Cloudera Data Engineering 1.5.5

Managing Cloudera Data Engineering jobs

Date published: 2020-07-30
Date modified: 2025-11-08

https://docs.cloudera.com/

Legal Notice

Contents

Creating jobs in Cloudera Data Engineering

A job in Cloudera Data Engineering consists of defined configurations and resources, including application code. Jobs can be run on demand or scheduled.

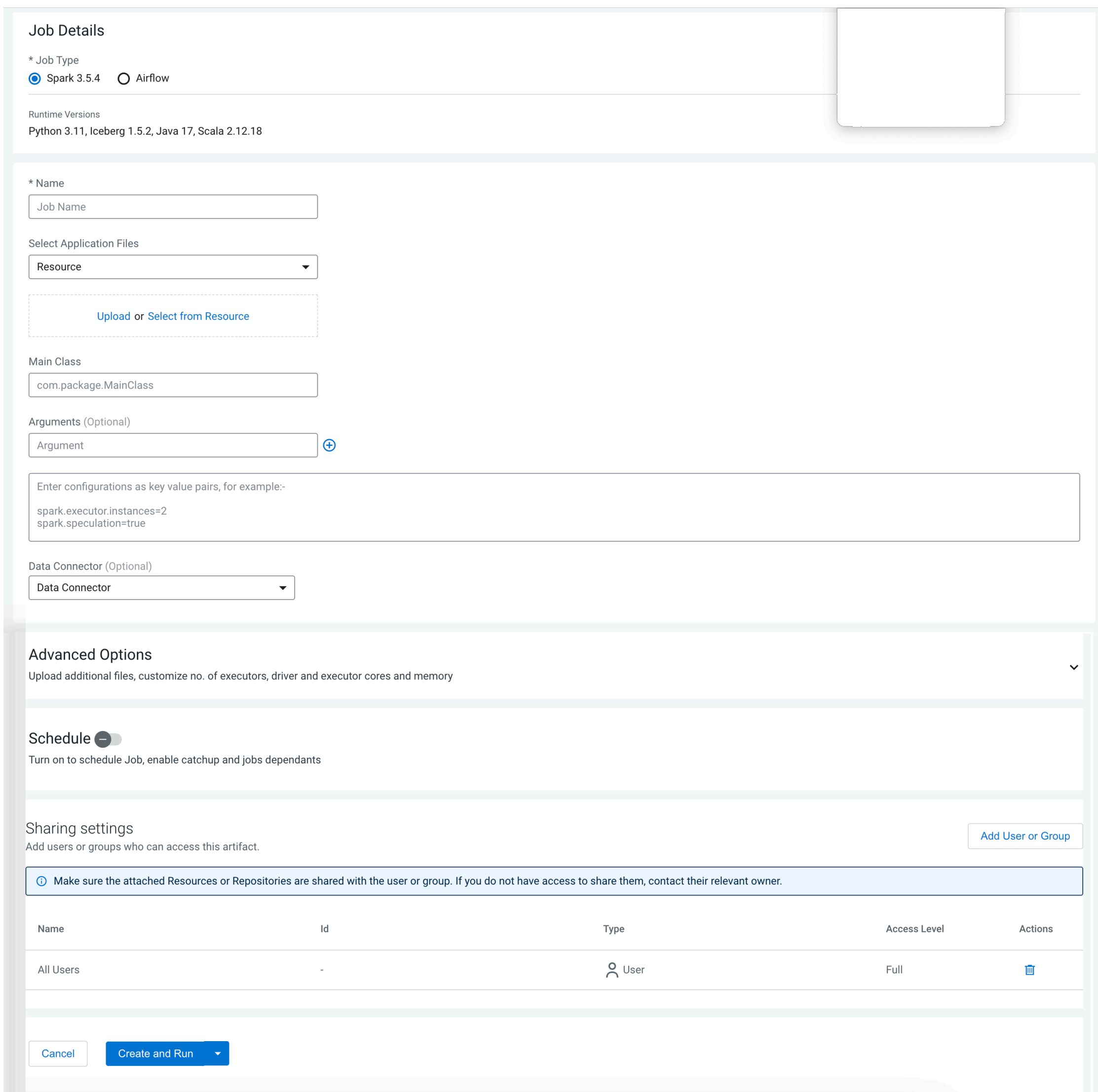
Before you begin

Important: You must create the cluster, initialize each cluster, and configure each user who need to submit jobs before creating jobs.

