Elastic MapReduce
EMR Development Guide

Tencent Cloud

Contents

EMR Development Guide
Hadoop Development Guide
HDFS Common Operations

Last updated：2023-12-25 11:03:40

Tencent Cloud's EMR has integrated Hadoop with Tencent Cloud Object Storage. If you have opted for COS support during your purchase, you can manipulate data on COS using common Hadoop commands. The following commands can be used to manage data within the cluster.

```
#cat Data
hadoop fs -cat /usr/hive/warehouse/hivewithhdfs.db/record/data.txt
#Modifying Directory or File Permissions
hadoop fs -chmod -R 777 /usr
#Changing the Owner of a File or Directory
hadoop fs -chown -R root:root /usr
#Creating a Directory
hadoop fs -mkdir <paths>
#Transferring Local Files to HDFS
hadoop fs -put <localsrc> ... <dst>
#Copying Local Files to HDFS
hadoop fs -copyFromLocal <localsrc> URI
#Viewing the Storage Usage of a File or Directory
hadoop fs -du URI [URI ...]
#Deleting a File
hadoop fs -rm URI [URI ...]
#Setting the Replication Factor for a Directory or File
hadoop fs-setrep [-R] [-w] REP PATH [PATH ...]
#Checking for Corrupted Blocks in the Cluster Files
hadoop fsck <path> [-move | -delete | -openforwrite] [-files [-blocks [-locations | -racks]]]
```

For additional HDFS commands, please refer to the Community Documentation. Furthermore, if your cluster is an HA cluster (dual namenode), you can determine which namenode is active by using the following command.

```
#nn1 is the ID of the namenode, typically designated as nn1 and nn2.
hdfs haadmin -getServiceState nn1
#Viewing the Current Cluster Report
hdfs dfsadmin -report
#Exiting the Namenode Safe Mode
hdfs dfsadmin -safemode leave
```

HDFS Federation Management Development Guide

Last updated：2023-12-25 15:00:45

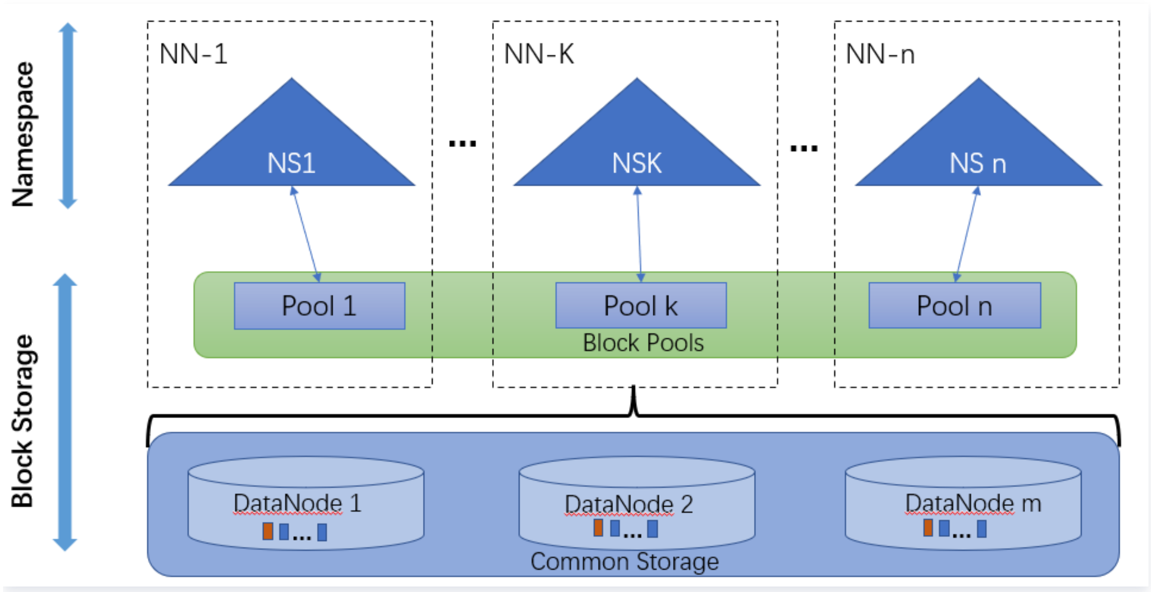
Selection of HDFS Federation Management Type

Note

The current HDFS Federation Management is open to a whitelist. If needed, you can submit a service ticket to enable it.

HDFS Federation Management Architecture

HDFS Federation utilizes multiple independent NameNode/Namespaces to enable horizontal scaling of HDFS naming services. In the federation, each DataNode is used by all NameNodes as a common storage block location; every DataNode registers with all NameNodes in its cluster, periodically sending heartbeat and block information reports, while also processing instructions from all NameNodes.



For more information on HDFS Federation, please refer to the Hadoop Documentation.

ViewFs Federation Principle

To facilitate the management of multiple namespaces, HDFS ViewFs Federation employs the classic Client Side Mount Table (community ViewFs feature). This maps different application paths to specific NameServices on the client side, thereby achieving storage separation or performance separation scenarios. HDFS ViewFs Federation uses a Client-Side Mount Table to distribute files and load, a method that requires more manual intervention (explicit planning) to achieve optimal load balancing.

Router-based Federation Principle

Router-based Federation provides a software layer to manage multiple NameSpace spaces. Compared to ViewFs, which maintains mount table information on the client side, Router-based Federation truly achieves complete transparency for the client. This is because this part of the mapping information will be additionally saved and persisted.

HDFS Federation Management Configuration

Two main aspects need to be considered: the required number of NameServices and the method of mounting business data directories.

Principles for Planning the Number of NameServices

1. The secure storage capacity limit for a single NameService is 100 million files. If this number is exceeded, the access speed and read/write throughput of the NameService will be significantly reduced. Therefore, if the file storage volume (anticipated) exceeds 100 million files, a new NameService needs to be added.

2. For heavy usage of HDFS (i.e., extensive reading and writing of files in HDFS), it is necessary to allocate a separate NameService to handle its requests. This ensures that the application's data can monopolize all the processing capabilities of the NameService, and also avoids the application's impact on other applications.
3. For applications with stringent reliability requirements, a separate NameService can be allocated to prevent high read/write frequencies from other applications causing this application to be unable to access HDFS, resulting in instability of the application's business operations.

Principles for Planning the Mounting Method of Business Data Directories

1. Data directories related to business data services should ideally be mounted under the same NameService. Otherwise, the file read/write speed across NameServices is slower, which can reduce the storage performance of the application.
2. For large volumes of data that are unrelated to the business of other services, a single NameService can be directly utilized.
3. For businesses with smaller volumes, it is recommended to directly mount their directories in the default NameService (HDFS${clusterid}, HDFS concatenated with numeric form clusterid). This eliminates the need for data migration and reduces the complexity of configuring for Federation.
4. It is recommended to map only the global top-level directories to NameService in order to reduce configuration complexity.

Comparative Analysis of Solutions

ViewFs Federation Type

1. Direct Use of ViewFs
   Advantages: Unified view, different applications have the same usage method.
   Disadvantages: Changes to the ViewFs mount table require all applications using the cluster to synchronously read the latest mount points.
2. Specifying NameService

   Preference: There is no need to synchronously change the configuration of all applications.
   Drawback: Different components and applications need to clearly define their respective usage directories. Scenarios where component paths are interlinked will become more complex.

Configuration Method for Router-based Federation Type

1. Direct Use of Router-based Federation

   Advantages: Unified view, different applications have the same usage method. Compared to ViewFs, changes to the Router-based Federation mount table take effect directly, without the need for applications using the cluster to synchronously update the mount table.
   Disadvantages: When the client accesses, an additional DFSRouter forwarding layer is added, which slightly impacts performance.
2. Specifying NameService

   Advantages: There is no need to synchronously change the configuration of all applications.
   Disadvantages: Different components and applications need to clearly define their respective usage directories. Scenarios where component paths are interlinked will become more complex.

HDFS Federation Management

Last updated：2023-12-25 15:01:02

Feature Overview

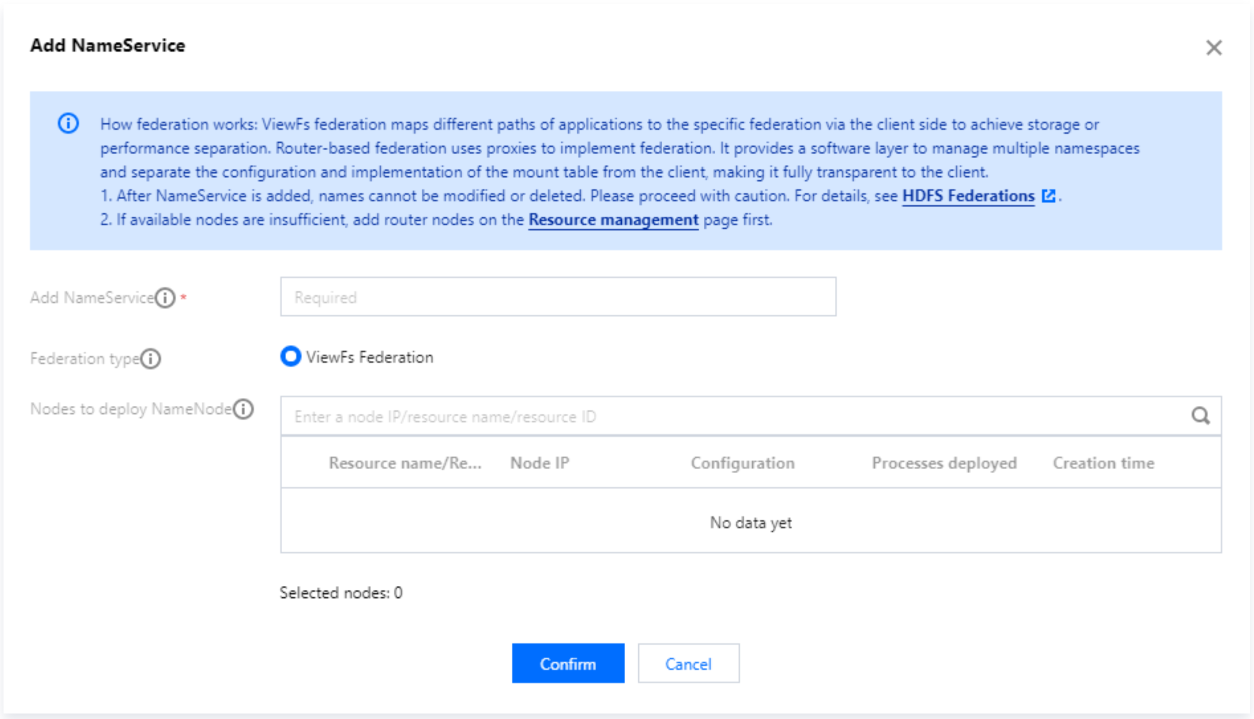
HDFS Federation Management is a deployment management capability for HDFS Federation clusters, based on the HDFS Federation feature. It includes NameService management and mount table management. Federation management is supported in the HA mode of Hadoop cluster types, with the option to choose between ViewFS Federation and Router-based Federation. Once the federation type is selected, it cannot be changed. Router nodes will be used for the deployment of newly expanded NameNodes. Once a Router node has been used for NameNode deployment, it does not support destruction or the start-stop of all roles at the node level.

Note

1. Currently, HDFS Federation Management is open to a whitelist. If needed, you can submit a ticket to contact us for activation.
2. All product versions of EMR support the ViewFs Federation type. As HDFS-2.9.0 and above support the Router-based Federation type, only EMR-V3.x.x and above product versions support this federation type. The EMR-V2.x.x series does not support it.
3. In the Role Management page, pausing the NameNode role process of the federation node will affect the cluster's expansion operation. The expansion can only be executed after the NameNode role process has been resumed.

Instructions

1. Log in to the EMR Console, and in the cluster list, click on the corresponding **Cluster ID/Name** to enter the **Cluster Details Page**.
2. In the Cluster Details Page, click on **Cluster Services**, then select **HDFS Component Top Right Operation>Federation Management** to enter the Federation Management page.
3. Click on **Add NameService** to create a new HDFS federation. You will need to input the NameService name, select the federation type, NameNode node, DFSRouter node (if Router-based Federation is selected), and so forth.



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 NameService is added, names cannot be modified or deleted. Please proceed with caution. For details, see **HDFS Federations**.
2. If available nodes are insufficient, add router nodes on the **Resource management** page first.

Add NameService * : Required

Federation type: ViewFs Federation

Nodes to deploy NameNode: Enter a node IP/resource name/resource ID

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

Selected nodes: 0

4. Select **Add Federation Node**
   The federation node uses the Router node in the cluster. You need to first expand and add Router nodes in the Resource Management page, then set them as federation nodes; The NameNode process requires two nodes, each of which will deploy the NameNode process and ZKFC process.

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 a federation again, the DFSRouter nodes can be reused, and the number of nodes can be zero or more. When the federation type is Router-based Federation, if the DFSRouter process load is high, you can go to the **Resource Management Page** to expand the Router node and deploy the DFSRouter process; if the DFSRouter process load is low, you can go to the **Resource Management Page** to reduce the Router federation node that deploys the DFSRouter process.

> ⚠ Note
> - For versions below HDFS-3.3.0, after successfully adding a NameService for the **first time** and when the federation type is Router-based Federation, it is necessary to restart the historical DFSRouter process on the Role Management page (to ensure normal business operations, it is recommended to perform this during off-peak business hours); HDFS-3.3.0 and above versions support hot loading configurations, so this operation is not required.
> - For clusters with Kerberos enabled, after adding a federation NameService, if tasks submitted to yarn need to use files on the new NameService, it is necessary to first restart yarn's ResourceManager (to ensure normal business operations, it is recommended to perform this during off-peak business hours).
> - Once the NameService name is set, it cannot be modified or deleted, and the NameService name cannot be system keywords such as "nsfed", "haclusterX", "ClusterX", etc.

5. Add Mount Table
Only after successfully adding a new NameService can you add a mount table. To avoid excessive configuration complexity, it is recommended to map the current cluster directory only at the global first-level directory for NameService, such as mounting "/tmp", "/user", "/srv", etc.; multiple mount paths can be added in batches.
When the federation type is Router-based Federation, it supports one-to-many mounting of global paths and target NameService, and also supports adding mount tables in batches through text.
   - Global Path: The path name in the ViewFs unified namespace, the path name in the Router-based Federation unified namespace, also known as the mount point.
   - Target NameService: The NameService corresponding to the real path to which the mount point is mapped.
   - Target Path: The real path on the corresponding NameService, which does not need to maintain consistency with the name of the global path.

   > ⚠ Note
   > - Path Direction: Log into the NameNode node, execute hdfs dfs -ls /, which points to the path under the namespace managed by this NameNode. For ViewFs federation, you need to use hdfs dfs -ls ViewFs://ClusterX/ to point to the global path. For Router-based federation, you need to use hdfs dfs -ls hdfs://nsfed/ to point to the global path. When logging into other nodes, such as Router nodes acting as clients, hdfs dfs -ls / points to the global path.
   > - Data from various business components should be placed under the first-level directory, and direct access from the root directory is not supported. The root directory does not support mounting.
   > - On the default NameService, there is an /emr directory that requires mounting.
   > - When using Router-based federation, if multiple target NameServices use the same mount point, and the same directory and file exist in multiple target NameServices, only one file with the same name will be retrieved. To avoid such situations, it is recommended to use hdfs://nsfed for unified reading and writing after enabling federation.

6. Synchronization of Mount Tables:
When the federation type is Router-based Federation, it supports synchronizing all background mount table information to the console, allowing users to manage mount tables uniformly.

Submitting MapReduce Tasks

Last updated：2023-12-25 15:01:14

This manual solely delineates the fundamental operations of MapReduce tasks under command-line mode, as well as how MapReduce computational tasks access data on Tencent Cloud Object Storage (COS). For comprehensive information, please refer to the community resources.

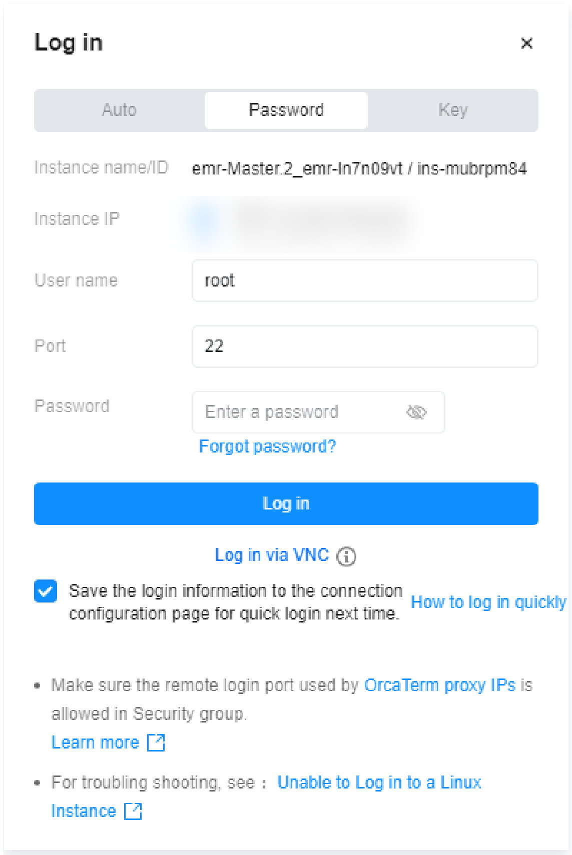
- The task submitted this time is a wordcount task, that is, to tally the number of words. It necessitates the preliminary uploading of the file to be counted within the cluster.
- The paths for related software such as Hadoop can be found under `/usr/local/service/`.
- The relevant log path is located under `/data/emr`.

1. Development Preparation

- Given that the task necessitates access to Tencent Cloud Object Storage (COS), it is imperative to initially create a storage bucket (Bucket) in COS.
- Ensure that you have activated Tencent Cloud and have established an EMR cluster. When creating the EMR cluster, select **Enable COS** on the basic configuration page, and fill in your own SecretId and SecretKey below. The SecretId and SecretKey can be viewed on the API Key Management Interface. If you do not yet have a key, please click on **Create New Key** to establish a new one.

2. Log into the EMR Server

Prior to performing related operations, it is necessary to log into any machine within the EMR cluster, with a recommendation to log into the Master node. As EMR is established on Tencent Cloud Server CVM based on the Linux operating system, using EMR in command line mode requires logging into the CVM server.
After creating the EMR cluster, select Elastic MapReduce in the console. Click on **Master Node** under **Cluster Resources > Resource Management**, select the **Resource ID** of the Master node, and you will be able to enter the cloud server console and locate the cloud server corresponding to EMR.
The method for logging into CVM can be referred to in Logging into Linux Instances. Here, we can opt to use WebShell for login. Click on **Log in** on the right side of the corresponding cloud server to enter the login interface. The default username is root, and the password is the one input by the user when creating EMR.


Log in

Auto | Password | Key

Instance name/ID: emr-Master.2_emr-ln7n09vt / ins-mubrpm84

Instance IP: [redacted]

User name: root

Port: 22

Password: Enter a password

Forgot password?

Log in

Log in via VNC

✓ Save the login information to the connection configuration page for quick login next time. How to log in quickly

- Make sure the remote login port used by OrcaTerm proxy IPs is allowed in Security group. Learn more
- For troubling shooting, see : Unable to Log in to a Linux Instance

Upon correct input, you can enter the command line interface of the EMR cluster. All Hadoop operations are under the Hadoop user. After logging into the EMR node, you are by default in the root user and need to switch to the Hadoop user. Use the following command to switch users and enter the Hadoop folder:

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

3. Data Preparation

You need to prepare the text files for statistics. There are two methods: **Data stored in HDFS cluster** and **Data stored in COS**. Firstly, local data needs to be uploaded to the cloud server. You can use scp or sftp services to upload local files to the cloud server in the EMR cluster. Use the following in the local command line:

```
scp $localfile root@PublicIPAddress:$remotefolder
```

In this context, $localfile is the path and name of your local file; root is the username of the CVM server; the public IP address can be viewed in the node information in the EMR console or in the cloud server console; $remotefolder is the path on the CVM server where you want to store the file.
After successful upload, you can check in the EMR cluster command line whether there are corresponding files in the respective folder.

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

Data stored in HDFS

After uploading the data to the Tencent cloud server, you can copy the data to the HDFS cluster. Here, the README.txt text file in the `/usr/local/service/hadoop` directory is used for illustration. Copy the file to the Hadoop cluster using the following command:

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

After the copy is complete, use 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 under Hadoop, users can create it themselves using the following command:

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

For more Hadoop commands, see Common HDFS Operations.

Data is stored in COS.

There are two methods for storing data in COS: **uploading locally via the COS console** and **uploading via Hadoop commands in the EMR cluster**.

- To upload directly via the COS console locally, if the data file is already in COS, it can be viewed using 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
  ```

  Where $bucketname should be replaced with the name and path of your storage bucket.
- To upload via Hadoop commands in the EMR cluster, use 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 tasks via MapReduce.

The task submitted this time is the built-in routine wordcount of the Hadoop cluster. Wordcount has been compressed into a jar package and uploaded to the created Hadoop, which users can directly utilize.
The example uses Hadoop version 3.2.2. For other versions, the corresponding version of the hadoop-mapreduce-examples package should be used in the example code. The actual existing versions can be viewed in the ./share/hadoop/mapreduce/ directory under the Hadoop directory.

Calculating text files within HDFS.

Navigate to the `/usr/local/service/hadoop` directory, as in the data preparation stage. Submit the task using the following command:

```
[hadoop@10 hadoop]$ bin/yarn jar ./share/hadoop/mapreduce/hadoop-mapreduce-examples-3.2.2.jar  wordcount
/user/hadoop/README.txt /user/hadoop/output
```

Note
The entire command above is a complete instruction. `/user/hadoop/README.txt` is the input file to be processed, and `/user/hadoop/output` is the output folder. Ensure that the output folder has not been created before submitting the command, otherwise, an error will occur.

Upon completion, view the execution output file using 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-00000
```

View the statistical results in part-r-00000 using 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
......
```

Statistical analysis of text files in COS

Navigate to the `/usr/local/service/hadoop` directory and submit the task using the following command:

```
[hadoop@10 hadoop]$ bin/yarn jar ./share/hadoop/mapreduce/hadoop-mapreduce-examples-3.2.2.jar  wordcount
cosn://$bucketname/README.txt /user/hadoop/output
```

The input file for the command has been changed to `cosn:// $bucketname /README.txt`, which processes files in COS, where $bucketname is the name and path of your bucket. The output is still directed to the HDFS cluster, but you can also choose to output to COS. The method for viewing the output remains the same as described above.

Review task logs

```
#Review task status
bin/mapred job -status jobid
#Inspect Task Logs
yarn logs -applicationId id
```

Automatically Adding Task Nodes Without Assigning ApplicationMasters

Last updated：2023-12-25 15:12:43

Feature Overview

In the context of automatic scaling, when a scaling-in rule is triggered, if the Task node that is about to be destroyed is running an ApplicationMaster (AM), the AM will also be destroyed along with the node, leading to the overall failure of the current Job.

In the context of automatic scaling, Task nodes added through scaling-out do not allocate an ApplicationMaster by default. This ensures that when a Task node is scaled-in, the running ApplicationMaster is not destroyed, allowing the Job to proceed normally.

Features

Through modifications to the YARN source code and the addition of configuration items to the automatic scaling process, Task nodes added during automatic scaling will not be allocated an ApplicationMaster (AM). Instead, all AMs will be allocated to Task nodes not involved in automatic scaling, with the Task nodes added during automatic scaling solely undertaking computational tasks. When scaling-in Task nodes, only nodes running computational tasks will be destroyed, while the corresponding AM remains active. The AM can retry the related computational tasks on other nodes, allowing the current Job to continue.

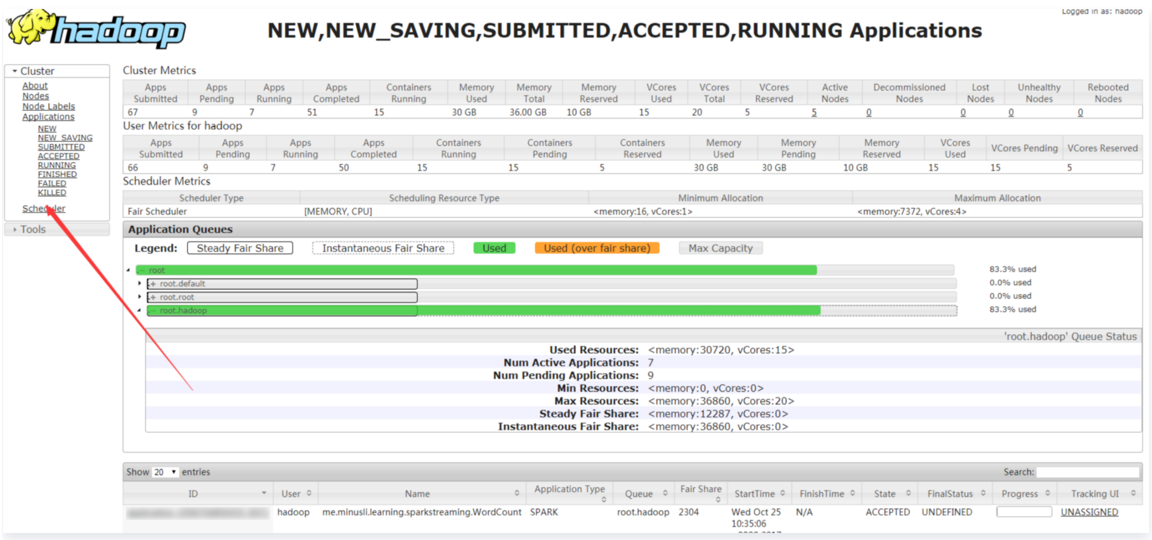
Application Scope

This feature is only applicable to Task nodes that are automatically scaled-out; it does not apply to manually scaled-out nodes.

YARN Task Queue Management

Last updated：2023-12-25 15:12:56

You may access the built-in YARN WebUI by utilizing the shortcut provided by EMR. For guidance on viewing the shortcut, please refer to Software WebUI Entrance. Upon logging in, click on **Scheduler** in the list on the left as shown in the figure below to view the task queue information.



NEW,NEW_SAVING,SUBMITTED,ACCEPTED,RUNNING Applications

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved	Active Nodes	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

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

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

Application Queues

Legend: Steady Fair Share | Instantaneous Fair Share | Used | Used (over fair share) | Max Capacity

Queue	Usage
root	83.3% used
root.default	0.0% used
root.root	0.0% used
root.hadoop	83.3% used

'root.hadoop' Queue Status

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>

Show 20 entries Search:

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

The image below provides a glimpse into some of the monitoring information for the entire cluster:

- Application Information: 9 awaiting, 7 in progress, 51 completed, totaling 67; of which, 15 containers are currently in execution.
- Memory Information: Total of 36GB, 30GB in use, 10GB reserved.
- Virtual Core Information: Total of 20, 15 in use, 5 reserved.
- Node Information: 5 nodes available, 0 decommissioned, 0 lost nodes, 0 unhealthy nodes, 0 rebooted nodes.
- Scheduler Information: Utilizing the Fair Scheduler, with details on maximum and minimum memory and CPU allocation.

The Application Queues section encompasses the task queue information of the cluster (taking the root.hadoop queue as an example):

- Resources Utilized: 30720MB (30GB) of memory, 15 virtual cores.
- Currently executing 7 applications.
- Nine applications pending.
- Minimum resources occupied: 0MB, 0 cores.
- Maximum resources occupied: 36860MB, 20 cores.
- Stable public sharing of queue resources.
- Instantaneous fair sharing of queue resources.
- The queue occupies 83.3% of the resources.

Practices on YARN Label Scheduling

Last updated：2023-12-25 15:13:13

Feature Overview

Spark on Yarn employs Yarn as its resource scheduler. Following Hadoop2.7.2, Yarn has enhanced its container scheduler, the Capacity Scheduler, with label scheduling.
The Capacity Scheduler is a multi-tenant resource scheduler, the central philosophy of which is the shared utilization of available resources in the Hadoop cluster by multiple organizations. These organizations contribute computational capacity to the cluster based on their individual computational needs. The Capacity Scheduler boasts features such as hierarchical queues, capacity guarantees, security, elastic resources, multi-tenancy, resource-based scheduling, and mapping. All resources of the cluster are allocated to multiple queues, and all applications submitted to the queue can utilize the resources allocated to the queue. Idle resources not used by other organizations can be non-preemptively allocated to applications running below capacity on the queue, ensuring that applications receive the minimum resource capacity while flexibly meeting the resource needs of different applications.
The Capacity Scheduler roughly allocates cluster resources to different queues, without specifying the running location of applications within the queue. Label scheduling, on top of the Capacity Scheduler, allows for a more granular resource division by assigning different Node Labels to each node in the cluster, enabling applications to specify their running location. Node labels possess the following characteristics:

1. Each node possesses a single node label (i.e., belongs to one node partition), dividing the cluster into multiple disjoint subclusters based on node partitions.
2. Based on matching strategies, node partitions are of two types: exclusive and non-exclusive. Exclusive node partitions allocate containers to nodes that perfectly match the node partition, while non-exclusive node partitions share idle resources with containers requesting the default partition.
3. Each queue designates the node partition where applications run by setting accessible node labels.
4. It is possible to set the resource proportion of different queues within each node partition.
5. When specifying the required node label in a resource request, allocation only occurs when the node possesses the same label. If no node label requirement is specified (which can be altered by modifying the default label name in the queue configuration), such resource requests are allocated only on nodes belonging to the default partition.

Configuration Notes

1. Enable Capacity Scheduler in ResourceManager.

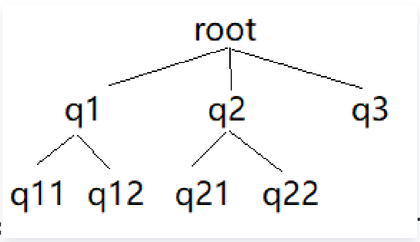
Label scheduling cannot be used independently; it must be used in conjunction with the Capacity Scheduler. Yarn employs the Capacity Scheduler as the default scheduler. If you are currently using another scheduler, please initiate the Capacity Scheduler first. Set it in `${HADOOP_HOME}/etc/hadoop/yarn-site.xml` :

```
<property>
    <name>yarn.resourcemanager.scheduler.class</name>
    <value>org.apache.hadoop.yarn.server.resourcemanager.scheduler.capacity.CapacityScheduler</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 for the Capacity Scheduler, with all other queues being sub-queues of the root queue. All queues are organized in a tree structure. `yarn.scheduler.capacity.<queue-path>.queues` is used to set the sub-queues under the queue-path, separated by commas.

Example:



The configuration of the above structure is 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 configurations, please refer to the documentation.

3. Enable Node Label in ResourceManager

Set in `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. Ensure that `yarn.node-labels.fs-store.root-dir` has been created and that ResourceManager has access rights.
2. If node labels are stored in the RM's local file system, paths such as `file://home/yarn/node-label` can be used. However, to ensure high availability of the cluster and prevent loss of label information due to RM downtime, it is recommended to store label information on HDFS.
3. Under hadoop2.8.2, the configuration item `yarn.node-labels.configuration-type` needs to be set.

4. Configure Node Label

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

Configuration items	Description
yarn.scheduler.capacity. `<queue-path>` .capacity	Set the percentage of nodes belonging to the DEFAULT partition that the queue can access. **The total DEFAULT capacity of direct sub-queues under each parent queue must equal 100.**
yarn.scheduler.capacity. `<queue-path>` .accessible-node-labels	Set the specific list of labels that the queue can access, separated by commas. For instance, "HBASE, STORM" implies that the queue can access nodes with the HBASE and STORM labels. All queues can access nodes without labels. If this field is not specified, it will be inherited from its parent field. If the user wishes to restrict the queue to only access nodes without labels, this field can simply be left blank.
yarn.scheduler.capacity. `<queue-path>` .accessible-node-labels. `<label>` .capacity	Establish the percentage of nodes belonging to the `<label>` partition that the queue can access. **The total `<label>` capacity of direct sub-queues under each parent queue must equal 100, defaulting to 0.**

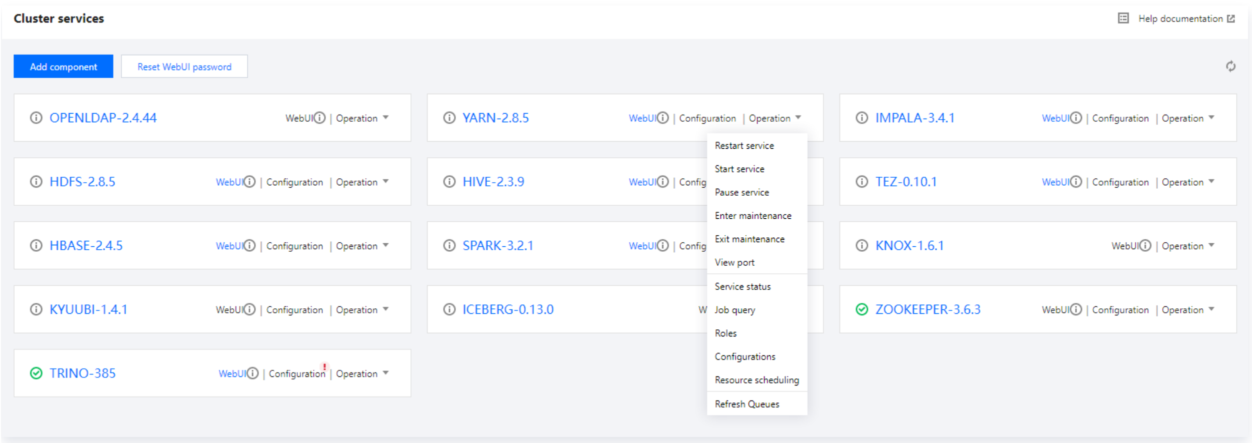
yarn.scheduler.capacity.`<queue-path>`.accessible-node-labels.`<label>`.maximum-capacity	Similar to the Capacity Scheduler configuration item yarn.scheduler.capacity.`<queue-path>`.maximum-capacity, it specifies the maximum capacity of `<queue-path>` in the `<label>` partition, defaulting to 100.
yarn.scheduler.capacity.`<queue-path>`.default-node-label-expression	If no node label is specified in the resource request, the application will be submitted to the container this value corresponds to. This value defaults to null, indicating the application will be assigned with a container in an untagged node.

Use Cases

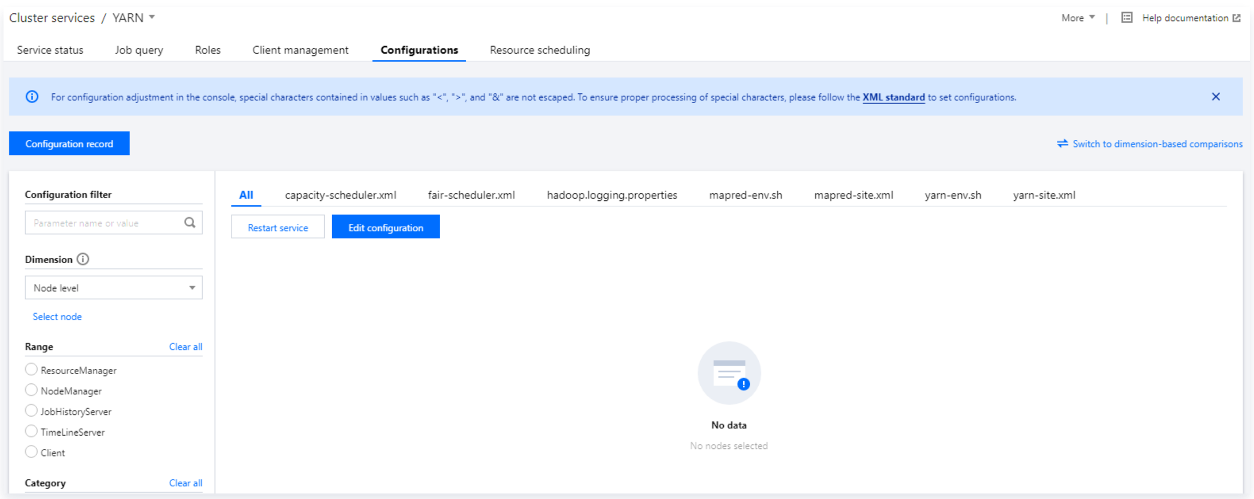
Preparations

1. Cluster Preparation
   Ensure that you have activated Tencent Cloud and have created an EMR cluster.
2. Inspecting YARN Component Configuration
   Navigate to the "Cluster Services" page, select the YARN component to enter the component management interface, then switch to the configuration management tab. Modify the relevant parameters in `yarn-site.xml`, save and restart all YARN components. Confirm the IP of the node where the ResourceManager service is located in the role management tab, then switch to the configuration management tab to modify the relevant parameters in `yarn-site.xml`, save and restart all YARN components.

- Click on the cluster instance ID in the cluster list to enter the cluster information page, then click on **Cluster Services** in the left menu bar, and select **Operations > Configuration Management** in YARN component management.

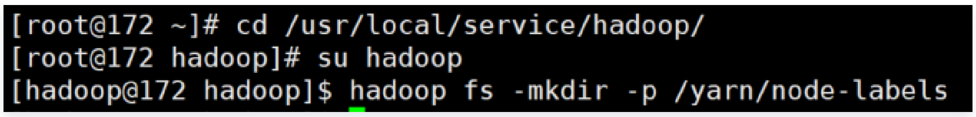

Cluster services
Help documentation
Add component
Reset WebUI password
OPENLDAP-2.4.44 WebUI | Operation
YARN-2.8.5 WebUI | Configuration | Operation
IMPALA-3.4.1 WebUI | Configuration | Operation
HDFS-2.8.5 WebUI | Configuration | Operation
HIVE-2.3.9 WebUI | Config
TEZ-0.10.1 WebUI | Configuration | Operation
HBASE-2.4.5 WebUI | Configuration | Operation
SPARK-3.2.1 WebUI | Config
KNOX-1.6.1 WebUI | Operation
KYUUBI-1.4.1 WebUI | Configuration | Operation
ICEBERG-0.13.0
ZOOKEEPER-3.6.3 WebUI | Configuration | Operation
TRINO-385 WebUI | Configuration | Operation
Restart service
Start service
Pause service
Enter maintenance
Exit maintenance
View port
Service status
Job query
Roles
Configurations
Resource scheduling
Refresh Queues

- Confirm the IP address of the ResourceManager (RM).
- On the YARN component "Configuration Management" page, select **Dimension** as Node Dimension, choose the node as the IP address of the ResourceManager (RM), and click on **Edit Configuration** to modify the `yarn.resourcemanager.scheduler.class` parameter in `yarn-site.xml` on the RM node.


For configuration adjustment in the console, special characters contained in values such as "<", ">", and "&" are not escaped. To ensure proper processing of special characters, please follow the XML standard to set configurations.

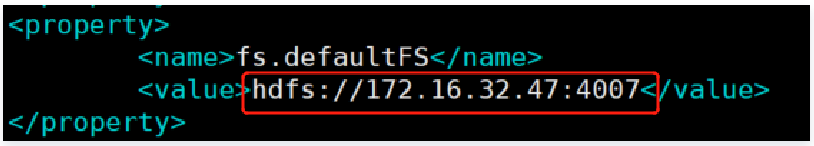
Configure the mapping relationship and proportion between Node Label and queue in Capacity-Scheduler.xml.

1. Create an HDFS directory for storing node labels.



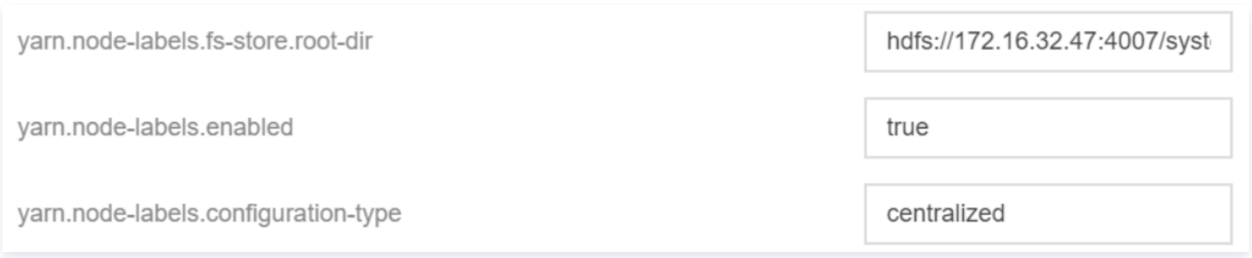
```
[root@172 ~]# cd /usr/local/service/hadoop/
[root@172 hadoop]# su hadoop
[hadoop@172 hadoop]$ hadoop fs -mkdir -p /yarn/node-labels
```

2. Obtain the NameNode (NN) IP and Port from `core-site.xml`.



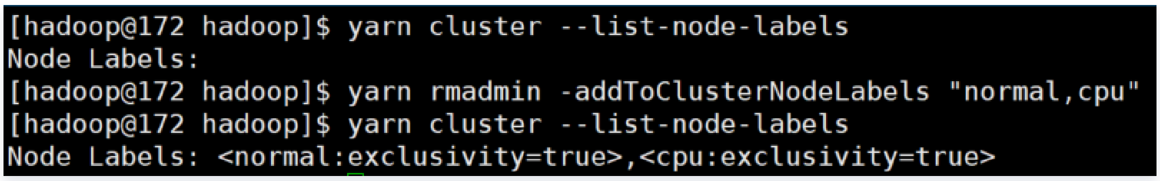
```
<property>
        <name>fs.defaultFS</name>
        <value>hdfs://172.16.32.47:4007</value>
</property>
```

3. After creating a new configuration item in `yarn-site.xml` on the master node, restart the ResourceManager.



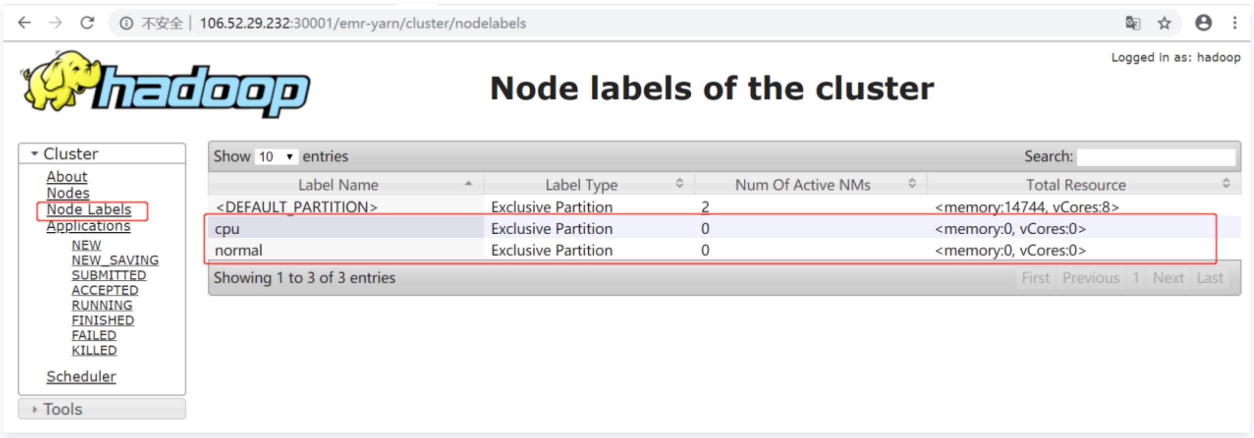
yarn.node-labels.fs-store.root-dir	hdfs://172.16.32.47:4007/syst
yarn.node-labels.enabled	true
yarn.node-labels.configuration-type	centralized

4. Use the `yarn rmadmin -addToClusterNodeLabels` command to add new labels.



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

Open the WebUI interface of the YARN component, and you can see all the labels of the cluster in the NodeLabels panel.

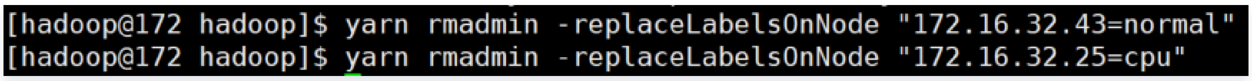


Node labels of the cluster

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>

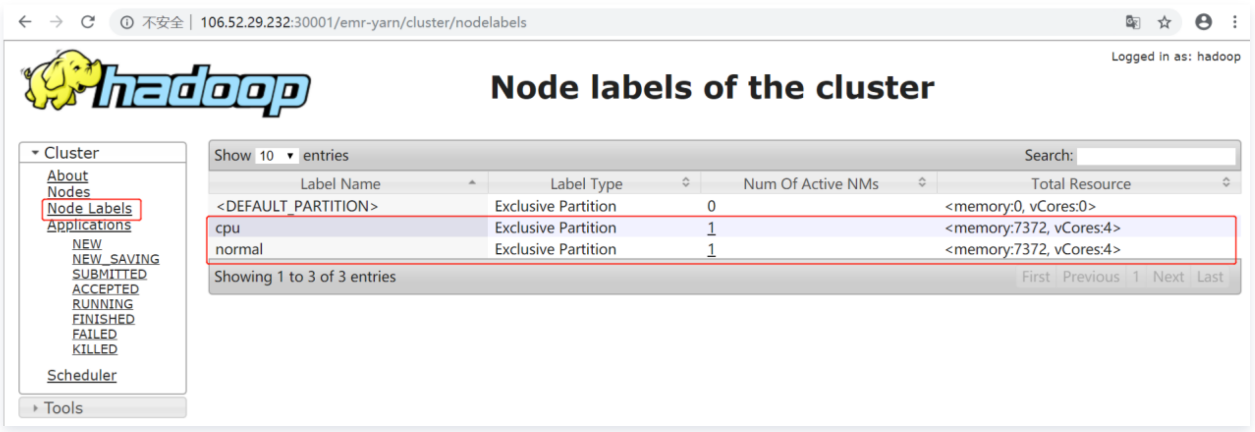
Showing 1 to 3 of 3 entries

5. Use the `yarn rmadmin -replaceLabelsOnNode` command to label nodes.



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

In the NodeLabels panel, you can see the number of nodes in the normal and cpu partitions change from 0 to 1.


不安全 | 106.52.29.232:30001/emr-yarn/cluster/nodelabels

Logged in as: hadoop

Node labels of the cluster

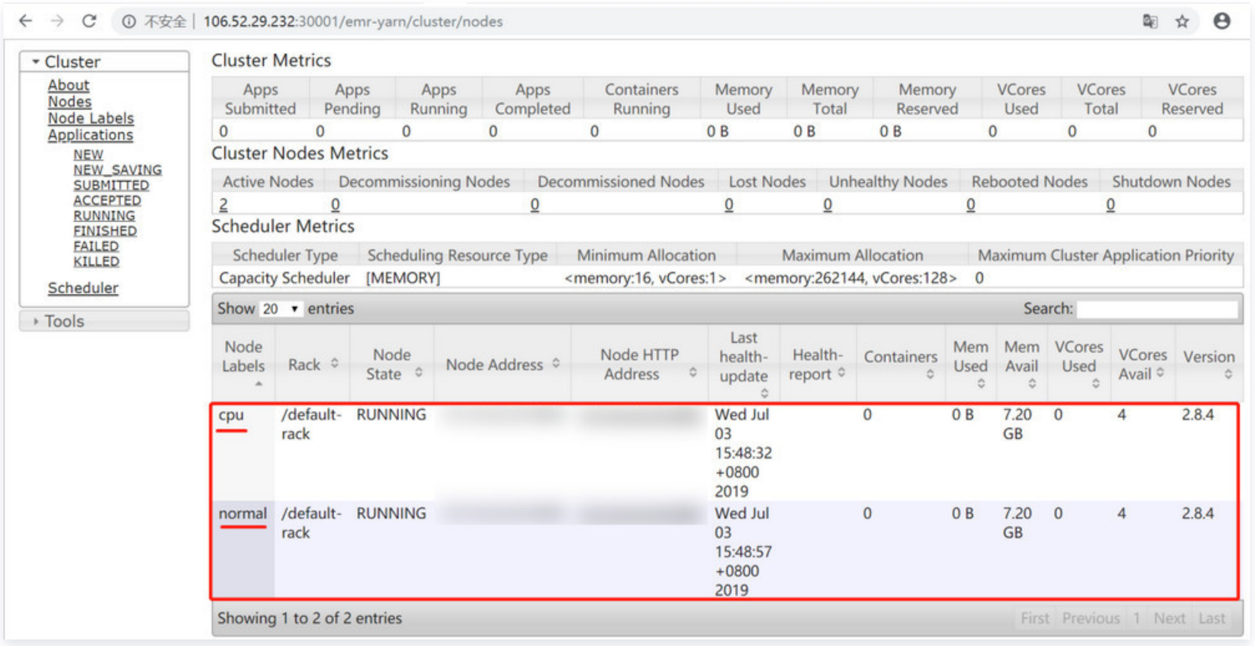
Cluster: About, Nodes, Node Labels, Applications (NEW, NEW_SAVING, SUBMITTED, ACCEPTED, RUNNING, FINISHED, FAILED, KILLED), Scheduler; Tools

Show 10 entries · Search:

Label Name	Label Type	Num Of Active NMs	Total Resource
<DEFAULT_PARTITION>	Exclusive Partition	0	<memory:0, vCores:0>
cpu	Exclusive Partition	1	<memory:7372, vCores:4>
normal	Exclusive Partition	1	<memory:7372, vCores:4>

Showing 1 to 3 of 3 entries · First Previous 1 Next Last

In the Scheduler panel, you can see that the labels corresponding to the two nodes of the test system have changed.


不安全 | 106.52.29.232:30001/emr-yarn/cluster/nodes

Cluster: About, Nodes, Node Labels, Applications (NEW, NEW_SAVING, SUBMITTED, ACCEPTED, RUNNING, FINISHED, FAILED, KILLED), Scheduler; Tools

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved
0	0	0	0	0	0 B	0 B	0 B	0	0	0

Cluster Nodes Metrics

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

Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation	Maximum Cluster Application Priority
Capacity Scheduler	[MEMORY]	<memory:16, vCores:1>	<memory:262144, vCores:128>	0

Show 20 entries · Search:

Node Labels	Rack	Node State	Node Address	Node HTTP Address	Last health-update	Health-report	Containers	Mem Used	Mem Avail	VCores Used	VCores Avail	Version
cpu	/default-rack	RUNNING	[redacted]	[redacted]	Wed Jul 03 15:48:32 +0800 2019		0	0 B	7.20 GB	0	4	2.8.4
normal	/default-rack	RUNNING	[redacted]	[redacted]	Wed Jul 03 15:48:57 +0800 2019		0	0 B	7.20 GB	0	4	2.8.4

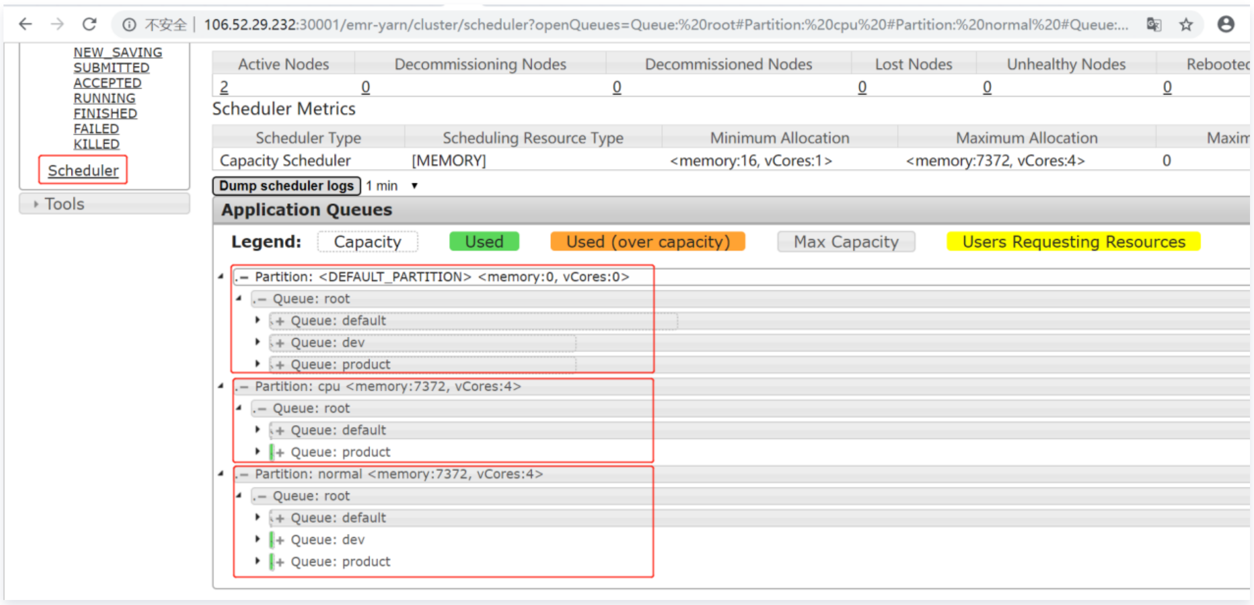
Showing 1 to 2 of 2 entries · First Previous 1 Next Last

6. Edit the configuration items in `Capacity-Scheduler.xml` , configure the cluster queue, the resource proportion of the queue, and the accessible labels of the queue. An example is as follows:

```
<?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>
```

In the Scheduler panel, you can see the three partitions of the test cluster, the resource allocation of the partitions, and the queues they contain. In the Application Queues panel, there are three partitions: default, normal, and cpu. The default partition is the default partition, the normal partition is the partition composed of nodes with the normal label, and the cpu partition is the partition composed of nodes with the cpu label. In the test environment, there are two nodes, which are respectively marked as normal and cpu. Clicking on the plus sign to the left of the partition will expand the queues contained in that partition.



106.52.29.232:30001/emr-yarn/cluster/scheduler?openQueues=Queue:%20root#Partition:%20cpu%20#Partition:%20normal%20#Queue:...

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

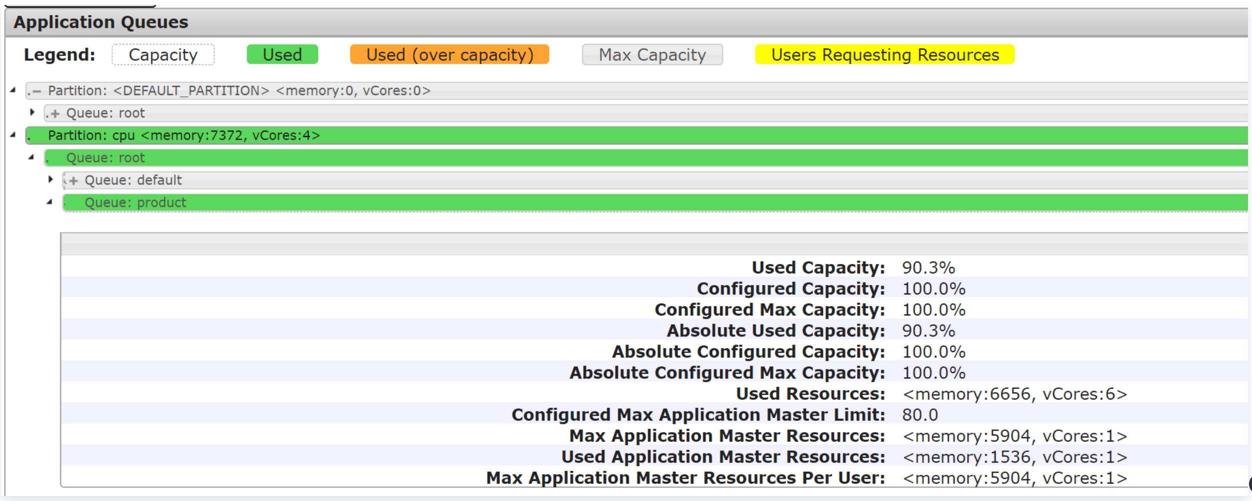
```
- Partition: <DEFAULT_PARTITION> <memory:0, vCores:0>
  - Queue: root
    + Queue: default
    + Queue: dev
    + Queue: product
- Partition: cpu <memory:7372, vCores:4>
  - Queue: root
    + Queue: default
    + Queue: product
- Partition: normal <memory:7372, vCores:4>
  - Queue: root
    + Queue: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

Verify label scheduling.

- **Test One: Submitting Tasks 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 1000
```

The resource usage of each partition's queue after task submission is as shown in the following figure:



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>

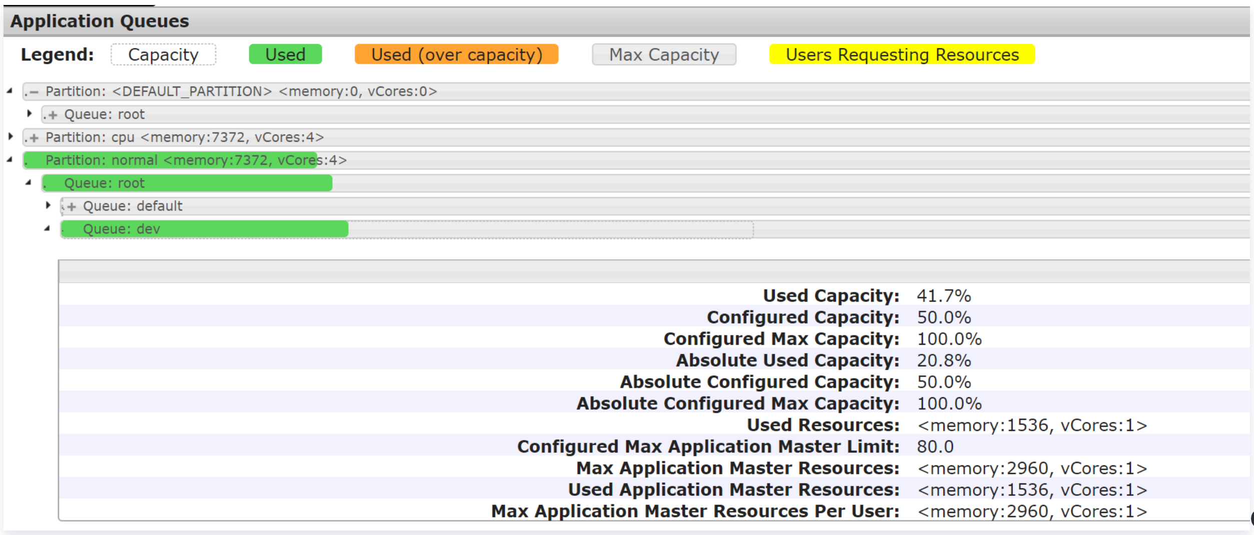
Conclusion:

The product queue is mapped to the cpu label and the default label used is cpu. Tasks submitted to the product queue run on nodes marked with cpu.

- **Test Two: Submitting Tasks 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
```

The resource usage of each partition's queue after task submission is as shown in the following figure:


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>
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:

The dev queue is mapped to the normal label and the default label used is normal. Tasks submitted to the dev queue run on nodes marked with normal.

Hadoop Best Practices

Last updated：2023-12-25 15:13:23

The Hadoop segment encompasses the distributed file system HDFS, the resource scheduling framework YARN, and the iterative computing framework MR. Tencent's version of Hadoop integrates Tencent Cloud Object Storage, enabling you to utilize object storage via the `hadoop fs` command line, thereby achieving computational storage separation. The best practices contained herein include the following aspects.

- Regardless of whether you are operating an HA cluster or a non-HA cluster, it is imperative that you **do not format the namenode**. Failure to heed this warning could result in data loss, for which Tencent Cloud assumes no responsibility if it is caused by your formatting of the namenode.
- By default, Tencent Cloud enables fair scheduling in YARN. However, you have the flexibility to modify the scheduler according to your specific requirements.

Using API to Analyze Data in HDFS and COS

Last updated：2023-12-25 15:15:35

The initial program one typically encounters when studying MapReduce is WordCount, which tallies the frequency of words in a given file. This section will elucidate how to construct a project and write a program independently, and how to utilize the compiled and packaged program to count data on HDFS and Tencent Cloud Object Storage (COS). The program used is fundamentally identical to the sample program provided by the Hadoop community.

1. Development Preparation

- Given that the task necessitates access to Tencent Cloud Object Storage (COS), it is imperative to initially create a storage bucket (Bucket) within COS.
- Ensure that you have activated Tencent Cloud and have established an EMR cluster. When creating the EMR cluster, it is necessary to select a cluster type that includes HDFS, and to enable object storage authorization on the basic configuration page.

2. Log into the EMR Server

Prior to performing related operations, it is necessary to log into any machine within the EMR cluster, preferably the Master node. As EMR is established on Tencent Cloud Server CVM, which operates on the Linux operating system, it is required to log into the CVM server when using EMR in command line mode.
Upon creating the EMR cluster, select Elastic MapReduce in the console. By clicking on **Master Node** under **Cluster Resources > Resource Management**, and selecting the resource ID of the Master node, you can access the cloud server console and locate the cloud server corresponding to EMR.
For methods on logging into CVM, please refer to Logging into Linux Instances. Here, we can opt to log in using WebShell. Click on **Login** on the right side of the corresponding cloud server to enter the login interface. The default username is root, and the password is the one input by the user when creating EMR.
Upon entering the correct information, you can access the command line interface of the EMR cluster. All Hadoop operations are under the Hadoop user. After logging into the EMR node, you are by default in the root user and need to switch to the Hadoop user. Use the following command to switch users and enter the Hadoop folder:

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

3. Data Preparation

You need to prepare the text files for statistics. There are two methods: **Storing data in the HDFS cluster** and **Storing data in COS**. Firstly, create a new txt file named test.txt locally, and add some English words:

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

Upload local data to the cloud server. You can use scp or sftp services to upload local files to the cloud server in the EMR cluster. Use the following in the local shell:

```
scp $localfile root@PublicIPAddress:$remotefolder
```

Here, $localfile is the path and name of your local file; root is the username of the CVM server; the public IP address can be viewed in the node information in the EMR console or in the cloud server console; $remotefolder is the path on the CVM server where you want to store the file.
Once the upload is complete, you can check in the EMR cluster command line whether there are corresponding files in the respective folder. In this case, the files have been uploaded to the `/usr/local/service/hadoop` path in the EMR cluster.

```
[hadoop@172 hadoop]$ ls –l
```

Data stored in HDFS

After uploading the data to the Tencent cloud server, you can copy the data to the HDFS cluster. Use the following command to copy the file to the Hadoop cluster:

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

After the copy is complete, use 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 under Hadoop, users can create it themselves using the following command:

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

For more Hadoop commands, see Common HDFS Operations.

Data is stored in COS.

There are two methods for storing data in COS: **direct upload from local via the COS console** and **upload via Hadoop commands**.

- For direct upload from local via the COS console, once the data file is uploaded, it can be viewed using 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
  ```

  Where $bucketname should be replaced with the name of your storage bucket plus the path.
- To upload via Hadoop commands, use the following instruction:

  ```
  [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 using Maven.

It is recommended to use Maven for project management. Maven is a project management tool that conveniently manages project dependencies. It can fetch jar packages through the configuration of the pom.xml file, eliminating the need for manual addition. First, download and install Maven, configure the Maven environment variables. If you are using an IDE, please set up the relevant Maven configurations within the IDE.

Create a new Maven project.

Navigate to the directory where you wish to create a new project from the command line, for instance, within `D://mavenWorkplace`, and input the following command to create a new Maven project:

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

Where `$yourgroupID` represents your package name; `$yourartifactID` is your project name; `maven-archetype-quickstart` indicates the creation of a Maven Java project. The project creation process requires the download of some files, please ensure a stable internet connection.
Upon successful creation, a project folder named `$yourartifactID` will be generated in the `D://mavenWorkplace` directory. The file structure within is as follows:

```
simple
        ---pom.xml           Core configuration, located at the root of the project
        ---src
              ---main
                    ---java           Directory for Java source code
              ---resources     Directory for Java configuration files
           ---test
                 ---java           Directory for test source code
                 ---resources     Directory for test configurations
```

The primary focus should be on the pom.xml file and the Java folder under main. The pom.xml file is primarily used for dependency and packaging configurations, while the Java folder is where your source code should be placed.
Firstly, add the Maven dependency in the 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>
```

Proceed to add packaging and compilation plugins in the 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 to create a new Java Class under src>main>java, input the Class name, here we use WordCount, and add 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);
    }
}
```

One can observe the presence of a Map function and a Reduce function.
If your Maven configuration is correct and the dependency packages have been successfully imported, then the entire project can be directly compiled. Navigate to the project directory in the local shell and execute the following command to package the entire project:

```
mvn package
```

During the execution, additional files may need to be downloaded until a 'build success' message appears, indicating successful packaging. Subsequently, you can locate the packaged jar file in the target folder under the project directory.
Utilize scp or sftp services to upload the packaged project files to the cloud server in the EMR cluster. Use the following in the local shell:

```
scp $jarpackage root@PublicIPAddress: /usr/local/service/hadoop
```

Herein, `$jarpackage` represents the path and name of your local jar package; 'root' is the username for the CVM server; the public IP address can be viewed in the node information in the EMR console or in the cloud server console. The upload is directed to the `/usr/local/service/hadoop` folder in the EMR cluster.

Calculating text files within HDFS.

Navigate to the `/usr/local/service/hadoop` directory, as in the data preparation stage. Submit the task using 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
The above is a complete command, where `/user/hadoop/test.txt` is the input file to be processed, and `/user/hadoop/ WordCount_output` is the output folder. Ensure that the `WordCount_output` folder has not been created prior to submitting the command, otherwise an error will occur.

Upon completion, view the execution output file using 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 statistical results in part-r-00000 using 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......
```

Statistical analysis of text files in COS

Navigate to the `/usr/local/service/hadoop` directory. Submit the task using 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 has been changed to `cosn:// $bucketname/test.txt` , where $bucketname is your bucket name plus path. The processing results are also output to COS. View the output file using the following command:

```
[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 results:

```
[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：2023-12-25 15:15:45

By default, Hadoop stores YARN job logs on HDFS. However, Tencent Cloud EMR also offers the option to store YARN job logs on external storage, COS.

