Cloudera Runtime 7.3.3

Securing Apache Hive

Date published: 2019-08-21
Date modified: 2026-11-13



https://docs.cloudera.com/

Legal Notice

© Cloudera Inc. 2026. All rights reserved.

The documentation is and contains Cloudera proprietary information protected by copyright and other intellectual property rights. No license under copyright or any other intellectual property right is granted herein.

Unless otherwise noted, scripts and sample code are licensed under the Apache License, Version 2.0.

Copyright information for Cloudera software may be found within the documentation accompanying each component in a particular release.

Cloudera software includes software from various open source or other third party projects, and may be released under the Apache Software License 2.0 (“ASLv2”), the Affero General Public License version 3 (AGPLv3), or other license terms. Other software included may be released under the terms of alternative open source licenses. Please review the license and notice files accompanying the software for additional licensing information.

Please visit the Cloudera software product page for more information on Cloudera software. For more information on Cloudera support services, please visit either the Support or Sales page. Feel free to contact us directly to discuss your specific needs.

Cloudera reserves the right to change any products at any time, and without notice. Cloudera assumes no responsibility nor liability arising from the use of products, except as expressly agreed to in writing by Cloudera.

Cloudera, Cloudera Altus, HUE, Impala, Cloudera Impala, and other Cloudera marks are registered or unregistered trademarks in the United States and other countries. All other trademarks are the property of their respective owners.

Disclaimer: EXCEPT AS EXPRESSLY PROVIDED IN A WRITTEN AGREEMENT WITH CLOUDERA, CLOUDERA DOES NOT MAKE NOR GIVE ANY REPRESENTATION, WARRANTY, NOR COVENANT OF ANY KIND, WHETHER EXPRESS OR IMPLIED, IN CONNECTION WITH CLOUDERA TECHNOLOGY OR RELATED SUPPORT PROVIDED IN CONNECTION THEREWITH. CLOUDERA DOES NOT WARRANT THAT CLOUDERA PRODUCTS NOR SOFTWARE WILL OPERATE UNINTERRUPTED NOR THAT IT WILL BE FREE FROM DEFECTS NOR ERRORS, THAT IT WILL PROTECT YOUR DATA FROM LOSS, CORRUPTION NOR UNAVAILABILITY, NOR THAT IT WILL MEET ALL OF CUSTOMER’S BUSINESS REQUIREMENTS. WITHOUT LIMITING THE FOREGOING, AND TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, CLOUDERA EXPRESSLY DISCLAIMS ANY AND ALL IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, QUALITY, NON-INFRINGEMENT, TITLE, AND FITNESS FOR A PARTICULAR PURPOSE AND ANY REPRESENTATION, WARRANTY, OR COVENANT BASED ON COURSE OF DEALING OR USAGE IN TRADE.

Contents

Hive access authorization **4**
Security risks of Hive user-defined functions **5**
Reviewing Hive UDFs for security risks 6
Transactional table access **6**
External table access **7**
Accessing Hive files in Ozone **7**
Configuring access to Hive on YARN **9**
Configuring HiveServer for ETL using YARN queues 11
Managing YARN queue users 11
Configuring queue mapping to use the user name from the application tag using Cloudera Manager 12
Disabling impersonation (doas) **13**
Connecting to an Apache Hive endpoint through Apache Knox **13**
HWC authorization **14**
Hive authentication **17**
Securing HiveServer using LDAP 17
Client connections to HiveServer 19
Pluggable authentication modules in HiveServer 20
JDBC connection string syntax 20
Configuring ZooKeeper SASL authentication for Hive clients 22
Communication encryption **23**
Enabling TLS/SSL for HiveServer 23
Enabling SASL in HiveServer 24
Securing an endpoint under AutoTLS **25**
Securing Hive metastore 26
Activating the Hive web UI **26**

Hive access authorization

As administrator, you need to understand that the Hive default authorization for running Hive queries is insecure and what you need to do to secure your data. You need to set up Apache Ranger.

To limit Apache Hive access to approved users, Cloudera recommends and supports only Ranger. Authorization is the process that checks user permissions to perform select operations, such as creating, reading, and writing data, as well as editing table metadata. Apache Ranger provides centralized authorization for all Cloudera Runtime Services.

You can set up Ranger to protect managed, ACID tables or external tables using a Hadoop SQL policy. You can protect external table data on the file system by using an HDFS policy in Ranger.

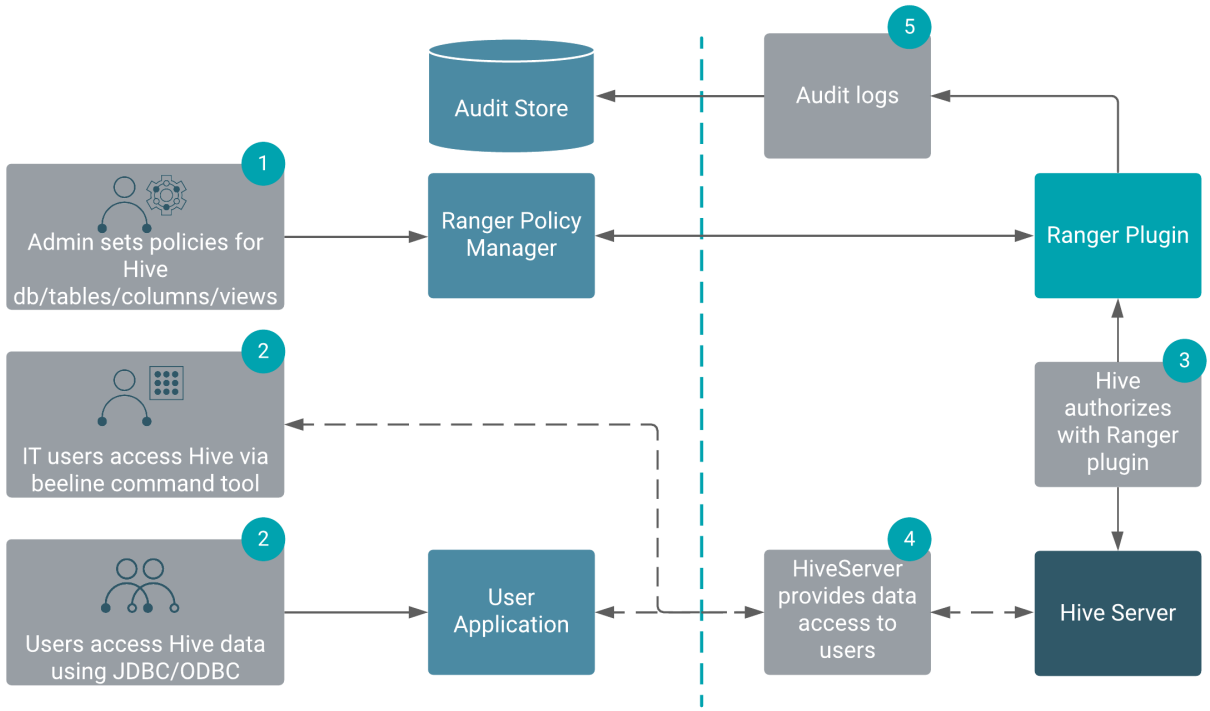
Preloaded Ranger Policies

In Ranger, preloaded Hive policies are available by default. Users covered by these policies can perform Hive operations. All users need to use the default database, perform basic operations such as listing database names, and query the information schema. To provide this access, preloaded default database tables columns and information_ schema database policies are enabled for group public (all users). Keeping these policies enabled for group public is recommended. For example, if the default database tables columns policy is disabled preventing use of the default database, the following error appears:

```
hive> USE default;
Error: Error while compiling statement: FAILED: HiveAccessControlException
Permission denied: user [hive] does not have [USE] privilege on [default]
```

Apache Ranger policy authorization

Apache Ranger provides centralized policy management for authorization and auditing of all Cloudera Runtime services, including Hive. All Cloudera Runtime services are installed with a Ranger plugin used to intercept authorization requests for that service, as shown in the following illustration.



The following table compares authorization models:

Authorization model	Secure?	Fine-grained authorization (column, row level)	Privilege management using GRANT/REVOKE statements	Centralized management GUI
Apache Ranger	Secure	Yes	Yes	Yes
Hive default	Not secure. No restriction on which users can run GRANT statements	Yes	Yes	No

When you run grant/revoke commands and Apache Ranger is enabled, a Ranger policy is created/removed.

Prerequisites for Ranger-Hive Integration

When setting up a Hive service in the Ranger Admin UI, you may encounter a ClassNotFoundException error when clicking the **Test Connection** button. This issue occurs because a key dependency, the Hive on Tez gateway role, is not installed on the Ranger Admin host. The missing role prevents Ranger from successfully creating and validating the Hive repository.

To avoid this error, you must install the Hive on Tez gateway role on the Ranger Admin host before creating the Hive service. This ensures all necessary dependencies are in place for Ranger to communicate properly with the Hive service.