In Cloudera Data Engineering, jobs are associated with virtual clusters. Before you can create a job, you must create a virtual cluster that can run it. For more information, see Creating virtual clusters.

Procedure

1. In the Cloudera console, click the Data Engineering tile. The Cloudera Data Engineering Home page displays.
2. In the left navigation menu click Jobs. The Jobs page is displayed.
3. Click Create Job. The Job Details page is displayed.



4. Provide the Job Details.

a) Select Spark for the job type. If you are creating the job from the Home page, select the virtual cluster where you want to create the job.

b) Specify the Name.

c) Select File or URL for your application file, and provide or specify the file. You can upload a new file or select a file from an existing resource.

If you select the URL option and specify an Amazon AWS S3 URL, add the following configuration to the job:

config_key: spark.hadoop.fs.s3a.delegation.token.binding

config_value: org.apache.knox.gateway.cloud.idbroker.s3a.IDBDelegationTokenBinding

d) If your application code is a JAR file, specify the Main Class.

e) Specify arguments if required. You can click the Add Argument button to add multiple command arguments as necessary.

f) Enter configurations if needed. You can add multiple configuration parameters as key-value pairs in the text box as necessary.

Important:

- For Spark jobs, setting the `spark.app.id` property at the Spark job level configuration or within the Spark application code is not supported in Cloudera Data Engineering.
- The Spark session is created during the job run or session creation. Most Spark configurations are not modifiable during runtime, you must specify them during job run or session creation. To check if you can modify a configuration, use spark.conf.isModifiable. For example,

```
spark.conf.isModifiable("spark.executor.memory")
False
```

g) Optional: Select the name of the data connector from the Data Connector drop-down list. The UI displays the storage information that is internally overwritten.

Note: Data connector option is only available in the Cloudera Control Plane UI and the job run which uses the data connector must be only created using the Cloudera Control Plane UI. Triggering the jobs with the data connector using the Cloudera Runtime UI is not supported and the job will fail.

h) If your application code is a Python file, select the Python Version, and optionally select a Python Environment.

5. Click Advanced Configurations to display more customizations, such as additional files, initial executors, executor range, driver and executor cores, and memory.

By default, the executor range is set to match the range of CPU cores configured for the virtual cluster. This improves resource utilization and efficiency by allowing jobs to scale up to the maximum virtual cluster resources available, without manually tuning and optimizing the number of executors per job.

GPU Acceleration (Technical Preview): You can accelerate your Spark jobs using GPUs. Click Enable GPU Accelerations checkbox to enable the GPU acceleration and configure selectors and tolerations if you want to run the job on specific GPU nodes. When this job is created and run, this particular job will request GPU resources.

Warning: You must ensure this virtual cluster has been configured with GPU resource quota. Otherwise, the jobs will be in the Pending state as no GPU resource can be allocated to the pod.

6. Click Schedule to display scheduling options.

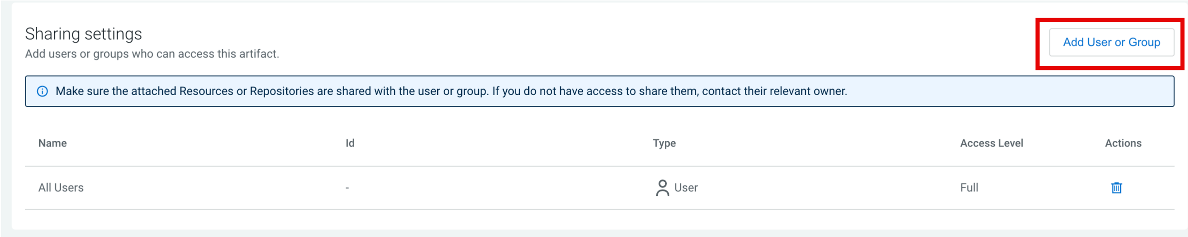
You can schedule the application to run periodically using the Basic controls or by specifying a Cron Expression.