Preparations

The EMR cluster needs to support COS. For more details, please refer to Using API to Analyze Data on HDFS/COS.

Instructions

1. Log in to the EMR Console, select the appropriate cluster from the cluster list, and click on **Cluster Services** to access the list of cluster services.
2. Navigate to the top right corner of the YARN service panel and select **Operations > Configuration Management**. Locate the `yarn-site.xml` configuration file and modify the following settings, then distribute to all nodes.

```
yarn.nodemanager.remote-app-log-dir=cosn://$bucketname/$logs_dirs
```

   Here, $bucketname refers to the name of your storage bucket, and $logs_dirs refers to the directory where you wish to transfer the yarn job logs.
3. Navigate to the top right corner of the HDFS service panel and select **Operations > Configuration Management**. Locate the `core-site.xml` file and add the following configuration items, then distribute to all nodes.

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

4. In **Cluster Services > HDFS > Role Management**, restart all `nodemanager/datanode` services within the cluster.
5. After running the `hive/spark` job, you can view the job logs stored on COS using the following command.

```
hdfs dfs -ls cosn://$bucketname/$logs_dirs
```

Spark Development Guide

Spark Environment Info

Last updated：2023-12-25 15:16:05

Tencent Cloud's EMR offers multi-version support for Spark 3.x and 2.x, with the following software environment information:

- Spark is installed on the master node by default.
- Upon logging into the machine, use the command `su hadoop` to switch to the Hadoop user.
- The Spark software path is located under `/usr/local/service/spark` .
- The relevant log path is located under `/data/emr` .

For more detailed information, please refer to the community documentation. This primarily introduces operations related to accessing Tencent Cloud Object Storage based on Spark.

Using Spark to Analyze Data in COS

Last updated：2023-12-25 15:16:17

As an advanced open-source project of Apache, Spark is a swift, universal engine for large-scale data processing. It shares similarities with Hadoop's MapReduce computational framework. However, compared to MapReduce, Spark, with its scalable, memory-based computing features and the ability to directly read and write data in any format on Hadoop, is more efficient in batch processing and has lower latency. In fact, Spark has become a unified platform for lightweight, rapid big data processing. Various applications, such as real-time stream processing, machine learning, interactive queries, and more, can be established on different storage and operating systems via Spark.
Spark is a parallel computing framework for big data based on in-memory computing. By leveraging in-memory computation, Spark enhances the real-time data processing capabilities in a big data environment, while ensuring high fault tolerance and scalability. It allows users to deploy Spark on a multitude of inexpensive hardware, forming a cluster.
This tutorial demonstrates the submission of a word count task, which necessitates the prior upload of the file to be counted within the cluster.

1. Development Preparation

- As the task requires access to Tencent Cloud Object Storage (COS), it is necessary to first create a storage bucket (Bucket) in COS.
- Ensure that you have activated Tencent Cloud and created an EMR cluster. When creating the EMR cluster, you need to select the Spark component in the software configuration interface, and enable the authorization of object storage in **Instance Information > Basic Configuration**.

2. Creating a Project Using Maven

In this demonstration, we will not use the system's built-in demo program. Instead, we will create our own project, compile and package it, and then upload it to the EMR cluster for execution. It is recommended to use Maven to manage your project. Maven is a project management tool that can help you conveniently manage project dependencies. That is, it can obtain jar packages through the configuration of the pom.xml file, eliminating the need for manual addition.
First, download and install Maven, and configure Maven's environment variables. If you are using an IDE, please set the Maven related configuration in the IDE.

Create a new Maven project

Navigate to the directory where you want to create a new project in your local shell, for example, in `D://mavenWorkplace`, and enter the following command to create a new Maven project:

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

Where $yourgroupID is your package name. $yourartifactID is your project name, and maven-archetype-quickstart indicates the creation of a Maven Java project. Some files need to be downloaded during the project creation process, so please ensure a stable internet connection.
Upon successful creation, a project folder named $yourartifactID will be generated in the `D://mavenWorkplace` directory. The file structure is as follows:

```
simple
    ---pom.xml           Core configuration, located at the root of the project
    ---src
        ---main
            ---java           Java source code directory
            ---resources      Directory for Java configuration files
        ---test
             ---java           Test source code directory
             ---resources      Test configuration directory
```

Our primary focus lies on the pom.xml file and the Java folder under main. The pom.xml file is primarily used for dependency and packaging configurations, while the Java folder houses your source code.
Firstly, add the Maven dependency in the pom.xml:

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

Proceed to add packaging and compilation plugins in the 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 to create a new Java Class under src>main>Java, input your Class name, here we use WordCountOnCos, 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 configuration is correct and the dependency packages have been successfully imported, then the entire project should be error-free and ready for compilation. Enter the project directory in the local command line mode and execute the following command to package the entire project:

```
mvn package
```

During the execution, additional files may need to be downloaded until a 'build success' message appears, indicating successful packaging. Subsequently, you can locate the packaged jar file in the target folder under the project directory.

Data Preparation

Initially, the compressed jar package needs to be uploaded to the EMR cluster using scp or sftp tools. Run the following command in the local command line mode:

```
scp $localfile root@PublicIPAddress:$remotefolder
```

Herein, $localfile represents the path and name of your local file; 'root' is the username of the CVM server; the public IP can be viewed in the node information of the EMR console or in the cloud server console; $remotefolder is the path on the CVM server where you wish to store the file. Upon completion of the upload, you can check in the EMR command line whether the corresponding folder contains the respective file.
The files to be processed need to be uploaded to COS in advance. If the files are local, they can be directly uploaded via the COS Console. If the files are on the EMR cluster, they can be uploaded using Hadoop commands. The directive is as follows:

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

Here, $testfile represents the full path and name of the file to be analyzed, and $bucketname is the name of your storage bucket. After the upload is complete, you can check in the COS console to see if the file is already in COS.

Execute Sample

Initially, it is necessary to log into any machine in the EMR cluster, preferably the Master node. For methods on how to log into EMR, please refer to Logging into Linux Instances. Here, we can opt to use WebShell for login. Click on the login button on the right side of the corresponding cloud server to enter the login interface. The default username is 'root', and the password is the one entered by the user when creating EMR. After entering correctly, you can access the command line interface.
In the EMR command line, use the following command to switch to the Hadoop user:

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

Then, navigate to the folder where your jar package is stored and execute the following command:

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

Here, $WordCountOnCOS is the name of your Java Class, $packagename is the name of the jar package generated in your new Maven project, $bucketname is the name and path of your storage bucket, and $testfile is the name of the file you want to analyze. The final output file is located in the 'output' folder, **this folder must not be created in advance, otherwise the operation will fail.**
Upon successful execution, the results of the wordcount can be viewed in the designated storage 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：2023-12-25 15:16:30

This section primarily focuses on executing word count tasks through Spark Python.

Development Preparations

- As the task requires access to Tencent Cloud Object Storage (COS), it is necessary to first create a storage bucket (Bucket) in COS.
- Ensure that you have activated Tencent Cloud and have created an EMR cluster. When creating the EMR cluster, you need to select the Spark component on the software configuration page, and enable object storage authorization on the basic configuration page.

Data Preparation

The files that need to be processed must be uploaded to COS in advance. If the files are local, they can be directly uploaded through the COS console. If the files are on the EMR cluster, the Hadoop command can be used for uploading. The command is as follows:

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

Where $testfile is the full path and name of the file to be counted, and $bucketname is the name of your storage bucket. After the upload is complete, you can check whether the file is already in COS.

Execute Sample

Initially, it is necessary to log into any machine in the EMR cluster, preferably the Master node. For methods on how to log into EMR, please refer to Logging into Linux Instances. Here, we can opt to use WebShell for login. Click on the login button on the right side of the corresponding cloud server to enter the login interface. The default username is 'root', and the password is the one entered by the user when creating EMR. After entering correctly, you can access the command line interface.
In the EMR command line, first use the following command to switch to the Hadoop user, and enter the Spark installation directory `/usr/local/service/spark`:

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

Create a new 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 using the following command:

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

Where $bucketname is the name of your COS bucket, $yourtestfile is the full path and name of your test file in the bucket. $output is your output folder. **$output is an uncreated folder, if this folder already exists before executing the command, it will cause the program to fail**.
Upon successful execution, the program runs automatically, and the output file can be viewed in the target storage 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
```

The final results can also be viewed using 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)
```

Similarly, the results can be output to HDFS by simply changing the output location in the command, as shown below:

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

Where `/user/hadoop/` is the path in HDFS, if it does not exist, users can create it themselves.
Upon completion of the task, the Spark operation logs can be viewed using the following command:

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

Where $yourId should be replaced with your task ID. The task ID can be viewed on the YARN WebUI.

SparkSQL Tutorial

Last updated：2023-12-25 15:16:40

Spark introduces a programming module for structured data processing known as Spark SQL. It offers a programming abstraction referred to as DataFrame, which can serve as a distributed SQL query engine.

1. Development Preparation

Ensure that you have activated Tencent Cloud and have established an EMR cluster. During the creation of the EMR cluster, the Spark component must be selected in the software configuration interface.

2. Utilizing the SparkSQL Interactive Console

Before utilizing SparkSQL, please log into the Master node of the EMR cluster. For the method of logging into EMR, please refer to Log into Linux Instance. Here, we can opt to log in using WebShell. Click on the login on the right side of the corresponding cloud server to enter the login interface. The default username is root, and the password is the one input by the user when creating EMR. Upon correct entry, you can access the EMR command line interface.
In the EMR command line, first use the following command to switch to the Hadoop user and enter the directory `/usr/local/service/spark`:

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

You can enter the SparkSQL interactive console through the following command:

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

The --master represents your master URL, --num-executors represents the number of executors, and --executor-memory represents the storage capacity of the executor. These parameters can be modified according to your actual situation. You can also start or stop a SparkSQL thrift server through `sbin/start-thriftserver.sh` or `sbin/stop-thriftserver.sh`.
Below are some basic operations of SparkSQL:

- Create a new database and view it:

```
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 newly created database and view it:

```
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 command line parameter usage tutorials, please refer to the community documentation.

3. Create a project using Maven

Firstly, download and install Maven, configure the Maven environment variables. If you are using an IDE, please set up the relevant Maven configurations within the IDE.

Create a new Maven project

Navigate to the directory where you wish to create a new project from the command line, for instance, within `D://mavenWorkplace`, and input the following command to create a new Maven project:

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

Wherein $yourgroupID is your package name. $yourartifactID is your project name, and maven−archetype−quickstart signifies the creation of a Maven Java project. Some files will need to be downloaded during the project creation process, so please ensure a stable internet connection.
Upon successful creation, a project folder named $yourartifactID will be generated in the `D://mavenWorkplace` directory. The file structure is as follows:

```
simple
        ---pom.xml           Core configuration, located at the root of the project
        ---src
              ---main
                    ---java           Directory for Java source code
              ---resources      Directory for Java configuration files
           ---test
                 ---java          Directory for test source code
                 ---resources     Directory for test configurations
```

Primarily, our focus lies on the pom.xml file and the Java folder under main. The pom.xml file is primarily used for dependency and packaging configurations, while the Java folder houses your source code.

Incorporate Hadoop dependencies and sample code

Identify a node within the cluster to ascertain the version of the dependent Spark:

```
cd /usr/local/service/spark/jars && ll

-rw-r--r-- 1 hadoop hadoop 31870390 Jun 20 16:05 hudi-spark3.2-bundle_2.12-0.11.0.jar
-rw-r--r-- 1 hadoop hadoop 16094752 Jun 20 16:05 kudu-spark3_2.12-1.15.0.jar
-rw-r--r-- 1 hadoop hadoop   186484 Jun 20 16:05 spark-avro_2.12-3.2.1.jar
-rw-r--r-- 1 hadoop hadoop 11645730 Jun 20 16:05 spark-catalyst_2.12-3.2.1.jar
-rw-r--r-- 1 hadoop hadoop 10841257 Jun 20 16:05 spark-core_2.12-3.2.1.jar
-rw-r--r-- 1 hadoop hadoop   431111 Jun 20 16:05 spark-graphx_2.12-3.2.1.jar
-rw-r--r-- 1 hadoop hadoop    11983 Jun 20 16:05 spark-hadoop-cloud_2.12-3.2.1.jar
-rw-r--r-- 1 hadoop hadoop   700945 Jun 20 16:05 spark-hive_2.12-3.2.1.jar
-rw-r--r-- 1 hadoop hadoop   570134 Jun 20 16:05 spark-hive-thriftserver_2.12-3.2.1.jar
-rw-r--r-- 1 hadoop hadoop   453567 Jun 20 16:05 spark-kubernetes_2.12-3.2.1.jar
-rw-r--r-- 1 hadoop hadoop    61811 Jun 20 16:05 spark-kvstore_2.12-3.2.1.jar
-rw-r--r-- 1 hadoop hadoop    76516 Jun 20 16:05 spark-launcher_2.12-3.2.1.jar
-rw-r--r-- 1 hadoop hadoop  6137585 Jun 20 16:05 spark-mllib_2.12-3.2.1.jar
-rw-r--r-- 1 hadoop hadoop   116164 Jun 20 16:05 spark-mllib-local_2.12-3.2.1.jar
-rw-r--r-- 1 hadoop hadoop  2412460 Jun 20 16:05 spark-network-common_2.12-3.2.1.jar
-rw-r--r-- 1 hadoop hadoop   160623 Jun 20 16:05 spark-network-shuffle_2.12-3.2.1.jar
-rw-r--r-- 1 hadoop hadoop    52799 Jun 20 16:05 spark-repl_2.12-3.2.1.jar
-rw-r--r-- 1 hadoop hadoop    30670 Jun 20 16:05 spark-sketch_2.12-3.2.1.jar
```

```
-rw-r--r-- 1 hadoop hadoop  8341311 Jun 20 16:05 spark-sql_2.12-3.2.1.jar
-rw-r--r-- 1 hadoop hadoop  1139792 Jun 20 16:05 spark-streaming_2.12-3.2.1.jar
-rw-r--r-- 1 hadoop hadoop    15156 Jun 20 16:05 spark-tags_2.12-3.2.1.jar
-rw-r--r-- 1 hadoop hadoop     9913 Jun 20 16:05 spark-tags_2.12-3.2.1-tests.jar
-rw-r--r-- 1 hadoop hadoop    51765 Jun 20 16:05 spark-unsafe_2.12-3.2.1.jar
-rw-r--r-- 1 hadoop hadoop   349957 Jun 20 16:05 spark-yarn_2.12-3.2.1.jar
```

Initially, incorporate Maven dependencies into the pom.xml file (modify the corresponding values of scala.version and spark.version based on the results from the above 'll' command):

```
    <properties>
        <scala.version>2.12</scala.version>
        <spark.version>3.2.1</spark.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.apache.spark</groupId>
            <artifactId>spark-core_${scala.version}</artifactId>
            <version>${spark.version}</version>
        </dependency>
        <dependency>
            <groupId>org.apache.spark</groupId>
            <artifactId>spark-sql_${scala.version}</artifactId>
            <version>${spark.version}</version>
        </dependency>
    </dependencies>
```

Proceed to incorporate packaging and compilation plugins into 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>
```

The complete pom.xml file is presented as follows:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>$yourgroupID </groupId>
    <artifactId>$yourartifactID </artifactId>
    <version>1.0-SNAPSHOT</version>

    <dependencies>
        <dependency>
            <groupId>org.apache.spark</groupId>
            <artifactId>spark-core_${scala.version}</artifactId>
            <version>${spark.version}</version>
        </dependency>
        <dependency>
            <groupId>org.apache.spark</groupId>
            <artifactId>spark-sql_${scala.version}</artifactId>
            <version>${spark.version}</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>
</project>
```

Note
Alter the $yourgroupID and $yourartifactID to reflect your personal settings.

Next, incorporate a sample code. Create a new Java Class named Demo.java under the main>Java folder, and insert the following code into it:

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

/**
 * 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]);
    }
}
```

Compile the code and package it for upload.

Navigate to the project directory using the local command line and execute the following command to compile and package the project:

```
mvn package
```

Upon seeing 'build success', the operation is successful. The packaged file can be found in the target folder within the project directory. Use scp or sftp tools to upload the packaged file to the EMR cluster. Run the following in the local command line mode:

```
scp $localfile root@PublicIPAddress:$remotefolder
```

In this context, $localfile refers to the path and name of your local file, 'root' is the username of the CVM server, and the public IP can be viewed in the node information of the EMR console or in the cloud server console. $remotefolder is the path on the CVM server where you wish to store the file. Once the upload is complete, you can check in the EMR cluster command line to see if the corresponding file is in the respective folder.

4. Prepare the data and run the sample.

Utilize SparkSQL to manipulate data stored on HDFS. First, upload the data to HDFS. Here, we use the built-in file 'people.json', located in the path `/usr/local/service/spark/examples/src/main/resources/` . Use the following command to upload this file to HDFS:

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

Users can also choose a different test file. Here, `/user/hadoop/` is a folder under HDFS. If there are no users, you can create one yourself.
Execute the sample, please first log in to the master node of the EMR cluster and switch to the Hadoop user, as shown in Step 2 Use SparkSQL Interactive Console. Execute the sample using the following command:

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

In this context, the −−class parameter indicates the entry class to be executed, which in this case is 'Demo', the name of the Java Class created in the addition of Hadoop dependencies and sample code. −−master is the primary URL of the cluster, $yourjarpackage is the name of your packaged bundle, and $output is the result output folder (**$output is an uncreated folder, if this folder already exists before executing the command, it will cause the program to fail**).
Upon successful execution, you can view the results at `/user/hadoop/$output` :

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

For more parameters of spark−submit, enter the following command in the command line for viewing, or please refer to the official documentation.

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

Integrating Spark Streaming with Ckafka

Last updated: 2023-12-25 15:16:55

Leveraging Tencent Cloud's EMR service, you can effortlessly integrate with Tencent Cloud's Ckafka service to implement the following stream-based applications:

- Streamlined processing of log information.
- Streamlined processing of user behavior logs.
- Collection and processing of alert information.
- Messaging system.

1. Development Preparation

- As the task requires access to Tencent Cloud's message queue CKafka, it is necessary to first create a CKafka instance, as detailed in Message Queue CKafka.
- Ensure that you have activated Tencent Cloud and have created an EMR cluster. When creating the EMR cluster, select the Spark component in the software configuration interface.

2. Utilizing the Kafka Toolkit in the EMR Cluster.

Initially, it is necessary to view the internal network IP and port number of CKafka. Log into the Message Queue CKafka console, select the CKafka instance you wish to use, and view its internal network IP as $kafkaIP in the basic messages. The port number is generally set to 9092 by default. Create a new topic named spark_streaming_test in the topic management interface.
Log into any machine in the EMR cluster, preferably the Master node. For the method of logging into EMR, please refer to Logging into Linux Instances. Here, you may choose to log in using WebShell. Click on the login on the right side of the corresponding cloud server to enter the login interface. The username is set to root by default, and the password is the one input by the user when creating EMR. After entering correctly, you can access the command line interface.
In the EMR command line, first use the following command to switch to the Hadoop user and enter the directory `/usr/local/service/spark`:

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

Download the installation package from the Kafka official website, ensuring to select the appropriate version. For specifics, refer to EMR Version Kafka and Spark Version Description. The Kafka client version is highly compatible with Tencent Cloud CKafka, so simply install the corresponding Kafka client version. This article uses kafka_2.10-0.10.2.0 as an example. Please ensure to use the correct version in commands and code, decompress the package, and move the decompressed 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/
```

Upon completion of decompression, the Kafka tool can be used directly. The `telnet` command can be employed to test whether the EMR cluster can connect to the CKafka instance:

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

Wherein, $kafkaIP represents the internal network IP address of the CKafka instance you have created.
You can now conduct a simple test of the Kafka toolkit. Log into the EMR cluster using two WebShells, switch to the Hadoop user, and enter 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 messages to it:

```
[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 another terminal and retrieve the data as a consumer:

```
[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. Interface CKafka service using SparkStreaming

On the consumer end, we utilize Spark Streaming to continuously pull data from CKafka for word frequency statistics, that is, to perform WordCount tasks on streaming data. On the producer end, we also continuously generate data through a program to constantly feed CKafka.
Firstly, download and install Maven, configure the Maven environment variables. If you are using an IDE, please set up the relevant Maven configurations within the IDE.

Create a Spark Streaming consumer project

Navigate to the directory where you wish to create a new project in your local command line, for instance, `D://mavenWorkplace` , and input the following command to create a new Maven project:

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

Wherein $yourgroupID is your package name. $yourartifactID is your project name, and maven-archetype-quickstart indicates the creation of a Maven Java project. Some files need to be downloaded during the project creation process, please ensure a stable internet connection.
Upon successful creation, a project folder named $yourartifactID will be generated in the `D://mavenWorkplace` directory. The file structure is as follows:

```
simple
        ---pom.xml           Core configuration, located at the root of the project
        ---src
             ---main
                  ---java          Directory for Java source code
             ---resources     Directory for Java configuration files
           ---test
                ---java          Directory for test source code
                ---resources     Directory for test configurations
```

Our primary focus lies on the pom.xml file and the Java folder under main. The pom.xml file is primarily used for dependency and packaging configurations, while the Java folder houses your source code.
Identify a node within the cluster to ascertain the version of the dependent Spark:

```
cd /usr/local/service/spark/jars && ll

-rw-r--r-- 1 hadoop hadoop 31870390 Jun 20 16:05 hudi-spark3.2-bundle_2.12-0.11.0.jar
-rw-r--r-- 1 hadoop hadoop 16094752 Jun 20 16:05 kudu-spark3_2.12-1.15.0.jar
-rw-r--r-- 1 hadoop hadoop   186484 Jun 20 16:05 spark-avro_2.12-3.2.1.jar
-rw-r--r-- 1 hadoop hadoop 11645730 Jun 20 16:05 spark-catalyst_2.12-3.2.1.jar
-rw-r--r-- 1 hadoop hadoop 10841257 Jun 20 16:05 spark-core_2.12-3.2.1.jar
-rw-r--r-- 1 hadoop hadoop   431111 Jun 20 16:05 spark-graphx_2.12-3.2.1.jar
-rw-r--r-- 1 hadoop hadoop    11983 Jun 20 16:05 spark-hadoop-cloud_2.12-3.2.1.jar
-rw-r--r-- 1 hadoop hadoop   700945 Jun 20 16:05 spark-hive_2.12-3.2.1.jar
```

```
-rw-r--r-- 1 hadoop hadoop   570134 Jun 20 16:05 spark-hive-thriftserver_2.12-3.2.1.jar
-rw-r--r-- 1 hadoop hadoop   453567 Jun 20 16:05 spark-kubernetes_2.12-3.2.1.jar
-rw-r--r-- 1 hadoop hadoop    61811 Jun 20 16:05 spark-kvstore_2.12-3.2.1.jar
-rw-r--r-- 1 hadoop hadoop    76516 Jun 20 16:05 spark-launcher_2.12-3.2.1.jar
-rw-r--r-- 1 hadoop hadoop  6137585 Jun 20 16:05 spark-mllib_2.12-3.2.1.jar
-rw-r--r-- 1 hadoop hadoop   116164 Jun 20 16:05 spark-mllib-local_2.12-3.2.1.jar
-rw-r--r-- 1 hadoop hadoop  2412460 Jun 20 16:05 spark-network-common_2.12-3.2.1.jar
-rw-r--r-- 1 hadoop hadoop   160623 Jun 20 16:05 spark-network-shuffle_2.12-3.2.1.jar
-rw-r--r-- 1 hadoop hadoop    52799 Jun 20 16:05 spark-repl_2.12-3.2.1.jar
-rw-r--r-- 1 hadoop hadoop    30670 Jun 20 16:05 spark-sketch_2.12-3.2.1.jar
-rw-r--r-- 1 hadoop hadoop  8341311 Jun 20 16:05 spark-sql_2.12-3.2.1.jar
-rw-r--r-- 1 hadoop hadoop  1139792 Jun 20 16:05 spark-streaming_2.12-3.2.1.jar
-rw-r--r-- 1 hadoop hadoop    15156 Jun 20 16:05 spark-tags_2.12-3.2.1.jar
-rw-r--r-- 1 hadoop hadoop     9913 Jun 20 16:05 spark-tags_2.12-3.2.1-tests.jar
-rw-r--r-- 1 hadoop hadoop    51765 Jun 20 16:05 spark-unsafe_2.12-3.2.1.jar
-rw-r--r-- 1 hadoop hadoop   349957 Jun 20 16:05 spark-yarn_2.12-3.2.1.jar
```

Firstly, incorporate Maven dependencies into the pom.xml file:

```
    <properties>
        <scala.version>2.12</scala.version>
        <spark.version>3.2.1</spark.version>
    </properties>
<dependencies>
        <dependency>
            <groupId>org.apache.spark</groupId>
            <artifactId>spark-core_${scala.version}</artifactId>
            <version>${spark.version}</version>
        </dependency>
        <dependency>
            <groupId>org.apache.spark</groupId>
            <artifactId>spark-streaming_${scala.version}</artifactId>
            <version>${spark.version}</version>
        </dependency>
        <dependency>
            <groupId>org.apache.spark</groupId>
            <artifactId>spark-streaming-kafka-0-10_${scala.version}</artifactId>
            <version>${spark.version}</version>
        </dependency>
</dependencies>
```

Proceed to incorporate packaging and compilation plugins into 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
Alter the $yourgroupID and $yourartifactID to reflect your personal settings.

Next, incorporate the sample code. Create a new Java Class named KafkaTest.java under the main>Java folder, and integrate the following code into 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 = "$kafkaIP:9092";
        String topics = "spark_streaming_test1";  // Subscribed topics, multiple topics separated by ','
        int durationSeconds = 60;  // Interval duration
        SparkConf conf = new SparkConf().setAppName("spark streaming word count");
        JavaSparkContext sc = new JavaSparkContext(conf);
        JavaStreamingContext ssc = new JavaStreamingContext(sc, Durations.seconds(durationSeconds));
        Collection<String> topicsSet = new HashSet<>(Arrays.asList(topics.split(",")));
        //Parameters related to Kafka
        Map<String, Object> kafkaParams = new HashMap<>();
        kafkaParams.put("metadata.broker.list", brokers) ;
        kafkaParams.put("bootstrap.servers", brokers);
        kafkaParams.put("group.id", "group1");
        kafkaParams.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");
        kafkaParams.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
        kafkaParams.put("value.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
        //Establishing connection
        JavaInputDStream<ConsumerRecord<Object,Object>> lines = KafkaUtils.createDirectStream(
                ssc,
                LocationStrategies.PreferConsistent(),
                ConsumerStrategies.Subscribe(topicsSet, kafkaParams)
        );
        //Logic for word count
        JavaPairDStream<String, Integer> counts = lines
                .flatMap(x -> Arrays.asList(x.value().toString().split(" ")).iterator())
                .mapToPair(x -> new Tuple2<String, Integer>(x, 1))
                .reduceByKey((x, y) -> x + y);
        // Storing results
        counts.dstream().saveAsTextFiles("$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 internal IP of the CKafka instance found in the second step.
- The 'topics' variable should be set to the name of the topic you created, in this case, 'spark_streaming_test1'.
- 'durationSeconds' refers to the time interval at which the program consumes data from CKafka, which is set to 60 seconds in this case.
- '$hdfsPath' refers to the path in HDFS where the results will be output.

Navigate to the project directory using the local command line and execute the following command to compile and package the project:

```
mvn package
```

A display of 'build success' indicates a successful operation. The packaged file can be found in the target folder within the project directory.
Utilize tools such as scp or sftp to upload the packaged file to the EMR cluster, ensuring to include the jar package that was bundled with the dependencies:

```
scp $localfile root@PublicIPAddress:$remotefolder
```

In this context, $localfile refers to the path and name of your local file, 'root' is the username of the CVM server, and the public IP can be viewed in the node information of the EMR console or in the cloud server console. $remotefolder is the path on the CVM server where you wish to store the file. Once the upload is complete, you can check in the EMR cluster command line to see if the corresponding file is in the respective folder.

Establish a Spark Streaming Producer Project

Navigate to the directory where you wish to create a new project in your local command line, for instance, `D://mavenWorkplace` , and input the following command to create a new Maven project:

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

Firstly, incorporate Maven dependencies into 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>
```

Proceed to incorporate packaging and compilation plugins into 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
Alter the $yourgroupID and $yourartifactID to reflect your personal settings.

Next, add the sample code. Create a new Java Class named SendData.java in the main>Java folder, and incorporate the following code into 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.StringSerializer");
        props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");
        //Producer dispatches message
        String topic = "spark_streaming_test1";
        org.apache.kafka.clients.producer.Producer<String, String> procuder = new KafkaProducer<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);
                procuder.send(msg);
            }

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

Alter the $kafkaIP to your CKafka's private IP address.
This program dispatches 10 messages to CKafka every 10 seconds, ranging from value_0 to value_9, with the starting sequence being random. The parameter information in the program refers to the consumer program.
Navigate to the project directory using the local command line and execute the following command to compile and package the project:

```
mvn package
```

A display of 'build success' indicates a successful operation. The packaged file can be found in the target folder within the project directory.
Utilize tools such as scp or sftp to upload the packaged file to the EMR cluster, ensuring to include the jar package that was bundled with the dependencies:

```
scp $localfile root@PublicIPAddress:$remotefolder
```

Utilize the program to consume CKafka's data

Access the Web Shell of the EMR cluster separately using two interfaces.
First Interface: Log into the Master node of the EMR cluster and switch to the Hadoop user as shown in Section 2, execute the sample using the following command:

```
[hadoop@172 ~]$ bin/spark-submit --class KafkaTest --master yarn --deploy-mode cluster $consumerpackage
```

The parameters are as follows:

- The −−class parameter signifies the entry class to be executed, which in this case is KafkaTest.
- −−master represents the primary URL of the cluster.
- $ consumerpackage refers to the package name after your consumer has been packaged.

Once the program begins execution, it will continuously run on the yarn cluster. The status of the program can be viewed using the following command:

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

Second Interface: Log into the EMR's Web Shell and run the producer program, enabling Spark Streaming to consume data from it.

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

Here, $producerpackage refers to the package name after your producer has been packaged. After a period of waiting, the wordcount results will be output in the specified HDFS folder. You can check the results of Spark Streaming consuming CKafka data in HDFS:

```
[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-1530793500000.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, it is necessary to exit the KafkaTest program within the yarn cluster:

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

Here, $Application−Id refers to the ID found using the `yarn application –list` command.
For more information related to Kafka, please refer to the official documentation.

Practices on Dynamic Scheduling of Spark Resources

Last updated：2024-01-12 11:22:19

Development Preparations

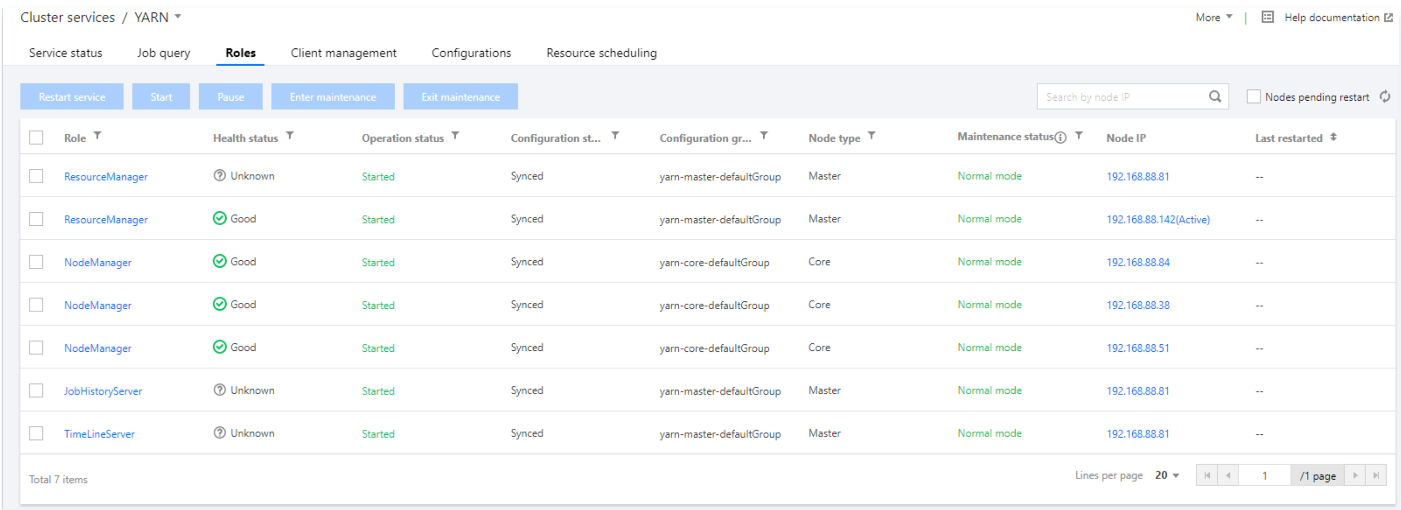
Ensure that you have activated Tencent Cloud and have established an EMR cluster. During the creation of the EMR cluster, it is necessary to select the spark_hadoop component in the software configuration interface.
Spark is installed under the path `/usr/local/service` on the EMR cloud server (`/usr/local/service/spark`).

Copying the JAR Package

It is required to copy `spark-<version>-yarn-shuffle.jar` to the directory path `/usr/local/service/hadoop/share/hadoop/yarn/lib` on each node of the cluster.

Method One: SSH Console Operation

1. Within the **Cluster Services > YARN** component, select **Operation > Role Management** to determine the IP of the NodeManager.

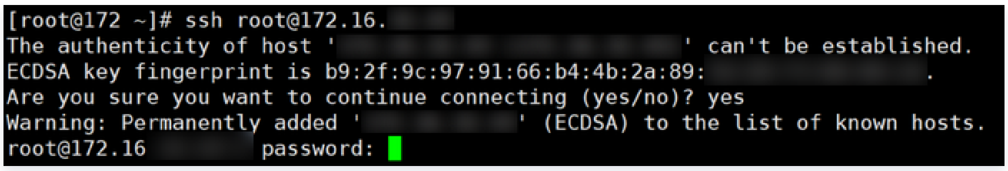


Role	Health status	Operation status	Configuration st...	Configuration gr...	Node type	Maintenance status	Node IP	Last restarted
ResourceManager	Unknown	Started	Synced	yarn-master-defaultGroup	Master	Normal mode	192.168.88.81	--
ResourceManager	Good	Started	Synced	yarn-master-defaultGroup	Master	Normal mode	192.168.88.142(Active)	--
NodeManager	Good	Started	Synced	yarn-core-defaultGroup	Core	Normal mode	192.168.88.84	--
NodeManager	Good	Started	Synced	yarn-core-defaultGroup	Core	Normal mode	192.168.88.38	--
NodeManager	Good	Started	Synced	yarn-core-defaultGroup	Core	Normal mode	192.168.88.51	--
JobHistoryServer	Unknown	Started	Synced	yarn-master-defaultGroup	Master	Normal mode	192.168.88.81	--
TimeLineServer	Unknown	Started	Synced	yarn-master-defaultGroup	Master	Normal mode	192.168.88.81	--

Total 7 items　Lines per page 20　1 /1 page

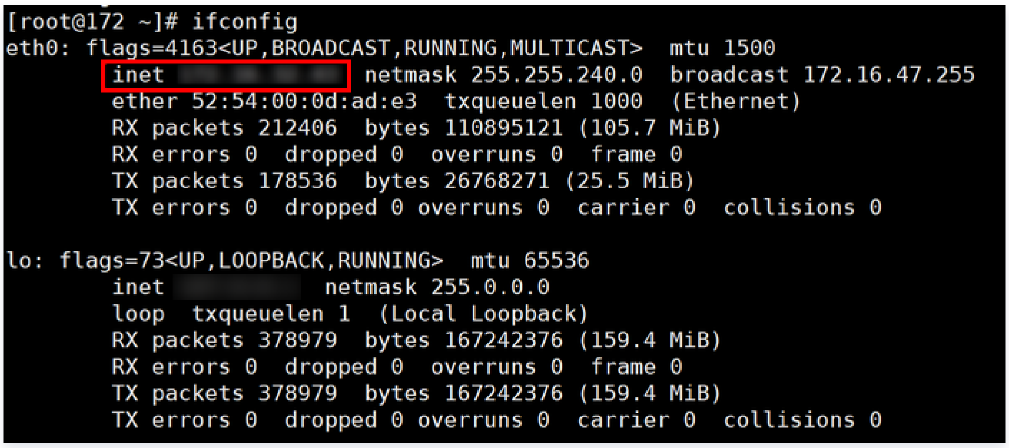
2. Sequentially log into each node where the NodeManager is located.

- Initially, it is necessary to log into any machine within the EMR cluster, preferably the Master node. Please refer to Logging into Linux Instances for the method of logging into EMR. Here, we can use XShell to log into the Master node.
- Utilize SSH to log into other nodes where the NodeManager is located. The command is `ssh $user@$ip` , where $user is the username for login and $ip is the IP of the remote node (i.e., the IP address determined in step 1).



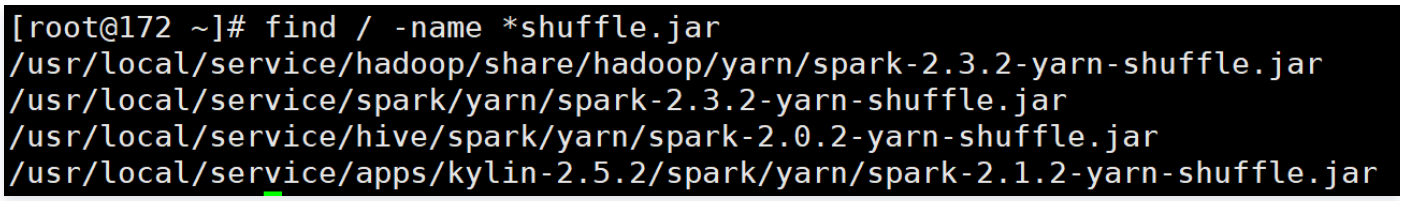
```
[root@172 ~]# ssh root@172.16.[redacted]
The authenticity of host '[redacted]' can't be established.
ECDSA key fingerprint is b9:2f:9c:97:91:66:b4:4b:2a:89:[redacted].
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '[redacted]' (ECDSA) to the list of known hosts.
root@172.16[redacted] password:
```

- Confirm successful switch.



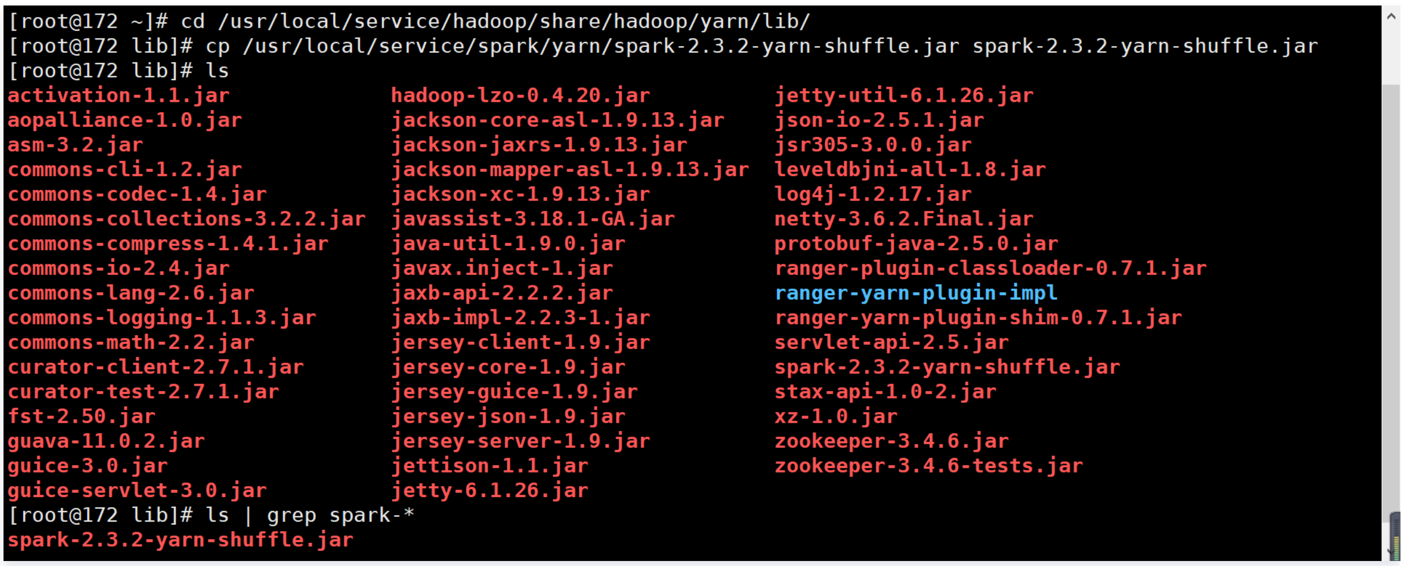
```
[root@172 ~]# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet [redacted]  netmask 255.255.240.0  broadcast 172.16.47.255
        ether 52:54:00:0d:ad:e3  txqueuelen 1000  (Ethernet)
        RX packets 212406  bytes 110895121 (105.7 MiB)
        RX errors 0  dropped 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 [redacted]  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 `spark-<version>-yarn-shuffle.jar` file path.



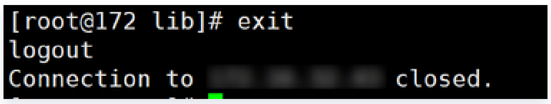
```
[root@172 ~]# find / -name *shuffle.jar
/usr/local/service/hadoop/share/hadoop/yarn/spark-2.3.2-yarn-shuffle.jar
/usr/local/service/spark/yarn/spark-2.3.2-yarn-shuffle.jar
/usr/local/service/hive/spark/yarn/spark-2.0.2-yarn-shuffle.jar
/usr/local/service/apps/kylin-2.5.2/spark/yarn/spark-2.1.2-yarn-shuffle.jar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

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 the remaining nodes.



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

Method Two: Batch Deployment Script

Firstly, it is necessary to log into any machine in the EMR cluster, preferably the Master node. For the method of logging into EMR, please refer to Logging into Linux Instances . Here, we can use XShell to log into the Master node.
Compose the following Shell script for batch file transfer. When there are many cluster nodes, to avoid multiple password entries, the sshpass tool can be used for transmission. The advantage of sshpass is that it can transmit without a password to avoid multiple entries, but its disadvantage is that the password is in plaintext and easily exposed, which can be found using the history command.

1. Install sshpass for password-free access.

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

Compose the following script:

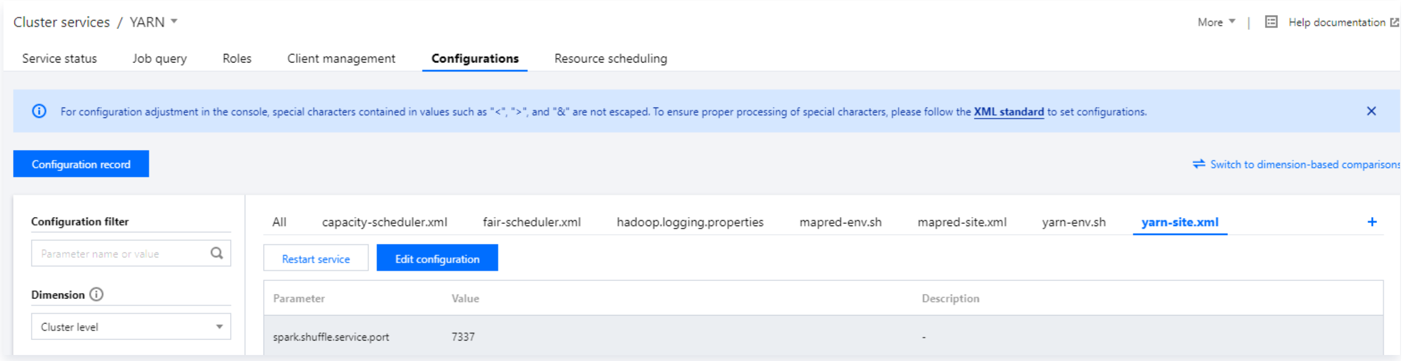
```
#!/bin/bash
nodes=(ip1 ip2 … ipn) # List of IP addresses for each node in the cluster, separated by spaces
len=${#nodes[@]}
password=<your password>
file=" spark-2.3.2-yarn-shuffle.jar "
source_dir="/usr/local/service/spark/yarn"
target_dir="/usr/local/service/hadoop/share/hadoop/yarn/lib"
echo $len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

2. Non-password-free.
   Compose the following script:

```
#!/bin/bash
nodes=(ip1 ip2 … ipn) # List of IP addresses for each node 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
```

Modify Yarn Configuration

1. In the **Cluster Services > YARN Component**, select **Operation > Configuration Management.** Select the configuration file yarn-site.xml, choose "Cluster Dimension" for the **Dimension Scope** (modifications to the configuration items in the cluster dimension will be applied to all nodes), then click on **Edit Configuration.**


Cluster services / YARN

Service status | Job query | Roles | Client management | **Configurations** | Resource scheduling

For configuration adjustment in the console, special characters contained in values such as "<", ">", and "&" are not escaped. To ensure proper processing of special characters, please follow the **XML standard** to set configurations.

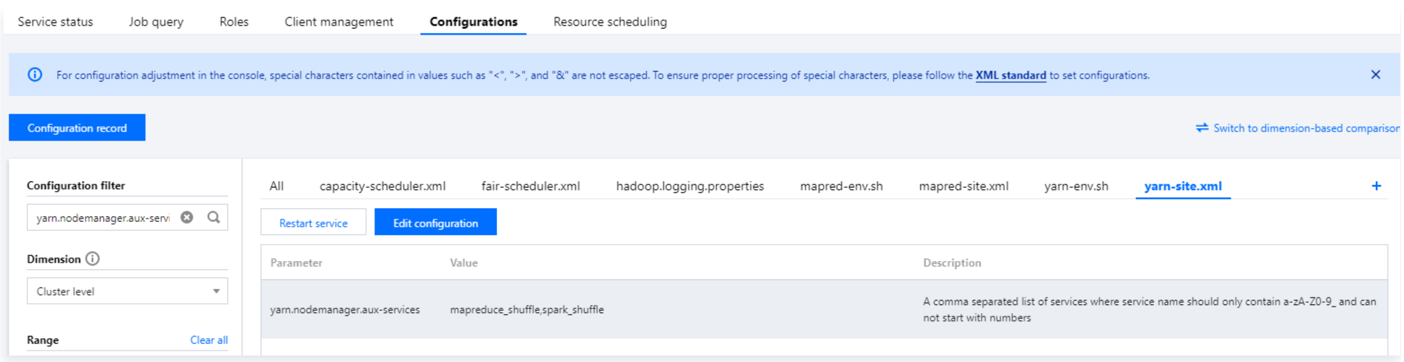
Configuration filter: Parameter name or value

Dimension: Cluster level

All | capacity-scheduler.xml | fair-scheduler.xml | hadoop.logging.properties | mapred-env.sh | mapred-site.xml | yarn-env.sh | **yarn-site.xml**

Parameter	Value	Description
spark.shuffle.service.port	7337	-

2. Modify the configuration item `yarn.nodemanager.aux-services` , adding spark_shuffle.



For configuration adjustment in the console, special characters contained in values such as "<", ">", and "&" are not escaped. To ensure proper processing of special characters, please follow the XML standard to set configurations.

yarn.nodemanager.aux-servi

Parameter	Value	Description
yarn.nodemanager.aux-services	mapreduce_shuffle,spark_shuffle	A comma separated list of services where service name should only contain a-zA-Z0-9_ and can not start with numbers

3. Add a new configuration item yarn.nodemanager.aux−services.spark_shuffle.class, setting its value to org.apache.spark.network.yarn.YarnShuffleService.
4. Introduce a new configuration item spark.yarn.shuffle.stopOnFailure, setting its value to false.
5. Save the settings and distribute them, then restart the YARN component to activate the configuration.

Modify Spark Configuration

1. Within the **Cluster Services > SPARK Component**, select **Operations > Configuration Management**.
2. Select the configuration file "spark−defaults.conf", then click on **Edit Configuration**. The new configuration items are as follows:

Configuration items	Value	Remarks
spark.shuffle.service.enabled	true	Initiate the shuffle service.
spark.dynamicAllocation.enabled	true	Initiate dynamic resource allocation.
spark.dynamicAllocation.minExecutors	1	The minimum number of executors allocated for each Application.
spark.dynamicAllocation.maxExecutors	30	The maximum number of executors allocated for each Application.
spark.dynamicAllocation.initialExecutors	1	Under normal circumstances, it is identical to the value of spark.dynamicAllocation.minExecutors.
spark.dynamicAllocation.schedulerBacklogTimeout	1s	If the backlog of pending tasks exceeds this duration, a request for a new executor will be initiated.
spark.dynamicAllocation.sustainedSchedulerBacklogTimeout	5s	If the task queue remains unprocessed, it will be triggered again every few seconds, with the number of executors requested in each round exhibiting exponential growth compared to the previous round.
spark.dynamicAllocation.executorIdleTimeout	60s	Executors will be removed from the Application after a certain number of seconds of inactivity.

3. Preserve configurations, distribute them, and restart the components.

Testing Spark's dynamic resource adjustment.

1. Description of resource configuration in the test environment.

In the test environment, there are two nodes deploying the NodeManager service, each with a resource configuration of 4−core CPU and 8GB memory, making the total cluster resources 8−core CPU and 16GB memory.

2. Description of the test task.

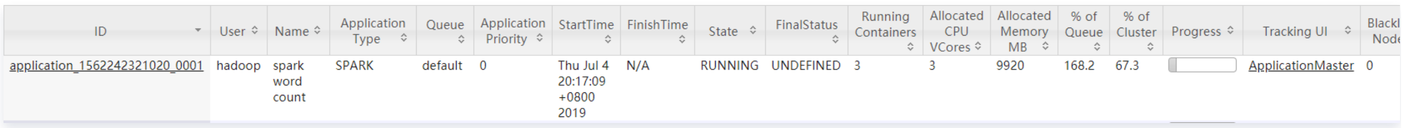
Test One:

- In the EMR console, navigate to the `/usr/local/service/spark` directory, switch to the hadoop user, and use `spark-submit` to submit a task, with data to be stored on hdfs.

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

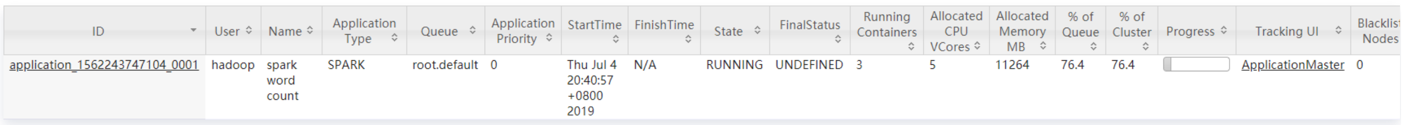
```
[root@172 spark]# su hadoop
[hadoop@172 spark]$  hadoop fs -put ./README.md /
[hadoop@172 spark]$ spark-submit --class org.apache.spark.examples.JavaWordCount --master yarn --num-executors 10 --
driver-memory 4g --executor-memory 4g --executor-cores 2 ./examples/jars/spark-examples_2.11-2.3.2.jar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

- In the Application panel of the YARN component's WebUI interface, one can observe the container and CPU allocation before and after 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	Blackl Node
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		ApplicationMaster	0

- Prior to setting dynamic resource scheduling, the maximum number of CPU allocations is three.



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	Blacklis 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		ApplicationMaster	0

- After setting dynamic resource scheduling, the maximum number of CPU allocations is five.

Conclusion: After configuring dynamic resource scheduling, the scheduler will dynamically increase allocated resources based on the needs of the application.

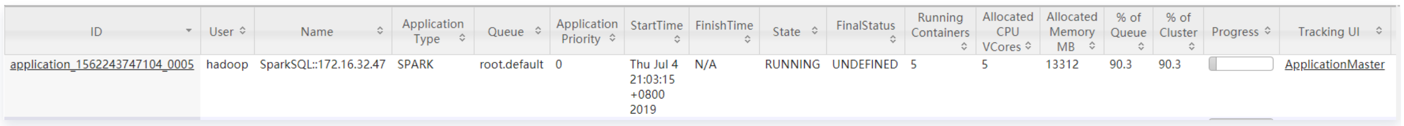
Test Two:

- Within the EMR console, navigate to the `/usr/local/service/spark` directory, switch to the hadoop user, and initiate the SparkSQL interactive console using `spark-sql` . The interactive console is set to occupy the majority of the test cluster's resources, allowing for observation of resource allocation before and after setting dynamic resource scheduling.

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

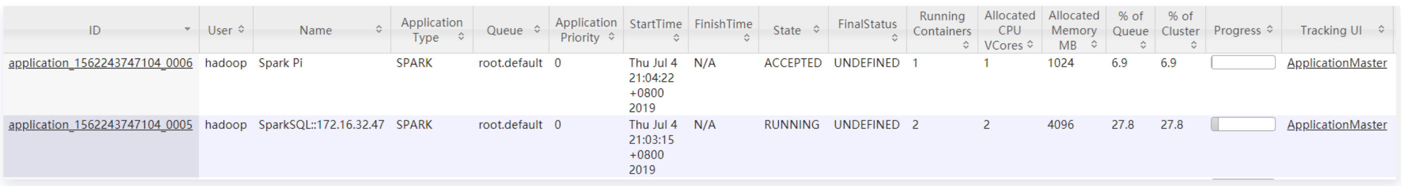
- Utilize the built-in example of calculating pi in spark2.3.0 as the test task. When submitting the task, set the number of task executors to five, the driver memory to 4g, the executor memory to 4g, and the number of executor cores to two.

```
[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 --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 VCores	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		ApplicationMaster

- When only running the SparkSQL task, the resource utilization rate is 90.3%.



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
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		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		ApplicationMaster

- After submitting the SparkPi task, the resource utilization rate of SparkSQL is 27.8%.

Conclusion: Although the SparkSQL task requested a large amount of resources upon submission, it did not execute any analysis tasks, thus a significant amount of resources remained idle. When the idle time exceeded the `spark.dynamicAllocation.executorIdleTimeout` setting, the idle executors were released, and other tasks received resources. In this test, the cluster resource usage rate of the SparkSQL task dropped from 90% to 28%, and the idle resources were allocated to the pi calculation task, demonstrating the effectiveness of automatic scheduling.

Note

The value of the configuration item `spark.dynamicAllocation.executorIdleTimeout` will influence the speed of dynamic resource scheduling. Testing has revealed that the time taken for resource scheduling is essentially equal to this value. It is recommended that you adjust the value of this configuration item according to actual needs to achieve optimal performance.

Spark Integration with Kafka

Last updated：2023-12-25 15:17:13

Dependency

Beginning with version 2.3, Spark no longer supports Kafka 0.8.2. The current network version of EMR has integrated Spark 2.4.3 and higher versions, necessitating the integration of Kafka 0.10.0 and subsequent versions.

Method of Inquiry

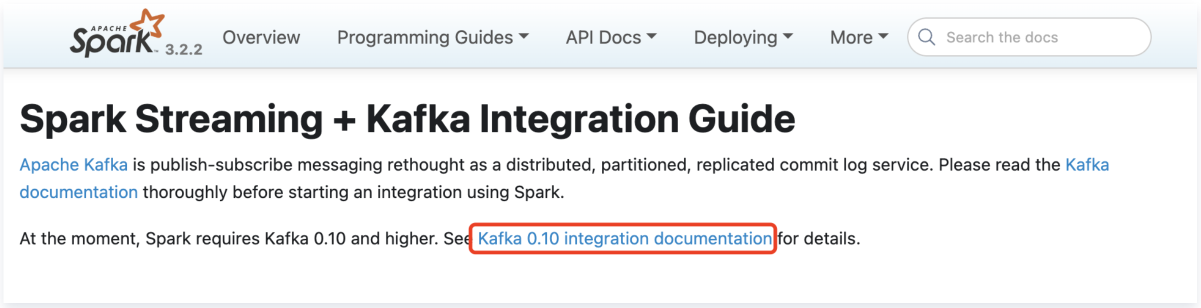
1. Visit the official website link and enter the version number link template:

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

   Replace `{spark.version}` with the corresponding Spark version. For instance, to view the dependencies of version 3.2.2, visit the following link:

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

2. Click on the corresponding link to view detailed integration instructions.



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：2023-12-25 15:17:24

Dependency

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

Method of Inquiry

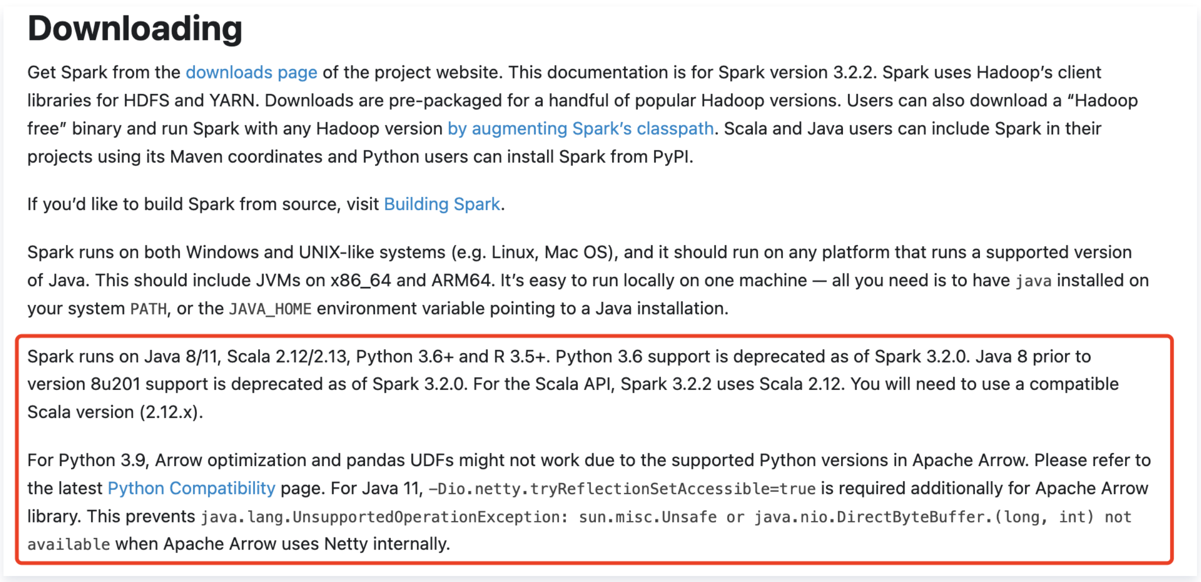
1. Visit the official website link and enter the version number link template:

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

   Replace `{spark.version}` with the corresponding Spark version. For instance, to view the dependencies of version 3.2.2, visit the following link:

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

2. View Dependencies.



Downloading

Get Spark from the downloads page of the project website. This documentation is for Spark version 3.2.2. Spark uses Hadoop's client libraries for HDFS and YARN. Downloads are pre-packaged for a handful of popular Hadoop versions. Users can also download a "Hadoop free" binary and run Spark with any Hadoop version by augmenting Spark's classpath. Scala and Java users can include Spark in their projects using its Maven coordinates and Python users can install Spark from PyPI.

If you'd like to build Spark from source, visit Building Spark.

Spark runs on both Windows and UNIX-like systems (e.g. Linux, Mac OS), and it should run on any platform that runs a supported version of Java. This should include JVMs on x86_64 and ARM64. It's easy to run locally on one machine — all you need is to have `java` installed on your system `PATH`, or the `JAVA_HOME` environment variable pointing to a Java installation.

Spark runs on Java 8/11, Scala 2.12/2.13, Python 3.6+ and R 3.5+. Python 3.6 support is deprecated as of Spark 3.2.0. Java 8 prior to version 8u201 support is deprecated as of Spark 3.2.0. For the Scala API, Spark 3.2.2 uses Scala 2.12. You will need to use a compatible Scala version (2.12.x).

For Python 3.9, Arrow optimization and pandas UDFs might not work due to the supported Python versions in Apache Arrow. Please refer to the latest Python Compatibility page. For Java 11, `-Dio.netty.tryReflectionSetAccessible=true` is required additionally for Apache Arrow library. This prevents `java.lang.UnsupportedOperationException: sun.misc.Unsafe or java.nio.DirectByteBuffer.(long, int) not available` when Apache Arrow uses Netty internally.

Hbase Development Guide
Using HBase Through API

Last updated：2023-12-25 16:20:03

HBase is a highly reliable, high-performance, column-oriented, scalable distributed storage system, serving as an open-source implementation of Google's BigTable. HBase employs Hadoop HDFS as its file storage system, utilizes Hadoop MapReduce to process the vast amount of data within HBase, and uses Zookeeper for coordination services.
HBase primarily consists of Zookeeper, HMaster, and HRegionServer. Specifically:

- ZooKeeper mitigates the single point of failure of HMaster, its master election mechanism ensures the provision of service by a single Master.
- HMaster manages user operations for adding, deleting, modifying, and querying tables, as well as managing the load balancing of HRegionServers. It can also adjust the distribution of Regions, migrating the HRegions within an exiting HRegionServer to other HRegionServers.
- HRegionServer is the most crucial module within HBase, primarily responsible for responding to user I/O requests and reading and writing data to the HDFS file system. Internally, HRegionServer manages a series of HRegion objects, each corresponding to a Region, and composed of multiple Stores within HRegion. Each Store corresponds to the storage of a Column Family.

This development guide, from a technical perspective, will assist users in developing with the EMR cluster. Considering user data security, currently, EMR only supports VPC network access.

1. Development Preparation

Ensure that you have activated Tencent Cloud and have created an EMR cluster. When creating the EMR cluster, you need to select the HBase and ZooKeeper components in the software configuration interface.

2. Utilizing HBase Shell

Before using HBase Shell, please log in to the Master node of the EMR cluster. The method of logging into EMR can be referred to in Log in to the Linux instance. Here, you can choose to log in using WebShell. Click on the login on the right side of the corresponding cloud server to enter the login interface. The username is set to root by default, and the password is the one entered by the user when creating EMR. After entering correctly, you can access the EMR command line interface.
In the EMR command line, first use the following command to switch to the Hadoop user, and enter the directory `/usr/local/service/hbase`:

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

You can enter HBase Shell using the following command:

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

In the HBase shell, you can view basic usage information and example commands by typing 'help'. Next, we will use the following command to create a new table:

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

After the table is established, you can use the `list` command to check whether the table you created already exists.

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

=> ["test"]
```

Utilize the `put` command to incorporate elements into the table you have 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
```

We have incorporated three values into the table we created. Initially, a value "value1" was inserted into the "cf:a" column of the "row1" row, and so forth.
Employ 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
```

Use the `get` command to obtain the value of a 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
```

Use the `drop` command to delete a table. Prior to deleting a table, it is necessary to first employ the `disable` command to deactivate a table:

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

Finally, you can employ the `quit` command to close the hbase shell.
For additional Hbase shell commands, please refer to the official documentation.

3. Utilizing Hbase through API

Initially, download and install Maven, configure Maven's environmental variables. If you are using an IDE, please set up the relevant Maven configurations within the IDE.

Create a new Maven project

Navigate to the directory where you wish to create a new project from the command line, for instance, within `D://mavenWorkplace`, and input the following command to create a new Maven project:

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

Where $yourgroupID is your package name. $yourartifactID is your project name, and maven-archetype-quickstart indicates the creation of a Maven Java project. Some files need to be downloaded during the project creation process, so please ensure a stable internet connection.
Upon successful creation, a project folder named $yourartifactID will be generated in the `D://mavenWorkplace` directory. The structure of the files within is as follows:

```
simple
```

```
        ---pom.xml            Core configuration, located at the root of the project
        ---src
            ---main
                ---java            Directory for Java source code
            ---resources      Directory for Java configuration files
          ---test
              ---java            Directory for test source code
              ---resources     Directory for test configurations
```

Our primary focus lies on the pom.xml file and the Java folder under main. The pom.xml file is primarily used for dependency and packaging configurations, while the Java folder houses your source code.

Incorporate Hadoop dependencies and sample code

Firstly, incorporate Maven dependencies into the pom.xml file:

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

Subsequently, incorporate packaging and compilation plugins into 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, users need to obtain the zookeeper address of the Hbase cluster. Log into any Master or Core node of EMR, navigate to the `/usr/local/service/hbase/conf` directory, and view the hbase.zookeeper.quorum configuration in hbase-site.xml to obtain the IP address $quorum of zookeeper, the hbase.zookeeper.property.clientPort configuration to obtain the port

number $clientPort of zookeeper, and the zookeeper.znode.parent configuration to obtain the znode path $znodePath used by hbase.
Next, add the sample code. Create a new Java Class named PutExample.java in the main>java folder, and incorporate the following code into 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.");
    }
}
```

Compile the code and package it for upload.