The error message you might see is similar to the following:

```
ERROR org.apache.ranger.services.hive.client.HiveClient: Unable to Connect t
o Hive
org.apache.ranger.plugin.client.HadoopException: initConnection: Meta info
 is not proper.So unable to initiate connection to hive thrift server insta
nce.
...
Caused by: java.lang.ClassNotFoundException: com.codahale.metrics.Metric
```

Related Information

HDFS ACLS

Configure a Resource-based Policy: Hive

Row-level Filtering and Column Masking in Hive

Query Hive

Security risks of Hive user-defined functions

Learn about security risks associated with the GenericUDFReflect class and how to restrict Hive user-defined functions (UDFs) to prevent unauthorized code execution or HDFS data exposure.

Hive UDFs support custom data processing, but granting incorrect UDF privileges exposes the Hive service to security vulnerabilities.

Risks of the TEMPUDFADMIN privilege

The TEMPUDFADMIN privilege allows you to create and invoke temporary functions without additional controls. Because temporary functions can be named differently than the blacklisted UDFs (hive.server2.builtin.udf.blacklist) they can bypass these restrictions, a user with this privilege can execute arbitrary code under the Hive service user. This access might lead to the exposure of sensitive HDFS data or the unauthorized use of the Hive Kerberos ticket.

Risks of the SELECT privilege

The SELECT privilege on a function acts as a safety gate.If you grant SELECT access to a function that points to malicious or risky code, you can inadvertently give a user access to sensitive data. Cloudera recommends that Ranger administrators review the function class and underlying code before granting this privilege.

Reviewing Hive UDFs for security risks

Verify that a Hive function does not use restricted classes or contain malicious code before you grant the SELECT privilege to a user.

About this task

Before you grant the SELECT privilege on a Hive function, you must verify that the function is secure. This process helps prevent the execution of unauthorized code and ensures that only trusted UDFs are accessible to users.

Before you begin

You must have administrative access to the Ranger UI and the ability to run commands in `Beeline` or the Data Explorer (Hue) Hive editor.

Procedure

1. Run the following command in `Beeline` or the Data Explorer (Hue) Hive editor to check the function details:

   ```
   DESCRIBE FUNCTION EXTENDED <db_name>.<udf_name>;
   ```

2. Review the Function class property in the output to identify the underlying Java class.

   Note: The following example shows a typical output for a function description:

   ```
   test.my_udf(class,method[,arg1[,arg2..]])
    calls method with reflection   Function
    class:org.apache.hadoop.hive.ql.udf.generic.GenericUDFReflect
   ```

3. Verify that the UDF does not use restricted classes.

   Confirm that the **Function class** is not one of the following elements:

   - org.apache.hadoop.hive.ql.udf.generic.GenericUDFReflect
   - org.apache.hadoop.hive.ql.udf.generic.GenericUDFReflect2
   - org.apache.hadoop.hive.ql.udf.generic.GenericUDFInFile

4. Conduct an internal code review of the custom UDF.

   Ensure the code does not perform the following actions:

   - Execute external scripts
   - Use unauthorized reflection or custom code execution

Results

After verifying that the function class is safe and the code is trusted, you can proceed with granting the SELECT privilege in Ranger.

Transactional table access

As administrator, you must enable the Apache Ranger service to authorize users who want to work with transactional tables. These types of tables are the default, ACID-compliant tables in Hive 3 and later.

ACID tables reside by default in /warehouse/tablespace/managed/hive. Only the Hive service can own and interact with files in this directory. Ranger is the only available authorization mechanism that Cloudera recommends for ACID tables.

External table access

As administrator, you must set up Apache Ranger to allow users to access external tables.

External tables reside by default in /warehouse/tablespace/external on your object store. To specify some other location of the external table, you need to include the specification in the table creation statement as shown in the following example:

```
CREATE EXTERNAL TABLE my_external_table (a string, b string)
LOCATION '/users/andrena';
```

Hive assigns a default permission of 777 to the hive user, sets a umask to restrict subdirectories, and provides a default ACL to give Hive read and write access to all subdirectories. External tables must be secured using Ranger.

Accessing Hive files in Ozone

Learn how to set up policies to give users access to Hive external files in Ozone. For example, if Ozone users are running SparkSQL statements that query Hive tables, you must set up an Ozone access policy and Ozone file system access policy.

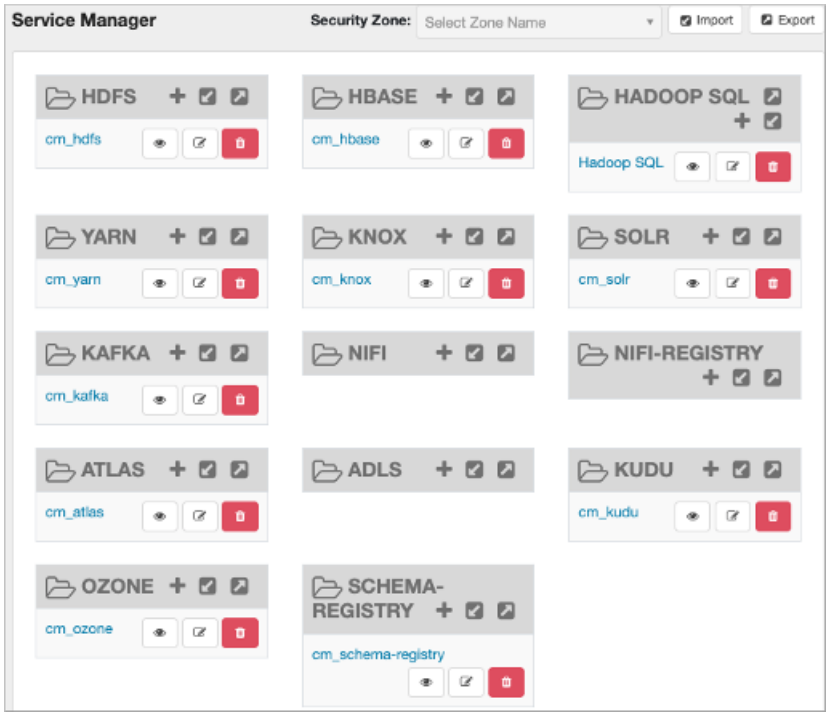
About this task

When Ranger is enabled in the cluster, any user other than the default admin user, "om" requires the necessary Ranger permissions and policy updates to access the Ozone filesystem. To create a Hive external table that points to the Ozone filesystem, the "hive" user should have the required permissions in Ranger.

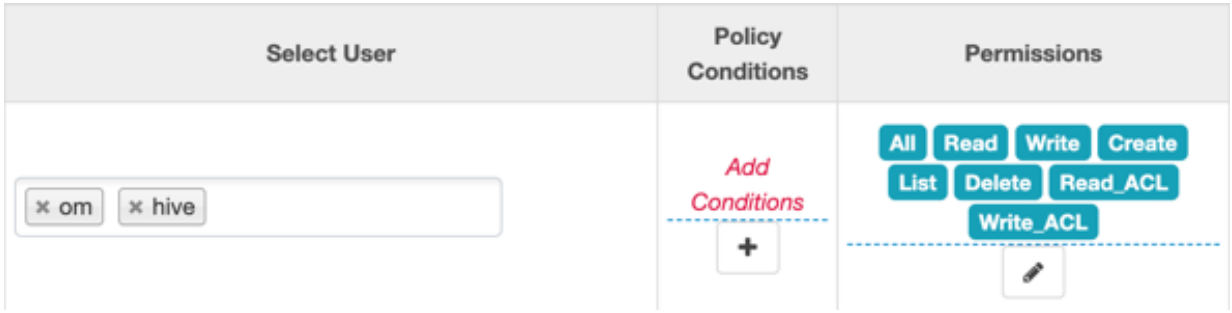
In this task, you first enable Ozone in the Ranger service, and then set up the required policies.

Procedure

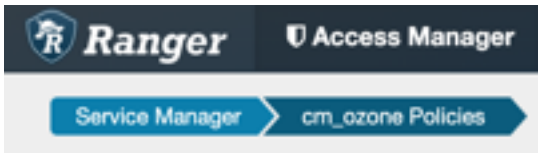
1. In Cloudera Manager, click Clusters Ozone Configuration to navigate to the configuration page for Ozone.
2. Search for ranger_service, and enable the property.
3. Click Clusters Ranger Ranger Admin Web UI , enter your user name and password, then click Sign In.
   The **Service Manager for Resource Based Policies** page is displayed in the Ranger console.



4. Click the cm_ozone preloaded resource-based service to modify an Ozone policy.
5. In the cm_ozone policies page, click the Policy ID or click  Edit against the "all - volume, bucket, key" policy to modify the policy details.
6. In the Allow Conditions pane, add the "hive" user, choose the necessary permissions, and then click Save.

