</version>
</dependency>
</dependencies>
```

Then, add the packaging and compiling plugins to pom.xml:

```
<build>
<plugins>
<plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-compiler-plugin</artifactId>
<configuration>
<source>1.8</source>
<target>1.8</target>
<encoding>utf-8</encoding>
</configuration>
</plugin>
<plugin>
<artifactId>maven-assembly-plugin</artifactId>
<configuration>
<descriptorRefs>
<descriptorRef>jar-with-dependencies</descriptorRef>
</descriptorRefs>
</configuration>
<executions>
<execution>
<id>make-assembly</id>
<phase>package</phase>
<goals>
<goal>single</goal>
</goals>
</execution>
</executions>
</plugin>
</plugins>
</build>
```

Right-click in src>main>Java and create a Java Class. Enter the Class name (e.g., `KyuubiJDBCTest.java` here) and add the sample code to the Class:

```
import java.sql.*;
public class KyuubiJDBCTest {
private static String driverName =
"org.apache.kyuubi.jdbc.KyuubiHiveDriver";
public static void main(String[] args)
throws SQLException {
try {
Class.forName(driverName);
} catch (ClassNotFoundException e) {
```



```
e.printStackTrace();
System.exit(1);
}
Connection con = DriverManager.getConnection(
"jdbc:hive2://$kyuubiserverhost:$kyuubiserverport/default", "hadoop", "");
Statement stmt = con.createStatement();
String tableName = "KyuubiTestByJava";
stmt.execute("drop table if exists " + tableName);
stmt.execute("create table " + tableName +
" (key int, value string)");
System.out.println("Create table success!");
// show tables
String sql = "show tables '" + tableName + "'";
System.out.println("Running: " + sql);
ResultSet res = stmt.executeQuery(sql);
if (res.next()) {
System.out.println(res.getString(1));
}
// describe table
sql = "describe " + tableName;
System.out.println("Running: " + sql);
res = stmt.executeQuery(sql);
while (res.next()) {
System.out.println(res.getString(1) + "\t" + res.getString(2));
}
sql = "insert into " + tableName + " values (42,\"hello\"), (48,\"world\")";
stmt.execute(sql);
sql = "select * from " + tableName;
System.out.println("Running: " + sql);
res = stmt.executeQuery(sql);
while (res.next()) {
System.out.println(String.valueOf(res.getInt(1)) + "\t"
+ res.getString(2));
}
sql = "select count(1) from " + tableName;
System.out.println("Running: " + sql);
res = stmt.executeQuery(sql);
while (res.next()) {
System.out.println(res.getString(1));
}
}
}
```

Note :

The `$kyuubiserverhost` and `$kyuubiserverport` parameters in the program should be replaced with the values of the IP and port number of the KyuubiServer you queried.

If your Maven is configured correctly and its dependencies are imported successfully, the project can be compiled directly. Enter the project directory in the local shell and run the following command to package the entire project:

```
mvn package
```

3. Upload and run the program.

First, use the SCP or SFTP tool to upload the compressed JAR package to the EMR cluster. Run the following command in your local shell:

```
scp $localfile root@public IP address:/usr/local/service/kyuubi
```

Be sure to upload a JAR package that contains the dependencies. Log in to the EMR cluster, switch to the Hadoop user, and go to the `/usr/local/service/kyuubi` directory. Then, you can run the following program:

```
[hadoop@172 kyuubi]$ yarn jar $package.jar KyuubiJDBCTest
```

Here, `$package.jar` is the path plus name of your JAR package, and `KyuubiJDBCTest` is the name of the previously created Java Class. The result is as follows:

```
Create table success!
Running: show tables 'KyuubiTestByJava'
default
Running: describe KyuubiTestByJava
key int
value string
Running: select * from KyuubiTestByJava
42 hello
48 world
Running: select count(1) from KyuubiTestByJava
2
```

Zeppelin Development Guide

Zeppelin Overview

Last updated : 2022-05-16 12:52:26

Apache Zeppelin is a web-based notebook that enables interactive data analysis. It allows you to create interactive collaborative documents with various prebuilt language backends (or interpreters), such as Scala (Apache Spark), Python (Apache Spark), Spark SQL, Hive, and Shell.

Prerequisites

- You have created a cluster and selected the Zeppelin service. For more information, see [Creating EMR Cluster](#).
- In the cluster's EMR security group, open ports 22 and 30001 (which are opened by default for new clusters) and the necessary IP ranges for communication over the private network. The new security group is named in the format of `emr-xxxxxxxx_yyyyMMdd`, which shouldn't be modified manually.
- Add services as needed, such as Spark, Flink, HBase, and Kylin.

Logging in to Zeppelin

1. Create a cluster and select the Zeppelin service. For more information, see [Creating EMR Cluster](#).
2. On the left sidebar in the [EMR console](#), select **Cluster Service**.
3. Click the Zeppelin block and click **Web UI Address** to access the web UI.
4. For EMR versions later than 3.1.0, the default login permission is set, and both the username and password are `admin`. To change the password, modify the `users` and `roles` options in the `/usr/local/service/zeppelin-0.8.2/conf/shiro.ini` configuration file. For more configuration instructions, see [Apache Shiro Configuration](#).
5. In EMR 2.6.0 and 3.3.0, Zeppelin login is integrated into the OpenLDAP account, so you can log in only with an OpenLDAP account and password. After a cluster is created, the default OpenLDAP accounts are `root` and `hadoop`, and the default password is the cluster password. Only the `root` account has the Zeppelin admin permissions and has access to the interpreter configuration page.

Performing `wordcount` with Spark

1. Click **Create new note** on the left of the page to create a notebook in the pop-up window.

Create new note

Note Name: test

Default Interpreter: spark

Use '/' to create folders. Example: /NoteDirA/Note1

Create Note

2. Configure Spark for integration with an EMR cluster (Spark on YARN). Modify and save the configuration.

spark %spark, %sq, %dep, %pyspark, %byspark, %r

Option

The interpreter will be instantiated Globaly in shared process

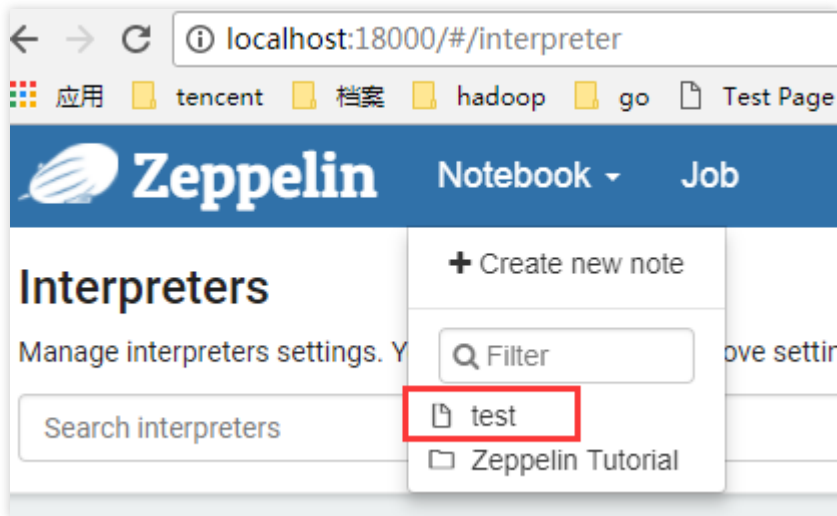
Connect to existing process

Set permission

Properties

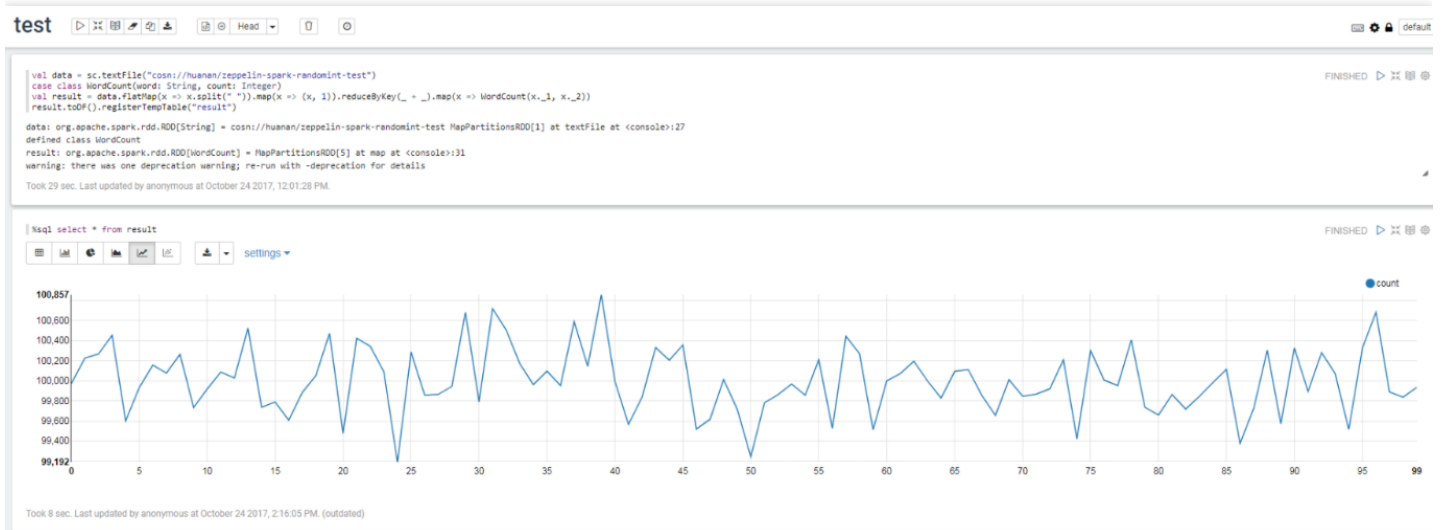
name	value
algse	
master	yarn-cluster
spark.app.name	Zeppelin
spark.cores.max	
spark.executor.memory	
zeppelin.R.cmd	R
zeppelin.R.image.width	100%
zeppelin.R.knitr	true
zeppelin.R.render.options	out.format = 'html', comment = NA, echo = FALSE, results = 'asis', message = F, warning = F, fig.retina = 2
zeppelin.dep.additionalRemoteRepository	spark-packages,http://dl.bintray.com/spark-packages/maven,false;
zeppelin.dep.localrepo	local-repo
zeppelin.pyspark.python	python
zeppelin.pyspark.useIPython	true
zeppelin.spark.concurrentSQL	false
zeppelin.spark.enableSupportedVersionCheck	true
zeppelin.spark.importImplicit	true
zeppelin.spark.maxResult	1000
zeppelin.spark.printREPLOutput	true
zeppelin.spark.sql.interpolation	false
zeppelin.spark.sql.stacktrace	false
zeppelin.spark.ui.hidden	false
zeppelin.spark.ui.WebUI	
zeppelin.spark.useHiveContext	true

3. Go to your own notebook.



4. Write a wordcount program and run the following commands:

```
val data = sc.textFile("cosn://huanan/zeppelin-spark-randomint-test")
case class WordCount(word: String, count: Integer)
val result = data.flatMap(x => x.split(" ")).map(x => (x, 1)).reduceByKey(_ + _)
                    .map(x => WordCount(x._1, x._2))
result.toDF().registerTempTable("result")
%sql select * from result
```



Zeppelin Interpreter Configuration

Last updated : 2022-05-16 12:52:26

This document describes how to configure and verify common Zeppelin interpreters.

Spark Interpreter

Configuration

```
SPARK_HOME: /usr/local/service/spark
spark.master: yarn
spark.submit.deployMode: cluster
spark.app.name: zeppelin-spark
```

Verification

1. Upload the `wordcount.txt` file to the `/tmp` path of `emr hdfs` first.
2. Find the `hdfs://HDFS45983` value of the `fs.defaultFS` configuration item in `core-site.xml`.
3. Run the Spark code in the notebook.

```
%spark
val data = sc.textFile("hdfs://HDFS45983/tmp/wordcount.txt")
case class WordCount(word: String, count: Integer)
val result = data.flatMap(x => x.split(" ")).map(x => (x, 1)).reduceByKey(_ + _)
  .map(x => WordCount(x._1, x._2))
result.toDF().registerTempTable("result")
%sql
select * from result
```

Flink Interpreter

Configuration

```
FLINK_HOME: /usr/local/service/flink
flink.execution.mode: yarn
```

Verification

```
%flink
val data = benv.fromElements("hello world", "hello flink", "hello hadoop")
data.flatMap(line => line.split("\\s"))
  .map(w => (w, 1))
  .groupBy(0)
  .sum(1)
  .print()
```

HBase Interpreter

Configuration

```
hbase.home: /usr/local/service/hbase
hbase.ruby.sources: lib/ruby
zeppelin.hbase.test.mode: false
```

Note :

As the JAR packages depended on by this interpreter have been integrated into the

`/usr/local/service/zeppelin/local-repo` path of the cluster, you don't need to configure dependencies. They are required only if you want to define JAR packages.

Verification

```
%hbase
help 'get'
%hbase
list
```

Livy Interpreter

Configuration

```
zeppelin.livy.url: http://ip:8998
```

Verification

```
%livy.spark
sc.version
%livy.pyspark
print "1"
%livy.sparkr
hello <- function( name ) {
  sprintf( "Hello, %s", name );
}
hello("livy")
```

Kylin Interpreter

Configuration

1. Create a default project in the Kylin console.
2. Configure the Kylin interpreter for Zeppelin.

```
kylin.api.url: http://ip:16500/kylin/api/query
kylin.api.user: ADMIN
kylin.api.password: KYLIN
kylin.query.project: default
```

Verification

```
%kylin(default)
select count(*) from table1
```

JDBC Interpreter

1. MySQL interpreter configuration

```
default.url: jdbc:mysql://ip:3306
default.user: xxx
default.password: xxx
default.driver: com.mysql.jdbc.Driver
```

Note :

As the JAR packages depended on by this interpreter have been integrated into the `/usr/local/service/zeppelin/local-repo` path of the cluster, you don't need to configure dependencies. They are required only if you want to define JAR packages.

Verification

```
%mysql
show databases
```

2. Hive interpreter configuration

```
default.url: jdbc:hive2://ip:7001
default.user: hadoop
default.password:
default.driver: org.apache.hive.jdbc.HiveDriver
```

Note :

As the JAR packages depended on by this interpreter have been integrated into the `/usr/local/service/zeppelin/local-repo` path of the cluster, you don't need to configure dependencies. They are required only if you want to define JAR packages.

Verification

```
%hive
show databases
%hive
use default;
show tables;
```

3. Presto interpreter configuration

```
default.url: jdbc:presto://ip:9000?user=hadoop
default.user: hadoop
default.password:
default.driver: io.prestosql.jdbc.PrestoDriver
```

Note :

As the JAR packages depended on by this interpreter have been integrated into the `/usr/local/service/zeppelin/local-repo` path of the cluster, you don't need to configure dependencies. They are required only if you want to define JAR packages.

Verification

```
%presto
show catalogs;
%presto
show schemas from hive;
%presto
show tables from hive.default;
```

For more information, see [Generic JDBC Interpreter for Apache Zeppelin](#).

Hudi Development Guide

Hudi Overview

Last updated : 2020-09-27 15:13:25

Apache Hudi provides the following streaming primitives over datasets on HDFS: upsert and incremental pull.

In general, we will store large amounts of data in HDFS and incrementally write new data but seldom change old data, particularly in scenarios where data is cleansed and placed in a data warehouse. In data warehouses such as Hive, the support for updates is very limited, and computing is expensive. In addition, in scenarios where only data that is added over a certain period of time needs to be analyzed, as Hive, Presto, and HBase do not provide native methods, it is necessary to filter and analyze it based on timestamp.

In view of this, Hudi enables you to update records and only query incremental data. In addition, it supports Hive, Presto, and Spark, so that you can directly use these components to query data managed by Hudi.

Hudi is a universal big data storage system that enables:

- Snapshot isolation between ingestion and query engines, including Apache Hive, Presto, and Apache Spark.
- Support for rollbacks and savepoints to recover datasets.
- Auto-management of file sizes and layout to optimize query performance and near real-time ingestion to feed queries with fresh data.
- Async compaction of both real-time and columnar data.

Timeline

At its core, Hudi maintains a **timeline** of all actions performed on the dataset at different **instants of time** that helps provide instantaneous views of the dataset.

A Hudi instant consists of the following components:

- Action type: type of action performed on the dataset.
- Instant time: typically a timestamp (such as 20190117010349), which monotonically increases in the order of action start time.
- State: current state of the instant.

File Organization

Hudi organizes datasets on DFS into a directory structure under a `basepath` . A dataset is broken down into partitions, which are folders containing data files for the corresponding partitions, very similar to Hive tables.

Each partition is uniquely identified by its `partitionpath` , which is relative to the `basepath` .

Within each partition, files are organized into `file groups` uniquely identified by `file ID` .

Each file group contains several `file slices` , where each slice contains a base columnar file (`*.parquet`) produced at a certain commit/compaction instant time, along with a set of log files (`*.log*`) that contain inserts/updates to the base file since the base file was produced.

Hudi adopts an MVCC design, where compaction action merges logs and base files to produce new file slices and cleaning action gets rid of unused/older file slices to reclaim space on DFS.

Hudi provides efficient upserts by mapping a given hoodie key (record key + partition path) to a file group through an indexing mechanism.

This mapping between record key and `file group / file id` never changes once the first version of a record has been written to a file. In short, the mapped file group contains all versions of a group of records.

Storage Types

Hudi supports the following storage types:

- Copy on write: it stores data only in columnar file formats (e.g., parquet), updates version, and rewrites the files by performing a sync merge during write.
- Merge on read: it stores data by using a combination of columnar (e.g., parquet) and row-based (e.g., avro) file formats. Updates are logged to incremental files and later compacted to produce new versions of columnar files synchronously or asynchronously.

The following table summarizes the trade-offs between these two storage types:

Trade-Off	Copy on Write	Merge on Read
Data latency	Higher	Lower
Update cost (I/O)	Higher (rewriting entire parquet)	Lower (appending to incremental log)
Parquet file size	Smaller (high update cost (I/O))	Larger (low update cost)
Write amplification	Higher	Lower (depending on compaction policy)

Hudi Support for EMR Underlying Storage

- HDFS
- COS

Installing Hudi

Enter the [EMR purchase page](#) and select EMR v2.2.0 as the **Product Version** and **Hudi 0.5.1** as the **Optional Component**. Hudi is installed on the master and router nodes by default.

Hudi relies on Hive and Spark components. If you choose to install Hudi, EMR will automatically install Hive and Spark.

Use Cases

For more information, please see [Hudi official demo](#):

1. Log in to a master node and switch to the `hadoop` user.
2. Load the Spark configuration.

```
cd /usr/local/service/hudi
ln -s /usr/local/service/spark/conf/spark-defaults.conf /usr/local/service/hudi
/demo/config/spark-defaults.conf
```

Upload the configuration to HDFS:

```
hdfs dfs -mkdir -p /hudi/config
hdfs dfs -copyFromLocal demo/config/* /hudi/config/
```

3. Modify the Kafka data source.

```
/usr/local/service/hudi/demo/config/kafka-source.properties
bootstrap.servers=kafka_ip:kafka_port
```

Upload the first batch of data:

```
cat demo/data/batch_1.json | kafkacat -b [kafka_ip] -t stock_ticks -P
```

4. Ingest the first batch of data.

```
spark-submit --class org.apache.hudi.utilities.deltastreamer.HoodieDeltaStream
er --master yarn ./hudi-utilities-bundle_2.11-0.5.1-incubating.jar --table-type
COPY_ON_WRITE --source-class org.apache.hudi.utilities.sources.JsonKafkaSource
--source-ordering-field ts --target-base-path /usr/hive/warehouse/stock_ticks_c
ow --target-table stock_ticks_cow --props /hudi/config/kafka-source.properties
--schemaprovider-class org.apache.hudi.utilities.schema.FilebasedSchemaProvider
spark-submit --class org.apache.hudi.utilities.deltastreamer.HoodieDeltaStream
er --master yarn ./hudi-utilities-bundle_2.11-0.5.1-incubating.jar --table-type
MERGE_ON_READ --source-class org.apache.hudi.utilities.sources.JsonKafkaSource
--source-ordering-field ts --target-base-path /usr/hive/warehouse/stock_ticks_m
or --target-table stock_ticks_mor --props /hudi/config/kafka-source.properties
--schemaprovider-class org.apache.hudi.utilities.schema.FilebasedSchemaProvider
--disable-compaction
```

5. View HDFS data.

```
hdfs dfs -ls /usr/hive/warehouse/
```

6. Sync Hive metadata.

```
bin/run_sync_tool.sh --jdbc-url jdbc:hive2://[hiveserver2_ip:hiveserver2_port]
--user hadoop --pass [password] --partitioned-by dt --base-path /usr/hive/wareh
ouse/stock_ticks_cow --database default --table stock_ticks_cow
bin/run_sync_tool.sh --jdbc-url jdbc:hive2://[hiveserver2_ip:hiveserver2_port]
--user hadoop --pass [password] --partitioned-by dt --base-path /usr/hive/wareh
ouse/stock_ticks_mor --database default --table stock_ticks_mor --skip-ro-suffix
```

7. Query data with a computing engine.

◦ Hive engine

```
beeline -u jdbc:hive2://[hiveserver2_ip:hiveserver2_port] -n hadoop --hivecon
f hive.input.format=org.apache.hadoop.hive ql.io.HiveInputFormat --hiveconf h
ive.stats.autogather=false
```

Or Spark engine

```
spark-sql --master yarn --conf spark.sql.hive.convertMetastoreParquet=false
```

In the Hive/Spark engine, execute the following SQL statement:

```
select symbol, max(ts) from stock_ticks_cow group by symbol HAVING symbol =
'GOOG';
select `_hoodie_commit_time`, symbol, ts, volume, open, close from stock_tick
s_cow where symbol = 'GOOG';
select symbol, max(ts) from stock_ticks_mor group by symbol HAVING symbol =
'GOOG';
select `_hoodie_commit_time`, symbol, ts, volume, open, close from stock_tick
```

```
s_mor where symbol = 'GOOG';
select symbol, max(ts) from stock_ticks_mor_rt group by symbol HAVING symbol
= 'GOOG';
select ` _hoodie_commit_time`, symbol, ts, volume, open, close from stock_tick
s_mor_rt where symbol = 'GOOG';
```

- o Enter the Presto engine

```
/usr/local/service/presto-client/presto --server localhost:9000 --catalog hiv
e --schema default --user Hadoop
```

You need to enclose a field with underlines in double quotation marks to query it with Presto, such as

"_hoodie_commit_time". Execute the following SQL statement:

```
select symbol, max(ts) from stock_ticks_cow group by symbol HAVING symbol = 'GOO
G';
select "_hoodie_commit_time", symbol, ts, volume, open, close from stock_ticks_co
w where symbol = 'GOOG';
select symbol, max(ts) from stock_ticks_mor group by symbol HAVING symbol = 'GOO
G';
select "_hoodie_commit_time", symbol, ts, volume, open, close from stock_ticks_mo
r where symbol = 'GOOG';
select symbol, max(ts) from stock_ticks_mor_rt group by symbol HAVING symbol = 'G
OOG';
```

8. Upload the second batch of data.

```
cat demo/data/batch_2.json | kafkacat -b 10.0.1.70 -t stock_ticks -P
```

9. Ingest the second batch of incremental data.

```
spark-submit --class org.apache.hudi.utilities.deltastreamer.HoodieDeltaStreame
r --master yarn ./hudi-utilities-bundle_2.11-0.5.1-incubating.jar --table-type
COPY_ON_WRITE --source-class org.apache.hudi.utilities.sources.JsonKafkaSource
--source-ordering-field ts --target-base-path /usr/hive/warehouse/stock_ticks_c
ow --target-table stock_ticks_cow --props /hudi/config/kafka-source.properties
--schemaprovider-class org.apache.hudi.utilities.schema.FilebasedSchemaProvider
spark-submit --class org.apache.hudi.utilities.deltastreamer.HoodieDeltaStreame
r --master yarn ./hudi-utilities-bundle_2.11-0.5.1-incubating.jar --table-type
MERGE_ON_READ --source-class org.apache.hudi.utilities.sources.JsonKafkaSource
--source-ordering-field ts --target-base-path /usr/hive/warehouse/stock_ticks_m
or --target-table stock_ticks_mor --props /hudi/config/kafka-source.properties
--schemaprovider-class org.apache.hudi.utilities.schema.FilebasedSchemaProvider
--disable-compactio
```

0. Query incremental data by following step 7.

1. Use the hudi-cli tool.

```
cli/bin/hudi-cli.sh
connect --path /usr/hive/warehouse/stock_ticks_mor
compactions show all
compaction schedule
Combine the execution plans
compaction run --compactionInstant [requestID] --parallelism 2 --sparkMemory 1G
--schemaFilePath /hudi/config/schema.avsc --retry 1
```

2. Query data with the Spark/Tez engine.

```
beeline -u jdbc:hive2://[hiveserver2_ip:hiveserver2_port] -n hadoop --hiveconf
hive.input.format=org.apache.hadoop.hive.ql.io.HiveInputFormat --hiveconf hive.
stats.autogather=false
set hive.execution.engine=tez;
set hive.execution.engine=spark;
```

Then, execute an SQL query by following step 7.

Working with COS

Like HDFS, you need to add `cosn://[bucket]` before the storage path. Please see the following steps:

```
bin/kafka-server-start.sh config/server.properties &
cat demo/data/batch_1.json | kafkacat -b kafkaip -t stock_ticks -P
cat demo/data/batch_2.json | kafkacat -b kafkaip -t stock_ticks -P
kafkacat -b kafkaip -L
hdfs dfs -mkdir -p cosn://[bucket]/hudi/config
hdfs dfs -copyFromLocal demo/config/* cosn://[bucket]/hudi/config/
```

```
spark-submit --class org.apache.hudi.utilities.deltastreamer.HoodieDeltaStreamer
--master yarn ./hudi-utilities-bundle_2.11-0.5.1-incubating.jar --table-type COPY
_ON_WRITE --source-class org.apache.hudi.utilities.sources.JsonKafkaSource --sour
ce-ordering-field ts --target-base-path cosn://[bucket]/usr/hive/warehouse/stock_
ticks_cow --target-table stock_ticks_cow --props cosn://[bucket]/hudi/config/kafk
a-source.properties --schemaprovider-class org.apache.hudi.utilities.schema.Fileb
asedSchemaProvider
```

```
spark-submit --class org.apache.hudi.utilities.deltastreamer.HoodieDeltaStreamer
--master yarn ./hudi-utilities-bundle_2.11-0.5.1-incubating.jar --table-type MERG
E_ON_READ --source-class org.apache.hudi.utilities.sources.JsonKafkaSource --sour
ce-ordering-field ts --target-base-path cosn://[bucket]/usr/hive/warehouse/stock_
```



```
ticks_mor --target-table stock_ticks_mor --props cosn://[bucket]/hudi/config/kafka-source.properties --schemaprovider-class org.apache.hudi.utilities.schema.FilebasedSchemaProvider --disable-compaction
```

```
bin/run_sync_tool.sh --jdbc-url jdbc:hive2://[hiveserver2_ip:hiveserver2_port] --user hadoop --pass isd@cloud --partitioned-by dt --base-path cosn://[bucket]/usr/hive/warehouse/stock_ticks_cow --database default --table stock_ticks_cow
```

```
bin/run_sync_tool.sh --jdbc-url jdbc:hive2://[hiveserver2_ip:hiveserver2_port] --user hadoop --pass hive --partitioned-by dt --base-path cosn://[bucket]/usr/hive/warehouse/stock_ticks_mor --database default --table stock_ticks_mor --skip-ro-suffix
```

```
beeline -u jdbc:hive2://[hiveserver2_ip:hiveserver2_port] -n hadoop --hiveconf hive.input.format=org.apache.hadoop.hive.ql.io.HiveInputFormat --hiveconf hive.stats.autogather=false
```

```
spark-sql --master yarn --conf spark.sql.hive.convertMetastoreParquet=false
```

hivesqls:

```
select symbol, max(ts) from stock_ticks_cow group by symbol HAVING symbol = 'GOOG';
select `_hoodie_commit_time`, symbol, ts, volume, open, close from stock_ticks_cow where symbol = 'GOOG';
select symbol, max(ts) from stock_ticks_mor group by symbol HAVING symbol = 'GOOG';
select `_hoodie_commit_time`, symbol, ts, volume, open, close from stock_ticks_mor where symbol = 'GOOG';
select symbol, max(ts) from stock_ticks_mor_rt group by symbol HAVING symbol = 'GOOG';
select `_hoodie_commit_time`, symbol, ts, volume, open, close from stock_ticks_mor_rt where symbol = 'GOOG';
```

prestoqls:

```
/usr/local/service/presto-client/presto --server localhost:9000 --catalog hive --schema default --user Hadoop
select symbol, max(ts) from stock_ticks_cow group by symbol HAVING symbol = 'GOOG';
select "_hoodie_commit_time", symbol, ts, volume, open, close from stock_ticks_cow where symbol = 'GOOG';
select symbol, max(ts) from stock_ticks_mor group by symbol HAVING symbol = 'GOOG';
select "_hoodie_commit_time", symbol, ts, volume, open, close from stock_ticks_mor where symbol = 'GOOG';
```

```
select symbol, max(ts) from stock_ticks_mor_rt group by symbol HAVING symbol = 'GOOG';  
select "_hoodie_commit_time", symbol, ts, volume, open, close from stock_ticks_mor_rt where symbol = 'GOOG';
```

```
cli/bin/hoodie-cli.sh
```

```
connect --path cosn://[bucket]/usr/hive/warehouse/stock_ticks_mor
```

```
compactions show all
```

```
compaction schedule
```

```
compaction run --compactionInstant [requestid] --parallelism 2 --sparkMemory 1G --schemaFilePath cosn://[bucket]/hoodie/config/schema.avsc --retry 1
```

Superset Development Guide

Superset Overview

Last updated : 2022-05-16 12:52:26

Apache Superset is a web-based data browsing and visualization application. Superset on EMR supports MySQL, Hive, Presto, Impala, Kylin, Druid, and ClickHouse.

Superset Features

- Supports almost all major databases such as MySQL, PostgreSQL, Oracle, SQL Server, SQLite, and Spark SQL as well as [Druid](#).
- Provides a wide variety of visual displays and allows you to create custom dashboards.
- Makes data display controllable and enables customization of displayed fields, aggregated data, and data sources.

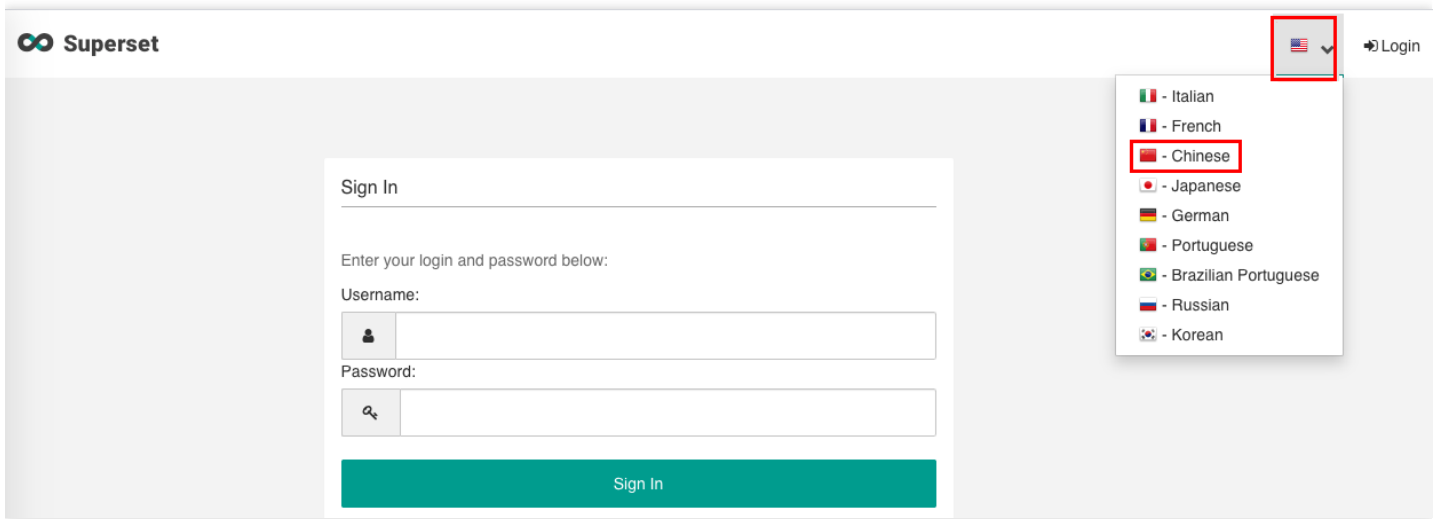
Prerequisites

1. You have created an EMR Hadoop or Druid cluster and selected the Superset service. For more information, see [Creating EMR Cluster](#).
2. By default, Superset is installed on the master node of your cluster. Enable the security group policy for the master node and make sure that your network can access port 18088 of the master node.

Login

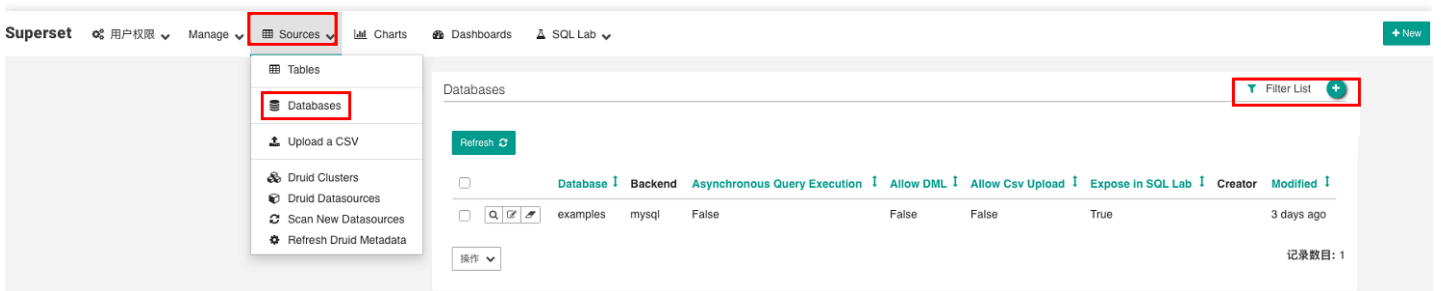
Enter `http://${master_ip}:18088` in your browser (or go to the [EMR console](#) > **Cluster Service**) to open the login page of Superset. The default username is `admin`, and the password is the one you set when creating

the cluster.



Adding Databases

Go to **Sources > Databases** and click **Filter List**.



On the following page, add the URI of the component to be added in **SQLAlchemy URI**.

The screenshot shows the 'Add Database' configuration page in Superset. The 'SQLAlchemy URI' field is highlighted with a red box and contains the value 'mysql+pymysql://root:XXXXXXXXXX@172.16.2:3306/mysql'. Below the form, there is a table with columns 'Name', 'SQLAlchemy URI', and 'Remarks'.

Name	SQLAlchemy URI	Remarks

The SQLAlchemy URI for each database is as follows:

Name	SQLAlchemy URI	Remarks

Name	SQLAlchemy URI	Remarks
MySQL	<pre>mysql+pymysql://<mysqlname>: <password>@<mysql_ip>: <mysql_port>/<your_database></pre>	<ul style="list-style-type: none"> <code>mysqlname</code> : Username used to connect to MySQL. <code>password</code> : MySQL password. <code>your_database</code> : The MySQL database to be connected to.

```
| Hive | `hive://hadoop@<master_ip>:7001/default?auth=NONE` | `master_ip`: Master IP of the EMR cluster. |
```

```
| Presto | presto://hive@<master_ip>:9000/hive/<hive_db_name> |
```

- Master_ip: `master_ip` of the EMR cluster
- hive_db_name: Name of the database in Hive. If this parameter is left empty, it will be `default` by default |

```
| Impala | impala://<core_ip>:27000 | core_ip : core IP of EMR cluster. |
```

```
| Kylin | kylin://<kylin_user>:<password>@<master_ip>:16500/<kylin_project> |
```

- kylin_user: Kylin username
- password: Kylin password
- master_ip: `master_ip` of the EMR cluster
- kylin_project: Kylin project |

```
| ClickHouse | clickhouse://<user_name>:<password>@<clickhouse-server-
endpoint>:8123/<database_name> |
```

```
clickhouse://default:password@localhost:8123/default
```

- user_name: Username
- password: Password
- clickhouse-server-endpoint: ClickHouse service endpoint
- database_name: Name of the database to be accessed |

Adding New Databases on Your Own

Superset supports databases. To install another database, follow the steps below:

- Log in to the server where the master node of EMR cluster resides.
- Run the `source /usr/local/service/superset/bin/activate` command.
- Install the corresponding Python library with pip3.
- Restart Superset.

Impala Development Guide

Impala Overview

Last updated : 2021-03-23 15:34:17

Apache Impala provides high-performance and low-latency SQL queries on data stored in Apache Hadoop file formats. Its fast response to queries enables interactive exploration and fine-tuning of analytical queries rather than long batch jobs traditionally associated with SQL-on-Hadoop technologies.

Impala differs from Hive in that Hive uses the MapReduce engine for execution and involves batch processing, while Impala streams intermediate results over the internet instead of writing them to the disk, which greatly reduces the IO overheads of nodes.

Impala integrates with Apache Hive database to share databases and tables between both components. The high level of integration with Hive and compatibility with the HiveQL syntax enable you to use either Impala or Hive to create tables, initiate queries, load data, and do more.

Prerequisites

- You have signed up for a Tencent Cloud account and created an EMR cluster. When creating the EMR cluster, select the Impala component on the software configuration page.
- Impala is installed in the `/data/Impala` directory of the CVM instance for the EMR cluster.

Data Preparations

First log in to any node (preferably a master one) in the EMR cluster. For more information on how to log in to EMR, please see [Logging in to Linux Instance Using Standard Login Method](#). On the CVM console, select the CVM that you want to log in to and click **Log In**. For **Password login**, enter **root** in **User Name** and your custom password when EMR is created in **Login password**. Then you can access the command line interface.

Run the following command in EMR command-line interface to switch to the Hadoop user and go to the Impala folder:

```
[root@10 ~]# su hadoop
[hadoop@10 root]$ cd /data/Impala/
```

Create a bash script file named `gen_data.sh` and add the following code to it:

```
#!/bin/bash
MAXROW=1000000 # Specify the number of data rows to be generated
for((i = 0; i < $MAXROW; i++))
do
echo $RANDOM, \"$RANDOM\"
done
```

Then, run the following command:

```
[hadoop@10 ~]$ ./gen_data.sh > impala_test.data
```

This script file will generate 1,000,000 random number pairs and save them to the `impala_test.data` file. Then, upload the generated test data to HDFS and run the following command:

```
[hadoop@10 ~]$ hdfs dfs -put ./impala_test.data $hdfspath
```

Here, `$hdfspath` is the path of your file on HDFS. Finally, you can run the following command to verify whether the data has been properly put on HDFS.

```
[hadoop@10 ~]$ hdfs dfs -ls $hdfspath
```

Basic Impala Operations

Connecting to Impala

Log in to a master node of the EMR cluster, switch to the Hadoop user, go to the Impala directory, and connect to Impala by running the following command:

```
[root@10 Impala]# cd /data/Impala/; bin/impala-shell.sh -i $core_ip:27001
```

Here, `core_ip` is the IP of the core node of the EMR cluster. The IP of a task node can also be used. After login succeeds, the following will be displayed:

```
Connected to $core_ip:27001
Server version: impalad version 2.10.0-SNAPSHOT (build Could not obtain git hash)
*****
**
Welcome to the Impala shell.
(Impala Shell v2.10.0-SNAPSHOT (Could) built on Tue Nov 20 17:28:10 CST 2018)
```


The '-B' command line flag turns off pretty-printing for query results. Use this flag to remove formatting from results you want to save for later, or to benchmark.

```
Impala.
*****
**
[$core_ip:27001] >
```

You can also directly connect to Impala by executing the following statement after logging in to the core node or task node:

```
cd /data/Impala/; bin/impala-shell.sh -i localhost:27001
```

Creating Impala database

Run the following statement in Impala to view the database:

```
[10.1.0.215:27001] > show databases;
Query: show databases
+-----+-----+
| name | comment |
+-----+-----+
| _impala_builtins | System database for Impala builtin functions |
| default | Default Hive database |
+-----+-----+
Fetched 2 row(s) in 0.09s
```

Run the `create` command to create a database:

```
[localhost:27001] > create database experiments;
Query: create database experiments
Fetched 0 row(s) in 0.41s
```

Run the `use` command to go to the test database you just created:

```
[localhost:27001] > use experiments;
Query: use experiments
```

View the current database and execute the following statement:

```
select current_database();
```

Creating Impala table

Run the `create` command to create an internal table named `impala_test` in the `experiments` database:

```
[localhost:27001] > create table t1 (a int, b string) ROW FORMAT DELIMITED FIELDS
TERMINATED BY ',';
Query: create table t1 (a int, b string)
Fetched 0 row(s) in 0.13s
```

View all tables:

```
[localhost:27001] > show tables;
Query: show tables
+-----+
| name |
+-----+
| t1 |
+-----+
Fetched 1 row(s) in 0.01s
```

View the table structure:

```
[localhost:27001] > desc t1;
Query: describe t1
+-----+-----+-----+
| name | type | comment |
+-----+-----+-----+
| a | int | |
| b | string | |
+-----+-----+-----+
Fetched 2 row(s) in 0.01s
```

Importing data into table

For data stored in HDFS, run the following command to import it into the table:

```
LOAD DATA INPATH '$hdfspath/impala_test.data' INTO TABLE t1;
```

Here, `$hdfspath` is the path of your file in HDFS. After the import is completed, the source data file in the import path in HDFS will be deleted and then stored in the `/usr/hive/warehouse/experiments.db/t1` path of the Impala internal table. You can also create an external table by executing the following statement:

Note :

There is only one command. If you do not enter the semicolon ";", you can put one command in multiple lines for input.