Navigate to the project directory using the local command line and execute the following command to compile and package the project:

```
mvn package
```

A display of 'build success' indicates a successful operation. The packaged file can be found in the target folder within the project directory.
Use scp or sftp tools to upload the packaged file to the EMR cluster. **It is essential to upload the jar package that has been packaged together with the dependencies**. Run the following in the local command line mode:

```
scp $localfile root@PublicIPAddress:$remotefolder
```

In this context, $localfile refers to the path and name of your local file, 'root' is the username of the CVM server, and the public IP can be viewed in the node information of the EMR console or in the cloud server console. $remotefolder is the path on the CVM server where you wish to store the file. Once the upload is complete, you can check in the EMR cluster command line to see if the corresponding file is in the respective folder.

4. Run the Sample

Log into the Master node of the EMR cluster and switch to the hadoop user. Execute the sample using the following command:

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

Upon the console outputting "Done", it indicates that all operations have been completed. You can switch to the hbase shell and use the `list` command to check whether the Hbase table created using the API was successful. If successful, you can use the `scan` command to view the specific contents 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 detailed instructions on API usage, please refer to the official documentation.

Using Hbase with Thrift

Last updated：2023-12-25 16:20:15

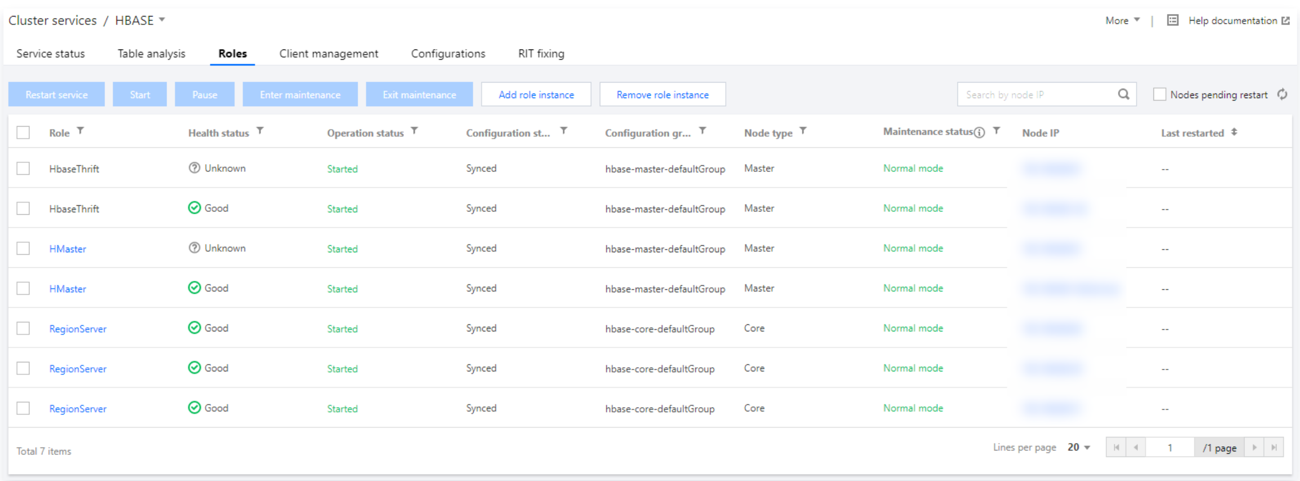
Apache Thrift is a cross-platform, cross-language development framework, offering multilingual compilation capabilities and a variety of server operating modes. Users describe interface functions and data types through Thrift's IDL (Interface Description Language), and then generate interface files of various language types through Thrift's compilation environment for the development of scalable and cross-language services.
It amalgamates a robust software stack and code generation engine to construct efficient services that seamlessly integrate across programming languages such as C++, Java, Go, Python, PHP, Ruby, Erlang, Perl, Haskell, C#, Cocoa, JavaScript, Node.js, Smalltalk, and OCaml.
The Thrift server is a service within HBase, primarily designed to support multilingual APIs and developed based on Apache Thrift. The Thrift API relies on client and server processes. This section will illustrate, using Python as an example, how to utilize HBase through Python programming via Thrift.

1. Development Preparation

Ensure that you have activated Tencent Cloud and have created an EMR cluster. When creating the EMR cluster, you need to select the Hbase component in the software configuration interface.

2. Utilizing HBase via Python API

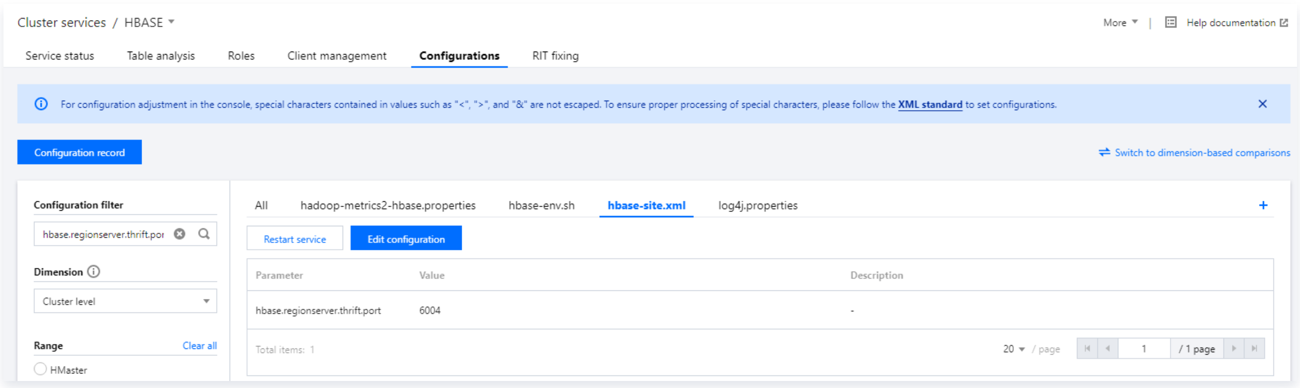
Within the EMR cluster, HBase integrates Thrift by default and initiates the Thrift Server on the Master1 node (external IP node).
1. Within the cluster details page, select **Cluster Services**, click on the top right corner of the Hbase component **Operation > Roles**, to view the running status and IP address of the Thrift Server.



Role	Health status	Operation status	Configuration st...	Configuration gr...	Node type	Maintenance status	Node IP	Last restarted
HbaseThrift	Unknown	Started	Synced	hbase-master-defaultGroup	Master	Normal mode	[redacted]	--
HbaseThrift	Good	Started	Synced	hbase-master-defaultGroup	Master	Normal mode	[redacted]	--
HMaster	Unknown	Started	Synced	hbase-master-defaultGroup	Master	Normal mode	[redacted]	--
HMaster	Good	Started	Synced	hbase-master-defaultGroup	Master	Normal mode	[redacted]	--
RegionServer	Good	Started	Synced	hbase-core-defaultGroup	Core	Normal mode	[redacted]	--
RegionServer	Good	Started	Synced	hbase-core-defaultGroup	Core	Normal mode	[redacted]	--
RegionServer	Good	Started	Synced	hbase-core-defaultGroup	Core	Normal mode	[redacted]	--

Total 7 items
Lines per page 20 1 /1 page

2. Click on the **Configuration Management** page, and view the port number of Thrift through the hbase.regionserver.thrift.port Thrift configuration item in the hbase-site.xml file:



For configuration adjustment in the console, special characters contained in values such as "<", ">", and "&" are not escaped. To ensure proper processing of special characters, please follow the XML standard to set configurations.

Configuration filter: hbase.regionserver.thrift.por
Dimension: Cluster level
Range: HMaster

Parameter	Value	Description
hbase.regionserver.thrift.port	6004	-

Total items: 1
20 / page 1 / 1 page

Now, we can directly manipulate HBase through Python programming.

Cloud Load Balancer

The HA cluster comprises two master nodes, both of which initiate the Thrift Server by default. To achieve load balancing, the client code needs to devise a custom strategy to distribute requests across the two Thrift Servers. These two Thrift Servers operate independently, with no communication between them.

Data Preparation

Utilize HBase Shell to create a new table in HBase. If you have used HBase in EMR and have already created your own table, this step can be skipped:

```
[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
```

Utilizing Python to view tables in HBase

Initially, it is necessary to install the Python dependency package. Switch to the root user, the password is the one you set when creating the EMR cluster. Install the python-pip tool first, then install the dependency package:

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

Subsequently, switch back to the Hadoop user and create a new Python file, Hbase_client.py, incorporating the following code:

```
#! /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
Wherein $thriftIP is the internal IP address of the Master node, and $port is the port number of the ThriftService, the same applies below.

Upon saving, run the program directly, it will output the existing tables in Hbase directly on the console:

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

Utilize Python to create an Hbase table.

Create a new Python file, Create_table.py, and incorporate the following code into 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
```

This program will add a new table named thrift_test_1 in Hbase, and output all existing tables, with the following operational effect:

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

Employ Python to insert data into an Hbase table.

Establish a new Python file, Insert.py, and integrate the following code into 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()
```

This program will add two rows of data in the Hbase's thrift_test_1 table, each row containing two pieces of data. The inserted data can be viewed in the 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
```

Utilize Python to view data within the Hbase table.

There are two viewing methods, one is to view a single row, and the other is to use Scan to view all data. Create a new Python file, Scan_table.py, and incorporate the following code:

```
#! /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()
```

In this context, GetRow is used to obtain a row of data, while scannerGetList is employed to retrieve data from the entire table. Upon running this program, the output 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(columns={'cf:a': TCell(timestamp=1530704886969, value='value3'), 'cf:b':
TCell(timestamp=1530704886975, value='value4')}, row='row2')]
```

It separately outputs the data of the first row and the data of the entire table.

Employ Python to delete data from the Hbase.

Create a new Python file, Delete_row.py, and incorporate the following code into 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()
```

This program will delete the second row of data in the test table. After running, the content of the table can be viewed in the 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 juncture, only the data from the first row remains in the table.
For more detailed operations regarding Thrift, refer to How to Use Thrift.

Spark on Hbase

Last updated：2023-12-25 16:20:25

For detailed operations concerning Spark on Hbase, one may refer to the Github homepage Spark-HBase Connector.

MapReduce on Hbase

Last updated：2023-12-25 16:20:34

For detailed information regarding read and write operations on MapReduce on Hbase, please refer to the section titled Usage Examples.

Phoenix on Hbase Development Guide
Phoenix Client Usage

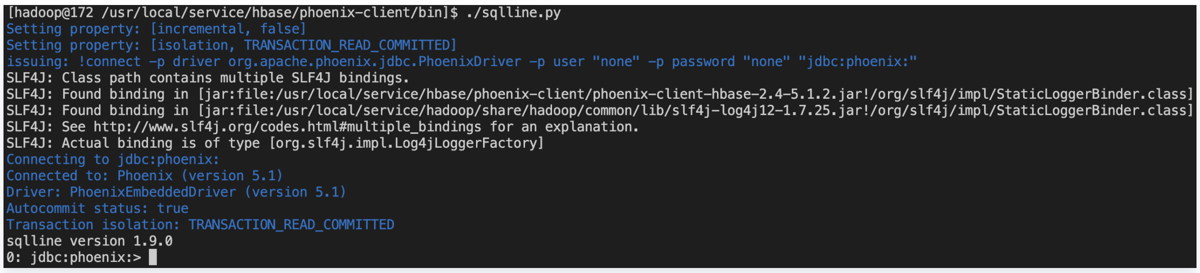
Last updated：2023-12-25 16:33:14

The Phoenix query engine facilitates the exploration of HBase data using SQL, transforming SQL queries into one or more HBase APIs, in conjunction with the implementation of coprocessors and custom filters, and orchestrating their execution. Utilizing Phoenix for simple queries yields millisecond-level performance, while for queries involving millions of rows, the performance scale is in seconds. **In EMR, when selecting the HBase component cluster, the Phoenix client is integrated by default.**

1. Initiating the Client
   Switch to the hadoop user, navigate to the /usr/local/service/hbase/phoenix-client/bin directory, and utilize the Python command-line tool of Phoenix:

   ```
   ./sqlline.py
   ```

   Upon successful execution, the following is displayed:

   ```
   [hadoop@172 /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. The Phoenix engine facilitates data querying using SQL, with some common operations as follows:
   - Create 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;
     ```

   - Dropping a Table

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

For additional operations and explanations, please refer to the community documentation.

Phoenix JDBC Usage

Last updated：2023-12-25 16:33:37

Establishing a Maven Project

Navigate to the directory where you wish to establish the project in your local shell, for instance, within D://mavenWorkplace, and input the following command to create a new Maven project:

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

Wherein, **$yourgroupID** corresponds to your package name. **$yourartifactID** signifies your project name, while **maven-archetype-quickstart** indicates the creation of a Maven Java project. The project establishment process necessitates the download of certain files, hence, ensure a stable internet connection.
Upon successful creation, a project folder named $yourartifactID will be generated within the D://mavenWorkplace directory. The structure of the files contained therein is as follows:

```
simple
    ---pom.xml          Core configuration, located at the root of the project
    ---src
        ---main
            ---java           Java source code directory
            ---resources     Directory for Java configuration files
        ---test
            ---java           Test source code directory
            ---resources     Test configuration directory
```

Our primary focus lies on the pom.xml file and the Java folder under main. The pom.xml file is primarily used for dependency and packaging configurations, while the Java folder houses your source code.

Incorporate Phoenix dependency

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

Ensure that the phoenix.version is consistent with the Phoenix version in the cluster.

Establish a JDBC connection object

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;

public class PhoenixJdbc {

    static {
        try {
            Class.forName("org.apache.phoenix.jdbc.PhoenixDriver");
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        String createSql = "create table if not exists user_tb(id VARCHAR,uname VARCHAR)";

        Connection connection = null;
        try {
            connection = DriverManager.getConnection("jdbc:phoenix:10.0.0.3:2181,10.0.0.5:2181,10.0.0.8:2181");
            Statement statement = connection.createStatement();

            statement.executeUpdate(createSql);
            statement.executeUpdate("upsert into user_tb values ('1','Zhang San')");
            statement.executeUpdate("upsert into user_tb values ('2','Li Si')");
            connection.commit();

        } catch (SQLException throwables) {
            throwables.printStackTrace();
        }
    }
}
```

Compile the code and package it for upload.

If your Maven configuration is correct and the dependency packages have been successfully imported, then the entire project should be error-free and ready for compilation. Enter the project directory in the local command line mode and execute the following command to package the entire project:

```
mvn package
```

Upload to the Hbase node of the EMR cluster and run in the local command line mode:

```
scp $localfile root@PublicIPAddress:$remotefolder
```

Herein, $localfile represents the path and name of your local file; 'root' is the username of the CVM server; the public IP can be viewed in the node information of the EMR console or in the cloud server console; $remotefolder is the path on the CVM server where you wish to store the file. Upon completion of the upload, you can check in the EMR command line whether the corresponding folder contains the respective file.

Execute Sample

Initially, it is necessary to log into the Hbase component node in the EMR cluster, preferably the Master node. For the method of logging into EMR, please refer to Log in to the Linux instance. Here, we can opt to use WebShell for login. Click on the login on the right side of the corresponding cloud server to enter the login interface. The default username is 'root', and the password is the one entered by the user when creating EMR. After entering correctly, you can access the command line interface.
In the EMR command line, use the following command to switch to the Hadoop user:

```
su hadoop
```

Then, navigate to the folder where your jar package is stored and execute the following command:

```
java -cp $jarPackageName $fullyQualifiedClassName
```

Wherein, **$jarPackageName** refers to the package name of the previously uploaded EMR node, and **$fullyQualifiedClassName** refers to the fully qualified name of the Java main method, package.mainclass.

Phoenix Best Practices

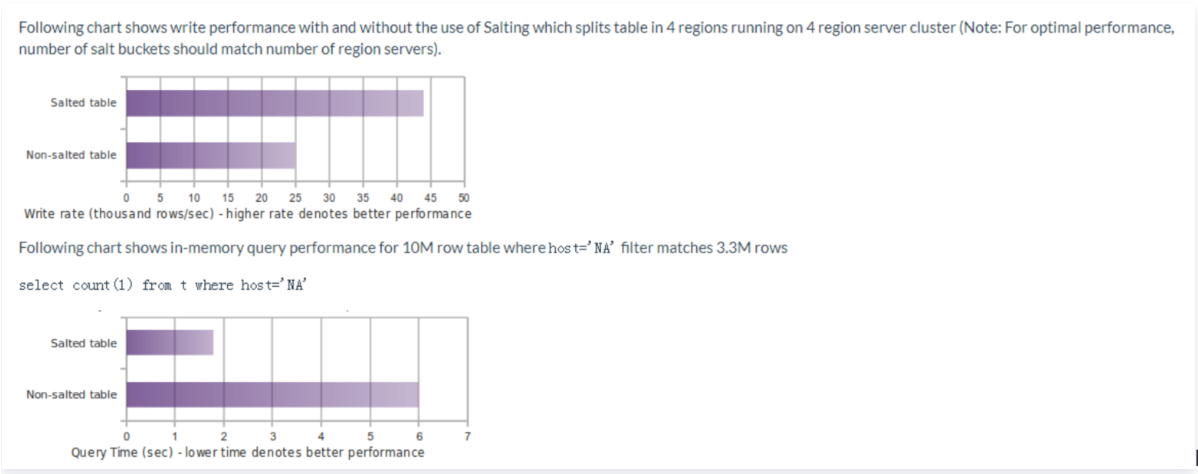
Last updated：2023-12-25 16:33:47

Utilizing Phoenix Salted Tables

If the Row Key of HBase is not meticulously designed, and it is also incrementing, sequential writing could potentially lead to hotspot issues. Phoenix offers a transparent method, associating salt and RowKey to a designated table's scheme. It only requires the addition of the SALT_BUCKETS keyword when creating the table, with a value range of 1 to 256. For instance:

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

This eliminates the need for meticulous RowKey design when using the HBase API. If users lack a comprehensive understanding of HBase Row Key design, it is recommended to use salted tables. The Phoenix official provides a comparison of the performance between salted and non-salted tables for writing and querying:



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 salted performance or operation instructions, refer to the Phoenix Salted Tables community documentation.

Phoenix Secondary Indexes

For HBase, if one wishes to precisely locate a specific row record, the only method is through a rowkey query. If data is not sought via rowkey, each column's value must be compared row by row, resulting in a full table scan. For larger tables, the cost of a full table scan is unacceptable. However, in many instances, data needs to be queried from multiple perspectives. This requires a secondary index to accomplish. The principle of a secondary index is straightforward, but maintaining it oneself can be somewhat cumbersome.

Phoenix Secondary Index Configuration

Phoenix within EMR directly supports Phoenix's secondary indexes. If non-transactional, mutable indexes are required, they can be configured following the steps below. Open the EMR component management page, click on **Hbase**, select **Configuration > Configuration Management**, and add three configuration items to 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 queues 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 queues for index and metadata
updates</description>
   </property>
```

Phoenix Secondary Index Usage

To create a secondary index, use the following command:

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

For more detailed instructions on secondary index operations, please refer to the Phoenix Secondary Index Community Documentation.

Hive Development Guide
Hive Overview

Last updated：2023-12-25 16:38:17

Hive is a data warehouse framework constructed atop the Hadoop file system, offering a plethora of functionalities for data warehouse management, inclusive of data ETL (Extraction, Transformation, and Loading) tools, data storage administration, and the capacity for querying and analyzing large data sets. Concurrently, Hive delineates a SQL-like development language, permitting users to map structured data files to a database table, and furnishing a simplistic SQL query function.

- Within EMR, Hive is installed in the path /usr/local/service/hive on the EMR node.
- For a more comprehensive introduction to Hive, please refer to the Official Apache Hive Website.

Hive Service Roles

Role name	Note
HiveServer2	Hive's ThriftServer service is designed to receive client query requests and perform SQL compilation and parsing. It supports concurrent multi-client access and authentication. An EMR cluster can deploy multiple HiveServer2 instances, supporting expansion to Router nodes and configuration of load balancing.
Hive MetaStore	Hive's metadata service is responsible for maintaining the metadata information of Hive Databases and Hive Tables. The metadata management capabilities of this module are also integrated into engines such as Spark and Trino. An EMR cluster can deploy multiple Hive MetaStores, supporting expansion to Router nodes.
Hive Client	The Hive client, offering application drivers such as Beeline and JDBC, can submit SQL jobs to HiveServer2. All nodes deploying Hive services will be installed.
Hive WebHCat	WebHCat is a service that provides REST API for HCatalog, offering a Rest interface to execute Hive commands and submit MapReduce tasks. Multiple WebHCat instances can be deployed within a cluster, supporting expansion to Router nodes.

Internal and External Tables in Hive

- **Internal Tables:** Hive manages the metadata and actual data of internal tables. When the DROP syntax is used to delete an internal table, both the table's metadata and corresponding data will be deleted. After creating an internal table, the HDFS file will be mapped to a Table, and then the Hive data warehouse will generate the corresponding directory. The default warehouse path in EMR is /usr/hive/warehouse/${tablename}, which is located on HDFS, where ${tablename} is the name of the table you created.
- **External Tables:** External tables in Hive are quite similar to internal tables, but their data is not stored in the directory belonging to their own table, but elsewhere. The advantage of this is that if you want to delete this external table, the data pointed to by this external table will not be deleted, it will only delete the metadata corresponding to the external table.

Hive Syntax

Hive in Elastic MapReduce is fully compatible with open-source community syntax, which can be referenced through the HiveQL Community Syntax Manual.

Fundamental Usage
Basic Hive Operations

Last updated：2023-12-25 16:38:41

This document demonstrates how to perform basic operations such as creating databases and tables, importing data, and executing queries on Hive in EMR.

Development Preparations

- Ensure that you have activated Tencent Cloud and have established an EMR cluster. For more details, refer to Creating a Cluster.
- Select the Hive component in the software configuration interface when creating an EMR cluster.
- The example includes optional content that requires access to Tencent Cloud Object Storage (COS). You may refer to Creating a Storage Bucket to create a bucket in COS, and enable object storage authorization on the EMR Console Instance Information page.

Note:
- For logging into the EMR node, refer to Logging into a Linux Instance. In the cluster details page, select **Cluster Resources > Resource Management**, click on the corresponding node resource ID to enter the cloud server list, click on **Login** on the right, and you can use WebShell to log into the instance.
- The default username for logging into a Linux instance is 'root', and the password is the one entered by the user when creating the EMR. Upon correct entry, you can access the command line interface.
- All operations in this document are performed as the 'hadoop' user. Please switch user identities after logging into the command line interface.

Preparing Sample Data

Log into the Master node, switch to the 'hadoop' user in the EMR command line using the following command, and enter the Hive folder:

```
su hadoop
cd /usr/local/service/hive
```

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

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

Assign permissions and execute the script in the following manner. This script file will generate 1,000,000 pairs of random numbers and save them to the file named 'hive_test.data':

```
chmod +x script_name
./gen_data.sh > hive_test.data
```

Use the following command to first upload the generated test data to HDFS, where ${hdfspath} is the path on HDFS where you store your files:

```
hdfs dfs -put ./hive_test.data /${hdfspath}
```

You can also use data from COS. Upload the data to COS; if the data is local, you can use the COS console to upload it. If the data is in the EMR cluster, use the following command to upload the data, where ${bucketname} is the name of the COS bucket you created:

```
hdfs dfs -put ./hive_test.data cosn://${bucketname}/
```

Basic Operations in Hive

Log into the Master node of the EMR cluster, switch to the hadoop user, and enter the Hive command line through the Hive client:

```
hive
```

Create Database and Table

Use the SHOW syntax to display all current databases:

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

Use the CREATE DATABASE syntax to create a database named 'test':

```
hive> create database if not exists test;
OK
Time taken: 0.176 seconds
```

Use the USE syntax to navigate to the recently created 'test' database:

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

Use the CREATE TABLE syntax to create a new internal table named 'hive_test' under the 'test' database:

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

Finally, use the SHOW TABLES syntax to verify if the table was created successfully:

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

Importing Data

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

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

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

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

Data stored locally in the EMR cluster can also be imported into Hive using the following command:

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

Note:
Where ${hdfspath} is the path to your file on HDFS, ${bucketname} is the name of your COS bucket, and ${localpath} is the path to your data stored locally on your EMR cluster. Upon completion of the import, the source data will be deleted.

Execute Query

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)
```

Calculate the total number of rows in the table:

```
hive> select count(*) from hive_test;
OK
1000000
Time taken: 18.504 seconds, Fetched: 1 row(s)
```

Drop databases & tables

Utilize the DROP TABLE syntax to delete a Hive table:

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

Employ the DROP DATABASE syntax to eliminate a Hive database:

```
hive> drop database if exists test;
OK
Time taken: 0.531 seconds
```

Hive Connection Method

Last updated: 2023-12-25 16:38:51

This document elucidates the process of connecting to Hive using three distinct methods – Hive client, Beeline, and Java – within the context of Elastic MapReduce.

Development Preparations

- Ensure that you have activated Tencent Cloud and have established an EMR cluster. For further details, please refer to the section titled 'Creating a Cluster'.
- Select the Hive component in the software configuration interface when creating an EMR cluster.

Note:
- To log into an EMR node, please refer to 'Logging into a Linux Instance'. In the cluster details page, select 'Cluster Resources > Resource Management', click on the corresponding node resource ID to access the cloud server list, and click 'Log In' on the right to use WebShell to log into the instance.
- The default username for logging into a Linux instance is 'root', and the password is the one entered by the user when creating the EMR. Upon correct entry, you can access the command line interface.
- All operations in this document are performed as the 'hadoop' user. After logging into the command line interface, use the 'su hadoop' command to switch user identities.

Connecting to Hive

The Hive service is deployed on the Master node by default. You can also refer to 'Cluster Expansion' to deploy HiveServer2 on the Router node. This document uses the Master node as an example to connect to the Hive service.

Method One: Via the Hive Client

Log into the Master node of the EMR cluster, switch to the Hadoop user, and execute the following command to enter the Hive command line:

```
hive
```

You can also use the '-h' parameter to obtain basic information about Hive commands.

Method Two: Connecting to HiveServer2 via Beeline

Log into the Master node of EMR and connect to Hive using the beeline command:

```
beeline -u "jdbc:hive2://${hs2_ip}:${hs2_port}" -n hadoop
```

Note:
1. ${hs2_ip} represents the internal IP of the node where the HiveServer2 service is deployed in the cluster. You can view this in the cluster details page under **Cluster Services > Hive > Role Management**.
2. ${hs2_port} represents the port number of the HiveServer2 in the cluster. The default value is 7001. You can view this in the hive-site.xml configuration file under the hive.server2.thrift.port configuration item in the cluster details page under **Cluster Services > Hive > Configuration Management**.

Method Three: Connecting to Hive via Java

This article uses Maven as an example to manage your project. Maven is a project management tool that can help you conveniently manage project dependencies. That is, it can obtain jar packages through the configuration of the pom.xml file, eliminating the need for manual addition.
Firstly, download and install Maven locally, and configure the Maven environment variables. If you are using an IDE, please set up the relevant Maven configurations within the IDE.

In the local shell, navigate to the directory where you want to create a new project, such as /tmp/mavenWorkplace, and enter the following command to create a new Maven project:

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

Note:

1. ${yourgroupID} refers to your package name; ${yourartifactID} refers to your project name;
2. 'maven−archetype−quickstart' signifies the creation of a Maven Java project. Some files need to be downloaded during the project creation process, so please ensure a stable internet connection.

Our primary focus lies on the pom.xml file and the Java folder under main. The pom.xml file is primarily used for dependency and packaging configurations, while the Java folder houses your source code.
Initially, configure the project dependencies (hadoop−common and hive−jdbc) in the pom.xml file:

```
<dependencies>
      <dependency>
         <groupId>org.apache.hive</groupId>
         <artifactId>hive-jdbc</artifactId>
         <version>${hive_version}</version>
      </dependency>
      <dependency>
         <groupId>org.apache.hadoop</groupId>
         <artifactId>hadoop-common</artifactId>
         <version>${hadoop_version}</version>
      </dependency>
</dependencies>
```

Note:
Here, ${hive_version} represents the Hive version in your cluster, and ${hadoop_version} represents the Hadoop version in your cluster.

Proceed to add packaging and compilation plugins in 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>
```

Right-click to create a new Java Class under src>main>java, input your Class name. In this example, we use App.java, and add sample code to the Class:

```
package org.example;

import java.sql.*;
/**
 * Created by tencent on 2023/8/11.
 */
public class App {
    private static final String DRIVER_NAME = "org.apache.hive.jdbc.HiveDriver";
    public static void main(String[] args) throws SQLException {
        try {
            // Load the hive-jdbc driver
            Class.forName(DRIVER_NAME);
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
            System.exit(1);
        }
        // Obtain the connection based on the connection information and username/password
        Connection conn = DriverManager.getConnection("jdbc:hive2://$hs2_ip:$hs2_port/default", "hadoop", "");
        // Create status parameters (use conn.prepareStatement(sql) to precompile SQL to prevent SQL injection, commonly used
for parameterized execution of SQL. For batch execution of different SQL, it is recommended to use the method below)
        Statement stmt = conn.createStatement();
        // The following are simple table creation and addition operations
        String tableName = "hive_test";
        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(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));
        }
    }
}
```

The program will first connect to the HiveServer2 service, then create a table named hive_test in the default database. Subsequently, it will insert two elements into this table and display the entire table's content.
If your Maven configuration is correct and the dependency packages have been successfully imported, then the entire project can be directly compiled. Navigate to the project directory in the local shell and execute the following command to package the entire project:

```
mvn clean package -DskipTests
```

During the execution, additional files may need to be downloaded until a 'build success' message appears, indicating successful packaging. Subsequently, you can locate the packaged jar file in the target folder under the project directory.

Upload and execute the program

Initially, the compressed jar package needs to be uploaded to the EMR cluster using scp or sftp tools. Run the following in the local shell: (Enter 'yes' followed by password verification)

```
scp ${localfile} root@${master_pubilc_ip}:/usr/local/service/hive
```

Note:

1. ${localfile} represents the path and name of your local file, where 'root' is the username of the CVM server. The public IP can be viewed in the node information in the EMR console or in the cloud server console.
2. ${master_pubilc_ip} refers to the public IP of your cluster's Master node.

Upload the packaged jar file to the /home/hadoop/ directory in the EMR cluster. Once the upload is complete, you can check in the EMR command line whether the corresponding folder contains the relevant file. **It is imperative to upload the jar package that has dependencies.**
Log into the EMR cluster, switch to the hadoop user, and navigate to the /home/hadoop/ directory. Execute the program:

```
yarn jar ./hive-test-1.0-SNAPSHOT-jar-with-dependencies.jar org.example.App
```

Note:
Here, ./hive−test−1.0−SNAPSHOT−jar−with−dependencies.jar represents the path + name of your jar package, and org.example.App refers to the package name + class name of the previous Java Class.

The execution result is as follows:

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

Configuring the Hive Execution Engine

Last updated: 2023-12-25 16:39:05

Hive accommodates MapReduce, Tez, and Spark as execution engines, with MapReduce being the default in Elastic MapReduce. This document elucidates the process of configuring the execution engine within Hive.
Under normal circumstances, it is recommended to utilize Tez as the execution engine in lieu of MapReduce, thereby achieving superior computational efficiency.

Development Preparations

- Ensure that you have activated Tencent Cloud and have established an EMR cluster. For more details, refer to Creating a Cluster.
- During the creation of an EMR cluster, select the Hive component in the software configuration interface. If there is a need to switch to Tez or Spark execution engines, the corresponding components must be selected when creating the cluster.

Engine Configuration

Log into the Hive client, for specific operations refer to Hive Connection Methods. The Hive execution engine is configured by the hive.execution.engine parameter, with the default engine being 'mr'. Execute the following command to view the current execution engine:

```
hive> set hive.execution.engine;
hive.execution.engine=mr
```

Switching Execution Engines in the Command Line

The following command can be used to modify the execution engine in the current client session.
Switching the Execution Engine to Tez:

```
hive> set hive.execution.engine=tez;
```

Switching the Execution Engine to Spark:

```
hive> set hive.execution.engine=spark;
```

Upon completion, the following command can be entered for verification:

```
hive> set hive.execution.engine;
```

If the returned information matches the execution engine you have set, it indicates a successful configuration.

Note:
The aforementioned operation to switch execution engines is only effective for individual client sessions. The default execution engine will be restored when other clients enter or exit and reconnect.

Switching the Execution Engine within the Configuration File:

To globally switch the default execution engine, navigate to the EMR Console **Cluster Services > HIVE > Configuration Management** page, locate the hive-site.xml configuration file, and set the following parameter values:
Switching the Execution Engine to Tez:

```
hive.execution.engine = tez
```

Switching the Execution Engine to Spark:

```
hive.execution.engine = spark
```

Proceed to save by clicking on **Save Configuration > Save and Distribute,** then **Restart HiveServer2**. Upon completion, you can modify the default execution engine of Hive.

Advanced Usage

Configuring LDAP Authentication

Last updated：2023-12-25 16:39:24

This document introduces the activation and utilization of LDAP configuration in Elastic MapReduce within Hive.

Development Preparations

- Ensure that you have activated Tencent Cloud and have established an EMR cluster. For more details, refer to Creating a Cluster.
- Establish a default Hadoop scenario cluster and select the Hive component within the software configuration interface.

Activating LDAP Authentication

Navigate to the EMR console, click on **Cluster Services > HIVE > Configuration Management**, select the hive-site.xml configuration file, add the following configuration items and set parameter values, save the configuration and distribute it, then restart HiveServer2:

Category	Value	Remarks
hive.server2.authentication	LDAP	Establish the authentication mechanism as LDAP.
hive.server2.authentication.ldap.url	ldap://$l{dap_ip}:389	Specify the URL of the LDAP service. ${ldap_ip} represents the IP of the node where the OPENLDAP service is located, which can be obtained under **Cluster Services > OPENLDAP** in the EMR console. Fill in according to the actual situation of the self-built LDAP service.
hive.server2.authentication.ldap.baseDN	ou=People,dc=emr,dc=cloud,dc=tencent,dc=com	The Base DN where the LDAP service users are located in EMR, please fill in according to the actual situation of the self-built LDAP service.
hive.server2.authentication.ldap.guidKey	cn	The format of the LDAP service username in EMR, please fill in according to the actual situation of the self-built LDAP service.

Accessing HiveServer2

Upon enabling LDAP authentication, it is necessary to provide the LDAP username and password when accessing HiveServer2.

Beeline client connecting to Hive

```
beeline -u "jdbc:hive2://${hs2_ip}:${hs2_port}"  -n ${user} -p ${password}
```

JDBC connection to Hive

```
jdbc:hive2://${hs2_ip}:${hs2_port}/default;user=${user};password=${password}
```

Note:

1. ${user} represents the LDAP username.
2. ${password} represents the LDAP password.
3. ${hs2_ip} represents the internal network IP of the node where the HiveSever2 service is deployed in the cluster. This can be viewed in the **Cluster Services>Hive>Role Management** section on the cluster details page.
4. ${hs2_port} denotes the port number of the HiveServer2 in the cluster. The default value is 7001. This can be viewed in the hive.server2.thrift.port configuration item of the hive-site.xml configuration file in the **Cluster Services>Hive>Configuration Management** section on the cluster details page.

Note:
Upon integrating Hive with EMR OpenLdap and adding new users, it is necessary to assign 644 permissions to the /emr/hive directory under hdfs for new user access.

HiveServer2 Load Balancing

Last updated：2023-12-25 16:39:33

When multiple HiveServer2 services exist within an EMR cluster, the utilization of Zookeeper services can facilitate the implementation of load balancing for accessing HiveServer2. This document provides a comprehensive guide on the usage of HiveServer2 load balancing.

Development Preparations

- Ensure that you have activated Tencent Cloud and have established an EMR cluster. For more details, refer to Creating a Cluster .
- Create a Hadoop default scenario **High Availability** cluster, and select the Hive component in the software configuration interface.

Implementing hs2 Load Balancing via Zookeeper

In the default Hadoop scenario, the High Availability cluster comes with Zookeeper services pre-installed. You can connect to HiveServer2 using the following methods, leveraging Zookeeper to achieve load balancing effects:

```
beeline -u 'jdbc:hive2://${hive.zookeeper.quorum}/;serviceDiscoveryMode=zooKeeper;zooKeeperNamespace=
<hive.server2.zookeeper.namespace>' -n ${user} -p ${password}
```

Note:

1. ${hive.zookeeper.quorum} and ${hive.server2.zookeeper.namespace} are Zookeeper Server related configurations in Hive. You can find the corresponding values for the hive.zookeeper.quorum and hive.server2.zookeeper.namespace parameters in the hive-site.xml configuration file under EMR Console **Cluster Services > HIVE > Configuration Management**.
2. The user and password are the LDAP username and login password you have set.

User-Defined Function (UDF)

Last updated：2023-12-25 16:40:26

This document introduces you to User-Defined Functions (UDF) and outlines the development and utilization process.

Classification of User-Defined Functions

Classification of User-Defined Functions	Description
UDF（User Defined Scalar Function）	Custom scalar functions, commonly referred to as UDFs, operate on a one-to-one input-output relationship, meaning they read one row of data and produce a single output value.
UDTF（User Defined Table-valued Function）	Custom table-valued functions are designed to address scenarios where a single function call outputs multiple rows of data. They are the only type of user-defined functions that can return multiple fields.
UDAF（User Defined Aggregation Function）	Custom aggregate functions operate on a many-to-one input-output relationship, aggregating multiple input records into a single output value. They can be used in conjunction with the Group By statement in SQL.

For further details, please refer to the community documentation: UDF, UDAF, UDTF.

Developing UDFs

Utilize an IDE to create a Maven project. The basic project information is as follows, and you have the flexibility to customize the groupId and artifactId:

```
<groupId>org.example</groupId>
<artifactId>hive-udf</artifactId>
<version>1.0-SNAPSHOT</version>
<packaging>jar</packaging>
```

Incorporate pom dependencies:

```
<!-- https://mvnrepository.com/artifact/org.apache.hive/hive-exec -->
<dependency>
    <groupId>org.apache.hive</groupId>
    <artifactId>hive-exec</artifactId>
    <version>3.1.3</version>
    <exclusions>
        <exclusion>
          <groupId>org.pentaho</groupId>
          <artifactId>*</artifactId>
        </exclusion>
      </exclusions>
</dependency>
```

Create a class with a name of your choosing. In this document, we will use 'nvl' as an example:
Method 1: Inherit UDF and override the evaluate method:

```
package org.example;

import org.apache.hadoop.hive.ql.exec.UDF;

public class nvl extends UDF {
    public String evaluate(final String s) {
        if (s == null) { return null; }
        return s + ":HelloWorld";
    }
}
```

Method 2 (Recommended for scenarios with complex parameter passing): Inherit GenericUDF and override initialize, evaluate, and getDisplayString:

```
package org.example;

import org.apache.hadoop.hive.ql.exec.Description;
import org.apache.hadoop.hive.ql.exec.UDFArgumentException;
import org.apache.hadoop.hive.ql.exec.UDFArgumentTypeException;
import org.apache.hadoop.hive.ql.metadata.HiveException;
import org.apache.hadoop.hive.ql.udf.generic.GenericUDF;
import org.apache.hadoop.hive.ql.udf.generic.GenericUDFUtils;
import org.apache.hadoop.hive.serde2.objectinspector.ObjectInspector;

@Description(name = "nvl",
        value = "nvl(value, default_value) - Returns default value if value is null else returns value",
        extended = "Example: SELECT nvl(null, default_value);")
public class MyUDF extends GenericUDF {

    private GenericUDFUtils.ReturnObjectInspectorResolver returnOIResolver;
    private ObjectInspector[] argumentOIs;

    /**
     * Determine the output parameter type based on the input parameter type of the function.
     */
    public ObjectInspector initialize(ObjectInspector[] arguments) throws UDFArgumentException {
        argumentOIs = arguments;
        if(arguments.length != 2) {
            throw new UDFArgumentException("The operator 'NVL' accepts 2 arguments.");
        }
        returnOIResolver = new GenericUDFUtils.ReturnObjectInspectorResolver(true);
        if(!(returnOIResolver.update(arguments[0]) && returnOIResolver.update(arguments[1]))) {
            throw new UDFArgumentTypeException(2, "The 1st and 2nd args of function NLV should have the same type, "
                    + "but they are different: \""+arguments[0].getTypeName()+"\" and \"" + arguments[1].getTypeName() + "\"");
        }
        return returnOIResolver.get();
    }

    /**
     * Compute the result. The data type of the final result will determine the return value type of the function, based on the
return value type of the initialize method.
     */
    public Object evaluate(DeferredObject[] arguments) throws HiveException {
        Object retVal = returnOIResolver.convertIfNecessary(arguments[0].get(), argumentOIs[0]);
        if(retVal == null) {
            retVal = returnOIResolver.convertIfNecessary(arguments[1].get(), argumentOIs[1]);
        }
        return retVal;
    }

    /**
     * Obtain the string to be displayed in the explain section.
     */
    public String getDisplayString(String[] children) {
        StringBuilder builder = new StringBuilder();
        builder.append("if ");
        builder.append(children[0]);
        builder.append(" is null ");
        builder.append("returns ");
        builder.append(children[1]);
        return builder.toString();
    }
}
```

Taking Method 2 as an example, compile the custom code into a jar package. In the directory where pom.xml is located, execute the following command to create a jar package.

```
mvn clean package -DskipTests
```

Upon completion of the UDF development work, a jar package named hive–udf–1.0–SNAPSHOT.jar will appear in the target directory.

Utilizing UDF

Upload the produced jar to the EMR cluster Master node:

```
scp ./target/hive-udf-1.0-SNAPSHOT.jar root@${master_public_ip}:/usr/local/service/hive
```

Switch to the hadoop user and execute the following command to upload the jar to HDFS:

```
su hadoop
hadoop fs -put ./hive-udf-1.0-SNAPSHOT.jar /
```

View the jar uploaded to HDFS:

```
hadoop fs -ls /
Found 5 items
drwxr-xr-x   - hadoop supergroup          0 2023-08-22 09:20 /data
drwxrwx---   - hadoop supergroup          0 2023-08-22 09:20 /emr
-rw-r--r--   2 hadoop supergroup       3235 2023-08-22 15:39 /hive-udf-1.0-SNAPSHOT.jar
drwx-wx-wx   - hadoop supergroup          0 2023-08-22 09:20 /tmp
drwxr-xr-x   - hadoop supergroup          0 2023-08-22 09:20 /user
```

Connecting to Hive:

```
hive
```

Execute the following command to create a function using the generated JAR package:

```
hive> create function nvl as "org.example.MyUDF" using jar "hdfs:///hive-udf-1.0-SNAPSHOT.jar";
```

Note:

1. 'nvl' is the name of the UDF function.
2. 'org.example.MyUDF' is the fully qualified name of the class created in the project.
3. 'hdfs:///user/hive/warehouse/hiveudf–1.0–SNAPSHOT.jar' is the path for uploading the JAR package to HDFS.

The following message indicates successful creation:

```
Added [/data/emr/hive/tmp/1b0f12a6-3406-4700-8227-37dec721297b_resources/hive-udf-1.0-SNAPSHOT.jar] to class path
Added resources: [hdfs:///hive-udf-1.0-SNAPSHOT.jar]
OK
Time taken: 1.549 seconds
```

You can also verify the successful creation of the function by using the command SHOW FUNCTIONS LIKE '*nvl*'.
Execute the following command to use the UDF function. This function can be accessed directly by using its name, just like an inbuilt function:

```
hive> select nvl("tur", "def");
OK
tur
Time taken: 0.344 seconds, Fetched: 1 row(s)
hive> select nvl(null, "def");
OK
def
Time taken: 0.471 seconds, Fetched: 1 row(s)
```

Best Practice
Mapping Hbase Tables

Last updated：2023-12-25 17:00:04

Utilize Hive to map Hbase tables, enabling the reading of data on Hbase via Hive. Execute queries, insertions, and other operations on Hbase tables using Hive-SQL statements.

Development Preparations

- Ensure that Tencent Cloud has been activated and an EMR cluster has been created. During the creation of the EMR cluster, select Hive and Hbase components in the software configuration interface.
- Hive and related software are installed in the `/usr/local/service/` directory of the EMR cloud server.

Create an Hbase Table

Initially, it is necessary to log into any machine in the EMR cluster, preferably the Master node. For methods on how to log into EMR, please refer to Logging into Linux Instances. Here, we can opt to use WebShell for login. Click on the login button on the right side of the corresponding cloud server to enter the login interface. The default username is 'root', and the password is the one entered by the user when creating EMR. After entering correctly, you can access the command line interface.
In the EMR command line, first use the following command to switch to the Hadoop user, enter the Hbase folder, and then enter the Hbase shell:

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

Establish a new 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 operations in Hbase, refer to the Hbase Operation Guide, or consult the Official Documentation.
Upon completion, 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 Hive Tables

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 to map it to the Hbase table created in the second step:

```
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
```

Thus, a mapping from a Hive table to an Hbase table has been established. The following command can be used 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)
```

Practices on Loading JSON Data to Hive

Last updated：2023-12-25 17:00:16

Connecting to Hive

Log into the Master node of the EMR cluster, switch to the Hadoop user, and navigate to the Hive directory:

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

Preparing the Data

Creating a data file (in JSON format):

```
vim test.data
```

Compile the following content and save it:

```
{"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]}
```

Storing the data file on HDFS:

```
hadoop fs -put ./test.data /
```

Creating the Table

Connecting to Hive:

```
[hadoop@10 hive]$ hive
```

Creating a table based on the mapping relationship:

```
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;
```

Importing Data

```
hive>LOAD DATA INPATH '/test.data' into table test;
```

Verifying the Successful Import of Data

Querying 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)
```

Querying the first score of each record:

```
hive> select grade[0] from test;
```

```
OK
99
91
Time taken: 0.374 seconds, Fetched: 2 row(s)
```

Querying the name and location of the first course for 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)
```

Accessing Iceberg Data via Hive

Last updated：2023-12-25 17:00:28

Development Preparations

- Ensure that you have activated Tencent Cloud and have established an EMR cluster. For more details, refer to Creating a Cluster.
- During the creation of the EMR cluster, select Hive, Spark, and Iceberg components in the software configuration interface.

Creating an Iceberg Table via Spark

Log into the Master node, switch to the Hadoop user, and execute the following command to initiate SparkSQL:

```
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-spark-runtime-3.2_2.12-
0.13.0.jar
```

Note:
The Iceberg-related packages are located in the /usr/local/service/iceberg/ directory. The dependency package version used by --jars may vary across different EMR versions. Please check and use the correct dependency package.

Table Creation:

```
spark-sql> CREATE TABLE local.default.t1 (id int, name string) USING iceberg;
Time taken: 2.752 seconds
```

Data Insertion:

```
spark-sql> INSERT INTO local.default.t1 values(1, "tom");
Time taken: 2.71 seconds
```

Data Query:

```
spark-sql> SELECT * from local.default.t1;
1       tom
Time taken: 0.558 seconds, Fetched 1 row(s)
```

Viewing Iceberg Data through Hive:

Log into the Master node and switch to the Hadoop user, then execute the following command to connect to Hive:

```
hive
```

Adding Iceberg Dependency Packages:

```
hive> add jar /usr/local/service/iceberg/iceberg-hive-runtime-0.13.0.jar;
```

Creating an External Table:

```
hive> CREATE EXTERNAL TABLE t1
STORED BY 'org.apache.iceberg.mr.hive.HiveIcebergStorageHandler'
LOCATION '/usr/hive/warehouse/default/t1'
TBLPROPERTIES ('iceberg.catalog'='location_based_table');
```

Querying the Record Count of Table t1:

```
hive> select count(*) from t1;
OK
1
Time taken: 26.255 seconds, Fetched: 1 row(s)
```

Hive Accessing Hudi Data

Last updated：2023-12-25 17:00:54

Development Preparations

- Ensure that you have activated Tencent Cloud and have established an EMR cluster. For more details, refer to Creating a Cluster.
- During the creation of the EMR cluster, select Hive, Spark, and Hudi components in the software configuration interface.

Reading and Writing Hudi via Spark

Log into the master node and switch to the Hadoop user. Utilize the HoodieSparkSessionExtension extension of SparkSQL for data reading and writing:

```
spark-sql --master yarn \
--num-executors 2 \
--executor-memory 1g \
--executor-cores 2 \
--jars /usr/local/service/hudi/hudi-bundle/hudi-spark3.3-bundle_2.12-0.13.0.jar \
--conf 'spark.serializer=org.apache.spark.serializer.KryoSerializer' \
--conf 'spark.sql.extensions=org.apache.spark.sql.hudi.HoodieSparkSessionExtension' \
--conf 'spark.sql.catalog.spark_catalog=org.apache.spark.sql.hudi.catalog.HoodieCatalog'
```

Note:
The --master represents your master URL, --num-executors denotes the number of executors, and --executor-memory signifies the storage capacity of the executor. These parameters can be modified according to your actual situation. The --jars dependency package version may vary across different EMR versions, **please check and use the correct dependency package in the /usr/local/service/hudi/hudi-bundle directory**.

Table Creation:

```
-- Creating a COW (Copy-On-Write) Non-Partitioned Table


spark-sql> create table hudi_cow_nonpcf_tbl (
  uuid int,
  name string,
  price double
) using hudi
tblproperties (
  primaryKey = 'uuid'
);


-- Creating a COW (Copy-On-Write) Partitioned Table


spark-sql> create table hudi_cow_pt_tbl (
  id bigint,
  name string,
  ts bigint,
  dt string,
  hh string
) using hudi
tblproperties (
  type = 'cow',
  primaryKey = 'id',
  preCombineField = 'ts'
 )
```

```
partitioned by (dt, hh);


-- Creating a MOR (Merge-On-Read) Partitioned Table


spark-sql> create table hudi_mor_tbl (
  id int,
  name string,
  price double,
  ts bigint,
  dt string
) using hudi
tblproperties (
  type = 'mor',
  primaryKey = 'id',
  preCombineField = 'ts'
)
partitioned by (dt);
```

Data Input:

```
-- insert into non-partitioned table
spark-sql> insert into hudi_cow_nonpcf_tbl select 1, 'a1', 20;


-- insert dynamic partition
spark-sql> insert into hudi_cow_pt_tbl partition (dt, hh) select 1 as id, 'a1' as name, 1000 as ts, '2021-12-09' as dt, '10' as hh;


-- insert static partition
spark-sql> insert into hudi_cow_pt_tbl partition(dt = '2021-12-09', hh='11') select 2, 'a2', 1000;
spark-sql> insert into hudi_mor_tbl partition(dt = '2021-12-09') select 1, 'a1', 20, 1000;
```

Utilizing Hive to Query Hudi Tables

Log into the Master node and switch to the Hadoop user, then execute the following command to connect to Hive:

```
hive
```

Incorporating Hudi Dependency Packages:

```
hive> add jar /usr/local/service/hudi/hudi-bundle/hudi-hadoop-mr-bundle-0.13.0.jar;
```

Inspecting the Table:

```
hive> show tables;
OK
hudi_cow_nonpcf_tbl
hudi_cow_pt_tbl
hudi_mor_tbl
hudi_mor_tbl_ro
hudi_mor_tbl_rt
Time taken: 0.023 seconds, Fetched: 5 row(s)
```

Data Query:

```
hive> select * from hudi_cow_nonpcf_tbl;
OK
```

```
20230905170525412	20230905170525412_0_0	1		8d32a1cc-11f9-437f-9a7b-8ba9532223d3-0_0-17-
15_20230905170525412.parquet	1	a1	20.0
Time taken: 1.447 seconds, Fetched: 1 row(s)

hive> set hive.input.format=org.apache.hadoop.hive.ql.io.HiveInputFormat;
hive> select * from hudi_mor_tbl_ro;
OK
20230808174602565  20230808174602565_0_1   id:1  dt=2021-12-09    af40667d-1dca-4163-89ca-2c48250985b2-0_0-34-
1617_20230808174602565.parquet   1  a1    20.0 1000    2021-12-09
Time taken: 0.159 seconds, Fetched: 1 row(s)


hive> set hive.vectorized.execution.enabled=false;
hive> select name, count(*) from hudi_mor_tbl_rt group by name;
a1    1
Time taken: 17.618 seconds, Fetched: 1 row(s)
```

Utilize Hive to create a library table within COS/CHDFS.

Last updated：2023-12-25 17:01:05

This document elucidates the process of utilizing Hive to establish libraries and tables on COS and CHDFS.

Development Preparations

- Ensure that you have activated Tencent Cloud and have established an EMR cluster. During the creation of the EMR cluster, it is necessary to select the Hive component in the software configuration interface.
- The example includes content that requires access to Tencent Cloud Object Storage (COS). You may refer to Create Bucket to create a bucket in COS, and enable object storage authorization on the EMR console Instance Information page.
- The example includes content that necessitates access to Tencent Cloud Object Storage CHDFS. You may refer to Mount CHDFS to create a mount point and attach it to the EMR cluster.

Creating Libraries and Tables on COS Using Hive

Note:
EMR has integrated Hadoop-COS by default. After enabling object storage authorization on the EMR console, the role bound to the Tencent Cloud EMR instance will be used to obtain a temporary key for accessing COS. This method is more secure compared to using a fixed key.
For other configuration methods, please refer to Object Storage-Hadoop Tools.

Method One: Establish the Entire Database on COS

Log into the Master node of the EMR cluster, switch to the Hadoop user, and execute the following command to enter the Hive command line:

```
hive
```

Execute the following command to create a database named 'hivewithcos' under your COS bucket.

```
hive> create database hivewithcos location 'cosn://${bucketname}/${path}';
```

Note:
Where ${bucketname} is the name of the COS bucket you created, and ${path} is the storage path.

Upon reviewing the execution results, you can see the 'hivewithcos' database we created on COS:

```
hive> show databases;
OK
default
hivewithcos
Time taken: 0.094 seconds, Fetched: 2 row(s)
```

Create a data table named 'record' (the method of loading data into the table is the same as HDFS):

```
hive> use hivewithcos;
hive> create table record(id int, name string) row format delimited fields terminated by ',' stored as textfile;
```

Inspecting the Table:

```
hive> show tables;
OK
```

```
record
Time taken: 0.063 seconds, Fetched: 1 row(s)
```

Method Two: Placing a Specified Table on COS

Create a database within Hive:

```
hive> create database test;
hive> use test;
```

Execute the following statement to create a table named 'record' under the COS bucket path (the method of loading data into the table is the same as HDFS):

```
hive> create table record(id int, name string) row format delimited fields terminated by ',' stored as textfile location
'cosn://$bucketname/$path';
```

Inspecting the Table:

```
hive> show tables;
OK
record
Time taken: 0.063 seconds, Fetched: 1 row(s)
```

Utilize Hive to Create a Library Table in CHDFS

Method One: Establishing the Entire Database on CHDFS

Log into the Master node of the EMR cluster, switch to the Hadoop user, and execute the following command to enter the Hive command line:

```
hive
```

Execute the command below to create a database named 'hivewithofs' in your CHDFS directory:

```
hive> create database hivewithofs location 'ofs://${mountpoint}/${path}';
```

Note:
Where ${mountpoint} is the CHDFS mount address you created, and ${path} is the path.

Upon reviewing the execution results, we can observe the 'hivewithofs' database that we have created on CHDFS:

```
hive> show databases;
OK
default
hivewithofs
Time taken: 0.094 seconds, Fetched: 2 row(s)
```

Create a data table named 'record' (the method of loading data into the table is the same as HDFS)

```
hive> use hivewithofs;
hive> create table record(id int, name string) row format delimited fields terminated by ',' stored as textfile;
```

Inspecting the Table:

```
hive> show tables;
OK
```

```
record
Time taken: 0.063 seconds, Fetched: 1 row(s)
```

Method Two: Placing a Specific Table on CHDFS

Create a database named 'test2' in Hive:

```
hive> create database test2;
hive> use test2;
```

Execute the following statement to create a table named 'record' under CHDFS:

```
hive> create table record(id int, name string) row format delimited fields terminated by ',' stored as textfile location
'cosn://$mountpoint/$path';
```

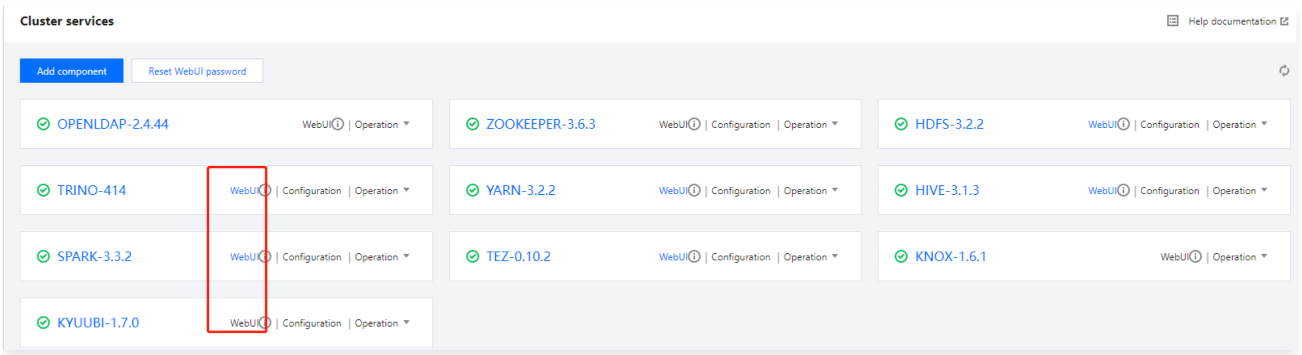
View the table:

```
hive> show tables;
OK
record
Time taken: 0.063 seconds, Fetched: 1 row(s)
```

Presto Development Guide
Presto Web UI

Last updated：2023-12-26 14:37:52

Presto is an open-source distributed SQL query engine, suitable for interactive analytical queries, supporting data volumes from gigabytes to petabytes. Presto's design and development were aimed at addressing the issues of interactive analysis and processing speed in large-scale, even ultra-large-scale, commercial data warehouses.
Presto supports online data queries, encompassing Hive, Cassandra, relational databases, and proprietary data stores. A single Presto query can amalgamate data from these multiple sources, enabling analysis across the entire organization.
EMR proxies the native Presto Web UI, which can be directly viewed in the EMR console. Log in to the EMR Console, click on the **Cluster Instance ID** to enter the instance management page. Click on the **Cluster Services** in the left menu bar to see the WebUI address page shortcut. Click on the Presto entry. The login username is root, and the password is the one set when creating the cluster. As shown below:


Cluster services
Help documentation
Add component
Reset WebUI password
OPENLDAP-2.4.44 WebUI | Operation
ZOOKEEPER-3.6.3 WebUI | Configuration | Operation
HDFS-3.2.2 WebUI | Configuration | Operation
TRINO-414 WebUI | Configuration | Operation
YARN-3.2.2 WebUI | Configuration | Operation
HIVE-3.1.3 WebUI | Configuration | Operation
SPARK-3.3.2 WebUI | Configuration | Operation
TEZ-0.10.2 WebUI | Configuration | Operation
KNOX-1.6.1 WebUI | Operation
KYUUBI-1.7.0 WebUI | Configuration | Operation

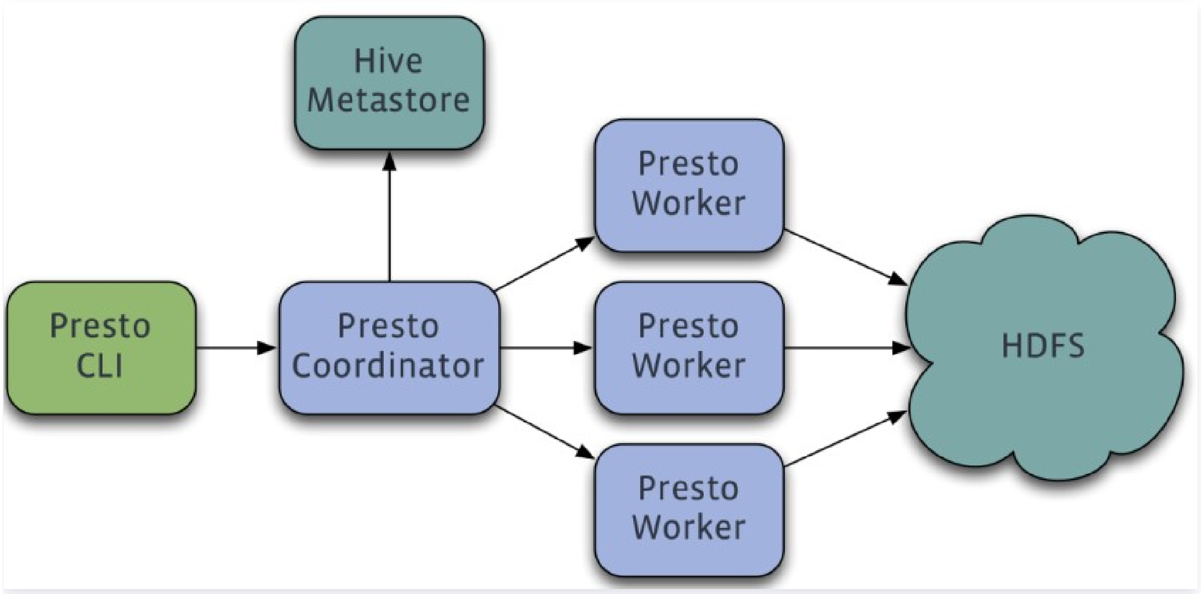
The access address requires authentication, the username is root, and the default password is the one entered when creating the cluster. If you need to change the password, you can click on **Reset WebUI Password** on this page to modify it.

Note:
Adding configuration files and using cluster scripts cannot utilize /data/emr/prestosql/etc, /data/emr/presto/etc.

Connector

Last updated：2023-12-26 14:38:48

Presto, a distributed SQL query engine developed by Facebook, is specifically designed for high-speed, real-time data analysis. It is suitable for interactive analytical queries, supporting data volumes from gigabytes to petabytes. It adheres to standard ANSI SQL, encompassing complex queries, aggregations, joins, and window functions, all implemented in Java. Presto's data sources range from Hive, HBase, and relational databases to proprietary data storage. The architecture diagram is as follows:



Presto is a distributed system operating across multiple servers, employing a master-slave architecture that includes a primary node, the Coordinator, and several secondary nodes, the Workers. The Presto CLI client is responsible for submitting queries to the Coordinator node. The Coordinator node, in turn, is tasked with parsing SQL statements, generating query execution plans, and managing Worker nodes. The Worker nodes are charged with the actual execution of query tasks.
Within EMR, the Presto component comes pre-configured with connectors such as Hive, MySQL, and Kafka. This section will illustrate the use of Presto to query table information from Hive, using the Hive connector as an example. The EMR cluster machine is configured with relevant environment variables for the presto-client, allowing for direct switching to the Hadoop user and utilization of the Presto client tool.

Development Preparations

- Ensure that you have activated Tencent Cloud and have created an EMR cluster. During the creation of the EMR cluster, it is necessary to select the Presto component in the software configuration interface.
- Presto and related software are installed in the `/usr/local/service/` directory of the EMR cloud server.