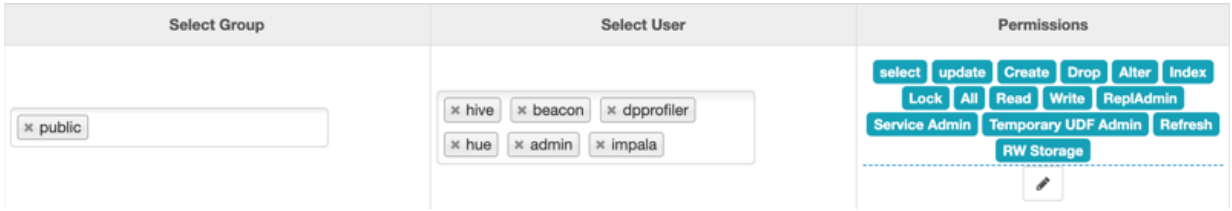


7. Click the Service Manager link in the breadcrumb trail and then click the Hadoop SQL preloaded resource-based service to update the Hadoop SQL URL policy.



8. In the Hadoop SQL policies page, click the Policy ID or click  Edit against the "all - url" policy to modify the policy details.

   By default, "hive", "Data Explorer", "impala", "admin" and a few other users are provided access to all the Ozone URLs. You can select users and groups in addition to the default. To grant everyone access, add the "public" group to the group list. Every user is then subject to your allow conditions.



What to do next

Create a Hive external table having source data in Ozone.

Also, it is recommended that you set certain Hive configurations before querying Hive tables in Ozone.

Related Information

Set up Ozone security

Cloudera's Ranger documentation

Creating an Ozone-based Hive external table

Configuring access to Hive on YARN

By default, access to Hive and YARN by unauthorized users is not allowed. You also cannot run unauthorized workloads on YARN. You need to know how to give end users and workloads the access rules necessary for querying Hive workloads in YARN queues.

About this task

You must configure the following access to query Hive workloads on YARN:

- Allow the end user to access Hive
- Allow the Hive workload on YARN
- Allow the end user to access YARN

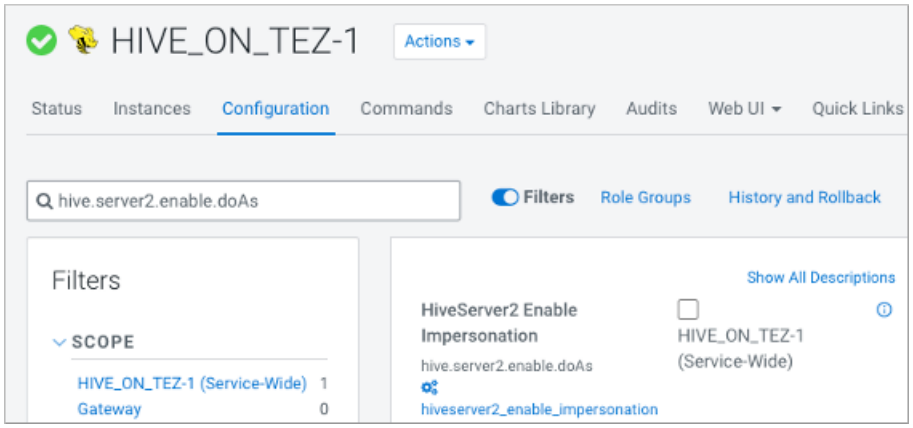
Before you begin

Ensure the Hive user has access to the required YARN queues in the cm_yarn Ranger repository for job submission.

Follow the steps in this topic to configure Hive and YARN for end users to access Hive on YARN.

Procedure

1. In Cloudera Manager, click Clusters Hive on Tez Configuration and search for hive.server2.enable.doAs.

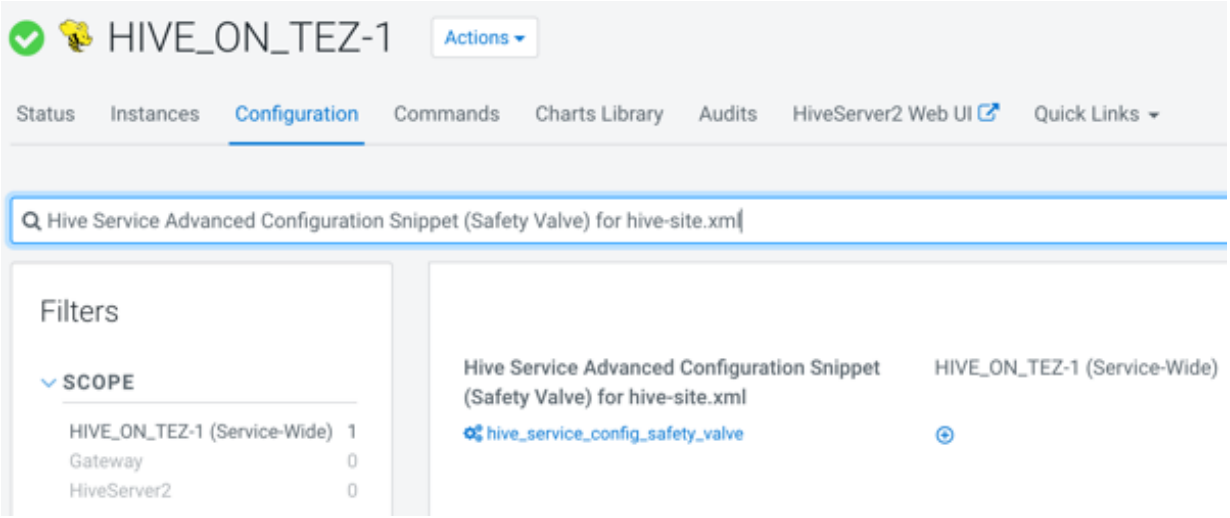


2. Set the value of doas to false. Uncheck Hive (Service-Wide) to disable impersonation.

For more information about configuring doas, see *Enabling or disabling impersonation*.

Save changes.

3. Search for the Hive Service Advanced Configuration Snippet (Safety Valve) for hive-site.xml setting.



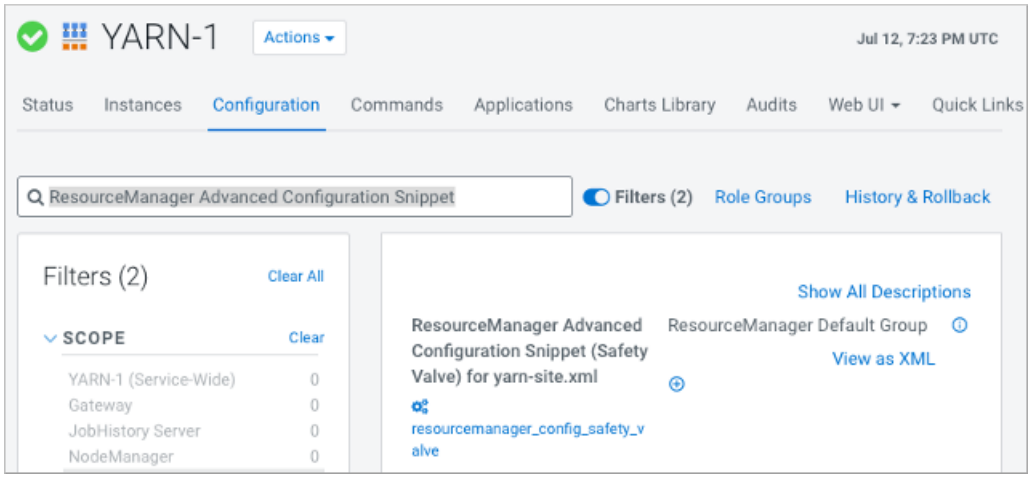
4. In the Hive Service Advanced Configuration Snippet (Safety Valve) for hive-site.xml setting, click ⊕.
5. Add the properties and values to allow the Hive workload on YARN.

```
Name: hive.server2.tez.initialize.default.sessions Value: false
Name: hive.server2.tez.queue.access.check Value: true
Name: hive.server2.tez.sessions.custom.queue.allowed Value: true
```

For more information about allowing the Hive workload on YARN, see *Configuring HiveServer for ETL using YARN queues*.

Save changes.

6. In Cloudera Manager, click Clusters YARN Configuration , and search for ResourceManager Advanced Configuration Snippet (Safety Valve) for yarn-site.xml.



7. In the ResourceManager Advanced Configuration Snippet (Safety Valve) for yarn-site.xml setting, click ⊕.
8. Add the properties and values to allow the end user to access YARN using placement rules.

```
Name: yarn.resourcemanager.application-tag-based-placement.enable Value:
 true
```

```
Name: yarn.resourcemanager.application-tag-based-placement.username.white
list Value: < Comma separated list of users who can use the application
tag based placement.>
```

For more information about allowing end user access to YARN, see *Configure queue mapping to use the user name from the application tag usingCloudera Manager*.

Save changes.

9. Restart the YARN ResourceManager service for the changes to apply.
End users you specified can now query Hive workloads in YARN queues.

