Cloudera Runtime 7.2.11

Migrating Data Using Sqoop

Date published: 2019-08-21
Date modified: 2021-09-08



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

Data migration to Apache Hive 4

Setting Up Sqoop 4

Atlas Hook for Sqoop 5

Sqoop enhancements to the Hive import process 6
Configuring custom Beeline arguments 6
Configuring custom Hive JDBC arguments 8
Configuring a custom Hive CREATE TABLE statement 10
Configuring custom Hive table properties 11

Secure options to provide Hive password during a Sqoop import 12
Providing the Hive password through a prompt 12
Providing the Hive password through a file 13
Providing the Hive password through an alias 14
Providing the Hive password through an alias in a file 15
Saving the password to Hive Metastore 16

Imports into Hive 17
Creating a Sqoop import command 17
Importing RDBMS data into Hive 20

HDFS to Apache Hive data migration 20
Importing RDBMS data to HDFS 20
Converting an HDFS file to ORC 23
Incrementally updating an imported table 24

Import command options 25

Data migration to Apache Hive

To migrate data from an RDBMS, such as MySQL, to Hive, you should consider Apache Sqoop in Cloudera with the Teradata connector. Apache Sqoop client CLI-based tool transfers data in bulk between relational databases and HDFS or cloud object stores including Amazon S3 and Microsoft ADLS.

The source of legacy system data that needs to undergo an extract, transform, and load (ETL) process typically resides on the file system or object store. You can also import data in delimited text (default) or SequenceFile format, and then convert data to ORC format recommended for Hive. Generally, for querying the data in Hive, ORC is the preferred format because of the performance enhancements ORC provides.

Teradata Connector for Sqoop

Cloudera does not support the Sqoop exports using the Hadoop jar command (the Java API). The connector documentation from Teradata includes instructions that include the use of this API. Cloudera users have reportedly mistaken the unsupported API commands, such as -forcestage, for the supported Sqoop commands, such as –-stagi ng-force. Cloudera supports the use of Sqoop only with commands documented in Using the Cloudera Connector Powered by Teradata. Cloudera does not support using Sqoop with Hadoop jar commands, such as those described in the Teradata Connector for Hadoop Tutorial.

Apache Sqoop documentation on the Cloudera web site

To access the latest Sqoop documentation, go to Sqoop Documentation 1.4.7.7.1.6.0.

Setting Up Sqoop

Cloudera Runtime includes the Sqoop Client for bulk importing and exporting data from diverse data sources to Hive. You learn how to install the RDBMS connector and Sqoop Client in Cloudera.

Procedure

1. In Cloudera Manager, in Clusters, select Add Service from the options menu.
2. Select the Sqoop Client and click Continue.
3. Choose a JDBC database driver, depending on the data source of the source or destination for a Sqoop import or export, respectively.
4. Install the JDBC database driver in /var/lib/sqoop on the Sqoop node.

   Do not install the/opt/cloudera/parcels/CDH because upgrades modify this directory.

   - MySQL: Download the MySQL driver https://dev.mysql.com/downloads/connector/j/ to /var/lib/sqoop, and then run tar -xvzf mysql-connector-java-<version>.tar.gz.
   - Oracle: Download the driver from https://www.oracle.com/database/technologies/appdev/jdbc-downloads.html and put it in /var/lib/sqoop.
   - PostgreSQL: Download the driver from https://jdbc.postgresql.org/download/ and put it in /var/lib/sqoop.
5. In Cloudera Manager, click Actions Deploy Client Configuration .

What to do next

After setting up the Sqoop client, you can enter Sqoop commands using the following connection string, depending on your data source.

Note:

Sqoop Netezza connector is not supported. If you want to use Netezza, you can request access to the Netezza connector source code repository and modify it to meet your requirements. To obtain access, contact your organization's legal representative to start the approval process.

- MySQL Syntax:

  ```
  jdbc:mysql://<HOST>:<PORT>/<DATABASE_NAME>
  ```

  Example:

  ```
  jdbc:mysql://my_mysql_server_hostname:3306/my_database_name
  ```

- Oracle Syntax:

  ```
  jdbc:oracle:thin:@<HOST>:<PORT>:<DATABASE_NAME>
  ```

  Example:

  ```
  jdbc:oracle:thin:@my_oracle_server_hostname:1521:my_database_name
  ```

- PostgreSQL Syntax:

  ```
  jdbc:postgresql://<HOST>:<PORT>/<DATABASE_NAME>
  ```

  Example:

  ```
  jdbc:postgresql://my_postgres_server_hostname:5432/my_database_name
  ```

- Netezza Syntax:

  ```
  jdbc:netezza://<HOST>:<PORT>/<DATABASE_NAME>
  ```

  Example:

  ```
  jdbc:netezza://my_netezza_server_hostname:5480/my_database_name
  ```

- Teradata Syntax:

  ```
  jdbc:teradata://<HOST>/DBS_PORT=1025/DATABASE=<DATABASE_NAME>
  ```

  Example:

  ```
  jdbc:teradata://my_teradata_server_hostname/DBS_PORT=1025/DATABASE=my_da
  tabase_name
  ```

Atlas Hook for Sqoop

Cloudera Manager configures the Atlas Hook for Sqoop automatically, but if you are upgrading, you need to know how to enable the Hook.

Cloudera Manager takes care of setting up the Atlas Hook for Sqoop in the following ways:

- Includes the required Atlas JARS in the correct location in your cluster.
- Configures the Sqoop Atlas Hook in sqoop-site.xml.
- Generates the atlas-application.properties file for Sqoop.
- If you are installing a fresh cluster that includes Atlas, Cloudera Manager enables the Sqoop Atlas Hook.

If you are upgrading from an older cluster, you need to enable the Hook manually.

Enabling the Sqoop Atlas Hook after upgrading

The upgrade process does not enable the Atlas Hook. To enable the hook:

1. Go to Sqoop's configuration page.
2. Search for Atlas.
3. Check the checkbox.
4. Re-deploy the client configurations using Cloudera Manager.

Related Information

Apache Sqoop Documentation (v1.4.7.7.1.6)
Sqoop Atlas Hook

Sqoop enhancements to the Hive import process

Learn how you can leverage the various Sqoop enhancements that enable you to configure how Sqoop imports data from relational databases into Hive.

Sqoop had limited capabilities in executing Hive processes that often resulted in a lack of control of the imported data and the corresponding Hive table properties. With the enhancements, you can now specify custom Beeline arguments, define custom Hive JDBC arguments, choose how tables are created in Hive using custom CREATE TABLE statements, and configure custom Hive table properties. The changes allow users to control the imported data according to their specific requirements.

In addition, the enhancements also address the variability across Hive versions and their corresponding table creation semantics. Depending on the Hive version, a CREATE TABLE statement can have different implications, such as creating an external table with purge or creating a managed table. Also, the ability to set custom configurations globally not only streamlines the import process but also simplifies the modification of a large number of Sqoop commands.

Configuring custom Beeline arguments

Learn how to configure Sqoop to enable users to include additional Beeline arguments while importing data into Hive using Sqoop.