Utilizing Connectors to Operate Hive

Initially, it is necessary to log into any machine in the EMR cluster, preferably the Master node. For methods on how to log into EMR, please refer to Logging into Linux Instances. Here, we can opt to use WebShell for login. Click on the login button on the right side of the corresponding cloud server to enter the login interface. The default username is 'root', and the password is the one entered by the user when creating EMR. After entering correctly, you can access the command line interface.
In the EMR command line, first use the following command to switch to the Hadoop user and enter the Presto directory:

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

Inspect 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
```

Where $host is your host address and $port is your port number. Then switch to the presto-client directory and use Presto to connect to Hive:

```
[hadoop@172 presto]# cd /usr/local/service/presto/presto-client
[hadoop@172 presto-client]$ ./presto --server $host:$port --catalog hive --schema default
```

Wherein, the --catalog parameter signifies the type of database to be manipulated, and --schema represents the database name, here entering the default 'default' database. For more parameter information, you can view it through the command `presto -h`, or refer to the official documentation.
Upon successful execution, you can enter the Presto interface and directly access the specified database. You can use Hive-SQL to view the tables 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' was established in the Hive Development Guide.
For more Presto operations, please refer to the official documentation.

Analyzing Data in COS

Last updated：2023-12-26 14:39:09

This section will elucidate on the extensive utilization methods of the Hive connector, based on Tencent Cloud Object Storage (COS). The data is derived from direct data insertion, COS data, and lzo compressed data.

Development Preparations

- As the task requires access to Tencent Cloud Object Storage (COS), it is necessary to first create a storage bucket (Bucket) in COS.
- Ensure that you have activated Tencent Cloud and have established an EMR cluster. During the creation of the EMR cluster, it is necessary to select the Presto component in the software configuration interface, and authorize object storage on the basic configuration page.
- Presto and other related software are installed in the `/usr/local/service/` directory of the EMR cloud server.

Data Preparation

Initially, it is necessary to log into any machine in the EMR cluster, preferably the Master node. For methods on how to log into EMR, please refer to Logging into Linux Instances. Here, we can opt to use WebShell for login. Click on the login button on the right side of the corresponding cloud server to enter the login interface. The default username is 'root', and the password is the one entered by the user when creating EMR. After entering correctly, you can access the command line interface.
In the EMR command line, first use the following command to switch to the Hadoop user and enter the Hive folder:

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

Create a new file named cos.txt and add the following data:

```
5,cos_patrick
6,cos_stone
```

Use the HDFS command to upload the file to COS. Here, $bucketname refers to the name and path of the storage bucket you created.

```
[hadoop@172 hive]# hdfs dfs -put cos.txt cosn://$bucketname/
```

Next, create a new file named lzo.txt and add the following data:

```
10,lzo_pop
11,lzo_tim
```

Compress it into a .lzo file:

```
[hadoop@172 hive]$ lzop -v lzo.txt
compressing hive_test.data into lzo.txt.lzo
```

Note
To compress the lzo file, you first need to install lzo and lzop. Execute the following command to install: `yum -y install lzo lzop` .

Create a new Hive table and query it using Presto.

A script file is utilized here to generate and create the Hive database and table. Create a new script file named presto_on_cos_test.sql and add the following program:

```
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;
```

Where $bucketname is the name of your COS bucket plus the path, and $yourpath is the path where you place the lzo.txt.lzo file. The script file initially creates a new database "test", and within this newly created database, a table "presto_on_cos" is established. The data insertion operation is carried out in three steps. Firstly, a direct insertion method is adopted. Subsequently, data from COS is inserted, and finally, data from the lzo compressed package is inserted.
It is recommended, as shown in the example, to use an external table for Hive testing to avoid deleting important data. Execute this script using hive-cli:

```
[hadoop@172 hive]$ hive -f "presto_on_cos_test.sql"
```

Upon completion of execution, you can enter Presto to view the data in the table. Use the method from the previous section to enter Presto, but the schema parameter needs to be modified.

```
[hadoop@172 presto-client]$ ./presto --server $host:$port --catalog hive --schema test
```

Conduct a query on the recently created Hive table:

```
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 Presto operations, please refer to the official documentation.

Sqoop Development Guide
Import/Export of Relational Database and HDFS

Last updated：2023-12-26 14:39:27

Sqoop is an open-source tool, primarily designed for data transfer between Hadoop and traditional databases such as MySQL, PostgreSQL, and others. It facilitates the importation of data from a relational database (like MySQL, Oracle, Postgres, etc.) into Hadoop's HDFS, and vice versa. A significant highlight of Sqoop is its ability to import data from relational databases into HDFS via Hadoop's MapReduce.
This document elucidates the methodology of utilizing Tencent Cloud's Sqoop service for importing/exporting data between MySQL and HDFS.

Development Preparations

- Ensure that Tencent Cloud is activated and an EMR cluster has been established. During the creation of the EMR cluster, it is necessary to select the Sqoop component in the software configuration interface.
- Sqoop and related software are installed in the path `/usr/local/service/` of the EMR cloud server.

Create a New MySQL Table

Initially, connect to the already created MySQL database, enter the EMR console, and copy the instance ID of the target cluster, which is the name of the cluster. Then, enter the relational database console, use Ctrl+F to search, locate the MySQL database corresponding to the cluster, and view the internal network address $mysqlIP of this database.
Log into any machine in the EMR cluster, preferably the Master node. For the method of logging into EMR, please refer to Logging into Linux Instance. Here, we can choose to log in using WebShell. Click on the login on the right side of the corresponding cloud server to enter the login interface. The username is set to root by default, and the password is the one entered by the user when creating EMR. After entering correctly, you can access the command line interface.
In the EMR command line, first use the following command to switch to the Hadoop user and enter the Sqoop folder:

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

Connect to MySQL Database:

```
[hadoop@172 sqoop]$ mysql -h $mysqlIP -p
Enter password:
```

The password is the one you set when creating the EMR cluster.
After connecting to the MySQL database, enter the test database and create a new table. Users can also choose their target 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 additional columns titled 'title', 'time', and 'content'. Data is inserted into this 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)
```

The following command can be used 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 MySQL Database:

```
Mysql> exit;
```

Importing MySQL data into HDFS

Use sqoop-import 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
```

The --connect is used to connect to the MySQL database, 'test' can be replaced with your database name, -P indicates that a password will be required, --table is the name of the database you wish to export, and --target-dir is the path to export to in HDFS. The ** `/sqoop` ** directory was not created prior to executing the command, and an error will occur if the directory already exists.
After pressing enter, you will be prompted to input your password, which is the password you set when creating EMR.
Upon 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 data from HDFS into MySQL

Initially, a new table needs to be created in MySQL 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)
```

After verifying the successful creation of the table, exit MySQL:

```
mysql> show tables;
+-----------------+
| Tables_in_test  |
+-----------------+
| sqoop_test      |
| sqoop_test_back |
+-----------------+
2 rows in set (0.00 sec)

mysql> exit;
```

Utilize sqoop-export to import the data from HDFS, which was imported in the previous step, back into MySQL:

```
[hadoop@172 sqoop]$ bin/sqoop-export --connect jdbc:mysql://$mysqlIP/test --username
root -P --table sqoop_test_back --export-dir /sqoop
```

The parameters are similar to sqoop-import, except it has been changed to --export-dir, which is the path for storing data in HDFS. After pressing enter, you will also need to input the password.
Upon successful execution, the data in the sqoop_test_back database can be verified:

```
[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 Sqoop operations, refer to the official documentation.

Incremental Data Import into HDFS

Last updated：2023-12-26 14:40:38

Sqoop is an open-source tool, primarily designed for data transfer between Hadoop and traditional databases such as MySQL, PostgreSQL, and others. It facilitates the importation of data from a relational database (like MySQL, Oracle, Postgres, etc.) into Hadoop's HDFS, and vice versa. A significant highlight of Sqoop is its ability to import data from relational databases into HDFS via Hadoop's MapReduce.
This document elucidates the incremental import operations of Sqoop, that is, synchronizing the modifications in the database to the data imported into HDFS when data in the database is increased or updated. This operation is divided into two modes: append and lastmodified. The append mode is exclusively applicable in scenarios where the database data increases but does not update, while the lastmodified mode is used in scenarios where data increases and updates.

Development Preparations

- Ensure that you have activated Tencent Cloud and have created an EMR cluster. During the creation of the EMR cluster, it is necessary to select the Sqoop component in the software configuration interface.
- Sqoop and related software are installed in the path `/usr/local/service/` of the EMR cloud server.

Utilizing the Append Mode

This section will continue to utilize the use case from the previous section.
Enter the EMR Console, copy the instance ID of the target cluster, which is the name of the cluster. Then, proceed to the relational database console, use Ctrl+F to search, locate the MySQL database corresponding to the cluster, and view the internal address $mysqlIP of this database.
Log into any machine in the EMR cluster, preferably the Master node. For the method of logging into EMR, please refer to Logging into Linux Instance. Here, we can choose to log in using WebShell. Click on the login on the right side of the corresponding cloud server to enter the login interface. The username is set to root by default, and the password is the one entered by the user when creating EMR. After entering correctly, you can access the command line interface.
In the EMR command line, first use the following command to switch to the Hadoop user and enter the Sqoop folder:

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

Connect to MySQL Database:

```
[hadoop@172 sqoop]$ mysql -h $mysqlIP -p
Enter password:
```

The password is the one you set when creating the EMR cluster.
After connecting to the MySQL database, add a new piece of data to the sqoop_test table as follows:

```
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)
```

Utilize the append mode to synchronize the newly added data to the HDFS path where the data was stored in the previous section:

```
[hadoop@172 sqoop]$ bin/sqoop-import --connect jdbc:mysql://$mysqlIP/test --username
root -P --table sqoop_test --check-column id  --incremental append --last-value 3 --target-dir
/sqoop
```

Where $mysqlIP is the internal address of your MySQL database.
Executing the command will require you to input the database password, which by default is the password you set when creating the EMR cluster. This command has more parameters than the standard sqoop-import command, where --check-column is the data referred to during import, --incremental is the import mode, in this case, append, and --last-value is the reference value of the reference data. All data updated beyond this value will be imported into HDFS.
Upon successful execution, you can view the updated data in the corresponding HDFS directory:

```
[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
```

Utilize Sqoop job

Using append to synchronize data in HDFS requires manual input of --last-value each time. Alternatively, you can use the sqoop job method, where Sqoop will automatically save the last-value from the last successful import. To use sqoop job, you need to start the sqoop-metastore process. The operation steps are as follows:
Firstly, initiate the sqoop-metastore process in conf/sqoop-site.xml:

```
<property>
  <name>sqoop.metastore.client.enable.autoconnect</name>
  <value>true</value>
</property>
```

Then, initiate the sqoop-metastore service in the bin directory:

```
./sqoop-metastore &
```

Use the following command to create a Sqoop job:

Note
This command is applicable for Sqoop version 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 id
--incremental append --last-value 4 --target-dir /sqoop
```

Where $mysqlIP is the internal address of your MySQL. By using this command, a Sqoop job is successfully created. Each execution will automatically update from the last-value updated previously.
Add a new record to the sqoop_test table in MySQL:

```
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
```

Executing this command will prompt you to enter the MySQL password. Upon successful execution, you can view the updated data in the corresponding HDFS directory:

```
[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
```

Utilizing the lastmodified mode:

To directly create a Sqoop job in lastmodified mode for sqoop-import, first query the last updated 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 --incremental lastmodified --merge-key id --last-value '2018-07-03 16:02:29' --target-dir
/sqoop
```

Parameter Description:

- $mysqlIP refers to the internal network address of your MySQL.
- The --check-column must utilize a column of timestamp type.
- The --incremental mode is set to lastmodified.
- The --merge-key is set to ID.
- The --last-value represents the most recent update time we have queried in the table. Any updates made after this time will be synchronized to HDFS, and the Sqoop job will automatically save and update this value each time.

Add data to the sqoop_test table in MySQL and make modifications:

```
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
```

Executing this command will prompt you to enter the MySQL password. Upon successful execution, you can view the updated data in the corresponding HDFS directory:

```
[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 Sqoop operations, refer to the official documentation.

Importing and Exporting Data Between Hive and TencentDB for MySQL

Last updated：2023-12-26 14:40:50

This document elucidates the methodology of utilizing Tencent Cloud's Sqoop service to reciprocally import and export data between MySQL and Hive.

Development Preparations

- Ensure that Tencent Cloud has been activated and an EMR cluster has been established. During the creation of the EMR cluster, it is necessary to select Sqoop and Hive components in the software configuration interface.
- Sqoop and related software are installed in the path `/usr/local/service/` of the EMR cloud server.

Importing Relational Database into Hive

This section will continue to utilize the use case from the previous section.
Enter the Elastic MapReduce Console, copy the instance ID of the target cluster, which is the name of the cluster. Then, enter the relational database console, use Ctrl+F to search, find the MySQL database corresponding to the cluster, and view the internal address $mysqlIP of this database.
Log into any machine in the EMR cluster, preferably the Master node. For the method of logging into EMR, please refer to Logging into Linux Instance. Here, we can choose to log in using WebShell. Click on the login on the right side of the corresponding cloud server to enter the login interface. The username is set to root by default, and the password is the one entered by the user when creating EMR. After entering correctly, you can access the command line interface.
In the EMR command line, first use the following command to switch to the Hadoop user and enter the Hive folder:

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

Create a new Hive database:

```
[hadoop@172 hive]$ hive
hive> create database hive_from_sqoop;
OK
Time taken: 0.167 seconds
```

Use the sqoop-import command to import the MySQL database created in the previous section into Hive:

```
[hadoop@172 hive]# cd /usr/local/service/sqoop
[hadoop@172 sqoop]$ bin/sqoop-import --connect  jdbc:mysql://$mysqlIP/test --username
root -P --table sqoop_test_back --hive-database hive_from_sqoop --hive-import --hive-table hive_from_sqoop
```

- $mysqlIP: The internal address of Tencent Cloud Relational Database (CDB).
- test: The name of the MySQL database.
- --table: The name of the MySQL table to be exported.
- --hive-database: The name of the Hive database.
- --hive-table: The name of the Hive table to be imported.

Executing the command requires entering your MySQL password, which by default is the password set when creating your EMR cluster. Upon successful execution, the imported database can be viewed 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)
```

Importing Hive into a relational database.

Sqoop facilitates the import of data from Hive tables into relational databases. Initially, a new table is created in Hive and data is imported.
Log into any machine in the EMR cluster, preferably the Master node. In the EMR command line, first use the following command to switch to the Hadoop user and enter the Hive directory:

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

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

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

And execute in the following manner:

```
[hadoop@172 hive]$ ./gen_data.sh > hive_test.data
```

This script file will generate 1,000,000 pairs of random numbers and save them to the file named hive_test.data.
Use the following command to upload the generated test data to HDFS:

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

Where $hdfspath is the path on HDFS where you store your files.
Connect to Hive and create a test table:

```
[hadoop@172 hive]$ bin/hive
hive> create database hive_to_sqoop;        #Create database hive_to_sqoop
OK
Time taken: 0.176 seconds
hive> use hive_to_sqoop;                    #Switch database
OK
Time taken: 0.176 seconds
hive> create table hive_test (a int, b string)
hive> ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';
                                       #Create data table 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 data
```

$hdfspath is the path where you store files on HDFS.
Upon successful completion, you can use the `quit` command to exit the Hive data warehouse. Connect to the relational database and create the corresponding tables:

```
[hadoop@172 hive]$ mysql -h $mysqlIP –p
Enter password:
```

Where $mysqlIP is the internal address of the database, and the password is the one you set when creating the cluster.

Create a table named test in MySQL, **The field names in MySQL must match exactly with the field names in Hive:**

```
mysql> create table table_from_hive (a int,b varchar(255));
```

Upon successful creation of the table, you may exit MySQL.
There are two methods to import data from the Hive data warehouse into the relational database using Sqoop. You can either use the Hive data stored directly in HDFS, or you can use Hcatalog for data import.

Utilizing the Hive data in HDFS

Switch to the Sqoop folder, then use the following command to export data from the Hive database to the relational database:

```
[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_test
```

Where $mysqlIP is the private IP address of your relational database, test is the name of the database in the relational database, the parameter following --table is the name of your table in the relational database, and the parameter following --export-dir is the location where the data in the Hive table is stored in HDFS.

Importing using Hcatalog

Switch to the Sqoop folder, then use the following command to export data from the Hive database to the relational database:

```
[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_test
```

Where $mysqlIP is the private IP address of your relational database, test is the name of the database in the relational database, the parameter following --table is the name of your table in the relational database, the parameter following --hcatalog-database is the name of the database where the Hive table to be exported is located, and the parameter following --hcatalog-table is the name of the table to be exported from Hive.
Upon completion of the operation, you can enter the relational database to verify whether the import was 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;     #The table now contains 1,000,000 records.
+----------+
| count(*) |
+----------+
| 1000000 |
+----------+
1 row in set (0.03 sec)
mysql> select * from table_from_hive limit 10;    #View the first 10 records 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)
```

For more parameters related to the sqoop-export command, you can view them using the following command:

```
[hadoop@172 bin]$ ./sqoop-export --help
```

Importing the Hive table in ORC format into the relational database.

ORC is a columnar file storage format that significantly enhances the performance of Hive. This section introduces how to create a table in ORC format, load data into it, and then use Tencent Cloud's Sqoop service to export data stored in Hive in ORC format to a relational database.

Note
Importing a Hive table in ORC storage format into a relational database cannot directly use the data stored in HDFS, but must be operated through Hcatalog.

This section will continue to utilize the use case from the previous section.
After logging into the Master node of the EMR cluster, switch to the Hadoop user in the EMR command line and enter the Hive folder using the following command:

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

Create a new table in the hive_from_sqoop database 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 FIELDS TERMINATED BY ',' stored as orc;
```

The storage format of the data in the table can be viewed using 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)
```

The returned data indicates that the data storage format in this table is ORC.
There are several methods to import data into a Hive table in ORC format. The following primarily introduces importing data into an ORC format table by creating a temporary Hive table in text storage format. Here, we use the hive_test table created in the previous section as a temporary table, and import data using the following command:

```
hive> insert into table orc_test select * from hive_test;
```

After successful import, the data in the table can be viewed using the `select` command.
Then, use Sqoop to export the Hive table in ORC format to MySQL. Connect to the relational database and create the corresponding table. For specific methods of connecting to the relational database, refer to the above text:

```
[hadoop@172 hive]$ mysql -h $mysqlIP –p
Enter password:
```

Where $mysqlIP is the internal address of the database, and the password is the one you set when creating the cluster.
Create a table named test in MySQL, **The field names in MySQL must match exactly with the field names in Hive:**

```
mysql> create table table_from_orc (a int,b varchar(255));
```

Upon successful creation of the table, you may exit MySQL.
Switch to the Sqoop folder, then use the following command to export data stored in ORC format in the Hive database to the relational database:

```
[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_test
```

Where $mysqlIP is the private IP address of your relational database, test is the name of the database in the relational database, the parameter following ––table is the name of your table in the relational database, the parameter following ––hcatalog–database is the name of the database where the Hive table to be exported is located, and the parameter following ––hcatalog–table is the name of the table to be exported from Hive.
After successful import, you can view the data in the corresponding table in MySQL:

```
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 Sqoop operations, refer to the official documentation.

Oozie Development Guide

Last updated：2023-12-26 14:41:26

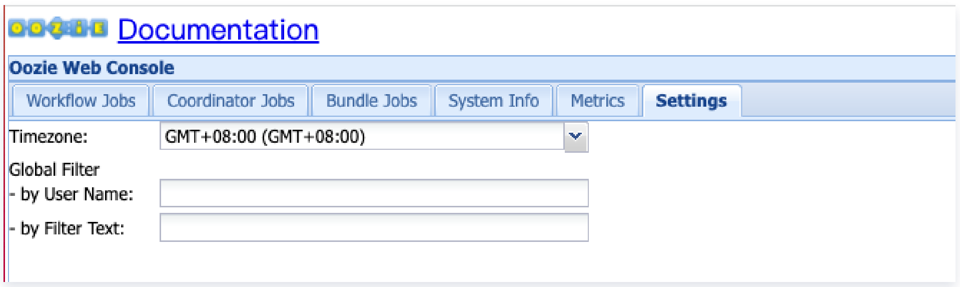
Apache Oozie is an open-source workflow engine, designed to orchestrate tasks from the Hadoop ecosystem into Workflows, subsequently scheduling, executing, and managing them. This article provides a brief introduction on how to utilize Oozie on EMR through an example. For a more comprehensive guide, please refer to the community documentation.
Additionally, it is recommended for users to operate Oozie through Hue's graphical interface. For usage instructions, please refer to the Hue Development Documentation.

Preparations

An Elastic MapReduce (EMR) Hadoop cluster has been established, with Oozie, Hive, and Spark services selected. For more details, please refer to Creating an EMR Cluster.

Accessing Oozie WebUI

- If you have enabled external network for cluster nodes during the purchase, you can access it by clicking on the WebUI link under **Cluster Services** in the EMR console.
- For domestic users, it is suggested to set the WebUI timezone to GMT+08:00.



Timezone: GMT+08:00 (GMT+08:00)
Global Filter
- by User Name:
- by Filter Text:

Updating the sharelib

Within the EMR cluster, sharelib has already been installed, thus there is no need for additional installation when submitting Workflow jobs using Oozie. However, you can still edit and update sharelib, with the steps as follows:

```
cd /usr/local/service/oozie
tar -xf oozie-sharelib.tar.gz
```

Add the jar package to the corresponding directory of the action to be supported in the unzipped share directory:

```
bin/oozie-setup.sh sharelib create -fs hdfs://active-namenode-ip:4007 -locallib shareoozie admin --oozie http://oozie-server-
ip:12000/oozie -sharelibupdate
```

Submitting Workflow Example

The installation directory for Oozie in the EMR cluster is /usr/local/service/oozie, which contains the Workflow examples oozie-examples.tar.gz for the components supported by Oozie. After entering the installation directory, switch to the Hadoop user and unzip the example files and upload them to hdfs using the following command:

```
su hadoop
tar -xf oozie-examples.tar.gz
hadoop fs -put examples
```

Note:
Parameters in the following example:
1, **$fs** is the access link for HDFS, which can be viewed in the core-site.xml configuration file fs.defaultFS value through **Cluster Services>HDFS>Configuration Management** on the EMR console cluster details page.
2, **$rm** is the address of the Yarn ResourceManager, which can be viewed in the yarn-site.xml configuration file through **Cluster Services>YARN>Configuration Management** on the EMR console cluster details page. The value is yarn.resourcemanager.ha.rm-ids in a high-availability deployment, and yarn.resourcemanager.address in a non-high-availability deployment.

3, **$hs2_ip** is the internal IP of the HiveServer2 service deployment node, which can be viewed in **Cluster Services>HIVE>Role Management** on the cluster details page.
4, **$oozie_server_ip** is the internal IP of the Oozie service deployment node, which can be viewed in **Cluster Services>OOZIE>Role Management** on the cluster details page.
5, **$job** is the id returned after submitting an Oozie Job.

Hive Task Example

The job.properties file under the example file ./examples/apps/hive2 is the configuration file for the Hive example task. Modify the HDFS and YARN addresses, HiveServer2 address, etc. in this file according to the actual situation:

```
nameNode=$fs
resourceManager=$rm
queueName=default
jdbcURL=jdbc:hive2://$hs2_ip:7001/default
examplesRoot=examples
```

Submitting an Oozie job:

```
oozie job -debug -oozie http://$oozie_server_ip:12000/oozie -config examples/apps/hive2/job.properties -run
```

You can execute the following command or visit the Oozie web UI to view job running information.

```
oozie job -info $job
```

Spark Task Example

The job.properties file under the example file ./examples/apps/spark is the configuration file for the Spark example task. Modify the HDFS and YARN addresses in this file according to the actual situation:

```
nameNode=$fs
resourceManager=$rm
master=local[*]
mode=client
queueName=default
examplesRoot=examples
oozie.use.system.libpath=true
```

Submitting an Oozie job:

```
oozie job -debug -oozie http://$oozie_server_ip:12000/oozie -config examples/apps/spark/job.properties -run
```

You can execute the following command or visit the Oozie web UI to view job running information.

```
oozie job -info $job
```

Flume Development Guide
Flume Overview

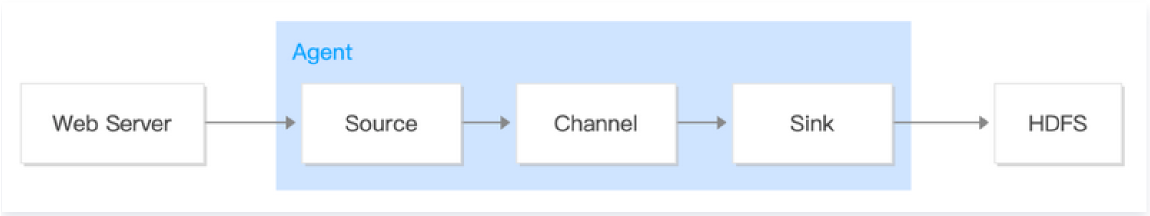
Last updated：2023-12-26 14:41:43

Introduction to Flume

Apache Flume is a tool/service capable of collecting data resources such as logs and events, and consolidating these vast quantities of data from various sources for storage. Flume boasts features such as high availability, distributed architecture, and configurable tools. Its design principle is to aggregate data streams (such as log data) from various web servers and store them in centralized storage systems like HDFS and HBase.

Flume Architecture

A Flume event is defined as a unit of data flow. A Flume agent is essentially a JVM process that encompasses all the components necessary to accomplish a task, with the three core components being the Source, Channel, and Sink.



- **Source**
  Consumes events passed to it from external sources (such as web servers or other Sources) and stores them in one or more Channels.
- **Channel**
  The Channel, positioned between the Source and the Sink, serves as a cache for incoming events. Once the Sink successfully dispatches the events to the next Channel or their final destination, the events are removed from the Channel.
- **Sink**
  The Sink is responsible for transmitting events to the next hop or their final destination. Upon successful completion, the events are removed from the Channel.

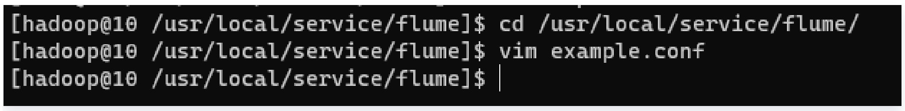
User guides

Preparations

- An EMR cluster has been created. During the creation of the EMR cluster, it is necessary to select the Flume component in the software configuration interface.
- Flume is installed on the EMR cloud server (core nodes and task nodes) under the `/usr/local/service/flume` path; the installation path for the master node is `/usr/local/service/apps/`.

Configuring Flume

Navigate to the `/usr/local/service/flume` directory and create an example.conf file.



```
[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
```

Launching Flume

```
bin/flume-ng agent --conf conf --conf-file example.conf --name a1 -Dflume.root.logger=INFO,console
```

Configuring Test Case

After configuration, you will see the previously launched Flume Agent 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：2023-12-26 14:41:55

Description

Collect data from Kafka via Flume and store it in Hive.

Development Preparations

- As the task requires access to Tencent Cloud's message queue CKafka, it is necessary to first create a CKafka instance, as detailed in Message Queue CKafka.
- Ensure that you have activated Tencent Cloud and have created an EMR cluster. When creating the EMR cluster, you need to select the Flume component in the software configuration interface.

Utilizing the Kafka toolkit in the EMR cluster

Initially, it is necessary to view the internal network IP and port number of CKafka. Log into the Message Queue CKafka console, select the CKafka instance you wish to use, and view its internal network IP as $kafkaIP in **Basic Information**. The port number is generally set to 9092 by default. Create a new topic named kafka_test in the topic management interface.

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
# Configuring the following 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
# Configuring the following 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 = weblogs
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
# Configuring the following channel
agent.channels.mem_channel.type = memory
agent.channels.mem_channel.capacity = 100000
agent.channels.mem_channel.transactionCapacity = 100000
```

The hive.metastore can be confirmed in the following manner:

```
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 weblogs ( 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**
> It is imperative that the table is partitioned and bucketed, stored as ORC, and set with TBLPROPERTIES ('transactional'='true'). None of these conditions can be omitted.

3. Initiate Hive transaction
Add the following configuration items to `hive-site.xml` via the console.

```
<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 issued and restarted, the `hadoop-hive` log will indicate that the metastore cannot connect. Please disregard this error. Due to the process startup sequence, the metastore needs to be launched before hiveserver2.

4. Copy Hive's `hive-hcatalog-streaming-xxx.jar` 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. Execute Flume

```
./bin/flume-ng agent --conf ./conf/ -f hive_kafka.properties -n agent -Dflume.root.logger=INFO,console
```

6. Execute Kafka producer

```
[hadoop@172 kafka]$ ./bin/kafka-console-producer.sh --broker-list $kafkaIP:9092 --topic kafka_test
1,hello
2,hi
```

Test

- Enter information in the Kafka producer client and press return.
- Observe whether there is corresponding data in the Hive table.

Reference

- Configuration Guide for Hive-Sink
- Configuration Guide for Hive Logs

Storing Kafka Data in HDFS or COS Through Flume

Last updated：2023-12-26 14:42:13

Description

Collect data from Kafka via Flume and store it in HDFS or COS.

Development Preparations

- As the task requires access to Tencent Cloud's message queue CKafka, it is necessary to first create a CKafka instance, as detailed in Message Queue CKafka.
- Ensure that you have activated Tencent Cloud and have created an EMR cluster. When creating the EMR cluster, you need to select the Flume component in the software configuration interface, and enable the authorization of object storage on the basic configuration page.

Utilizing the Kafka toolkit in the EMR cluster

Initially, it is necessary to view the internal network IP and port number of CKafka. Log into the Message Queue CKafka console, select the CKafka instance you wish to use, and view its internal network IP as $kafkaIP in the basic message, while the port number is generally set to 9092 by default. Create a new topic named kafka_test in the topic management interface.

Configuring Flume

1. Create the Flume configuration file `kafka.properties`

```
vim kafka.properties
agent.sources = kafka_source
agent.channels = mem_channel
agent.sinks = hdfs_sink
# Configuring the following 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
# Configuring the following sink
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/xxx)
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
# Configuring the following channel
agent.channels.mem_channel.type = memory
agent.channels.mem_channel.capacity = 100000
agent.channels.mem_channel.transactionCapacity = 10000
```

2. Execute Flume

```
./bin/flume-ng agent --conf ./conf/ -f kafka.properties -n agent -Dflume.root.logger=INFO,console
```

3. Execute Kafka producer

```
[hadoop@172 kafka]$ ./bin/kafka-console-producer.sh --broker-list $kafkaIP:9092 --topic kafka_test
test
hello
```

Test

- Enter information in the Kafka producer client and press return.
- Observe whether HDFS generates the corresponding directory and file `hadoop fs -ls /data/flume/kafka/` .

Reference

Kafka–source Configuration Guide

Storing Kafka Data in Hive Through Flume

Last updated：2023-12-26 14:42:24

Description

Collect data from Kafka via Flume and store it in Hbase.

Development Preparations

- As the task requires access to Tencent Cloud's message queue CKafka, it is necessary to first create a CKafka instance, as detailed in Message Queue CKafka.
- Ensure that you have activated Tencent Cloud and have created an EMR cluster. When creating the EMR cluster, you need to select the Flume component in the software configuration interface.

Utilizing the Kafka toolkit in the EMR cluster

Initially, it is necessary to view the internal network IP and port number of CKafka. Log into the Message Queue CKafka console, select the CKafka instance you wish to use, and view its internal network IP as $kafkaIP in the basic message, while the port number is generally set to 9092 by default. Create a new topic named kafka_test in the topic management interface.

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
# Configuring the following 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
# Configuring the following sink
agent.sinks.hbase_sink.type = hbase2
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.RegexHBase2EventSerializer
# Configuring the following channel
agent.channels.mem_channel.type = memory
agent.channels.mem_channel.capacity = 100000
agent.channels.mem_channel.transactionCapacity = 10000
```

2. Creating an HBase table

```
hbase shell
create 'foo_table','cf'
```

3. Execute Flume

```
./bin/flume-ng agent --conf ./conf/ -f hbase_kafka.properties -n agent -Dflume.root.logger=INFO,console
```

4. Execute Kafka producer

```
[hadoop@172 kafka]$ ./bin/kafka-console-producer.sh --broker-list $kafkaIP:9092 --topic kafka_test
hello
hbase_test
```

Test

- Enter information in the Kafka producer client and press return.
- Observe whether there is corresponding data in the HBase table.

Reference

HBase-Sink Configuration Guide

Kerberos Development Guide
Kerberos Overview

Last updated：2023-12-26 14:42:43

The current EMR-V2.1.0 and subsequent versions all support the creation of secure clusters via Kerberos, meaning that all supported open-source components within the cluster, with the exception of Alluxio, Zeppelin, and Kylin, are initiated in Kerberos' secure mode. In this secure environment, only authenticated clients can access the cluster's services, such as HDFS.

Note
The Impala service component only supports EMR-V3.1.0 and subsequent versions, and is initiated in Kerberos' secure mode.

Key Concepts

Full Name	Short Name	Effect
key distributed center	KDC	The entire security authentication process involves the generation and management of tickets, which includes two services: AS and TGS.
authentication service	AS	The service that generates TGT for the client.
ticket granting service	TGS	The service that generates a specific ticket for the client.
account database	AD	Stores the whitelist of all clients, only those present on the whitelist can successfully apply for a TGT.
ticket-granting ticket	TGT	Used to obtain the ticket credentials.
client	-	The client intending to access a particular server.
server	-	Provides a certain type of business service.

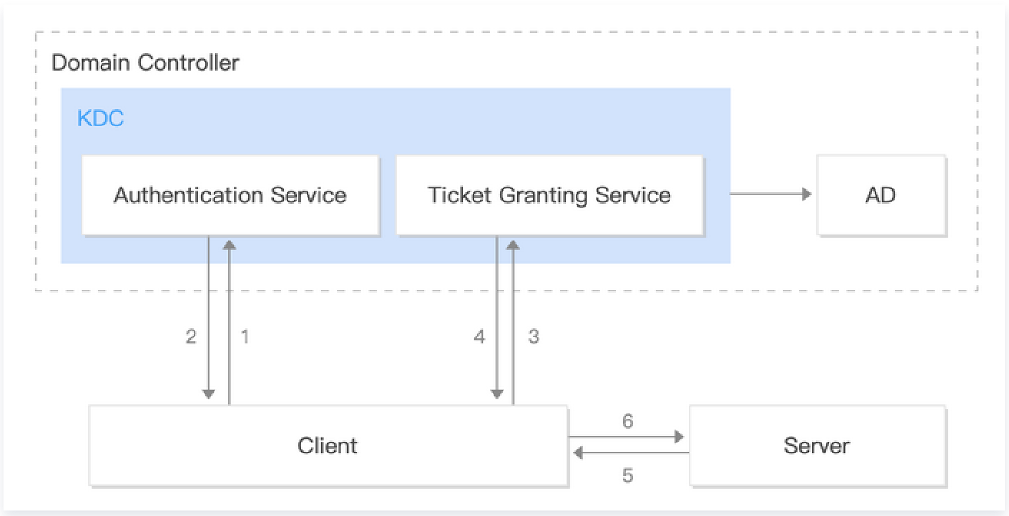
Additional Concepts

Full Name	Effect
principal	The principal of authentication, that is, the "username".
realm	A realm is somewhat akin to a namespace in programming languages. In such languages, a variable name only holds significance within a specific namespace. Similarly, a principal only carries meaning under a particular realm. Therefore, a realm can be viewed as a "container" or "space" for the principal. Correspondingly, the naming convention for a principal is what_name_you_like@realm. In Kerberos, it is a common practice to name realms in uppercase, for instance: EXAMPLE.COM.
password	The password of a user, corresponding to the master_key in Kerberos. The password can be stored in a keytab file. Therefore, in Kerberos, any scenario requiring a password can use a keytab as input.
credential	A credential serves as evidence that "a certain individual is indeed who they claim to be" or "a certain action can indeed occur". In different usage scenarios, the specific meaning of a credential may vary slightly: • For a given principal entity, their credential is their password. • In the process of Kerberos authentication, a credential signifies various types of tickets.

Authentication process

During the process of a client accessing a server, to ensure both the client and server are reliable, it is inevitable to introduce a third-party notary platform. Hence, the AS and TGS services are introduced. AS and TGS are typically located within the same service process. In the case of MIT's Kerberos implementation, both are provided by the KDC.
The authentication process is divided into the following three steps:

1. The client sends a request to the Kerberos service, seeking permission to access the server. Kerberos first determines whether the client is trustworthy, distinguishing clients through blacklists and whitelists stored in the AD. Upon successful verification, the AS returns a TGT to the client.
2. Upon obtaining the TGT, the client continues to request Kerberos for permission to access the server. Kerberos, through the TGT in the client's message, determines the client's authorization and provides the client with a permission ticket to access the server.
3. Once the client has obtained the ticket, it can access the server. However, this ticket is specific to this server. To access other servers, the client needs to reapply to the TGS.



Kerberos Use Instructions

Last updated：2023-12-26 14:42:53

This document employs MIT's Kerberos as the KDC service. Presuming the KDC service has been installed and initiated, the utilization of Kerberos initially necessitates the creation of a realm, followed by the addition of pertinent principals for various roles, inclusive of server and client, culminating in the generation of a keytab file.

Create Database

Utilize the `kdb5_util` command to establish a database, designated for the storage of information pertinent to the principal.

```
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>
```

Incorporate 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.
```

Generate 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.
```

Herein, we have established a new user: test/host@EXAMPLE.COM, and have positioned this user's key within the `/var/krb5kdc/test.keytab` file. Please modify the relevant paths in the **example code to reflect the actual paths within your project**.

Initiate KDC

```
 service krb5-kdc start
 * Starting Kerberos KDC krb5kdc
```

Kinit Authentication

```
kinit -k -t /etc/krb5.keytab test-client/host@EXAMPLE.COM
```

Kinit corresponds to the step of acquiring TGT from KDC. It will send a request to the KDC server specified in `/etc/krb5.conf`. If the TGT request is successful, it can be viewed 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
```

Utilization within the Project

Upon successful Kinit authentication, the keytab file can be duplicated onto the server and client servers that require its use, and the corresponding principal can be configured for utilization.

Accessing Hadoop Secure Cluster

Last updated：2023-12-26 14:43:05

Hadoop Command

Ticket not retrieved

When Kerberos is enabled, it is necessary to obtain a ticket in advance before executing a Hadoop command. If a ticket is not retrieved, 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 GSSException: No valid credentials provided (Mechanism level: Failed to find any Kerberos tgt)]
ls: Failed on local exception: java.io.IOException: javax.security.sasl.SaslException: GSS initiate failed [Caused by
GSSException: 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;
```

Obtain a ticket

Prerequisite: The principal of hadoop@EMR has been added.

```
kinit -kt /var/krb5kdc/emr.keytab hadoop@EMR
```

Executing the command will allow normal access.

```
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
```

Java code accessing HDFS

Utilizing a local ticket

Note
It is necessary to execute kinit in advance to obtain a ticket. The program will experience abnormal access after the ticket expires.

```
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());
    }
}
```

Employing 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：2023-12-26 14:43:16

This document elucidates the process of modifying Hadoop's configuration to integrate with Kerberos. If you have procured a secure cluster through Tencent Cloud's EMR, the system will be automatically configured, eliminating the need for manual configuration.

Preparations

- The KDC service has been successfully established.
- The creation of relevant Hadoop principals has been successfully completed.
- The keytab file has been distributed to each server (assuming the keytab file path is `/var/krb5kdc/emr.keytab`).

Hadoop Integration with Kerberos

Hadoop primarily comprises HDFS and Yarn services. It is necessary to modify the configurations of these two components separately and restart the service processes.

HDFS Integration

Modify core-site.xml

```
hadoop.security.authentication: kerberos
hadoop.security.authorization: true
```

Alter 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 option 'ignore.secure.ports.for.testing' must be set to true; otherwise, the SASL mode must be configured, and webhdfs must enable HTTPS.

Revise 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 Integration

Modify 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
```

Alter mapred-site.xml

```
mapreduce.jobhistory.keytab: /var/krb5kdc/emr.keytab
mapreduce.jobhistory.principal: hadoop/_HOST@EMR
```

Kerberos HA Support

Last updated：2023-12-26 14:43:26

For a formal production system enabled with Kerberos, the high availability of KDC must also be considered. The Kerberos service supports configuration in a primary-secondary mode, with data synchronization achieved through the kprop service, synchronizing data from the primary node to the secondary node. Upon purchasing Tencent Cloud's EMR high-availability security cluster, Kerberos is by default highly available, requiring no configuration on the part of the user.
This document primarily elucidates the relevant configuration and usage pertaining to the high availability of the Kerberos service.

Preparations

- Tencent Cloud's EMR high-availability cluster has been procured.
- During the procurement of the cluster, the option to enable Kerberos authentication was selected.

Introduction to High Availability Configuration for KDC

Configure the `/etc/krb5.conf` file as follows:

Note
In the example, two KDC addresses are configured, the active KDC and the backup KDC. This ensures that normal KDC services can still be provided to the cluster even if any one of the KDC services experiences an anomaly.

```
[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 Synchronization

- Kprop Configuration
  By default, there are `/var/kerberos/krb5kdc/kpropd.acl` configuration files on the two KDC servers.

```
host/active_kdc@REALM
host/backup_kdc@REALM
```

1. Execute the `/var/kerberos/krb5kdc/kpropd.acl` file
   After executing the default configuration file `/var/kerberos/krb5kdc/kpropd.acl`, the `/etc/krb5.keyta` file will be generated on the two KDC servers. Simultaneously, the two KDC servers will also initiate 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
```

2. Periodic KDC Data Synchronization Command
   The EMR agent will periodically (default is every 5 minutes) synchronize the principals from the active KDC to the backup KDC, ensuring that the data on both KDCs is synchronized. The following synchronization command will be executed periodically on the 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. Execute the following command on the active KDC.

```
kadmin.local "-q addprinc -randkey test"
```

2. On the backup KDC, you can see the newly added principal, indicating that the data synchronization between the KDCs has been completed. This process requires a waiting period (default is 5 minutes).

```
kadmin.local "-q listprincs"|grep test
```

Knox Development Guide

Knox Development Guide

Last updated: 2023-12-29 10:04:51

All versions of EMR now support Apache Knox. After completing the following preparatory tasks, you can directly access the Web UI services of Yarn, HDFS, and others from the public network.

Preparations

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

Initiating Access to Knox

Access via the public IP address of the cluster is recommended. It is advisable to modify the security group rules of the CVM with a public IP in the cluster, restricting the access IP end of the TCP:30002 port to your IP end.

1. View the public IP through the cluster details.
2. Access the corresponding service's URL in your browser.
    - HDFS UI: https://{public IP of the cluster}:30002/gateway/emr/hdfs
    - Yarn UI: https://{public IP of the cluster}:30002/gateway/emr/yarn
    - Hive UI: https://{public IP of the cluster}:30002/gateway/emr/hive
    - Hbase UI: https://{public IP of the cluster}:30002/gateway/emr/hbase/webui
    - Storm UI: https://{public IP of the cluster}:30002/gateway/emr/stormui
    - Ganglia UI: https://{public IP of the cluster}:30002/gateway/emr/ganglia/
    - Presto UI: https://{public IP of the cluster}:30002/gateway/emr/presto/
    - Oozie UI: https://{public IP of the cluster}:30002/gateway/emr/oozie/
    - Livy UI: https://{public IP of the cluster}:30002/gateway/emr/livy/v1/
    - Ranger UI: https://{public IP of the cluster}:30002/gateway/emr/ranger/
    - Alluxio UI: https://{public IP of the cluster}:30002/gateway/emr/alluxio/
    - Impala UI: https://{public IP of the cluster}:30002/gateway/emr/impalastore/
    - Ganglia UI: https://{public IP of the cluster}:30002/gateway/emr/ganglia/
    - Spark UI: https://{public IP of the cluster}:30002/gateway/emr/sparkhistory/
    - Tez UI: https://{public IP of the cluster}:30002/gateway/emr/tez/
3. If your browser displays **Your connection is not private**, it is because the Knox service uses a self-signed certificate. Please reconfirm that you are accessing the public IP of your own cluster and that the port is 30002. Choose **Advanced > Proceed**.
4. In the authentication login box that appears, the username is 'root', and the default password is the one entered when creating the cluster. It is recommended that you change your password. You can do so by clicking on **Reset Native UI Password** on this page.

Alluxio Development Guide
Alluxio Development Documentation

Last updated：2023-12-29 10:05:37

Example

Alluxio provides data access through a file system interface. Files within Alluxio offer write-once semantics: they are immutable once fully written, and unreadable before completion. Alluxio offers two distinct file system APIs: the Alluxio API and the Hadoop-compatible API. The Alluxio API provides additional functionality, while the Hadoop-compatible API offers users the flexibility to use the Hadoop API without modifying existing code.
All resources utilizing the Alluxio Java API are implemented via paths specified by AlluxioURI.

Acquiring the File System Client

To acquire the Alluxio file system client using Java code, please utilize:

```
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. As Alluxio files are immutable once written, the conventional method for creating a file is to use `FileSystem#createFile(AlluxioURI)` , which returns a stream object that can be used for writing to the file. For instance:

```
FileSystem fs = FileSystem.Factory.get();
AlluxioURI path = new AlluxioURI("/myFile");
// Create a file and get its output stream
FileOutStream out = fs.createFile(path);
// Write data
out.write(...);
// Close and complete file
out.close();
```

Specifying Operation Options

For all file system operations, an additional 'options' field can be specified, allowing users to define non-default settings for the operation. For instance:

```
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);
FileOutStream out = fs.createFile(path, options);
```

IO Options

Alluxio utilizes two distinct types of storage: Alluxio managed storage and under storage. Alluxio managed storage refers to memory, SSD, or HDD allocated to the Alluxio worker. Under storage, on the other hand, pertains to resources managed by the underlying storage system, such as S3, Swift, or HDFS. Users can specify their interaction with Alluxio managed storage through ReadType and WriteType. ReadType determines the data reading behavior when accessing a file, while WriteType outlines the writing behavior when creating a new file, for instance, whether the data should be written to Alluxio Storage.
Below is the expected behavior table for ReadType. Reading always prefers Alluxio storage over under storage.

Read Type	Behavior
CACHE_PROMOTE	• If the data being read is on the Worker, it is moved to the topmost tier of the Worker. • If the data is not present in the Alluxio storage of the local Worker, then a replica is added to the local

	Alluxio Worker. • If `alluxio.user.file.cache.partially.read.block` is set to true, data blocks that have not been fully read will also be **completely** stored in Alluxio. Conversely, a data block can only be cached once it has been fully read.
CACHE	• If the data is not present in the Alluxio storage of the local Worker, then a replica is added to the local Alluxio Worker. • If `alluxio.user.file.cache.partially.read.block` is set to true, data blocks that have not been fully read will also be **completely** stored in Alluxio. Conversely, a data block can only be cached once it has been fully read.
NO_CACHE	Simply read the data, without storing a replica in Alluxio.

Below is the expected behavior table for WriteType.

Write Type	Behavior
CACHE_THROUGH	The data is synchronously written to both the Alluxio Worker and the underlying storage system.
MUST_CACHE	The data is synchronously written to the Alluxio Worker, but it is not written to the underlying storage system. This is the default write type.
THROUGH	The data is synchronously written to the underlying storage system, but it is not written to the Alluxio Worker.
ASYNC_THROUGH	The data is synchronously written to the Alluxio Worker and asynchronously written to the underlying storage system. This is in the experimental stage.

Location Strategy

Alluxio provides a location strategy to determine which worker should store the file block. Using Alluxio's Java API, users can set the strategy in CreateFileOptions for writing files to Alluxio and in OpenFileOptions for reading files.
Users can easily override the default policy class property `alluxio.user.file.write.location.policy.class` in the configuration file. The built-in policies include:

- LocalFirstPolicy (alluxio.client.file.policy.LocalFirstPolicy): Initially, the local node is returned. If the local worker does not have sufficient block capacity, a worker is randomly selected from the list of active workers. This is the default policy.
- MostAvailableFirstPolicy (alluxio.client.file.policy.MostAvailableFirstPolicy): Returns the worker with the most available bytes.
- RoundRobinPolicy (alluxio.client.file.policy.RoundRobinPolicy): Selects the next worker in a cyclical manner, skipping any worker that does not have sufficient capacity.
- SpecificHostPolicy (alluxio.client.file.policy.SpecificHostPolicy): Returns the worker with the specified node name. This policy cannot be set as the default policy.

Alluxio supports custom policies, so you can also formulate your own policy by implementing the interface `alluxio.client.file.policyFileWriteLocationPolicy` .

Note
The default policy must have an empty constructor. And when using the ASYNC_THROUGH write type, all blocks of the file must be written to the same worker.

Alluxio allows clients to select a tier preference when writing data blocks to a local worker. Currently, this policy preference is only applicable to local workers and not remote workers; remote workers will write to the top tier. By default, data is written to the top tier. Users can modify the default settings through the `alluxio.user.file.write.tier.default` configuration item, or override it by calling the `FileSystem#createFile(AlluxioURI)API` .
All operations on existing files or directories require the user to specify an AlluxioURI. With AlluxioURI, users can use any method in FileSystem to access resources. AlluxioURI can be used to perform Alluxio FileSystem operations, such as modifying file metadata, ttl or pin status, or obtaining an input stream to read a file. For example, to read a file:

```
FileSystem fs = FileSystem.Factory.get();
AlluxioURI path = new AlluxioURI("/myFile");
```

```
// Open the file for reading
FileInStream in = fs.openFile(path);
// Read data
in.read(...);
// Close file relinquishing the lock
in.close();
```

Common Alluxio Commands

Last updated：2023-12-29 10:06:01

Action	Syntax	Description
cat	cat "path"	Display the content of a file from Alluxio on the console.
checkConsistency	checkConsistency "path"	Verify the metadata consistency between Alluxio and the underlying storage system.
checksum	checksum "path"	Calculate the MD5 checksum of a file.
chgrp	chgrp "group" "path"	Modify the ownership group of a file or folder in Alluxio.
chmod	chmod "permission" "path"	Alter the access permissions of a file or folder in Alluxio.
chown	chown "owner" "path"	Change the owner of a file or folder in Alluxio.
copyFromLocal	copyFromLocal "source path""remote path"	The "remote path" copies the file from the "source path" in the local file system to the specified path in Alluxio. If the "remote path" already exists, the command will fail.
copyToLocal	copyToLocal "remote path" "local path"	Copy the file specified by the "remote path" in Alluxio to the local file system.
count	count "path"	Output the total number of files and folders in "path" that match a given prefix.
cp	cp "src" "dst"	Duplicate a file or directory within the Alluxio file system.
du	du "path"	Output the size of a specified file or folder.
fileInfo	fileInfo "path"	Output the block data information of the specified file.
free	free "path"	Remove a file or folder in Alluxio. If the file or folder exists in the underlying storage, it can still be accessed there.
getCapacityBytes	getCapacityBytes	Obtain the capacity of the Alluxio file system.
getUsedBytes	getUsedBytes	Retrieve the number of bytes utilized in the Alluxio file system.
help	help "cmd"	Print the help information for a given command. If no command is given, print the help information for all supported commands.
leader	leader	Print the name of the current Alluxio leader master node.
load	load "path"	Load files or directories from the underlying file system into Alluxio.
loadMetadata	loadMetadata "path"	Load the metadata of files or directories from the underlying file system into Alluxio.
location	location "path"	Output the nodes containing data of a specific file.
ls	ls "path"	List all the information of direct files and directories under the given path, such as size.
masterInfo	masterInfo	Print information related to Alluxio master fault tolerance, such as the address of the leader, the list of all master addresses, and the configured Zookeeper address.
mkdir	mkdir "path1" ... "pathn"	Create directories and necessary parent directories at the given path, with multiple paths separated by spaces or tabs. If any of the paths already exist, the command fails.
mount	mount "path" "uri"	Mount the "uri" path from the underlying file system to the "path" in

		the Alluxio namespace. The "path" must not pre-exist and is generated by this command. No data or metadata is loaded from the underlying file system. Once mounted, operations on this mount path will also affect the mount point in the underlying file system.
mv	mv "source" "destination"	Move the file or directory specified by "source" to the new path specified by "destination". If "destination" already exists, the command fails.
persist	persist "path1" ... "pathn"	Persist files or directories that exist only in Alluxio to the underlying file system.
pin	pin "path"	Lock the given file to the content to prevent eviction. If it is a directory, it recursively applies to its subfiles and newly created files within.
report	report "path"	Report to the master that a file has been lost.
rm	rm "path"	Delete a file. If the input path is a directory, the command fails.
setTtl	setTtl "path" "time"	Set a file's Time To Live (TTL) duration, measured in milliseconds.
stat	stat "path"	Display information for the specified path of files and directories.
tail	tail "path"	Output the last 1KB of content from the specified file to the console.
test	test "path"	Test the attributes of a path. If the attributes are correct, return 0; otherwise, return 1.
touch	touch "path"	Create an empty file at the specified path.
unmount	unmount "path"	Unmount the underlying file path mounted at the "path" specified in Alluxio. All objects at this mount point in Alluxio will be deleted, but the underlying file system will retain them.
unpin	unpin "path"	Unlock a file to make it eligible for eviction, if it's a directory, the action is recursive.
unsetTtl	unsetTtl "path"	Remove the ttl value of the file.

cat

Example

The cat command prints the entire content of a file in Alluxio to the console, which is very useful when users need to confirm whether the content of a file matches their expectations. If you want to copy the file to the local file system, use the copyToLocal command.

Sample:

For instance, when testing a new computational task, the cat command can be utilized to swiftly verify its output results.

```
$ alluxio fs cat /output/part-00000
```

checkConsistency

Example

The checkConsistency command compares the metadata of Alluxio and the underlying storage system for the given path. If the path is a directory, all its sub-contents will be compared. The command returns a list of all inconsistent files and directories, and the system administrator decides whether to adjust these inconsistent data. To prevent inconsistency between Alluxio and the metadata of the underlying storage system, your system should be set to modify files and directories through Alluxio, rather than directly accessing the underlying storage system for modifications.

If the –r option is used, the checkConsistency command will repair inconsistent files or directories. Files or folders that only exist in the underlying storage will be deleted from Alluxio. For files in the underlying file system that have changed content, the metadata of these files will be reloaded into Alluxio.

Note
This command necessitates a read lock request on the directory subtree to be inspected, implying that write or update operations on the files or directories of this subtree cannot be performed until the command is completed.

Sample:

For instance, the checkConsistency command can be employed to periodically inspect the integrity of the namespace.
Listing inconsistent files or directories:

```
$ alluxio fs checkConsistency /
```

Repairing inconsistent files or directories:

```
$ alluxio fs checkConsistency -r /
```

checksum

Example

The checksum command outputs the md5 value of a specific Alluxio file.

Sample:

For instance, checksum can be utilized to verify whether the content of files in Alluxio matches the content of files stored in the underlying or local file system.

```
$ alluxio fs checksum /LICENSE
```

chgrp

Example

The chgrp command can alter the group ownership of files or folders within Alluxio. Alluxio supports POSIX standard file permissions, where a group is an authorized entity within the POSIX file permission model. The file owner or superuser can execute this command to change the group ownership of a file or folder.
Adding the –R option allows for the recursive change of group ownership for subfiles and subfolders within a folder.

Sample:

The chgrp command can be employed to swiftly modify the group ownership of a file.

```
$ alluxio fs chgrp alluxio-group-new /input/file1
```

chmod

Example

The chmod command modifies the access permissions of files or folders within Alluxio, currently supporting octal mode: three octal digits correspond to the permissions of the file owner, the group, and other users respectively. Below is the correspondence table between the digits and permissions:

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	–––

Incorporating the –R option allows for the recursive alteration of permissions for subfiles and subfolders within a directory.

Sample:

The chmod command can be utilized to swiftly alter the permissions of a file.

```
$ alluxio fs chmod 755 /input/file1
```

chown

Example

The chown command is utilized to alter the owner of files or folders within Alluxio. For security considerations, only a superuser has the capability to change the ownership of a file.
Incorporating the –R option allows for the recursive alteration of ownership for subfiles and subfolders within a directory.

Use Case

The chown command can be utilized to swiftly alter the ownership of a file.

```
$ alluxio fs chown alluxio-user /input/file1
```

copyFromLocal

Example

The copyFromLocal command copies files from the local file system into Alluxio. If an Alluxio worker is present on the machine running this command, the data will be stored on this worker. Otherwise, the data will be randomly replicated to a remote node running an Alluxio worker. If the target specified by the command is a folder, then the folder and all its contents will be recursively copied into Alluxio.

Sample:

The copyFromLocal command facilitates the rapid replication of data into the Alluxio system for subsequent processing.

```
$ alluxio fs copyFromLocal /local/data /input
```

copyToLocal

Example

The copyToLocal command replicates files from Alluxio into the local file system. If the target specified by the command is a folder, then the folder and all its contents will be recursively copied.

Sample:

The copyToLocal command can be employed to swiftly download output data, thereby facilitating subsequent research or debugging.

```
$ alluxio fs copyToLocal /output/part-00000 part-00000
```

count

Example

The count command outputs the total number of files and folders in Alluxio that match a given prefix, as well as their cumulative size. This command recursively processes the contents of folders. The count command proves useful when users have predefined naming conventions for files.

Sample:

If files are named according to their creation dates, the count command can be utilized to ascertain the number and cumulative size of all files for any given date, month, or year.

```
$ alluxio fs count /data/2014
```

cp

Example

The cp command copies a file or directory within the Alluxio file system, and can also facilitate copying between the local file system and the Alluxio file system. The filescheme denotes the local file system, while the alluxioscheme or absence of a scheme signifies the Alluxio file system. If the −R option is used and the source path is a directory, the cp command will copy the entire subtree from the source path to the target path.

Sample:

For instance, the cp command can facilitate file copying between underlying file systems.

```
$ alluxio fs cp /hdfs/file1 /s3/
```

du

Example

The du command outputs the size of a file. If the specified target is a folder, this command outputs the cumulative size of all subfiles and the contents of subfolders within that folder.

Sample:

If Alluxio space is being excessively utilized, the du command can be employed to identify which folders are occupying the majority of the space.

```
$ alluxio fs du /\\*
```

fileInfo

Example

The fileInfo command is no longer supported from version 1.5 onwards, please use the stat command instead. The fileInfo command outputs the primary information of a file to the console, primarily to assist users in debugging their systems. Generally speaking, viewing file information on the Web UI is much more comprehensible.

Sample:

Utilizing the fileInfo command allows for the acquisition of a file's data block location, which proves extremely beneficial when obtaining data locality in computational tasks.

```
$ alluxio fs fileInfo /data/2015/logs-1.txt
```

free

Example

The free command requests the Alluxio master to expel all data blocks of a file from the Alluxio worker. If the command parameter is a folder, it will recursively apply to its subfiles and subfolders. This request does not guarantee immediate effect, as the data blocks of the file may be in the process of being read. The free command will return immediately after being received by the master. Note that this command does not delete any data in the underlying file system, but only affects data stored in Alluxio. Additionally, this operation does not affect metadata, which means that if the ls command is run, the file will still be displayed.

Sample:

The free command can be employed for manual management of Alluxio's data cache.

```
$ alluxio fs free /unused/data
```

getCapacityBytes

Example

The getCapacityBytes command returns the maximum byte capacity configured for Alluxio.

Sample:

Employing the getCapacityBytes command enables the verification of whether your system has been correctly initiated.

```
$ alluxio fs getCapacityBytes
```

getUsedBytes

Example

The getUsedBytes command returns the number of space bytes already utilized within Alluxio.

Sample:

Utilizing the getUsedBytes command facilitates the monitoring of the cluster's health status.

```
$ alluxio fs getUsedBytes
```

leader

Example

The leader command prints the name of the current leader master node in Alluxio.

Sample:

```
$ alluxio fs leader
```

load

Example

The load command imports data from the underlying file system into Alluxio. If an Alluxio worker is operating on the machine running this command, the data will be transferred to that worker; otherwise, the data will be randomly relocated to a worker. If the file already exists within Alluxio, the command will not perform any action. If the target of the command is a directory, its subfiles and subdirectories will be recursively loaded.

Sample:

Utilizing the load command allows for the acquisition of data intended for analytical purposes.

```
$ alluxio fs load /data/today
```

loadMetadata

Example

The loadMetadata command queries all files and directories in the local file system that match the given pathname, and creates mirrors of these files in Alluxio. This command only creates metadata, such as filenames and file sizes, and does not transfer data.

Sample:

The loadMetadata command can be employed when other systems output data to the underlying file system (bypassing Alluxio), and an application running on Alluxio requires access to this output data.

```
$ alluxio fs loadMetadata /hdfs/data/2015/logs-1.txt
```

location

Example

The location command returns the addresses of all Alluxio workers containing the data blocks of a given file.

Sample:

When employing a computational framework for a job, the location command can be utilized to debug data locality.

```
$ alluxio fs location /data/2015/logs-1.txt
```

ls

Example

The ls command lists all subfiles and subfolders under a folder, along with their file sizes, last modification times, and memory statuses. Using the ls command on a file will only display information about that file. The ls command also loads the metadata of any subdirectories under any file or directory from the underlying storage system into the Alluxio namespace, if Alluxio does not yet have this metadata. The ls command queries the underlying file system for files or directories that match the given path, and then creates a mirror file of this file in Alluxio. Only metadata, such as filenames and sizes, are loaded in this manner without any data transfer occurring.
Options:

- The −d option lists directories as regular files. For instance, `ls -d /` displays the attributes of the root directory.
- The −f option forces the loading of metadata for subdirectories within a directory. By default, metadata is only loaded when the directory is listed for the first time.
- The −h option displays file sizes in a human−readable format.
- The −p option lists all fixed files.
- The −R option recursively lists all subfiles and subfolders under the input path, and enumerates all subtrees starting from the input path.

Sample:

The ls command can be utilized to navigate the file system.

```
$ alluxio fs mount /cos/data cosn://data-bucket/
```

Verification:

```
$ alluxio fs ls /s3/data/
```

masterInfo

Example

The masterInfo command prints information related to Alluxio master fault tolerance, such as the leader's address, the address list of all masters, and the configured Zookeeper address. If Alluxio is running in single master mode, the masterInfo command will print the address of that master; if Alluxio is operating in multi-master fault tolerance mode, the masterInfo command will print the current leader's address, the address list of all masters, and the Zookeeper's address.

Sample:

The masterInfo command can be employed to print information pertinent to Alluxio master fault tolerance.

```
$ alluxio fs masterInfo
```

mkdir

Example

The mkdir command creates a new folder in Alluxio. This command can recursively create non-existent parent directories. Note that this folder will not be created in the underlying file system until a file in this folder is persisted. Using the mkdir command on an invalid or already existing path will fail.

Sample:

Administrators can employ the mkdir command to establish a fundamental folder structure.

```
$ alluxio fs mkdir /users
$ alluxio fs mkdir /users/Alice
$ alluxio fs mkdir /users/Bob
```

mount

Example

The mount command links a path in the underlying storage to an Alluxio path, and files and folders created under this path in Alluxio will be backed up in the corresponding path in the underlying file system. For more related information, access the unified namespace.
Options:
The --readonly option sets the mount point in Alluxio as read-only.
The --option `<key>=<val>` option passes an attribute to this mount point (such as S3 credential).

Sample:

The mount command can be utilized to make data from other storage systems accessible within Alluxio.

```
$ alluxio fs mount /mnt/hdfs hdfs://host1:9000/data/
```

mv

Example

The mv command moves files or folders within Alluxio to another path. The target path must not pre-exist or be a directory. If it is a directory, the file or folder will become a subfile or subfolder of that directory. The mv command only operates on metadata and does not affect the data blocks of the file. The mv command cannot operate between mount points of different underlying storage systems.

Sample:

The mv command can be utilized to relocate outdated data to non-working directories.

```
$ alluxio fs mv /data/2014 /data/archives/2014
```

persist

Example

The persist command perpetuates data from Alluxio into the underlying file system. This command operates on data, hence its execution time depends on the size of the file. Upon completion of the persistence, the file is backed up in the underlying file system, thus it remains accessible even if the data blocks of the file in Alluxio are evicted or lost.

Sample:

Upon filtering out files containing useful data from a series of temporary files, the persist command can be employed to effectuate their permanence.

```
$ alluxio fs persist /tmp/experimental-logs-2.txt
```

pin

Example

The pin command marks files or folders within Alluxio. This command operates solely on metadata and does not result in any data being loaded into Alluxio. If a file is marked within Alluxio, none of its data blocks will be evicted from the Alluxio worker. If there are too many locked files, the Alluxio worker will be left with minimal storage space, thereby preventing the caching of other files.

Sample:

If the administrator has a thorough understanding of the job execution process, the pin command can be manually employed to enhance performance.

```
$ alluxio fs pin /data/today
```

report

Example

The report command flags a file as lost to the Alluxio master. This command should only be used on files created with the Lineage API. Marking a file as lost will prompt the master to schedule a recomputation job to regenerate the file.

Sample:

The report command can be utilized to forcibly instigate the recomputation of a file.

```
alluxio fsadmin report /tmp/lineage-file
```

rm

Example

The rm command eradicates a file from both Alluxio and the underlying file system. Upon the command's return, the file becomes immediately inaccessible, though the actual data takes some time to be genuinely obliterated.
Incorporating the –R option allows for the recursive deletion of all contents within a folder before the folder itself is deleted. Adding the –U option will bypass the check for consistency between the UFS content to be deleted and Alluxio prior to attempting to delete the persistent directory.

Sample:

The rm command can be utilized to eliminate temporary files that are no longer required.

```
$ alluxio fs rm /tmp/unused-file
```

setTtl

Example

The setTtl command establishes the ttl (time-to-live) duration for a file or folder, measured in milliseconds. If the current time exceeds the sum of the file's creation time and ttl duration, the action parameter will dictate the operation to be executed. The delete operation (default) will eradicate the file from both Alluxio and the underlying file system, while the free operation will solely remove the file from Alluxio.