7. Click Alerts and provide the email ID to receive alerts. Click + to add more email IDs. Optionally, you can select when you want email alerts for job failures, missed job service-level agreements, or both.

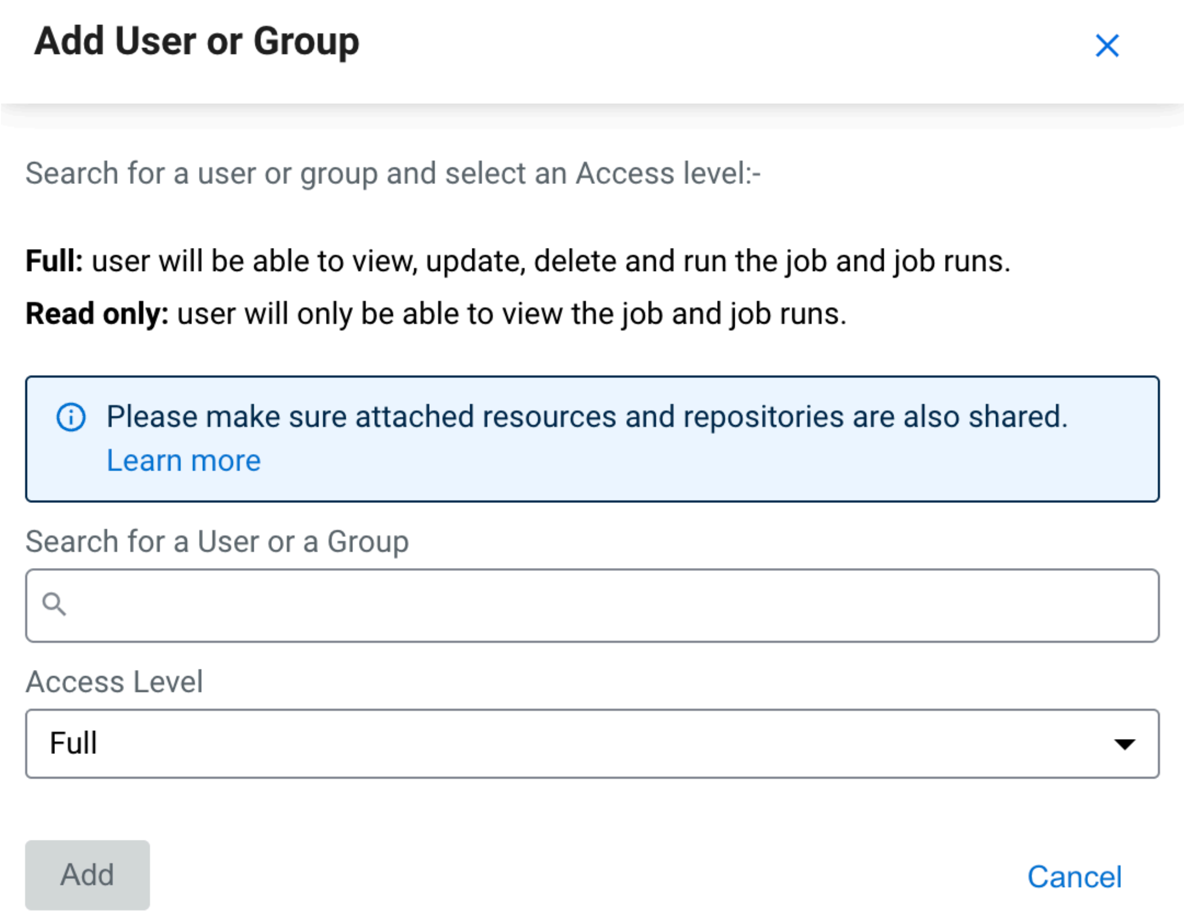
Note: You must configure the Configure Email Alerting option while creating a virtual cluster to send your email alerts. For more information about configuring email alerts, see Creating virtual clusters.