About this task

You can configure the required property either through Cloudera Manager or by using the --beeline-args argument in your Sqoop import command.

Order of precedence

The configuration set through the Sqoop argument in the command line takes precedence over the configuration specified through Cloudera Manager. If custom Beeline arguments are specified through both the command line and Cloudera Manager, the command line Sqoop argument overwrites the Cloudera Manager configuration.

Limitations

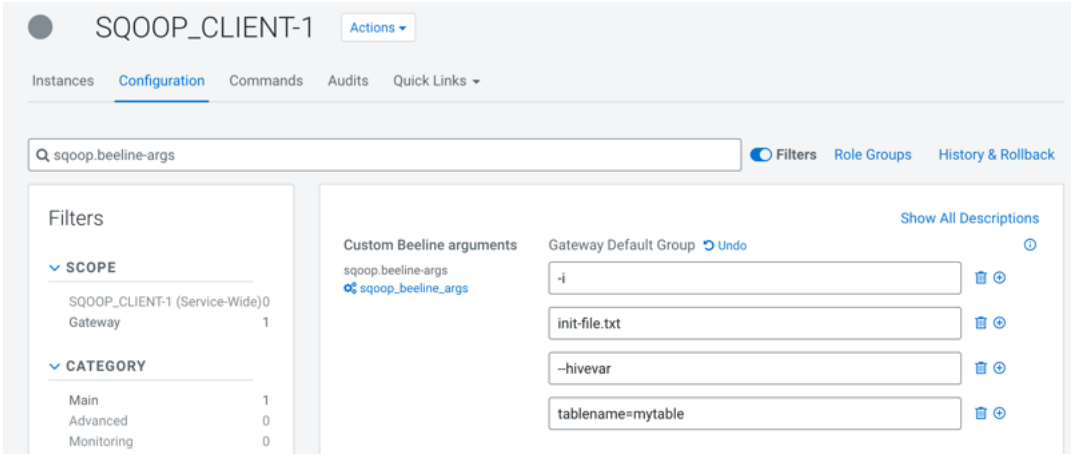
You must be aware of certain Beeline arguments that cannot be specified. This limitation ensures the integrity and proper functioning of Sqoop's core processes. The following Beeline arguments are not supported:

- -e
- -f
- -n
- -p

- -w
- --password-file
- -a (if the HADOOP_TOKEN_FILE_LOCATION is not specified)

Procedure

1. If you are specifying the custom Beeline arguments through Cloudera Manager, perform the following steps:
   a) In Cloudera Manager, click Clusters and then select the SQOOP_CLIENT-1 service.
   b) From the Sqoop service, go to the **Configuration** tab and search for sqoop.beeline-args.
   c) Click ⊕ and specify the custom Beeline argument and its value.



Important: Each argument and value must be added in a new line using ⊕.

   d) Click Save Changes.
2. If you are specifying the custom Beeline arguments through the Sqoop argument, specify the required argument using the --beeline-args argument while constructing the Sqoop import command.

```
/opt/cloudera/parcels/CDH/bin/sqoop import \
  -Dsqoop.beeline.env.preserve=KRB5CCNAME \
  --connection-manager org.apache.sqoop.manager.MySQLManager \
  --connect jdbc:mysql://db.foo.com:3306/employees \
  --username [***USERNAME***] \
  --password [***PASSWORD***] \
  --table employees \
  --warehouse-dir /user/hrt_qa/test-sqoop \
  --hive-import \
  --delete-target-dir \
  --hive-overwrite \
  --beeline-args -i init_file2.txt --hivevar tablename=mytable2 -r
```

Important: Every argument that you provide must line up following the --beeline-args key.

Note: As part of the upgrade, Sqoop imports into Hive using Beeline instead of the Hive CLI. This change requires that Ranger policies be in place for the target-dir input location, as Cloudera mandates explicit permissions on the target directory to ensure successful data import.

Configuring custom Hive JDBC arguments

Learn how to configure Sqoop to enable users to specify custom Hive JDBC arguments while importing data into Hive using Sqoop. This allows users to set Hive session variables, Hive configuration values, and Hive user variables.

About this task

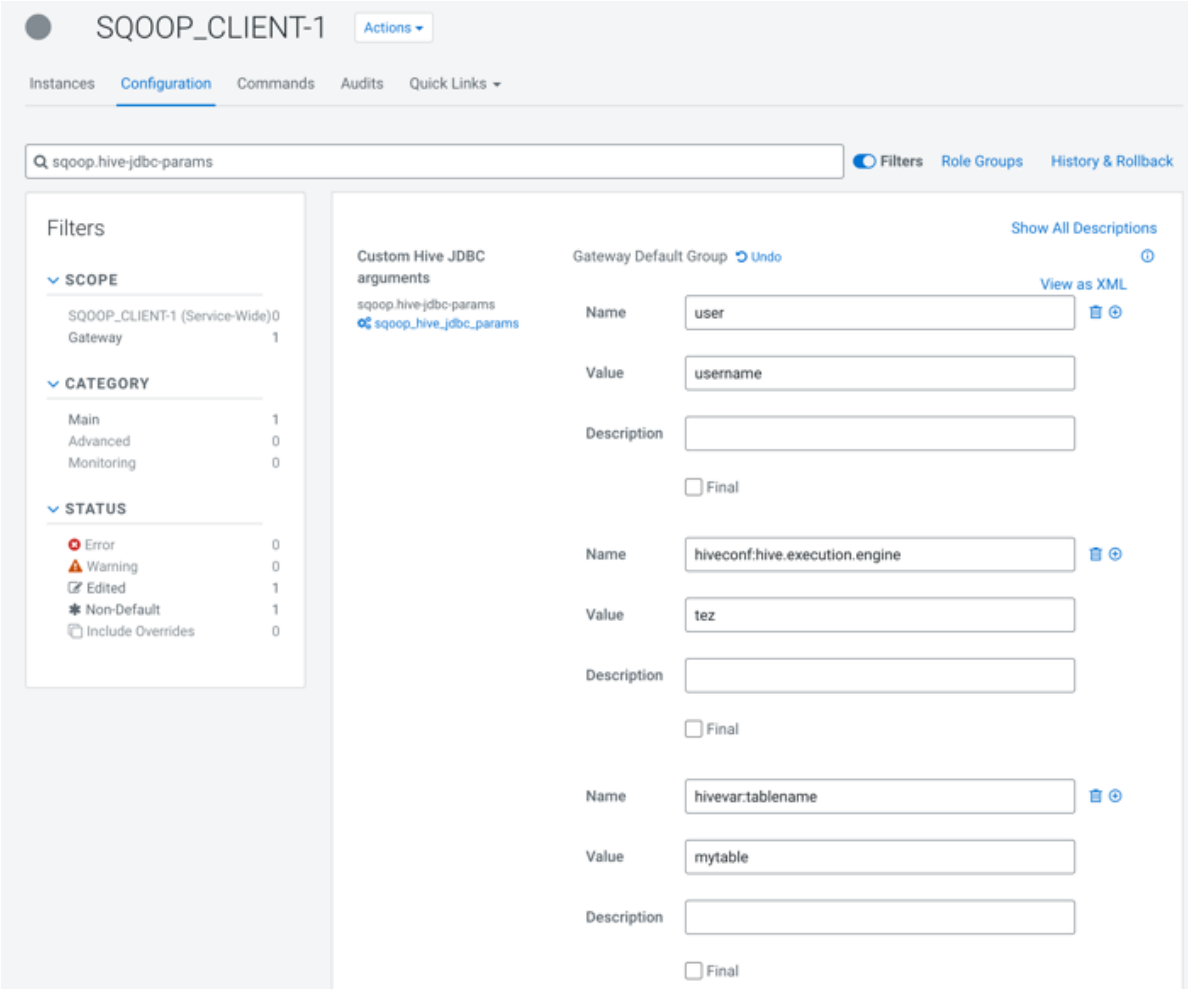
You can configure the required property either through Cloudera Manager or by using the --jdbc-arg argument in your Sqoop import command. Use the hiveconf: prefix to specify Hive configuration values and the hivevar: prefix to specify Hive user variables.