```
CREATE EXTERNAL TABLE t2
(
a INT,
b string
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
LOCATION '/impala_test_dir';
```

Running query

```
[localhost:27001] > select count(*) from experiments.t1;
Query: select count(*) from experiments.t1
Query submitted at: 2019-03-01 11:20:20 (Coordinator: http://10.1.0.215:20004)
Query progress can be monitored at: http://10.1.0.215:20004/query_plan?query_id=f
1441478dba3a1c5:fa7a8eef00000000
+-----+
| count(*) |
+-----+
| 1000000 |
+-----+
Fetched 1 row(s) in 0.63s
```

The final output is 1000000.

Deleting table

```
[localhost:27001] > drop table experiments.t1;
Query: drop table experiments.t1
```

For more information on Impala operations, please see the [official documentation](#).

Connecting to Impala Through JDBC

Impala can also be connected through Java code by following the steps similar to those described in [Connecting to Hive Through Java](#).

The only difference is `$hs2host` and `$hsport`, where `$hs2host` is the IP of any core node or task node in the EMR cluster and `$hsport` can be viewed in the `conf/impalad.flgs` configuration file under the Impala directory of the corresponding node.

```
[root@10 ~]# su hadoop
[hadoop@10 root]$ cd /data/Impala/
[hadoop@10 Impala]$ grep hs2_port conf/impalad.flgs
```

How to Map HBase Tables

Impala uses Hive metadata, and all tables in Hive can be read in Impala. For more information on how to map an HBase table in Impala, please see [Mapping HBase Table in Hive](#).

Impala OPS Manual

Last updated : 2020-05-13 21:17:10

Impala failed to start as the data volume increased

Background

When there is too much metadata (such as hundreds of databases or tens of thousands of tables) in Impala, Impala needs to broadcast such metadata to all nodes when starting, with a timeout period of 10 seconds by default. If there is a large amount of metadata and the broadcasting is easy to trigger, you can set `statestore_subscriber_timeout_seconds=100` in the `/data/Impala/conf/impalad.flgs` launch configuration file to fix this problem.

Troubleshooting

Generally, when this issue occurs, the following content will appear in the Impala log at

```
/data/emr/impala/logs :
```

```
Connection with state-store lost  
Trying to re-register with state-store
```

Impala queries are slow due to a low configuration

Although Impala is not an in-memory database, it is still necessary to allocate more physical memory to Impala when dealing with large tables or high volumes of data. You are generally recommended to use a memory of 128 GB or more and allocate 80% of it to the Impala process.

A SELECT statement failed

Possible reasons:

1. Timeout was caused by a performance, capacity, or network issue with a particular node. View the Impala log to identify the node and check whether the problem persists after changing the node network.
2. Automatic cancellation of queries was caused due to excessive memory usage by `join` queries. Check whether the `join` statement is appropriate or increase the server memory.

3. The way how a node generates native code to process a specific `WHERE` clause in a query was incorrect, such as server instructions that are not supported by the processor that can generate a specific node. If the error message in the log indicates that the cause is an invalid instruction, please consider disabling native code generation before trying a query again.
4. The input data format is incorrect, such as text data files with very long lines or delimiters that do not match the characters specified in the `FIELDS TERMINATED BY` clause of the `CREATE TABLE` statement. Check whether there is extra-long data and whether correct delimiters are used in the `CREATE TABLE` statement.

Setting a limit on the memory usage of queries

```
[localhost:27001] > set mem_limit=3000000000;
MEM_LIMIT set to 3000000000
[localhost:27001] > select 5;
Query: select 5
+----+ |5 | +----+ |5 | +----+
[localhost:27001] > set mem_limit=3g;
MEM_LIMIT set to 3g
[localhost:27001] > select 5;
Query: select 5
+----+ |5 | +----+ |5 | +----+
[localhost:27001] > set mem_limit=3gb;
MEM_LIMIT set to 3gb
[localhost:27001] > select 5;
+----+
|5 | +----+ |5 | +----+
[localhost:27001] > set mem_limit=3m;
MEM_LIMIT set to 3m
[localhost:27001] > select 5;
+----+
|5 |
+----+
|5 |
+----+
[localhost:27001] > set mem_limit=3mb; MEM_LIMIT set to 3mb [localhost:21000] > s
elect 5;
+----+ |5 | +----+
```

Analyzing Data on COS/CHDFS

Last updated : 2021-07-08 10:43:44

This document describes more ways to use Impala based on COS with data from direct data insertion and COS.

Development Preparations

1. This task requires access to COS, so you need to [create a bucket](#) in COS first.
2. Create an EMR cluster. When creating the EMR cluster, you need to select the Impala component on the software configuration page and enable access to COS on the basic configuration page.
3. Relevant software programs such as Impala are installed in the `/usr/local/service/` directory of the CVM instance for the EMR cluster.

Directions

Log in to any node (preferably a master one) in the EMR cluster first. For more information on how to log in to EMR, see [Logging in to Linux Instance Using Standard Login Method](#). You can choose to log in with WebShell. Click **Login** on the right of the desired CVM instance to enter the login page. The default username is `root`, and the password is the one you set when creating the EMR cluster. Once the correct information is entered, you can enter the command line interface.

Run the following commands on the EMR command line to switch to the Hadoop user and connect to Impala:

```
[root@172 ~]# su hadoop
[hadoop@172 ~]$ impala-shell.sh -i $host:27001
```

`$host` is the private IP of your Impala data node.

Step 1. Create a table (record)

```
[$host:27001 ] > create table record(id int, name string) row format delimited fi
elds terminated by ',' stored as textfile location 'cosn://$bucketname/';
Query: create table record(id int, name string) row format delimited fields termi
nated by ',' stored as textfile location 'cosn://$bucketname/'
Fetched 0 row(s) in 3.07s
```

Here, ``$bucketname`` is the **name** plus **path** of your COS bucket. If you use CHDFS, replace the ``location`` value with ``ofs://$mountname/``, where ``$mountname`` is your CHDFS instance mount address plus **path**.

View the **table** information and confirm that ``location`` is the COS **path**.

```

[$host:27001 ] > show create table record2;
Query: show create table record2
+-----+
| result |
+-----+
| CREATE TABLE default.record2 ( |
| id INT, |
| name STRING |
| ) |
| ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' |
| WITH SERDEPROPERTIES ('field.delim'=',', 'serialization.format=',') |
| STORED AS TEXTFILE |
| LOCATION 'cosn://$bucketname' |
| TBLPROPERTIES ('numFiles'='19', 'totalSize'='1870') |
+-----+
Fetched 1 row(s) in 5.90s

```

Step 2. Insert data into the table

```

[$host:27001] > insert into record values (1,"test");
Query: insert into record values (1,"test")
Query submitted at: 2020-08-03 11:29:16 (Coordinator: http://$host:27004)
Query progress can be monitored at: http://$host:27004/query_plan?query_id=b246d31
94efb7a8f:bc60721600000000
Modified 1 row(s) in 0.64s

```

Step 3. Use Impala to query the table

```

[$host:27001] > select * from record;
Query: select * from record
Query submitted at: 2020-08-03 11:29:31 (Coordinator: http://172.30.1.136:27004)
Query progress can be monitored at: http://$host:27004/query_plan?query_id=8148da
96f8c0d369:4b26432a00000000
+----+-----+
| id | name |
+----+-----+
| 1 | test |
+----+-----+
Fetched 1 row(s) in 0.37s

```


ClickHouse Development Guide

ClickHouse Overview

Last updated : 2022-04-06 11:18:57

Tencent Cloud Elastic MapReduce (EMR) ClickHouse provides cloud-hosted services for ClickHouse, a popular open-source column-oriented database management system (DBMS). With features such as convenient ClickHouse cluster deployment, configuration modification, and monitoring and alarming, it is a secure and stable OLAP solution for both individual and corporate users. Moreover, it has excellent query performance and is therefore ideal for online data query and analysis scenarios.

ClickHouse Features

Architecture

ClickHouse is available in single-node, multi-node, and multi-replica architectures for your choice according to your business needs.

OPS

Out-of-the-box services such as monitoring, log search, and parameter modification are provided in the console.

Data

EMR ClickHouse has complete data import and export capabilities. You can easily import data from data sources such as COS, HDFS, Kafka, and MySQL to the ClickHouse cluster or vice versa.

ClickHouse Strengths

High performance

By giving full play to multi-core concurrency (with the aid of high-efficiency SIMD instruction set and vectorized execution engine) and leveraging distributed technologies and accelerated computing, ClickHouse provides real-time analysis capabilities. As an open-source solution, ClickHouse's benchmark test shows that its processing is 100–1,000 times faster than that of traditional methods, with a 50–200 MB/s high throughput during real-time import.

Low costs

Based on the well-designed column-oriented storage and efficient data compression algorithm, ClickHouse is an optimal scheme for building large-scale data warehouses, as it provides a compression ratio of up to 1,000%, greatly

improving the data storage and computing capabilities of a single server and reducing the use costs.

Ease of use

- ClickHouse comprehensively supports SQL, so that you can easily get started with it.
- It is compatible with various data types such as JSON, map, and array for quick adaption to ever-changing businesses.
- It supports cutting-edge technologies such as approximation calculation and probabilistic data structure to process massive amounts of data.

EMR ClickHouse Table Engine Overview

Table engine purposes

The table engine (i.e., table type) determines the following:

- Data storage method and location, and where to write the data to or read the data from.
- Supported queries and how they are supported.
- Concurrent data access.
- Use of indexes (if any).
- Whether multi-thread requests can be executed.
- Data replication parameters.

Engine types

MergeTree family

`MergeTree` engines are the most universal and powerful table engines for high-load tasks. A common feature among them is quick data insertion with subsequent backend data processing. They support data replication (with `Replicated*` versions of engines), partitioning, and some features not supported by other engines.

Engines in this family:

- [MergeTree](#): it is the most powerful table engine in ClickHouse.
- [ReplacingMergeTree](#): it differs from `MergeTree` in that it removes duplicate entries with the same primary key value.
- [SummingMergeTree](#): it inherits from `MergeTree`. The difference is that when merging data parts for `SummingMergeTree` tables, ClickHouse replaces all the rows with the same primary key with one row which contains summarized values for the columns with the numeric data type. If the primary key is composed in a way that a single key value corresponds to a large number of rows, this significantly reduces storage volume and speeds up data query.

- **AggregatingMergeTree**: it inherits from `MergeTree` while changing the logic for data parts merging. ClickHouse replaces all rows with the same primary key (within one data part) with a single row that stores a combination of states of aggregate functions.
- **CollapsingMergeTree**: it inherits from `MergeTree` and adds the logic of rows collapsing to data parts merge algorithm.
- **VersionedCollapsingMergeTree**: as an upgrade from `CollapsingMergeTree`, it uses a different collapsing algorithm that allows inserting data in any order with multiple threads.
- **GraphiteMergeTree**: it is used for aggregating Graphite data. It reduces the used storage capacity and increases the efficiency of queries from Graphite.

Log family

Log engines are lightweight engines with minimum functionality. They are most effective when you need to quickly write many small tables (up to around 1 million rows) and read them later as a whole.

Engines in this family:

- **TinyLog**: it is the simplest table engine and is used to store data on disks. Each column is stored in an independent compressed file. During writing, data will be added to the end of the file.
- **StripeLog**: it belongs to the family of log engines and can be used in scenarios when you need to write many tables with a small amount of data (less than 1 million rows).
- **Log**: it differs from `TinyLog` in that a small file of "marks" resides with the column files. These marks are written on every data block and contain offsets that indicate where to start reading the file in order to skip the specified number of rows. This makes it possible to read table data in multiple threads. For concurrent data access, the read operations can be performed at the same time, while write operations block reads and one another. It does not support indexes. Similarly, if writing to a table failed, the table is broken, and reading from it returns an error. It is appropriate for temporary data, write-once tables, and testing or demonstration purposes.

Integration engines

Integration engines are for integration with other data storage and processing systems.

- **Kafka**: it works with Apache Kafka.
- **MySQL**: it allows you to perform `SELECT` queries on data stored on remote MySQL servers.
- **ODBC**: it allows ClickHouse to connect to external databases through ODBC.
- **JDBC**: it allows ClickHouse to connect to external databases through JDBC.
- **HDFS**: it allows ClickHouse to access data on HDFS.

Special engines

- **Distributed**: tables with the Distributed engine do not store any data by themselves but allow distributed queries on multiple servers. Reading is automatically parallelized. During a read, the table indexes on remote servers are

used, if there are any.

- **MaterializedView**: it is used for implementing materialized views (for more information, please see "CREATE TABLE"). For storing data, it uses a different engine that was specified when the view was created. When reading from a table, it just uses this engine.
- **Dictionary**: it displays the dictionary data as a ClickHouse table.
- **Merge**: it (not to be confused with `MergeTree`) does not store data itself but allows reading from any number of other tables at the same time.
- **File**: a data source is a file used to store data in one of the file formats supported by ClickHouse (TabSeparated, Native, etc.).
- **Null**: when a `Null` table is written to, data is ignored. When a `Null` table is read from, the response is empty. However, you can create a materialized view on a `Null` table, so the data written to the table will be forwarded to the view.
- **Set**: it is a data set always in RAM.
- **Join**: loaded `Join` table data is permanently retained in the memory.
- **URL**: it manages data on a remote HTTP/HTTPS server and is similar to the `File` engine.
- **View**: it is used for implementing views (for more information, please see the `CREATE VIEW` query). It does not store data but only stores the specified `SELECT` query. When reading from a table, it runs this query (and deletes all unnecessary columns from the query).
- **Memory**: it stores data in uncompressed form in RAM.
- **Buffer**: it buffers the data to write into the RAM, periodically flushing it to another table. During the read operation, data is read from the buffer and the other table simultaneously.

Virtual column

Virtual column is an integral table engine attribute that is defined in the engine source code.

You should not specify virtual columns in the `CREATE TABLE` query and you cannot see them in `SHOW CREATE TABLE` and `DESCRIBE TABLE` query results. Virtual columns are also read-only, so you can't insert data into virtual columns.

To select data from a virtual column, you must specify its name in the `SELECT` query. `SELECT *` does not return values from virtual columns.

If you create a table with a column that has the same name as one of the table virtual columns, the virtual column becomes inaccessible. To help avoid conflicts, virtual column names are usually prefixed with an underscore.

ClickHouse Usage

ClickHouse SQL Syntax

Last updated : 2021-03-03 19:24:17

Data Type

ClickHouse supports multiple data types such as integer, floating point, character, date, enumeration, and array.

Type list

Type	Name	Type Identifier	Value Range or Description
Integer	1-byte integer	Int8	-128-127
	2-byte integer	Int16	-32768-32767
	4-byte integer	Int32	-2147483648-2147483647
	8-byte integer	Int64	-9223372036854775808-9223372036854775807
	1-byte unsigned integer	UInt8	0-255
	2-byte unsigned integer	UInt16	0-65535
	4-byte unsigned integer	UInt32	0-4294967295
	8-byte unsigned integer	UInt64	0-18446744073709551615
Floating point	Single-precision floating point	Float32	6-7 significant digits
	Double-precision floating point	Float64	15-16 significant digits

	Custom-precision floating point	Decimal32(S)	1–9 significant digits (specified by `S`)
		Decimal64(S)	10–18 significant digits (specified by `S`)
		Decimal128(S)	19–38 significant digits (specified by `S`)
Character	Varchar	String	The string length is unlimited
	Char	FixedString(N)	The string length is fixed
	UUID	UUID	The `UUID` is generated by the built-in function `generateUUIDv4`
Time	Date	Date	It stores the year, month, and day in the format of `yyyy-MM-dd`
	Timestamp (second-level)	DateTime(timezone)	Unix timestamp that is accurate down to the second
	Timestamp (custom precision)	DateTime(precision, timezone)	You can specify the time precision
Enumeration	1-byte enumeration	Enum8	256 values (-128–127) are provided
	2-byte enumeration	Enum16	65,536 values (-32768–32767) are provided
Array	Array	Array(T)	It indicates an array consisting of data in `T` type. You are not recommended to use nested arrays

- You can use UInt8 to store boolean values and limit the value to 0 or 1.
- For more information on other data types, please see the [official documentation](#).

Use cases

Enumeration

The following sample code is used to store the gender information of users in a site:

```
CREATE TABLE user (uid Int16, name String, gender Enum('male'=1, 'female'=2)) ENGINE=Memory;

INSERT INTO user VALUES (1, 'Gary', 'male'), (2, 'Jaco', 'female');
```

```
# Query data
SELECT * FROM user;

┌uid┆name┆gender┆
├──┆──┆──┆
│ 1 │ Gary │ male │
│ 2 │ Jaco │ female │
└──┆──┆──┆

# Use the `cast` function to query the enumerated integers
SELECT uid, name, CAST(gender, 'Int8') FROM user;

┌uid┆name┆CAST(gender, 'Int8')┆
├──┆──┆──┆
│ 1 │ Gary │ 1 │
│ 2 │ Jaco │ 2 │
└──┆──┆──┆
```

Array

The following sample code is used to record the IDs of users who log in to the site every day so as to analyze active users.

```
CREATE TABLE userloginlog (logindate Date, uids Array(String)) ENGINE=TinyLog;

INSERT INTO userloginlog VALUES ('2020-01-02', ['Gary', 'Jaco']), ('2020-02-03',
['Jaco', 'Sammie']);

# Query result
SELECT * FROM userloginlog;
```

```
┌logindate┆uids┆
├──┆──┆
│ 2020-01-02 │ ['Gary', 'Jaco'] │
│ 2020-02-03 │ ['Jaco', 'Sammie'] │
└──┆──┆
```

Creating Database/Table

ClickHouse uses the `CREATE` statement to create a database or table.

```
CREATE DATABASE [IF NOT EXISTS] db_name [ON CLUSTER cluster] [ENGINE = engine
(...)]

CREATE TABLE [IF NOT EXISTS] [db.]table_name [ON CLUSTER cluster]
```

```
(  
  name1 [type1] [DEFAULT|MATERIALIZED|ALIAS expr1] [compression_codec] [TTL expr1],  
  name2 [type2] [DEFAULT|MATERIALIZED|ALIAS expr2] [compression_codec] [TTL expr2],  
  ...  
) ENGINE = engine
```

Databases and tables can be created on local disks or in a distributed manner. Distributed creation can be implemented in the following two methods:

- Run the `CREATE` statement on all servers where clickhouse-server resides.
- Run the `ON CLUSTER` clause to create a database/table with ZooKeeper.

If you use clickhouse-client to query a local table of server B on server A, the error "Table xxx doesn't exist.." will be reported. If you want that all servers in the cluster can query a table, you are recommended to use distributed tables.

For more information, please see [CREATE Queries](#).

Query

ClickHouse uses the `SELECT` statement to query data.

```
SELECT [DISTINCT] expr_list  
[FROM [db.]table | (subquery) | table_function] [FINAL]  
[SAMPLE sample_coeff]  
[GLOBAL] [ANY|ALL] [INNER|LEFT|RIGHT|FULL|CROSS] [OUTER] JOIN (subquery) |table USING columns_list  
[PREWHERE expr]  
[WHERE expr]  
[GROUP BY expr_list] [WITH TOTALS]  
[HAVING expr]  
[ORDER BY expr_list]  
[LIMIT [offset_value, ]n BY columns]  
[LIMIT [n, ]m]  
[UNION ALL ...]  
[INTO OUTFILE filename]  
[FORMAT format]
```

For more information, please see [SELECT Queries Syntax](#).

Batch Write

ClickHouse uses the `INSERT INTO` statement to write data.


```
INSERT INTO [db.].table [(c1, c2, c3)] VALUES (v11, v12, v13), (v21, v22, v23),
...

INSERT INTO [db.].table [(c1, c2, c3)] SELECT ...
```

For more information, please see [INSERT](#).

Data Deletion

ClickHouse uses the `DROP` or `TRUNCATE` statement to delete data.

Note :

`DROP` deletes metadata and data, while `TRUNCATE` deletes only data.

```
DROP DATABASE [IF EXISTS] db [ON CLUSTER cluster]
DROP [TEMPORARY] TABLE [IF EXISTS] [db.].name [ON CLUSTER cluster]

TRUNCATE TABLE [IF EXISTS] [db.].name [ON CLUSTER cluster]
```

Table Structure Modification

ClickHouse uses the `ALTER` statement to modify the table structure.

```
# Column operations on table
ALTER TABLE [db].name [ON CLUSTER cluster] ADD COLUMN [IF NOT EXISTS] name [type]
[default_expr] [codec] [AFTER name_after]
ALTER TABLE [db].name [ON CLUSTER cluster] DROP COLUMN [IF EXISTS] name
ALTER TABLE [db].name [ON CLUSTER cluster] CLEAR COLUMN [IF EXISTS] name IN PARTI
TION partition_name
ALTER TABLE [db].name [ON CLUSTER cluster] COMMENT COLUMN [IF EXISTS] name 'comme
nt'
ALTER TABLE [db].name [ON CLUSTER cluster] MODIFY COLUMN [IF EXISTS] name [type]
[default_expr] [TTL]

# Partition operations on table
ALTER TABLE table_name DETACH PARTITION partition_expr
ALTER TABLE table_name DROP PARTITION partition_expr
ALTER TABLE table_name CLEAR INDEX index_name IN PARTITION partition_expr
```

```
# Attribute operations on table
ALTER TABLE table-name MODIFY TTL ttl-expression
```

For more information, please see [ALTER](#).

Information Viewing

- `SHOW` statement

It is used to display information such as databases, processing lists, tables, and dictionaries.

```
SHOW DATABASES [INTO OUTFILE filename] [FORMAT format]
SHOW PROCESSLIST [INTO OUTFILE filename] [FORMAT format]
SHOW [TEMPORARY] TABLES [{FROM | IN} <db>] [LIKE '<pattern>' | WHERE expr] [LIMIT <N>] [INTO OUTFILE <filename>] [FORMAT <format>]
SHOW DICTIONARIES [FROM <db>] [LIKE '<pattern>'] [LIMIT <N>] [INTO OUTFILE <filename>] [FORMAT <format>]
```

For more information, please see [SHOW Queries](#).

- `DESCRIBE` statement

It is used to view table metadata.

```
DESC | DESCRIBE TABLE [db.]table [INTO OUTFILE filename] [FORMAT format]
```

Functions

There are two types of ClickHouse functions: regular functions and aggregate functions. The difference is that a regular function can generate the result for each row, while an aggregate function needs a set of data to generate the result.

Regular functions

Arithmetic functions

In this type of functions, all fields in the table engage in the arithmetic calculation.

Function Name	Purpose	Use Case
plus(a, b), a + b	Calculates the sum of two fields	plus(table.field1, table.field2)
minus(a, b), a - b	Calculates the difference between two fields	minus(table.field1, table.field2)
multiply(a, b), a * b	Calculates the product of two fields	multiply(table.field1, table.field2)

divide(a, b), a / b	Calculates the quotient of two fields	divide(table.field1, table.field2)
modulo(a, b), a % b	Calculates the remainder between two fields	modulo(table.field1, table.field2)
abs(a)	Calculates absolute value	abs(table.field1)
negate(a)	Calculates opposite	negate(table.field1)

Comparison functions

Function Name	Purpose	Use Case
=, ==	Determines whether the values are the same	table.field1 = value
!=, <>	Determines whether the values are different	table.field1 != value
>	Determines whether the former value is greater than the latter value	table.field1 > value
>=	Determines whether the former value is greater than or equal to the latter value	table.field1 >= value
<	Determines whether the former value is smaller than the latter value	table.field1 < value
<=	Determines whether the former value is smaller than or equal to the latter value	table.field1 <= value

Logical operation functions

Function Name	Purpose	Use Case
AND	Returns result if both two conditions are met	-
OR	Returns result if either condition is met	-
NOT	Returns result if none conditions are met	-

Type conversion functions

Overflow may occur when you use a type conversion function. The data types of overflowed values are the same as those in C.

Function Name	Purpose	Use Case
---------------	---------	----------

<code>toInt(8 16 32 64)</code>	Converts <code>String</code> value to <code>Int</code> value	The result of <code>toInt8('128')</code> is -127
<code>toUInt(8 16 32 64)</code>	Converts <code>String</code> value to <code>UInt</code> value	The result of <code>toUInt8('128')</code> is 128
<code>toInt(8 16 32 64)OrZero</code>	Converts <code>String</code> value to <code>Int</code> value and returns 0 if failed	The result of <code>toInt8OrZero('a')</code> is 0
<code>toUInt(8 16 32 64)OrZero</code>	Converts <code>String</code> value to <code>UInt</code> value and returns 0 if failed	The result of <code>toUInt8OrZero('a')</code> is 0
<code>toInt(8 16 32 64)OrNull</code>	Converts <code>String</code> value to <code>Int</code> value and returns <code>NULL</code> if failed	The result of <code>toInt8OrNull('a')</code> is <code>NULL</code>
<code>toUInt(8 16 32 64)OrNull</code>	Converts <code>String</code> value to <code>UInt</code> value and returns <code>NULL</code> if failed	The result of <code>toUInt8OrNull('a')</code> is <code>NULL</code>

Functions similar to those above are also provided for the floating point and date types.

For more information, please see [Type Conversion Functions](#).

Date functions

For more information, please see [Functions for Working with Dates and Times](#).

String functions

For more information, please see [Functions for Working with Strings](#).

UUID

For more information, please see [Functions for Working with UUID](#).

JSON processing functions

For more information, please see [Functions for Working with JSON](#).

Aggregate functions

Function Name	Purpose	Use Case
<code>count</code>	Counts the number of rows or non-NULL values	<code>count(expr)</code> , <code>COUNT(DISTINCT expr)</code> , <code>count()</code> , <code>count(*)</code>

<code>any(x)</code>	Returns the first encountered value. The result is indeterminate	<code>any(column)</code>
<code>anyHeavy(x)</code>	Returns a frequently occurring value using the heavy hitters algorithm. The result is generally indeterminate	<code>anyHeavy(column)</code>
<code>anyLast(x)</code>	Returns the last encountered value. The result is indeterminate	<code>anyLast(column)</code>
<code>groupBitAnd</code>	Applies bitwise <code>AND</code>	<code>groupBitAnd(expr)</code>
<code>groupBitOr</code>	Applies bitwise <code>OR</code>	<code>groupBitOr(expr)</code>
<code>groupBitXor</code>	Applies bitwise <code>XOR</code>	<code>groupBitXor(expr)</code>
<code>groupBitmap</code>	Returns cardinality	<code>groupBitmap(expr)</code>
<code>min(x)</code>	Calculates the minimum	<code>min(column)</code>
<code>max(x)</code>	Calculates the maximum	<code>max(x)</code>
<code>argMin(arg, val)</code>	Returns the <code>arg</code> value for a minimal <code>val</code> value	<code>argMin(c1, c2)</code>
<code>argMax(arg, val)</code>	Returns the <code>arg</code> value for a maximum <code>val</code> value	<code>argMax(c1, c2)</code>
<code>sum(x)</code>	Calculates the sum	<code>sum(x)</code>
<code>sumWithOverflow(x)</code>	Calculates the sum. If the sum exceeds the maximum value for this data type, the function will return an error	<code>sumWithOverflow(x)</code>
<code>sumMap(key, value)</code>	Sums the <code>value</code> array of the same <code>key</code> and returns the tuples of <code>value</code> and <code>key</code> arrays: <code>keys</code> in sorted order, and <code>value</code> sum of corresponding <code>keys</code>	-
<code>skewPop</code>	Calculates skewness	<code>skewPop(expr)</code>
<code>skewSamp</code>	Calculates sample skewness	<code>skewSamp(expr)</code>
<code>kurtPop</code>	Calculates kurtosis	<code>kurtPop(expr)</code>
<code>kurtSamp</code>	Calculates sample kurtosis	<code>kurtSamp(expr)</code>

<code>timeSeriesGroupSum(uid, timestamp, value)</code>	Sums the timestamps in time-series grouped by <code>uid</code> . It uses linear interpolation to add missing sample timestamps before summing the values	-
<code>timeSeriesGroupRateSum(uid, ts, val)</code>	Calculates the rate of time-series grouped by <code>uid</code> and then sum rates together	-
<code>avg(x)</code>	Calculates the average	-
<code>uniq</code>	Calculates the approximate number of different values	<code>uniq(x[, ...])</code>
<code>uniqCombined</code>	Calculates the approximate number of different values. Compared with <code>uniq</code> , it consumes less memory and is more accurate but has slightly lower performance	<code>uniqCombined(HLL_precision)(x[, ...]), uniqCombined(x[, ...])</code>
<code>uniqCombined64</code>	Functions in the same way as <code>uniqCombined</code> but uses 64-bit values to reduce the probability of result value overflow	-
<code>uniqHLL12</code>	Calculates the approximate number of different values. It is not recommended. Please use <code>uniq</code> and <code>uniqCombined</code> instead	-
<code>uniqExact</code>	Calculates the exact number of different values	<code>uniqExact(x[, ...])</code>
<code>groupArray(x), groupArray(max_size)(x)</code>	Returns an array of <code>x</code> values. The array size can be specified by <code>max_size</code>	-
<code>groupArrayInsertAt(value, position)</code>	Inserts value into array at specified position	-
<code>groupArrayMovingSum</code>	-	-
<code>groupArrayMovingAvg</code>	-	-
<code>groupUniqArray(x), groupUniqArray(max_size)(x)</code>	-	-
<code>quantile</code>	-	-
<code>quantileDeterministic</code>	-	-

quantileExact	-	-
quantileExactWeighted	-	-
quantileTiming	-	-
quantileTimingWeighted	-	-
quantileTDigest	-	-
quantileTDigestWeighted	-	-
median	-	-
quantiles(level1, level2, ...)(x)	-	-
varSamp(x)	-	-
varPop(x)	-	-
stddevSamp(x)	-	-
stddevPop(x)	-	-
topK(N)(x)	-	-
topKWeighted	-	-
covarSamp(x, y)	-	-
covarPop(x, y)	-	-
corr(x, y)	-	-
categoricalInformationValue	-	-
simpleLinearRegression	-	-
stochasticLinearRegression	-	-
stochasticLogisticRegression	-	-
groupBitmapAnd	-	-
groupBitmapOr	-	-
groupBitmapXor	-	-

Dictionary

A dictionary is a mapping between a key and attributes and can be used as a function for query, which is simpler and more efficient than the method of combining referencing tables with a `JOIN` clause.

There are two types of data dictionaries, namely, internal and external dictionaries.

Internal dictionary

ClickHouse supports one type of [internal dictionaries](#), i.e., geobase. For more information on the supported functions, please see [Functions for Working with Yandex.Metrica Dictionaries](#).

External dictionary

ClickHouse allows you to add [external dictionaries](#) from multiple data sources. For more information on the supported data sources, please see [Sources of External Dictionaries](#).

`p](https://clickhouse.tech/docs/en/query_language/agg_functions/reference/#skewpop) | Calculates skewness |`
`skewPop(expr) |`
`| skewSamp | Calculates sample skewness | skewSamp(expr) |`
`| kurtPop | Calculates kurtosis | kurtPop(expr) |`
`| kurtSamp | Calculates sample kurtosis | kurtSamp(expr) |`
`| timeSeriesGroupSum\(uid, timestamp, value\) | Sums the timestamps in time-series grouped by uid . It uses linear interpolation to add missing sample timestamps before summing the values | - |`
`| timeSeriesGroupRateSum\(uid, ts, val\) | Calculates the rate of time-series grouped by uid and then sum rates together | - |`
`| avg\(x\) | Calculates the average | - |`
`| uniq | Calculates the approximate number of different values | uniq(x[, ...]) |`
`| uniqCombined | Calculates the approximate number of different values. Compared with uniq , it consumes less memory and is more accurate but has slightly lower performance | uniqCombined(HLL_precision)(x[, ...]),
 uniqCombined(x[, ...]) |`
`| uniqCombined64 | Functions in the same way as uniqCombined but uses 64-bit values to reduce the probability of result value overflow | - |`
`| uniqHLL12 | Calculates the approximate number of different values. It is not recommended. Please use uniq and uniqCombined instead | - |`
`| uniqExact | Calculates the exact number of different values | uniqExact(x[, ...]) |`
`| groupArray\(x\), groupArray\(max_size\)\(x\) | Returns an array of x values. The array size can be specified by max_size | - |`
`| groupArrayInsertAt\(value, position\) | Inserts value into array at specified position | - |`
`| groupArrayMovingSum | - | - |`

[groupByMovingAvg](#) | - | - |
[groupUniqArray\(x\), groupUniqArray\(max_size\)\(x\)](#) | - | - |
[quantile](#) | - | - |
[quantileDeterministic](#) | - | - |
[quantileExact](#) | - | - |
[quantileExactWeighted](#) | - | - |
[quantileTiming](#) | - | - |
[quantileTimingWeighted](#) | - | - |
[quantileTDigest](#) | - | - |
[quantileTDigestWeighted](#) | - | - |
[median](#) | - | - |
[quantiles\(level1, level2, ...\)\(x\)](#) | - | - |
[varSamp\(x\)](#) | - | - |
[varPop\(x\)](#) | - | - |
[stddevSamp\(x\)](#) | - | - |
[stddevPop\(x\)](#) | - | - |
[topK\(N\)\(x\)](#) | - | - |
[topKWeighted](#) | - | - |
[covarSamp\(x, y\)](#) | - | - |
[covarPop\(x, y\)](#) | - | - |
[corr\(x, y\)](#) | - | - |
[categoricalInformationValue](#) | - | - |
[simpleLinearRegression](#) | - | - |
[stochasticLinearRegression](#) | - | - |
[stochasticLogisticRegression](#) | - | - |
[groupByBitmapAnd](#) | - | - |
[groupByBitmapOr](#) | - | - |
[groupByBitmapXor](#) | - | - |

Dictionary

A dictionary is a mapping between a key and attributes and can be used as a function for query, which is simpler and more efficient than the method of combining referencing tables with a `JOIN` clause.

There are two types of data dictionaries, namely, internal and external dictionaries.

Internal dictionary

ClickHouse supports one type of [internal dictionaries](#), i.e., geobase. For more information on the supported functions, please see [Functions for Working with Yandex.Metrica Dictionaries](#).

External dictionary

ClickHouse allows you to add [external dictionaries](#) from multiple data sources. For more information on the supported data sources, please see [Sources of External Dictionaries](#).

Client Overview

Last updated : 2020-05-28 16:41:32

ClickHouse provides two types of client APIs over HTTP and TCP protocols respectively.

Over HTTP

HTTP is mainly used to support simple lightweight operations and suitable in cross-platform and cross-programming language scenarios. The `clickhouse-server` process on the EMR cluster can start the HTTP service at port 8123 to send simple `GET` requests in order to check whether the service is normal.

```
$ curl http://127.0.0.1:8123
Ok.
```

You can also send requests through `query` parameters. For example, the following sample code is to query data in the `account` table in `testdb` :

```
$ wget -q -O- 'http://127.0.0.1:8123/?query=SELECT * from testdb.account'
1 GHua WuHan Hubei 1990
2 SLiu ShenZhen Guangzhou 1991
3 JPong Chengdu Sichuan 1992
```

For other methods, please see [HTTP Interface](#).

Over TCP

TCP is used mainly on `clickhouse-client` . You can enter the `clickhouse-client` command in the EMR cluster to get information such as the version information, address of the connected `clickhouse-server` , and database used by default. **You can run `quit` , `exit` , or `q` to exit.**

```
$ clickhouse-client
ClickHouse client version 19.16.12.49.
Connecting to localhost:9000 as user default.
Connected to ClickHouse server version 19.16.12 revision 54427.
```

Key parameters used by `clickhouse-client` :

- `-C --config-file`: configuration file used by client.
- `-h --host`: IP address of `clickhouse-server` .

- --port: port address of `clickhouse-server` .
- -u --user: username.
- --password: password.
- -d --database: database name.
- -V --version: displays client version.
- -E --vertical: displays query results in vertical format.
- -q --query: passed-in SQL statement in non-interactive mode.
- -t --time: displays execution time in non-interactive mode.
- --log-level: client log level.
- --send_logs_level: level of log data returned by server.
- --server_logs_file: server log storage path.

For other parameters, please see [Command-line Client](#).

Getting Started

Last updated : 2020-05-28 16:56:24

1. In the EMR Console, create a ClickHouse cluster and log in to a server in it at the public IP address.
2. Prepare data in CSV format by creating an `account.csv` file in the `/data` directory.

```
AccountId, Name, Address, Year
1, 'GHua', 'WuHan Hubei', 1990
2, 'SLiu', 'ShenZhen Guangzhou', 1991
3, 'JPong', 'Chengdu Sichuan', 1992
```

3. Use `clickhouse-client` to connect to the service and create a database and table.

```
CREATE DATABASE testdb;
CREATE TABLE testdb.account (accountid UInt16, name String, address String, year UInt64) ENGINE=MergeTree ORDER BY(accountid);
```

4. Import data into the data table.

```
cat /data/account.csv | clickhouse-client --database=testdb --query="INSERT INTO account FORMAT CSVWithNames"
```

5. Query the imported data.

```
select * from testdb.account;
```

```
SELECT *
FROM testdb.account
```

accountid	name	address	year
1	GHua	WuHan Hubei	1990
2	SLiu	ShenZhen Guangzhou	1991
3	JPong	Chengdu Sichuan	1992

```
3 rows in set. Elapsed: 0.001 sec.
```

ClickHouse Data Import

COS Data Import

Last updated : 2020-09-14 15:21:34

ClickHouse allows you to import data stored in COS to tables in two methods:

- Table engine: you can create an `Engine=S3` external table to import data.
- Table function: you can use a built-in function to import data.

Before using the two methods above, you need to see the table below to set COS access permission:

EMR	ClickHouse	COS Access Permission	Table Engine
1.0.0	19.16.12.49	Public read/write	S3
1.1.0	20.3.10.75	Public read/write, authenticated read/write (by secretId and secretKey)	S3
1.2.0+	20.7.2.30	Public read/write, authenticated read/write (by secretId and secretKey)	S3, COSN

Using Table Engine

You can create an external table and a destination table whose engine is `S3` and use the `INSERT INTO` statement to insert data in batches.

```
CREATE TABLE testdb.costb (
  column1 UInt32,
  column2 String,
  column3 String
) ENGINE=S3 ('http://${bucket-name}.cos.${region}.myqcloud.com/data1.csv', 'CSV'
);
Or
CREATE TABLE testdb.costb (
  column1 UInt32,
  column2 String,
  column3 String
) ENGINE=S3('http://${bucket-name}.cos.${region}.myqcloud.com/data1.csv', 'secret
Id', 'secretKey', 'CSV');
Or
```

```
CREATE TABLE testdb.costb (  
  column1 UInt32,  
  column2 String,  
  column3 String  
) ENGINE=COSN('http://{bucket-name}.cos.{region}.myqcloud.com/data1.csv', 'secretId', 'secretKey', 'CSV');  
  
CREATE TABLE testdb.chtb (  
  column1 UInt32,  
  column2 String,  
  column3 String  
) ENGINE=MergeTree() ORDER BY(column1);  
  
INSERT INTO testdb.chtb SELECT * FROM testdb.costb;
```

If you are using version EMR 1.2.0 or later (ClickHouse 20.7.2.30+), you can change the table engine S3 to COSN, and get the same result.

Using Table Function

When creating a table, you can use the `s3` built-in function to directly import data into it.