Sample:

Administrators, aware that certain files become redundant after a period, can employ the setTtl command with the delete operation to cleanse these files. If the aim is merely to liberate more space for Alluxio, the setTtl command with the free operation can be used to purge the file content within Alluxio.

```
$ alluxio fs setTtl -action free /data/good-for-one-day 86400000
```

stat

Example

The stat command outputs the principal information of a file or folder to the console, primarily to facilitate users in debugging their systems. Generally speaking, viewing file information on the Web UI is considerably more comprehensible.
The -f option can be specified to display information in a designated format:

- "%N": Filename.
- "%z": File size (in bytes).
- "%u": File owner.
- "%g": Name of the group to which the owner belongs.
- "%y" or "%Y": Edit time, `%y displays 'yyyy-MM-dd HH:mm:ss' (the UTC date), %Y` represents the number of milliseconds since January 1, 1970 UTC.
- "%b": Number of data blocks allocated for the file.

Sample:

For instance, utilizing the stat command can ascertain the location of a file's data blocks, which proves to be immensely beneficial when acquiring data locality in computational tasks.

```
$ alluxio fs stat /data/2015/logs-1.txt
$ alluxio fs stat /data/2015
$ alluxio fs stat -f %z /data/2015/logs-1.txt
```

tail

Example

The tail command outputs the final 1kb of content from a file to the console.

Sample:

Employing the tail command allows for the verification of whether a job's output adheres to the expected format or contains the anticipated values.

```
$ alluxio fs tail /output/part-00000
```

test

Example

The test command evaluates the attributes of a path.
Options:
-d option tests if the path is a directory.
-e option tests if the path exists.
-f option tests if the path is a file.

−s option tests if the path is empty.
−z option tests if the file length is zero.

Sample:

```
$ alluxio fs test -d /someDir
```

touch

Example

The touch command generates an empty file. The file created by this command cannot be overwritten and is predominantly used as a marker.

Sample:

Utilizing the touch command, one can generate an empty file to signify the completion of an analysis task for a folder.

```
$ alluxio fs touch /data/yesterday/_DONE_
```

unmount

Example

The unmount operation severs the link between an Alluxio path and a directory in the underlying file system. All metadata and file data from the mount point will be eradicated, yet the underlying file system will retain it. For further information, access the Unified Namespace.

Sample:

When the data in an underlying storage system is no longer required, the unmount command can be employed to remove that particular storage system.

```
$ alluxio fs unmount /s3/data
```

unpin

Example

The unpin command unmarks files or folders within Alluxio. This command solely affects metadata and does not expunge or delete any data blocks. Once a file is unpinned, the Alluxio worker can evict the data blocks of that file.

Sample:

The unpin command can be employed when the administrator is aware of a change in the data access pattern.

```
$ alluxio fs unpin /data/yesterday/join-table
```

unsetTtl

Example

The unsetTtl command eradicates the TTL of a file within Alluxio. This command solely affects metadata and does not expunge or delete any data blocks within Alluxio. The TTL value of the file can be reestablished by the setTtl command.

Sample:

In certain exceptional circumstances, when a file that was originally managed automatically requires manual management, the unsetTtl command can be utilized.

```
$ alluxio fs unsetTtl /data/yesterday/data-not-yet-analyzed
```

Mounting File System to Unified Alluxio File System

Last updated：2023-12-29 10:06:13

Example

Alluxio offers a unified namespace mechanism, permitting the mounting of other file systems within the Alluxio file system. This mechanism allows upper-level applications to utilize a unified namespace, accessing data dispersed across various systems.

Mounting COS

Example: Mounting a specific bucket from COS into the Alluxio directory.

```
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>/
```

The configuration information for COS in --option is as follows, where fs.cos.secret.key and fs.cos.secret.id can be obtained through API Key Management.

Configuration name	Explanation
fs.cos.access.key	cos secret id
fs.cos.secret.key	cos secret key
fs.cos.region	COS region name, for instance, ap-beijing.
fs.cos.app.id	User `appid`
COS_BUCKET	COS BUCKET name. Only the name is required, do not include the AppID suffix.

This command mounts the COS directory (specified by cos://bucket/xxx) under the Alluxio /cos directory.

Mounting HDFS

Example: Mounting a specific HDFS directory into the Alluxio directory.

```
alluxio fs mount /hdfs hdfs://data
```

This command mounts the HDFS /data directory under the Alluxio /hdfs subdirectory. After successful mounting, use the alluxio fs ls command to view the mounted content.

Mounting CHDFS

Example: Mounting CHDFS to Alluxio using mount.

Note
Only EMR2.5.0 and Alluxio2.3.0 or higher versions are supported.

```
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：2023-12-29 10:07:28

Overview

Tencent Cloud EMR offers an out-of-the-box Alluxio service to assist Tencent Cloud customers in rapidly implementing distributed memory-level caching acceleration and simplifying data management. Additionally, through the Tencent Cloud EMR console or API interface, users can utilize the configuration delivery function to swiftly set up multi-level caching and metadata management, and gain access to comprehensive monitoring and alerting capabilities.

Preparations

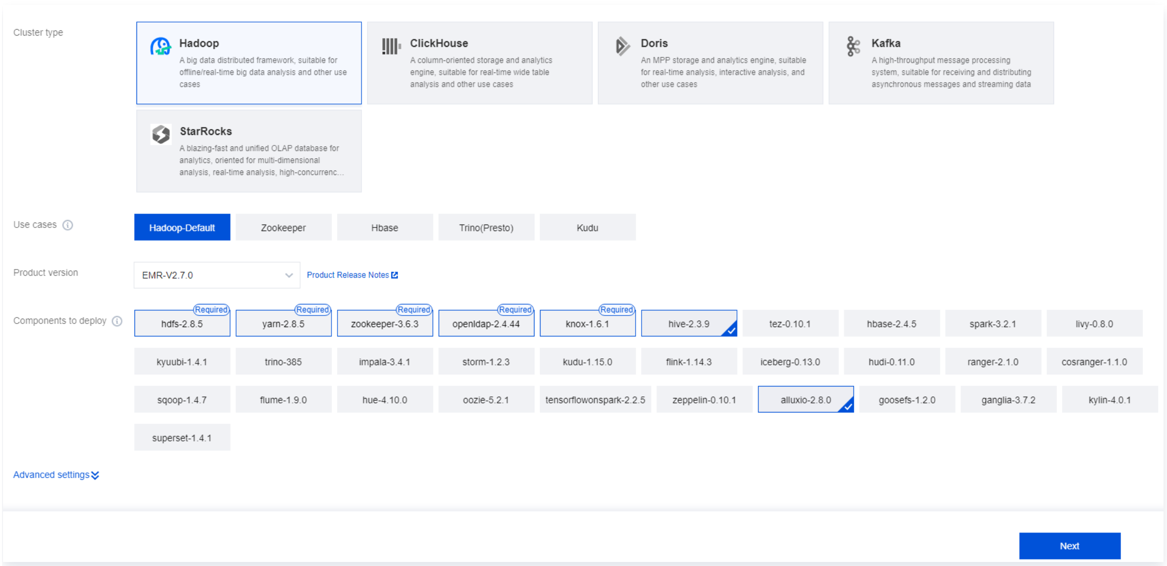
Tencent Cloud EMR supports Hadoop standard version 2.1.0 and above. For specific Alluxio versions supported by EMR, please refer to the Component Versions.

Creating an EMR Cluster Based on Alluxio

This section primarily details how to create a ready-to-use Alluxio cluster on Tencent Cloud EMR. EMR offers two methods for cluster creation: through the purchase page and via API.

Creating a Cluster via the Purchase Page

You need to log in to the Tencent Cloud EMR Purchase Page, select the supported Alluxio release version on the purchase page, and check the Alluxio component in the optional components list.



Cluster type

[✓ **Hadoop**] A big data distributed framework, suitable for offline/real-time big data analysis and other use cases

ClickHouse A column-oriented storage and analytics engine, suitable for real-time wide table analysis and other use cases

Doris An MPP storage and analytics engine, suitable for real-time analysis, interactive analysis, and other use cases

Kafka A high-throughput message processing system, suitable for receiving and distributing asynchronous messages and streaming data

StarRocks A blazing-fast and unified OLAP database for analytics, oriented for multi-dimensional analysis, real-time analysis, high-concurrenc...

Use cases: [✓ Hadoop-Default] Zookeeper Hbase Trino(Presto) Kudu

Product version: EMR-V2.7.0 Product Release Notes

Components to deploy: [✓ hdfs-2.8.5 (Required)] [✓ yarn-2.8.5 (Required)] [✓ zookeeper-3.6.3 (Required)] [✓ openldap-2.4.44 (Required)] [✓ knox-1.6.1 (Required)] [✓ hive-2.3.9] tez-0.10.1 hbase-2.4.5 spark-3.2.1 livy-0.8.0 kyuubi-1.4.1 trino-385 impala-3.4.1 storm-1.2.3 kudu-1.15.0 flink-1.14.3 iceberg-0.13.0 hudi-0.11.0 ranger-2.1.0 cosranger-1.1.0 sqoop-1.4.7 flume-1.9.0 hue-4.10.0 oozie-5.2.1 tensorflowonspark-2.2.5 zeppelin-0.10.1 [✓ alluxio-2.8.0] goosefs-1.2.0 ganglia-3.7.2 kylin-4.0.1 superset-1.4.1

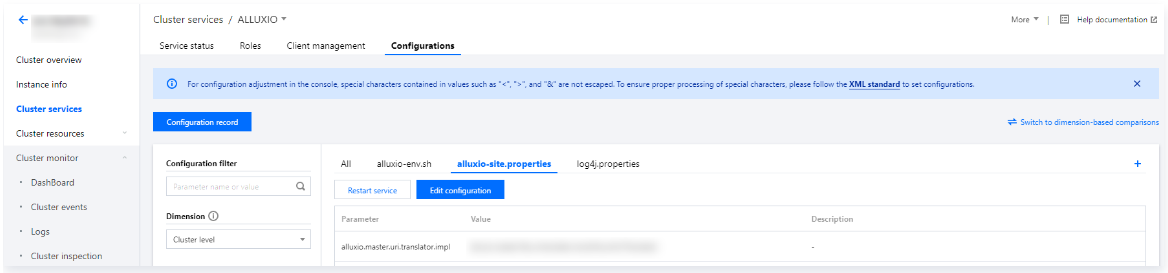
Other options can be customized according to specific business scenarios. For specific options during the creation process, please refer to Creating an EMR Cluster.

Creating a Cluster via API

Tencent Cloud EMR also offers an API method to construct a big data cluster based on Alluxio. For specifics, please refer to Querying Hardware Node Information.

Basic configurations

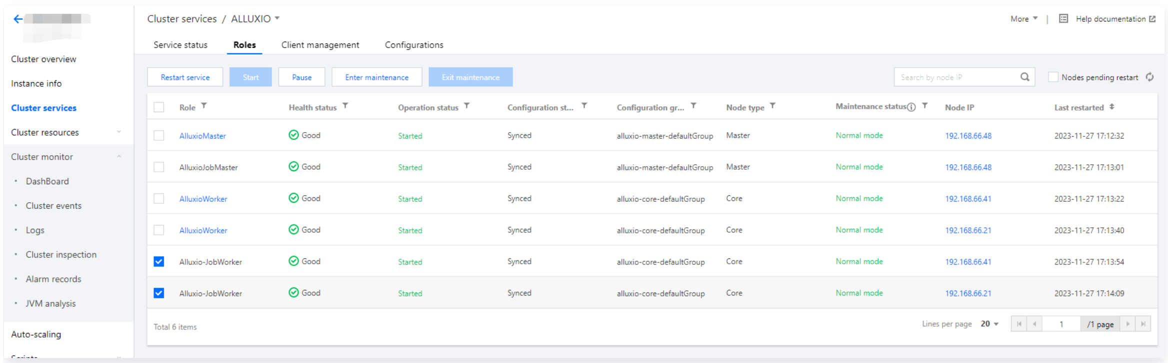
When creating a Tencent Cloud EMR cluster with an Alluxio component, HDFS will be mounted to Alluxio by default, using memory as a single-layer level0 storage. If there is a need to change to multi-level storage that better suits business characteristics, or other corresponding optimization items, you can use the configuration delivery function to complete the relevant configuration.



For configuration adjustment in the console, special characters contained in values such as "<", ">", and "&" are not escaped. To ensure proper processing of special characters, please follow the XML standard to set configurations.

Parameter	Value	Description
alluxio.master.uri.translator.impl	[redacted]	-

After configuration delivery, some configurations require a restart of the Alluxio service to take effect.



	Role	Health status	Operation status	Configuration st...	Configuration gr...	Node type	Maintenance status	Node IP	Last restarted
☐	AlluxioMaster	✓ Good	Started	Synced	alluxio-master-defaultGroup	Master	Normal mode	192.168.66.48	2023-11-27 17:12:32
☐	AlluxioJobMaster	✓ Good	Started	Synced	alluxio-master-defaultGroup	Master	Normal mode	192.168.66.48	2023-11-27 17:13:01
☐	AlluxioWorker	✓ Good	Started	Synced	alluxio-core-defaultGroup	Core	Normal mode	192.168.66.41	2023-11-27 17:13:22
☐	AlluxioWorker	✓ Good	Started	Synced	alluxio-core-defaultGroup	Core	Normal mode	192.168.66.21	2023-11-27 17:13:40
☑	Alluxio-JobWorker	✓ Good	Started	Synced	alluxio-core-defaultGroup	Core	Normal mode	192.168.66.41	2023-11-27 17:13:54
☑	Alluxio-JobWorker	✓ Good	Started	Synced	alluxio-core-defaultGroup	Core	Normal mode	192.168.66.21	2023-11-27 17:14:09

Total 6 items   Lines per page 20   1 /1 page

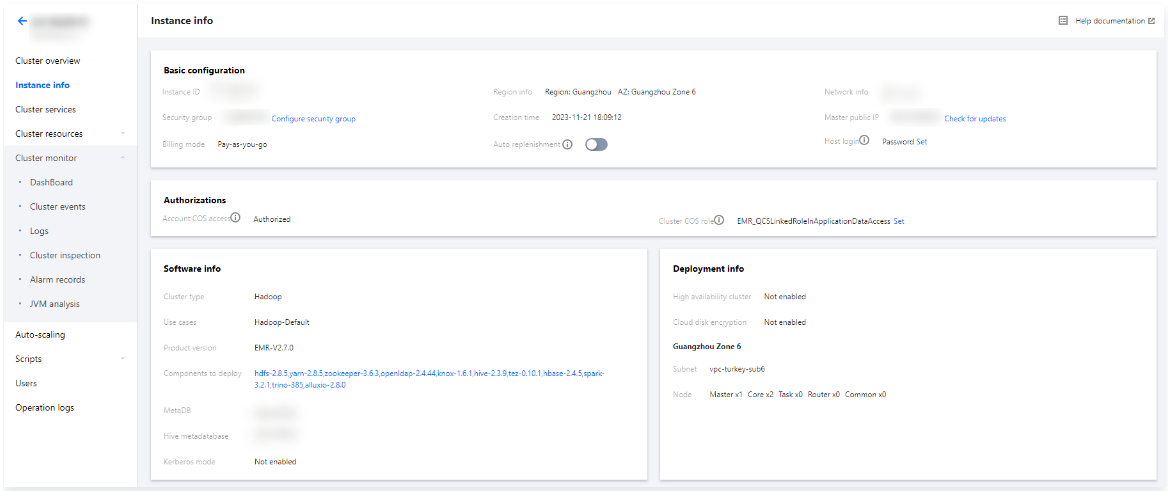
For more details on configuration delivery and restart strategies, please refer to Configuration Delivery and Restarting Components.

Accelerating Compute-Storage Separation Based on Alluxio

Tencent Cloud EMR, based on Tencent Cloud Object Storage (COS), provides compute-storage separation capabilities. By default, when directly accessing data in object storage, applications lack node-level data locality or cross-application caching. Utilizing Alluxio for acceleration will alleviate these issues.
On the Tencent Cloud EMR cluster, the dependency jar package using COS as UFS has been deployed by default. It only requires granting access permissions to COS and mounting COS to Alluxio for use.

Authorize

If the current cluster has not enabled object storage, you can authorize it in Access Management Console > Roles. After authorization, nodes in EMR can access data in COS using temporary keys.



Basic configuration

Instance ID [redacted]
Security group [redacted] Configure security group
Billing mode Pay-as-you-go
Region info Region: Guangzhou AZ: Guangzhou Zone 6
Creation time 2023-11-21 18:09:12
Auto replenishment
Network info [redacted]
Master public IP [redacted] Check for updates
Host login Password Set

Authorizations

Account COS access Authorized
Cluster COS role EMR_QCSLinkedRoleInApplicationDataAccess Set

Software info

Cluster type Hadoop
Use cases Hadoop-Default
Product version EMR-V2.7.0
Components to deploy hdfs-2.8.5,yarn-2.8.5,zookeeper-3.6.3,openldap-2.4.44,knox-1.6.1,hive-2.3.9,tez-0.10.1,hbase-2.4.5,spark-3.2.1,trino-385,alluxio-2.8.0
MetaDB [redacted]
Hive metadatabase [redacted]
Kerberos mode Not enabled

Deployment info

High availability cluster Not enabled
Cloud disk encryption Not enabled
Guangzhou Zone 6
Subnet vpc-turkey-sub6
Node Master x1 Core x2 Task x0 Router x0 Common x0

Mount

Log into any EMR machine and mount COS to Alluxio.

```
alluxio fs mount <alluxio-path> <source-path>
```

For more details on using Alluxio in Tencent Cloud EMR, please refer to the Alluxio Development Documentation.

Support for COS Transparent-URI

Last updated：2023-12-29 10:07:44

Alluxio users typically have the need to incorporate Alluxio into their existing ecosystem, accessing their underlying storage system (Under-FileSystem) through existing applications. However, the existing applications must be modified to use Alluxio's URI. The Transparent URI feature allows users to access their existing storage systems without the necessity of altering the URI 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 Edition, EMR-V3.4.0 Edition.
3. Configuration supports Transparent URI. To utilize Alluxio's Transparent URI, a new Hadoop-compatible file system client implementation needs to be configured. As long as the client is configured to accept external URIs, this new ShimFileSystem will replace the existing FileSystem. The Hadoop-compatible computing framework - Hadoop FileSystem interface defines the mapping from FileSystem scheme to FileSystem implementation. To configure ShimFileSystem, ensure the following configuration items are set in core-site.xml:

Configuration items	Configuration ItemValue
fs.cosn.impl	alluxio.hadoop.ShimFileSystem

4. When Alluxio is compatible with the Transparent URL Schema, it needs to be converted for compatibility. Please ensure the following configuration items are set in alluxio-site.properties:

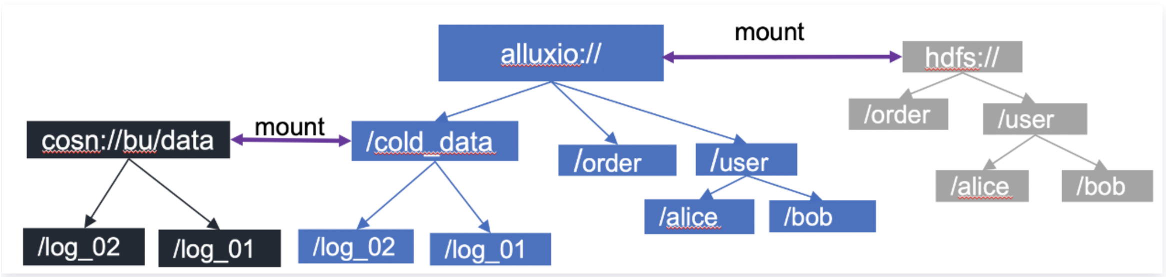
Configuration items	Configuration ItemValue
alluxio.master.uri.translator.impl	alluxio.master.file.uritranslator.AutoMountUriTranslator
alluxio.user.shimfs.bypass.ufs.impl.list	fs.cosn.impl:org.apache.hadoop.fs.cosnative.NativeCosFileSystem

Note

- After making changes to the alluxio-site.properties configuration, the Alluxio service needs to be restarted.
- Once the ShimFileSystem is configured, the master will need to route the local URI of the external storage system to the Alluxio namespace. This requires that cosn is already mounted in the Alluxio namespace.
- To disable the Transparent URI feature, simply rollback the fs.cosn.impl configuration item in core-site.xml.

mount

The mount command can be considered one of the most distinctive features of Alluxio. It is akin to the mount command in Linux - Linux users can use Linux mount to load storage devices such as hard drives and SSDs into the local file system of the Linux system. In the Alluxio system, the concept of mount is further extended to the layer of distributed systems: users can use Alluxio mount to mount one or more other storage systems/cloud storage services (such as HDFS, COS, etc.) into the Alluxio distributed file system. Consequently, distributed applications running on Alluxio, such as Spark, Presto, or MapReduce, do not need to adapt or even understand the specific data access protocols and paths, but only need to know the corresponding path in the Alluxio file system. This greatly facilitates the development and maintenance of applications.



By default, EMR-Alluxio uses hdfs as the root directory mount point.

In EMR-Alluxio2.5.1 and later, Alluxio's UFS began to support the COSN protocol. The COS UFS has issues with poor read and write performance and instability. To address these issues, the community contributed the COSN UFS underlying file system. Both COS and COSN UFS are used to access Tencent Cloud Object Storage. Compared to COS, COSN has been deeply optimized, with its read and write performance being significantly improved over COS, and it also brings better stability. Therefore, the use of COSN is highly recommended. COS UFS will cease to be maintained after the EMR-Alluxio2.6.0 version.

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
```

In this case, the COS configuration is set in the --options.

Configuration Item Name	Explanation
fs.cosn.userinfo.secretId	cos secret id
fs.cosn.userinfo.secretKey	cos secret key
fs.cosn.impl	Fixed Value: `org.apache.hadoop.fs.CosFileSystem`
fs.AbstractFileSystem.cosn.impl	Fixed Value: `org.apache.hadoop.fs.CosN`
fs.cosn.bucket.region cos region	Name, for instance, ap-beijing
fs.cosn.userinfo.appid	User's Primary Account AppID
COS_BUCKET COS BUCKET	Name. The name alone is required, without the AppID suffix.

Support for Authentication

Last updated：2023-12-29 10:08:06

Alluxio users, when accessing data on COS, HDFS, CHDFS via the existing unified namespace, or when utilizing transparent URLs to access cached data within Alluxio, may encounter situations devoid of authentication. In other words, any user possessing the corresponding URI can access the data. To address such scenarios, Alluxio in the cloud has enhanced the authentication process in conjunction with Ranger and CosRanger.

Note

To accommodate the authentication feature, please ensure the integration of the following components within the cluster:
- Should Alluxio only have HDFS mounted, the integration of the Ranger component is required.
- Should Alluxio have COS and CHDFS mounted, the integration of the CosRanger component is necessitated.

Supported Versions

- Supported Service Component Version: Alluxio 2.8.0.
- Product Version: Standard Hadoop 3.x, EMR-V3.4.0.

Configure authentication

Preliminary Configuration

```
#Add new configuration item for Hive component ranger-hive-security.xml
ranger.plugin.hive.urlauth.filesystem.schemes==hdfs:,file:,wasb:,adl:,alluxio:

#Add new configuration item for Presto component hive.properties
hive.hdfs.authentication.type=NONE
hive.metastore.authentication.type=NONE
hive.hdfs.impersonation.enabled=true
hive.metastore.thrift.impersonation.enabled=true
```

Note

The aforementioned preliminary configurations should be set according to the existing components of the cluster.

HDFS Authentication

Create symbolic links for Ranger-related configuration files:

```
[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
```

Configuration of alluxio-site.properties

It is recommended to use the EMR console for cluster-level configuration distribution.

```
# Authentication switch (default is false)
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
Upon completion of the distribution, it is necessary to restart the Alluxio service.

COS and CHDFS Authentication

```
#Add new configuration items to core-site.xml
fs.ofs.ranger.enable.flag=true
```

Configuration of alluxio-site.properties

It is recommended to use the EMR console for cluster-level configuration distribution.

```
# Authentication switch (default is false)
# Authentication switch (default is false)
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.cos.qcloud.object.storage.ranger.service.config.dir=/usr/local/service/cosranger/conf
alluxio.master.security.impersonation.hadoop.users=*
alluxio.security.login.impersonation.username=_HDFS_USER_
# The default number of retry attempts is five.
alluxio.cos.qcloud.object.storage.permission.check.max.retry=5
```

Note
Upon completion of the distribution, it is necessary to restart the Alluxio service.

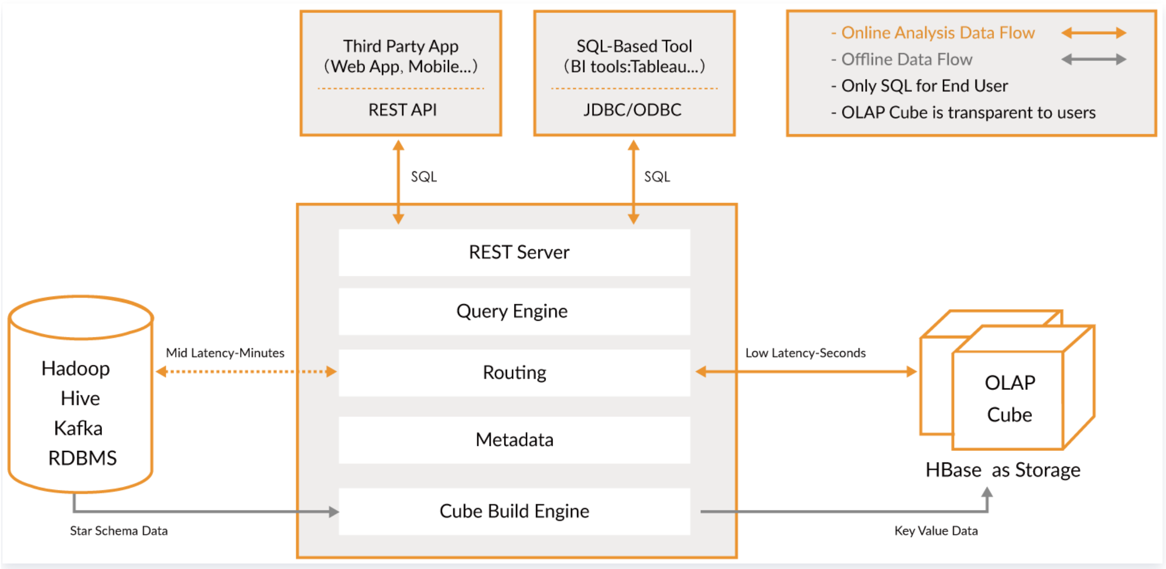
Kylin Development Guide
Kylin Overview

Last updated：2023-12-29 10:09:39

Apache Kylin™ is an open-source, distributed analytical data warehouse that offers a SQL query interface atop Hadoop/Spark, along with multidimensional analysis (OLAP) capabilities to support massive-scale data. Initially developed and contributed to the open-source community by eBay, it possesses the ability to query enormous tables within microseconds.

Introduction to the Kylin Framework

The secret to Kylin's provision of low latency (sub-second latency) lies in precomputation. Specifically, it precomputes metrics for various dimension combinations of a star topology data cube, then stores the results in HBase. By offering JDBC, ODBC, and Rest API query interfaces, it facilitates real-time queries.



Core Concepts of Kylin

- Table: Defined within Hive, a table serves as the data source for the data cube. Prior to building the cube, it must be synchronized within Kylin.
- Model: A model describes the data structure of a star schema, defining the connections and filtering relationships between a fact table and multiple lookup tables.
- Cube Description: This outlines the definition and configuration options of a Cube instance, including the data model used, the dimensions and metrics included, how data is partitioned, and how automatic merging is handled.
- Cube Instance: Derived from the Cube description, it comprises one or more Cube Segments.
- Partition: Users can utilize a DATA/STRING column as the partition column in the Cube description, thereby dividing a Cube into multiple segments based on the date.
- Cube Segment: This is the data carrier after the cube is built. A segment maps to a table in HBase. After the cube instance is built, a new segment is generated. If the original data of a built cube changes, it is only necessary to refresh the segment associated with the changed time period.
- Aggregation Group: Each aggregation group is a subset of dimensions, internally constructing cuboids through combinations.
- Job: Upon issuing a build request for a cube instance, a job is generated. This job records the task information for each step of the cube instance build. The job's status information reflects the result information of the cube instance build. For instance, when the job execution status information is RUNNING, it indicates that the cube instance is being built; when the job status information is FINISHED, it signifies that the cube instance has been successfully built; when the job status information is ERROR, it denotes that the cube instance build has failed.
- **DIMENSION & MEASURE Types**
- Mandatory: These are compulsory dimensions that all cuboids must contain.

- Hierarchy: These are hierarchical relationship dimensions. There is a hierarchical relationship between dimensions, and it is only necessary to retain cuboids with certain hierarchical relationships.
- Derived: These are derived dimensions. In the lookup table, some dimensions can be derived from its primary key, hence these dimensions will not participate in the construction of the cuboid.
- Count Distinct(HyperLogLog): Direct count distinct calculations can be challenging. An approximate algorithm, HyperLogLog, can maintain the error rate within a very low range.
- Count Distinct(Precise): Calculations will be based on RoaringBitMap, currently only supporting int and BigInt.
- **Types of Cube Actions**
- BUILD: Given a time interval specified by a partition column, a new cube segment is built by performing a Build on the Cube.
- REFRESH: This operation will rebuild the cube Segment within certain installment periods.
- MERGE: This operation will amalgamate multiple cube segments. This operation can be set to complete automatically during the cube construction process.
- PURGE: This operation cleanses the segments under a Cube instance, but does not delete the Tables in the HBase table.
- **Job Status**
- NEW: Indicates that a job has been created.
- PENDING: Indicates that a job has been submitted by the job Scheduler and is awaiting execution resources.
- RUNNING: Denotes that a job is currently in operation.
- FINISHED: Signifies that a job has been successfully completed.
- ERROR: Signifies that a job has exited due to an error.
- DISCARDED: Denotes that a job has been cancelled by the user.
- **Job Execution**
- RESUME: This operation will continue the execution of the job from the last successful point of the failed job.
- DISCARD: Regardless of the job's status, the user can terminate it and free up resources.

For more usage methods, please refer to the official documentation.

Livy Development Guide
Livy Overview

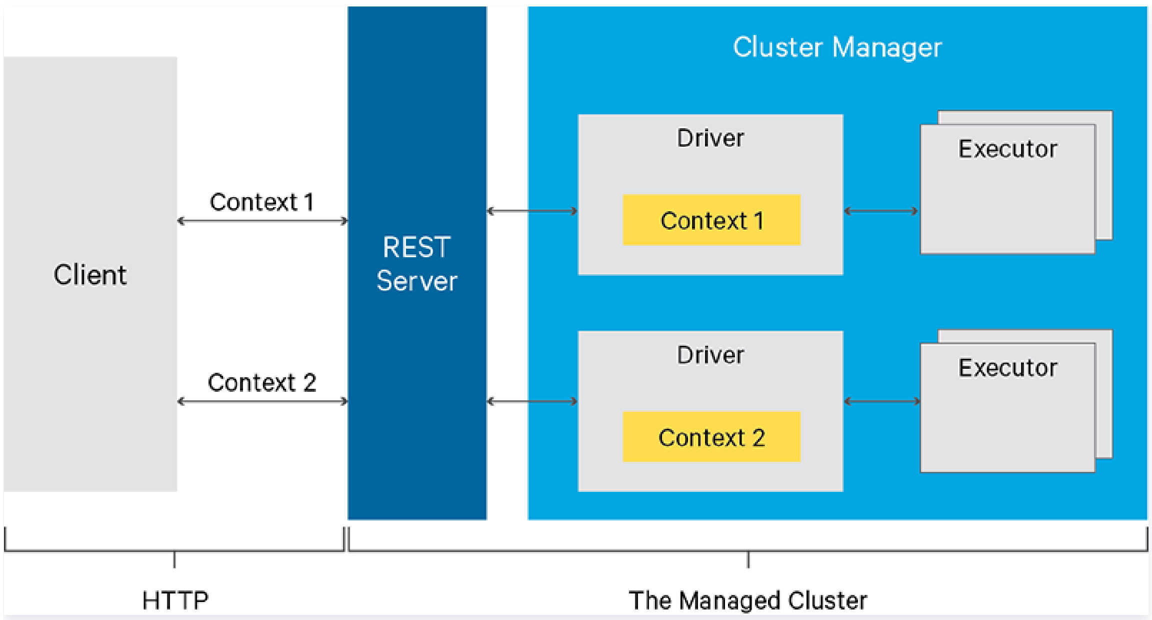
Last updated：2023-12-29 10:10:13

Apache Livy is a service that facilitates interaction with Spark clusters via a REST interface. It enables the submission of Spark jobs or Spark code snippets, synchronous or asynchronous retrieval of results, and management of Spark Context. By simplifying the interaction between Spark and application servers, Apache Livy empowers Spark to be utilized for interactive web/mobile applications.

Livy Features

Livy also supports the following features:

- A Spark context, capable of running for extended periods, can be utilized by multiple clients for various Spark jobs.
- Shared caching of RDDs or data frames across multiple jobs and clients.
- It is capable of managing multiple Spark contexts concurrently, with the Spark contexts operating within the cluster (YARN/Mesos) rather than the Livy server, thereby ensuring robust fault tolerance and concurrency.
- Jobs can be submitted as precompiled jars, code snippets, or via the java/scala client API.
- Ensuring security through secure authentication communication.



Utilizing Livy.

1. Access the Livy UI page at `https://IP:8998/ui` (**IP refers to the public IP, please apply for a public IP for the machine where Livy is installed, and edit the security group policy to open the corresponding port for access**).
2. Establish an interactive session. (The following example code is **based on Linux/Mac OS**)

```
curl -X POST --data '{"kind":"spark"}' -H "Content-Type:application/json" IP:8998/sessions
```

3. View the active sessions on Livy.

```
curl IP:8998/sessions
```

4. Execute a code snippet, a simple addition operation (here it corresponds to the specified session 0, if there are multiple sessions, other sessions can also be specified).

```
curl -X POST IP:8998/sessions/0/statements -H "Content-Type:application/json" -d '{"code":"1+1"}'
```

5. Calculating Pi (Executing jar package).
   Step 1: Upload the jar package to hdfs, for example, upload to /usr/local/spark−examples_2.11−2.4.3.jar.
   Step 2: Execute the 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. To check whether the code snippet has been executed successfully, you can directly view it on the UI page at `https://IP:8998/ui/session/0` .

```
curl IP:8998/sessions/0/statements/0
```

7. Terminate the session.

```
curl -X DELETE IP:8998/sessions/0
```

Supports and Limits

Accessible Configuration Files

The currently accessible configuration files include `livy.conf` and `livy-env.sh` . Both of these configuration files can be modified through the configuration delivery method. For the specific configuration files that are accessible, please refer to the actual EMR console.

Method for Modifying Livy Port

The current default port is 8998, which users can modify at their discretion. This can be done by altering the `livy.server.port` attribute in the `livy.conf` configuration file.
If the cluster is equipped with Hue, due to the connectivity between Hue and Livy, the corresponding configuration port for Hue also needs to be modified. The corresponding configuration file is `pseudo-distributed.ini` , located at `/usr/local/service/hue/desktop/conf` , with the corresponding configuration item being `spark_livy_server_port=8998` . After modification, the corresponding services must be restarted.
Unless necessary, it is recommended not to modify the Livy port. If security requirements are involved, control can be achieved through the use of security groups. Modifications may lead to some other potential issues.

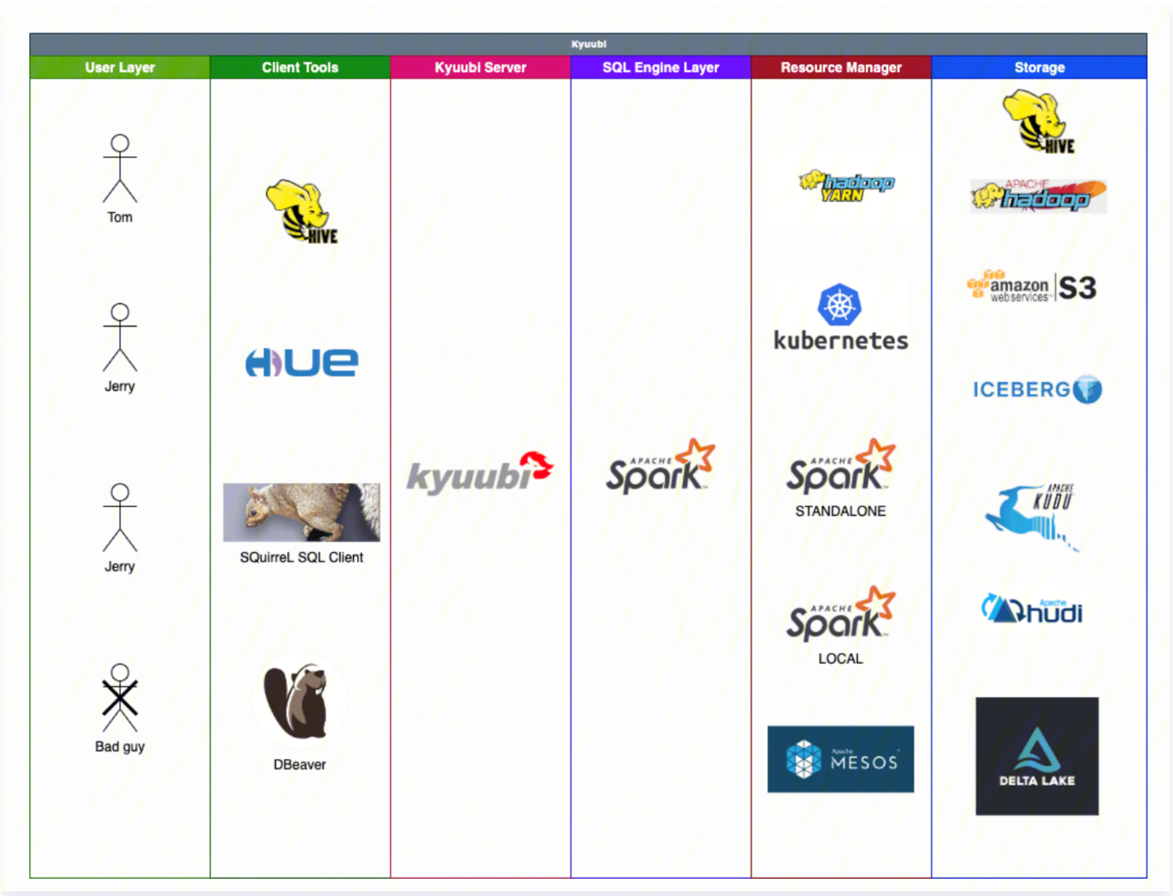
Livy Deployment Method

Currently, Livy is deployed on all master nodes by default. Users can also deploy it on the router by expanding the router.

Kyuubi Development Guide
Kyuubi Overview

Last updated：2023-12-29 10:10:48

Apache Kyuubi (Incubating) is a Thrift JDBC/ODBC service, currently interfaced with the Apache Spark computing framework (in the process of interfacing with the Apache Flink computing framework and Trino), boasting features such as multi-tenancy and distribution. It is capable of satisfying a variety of big data application scenarios within enterprises, such as ETL and BI reporting.



Use Cases

- Supplanting HiveServer2, effortlessly achieving a performance enhancement of 10 to 100 times.
  - Kyuubi is highly compatible with the HiveServer2 interface and behavior, supporting seamless migration.
  - Kyuubi's layered architecture eradicates client compatibility issues, facilitating imperceptible upgrades.
  - Kyuubi supports full-link optimization and further enhancement of Spark SQL, delivering remarkable performance.
  - It encompasses various enterprise-level features such as high availability, multi-tenancy, and fine-grained permission authentication.
- Constructing a Serverless Spark platform.
  - The absolute goal of Serverless Spark is not to have users call Spark's API or continue writing Spark jobs.
  - With Kyuubi's pre-set Engine module, users can get started with minimal understanding of Spark logic, making the entry threshold extremely low.
  - Users only need to focus on their own business development by operating data through JDBC and SQL. The platform offers elastic resource scaling and zero maintenance.
  - It supports resource managers (such as Kubernetes, YARN, etc.), Engine lifecycle, and Spark's dynamic resource allocation, providing a comprehensive resource elasticity strategy at three different levels of granularity.
  - It supports simultaneous scheduling by multiple resource managers such as YARN/Kubernetes, ensuring the safe migration of historical jobs to the cloud.

  - Spark's Adaptive Query Engine (AQE) and Kyuubi AQE plus provide powerful performance.
- Construct a unified data lake exploration, analysis, and management platform (kyuubi–1.5 and above versions).
  - It supports all official Spark data sources as well as third–party data sources.
  - It supports Spark DSv2 metadata management, allowing for intuitive construction and management of data lakes.
  - It supports all mainstream data lake frameworks such as Apache Iceberg/Hudi, DeltaLake.
  - A single interface, a single engine, and a single data set provide a unified platform for analysis queries, data ingestion, and data lake management.
  - Unified batch and stream processing, with support for streaming jobs (Upcoming).

Zeppelin Development Guide
Zeppelin Overview

Last updated：2023-12-29 10:11:15

Apache Zeppelin is a web-based notebook product, designed for interactive data analysis. With Zeppelin, you can create interactive collaborative documents using a wealth of pre-built language backends (or interpreters), such as Scala (Apache Spark), Python (Apache Spark), SparkSQL, Hive, Shell, and more.

Note

For EMR-V3.3.0 and above, as well as EMR-V2.6.0 and above, interpreters for flink, hbase, kylin, livy, and spark have been configured by default. For other versions and components, please refer to the official documentation for configuration according to the Zeppelin version.

Preparations

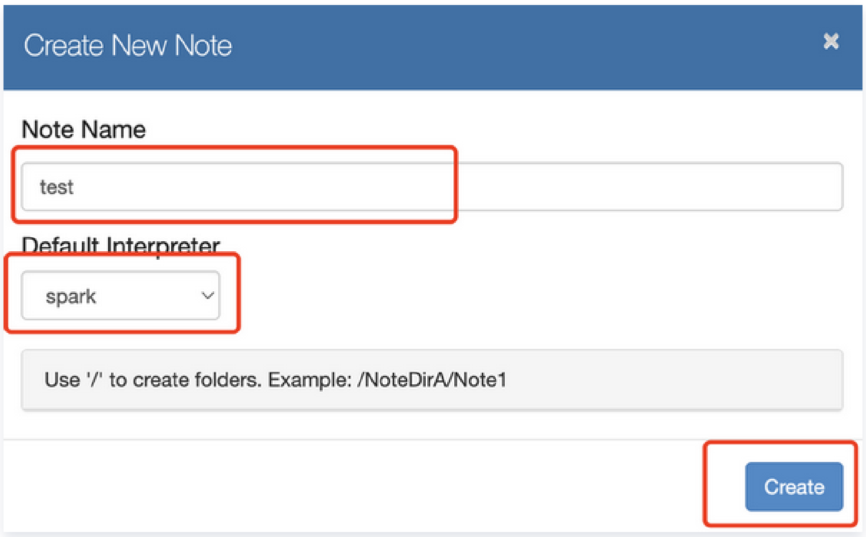
- A cluster has been created and the Zeppelin service has been selected. For more details, please refer to Creating an EMR Cluster.
- Within the EMR cluster's security group, open ports 22, 30001, and 18000 (ports 22 and 30001 are opened by default in a new cluster) along with the necessary internal communication network segments. The new security group is named emr-xxxxxxxx_yyyyMMdd. Please refrain from manually altering the security group's name.
- Add the required services as needed, such as Spark, Flink, HBase, and Kylin.

Accessing Zeppelin

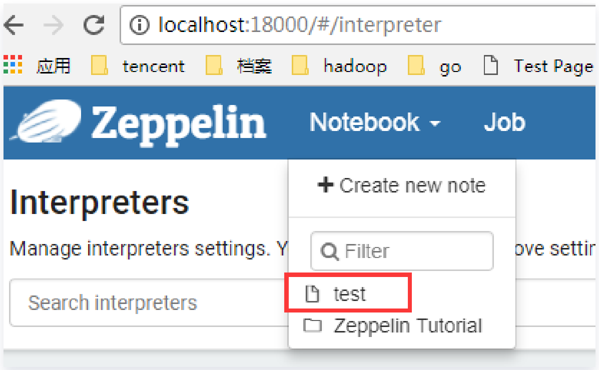
1. Establish a cluster and select the Zeppelin service. For more information, please refer to Creating an EMR Cluster.
2. In the navigation bar on the left side of the EMR Console, select Cluster Services.
3. Click on the card where Zeppelin is located, then click on the **Web UI Address** to access the Web UI page.
4. In versions EMR-V2.5.0 and earlier, as well as EMR-V3.2.1 and earlier, default login permissions have been set with the username and password as admin:admin. If you need to change the password, you can modify the users and roles options in the configuration file /usr/local/service/zeppelin-0.8.2/conf/shiro.ini. For more configuration instructions, please refer to the documentation.
5. In versions EMR-V2.6.0 and later, as well as EMR-V3.3.0 and later, Zeppelin login has been integrated with Openldap accounts. Only Openldap account passwords can be used for login. After a new cluster is created, the default Openldap accounts are root and hadoop. The default password is the cluster password, and only the root account has Zeppelin administrator privileges, granting access to the parser configuration page.

Utilize the Spark feature to accomplish a word count.

1. Click on **Create new note** on the left side of the page, and create a notebook in the pop-up window.



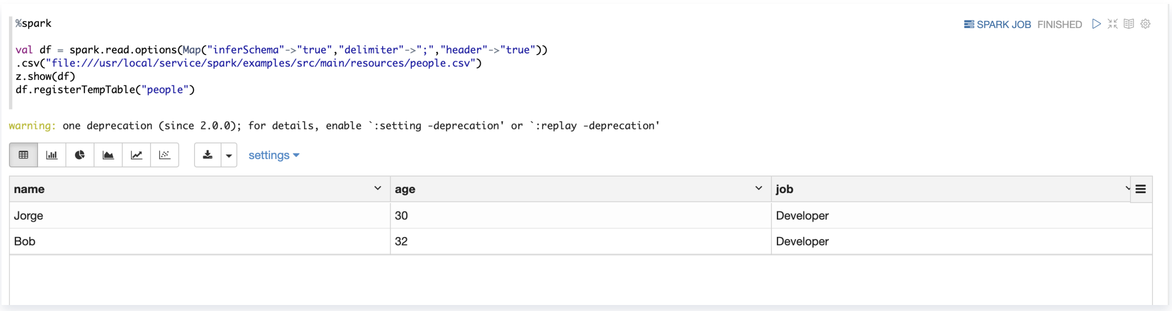
2. In versions EMR-V3.3.0 and above, as well as EMR-V2.6.0 and above, Spark has been configured by default to interface with the EMR cluster (Spark On Yarn).
   - If your version is EMR-V3.1.0, EMR-V2.5.0, or EMR-V2.3.0, please refer to the documentation for Spark interpreter configuration.
   - If your version is EMR-V3.2.1, please refer to the documentation for Spark interpreter configuration.
3. Navigate to your own notebook.



4. Compose a Spark program, using Spark Scala as an example below, where %spark denotes the execution of Spark Scala code:

```
%spark

val df = spark.read.options(Map("inferSchema"->"true","delimiter"->";","header"->"true"))
.csv("file:///usr/local/service/spark/examples/src/main/resources/people.csv")
z.show(df)
df.registerTempTable("people")
```

5. The returned information results are depicted as follows:



```
%spark

val df = spark.read.options(Map("inferSchema"->"true","delimiter"->";","header"->"true"))
.csv("file:///usr/local/service/spark/examples/src/main/resources/people.csv")
z.show(df)
df.registerTempTable("people")

warning: one deprecation (since 2.0.0); for details, enable `:setting -deprecation' or `:replay -deprecation'
```

name	age	job
Jorge	30	Developer
Bob	32	Developer

Zeppelin Interpreter Configuration

Last updated：2023-12-29 10:11:25

This document, using Zeppelin 0.91 or later as an example, primarily discusses the configuration and verification methods of common Zeppelin parsers.
Log in to the Zeppelin WebUI as a root user, the password is by default the same as the one set during the cluster creation. Click on the top right corner **User > Interpreter**, search for the component that requires parser configuration, and click on **edit** to proceed with the configuration.

Spark Parser

Configuration

```
SPARK_HOME: /usr/local/service/spark
spark.master: yarn
spark.submit.deployMode: cluster
spark.app.name: zeppelin-spark
```

Verification

1. Firstly, upload the wordcount.txt file to the /tmp path of the emr hdfs.
2. Locate the fs.defaultFS configuration item value hdfs://HDFS45983 through core-site.xml.
3. Execute spark-related code within 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 Parser

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 Parser

Configuration

```
hbase.home: /usr/local/service/hbase
hbase.ruby.sources: lib/ruby

zeppelin.hbase.test.mode: false
```

Note
The jar package required by this parser has been integrated into the cluster path /usr/local/service/zeppelin/local-repo, thus there is no need to configure dependencies. If a predefined jar package is needed, then dependencies must be configured.

Verification

```
%hbase
help 'get'

%hbase
list
```

Livy Parser

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 Parser

Configuration

1. Create a default Project within the Kylin console page.
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 Parser

1. Configuration of MySQL Parser

```
default.url: jdbc:mysql://ip:3306

default.user: xxx

default.password: xxx

default.driver: com.mysql.jdbc.Driver
```

Note
The jar package required by this parser has been integrated into the cluster path /usr/local/service/zeppelin/local-repo, thus there is no need to configure dependencies. If a predefined jar package is needed, then dependencies must be configured.

Verification

```
%mysql
show databases
```

2. Configuration of Hive Parser

```
default.url: jdbc:hive2://ip:7001

default.user: hadoop

default.password:

default.driver: org.apache.hive.jdbc.HiveDriver
```

Note
The jar package required by this parser has been integrated into the cluster path /usr/local/service/zeppelin/local-repo, thus there is no need to configure dependencies. If a predefined jar package is needed, then dependencies must be configured.

Verification

```
%hive
show databases

%hive
use default;
show tables;
```

3. Configuration of Presto Parser

```
default.url: jdbc:presto://ip:9000?user=hadoop

default.user: hadoop

default.password:

default.driver: io.prestosql.jdbc.PrestoDriver
```

Note

The jar package required by this parser has been integrated into the cluster path /usr/local/service/zeppelin/local-repo, thus there is no need to configure dependencies. If a predefined jar package is needed, then dependencies must be configured.

Verification

```
%presto
show catalogs;

%presto
show schemas from hive;

%presto
show tables from hive.default;
```

For additional versions and parser configurations, please refer to the official Zeppelin documentation.

Hudi Development Guide
Hudi Overview

Last updated：2023-12-29 10:11:42

Apache Hudi offers stream primitives for insertions, updates, and incremental pulls on HDFS datasets. Typically, we store vast amounts of data in HDFS, with new data incrementally written and old data seldom modified, especially in scenarios where data is cleansed and placed into a data warehouse. Moreover, in data warehouses like Hive, the support for updates is quite limited and computationally expensive. On the other hand, if there are scenarios where analysis is only required for newly added data within a certain time frame, native methods are not provided by Hive, Presto, Hbase, etc., instead, analysis is performed based on timestamp filtering. In response to these needs, Hudi can provide implementations for both. The first is record-level updates, and the second is queries only on incremental data. Furthermore, Hudi offers support for Hive, Presto, and Spark, allowing these components to directly query data managed by Hudi.
Hudi is a versatile big data storage system, with the following key features:

- Snapshot isolation between ingestion and query engines, including Hive, Presto, and Spark.
- Supports rollback and savepoints, enabling data set recovery.
- Automatically manages file size and layout to optimize query performance, providing the most recent data for near real-time ingestion.
- Asynchronous compression of real-time data and columnar data.

Timeline

Hudi maintains a timeline that encompasses all dataset operations at various instantaneous moments, thereby offering different views of the dataset starting from different points in time.
Hudi instantaneously includes the following components:

- Operation Type: The type of operation performed on the dataset.
- Instant Time: Instant time is typically a timestamp (for instance, 20190117010349) that monotonically increases in the order of operation start times.
- Status: The state of the instant.

File Organization

Hudi organizes the dataset on the DFS into a directory structure under the `base path`. The dataset is divided into multiple partitions, which are folders containing the data files of that partition, very similar to Hive tables.
Each partition is distinguished by a specific `partition path` relative to the base path. Within each partition, files are organized into `file groups`, uniquely identified by a `file id`. Each file group contains multiple `file slices`, each of which includes a base column file `*.parquet` generated at a certain commit/compaction instant time, as well as a set of log files `*.log*` that contain inserts/updates to the base file since its creation.
Hudi employs an MVCC design where compaction operations merge logs and base files to produce new file slices, while cleaning operations delete unused/older file slices to reclaim space on the DFS. Hudi maps a given hoodie key (record key + partition path) to a file group via an indexing mechanism, thereby providing efficient Upsert.
Once the first version of a record is written into a file, the mapping between the record key and the `file group` / `file id` will never change. In essence, the mapped file group contains all versions of a set of records.

Storage Types

Hudi supports the following storage types:

- Copy on Write: Data is stored exclusively in a columnar file format (such as parquet). Updates to versions are managed by performing synchronous merging during the write process and rewriting the files.
- Merge on Read: Updates are written to incremental files, followed by synchronous or asynchronous compaction to generate new versions of the columnar files.

The table below summarizes the trade-offs between these two storage types:

Trade-offs	Copy on Write	Merge on Read
Data latency	Higher	Lower

Query Latency	Lower	Higher
Update Cost (I/O)	Higher (Entire Parquet file rewrite)	Lower (Append to Incremental Log)
Write Amplification	Higher	Lower (Dependent on Compression Strategy)

Hudi Support for Underlying EMR Storage

- HDFS
- COS

Hudi Installation

Navigate to the EMR Purchase Page, select the product version as **EMR–V3.6.0**, and choose the optional component as **Hudi 0.13.0**. The Hudi component versions corresponding to each product version can be viewed on the Component Version Overview page.

Note
The Hudi component relies on Hive and Spark components. If you opt to install the Hudi component, you must also choose to install the Hive and Spark components.

Use Case

This example will demonstrate the creation of tables and data insertion using SparkSQL, followed by query analysis using Hive and Trino.

Utilizing SparkSQL for Reading and Writing Hudi Tables

- Log in to the master node and switch to the hadoop user.
- Hudi supports data reading and writing using the SparkSQL's **HoodieSparkSessionExtension** extension:

```
spark-sql --master yarn \
--num-executors 2 \
--executor-memory 1g \
--executor-cores 2 \
--jars /usr/local/service/hudi/hudi-bundle/hudi-spark3.3-bundle_2.12-0.13.0.jar \
--conf 'spark.serializer=org.apache.spark.serializer.KryoSerializer' \
--conf 'spark.sql.extensions=org.apache.spark.sql.hudi.HoodieSparkSessionExtension' \
--conf 'spark.sql.catalog.spark_catalog=org.apache.spark.sql.hudi.catalog.HoodieCatalog'
```

Note:
The ––master represents your master URL, ––num–executors denotes the number of executors, and ––executor–memory signifies the storage capacity of the executor. These parameters can be modified according to your actual situation. The ––jars dependency package version may vary across different EMR versions. Please check and use the correct dependency package in the /usr/local/service/hudi/hudi–bundle directory.

- Create

```
-- Creating a COW (Copy-On-Write) Non-Partitioned Table

spark-sql> create table hudi_cow_nonpcf_tbl (
  uuid int,
  name string,
  price double
) using hudi
tblproperties (
  primaryKey = 'uuid'
);

-- Creating a COW (Copy-On-Write) Partitioned Table
```

```
spark-sql> create table hudi_cow_pt_tbl (
  id bigint,
  name string,
  ts bigint,
  dt string,
  hh string
) using hudi
tblproperties (
  type = 'cow',
  primaryKey = 'id',
  preCombineField = 'ts'
 )
partitioned by (dt, hh);

-- Creating a MOR (Merge-On-Read) Partitioned Table

spark-sql> create table hudi_mor_tbl (
  id int,
  name string,
  price double,
  ts bigint,
  dt string
) using hudi
tblproperties (
  type = 'mor',
  primaryKey = 'id',
  preCombineField = 'ts'
)
partitioned by (dt);
```

- Data Ingestion

```
-- insert into non-partitioned table
spark-sql> insert into hudi_cow_nonpcf_tbl select 1, 'a1', 20;

-- insert dynamic partition
spark-sql> insert into hudi_cow_pt_tbl partition (dt, hh) select 1 as id, 'a1' as name, 1000 as ts, '2021-12-09' as dt, '10' as hh;

-- insert static partition
spark-sql> insert into hudi_cow_pt_tbl partition(dt = '2021-12-09', hh='11') select 2, 'a2', 1000;
spark-sql> insert into hudi_mor_tbl partition(dt = '2021-12-09') select 1, 'a1', 20, 1000;
```

- Query data

```
spark-sql> select * from hudi_cow_nonpcf_tbl;
20230808165859974  20230808165859974_0_1  uuid:1     f564e6c7-631c-4f32-a01d-e042f37ad6e6-0_0-21-
19_20230808165859974.parquet  1  a1   20.0
Time taken: 6.255 seconds, Fetched 1 row(s)

spark-sql> select count(*) from hudi_mor_tbl;
1
Time taken: 0.955 seconds, Fetched 1 row(s)

spark-sql> select name, count(*) from hudi_cow_pt_tbl group by name;
a2    1
a1    1
Time taken: 2.049 seconds, Fetched 2 row(s)
```

Utilizing Hive to Query Hudi Tables

The following illustrates how to query the aforementioned Hudi tables within the Hive CLI:

Note:
The version of the Hudi dependency package loaded may vary across different EMR versions. Please select the correct version of the dependency package for loading from the /usr/local/service/hudi/hudi-bundle directory.

```
hive> add jar /usr/local/service/hudi/hudi-bundle/hudi-hadoop-mr-bundle-0.13.0.jar;
Added [/usr/local/service/hudi/hudi-bundle/hudi-hadoop-mr-bundle-0.13.0.jar] to class path

hive> show tables;
OK
hudi_cow_nonpcf_tbl
hudi_cow_pt_tbl
hudi_mor_tbl
hudi_mor_tbl_ro
hudi_mor_tbl_rt
Time taken: 0.023 seconds, Fetched: 5 row(s)

hive> set hive.input.format=org.apache.hadoop.hive.ql.io.HiveInputFormat;
hive> select * from hudi_mor_tbl_ro;
OK
20230808174602565  20230808174602565_0_1   id:1  dt=2021-12-09    af40667d-1dca-4163-89ca-2c48250985b2-0_0-34-
1617_20230808174602565.parquet   1  a1    20.0 1000    2021-12-09
Time taken: 0.159 seconds, Fetched: 1 row(s)

hive> set hive.vectorized.execution.enabled=false;
hive> select name, count(*) from hudi_mor_tbl_rt group by name;
a1    1
Time taken: 17.618 seconds, Fetched: 1 row(s)
```

Employing Trino to Query Hudi Tables

The following demonstrates how to query the previously created Hudi tables within Trino.

- Versions prior to EMR 3.6.0:
  - Trino queries Hudi using the Hive Connector, eliminating the need for additional configurations.
  - Copy **/usr/local/service/hudi/hudi-bundle/hudi-hadoop-mr-bundle-$hudi_version.*jar*** to the **/usr/local/service/trino/plugin/hive/** path on all cluster nodes, and subsequently restart the Trino service.
- Versions EMR-3.6.0 and subsequent:
  - It is necessary to add a hudi connector configuration file within the Trino component configuration management. Refer to the configuration update instructions for adding a new configuration file. The **${hivemetastore_uri} should be changed to the Hive Metastore service address of the cluster, which can be found in the hive.metastore.uris** configuration item in the Hive component configuration file hive-site.xml.
  - Upon completion of the configuration, restart the Trino service.