Order of precedence

The configuration set through the Sqoop argument in the command line takes precedence over the configuration specified through Cloudera Manager. If custom Hive JDBC arguments are specified through both the command line and Cloudera Manager, the command line Sqoop argument does not entirely overwrite the Cloudera Manager configuration. Instead, the distinct values are retained and the values with matching keys are replaced with arguments specified through the command line Sqoop argument.

Procedure

1. If you are specifying the custom Hive JDBC arguments through Cloudera Manager, perform the following steps:
   a) In Cloudera Manager, click Clusters and then select the SQOOP_CLIENT-1 service.
   b) From the Sqoop service, go to the **Configuration** tab and search for sqoop.hive-jdbc-params.
   c) Click ⊕ and specify the custom Hive JDBC argument and its value.



Important: Each key-value pair must be added to a new key-value field using ⊕.

d) Click Save Changes.

2. If you are specifying the custom Hive JDBC arguments through the Sqoop argument, specify the required key-value pair using the --hive-jdbc-arg argument while constructing the Sqoop import command.

```
/opt/cloudera/parcels/CDH/bin/sqoop import \
  -Dsqoop.beeline.env.preserve=KRB5CCNAME \
  --connection-manager org.apache.sqoop.manager.MySQLManager \
  --connect jdbc:mysql://db.foo.com:3306/employees \
  --username [***USERNAME***] \
  --password [***PASSWORD***] \
  --table employees \
  --warehouse-dir /user/hrt_qa/test-sqoop \
  --hive-import \
  --delete-target-dir \
  --hive-overwrite \
  --hs2-url "jdbc:hive2://[***HOST***]:[***PORT***];serviceDiscoveryMode=z
ooKeeper;zooKeeperNamespace=hiveserver2;transportMode=http;httpPath=clis
ervice;ssl=true;sslTrustStore=[***TRUSTSTORE PATH***];trustStorePassword
=[***TRUSTSTORE PASSWORD***]" \
  --hive-jdbc-arg user=username \
  --hive-jdbc-arg hiveconf:hive.execution.engine=tez \
  --hive-jdbc-arg hivevar:tablename=mytable
```

Important: Every key-value pair should be added in the following format: --hive-jdbc-arg key1=value1 --hive-jdbc-arg key2=value2

Configuring a custom Hive CREATE TABLE statement

Learn how to configure a custom CREATE TABLE statement that Sqoop uses during the Hive table creation process. For example, if you have configured Sqoop to use the CREATE EXTERNAL TABLE statement, then external tables are created by default during the table creation process.

About this task

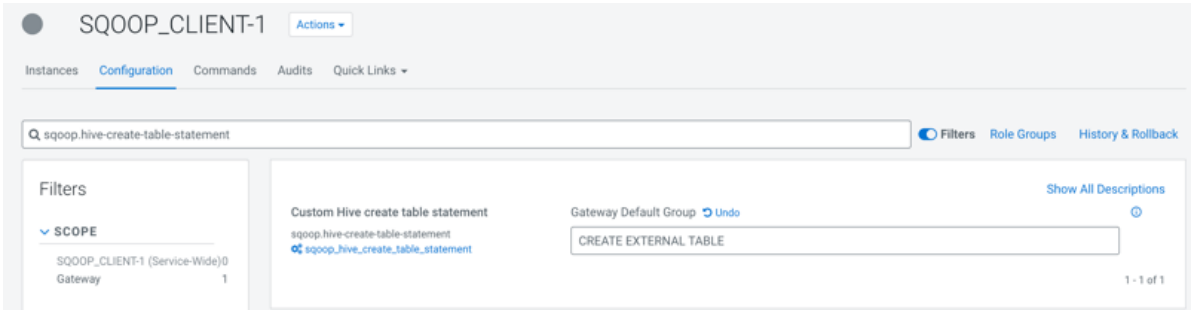
You can configure the required property either through Cloudera Manager or by using the --hive-create-table-statement argument in your Sqoop import command.

Order of precedence

The configuration set through the Sqoop argument in the command line takes precedence over the configuration specified through Cloudera Manager.

Procedure

1. If you are specifying the custom CREATE TABLE statement through Cloudera Manager, perform the following steps:
   a) In Cloudera Manager, click Clusters and then select the SQOOP_CLIENT-1 service.
   b) From the Sqoop service, go to the **Configuration** tab and search for sqoop.hive-create-table-statement.
   c) Enter the custom CREATE TABLE statement.



   d) Click Save Changes.
2. If you are specifying the custom CREATE TABLE statement through the Sqoop argument, specify the required statement using the --hive-create-table-statement argument while constructing the Sqoop import command.

```
/opt/cloudera/parcels/CDH/bin/sqoop import \
  -Dsqoop.beeline.env.preserve=KRB5CCNAME \
  --connection-manager org.apache.sqoop.manager.MySQLManager \
  --connect jdbc:mysql://db.foo.com:3306/employees \
  --username [***USERNAME***] \
  --password [***PASSWORD***] \
  --table employees \
  --warehouse-dir /user/hrt_qa/test-sqoop \
  --hive-import \
  --delete-target-dir \
  --hive-overwrite \
  --hive-create-table-statement "CREATE EXTERNAL TABLE"
```

Configuring custom Hive table properties

Learn how to specify custom Hive table properties for the CREATE TABLE statement that Sqoop uses during the Hive table creation process.

About this task

You can configure the required property either through Cloudera Manager or by using the --hive-table-property argument in your Sqoop import command.