```
CREATE TABLE testdb.chtb  
ENGINE=MergeTree()  
ORDER BY(column1)  
AS SELECT * FROM s3(  
  'http://{bucket-name}.cos.{region}.myqcloud.com/data1.csv',  
  'CSV', 'column1 UInt32, column2 String, column3 String');  
Or  
CREATE TABLE testdb.chtb  
ENGINE=MergeTree()  
ORDER BY(column1)  
AS SELECT * FROM s3(  
  'http://{bucket-name}.cos.{region}.myqcloud.com/data1.csv', 'secretId', 'secretKey',  
  'CSV', 'column1 UInt32, column2 String, column3 String');
```

References

- [CREATE Queries](#)
- [Support S3 as the persistent storage](#)

HDFS Data Import

Last updated : 2021-07-01 11:25:51

Overview

This document describes two methods to import HDFS data to a ClickHouse cluster suitable for scenarios with low and high data volumes respectively. **In this document, v19.16.12.49 is used as an example.**

Note :

To share your thoughts on ClickHouse, please [submit a ticket](#) to join the ClickHouse technical exchange group.

Directions

Importing data from external table

This method is suitable for scenarios where the data volume is small and can be implemented in the following steps:

- Create an HDFS engine external table in ClickHouse to read the HDFS data.
- Create a regular table (generally in the `MergeTree` family) in ClickHouse to store the HDFS data.
- Select data from the external table (with the `SELECT` statement) and insert it into the regular table (with the `INSERT` statement) to import the data.

Step 1. Create an HDFS engine external table

```
CREATE TABLE source
(
  `id` UInt32,
  `name` String,
  `comment` String
)
ENGINE = HDFS('hdfs://172.30.1.146:4007/clickhouse/globs/*.csv', 'CSV')
```

For more information on how to use the HDFS engine (`ENGINE = HDFS(URI, format)`), please see [HDFS](#).

`URI` is the HDFS path. If it contains wildcards, the table is read-only. File match with wildcards is performed during the query rather than during table creation. Therefore, if the number or content of matched files changes between two queries, the difference will be shown in the query results. Supported wildcards are as follows:

- `*` can match a random number of any characters except the path separator `/`, including an empty string.
- `?` can match a character.
- `{some_string, another_string, yet_another_one}` can match `some_string`, `another_string`, or `yet_another_one`.
- `{N..M}` can match numbers from `N` to `M` (including `N` and `M`); for example, `{1..3}` can match 1, 2, and 3.

For more information on the formats supported for `format`, please see [Formats for Input and Output Data](#).

Step 2. Create a regular table

```
CREATE TABLE dest
(
  `id` UInt32,
  `name` String,
  `comment` String
)
ENGINE = MergeTree()
ORDER BY id
```

Step 3. Import data

```
INSERT INTO dest SELECT *
FROM source
```

Step 4. Query data

```
SELECT *
FROM dest
LIMIT 2
```

Concurrent JDBC driver import scheme

ClickHouse provides methods to access JDBC and an official driver. You can also use third-party drivers. For more information, please see [JDBC Driver](#).

ClickHouse is deeply integrated with big data ecosystems such as Hadoop and Spark. By developing Spark or MapReduce applications and leveraging the concurrent processing capabilities of the big data platform, you can quickly import a high volume of data from HDFS to ClickHouse. Spark also supports other data sources such as Hive; therefore, you can import data from other data sources in a similar way.

The following uses Spark Python as an example to describe how to import data concurrently:

Step 1. Create a regular table

```
CREATE TABLE default.hdfs_loader_table
(
  `id` UInt32,
  `name` String,
  `comment` String
)
ENGINE = MergeTree()
PARTITION BY id
ORDER BY id
```

Step 2. Develop a Spark Python application

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from pyspark.sql import SparkSession
import sys
if __name__ == '__main__':
if len(sys.argv) != 2:
print("Usage: clickhouse-spark <path>", file=sys.stderr)
sys.exit(-1)
spark = SparkSession.builder \
  .appName("clickhouse-spark") \
  .enableHiveSupport() \
  .getOrCreate()
url = "jdbc:clickhouse://172.30.1.15:8123/default"
driver = 'ru.yandex.clickhouse.ClickHouseDriver'
properties = {
  'driver': driver,
  "socket_timeout": "300000",
  "rewriteBatchedStatements": "true",
  "batchsize": "1000000",
  "numPartitions": "4",
  'user': 'default',
  'password': 'test'
}
spark.read.csv(sys.argv[1], schema="\"id INT, name String, comment String\"").write.jdbc(
  url=url,
  table='hdfs_loader_table',
  mode='append',
  properties=properties,
)
```

The URL format is `jdbc:clickhouse://host:port/database` where `port` is the HTTP protocol port of ClickHouse, which is 8123 by default.

The meanings of some parameters in `properties` are as follows:

- `socket_timeout` is the timeout period in milliseconds. For more information, please see [here](#).
- `rewriteBatchedStatements` is used to enable batch SQL execution in the JDBC driver.
- `batchsize` specifies the number of data entries that can be written at a time. You can increase the value appropriately to improve the write performance.
- `numPartitions` specifies the data write concurrency, which also determines the number of JDBC concurrent connections. For more information on `batchsize` and `numPartitions`, please see [JDBC to Other Databases](#).

Step 3. Submit a Spark task

```
#!/usr/bin/env bash
spark-submit \
--master yarn \
--jars ./clickhouse-jdbc-0.2.4.jar,./guava-19.0.jar \
clickhouse-spark.py hdfs:///clickhouse/globs
```

For Spark Python, you need to check the JAR version depended on by `clickhouse-jdbc-0.2.4.jar`. You can decompress the JAR file and view the configuration in `pom.xml` to check whether the JAR package for the Spark environment matches the version; and if not, the error "[Could not initialize class ru.yandex.clickhouse.ClickHouseUtil](#)" may occur. In this case, you need to download the JAR package on the correct version and submit it with the parameter `--jars` in the `spark-submit` command.

Step 4. Query data

```
SELECT *
FROM hdfs_loader_table
LIMIT 2
```

Notes

This section describes two ways to directly read/write HDFS data, which are generally used to import data from HDFS to ClickHouse. They are relatively slow in read/write and do not support the following features (for more information, please see [HDFS](#)):

- `ALTER` and `SELECT...SAMPLE` operations

- Indexes
- Replication

Table engine

1. Create a table

```
CREATE TABLE hdfs_engine_table(id UInt32, name String, comment String) ENGINE=HDFS('hdfs://172.30.1.146:4007/clickhouse/hdfs_engine_table', 'CSV')
```

2. Insert the testing data

```
INSERT INTO hdfs_engine_table VALUES(1, 'zhangsan', 'hello zhangsan'), (2, 'lisi', 'hello lisi')
```

3. Query the data

```
SELECT * FROM hdfs_engine_table
```

id	name	comment
1	zhangsan	hello zhangsan
2	lisi	hello lisi

4. View the HDFS file

```
hadoop fs -cat /clickhouse/hdfs_engine_table  
1,"zhangsan","hello zhangsan"  
2,"lisi","hello lisi"
```

Table function

There is only a slight difference in the table creation syntax between using a table function and using a table engine.

The sample code is as follows:

```
CREATE TABLE hdfs_function_table AS hdfs('hdfs://172.30.1.146:4007/clickhouse/hdfs_function_table', 'CSV', 'id UInt32, name String, comment String')
```

References

- [ClickHouse Documentation - Table Engine HDFS](#)
- [ClickHouse Documentation - Table Function hdfs](#)
- [How to Import Data from HDFS to ClickHouse?](#)

- [How to import my data from HDFS?](#)
- [ClickHouse Documentation - JDBC Driver](#)
- [Summary for Writing Data to ClickHouse from Spark JDBC](#)

Kafka Data Import

Last updated : 2020-12-21 09:50:56

Overview

This article describes how to import data from Kafka into a ClickHouse cluster.

Note :

For more technical exchanges on ClickHouse, [submit a ticket](#) to us, and we will add you into the ClickHouse technical exchange group.

Kafka is a widely used open source messaging middleware. A common use case of Kafka is to collect data from services as a data bus, including service, subscription, and spending data, and then generate reports or data applications. ClickHouse has a built-in Kafka engine, making it easy to integrate ClickHouse and Kafka.

Standard process for importing data from Kafka into ClickHouse:

- Create an external Kafka engine table in ClickHouse as a connector to access the Kafka data source.
- Create a common table in ClickHouse (usually MergeTree family) to store the data from Kafka.
- Create a materialized view in ClickHouse to listen to the data in Kafka and write the data from Kafka to a ClickHouse table.

After completing the above three steps, you can import the data from Kafka to the ClickHouse cluster.

Importing data from Kafka to ClickHouse

ClickHouse provides a Kafka engine that serves as a connector (or a data stream) to access Kafka clusters. The detailed steps are as shown below:

- **Step 1:** Create an external Kafka engine table

```
CREATE TABLE source
(
  `ts` DateTime,
  `tag` String,
  `message` String
)
ENGINE = Kafka()
```

```
SETTINGS kafka_broker_list = '172.19.0.47:9092',
kafka_topic_list = 'tag',
kafka_group_name = 'clickhouse',
kafka_format = 'JSONEachRow',
kafka_skip_broken_messages = 1,
kafka_num_consumers = 2
```

Parameter	Required	Description
kafka_broker_list	Yes	Enter Kafka brokers and separate each one with a comma
kafka_topic_list	Yes	Enter Kafka topics and separate each one with a comma
kafka_group_name	Yes	Enter the consumer group name
kafka_format	Yes	Kafka data format. For formats supported by ClickHouse, see Formats for Input and Output Data
kafka_skip_broken_messages	No	Enter an integer greater than or equal to 0, which indicates the number of errors to tolerate when parsing messages. Where N errors occur, background threads end. The materialized view will rearrange background threads to listen to data
kafka_num_consumers	No	Number of consumers of a single external Kafka engine table. By giving a higher value for this parameter, you can increase the throughput of consumed data, but the number of consumers should not be greater than the number of partitions in the topic
kafka_row_delimiter	No	Message delimiter
kafka_schema	No	If kafka_format requires a schema definition, this parameter determines the schema
kafka_max_block_size	No	This parameter determines the maximum block size allowed for writing Kafka data into the target table. The data will be flushed to the disk if the block size exceeds this value

- **Step 2:** Create a target table to store Kafka data. This table is the destination where the Kafka data will be stored. This article uses MergeTree to store Kafka data:

```
CREATE TABLE target
(
  `ts` DateTime,
  `tag` String
```

```
)  
ENGINE = MergeTree()  
PARTITION BY toYYYYMM(ts)  
ORDER BY tag
```

- **Step 3:** Create a materialized view to capture data

This article uses the following statements to create the materialized view:

```
CREATE MATERIALIZED VIEW source_mv TO target AS  
SELECT  
ts,  
tag  
FROM source
```

After completing the above three steps, you can query the data from Kafka in the target table.

In the above process, the materialized view acts as an intermediate pipeline to write the data streams represented by Kafka engine to the target table. In fact, a data stream can be attached to multiple materialized views to import the data from Kafka to multiple target tables simultaneously. You can also detach a data stream from a table or attach it to a target table.

MySQL Data Import

Last updated : 2020-05-28 16:00:19

Overview

This document describes how to import data in MySQL into a ClickHouse cluster in the following two methods:

- Through ClickHouse's support for MySQL external tables.
- By using the `clickhouse-mysql-data-reader` tool provided by Altinity.

In the examples below, data is imported from the MySQL data table `test.clickhouse_test` into the ClickHouse cluster. The table schema is as shown below:

```
MySQL [test]> desc clickhouse_test;
+-----+-----+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default          | Extra          |
+-----+-----+-----+-----+-----+-----+
| id    | int(20)       | YES  |     | NULL             |               |
| name  | varchar(64)   | YES  |     | NULL             |               |
| ts    | timestamp    | NO   |     | CURRENT_TIMESTAMP | on update CURRENT_TIMESTAMP |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

Importing Data by Using MySQL Table Engine (Simple Scheme)

ClickHouse's MySQL table engine allows you to perform `SELECT` queries on data stored on remote MySQL servers. Based on this capability, you can use the `CREATE ... SELECT * FROM` or `INSERT INTO ... SELECT * FROM` statement to import data.

Directions:

- Step 1. Create a MySQL table engine in ClickHouse

```
CREATE TABLE clickhouse_test2
(
  `id` Nullable(Int32),
  `name` Nullable(String),
  `ts` DateTime
)
ENGINE = MySQL('172.19.0.31:3306', 'test', 'clickhouse_test', 'root', 'tencentcloud')
```

- Step 2. Create a ClickHouse table

```
CREATE TABLE clickhouse_test3
(
  `id` Nullable(Int32),
  `name` Nullable(String),
  `ts` DateTime
)
ENGINE = TinyLog()
```

- Step 3. Import data in the external table created in step 1 into the ClickHouse table

```
INSERT INTO clickhouse_test3 (id, name, ts) SELECT *
FROM clickhouse_test2
```

You can combine steps 2 and 3 into one step, i.e., using `CREATE TABLE AS SELECT * FROM` to achieve the same result.

ClickHouse supports MySQL external table engines. Is it necessary to import data into ClickHouse?

Yes. A MySQL external table engine does not store data itself; instead, data is stored in MySQL. In copy queries, especially when there are `JOIN` statements, access to an external table is very slow and even impossible. This scheme has obvious flaws and does not support importing incremental data.

Importing Data by Using Altinity Tool (Recommended Scheme)

Altinity provides the [clickhouse-mysql-data-reader](#) tool for data import. This tool can export both existing and incremental data from MySQL.

As described on the official website, use of the [pypy](#) tool can significantly improve the data import performance of `clickhouse-mysql-data-reader`.

Tool preparations

- Step 1. Download [pypy3.6-7.2.0](#) and decompress it to the `pypy` directory
- Step 2. Install `clickhouse-mysql`. **If you are performing the operations in a Tencent Cloud ClickHouse cluster, after completing the installation operations below, the tool will be integrated and can be used out of the box with no configuration required**
 - Install `pip`: `run pypy/bin/pypy3 -m ensurepip`.
 - Install `mysql-replication` and `clickhouse-driver`: `run pypy/bin/pip3 install mysql-replication` and `pypy/bin/pip3 install clickhouse-driver`.

- Install and initialize `clickhouse-mysql` :run `pypy/bin/pip3 install clickhouse-mysql` and `pypy/bin/clickhouse-mysql --install` .
- Install `clickhouse-client` :run `yum install -y clickhouse-client` .
- Install `mysql-community-devel` :run `yum install -y mysql-community-devel` .
- Step 3. Get the database permissions `SUPER` and `REPLICATION CLIENT`

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

Importing Data

After completing the preparations, you can use the tool to import data from MySQL to the ClickHouse cluster in the following steps:

1. Use `clickhouse-mysql-data-reader` to generate the SQL statement for table creation.

```
pypy/bin/pypy3 pypy/bin/clickhouse-mysql \
  --src-host=172.19.0.31 \
  --src-user=root \
  --src-password=cloud \
  --create-table-sql-template \
  --with-create-database \
  --src-tables=test.clickhouse_test > create.sql
```

Modify the SQL statement and select an appropriate table engine (`TinyLog` is used in this example). Run the table creation statement `clickhouse-client -m < create.sql` .

2. Import existing data.

```
pypy/bin/pypy3 pypy/bin/clickhouse-mysql \
  --src-host=172.19.0.31 \
  --src-user=root \
  --src-password=cloud \
  --with-create-database \
  --src-tables=test.clickhouse_test \
  --migrate-table \
  --dst-host=localhost
```

3. Import incremental data.

```
pypy/bin/pypy3 pypy/bin/clickhouse-mysql \  
--src-host=172.19.0.31 \  
--src-user=root \  
--src-password=cloud \  
--src-tables=test.clickhouse_test \  
--dst-host=127.0.0.1 \  
--mempool-max-flush-interval=60 \  
--mempool-max-events-num=10000 \  
--pump-data \  
--src-server-id=1 \  
--src-resume \  
--src-wait \  
--nice-pause=1
```

The meanings of the parameters are as follows:

- **src-host**: MySQL database IP.
- **src-user**: MySQL database username.
- **src-password**: MySQL database password.
- **create-table-sql-template**: generates the table creation script of ClickHouse.
- **with-create-database**: adds the database creation statement to the table creation script.
- **src-tables**: source table (MySQL table).
- **mempool-max-flush-interval**: time interval for `mempool flush`.
- **src-server-id**: whether the MySQL server is a master node.
- **src-resume**: resumable transfer.
- **src-wait**: data wait.
- **nice-pause**: sleep time interval if there is no data.

ClickHouse OPS Monitoring

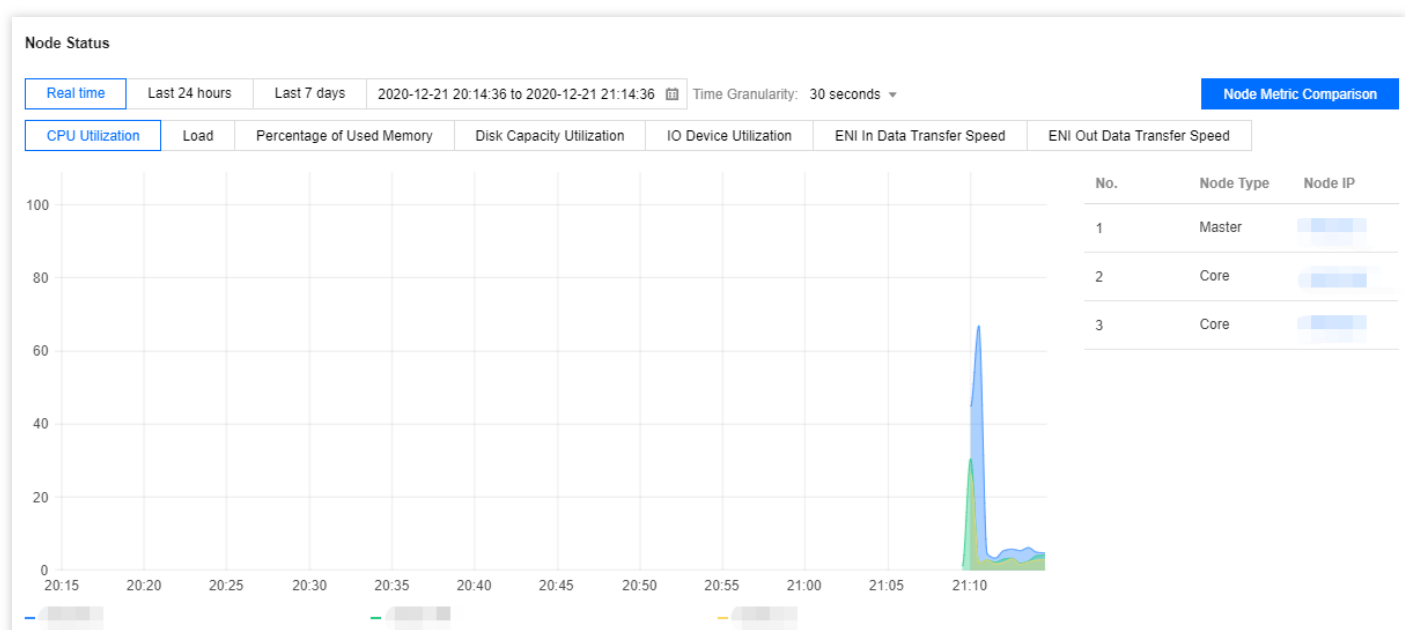
Last updated : 2021-10-08 15:36:37

Tencent Cloud Elastic MapReduce (EMR) provides a complete monitoring system for ClickHouse clusters. The system is divided into three dimensions: **Cluster Overview**, **Service Monitoring**, and **Node Monitoring**.

Cluster Overview

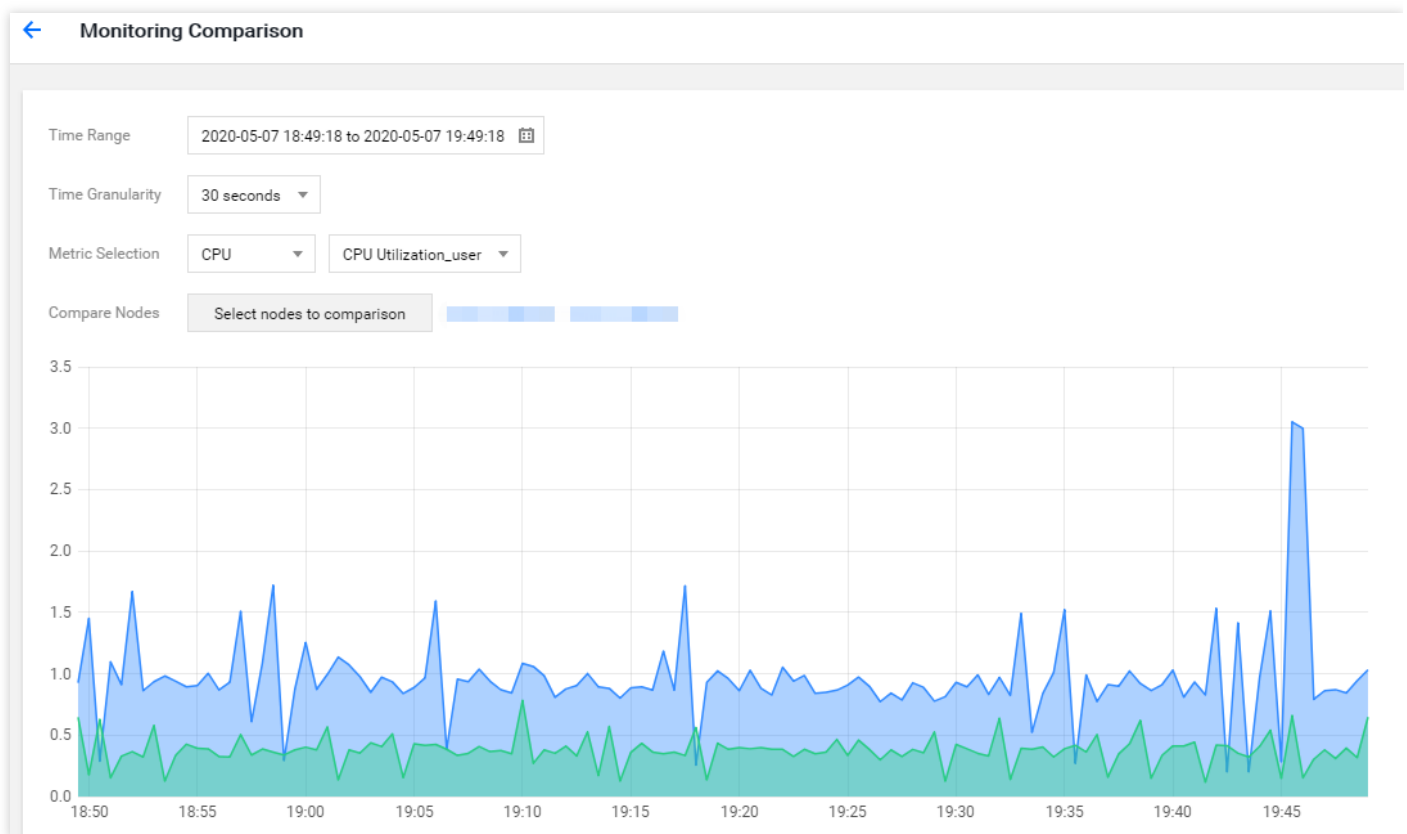
The cluster overview page displays the overview information of a ClickHouse cluster, such as the running status, number of nodes, and Zookeeper status. This page also shows service metrics and node metrics in the cluster dimension, allowing you to see the overall running status of the cluster visually.

- The service monitoring section has four aggregate metrics: the number of queries, number of active data blocks, size of the operation queue, and number of network connections. The graphs on the cluster overview page show the aggregate of the corresponding metrics for all nodes in the ClickHouse cluster.
- The node monitoring section also has four aggregate metrics: CPU utilization, memory utilization, disk utilization, and network traffic. The graphs on the cluster overview page show the overall usage of the node resources in the ClickHouse cluster.



- The deployment status section provides real-time monitoring on the cluster process status. If a process is missing, it will be immediately displayed on this page.
- In the node status section, you can view the resource usage of the 10 most used machines for the last seven days to quickly locate the cluster bottleneck.

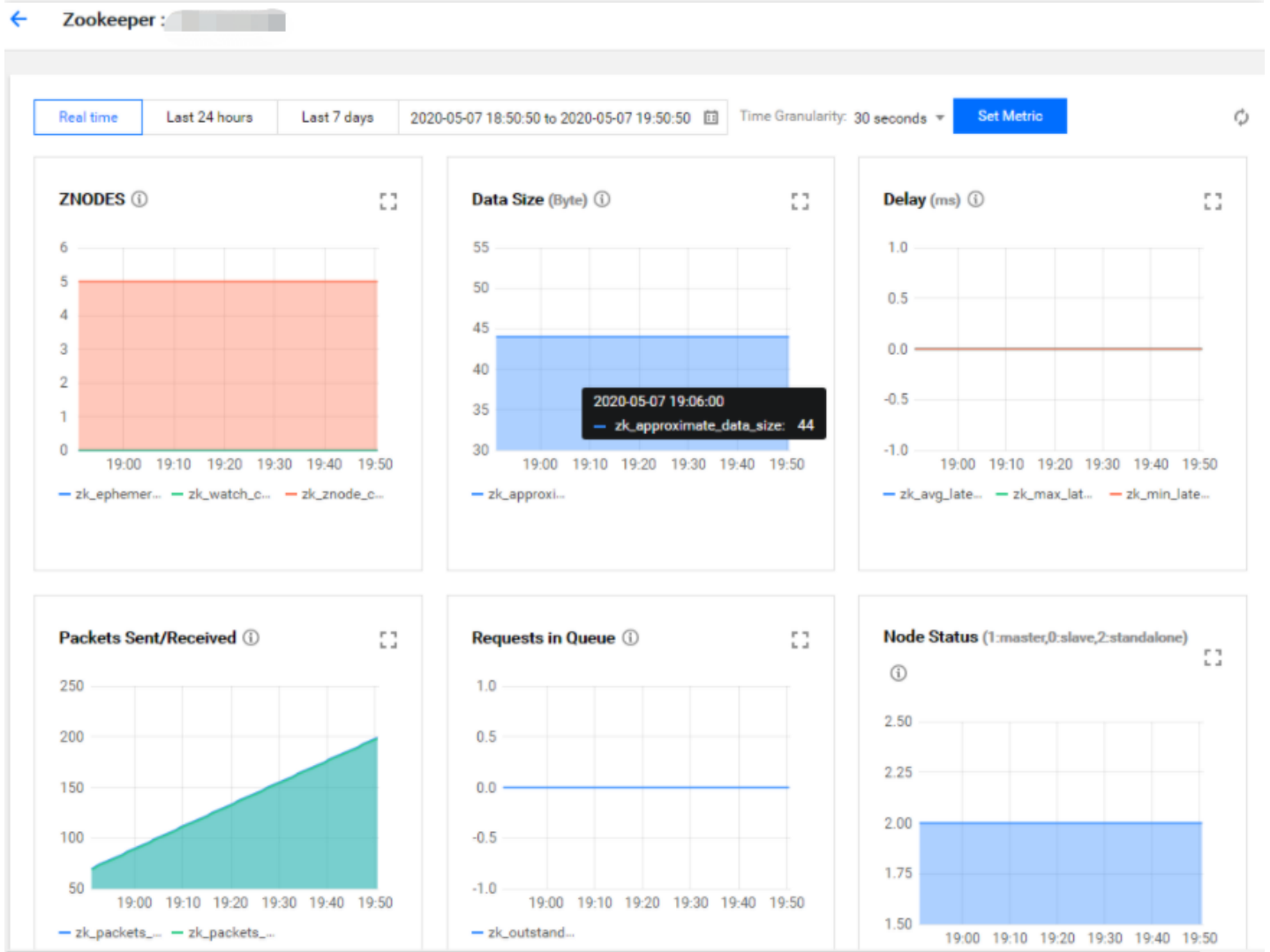
You can also click **Node Metric Comparison** to compare the resource usage of multiple nodes during a certain period of time.



Service Monitoring

The cluster service monitoring of ClickHouse is relatively simple, only consisting of ClickHouse and Zookeeper (if it is an HA cluster). You can see the basic information of the roles in the role list on the service monitoring page. Both Zookeeper and ClickHouse have only one role named Zookeeper and ClickHouse-Server, respectively. You can go to the monitoring page of a specific node from the node IP column.

On the monitoring page of a specific node, you can view the detailed data for up to 30 days, and you can select the time period as needed. Moreover, you can customize the metrics to be displayed by clicking **Set Metrics** and selecting those that you want. Currently, up to 12 metrics can be displayed at a time.



The monitoring metrics of ClickHouse are divided into three groups, which come from the three system tables of ClickHouse: metrics, events, and asynchronous_metrics.

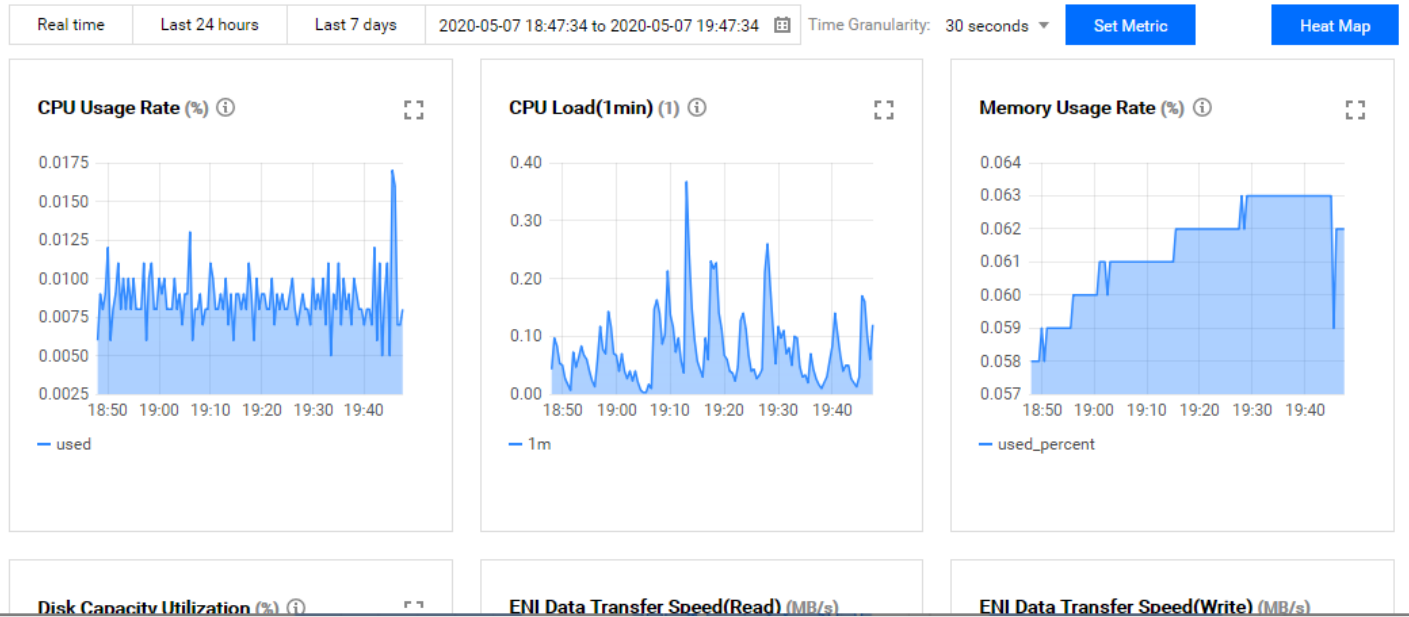
Node Monitoring

Node monitoring consists of the monitoring overview page and the monitoring details page.

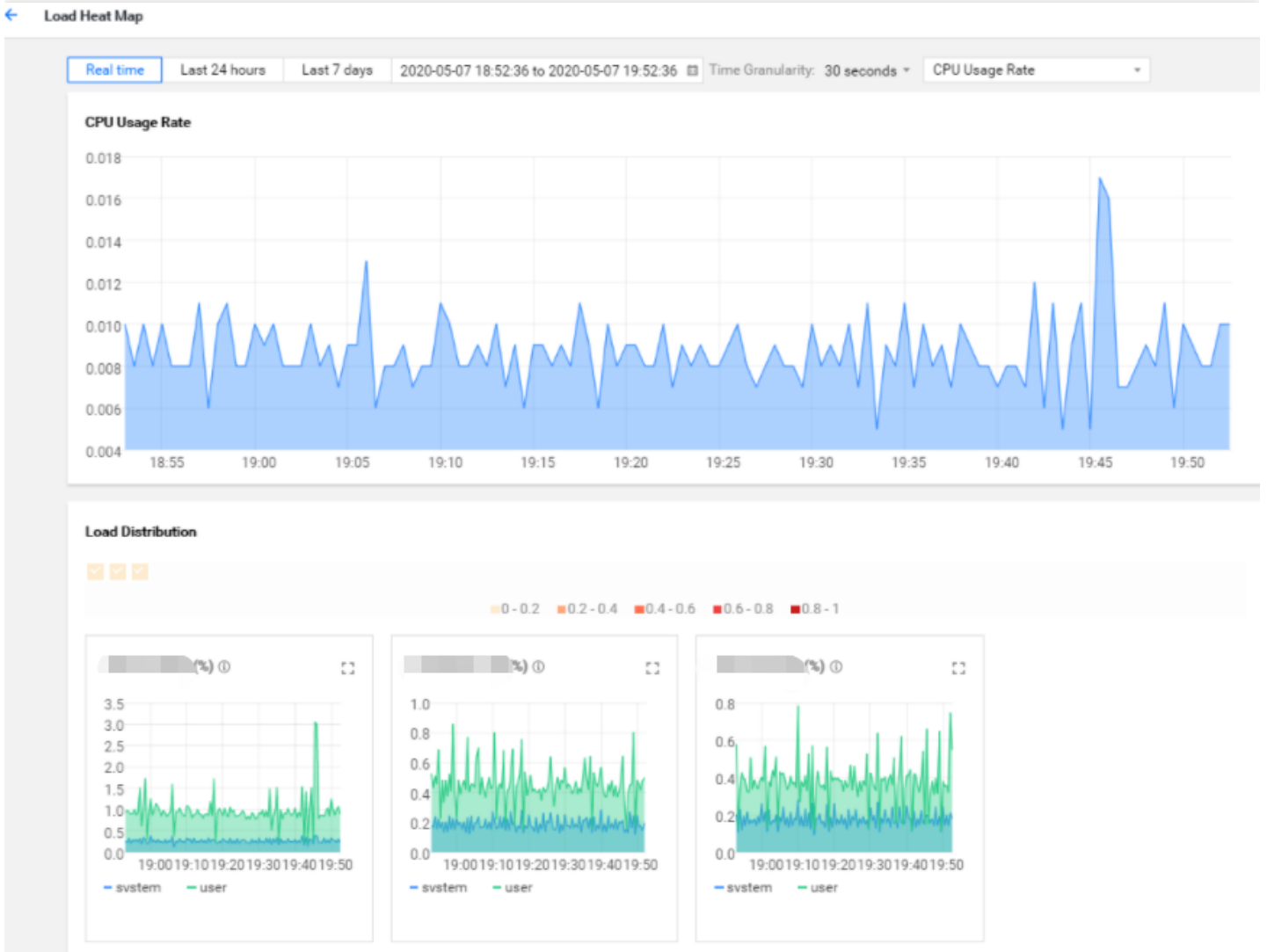
Monitoring overview

The monitoring overview page displays the aggregate monitoring metrics of the nodes in the cluster. Currently, it provides 12 aggregate metrics in dimensions such as the CPU, memory, disk, and network, reflecting the overall resource utilization of the nodes in the cluster. Like service monitoring, you can select the metrics to be displayed by clicking **Set Metrics**.

Overview



Node monitoring also provides heat maps that allow you to view the load of each machine in terms of a certain node metric during a certain period of time. For example, the curve on a heap map shows the aggregate memory utilization in the cluster dimension. In load distribution, each small square represents a node, and different colors indicate that the memory utilization of different nodes is in different categories. The darker the color, the higher the memory utilization. Load distribution is displayed in descending order by default and the memory utilization of the top 3 nodes is displayed by default, making it convenient to compare the differences between machines.



The node monitoring overview page also has a list of all nodes in the cluster. You can filter by node type, search by IP, or sort and display by CPU utilization, memory utilization, or disk utilization. Click a node IP to go to the node monitoring details page of the specific machine.

Node List

Node IP	Node Type ▾	CPU Utilization ⚡	Memory Utilization ⚡	Disk Utilization ⚡
	Master	1.3%	10.1%	0.3%
	Core	0.7%	4.7%	0.3%
	Core	0.4%	4%	0.3%

Total 3 items Lines per page 10 ▾ 1 /1 page

Monitoring details

The monitoring details page consists of four sections: basic configuration, deployment status, load status, and node monitoring.

- Basic configuration displays the basic hardware information and a disk list showing the disk information of the node. If the disk name or mount point is changed, it will be detected within 30 minutes. You can view the monitoring metrics of a disk by clicking the disk name.

Basic Configuration

IP :

Node Type: **Master** Instance ID: emr-vm-0vxndmf Resource ID: __BLANK__ ins-e1doytfo Billing Mode: Pay-as-you-go

Node Specification: **EMR Big DataD2** CPU : 8Core MEM: 32GB Local disk: 11176GB * 1

Disk	Mount Point	Usage	Disk Read-Write Speed
vdb	/data	5% 600.09GB/11998.04GB	Read: 0MB/s Write: 0.14MB/s
vda1	/	66% 35.03GB/52.71GB	Read: 0MB/s Write: 0.02MB/s

- Deployment status shows the real-time status of the deployment service processes of the node, making it convenient to monitor the machine processes.
- Load status shows the snapshot information of the machine at a certain time, including the CPU utilization, memory utilization, IO, and network condition of the processes. You can also see the process list of the machine at a certain time.

Load Status

2020-05-07 19:50:50 📄

Top CPU Processes
Top Memory Processes
Top IO Processes
Top Network Processes
Process List

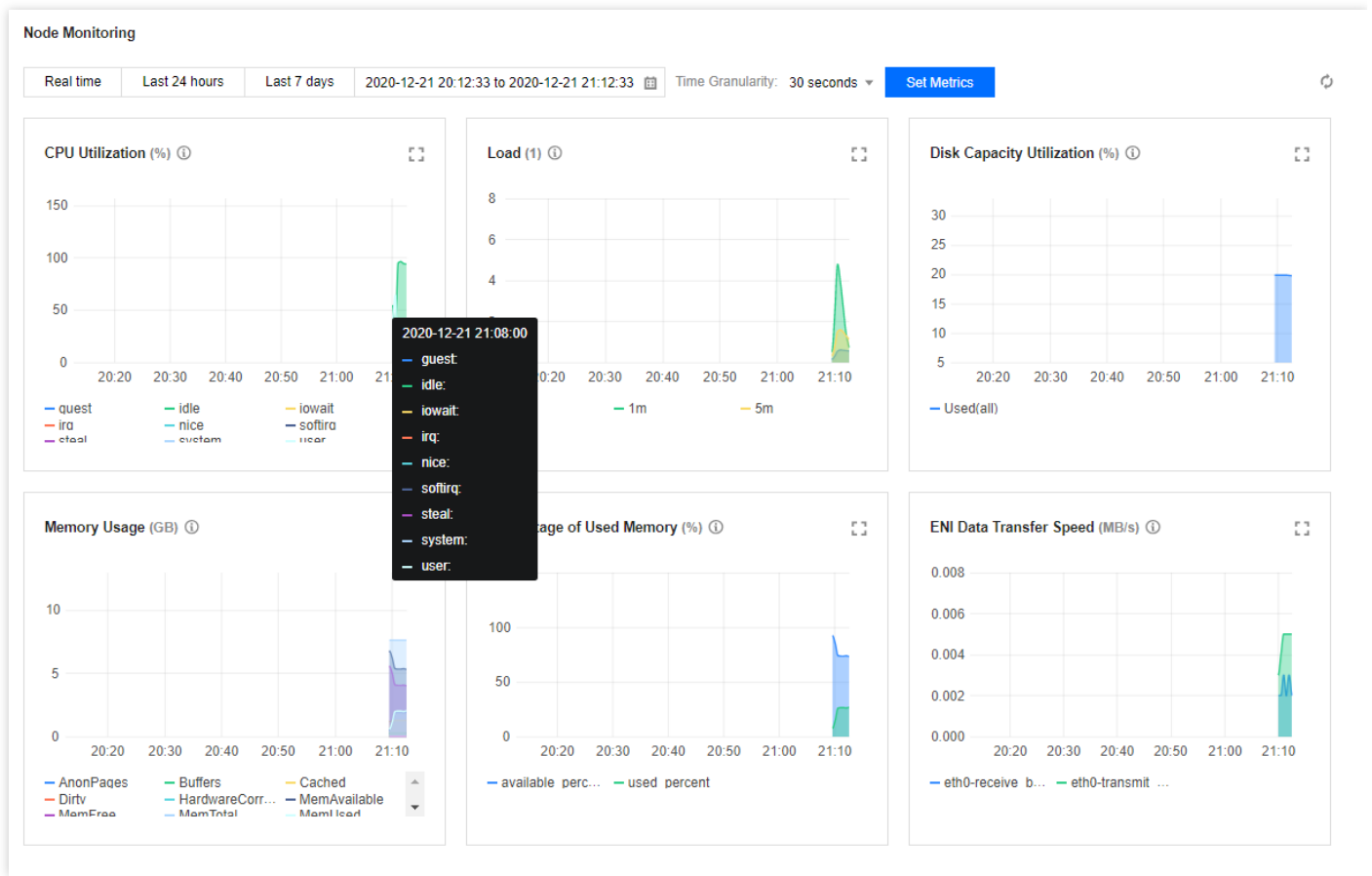
```

top - 19:50:23 up 1:37, 0 users, load average: 0.06, 0.06, 0.05
Tasks: 163 total, 1 running, 84 sleeping, 0 stopped, 0 zombie
%Cpu(s): 1.1 us, 0.3 sy, 0.0 ni, 98.5 id, 0.1 wa, 0.0 hi, 0.0 si, 0.0 st
KiB Mem : 32677364 total, 28448288 free, 3236584 used, 992492 buff/cache
KiB Swap: 0 total, 0 free, 0 used, 29007272 avail Mem

  PID USER      PR  NI   VIRT   RES   SHR  S  %CPU  %MEM    TIME+  COMMAND
    1 root        20   0 192224  5500  3748  S   0.0   0.0   0:01.72  systemd
    2 root        20   0     0     0     0  S   0.0   0.0   0:00.00  kthreadd
    3 root         0 -20     0     0     0  I   0.0   0.0   0:00.00  rcu_gp
    4 root         0 -20     0     0     0  I   0.0   0.0   0:00.00  rcu_par_gp
    6 root         0 -20     0     0     0  I   0.0   0.0   0:00.00  kworker/0+
    8 root         0 -20     0     0     0  I   0.0   0.0   0:00.00  mm_percpu+
    9 root        20   0     0     0     0  S   0.0   0.0   0:00.01  ksoftirqd+
   10 root        20   0     0     0     0  I   0.0   0.0   0:00.82  rcu_sched
   11 root        rt   0     0     0     0  S   0.0   0.0   0:00.04  migration+
   13 root        20   0     0     0     0  S   0.0   0.0   0:00.00  cpuhp/0
   14 root        20   0     0     0     0  S   0.0   0.0   0:00.00  cpuhp/1
   15 root         0   0     0     0     0  S   0.0   0.0   0:00.00  cpuhp/1

```

- Node monitoring shows the specific monitoring metrics of the node, including the CPU, memory, file handle, disk, network, process, and more. Like service monitoring, you can select the metrics to be displayed.



Summary

Cluster Overview, **Service Monitoring**, and **Node Monitoring** together form a complete monitoring system for ClickHouse clusters, which is very helpful for the OPS of ClickHouse clusters.

Configuration Description

Last updated : 2021-03-03 19:32:50

The ClickHouse server configuration files are located in `/etc/clickhouse-server` and include `config.xml`, `metrika.xml`, and `users.xml`, among which `config.xml` is the primary configuration file of the ClickHouse server.

config.xml

Under the `conf.d` and `config.d` folders at the same level as the `config.xml` configuration file, you can create an `*.xml` file to override the configurations in the `config.xml` file (**however, you are not recommended to do so; instead, you are recommended to centrally distribute configurations by using the configuration delivery feature in the console**).

For example, create a `config.d` directory at the same level as the `config.xml` file:

1. Modify the TCP port listened on by the `clickhouse-server`, which is port 9000 in the `config.xml` file by default. Create a `tcp_port.xml` file in the `config.d` folder and add the following content to it:

```
<yandex>
<tcp_port>9900</tcp_port>
</yandex>
```

Restart the `clickhouse-server` and you can find that the TCP port listened on has been changed to 9000.

2. Add the `metric_log` configuration. Create a `metric_log.xml` file and add the following content to it:

```
<yandex>
<metric_log>
<database>system</database>
<table>metric_log</table>
<flush_interval_milliseconds>7500</flush_interval_milliseconds>
<collect_interval_milliseconds>1000</collect_interval_milliseconds>
</metric_log>
</yandex>
```

Restart the `clickhouse-server` and `clickhouse-client` and run `SELECT * FROM system.metric_log LIMIT 1 FORMAT Vertical;` to create the `system.metric_log` table automatically.

metrika.xml

ClickHouse configurations can be "replaced". You can use the **incl** attribute to replace configurations in `config.xml` with those in the specified file, so that the primary configuration file will not become too redundant or hard to maintain. In the default `config.xml` configuration file, you can see that the three tags `<remote_servers>`, `<macros>`, and `<zookeeper>` all have the **incl** attribute. Therefore, you can load the corresponding `<clickhouse_remote_servers>`, `<macros>`, and `<zookeeper-servers>` configuration items in the `metrika.xml` file into the `config.xml` file. The path of the file for replacement is `/etc/metrika.xml` by default and can be modified through the `<include_from>` configuration item.

Note :

If the configuration items to be replaced in the `incl` attribute do not exist, the event will be recorded in a log. If you do not want to log such events, you can use the `optional="true"` attribute.

users.xml

In the `config.xml` configuration file, `users_config` indicates the path of the user-related configuration file, which is `users.xml` by default. In `users.xml`, you can configure information such as user password, permission, profile, and quota. Different users can have different configurations.

You can create a `users.d` directory in the `user.xml` statistics directory and add user-related configurations (e.g., `/etc/clickhouse-server/users.d/testUser.xml`) under it so as to reduce redundancy of the `users.xml` file and simplify the maintenance.