Configuring HiveServer for ETL using YARN queues

You need to set several configuration properties to allow placement of the Hive workload on the Yarn queue manager, which is common for running an ETL job. You need to set several parameters that effectively disable the reuse of containers. Each new query gets new containers routed to the appropriate queue.

About this task

Hive configuration properties affect mapping users and groups to YARN queues. You set these properties to use with YARN Placement Rules.

To set Hive properties for YARN queues:

Procedure

1. In Cloudera Manager, click Clusters Hive-on-Tez Configuration .
2. Search for the Hive Service Advanced Configuration Snippet (Safety Valve) for hive-site.xml setting.
3. In the Hive Service Advanced Configuration Snippet (Safety Valve) for hive-site.xml setting, click +.
4. In Name enter the property hive.server2.tez.initialize.default.sessions and in value enter false.
5. In Name enter the property hive.server2.tez.queue.access.check and in value enter true.
6. In Name enter the property hive.server2.tez.sessions.custom.queue.allowed and in value enter true.

Managing YARN queue users

To manage users of secure YARN queues, you need to know how to configure impersonation for Ranger.

To allow access to YARN queues, as Administrator, you configure HiveServer user impersonation to false. You also need to configure hive.server2.tez.queue.access.check=true. To manage YARN queues, you need the following behavior:

- User submits the query through HiveServer (HS2) to the YARN queue
- Tez app starts for the user
- Access to the YARN queue is allowed for this user.

  As administrator, you can allocate resources to different users.

Managing YARN queues under Ranger

When you use Ranger, you configure HiveServer not to use impersonation (doas=false). HiveServer authorizes only the hive user, not the connected end user, to access Hive tables and YARN queues unless you also configure the following parameter:

hive.server2.tez.queue.access.check=true

Configuring queue mapping to use the user name from the application tag using Cloudera Manager

You learn how to add service users to the YARN queue by following a mapping procedure.

You can configure queue mapping to use the user name from the application tag instead of the proxy user who submitted the job. You can add only service users like hive using the `yarn.resourcemanager.application-tag-based-placement.username.whitelist` property and not normal users.

When a user runs Hive queries, HiveServer2 submits the query in the queue mapped from an end user instead of a hive user. For example, when user *ALICE* submits a Hive query with doAs=false mode, job will run in YARN as hive user. If application-tag based scheduling is enabled, then the job will be placed to a target queue based on the queue mapping configuration of user *ALICE*.

For more information about queue mapping configurations, see Manage placement rules. For information about Hive access, see Apache Hive documentation.

How to configure queue mapping

1. In Cloudera Manager, select the YARN service.
2. Click the Configuration tab.
3. Search for ResourceManager. In the Filters pane, under Scope, select ResourceManager.
4. In ResourceManager Advanced Configuration Snippet (Safety Valve) for yarn-site.xml add the following:
   a. Enable the application-tag-based-placement property to enable application placement based on the user ID passed using the application tags.

```
Name: yarn.resourcemanager.application-tag-based-placement.enable
Value: true
Description: Set to "true" to enable application placement based on the
 user ID passed using the application tags. When it is enabled, it chec
ks for the userid=<userId> pattern and if found, the application will be
 placed onto the found user's queue, if the original user has the requir
ed rights on the passed user's queue.
```

   b. Add the list of users in the allowlist who can use application tag based placement. The applications when the submitting user is included in the allowlist, will be placed onto the queue defined in the yarn.scheduler.capacity.queue-mappings property defined for the user from the application tag. If there is no user defined, the submitting user will be used.

```
Name: yarn.resourcemanager.application-tag-based-placement.username.whit
elist
Value: <Hive process user(s)>
Description: Comma separated list of users who can use the application t
ag based placement, if "yarn.resourcemanager.application-tag-based-place
ment.enable" is enabled.
```

   Note: Check the Hive system user value(s) and add the value(s) to the allowlist:
   1. In Cloudera Manager, navigate to Hive Configuration .
   2. Search for System User.
   3. Note down the value(s) set as process username(s), and add the value(s) to the allowlist.
   4. When there are no rules mapping for the tag user, it will fallback to the proxy user's (Hive) placement rule.
5. Restart the ResourceManager service for the changes to apply.

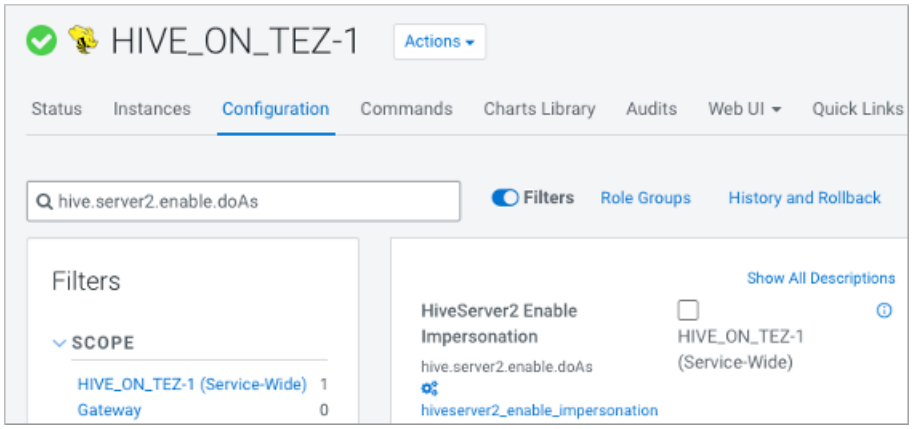
Disabling impersonation (doas)

As administrator, you must understand the permissions model supported in Cloudera Base on premises is Apache Ranger.

Disable impersonation to use Ranger

When you enable Ranger, you disable user impersonation (doAs=false). This is the Hive default and Ranger is the only supported and recommended security model. Managed, ACID tables as well as external tables, secured by Ranger, are supported in this configuration. Impersonation of the end user is disabled, which is the state required by Hive for managing ACID tables.

In Cloudera Manager, click Hive on Tez Configuration and search for (hive.server2.enable.doAs).



Uncheck Hive (Service-Wide) to disable impersonation.

With no impersonation, HiveServer authorizes only the hive user to access Hive tables.

Related Information

Apache Software Foundation HDFS Permissions Guide

HDFS ACLS

Connecting to an Apache Hive endpoint through Apache Knox

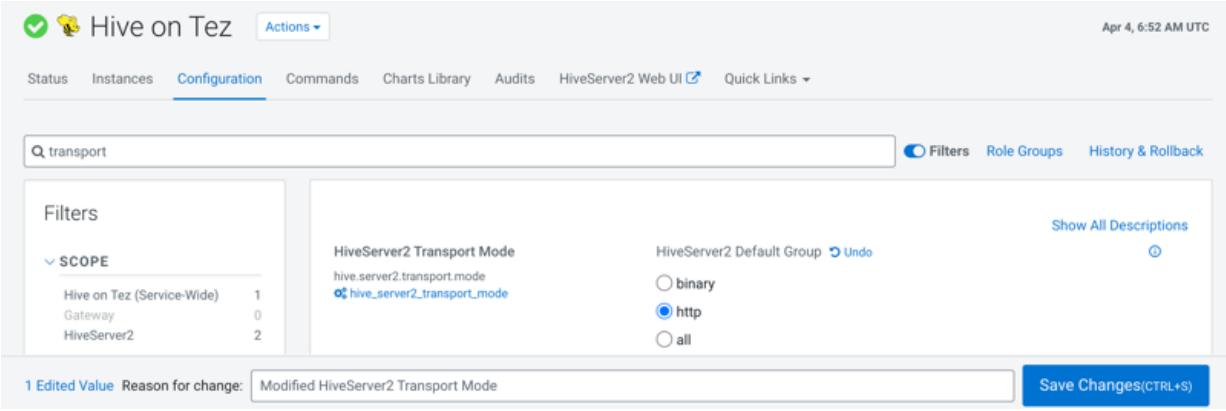
If your cluster uses Apache Knox for perimeter security in Cloudera Base on premises, you can connect to an Apache Hive endpoint through Knox. You set the HiveServer transport mode and reference your Java keystore.

Before you begin

Automate the creation of an internal certificate authority (CA) using Auto-TLS (see link below). Set up SSL, including trust, for Knox Gateway clients.

Procedure

1. In Cloudera Manager, click Clusters Hive on Tez Configuration , and change the Hive on Tez service transport mode in Cloudera Manager to http.



KNOX discovers the service automatically and builds a proxy URL for Hive on Tez only when the transport mode is http.

2. Download the Knox Gateway TLS/SSL client trust store JKS file from Knox, and save it locally.

   You can find the location of the JKS file from value of the Knox property gateway.tls.keystore.path.

3. In the Hive connection string, include parameters as follows:

```
jdbc:hive2://<host>:8443/;ssl=true;transportMode=http;  \
httpPath=gateway/cdp-proxy-api/hive;  \
sslTrustStore=/<path to JKS>/bin/certs/gateway-client-trust.jks;  \
trustStorePassword=<Java default password>
```

   In this example, changeit is the Java default password for the trust store.