8. Share the job with a user or group.

a) In the Sharing Settings section, click Add User or Group. The Add User or Group pop-up appears.



b) In the Search for a User or a Group field, type the user or group name and select the required user or group from the list.



c) Select Full or Read Only depending on the access you want to provide from the Access Level drop-down list.

d) Click Add.

9. If you provided a schedule, click Schedule to create the job. If you did not specify a schedule, and you do not want the job to run immediately, click the drop-down arrow on Create and Run and select Create. Otherwise, click Create and Run to run the job immediately.

Related Information

Generate Access Key

Hadoop Authentication

The Hadoop Authentication allows you to update the Kerberos keytab file. This allows you to run the jobs or sessions.

Note:

- If the Kerberos Keytab file is not authenticated properly, then you cannot run the jobs or sessions.
- Make sure that user who is accessing the Virtual Cluster has the appropriate Ranger permissions on the base cluster for HDFS, Ozone, Hive, Impala, and so on.

Configuring Hadoop Authentication

Learn about how to configure Hadoop Authentication for a service.

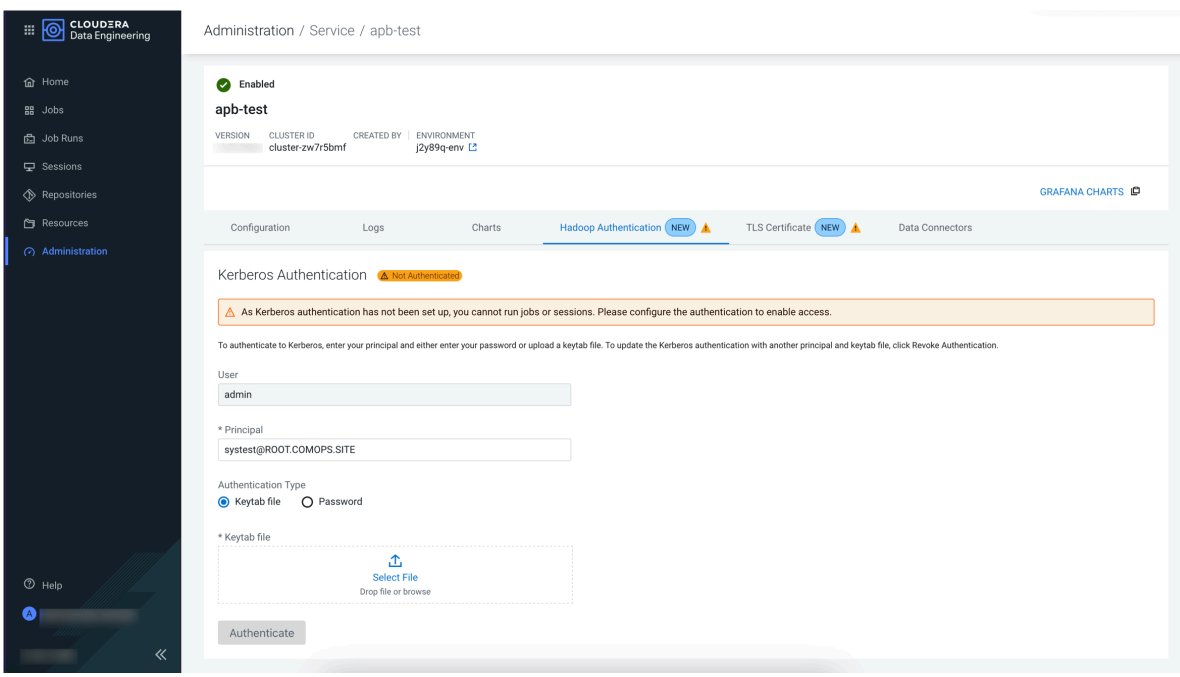
Important: Hadoop Authentication for Machine Users can be only configured through Cloudera Data Engineering API.