```
<yandex>
<users>
<testUser>
<profile>default</profile>
<networks>
<ip>:::0</ip>
</networks>
<password>12345</password>
<quota>default</quota>
</testUser>
</users>
</yandex>
```

For the specific server parameter configurations and settings, please see [Server Settings](#) and [Settings](#) on the official website.

Log Description

Last updated : 2020-05-28 16:56:24

ClickHouse Server Log Description

ClickHouse server log configuration items are in the `config.xml` file in the `/etc/clickhouse-server` directory by default.

```
<logger>
<level>trace</level>
<log>/data/clickhouse/clickhouse-server/logs/clickhouse-server.log</log>
<errorlog>/data/clickhouse/clickhouse-server/logs/clickhouse-server.err.log</errorlog>
<size>100M</size>
<count>10</count>
</logger>
```

- `level` records the server log level, which can be `trace`, `debug`, `information`, `warning`, or `error`.
- `log` records the file path.
- `errlog` records the error log file path.
- `size` and `count` record the maximum size of historical log file and maximum number of historical log entries that can be retained respectively.

ClickHouse Client Log Description

When using the `clickhouse-client` command line to run SQL statements, you can set the `send_logs_level` parameter in interactive mode to view the logs of each execution.

```
172.30.1.15 :) set send_logs_level='trace';
SET send_logs_level = 'trace'
Ok.

0 rows in set. Elapsed: 0.001 sec.
```

When starting `clickhouse-client`, you can specify the `send_logs_level` and `log-level` parameters.

```
clickhouse-client --send_logs_level=trace --log-level=trace
```

You can use the command with `--server_logs_file` to save the logs to the specified file.

```
clickhouse-client --send_logs_level=trace --log-level=trace --server_logs_file='/  
data/query.log'
```

Data Backup

Last updated : 2020-05-28 16:56:25

The ClickHouse data directory can be configured in the `path` field in `config.xml` by default, and the default directory is `/data/clickhouse/clickhouse-server/data` .

Data Directory Format

The structure of a `clickhouse-server` data directory is as follows:

```
test
|-- account
| |-- all_1_1_0
| | |-- checksums.txt
| | |-- columns.txt
| | |-- count.txt
| | |-- name.bin
| | |-- name.mrk2
| | |-- id.bin
| | |-- id.mrk2
| | |-- age.bin
| | |-- age.mrk2
| | |-- primary.idx
| |-- detached
|-- |-- format_version.txt
```

- The first-level directory displays the database name such as `test` .
- The second-level directory is the table name. In the example above, the `account` table exists in the `test` database.
- The third-level directory displays the partition directories.
 - `checksum.txt` stores the data checksum information.
 - `columns.txt` stores the data column names and data types.
 - `count.txt` stores the total number of data entries in the partition.
 - `.bin` and `.mrk2` store the data.

Data Backup Method

ClickHouse officially provides the `clickhouse-backup` backup tool that can back up the data directories of the `clickhouse-server` process to COS in the public cloud.

Step 1. Prepare the `config.yml` configuration file

Add the COS information such as URL, `secret_id`, `secret_key`, `clickhouse-server` address, and access information to the file. Create a `/etc/clickhouse-backup` directory and move the configuration file into it.

```
general:
  remote_storage: cos
  disable_progress_bar: false
  backups_to_keep_local: 0
  backups_to_keep_remote: 0
  clickhouse:
    username: default
    password: ""
    host: 127.0.0.1
    port: 9000
    data_path: ""
    skip_tables:
      - system.*
    timeout: 5m
  cos:
    url: "https://${bucket-name}.cos.${region}.myqcloud.com"
    timeout: 100
    secret_id: "xxxxxx"
    secret_key: "yyyyy"
    path: "/"
    compression_format: gzip
    compression_level: 1
  debug: false
```

Step 2. Use the `clickhouse-backup` tool to create a backup directory

```
$ ./clickhouse-backup create testbak1
# Run `list` to check whether the creation succeeds
$ ./clickhouse-backup list local
- 'testbak1' (created at 26-03-2020 02:55:23)
```

Step 3. Use the `clickhouse-backup` tool to package data and upload it to COS

After running the following command, check whether the compressed file `testbak1.tar.gz` exists in the corresponding directory in the COS bucket:

```
$ ./clickhouse-backup upload testbak1 2020/03/25 12:58:25 Upload backup 'testbak
1' 1.27 MiB / 1.27 MiB [=====] 100.00% 0s 2020/03/25 12:58:
26 Done.
```

References

[clickhouse-backup tool](#)

System Table Description

Last updated : 2022-04-25 12:29:36

This document is composed based on System Tables at ClickHouse official website.

- System tables are used to implement certain system features and provide access to information on how the system is working.
- You cannot delete a system table (but you can run `DETACH`).
- System tables do not have files with data or metadata on the disk. The server creates all the system tables when it starts.
- System tables are read-only.
- System tables are located in the `system` database.

system.asynchronous_metrics

This system table contains metrics that are calculated periodically in the background, such as the RAM capacity in use.

Column information:

Column Name	Type	Description
metric	String	Metric name
value	Float64	Metric value

Sample code:

```
SELECT * FROM system.asynchronous_metrics LIMIT 10
```

metric	value
jemalloc.background_thread.run_interval	0
jemalloc.background_thread.num_runs	0
jemalloc.background_thread.num_threads	0
jemalloc.retained	299642880
jemalloc.mapped	964415488
jemalloc.resident	942026752
jemalloc.metadata_thp	0
jemalloc.metadata	5601816
jemalloc.allocated	936427032
ReplicasMaxQueueSize	0

Column Name	Type	Description
cluster	String	Cluster name
shard_num	UInt32	Shard number in cluster
shard_weight	UInt32	Shard weight during data writing
replica_num	UInt32	Replica number in shard
host_name	String	Host name
host_address	String	Host IP address
port	UInt16	Port for connecting to server
user	String	Username for connecting to server
errors_count	UInt32	Number of host's failures in reaching replica
estimated_recovery_time	UInt32	Seconds left until replica error count is zeroed and status is back to normal

system.clusters

This system table contains information on clusters available in the configuration file and the servers in them.

Column information:

`errors_count` is updated once per query in the cluster, while `estimated_recovery_time` is recalculated on demand. Therefore, there could be a case of non-zero `errors_count` and zero

`estimated_recovery_time` . In this case, the next query will zero `errors_count` and try to use the replica as if it has no errors.

system.columns

This system table contains information on columns in all the tables. You can use this table to get information similar to the `DESCRIBE TABLE` query, but for multiple tables at once.

Column information:

Column Name	Type	Description
database	String	Database name
table	String	Table name
name	String	Column name
type	String	Column type
default_kind	String	Expression type (<code>DEFAULT</code> , <code>MATERIALIZED</code> , or <code>ALIAS</code>) for the default value, or an empty string if it is not defined
default_expression	String	Expression for the default value, or an empty string if it is not defined
data_compressed_bytes	UInt64	Size of compressed data in bytes
data_uncompressed_bytes	UInt64	Size of decompressed data in bytes
marks_bytes	UInt64	Size of marks in bytes
comment	String	Column comment, or an empty string if it is not defined
is_in_partition_key	UInt8	Flag that indicates whether the column is in the partition expression
is_in_sorting_key	UInt8	Flag that indicates whether the column is in the sorting key expression
is_in_primary_key	UInt8	Flag that indicates whether the column is in the primary key expression
is_in_sampling_key	UInt8	Flag that indicates whether the column is in the sampling key expression

system.contributors

This system table contains information on contributors. All contributors are in random order during query execution.

system.databases

This system table contains a single `String` column called `name`, which is the name of a database. Each database that the server knows about has a corresponding entry in it. It is used to implement the `SHOW DATABASES` query.

system.detached_parts

This system table contains information on detached parts of `MergeTree` tables. The `reason` column specifies why the part was detached. For user-detached parts, the `reason` is empty. Such parts can be attached with the `ALTER TABLE ATTACH PARTITION|PART` command. For the description of other columns, please see [system.parts](#). If the part name is invalid, values of some columns may be `NULL`. Such parts can be deleted with the `ALTER TABLE DROP DETACHED PART` command.

system.dictionaries

This system table contains information on external dictionaries.

Column information:

Column Name	Type	Description
name	String	Dictionary name
type	String	Dictionary type, which can be <code>Flat</code> , <code>Hashed</code> , or <code>Cache</code>
origin	String	Path to the configuration file that describes the dictionary
attribute.names	Array(String)	Array of attribute names provided by dictionary
attribute.types	Array(String)	Corresponding array of attribute types provided by dictionary
has_hierarchy	UInt8	Whether the dictionary is hierarchical
bytes_allocated	UInt64	RAM capacity allocated for dictionary
hit_rate	Float64	Percentage of the usage of the value in the cache for cache dictionaries
element_count	UInt64	Number of items stored in dictionary
load_factor	Float64	Percentage filled in dictionary
creation_time	DateTime	Time of creation or last successful reload of dictionary
last_exception	String	Text of the error that occurs when the dictionary is created or reloaded if the dictionary cannot be created
source	String	Text describing dictionary data source

system.events

This system table contains information on the number of events that have occurred in the system. For example, in this table, you can find how many `SELECT` queries have been processed since the ClickHouse server was started.

Column information:

Column Name	Type	Description
event	String	Event name
value	UInt64	Number of occurred events
description	String	Event description

Sample code:

```
SELECT * FROM system.events LIMIT 5
```

event	value	description
SelectQuery	31663	Same as Query, but only for SELECT queries.
ReadBufferFromFileDescriptorRead	2	Number of reads (read/pread) from a file descriptor. Does not include sockets.
WriteBufferFromFileDescriptorWrite	1	Number of writes (write/pwrite) to a file descriptor. Does not include sockets.
ReadCompressedBytes	360	
CompressedReadBufferBlocks	10	

system.functions

This system table contains information on regular and aggregate functions.

Column information:

Column Name	Type	Description
name	String	Function name
is_aggregate	UInt8	Whether the function is aggregate

system.graphite_retentions

This system table contains information on parameters `graphite_rollup` which are used in tables with `* GraphiteMergeTree` engines.

Column information:

Column Name	Type	Description
-------------	------	-------------

Column Name	Type	Description
config_name	String	<code>graphite_rollup</code> variable name
regexp	String	Metric name pattern
function	String	Aggregate function name
age	UInt64	Minimum data usage duration in seconds
precision	UInt64	Data usage duration precision in seconds
priority	UInt16	Pattern priority
is_default	UInt8	Whether the pattern is the default one
Tables.database	Array(String)	Array of names of the database tables that use the <code>config_name</code> parameter
Tables.table	Array(String)	Array of table names that use the <code>config_name</code> parameter

system.merges

This system table contains information on merges and part mutations currently in process for tables in the `MergeTree` family.

Column information:

Column Name	Type	Description
database	String	Name of the database where the table resides
table	String	Table name
elapsed	Float64	Time elapsed (in seconds) since the merge started
progress	Float64	Percentage of work completion (from 0 to 1)
num_parts	UInt64	Number of the parts to be merged
result_part_name	String	Name of the part that will be formed as the result of merging
is_mutation	UInt8	1 if this process is a part mutation
total_size_bytes_compressed	UInt64	Total size of the compressed data in the merged blocks
total_size_marks	UInt64	Total number of marks in merged parts

Column Name	Type	Description
bytes_read_uncompressed	UInt64	Number of uncompressed read bytes
rows_read	UInt64	Number of read rows
bytes_written_uncompressed	UInt64	Number of uncompressed written bytes
rows_written	UInt64	Number of written rows

system.metrics

This system table contains metrics which can be calculated instantly or have a current value, such as the number of simultaneously processed queries or the current replica delay. This table is always up to date.

Column information:

Column Name	Type	Description
metric	String	Metric name
value	Int64	Metric value
description	String	Metric description

You can find the list of supported metrics in the [dbms/src/Common/CurrentMetrics.cpp](#) source code file of ClickHouse.

Sample code:

```
SELECT * FROM system.metrics LIMIT 10
```

metric	value	description
Query	1	Number of executing queries
Merge	0	Number of executing background merges
PartMutation	0	Number of mutations (ALTER DELETE/UPDATE)
ReplicatedFetch	0	Number of data parts being fetched from replica
ReplicatedSend	0	Number of data parts being sent to replicas

```

Row 1:
-----
event_date:                2020-03-24
event_time:                2020-03-24 11:06:10
milliseconds:             437
ProfileEvent_Query:       0
ProfileEvent_SelectQuery: 0
ProfileEvent_InsertQuery: 0
ProfileEvent_FileOpen:    0
ProfileEvent_Seek:        0
ProfileEvent_ReadBufferFromFileDescriptorRead: 1
ProfileEvent_ReadBufferFromFileDescriptorReadFailed: 0
ProfileEvent_ReadBufferFromFileDescriptorReadBytes: 0
ProfileEvent_WriteBufferFromFileDescriptorWrite: 1
ProfileEvent_WriteBufferFromFileDescriptorWriteFailed: 0
ProfileEvent_WriteBufferFromFileDescriptorWriteBytes: 60
ProfileEvent_ReadBufferAIORead: 0
ProfileEvent_ReadBufferAIOReadBytes: 0
ProfileEvent_WriteBufferAIOWrite: 0
ProfileEvent_WriteBufferAIOWriteBytes: 0

```

system.metric_log

This system table contains the history of metric values from the `system.metrics` and `system.events` tables, which are regularly flushed to the disk.

To enable metric history collection on `system.metric_log`, create `/etc/clickhouse-server/config.d/metric_log.xml` with the following content and restart `clickhouse-server`:

```

<yandex>
<metric_log>
<database>system</database>
<table>metric_log</table>
<flush_interval_milliseconds>7500</flush_interval_milliseconds>
<collect_interval_milliseconds>1000</collect_interval_milliseconds>
</metric_log>
</yandex>

```

Sample code:

```

SELECT * FROM system.metric_log LIMIT 1 FORMAT Vertical;

```

Row 1:

```

event_date:                2020-03-24
event_time:                2020-03-24 11:06:10
milliseconds:             437
ProfileEvent_Query:       0
ProfileEvent_SelectQuery: 0
ProfileEvent_InsertQuery: 0
ProfileEvent_FileOpen:    0
ProfileEvent_Seek:        0
ProfileEvent_ReadBufferFromFileDescriptorRead: 1
ProfileEvent_ReadBufferFromFileDescriptorReadFailed: 0
ProfileEvent_ReadBufferFromFileDescriptorReadBytes: 0
ProfileEvent_WriteBufferFromFileDescriptorWrite: 1
ProfileEvent_WriteBufferFromFileDescriptorWriteFailed: 0
ProfileEvent_WriteBufferFromFileDescriptorWriteBytes: 60
ProfileEvent_ReadBufferAIORead: 0
ProfileEvent_ReadBufferAIOReadBytes: 0
ProfileEvent_WriteBufferAIOWrite: 0
ProfileEvent_WriteBufferAIOWriteBytes: 0

```

system.numbers

This system table contains a single UInt64 column named `number` that contains almost all the natural numbers starting from zero. You can use this table for tests or for a brute force search as needed. **Reads from this table are not parallelized.**

system.numbers_mt

This system table is the same as `system.numbers`, but reads from it are parallelized. The numbers can be returned in any order (used for tests).

system.one

This system table contains a single row with a single `dummy` UInt8 column containing the value 0. It is used if a `SELECT` query does not specify the `FROM` clause. It is similar to the `DUAL` table found in other DBMSs.

system.parts

This system table contains information on parts of `MergeTree` tables. Each row describes one data part.

Column information:

Column Name	Type	Description
partition	String	Partition name
name	String	Data part name

Column Name	Type	Description
active	UInt8	Flag that indicates whether the data part is active. If it is active, it will be used in a table; otherwise, it will be deleted. Inactive data parts will be retained after merging
marks	UInt64	Number of marks
rows	UInt64	Number of rows
bytes_on_disk	UInt64	Total size of all data part files in bytes
data_compressed_bytes	UInt64	Total size of compressed data in data part. All the auxiliary files (e.g., files with marks) are not included
data_uncompressed_bytes	UInt64	Total size of uncompressed data in data part. All the auxiliary files (e.g., files with marks) are not included
marks_bytes	UInt64	Size of the file with marks
modification_time	DateTime	Time when the directory with the data part was modified
remove_time	DateTime	Time when the data part became inactive
refcount	UInt32	Number of places where the data part is used. A value greater than 2 indicates that the data part is used in queries or merges
min_date	Date	Minimum value of date key in data part
max_date	Date	Maximum value of date key in data part
min_time	DateTime	Minimum value of date and time key in data part
max_time	DateTime	Maximum value of date and time key in data part
partition_id	String	Partition ID
min_block_number	UInt64	Minimum number of data parts that make up the current part after merging
max_block_number	UInt64	Maximum number of data parts that make up the current part after merging

Column Name	Type	Description
level	UInt32	Depth of merge tree. 0 indicates that the current part was created by insertion rather than by merging other parts
data_version	UInt64	Mutations that should be applied to the data part (mutations with a version above <code>data_version</code>)
primary_key_bytes_in_memory	UInt64	Memory capacity in bytes used by primary key values
primary_key_bytes_in_memory_allocated	UInt64	Memory capacity (in bytes) reserved for primary key values
is_frozen	UInt8	Flag that indicates whether a partition data backup exists. 1: yes; 0: no
database	String	Database name
table	String	Table name
engine	String	Table engine name without parameters
path	String	Absolute path to the folder containing data part files
disk	String	Name of the disk that stores data parts
hash_of_all_files	String	Compressed files of SipHash128
hash_of_uncompressed_files	String	Uncompressed files of SipHash128 (files with marks, index files, etc.)
uncompressed_hash_of_compressed_files	String	Data in the SipHash128 compressed files as if they were uncompressed
bytes	UInt64	Alias for <code>bytes_on_disk</code>
marks_size	UInt64	Alias for <code>mark_bytes</code>

system.part_log

This system table can be created only if the `part_log` server setting is specified. It contains information on events that occurred with data parts in the `MergeTree` family tables, such as adding or merging data.

Column information:

Column Name	Type	Description
event_type	Enum	Type of the event that occurred with the data part. Valid values: NEW_PART, MERGE_PARTS, DOWNLOAD_PART, REMOVE_PART, MUTATE_PART, MOVE_PART
event_date	Date	Event date
event_time	DateTime	Event time
duration_ms	UInt64	Duration
database	String	Name of the database where the data part resides
table	String	Name of the table where the data part resides
part_name	String	Data part name
partition_id	String	ID of the partition into which the data part is inserted
rows	UInt64	Number of rows in data part
size_in_bytes	UInt64	Data part size in bytes
merged_from	Array(String)	Array of names of the parts that make up the current part (after merging)
bytes_uncompressed	UInt64	Size of uncompressed bytes
read_rows	UInt64	Number of rows read during merging
read_bytes	UInt64	Number of bytes read during merging
error	UInt16	Error code
exception	String	Error message

The `system.part_log` table will be created after the first insertion of data into the `MergeTree` table.

system.processes

This system table is used to implement the `SHOW PROCESSLIST` query.

Column information:

Column Name	Type	Description
-------------	------	-------------

Column Name	Type	Description
user	String	User who made the query
address	String	IP address from which the request was initiated
elapsed	Float64	Elapsed time in seconds since request execution started
rows_read	UInt64	Number of rows read from table. For distributed processing, on the requester server, this parameter is the total for all remote servers
bytes_read	UInt64	Number of uncompressed bytes read from table. For distributed processing, on the requester server, this parameter is the total for all remote servers
total_rows_approx	UInt64	Approximation of total number of the rows that should be read. For distributed processing, on the requester server, this parameter is the total for all remote servers.
memory_usage	UInt64	RAM capacity used by request. It might not include some types of dedicated memory
query	String	Query text. For <code>INSERT</code> , it does not include the data to insert
query_id	String	Query ID (if defined)

system.text_log

This system table contains log entries. Levels of logs that can be stored in this table can be limited with the `text_log.level` server setting.

Column information:

Column Name	Type	Description
event_date	Date	Entry date
event_time	DateTime	Entry time
microseconds	UInt32	Entry time
thread_name	String	Name of thread for logging
thread_id	UInt64	OS thread ID
level	Enum8	Log level. Valid values: 'Fatal' = 1, 'Critical' = 2, 'Error' = 3, 'Warning' = 4, 'Notice' = 5, 'Information' = 6, 'Debug' = 7, 'Trace' = 8

Column Name	Type	Description
query_id	String	Query ID
logger_name	LowCardinality(String)	Logger name (such as <code>DDLWorker</code>)
message	String	Log message
revision	UInt32	ClickHouse version
source_file	LowCardinality(String)	Source file from which the logging was done
source_line	UInt64	Source line from which the logging was done

system.query_log

This system table contains information on execution of queries. For each query, you can view the processing start time, processing duration, error message, and other information.

- This table does not contain input data for `INSERT` queries.
- ClickHouse will create this table only if the `query_log server` parameter is specified, which sets the logging rules, such as the logging interval or the name of the table into which the queries will be logged.
- To enable query logging, please set the `log_queries` parameter to 1.

The `system.query_log` table registers two types of queries:

- Initial queries that were run directly by the client.
- Subqueries that were initiated by other queries (for distributed query execution). For this type of queries, information on the parent queries is displayed in the `initial_*` columns.

Each query creates one or two rows in the `query_log` table based on the query status:

- If the query execution succeeds, two events of types 1 and 2 will be created (for more information, please see the `type` column).
- If an error occurs during query processing, two events of types 1 and 4 will be created.
- If an error occurs before the query is started, one event of type 3 will be created.

By default, logs are added to the table at intervals of 7.5 seconds. You can set this interval in the `query_log server` setting (for more information, please see the `flush_interval_milliseconds` parameter). To flush the logs forcibly from the memory buffer into the table, please use the `SYSTEM FLUSH LOGS` query.

When the table is deleted manually, it will be automatically created immediately. **All the previous logs will be deleted.**

Note: the log storage period is unlimited. Logs will not be automatically deleted from the table, and you need to delete outdated logs by yourself. You can specify an arbitrary partitioning key for the `system.query_log` table in the `query_log server` setting (for more information, please see the `partition_by` parameter).

system.query_thread_log

This system table contains information on each query execution thread.

- ClickHouse will create this table only if the `query_thread_log server` parameter is specified, which sets the logging rules, such as the logging interval or the name of the table into which the queries will be logged.
- To enable query logging, please set the `log_query_threads` parameter to 1.

By default, logs are added to the table at intervals of 7.5 seconds. You can set this interval in the `query_log server` setting (for more information, please see the `flush_interval_milliseconds` parameter). To flush the logs forcibly from the memory buffer into the table, please use the `SYSTEM FLUSH LOGS` query.

When the table is deleted manually, it will be automatically created immediately. **All the previous logs will be deleted.**

Note: the log storage period is unlimited. Logs will not be automatically deleted from the table, and you need to delete outdated logs by yourself. You can specify an arbitrary partitioning key for the `system.query_log` table in the `query_log server` setting (for more information, please see the `partition_by` parameter).

system.trace_log

This system table contains stack traces collected by the sampling query profiler. ClickHouse will create this table when the `trace_log server` configuration is set. You also need to set `query_profiler_real_time_period_ns` and `query_profiler_cpu_time_period_ns`. To analyze logs, please use the `addressToLine`, `addressToSymbol`, and `demangle` introspection functions.

Column information:

Column Name	Type	Description
event_date	Date	Sampling date
event_time	DateTIme	Sampling time
revision	UInt32	ClickHouse server version revision
timer_type	Enum8	Timer type. Valid values: Real, CPU

Column Name	Type	Description
thread_number	UInt32	Thread identifier
query_id	String	Query identifier that can be used to get details of a running query from the <code>query_log</code> system table
trace	Array(UInt64)	Stack trace at the moment of sampling. Each element is a virtual memory address inside the ClickHouse server process

Sample code:

```
SELECT * FROM system.trace_log LIMIT 1 \G;
```

Row 1:

```
event_date:      2020-03-24
event_time:      2020-03-24 11:12:36
revision:        54427
timer_type:      CPU
thread_number:   47
query_id:        2063fbac-110b-4afd-b7b4-f6cdf3ce507
trace:          [99887571,100858901,108095547,100858901,102292972,10
0858901,108755294,107833405,100858901,108095547,100858901,108095547
,100858901,100882409,100858901,100831379,100832522,55861763,5586940
0,55857811,126629183,140458940837317,140458935768941]
```

```
1 rows in set. Elapsed: 0.003 sec.
```

system.replicas

This system table contains information and status of replicated tables residing on the local server. It can be used for monitoring and contains a row for every `Replicated *` table.

Column information:

Column Name	Type	Description
database	String	Database name
table	String	Table name
engine	String	Table engine name
is_leader	UInt8	Whether the replica is the leader

Column Name	Type	Description
can_become_leader	UInt8	Whether the replica can be elected as the leader
is_readonly	UInt8	Whether the replica is in read-only mode
is_session_expired	UInt8	Expiration of session with ZooKeeper. It is basically the same as <code>is_readonly</code>
future_parts	UInt32	Number of data parts that will appear as the result of uncompleted <code>INSERT</code> statements or merges
parts_to_check	UInt32	Number of data parts in queue for verification. A part will be put into the verification queue if it is suspected to be corrupted
zookeeper_path	String	Path to table data in ZooKeeper
replica_name	String	Replica name in ZooKeeper. Different replicas of the same table have different names
replica_path	String	Path to replica data in ZooKeeper
columns_version	Int32	Version number of table structure, which indicates how many times <code>ALTER</code> has been executed
queue_size	UInt32	Size of queue for operations waiting to be performed. Operations include inserting blocks of data, merging, and certain other actions. It is usually the same as <code>future_parts</code>
inserts_in_queue	UInt32	Number of data blocks that need to be inserted. Insertions are usually replicated quickly. If this number is large, it means that something is wrong
merges_in_queue	UInt32	Number of merges waiting to be made. Sometimes merges are lengthy, so this value may be greater than zero for a long time
part_mutations_in_queue	UInt32	Number of mutations waiting to be made
queue_oldest_time	DateTime	Time when the earliest operation was added to the queue (this will be displayed only if <code>queue_size</code> is greater than 0)
inserts_oldest_time	DateTime	For more information, please see <code>queue_oldest_time</code>
merges_oldest_time	DateTime	For more information, please see <code>queue_oldest_time</code>
part_mutations_oldest_time	DateTime	For more information, please see <code>queue_oldest_time</code>
log_max_index	UInt64	Maximum number of entries in the log of general activity

Column Name	Type	Description
log_pointer	UInt64	Maximum number of entries in the log of general activity to whose execution queue the replica is copied plus one. If <code>log_pointer</code> is much smaller than <code>log_max_index</code> , something is wrong
last_queue_update	DateTime	Time when the queue was last updated
absolute_delay	UInt64	Delay in seconds of the current replica
total_replicas	UInt8	Total number of known replicas of this table
active_replicas	UInt8	Number of replicas of this table that have a session in ZooKeeper (i.e., number of running replicas)

Only one replica can be the leader at a time. The leader is responsible for selecting backend merges to perform. **Data can be written into any replica that is available and has a session in ZooKeeper, regardless of whether it is a leader.**

- If you request all the columns, the table may run a bit slowly, as several reads from ZooKeeper are made for each row.
- If you do not request the last 4 columns (`log_max_index`, `log_pointer`, `total_replicas`, and `active_replicas`), the table runs quickly.

For example, you can check whether everything is running correctly as follows:

```
SELECT
database,
table,
is_leader,
is_readonly,
is_session_expired,
future_parts,
parts_to_check,
columns_version,
queue_size,
inserts_in_queue,
merges_in_queue,
log_max_index,
log_pointer,
total_replicas,
active_replicas
FROM system.replicas
```

```

WHERE
is_readonly
OR is_session_expired
OR future_parts > 20
OR parts_to_check > 10
OR queue_size > 20
OR inserts_in_queue > 10
OR log_max_index - log_pointer > 10
OR total_replicas < 2
OR active_replicas < total_replicas

```

If this query does not return anything, it means that everything is normal.

```

SELECT
  database,
  table,
  is_leader,
  is_readonly,
  is_session_expired,
  future_parts,
  parts_to_check,
  columns_version,
  queue_size,
  inserts_in_queue,
  merges_in_queue,
  log_max_index,
  log_pointer,
  total_replicas,
  active_replicas
FROM system.replicas
WHERE is_readonly OR is_session_expired OR (future_parts > 20) OR (
parts_to_check > 10) OR (queue_size > 20) OR (inserts_in_queue > 10
) OR ((log_max_index - log_pointer) > 10) OR (total_replicas < 2) O
R (active_replicas < total_replicas)

Ok.

0 rows in set. Elapsed: 0.003 sec.

```

system.settings

This system table contains information on settings that are being used currently, i.e., used for executing queries to read from the `system.settings` table.

Column information:

Column Name	Type	Description
name	String	Setting name

Column Name	Type	Description
value	String	Setting value
changed	UInt8	Whether a setting is explicitly defined or changed in the configuration

Sample code:

```
SELECT name,value,changed FROM system.settings WHERE changed;
```

name	value	changed
use_uncompressed_cache	0	1
load_balancing	random	1
max_memory_usage	10000000000	1

system.table_engines

This system table contains descriptions of table engines supported by the server and their feature support information. It has only one column indicating the table engine name.

Sample code:

```
select * from system.table_engines limit 5;
```

name
JDBC
ODBC
HDFS
LiveView
Kafka
MaterializedView

system.tables

This system table contains metadata of each table that the server knows about. Detached tables are not displayed in

```
system.tables .
```

Column information:

Column Name	Type	Description
database	String	Name of the database where the table resides
name	String	Table name
engine	String	Table engine name (without parameters)
is_temporary	UInt8	Flag that indicates whether the table is temporary
data_path	String	Path to table data in file system
metadata_path	String	Path to table metadata in file system
metadata_modification_time	DateTime	Last modified time of table metadata
dependencies_database	Array(String)	Database dependencies
dependencies_table	Array(String)	Table dependencies (<code>MaterializedView</code> tables based on current table)
create_table_query	String	Query that was used to create the table
engine_full	String	Table engine parameters
partition_key	String	Partition key expression specified in table
sorting_key	String	Sorting key expression specified in table
primary_key	String	Primary key expression specified in table
sampling_key	String	Sampling key expression specified in table

The `system.tables` table is used to implement `SHOW TABLES` queries.

system.zookeeper

This system table can't exist if ZooKeeper is not configured. It allows reading data from the ZooKeeper cluster defined in the configuration. The query must have a `path` equality condition in the `WHERE` clause. This is the path to the descendants in ZooKeeper from which you want to get data.

The `SELECT * FROM system.zookeeper WHERE path = '/clickhouse'` query outputs data of all descendants on the `/clickhouse` node. To output data of all root nodes, write the path `='/'`. If the path specified in `path` does not exist, an exception will be thrown.

Column information:

Column Name	Type	Description
name	String	Node name
path	String	Node path
value	String	Node value
dataLength	Int32	Value size
numChildren	Int32	Number of descendants
czxid	Int64	ID of the transaction that created the node
mzxid	Int64	ID of the transaction that last changed the node
pzxid	Int64	ID of the transaction that last deleted or added descendants
ctime	DateTime	Node creation time
mtime	DateTime	Last modified time of node
version	Int32	Node version, i.e., number of times that the node was changed
cversion	Int32	Number of added or deleted descendants
aversion	Int32	Number of changes to ACL
ephemeralOwner	Int64	ID of the session that owns this node for an ephemeral node

system.mutations

This system table contains information on mutations of `MergeTree` tables and their progress. Each mutation command is represented by a single row.

Column information:

Column Name	Type	Description
database	String	Name of the database to which the mutation was applied
table	String	Name of the table to which the mutation was applied
mutation_id	String	Mutation ID
command	String	Mutation command
create_time	DateTime	Time when this mutation command was submitted for execution

Column Name	Type	Description
block_numbers.partition_id	Array(String)	Partition ID
block_numbers.number	Array(Int64)	Block number
parts_to_do	Int64	Number of data parts that need to be mutated for the mutation to be completed
is_done	UInt8	Flag indicates whether the mutation is completed
latest_failed_part	String	Name of the last part that could not be mutated
latest_fail_time	DateTime	Time of the last part mutation failure
latest_fail_reason	String	Message of the exception that caused the last part mutation failure

system.disks

This system table contains information on disks defined in the server configuration.

Column information:

Column Name	Type	Description
name	String	Disk name in server configuration
path	String	Path to mount point in file system
free_space	UInt64	Available disk space in bytes
total_space	UInt64	Disk volume in bytes
keep_free_space	UInt64	Disk space that should stay available on disk in bytes. It is defined in the <code>keep_free_space_bytes</code> parameter in disk configuration

system.storage_policies

This system table contains information on storage policies and volumes defined in the server configuration.

Column information:

Column Name	Type	Description
policy_name	String	Storage policy name

Column Name	Type	Description
volume_name	String	Volume name as defined in storage policy
volume_priority	UInt64	Volume priority in configuration
disks	Array(String)	Disk names as defined in storage policy
max_data_part_size	UInt64	Maximum size of data part that can be stored on volume disks (from 0 to infinite)
move_factor	Float32	Ratio of available disk capacity. When the ratio exceeds this value, ClickHouse will start to move data to the next volume in order

If the storage policy contains more than one volume, then information for each volume will be stored in individual rows of the table.

Access Permission Control

Last updated : 2020-05-28 17:44:45

User access permissions generally can be configured in the configuration file which is named `user.xml` by default and generally contains three sessions, namely, `users` , `profiles` , and `quotas` .

users Configuration

Username are recorded in this configuration file. The `users` session in the `user.xml` configuration file contains user access permissions. In the `users` session, each user has an independent subsession named after the username, which contains the following content:

- password: plaintext password.
- password_sha256_hex: SHA256 encryption.
- password_double_sha1_hex: double SHA1 encryption.
- networks: access source configuration. You can configure the IP and IP range with wildcards supported.
- profile: `profile` attribute used to control user resource usage.
- quota: `quota` attribute used to limit user resource usage with multi-dimensional thresholds in a time period.

profiles Configuration

A profile specifies the resource usage limits and can be configured in the `profiles` session in `user.xml` . You can configure multiple profiles at the same time, each of which has an independent subsession in the `profiles` session and contains the following content:

- max_memory_usage: maximum memory available for one single query in bytes.
- use_uncompressed_cache: whether to enable caching for uncompressed data. `0` indicates no, and `1` indicates yes.
- load_balancing: access policy used when distributed queries are performed on multiple replicas, which is `random` by default.
- readonly: read-only flag. `1` indicates read-only permission, and `0` indicates non-read-only permission.

quotas Configuration

A quota allows you to set multi-dimensional thresholds to control resource access in a configurable time period. The dimensions include number of queries, number of exceptional queries, total number of rows of query results, number of rows that are read within the cluster (the number of read columns is usually higher than the number of result rows because of filtering) in the time period, and query execution time (wall time in seconds). ClickHouse allows you to configure multiple quotas, each of which is an independent subsession in the `quotas` session and contains the following content:

- `duration`: time period length in seconds.
- `queries`: maximum number of queries in time period.
- `errors`: maximum number of exceptional queries in time period.
- `result_rows`: maximum number of query result rows in time period.
- `read_rows`: maximum number of rows that can be read for queries in the time period.
- `execution_time`: query execution time in time period in seconds.

If any of the limits above is reached in a time period, an exception will occur. If a limit is set to 0, it indicates that there is no limit for the corresponding metric. By default, the limits are set to 0.

Appendix

```
<?xml version="1.0"?>
<yandex>
  <!-- Profiles of settings. -->
  <profiles>
    <!-- Default settings. -->
    <default>
      <!-- Maximum memory usage for processing single query, in bytes. -->
      <max_memory_usage>10000000000</max_memory_usage>

      <!-- Use cache of uncompressed blocks of data. Meaningful only for processing many of very short queries. -->
      <use_uncompressed_cache>0</use_uncompressed_cache>

      <!-- How to choose between replicas during distributed query processing.
      random - choose random replica from set of replicas with minimum number of errors
      nearest_hostname - from set of replicas with minimum number of errors, choose replica
      with minimum number of different symbols between replica's hostname and local hostname
      (Hamming distance).
      in_order - first live replica is chosen in specified order.
      first_or_random - if first replica one has higher number of errors, pick a random
```

```
one from replicas with minimum number of errors.
-->
<load_balancing>random</load_balancing>
</default>

<!-- Profile that allows only read queries. -->
<readonly>
<readonly>1</readonly>
</readonly>
</profiles>

<!-- Users and ACL. -->
<users>
<!-- If user name was not specified, 'default' user is used. -->
<default>
<!-- Password could be specified in plaintext or in SHA256 (in hex format).

If you want to specify password in plaintext (not recommended), place it in 'password' element.
Example: <password>qwerty</password>.
Password could be empty.

If you want to specify SHA256, place it in 'password_sha256_hex' element.
Example: <password_sha256_hex>65e84be33532fb784c48129675f9eff3a682b27168c0ea744b2cf58ee02337c5</password_sha256_hex>
Restrictions of SHA256: impossibility to connect to ClickHouse using MySQL JS client (as of July 2019).

If you want to specify double SHA1, place it in 'password_double_sha1_hex' element.
Example: <password_double_sha1_hex>e395796d6546b1b65db9d665cd43f0e858dd4303</password_double_sha1_hex>

How to generate decent password:
Execute: PASSWORD=$(base64 < /dev/urandom | head -c8); echo "$PASSWORD"; echo -n "$PASSWORD" | sha256sum | tr -d '-'
In first line will be password and in second - corresponding SHA256.