For example, if you have specified the custom Hive table properties with the "transactional=true" and "transactional_properties=insert_only" key-value pairs, the Hive CREATE TABLE statement is constructed as follows:

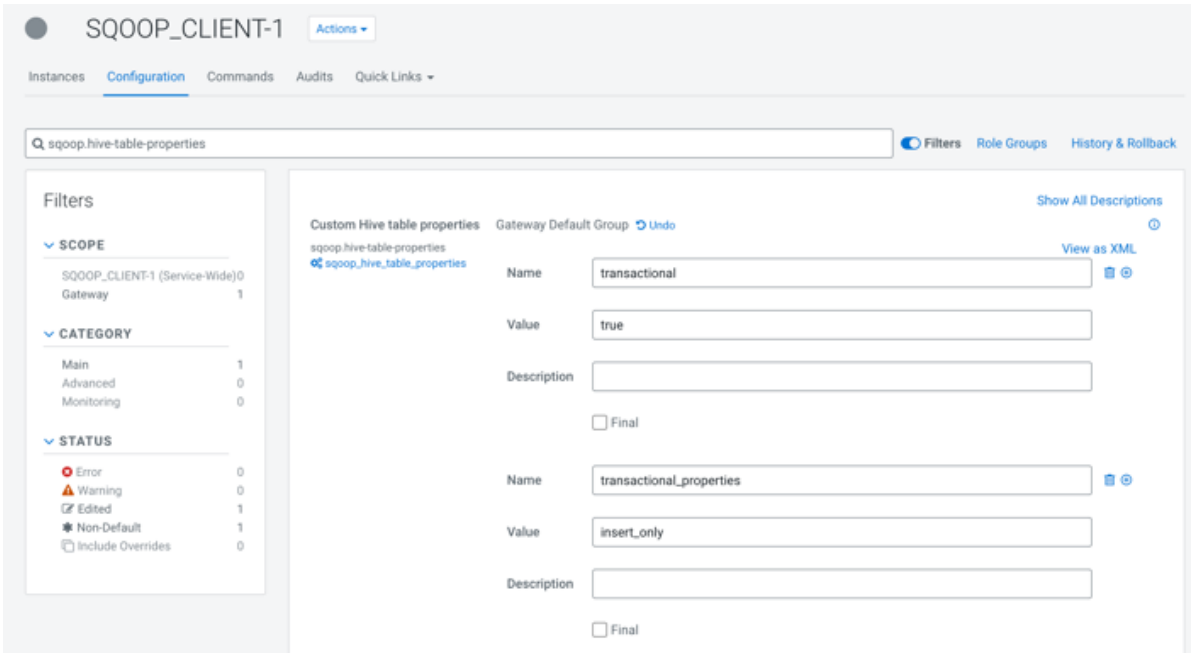
```
CREATE TABLE foo ....
....
TBLPROPERTIES ('transactional'='true', 'transactional_properties'='insert_only');
```

Order of precedence

The configuration set through the Sqoop argument in the command line takes precedence over the configuration specified through Cloudera Manager. If custom Hive table properties are specified through both the command line and Cloudera Manager, the command line Sqoop argument does not entirely overwrite the Cloudera Manager configuration. Instead, the distinct values are retained and the values with matching keys are replaced with arguments specified through the command line Sqoop argument.

Procedure

1. If you are specifying the custom Hive table properties through Cloudera Manager perform the following steps:
   a) In Cloudera Manager, click Clusters and then select the SQOOP_CLIENT-1 service.
   b) From the Sqoop service, go to the **Configuration** tab and search for sqoop.hive-table-properties.
   c) Click ⊕ and specify the custom Hive table property and its value.



Important: Each key-value pair must be added to a new key-value field using ⊕.

d) Click Save Changes.

2. If you are specifying the custom Hive table properties through the Sqoop argument, specify the required table properties using the --hive-table-property argument while constructing the Sqoop import command.

```
/opt/cloudera/parcels/CDH/bin/sqoop import \
  -Dsqoop.beeline.env.preserve=KRB5CCNAME \
  --connection-manager org.apache.sqoop.manager.MySQLManager \
  --connect jdbc:mysql://db.foo.com:3306/employees \
  --username [***USERNAME***] \
  --password [***PASSWORD***] \
  --table employees \
  --warehouse-dir /user/hrt_qa/test-sqoop \
  --hive-import \
  --delete-target-dir \
  --hive-overwrite \
  --hive-table-property transactional=true
  --hive-table-property transactional_properties=insert_only
```

Important: Every key-value pair should be added in the following format: --hive-table-property key1=value1 --hive-table-property key2=value2

Secure options to provide Hive password during a Sqoop import

Learn about the secure options that you can use to provide the Hive password during Sqoop-Hive imports instead of the earlier way of providing the password as plaintext in the command-line interface.

When importing data into Hive using Sqoop and if LDAP authentication is enabled for Hive, the necessity to set the Hive password parameter directly in the command-line poses a potential vulnerability. Passwords provided in plaintext within command-line interfaces are susceptible to unauthorized access or interception, compromising sensitive credentials and, subsequently, the security of the entire data transfer process.

You can use the following Sqoop arguments in your Sqoop import command:

The following Sqoop arguments are introduced that allow you to provide the Hive password in a secure way during the Sqoop-Hive import process:

Sqoop argument	Description
-promptHivePassword	Prompts the user to enter the Hive password
--hive-password-file <***FILE PATH***>	Stores the Hive password in a file and uses it during the import
--hive-password-alias <***ALIAS NAME***>	Stores the Hive password in a Credential Provider facility with an alias associated with the actual value in the credential storage

Along with securely providing the Hive password to Sqoop, it is essential that the password is safely persisted in the Hive metastore when saving a Sqoop job related to Hive.

Providing the Hive password through a prompt

Use the -promptHivePassword argument in the Sqoop import command to prompt users to enter the Hive password manually in the command-line for Sqoop-Hive import processes when LDAP authentication is enabled.

About this task

Procedure

1. Specify the -promptHivePassword argument while constructing the Sqoop import command.

```
/opt/cloudera/parcels/CDH/bin/sqoop import \
  -Dsqoop.beeline.env.preserve=KRB5CCNAME \
  --connection-manager org.apache.sqoop.manager.PostgresqlManager \
  --connect "jdbc:postgresql://db.foo.com:5432/employees" \
  --username [***USERNAME***] \
  --password [***PASSWORD***] \
  --table employees \
  --warehouse-dir \
  /user/hrt_qa/test-sqoop \
  --hive-import \
  --delete-target-dir \
  --hive-overwrite \
  --external-table-dir hdfs:///warehouse/tablespace/external/hive/employee
s \
  --hs2-url "jdbc:hive2://[***HOST***]:[***PORT***];serviceDiscoveryMode
=zooKeeper;zooKeeperNamespace=hiveserver2;transportMode=http;httpPath=cl
iservice;ssl=true;sslTrustStore=[***TRUSTSTORE PATH***];trustStorePasswo
rd=[***TRUSTSTORE PASSWORD***]" \
  --hive-user guest \
  -promptHivePassword \
  -m 1
```

2. Enter the Hive password in the command-line when you are prompted with the "Enter Hive password:" message.

Providing the Hive password through a file

You can save the Hive password in a file and then set up Sqoop to use this password for Sqoop-Hive import processes when LDAP authentication is enabled.

Procedure

1. Create a file containing the appropriate Hive password. Ensure to set '400' permission on the file so that only the user or owner of the file has read permissions.

   You can save this file either in a local file system or on HDFS.