HWC authorization

The way you configure Hive Warehouse Connector (HWC) affects the query authorization process and your security. There are a number of ways to access Hive through HWC, and not all operations go through HiveServer (HS2). Some operations, such as Spark Direct Reader and Hive Streaming, go to Hive directly through HMS where storage-based permissions generally apply.

As a client user, you must be logged in using kerberos before using HWC. You need appropriate storage permissions to write to destination partition or table location. You need to configure an HWC read option. HWC read configuration options are shown in the following table:

Table 1:

Capabilities	JDBC mode	Spark Direct Reader mode	Secure Access mode
Ranger integration with fine-grained access control	#	N/A	#
Hive ACID reads	#	#	#
Workloads handled	Non-production workloads, small datasets	Production workloads, ETL without fine-grained access control	Large workloads with fine-grained access control, row filtering, and column masking

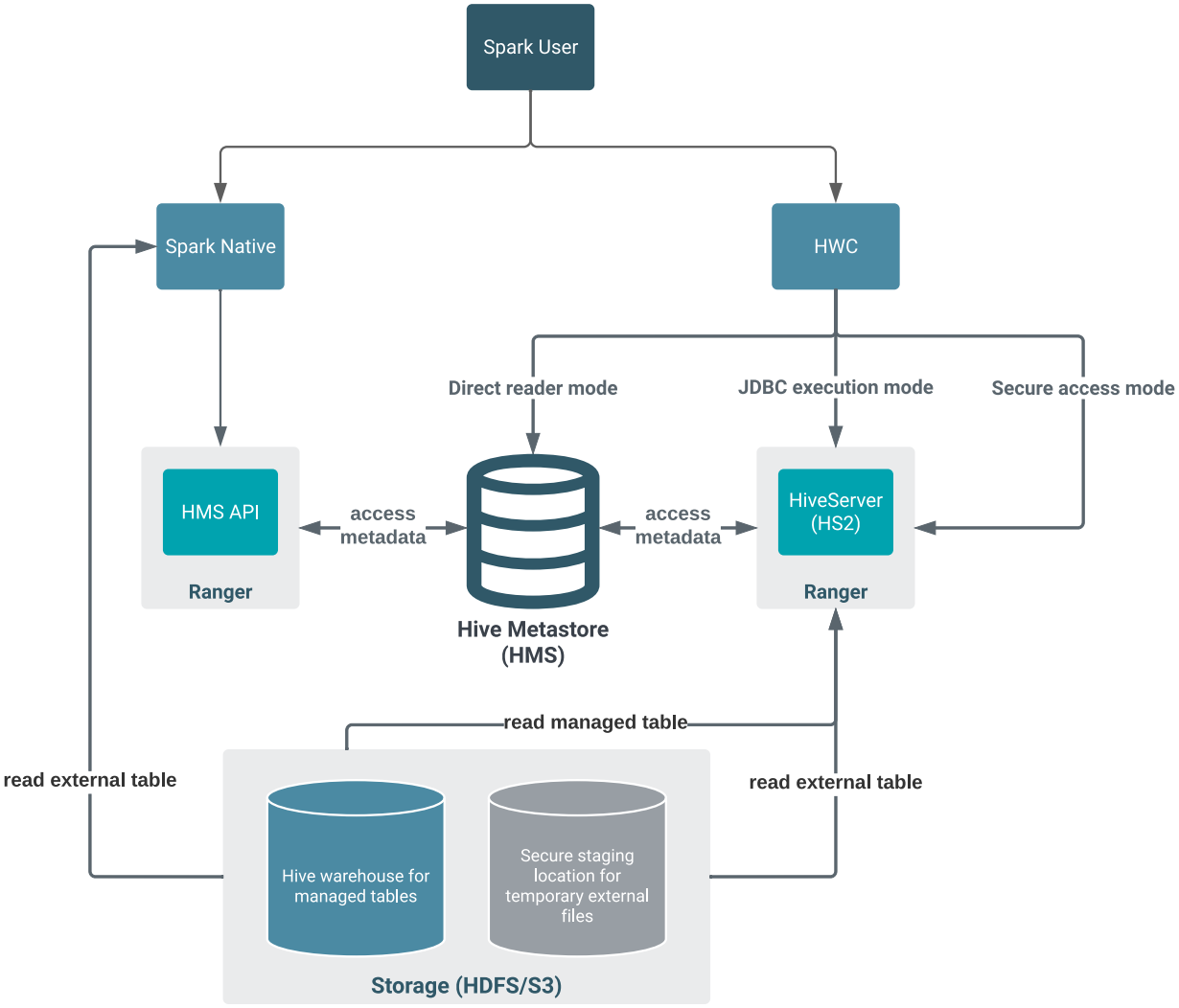
These read configuration options require connections to different Hive components:

- Direct Reader configuration: Connects to Hive Metastore (HMS)
- JDBC configuration: Connects to HiveServer (HS2)
- Secure Access configuration: Connects to HiveServer (HS2)

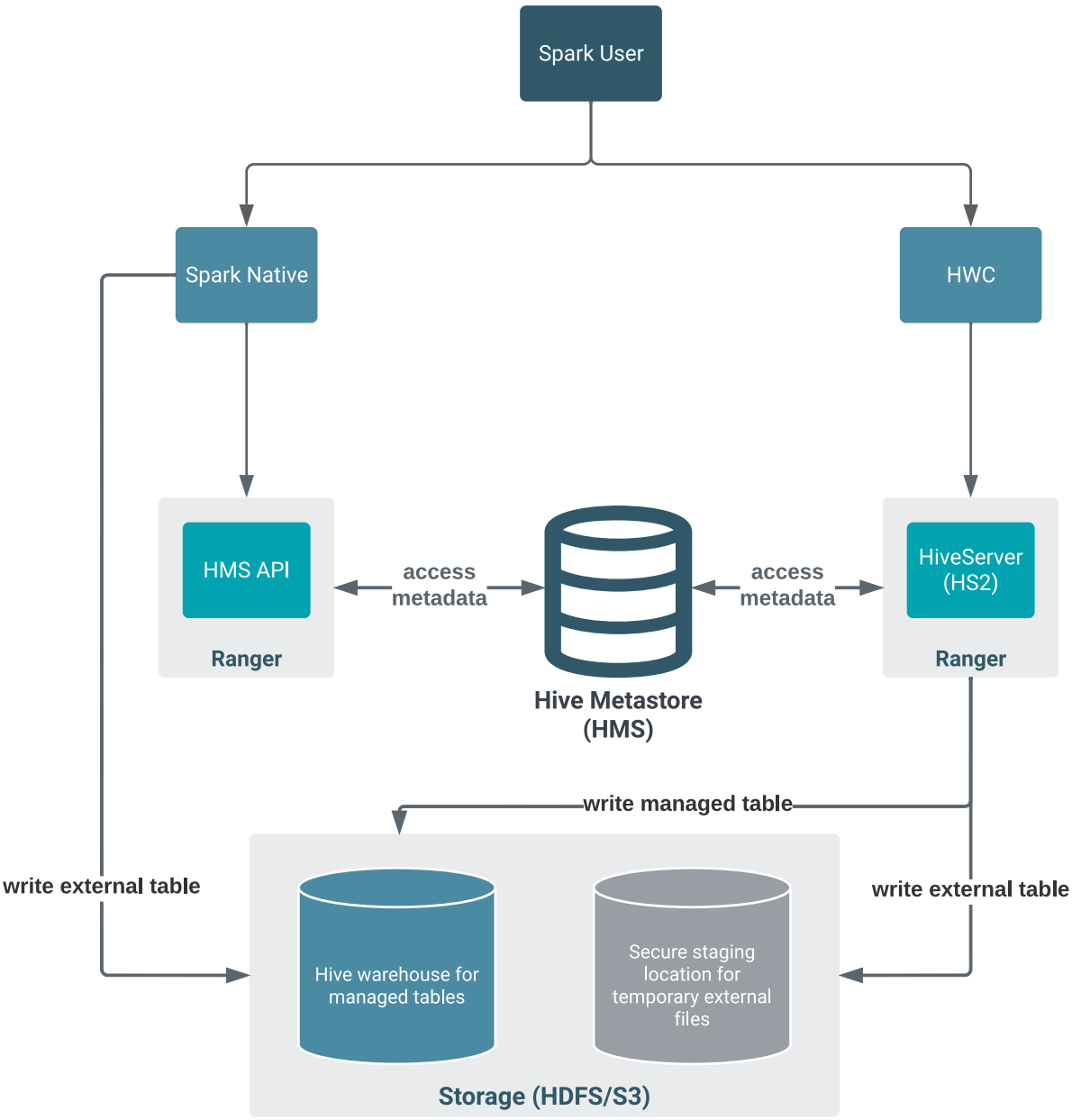
Ranger authorizes access to Hive tables from Spark through HiveServer (HS2) or the Hive metastore API (HMS API).