How to generate double SHA1:
Execute: PASSWORD=$(base64 < /dev/urandom | head -c8); echo "$PASSWORD"; echo -n "$PASSWORD" | openssl dgst -sha1 -binary | openssl dgst -sha1
In first line will be password and in second - corresponding double SHA1.
-->
<password></password>

<!-- List of networks with open access.
```

To open access from everywhere, specify:

```
<ip>::/0</ip>
```

To open access only from localhost, specify:

```
<ip>::1</ip>
```

```
<ip>127.0.0.1</ip>
```

Each element of list has one of the following forms:

<ip> IP-address or network mask. Examples: 213.180.204.3 or 10.0.0.1/8 or 10.0.0.1/255.255.255.0

2a02:6b8::3 or 2a02:6b8::3/64 or 2a02:6b8::3/ffff:ffff:ffff:ffff::

<host> Hostname. Example: server01.yandex.ru.

To check access, DNS query is performed, and all received addresses compared to peer address.

<host_regexp> Regular expression for host names. Example, ^server\d\d-\d\d-\d\d\.yandex\.ru\$

To check access, DNS PTR query is performed for peer address and then regexp is applied.

Then, for result of PTR query, another DNS query is performed and all received addresses compared to peer address.

Strongly recommended that regexp is ends with \$

All results of DNS requests are cached till server restart.

-->

```
<networks incl="networks" replace="replace">
```

```
<ip>::/0</ip>
```

```
</networks>
```

```
<!-- Settings profile for user. -->
```

```
<profile>default</profile>
```

```
<!-- Quota for user. -->
```

```
<quota>default</quota>
```

```
<!-- For testing the table filters -->
```

```
<databases>
```

```
<test>
```

```
<!-- Simple expression filter -->
```

```
<filtered_table1>
```

```
<filter>a = 1</filter>
```

```
</filtered_table1>
```

```
<!-- Complex expression filter -->
```

```
<filtered_table2>
```

```
<filter>a + b < 1 or c - d > 5</filter>
```

```
</filtered_table2>
```

```
<!-- Filter with ALIAS column -->
```

```
<filtered_table3>
<filter>c = 1</filter>
</filtered_table3>
</test>
</databases>
</default>

<!-- Example of user with readonly access. -->
<!-- <readonly>
<password></password>
<networks incl="networks" replace="replace">
<ip>:1</ip>
<ip>127.0.0.1</ip>
</networks>
<profile>readonly</profile>
<quota>default</quota>
</readonly> -->
</users>

<!-- Quotas. -->
<quotas>
<!-- Name of quota. -->
<default>
<!-- Limits for time interval. You could specify many intervals with different li
mits. -->
<interval>
<!-- Length of interval. -->
<duration>3600</duration>

<!-- No limits. Just calculate resource usage for time interval. -->
<queries>0</queries>
<errors>0</errors>
<result_rows>0</result_rows>
<read_rows>0</read_rows>
<execution_time>0</execution_time>
</interval>
</default>
</quotas>
</yandex>
```

ClickHouse Data Migration Guide

Last updated : 2020-09-15 14:32:38

Feature Overview

A ClickHouse cluster consists of at least one node and allows you to define one or more virtual clusters through configuration. If the node quantity or storage volume of a virtual cluster changes, you can use the data migration feature to balance data distribution and improve the cluster resource utilization. This feature supports deactivation mode and balancing mode.

Use Cases

- In balancing mode, each target node will receive an equal amount of data migrated from source nodes, and only partition tables rather than non-partition tables will be migrated. It is suitable for cluster scale-out and node load balancing scenarios.
- In deactivation mode, all data including partition and non-partition tables from source nodes will be migrated to target nodes. It is suitable for node termination and data backup scenarios.

Directions

Log in to the [EMR Console](#), select **ClickHouse Cluster** > **Cluster Service** in the cluster list, and click **Operation** > **Data Migration** in the "ClickHouse" block.

← emr-hk8tzmed
ceshi-1234

Basic Info ^

- Instance Information
- Bootstrap Actions

Cluster Services

Cluster Resources v

Cluster Monitoring v

Operation Logs

Cluster Services

To receive or forward data in batches, it is recommended to use this service with CKafka.

[Add Component](#) [Restart Service](#)

CLICKHOUSE
Running | Version20.3.10.75

WebUI Address ⓘ

Operation ▾

Service St...

Role Man...

Configurat...

View port

Data Migr...

ZOOKEEPER
Running | Version3.4.9

WebUI Address ⓘ

Operation ▾

You can migrate data in the following four steps:

- Select Cluster:** select a cluster and the migration mode, which is **balance mode** by default. You can select only one cluster for each migration task.

Data Migration ✕

1 **Select a Cluster** >
 2 **Select Nodes** >
 3 **Select Data Tables** >
 4 **Confirm Information**

A ClickHouse cluster consists of one or multiple nodes. Users can configure one or multiple clusters in a ClickHouse cluster. When the number of cluster nodes or storage capacity changes, ClickHouse supports data migration so as to improve the utilization of cluster resources.

Select a Data Migration Type

Data Migration Type ⓘ Balancing Mode Deactivation Mode

Select a Cluster

Search by cluster nam

Cluster	Capacity ↕	Nodes ↕
<input checked="" type="radio"/> default_cluster	10.91 TB	1

共 1 条 10 ▾ 条 / 页 / 1 页

2. **Next: Select Nodes:** select the migration nodes and set the maximum data migration bandwidth, which is 200 MB/s by default (recommended) and can be customized. The system will check all nodes in the cluster by default and automatically mark source and target nodes, which can be slightly adjusted on your own. A node can be either a source node or target node, and a cluster must contain at least one source node and one target node.

Progress: **1** Select a Cluster > **2** Select Nodes > 3 Select Data Tables > 4 Confirm Information

Set a Data Bandwidth Cap

Data Bandwidth Cap MB/s

Select Nodes

Search for node IP

<input checked="" type="checkbox"/> Node IP	Spec	Storage	Used Cap	Used Storage	Operation
<input checked="" type="checkbox"/>	EMR Big DataD2 CPU: 8-core; memory: 32GB Local disk: 11176G x 1	10.91 TB	77.32 MB	0.00%	<input checked="" type="radio"/> Migrated From <input type="radio"/> Migrated To
<input checked="" type="checkbox"/>	EMR Big DataD2 CPU: 8-core; memory: 32GB Local disk: 11176G x 1	10.91 TB	33.00 MB	0.00%	<input type="radio"/> Migrated From <input checked="" type="radio"/> Migrated To

共 2 条 10 条 / 页 1 / 1 页

[Back](#) [Next: Select Data Tables](#)

3. **Next: Select Data Tables:** select the data tables to be migrated. The system will pull all tables from the source node and sort them in descending order by data volume. 10 tables are displayed per page and checked by default. You can search for tables by their thresholds and whether they contain or do not contain certain conditions.

4. **Next: Information Confirmation:** confirm the final migration information.

Note :

- Balancing mode: only partition tables will be migrated.

- Deactivation mode: partition and non-partition tables will be migrated.

Druid Development Guide

Druid Overview

Last updated : 2021-07-01 10:57:03

Apache Druid is a distributed data processing system supporting real-time and multi-dimensional online analytical processing (OLAP). It is used to implement quick and interactive query and analysis for large data sets.

Basic Characteristics

Characteristics of Apache Druid:

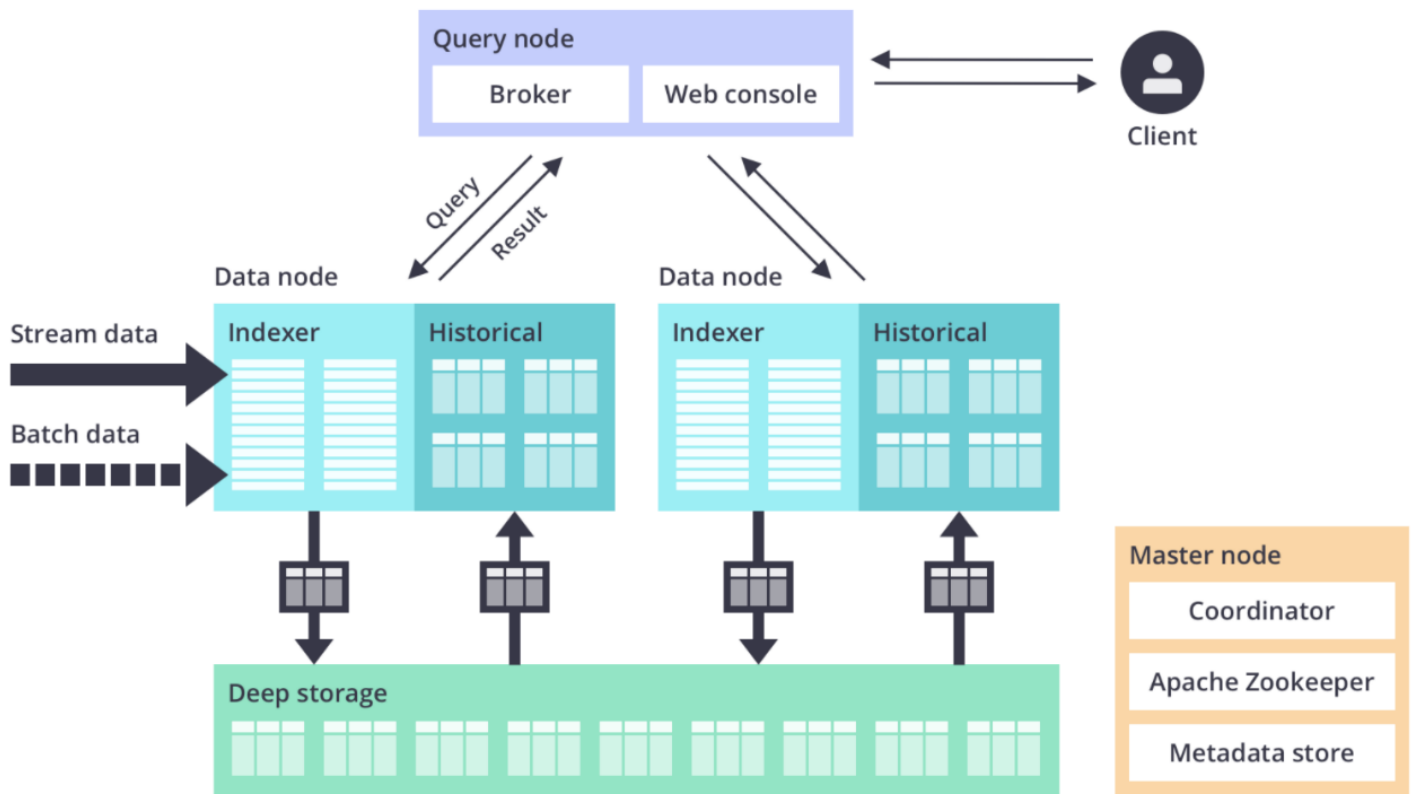
- It supports interactive queries with a subsecond response time and has various features such as multi-dimensional filtering, ad hoc attribute grouping, and quick data aggregation.
- It supports highly concurrent and real-time data ingestion to ensure real-timeliness for data ingestion and query.
- It features high scalability. With the distributed shared-nothing architecture, it supports quick processing of petabytes of data with hundreds of billions of events and sustains thousands of concurrent queries per second.
- It allows simultaneous online queries by multiple tenants.
- It supports high availability (HA) and rolling update.

Use Cases

Druid is most frequently used for flexible, quick, multi-dimensional OLAP analysis on big data. In addition, as it supports ingestion of pre-aggregated data and analysis of aggregated data based on timestamps, it is usually used in time-series data processing and analysis, such as ad platform, real-time metric monitoring, recommendation model, and search model.

System and Architecture

Druid uses a microservice-based architecture. All core services in it can be deployed on different hardware devices either separately or jointly.



Enhanced EMR Druid

A lot of improvements have been made on EMR Druid based on Apache Druid, including integration with EMR Hadoop and relevant Tencent Cloud ecosystem, convenient monitoring and OPS, and easy-to-use product APIs, so that you can use it out of the box in an OPS-free manner.

Currently, EMR Druid supports the following features:

- Easy integration with EMR Hadoop cluster
- Easy and quick elastic scalability
- HA
- Using COS as deep storage
- Using COS file as data source for batch indexing
- Metadata storage in TencentDB
- Integration with tools such as Superset
- Various monitoring metrics and alarm rules
- Failover
- High security

Druid Usage

Last updated : 2022-11-25 16:06:47

EMR allows you to deploy an E-MapReduce Druid cluster as an independent cluster based on the following considerations:

- Use case: E-MapReduce Druid can be used without Hadoop to adapt to different business use cases.
- Resource preemption: E-MapReduce Druid has high requirements for the memory, especially with the Broker and Historical nodes. The resource usage of E-MapReduce Druid is not scheduled by Hadoop YARN; therefore, resource preemption tends to occur during operations.
- Cluster size: As an infrastructure, Hadoop generally has a large size, while E-MapReduce Druid is relatively small. When they are deployed in the same cluster, resources may be wasted due to their different sizes. Therefore, separate deployment is more flexible.

Purchase suggestions

To purchase a Druid cluster, select Druid as the cluster type when creating the EMR cluster. The Druid cluster has built-in Hadoop HDFS and YARN services integrated with Druid, which are recommended for testing only. **We strongly recommend you use a dedicated Hadoop cluster in the production environment.**

To disable the built-in Hadoop services for the Druid cluster, you can select and suspend the target services on the [Cluster Service](#) page in the EMR console.

Configuring connectivity between Hadoop and Druid clusters

This section describes how to configure the connectivity between the Hadoop and Druid clusters. If you use the built-in Hadoop cluster in the Druid cluster (which is not recommended for the production environment), they can be properly connected with no additional settings required, and you can skip this section.

If you want to store the index data in the HDFS of another independent Hadoop cluster (which is recommended for the production environment), you need to configure the connectivity between the two clusters in the following steps:

1. Make sure that the Druid and Hadoop clusters can properly communicate with each other.
The two clusters should be in the same VPC. If they are in different VPCs, the two VPCs should be able to communicate with each other (through CCN or Peering Connection, for example).
2. Copy the `core-site.xml`, `hdfs-site.xml`, `yarn-site.xml`, and `mapred-site.xml` files in `/usr/local/service/hadoop/etc/hadoop` in the Hadoop cluster and paste them in `/usr/local/service/druid/conf/druid/_common` on each node in the E-MapReduce Druid cluster.

Note :

As the Druid cluster has a built-in Hadoop cluster, the relevant soft links to the files above already exist in the Druid path. You need to delete them first before copying the configuration files of another Hadoop cluster. In addition, you need to make sure that the file permissions are correct so that the files can be accessed by the `hadoop` user.

3. Modify the `common.runtime.properties` configuration file in Druid configuration management, save the change, and restart the Druid cluster services.

- `druid.storage.type`: It defaults to `hdfs` and does not need to be modified
- `druid.storage.storageDirectory`:

```
If the target Hadoop cluster is non-HA: hdfs://{namenode_ip}:4007  
If the target Hadoop cluster is HA: hdfs://HDFSXXXXX  
Configure the full path, which can be found in the `fs.defaultFS` configuration  
item in the `core-site.xml` file of the target Hadoop cluster.
```

Using COS

E-MapReduce Druid can use COS as the deep storage. This section describes how to configure COS as the deep storage of the Druid cluster.

First, you need to make sure that COS has been activated for both the Druid cluster and the target Hadoop cluster. You can activate COS when purchasing the clusters or configure COS in the EMR console after purchasing them.

1. Modify the `common.runtime.properties` configuration file in Druid configuration management:

- `druid.storage.type`: `hdfs`
- `druid.storage.storageDirectory`: `cosn://{bucket_name}/druid/segments`

You can create the `segments` directory on COS and set its permissions in advance.

2. Modify the `core-site.xml` configuration file in HDFS configuration management:

- Set `fs.cosn.impl` to `org.apache.hadoop.fs.CosFileSystem` .
- Add a new configuration item `fs.AbstractFileSystem.cosn.impl` and set it to `org.apache.hadoop.fs.CosN` .

- Put the JAR packages related to [hadoop-cos](#) (such as `cos_api-bundle-5.6.69.jar` and `hadoop-cos-2.8.5-8.1.6.jar`) into the `/usr/local/service/druid/extensions/druid-hdfs-storage`, `/usr/local/service/druid/hadoopdependencies/hadoop-client/2.8.5`, and `/usr/local/service/hadoop/share/hadoop/common/lib/` directories on each node of the cluster.

Save the configuration and restart the Druid cluster services.

Modifying Druid parameters

After you create the E-MapReduce Druid cluster, a set of configuration items will be generated automatically. However, we recommend you modify the memory configuration as needed to achieve the optimal performance. You can do so on the [Component Management](#) page in the EMR console.

When modifying the configuration, make sure that the modification is correct:

```
MaxDirectMemorySize >= druid.processing.buffer.sizeByte * (druid.processing.numMergeBuffers + druid.processing.numThreads + 1)
```

Modification suggestion:

```
druid.processing.numMergeBuffers = max(2, druid.processing.numThreads / 4)
druid.processing.numThreads = Number of cores - 1 (or 1)
druid.server.http.numThreads = max(10, (Number of cores * 17) / 16 + 2) + 30
```

For more information on the configuration, see [Configuration reference](#).

Using a router as a query node

Currently, a Druid cluster deploys the Broker process on the EMR master node by default. As there are many processes deployed on the master node, they may interfere with each other, which may lead to insufficient memory and compromise the query efficiency. In addition, many businesses require that the query nodes and core nodes be separately deployed. In this case, you can add one or more router nodes in the console and install the Broker processes so as to scale out the query nodes of the Druid cluster.

Accessing the web

You can access the Druid cluster in the console through the port 18888 on the master node and configure the public IP on your own. After opening port 18888 in the security group and setting the bandwidth, you can access the cluster

at [http://{masterIp}:18888] () .

Ingesting Data from Hadoop in Batches

Last updated : 2021-01-04 16:01:05

This document describes how to load data files from a remote Hadoop cluster into a Druid cluster in batches.

Operations in this document are all performed by the `Hadoop` user; therefore, please switch to the `Hadoop` user in both the Druid and Hadoop clusters.

Loading Data into Druid Cluster in Batches

1. Run the following commands to create directories as the `Hadoop` user in the remote Hadoop cluster:

```
hdfs dfs -mkdir /druid
hdfs dfs -mkdir /druid/segments
hdfs dfs -mkdir /quickstart
hdfs dfs -chmod 777 /druid
hdfs dfs -chmod 777 /druid/segments
hdfs dfs -chmod 777 /quickstart
```

Note :

If the Druid and Hadoop clusters are both self-deployed clusters, the directories need to be created on the corresponding Hadoop cluster (you also need to perform subsequent operations in the correct cluster). If the Druid and Hadoop clusters are the same cluster in the testing environment, you can perform the operations in the same cluster.

2. Upload the testing package.

The Druid cluster comes with a sample dataset named `Wikiticker` (located in

`/usr/local/service/druid/quickstart/tutorial/wikiticker-2015-09-12-`

`sampled.json.gz` by default). The operation of uploading the dataset in the Druid cluster to the corresponding remote Hadoop cluster **should be performed on the remote Hadoop cluster**.

```
hdfs dfs -put wikiticker-2015-09-12-sampled.json.gz /quickstart/wikiticker-2015-09-12-sampled.json.gz
```

3. Compile the index file.

Prepare an index file by running the following commands, which is still the Druid cluster's sample file

`/usr/local/service/druid/quickstart/tutorial/wikipedia-index-hadoop.json` :

```
{
  "type" : "index_hadoop",
  "spec" : {
    "dataSchema" : {
      "dataSource" : "wikipedia",
      "parser" : {
        "type" : "hadoopyString",
        "parseSpec" : {
          "format" : "json",
          "dimensionsSpec" : {
            "dimensions" : [
              "channel",
              "cityName",
              "comment",
              "countryIsoCode",
              "countryName",
              "isAnonymous",
              "isMinor",
              "isNew",
              "isRobot",
              "isUnpatrolled",
              "metroCode",
              "namespace",
              "page",
              "regionIsoCode",
              "regionName",
              "user",
              { "name": "added", "type": "long" },
              { "name": "deleted", "type": "long" },
              { "name": "delta", "type": "long" }
            ]
          },
          "timestampSpec" : {
            "format" : "auto",
            "column" : "time"
          }
        }
      },
      "metricsSpec" : [],
      "granularitySpec" : {
        "type" : "uniform",
        "segmentGranularity" : "day",
        "queryGranularity" : "none",
        "intervals" : ["2015-09-12/2015-09-13"],
        "rollup" : false
      }
    }
  }
}
```

```
},
"ioConfig" : {
  "type" : "hadoop",
  "inputSpec" : {
    "type" : "static",
    "paths" : "/quickstart/wikiticker-2015-09-12-sampled.json.gz"
  }
},
},
"tuningConfig" : {
  "type" : "hadoop",
  "partitionsSpec" : {
    "type" : "hashed",
    "targetPartitionSize" : 5000000
  },
  "forceExtendableShardSpecs" : true,
  "jobProperties" : {
    "yarn.nodemanager.vmem-check-enabled" : "false",
    "mapreduce.map.java.opts" : "-Duser.timezone=UTC -Dfile.encoding=UTF-8",
    "mapreduce.job.user.classpath.first" : "true",
    "mapreduce.reduce.java.opts" : "-Duser.timezone=UTC -Dfile.encoding=UTF-8",
    "mapreduce.map.memory.mb" : 1024,
    "mapreduce.reduce.memory.mb" : 1024
  }
},
},
"hadoopDependencyCoordinates" : ["org.apache.hadoop:hadoop-client:2.8.5"]
}
```

Note:

- `hadoopDependencyCoordinates` is the dependent Hadoop version.
 - `spec.ioConfig.inputSpec.paths` is the input file path. If you have already set cluster connectivity in the `common.runtime.properties` configuration item, you can use the relative path (for more information, please see [Druid Usage](#)); otherwise, you need to use a relative path starting with `hdfs://` or `cosn://` based on the actual conditions.
 - The `tuningConfig.jobProperties` parameter is used to set MapReduce job parameters.
- iv. Submit the indexing task.

Submit the task in the Druid cluster to ingest the data. Run the following command as the `Hadoop` user under the Druid directory:

```
./bin/post-index-task --file quickstart/tutorial/wikipedia-index-hadoop.json
--url http://localhost:8090
```

If the command succeeds, an output similar to the one below will be displayed:

...

Task finished **with status: SUCCESS**Completed **indexing data for** wikipedia. **Now** loading indexed **data** onto the cluster...wikipedia loading **complete!** You may **now query** your **data**

Data Query

Druid supports SQL-like and native JSON queries as described below. For more information, please see [Tutorial: Querying data](#).

Querying with SQL

Druid supports multiple SQL query methods:

- Perform queries in the `Query` menu on the web UI.

```
SELECT page, COUNT(*) AS Edits
FROM wikipedia
WHERE TIMESTAMP '2015-09-12 00:00:00' <= "__time" AND "__time" < TIMESTAMP '2015-09-13 00:00:00'
GROUP BY page
ORDER BY Edits DESC
LIMIT 10
```

- Use the command line tool `bin/dsql` for interactive queries on a query node.

```
[hadoop@172 druid]$ ./bin/dsql
Welcome to dsql, the command-line client for Druid SQL.
Connected to [http://localhost:8082/].
Type "\h" for help.
dsql> SELECT page, COUNT(*) AS Edits FROM wikipedia WHERE "__time" BETWEEN TIME
STAMP '2015-09-12 00:00:00' AND TIMESTAMP '2015-09-13 00:00:00' GROUP BY page O
RDER BY Edits DESC LIMIT 10;
```

page	Edits
Wikipedia:Vandalismmeldung	33
User:Cyde/List of candidates for speedy deletion/Subpage	28
Jeremy Corbyn	27
Wikipedia:Administrators' noticeboard/Incidents	21
Flavia Pennetta	20
Total Drama Presents: The Ridonculous Race	18
User talk:Dudeperson176123	18

```
| Wikipédia:Le Bistro/12 septembre 2015 | 18 |  
| Wikipedia:In the news/Candidates | 17 |  
| Wikipedia:Requests for page protection | 17 |
```

```
Retrieved 10 rows in 0.06s.
```

- Submit SQL queries over HTTP.

```
curl -X 'POST' -H 'Content-Type:application/json' -d @quickstart/tutorial/wikipedia-top-pages-sql.json http://localhost:18888/druid/v2/sql
```

The formatted output is as follows:

```
[  
  {  
    "page": "Wikipedia:Vandalismmeldung",  
    "Edits": 33  
  },  
  {  
    "page": "User:Cyde/List of candidates for speedy deletion/Subpage",  
    "Edits": 28  
  },  
  {  
    "page": "Jeremy Corbyn",  
    "Edits": 27  
  },  
  {  
    "page": "Wikipedia:Administrators' noticeboard/Incidents",  
    "Edits": 21  
  },  
  {  
    "page": "Flavia Pennetta",  
    "Edits": 20  
  },  
  {  
    "page": "Total Drama Presents: The Ridonculous Race",  
    "Edits": 18  
  },  
  {  
    "page": "User talk:Dudeperson176123",  
    "Edits": 18  
  },  
  {  
    "page": "Wikipédia:Le Bistro/12 septembre 2015",  
    "Edits": 18  
  },  
  {
```

```
"page": "Wikipedia:In the news/Candidates",
"Edits": 17
},
{
"page": "Wikipedia:Requests for page protection",
"Edits": 17
}
]
```

Querying through native JSON

- Directly enter JSON queries in the `Query` menu on the web UI.

```
{
"queryType" : "topN",
"dataSource" : "wikipedia",
"intervals" : ["2015-09-12/2015-09-13"],
"granularity" : "all",
"dimension" : "page",
"metric" : "count",
"threshold" : 10,
"aggregations" : [
{
"type" : "count",
"name" : "count"
}
]
}
```

- Submit the queries over HTTP under the Druid directory on a query node.

```
curl -X 'POST' -H 'Content-Type:application/json' -d @quickstart/tutorial/wikipedia-top-pages.json http://localhost:18888/druid/v2?pretty
```

The output is as follows:

```
[ {
"timestamp" : "2015-09-12T00:46:58.771Z",
"result" : [ {
"count" : 33,
"page" : "Wikipedia:Vandalismmeldung"
}, {
"count" : 28,
"page" : "User:Cyde/List of candidates for speedy deletion/Subpage"
}, {
"count" : 27,
"page" : "Jeremy Corbyn"
}
]
```

```
}, {  
  "count" : 21,  
  "page" : "Wikipedia:Administrators' noticeboard/Incidents"  
}, {  
  "count" : 20,  
  "page" : "Flavia Pennetta"  
}, {  
  "count" : 18,  
  "page" : "Total Drama Presents: The Ridonculous Race"  
}, {  
  "count" : 18,  
  "page" : "User talk:Dudeperson176123"  
}, {  
  "count" : 18,  
  "page" : "Wikipédia:Le Bistro/12 septembre 2015"  
}, {  
  "count" : 17,  
  "page" : "Wikipedia:In the news/Candidates"  
}, {  
  "count" : 17,  
  "page" : "Wikipedia:Requests for page protection"  
} ]  
} ]
```


Ingesting Data from Kafka in Real Time

Last updated : 2020-11-23 15:55:41

This document describes how to consume Kafka data in real time by using the Apache Druid Kafka indexing service. Before performing operations described in this document, just like in a Hadoop cluster, you need to make sure that the Kafka and Druid clusters can properly communicate with each other.

Note :

- The two clusters should be in the same VPC. If they are in different VPCs, the two VPCs should be able to communicate with each other (through CCN or Peering Connection, for example).
- If necessary, you should configure the Druid cluster with the host information of the Kafka cluster.

Using Command Line

1. Start the Kafka broker in the Kafka cluster.

```
./bin/kafka-server-start.sh config/server.properties
```

2. Create a Kafka topic named `mytopic` .

```
./bin/kafka-topics.sh --create --zookeeper {kafka_zk_ip}:2181 --replication-factor 1 --partitions 1 --topic mytopic  
Output:  
Created topic "mytopic".
```

{kafka_zk_ip}:2181 is the ZooKeeper address of the Kafka cluster.

3. Prepare a data description file `kafka-mytopic.json` in the Druid cluster.

```
{  
  "type": "kafka",  
  "dataSchema": {  
    "dataSource": "mytopic-kafka",  
    "parser": {  
      "type": "string",  
      "parseSpec": {  
        "timestampSpec": {  
          "column": "time",  
          "format": "auto"  
        },  
        "dimensionsSpec": {
```

```
"dimensions": ["url", "user"]
},
"format": "json"
},
"granularitySpec": {
  "type": "uniform",
  "segmentGranularity": "hour",
  "queryGranularity": "none"
},
"metricsSpec": [{
  "type": "count",
  "name": "views"
},
{
  "name": "latencyMs",
  "type": "doubleSum",
  "fieldName": "latencyMs"
}
],
"ioConfig": {
  "topic": "mytopic",
  "consumerProperties": {
    "bootstrap.servers": "{kafka_ip}:9092",
    "group.id": "kafka-indexing-service"
  },
  "taskCount": 1,
  "replicas": 1,
  "taskDuration": "PT1H"
},
"tuningConfig": {
  "type": "kafka",
  "maxRowsInMemory": "100000"
}
}
```

`{kafka_ip}:9092` is the IP and port of the `bootstrap.servers` in your Kafka cluster.

4. Add the Kafka supervisor on the master nodes in the Druid cluster.

```
curl -XPOST -H 'Content-Type: application/json' -d @kafka-mytopic.json http://
{druid_master_ip}:8090/druid/indexer/v1/supervisor
Output:
{"id": "mytopic-kafka"}
```

`{druid_master_ip}:8090` is the node where the `overload` process is deployed, which is generally a master node.

5. Enable a console producer in the Kafka cluster.

```
./bin/kafka-console-producer.sh --broker-list {kafka_ip}:9092 --topic mytopic
```

`{kafka_ip}:9092` is the IP and port of the `bootstrap.servers` in your Kafka cluster.

6. Prepare a query file named `query-mytopic.json` in the Druid cluster.

```
{
  "queryType" : "search",
  "dataSource" : "mytopic-kafka",
  "intervals" : ["2020-03-13T00:00:00.000/2020-03-20T00:00:00.000"],
  "granularity" : "all",
  "searchDimensions": [
    "url",
    "user"
  ],
  "query": {
    "type": "insensitive_contains",
    "value": "roni"
  }
}
```

7. Enter some data on Kafka in real time.

```
{"time": "2020-03-19T09:57:58Z", "url": "/foo/bar", "user": "brozo", "latencyMs": 62}
{"time": "2020-03-19T16:57:59Z", "url": "/", "user": "roni", "latencyMs": 15}
{"time": "2020-03-19T17:50:00Z", "url": "/foo/bar", "user": "roni", "latencyMs": 25}
```

Timestamp generation command:

```
python -c 'import datetime; print(datetime.datetime.utcnow().strftime("%Y-%m-%dT%H:%M:%SZ"))'
```

8. Perform a query in the Druid cluster.

```
curl -XPOST -H 'Content-Type: application/json' -d @query-mytopic.json http://{druid_ip}:8082/druid/v2/?pretty
```

`{druid_ip}:8082` is the broker node of your Druid cluster, which generally resides on a master or router node.

Query result:

```
[ {  
  "timestamp" : "2020-03-19T16:00:00.000Z",  
  "result" : [ {  
    "dimension" : "user",  
    "value" : "roni",  
    "count" : 2  
  } ]  
} ]
```

Using Web-Based Visualization

You can ingest data from a Kafka cluster and perform queries in the Druid Web UI Console in a visualized manner.

For detailed directions, please see [Loading data with the data loader](#).

TensorFlow Development Guide

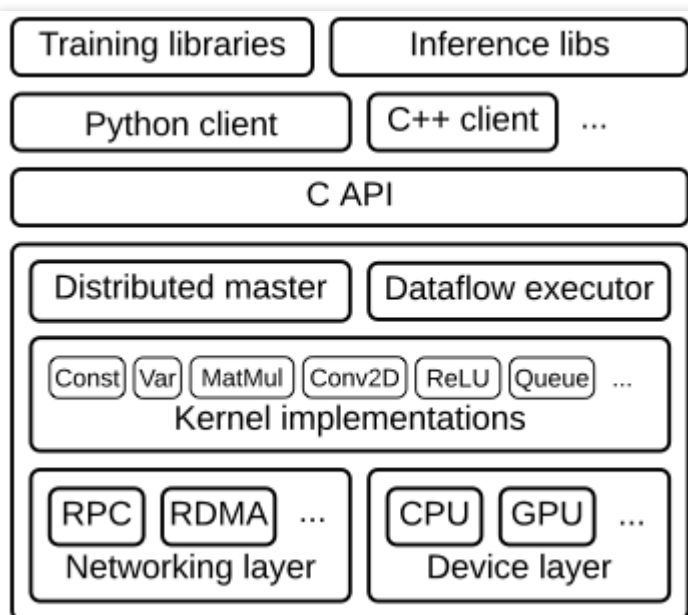
TensorFlow Overview

Last updated : 2021-07-01 15:23:49

TensorFlow is an end-to-end open source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries, and community resources that lets researchers push the state of the art in machine learning and developers easily build and deploy machine learning-powered applications.

- Easy model building
Build and train ML models easily using intuitive high-level APIs like Keras with eager execution, which makes for immediate model iteration and easy debugging.
- Reliable machine learning production anywhere
Easily train and deploy models in the cloud, on-prem, in the browser, or on-device no matter what language you use.
- Powerful experimentation for research
A simple and flexible architecture to take new ideas from concept to code, to state-of-the-art models, and to publication faster.

TensorFlow Architecture



- Client

It defines the computation process as a data flow graph and initializes graph execution by using `_Session_`.

- Distributed master

It prunes specific subgraphs in a graph, i.e, parameters defined in `Session.run()`, partitions a subgraph into multiple parts that run in different processes and devices, and distributes the graph parts to different worker services, which initialize subgraph computation.

- Worker service (one for each task)

It schedules the execution of graph operations by using kernel implementations appropriate to the available hardware (CPUs, GPUs, etc.) and sends/receives operation results to/from other worker services.

- Kernel implementation

It performs the computation for individual graph operations.

EMR's Support for TensorFlow

- TensorFlow version: v1.14.0
- Currently, TensorFlow can only run on CPU models instead of GPU models.
- TensorFlowOnSpark can be used for distributed training.

TensorFlow Development Sample

Write code: `test.py`

```
import tensorflow as tf
hello = tf.constant('Hello, TensorFlow!')
sess = tf.Session()
print sess.run(hello)
a = tf.constant(10)
b = tf.constant(111)
print sess.run(a+b)
exit()
```

Run the following command:

```
python test.py
```

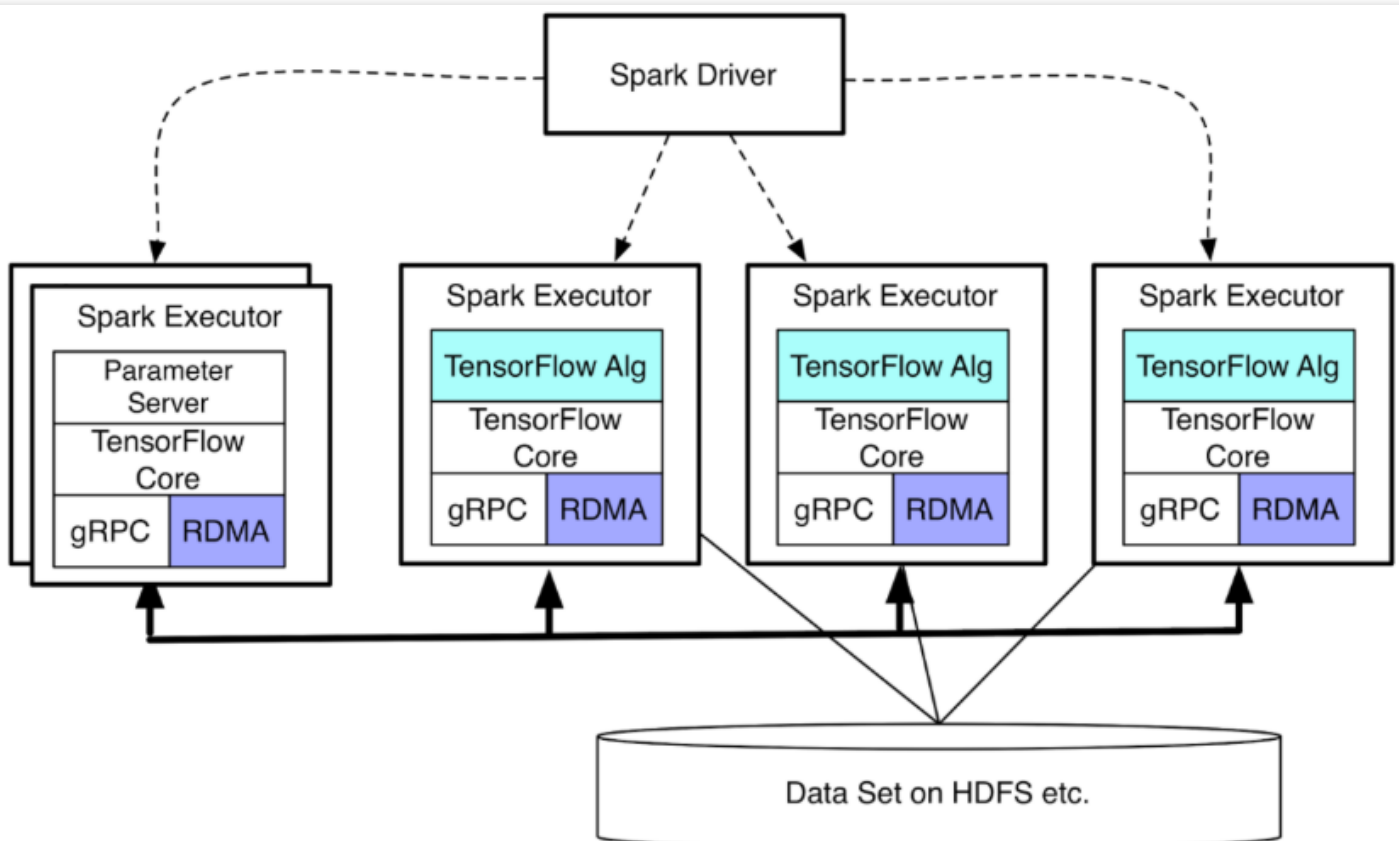
For more usage, please visit the TensorFlow official website.

TensorFlowOnSpark Overview

Last updated : 2022-04-18 15:29:03

TensorFlowOnSpark brings scalable deep learning to Apache Hadoop and Apache Spark clusters with support for TensorFlow programs in all types, async/sync training and inferencing, model parallelism, and parallel data processing. For more information, please visit [TensorFlowOnSpark official website](#).

TensorFlowOnSpark Architecture Diagram



TensorFlowOnSpark supports direct tensor communication among TensorFlow processes (workers and parameter servers). Process-to-process direct communication enables TensorFlowOnSpark programs to scale easily by adding machines. As TensorFlowOnSpark doesn't involve Spark drivers in tensor communication, it can achieve similar scalability as standalone TensorFlow clusters.

Installing TensorFlowOnSpark

1. Enter the EMR [purchase page](#) and select EMR v2.3.0 or above.
2. Select the `tensorflowonspark 1.4.4` component in the **Optional Component** list.
3. TensorFlowOnSpark will be installed in the `/usr/local/service/tensorflowonspark` directory by default.

Note :

The components depended on by TensorFlowOnSpark include Hive, Spark, etc., which will be installed together with TensorFlowOnSpark.

Use Cases

There is complete sample code in the directory of the installed TensorFlowOnSpark component. You can use TensorFlowOnSpark in the following steps:

- Download testing data

Run the following command in the `/usr/local/service/tensorflowonspark` directory as the `hadoop` user:

```
sh mnist_download.sh
cat mnist_download.sh
mkdir ${HOME}/mnist
pushd ${HOME}/mnist >/dev/null
curl -O "http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz"
curl -O "http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz"
curl -O "http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz"
curl -O "http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz"
zip -r mnist.zip *
popd >/dev/null
```

- Upload the original data and dependency packages

```
hdfs dfs -mkdir -p /mnist/tools/
hdfs dfs -put ~/mnist/mnist.zip /mnist/tools
hdfs dfs -mkdir /tensorflow
hdfs dfs -put TensorFlowOnSpark/tensorflow-hadoop-1.10.0.jar /tensorflow
```


- Prepare the feature data

```
sh prepare_mnist.sh
```

You can see that the feature data has been prepared:

```
hdfs dfs -ls /user/hadoop/mnist
Found 2 items
drwxr-xr-x - hadoop supergroup 0 2020-05-21 11:40 /user/hadoop/mnist/csv
drwxr-xr-x - hadoop supergroup 0 2020-05-21 11:41 /user/hadoop/mnist/tfr
```

- Train the model based on InputMode.SPARK

```
sh mnist_train_with_spark_cpu.sh
```

View the trained model:

```
[hadoop@10 tensorflow-on-spark]$ hdfs dfs -ls /user/hadoop/mnist_model
Found 10 items
-rw-r--r-- 1 hadoop supergroup 128 2020-05-21 11:46 /user/hadoop/mnist_model/che
kpoint
-rw-r--r-- 1 hadoop supergroup 243332 2020-05-21 11:46 /user/hadoop/mnist_model/e
vents.out.tfevents.1590032704.10.0.0.114
-rw-r--r-- 1 hadoop supergroup 164619 2020-05-21 11:45 /user/hadoop/mnist_model/g
raph.pbtxt
-rw-r--r-- 1 hadoop supergroup 814168 2020-05-21 11:45 /user/hadoop/mnist_model/m
odel.ckpt-0.data-00000-of-00001
-rw-r--r-- 1 hadoop supergroup 375 2020-05-21 11:45 /user/hadoop/mnist_model/mode
l.ckpt-0.index
-rw-r--r-- 1 hadoop supergroup 64658 2020-05-21 11:45 /user/hadoop/mnist_model/mo
del.ckpt-0.meta
-rw-r--r-- 1 hadoop supergroup 814168 2020-05-21 11:46 /user/hadoop/mnist_model/m
odel.ckpt-595.data-00000-of-00001
-rw-r--r-- 1 hadoop supergroup 375 2020-05-21 11:46 /user/hadoop/mnist_model/mode
l.ckpt-595.index
-rw-r--r-- 1 hadoop supergroup 64658 2020-05-21 11:46 /user/hadoop/mnist_model/mo
del.ckpt-595.meta
drwxr-xr-x - hadoop supergroup 0 2020-05-21 11:46 /user/hadoop/mnist_model/train
```

- Predict the model based on InputMode.SPARK

```
sh mnist_inference_with_spark_cpu.sh
```

View the prediction result:

```
hdfs dfs -cat /user/hadoop/predictions/part-00000 |more
2020-05-21T11:49:56.561506 Label: 7, Prediction: 7
2020-05-21T11:49:56.561535 Label: 2, Prediction: 2
2020-05-21T11:49:56.561541 Label: 1, Prediction: 1
2020-05-21T11:49:56.561545 Label: 0, Prediction: 0
2020-05-21T11:49:56.561550 Label: 4, Prediction: 4
2020-05-21T11:49:56.561555 Label: 1, Prediction: 1
2020-05-21T11:49:56.561559 Label: 4, Prediction: 4
2020-05-21T11:49:56.561564 Label: 9, Prediction: 9
2020-05-21T11:49:56.561568 Label: 5, Prediction: 6
2020-05-21T11:49:56.561573 Label: 9, Prediction: 9
2020-05-21T11:49:56.561578 Label: 0, Prediction: 0
2020-05-21T11:49:56.561582 Label: 6, Prediction: 6
2020-05-21T11:49:56.561587 Label: 9, Prediction: 9
2020-05-21T11:49:56.561603 Label: 0, Prediction: 0
2020-05-21T11:49:56.561608 Label: 1, Prediction: 1
2020-05-21T11:49:56.561612 Label: 5, Prediction: 5
```

- Train the model based on InputMode.TENSORFLOW

```
sh mnist_train_with_tf_cpu.sh
```

View the model:

```
hdfs dfs -ls mnist_model
Found 25 items
-rw-r--r-- 1 hadoop supergroup 265 2020-05-21 14:58 mnist_model/checkpoint
-rw-r--r-- 1 hadoop supergroup 40 2020-05-21 14:53 mnist_model/events.out.tfevent
s.1590044017.10.0.0.144
-rw-r--r-- 1 hadoop supergroup 40 2020-05-21 14:57 mnist_model/events.out.tfevent
s.1590044221.10.0.0.144
-rw-r--r-- 1 hadoop supergroup 40 2020-05-21 14:57 mnist_model/events.out.tfevent
s.1590044227.10.0.0.144
-rw-r--r-- 1 hadoop supergroup 40 2020-05-21 14:57 mnist_model/events.out.tfevent
s.1590044232.10.0.0.144
-rw-r--r-- 1 hadoop supergroup 40 2020-05-21 14:57 mnist_model/events.out.tfevent
s.1590044238.10.0.0.144
```

```
-rw-r--r-- 1 hadoop supergroup 40 2020-05-21 14:58 mnist_model/events.out.tfevent
s.1590044303.10.0.0.114
-rw-r--r-- 1 hadoop supergroup 198078 2020-05-21 14:58 mnist_model/graph.pbtxt
drwxr-xr-x - hadoop supergroup 0 2020-05-21 14:58 mnist_model/inference
-rw-r--r-- 1 hadoop supergroup 814168 2020-05-21 14:57 mnist_model/model.ckpt-238
.data-00000-of-00001
-rw-r--r-- 1 hadoop supergroup 375 2020-05-21 14:57 mnist_model/model.ckpt-238.in
dex
-rw-r--r-- 1 hadoop supergroup 76255 2020-05-21 14:57 mnist_model/model.ckpt-238.
meta
-rw-r--r-- 1 hadoop supergroup 814168 2020-05-21 14:57 mnist_model/model.ckpt-277
.data-00000-of-00001
-rw-r--r-- 1 hadoop supergroup 375 2020-05-21 14:57 mnist_model/model.ckpt-277.in
dex
-rw-r--r-- 1 hadoop supergroup 76255 2020-05-21 14:57 mnist_model/model.ckpt-277.
meta
-rw-r--r-- 1 hadoop supergroup 814168 2020-05-21 14:57 mnist_model/model.ckpt-315
.data-00000-of-00001
-rw-r--r-- 1 hadoop supergroup 375 2020-05-21 14:57 mnist_model/model.ckpt-315.in
dex
-rw-r--r-- 1 hadoop supergroup 76255 2020-05-21 14:57 mnist_model/model.ckpt-315.
meta
-rw-r--r-- 1 hadoop supergroup 814168 2020-05-21 14:57 mnist_model/model.ckpt-354
.data-00000-of-00001
-rw-r--r-- 1 hadoop supergroup 375 2020-05-21 14:57 mnist_model/model.ckpt-354.in
dex
-rw-r--r-- 1 hadoop supergroup 76255 2020-05-21 14:57 mnist_model/model.ckpt-354.
meta
-rw-r--r-- 1 hadoop supergroup 814168 2020-05-21 14:57 mnist_model/model.ckpt-393
.data-00000-of-00001
-rw-r--r-- 1 hadoop supergroup 375 2020-05-21 14:57 mnist_model/model.ckpt-393.in
dex
-rw-r--r-- 1 hadoop supergroup 76255 2020-05-21 14:57 mnist_model/model.ckpt-393.
meta
drwxr-xr-x - hadoop supergroup 0 2020-05-21 14:53 mnist_model/train
```

- Predict the model based on InputMode.TENSORFLOW

```
sh mnist_train_with_tf_cpu.sh
```

View the prediction result:

```
hdfs dfs -cat predictions/part-00000 |more
9 4
9 9
```

4 4
1 1
4 4
8 8
9 9
2 2
3 5
6 6
9 9
2 2
6 6
0 0
7 7
5 5
3 3

Jupyter Development Guide

Jupyter Notebook Overview

Last updated : 2022-04-18 15:35:45

Jupyter Notebook Overview

Jupyter Notebook is a web application for interactive computing throughout the whole process from development and documentation writing to code execution and result display. For more information, please see [The Jupyter Notebook](#).

In short, Jupyter Notebook is opened on a webpage where you can directly write and run code, and the code execution result will be directly displayed below the code block. If you need to write help documents during programming, you can write them directly on the same page to describe and explain the code promptly.

Components

- Web application
A web application is a tool in the form of a webpage that provides help documentation writing, mathematical formulas, interactive computation, and other rich media features. In brief, it is a tool that can implement various features.
- Document
All interactive computations, explanatory text writing, mathematical formulas, images, and other rich media inputs and outputs in Jupyter Notebook are represented as documents. These documents are `JSON` files with the `.ipynb` extension, which facilitate version control and file sharing. In addition, documents can be exported in various formats such as HTML, LaTeX, and PDF.

Main features of Jupyter Notebook

1. It provides syntax highlighting, indentation, and tab completion for programming.
2. It can run code directly in a browser with the result displayed below the code block.
3. It displays the computation results in rich media formats such as HTML, LaTeX, PNG, and SVG.
4. It supports Markdown syntax for writing of code help documents or statements.
5. It allows you to use LaTeX to write code comments containing mathematical notations.

Installing Jupyter

Enter the EMR [purchase page](#).

- Select `EMR-V2.3.0` as the product version.
- Select **tensorflowspark 1.4.4** in the **Optional Component** list, and `Jupyter` will be automatically installed in the `/usr/local/service/jupyter` directory. Jupyter will not start any service. If you have not installed TensorFlowOnSpark, the default installation path will be `/usr/local/service/apps/jupyter`.

Using Jupyter

Initializing Jupyter configuration

```
Usage: init.sh [password] [port]
# Sample
./init.sh 123456 10086
```

Press Enter consecutively until the following information is displayed:

```
[hadoop@10 jupyter]$ ./init.sh 123456 10086
Your password is: 123456
Your signature is: sha1:139fa061bae6:bcdc6a7870878458c7c14594fe65dd21f85f84a4
Generating a 4096 bit RSA private key
.....++
.....
++
writing new private key to '/usr/local/service/jupyter/conf/jkey.key'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [XX]:
State or Province Name (full name) []:
Locality Name (eg, city) [Default City]:
Organization Name (eg, company) [Default Company Ltd]:
Organizational Unit Name (eg, section) []:
Common Name (eg, your name or your server's hostname) []:
Email Address []:
Jupyter config has already generated at /usr/local/service/jupyter/conf/jupyter_n
otebook_config.py
Now, you can execute the following command to start jupyter:
jupyter notebook --config=/usr/local/service/jupyter/conf/jupyter_notebook_conf
```

```
g.py --allow-root
```

At this point, Jupyter configuration has been generated. You can also modify relevant parameters in the generated configuration file `jupyter_notebook_config.py` as instructed on the Jupyter official website.

Note :

The last row is the startup command. You can copy it to start Jupyter.

Starting Jupyter Notebook

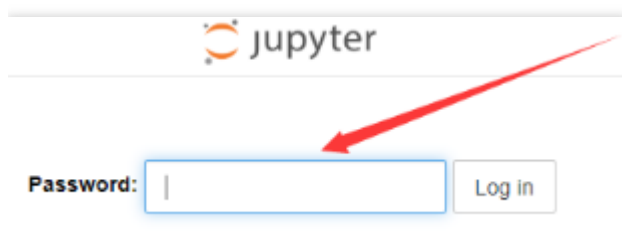
```
jupyter notebook --config=/usr/local/service/jupyter/conf/jupyter_notebook_config.py --allow-root
[I 10:47:46.972 NotebookApp] Writing notebook server cookie secret to /home/hadoop/.local/share/jupyter/runtime/notebook_cookie_secret
[I 10:47:47.748 NotebookApp] Serving notebooks from local directory: /usr/local/service/jupyter
[I 10:47:47.749 NotebookApp] The Jupyter Notebook is running at:
[I 10:47:47.749 NotebookApp] https://(10.0.0.7 or 127.0.0.1):10086/
[I 10:47:47.749 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
```

Accessing Jupyter

Open Jupyter on a webpage (before this operation, you may need to open the Jupyter port in the security group):

```
https://IP:10086/
```

Here, port 10086 is a parameter in the `init.sh` initialization above.



You can enter Jupyter's homepage by simply entering the password configured just now.

Using Jupyter for development

Creating directory

Renaming directory

Writing TensorFlow code

For more information, please visit the [TensorFlow official website](#).

Note :

You need to download the data set here. The download may be slow in Mainland China.

```
import tensorflow as tf
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(512, activation=tf.nn.relu),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

Running code

Train the model in Jupyter again.

Stopping Jupyter service

```
./stop_jupy.sh [jupyter_port]
```


Kudu Development Guide

Kudu Overview

Last updated : 2022-12-02 14:52:56

Apache Kudu is a distributed and horizontally scalable columnar storage system. It improves the storage layer of Hadoop and can quickly analyze rapidly changing data.

Basic Kudu Features

- Fast processing of OLAP workloads.
- Integration with MapReduce, Spark, and other Hadoop ecosystem components.
- Tight integration with Apache Impala, making it a good and mutable alternative to using HDFS with Apache Parquet.
- Flexible consistency model.
- Strong performance for running sequential and random workloads simultaneously.
- High data availability and storage reliability backed by the Raft protocol.
- Structured data model.

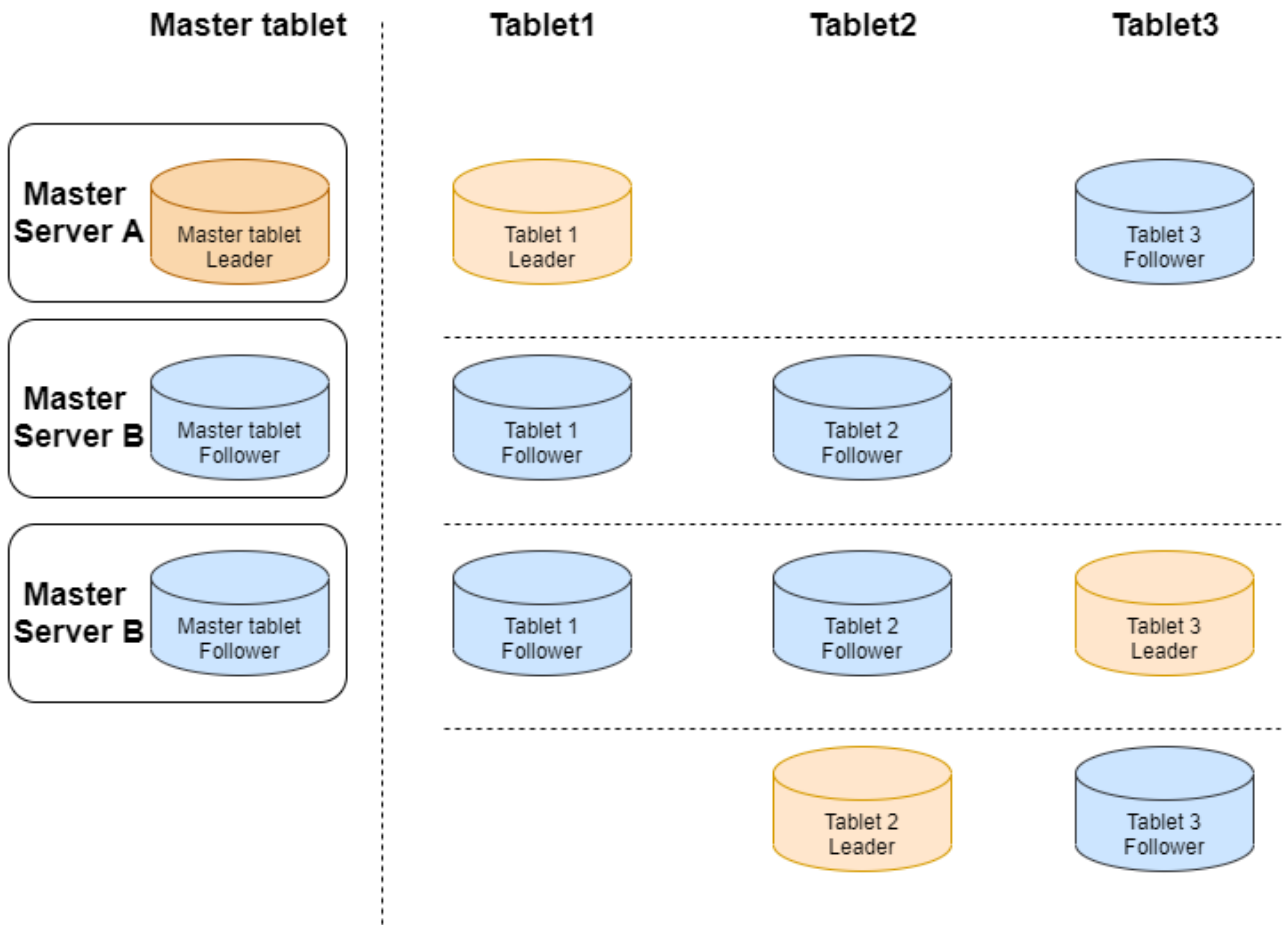
Kudu Use Cases

- Complex scenarios involving both random access and batch data scanning.
- Scenarios with high computational load.
- Application of real-time predication models, which supports periodic model update based on all historical data.
- Data update, which avoids repeated data migration.
- Cross-region real-time data backup and query.

Basic Kudu Architecture

Kudu contains the following two types of components:

- Master, which is mainly responsible for managing metadata information, listening on servers, and reassigning tablets in case of server failures.
- Tablet server, which is mainly responsible for tablet storage and data CRUD.



Kudu Usage

EMR 2.4.0 supports the Kudu component. If you check the Kudu component when creating a Hadoop cluster, a Kudu cluster will be created. By default, it contains 3 Kudu masters, and high availability is enabled for it.

Note :
All IPs used below are private IPs.

- Integrate Impala with Kudu

```
[172.30.0.98:27001] > CREATE TABLE t2(id BIGINT, name STRING, PRIMARY KEY(id)) PARTITION BY HASH PARTITIONS 2 STORED AS KUDU TBLPROPERTIES ('kudu.master_addresses' = '172.30.0.240,172.30.1.167,172.30.0.96,172.30.0.94,172.
```

```
30.0.214',
'kudu.num_tablet_replicas' = '1');
Query: create TABLE t2 (id BIGINT,name STRING,PRIMARY KEY(id)) PARTITION BY HASH
PARTITIONS 2 STORED AS KUDU TBLPROPERTIES (
'kudu.master_addresses' = '172.30.0.240,172.30.1.167,172.30.0.96,172.30.0.94,172.
30.0.214',
'kudu.num_tablet_replicas' = '1')
Fetched 0 row(s) in 0.12s
[hadoop@172 root]$ /usr/local/service/kudu/bin/kudu table list 172.30.0.240,172.3
0.1.167,172.30.0.96,172.30.0.94,172.30.0.214
impala::default.t2
```

- Insert data

```
[172.30.0.98:27001] > insert into t2 values(1, 'test');
Query: insert into t2 values(1, 'test')
Query submitted at: 2020-08-10 20:07:21 (Coordinator: http://172.30.0.98:27004)
Query progress can be monitored at: http://172.30.0.98:27004/query_plan?query_id=
b44fe203ce01254d:b055e98200000000
Modified 1 row(s), 0 row error(s) in 5.63s
```

- Query data based on Impala

```
[172.30.0.98:27001] > select * from t2;
Query: select * from t2
Query submitted at: 2020-08-10 20:09:47 (Coordinator: http://172.30.0.98:27004)
Query progress can be monitored at: http://172.30.0.98:27004/query_plan?query_id=
ec4c9706368f135d:f20ccb6e00000000
+----+-----+
| id | name |
+----+-----+
| 1 | test |
+----+-----+
Fetched 1 row(s) in 0.20s
```

- Other commands

- i. Perform health check for the cluster

```
[hadoop@172 root]$ /usr/local/service/kudu/bin/kudu cluster ksck 172.30.0.240,17
2.30.1.167,172.30.0.96,172.30.0.94,172.30.0.214
```

- ii. Create a table

```
[hadoop@172 root]$ /usr/local/**service**/**kudu**/bin/kudu table create '172.30.0.240,172.30.1.167,172.30.0.96,172.30.0.94,172.30.0.214' '{"table_name":"test","schema":{"columns":[{"column_name":"id","column_type":"INT32","default_value":"1"}, {"column_name":"key","column_type":"INT64","is_nullable":false,"comment":"range key"}, {"column_name":"name","column_type":"STRING","is_nullable":false,"comment":"user name"}],"key_column_names":["id","key"]},"partition":{"hash_partitions":[{"columns":["id"],"num_buckets":2,"seed":100}],"range_partition":{"columns":["key"],"range_bounds":[{"upper_bound":{"bound_type":"inclusive","bound_values":["2"]}}, {"lower_bound":{"bound_type":"exclusive","bound_values":["2"]},"upper_bound":{"bound_type":"inclusive","bound_values":["3"]}}]}}},"extra_configs":{"configs":{"kudu.table.history_max_age_sec":"3600"}},"num_replicas":1}'
```

iii. Query the created `test` table

```
[hadoop@172 root]$ /usr/local/service/kudu/bin/kudu table list 172.30.0.240,172.30.1.167,172.30.0.96,172.30.0.94,172.30.0.214
test
```

iv. View table structure

```
[hadoop@172 root]$ /usr/local/service/kudu/bin/kudu table describe 172.30.0.240,172.30.1.167,172.30.0.96,172.30.0.94,172.30.0.214 test
TABLE test (
id INT32 NOT NULL,
key INT64 NOT NULL,
name STRING NOT NULL,
PRIMARY KEY (id, key)
)
HASH (id) PARTITIONS 2 SEED 100,
RANGE (key) (
PARTITION VALUES < 3,
PARTITION 3 <= VALUES < 4
)
REPLICAS 1
```

Ranger Development Guide

Ranger Overview

Last updated : 2021-06-07 17:25:20

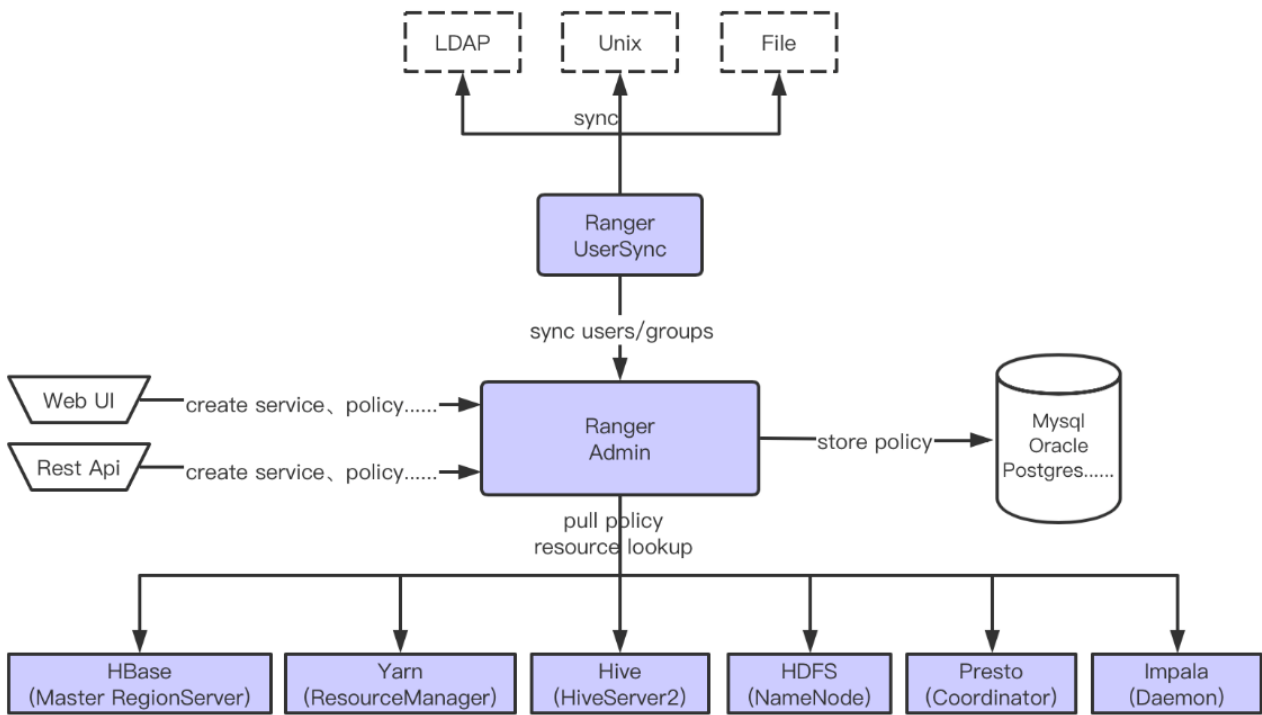
Ranger Overview

Ranger is a framework for centralized security management of Hadoop components in the field of big data. Users can use Ranger to securely access data in a cluster. It is mainly designed to monitor Hadoop components, and control service launch and resource access. The major goals of Ranger include:

- Centralized security management of big data components in the web UI or using RESTful APIs provided by Ranger.
- Authorization to perform specific operations with big data components based on roles and attributes.
- Centralized auditing of user access and administrative operations (security-related) with big data components.

Ranger Architecture

Ranger is mainly composed of Ranger Admin, Ranger UserSync, and Ranger Plugin. Both Ranger Admin and Ranger UserSync are a separate JVM process, while Ranger Plugin needs to be installed on different nodes depending on different components.



- Ranger Admin is used to manage configured policies, created services, and audit logs and reports. It also persistently saves the policies and services to the database for regular queries from Ranger Plugin.
- Ranger UserSync is used to synchronize information from LDAP, File, and Unix to Ranger Admin, for example, user and group information from users' Unix or LDAP directory access systems. To synchronize user and group information from Unix, you need to enable the `unixAuthenticationService` process and persistently store the synchronized information.
- Ranger Plugin will be deployed on service nodes as needed and periodically synchronize policy information from Ranger Admin.

The following table lists the components that can be integrated with Ranger.

Service	Installation Nodes	EMR Versions
HDFS	NameNode	EMR v2.0.1 and above
HBase	Master, RegionServer	EMR v2.0.1 and above
Hive	HiveServer2	EMR v2.0.1 and above
YARN	ResourceManager	EMR v2.0.1 and above
Presto	All coordinators	EMR v2.0.1 and above

Service	Installation Nodes	EMR Versions
Impala	All daemons	EMR v2.2.0 and above
Kudu	All masters	EMR v3.2.0

Ranger User Guide

Integrating HDFS with Ranger

Last updated : 2021-06-07 17:25:20

Preparations

Ranger is available only when it is selected in **Optional Components** when you purchase a cluster. If you add the Ranger component after purchasing the cluster, the Web UI may be inaccessible. By default, when Ranger is installed, Ranger Admin and Ranger UserSync are deployed on the master node, and Ranger Plugin is deployed on the main daemon node of the embedded component.

When creating a cluster of the Hadoop type, you can select Ranger in **Optional Components**. The Ranger version varies depending on the EMR version you choose.

Note :

When the cluster type is Hadoop and the Ranger optional component is selected, EMR-Ranger will create services for HDFS and YARN by default and set default policies.

The screenshot displays the configuration interface for creating an EMR cluster. It includes the following sections:

- Cluster Type:** HADOOP (selected), DRUID, CLICKHOUSE, DORIS, KAFKA.
- Product Version:** EMR-V2.5.0 (selected), with a link to [Product Version Introduction](#).
- Required Components:** hadoop 2.8.5, zookeeper 3.6.1, Knox 1.2.0.
- Optional Components:** hbase 1.4.9, hive 2.3.7, hue 4.6.0, oozie 5.1.0, **ranger 1.2.0** (highlighted with a red box), spark_hadoop2.8 3.0.0, sqoop 1.4.7, tez 0.9.2, flume 1.9.0, impala 2.10.0, alluxio 2.3.0, flink 1.10.0, storm 1.2.3, kylin 2.5.2, ganglia 3.7.2, superset 0.35.2, livy 0.7.0, zeppelin 0.8.2, tensorflowspark 1.4.4, kudu 1.12.0, prestosql 332.
- Advanced Settings:** A link to expand more options.
- Next Step:** Hardware Configuration.

Ranger Web UI

Before accessing the Ranger Web UI, make sure that the current cluster is configured with a public IP and click the Ranger Web UI URL on the **Cluster Service** page.

The screenshot shows the 'Cluster Service' page in the Elastic MapReduce console. The left sidebar has 'Cluster Service' selected. The main area displays a table of services:

Service Name	Status	Version	WebUI Address
ZOOKEEPER	Running	Version3.6.1	--
HDFS	Running	Version2.8.5	https://1.1.1.1:8080/gateway/emr/hdfs
YARN	Running	Version2.8.5	https://1.1.1.1:8080/gateway/emr/yarn
KNOX	Running	Version1.2.0	--
RANGER	Running	Version1.2.0	https://1.1.1.1:8080/gateway/emr/ranger/

After you are redirected, enter the username and password that you set when you purchased the cluster.

The screenshot shows the Ranger Service Manager interface. The 'Service Manager' section is active, displaying a grid of services:

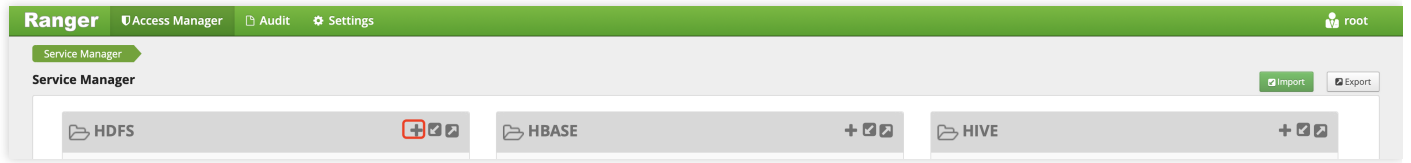
- HDFS (hdfs)
- YARN (yarn)
- SOLR
- KYLIN
- ATLAS
- HBASE (hbase)
- KNOX
- KAFKA
- NIFI-REGISTRY
- PRESTO
- HIVE (hive)
- STORM
- NIFI
- SQOOP

Integrating HDFS with Ranger

Note :

Make sure that HDFS related services are running normally and Ranger has been installed in the current cluster.

1. Add an EMR Ranger HDFS service on the EMR Ranger Web UI.



2. Configure EMR Ranger HDFS service parameters.

Service Manager > Edit Service

Edit Service

Service Details :

Service Name *

Description

Active Status Enabled Disabled

Select Tag Service

Config Properties :

Username *

Password *

Namenode URL *

Authorization Enabled

Authentication Type *

hadoop.security.auth_to_local

dfs.datanode.kerberos.principal

dfs.namenode.kerberos.principal

dfs.secondary.namenode.kerberos.principal

RPC Protection Type

Common Name for Certificate

Add New Configurations

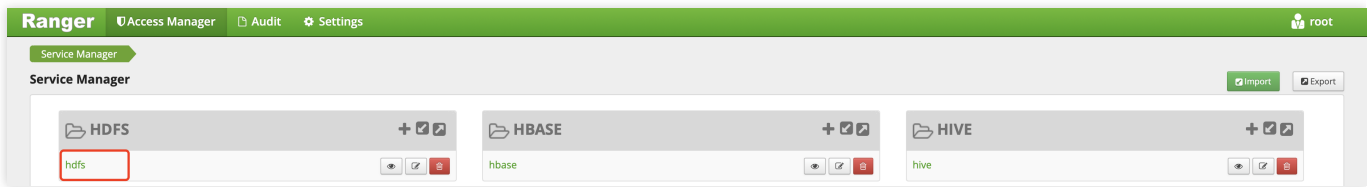
Name	Value
policy.grantrevoke.auth.users	<input type="text" value="hadoop"/> ✕

Parameter	Required	Description
-----------	----------	-------------

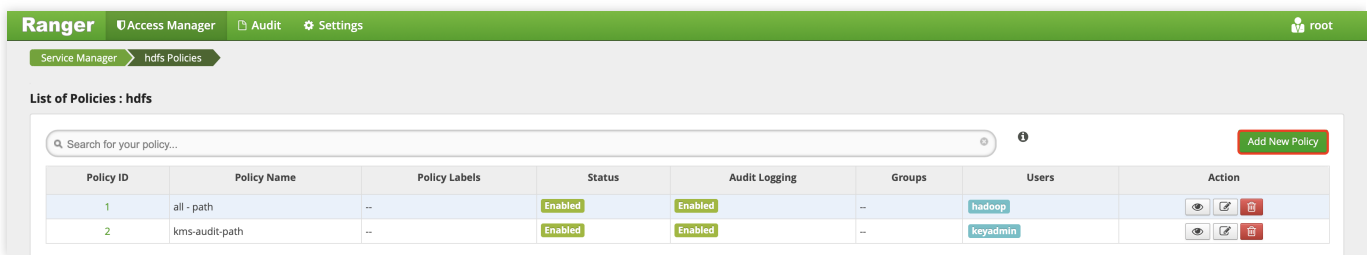
Parameter	Required	Description
Service Name	Yes	Service name, which is displayed on the main HDFS component on the Ranger Web UI
Description	No	Service description
Active Status	Default	Service status, which is Enabled by default
Username	Yes	Username of the resource
Password	Yes	User password
NameNode URL	Yes	HDFS URL
Authorization Enabled	Default	Select No for standard clusters and Yes for high-security clusters.
Authorization Type	Yes	Simple : standard cluster; Kerberos : high-security cluster

3. Configure EMR Ranger HDFS resource permissions.

- Click the configured EMR Ranger HDFS service.



- Configure a policy.



Service Manager > Hdfs Policies > Create Policy

Create Policy

Policy Details :

Policy Type: **Access** Add Validity Period

Policy Name *: user1_proxy enabled normal

Policy Label: Policy_Label

Resource Path *: /user recursive

Description: [Redacted]

Audit Logging: **YES**

Allow Conditions : hide

Select Group	Select User	Permissions	Delegate Admin	
/user	/user	Read Write	<input type="checkbox"/>	<input type="checkbox"/>

Exclude from Allow Conditions : hide

Select Group	Select User	Permissions	Delegate Admin	
Select Group	Select User	Add Permissions	<input type="checkbox"/>	<input type="checkbox"/>

4. After the policy is added, it will take effect in about 30 seconds, then you can use **user1** to read and write the **/user** of the HDFS file system.

Integrating YARN with Ranger

Last updated : 2021-06-07 17:25:20

Preparations

Ranger is available only when it is selected in **Optional Components** when you purchase a cluster. If you add the Ranger component after purchasing the cluster, the Web UI may be inaccessible. By default, when Ranger is installed, Ranger Admin and Ranger UserSync are deployed on the master node, and Ranger Plugin is deployed on the main daemon node of the embedded component.

When creating a cluster of the Hadoop type, you can select Ranger in **Optional Components**. The Ranger version varies depending on the EMR version you choose.

Note :

When the cluster type is Hadoop and the Ranger optional component is selected, EMR-Ranger will create services for HDFS and YARN by default and set default policies.

Cluster Type: **HADOOP** | DRUID | CLICKHOUSE | DORIS | KAFKA

Product Version: **EMR-V2.5.0** [Product Version Introduction](#)

Required Components: hadoop 2.8.5 | zookeeper 3.6.1 | knox 1.2.0

Optional Components:

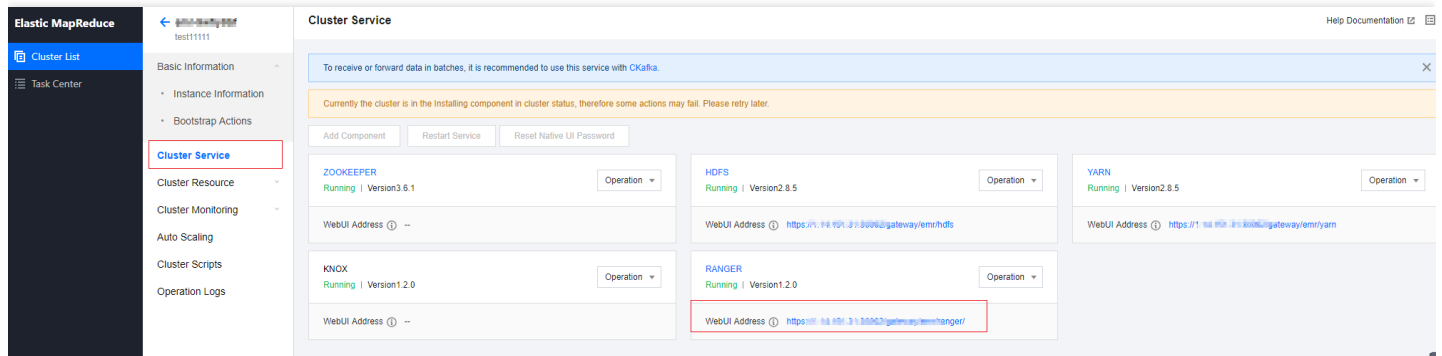
- hbase 1.4.9 | hive 2.3.7 | hue 4.6.0 | oozie 5.1.0 | **ranger 1.2.0** | spark_hadoop2.8 3.0.0 | sqoop 1.4.7 | tez 0.9.2
- flume 1.9.0 | impala 2.10.0 | alluxio 2.3.0 | flink 1.10.0 | storm 1.2.3 | kylin 2.5.2 | ganglia 3.7.2 | superset 0.35.2
- livy 0.7.0 | zeppelin 0.8.2 | tensorflowspark 1.4.4 | kudu 1.12.0 | prestosql 332

[Advanced Settings](#)

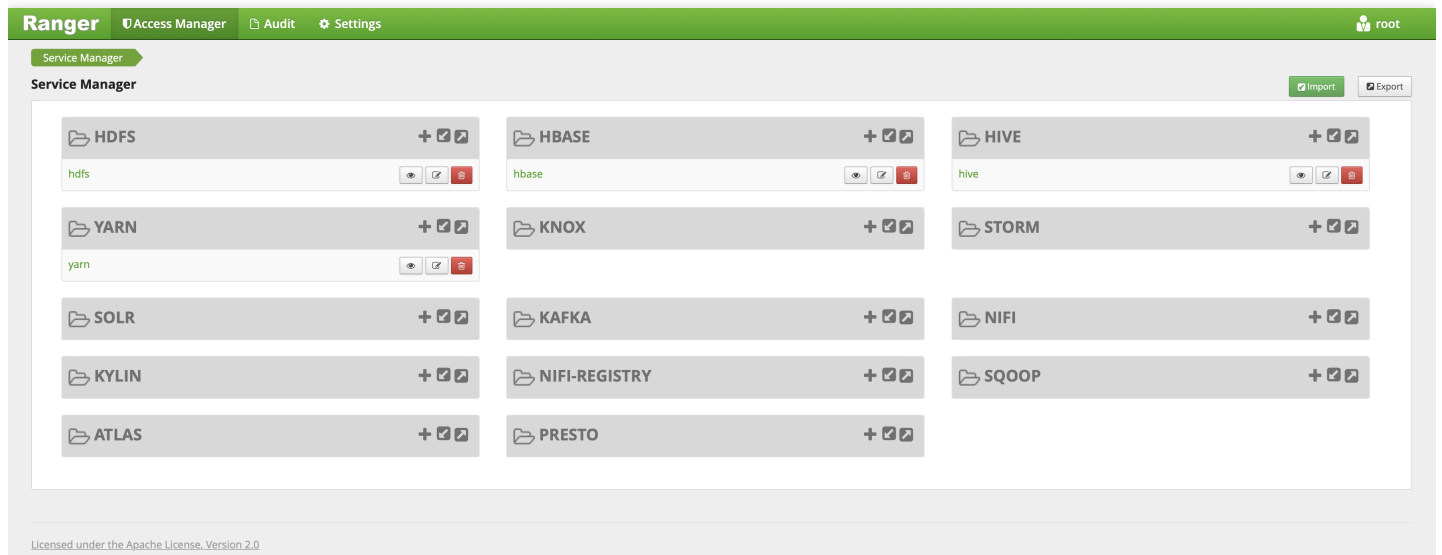
Next Step: Hardware Configuration

Ranger Web UI

Before accessing the Ranger Web UI, make sure that the current cluster is configured with a public IP and click the Ranger Web UI URL on the **Cluster Service** page.



After you are redirected, enter the username and password that you set when you purchased the cluster.



Integrating YARN with Ranger

Note :

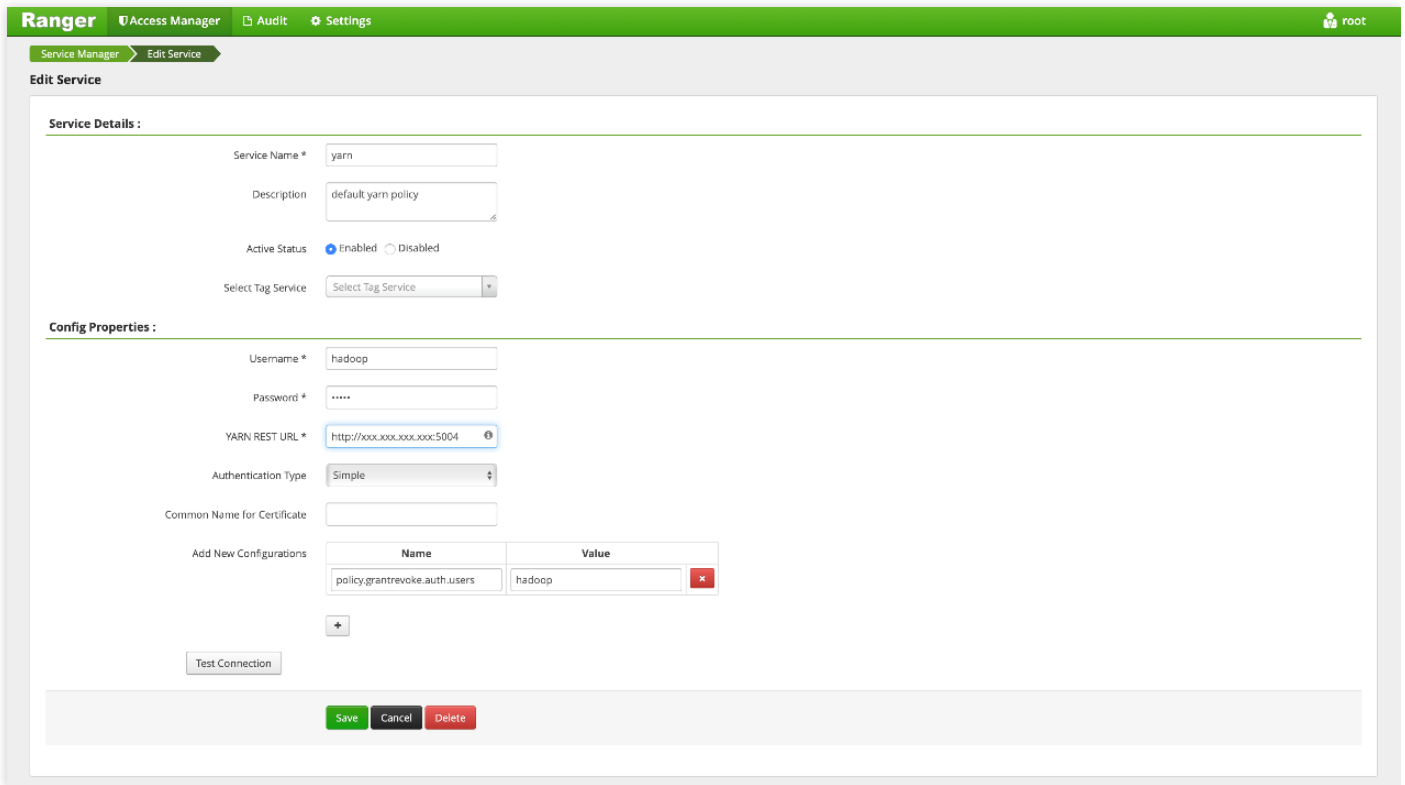
Make sure that YARN related services are running normally and Ranger has been installed in the current cluster.