```
connector.name=hudi
hive.metastore.uri=${hivemetatore_uri}
```

Upon completion of the aforementioned steps, Hudi tables can be queried within Trino:

```
trino:default> show tables;
       Table
---------------------
 hudi_cow_nonpcf_tbl
 hudi_cow_pt_tbl
 hudi_mor_tbl
 hudi_mor_tbl_ro
 hudi_mor_tbl_rt
(5 rows)
```

```
Query 20230808_113217_00005_r3tqk, FINISHED, 3 nodes
Splits: 12 total, 12 done (100.00%)
0.34 [11 rows, 359B] [32 rows/s, 1.03KB/s]

trino:default> select uuid, name, price from hudi_cow_nonpcf_tbl;
 uuid | name | price
------+------+-------
    1 | a1   |  20.0
(1 row)

Query 20230808_114808_00001_72vwk, FINISHED, 1 node
Splits: 5 total, 5 done (100.00%)
3.48 [1 rows, 440KB] [0 rows/s, 126KB/s]

trino:default> select id, name, price from hudi_mor_tbl_ro;
 id | name | price
----+------+-------
  1 | a1   |  20.0
(1 row)

Query 20230808_115532_00006_72vwk, FINISHED, 1 node
Splits: 5 total, 5 done (100.00%)
1.75 [1 rows, 440KB] [0 rows/s, 251KB/s]

trino:default> select name, count(*) from hudi_mor_tbl_rt group by name;
 name | _col1
------+-------
 a1   |     1
(1 row)

Query 20230808_115728_00009_72vwk, FINISHED, 2 nodes
Splits: 21 total, 21 done (100.00%)
0.63 [1 rows, 440KB] [1 rows/s, 698KB/s]
```

Superset Development Guide
Superset Overview

Last updated：2023-12-29 10:13:22

Apache Superset is a data exploration and visualization web application. The Superset on EMR comes pre-equipped with support for Mysql, Hive, Presto, Impala, and Kylin.

Superset Features

- It supports nearly all mainstream databases, including MySQL, PostgreSQL, Oracle, SQL Server, SQLite, SparkSQL, and so forth.
- It boasts a rich array of visualization options and supports the custom creation of dashboards.
- The data display is fully controllable, allowing for the customization of display fields, data aggregation, data sources, and so on.

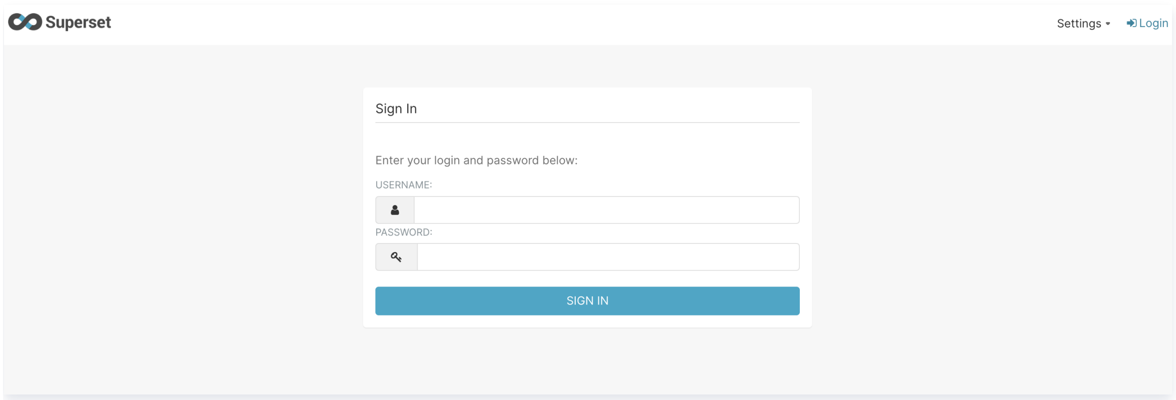
Preparations

1. An Elastic MapReduce (EMR) Hadoop or Druid cluster has been created, with the Superset service selected. For more details, please refer to Creating an EMR Cluster.
2. By default, Superset is installed on the master node of the cluster. Ensure that your network can access port 18088 of the master node by opening the security group policy of the master node.

Login

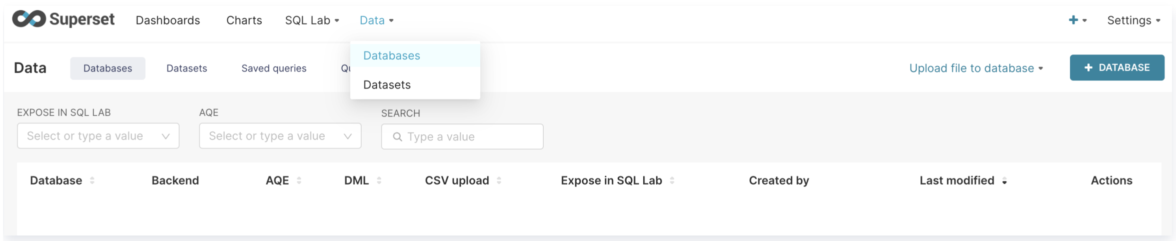
Enter `http://${master_ip}:18088` in the browser's address bar (or go through EMR Console > **Cluster Services**), to open the Superset login interface. The default username is 'admin', and the password is the one you set when creating the cluster.

⚠ Note
Different versions of the Superset community may have variations in web page interactions. This document uses Superset version 1.5.1 as an example. For more versions and usage methods, please refer to the Official Superset Documentation.

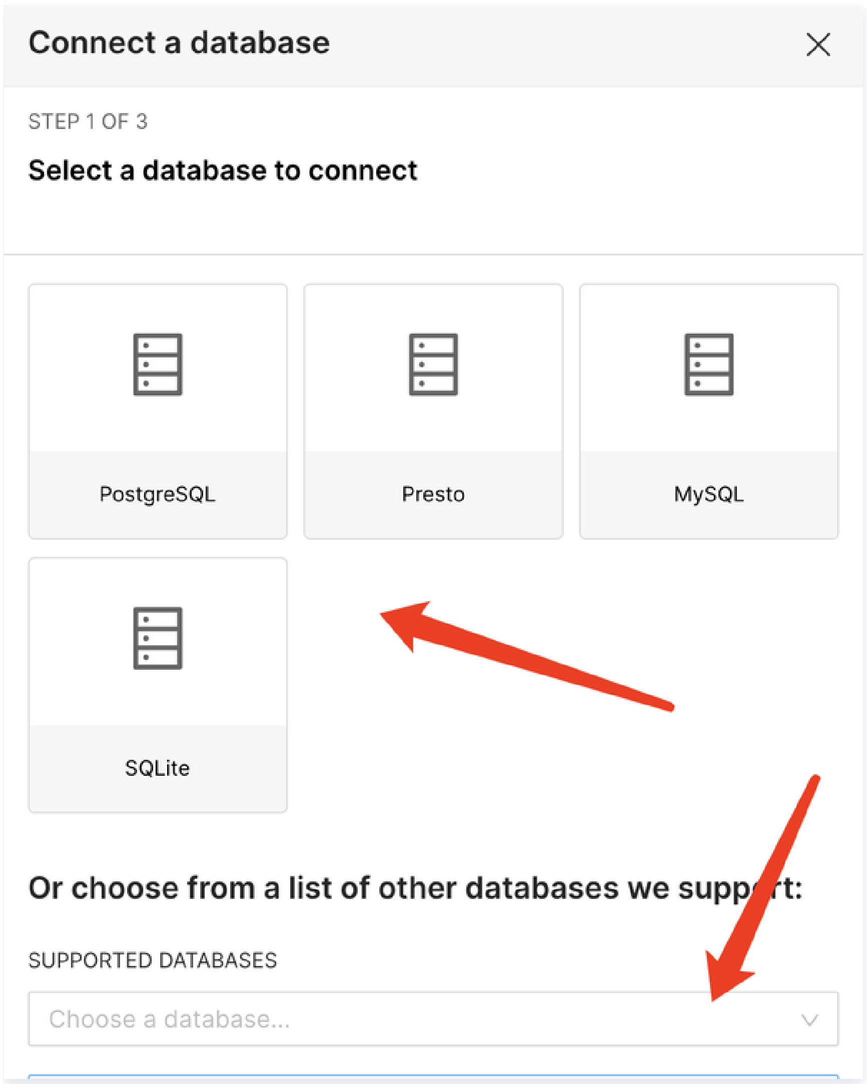


Add DataBase

Navigate to the **Data > Databases** interface and click on **+DATABASE**.



You will be directed to the following page where you can select the data source you wish to add.



The SQLAlchemy URI links for each database are as follows:

Name	SQLAlchemy URI	Remarks
Mysql	mysql+pymysql://<mysqlname>:<password>@<mysql_ip>:<mysql_port>/<your_database>	• mysqlname: The username used for connecting to MySQL. • password: MySQL password • your_database: The MySQL database that needs to be connected.
Hive	hive://hadoop@<master_ip>:7001/default?auth=NONE	Master_ip: The master_ip of the EMR cluster.
presto	presto://hive@<master_ip>:9000/hive/<hive_db_name>	• Master_ip: The master_ip of the EMR cluster. • hive_db_name: The database name in Hive, defaults to 'default' if not specified.
impala	impala://<core_ip>:27000	core_ip: The core IP within the EMR cluster.
kylin	kylin://<kylin_user>:<password>@<master_ip>:16500/<kylin_project>	• kylin_user: Username for Kylin. • password: Password for Kylin. • Master_ip: The master_ip of the EMR cluster. • kylin_project: Project in Kylin.

Add a new Database independently.

Superset supports Database. If you need to install additional databases, you can proceed as follows:

1. Log into the machine where the EMR cluster master is located.
2. Execute the command `source /usr/local/service/superset/bin/activate` .
3. Execute pip3 install for the corresponding Python library.
4. Restart Superset.

Impala Development Guide
Impala Overview

Last updated：2023-12-29 10:13:48

The Apache Impala project offers high-performance, low-latency SQL queries for data stored in Apache Hadoop file formats. It provides swift responses to queries, while facilitating interactive data exploration and query adjustments for analytical queries, as opposed to the traditional long-duration batch jobs associated with SQL-on-Hadoop technologies.
Impala differs from Hive, which utilizes the MapReduce engine at its core, remaining a batch processing procedure. Conversely, Impala's intermediate results are not written to disk, but are instantaneously transmitted over the network in a stream-like manner, significantly reducing the IO overhead of the nodes.
Impala integrates with the Apache Hive database, sharing databases and tables between the two components. Through its high degree of integration with Hive, as well as compatibility with HiveQL syntax, you can use Impala or Hive to create tables, initiate queries, load data, and more.

Preparations

- Ensure that Tencent Cloud has been activated and an EMR cluster has been created. When creating the EMR cluster, it is necessary to select the Impala component in the software configuration interface.
- Impala is installed in the `/data/` directory of the EMR cloud server (`/data/Impala`).

Data Preparation

First, log into any machine in the EMR cluster, preferably the Master node. For methods on how to log into EMR, please refer to Logging into Linux Instances. You can choose to log in using WebShell. Click on the login button on the right side of the corresponding cloud server to enter the login interface. The default username is root, and the password is the one entered by the user when creating EMR. After entering correctly, you can access the command line interface.
In the EMR command line, first use the following command to switch to the Hadoop user and enter the Impala folder.

```
[root@10 ~]# su hadoop
[hadoop@10 root]$ cd /data/Impala/
```

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

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

After granting executable permissions to gen_data.sh using the chmod command, execute the following command:

```
[hadoop@10 ~]$ ./gen_data.sh > impala_test.data
```

This script file will generate 1,000,000 pairs of random numbers and save them to the file `impala_test.data` . Then, upload the generated test data to HDFS by executing the following command:

```
[hadoop@10 ~]$ hdfspath="/impala_test_dir"
[hadoop@10 ~]$ hdfs dfs -mkdir $hdfspath
[hadoop@10 ~]$ hdfs dfs -put ./impala_test.data $hdfspath
```

Where $hdfspath is the path in HDFS where you store your files. Finally, use the following command to verify whether the data has been properly placed on HDFS.

```
[hadoop@10 ~]$ hdfs dfs -ls $hdfspath
```

Impala Basic Operations

Due to variations in community component interface protocols and default path values across different Impala versions, the paths for different versions of impala-shell are as shown in the table below.

Impala Version	Impala-shell Path	Impala-shell Default Connection Port
4.1.0/4.0.0	/data/Impala/shell	27009
3.4.0	/data/Impala/shell	27001
2.10.0	/data/Impala/bin	27001

The following operations are exemplified using the Impala 3.4.0 version:

Connecting to Impala

Log into the Master node of the EMR cluster, switch to the Hadoop user, enter the Impala directory, and establish a connection with Impala:

```
[root@10 Impala]# cd /data/Impala/shell;./impala-shell -i $core_ip:27001
```

Wherein, the core_ip refers to the IP of the core node in the EMR cluster, or the IP of the task node can also be used. Upon successful login, the following will be displayed:

```
Connected to $core_ip:27001
Server version: impalad version 3.4.1-RELEASE RELEASE (build Could not obtain git hash)
***********************************************************************************
Welcome to the Impala shell.
(Impala Shell 3.4.1-RELEASE (ebled66) built on Tue Nov 20 17:28:10 CST 2021)

The SET command shows the current value of all shell and query options.
***********************************************************************************
[$core_ip:27001] >
```

Alternatively, after logging into the core node or task node, you can directly establish a connection and execute the following command:

```
cd /data/Impala/shell;./impala-shell -i localhost:27001
```

Creating an Impala database

Execute the following command under 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
```

Utilize the `create` command to establish a new database:

```
[localhost:27001] > create database experiments;
Query: create database experiments
Fetched 0 row(s) in 0.41s
```

Employ the `use` command to navigate to the recently established test database:

```
[localhost:27001] > use experiments;
Query: use experiments
```

To view the current database, execute the following command:

```
select current_database();
```

Establishing an Impala table

Utilize the `create` command to establish a new internal table named impala_test under 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
```

To view all tables:

```
[localhost:27001] > show tables;
Query: show tables
+------+
| name |
+------+
| t1   |
+------+
Fetched 1 row(s) in 0.01s
```

To examine 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 the table

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

```
LOAD DATA INPATH '$hdfspath/impala_test.data' INTO TABLE t1;
```

Where $hdfspath is the path of your file stored in HDFS. Upon completion of the import, the source data file on the import path in HDFS will be deleted. It is stored under the Impala internal table storage path `/usr/hive/warehouse/experiments.db/t1` . An external table can also be established, with the statement as follows:

Note
Here, there is only one command. If the semicolon ";" is not entered, a single command can be distributed across multiple lines for input.

```
CREATE EXTERNAL TABLE t2
(
  a INT,
  b string
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
LOCATION '/impala_test_dir';
```

Execute 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=f1441478dba3a1c5:fa7a8eef00000000
+----------+
| count(*) |
+----------+
| 1000000  |
+----------+
Fetched 1 row(s) in 0.63s
```

The final output result is 1,000,000.

Deleting a table

```
[localhost:27001] > drop table experiments.t1;
Query: drop table experiments.t1
```

For more operations with Impala, refer to the official documentation.

Connecting to Impala via JDBC

Impala can also be connected via Java code, the process is similar to connecting to Hive through Java.
The only difference lies in `$hs2host` and `$hsport`, where `$hs2host` is the IP of any core node or task node in the EMR cluster. The hsport can be viewed in the corresponding node's Impala directory, in the configuration file `conf/impalad.flgs`.

```
[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 utilizes Hive's metadata information, hence all tables in Hive can be read in Impala. This can be achieved by mapping Hbase tables in Hive to map Hbase tables in Impala.

Impala OPS Manual

Last updated：2023-12-29 10:14:13

Data expansion results in Impala launch failure.

Example

When the metadata information in Impala is excessive (for instance, hundreds of libraries, tens of thousands of tables), Impala needs to broadcast this metadata information to all nodes during startup. The default timeout for this action is 10 seconds. When the metadata is large and easily triggered, this can be adjusted by setting `-statestore_subscriber_timeout_seconds=100` in Impala's startup configuration file `/data/Impala/conf/impalad.flgs`.

Problem Diagnosis Confirmation

Typically, when this issue arises, the following content can be found in Impala's log `/data/emr/impala/logs`:

```
Connection with state-store lost
Trying to re-register with state-store
```

Impala's query speed is reduced due to low configuration.

Although Impala is not an in-memory database, it is advisable to allocate more physical memory to Impala when dealing with large tables and large data. The general recommendation is to use 128GB or more memory, and allocate 80% to the Impala process.

SELECT Statement Failure

General possible causes:

1. Timeouts may be caused by performance, capacity, or network issues on a specific node. Review the Impala log to identify the problematic node and check if there are any network issues on that machine.
2. Excessive memory usage in join queries leads to automatic query cancellation. Check the reasonableness of the join statements or consider increasing the machine's memory.
3. The issue may be influenced by how native code is generated on a node to handle specific WHERE clauses in a query. For instance, machine instructions that the processor of a specific node does not support could be generated. If the error message in the log indicates that the cause is an illegal instruction, consider temporarily disabling native code generation and then retry the query.
4. Input data with incorrect formatting, such as a text data file with excessively long lines, or separators that do not match the characters specified in the FIELDS TERMINATED BY clause of the CREATE TABLE statement. Check for excessively long data and verify the table creation statement to ensure the correct separator has been specified.

Setting memory usage limits for 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] > select 5;
+---+
|5 |
+---+
```

Analyzing Data on COS/CHDFS

Last updated：2023-12-29 10:14:23

This section will demonstrate additional usage methods of Impala, based on Tencent Cloud Object Storage (COS). The data originates from direct data insertion and COS data.

Development Preparations

1. Given that the task necessitates access to Tencent Cloud Object Storage (COS), it is imperative to initially create a storage bucket (Bucket) within COS.
2. Ensure that you have activated Tencent Cloud and have established an EMR cluster. When creating the EMR cluster, it is necessary to select the Impala component on the software configuration interface, and to authorize object storage on the basic configuration page.
3. Impala and related software are installed in the `/usr/local/service/` directory of the EMR cloud server.

Instructions

Log into any machine within the EMR cluster, preferably the Master node. For methods on how to log into EMR, please refer to Logging into Linux Instances. You may opt to use WebShell for logging in. Click on the login option on the right side of the corresponding cloud server to access the login interface. The default username is 'root', and the password is the one input by the user during the creation of EMR. Upon correct input, you will gain access to the command line interface.
In the EMR command line, first use the following command to switch to the Hadoop user and connect to Impala:

```
[root@172 ~]# su hadoop
[hadoop@172 ~]$ impala-shell -i $host:27001
```

$host represents the internal IP where your Impala-Daemon role is located, and 27001 is the value of the beeswax_port configuration item in impalad.flgs.

Step One: Create a Table (Record)

```
[$host:27001 ] > create table record(id int, name string) row format delimited fields terminated by ',' stored as textfile location
'cosn://$bucketname/';
Query: create table record(id int, name string) row format delimited fields terminated by ',' stored as textfile location
'cosn://$bucketname/'
Fetched 0 row(s) in 3.07s
Where $bucketname is your COS bucket name plus path. If using CHDFS, replace the value of location with
ofs://$mountname/, where $mountname is your CHDFS mount address plus path.
Inspect the table information to confirm that the location is a COS path.
[$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 Two: 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=b246d3194efb7a8f:bc60721600000000
Modified 1 row(s) in 0.64s
```

Step Three: Query the Table Using Impala

```
[$host:27001] > select * from record;
Query: select * from record
Query submitted at: 2020-08-03 11:29:31 (Coordinator: http://172.30.1.136:27004)
Query progress can be monitored at: http://$host:27004/query_plan?query_id=8148da96f8c0d369:4b26432a00000000
+----+---------+
| id | name    |
+----+---------+
| 1  | test    |
+----+---------+
Fetched 1 row(s) in 0.37s
```

Druid Development Guide
Druid Overview

Last updated：2023-12-29 10:19:18

Apache Druid is a distributed system, designed to facilitate real-time multidimensional OLAP analysis, adept at addressing the challenges of rapid, interactive querying and analysis within extensive data sets.

Fundamental Characteristics

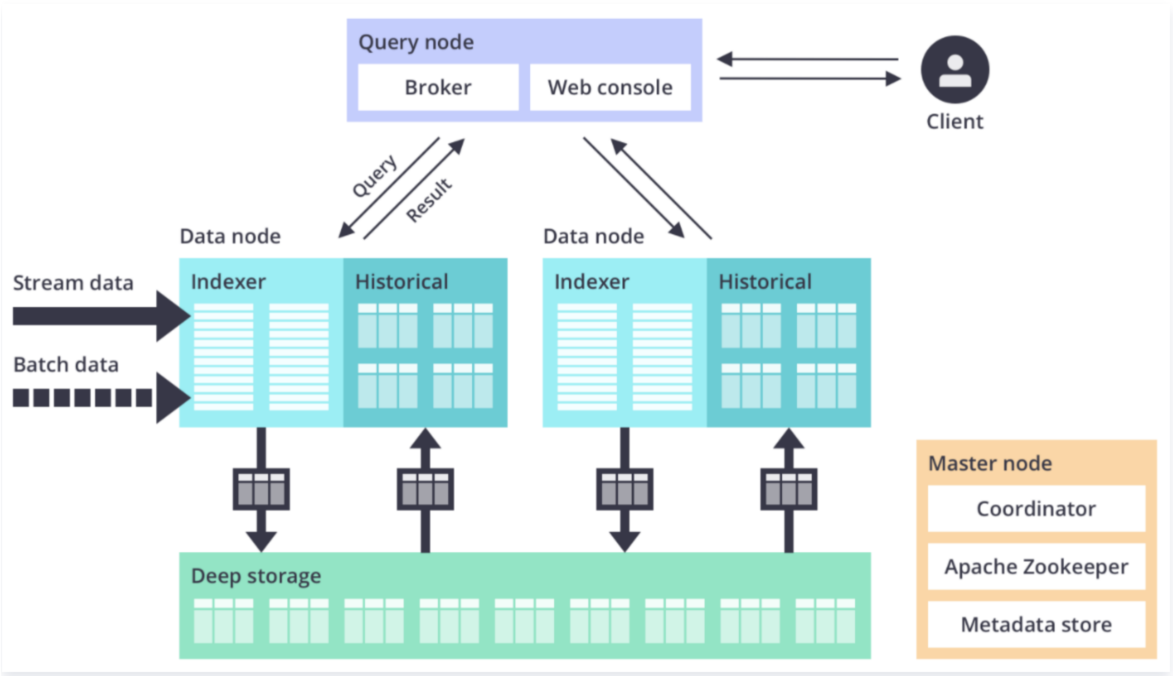
Apache Druid is characterized by the following features:
- It offers sub-second response times for interactive queries, with robust support for multidimensional filtering, ad-hoc attribute grouping, and rapid data aggregation.
- It accommodates high concurrency and real-time data ingestion, truly achieving instantaneous data intake and immediate query results.
- It boasts strong scalability, employing a distributed shared-nothing architecture, capable of swiftly processing petabyte-scale data and hundreds of billions of events, while supporting thousands of concurrent queries per second.
- It facilitates simultaneous online queries from multiple tenants.
- It ensures high availability and supports rolling upgrades.

Scenarios

Druid is most commonly utilized for flexible and rapid multi-dimensional OLAP analysis in the context of big data. Additionally, a key feature of Druid is its support for pre-aggregated data ingestion and aggregate analysis based on timestamps. Consequently, it is frequently employed in scenarios involving time-series data processing and analysis, such as advertising platforms, real-time metric monitoring, recommendation models, and search models.

System Architecture

Druid is an architecture based on microservices. Each core service within Druid can be deployed individually or in conjunction on different hardware.



Enhanced EMR Druid
EMR Druid, based on Apache Druid, has undergone numerous enhancements, including integration with EMR Hadoop and the surrounding cloud ecosystem, convenient monitoring and operational support, and user-friendly product interfaces, achieving immediate usability and maintenance-free operation.
The features currently supported by EMR Druid are as follows:

- Ease of integration with EMR Hadoop clusters
- Convenient and rapid elastic scaling
- Supports High Availability (HA)
- Supports using COS as deep storage
- Supports using COS files as data sources for batch indexing
- Supports storing metadata in TencentDB
- Integrated with tools such as Superset
- Abundant monitoring metrics and alert rules
- Fault migration
- High security

Druid Usage

Last updated：2023-12-29 10:19:29

EMR facilitates the utilization of E-MapReduce Druid clusters as an independent cluster type, primarily based on the following considerations:

- Usage Scenarios: E-MapReduce Druid can operate independently from Hadoop to accommodate various business application contexts.
- Resource Preemption: E-MapReduce Druid has a high demand for memory, particularly for Broker nodes and Historical nodes. The resource usage of E-MapReduce Druid is not subject to the unified scheduling of Hadoop YARN, which can lead to resource contention during operation.
- Cluster Scale: As a foundational infrastructure, Hadoop typically operates on a larger scale, whereas E-MapReduce Druid clusters may be relatively smaller. Deploying them on the same cluster could lead to resource wastage due to the disparity in scale. Independent deployment offers greater flexibility.

Purchasing Recommendations

When creating an EMR cluster, simply select the Druid cluster type. The Druid cluster comes with integrated Hadoop HDFS and YARN services. However, it is recommended for testing purposes only. For live production environments, it is strongly advised to utilize a dedicated Hadoop cluster.

If there is a need to disable the Hadoop-related services that come with the Druid cluster, you can do so in the EMR Console's cluster services page. Select the corresponding service card and click on 'Actions > Pause Service' to suspend the service.

Hadoop and Druid Cluster Connectivity Configuration

This section outlines how to configure the connectivity between Hadoop and Druid clusters. If you are using the Hadoop cluster that comes with the Druid cluster (not recommended for production environments), no additional settings are required for normal connectivity and use, and this section can be skipped.
If you need to store index data on the HDFS of a separate Hadoop cluster (recommended for production environments), you first need to establish connectivity between the two clusters. The specific steps are as follows:

1. Ensure that the Druid and Hadoop clusters can communicate effectively.

   Both clusters should either be under the same VPC, or if they are in different VPCs, there should be normal communication between the two VPCs (for instance, through Cloud Connect Network or Peer Connection).
2. Place a copy of the core-site.xml, hdfs-site.xml, yarn-site.xml, and mapred-site.xml files from the Hadoop cluster's '/usr/local/service/hadoop/etc/hadoop' directory into each node of the E-MapReduce Druid cluster at the '/usr/local/service/druid/conf/druid/_common' path.

   Note
   Since the Druid cluster comes with a Hadoop cluster, the relevant soft links for the above files have already been created in the Druid path. These need to be deleted first, before copying the configuration from another Hadoop cluster. At the same time, it is necessary to ensure that the file permissions are correct and can be accessed normally by the Hadoop user.

3. Modify the common.runtime.properties configuration file within Druid's configuration management. After making the changes, save the configuration and restart the relevant services in the Druid cluster.

- druid.storage.type: The default is set to hdfs, no modification is required.
- druid.storage.storageDirectory：

```
If the other Hadoop cluster is non-HA: hdfs://{namenode_ip}:4007
If the other Hadoop cluster is HA: hdfs://HDFSXXXXX
Please configure the full path. The detailed address can be found in the fs.defaultFS configuration item in the core-site.xml
file of the target Hadoop cluster.
```

Utilize COS

E-MapReduce Druid supports the use of COS as deep storage. This section provides guidance on how to utilize COS as the deep storage for your Druid cluster.
Initially, you need to ensure that both the Druid cluster and the target Hadoop have activated the COS service. This can be done during the purchase of the Druid and Hadoop clusters, or it can be configured post-purchase via the EMR console.

1. Modify the common.runtime.properties configuration file within the Druid configuration management:
   - druid.storage.type: Remains as hdfs.
   - druid.storage.storageDirectory: `cosn://{bucket_name}/druid/segments` .
     You have the option to pre-create and set the segments directory and permissions on COS.
2. Modify the core-site.xml configuration file within the hdfs configuration management:
   - fs.cosn.impl: Modify to `org.apache.hadoop.fs.CosFileSystem` .
   - Add new configuration item `fs.AbstractFileSystem.cosn.impl` : Modify to `org.apache.hadoop.fs.CosN` .
3. Place the hadoop-cos related jar packages (for example: cos_api-bundle-5.6.69.jar, hadoop-cos-2.8.5-8.1.6.jar) into the directories of each node in the cluster: /usr/local/service/druid/extensions/druid-hdfs-storage, /usr/local/service/druid/hadoopdependencies/hadoop-client/2.8.5, /usr/local/service/hadoop/share/hadoop/common/lib/.

Save the configuration and restart the related services of the Druid cluster.

Adjust the Druid parameters

Upon creation, E-MapReduce Druid automatically generates a set of configurations. However, it is recommended to adjust the optimal memory configuration according to your business needs. To adjust the configuration, you can use the Configuration Management feature.
When adjusting the configuration, please ensure it is done correctly:

```
MaxDirectMemorySize >= druid.processing.buffer.sizeByte *(druid.processing.numMergeBuffers +
druid.processing.numThreads + 1)
```

Recommended adjustments:

```
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 configurations, please refer to Druid Component Configuration.

Expand the Router as a query node

The current Druid cluster defaults to deploying the Broker process on the EMR Master node. Given the multitude of processes deployed on the Master node, inter-process interference may lead to insufficient memory, affecting query efficiency. At the same time, many businesses prefer to deploy query nodes separately from the central node. In such cases, you can expand one or more Router nodes through the console and choose to install the Broker process, thereby conveniently expanding the query nodes of the Druid cluster.

Access the Web

Access the Druid cluster uniformly through the console, with the port opened on the master node's 18888 port. You can configure the public IP address yourself. After opening the 18888 port in the security group and setting the bandwidth, you can access it via `[http://{masterIp}:18888]()` .

Ingesting Data from Hadoop in Batches

Last updated：2023-12-29 10:20:42

This section succinctly elucidates the process of bulk loading data files from a remote Hadoop cluster into a Druid cluster. All operations discussed herein are performed as a Hadoop user, hence it is imperative to switch to the Hadoop user on both the Druid and Hadoop clusters.

Bulk loading of data into the Druid cluster

1. On the corresponding remote Hadoop cluster, execute the following command to create a new directory as a Hadoop user:

```
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
Should the Druid and Hadoop clusters be two distinct entities, the directory must be established on the corresponding Hadoop cluster (subsequent operations are similar, with due attention paid to correctly identifying the appropriate cluster). However, if in a testing environment the Druid and Hadoop clusters are one and the same, operations can be carried out within the same cluster.

2. Upload Test Package
The Druid cluster inherently includes a dataset example named 'Wikiticker' (default path `/usr/local/service/druid/quickstart/tutorial/wikiticker-2015-09-12-sampled.json.gz`). This dataset within the Druid cluster should be uploaded to the corresponding remote Hadoop cluster, **which is uploaded 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 Index File
Prepare an index file, continuing to use the sample file from the Druid cluster `/usr/local/service/druid/quickstart/tutorial/wikipedia-index-hadoop.json` , with the command as follows:

```
{
  "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 refers to the version of Hadoop that is depended upon.
- spec.ioConfig.inputSpec.paths denotes the input file path. If cluster connectivity has already been configured in the common.runtime.properties, a relative path can be utilized (refer to Druid Usage for guidance). Otherwise, depending on the circumstances, a relative path beginning with either `hdfs://` or `cosn://` should be employed.
- The tuningConfig.jobProperties parameter can be used to set relevant parameters for the mapreduce job.

4. Submit Index Task

   Next, a task can be submitted on the Druid cluster to ingest the data. This is executed under the Druid directory as a Hadoop user:

```
./bin/post-index-task --file quickstart/tutorial/wikipedia-index-hadoop.json --url http://localhost:8090
```

Upon successful completion, an output similar to the following will be generated:

```
...
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 both SQL-like and native JSON queries. These will be discussed separately below. For more detailed information, please refer to the official documentation .

SQL Query Method

Druid supports a variety of SQL query methods:

- Queries can be executed within the Query menu of 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
```

- Interactive queries can be executed on the query node using the command-line tool `bin/dsql` .

```
[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 TIMESTAMP '2015-09-12 00:00:00' AND
TIMESTAMP '2015-09-13 00:00:00' GROUP BY page ORDER BY Edits DESC LIMIT 10;
┌──────────────────────────────────────────────────────────┬───────┐
│ page                                                     │ Edits │
├──────────────────────────────────────────────────────────┼───────┤
│ Wikipedia:Vandalismusmeldung                             │    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.
```

- Execute SQL queries via HTTP service.

```
curl -X 'POST' -H 'Content-Type:application/json' -d @quickstart/tutorial/wikipedia-top-pages-sql.json
http://localhost:18888/druid/v2/sql
```

Formatted Output Results:

```
[
  {
      "page":"Wikipedia:Vandalismusmeldung",
      "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
   }
]
```

Native JSON Queries

- Directly input 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 via HTTP under the Druid directory on the query node.

```
curl -X 'POST' -H 'Content-Type:application/json' -d @quickstart/tutorial/wikipedia-top-pages.json
http://localhost:18888/druid/v2?pretty
```

Output Results:

```
[ {
"timestamp" : "2015-09-12T00:46:58.771Z",
"result" : [ {
  "count" : 33,
  "page" : "Wikipedia:Vandalismusmeldung"
}, {
  "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：2023-12-29 10:34:00

This document elucidates the process of utilizing Apache Druid Kafka Indexing Service for real-time consumption of Kafka data. Prior to commencing this section, akin to a Hadoop cluster, it is imperative to ensure seamless communication between the Kafka cluster and the Druid cluster.

Note
- Both clusters reside within the same VPC, or they inhabit different VPCs, but communication between the two VPCs is unimpeded (for instance, via Cloud Connect Network or Peer Connection).
- If necessary, the host information of the Kafka cluster should be configured into the Druid cluster.

Command Line Methodology

1. Initially, initiate the Kafka broker within the Kafka cluster.

```
./bin/kafka-server-start.sh config/server.properties
```

2. Establish a Kafka topic, christened as '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` represents the Zookeeper address of the Kafka cluster.

3. Prepare a data description file named kafka-mytopic.json on 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` signifies the bootstrap.servers IP and port of your Kafka cluster.

4. Incorporate a Kafka supervisor on the Master node of 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` denotes the node where the overlord process is deployed, typically the Master node.

5. Initiate a console producer on the Kafka cluster.

```
./bin/kafka-console-producer.sh --broker-list {kafka_ip}:9092 --topic mytopic
```

`{kafka_ip}:9092` signifies the bootstrap.servers IP and port of your Kafka cluster.

6. Prepare a query file on the Druid cluster, named as query-mytopic.json.

```
{
  "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. Input some real-time data on Kafka.

```
{"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. Execute a query on the Druid cluster.

```
curl -XPOST -H 'Content-Type: application/json' -d @query-mytopic.json http://{druid_ip}:8082/druid/v2/?pretty
```

`{druid_ip}:8082` represents the broker node of your Druid cluster, typically located on the Master or Router node.

Query results:

```
[ {
  "timestamp" : "2020-03-19T16:00:00.000Z",
  "result" : [ {
 "dimension" : "user",
 "value" : "roni",
 "count" : 2
  } ]
} ]
```

Web Visualization Method

You can ingest and query data from the Kafka cluster through the visual interface of the Druid Web UI console. For more details, please refer to Loading Kafka Data through Data Loader.

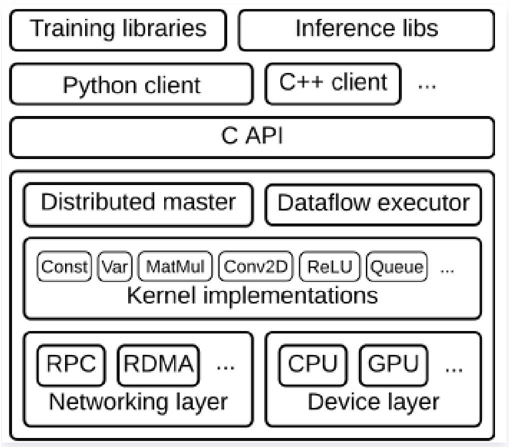
TensorFlow Development Guide
TensorFlow Overview

Last updated：2023-12-29 10:46:18

TensorFlow is an end-to-end open-source platform for machine learning. It possesses a comprehensive and flexible ecosystem, encompassing a variety of tools, libraries, and community resources. This ecosystem empowers researchers to propel the advancement of cutting-edge machine learning technologies, and enables developers to effortlessly construct and deploy applications bolstered by machine learning.

- Effortlessly construct models using intuitive high-level APIs such as Keras in an eager execution environment, facilitating the swift iteration of machine learning models and the straightforward debugging of these models.
- Conduct reliable machine learning production anytime, anywhere. Regardless of the language you utilize, you can effortlessly train and deploy models in the cloud, locally, within browsers, or on devices.
- Powerful research experimentation
  A simple yet flexible architecture that expedites the transformation of novel ideas from concept to code, subsequently facilitating the creation of advanced models, and ultimately enabling their release to the public.

TensorFlow Architecture



- Client
  Defines the computational process as a data flow graph. Utilizes `_Session_` to initialize the execution of the data flow graph.
- Distributed Master
  Prunes certain specific subgraphs within the graph, namely the parameters defined in `Session.run()`. Segments the subgraph into multiple parts that run across different processes and devices. Distributes the graph to various worker processes. Initiates the computation of the subgraph by the worker processes.
- Worker Service (for each task)
  Implements scheduling graph operations using kernels and executes on appropriate hardware (CPU, GPU, etc.). Sends or receives the results of operations to or from other worker processes.
- Kernel Implementation
  Executes an independent graph operation computation.

EMR Supports TensorFlow

- TensorFlow Version: v1.14.0
- Currently, TensorFlow only supports operation on CPU models and does not yet support GPU models.
- Supports distributed training using TensorFlow on Spark.

TensorFlow Development Examples

This document uses TensorFlow v1.4.4 as an example. First, TensorFlow needs to be installed. Switch to the root user, the password is the one set when creating the EMR cluster. Install the python-pip tool first, then install the dependent packages:

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

Code Composition: `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
```

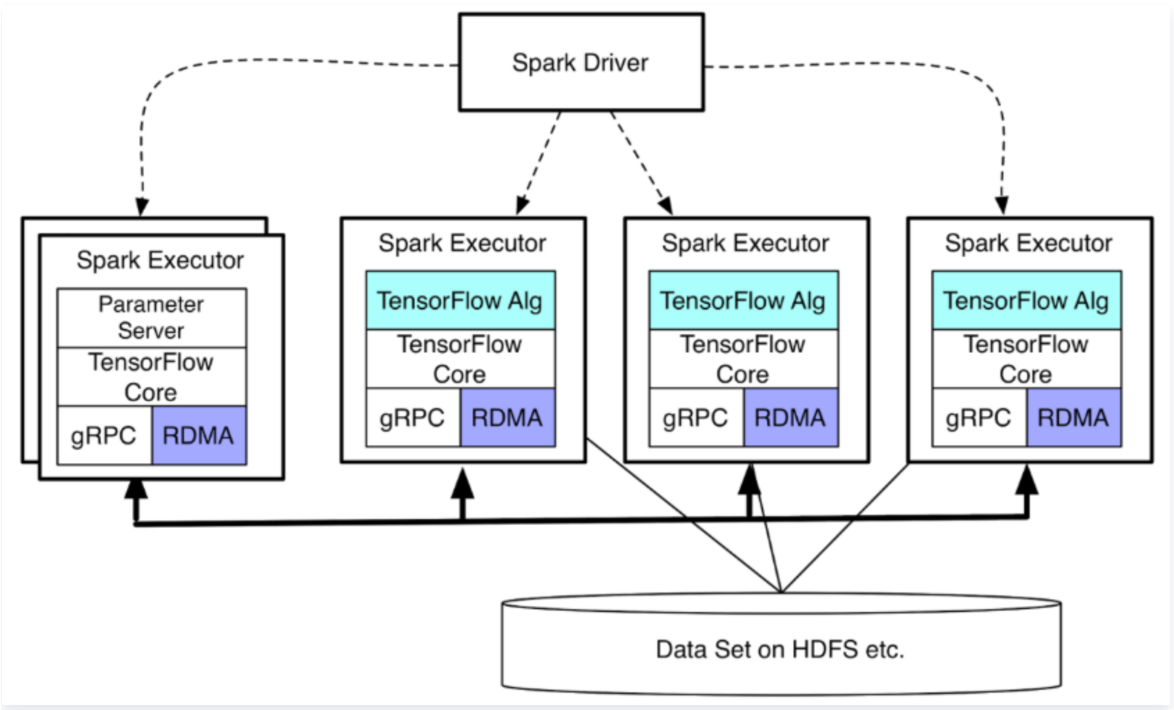
For more usage details, please refer to the official TensorFlow website.

TensorFlowOnSpark Overview

Last updated：2023-12-29 10:46:30

TensorFlowOnSpark offers scalable deep learning for Apache Hadoop and Apache Spark clusters. TensorFlowOnSpark supports all types of TensorFlow programs, enabling asynchronous/synchronous training and inference, while also facilitating model parallelism and parallel data processing. For more information, please refer to the official TensorFlowOnSpark website.

TensorFlowOnSpark Architecture Diagram



TensorFlowOnSpark facilitates direct tensor communication between TensorFlow processes (computation nodes and parameter service nodes). This process-to-process direct communication mechanism allows TensorFlowOnSpark programs to easily scale on additional machines. TensorFlowOnSpark does not involve the Spark driver in tensor communication, thus achieving scalability similar to that of standalone TensorFlow clusters.

Installing TensorFlowOnSpark

1. Navigate to the EMR purchase page and select the product version EMR-2.x.x.
2. In the **optional components** list, select the tensorflowonspark component.
3. By default, tensorflowonspark is installed in the `/usr/local/service/tensorflowonspark` directory.

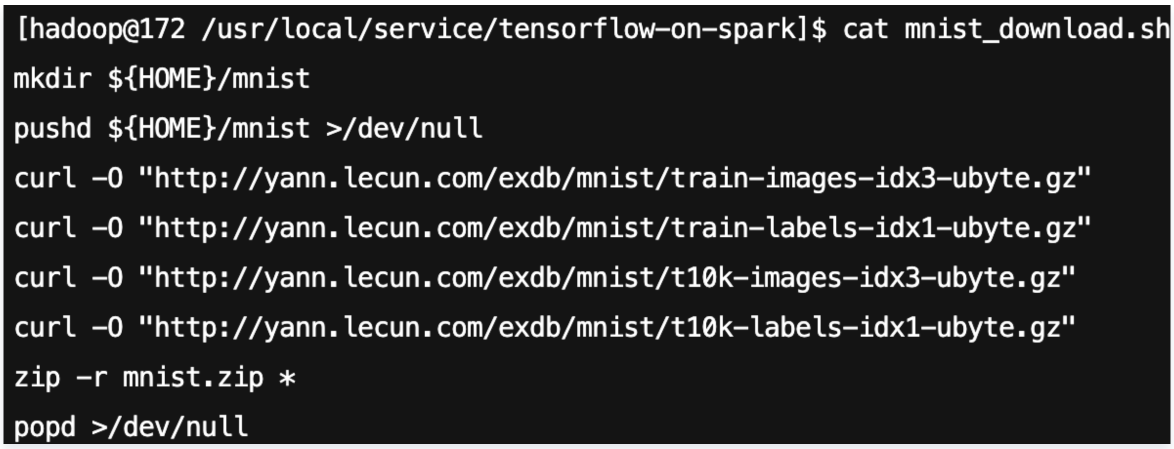
> **Note**
> The components that tensorflowonspark depends on include hive and spark. Therefore, when installing tensorflowonspark, the hive and spark components must also be installed.

Use Case

Within the installed tensorflowonspark component directory, there is a complete set of example codes. For **tensorflowonspark version 1.4.4**, you can proceed as follows:

- Downloading Test Data
  As a Hadoop user, execute the following command in the /usr/local/service/tensorflow-on-spark directory:

```
sh mnist_download.sh
cat mnist_download.sh
```



```
[hadoop@172 /usr/local/service/tensorflow-on-spark]$ 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 Raw 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
```

- Preparation of Feature Data

```
sh prepare_mnist.sh
```

The feature data is now ready for use:

```
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
```

- Training Based on InputMode.SPARK Model

```
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/checkpoint
-rw-r--r-- 1 hadoop supergroup 243332 2020-05-21 11:46
/user/hadoop/mnist_model/events.out.tfevents.1590032704.10.0.0.114
-rw-r--r-- 1 hadoop supergroup 164619 2020-05-21 11:45 /user/hadoop/mnist_model/graph.pbtxt
-rw-r--r-- 1 hadoop supergroup 814168 2020-05-21 11:45 /user/hadoop/mnist_model/model.ckpt-0.data-00000-of-00001
-rw-r--r-- 1 hadoop supergroup 375 2020-05-21 11:45 /user/hadoop/mnist_model/model.ckpt-0.index
-rw-r--r-- 1 hadoop supergroup 64658 2020-05-21 11:45 /user/hadoop/mnist_model/model.ckpt-0.meta
-rw-r--r-- 1 hadoop supergroup 814168 2020-05-21 11:46 /user/hadoop/mnist_model/model.ckpt-595.data-00000-of-00001
-rw-r--r-- 1 hadoop supergroup 375 2020-05-21 11:46 /user/hadoop/mnist_model/model.ckpt-595.index
-rw-r--r-- 1 hadoop supergroup 64658 2020-05-21 11:46 /user/hadoop/mnist_model/model.ckpt-595.meta
drwxr-xr-x - hadoop supergroup 0 2020-05-21 11:46 /user/hadoop/mnist_model/train
```

- Prediction Based on InputMode.SPARK Model

```
sh mnist_inference_with_spark_cpu.sh
```

View the Prediction Results:

```
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
```

- Training 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.tfevents.1590044017.10.0.0.144
-rw-r--r-- 1 hadoop supergroup 40 2020-05-21 14:57 mnist_model/events.out.tfevents.1590044221.10.0.0.144
-rw-r--r-- 1 hadoop supergroup 40 2020-05-21 14:57 mnist_model/events.out.tfevents.1590044227.10.0.0.144
-rw-r--r-- 1 hadoop supergroup 40 2020-05-21 14:57 mnist_model/events.out.tfevents.1590044232.10.0.0.144
-rw-r--r-- 1 hadoop supergroup 40 2020-05-21 14:57 mnist_model/events.out.tfevents.1590044238.10.0.0.144
-rw-r--r-- 1 hadoop supergroup 40 2020-05-21 14:58 mnist_model/events.out.tfevents.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.index
-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.index
-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.index
-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.index
-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 14:58 mnist_model/model.ckpt-393.data-00000-of-00001
-rw-r--r-- 1 hadoop supergroup 375 2020-05-21 14:58 mnist_model/model.ckpt-393.index
-rw-r--r-- 1 hadoop supergroup 76255 2020-05-21 14:58 mnist_model/model.ckpt-393.meta
drwxr-xr-x - hadoop supergroup 0 2020-05-21 14:53 mnist_model/train
```

- Prediction Based on InputMode.TENSORFLOW Model

```
sh mnist_train_with_tf_cpu.sh
```

View the Prediction Results:

```
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
```

Kudu Development Guide
Kudu Overview

Last updated：2023-12-29 10:46:52

Apache Kudu is a distributed, horizontally scalable columnar storage system that enhances Hadoop's storage layer, enabling rapid analysis of rapidly changing data.

Fundamental Characteristics of Kudu

- Efficiently manages OLAP-type workloads.
- Harmoniously integrates with MapReduce, Spark, and other components within the Hadoop ecosystem.
- It can integrate with Impala, serving as a substitute for the commonly used HDFS + Parquet combination in Impala.
- Flexible consistency model.
- Maintains commendable performance in scenarios where sequential and random writes coexist.
- Highly available, utilizing the Raft protocol to ensure reliable data storage.
- Structured data model.

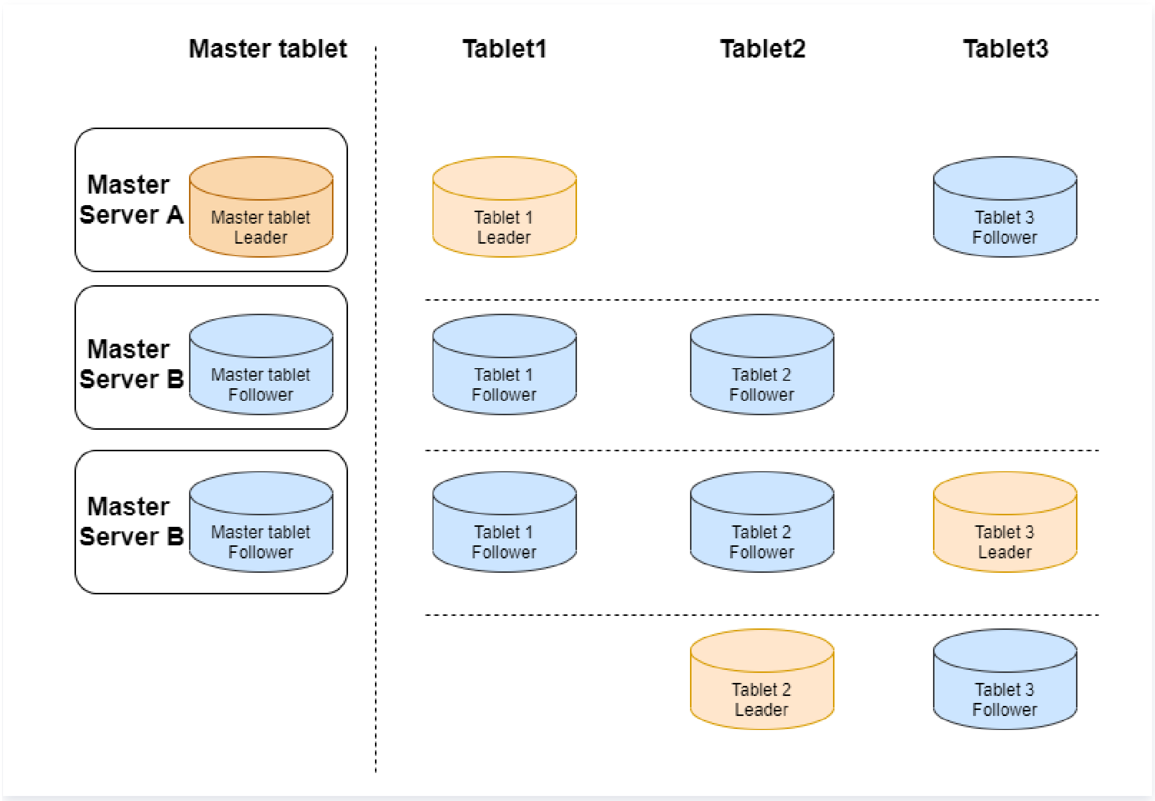
Kudu use cases.

- Applicable to composite scenarios that involve both random access and bulk data scanning.
- Scenarios with high computational demands.
- Application of real-time predictive models, supporting periodic updates based on all historical data.
- Supports data updates, preventing repetitive data migration.
- Supports real-time data backup and querying across different regions.

Fundamental Architecture of Kudu

Kudu comprises the following two types of components:

- The master primarily manages metadata information, monitors servers, and is responsible for tablet reallocation when a server goes down.
- The tserver primarily handles the storage of tablets and the addition, deletion, modification, and retrieval of data.



Utilization of Kudu

EMR-2.4.0 and later versions support the Kudu component. When creating a Hadoop cluster, selecting the Kudu component will create a Kudu cluster. By default, the Kudu cluster includes three Kudu Master services and enables High Availability (HA).

Note
The IPs used in Kudumaster_ip1, Kudumaster_ip2, and Kudumaster_ip3 correspond to the internal network IPs of the nodes where the KuduMaster role is located.

- Integration of Impala and Kudu
  Refer to the Impala Overview to access the Impala command line, and execute the following command to create a new table:

```
CREATE TABLE t2(id BIGINT,name STRING,PRIMARY KEY(id))PARTITION BY HASH PARTITIONS 2 STORED AS KUDU
TBLPROPERTIES (
'kudu.master_addresses' = '$Kudumaster_ip1,$Kudumaster_ip2,$Kudumaster_ip3',
'kudu.num_tablet_replicas' = '1');
```

  Upon successful execution, the following prompt message will be displayed:

```
Query: create TABLE t2 (id BIGINT,name STRING,PRIMARY KEY(id)) PARTITION BY HASH PARTITIONS 2 STORED AS KUDU
TBLPROPERTIES (
'kudu.master_addresses' = '$Kudumaster_ip1,$Kudumaster_ip2,$Kudumaster_ip3',
'kudu.num_tablet_replicas' = '1')
Fetched 0 row(s) in 0.12s
```

  To view the created table, enter the following command:

```
usr/local/service/kudu/bin/kudu table list $Kudumaster_ip1,$Kudumaster_ip2,$Kudumaster_ip3
```

- Data Insertion
  In the Impala command line, execute the following command to insert data into the table:

```
insert into t2 values(1, 'test');
```

- Data Query Based on Impala
  In the Impala command line, execute the following command to query data in the table:

```
[172.30.0.98:27001] > select * from t2;
```

  You can query the data that has been inserted into the table.
- Additional Commands
- Cluster Health Check

```
[hadoop@172 root]$ /usr/local/service/kudu/bin/kudu cluster ksck
172.30.0.240,172.30.1.167,172.30.0.96,172.30.0.94,172.30.0.214
```

- Create 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}'
```

- 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
```

- Inspect 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
```

Data Migration Guide for Kudu Node Scale-In

Last updated：2023-12-29 10:47:10

This document primarily elucidates the data migration process required when scaling down the tserver of Core nodes in a Kudu cluster.

Note
The Kudu cluster type supports the scaling down of Core nodes. This feature is not open by default. If required, please submit a ticket to request activation.

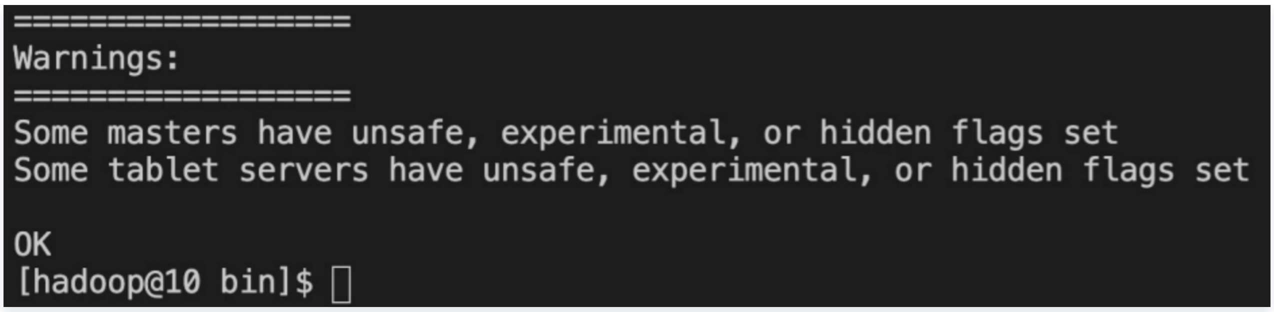
Prior to decommissioning a tserver node, the rebalance tool can be utilized for data migration. Please be aware that only one tserver can be taken offline at a time. If multiple servers need to be decommissioned, the following steps must be repeated for each.

Migration Based on Kudu's Rebalance Tool

1. Ensure the cluster status is optimal.

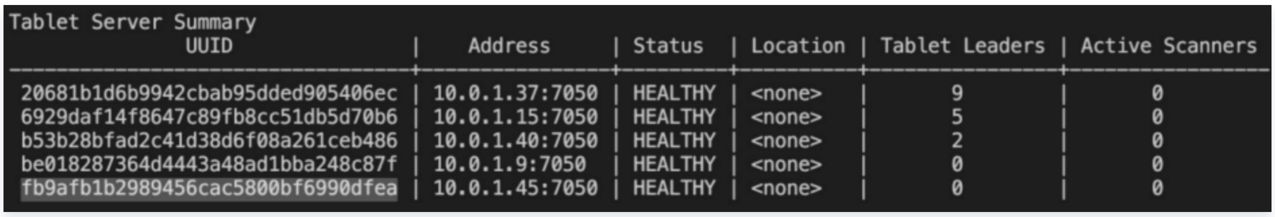
```
/usr/local/service/kudu/bin/kudu cluster ksck $KuduMaster1_ip::port,$KuduMaster2_ip::port,$KuduMaster3_ip::port
```

   Here, $KuduMaster1_ip, $KuduMaster2_ip, and $KuduMaster3_ip represent the IPs of the three nodes in the EMR cluster where the KuduMaster roles are located; port refers to the port number for connecting to Kudu, which is 7051 by default.



```
==================
Warnings:
==================
Some masters have unsafe, experimental, or hidden flags set
Some tablet servers have unsafe, experimental, or hidden flags set

OK
[hadoop@10 bin]$ 
```

2. Use the ksck command from Step 1 to obtain the uid of the node to be taken offline.



```
Tablet Server Summary
               UUID               |     Address      | Status  | Location | Tablet Leaders | Active Scanners
----------------------------------+------------------+---------+----------+----------------+-----------------
 20681b1d6b9942cbab95dded905406ec | 10.0.1.37:7050   | HEALTHY | <none>   |       9        |       0
 6929daf14f8647c89fb8cc51db5d70b6 | 10.0.1.15:7050   | HEALTHY | <none>   |       5        |       0
 b53b28bfad2c41d38d6f08a261ceb486 | 10.0.1.40:7050   | HEALTHY | <none>   |       2        |       0
 be018287364d4443a48ad1bba248c87f | 10.0.1.9:7050    | HEALTHY | <none>   |       0        |       0
 fb9afb1b2989456cac5800bf6990dfea | 10.0.1.45:7050   | HEALTHY | <none>   |       0        |       0
```

   For instance, consider the node with uid fb9afb1b2989456cac5800bf6990dfea.

3. Transition the node with uid fb9afb1b2989456cac5800bf6990dfea into maintenance mode.

```
/usr/local/service/kudu/bin/kudu tserver state enter_maintenance
$KuduMaster1_ip::port,$KuduMaster2_ip::port,$KuduMaster3_ip::port fb9afb1b2989456cac5800bf6990dfea
```

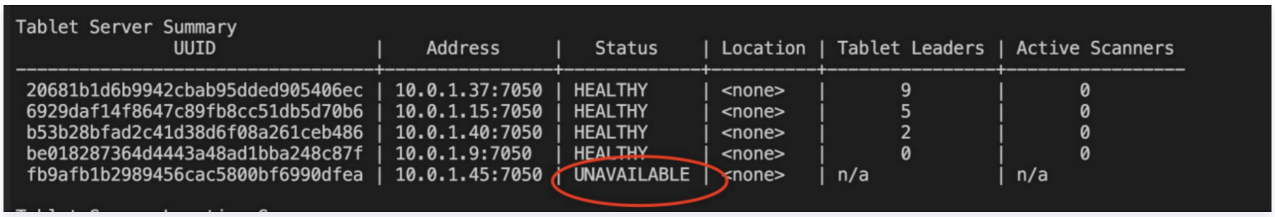
4. Execute the rebalance command

```
/usr/local/service/kudu/bin/kudu cluster rebalance $KuduMaster1_ip::port,$KuduMaster2_ip::port,$KuduMaster3_ip::port --
ignored_tservers fb9afb1b2989456cac5800bf6990dfea --move_replicas_from_ignored_tservers
```

   Upon completion of the command execution, use the ksck command for verification once more. If the status is optimal, proceed with the subsequent steps.

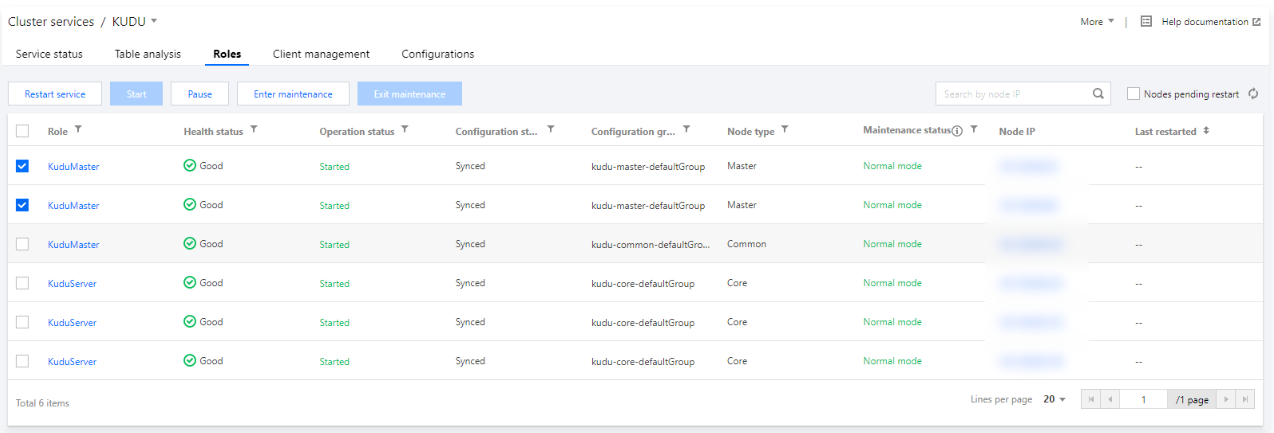
5. Suspend the tserver process for the node 10.0.1.45 corresponding to fb9afb1b2989456cac5800bf6990dfea. Note that at this

point, using the ksck command indicates that the cluster status is unhealthy, necessitating a master restart.