To write ACID managed tables from Spark to Hive, you must use HWC. To write external tables from Spark to Hive, you can use native Spark or HWC.

The following diagram shows the typical read authorization process:



The following diagram shows the typical write authorization process:



When writing, HWC always enforces authorization through HiveServer (HS2). Reading managed tables in JDBC mode or Secure access mode enforces Ranger authorization, including fine-grained features, such as row masking and column mapping. In Direct Reader mode, the Ranger and HMS integration provides authorization.

External table queries go through the HMS API, which is also integrated with Ranger. If you do not use HWC, the Hive metastore (HMS) API, integrated with Ranger, authorizes external table access. HMS API-Ranger integration enforces the Ranger Hive ACL in this case. When you use HWC, queries such as DROP TABLE affect file system data as well as metadata in HMS.

Using the Direct Reader option, SparkSQL queries read managed table metadata directly from the HMS, but only if you have permission to access files on the file system. You cannot write to managed tables using the Direct Reader option.

Using the Secure access mode, you can enable fine-grained access control (FGAC) column masking and row filtering to secure managed (ACID), or even external, Hive table data that you read from Spark.

Managed table authorization

A Spark job impersonates the end user when attempting to access an Apache Hive managed table. As an end user, you do not have permission to access, managed files in the Hive warehouse. Managed tables have default file system permissions that disallow end user access, including Spark user access.

As Administrator, you set permissions in Ranger to access the managed tables when you configure HWC for JDBC mode or Secure access mode reads. You can fine-tune Ranger to protect specific data. For example, you can mask data in certain columns, or set up tag-based access control.

When you configure HWC for Direct Reader mode, you cannot use Ranger in this way. You must set read access to the file system location for managed tables. You must have Read and Execute permissions on the Hive warehouse location (hive.metastore.warehouse.dir).

External table authorization

Ranger authorization of external table reads and writes is supported. You need to configure a few properties in Cloudera Manager for authorization of external table writes. You must be granted file system permissions on external table files to allow Spark direct access to the actual table data instead of just the table metadata.

Direct Reader Authorization Limitation

As Spark allows users to run arbitrary code, Ranger fine grained access control, such as row level filtering or column level masking, is not possible within Spark itself. This limitation extends to data read using Direct Reader.

To restrict data access at a fine-grained level, use a read option that supports Ranger. Only consider using the Direct Reader option to read Hive data from Spark if you do not require fine-grained access. For example, use Direct Reader for ETL use cases.

Hive authentication

HiveServer supports authentication of clients using Kerberos or user/password validation backed by LDAP.

If you configure HiveServer to use Kerberos authentication, HiveServer acquires a Kerberos ticket during startup. HiveServer requires a principal and keytab file specified in the configuration. Client applications (for example, JDBC or Beeline) must have a valid Kerberos ticket before initiating a connection to HiveServer2. JDBC-based clients must include principal=<hive.server2.authentication.principal> in the JDBC connection string. For example:

```
String url = "jdbc:hive2://node1:10000/default;principal=hive/HiveServerHost
@YOUR-REALM.COM"
Connection con = DriverManager.getConnection(url);
```

where hive is the principal configured in hive-site.xml and HiveServerHost is the host where HiveServer is running.

To start Beeline and connect to a secure HiveServer, enter a command as shown in the following example:

```
beeline -u "jdbc:hive2://10.65.13.98:10000/default;principal=hive/_HOST@CLOU
DERA.SITE"
```

Securing HiveServer using LDAP

You can secure the remote client connection to Hive by configuring HiveServer to use authentication with LDAP.

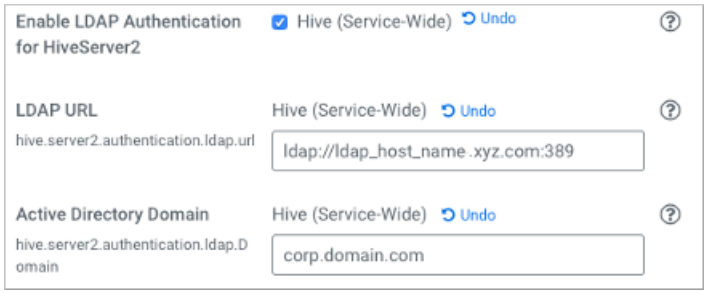
About this task

When you configure HiveServer to use user and password validation backed by LDAP, the Hive client sends a username and password during connection initiation. HiveServer validates these credentials using an external LDAP service. You can enable LDAP Authentication with HiveServer using Active Directory or OpenLDAP.

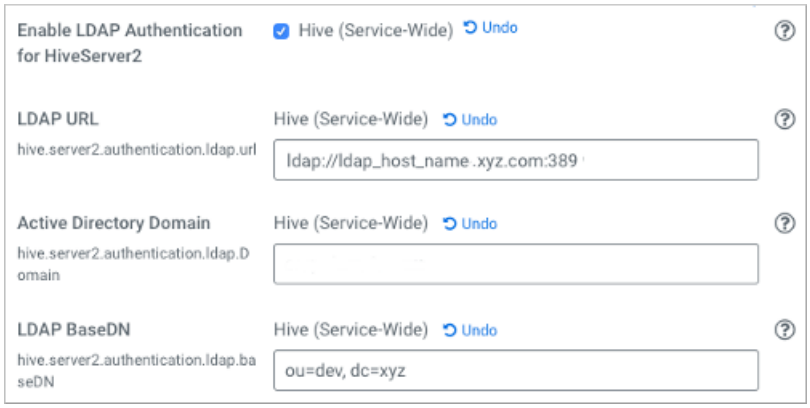
Procedure

1. In Cloudera Manager, select Hive-on-Tez Configuration .
2. Search for ldap.
3. Check Enable LDAP Authentication for HiveServer2 for Hive (Service Wide).
4. Enter your LDAP URL in the format ldap[s]://<host>:<port>.
   LDAP_URL is the access URL for your LDAP server. For example, ldap://ldap_host_name.xyz.com:389
5. Enter the Active Directory Domain or LDAP Base DN for your environment.
   - Active Directory (AD)
   - LDAP_BaseDN

   Enter the domain name of the AD server. For example, corp.domain.com.



   Enter the base LDAP distinguished name (DN) for your LDAP server. For example, ou=dev, dc=xyz.



6. Click Save Changes.
7. Restart the Hive service.
8. Construct the LDAP connection string to connect to HiveServer.
   The following simple example is insecure because it sends clear text passwords.

```
String URL = "jdbc:hive2://node1:10000/default;user=LDAP_Userid;password
=LDAP_Password"
Connection con = DriverManager.getConnection(url);
```

   The following example shows a secure connection string that uses encrypted passwords.

```
String url ="jdbc:hive2://node1:10000/default;ssl=true;sslTrustStore=/my
truststore_path;trustStorePassword=my_truststore_password"
Connection con = DriverManager.getConnection(url);
```

   For information about encrypting communication, see links below.

Related Information

Custom Configuration (about Cloudera Manager Safety Valve)

Client connections to HiveServer

You can use Beeline, a JDBC, or an ODBC connection to HiveServer.

JDBC Client-HiveServer Authentication

The JDBC client requires a connection URL as shown below. JDBC-based clients must include a user name and password in the JDBC connection string. For example:

```
String url = "jdbc:hive2://node1:10000/default;user=LDAP_Userid;password=LDA
P_Password" Connection con = DriverManager.getConnection(url);
```

where the LDAP_Userid value is the user ID and LDAP_Password is the password of the client user.

HiveServer modes of operation

HiveServer supports the following modes for interacting with Hive:

Operating Mode	Description
Embedded	The Beeline client and the Hive installation reside on the same host machine or virtual machine. No TCP connectivity is required.
Remote	Use remote mode to support multiple, concurrent clients executing queries against the same remote Hive installation. Remote transport mode supports authentication with LDAP and Kerberos. It also supports encryption with SSL. TCP connectivity is required.

Remote mode: Launch Hive using the following URL:

```
jdbc:hive2://<host>:<port>/<db>.
```

The default HiveServer2 port is 10000.

Embedded mode: Launch Hive using the following URL:

```
jdbc:hive2://
```

Transport Modes

As administrator, you can start HiveServer in one of the following transport modes:

Transport Mode	Description
TCP	HiveServer uses TCP transport for sending and receiving Thrift RPC messages.
HTTP	HiveServer uses HTTP transport for sending and receiving Thrift RPC messages.

Pluggable Authentication Modules in HiveServer

While running in TCP transport mode, HiveServer supports Pluggable Authentication Modules (PAM). Using Pluggable Authentication Modules, you can integrate multiple authentication schemes into a single API. You use the Cloudera Manager Safety Valve technique on HIVE_ON_TEZ-1 Configuration to set the following properties:

- hive.server2.authentication

  Value = CUSTOM
- hive.server2.custom.authentication.class

Value = <the pluggable auth class name>

The class you provide must be a proper implementation of the org.apache.hive.service.auth.PasswdAuthenticationProvider. HiveServer calls its Authenticate(user, passed) method to authenticate requests. The implementation can optionally extend the Hadoop's org.apache.hadoop.conf.Configured class to grab the Hive Configuration object.