Currently, EMR Ranger YARN only supports ACLs for Capacity Scheduler queues, not for Fair Scheduler queues. Ranger YARN's queue ACLs take effect together with YARN's built-in Capacity Scheduler configuration, but with lower priority. Ranger YARN permissions will be verified only when YARN's built-in Capacity Scheduler configuration denies verification. **You are advised to set ACLs via Ranger, instead of in the configuration files.**

1. Add an EMR Ranger YARN service on the EMR Ranger Web UI.



2. Configure EMR Ranger YARN service parameters.



Parameter	Required	Description
Service Name	Yes	Service name, which is displayed on the main YARN component on the Ranger Web UI
description	No	Service description
Active Status	Default	Service status, which is Enabled by default
Username	Yes	Username of the resource
Password	Yes	User password
NameNode URL	Yes	YARN URL

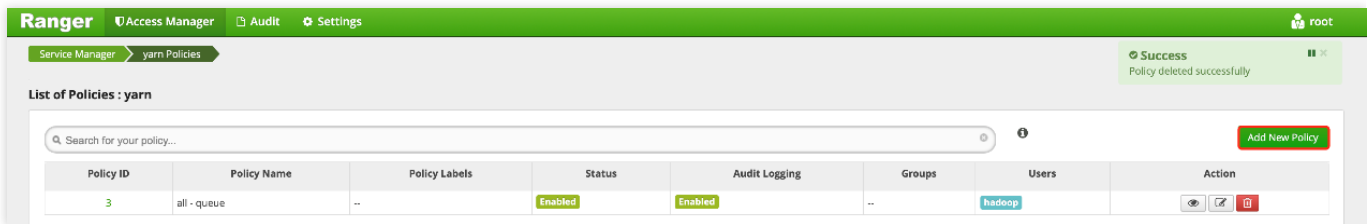
Parameter	Required	Description
Authorization Enabled	Default	Select No for standard clusters and Yes for high-security clusters.
Authorization Type	Yes	Simple : standard cluster; Kerberos : high-security cluster

3. Configure EMR Ranger YARN resource permissions.

- Click the configured EMR Ranger YARN service.



- Configure a policy.



Ranger Access Manager Audit Settings root

Service Manager > yarn Policies > Create Policy

Create Policy

Policy Details :

Policy Type: **Access** Add Validity Period

Policy Name *: user1_proxy enabled normal

Policy Label: Policy Label

Queue *: root.default recursive

Description: [Redacted]

Audit Logging: **YES**

Allow Conditions :

Select Group	Select User	Permissions	Delegate Admin	
user1	user1	admin-queue submit-app	<input type="checkbox"/>	<input type="checkbox"/>
Exclude from Allow Conditions :				
Select Group	Select User	Add Permissions	<input type="checkbox"/>	<input type="checkbox"/>

- After the policy is added, it will take effect in about 30 seconds, then you can use **user1** to submit, kill, or query jobs in the `root.default` queue of YARN.

Note :

When configuring Ranger YARN services and policies, make sure that there are no YARN jobs during this period; otherwise, issues about job submission permissions may occur.

Integrating HBase with Ranger

Last updated : 2021-06-07 17:25:20

Preparations

Ranger is available only when it is selected in **Optional Components** when you purchase a cluster. If you add the Ranger component after purchasing the cluster, the Web UI may be inaccessible. By default, when Ranger is installed, Ranger Admin and Ranger UserSync are deployed on the master node, and Ranger Plugin is deployed on the main daemon node of the embedded component.

When creating a cluster of the Hadoop type, you can select Ranger in **Optional Components**. The Ranger version varies depending on the EMR version you choose.

Note :

When the cluster type is Hadoop and the Ranger optional component is selected, EMR-Ranger will create services for HDFS and YARN by default and set default policies.

Cluster Type: **HADOOP** | DRUID | CLICKHOUSE | DORIS | KAFKA

Product Version: **EMR-V2.5.0** [Product Version Introduction](#)

Required Components: hadoop 2.8.5 | zookeeper 3.6.1 | knox 1.2.0

Optional Components:

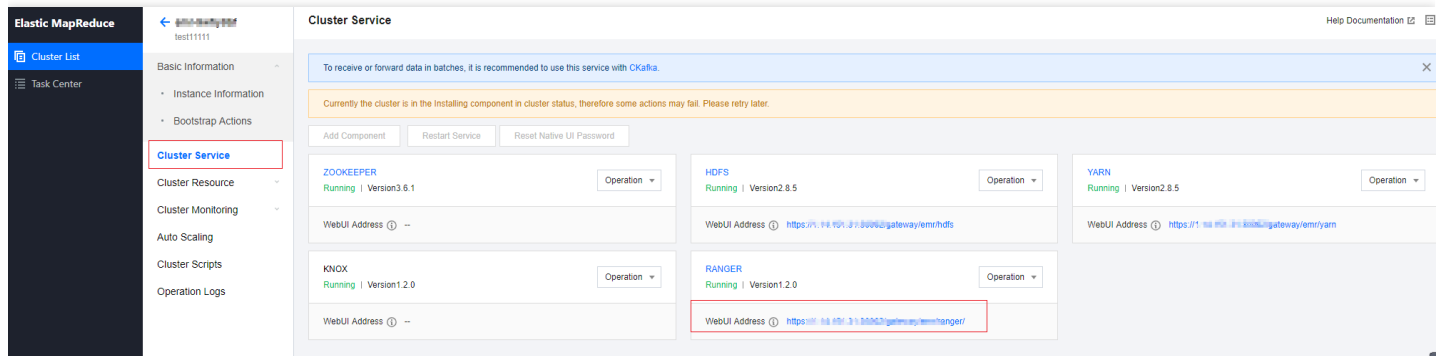
- hbase 1.4.9
- hive 2.3.7
- hue 4.6.0
- oozie 5.1.0
- ranger 1.2.0**
- spark_hadoop2.8 3.0.0
- sqoop 1.4.7
- tez 0.9.2
- flume 1.9.0
- impala 2.10.0
- alluxio 2.3.0
- flink 1.10.0
- storm 1.2.3
- kylin 2.5.2
- ganglia 3.7.2
- superset 0.35.2
- livy 0.7.0
- zeppelin 0.8.2
- tensorflowspark 1.4.4
- kudu 1.12.0
- prestosql 332

[Advanced Settings](#)

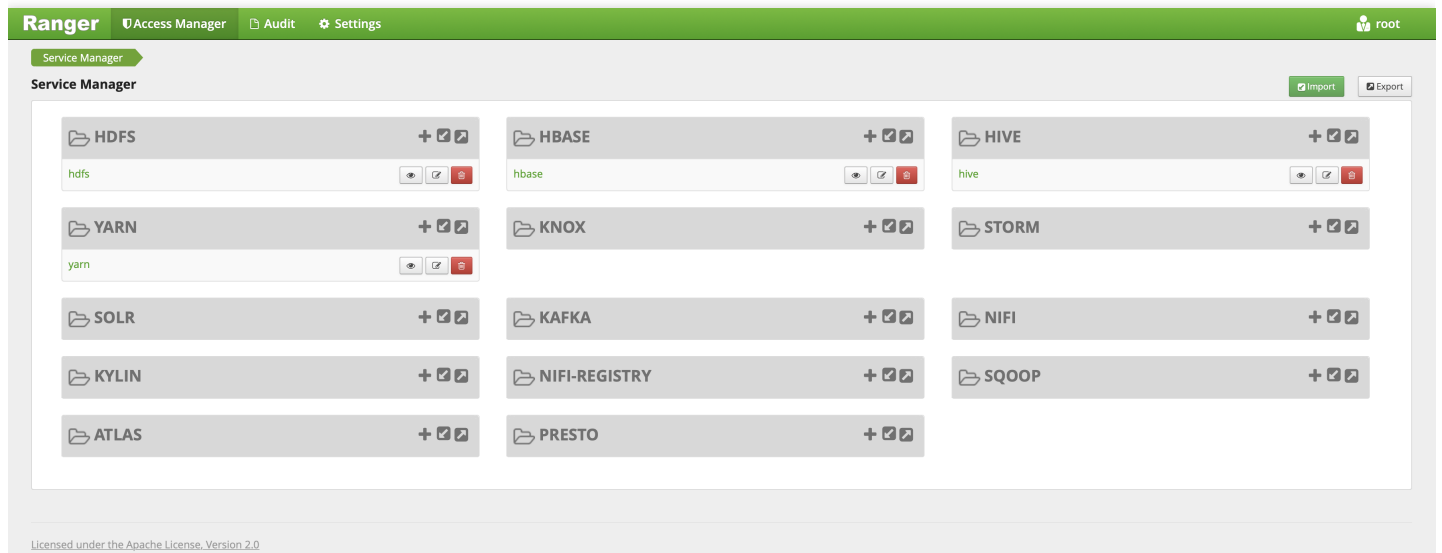
Next Step: Hardware Configuration

Ranger Web UI

Before accessing the Ranger Web UI, make sure that the current cluster is configured with a public IP and click the Ranger Web UI URL on the **Cluster Service** page.



After you are redirected, enter the username and password that you set when you purchased the cluster.

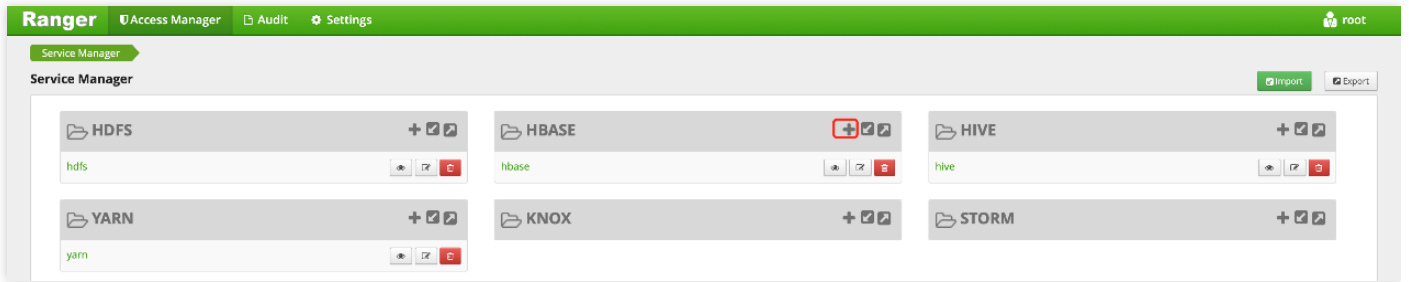


Integrating HBase with Ranger

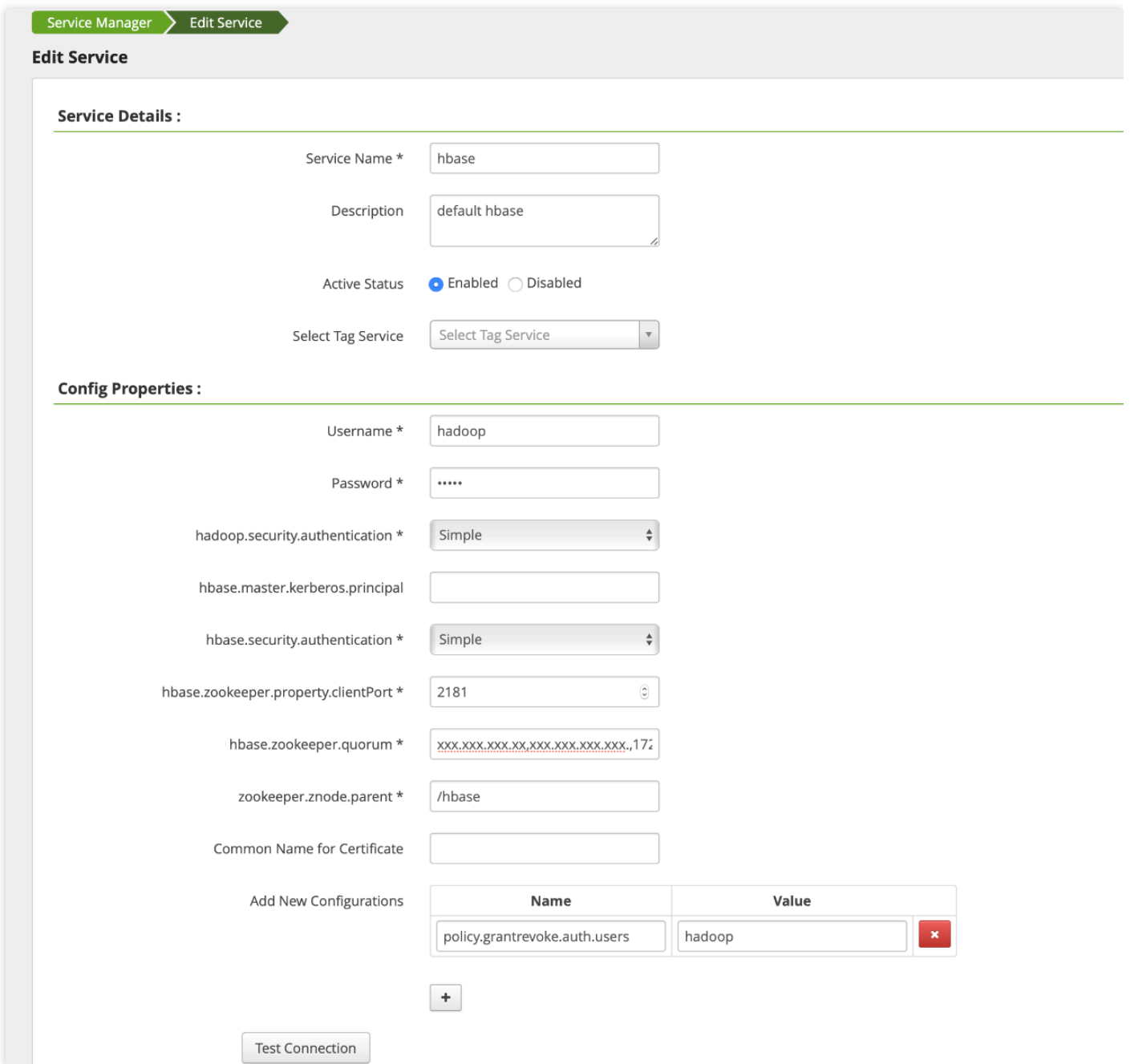
Note :

Make sure that HBase related services are running normally and Ranger has been installed in the current cluster.

1. Add an EMR Ranger HBase service on the EMR Ranger Web UI.



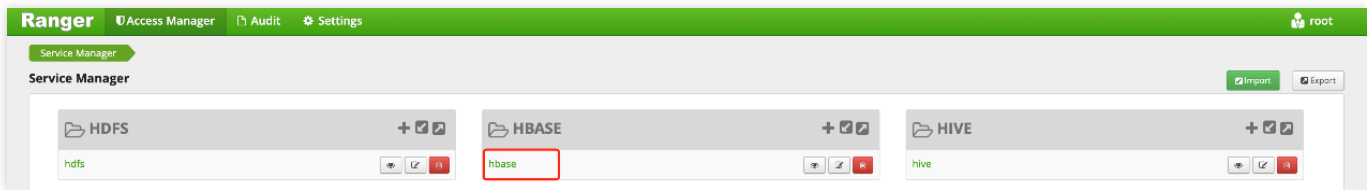
2. Configure EMR Ranger HBase service parameters.



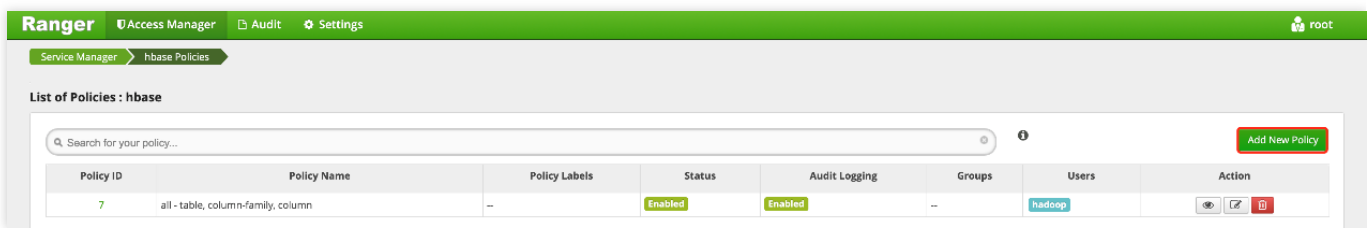
Parameter	Required	Description
Service Name	Yes	Service name, which is displayed on the main HBase component on the Ranger Web UI
Description	No	Service description
Active Status	Default	Service status, which is Enabled by default
Username	Yes	Username of the resource
Password	Yes	User password
Hbase.zookeeper.property.clientPort	Yes	Request port of the ZooKeeper client
Hbase.zookeeper.quorum	Yes	ZooKeeper cluster IP
Zookeeper.znode.parent	Yes	ZooKeeper node information

3. Configure EMR Ranger HBase resource permissions.

- Click the configured EMR Ranger HBase service.

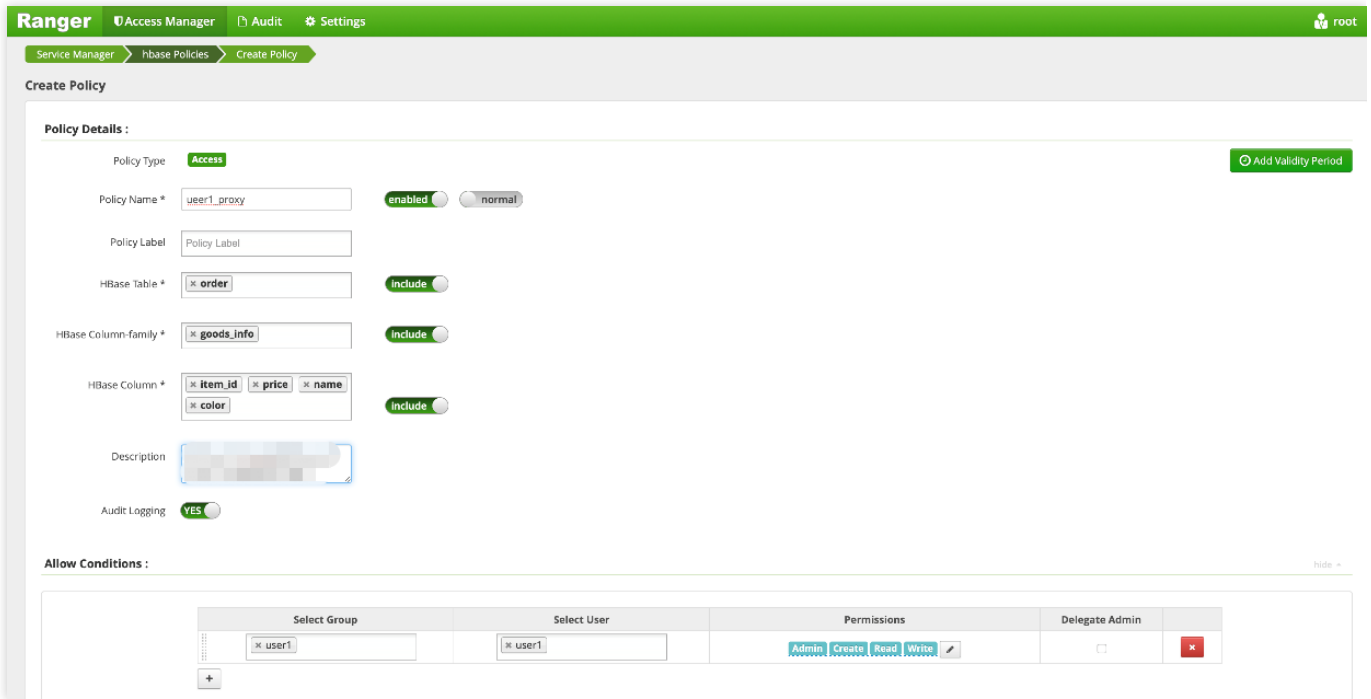


- Configure a policy.



In the above figure, the **Users** is **hadoop** and **Policy Name** is **all-table, column-family, column**, meaning HBase users have Region Balance, MemStore Flush, Compaction, and Split permissions. **Make sure that the**

created service has these permissions.



Parameter	Required	Description
HBase Table	Yes	HBase table name
HBase Column-family	Yes	Column family of the HBase table
HBase Column	Yes	Qualifiers of the column family

4. After the policy is added, it will take effect in about 30 seconds, then you can use **user1** to perform operations on the column family and qualifiers of the **order** table.

Integrating Presto with Ranger

Last updated : 2021-06-07 17:25:20

Preparations

Ranger is available only when it is selected in **Optional Components** when you purchase a cluster. If you add the Ranger component after purchasing the cluster, the Web UI may be inaccessible. By default, when Ranger is installed, Ranger Admin and Ranger UserSync are deployed on the master node, and Ranger Plugin is deployed on the main daemon node of the embedded component.

When creating a cluster of the Hadoop type, you can select Ranger in **Optional Components**. The Ranger version varies depending on the EMR version you choose.

Note :

When the cluster type is Hadoop and the Ranger optional component is selected, EMR-Ranger will create services for HDFS and YARN by default and set default policies.

Cluster Type: **HADOOP** | DRUID | CLICKHOUSE | DORIS | KAFKA

Product Version: **EMR-V2.5.0** [Product Version Introduction](#)

Required Components: hadoop 2.8.5 | zookeeper 3.6.1 | Knox 1.2.0

Optional Components:

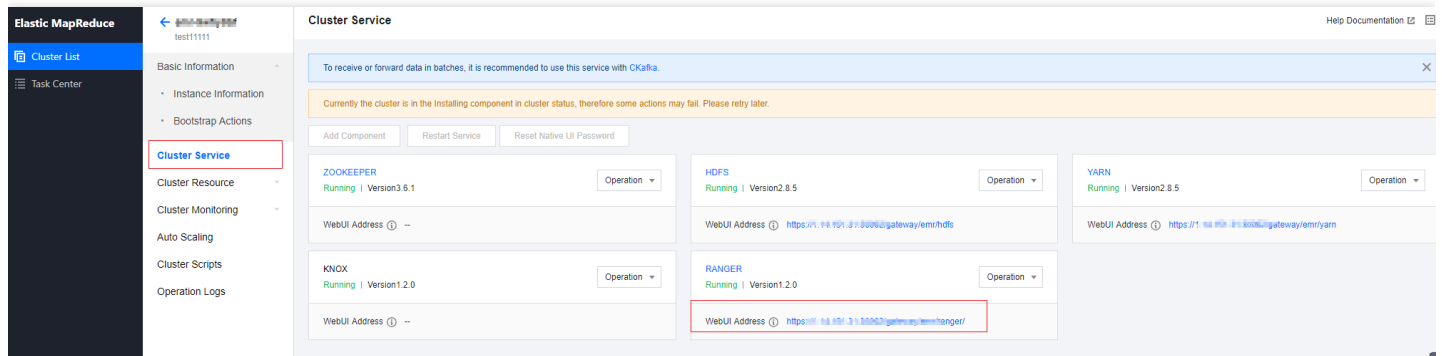
- hbase 1.4.9
- hive 2.3.7
- hue 4.6.0
- oozie 5.1.0
- ranger 1.2.0**
- spark_hadoop2.8 3.0.0
- sqoop 1.4.7
- tez 0.9.2
- flume 1.9.0
- impala 2.10.0
- alluxio 2.3.0
- flink 1.10.0
- storm 1.2.3
- kylin 2.5.2
- ganglia 3.7.2
- superset 0.35.2
- livy 0.7.0
- zeppelin 0.8.2
- tensorflowspark 1.4.4
- kudu 1.12.0
- prestosql 332

[Advanced Settings](#)

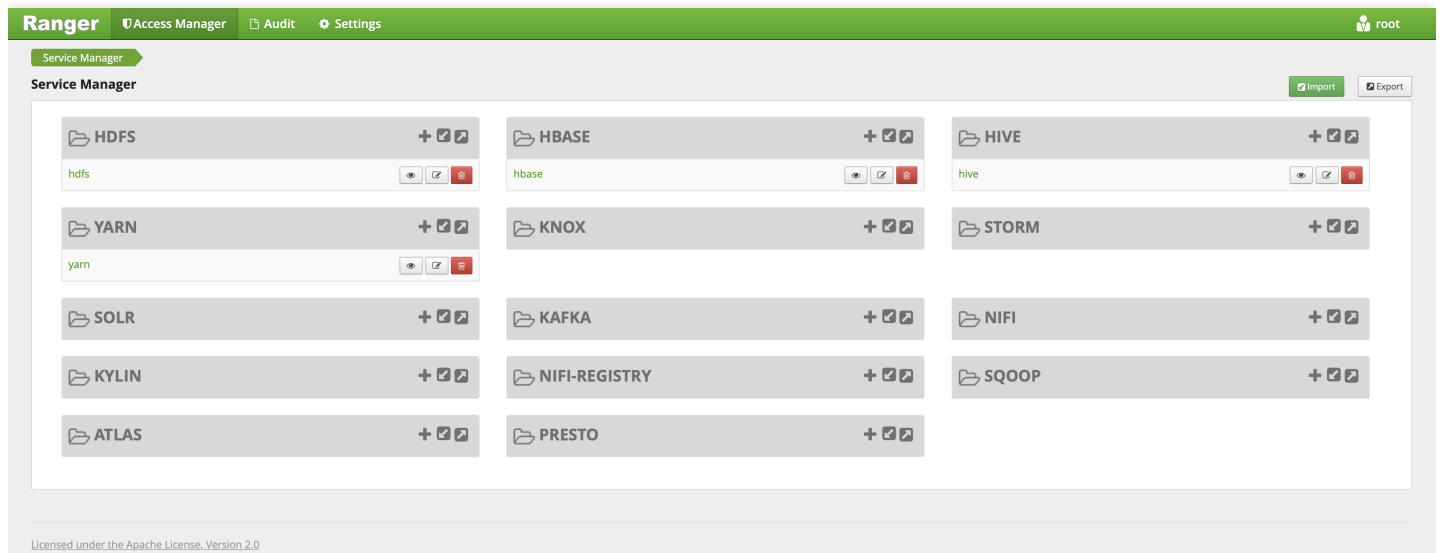
Next Step: Hardware Configuration

Ranger Web UI

Before accessing the Ranger Web UI, make sure that the current cluster is configured with a public IP and click the Ranger Web UI URL on the **Cluster Service** page.



After you are redirected, enter the username and password that you set when you purchased the cluster.

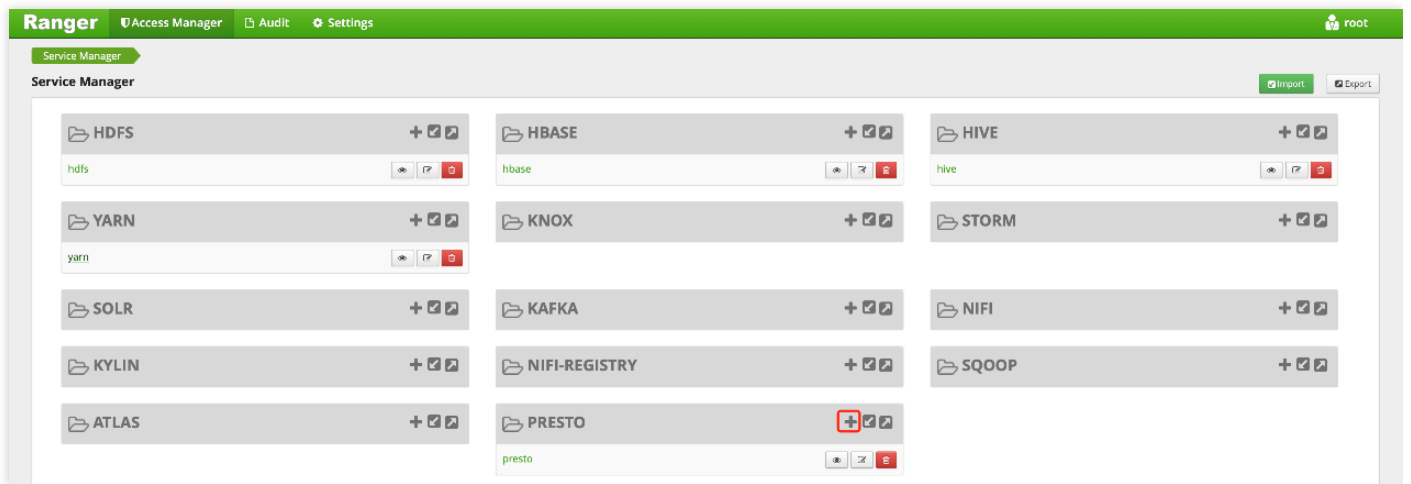


Integrating Presto with Ranger

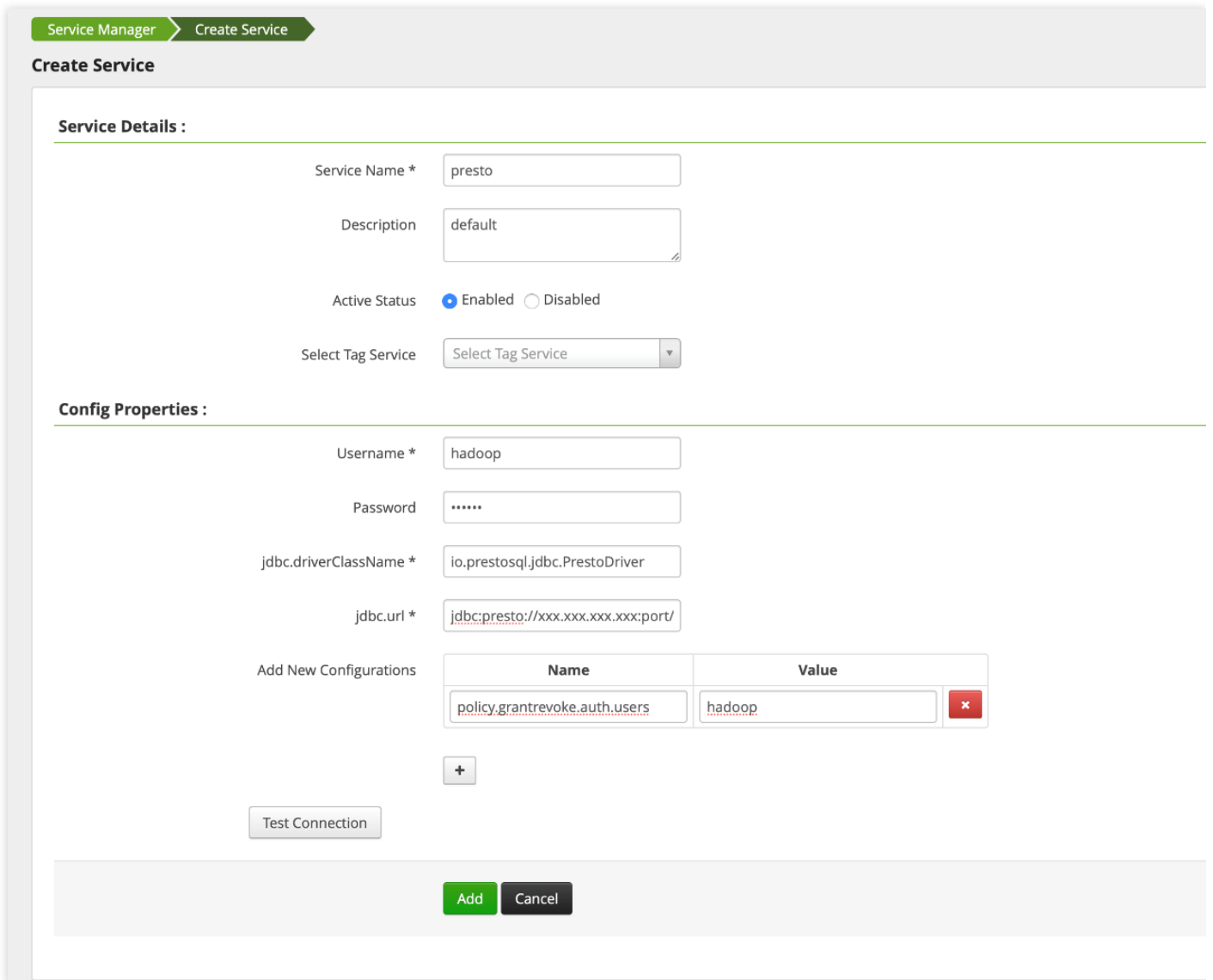
Note :

Make sure that Presto related services are running normally and Ranger has been installed in the current cluster.

1. Add an EMR Ranger Presto service on the EMR Ranger Web UI.



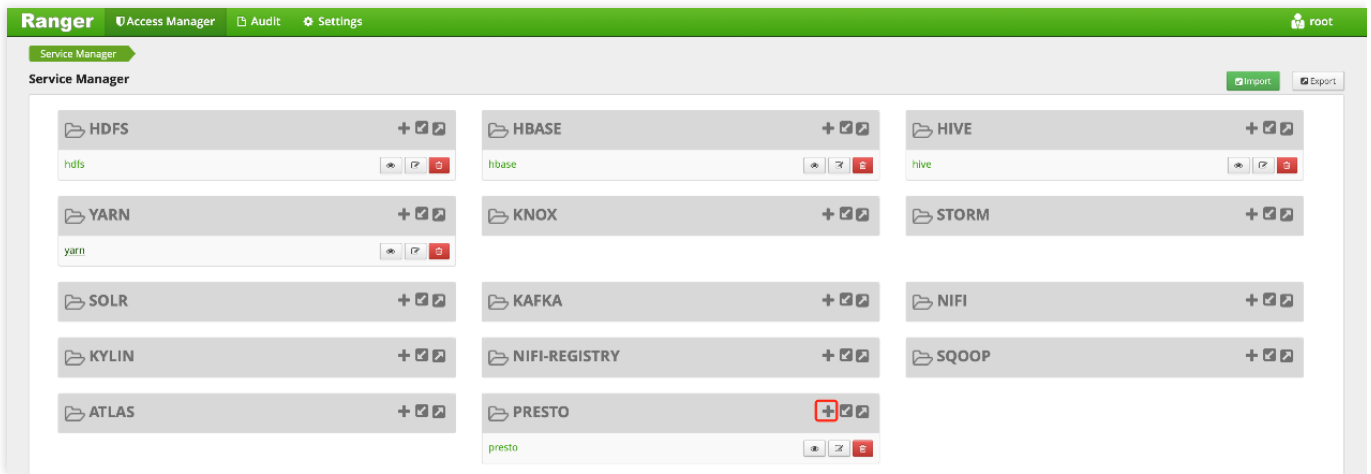
2. Configure EMR Ranger Presto service parameters.



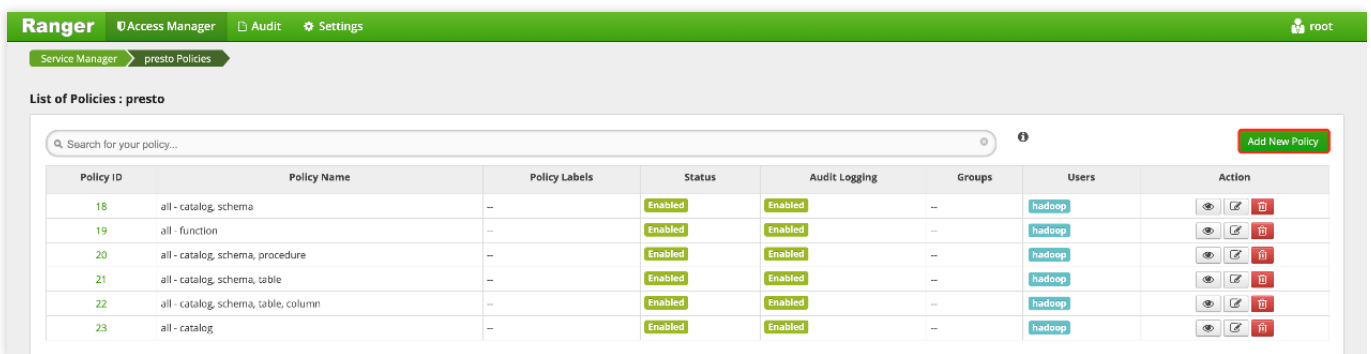
Parameter	Required	Description
Service Name	Yes	Service name, which is displayed on the main Presto component on the Ranger Web UI
Description	No	Service description
Active Status	Default	Service status, which is Enabled by default
Username	Yes	Username of the resource
Password	Yes	User password
JDBC.driverClassName	Yes	Full path of the drive class name
jdbc.url	Yes	Presto JDBC connection URL, for example, jdbc:presto://ip/hostname:port

3. Configure EMR Ranger Presto resource permissions.

- Click the configured EMR Ranger Presto service.



- Configure a policy.



Service Manager > presto Policies > Create Policy

Create Policy

Policy Details :

Policy Type: **Access** Add Validity Period

Policy Name *: user1.proxy enabled normal

Policy Label: Policy Label

catalog: none hive include

Description: [Redacted]

Audit Logging: **YES**

Allow Conditions :

Select Group	Select User	Permissions	Delegate Admin	
user1	user1	Select Use	<input type="checkbox"/>	<input type="checkbox"/>

4. After the policy is added, it will take effect in about 30 seconds, then you can use **user1** to view and use the catalog of Presto.

Doris Development Guide

Doris Overview

Last updated : 2021-06-07 17:25:20

Tencent Cloud EMR-Doris provides the cloud semi-hosting service of Doris, which is an MPP analytical database product. It features convenient Doris cluster deployment, configuration modification, monitoring and alarming, etc. Doris is compatible with the MySQL protocol, uses the standard SQL language, and supports highly concurrent queries on PB-scale data, making it suitable for offline data analysis, real-time data analysis, interactive data analysis, exploratory data analysis, and more.

Features

MySQL protocol compatibility

Doris provides a connection API compatible with the MySQL protocol. Users can directly use MySQL libraries or tools without the need to deploy new client libraries or tools separately. It also provides MySQL APIs for compatibility with upper-layer applications. All this reduces the learning costs and makes it easy to get started.

High throughput

The MPP architecture allows the concurrent execution of queries on multiple nodes in a distributed manner, making full use of the overall compute resources of clusters and improving the throughput for large queries.

High concurrence

Leveraging technologies such as partition pruning, pre-aggregation, predicate pushdown, vectorized execution, and asynchronous RPC, Doris supports highly concurrent query scenarios.

Data update

Doris supports deleting and updating data by the primary key, making it easy to synchronize real-time data from transactional databases such as MySQL.

High availability and reliability

Doris use three-replica storage for both data and metadata by default (three or more replicas for BE nodes). This ensures data reliability even when a small number of nodes are down. Doris will automatically identify and repair corrupted data, and route query requests to healthy nodes, ensuring data availability on a 24/7 basis.

Horizontal scaling and data balancing

Both frontend (FE) nodes and backend (BE) nodes can be scaled horizontally. You can flexibly increase nodes based on compute and storage needs. After BE nodes are added, Doris will automatically balance the data partitions according to the load of the nodes without manual intervention.

Materialized views and pre-aggregation engine

Doris supports storing the results of data pre-aggregation and compute in the form of materialized views or rollups, thereby improving the query efficiency of some aggregation scenarios. Doris also guarantees the data consistency between materialized views and underlying tables, making queries and imports via materialized views completely transparent. Doris automatically selects the appropriate materialized views for data ingestion based on your query statements.

Efficient column-oriented storage engine

Doris uses proprietary column-oriented storage formats to improve the query efficiency in the OLAP field. Coupled with the characteristics of column-oriented storage, it adopts various encoding methods such as dictionary encoding and RLE for storage to provide a high data compression ratio and help save storage space. Meanwhile, it uses technologies such as smart min/max index, sparse index, Bloom filter, and bitmap inverted index to improve query efficiency.

Online table structure modification

You can modify the table structure after data is imported, including adding columns, deleting columns, modifying column types, and changing the column order. These operations will not affect the current database queries and writes.

Scenarios

Big data analysis and statistics

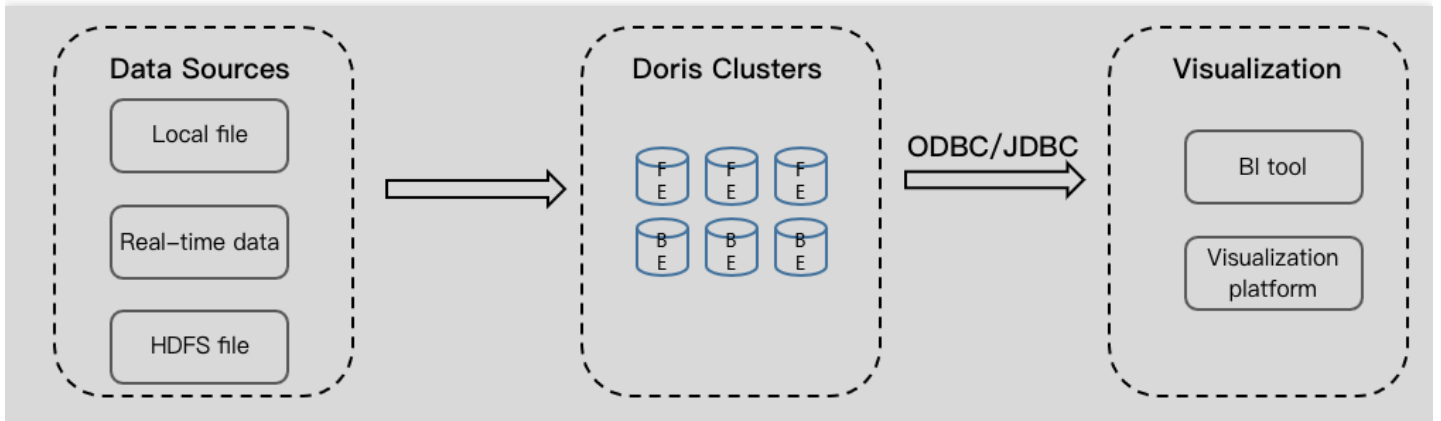
There are mainly two types of data analysis scenarios: report analysis and multi-dimensional analysis.

Report analysis

For report analysis scenarios, data analysis and query modes are relatively fixed, and the backend SQL mode is definite. In such cases, you are advised to use MySQL to store result data. You can choose to execute batch processing and send emails. In the Doris platform, the latency of a report query is generally within one second.