2. While creating the Sqoop import command, specify the --hive-password-file argument along with the path of the password file you created in the previous step.

```
/opt/cloudera/parcels/CDH/bin/sqoop import \
  -Dsqoop.beeline.env.preserve=KRB5CCNAME \
  --connection-manager org.apache.sqoop.manager.PostgresqlManager \
  --connect "jdbc:postgresql://db.foo.com:5432/employees" \
  --username [***USERNAME***] \
  --password [***PASSWORD***] \
  --table employees \
  --warehouse-dir \
  /user/hrt_qa/test-sqoop \
  --hive-import \
  --delete-target-dir \
  --hive-overwrite \
  --external-table-dir hdfs:///warehouse/tablespace/external/hive/employee
s \
  --hs2-url "jdbc:hive2://[***HOST***]:[***PORT***];serviceDiscoveryMode
=zooKeeper;zooKeeperNamespace=hiveserver2;transportMode=http;httpPath=cl
```

```
iservice;ssl=true;sslTrustStore=[***TRUSTSTORE PATH***];trustStorePasswo
rd=[***TRUSTSTORE PASSWORD***]" \
  --hive-user guest \
  --hive-password-file /user/hrt_qa/hivepasswd-storefile \
  -m 1
```

Providing the Hive password through an alias

Learn how you can use an alias to represent the Hive password during Sqoop-Hive import processes when LDAP authentication is enabled.

About this task

The Hive password is stored in a Credential Provider facility and is associated with the alias. During the import, Sqoop resolves the alias and uses the linked password.

Procedure

1. Using the CredentialProvider API, store the Hive password in a user specified provider path and associate it with an alias.

```
/opt/cloudera/parcels/CDH/lib/hadoop/bin/hadoop credential \
  create sqoophive.password.alias \
  -value guest-password \
  -provider jceks://hdfs/user/hive/sqoophivepasswd.jceks
```

2. Add the provider path property in the Sqoop import command pointing to the credential provider URI that should be considered while resolving the credential alias.

   -D hadoop.security.credential.provider.path=<***PROVIDER PATH***>

```
-D hadoop.security.credential.provider.path=jceks://hdfs/user/hive/sqoop
hivepasswd.jceks \
```

3. While creating the Sqoop import command, specify the --hive-password-alias argument with the alias name that you want to resolve.

```
/opt/cloudera/parcels/CDH/bin/sqoop import \
  -Dsqoop.beeline.env.preserve=KRB5CCNAME \
  -D hadoop.security.credential.provider.path=jceks://hdfs/user/hive/
sqoophivepasswd.jceks \
  --connection-manager org.apache.sqoop.manager.PostgresqlManager \
  --connect "jdbc:postgresql://db.foo.com:5432/employees" \
  --username [***USERNAME***] \
  --password [***PASSWORD***] \
  --table employees \
  --warehouse-dir \
  /user/hrt_qa/test-sqoop \
  --hive-import \
  --delete-target-dir \
  --hive-overwrite \
  --external-table-dir hdfs:///warehouse/tablespace/external/hive/employee
s \
  --hs2-url "jdbc:hive2://[***HOST***]:[***PORT***];serviceDiscoveryMode
=zooKeeper;zooKeeperNamespace=hiveserver2;transportMode=http;httpPath=cl
iservice;ssl=true;sslTrustStore=[***TRUSTSTORE PATH***];trustStorePasswo
rd=[***TRUSTSTORE PASSWORD***]" \
  --hive-user guest \
  --hive-password-alias sqoophive.password.alias \
  -m 1
```

Providing the Hive password through an alias in a file

Learn how you can use an alias stored in a file to represent the Hive password during Sqoop-Hive import processes when LDAP authentication is enabled.

About this task

The Hive password is stored in a Credential Provider facility and is associated with the alias. During the import, Sqoop resolves the alias from the file and uses the linked password.

Procedure

1. Using the CredentialProvider API, store the Hive password in a user specified provider path and associate it with an alias.

```
/opt/cloudera/parcels/CDH/lib/hadoop/bin/hadoop credential \
  create sqoophive.password.alias \
  -value guest-password \
  -provider jceks://hdfs/user/hive/sqoophivepasswd.jceks
```

2. Create a file containing the alias that you created in the previous step. Ensure to set '400' permission on the file so that only the user or owner of the file has read permissions.

   You can save this file either in a local file system or on HDFS.

3. Add the provider path property in the Sqoop import command pointing to the credential provider URI that should be considered while resolving the credential alias.

   -D hadoop.security.credential.provider.path=<***PROVIDER PATH***>

```
-D hadoop.security.credential.provider.path=jceks://hdfs/user/hive/sqoop
hivepasswd.jceks \
```

4. Include the following argument in the Sqoop import command to ensure that the content of the file you created will be handled as an alias.

   -D org.apache.sqoop.credentials.loader.class=org.apache.sqoop.util.password.CredentialProviderPasswordLoader

5. While creating the Sqoop import command, specify the --hive-password-file argument along with the path of the alias file you created earlier.

```
/opt/cloudera/parcels/CDH/bin/sqoop import \
  -Dsqoop.beeline.env.preserve=KRB5CCNAME \
  -D hadoop.security.credential.provider.path=jceks://hdfs/user/hive/
sqoophivepasswd.jceks \
  -D
 org.apache.sqoop.credentials.loader.class=org.apache.sqoop.util.password.CredentialP
  --connection-manager org.apache.sqoop.manager.PostgresqlManager \
  --connect "jdbc:postgresql://db.foo.com:5432/employees" \
  --username [***USERNAME***] \
  --password [***PASSWORD***] \
  --table employees \
  --warehouse-dir \
  /user/hrt_qa/test-sqoop \
  --hive-import \
  --delete-target-dir \
  --hive-overwrite \
  --external-table-dir hdfs:///warehouse/tablespace/external/hive/employee
s \
  --hs2-url "jdbc:hive2://[***HOST***]:[***PORT***];serviceDiscoveryMode
=zooKeeper;zooKeeperNamespace=hiveserver2;transportMode=http;httpPath=cl
```

```
iservice;ssl=true;sslTrustStore=[***TRUSTSTORE PATH***];trustStorePasswo
rd=[***TRUSTSTORE PASSWORD***]" \
  --hive-user guest \
  --hive-password-file /user/hrt_qa/hivepasswd-storefile \
  -m 1
```

Saving the password to Hive Metastore

Learn how Sqoop stores the password in the Hive Metastore (HMS) when the Sqoop job is saved and how this password is retrieved from the metastore when the job is run.

Along with securely providing the Hive password to Sqoop, it is essential that the password is safely persisted in the Hive metastore when saving a Sqoop job related to Hive.

The following sections describe how Sqoop stores the Hive password in HMS based on the secure option that you have chosen to provide the password during the Sqoop-Hive import process:

Hive password is provided through a file

When you save the Hive password in a file and then set up Sqoop to use this password for Sqoop-Hive import processes, Sqoop stores the path of the password file in HMS when the Sqoop job is saved.

On running the Sqoop job, the file path is retrieved from HMS and the content of the file is used as the Hive password during the job execution.

Note: The file might include the actual Hive password or an alias linked to the password in a Credential Provider facility, provided that the CredentialProviderPasswordLoader and the Credential Provider path is configured correctly.