HiveServer Trusted Delegation

HiveServer determines the identity of the connecting user from the authentication subsystem (Kerberos or LDAP). Any new session started for this connection runs on behalf of this connecting user. If the server is configured to proxy the user, the identity of the connecting user is used to connect to Hive. Users with Hadoop superuser privileges can request an alternate user for the given session. HiveServer checks that the connecting user can proxy the requested userid, and if so, runs the new session as the alternate user.

Pluggable authentication modules in HiveServer

While running in TCP transport mode, HiveServer supports Pluggable Authentication Modules (PAM). Using Pluggable Authentication Modules, you can integrate multiple authentication schemes into a single API.

You use the Cloudera Manager Safety Valve technique on HIVE_ON_TEZ-1 Configuration to set the following properties:

- hive.server2.authentication

  Value = CUSTOM
- hive.server2.custom.authentication.class

  Value = <the pluggable auth class name>

The class you provide must be a proper implementation of the org.apache.hive.service.auth.PasswdAuthenticationProvider. HiveServer calls its Authenticate(user, passed) method to authenticate requests. The implementation can optionally extend the Hadoop's org.apache.hadoop.conf.Configured class to grab the Hive Configuration object.

JDBC connection string syntax

The JDBC connection string for connecting to a remote Hive client requires a host, port, and Hive database name. You can optionally specify a transport type and authentication.

jdbc:hive2://<host>:<port>/<dbName>;<sessionConfs>?<hiveConfs>#<hiveVars>

Connection string parameters

The following table describes the parameters for specifying the JDBC connection.

JDBC Parameter	Description	Required
host	The cluster node hosting HiveServer.	yes
port	The port number to which HiveServer listens.	yes
dbName	The name of the Hive database to run the query against.	yes
sessionConfs	Optional configuration parameters for the JDBC/ODBC driver in the following format: <key1>=<value1>;<key2>=<key2>...;	no

JDBC Parameter	Description	Required
hiveConfs	Optional configuration parameters for Hive on the server in the following format: <key1>=<value1>;<key2>=<key2>; ... The configurations last for the duration of the user session.	no
hiveVars	Optional configuration parameters for Hive variables in the following format: <key1>=<value1>;<key2>=<key2>; ... The configurations last for the duration of the user session.	no

TCP and HTTP Transport

The following table shows variables for use in the connection string when you configure HiveServer. The JDBC client and HiveServer can use either HTTP or TCP-based transport to exchange RPC messages. Because the default transport is TCP, there is no need to specify transportMode=binary if TCP transport is desired.

transportMode Variable Value	Description
http	Connect to HiveServer2 using HTTP transport.
binary	Connect to HiveServer2 using TCP transport.

The syntax for using these parameters is:

```
jdbc:hive2://<host>:<port>/<dbName>;transportMode=http;httpPath=<http_endpoi
nt>; \
  <otherSessionConfs>?<hiveConfs>#<hiveVars>
```

User Authentication

If configured in remote mode, HiveServer supports Kerberos, LDAP, Pluggable Authentication Modules (PAM), and custom plugins for authenticating the JDBC user connecting to HiveServer. The format of the JDBC connection URL for authentication with Kerberos differs from the format for other authentication models. The following table shows the variables for Kerberos authentication.

User Authentication Variable	Description
principal	A string that uniquely identifies a Kerberos user.
saslQop	Quality of protection for the SASL framework. The level of quality is negotiated between the client and server during authentication. Used by Kerberos authentication with TCP transport.
user	Username for non-Kerberos authentication model.
password	Password for non-Kerberos authentication model.

The syntax for using these parameters is:

```
jdbc:hive://<host>:<port>/<dbName>;principal=<HiveServer2_kerberos_principal
>;<otherSessionConfs>?<hiveConfs>#<hiveVars>
```

Transport Layer Security

HiveServer2 supports SSL and Sasl QOP for transport-layer security. The format of the JDBC connection string for SSL uses these variables:

SSL Variable	Description
ssl	Specifies whether to use SSL
sslTrustStore	The path to the SSL TrustStore.
trustStorePassword	The password to the SSL TrustStore.

The syntax for using the authentication parameters is:

```
jdbc:hive2://<host>:<port>/<dbName>; \
ssl=true;sslTrustStore=<ssl_truststore_path>;trustStorePassword=<truststo
re_password>; \
<otherSessionConfs>?<hiveConfs>#<hiveVars>
```

Tip: To avoid passing the truststore password in the Beeline connection string, import the truststore certificate into the default Java truststore (jssecacerts or cacerts) on the client. You must import the certificate on all nodes. In this case, you do not need to specify sslTrustStore and trustStorePassword in the Beeline connection string.

When using TCP for transport and Kerberos for security, HiveServer2 uses Sasl QOP for encryption rather than SSL.

Sasl QOP Variable	Description
principal	A string that uniquely identifies a Kerberos user.
saslQop	The level of protection desired. For authentication, checksum, and encryption, specify auth-conf. The other valid values do not provide encryption.

The JDBC connection string for Sasl QOP uses these variables.

```
jdbc:hive2://FQDN.EXAMPLE.COM:10000/default;principal=hive/_H
OST@EXAMPLE.COM;saslQop=auth-conf
```

The _HOST is a wildcard placeholder that gets automatically replaced with the fully qualified domain name (FQDN) of the server running the HiveServer daemon process.

Configuring ZooKeeper SASL authentication for Hive clients

Learn about how to configure Hive clients to authenticate with a ZooKeeper ensemble that enforces Simple Authentication and Security Layer (SASL).

ZooKeeper client authentication

When a ZooKeeper ensemble is configured to enforce SASL authentication, it ensures that only authenticated clients can establish a session. Without this enforcement, the ZooKeeper server remains open to anonymous client connections.

In Kerberized environments, Hive components such as HiveServer2 (HS2) and the Metastore use ZooKeeper for service discovery, high availability, and locking. While server-side components like HS2 can easily authenticate using a predefined principal and keytab, client-side discovery (such as JDBC or Metastore clients) requires specific configurations to handle different Kerberos login methods, including keytabs or the local ticket cache.

Hive ZooKeeper authentication properties

The following table describes the properties used to manage ZooKeeper authentication for Hive components:

Property	Description
hive.zookeeper.kerberos.enabled	Specifies whether hive beeline/JDBC HS2 uses Kerberos to secure the ZooKeeper connection. This property is evaluated as true only when HS2 is Kerberized.

Configuring Hive clients for SASL-enforced ZooKeeper

To enable successful authentication when a client connects to a SASL-enforced ZooKeeper ensemble, perform the following:

1. Authenticate by using `kinit` if you are using the ticket cache before running the Hive client.

Communication encryption

Encryption between HiveServer2 and its clients is independent from Kerberos authentication. HiveServer supports the following types of encryption between the service and its clients (Beeline, JDBC/ODBC):

- SASL (Simple Authentication and Security Layer)
- TLS/SSL (Transport Layer Security/Secure Sockets Layer)

TLS/SSL requires certificates. SASL QOP encryption does not. SASL QOP is aimed at protecting core Hadoop RPC communications. SASL QOP might cause performance problems when handling large amounts of data. You can configure HiveServer to support TLS/SSL connections from JDBC/ODBC clients using Cloudera Manager.

Client connections to HiveServer2 over TLS/SSL

A client connecting to a HiveServer2 over TLS/SSL must access the trust store on HiveServer to establish a chain of trust and verify server certificate authenticity. The trust store is typically not password protected. The trust store might be password protected to prevent its contents from being modified. However, password protected trust stores can be read from without using the password.

The client needs the path to the trust store when attempting to connect to HiveServer2 using TLS/SSL. You can specify the trust store in one of the following ways:

- Pass the path to the trust store each time you connect to HiveServer in the JDBC connection string:

```
jdbc:hive2://fqdn.example.com:10000/default;ssl=true;\
 sslTrustStore=$JAVA_HOME/jre/lib/security/jssecacerts;trustStorePassword
=extraneous
```

- Set the path to the trust store one time in the Java system javax.net.ssl.trustStore property:

```
java -Djavax.net.ssl.trustStore=/usr/java/jdk1.8.0_141-cloudera/jre/lib/
security/jssecacerts \
 -Djavax.net.ssl.trustStorePassword=extraneous MyClass \
 jdbc:hive2://fqdn.example.com:10000/default;ssl=true
```

Enabling TLS/SSL for HiveServer

You can secure client-server communications using symmetric-key encryption in the TLS/SSL (Transport Layer Security/Secure Sockets Layer) protocol. To encrypt data exchanged between HiveServer and its clients, you can use Cloudera Manager to configure TLS/SSL.