Multi-dimensional analysis

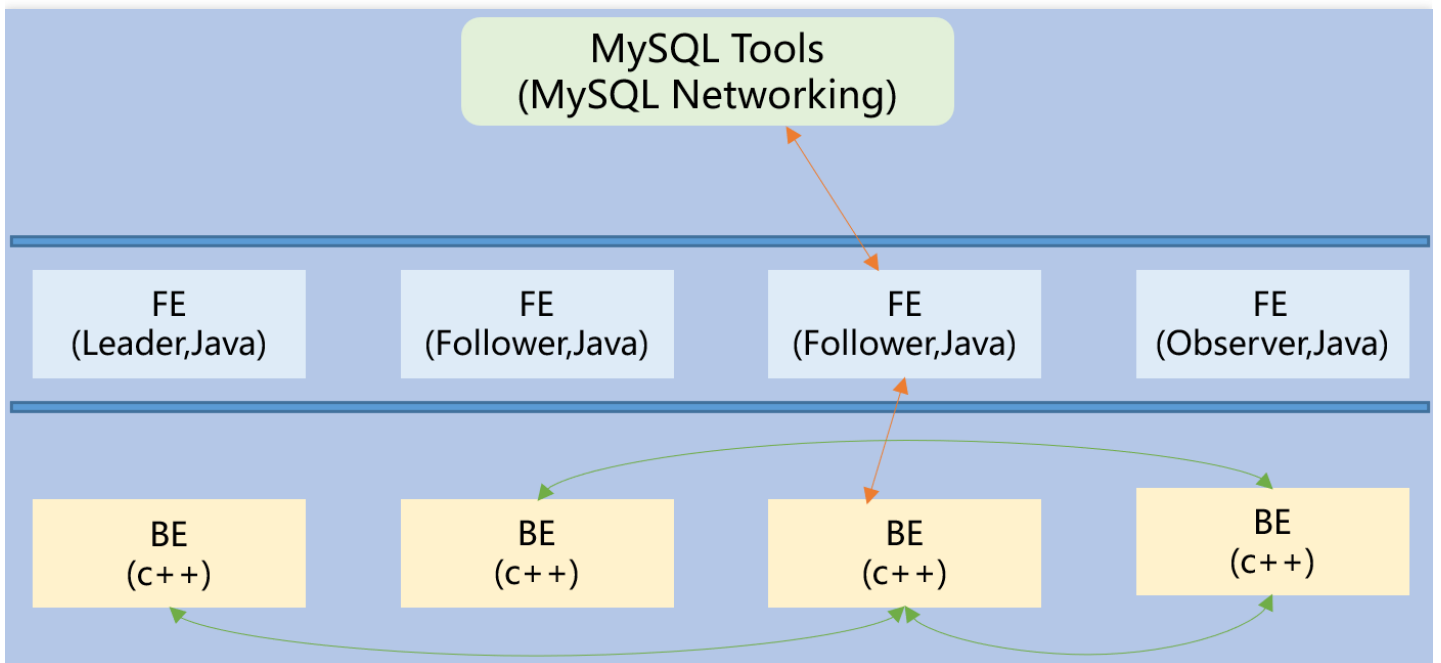
Multi-dimensional analysis requires data to be structured and is suitable for scenarios where queries are relatively flexible, for example, data analysis criteria and aggregation dimensions are not definite.



Structure

Doris has three major components:

- FE refers to the frontend nodes of Doris, which are mainly responsible for receiving client requests and returning results, managing metadata and clusters, and generating query plans.
- BE refers to the backend nodes of Doris, which are mainly responsible for storing and managing data, and executing query plans.
- Broker is an optional process in a Doris cluster, which are mainly used to read and write files and directories on remote storage systems or services, such as HDFS and Tencent Cloud COS.



User Guide

Creating a User

Last updated : 2021-06-01 17:27:01

Doris uses the MySQL protocol to communicate. You can connect to a Doris cluster through a MySQL client or MySQL JDBC. When selecting the MySQL client version, you are advised to use a version after 5.1, because usernames of more than 16 characters are not supported before 5.1. This document takes MySQL client as an example to show you the basic usage of Doris through a complete process.

Note :

Change the Knox password only when resetting the native UI password. Doris' UI authentication password is the same as the cluster password.

Root User Login and Password Modification

Doris has built-in root and admin users, and the password is the same as the cluster password. After starting the Doris program, you can connect to the Doris cluster through the root or admin user. Use the following command to log in to Doris:

```
mysql -h FE_HOST -P9030 -uroot
```

Parameter description:

- `FE_HOST` is the IP address of any FE node.
- `9030` is the `query_port` configuration in `fe.conf`.

After login, you can modify the root password with the following command:

```
SET PASSWORD FOR 'root' = PASSWORD('your_password');
```

Creating a User

Create an ordinary user with the following command:

```
CREATE USER 'test' IDENTIFIED BY 'test_passwd';
```

Subsequent login can be done with the following connection command:

```
mysql -h FE_HOST -P9030 -utest -ptest_passwd
```

By default, the newly created ordinary user does not have any permissions. To grant permissions, see [Account Authorization](#).

Creating a Data Table and Importing Data

Last updated : 2022-07-08 11:51:12

Creating a Database

Initially, you can create a database through the root or admin account with the following command:

```
CREATE DATABASE example_db;
```

All commands can use `HELP command` to see detailed syntax help. For example: `HELP CREATE DATABASE;` .

If you don't know the full name of the command, you can use "help + a field" for fuzzy query. For example, if you type

`HELP CREATE` , you can get commands like `CREATE DATABASE` , `CREATE TABLE` , and `CREATE USER` .

After the database is created, you can view the database information through `SHOW DATABASES;` .

```
MySQL> SHOW DATABASES;
+-----+
| Database |
+-----+
| example_db |
| information_schema |
+-----+
2 rows in set (0.00 sec)
```

`information_schema` exists to be compatible with the MySQL protocol. In practice, information may not be accurate. Therefore, you are advised to obtain the information about specific databases by directly querying the corresponding databases.

Account Authorization

After the `example_db` database is created, you can authorize the read/write permissions for `example_db` to an ordinary account, such as `test`, through the root or admin account. After authorization, you can log in to and operate `example_db` using the `test` account.

```
GRANT ALL ON example_db TO test;
```

Creating a table

Create a table with the `CREATE TABLE` command. More parameter information can be seen by running the `HELP CREATE TABLE;` command.

Switch the database using the following command:

```
USE example_db;
```

Doris supports two ways to create a table: single partitioning and composite partitioning.

In composite partitioning:

- The first level is called Partition, or partitioning. Users can specify a dimension column as a partition column (currently only integer and time columns are supported), and specify the value range for each partition.
- The second level is called Distribution, or bucketing. Users can specify one or more dimension columns and the number of buckets for HASH distribution of data.

Composite partitioning is recommended for the following scenarios:

- There are time dimensions or similar dimensions with ordered values, which can be used as partition columns. The partition granularity can be evaluated according to the frequency of import and the amount of partition data.
- Historical data deletion requirements: for example, only to retain the data of the last N days. Using composite partitioning, you can achieve this by deleting historical partitions. You can also delete data by sending a `DELETE` statement within a specified partition.
- Solving the data skew issue: you can specify the number of buckets for each partition separately. For partitioning by day, when the amount of data varies greatly every day, you can divide the data of different partitions by specifying the number of buckets in the partitions. You are advised to choose columns with high differentiation as the bucket columns.
- You can use single partitioning instead of composite partitioning. Then the data are only distributed by HASH.

The following takes the aggregation model as an example to separately illustrate the table creation statements of the two kinds of partitioning.

Single partitioning

Create a logical table named `table1`. The bucket column is `siteid` and the number of buckets is 10. The schema of this table is as follows:

- `siteid`: the type is `INT` (4 bytes); the default value is `10`.
- `citycode`: the type is `SMALLINT` (2 bytes).
- `username`: the type is `VARCHAR`; the maximum length is `32`; the default value is an empty string.
- `pv`: the type is `BIGINT` (8 bytes); the default value is `0`. This is a metric column. Doris will aggregate the metric column internally. The aggregation method of this column is `SUM`.

The table creation statement is as follows:

```
CREATE TABLE table1
(
  siteid INT DEFAULT '10',
  citycode SMALLINT,
  username VARCHAR(32) DEFAULT '',
  pv BIGINT SUM DEFAULT '0'
)
AGGREGATE KEY(siteid, citycode, username)
DISTRIBUTED BY HASH(siteid) BUCKETS 10
PROPERTIES("replication_num" = "1");
```

Composite partitioning

Create a logical table named `table2`. The schema of this table is as follows:

- `event_day`: the type is `DATE`; no default value.
- `siteid`: the type is `INT` (4 bytes); the default value is `10`.
- `citycode`: the type is `SMALLINT` (2 bytes).
- `username`: the type is `VARCHAR`; the maximum length is `32`; the default value is an empty string.
- `pv`: the type is `BIGINT` (8 bytes); the default value is `0`. This is a metric column. Doris will aggregate the metric column internally. The aggregation method of this column is `SUM`.

The `event_day` column is used as the partition column to create three partitions: `p201706`, `p201707`, and `p201708`.

- `p201706`: the range is [Minimum, 2017-07-01).
- `p201707`: the range is [2017-07-01, 2017-08-01).
- `p201708`: the range is [2017-08-01, 2017-09-01).

Note :

Note that the interval is left-closed and right-open.

Each partition uses `siteid` to hash buckets, with a bucket count of 10. The table creation statement is as follows:

```
CREATE TABLE table2
(
  event_day DATE,
  siteid INT DEFAULT '10',
```

```

citycode SMALLINT,
username VARCHAR(32) DEFAULT '',
pv BIGINT SUM DEFAULT '0'
)
AGGREGATE KEY(event_day, siteid, citycode, username)
PARTITION BY RANGE(event_day)
(
PARTITION p201706 VALUES LESS THAN ('2017-07-01'),
PARTITION p201707 VALUES LESS THAN ('2017-08-01'),
PARTITION p201708 VALUES LESS THAN ('2017-09-01')
)
DISTRIBUTED BY HASH(siteid) BUCKETS 10
PROPERTIES("replication_num" = "1");

```

After the table is created, you can view the information of the table in `example_db` :

```
MySQL> SHOW TABLES;
```

```

+-----+
| Tables_in_example_db |
+-----+
| table1 |
| table2 |
+-----+
2 rows in set (0.01 sec)

```

```
MySQL> DESC table1;
```

```

+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| siteid | int(11) | Yes | true | 10 | |
| citycode | smallint(6) | Yes | true | N/A | |
| username | varchar(32) | Yes | true | | |
| pv | bigint(20) | Yes | false | 0 | SUM |
+-----+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)

```

```
MySQL> DESC table2;
```

```

+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| event_day | date | Yes | true | N/A | |
| siteid | int(11) | Yes | true | 10 | |
| citycode | smallint(6) | Yes | true | N/A | |
| username | varchar(32) | Yes | true | | |
| pv | bigint(20) | Yes | false | 0 | SUM |
+-----+-----+-----+-----+-----+-----+
5 rows in set (0.00 sec)

```

Notes

Note :

For more syntax description regarding the use of Doris, see [Data Table Creation and Data Import](#).

1. The above tables created by setting `replication_num` are all single-replica tables. Doris recommends that users adopt the default three-replica settings to ensure high availability.
2. Partitions in composite partitioning tables can be added or deleted dynamically.
3. Data can be imported by importing a specified partition.
4. The schema of table can be dynamically modified.
5. Rollup can be added to a table to improve the query performance.
6. The default value of the `Null` attribute for column is `true` , which may affect the query performance.

Importing Data

Doris supports a variety of data import methods. The following uses streaming import and broker import as examples.

Streaming import

Streaming import transfers data to Doris over the HTTP protocol. It can import local data directly without relying on other systems or components.

Example 1

With `table1_20170707` as the label, import the `table1` table using the `table1_data` local file.

```
curl --location-trusted -u test:test -H "label:table1_20170707" -H "column_separator:," -T table1_data http://FE_HOST:8030/api/example_db/table1/_stream_load
```

1. `FE_HOST` is the IP of any FE node and `8030` is the `http_port` in `fe.conf` .
2. You can use the IP of any BE and the `webserver_port` in `be.conf` for import. For example:

```
BE_HOST:8040 .
```

The `table1_data` local file uses a comma (,) as the separator between data. The specific content is as follows:

```
1,1,jim,2
2,1,grace,2
3,2,tom,2
```

```
4,3,bush,3
5,3,helen,3
```

Example 2

With `table2_20170707` as the label, import the `table2` table using the `table2_data` local file.

```
curl --location-trusted -u test:test -H "label:table2_20170707" -H "column_separator:|" -T table2_data http://127.0.0.1:8030/api/example_db/table2/_stream_load
```

The `table2_data` local file uses a vertical bar (|) as the separator between data. The specific content is as follows:

```
2017-07-03|1|1|jim|2
2017-07-05|2|1|grace|2
2017-07-12|3|2|tom|2
2017-07-15|4|3|bush|3
2017-07-12|5|3|helen|3
```

Precautions

1. The recommended file size for streaming imports is no greater than 10 GB. Excessive file size will result in failure and increase the costs of retry.
2. Each batch of imported data needs to take a label. The label is preferably a string related to the batch of data for easy reading and management. Doris guarantees that the same batch of data can be imported only once in a database based on label. Labels for failed tasks can be reused.
3. Streaming imports are synchronous commands. The successful return of the command indicates that the data has been imported, and the failed return indicates that the data has not been imported.

Broker import

Broker imports import data from external storage through deployed broker processes.

With `table1_20170708` as the label, import files on HDFS into the `table1` table.

```
LOAD LABEL table1_20170708
(
  DATA INFILE("hdfs://your.namenode.host:port/dir/table1_data")
  INTO TABLE table1
)
WITH BROKER hdfs
(
  "username"="hdfs_user",
  "password"="hdfs_password"
```

```
)  
PROPERTIES  
(  
  "timeout"="3600",  
  "max_filter_ratio"="0.1"  
)  
);
```

Broker imports are asynchronous commands. Successful execution of the above commands only indicates successful submission of tasks. Successful imports need to be checked through `SHOW LOAD;`. The command is as follows:

```
SHOW LOAD WHERE LABEL = "table1_20170708";
```

In the return result, `FINISHED` in the `State` field indicates that the import was successful.

Asynchronous import tasks can be canceled before the end by using the following command:

```
CANCEL LOAD WHERE LABEL = "table1_20170708";
```

Querying Data

Last updated : 2021-06-01 17:48:35

Simple Query

```
MySQL> SELECT * FROM table1 LIMIT 3;
+-----+-----+-----+-----+
| siteid | citycode | username | pv |
+-----+-----+-----+-----+
| 2 | 1 | 'grace' | 2 |
| 5 | 3 | 'helen' | 3 |
| 3 | 2 | 'tom' | 2 |
+-----+-----+-----+-----+
3 rows in set (0.01 sec)
MySQL> SELECT * FROM table1 ORDER BY citycode;
+-----+-----+-----+-----+
| siteid | citycode | username | pv |
+-----+-----+-----+-----+
| 2 | 1 | 'grace' | 2 |
| 1 | 1 | 'jim' | 2 |
| 3 | 2 | 'tom' | 2 |
| 4 | 3 | 'bush' | 3 |
| 5 | 3 | 'helen' | 3 |
+-----+-----+-----+-----+
5 rows in set (0.01 sec)
```

Join Query

```
MySQL> SELECT SUM(table1.pv) FROM table1 JOIN table2 WHERE table1.siteid = table
2.siteid;
+-----+
| sum(`table1`.`pv`) |
+-----+
| 12 |
+-----+
1 row in set (0.20 sec)
```


Subquery

```
MySQL> SELECT SUM(pv) FROM table2 WHERE siteid IN (SELECT siteid FROM table1 WHERE siteid > 2);
+-----+
| sum(`pv`) |
+-----+
| 8 |
+-----+
1 row in set (0.13 sec)
```

Kafka Development Guide

Kafka Overview

Last updated : 2021-06-07 17:25:21

Tencent Cloud EMR-Kafka offers the cloud hosting service of open-source Kafka, with convenient Kafka cluster deployment, configuration modification, monitoring and alarming, and other features, providing enterprises and users with safe, stable OLAP solutions. Kafka data pipelines have been the most commonly used data sources and data sinks in stream computing systems. You can import streaming data into a certain topic in Kafka, process it through Flink operators, and output it to another topic in the same or a different Kafka instance. Kafka supports reading/writing data from/to multiple partitions in the same topic, which increases throughput and reduces data skew and hotspots.

Architecture

Both single-node and multi-node architectures are available for your choice based on business needs.

OPS

The console provides out-of-the-box services such as monitoring, log search, and parameter adjustment.

Features

- **Sending-receiving decoupling:** the relationship between producers and consumers is effectively decoupled. Under the premise that the same API constraint is ensured, the processing between producers and consumers can be independently expanded or modified.
- **Flexibility:** Kafka clusters are able to withstand sudden increases in requests without breakdown, effectively improving the robustness of the system.
- **Orderly reading and writing:** Kafka clusters can guarantee the order of messages in a partition. Just like most message queue services, they can also ensure that data is processed in order, greatly improving disk efficiency.
- **Asynchronous communication:** in scenarios where the business does not need to process messages immediately, Kafka clusters provide the asynchronous message processing mechanism. When the traffic is heavy, messages are put into the queue only, and they will be processed after the traffic become lighter, which relieves the system pressure.

Strengths

100% compatibility with open-source Kafka and easy migration

- Kafka clusters are compatible with open-source Kafka v1.1.1.
- The business system of Kafka clusters is based on the existing code of the open-source Apache Kafka ecosystem. Without any changes to your existing project, you can migrate to the cloud and enjoy the high-performance Kafka services provided by Tencent Cloud.

High performance

- Tencent Cloud has improved the service performance, eliminating the need for complicated parameter configuration.
- You can upgrade or downgrade configurations on the UI and enjoy high-performance IaaS layer support.

High availability

- Leveraging Tencent's years of experience in monitoring technologies, EMR offers comprehensive monitoring on clusters and has a professional OPS team in place that responds to alarms on a 24/7 basis to ensure the high availability of Kafka clusters.
- Custom multi-AZ deployment in the same region is supported to improve disaster recovery ability.

High reliability

- The disks are highly reliable, making it possible to keep services running even if 50% of the disks become faulty.
- Two replicas are created by default, and up to three replicas can be used. The more replicas, the higher the reliability.

Use Cases

Last updated : 2022-01-05 11:00:14

Webpage Behavior Analysis

Kafka clusters process website activities (PV, search, etc.) in real time and publish them to topics by type. These information flows can be used for real-time monitoring or offline statistical analysis.

A large amount of activity information is generated in each user's PV, therefore, website activity tracking requires high throughput. Kafka clusters perfectly meet the requirements of high throughput and offline processing.

Log Aggregation

Kafka clusters feature low-latency processing, supporting multiple data sources and distributed data processing (consumption). Compared with centralized log aggregation systems, Kafka provides better persistence and lower end-to-end latency while delivering the same performance.

The above features make Kafka clusters an ideal log collection center. Multiple servers/applications can asynchronously send operation logs in batches to Kafka clusters instead of saving them locally or in a DB. Kafka clusters can submit/compress messages in batches, and producers can hardly perceive the performance overhead. Consumers can use systematic storage and analysis systems such as Hadoop to analyze the pulled logs.

Online/Offline Analysis

In some big data scenarios, a large amount of concurrent data needs to be processed and aggregated. This requires clusters to have excellent processing performance and high scalability. Moreover, Kafka clusters' data distribution mechanism, in terms of disk space allocation, message format processing, server selection, and data compression, also makes them suitable for handling high numbers of real-time messages and aggregating distributed application data, which facilitates system OPS.

Kafka clusters can better aggregate, process, and analyze offline and streaming data.

Kafka Usage

Last updated : 2021-06-07 17:25:21

Generating Data

By Java code

```
@Component
@Slf4j
public class KafkaProducer {
    @Autowired
    private KafkaTemplate<String, Object> kafkaTemplate;
    // Custom topics.
    public static final String TOPIC_TEST = "topic.test";
    //
    public static final String TOPIC_GROUP1 = "topic.group1";
    //
    public static final String TOPIC_GROUP2 = "topic.group2";
    public void send(Object obj) {
        String obj2String = JSONObject.toJSONString(obj);
        log.info("the message to send: {}", obj2String);
        // Send a message.
        ListenableFuture<SendResult<String, Object>> future = kafkaTemplate.send(TOPIC_TEST, obj);
        future.addCallback(new ListenableFutureCallback<SendResult<String, Object>>() {
            @Override
            public void onFailure(Throwable throwable) {
                // Returned result for failed sending
                log.info(TOPIC_TEST + " - the producer failed to send the message:" + throwable.getMessage());
            }
            @Override
            public void onSuccess(SendResult<String, Object> stringObjectSendResult) {
                // Returned result for successful sending
                log.info(TOPIC_TEST + " - the producer sent the message successfully:" + stringObjectSendResult.toString());
            }
        });
    }
}
```

By command

```
bin/kafka-console-producer.sh --broker-list node86:9092 --topic t_cdr
```

Consuming Data

By Java code

```
@Component
@Slf4j
public class KafkaConsumer {
    @KafkaListener(topics = KafkaProducer.TOPIC_TEST, groupId = KafkaProducer.TOPIC_GROUPO1)
    public void topic_test(ConsumerRecord<?, ?> record, Acknowledgment ack, @Header(KafkaHeaders.RECEIVED_TOPIC) String topic) {
        Optional message = Optional.ofNullable(record.value());
        if (message.isPresent()) {
            Object msg = message.get();
            log.info("topic_test consumed: Topic:" + topic + ",Message:" + msg);
            ack.acknowledge();
        }
    }
    @KafkaListener(topics = KafkaProducer.TOPIC_TEST, groupId = KafkaProducer.TOPIC_GROUPO2)
    public void topic_test1(ConsumerRecord<?, ?> record, Acknowledgment ack, @Header(KafkaHeaders.RECEIVED_TOPIC) String topic) {
        Optional message = Optional.ofNullable(record.value());
        if (message.isPresent()) {
            Object msg = message.get();
            log.info("topic_test1 consumed: Topic:" + topic + ",Message:" + msg);
            ack.acknowledge();
        }
    }
}
```

By command

```
bin/kafka-console-consumer.sh --zookeeper node01:2181 --topic t_cdr --from-beginning
```

Adding a topic (by command)

```
bin/kafka-topics.sh --zookeeper node01:2181 --create --topic t_cdr --partitions 3  
0 --replication-factor 2
```

For more information, see [Kafka Documentation](#).

Iceberg Development Guide

Last updated : 2022-11-25 16:06:47

Iceberg overview

Apache Iceberg is an open-source table format for large-scale data analytics and large, slow-moving tabular data storage. It is designed to improve the de facto standard table layout built into Hive, Trino (PrestoSQL), and Spark. Iceberg helps solve issues caused by the differences between data storage formats in lower layers and provides unified APIs for upper layers, so that different engines can access through the APIs.

Apache Iceberg capabilities:

- Schema evolution: Supports five actions - Add, Drop, Update, Rename, and Reorder.
- Partition layout evolution: Updates the layout of a table as data volume or query patterns change.
- Hidden partitioning: With this capability, queries no longer depend on a table's physical layout. Through a separation between physical and logical, Iceberg tables can evolve partition schemes over time as data volume changes. Misconfigured tables can be fixed without an expensive migration.
- Time travel: Enables reproducible queries that use exactly the same table snapshot, or lets users easily examine changes.
- Version rollback: Allows users to quickly correct problems by resetting tables to a good state.

Iceberg boasts high reliability and performance. It can be used in production where a single table contain tens of petabytes of data and these huge tables can be read even without a distributed SQL engine.

- Fast scan planning: A distributed SQL engine isn't needed to read a table or find files.
- Advanced filtering: Data files are pruned with partition and column-level statistics, based on table metadata.

Iceberg is designed to solve correctness problems in eventually-consistent cloud object stores.

- Works with any cloud store and reduces NameNode congestion in HDFS by avoiding listing and renaming.
- Serializable isolation: Table changes are atomic and readers never see partial or uncommitted changes.
- Multiple concurrent writers use optimistic concurrency control and will retry to ensure that compatible updates succeed, even when the writes conflict.

Iceberg is designed to manage data in tables in the form of snapshots. A snapshot is the state of a table at some time. Each snapshot is a complete list of files in a table at some time. Data files are stored across multiple manifest files, and the manifest files are listed in a single manifest list file. Manifest files can be shared among different manifest list files, and a manifest list file represents a snapshot.

- A manifest list file is a metadata file that lists the manifest files. Each manifest file takes up a row.
- A manifest file is a metadata file that lists the data files that make up a snapshot. Each row is a detailed description of each data file, including the file status, file path, partition information, column-level statistics (such as the maximum/minimum values and number of empty values in each column), file size, number of rows, etc.
- A data file is where data is stored. Generally, data files are saved in the `data` directory under the table's data storage directory.

Example

For more examples, see [here](#).

This document takes Iceberg 0.11.0 in EMR v3.3.0 as an example. The names of JAR packages may vary by EMR version.

1. Log in to the master node and switch to the `hadoop` user.
2. Iceberg packages are saved in `/usr/local/service/iceberg/`.
3. Use a compute engine to query data.

- Spark engine

- Spark-SQL interactive command line

```
spark-sql --master local[*] --conf spark.sql.extensions=org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions --conf spark.sql.catalog.local=org.apache.iceberg.spark.SparkCatalog --conf spark.sql.catalog.local.type=hadoop --conf spark.sql.catalog.local.warehouse=/usr/hive/warehouse --jars /usr/local/service/iceberg/iceberg-spark3-runtime-0.11.0.jar
```

- Insert and query data.

```
CREATE TABLE local.default.t1 (id int, name string) USING iceberg;
INSERT INTO local.default.t1 values (1, "tom");
SELECT * from local.default.t1;
```

- Hive engine

- Use Beeline.

```
beeline -u jdbc:hive2://[hiveserver2_ip:hiveserver2_port] -n hadoop --hiveconf hive.input.format=org.apache.hadoop.hive.ql.io.HiveInputFormat --hiveconf hive.stats.autogather=false
```

- Query data

```
ADD JAR /usr/local/service/iceberg/iceberg-hive-runtime-0.11.0.jar;  
CREATE EXTERNAL TABLE t1 STORED BY 'org.apache.iceberg.mr.hive.HiveIcebergStorageHandler' LOCATION '/usr/hive/warehouse/default/t1' TBLPROPERTIES ('iceberg.catalog'='location_based_table');  
select count(*) from t1;
```

- Flink engine

- Download the corresponding version of the `flink-sql-connector-hive` package from the [Maven repository](#) based on the Flink and Hive versions. The following takes the Flink standalone mode as an example and uses the Flink Shell interactive command line.

```
wget https://repo1.maven.org/maven2/org/apache/flink/flink-sql-connector-hive-3.1.2_2.11/1.12.1/flink-sql-connector-hive-3.1.2_2.11-1.12.1.jar  
/usr/local/service/flink/bin/start-cluster.sh  
sql-client.sh embedded -j /usr/local/service/iceberg/iceberg-flink-runtime-0.11.0.jar -j flink-sql-connector-hive-3.1.2_2.11-1.12.1.jar shell
```

- Query data

```
CREATE CATALOG hive_catalog WITH ('type'='iceberg', 'catalog-type'='hive', 'uri'='hivemetastore_ip:hivemetastore_port', 'clients'='5', 'property-version'='1', 'warehouse'='hdfs:///usr/hive/warehouse/');  
CREATE DATABASE hive_catalog.iceberg_db;  
CREATE TABLE hive_catalog.iceberg_db.t1 (id BIGINT COMMENT 'unique id', data STRING);  
INSERT INTO hive_catalog.iceberg_db.t1 values(1, 'tom');  
SELECT count(*) from hive_catalog.iceberg_db.t1;
```

StarRocks Development Guide

StarRocks Overview

Last updated : 2022-05-24 09:31:14

StarRocks Overview

- StarRocks is a new-generation and high-speed MPP database for nearly all data analytics scenarios.
- StarRocks takes advantage of the relational Online Analytical Processing (OLAP) database and distributed storage system. Through architectural upgrades and functional optimization, StarRocks has developed into an enterprise-level product.
- StarRocks is committed to accommodating multiple data analysis scenarios for enterprise users. It supports multiple data warehouse schemas (flat tables, pre-aggregations, star or snowflake schema) and various data import methods (batch and streaming) for up to 10,000 columns of data. It allows direct access to data from Hive, MySQL and Elasticsearch without importing.
- StarRocks is compatible with the MySQL protocol. You can use the MySQL client and common Business Intelligence (BI) tools to connect to StarRocks for data analysis.
- StarRocks uses a distributed architecture to divide the table horizontally and store it in multiple replications. The clusters are highly scalable and therefore support 10PB-level data analysis, Massively Parallel Processing (MPP), and data replication and elastic fault tolerance.
- Leveraging a relational model, strong data typing, and a columnar storage engine, StarRocks reduces read-write amplification through encoding and compression techniques. Using vectorized query execution, it fully unleashes the power of parallel computing on multicore CPUs, therefore significantly improves query performance.

StarRocks Features

StarRocks integrates the design ideas of MPP databases and distributed systems into its architecture and has the following features:

Simplified architecture

StarRocks performs the specific execution of SQL through the MPP computing framework. The framework itself can make full use of the computing power of multiple nodes and execute the entire query in parallel, so as to deliver an excellent interactive analysis experience. StarRocks does not rely on any external systems. Its clean architecture makes it easier to deploy, maintain, and scale out. Its minimalist architectural design reduces its complexity and

maintenance cost and increases its reliability and scalability. Admins only need to focus on the StarRocks system itself, with no need to learn and manage other external systems.

Native vectorized SQL engine

The computing layer of StarRocks fully adopts the vectorization technology to systematically optimize all operators, functions, scanning, filtering, and import and export modules. Through the columnar memory layout and the SIMD instruction set adapted to CPU, it makes full use of the parallel computing power of CPU, achieving sub-second query returns in multidimensional analyses.

Query optimization

StarRocks can optimize complex queries through cost-based optimizer (CBO). The execution costs can be reasonably estimated based on the statistical information with no human intervention required. With a better execution plan, the data analysis efficiency in ad hoc and ETL scenarios can be greatly improved.

Query federation

StarRocks enables you to perform federated queries by using external tables. Currently, it supports tables from Hive, MySQL, and Elasticsearch. In this way, you can quickly query data without importing data.

Efficient update

StarRocks supports multiple data models. Among them, the update model can perform UPSERT/DELETE operations according to the primary key and achieve efficient query during concurrent updates through storage and indexing optimization. This better serves real-time data warehouses.

Intelligent materialized view

StarRocks supports intelligent materialized views. Users can create materialized views and generate pre-aggregated tables to speed up aggregate queries. StarRocks' materialized view automatically runs the aggregation when data is imported, keeping it consistent with the original table. When querying, users do not need to specify a materialized view, StarRocks can automatically select the best-materialized view to satisfy the query.

Standard SQL

StarRocks supports standard SQL syntax, including aggregation, join, sorting, window functions, and custom functions. It also fully supports 22 SQL queries from TPC-H and 99 SQL queries from TPC-DS. In addition, it is compatible with the MySQL protocol, so you can use various existing client tools and BI software programs to access StarRocks and perform data analysis with simple drag-and-drops in StarRocks.

Unified batch processing and streaming

StarRocks supports batch and streaming import of up to 10,000 columns of data in ORC, Parquet, and CSV formats from Kafka, HDFS, and local files. It can consume real-time Kafka data to import the data, which avoids data loss or

duplication (i.e., exactly once). It can also import data in batches from local or remote (HDFS) data sources.

High availability and scalability

StarRocks metadata and data are stored in multiple replicas. A StarRocks cluster provides hot standby services and can be deployed as multiple instances, eliminating single points of failure. It has the ability of self-healing and elastic recovery. Therefore, the overall stability of the cluster service will not be affected by node failures, disconnections, or exceptions. StarRocks adopts a distributed architecture that makes possible to horizontally scale the storage capacity and computing power. Specifically, a cluster can be expanded to hundreds of nodes to support up to 10 PB data storage. It can normally provide the query service during scaling. In addition, the table schema of StarRocks supports hot changes, so you can use a simple SQL command to dynamically modify the table definition, for example, adding or deleting a column or creating a materialized view. You can also import data into or query data from tables during schema changes.

Use Cases

StarRocks can meet a variety of analysis needs, including OLAP analysis, custom reporting, real-time data analysis, and ad hoc data analysis. Specific business scenarios include:

- Multidimensional OLAP. User behavior analysis.
 - User profiling, tag analysis, and user tagging.
 - High-dimensional business metric reporting.
 - Self-service reporting.
 - Business problem identification and analysis.
 - Cross-topic business analysis.
 - Financial reporting.
 - System monitoring analysis.
- Real-time data analysis. Ecommerce promotion data analysis.
 - Education live streaming quality analysis.
 - Waybill analysis.
 - Finance performance analysis and metric calculation.
 - Ad placement analysis.
 - Management cockpit.
 - Application performance management (APM).
- High-concurrency queries. Advertiser report analysis.
 - Retail channel personnel analysis.
 - User-oriented analysis reporting in the SaaS industry.
 - Multi-page dashboard analysis.

- Unified analysis. The same system is used to address the needs in different scenarios, such as multidimensional analysis, high-concurrency queries, precomputing, real-time analysis, and ad hoc query, simplifying the system and reducing the cost of multi-technology stack development and maintenance.

User Guide

Last updated : 2022-05-16 12:52:26

Currently, StarRocks can be connected to in different ways. This document describes how to use a MySQL client to connect to StarRocks for development.

Logging in as Root User

Use the MySQL client to connect to the query port (9030) of an FE instance. StarRocks is preconfigured with the root user, whose password is the cluster password:

```
mysql -h fe_host -P9030 -u root -p
```

Clean up the environment:

```
mysql > drop database if exists example_db;  
mysql > drop user test;
```

Viewing Deployed Nodes

1. View the FE node.

```
mysql> SHOW PROC '/frontends'\G  
***** 1. row *****  
Name: 172.16.139.24_9010_1594200991015  
IP: 172.16.139.24  
HostName: starrocks-sandbox01  
EditLogPort: 9010  
HttpPort: 8030  
QueryPort: 9030  
RpcPort: 9020  
Role: FOLLOWER  
IsMaster: true  
ClusterId: 861797858  
Join: true  
Alive: true  
ReplayedJournalId: 64  
LastHeartbeat: 2020-03-23 20:15:07  
IsHelper: true
```

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

The value of `Role` is `FOLLOWER`, indicating that the FE is eligible for master election. The value of `IsMaster` is `true`, indicating that the FE is the current master node.

2. Viewing the BE node.

```
mysql> SHOW PROC '/backends'\G
***** 1. row *****
BackendId: 10002
Cluster: default_cluster
IP: 172.16.139.24
HostName: starrocks-sandbox01
HeartbeatPort: 9050
BePort: 9060
HttpPort: 8040
BrpcPort: 8060
LastStartTime: 2020-03-23 20:19:07
LastHeartbeat: 2020-03-23 20:34:49
Alive: true
SystemDecommissioned: false
ClusterDecommissioned: false
TabletNum: 0
DataUsedCapacity: .000
AvailCapacity: 327.292 GB
TotalCapacity: 450.905 GB
UsedPct: 27.41 %
MaxDiskUsedPct: 27.41 %
ErrMsg:
Version:
1 row in set (0.01 sec)
```

If the value of `isAlive` is `true`, the BE is connected to the cluster normally; if not, view the `be.WARNING` log file in the `log` directory to identify the cause.

3. View the broker node.

```
MySQL> SHOW PROC "/brokers"\G
***** 1. row *****
Name: broker1
IP: 172.16.139.24
Port: 8000
Alive: true
LastStartTime: 2020-04-01 19:08:35
```



```
LastUpdateTime: 2020-04-01 19:08:45
ErrMsg:
1 row in set (0.00 sec)
```

If the value of `Alive` is `true`, the status is normal.

Creating User

Create an ordinary user with the following command:

```
mysql > create user 'test' identified by '123456';
```

Creating Database

In StarRocks, only the root account has the permission to create databases. Log in as the root user and create the `example_db` database:

```
mysql > create database example_db;
```

After creating the database, you can view the database information through `show databases`:

```
mysql > show databases;
+-----+
| Database |
+-----+
| example_db |
| information_schema |
+-----+
2 rows in set (0.00 sec)
```

`information_schema` exists to be compatible with the MySQL protocol. In practice, information may not be accurate. Therefore, we recommend you get the information of a specific database by directly querying the database.

Account Authorization

After creating the `example_db` database, you can grant its read/write permissions to the test account through the root account and then log in with the test account to manipulate the database.

```
mysql > grant all on example_db to test;
```

Log out of the root account and log in to the StarRocks cluster with the test account:

```
mysql > exit
mysql -h 127.0.0.1 -P9030 -utest -p123456
```

Creating Table

StarRocks supports two table creation methods: single partitioning and composite partitioning.

In composite partitioning:

- The first level is called Partition, or partitioning. You can specify a dimension column as a partitioning column (currently only integer and time columns are supported) and specify the value range for each partition.
- The second level is called Distribution, i.e., bucketing. You can specify multiple dimension columns (or specify none, in which case, all `KEY` columns will be selected) and the number of buckets for HASH distribution of data.

Composite partitioning is recommended for the following scenarios:

- Scenarios involving time dimensions or similar dimensions with ordered values: You can use such dimension columns as a partitioning column and assess the partitioning granularity based on the import frequency and the amount of partitioned data.
- Historical data deletion: If you want to retain only data from the past N days, you can delete historical partitions through composite partitioning or delete data by sending a `DELETE` statement to the specified partition.
- Data skew elimination: You can specify the number of buckets for each partition separately. If you partition data by day, when the amount of data varies greatly every day, you can reasonably partition the data by specifying the number of buckets for the partitions. We recommend you select distinguishing columns as the bucket columns.

You can also use single partitioning instead of composite partitioning. Then, the data will be only distributed by HASH.

The following demonstrates the table creation statements for the two partitioning methods respectively:

1. Switch the database from `mysql` to `use example_db`.
2. Create a single-partitioned logical table named `table1` by assigning ten hash buckets based on `siteid`, with the schema as shown below:
 - `siteid`: The type is `INT` (4 bytes); the default value is `10`.
 - `city_code`: The type is `SMALLINT` (two bytes).

- `username` : The type is `VARCHAR` . The maximum length is `32` . The default value is an empty string.
- `pv` : The type is `BIGINT` (eight bytes), and the default value is `0` . It is a metric column and will be aggregated by StarRocks with the `SUM` method. The aggregation model is used here. In addition, StarRocks supports the detail model and update model. For more information, see [Data Model](#).

The table creation statement is as follows:

```
mysql >
CREATE TABLE table1
(
  siteid INT DEFAULT '10',
  citycode SMALLINT,
  username VARCHAR(32) DEFAULT '',
  pv BIGINT SUM DEFAULT '0'
)
AGGREGATE KEY(siteid, citycode, username)
DISTRIBUTED BY HASH(siteid) BUCKETS 10
PROPERTIES("replication_num" = "1");
```

3. Create a composite partitioned table

Create a logical table named `table2` with the schema as shown below:

- `event_day` : The type is `DATE` ; no default value.
- `siteid` : The type is `INT` (4 bytes); the default value is `10` .
- `city_code` : The type is `SMALLINT` (two bytes).
- `username` : The type is `VARCHAR` . The maximum length is `32` . The default value is an empty string.
- `pv` : The type is `BIGINT` (eight bytes), and the default value is `0` . It is a metric column and will be aggregated by StarRocks with the `SUM` method.

Use the `event_day` column as the partitioning column to create three partitions: p1, p2, and p3.

- p1: Its value range is [minimum value, 2017-06-30].
- p2: Its value range is [2017-06-30, 2017-07-31).
- p3: Its value range is [2017-07-31, 2017-08-31).

Use `siteid` to assign ten hash buckets to each partition.

The table creation statement is as follows:

```
CREATE TABLE table2
(
  event_day DATE,
  siteid INT DEFAULT '10',
  citycode SMALLINT,
  username VARCHAR(32) DEFAULT '',
```

```

pv BIGINT SUM DEFAULT '0'
)
AGGREGATE KEY(event_day, siteid, citycode, username)
PARTITION BY RANGE(event_day)
(
PARTITION p1 VALUES LESS THAN ('2017-06-30'),
PARTITION p2 VALUES LESS THAN ('2017-07-31'),
PARTITION p3 VALUES LESS THAN ('2017-08-31')
)
DISTRIBUTED BY HASH(siteid) BUCKETS 10
PROPERTIES("replication_num" = "1");

```

After creating the table, you can view the information of the table in `example_db` :

```
mysql> show tables;
```

```

+-----+
| Tables_in_example_db |
+-----+
| table1 |
| table2 |
+-----+
2 rows in set (0.01 sec)

```

```
mysql> desc table1;
```

```

+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| siteid | int(11) | Yes | true | 10 | |
| citycode | smallint(6) | Yes | true | N/A | |
| username | varchar(32) | Yes | true | | |
| pv | bigint(20) | Yes | false | 0 | SUM |
+-----+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)

```

```
mysql> desc table2;
```

```

+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| event_day | date | Yes | true | N/A | |
| siteid | int(11) | Yes | true | 10 | |
| citycode | smallint(6) | Yes | true | N/A | |
| username | varchar(32) | Yes | true | | |
| pv | bigint(20) | Yes | false | 0 | SUM |
+-----+-----+-----+-----+-----+-----+
5 rows in set (0.00 sec)

```