Hive password is provided through an alias

When you use an alias to represent the Hive password during the Sqoop-Hive import process, Sqoop stores the alias in the HMS when the Sqoop job is saved.

On running the Sqoop job, the value of the alias is retrieved from the HMS and the value is read by Sqoop. This value is then used to obtain the actual Hive password that is stored in a Credential Provider facility and referenced by a provider path. Once the password is acquired, it is used as the Hive password during the job execution.

Hive password is provided in an insecure way

When the Hive password is provided using one of the following arguments, Sqoop does not store the raw or unencrypted password in the HMS for security reasons and instead prompts the user to enter the Hive password.

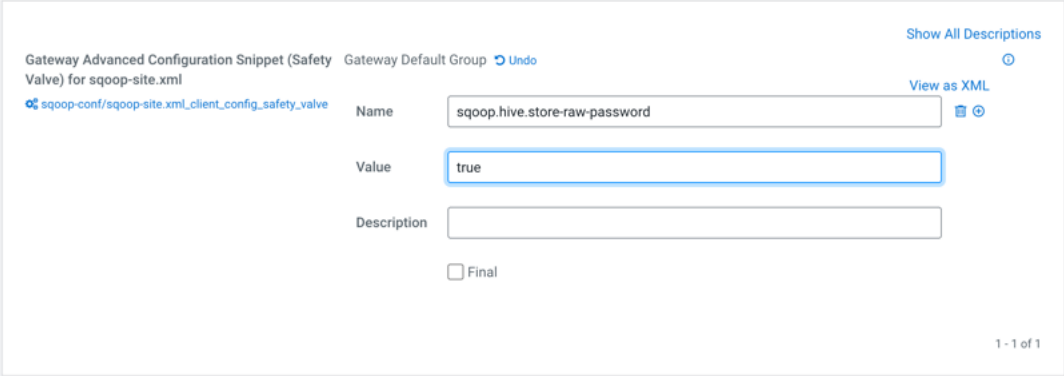
- --hive-password
- --hs2-password
- -promptHivePassword

However, if you still want to save the password in HMS although it is insecure, you can set the sqoop.hive.store-raw-password to true either by using this as an argument in your Sqoop import command or by adding this to Cloudera Manager Advanced Configuration Snippet (Safety Valve) for sqoop-site.xml. This ensures that the password is persisted in HMS and you can run the Sqoop job without having to enter the password.

Important: The configuration set through the Sqoop argument in the command-line takes precedence over the configuration specified through Cloudera Manager.

Setting the sqoop.hive.store-raw-password property through Cloudera Manager

From Cloudera Manager, go to Clusters > SQOOP CLIENT > Configuration and search for 'Gateway Advanced Configuration Snippet (Safety Valve) for sqoop-site.xml'.



Setting the sqoop.hive.store-raw-password property through the Sqoop command

```
/opt/cloudera/parcels/CDH/bin/sqoop job \
 -Dsqoop.hive.store-raw-password=true \
 --meta-connect "jdbc:postgresql://db.foo.com:5432/employees" \
 --meta-username [***USERNAME***] \
 --meta-password [***PASSWORD***] \
 --create myjob -- import \
 --connection-manager org.apache.sqoop.manager.PostgresqlManager
\
 --connect "jdbc:postgresql://db.foo.com:5432/employees" \
 --username [***USERNAME***] \
 --password [***PASSWORD***] \
 --table employees \
 --warehouse-dir /user/hrt_qa/test-sqoop \
 --hive-import \
 --delete-target-dir \
 --hive-overwrite \
 --external-table-dir hdfs:///warehouse/tablespace/external/hive/
employees \
 --hs2-url "jdbc:hive2://[***HOST***]:[***PORT***];serviceDiscov
eryMode=zooKeeper;zooKeeperNamespace=hiveserver2;transportMode=h
ttp;httpPath=cliservice;ssl=true;sslTrustStore=[***TRUSTSTORE PA
TH***];trustStorePassword=[***TRUSTSTORE PASSWORD***]" \
 --hive-user guest \
 --hive-password [***PASSWORD***] \
 -m 1
```

Imports into Hive

You can use Sqoop to import data from a relational database into for use with Hive. You can import the data directly to Hive. Assuming the Sqoop client service is available in your cluster, you can issue import and export commands from the command line.

Related Information

Apache Sqoop Documentation (v1.4.7.7.1.6)

Creating a Sqoop import command

You create a single Sqoop import command that imports data from diverse data sources, such as a relational database on a different network, into Apache Hive using Apache Sqoop.

About this task

You enter the Sqoop import command on the command line of your Hive cluster to import data from a data source into the cluster file system and Hive. The import can include the following information, for example:

- Database connection information: database URI, database name, and connection protocol, such as jdbc:mysql:
- The data to import
- Parallel processing directives for fast data transfer
- Destination for imported data

Sqoop is tested to work with Connector/J 5.1. If you have upgraded to Connector/J 8.0, and want to use the zeroDate TimeBehavior property to handle values of '0000-00-00\' in DATE columns, explicitly specify zeroDateTimeBeha vior=CONVERT_TO_NULL in the connection string. For example, --connect jdbc:mysql://<MySQL host>/<DB>? zeroDateTimeBehavior=CONVERT_TO_NULL.

Before you begin

It is recommended that you familiarize yourself with the Sqoop configurations that can enable you to control the imported data according to specific requirements.

Procedure

1. Create an import command that specifies the Sqoop connection to the RDBMS.
   - To enter a password for the data source on the command line, use the -P option in the connection string.
   - To specify a file where the password is stored, use the --password-file option.

   Password on command line:

   ```
   sqoop import --connect jdbc:mysql://db.foo.com:3306/bar \
   <data to import> \
   --username <username> \
   -P
   ```

   Specify password file:

   ```
   sqoop import --connect jdbc:mysql://db.foo.com:3306/bar \
   --table EMPLOYEES \
   --username <username> \
   --password-file ${user.home}/.password
   ```

2. Specify the data to import in the command.
   - Import an entire table.
   - Import a subset of the columns.
   - Import data using a free-form query.

   Entire table:

   ```
   sqoop import \
   --connect jdbc:mysql://db.foo.com:3306/bar \
   --table EMPLOYEES
   ```

   Subset of columns:

   ```
   sqoop import
   --connect jdbc:mysql://db.foo.com:3306/bar \
   --table EMPLOYEES \
   --columns "employee_id,first_name,last_name,job_title"
   ```

Free-form query to import the latest data:

```
sqoop import \
--connect jdbc:mysql://db.foo.com:3306/bar \
--table EMPLOYEES \
--where "start_date > '2018-01-01'"
```

3. Optionally, specify write parallelism in the import statement to run a number of map tasks in parallel:
   - Set mappers: If the source table has a primary key, explicitly set the number of mappers using --num-mappers.
   - Split by: If primary keys are not evenly distributed, provide a split key using --split-by
   - Sequential: If you do not have a primary key or split key, import data sequentially using --num-mappers 1 or --autoreset-to-one-mapper in query.
   - Set mappers:

     ```
     sqoop import --connect jdbc:mysql://db.foo.com:3306/bar \
      --table EMPLOYEES \
      --num-mappers 8 \
     ```

   - Split by:

     ```
     sqoop import --connect jdbc:mysql://db.foo.com:3306/bar \
      --table EMPLOYEES \
      --split-by dept_id
     ```

   - Setting mappers evenly splits the primary key range of the source table.
   - Split by evenly splits the data using the split key instead of a primary key.
4. Specify importing the data into Hive using Hive default delimiters --hive-import.
5. Specify the Hive destination of the data.
   - If you think the table does not already exist in Hive, name the database and table, and use the --create-hive-table option.
   - If you want to insert the imported data into an existing Hive external table, name the database and table, but do not use the --create-hive-table option.