Before you begin

- HiveServer has the necessary server key, certificate, keystore, and trust store set up on the host system.

- The hostname variable ($(hostname -f)-server.jks) is used with Java keytool commands to create keystore, as shown in this example:

```
$ sudo keytool -genkeypair -alias $(hostname -f)-server -keyalg RSA -key
store \
    /opt/cloudera/security/pki/$(hostname -f)-server.jks -keysize 2048 -
dname \
    "CN=$(hostname -f),OU=DEPT-NAME-OPTIONAL,O=COMPANY-
NAME,L=CITY,ST=STATE,C=TWO-DIGIT-NATION" \
    -storepass PASSWORD -keypass PASSWORD
```

About this task

On the beeline command line, the JDBC URL requirements include specifying ssl=true;sslTrustStore=<path_to_trustore>. Truststore password requirements depend on the version of Java running in the cluster:

- Java 11: the truststore format has changed to PKCS and the truststore password is required; otherwise, the connection fails.
- Java 8: The trust store password does not need to be specified.

Procedure

1. In Cloudera Manager, navigate to Clusters Hive Configuration .
2. In Filters, select HIVE for the scope.
3. Select Security for the category.
4. Accept the default Enable TLS/SSL for HiveServer2, which is checked for Hive (Service-Wide).
5. Enter the path to the Java keystore on the host system.
   /opt/cloudera/security/pki/*KEYSTORE_NAME*.jks
6. Enter the password for the keystore you used on the Java keytool command-line when the key and keystore were created.

   The password for the keystore must match the password for the key.
7. Enter the path to the Java trust store on the host system.
8. Click Save Changes.
9. Restart the Hive service.
10. Construct a connection string for encrypting communications using TLS/SSL.

```
jdbc:hive2://#<host>:#<port>/#<dbName>;ssl=true;sslTrustStore=#<ssl_trus
tstore_path>; \
trustStorePassword=#<truststore_password>;#<otherSessionConfs>?#<hiveCon
fs>#<hiveVars>
```

Enabling SASL in HiveServer

You can provide a Quality of Protection (QOP) that is higher than the cluster-wide default using SASL (Simple Authentication and Security Layer).

About this task

HiveServer2 by default uses hadoop.rpc.protection for its QOP value. Setting hadoop.rpc.protection to a higher level than HiveServer (HS2) does not usually make sense. HiveServer ignores hadoop.rpc.protection in favor of hive.server2.thrift.sasl.qop.

You can determine the value of hadoop.rpc.protection: In Cloudera Manager, click Clusters HDFS Configuration Hadoop , and search for hadoop.rpc.protection.

If you want to provide a higher QOP than the default, set one of the SASL Quality of Protection (QOP) levels as shown in the following table:

auth	Default. Authentication only.
auth-int	Authentication with integrity protection. Signed message digests (checksums) verify the integrity of messages sent between client and server.
auth-conf	Authentication with confidentiality (transport-layer encryption) and integrity. Applicable only if HiveServer is configured to use Kerberos authentication.

Procedure

1. In Cloudera Manager, navigate to Clusters Hive Configuration .
2. In HiveServer2 Advanced Configuration Snippet (Safety Valve) for hive-site click + to add a property and value.
3. Specify the QOP auth-conf setting for the SASL QOP property.
   For example,

   Name:hive.server2.thrift.sasl.qop

   Value: auth-conf
4. Click Save Changes.
5. Restart the Hive service.
6. Construct a connection string for encrypting communications using SASL.

   ```
   jdbc:hive2://fqdn.example.com:10000/default;principal=hive/_HOST@EXAMPLE
   .COM;saslqop=auth-conf
   ```

   The _HOST is a wildcard placeholder that gets automatically replaced with the fully qualified domain name (FQDN) of the server running the HiveServer daemon process.

Securing an endpoint under AutoTLS

The default cluster configuration for HiveServer (HS2) with AutoTLS secures the HS2 WebUI Port, but not the JDBC/ODBC endpoint.

About this task

The default cluster configuration for HS2 with AutoTLS will secure the HS2 Server WebUI Port, but not the JDBC/ODBC endpoint.

Assumptions:

- Auto-TLS Self-signed Certificates.
- Proper CA Root certs eliminate the need for any of the following truststore actions.

When HS2 TLS is enabled hive.server2.use.SSL=true, the auto-connect feature on gateway servers is not supported. The auto-connect feature uses /etc/hive/conf/beeline-site.xml to automatically connect to Cloudera Manager controlled HS2 services. Also, with hive.server2.use.SSL=true, ZooKeeper discovery mode is not supported because the HS2 reference stored in ZooKeeper does not include the ssl=true and other TLS truststore references (self-signed) needed to connect with TLS.

The beeline-site.xml file managed for gateways doesn't not include ssl=true or a reference to a truststore that includes a CA for the self-signed TLS certificate used by ZooKeeper or HiveServer.

The best practice, under the default configuration, is to have all external clients connect to Hive (JDBC/ODBC) through the Apache Knox proxy. With TLS enabled via Auto-TLS with a self-signed cert, you can use the jks file downloaded from Knox as the client trusted CA for the Knox host. That cert will only work for KNOX. And since KNOX and HS2 TLS server certs are from the same CA, Knox connects without adjustments.

To connect through Knox:

Procedure

1. Configure the HS2 transport mode as http to support the Knox proxy interface.

```
jdbc:hive2://<host>:8443/;ssl=true;\
transportMode=http;httpPath=gateway/cdp-proxy-api/hive;\
...
```

   The TLS Public Certificate in <path>/bin/certs/gateway-client-trust.jks will not work.
2. Build a TLS Public Certificate from the self-signed root CA used for the cluster in Cloudera Manager.

```
keytool -import -v -trustcacerts -alias home90-ca -file \
/var/lib/cloudera-scm-agent/agent-cert/cm-auto-global_cacerts.pem \
-keystore <my_cacert>.jks
```

3. Connect to HS2 on the Beeline command line, using the -u option.

```
hive -u jdbc:hive2://<HS2 host>:10001/default;ssl=true;\
transportMode=http;httpPath=cliservice;\
principal=hive/_HOST@<realm>;user=<user name>;\
sslTrustStore=<path>/certs/home90_cacert.jks;\
trustStorePassword=changeit
```

   The httpPath default is configured in Cloudera Manager. The sslTrustStore is required is you are using a self-signed certificate.

Securing Hive metastore

Cloudera recommends using Apache Ranger policies to secure Hive data in Hive metastore. You need to perform a few actions to prevent users from bypassing HiveServer to access the Hive metastore and the Hive metastore database.

Procedure

1. Add a firewall rule on the metastore service host to allow access to the metastore port only from the HiveServer2 host. You can do this using iptables.
2. Grant access to the metastore database only from the metastore service host.
   For example, in MySQL: GRANT ALL PRIVILEGES ON metastore.* TO 'hive'@'metastorehost'; where metastorehost is the host where the metastore service is running.
3. Make sure users who are not administrators cannot log into the HiveServer host.

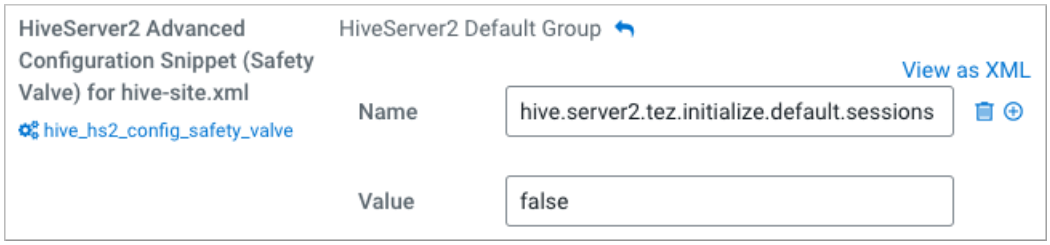
Activating the Hive web UI

The HiveServer2 GUI/ web UI does not display active client connections after enabling Kerberos, which leads to a Kerberos ticket not being issued for a browser client.

HiveServer2 GUI/ web UI does not display active client connections after enabling Kerberos when SPENGO authentication is disabled.

Procedure

1. In Cloudera Manager, go to Clusters Hive-on-Tez Configuration .
2. Search for HiveServer2 Advanced Configuration Snippet (Safety valve) for hive-site.xml



3. Click ⊕ and add the property hive.server2.webui.spnego.keytab and value hive.keytab
4. Click ⊕ and add the property hive.server2.webui.spnego.principal and value HTTP/_HOST@[***REALM NAME***]
5. Click ⊕ and add the property hive.server2.webui.use.spnegoand value true
6. Click ⊕ and add the property hive.users.in.admin.roleand value *[***USERNAME1,USERNAME2,…***]*

   Note: [***USERNAME1,USERNAME2,…***] is the list of comma separated users who want to access historic query detail from the web UI.
7. Save changes, and restart Hive-on-Tez.

   The Hive Web UI shows active client connections.