```
Tablet Server Summary
               UUID               |     Address      |   Status    | Location | Tablet Leaders | Active Scanners
----------------------------------+------------------+-------------+----------+----------------+-----------------
 20681b1d6b9942cbab95dded905406ec | 10.0.1.37:7050   | HEALTHY     | <none>   |       9        |       0
 6929daf14f8647c89fb8cc51db5d70b6 | 10.0.1.15:7050   | HEALTHY     | <none>   |       5        |       0
 b53b28bfad2c41d38d6f08a261ceb486 | 10.0.1.40:7050   | HEALTHY     | <none>   |       2        |       0
 be018287364d4443a48ad1bba248c87f | 10.0.1.9:7050    | HEALTHY     | <none>   |       0        |       0
 fb9afb1b2989456cac5800bf6990dfea | 10.0.1.45:7050   | UNAVAILABLE | <none>   | n/a            | n/a
```

6. Restart the master via the EMR Console. Note that the restart should be performed manually, one by one (it is not recommended to use the console's rolling restart). After the restart, use the ksck command to ensure the cluster status is healthy.



	Role	Health status	Operation status	Configuration st...	Configuration gr...	Node type	Maintenance status	Node IP	Last restarted
✓	KuduMaster	Good	Started	Synced	kudu-master-defaultGroup	Master	Normal mode	[redacted]	--
✓	KuduMaster	Good	Started	Synced	kudu-master-defaultGroup	Master	Normal mode	[redacted]	--
	KuduMaster	Good	Started	Synced	kudu-common-defaultGro...	Common	Normal mode	[redacted]	--
	KuduServer	Good	Started	Synced	kudu-core-defaultGroup	Core	Normal mode	[redacted]	--
	KuduServer	Good	Started	Synced	kudu-core-defaultGroup	Core	Normal mode	[redacted]	--
	KuduServer	Good	Started	Synced	kudu-core-defaultGroup	Core	Normal mode	[redacted]	--

Total 6 items   Lines per page 20   1 /1 page

Ranger Development Guide
Ranger Overview

Last updated：2024-01-02 14:16:27

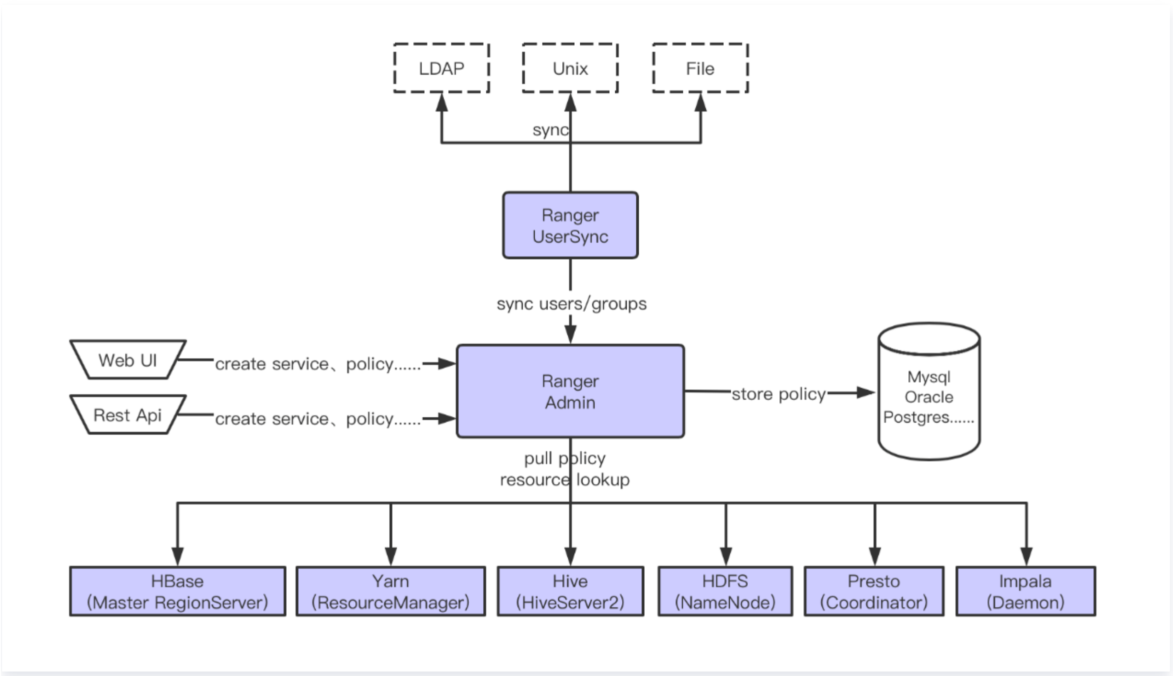
Introduction to Ranger

Ranger is a centralized security management framework in the field of big data, enabling centralized security management of Hadoop ecosystem components. Users can secure access to data in the cluster through Ranger. It primarily oversees Hadoop platform components, initiates services, and controls resource access, with its core philosophy being:

- Users can utilize the REST API provided by Ranger or employ the Web UI offered by Ranger for centralized management of big data components.
- Authorization can be implemented based on roles and attributes for big data components.
- Centralized management is implemented for the security audits involved with big data components.

Ranger Architecture

Ranger is primarily composed of three components: Ranger Admin, Ranger UserSync, and Ranger Plugin. Both Ranger Admin and Ranger UserSync are individual JVM processes, while Ranger Plugin needs to be installed on different nodes depending on the component.



- Ranger Admin: Manages user-configured policies and created services, audit logs and reports, persists configured policies and created services to the database, and provides them for regular queries by the Plugin.
- Ranger UserSync: Synchronizes relevant information from LDAP, File, and Unix into Ranger Admin. For instance, it synchronizes user information and group information from the user's LDAP directory access system or Unix. To synchronize Unix user and group information, the unixAuthenticationService process needs to be activated. Concurrently, the synchronized information is persisted.
- Ranger Plugin: The Plugin is deployed in the service nodes as required and synchronizes policy information with Ranger Admin regularly.

For component integration with Ranger, please refer to the following table:

Service	install Node	EMR Version
HDFS	NameNode	EMR-V 2.0.1 and higher versions
Hbase	Master、RegionServer	EMR-V 2.0.1 and higher versions

Hive	HiveServer2	EMR-V 2.0.1 and higher versions
Yarn	ResourceManager	EMR-V 2.0.1 and higher versions
Presto	All Coordinator	EMR-V 2.0.1 and higher versions
Impala	All Daemon	EMR-V 2.2.0 and higher versions
Kudu	All Master	EMR-V 3.2.0 version

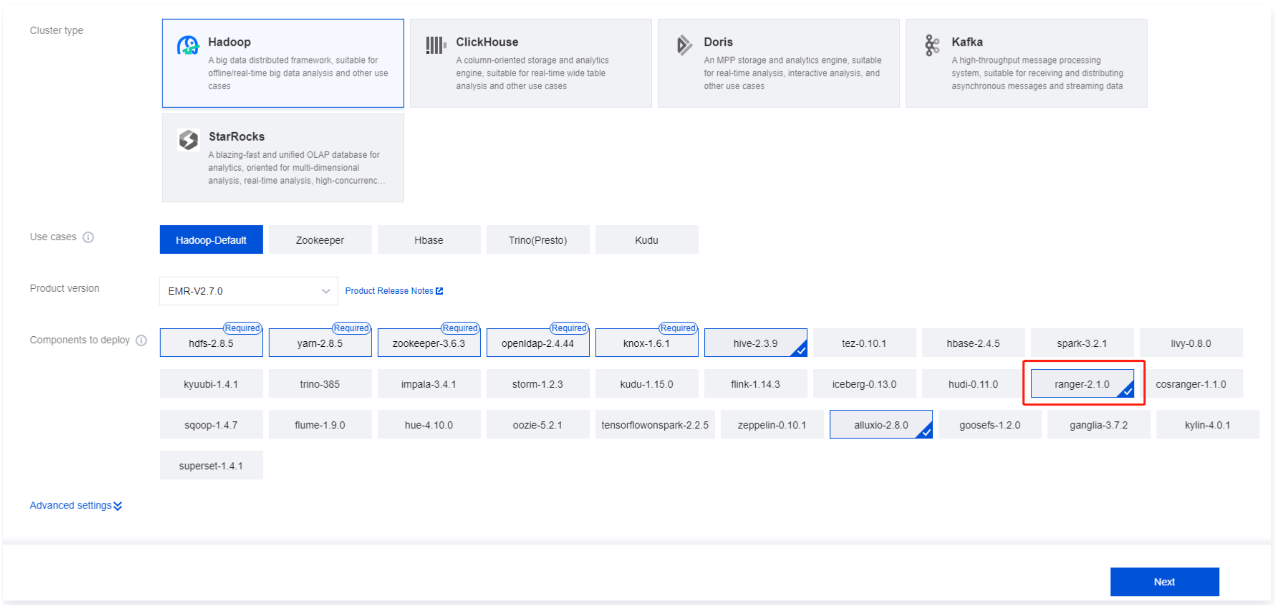
Ranger User Guide
Integrating HDFS with Ranger

Last updated：2024-01-02 14:16:53

Preparations

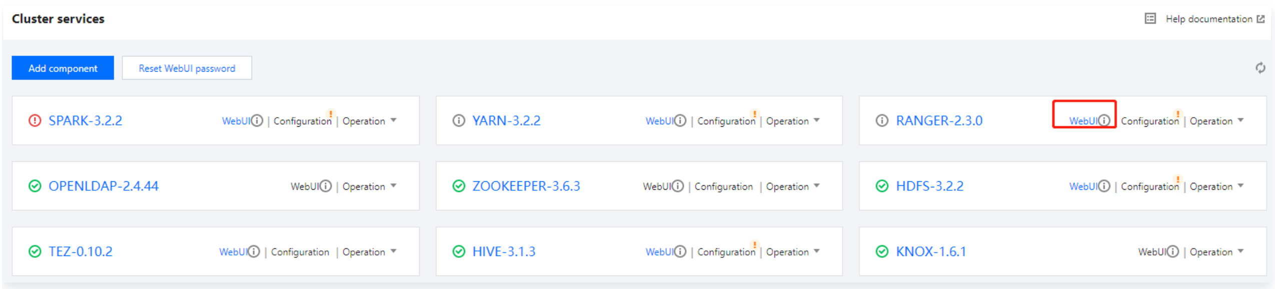
Support is exclusively provided for clusters that have selected the optional Ranger component at the time of purchase. If the Ranger component is added to an already established cluster, there may be instances where the Web UI becomes inaccessible. By default, upon Ranger installation, both Ranger Admin and Ranger UserSync are deployed on the Master node, while the Ranger Plugin is embedded and deployed on the main daemon node of the component.
During the creation of a cluster, when selecting Hadoop as the cluster type, Ranger can be chosen from the optional components. The version of Ranger may vary depending on the selected version of EMR.

Note
When the cluster type is Hadoop and the optional Ranger component is selected, EMR-Ranger will, by default, create services for HDFS and YARN and establish default policies.


Cluster type: Hadoop
Use cases: Hadoop-Default
Product version: EMR-V2.7.0
Components to deploy: hdfs-2.8.5, yarn-2.8.5, zookeeper-3.6.3, openldap-2.4.44, knox-1.6.1, hive-2.3.9, ranger-2.1.0, alluxio-2.8.0
Hadoop
A big data distributed framework, suitable for offline/real-time big data analysis and other use cases
ClickHouse
A column-oriented storage and analytics engine, suitable for real-time wide table analysis and other use cases
Doris
An MPP storage and analytics engine, suitable for real-time analysis, interactive analysis, and other use cases
Kafka
A high-throughput message processing system, suitable for receiving and distributing asynchronous messages and streaming data
StarRocks
A blazing-fast and unified OLAP database for analytics, oriented for multi-dimensional analysis, real-time analysis, high-concurrenc...
Zookeeper
Hbase
Trino(Presto)
Kudu
Product Release Notes
Required
tez-0.10.1
hbase-2.4.5
spark-3.2.1
livy-0.8.0
kyuubi-1.4.1
trino-385
impala-3.4.1
storm-1.2.3
kudu-1.15.0
flink-1.14.3
iceberg-0.13.0
hudi-0.11.0
cosranger-1.1.0
sqoop-1.4.7
flume-1.9.0
hue-4.10.0
oozie-5.2.1
tensorflowonspark-2.2.5
zeppelin-0.10.1
goosefs-1.2.0
ganglia-3.7.2
kylin-4.0.1
superset-1.4.1
Advanced settings
Next

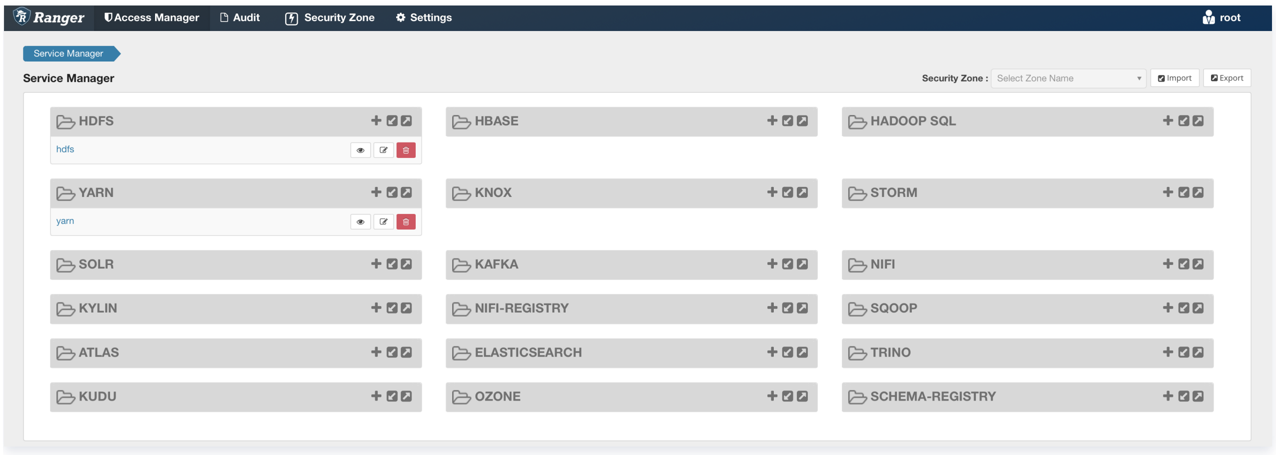
Ranger Web UI

Before accessing the Ranger Web UI, please ensure that the purchased cluster is configured with a public IP. Then, in the cluster service, click on the Web UI address link of the Ranger component.


Cluster services
Help documentation
Add component
Reset WebUI password
SPARK-3.2.2 WebUI | Configuration | Operation
YARN-3.2.2 WebUI | Configuration | Operation
RANGER-2.3.0 WebUI | Configuration | Operation
OPENLDAP-2.4.44 WebUI | Operation
ZOOKEEPER-3.6.3 WebUI | Configuration | Operation
HDFS-3.2.2 WebUI | Configuration | Operation
TEZ-0.10.2 WebUI | Configuration | Operation
HIVE-3.1.3 WebUI | Configuration | Operation
KNOX-1.6.1 WebUI | Operation

After the Web UI address link redirects, you will be prompted to enter a username and password, which are the ones set during the

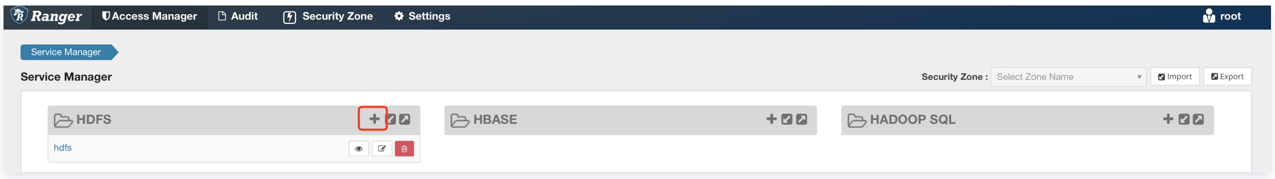
purchase of the cluster.



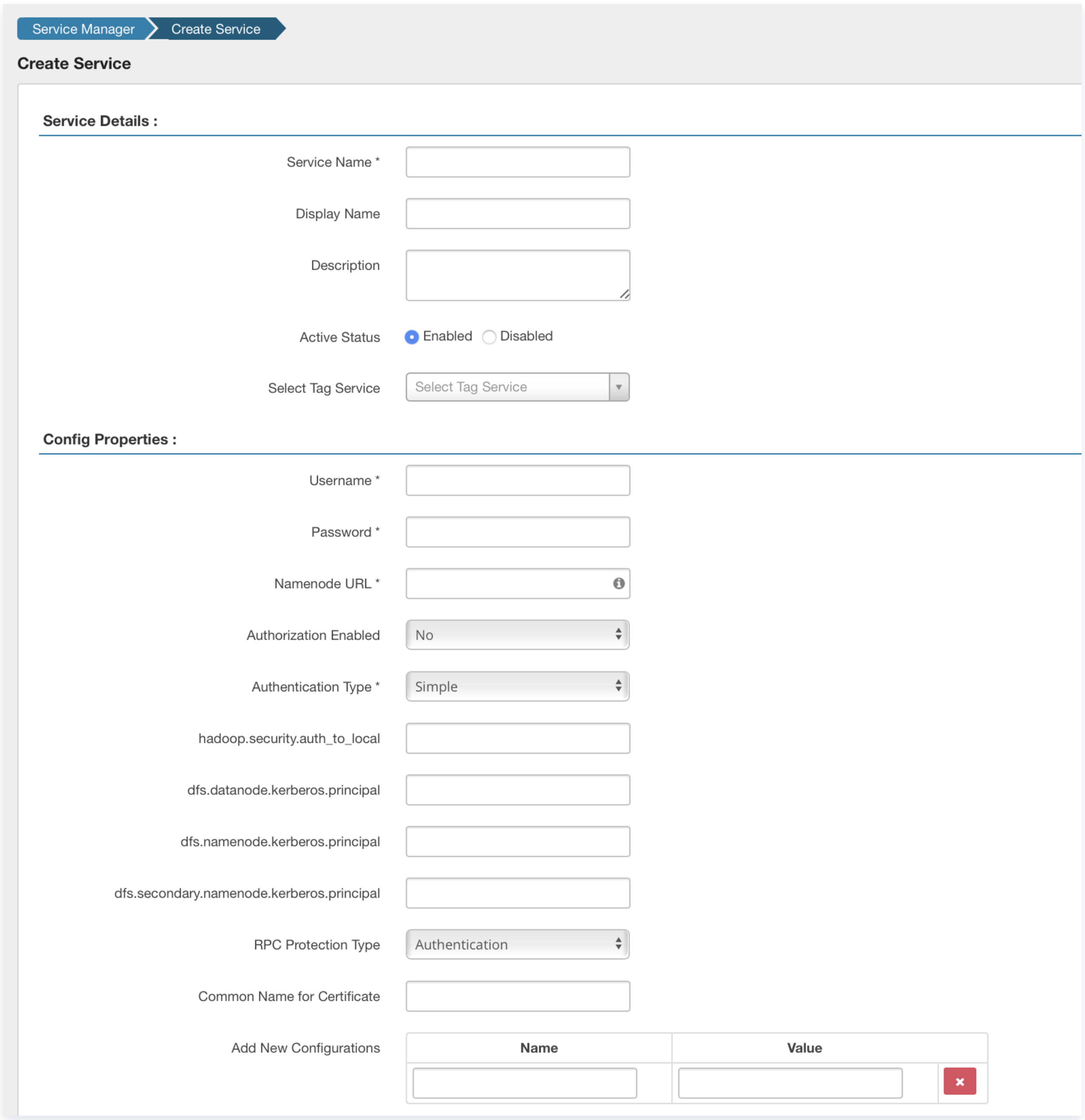
HDFS Integration with Ranger

Note
Please ensure that the HDFS related services are functioning properly and that the current cluster has Ranger installed.

1. Utilize the EMR Ranger Web UI page to add the EMR Ranger HDFS service.



2. Configure the relevant parameters for the EMR Ranger HDFS Service.



Service Manager > Create Service

Create Service

Service Details :

Service Name *	
Display Name	
Description	
Active Status	[✓ Enabled] Disabled
Select Tag Service	Select Tag Service

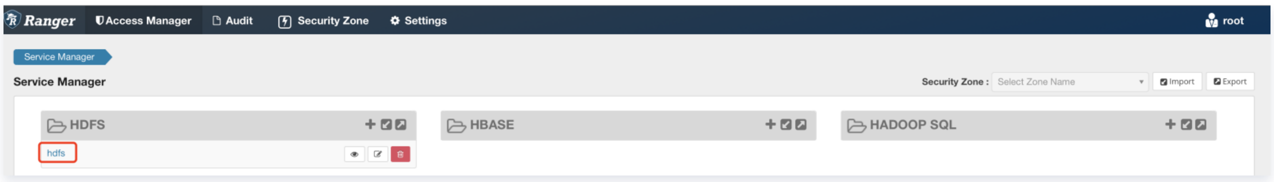
Config Properties :

Username *	
Password *	
Namenode URL *	
Authorization Enabled	No
Authentication Type *	Simple
hadoop.security.auth_to_local	
dfs.datanode.kerberos.principal	
dfs.namenode.kerberos.principal	
dfs.secondary.namenode.kerberos.principal	
RPC Protection Type	Authentication
Common Name for Certificate	
Add New Configurations	Name / Value

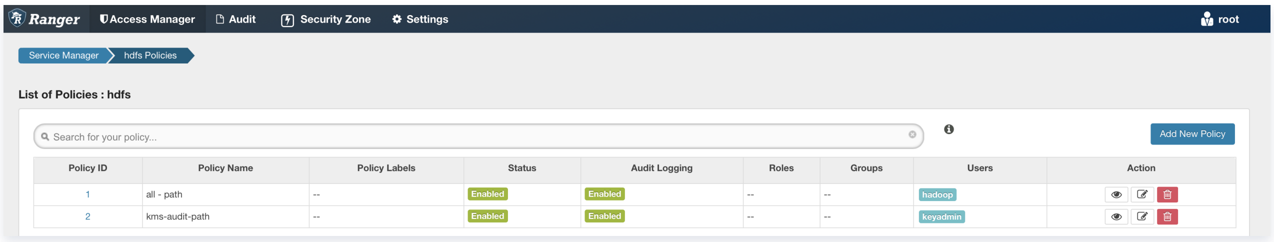
Parameter	Mandatory Field	Explanation
Service Name	Supported	Service name, as displayed on the main HDFS component of the Ranger Web UI.
Description	Not required	Service Description Information
Active Status	Default Value	Service activation status, enabled by default.
UserName	Supported	Username for resource utilization.
Password	Supported	User Password
NameNode URL	Supported	HDFS Address
Authorization Enable	Default Value	Select 'No' for standard clusters; opt for 'Yes' for high-security clusters.
Authorization Type	Supported	Simple: Standard Cluster; Kerberos: High-Security Cluster.

3. EMR Ranger HDFS Resource Permission Configuration.

- Click on the configured EMR Ranger HDFS Service.

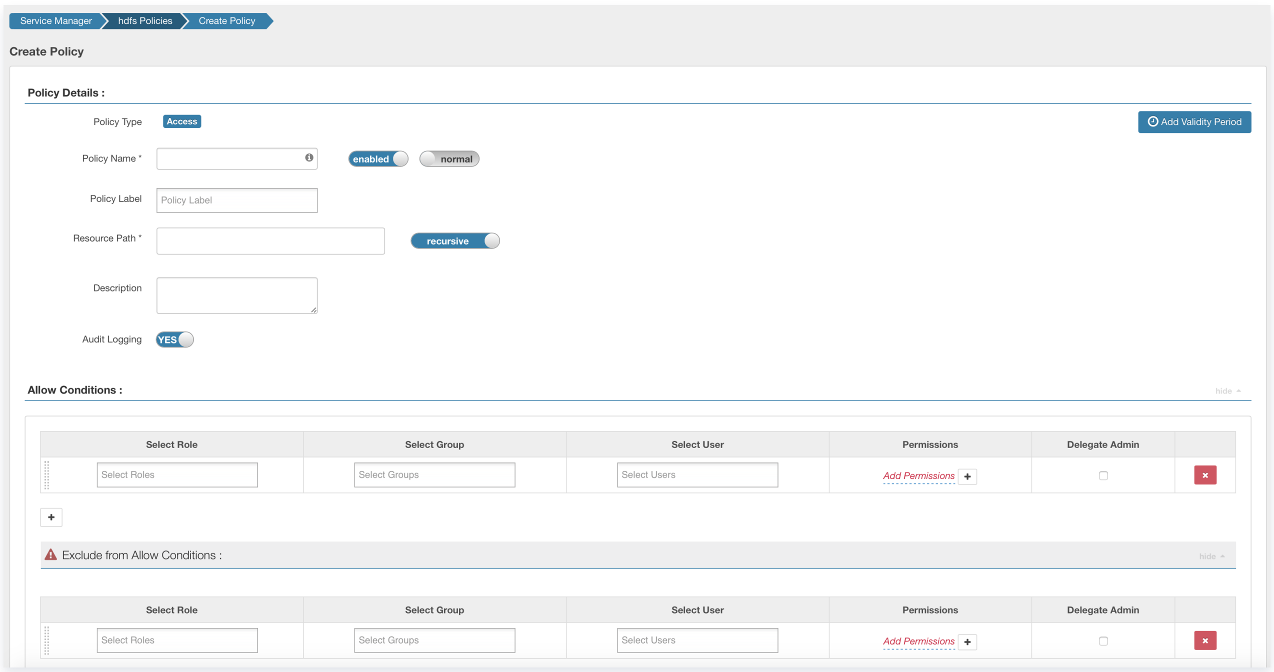


- Configure Policy



List of Policies : hdfs

Policy ID	Policy Name	Policy Labels	Status	Audit Logging	Roles	Groups	Users	Action
1	all - path	--	Enabled	Enabled	--	--	hadoop	
2	kms-audit-path	--	Enabled	Enabled	--	--	keyadmin	



Create Policy

Policy Details :

Policy Type	Access
Policy Name *	
Policy Label	Policy Label
Resource Path *	
Description	
Audit Logging	YES

Allow Conditions :

Select Role	Select Group	Select User	Permissions	Delegate Admin
Select Roles	Select Groups	Select Users	Add Permissions	

Exclude from Allow Conditions :

Select Role	Select Group	Select User	Permissions	Delegate Admin
Select Roles	Select Groups	Select Users	Add Permissions	

4. Upon completion of the Policy addition, please wait approximately half a minute for the Policy to take effect. Once activated, 'user1' will have read and write access to the HDFS file system's /user.

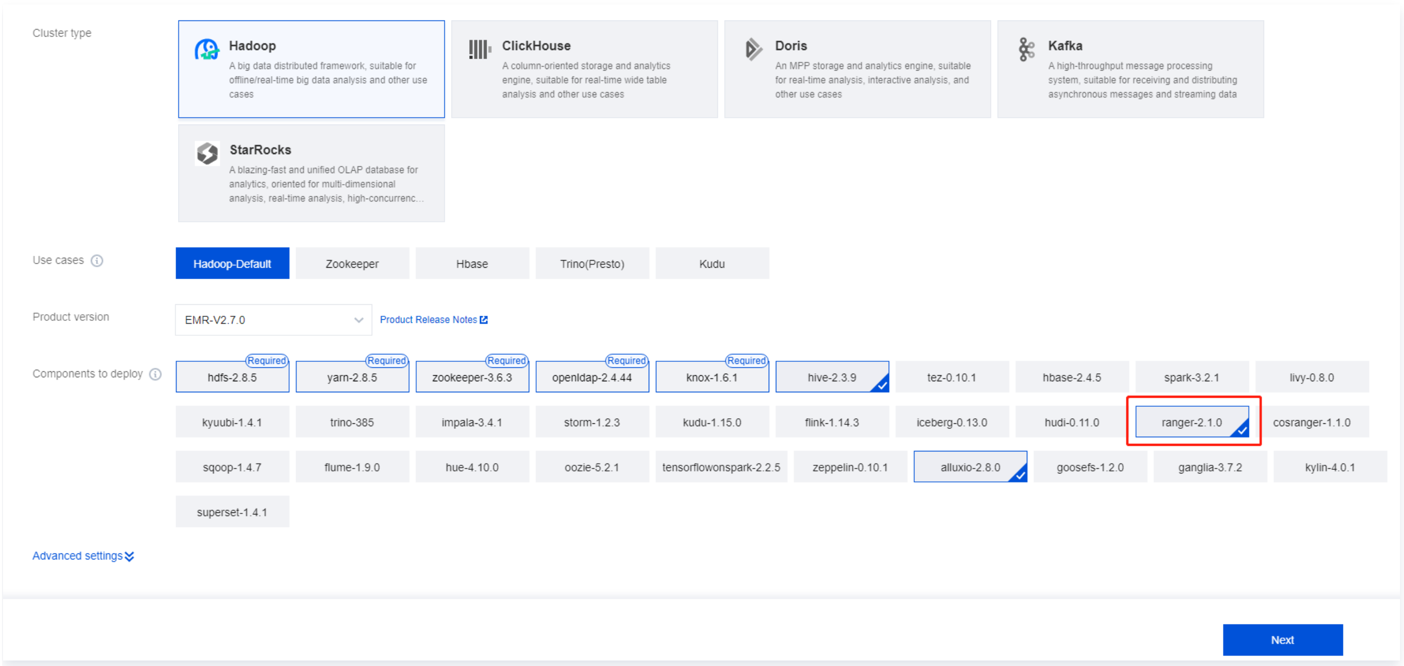
Integrating YARN with Ranger

Last updated：2024-01-02 14:17:10

Preparations

During the installation of Ranger, both Ranger Admin and Ranger UserSync are deployed on the Master node, while Ranger Plugin is deployed on the node hosting the embedded component's primary daemon process.
During the creation of a cluster, when selecting Hadoop as the cluster type, Ranger can be chosen from the optional components. The version of Ranger may vary depending on the selected version of EMR.

Note
When the cluster type is Hadoop and the optional Ranger component is selected, EMR-Ranger will, by default, create services for HDFS and YARN and establish default policies.



Cluster type

- [✓ Hadoop] A big data distributed framework, suitable for offline/real-time big data analysis and other use cases
- ClickHouse A column-oriented storage and analytics engine, suitable for real-time wide table analysis and other use cases
- Doris An MPP storage and analytics engine, suitable for real-time analysis, interactive analysis, and other use cases
- Kafka A high-throughput message processing system, suitable for receiving and distributing asynchronous messages and streaming data
- StarRocks A blazing-fast and unified OLAP database for analytics, oriented for multi-dimensional analysis, real-time analysis, high-concurrenc...

Use cases: [✓ Hadoop-Default] Zookeeper Hbase Trino(Presto) Kudu

Product version: EMR-V2.7.0 Product Release Notes

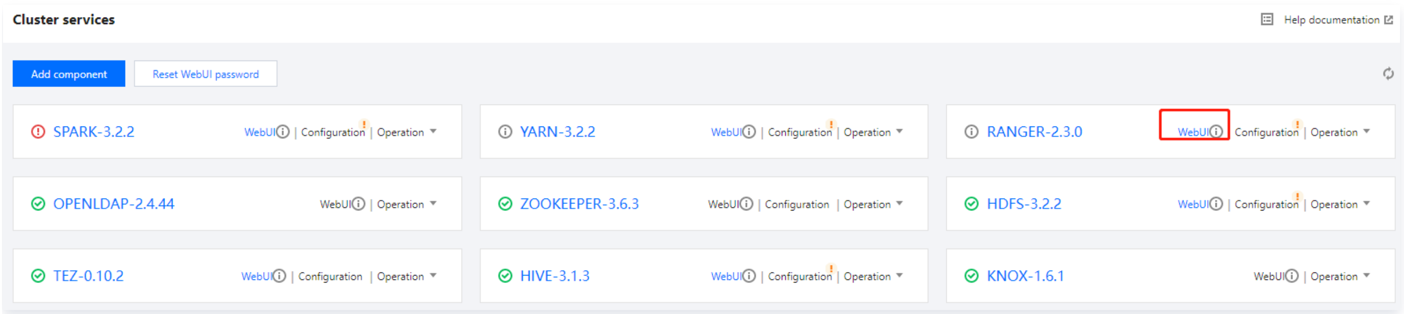
Components to deploy: [✓ hdfs-2.8.5 (Required)] [✓ yarn-2.8.5 (Required)] [✓ zookeeper-3.6.3 (Required)] [✓ openldap-2.4.44 (Required)] [✓ knox-1.6.1 (Required)] [✓ hive-2.3.9] tez-0.10.1 hbase-2.4.5 spark-3.2.1 livy-0.8.0 kyuubi-1.4.1 trino-385 impala-3.4.1 storm-1.2.3 kudu-1.15.0 flink-1.14.3 iceberg-0.13.0 hudi-0.11.0 [✓ ranger-2.1.0] cosranger-1.1.0 sqoop-1.4.7 flume-1.9.0 hue-4.10.0 oozie-5.2.1 tensorflowonspark-2.2.5 zeppelin-0.10.1 [✓ alluxio-2.8.0] goosefs-1.2.0 ganglia-3.7.2 kylin-4.0.1 superset-1.4.1

Advanced settings

Next

Ranger Web UI

Before accessing the Ranger Web UI, please ensure that the purchased cluster is configured with a public IP. Then, in the cluster service, click on the Web UI address link of the Ranger component.



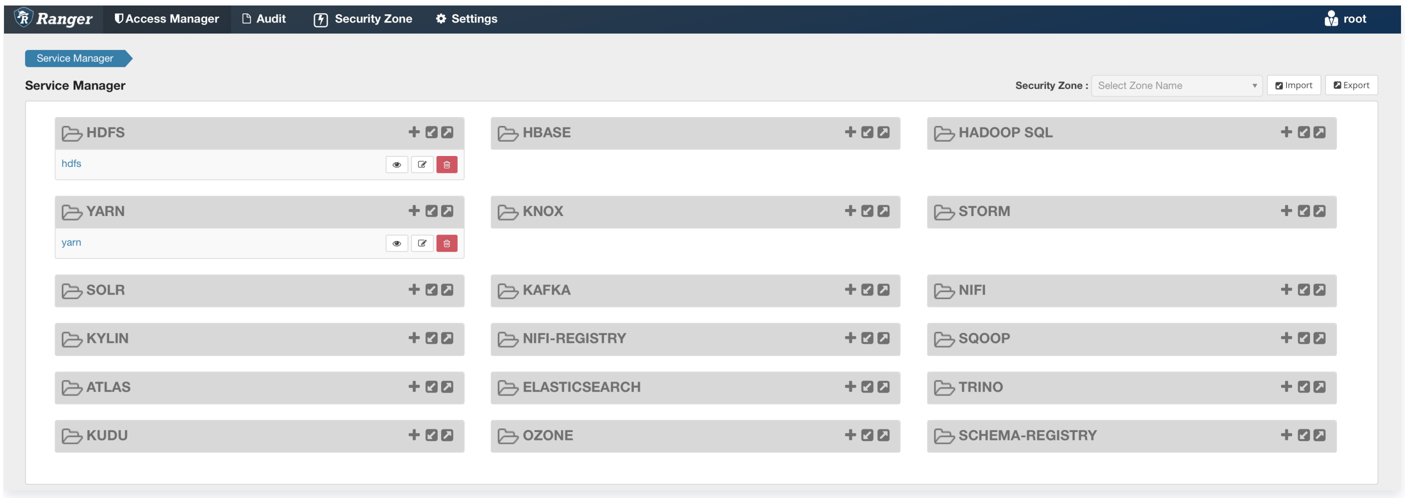
Cluster services — Help documentation

Add component | Reset WebUI password

SPARK-3.2.2	WebUI | Configuration | Operation
YARN-3.2.2	WebUI | Configuration | Operation
RANGER-2.3.0	WebUI | Configuration | Operation
OPENLDAP-2.4.44	WebUI | Operation
ZOOKEEPER-3.6.3	WebUI | Configuration | Operation
HDFS-3.2.2	WebUI | Configuration | Operation
TEZ-0.10.2	WebUI | Configuration | Operation
HIVE-3.1.3	WebUI | Configuration | Operation
KNOX-1.6.1	WebUI | Operation

After the Web UI address link redirects, you will be prompted to enter a username and password, which are the ones set during the

purchase of the cluster.

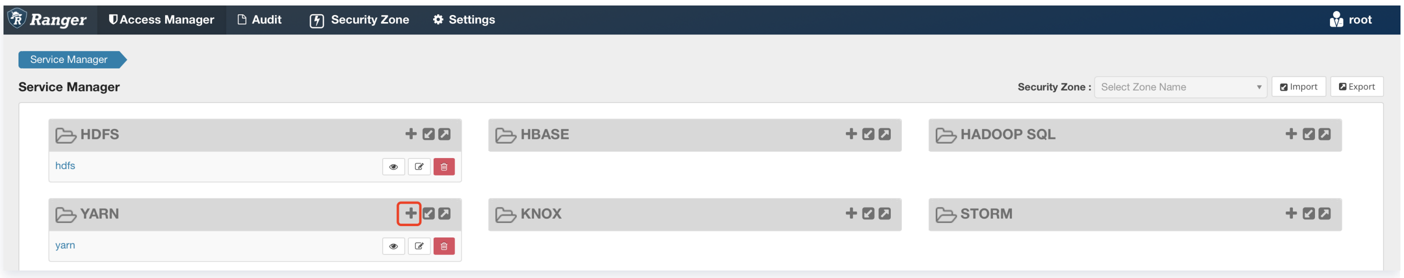


Integration of YARN with Ranger

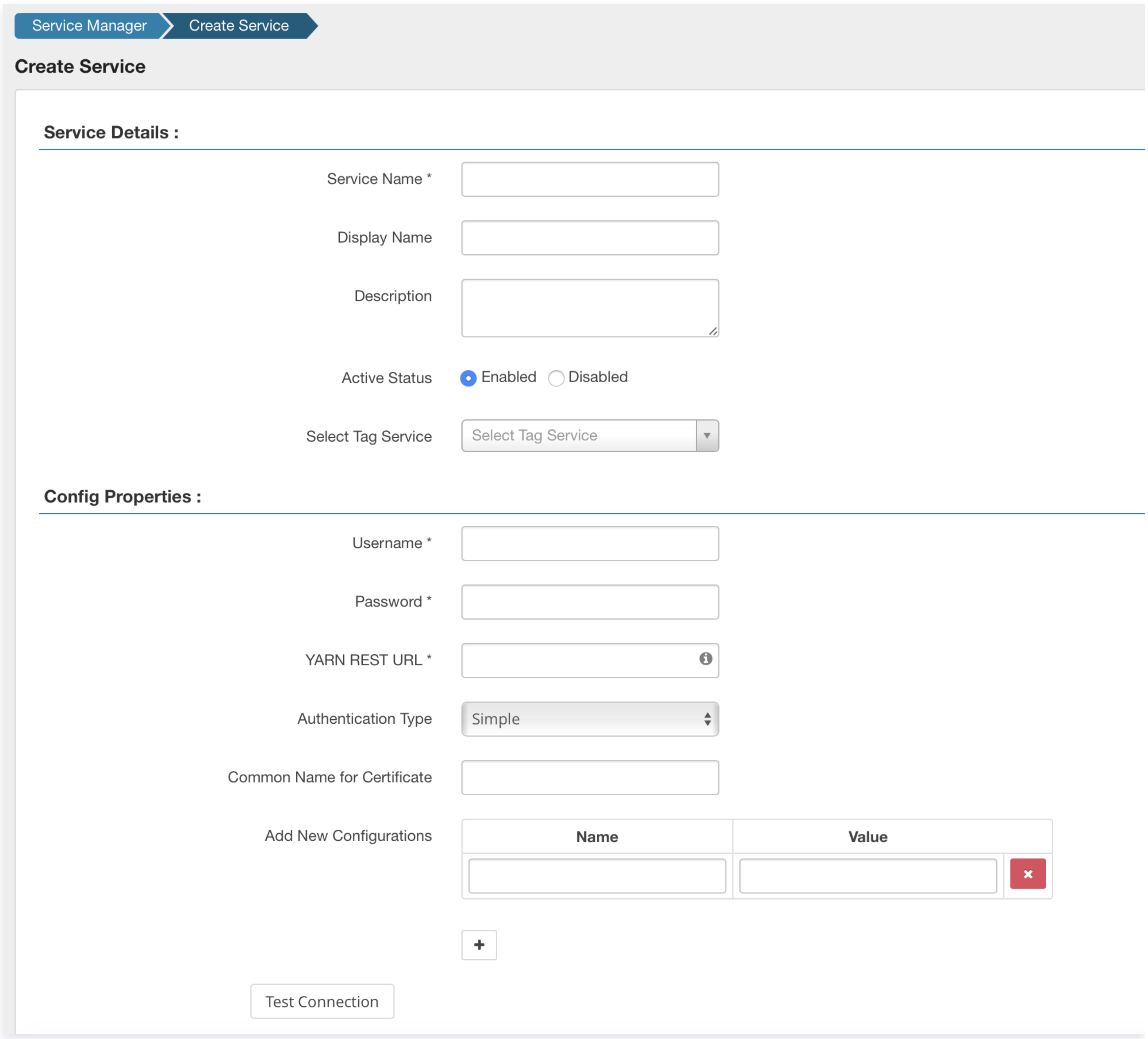
Note
Please ensure that YARN-related services are operating normally and that the current cluster has Ranger installed.

EMR Ranger YARN currently only supports the ACL of the Capacity Scheduler queue and does not support the ACL of the Fair Scheduler queue. The Ranger YARN queue ACL and the built-in Capacity Scheduler configuration of YARN take effect together, with the latter having a higher priority. Only when the built-in Capacity Scheduler configuration of YARN rejects the verification will the Ranger YARN permissions be verified. **It is recommended not to set the ACL in the configuration file, but to use Ranger to set the ACL.**

1. Add the EMR Ranger YARN service through the EMR Ranger Web UI page.



2. Configure the relevant parameters for the EMR Ranger YARN Service.


Service Manager > Create Service

Create Service

Service Details :

Service Name *

Display Name

Description

Active Status: [✓ Enabled] Disabled

Select Tag Service: Select Tag Service

Config Properties :

Username *

Password *

YARN REST URL *

Authentication Type: Simple

Common Name for Certificate

Add New Configurations

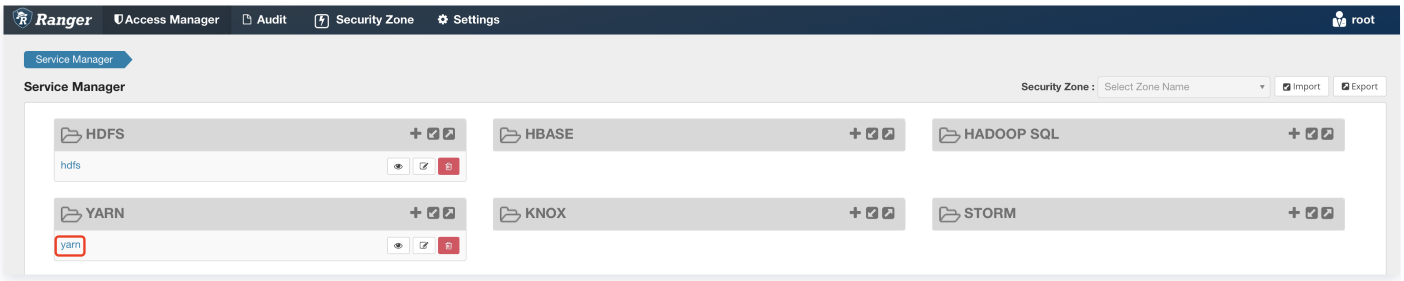
Name	Value

Test Connection

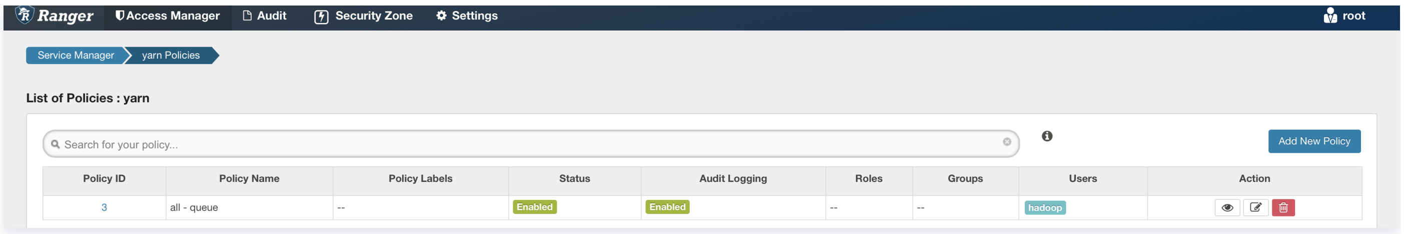
Parameter	Mandatory Field	Explanation
Service Name	Supported	Service name, as displayed on the main YARN component of the Ranger Web UI.
Description	Not required	Service Description Information
Active Status	Default Value	Service activation status, enabled by default.
UserName	Supported	Username for resource utilization.
Password	Supported	User Password
NameNode URL	Supported	YARN Address
Authorization Enable	Default Value	Select 'No' for standard clusters; opt for 'Yes' for high-security clusters.
Authorization Type	Supported	Simple: Standard Cluster; Kerberos: High-Security Cluster.

3. EMR Ranger YARN Resource Permission Configuration.

- Click on the configured EMR Ranger HDFS Service.

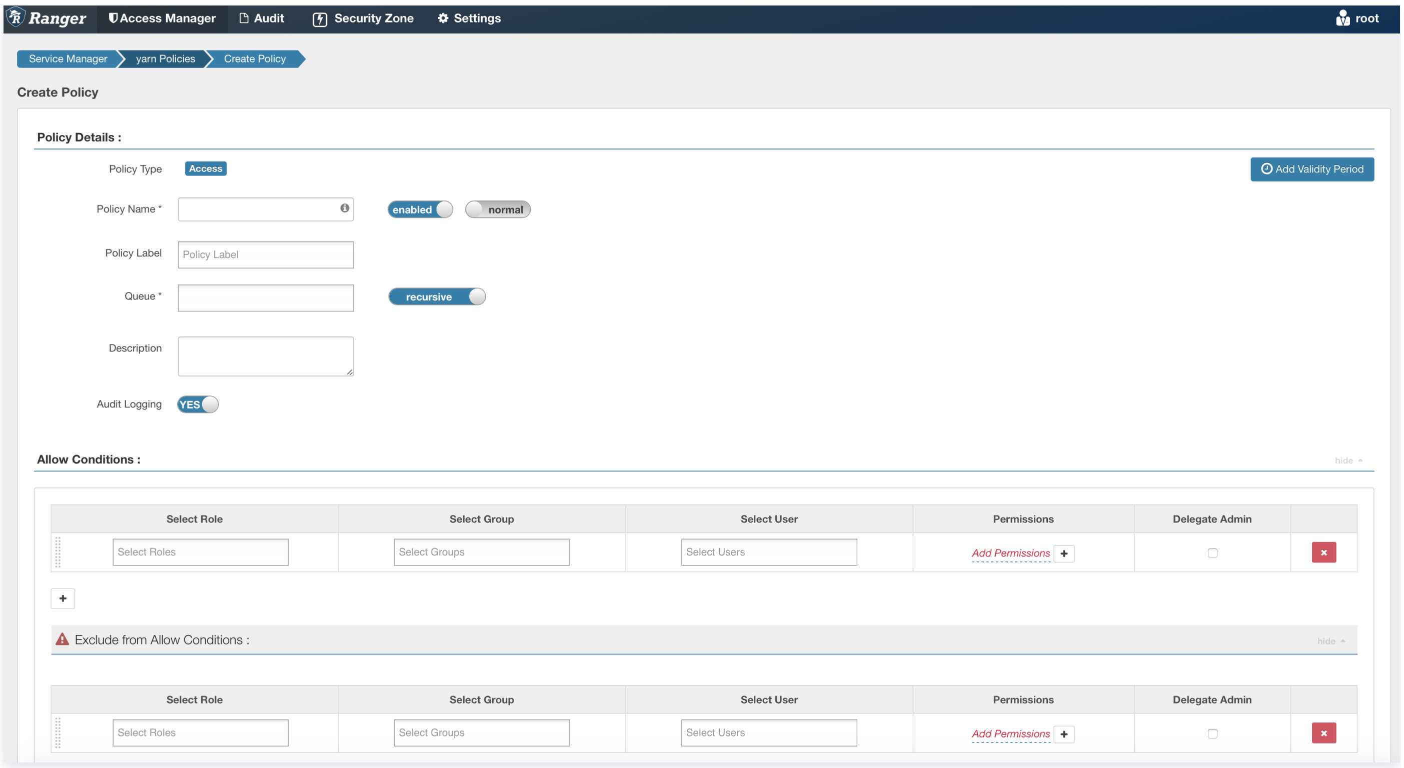


- Configure Policy



List of Policies : yarn

Policy ID	Policy Name	Policy Labels	Status	Audit Logging	Roles	Groups	Users	Action
3	all - queue	--	Enabled	Enabled	--	--	hadoop	



Create Policy

Policy Details :

Policy Type	Access
Policy Name *	
Policy Label	Policy Label
Queue *	
Description	
Audit Logging	YES

Allow Conditions :

Select Role	Select Group	Select User	Permissions	Delegate Admin
Select Roles	Select Groups	Select Users	Add Permissions	

Exclude from Allow Conditions :

Select Role	Select Group	Select User	Permissions	Delegate Admin
Select Roles	Select Groups	Select Users	Add Permissions	

4. Upon adding the Policy, please wait approximately half a minute for the Policy to take effect. Once effective, 'user1' can perform operations such as submitting, deleting, and querying jobs in the YARN's root.default queue.

Note
While configuring the Ranger YARN Service and Policy, it is imperative to ensure that there are no YARN jobs in progress, as this could lead to issues with certain user job submission permissions.

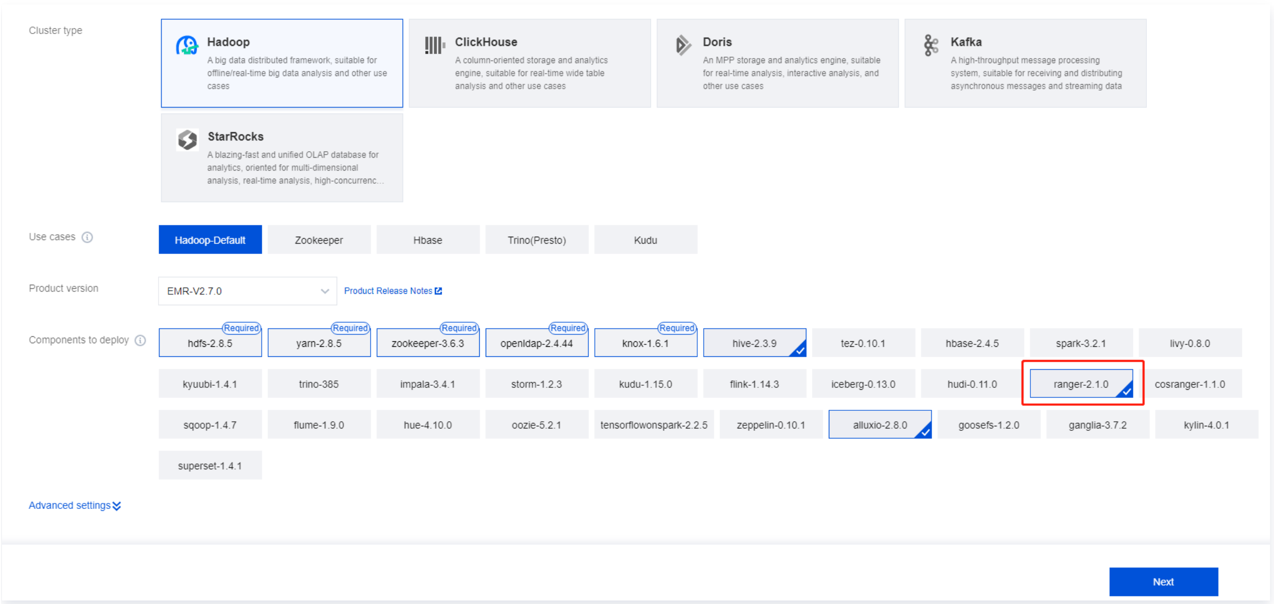
Integrating HBase with Ranger

Last updated: 2024-01-02 14:17:32

Preparations

During the installation of Ranger, both Ranger Admin and Ranger UserSync are deployed on the Master node, while Ranger Plugin is deployed on the node hosting the embedded component's primary daemon process.
During the creation of a cluster, when selecting Hadoop as the cluster type, Ranger can be chosen from the optional components. The version of Ranger may vary depending on the selected version of EMR.

Note
When the cluster type is Hadoop and the optional Ranger component is selected, EMR-Ranger will, by default, create services for HDFS and YARN and establish default policies.



Cluster type: [✓ Hadoop] ClickHouse Doris Kafka StarRocks

Use cases: [✓ Hadoop-Default] Zookeeper Hbase Trino(Presto) Kudu

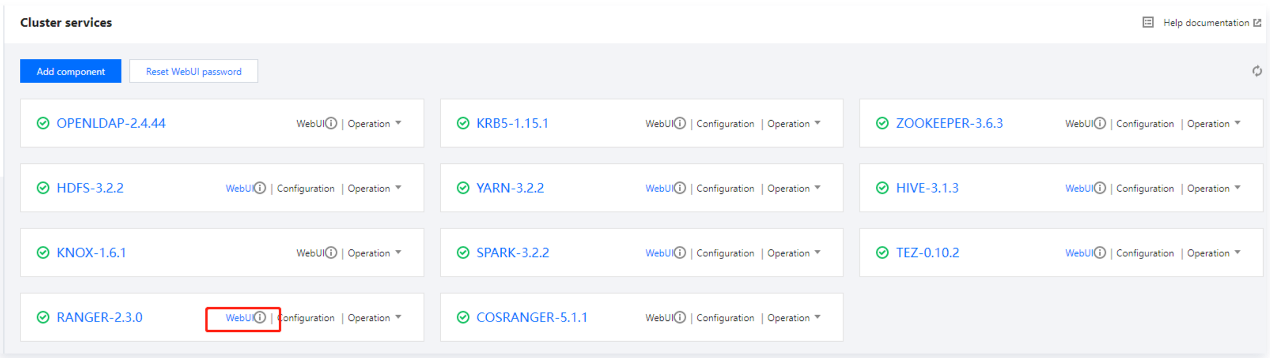
Product version: EMR-V2.7.0

Components to deploy: [✓ hdfs-2.8.5 (Required)] [✓ yarn-2.8.5 (Required)] [✓ zookeeper-3.6.3 (Required)] [✓ openldap-2.4.44 (Required)] [✓ knox-1.6.1 (Required)] [✓ hive-2.3.9] tez-0.10.1 hbase-2.4.5 spark-3.2.1 livy-0.8.0 kyuubi-1.4.1 trino-385 impala-3.4.1 storm-1.2.3 kudu-1.15.0 flink-1.14.3 iceberg-0.13.0 hudi-0.11.0 [✓ ranger-2.1.0] cosranger-1.1.0 sqoop-1.4.7 flume-1.9.0 hue-4.10.0 oozie-5.2.1 tensorflowonspark-2.2.5 zeppelin-0.10.1 [✓ alluxio-2.8.0] goosefs-1.2.0 ganglia-3.7.2 kylin-4.0.1 superset-1.4.1

Hadoop
A big data distributed framework, suitable for offline/real-time big data analysis and other use cases
ClickHouse
A column-oriented storage and analytics engine, suitable for real-time wide table analysis and other use cases
Doris
An MPP storage and analytics engine, suitable for real-time analysis, interactive analysis, and other use cases
Kafka
A high-throughput message processing system, suitable for receiving and distributing asynchronous messages and streaming data
StarRocks
A blazing-fast and unified OLAP database for analytics, oriented for multi-dimensional analysis, real-time analysis, high-concurrenc...
Product Release Notes
Advanced settings
Next

Ranger Web UI

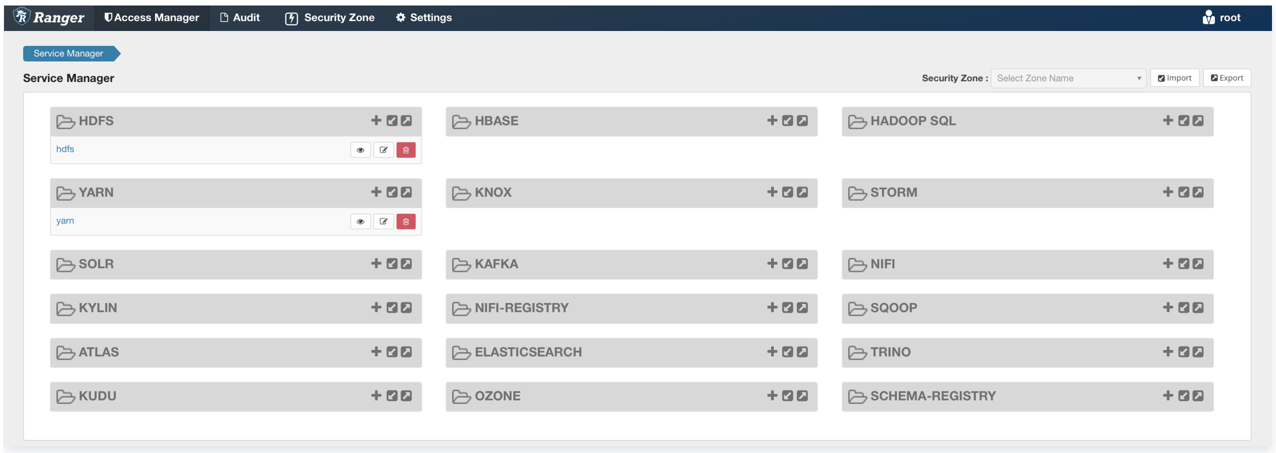
Before accessing the Ranger Web UI, please ensure that the purchased cluster is configured with a public IP. Then, in the cluster service, click on the Web UI address link of the Ranger component.



Cluster services: OPENLDAP-2.4.44, KRB5-1.15.1, ZOOKEEPER-3.6.3, HDFS-3.2.2, YARN-3.2.2, HIVE-3.1.3, KNOX-1.6.1, SPARK-3.2.2, TEZ-0.10.2, RANGER-2.3.0, COSRANGER-5.1.1

After the Web UI address link redirects, you will be prompted to enter a username and password, which are the ones set during the

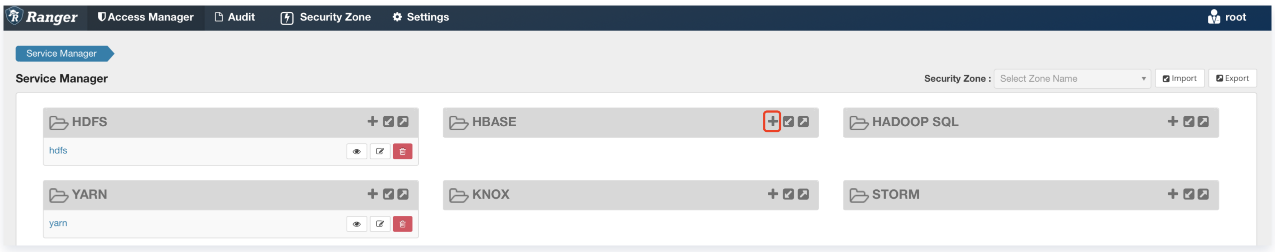
purchase of the cluster.



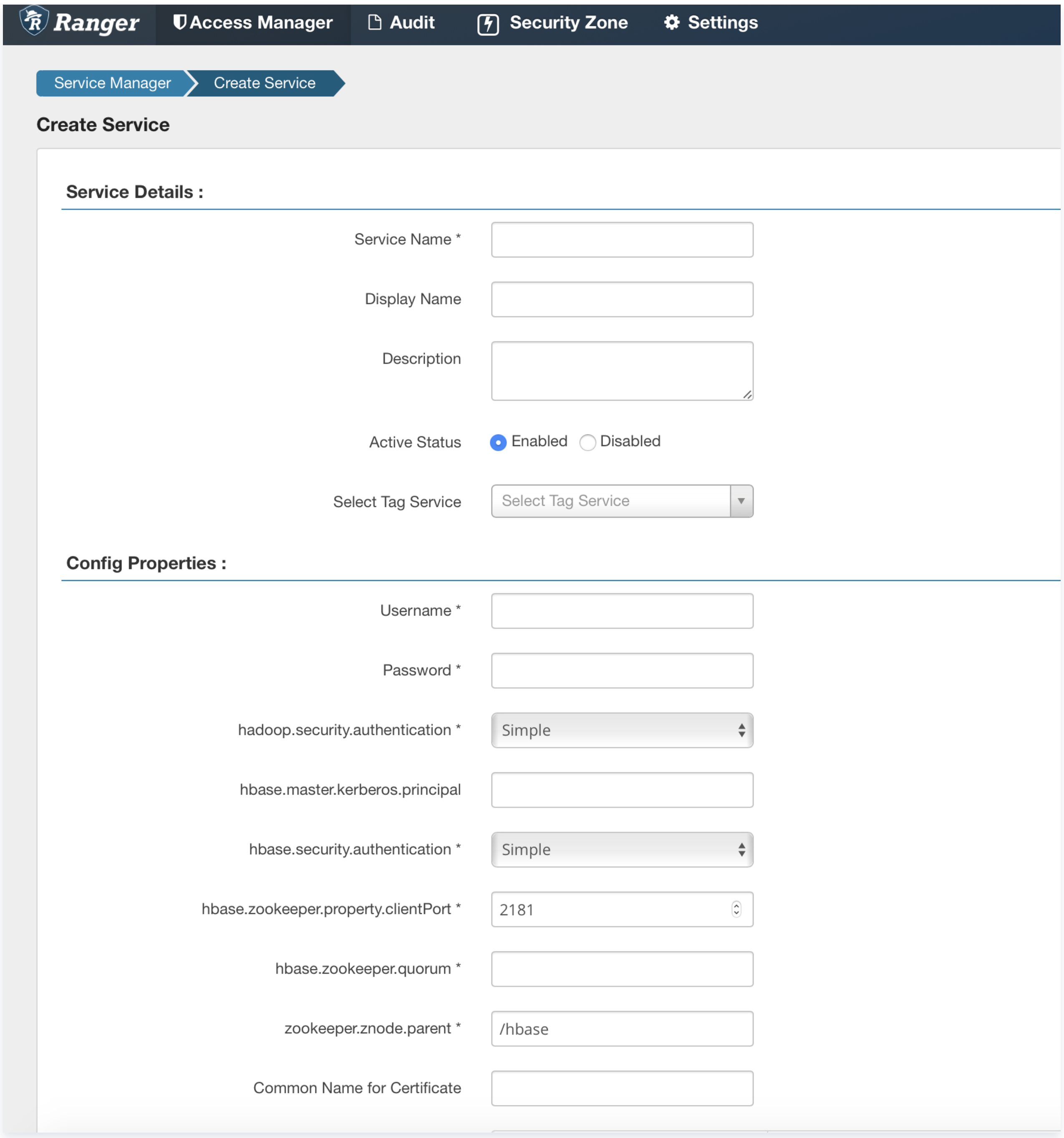
Hbase Integration with Ranger

Note
Please ensure that HBase related services are operating normally and that the current cluster has Ranger installed.

1. Utilize the EMR Ranger Web UI page to add the EMR Ranger Hbase service.



2. Configure the relevant parameters for the EMR Ranger Hbase Service.



Create Service

Service Details :

Service Name *	
Display Name	
Description	
Active Status	[✓ Enabled] Disabled
Select Tag Service	Select Tag Service

Config Properties :

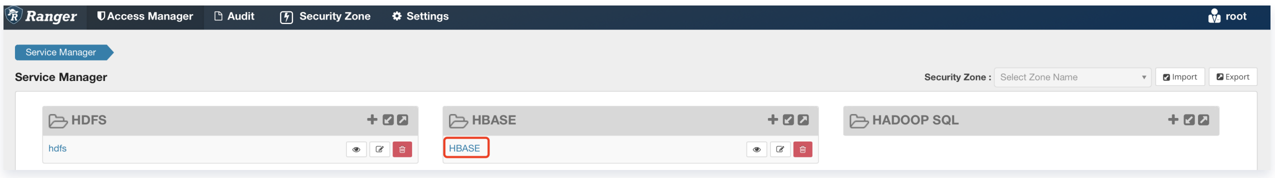
Username *	
Password *	
hadoop.security.authentication *	Simple
hbase.master.kerberos.principal	
hbase.security.authentication *	Simple
hbase.zookeeper.property.clientPort *	2181
hbase.zookeeper.quorum *	
zookeeper.znode.parent *	/hbase
Common Name for Certificate	

Parameter	Mandatory Field	Explanation
Service Name	Supported	Service name, as displayed on the main HBase component of the Ranger Web UI.
Description	Not required	Service Description Information
Active Status	Default Value	Service activation status, enabled by default.
UserName	Supported	Username for resource utilization.
Password	Supported	User Password
Hbase.zookeeper.property.clientPort	Supported	ZK Client Request Port

Hbase.zookeeper.quorum	Supported	ZK Cluster IP Address
Zookeeper.znode.parent	Supported	ZK Node Information

3. EMR Ranger Hbase Resource Permission Configuration.

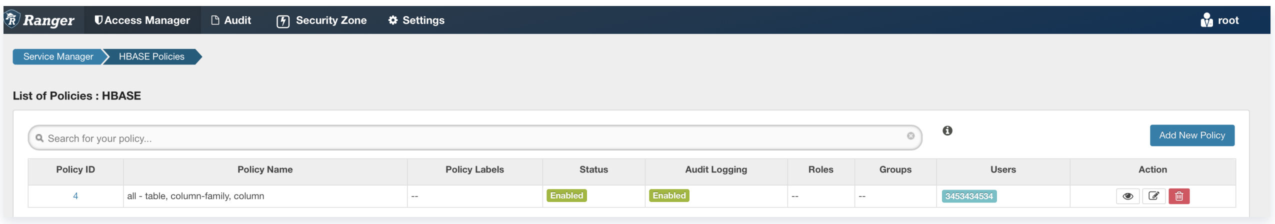
- Click on the Configured EMR Ranger Hbase Service



Service Manager

Security Zone : Select Zone Name

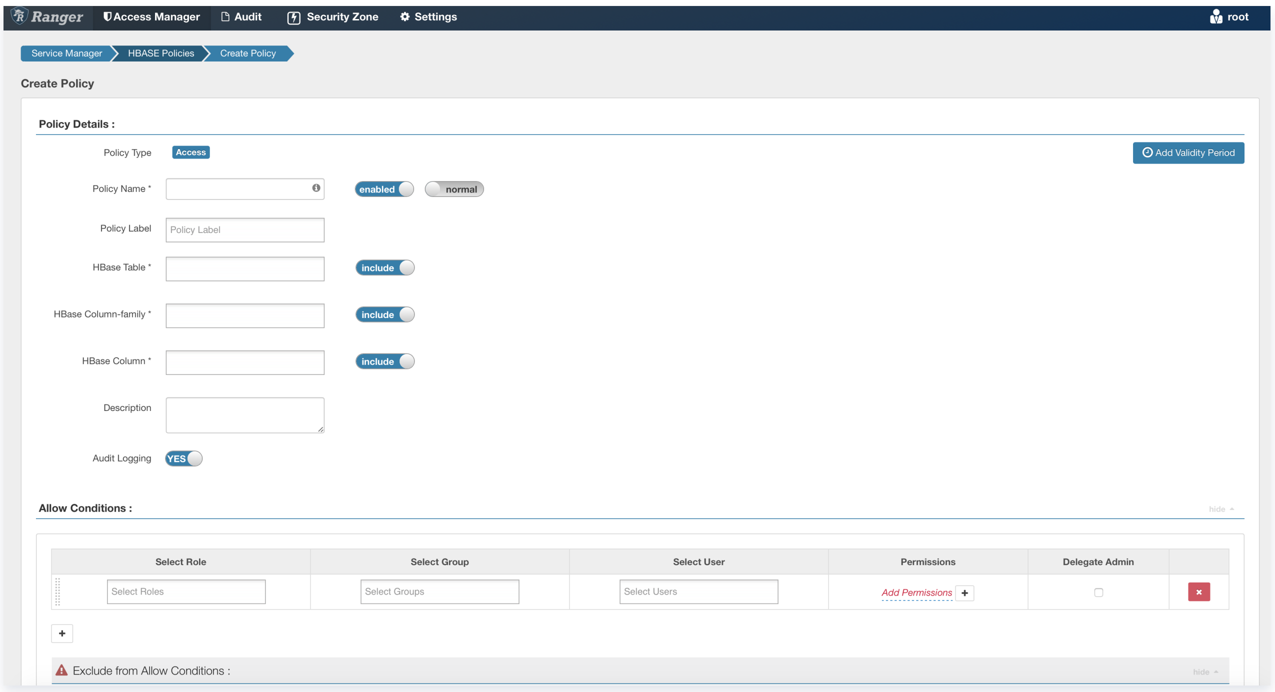
- Configure Policy



List of Policies : HBASE

Policy ID	Policy Name	Policy Labels	Status	Audit Logging	Roles	Groups	Users	Action
4	all - table, column-family, column	--	Enabled	Enabled	--	--	3453434534	

The Users depicted above is Hbase, with a Policy Name of all–table, column–family, and column, signifying that the Hbase user possesses permissions for Region Balance, MemStore Flush, Compaction, and Split. Please ensure that the Service you create has these permissions.



Create Policy

Policy Details :

Policy Type: Access

Policy Name * : enabled, normal

Policy Label: Policy Label

HBase Table * : include

HBase Column-family * : include

HBase Column * : include

Description

Audit Logging: YES

Allow Conditions :

Select Role	Select Group	Select User	Permissions	Delegate Admin
Select Roles	Select Groups	Select Users	Add Permissions	

Exclude from Allow Conditions :

Category	Is it a mandatory option?	Explanation
HBase Table	Supported	HBase Table Name
HBase Column–family	Supported	Column Family in HBase Table
HBase Column	Supported	Qualifier under the Column Family in HBase Table

4. Upon adding the Policy, please wait approximately half a minute for it to take effect. Once effective, 'user1' will be able to perform relevant operations on the related column family and qualifiers of the HBase 'order' table.

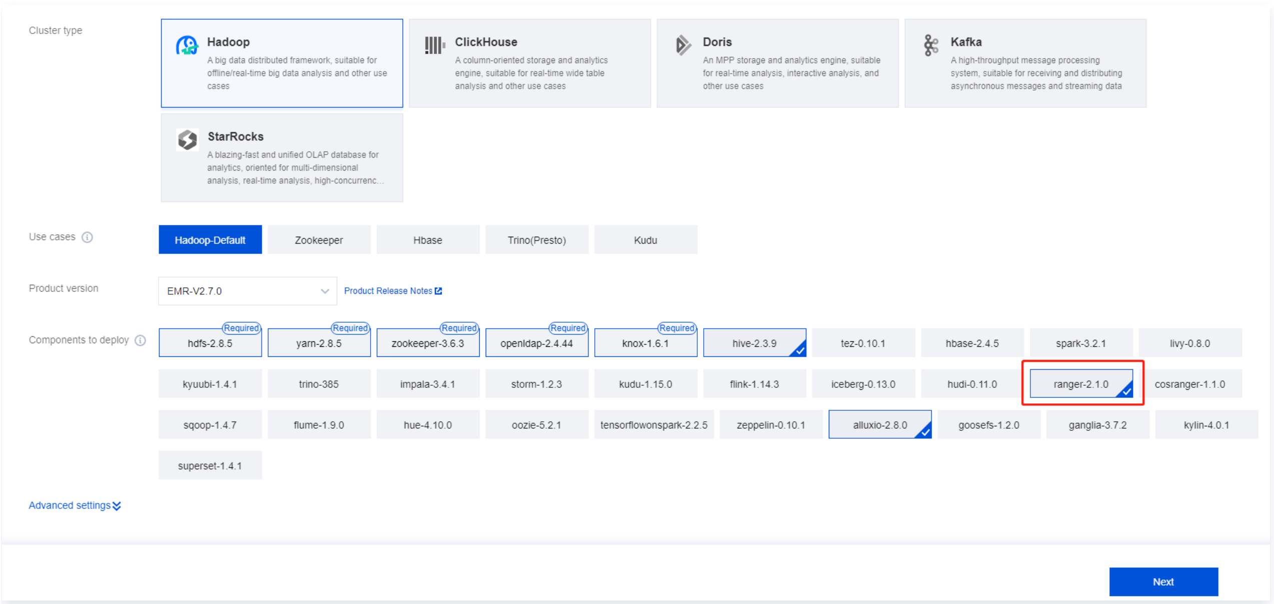
Integrating Presto with Ranger

Last updated：2024-01-02 14:17:44

Preparations

During the installation of Ranger, both Ranger Admin and Ranger UserSync are deployed on the Master node, while Ranger Plugin is deployed on the node hosting the embedded component's primary daemon process.
During the creation of a cluster, when selecting Hadoop as the cluster type, Ranger can be chosen from the optional components. The version of Ranger may vary depending on the selected version of EMR.

Note
When the cluster type is Hadoop and the optional Ranger component is selected, EMR-Ranger will, by default, create services for HDFS and YARN and establish default policies.



Cluster type: Hadoop — A big data distributed framework, suitable for offline/real-time big data analysis and other use cases | ClickHouse — A column-oriented storage and analytics engine, suitable for real-time wide table analysis and other use cases | Doris — An MPP storage and analytics engine, suitable for real-time analysis, interactive analysis, and other use cases | Kafka — A high-throughput message processing system, suitable for receiving and distributing asynchronous messages and streaming data | StarRocks — A blazing-fast and unified OLAP database for analytics, oriented for multi-dimensional analysis, real-time analysis, high-concurrenc...

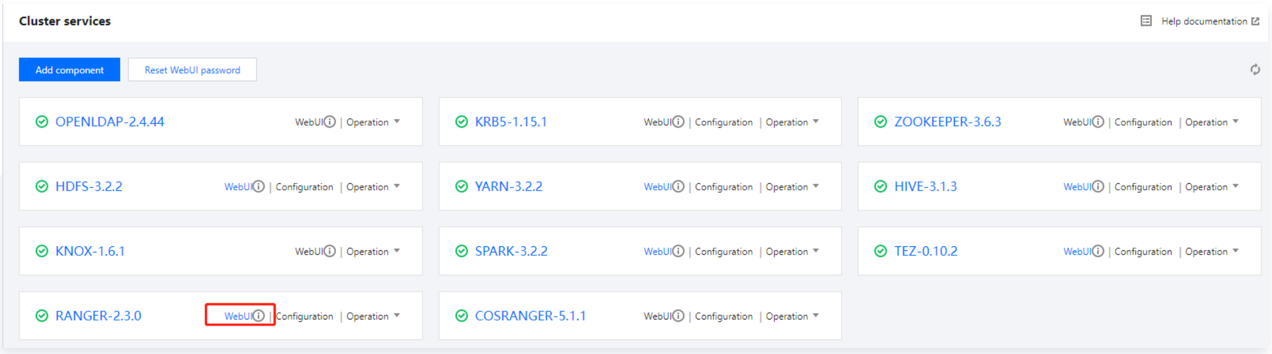
Use cases: Hadoop-Default | Zookeeper | Hbase | Trino(Presto) | Kudu

Product version: EMR-V2.7.0 Product Release Notes

Components to deploy: hdfs-2.8.5 (Required), yarn-2.8.5 (Required), zookeeper-3.6.3 (Required), openldap-2.4.44 (Required), knox-1.6.1 (Required), hive-2.3.9 ✓, tez-0.10.1, hbase-2.4.5, spark-3.2.1, livy-0.8.0, kyuubi-1.4.1, trino-385, impala-3.4.1, storm-1.2.3, kudu-1.15.0, flink-1.14.3, iceberg-0.13.0, hudi-0.11.0, ranger-2.1.0 ✓, cosranger-1.1.0, sqoop-1.4.7, flume-1.9.0, hue-4.10.0, oozie-5.2.1, tensorflowonspark-2.2.5, zeppelin-0.10.1, alluxio-2.8.0 ✓, goosefs-1.2.0, ganglia-3.7.2, kylin-4.0.1, superset-1.4.1

Ranger Web UI

Before accessing the Ranger Web UI, please ensure that the purchased cluster is configured with a public IP. Then, in the cluster service, click on the Web UI address link of the Ranger component.

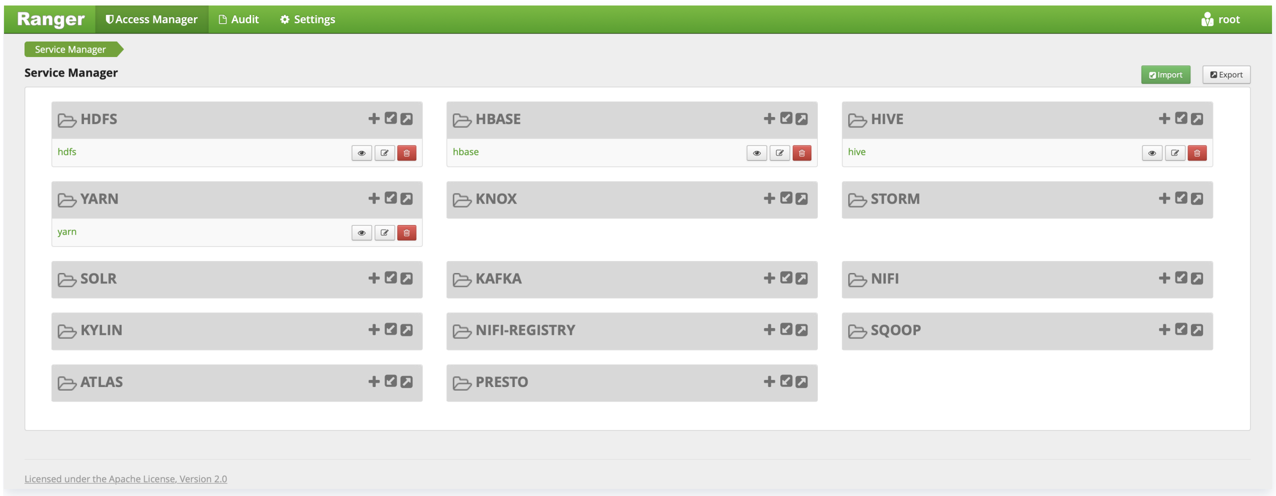


Cluster services

OPENLDAP-2.4.44, KRB5-1.15.1, ZOOKEEPER-3.6.3, HDFS-3.2.2, YARN-3.2.2, HIVE-3.1.3, KNOX-1.6.1, SPARK-3.2.2, TEZ-0.10.2, RANGER-2.3.0, COSRANGER-5.1.1

After the Web UI address link redirects, you will be prompted to enter a username and password, which are the ones set during the

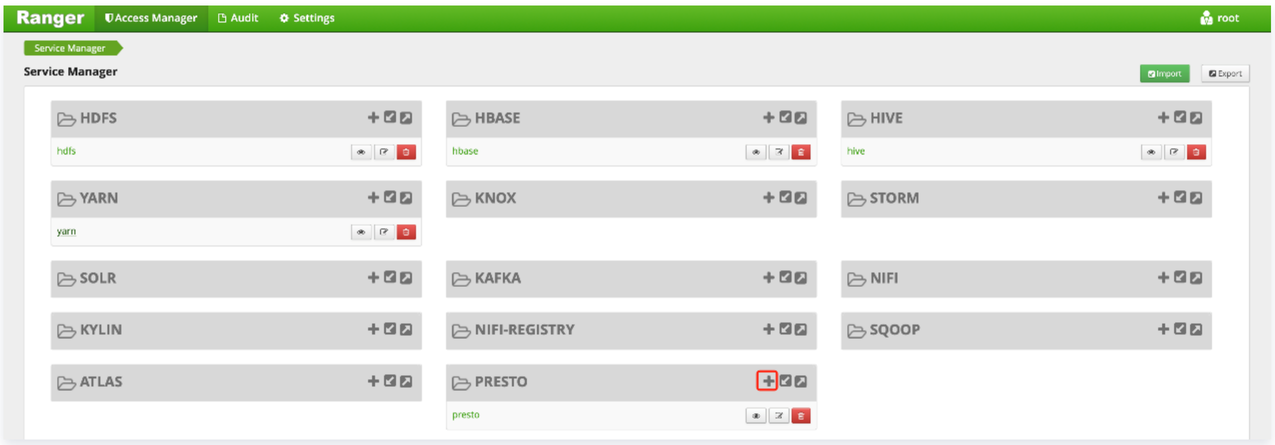
purchase of the cluster.



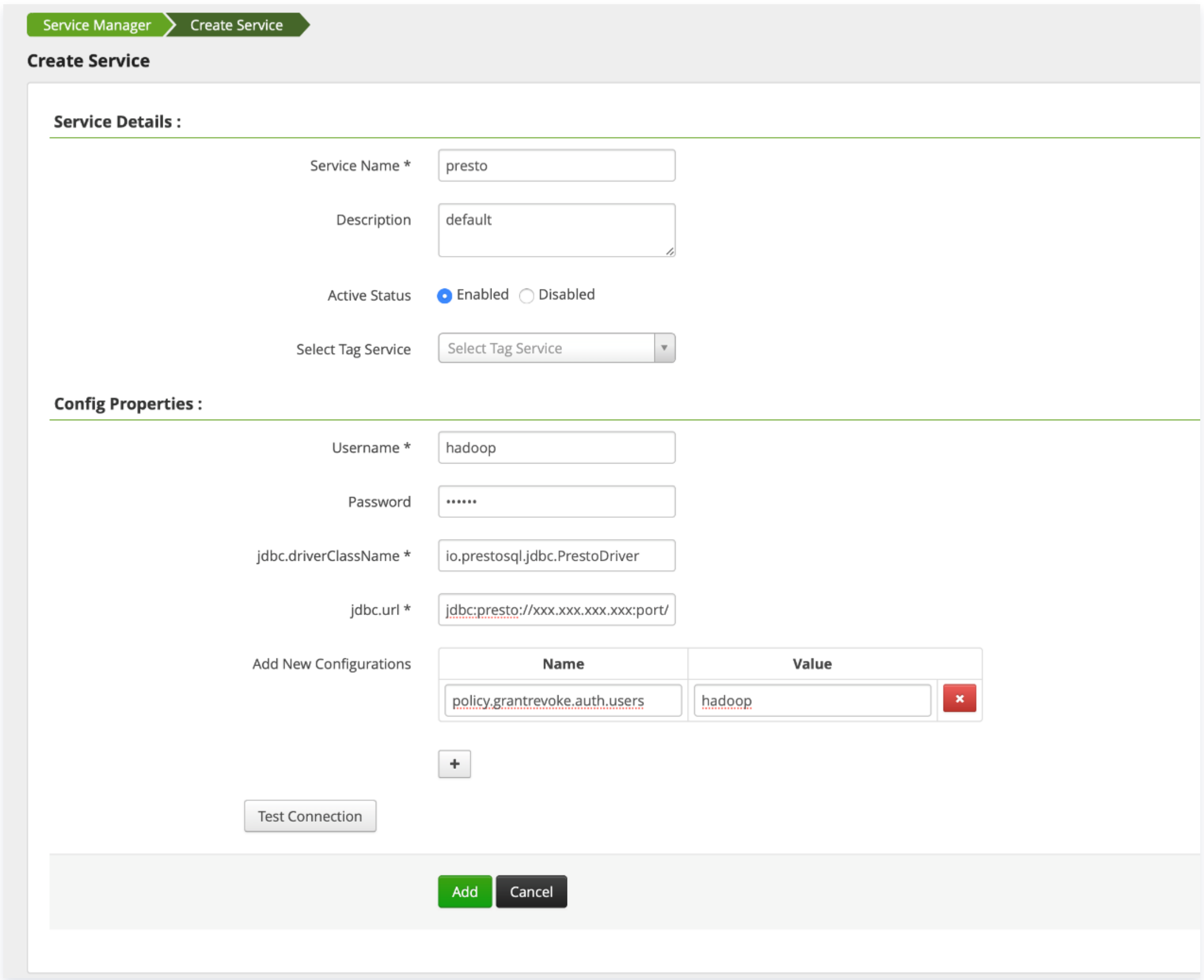
Presto Integration with Ranger

Note
Please ensure that the Presto-related services are running smoothly and that the current cluster has Ranger installed.

1. Utilize the EMR Ranger Web UI page to add the EMR Ranger Presto service.



2. Configure the relevant parameters for the EMR Ranger Presto Service.



Create Service

Service Details :

Service Name *	presto
Description	default
Active Status	[✓ Enabled] Disabled
Select Tag Service	Select Tag Service

Config Properties :

Username *	hadoop
Password	••••••
jdbc.driverClassName *	io.prestosql.jdbc.PrestoDriver
jdbc.url *	jdbc:presto://xxx.xxx.xxx.xxx:port/

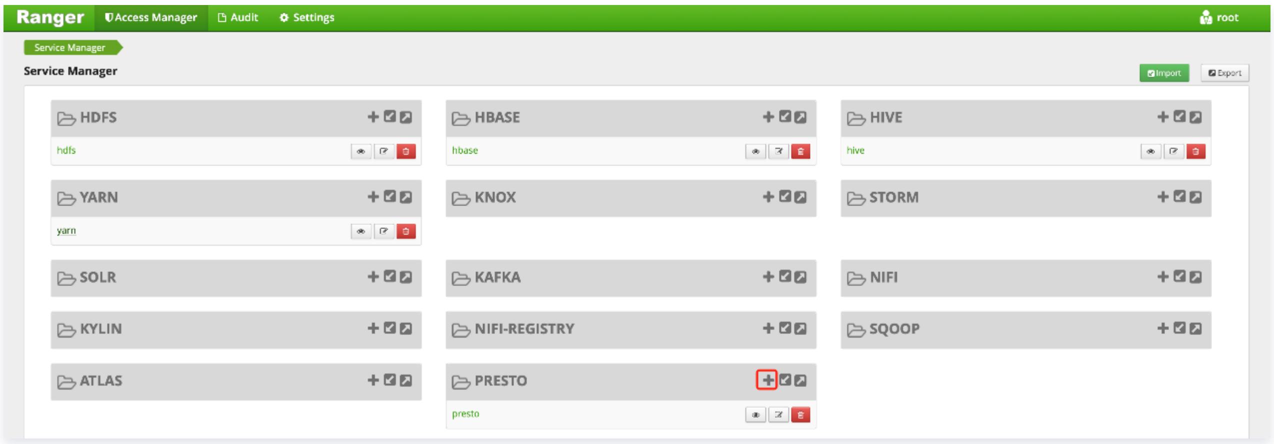
Add New Configurations

Name	Value
policy.grantrevoke.auth.users	hadoop

Category	Mandatory Field	Explanation
Service Name	Supported	The service name, as displayed on the Ranger Web UI's main Presto component.
Description	Not required	Service Description Information
Active Status	Default Value	Service activation status, enabled by default.
UserName	Supported	Username for resource utilization.
Password	Supported	User Password
JDBC.driverClassName	Supported	Complete Path of the Driver Class Name
jdbc.url	Supported	Address in the form of a Presto JDBC connection, for instance, jdbc:presto://ip/hostname:port.

3. EMR Ranger Presto Resource Permission Configuration

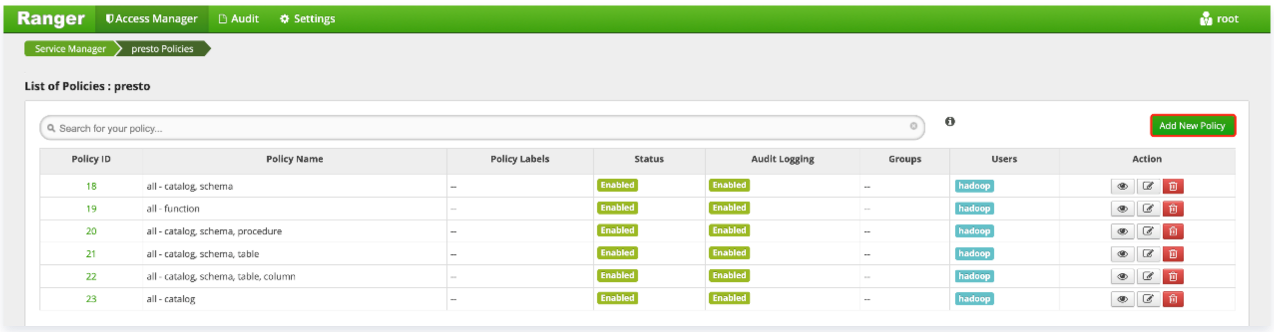
- Click on the configured EMR Ranger Presto Service.



Service Manager

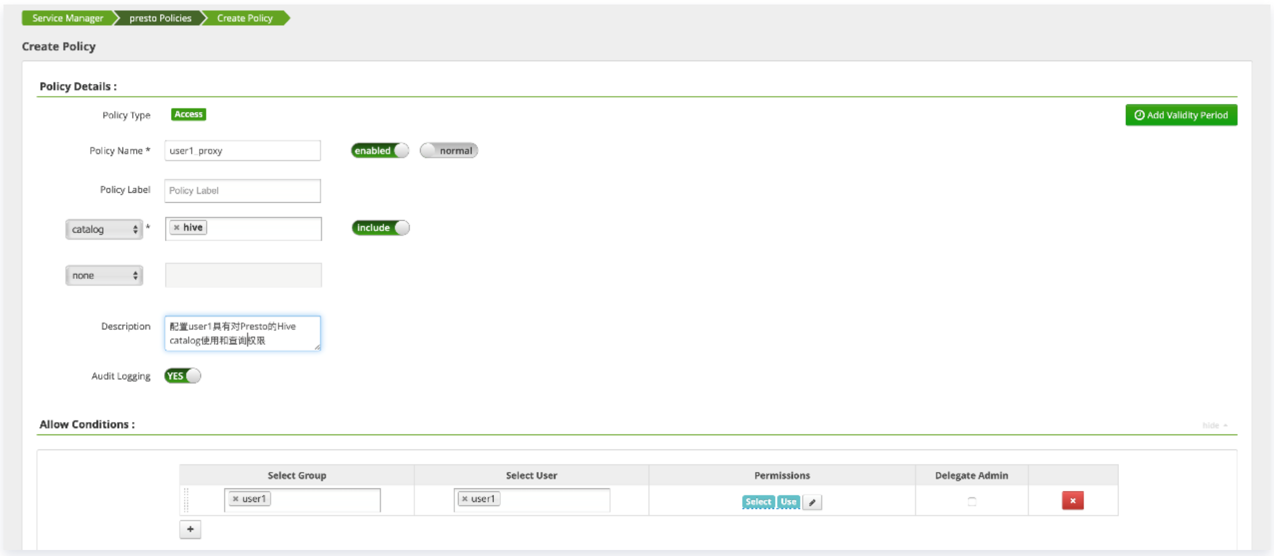
HDFS: hdfs	HBASE: hbase	HIVE: hive
YARN: yarn	KNOX	STORM
SOLR	KAFKA	NIFI
KYLIN	NIFI-REGISTRY	SQOOP
ATLAS	PRESTO: presto	

- Configure Policy



List of Policies : presto

Policy ID	Policy Name	Policy Labels	Status	Audit Logging	Groups	Users	Action
18	all - catalog, schema	--	Enabled	Enabled	--	hadoop	
19	all - function	--	Enabled	Enabled	--	hadoop	
20	all - catalog, schema, procedure	--	Enabled	Enabled	--	hadoop	
21	all - catalog, schema, table	--	Enabled	Enabled	--	hadoop	
22	all - catalog, schema, table, column	--	Enabled	Enabled	--	hadoop	
23	all - catalog	--	Enabled	Enabled	--	hadoop	



Create Policy

Policy Details :

Policy Type	Access	
Policy Name *	user1_proxy	enabled normal
Policy Label	Policy Label	
catalog *	hive	include
none		
Description	配置user1具有对Presto的Hive catalog使用和查询权限	
Audit Logging	YES	

Allow Conditions :

Select Group	Select User	Permissions	Delegate Admin
user1	user1	Select Use	

4. Upon adding the Policy, please wait approximately half a minute for the Policy to take effect. Once effective, user1 will be able to view and utilize the Presto catalog.

Ranger Audit Log Guide
Storing Ranger audit logs in Solr

Last updated：2024-01-02 14:18:08

By default, the audit log function of Ranger is deactivated. However, it can be activated as per necessity by modifying the configuration file to enable the corresponding component's auditing capabilities.

Note:
When installing Ranger on EMR-2.7.0 and later versions, by default, a standalone version of Solr service is installed on a single Master node. This service is utilized for storing Ranger's audit logs, which are retained for a period of seven days.

1. For instance, to enable this feature for HDFS, navigate to EMR Console **> Cluster Services > HDFS > Configuration Management**, and modify the configuration items in the ranger-hdfs-audit.xml file as follows:

```
xasecure.audit.is.enabled=true
xasecure.audit.solr.is.enabled=true
```

2. Navigate to **Cluster Services > HDFS > Role Management** and restart the NameNode role.
3. Should clients require long-term retention of Ranger audit logs, it is recommended to store the Ranger audit logs in Tencent Cloud's ElasticSearch.

Kafka Development Guide
Kafka Overview

Last updated：2024-01-02 14:18:28

Tencent Cloud's Elastic MapReduce (EMR) - Kafka offers a cloud-hosted service for the open-source Kafka, providing convenient features such as Kafka cluster deployment, configuration modification, and monitoring alerts. It delivers a secure and stable OLAP solution for businesses and users. Kafka data pipelines are the most commonly used data sources (Sources) and data destinations (Sinks) in stream computing systems. Users can import stream data into a specific Kafka Topic, process it through Flink operators, and output it to another Topic in the same or different Kafka instance. Kafka supports multi-partition reading and writing for the same Topic, allowing data to be read from multiple partitions and written to multiple partitions. This enhances throughput, reduces data skew, and alleviates hotspots.

Architecture

Supports both single-node and multi-node architectures. Flexibly choose according to business requirements.

OPS

The console provides ready-to-use services such as monitoring, log retrieval, and parameter adjustment.

Feature

- Decoupling of sending and receiving: Effectively decouples the relationship between producers and consumers. While ensuring the same interface constraints, it allows for independent expansion or modification of the processing process between producers/consumers.
- Peak shaving and valley filling: The Kafka cluster can withstand sudden increases in access pressure, and will not completely collapse due to sudden overload requests, effectively enhancing system robustness.
- Sequential read and write: The Kafka cluster can ensure the orderliness of messages within a Partition. Consistent with most message queues, the Kafka cluster can ensure data is processed in order, greatly enhancing disk efficiency.
- Asynchronous communication: In scenarios where the business does not need to process messages immediately, the Kafka message queue cluster provides an asynchronous message processing mechanism. When the access volume is high, messages are simply placed in the queue, and are processed after the access volume decreases, alleviating system pressure.

Advantages

Full compatibility with Apache Kafka and easy migration

- The Kafka cluster is compatible with the open-source Kafka version 1.1.1.
- The Kafka cluster business system is based on the existing open-source Apache Kafka ecosystem code. Without any modifications, it can be migrated to the cloud to enjoy the high-performance Kafka message queue service provided by Tencent Cloud.

High Performance

- Tencent Cloud's Professional team further optimizes service performance, eliminating complex parameter configurations, and providing superior performance.
- Graphical interface for scaling capabilities, bolstered by high-performance IaaS layer support.

High availability

- Leveraging years of technical accumulation from Tencent's engineering monitoring platform, the cluster is monitored from all angles, with a professional operations team available 24/7 to handle alerts and ensure the high availability of the Kafka cluster service.
- Supports custom multi-availability zone deployment within the same region, enhancing disaster recovery capabilities.

High reliability

- CKafka uses highly reliable disks and can keep its services running even if 50% of the disks are unavailable.

- 2 replicas are created by default, and up to 3 replicas can be used. The more replicas, the higher the reliability.

Scenarios

Last updated：2024-01-02 14:18:38

Web Behavior Analysis

The Kafka cluster processes website activities (page views, searches, and other user activities) in real time and publishes them to the Topic based on their type. This stream of information can be utilized for real-time monitoring or offline statistical analysis. Given that each user's page view generates a plethora of activity data, tracking website activities necessitates a high throughput. The Kafka cluster is perfectly equipped to meet the demands for high throughput and offline processing.

Log aggregation

The Kafka cluster's low-latency processing feature facilitates the support of multiple data sources and distributed data processing (consumption). Compared to centralized log aggregation systems, the Kafka message queue can provide stronger persistence guarantees and lower end-to-end latency while maintaining equivalent performance.
The characteristics of the Kafka cluster make it highly suitable as a log collection hub. Multiple hosts/applications can asynchronously send operation logs in bulk to the Kafka cluster, eliminating the need for local or DB storage. The Kafka cluster can batch submit/compress messages, resulting in virtually imperceptible performance overhead for the producer. At this point, consumers can use systematic storage and analysis systems like Hadoop to statistically analyze the pulled logs.

Online/Offline Analysis

In certain big data-related business scenarios, there is a need to process and aggregate a large volume of concurrent data, which places high demands on the processing performance and scalability of the cluster. The data distribution mechanism implemented by the Kafka cluster, including disk storage space allocation, message format processing, server selection, and data compression, makes it suitable for handling massive real-time messages and aggregating data from distributed applications, thereby facilitating system operations and maintenance.
In specific big data scenarios, the Kafka cluster can effectively support the processing of offline and streaming data, and conveniently carry out operations such as data aggregation and analysis.

Kafka Usage

Last updated：2024-01-02 14:18:47

Generate Data

Java Code Method

```
@Component
@Slf4j
public class KafkaProducer {

    @Autowired
    private KafkaTemplate<String, Object> kafkaTemplate;

    // Define Custom Topic
    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("Preparing to send message: {}", obj2String);
        // Send Message
        ListenableFuture<SendResult<String, Object>> future = kafkaTemplate.send(TOPIC_TEST, obj);
        future.addCallback(new ListenableFutureCallback<SendResult<String, Object>>() {
            @Override
            public void onFailure(Throwable throwable) {
                // Handling Failed Sends
                log.info(TOPIC_TEST + " - Producer encountered an error while sending message: " + throwable.getMessage());
            }

            @Override
            public void onSuccess(SendResult<String, Object> stringObjectSendResult) {
                // Successful Handling
                log.info(TOPIC_TEST + " - Producer successfully sent message: " + stringObjectSendResult.toString());
            }
        });
    }
}
```

Command Method

```
bin/kafka-console-producer.sh --broker-list node86:9092 --topic t_cdr
```

Consuming Data

Java Code Method

```
@Component
@Slf4j
public class KafkaConsumer {

    @KafkaListener(topics = KafkaProducer.TOPIC_TEST, groupId = KafkaProducer.TOPIC_GROUP1)
    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 has consumed: Topic: " + topic + ", Message: " + msg);
            ack.acknowledge();
        }
    }

    @KafkaListener(topics = KafkaProducer.TOPIC_TEST, groupId = KafkaProducer.TOPIC_GROUP2)
    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 has consumed: Topic: " + topic + ", Message: " + msg);
            ack.acknowledge();
        }
    }
}
```

Command Method

```
bin/kafka-console-consumer.sh --zookeeper node01:2181 --topic t_cdr --from-beginning
```

Adding a new topic (Command Method)

```
bin/kafka-topics.sh --zookeeper node01:2181 --create --topic t_cdr --partitions 30  --replication-factor 2
```

For detailed usage, please refer to the official Kafka documentation.

Iceberg Development Guide

Last updated：2024-01-02 14:19:02

Introduction to Iceberg

Apache Iceberg is a novel open-source table format designed for large-scale data analytics. It is engineered to store massive, slowly changing tabular data, aiming to enhance the de facto standard table layouts inherent in Hive, Trino (PrestoSQL), and Spark. Iceberg can obscure differences in underlying data storage formats, offering a unified operational API, thereby enabling various engines to connect through its provided API.
Apache Iceberg possesses the following capabilities:

- Schema Evolution: Supports the addition, deletion, updating, renaming, and reordering of table format definitions.
- Partition Layout Evolution: Allows for the updating of table layouts in response to changes in data volume or query patterns.
- Hidden Partitioning: Queries are no longer dependent on the physical layout of the table. Through the separation of physical and logical aspects, Iceberg tables can evolve partitioning schemes with changes in data volume and over time. Misconfigured tables can be rectified without the need for costly migrations.
- Time Travel: Facilitates users in executing repeated queries using identical snapshots, or effortlessly inspecting modifications.
- Version Rollback: Empowers users to swiftly rectify issues by resetting the table to a stable state.

In terms of reliability and performance, Iceberg can be applied to data tables of tens of PB in production, and these large-scale tables can be read even without a distributed SQL engine:

- Fast Scanning: Enables reading of tables or locating files without the necessity of a distributed SQL engine.
- Advanced Filtering: Utilizes partition and column-level statistical information for data file pruning, based on table metadata.

Iceberg is designed to address the issues of correctness in eventually consistent cloud object storage:

- Compatible with any cloud storage, it mitigates HDFS NameNode congestion by circumventing list and rename calls.
- Serializable Isolation: Table modifications are atomic, ensuring users never encounter partial or uncommitted changes.
- Concurrent Writes: Employs an optimistic locking mechanism for concurrency control, ensuring successful compatible updates through retries, even in the event of write conflicts.

Iceberg is designed to manage historical versions of table data in the form of snapshots. A snapshot represents the state of a table at a specific moment in time, listing all data files of the table at that moment. Data files are stored in various Manifest files, which are in turn stored in a Manifest List file. Manifest files can be shared among different Manifest List files, with each Manifest List file representing a snapshot.

- Manifest List Files: These are metadata files that store a list of Manifest files, with each Manifest file occupying a single line.
- Manifest Files: These are metadata files that list the data files comprising a specific snapshot. Each line provides a detailed description of each data file, including the file's status, path, partition information, column-level statistics (such as the maximum and minimum values for each column, number of null values, etc.), file size, and the number of data rows within the file.
- Data Files: These are the files where Iceberg tables store actual data, typically located under the data directory of the table's data storage directory.

Use Case

For more examples, please refer to the official Iceberg examples.
This document uses Iceberg version 0.11.0 in EMR-V3.3.0 as an example. The names of related jar packages may vary with different EMR versions, so please use the actual name found in the path.

1. Log in to the master node and switch to the hadoop user.
2. Iceberg-related packages are located under `/usr/local/service/iceberg/`.
3. Utilize the computation 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
```

- Insertion and Querying of 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
  - Utilizing 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. This example uses the Flink standalone mode and 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：2024-01-02 14:19:20

What is StarRocks?

- StarRocks is a next-generation, high-speed, multi-purpose, massively parallel processing (MPP) database.
- StarRocks is a comprehensive amalgamation of the superior research outcomes from relational OLAP databases and distributed storage systems in the era of big data. Based on industry practices, it has been further refined and optimized, its architecture upgraded, and numerous innovative features have been added, culminating in a brand-new enterprise-level product.
- StarRocks is dedicated to creating a high-speed unified analytical experience, catering to a variety of data analysis scenarios for enterprise users. It supports multiple data models (detail model, aggregate model, update model), various import methods (batch and real-time), and can import data with up to 10,000 columns. Furthermore, it can integrate and connect with a multitude of existing systems such as Spark, Flink, Hive, and ElasticSearch.
- StarRocks is compatible with the MySQL protocol, allowing for data analysis through interfacing with MySQL clients and commonly used Business Intelligence (BI) tools.
- StarRocks employs a distributed architecture, horizontally partitioning data tables and storing them in multiple replicas. The cluster size can be flexibly scaled, supporting data analysis at the 10PB level. It supports the MPP framework for parallel accelerated computing and multiple replicas, demonstrating resilient fault tolerance.
- StarRocks employs a relational model, utilizing strict data types and a columnar storage engine. Through encoding and compression techniques, it minimizes read-write amplification. By using a vectorized execution method, it fully exploits the parallel computing capabilities of multi-core CPUs, thereby significantly enhancing query performance.

StarRocks Features

The architectural design of StarRocks incorporates the concepts of MPP databases and distributed systems, exhibiting the following features:

Streamlined Architecture

StarRocks internally executes SQL tasks through the MPP computing framework. The MPP framework itself fully leverages the computing capabilities of multiple nodes, executing the entire query in parallel, thereby delivering an excellent interactive analysis experience. The StarRocks cluster does not rely on any other components, making it easy to deploy and maintain. The minimalist architectural design reduces the complexity and maintenance cost of the StarRocks system, while also enhancing its reliability and scalability. Administrators only need to focus on the StarRocks system, without the need to learn and manage any other external systems.

Comprehensive Vectorized Engine

The computational layer of StarRocks fully adopts vectorization technology, systematically optimizing all operators, functions, scan filters, and import/export modules. By employing columnar memory layout and adapting to the CPU's SIMD instruction set, it fully exploits the parallel computing capabilities of modern CPUs, thereby achieving sub-second multi-dimensional analysis capabilities.

Intelligent Query Optimization

StarRocks, through its Cost Based Optimizer (CBO), can automatically optimize complex queries. Without manual intervention, it can reasonably estimate execution costs through statistical information, generate superior execution plans, and significantly enhance the data analysis efficiency in Adhoc and ETL scenarios.

Federated Query

StarRocks supports federated queries using external tables, currently accommodating Hive, MySQL, and Elasticsearch types of external tables. Users can directly accelerate data queries without the need for data import.

Efficient Updates

StarRocks supports various data models, among which the update model allows for upsert/delete operations based on the primary key. Through storage and index optimization, it can achieve efficient query optimization while concurrently updating, thereby better serving real-time data warehouse scenarios.

Intelligent Materialized View

StarRocks supports intelligent materialized views. Users can create materialized views to pre-calculate and generate pre-aggregated tables for accelerating aggregate query requests. StarRocks' materialized views can automatically complete aggregation during data import, maintaining consistency with the original table data. Moreover, during a query, users do not need to specify the materialized view, as StarRocks can automatically select the optimal materialized view to satisfy the query request.

Standard SQL

StarRocks supports standard SQL syntax, including features such as aggregation, JOIN, sorting, window functions, and custom functions. StarRocks can fully support the 22 SQLs of TPC-H and the 99 SQLs of TPC-DS. In addition, StarRocks is also compatible with MySQL protocol syntax, allowing the use of various existing client tools and BI software to access StarRocks and perform drag-and-drop analysis on the data within StarRocks.

Stream-Batch Integration

StarRocks supports both real-time and batch data import methods, with supported data sources including Kafka, HDFS, and local files. It supports data formats such as ORC, Parquet, and CSV, and can import data with up to 10,000 columns. StarRocks can consume Kafka data in real-time to complete data import, ensuring data is neither lost nor duplicated (exactly once). StarRocks can also import data in batches from local or remote sources (HDFS).

Highly Available and Easily Scalable

StarRocks stores both metadata and data in multiple copies, and the services in the cluster have hot backups and are deployed in multiple instances, avoiding single point failures. The cluster has self-healing capabilities and can recover elastically. Failures, offline incidents, or anomalies of nodes will not affect the overall stability of the StarRocks cluster service. StarRocks adopts a distributed architecture, allowing nearly linear horizontal scaling of storage capacity and computing power. The node scale of a single StarRocks cluster can be expanded to hundreds of nodes, and the data scale can reach the level of 10PB. During the scaling process, there is no need to stop the service, and it can continue to provide query services. Additionally, StarRocks supports hot changes to table schemas, allowing the dynamic modification of table definitions through a simple SQL command, such as adding columns, reducing columns, creating new materialized views, etc. At the same time, tables undergoing schema changes can also normally import and query data.

What scenarios is StarRocks suitable for?

StarRocks can meet a variety of analytical needs for enterprise-level users, including OLAP multidimensional analysis, custom reports, real-time data analysis, and Ad-hoc data analysis. Specific business scenarios include:

- OLAP multidimensional analysis for user behavior analysis.
  - User profiling, tag analysis, and audience segmentation.
  - High-dimensional business indicator reports.
  - Self-service report platform.
  - Business issue exploration and analysis.
  - Cross-thematic business analysis.
  - Financial statements.
  - System monitoring and analysis.
- Real-time data analysis for e-commerce promotions.
  - Quality analysis of live streaming in the education industry.
  - Analysis of consignment notes in the logistics industry.
  - Performance analysis and metric calculation in the finance industry.
  - Analysis of advertising placements.
  - Management cockpit.
  - Probe analysis for Application Performance Management (APM).
- High-concurrency query analysis of advertiser reports.

    - Channel personnel analysis in the retail industry.
    - User–oriented analysis reports in the SaaS industry.
    - Multi–page analysis on the dashboard.
- Unified analysis reduces system complexity and the cost of multi–technology stack development and maintenance by utilizing a single system to address scenarios such as multi–dimensional analysis, high–concurrency queries, precomputation, real–time analysis, and Adhoc queries.

User Guide

Last updated：2024-01-02 14:19:31

At present, StarRocks supports a variety of connection methods. The following is a brief introduction on utilizing the MySQL client to establish a connection with StarRocks for development purposes.

Root User Login

Utilize the MySQL client to connect to a specific FE instance's query_port (9030). StarRocks comes with a built-in root user, the password for which is identical to the cluster password by default:

```
mysql -h fe_host -P9030 -u root -p
```

Environment Cleanup:

```
mysql > drop database if exists example_db;

mysql > drop user test;
```

Inspecting Deployed Nodes

1. Inspecting FE Nodes.

```
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)
```

A role designated as FOLLOWER indicates that this is a FE capable of participating in leader selection; IsMaster set to true signifies that the current FE is the master node.

2. Inspecting BE Nodes.

```
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 isAlive is set to true, it indicates that the BE has successfully integrated into the cluster. If the BE has not properly joined the cluster, please refer to the be.WARNING log file in the log directory to ascertain the cause.

3. Inspecting Broker Nodes.

```
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)
```

Alive set to true signifies a normal status.

Establishing a New User.

Establish a standard user through the following command:

```
mysql > create user 'test' identified by '123456';
```

Create Database

In StarRocks, only the root account possesses the authority to establish a database. Log in using the root user to create the 'example_db' database:

```
mysql > create database example_db;
```

Upon the successful creation of the database, one can view the database information through the 'show databases' command:

```
mysql > show databases;

+--------------------+
| Database           |
+--------------------+
| example_db         |
| information_schema |
+--------------------+
2 rows in set (0.00 sec)
```

The 'information_schema' exists for the purpose of compatibility with the MySQL protocol. However, the information it provides may not always be accurate. Therefore, it is recommended to obtain specific database information through direct queries to the respective database.

Account Authorization

Upon the successful creation of 'example_db', the root account can grant read and write permissions for 'example_db' to the 'test' account. Once authorized, the 'test' account can log in and operate the 'example_db' database:

```
mysql > grant all on example_db to test;
```

Exit the root account and log into the StarRocks cluster using the 'test' account:

```
mysql > exit

mysql -h 127.0.0.1 -P9030 -utest -p123456
```

Create

StarRocks supports two methods of table creation: single partition and composite partition.
In the case of composite partition:

- The first level is known as Partition, or partitioning. Users can designate a specific dimension column as the partition column (currently, only integer and time type columns are supported), and specify the value range for each partition.
- The second level is referred to as Distribution, or bucketing. Users can specify a few dimension columns (or none, meaning all KEY columns) and the number of buckets to distribute the data using HASH.

The following scenarios are recommended for using composite partitioning:

- For dimensions with a time aspect or similar ordered values: such dimension columns can be used as partition columns. The granularity of the partition can be evaluated based on factors such as import frequency and the amount of data in the partition.
- Historical data deletion requirements: If there is a need to delete historical data (for example, only retaining data from the most recent N days). Using composite partitioning, this can be achieved by deleting historical partitions. Data can also be deleted by sending DELETE statements within a specified partition.
- Addressing data skew issues: The number of buckets can be individually specified for each partition. For instance, when partitioning by day, if the volume of data varies greatly each day, the number of buckets for the partition can be specified to reasonably divide the data across different partitions. It is recommended to choose columns with high differentiation for bucketing.

Users can also opt not to use composite partitioning, that is, use single partitioning. In this case, the data is only distributed using HASH.
Below are demonstrations of table creation statements for both types of partitioning:

1. First, switch the database: mysql > use example_db;
2. Establish a single partition table named table1. Use full hash bucketing, with siteid as the bucketing column and the number of buckets as 10. The schema of this table is as follows:
   - siteid: The type is INT (4 bytes), with a default value of 10.
   - city_code: The type is SMALLINT (2 bytes).
   - username: The type is VARCHAR, with a maximum length of 32, and a default value of an empty string.
   - pv: The type is BIGINT (8 bytes), with a default value of 0; this is a metric column, and StarRocks will perform aggregation operations on the metric columns, with the aggregation method for this column being summation (SUM). Here, the aggregation model is used. In addition to this, StarRocks also supports detail models and update models, for more details refer to Introduction to Data Models.
   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. Establish a composite partition table
   Create a logical table named table2. The schema of this table is as follows:
   - event_day: The type is DATE, with no default value.
   - siteid: The type is INT (4 bytes), with a default value of 10.
   - city_code: The type is SMALLINT (2 bytes).
   - username: The type is VARCHAR, with a maximum length of 32, and a default value of an empty string.
   - pv: The type is BIGINT (8 bytes), with a default value of 0; this is a metric column, and StarRocks will perform aggregation operations on the metric columns, with the aggregation method for this column being summation (SUM).
     We use the event_day column as the partition column, establishing three partitions: p1, p2, p3.
   - p1: The range is [minimum value, 2017-06-30).
   - p2: The range is [2017-06-30, 2017-07-31).
   - p3: The range is [2017-07-31, 2017-08-31).

Each partition uses siteid for hash bucketing, with a total of 10 buckets.
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");
```

Upon completion of the table creation, one can view the information of the table in example_db:

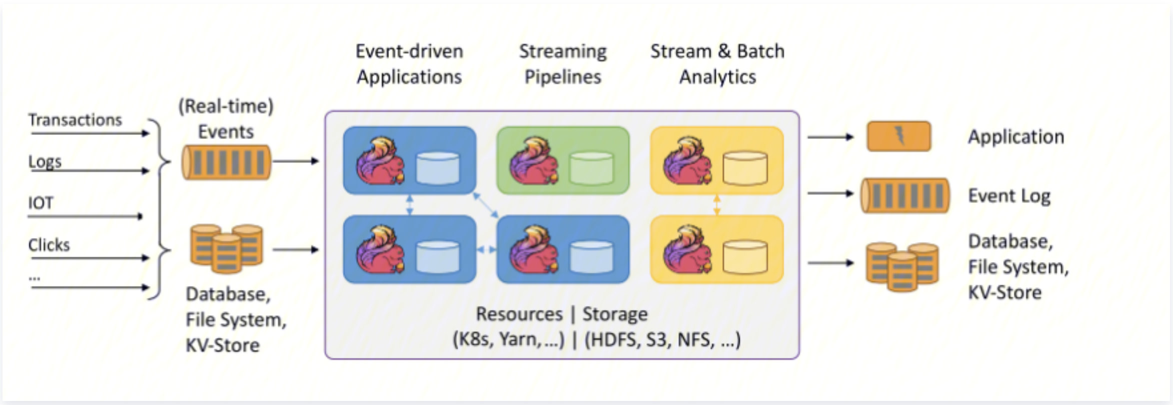
```
mysql> show tables;

+------------------------+
| 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)
```

Flink Development Guide
Flink Overview

Last updated：2024-01-02 14:19:48

Flink's core is an open-source, distributed, high-performance, highly available, and accurate data stream execution engine, offering functionalities such as data distribution, data communication, and fault tolerance mechanisms for distributed computation of data streams. Leveraging this stream execution engine, Flink provides a higher level of abstraction in its API to facilitate the writing of distributed tasks.

- Distributed: This signifies that Flink programs can operate across multiple machines.
- High Performance: This denotes that Flink possesses superior processing capabilities.
- High Availability: This indicates that Flink supports an automatic restart mechanism for programs.
- Accurate: This implies that Flink can ensure the precision of data processing.



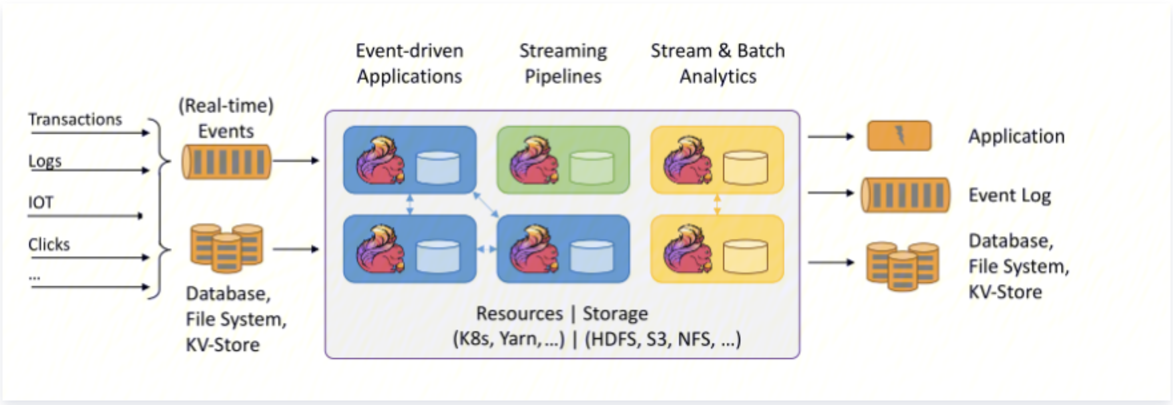
The diagram below illustrates the data source on the left, indicating that these data are real-time logs produced, or data from databases, file systems, or key-value storage systems. At the center is Flink, responsible for organizing the data. On the right is the destination, where Flink can output the processed data to other application systems or storage systems. The three core components of Flink are as follows:

- Data Source: This refers to the data source depicted on the left side of the diagram.
- Transformations: Operators (responsible for data processing).
- Data Sink: Output component (responsible for exporting the processed data to other application systems).

Use Cases

Flink primarily has the following three application scenarios:

1. Event-Driven Applications
   Event-driven applications are a category of stateful applications that extract data from one or more event streams, triggering computations, state updates, or other external actions based on incoming events.

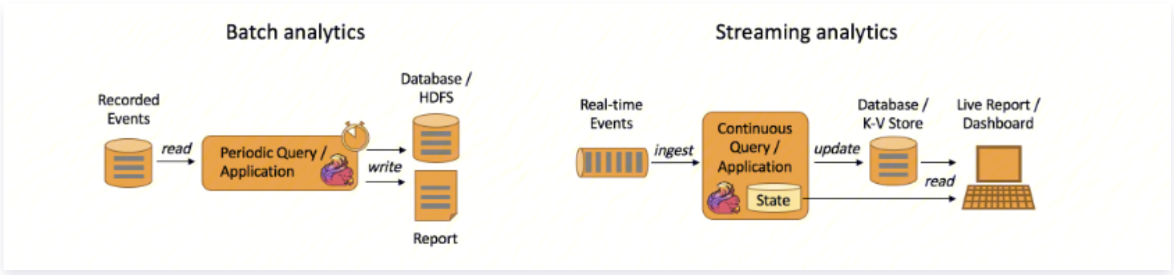


In traditional architectures (as depicted on the left), we need to read and write to remote transactional databases, such as

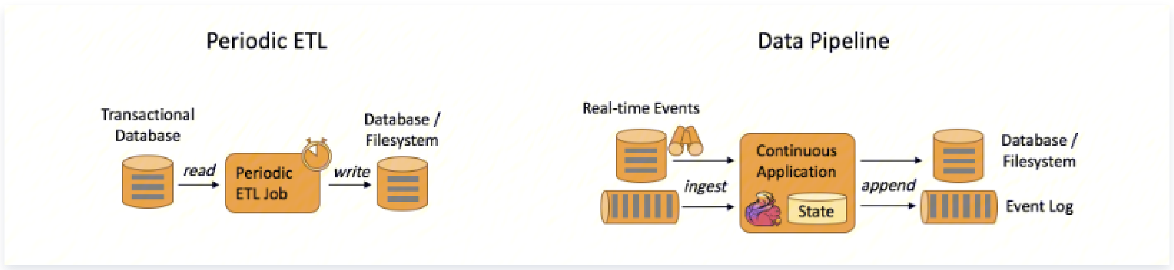
MySQL. In event-driven applications, data and computation are not separated; the application only needs to access local data (either in memory or on disk), resulting in higher throughput and lower latency.

- The following features of Flink perfectly support event-driven applications.
- Efficient state management, Flink's inherent State Backend can adeptly store intermediate state information.
- With its extensive window support, Flink accommodates a variety of windows, including tumbling windows, sliding windows, and others.
- Multiple time semantics, Flink supports Event Time, Processing Time, and Ingestion Time.
  Various levels of fault tolerance, Flink supports At Least Once or Exactly Once fault tolerance levels.

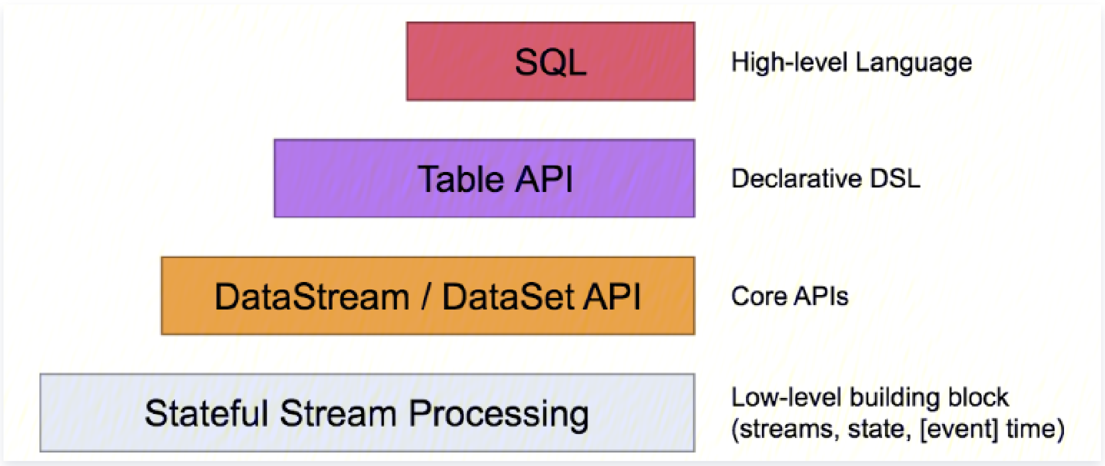
2. Real-Time Data Analysis Applications:
   Data analysis tasks require extracting valuable information and metrics from raw data. Traditional analysis methods typically involve batch queries or recording events and building applications based on this limited dataset. To obtain the analysis results of the latest data, they must first be added to the analysis dataset and the query or application must be re-executed, followed by writing the results to the storage system or generating a report.



3. Real-Time Data Warehousing and ETL
   The purpose of ETL (Extract-Transform-Load) is the process of extracting data from business systems, cleaning and transforming it, and then loading it into a data warehouse.
   Traditional offline data warehouses store business data centrally and use fixed computational logic to periodically perform ETL and other modeling to produce reports and other applications. Offline data warehouses primarily build T+1 offline data, pulling incremental data daily through scheduled tasks, then creating various business-related topic dimension data, and providing a T+1 data query interface.



The above diagram illustrates the differences between offline data warehouse ETL and real-time data warehousing, showing that both the computation and real-time nature of data in offline data warehouses are relatively poor. The value of data itself gradually diminishes over time, so once data is generated, it must reach users as quickly as possible, hence the need for real-time data warehouse construction arises.



For related layered APIs, please refer to the following documentation:

- Table API & SQL: The Table API is generally closely associated with DataSet or DataStream. It can create a Table from a DataSet or DataStream, and then perform operations such as filter, sum, join, select, etc. Recently, it has also been possible to convert a Table object into a DataSet or DataStream. The underlying layer of the SQL API is based on Apache Calcite, which implements standard SQL, making it more flexible to use than other APIs, as it allows for the direct use of SQL statements. The Table API and SQL API can be easily combined for use, as they both return Table objects.
- DataStream API & DataSet API: These primarily offer processing for stream data and batch data. They encapsulate some of the lower-level APIs, providing higher-order functions such as filter, sum, max, min, etc. These APIs are simple to use, hence their widespread application in actual production.
- Stateful Stream Processing: Offers fine-grained control over time and state, albeit with less simplicity and ease of use. It is primarily applied in complex event processing logic.

Environment information

- Flink is deployed by default on the cluster's Master and Core nodes. The functionality of a cluster with the Flink component installed is ready to use out of the box.
- After logging into the machine, use the command 'su hadoop' to switch to the hadoop user for conducting some local tests with Flink.
- The Flink software path is located under /usr/local/service/flink.
- The relevant log path is located under /data/emr/flink/logs.

For more detailed information, please refer to the community documentation.

Analyzing COS Data with Flink

Last updated：2024-01-02 14:19:57

Flink excels in handling both unbounded and bounded data sets. Its precise time control and statefulness enable Flink's runtime to operate any application that processes unbounded streams. Bounded streams, on the other hand, are internally managed by algorithms and data structures specifically designed for fixed-size data sets, resulting in outstanding performance.

The following demonstration operates on data from bounded or unbounded data sets stored in object storage. To better observe the job's operation, we use the yarn-session mode to submit tasks. Flink On Yarn supports Session Mode, Application Mode, and Per-Job Mode (deprecated in flink 1.15+), for more details, please refer to the community documentation.

This tutorial demonstrates the submission of a wordcount task, which counts the number of words. It is necessary to upload the file to be counted in the cluster in advance.

Development Preparations

1. As the task requires access to Tencent Cloud Object Storage (COS), it is necessary to first create a storage bucket in COS.
2. Ensure that you have activated Tencent Cloud and have created an EMR cluster. When creating the EMR cluster, you need to select the Flink component in the software configuration interface, and enable object storage authorization on the **Cluster>Cluster Information** page.
3. Upon completion of the cluster purchase, you can use HDFS to access object storage to ensure its basic functionality is available. The specific command is as follows:

```
[hadoop@10 ~]$ hdfs dfs -ls cosn://$BUCKET_NAME/path
Found 1 items
-rw-rw-rw-   1 hadoop hadoop      27040 2022-10-28 15:08 cosn://$BUCKET_NAME/path/LICENSE
```

   Here, $BUCKET_NAME/path represents the name and path of your storage bucket.

Sample

```
# -tm represents the memory size of each TaskManager.
# -s represents the number of slots for each TaskManager.
# -d signifies running as a background process, followed by the session name.
[hadoop@10 ~]$ yarn-session.sh -jm 1024 -tm 1024 1 -s 1 -nm wordcount-example -d
```

```
/usr/local/service/flink/bin/flink run -m yarn-cluster /usr/local/service/flink/examples/batch/WordCount.jar --input
cosn://$BUCKET_NAME/path/LICENSE -output cosn://$BUCKET_NAME/path/wdp_test
[hadoop@10 ~]$ hdfs dfs -ls cosn://$BUCKET_NAME/path/wdp_test
-rw-rw-rw-   1 hadoop hadoop       7484 2022-11-04 00:47 cosn://$BUCKET_NAME/path/wdp_test
```

Maven Project Sample

In this demonstration, we will not use the system's built-in demo program. Instead, we will create our own project, compile and package it, and then upload it to the EMR cluster for execution. It is recommended to use Maven to manage your project. Maven is a project management tool that can help you conveniently manage project dependencies. That is, it can obtain jar packages through the configuration of the pom.xml file, eliminating the need for manual addition.

1. Firstly, download and install Maven, and configure the Maven environment variables. If you are using an IDE, please set the Maven related configuration within the IDE.
2. Navigate to the directory where you want to create a new project in your local shell, for example, in D://mavenWorkplace, and enter the following command to create a new Maven project:

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

Wherein, yourgroupID is your package name. YourartifactID is your project name, and maven−archetype−quickstart indicates the creation of a Maven Java project. Some files need to be downloaded during the project creation process, so please ensure a stable internet connection.
Upon successful creation, a project folder named $yourartifactID will be generated in the D://mavenWorkplace directory. The file structure is as follows:

```
simple
 ---pom.xml           Core configuration, located at the root of the project
 ---src
     ---main
         ---java           Java source code directory
         ---resources     Java configuration files directory
     ---test
          ---java           Test source code directory
          ---resources     Test configuration directory
```

Our primary focus lies on the pom.xml file and the Java folder under main. The pom.xml file is primarily used for dependency and packaging configurations, while the Java folder houses your source code.
Firstly, add the Maven dependency in the pom.xml:

```
<properties>
 <scala.version>2.12</scala.version>
 <flink.version>1.14.3</flink.version>
</properties>

<dependencies>
 <dependency>
     <groupId>org.apache.flink</groupId>
     <artifactId>flink-java</artifactId>
     <version>1.14.3</version>
     <scope>provided</scope>
 </dependency>
 <dependency>
     <groupId>org.apache.flink</groupId>
     <artifactId>flink-streaming-scala_${scala.version}</artifactId>
     <version>${flink.version}</version>
     <scope>provided</scope>
 </dependency>
 <dependency>
     <groupId>org.apache.flink</groupId>
     <artifactId>flink-clients_${scala.version}</artifactId>
     <version>${flink.version}</version>
     <scope>provided</scope>
 </dependency>
</dependencies>
```

Note
Please ensure the scala.version and flink.version are consistent with the component versions in the EMR version you are using.

Proceed to add packaging and compilation plugins in the 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>
```

If your Maven configuration is correct and the dependency packages have been successfully imported, then the entire project should be error-free and ready for compilation. Enter the project directory in the local command line mode and execute the following command to package the entire project:

```
mvn package
```

During the execution, additional files may need to be downloaded until a 'build success' message appears, indicating successful packaging. Subsequently, you can locate the packaged jar file in the target folder under the project directory.

Data Preparation

Initially, the compressed jar package needs to be uploaded to the EMR cluster using scp or sftp tools. Run the following command in the local command line mode:

```
scp $localfile root@PublicIPAddress:$remotefolder
```

Here, $localfile refers to the combination of the path and name of your local file; 'root' serves as the username of your server. You may find the public internet protocol address in the node information of the control panel or the cloud server console. $remotefolder represents the CVM server path where you wish to store your files. Once the upload finishes, you can confirm the presence of your desired files by checking the corresponding folders through the EMR command line.
The files you require to process should have been previously uploaded to COS. If the file is locally stored, it can be uploaded directly via the COS console. If the file is on the EMR cluster, you can utilize the Hadoop command for upload. The directive as follows:

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

Execute Sample

You first need to log into any machine in the EMR cluster, preferably the Master node. Refer to the Linux Instance for help on logging into EMR. Here, we can opt for using WebShell for log-in. By clicking on the 'log-in' to the right of the corresponding cloud server, you enter the log-in page. The username by default is 'root', while the password is the one set by the user during the creation of EMR. Upon entering correctly, you gain access to the command line interface.
On the EMR command line, employ the following command to switch to the Hadoop user initially:

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

```
[hadoop@172 ~]$ flink  run  -m yarn-cluster -c com.tencent.flink.CosWordcount ./flink-example-1.0-SNAPSHOT.jar
cosn://$BUCKET_NAME/test/data.txt cosn://$BUCKET_NAME/test/result
[hadoop@172 ~]$ hdfs dfs -cat cosn://becklong-cos/test/result
(Flink,8)
(Hadoop,3)
(Spark,7)
(Hbase,3)
```

RSS Development Guide

Last updated：2024-01-02 14:20:08

Introduction to RSS

Apache Uniffle is a unified Remote Shuffle Service (RSS) for computational engines, possessing the capability to aggregate and store shuffle data on remote servers, thereby significantly enhancing the performance and reliability of large-scale tasks.

Background

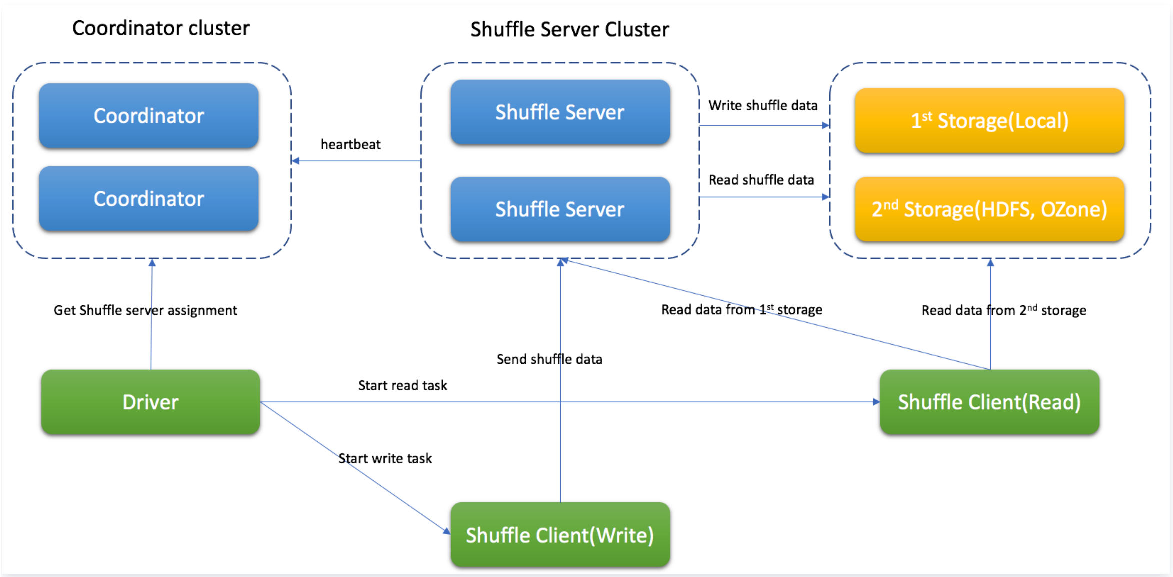
Issues with Existing Shuffle Schemes:

- Existing shuffle schemes cannot achieve serverless computation, leading to the recalculation of shuffle data when machine resources are preempted.
- Shuffle Partition can result in a multitude of shuffle connections and small network packets, which are highly susceptible to timeout issues in large-scale scenarios.
- The shuffle process is plagued by write amplification and random IO issues. When shuffle data becomes excessively large, it can severely impact the performance and stability of the cluster.
- When the Spark Shuffle service and NodeManager operate within the same process, high IO loads can easily trigger a NodeManager restart, thereby affecting Yarn scheduling.

Fundamental Characteristics of RSS:

1. Storing shuffle data on remote servers supports the separation of computation and storage, as well as mixed cluster deployment modes.
2. Supports shuffle data aggregation and data caching mechanisms, maximizing the utilization of memory resources and reducing random disk access.
3. Supports a variety of storage methods for shuffle data, such as local files, HDFS files, and hybrid modes.
4. Supports the integrity verification of shuffle data, filtering invalid data while ensuring the accuracy of data during the task computation process.
5. Adopts a master-slave architecture and active-standby mode, enhancing the utilization of cluster resources and the stability of services.

RSS Basic Architecture



The functions of each component are as follows:

- The Coordinator, managing the Shuffle Server based on a heartbeat mechanism, stores metadata information such as the resource usage of the Shuffle Server. It also undertakes task allocation responsibilities, assigning the appropriate Shuffle Server to the Spark application to handle different Partition data based on the load of the Shuffle Server.

- The Shuffle Server primarily handles the reception of shuffle data, aggregating it before writing to storage. Based on different storage methods, it can also be used to read shuffle data (such as in LocalFile storage mode).
- The Shuffle Client primarily communicates with the Coordinator and Shuffle Server, sending read and write requests for shuffle data, and maintaining the heartbeat between the application and the Coordinator.

RSS Usage

The RSS cluster provides remote shuffle services and cannot be used independently. It requires association with a Spark container cluster for use.
To associate, click on **Spark Cluster > Cluster Information > Associate RSS Cluster**, then select an RSS cluster that is running. For more details, please refer to Associate RSS Cluster .

RSS Configuration

You can view the configuration files and common configuration items of different RSS roles on the **Cluster Services > Configuration Management** page.
The configuration files for the Coordinator role are coordinator.conf and log4j.properties, while the Shuffle Server role has server.conf and log4j.properties files.
Common configuration items in coordinator.conf are not recommended to be modified:

Category	Default value	Description
rss.coordinator.app.expired	60000	Application Expiration Time (ms)
rss.coordinator.dynamicClientConf.enabled	false	Whether to enable dynamic client configuration, to be obtained by the Spark client.
rss.coordinator.exclude.nodes.file.path	file:///usr/local/service/rss/conf/exclude_nodes	Configuration file path for excluding nodes
rss.coordinator.server.heartbeat.timeout	30000	Timeout if unable to receive heartbeat from the shuffle server.
rss.coordinator.shuffle.nodes.max	1000	Maximum number of Shuffle Servers during allocation
rss.jetty.http.port	19998	HTTP Port of the Coordinator
rss.rpc.server.port	19999	RPC Port of the Coordinator
rss.storage.type	MEMORY_LOCALFILE	RSS storage types include MEMORY_ONLY, MEMORY_LOCALFILE, and MEMORY_LOCALFILE_HDFS. Due to the unavailability of a Hadoop cluster, the current default is MEMORY_LOCALFILE.

Common configuration items in server.conf, modification is not recommended:

Category	Default value	Description
rss.coordinator.quorum	rss-coordinator-rss-<ClusterId>-0:19999, rss-coordinator-rss-<ClusterId>-1:19999	Coordinator Address Information
rss.jetty.http.port	19998	HTTP Port of the Shuffle Server
rss.rpc.server.port	19999	RPC Port of the Shuffle Server
rss.server.buffer.capacity	Memory Limit Value * 0.75 * 0.6	Maximum Memory of the Shuffle Server Buffer Manager

rss.server.disk.capacity	Single Data Disk Capacity * 0.9	Disk Capacity Available to the Shuffle Server
rss.server.flush.thread.alive	Number of Data Disks	Number of Threads Flushing Data to Files
rss.server.flush.threadPool.size	Number of Data Disks * 2	Size of the Thread Pool for Flushing Data to Files
rss.server.heartbeat.interval	10000	Heartbeat Interval to the Coordinator
rss.server.heartbeat.timeout	60000	Heartbeat Timeout Duration
rss.server.read.buffer.capacity	Memory Limit Value * 0.75 * 0.2	Maximum Buffer Size for Data Reading
rss.storage.basePath	The default number of /data1/rssdata, /data2/rssdata... paths is equivalent to the quantity of data disks.	Path for Writing Shuffle Data to the Data Disk
rss.storage.type	MEMORY_LOCALFILE	RSS Storage Type, which must be consistent with the Coordinator.

For more information, refer to incubator-uniffle.