For Cloudera Data Engineering UI

1. In the Cloudera console, click the Data Engineering tile. The Cloudera Data Engineering Home page displays.
2. Click Administration in the left navigation menu. The Administration page displays.
3. In the Services column, select the environment for which you want to configure the Hadoop Authentication and click Service Details.
4. Click Hadoop Authentication.

Note:

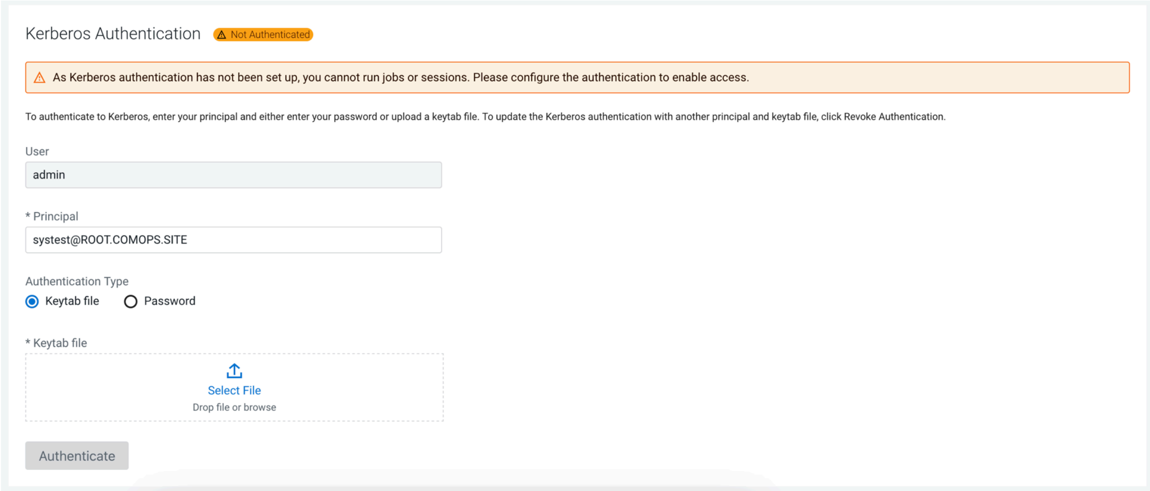
- In the Hadoop Authentication tab, the User field is automatically populated with the username of the logged-in user.
- Only logged-in users can access their Hadoop authentication details in Cloudera Data Engineering Service. No user can view another user's Hadoop authentication details, even if they have DEAdmin or Service Admin roles.



5. Enter the Principal value.

6. Under Authentication Type, choose one of the following options:

- Using the Keytab file:



a. Under Authentication Type, select the Keytab file checkbox.

Important: To generate a keytab named <*USERNAME*>.keytab manually for a user using `ktutil`, run the following commands:

```
sudo ktutil
ktutil:  addent -password -p <USERNAME>@EXAMPLE.COM -k 1 -f
Password for <USERNAME>@EXAMPLE.COM:
ktutil:  addent -password -p <USERNAME>@EXAMPLE.COM -k 2 -f
Password for <USERNAME>@EXAMPLE.COM:
ktutil:  wkt <USERNAME>.keytab
ktutil:  q
```