   This command imports the MySQL EMPLOYEES table to a new Hive table named in the warehouse.

   ```
   sqoop import --connect jdbc:mysql://db.foo.com:3306/corp \
   --table EMPLOYEES \
   --hive-import \
   --create-hive-table \
   --hive-database 'mydb' \
   --hive-table 'newtable'
   ```

   This command imports the MySQL EMPLOYEES table to an external table in HDFS.

   ```
   sqoop import --connect jdbc:mysql://db.foo.com:3306/corp \
   --table EMPLOYEES \
   --hive-import \
   --hive-database 'mydb' \
   --hive-table 'myexternaltable'
   ```

   Specify the database and table names, enclosed in single quotation marks, on separate lines (recommended) as shown above. Alternatively specify the database and table names on one line, and enclose the database and table names in backticks.

   --hive-table `mydb`.`myexternaltable`

Due to the Hive-16907 bug fix, Hive rejects `db.table` in SQL queries. A dot (.) is no longer allowed in table names. You need to change queries that use such references to prevent Hive from interpreting the entire db.table string as the table name.

Related Information

Apache Sqoop Documentation (v1.4.7.7.1.6)

Sqoop enhancements to the Hive import process

Importing RDBMS data into Hive

You can test the Apache Sqoop import command and then run the command to import relational database tables into Apache Hive.

About this task

You enter the Sqoop import command on the command line of your Hive cluster to import data from a data source to Hive. You can test the import statement before actually executing it.

Before you begin

- The Apache Sqoop client service is available.
- The Hive Metastore and Hive services are available.

Procedure

1. Optionally, test the import command before execution using the eval option.

```
sqoop eval --connect jdbc:mysql://db.foo.com/bar \
--query "SELECT * FROM employees LIMIT 10"
```

   The output of the select statement appears listing 10 rows of data from the RDBMS employees table.

2. Rn a Sqoop import command that specifies the Sqoop connection to the RDBMS, the data you want to import, and the destination Hive table name.
   This command imports the MySQL EMPLOYEES table to a new Hive table named in the warehouse.

```
sqoop import --connect jdbc:mysql://db.foo.com:3306/corp \
--table EMPLOYEES \
--hive-import \
--create-hive-table \
--hive-table mydb.newtable
```

Related Information

Apache Sqoop Documentation (v1.4.7.7.1.6)

HDFS to Apache Hive data migration

In Cloudera Base on premises, you can import data from diverse data sources into HDFS, perform ETL processes, and then query the data in Apache Hive.

Importing RDBMS data to HDFS

In Cloudera Base on premises, you create a single Sqoop import command that imports data from a relational database into HDFS.

About this task

You enter the Sqoop import command on the command line of your cluster to import data into HDFS. The import command needs to include the database URI, database name, and connection protocol, such as jdbc:mysql:m and the data to import. Optionally, the command can include parallel processing directives for fast data transfer, the HDFS destination directory for imported data, data delimiters, and other information. The default HDFS directory is used if you do not specify another location. Fields are comma-delimited and rows are line-delimited.You can test the import statement before actually executing it.

Before you begin

- Apache Sqoop is installed and configured.

Procedure

1. Create an import command that specifies the Sqoop connection to the data source you want to import.
   - If you want to enter a password for the data source on the command line, use the -P option in the connection string.
   - If you want to specify a file where the password is stored, use the --password-file option.

   Password on command line:

   ```
   sqoop import --connect jdbc:mysql://db.foo.com/bar \
   <data to import> \
   --username <username> \
   -P
   ```

   Specify password file:

   ```
   sqoop import --connect jdbc:mysql://db.foo.com/bar \
   --table EMPLOYEES \
   --username <username> \
   --password-file ${user.home}/.password
   ```

2. Specify the data to import in the command.
   - Import an entire table.
   - Import a subset of the columns.
   - Import data using a free-form query.

   Entire table:

   ```
   sqoop import \
   --connect jdbc:mysql://db.foo.com/bar \
   --table EMPLOYEES
   ```

   Subset of columns:

   ```
   sqoop import \
   --connect jdbc:mysql://db.foo.com/bar \
   --table EMPLOYEES \
   --columns "employee_id,first_name,last_name,job_title"
   ```

   Free-form query to import the latest data:

   ```
   sqoop import \
   --connect jdbc:mysql://db.foo.com/bar \
   --table EMPLOYEES \
   --where "start_date > '2018-01-01'"
   ```

3. Specify the destination of the imported data using the --target-dir option.

This command appends data imported from the MySQL EMPLOYEES table to the output files in the HDFS target directory using default text file delimiters.

```
sqoop import \
--connect jdbc:mysql://db.foo.com:3600/bar \
--table EMPLOYEES \
--where "id > 100000" \
--target-dir /incremental_dataset \
--append
```

This command splits imported data by column and specifies importing the data into output files in the HDFS target directory.

```
sqoop import \
--connect jdbc:mysql://db.foo.com:3600/bar \
--query 'SELECT a.*, b.* \
FROM a JOIN b on (a.id == b.id) \
WHERE $CONDITIONS' \
--split-by a.id \
--target-dir /user/foo/joinresults
```

This command executes once and imports data serially using a single map task as specified by the -m 1 options:

```
sqoop import \
--connect jdbc:mysql://db.foo.com:3600/bar \
--query \
'SELECT a.*, b.* \
FROM a \
JOIN b on (a.id == b.id) \
WHERE $CONDITIONS' \
-m 1 \
--target-dir /user/foo/joinresults
```

4. Optionally, specify write parallelism in the import statement to run a number of map tasks in parallel:
   - Set mappers: If the source table has a primary key, explicitly set the number of mappers using --num-mappers.
   - Split by: If primary keys are not evenly distributed, provide a split key using --split-by
   - Sequential: If you do not have a primary key or split key, import data sequentially using --num-mappers 1 or --autoreset-to-one-mapper in query.
   - Set mappers:

     ```
     sqoop import --connect jdbc:mysql://db.foo.com:3306/bar \
     --table EMPLOYEES \
     --num-mappers 8
     ```

   - Split by:

     ```
     sqoop import --connect jdbc:mysql://db.foo.com:3306/bar \
     --table EMPLOYEES \
     --split-by dept_id
     ```

   - Setting mappers evenly splits the primary key range of the source table.
   - Split by evenly splits the data using the split key instead of a primary key.
5. Optionally, test the import command before execution using the eval option.

   ```
   sqoop eval --connect jdbc:mysql://db.foo.com:3306/bar \
   --query "SELECT * FROM employees LIMIT 10"
   ```

   The output of the select statement appears.

Related Information

Apache Sqoop Documentation (v1.4.7.7.1.6)

Converting an HDFS file to ORC

In Cloudera Base on premises, to use Hive to query data in HDFS, you apply a schema to the data and then store data in ORC format.

About this task

To convert data stored in HDFS into the recommended format for querying in Hive, you create a schema for the HDFS data by creating a Hive external table, and then create a Hive-managed table to convert and query the data in ORC format. The conversion is a parallel and distributed action, and no standalone ORC conversion tool is necessary. Suppose you have the following CSV file that contains a header line that describes the fields and subsequent lines that contain the following data:

```
Name,Miles_per_Gallon,Cylinders,Displacement,Horsepower,Weight_in_lbs, \
  Acceleration,Year,Origin
"chevrolet chevelle malibu",18,8,307,130,3504,12,1970-01-01,A
"buick skylark 320",15,8,350,165,3693,11.5,1970-01-01,A
"plymouth satellite",18,8,318,150,3436,11,1970-01-01,A
"amc rebel sst",16,8,304,150,3433,12,1970-01-01,A
"ford torino",17,8,302,140,3449,10.5,1970-01-01,A
```

Before you begin

You removed the header from the CSV file.

Procedure

1. Create an external table:

```
CREATE EXTERNAL TABLE IF NOT EXISTS Cars(
    Name STRING,
    Miles_per_Gallon INT,
    Cylinders INT,
    Displacement INT,
    Horsepower INT,
    Weight_in_lbs INT,
    Acceleration DECIMAL,
    Year DATE,
    Origin CHAR(1))
COMMENT 'Data about cars from a public database'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE
location '/user/<username>/visdata';
```

2. Create a Hive-managed table to convert the data to ORC.

```
CREATE TABLE IF NOT EXISTS mycars(
Name STRING,
Miles_per_Gallon INT,
Cylinders INT,
Displacement INT,
Horsepower INT,
Weight_in_lbs INT,
Acceleration DECIMAL,
Year DATE,
Origin CHAR(1))
```

```
COMMENT 'Data about cars from a public database'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS ORC;
```

3. Insert the data from the external table into the Hive-managed table.

```
INSERT OVERWRITE TABLE mycars SELECT * FROM cars;
```

4. Verify that you imported the data into the ORC-formatted table correctly:

```
hive> SELECT * FROM mycars LIMIT 3;
OK
"chevrolet chevelle malibu" 18 8 307 130 3504 12 1970-01-01 A
"buick skylark 320" 15 8 350 165 3693 12 1970-01-01 A
"plymouth satellite" 18 8 318 150 3436 11 1970-01-01 A
Time taken: 0.144 seconds, Fetched: 3 row(s)
```

Incrementally updating an imported table

In Cloudera Base on premises, updating imported tables involves importing incremental changes made to the original table using Apache Sqoop and then merging changes with the tables imported into Apache Hive.

About this task

After ingesting data from an operational database to Hive, you usually need to set up a process for periodically synchronizing the imported table with the operational database table. The base table is a Hive-managed table that was created during the first data ingestion. Incrementally updating Hive tables from operational database systems involves merging the base table and change records to reflect the latest record set. You create the incremental table as a Hive external table, typically from CSV data in HDFS, to store the change records. This external table contains the changes (INSERTs and UPDATEs) from the operational database since the last data ingestion. Generally, the table is partitioned and only the latest partition is updated, making this process more efficient.

You can automate the steps to incrementally update data in Hive by using Oozie.

Before you begin

- The first time the data was ingested into hive, you stored entire base table in Hive in ORC format.
- The base table definition after moving it from the external table to a Hive-managed table has the following schema:

```
CREATE TABLE base_table (
id STRING,
field1 STRING,
modified_date DATE)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ',';
```

Procedure

1. Store the incremental table as an external table in Hive and to fetch records newer than last_import_date, which is the date of the last incremental data update.

   You frequently import incremental changes since the last time data was updated and then merging it.

   - using --check-column to fetch records
   - use --query to fetch records

```
sqoop import --connect jdbc:teradata://{host name}/Database=retail --con
nection-manager org.apache.sqoop.teradata.TeradataConnManager --username
```

```
 dbc --password dbc --table SOURCE_TBL --target-dir /user/hive/increment
al_table -m 1 --check-column modified_date --incremental lastmodified --
last-value {last_import_date}
```

```
sqoop import --connect jdbc:teradata://{host name}/Database=retail --con
nection-manager org.apache.sqoop.teradata.TeradataConnManager --username
 dbc --password dbc --target-dir /user/hive/incremental_table -m 1 --query
 'select * from SOURCE_TBL where modified_date > {last_import_date} AND $C
ONDITIONS'
```

2. After the incremental table data is moved into HDFS using Sqoop, you can define an external Hive table over it using the following command

```
CREATE EXTERNAL TABLE incremental_table (
  id STRING,
  field1 STRING,
  modified_date DATE)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE
location '/user/hive/incremental_table';
```

3. Use the MERGE command to merge the data and reconcile the base table records with the new records:

```
MERGE INTO base_table
USING incremental_table ON base.id = incremental_table.id
WHEN MATCHED THEN UPDATE SET
fieldl1=incremental_table.email,
modified_date=incremental_table.state
WHEN NOT MATCHED THEN INSERT
VALUES(incremental_table.id, incremental_table.field1, incremental_tabl
e.modified_data);
```

Import command options

You can use Sqoop command options to import data into Apache Hive.

Table 1: Sqoop Command Options for Importing Data into Hive

Sqoop Command Option	Description
--hive-home <directory>	Overrides $HIVE_HOME.
--hive-import	Imports tables into Hive using Hive's default delimiters if none are explicitly set.
--hive-overwrite	Overwrites existing data in the Hive table.
--create-hive-table	Creates a hive table during the operation. If this option is set and the Hive table already exists, the job will fail. Set to false by default.
--hive-create-table-statement	Specifies a custom CREATE TABLE statement that Sqoop uses during the Hive table creation process.
--hive-table <table_name>	Specifies the table name to use when importing data into Hive.
--hive-table-property	Specifies custom Hive table properties for the CREATE TABLE statement that Sqoop uses during the Hive table creation process.
--hive-drop-import-delims	Drops the delimiters \n, \r, and \01 from string fields when importing data into Hive.

Sqoop Command Option	Description
--hive-delims-replacement	Replaces the delimiters \n, \r, and \01 from strings fields with a user-defined string when importing data into Hive.
--hive-partition-key	Specifies the name of the Hive field on which a sharded database is partitioned.
--hive-partition-value <value>	A string value that specifies the partition key for data imported into Hive.
--map-column-hive <map>	Overrides the default mapping from SQL type to Hive type for configured columns.
--beeline-args	Enables you to include additional Beeline arguments while importing data into Hive.
--hive-jdbc-arg	Enables you to specify custom Hive JDBC arguments, such as Hive session variables, user variables, and configuration values.