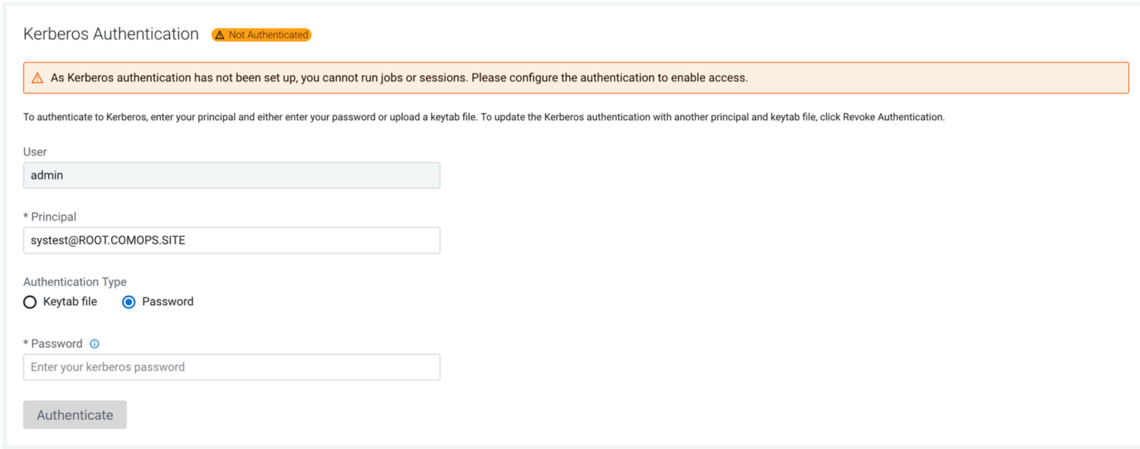
For example, if you want to generate a keytab named psherman.keytab, run the following commands:

```
sudo ktutil
ktutil:  addent -password -p psherman@EXAMPLE.COM -k 1 -f
Password for psherman@EXAMPLE.COM:
ktutil:  addent -password -p psherman@EXAMPLE.COM -k 2 -f
Password for psherman@EXAMPLE.COM:
ktutil:  wkt psherman.keytab
ktutil:  q
```

b. Click Select File and select the relevant keytab file. For instructions about getting the keytab file, see Kerberos Configuration Strategies for CDP.

- Using password:

Note: This option is available only from Cloudera Data Engineering 1.5.5 SP1 onward.



a. Under the Authentication Type, select the Password checkbox.
b. In the Password text box, enter the Kerberos password.

7. Click Authenticate.

For Cloudera Data Engineering CLI

Note: You can perform Hadoop authentication using CDE CLI from Cloudera Data Engineering 1.5.5 SP1 onward.

Login to CDE CLI and run the following command to configure the Hadoop authentication:

```
cde kerberos authenticate [***KERBEROS-AUTHENTICATION-FLAG***]
```

Use one of the following flag value pairs depending on the type of authentication you want to use:

- Using a password:

  Note: This option is available only from Cloudera Data Engineering 1.5.5 SP1 onward.

  - --principal string – Enter the Kerberos principal when prompted
  - --password – Enter the password when prompted
- Using the Keytab file:
  - --principal string – Enter the Kerberos principal when prompted
  - --keytab-file string – Enter the keytab file path when prompted

For Cloudera Data Engineering API

1. Get the Keytab metadata. For instructions about getting the keytab file, see Kerberos Configuration Strategies for CDP.

   ```
   curl -X GET -H "Authorization: Bearer ${CDE_TOKEN}" \
   <SERVICE-URL>/user-auth/api/v1/kerberos
   ```

   To get the CDE_TOKEN and service-url values, see Getting a Cloudera Data Engineering API access token.

   Important: To get the CDE_TOKEN value for Machine Users, see Using CDP Access keys tab in Getting a Cloudera Data Engineering API access token page.

2. Update Hadoop Authenticaion using one of the following methods:
   - Using a password:

     Note: This option is available only from Cloudera Data Engineering 1.5.5 SP1 onward.

     ```
     curl -H "Authorization: Bearer ${CDE_TOKEN}" <SERVICE-URL>/user-auth/
     api/v1/kerberos \
     -H "Content-Type: application/json" \
     -X POST
     --form 'principal="<PRINCIPAL>"' \
     --form 'password="<PASSWORD>"'
     ```
   - Using the Keytab file:

     ```
     curl -H "Authorization: Bearer ${CDE_TOKEN}" <SERVICE-URL>/user-auth/
     api/v1/kerberos \
     -H "Content-Type: application/json" \
     -X POST
     --form 'principal="<PRINCIPAL>"' \
     --form 'file=@"</PATH/TO/KEYTAB/FILE>"'
     ```

Related Information

Creating a machine user in Cloudera

Generate Access Key

Updating Hadoop Authentication

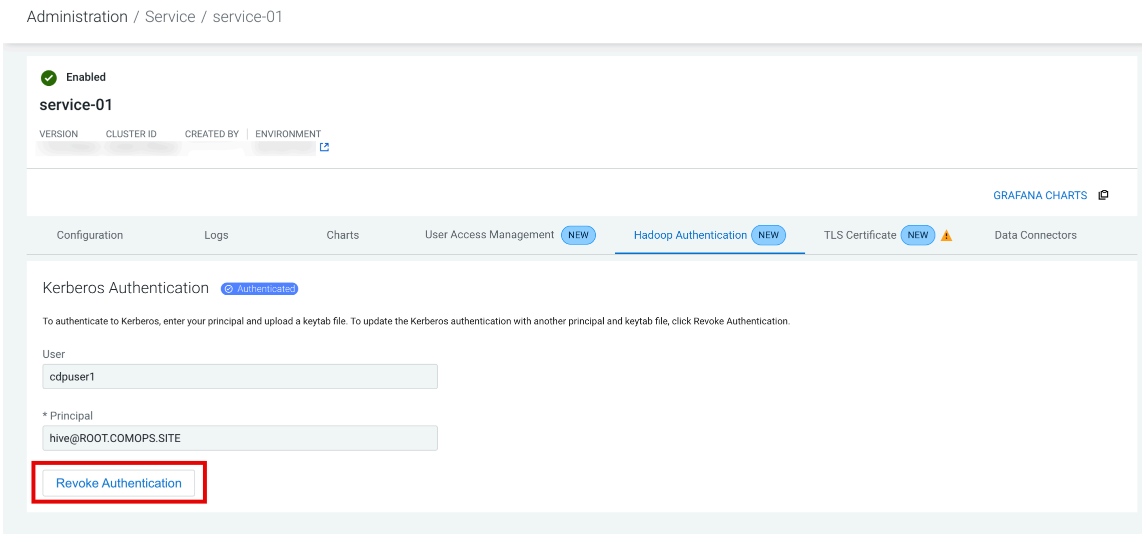
Learn about how to update the Kerberos Keytab file for a Cloudera Data Engineering service.

Important:

- Before you revoke Hadoop Authentication for a Cloudera Data Engineering service, ensure that no active jobs or sessions are running in any of the Virtual Clusters. Using the Revoke Authentication is not allowed if any job or session for the logged-in user is running. Ensure that all the jobs or sessions for the logged-in user are stopped and the scheduled jobs are paused, when Revoke Authentication is clicked.
- Hadoop Authentication for Machine Users can be only configured through Cloudera Data Engineering API.

For Cloudera Data Engineering UI

1. Revoke Hadoop Authentication:
    a. In the Cloudera console, click the Data Engineering tile. The Cloudera Data Engineering Home page displays.
    b. Click Administration in the left navigation menu. The Administration page displays.
    c. In the Services column, select the environment for which you want to revoke the Hadoop Authentication and click Service Details.
    d. Go to the Hadoop Authentication tab.
    e. Click Revoke Authentication.



    f. Click Confirm.
2. Update the new Kerberos keytab file. For instructions about updating the new keytab file, see Configuring Hadoop Authentication.

For Cloudera Data Engineering CLI

Note: You can revoke Hadoop authentication using CDE CLI from Cloudera Data Engineering 1.5.5 SP1 onward.

Login to CDE CLI and run the following command to revoke the Hadoop authentication:

```
cde kerberos revoke
```

For Cloudera Data Engineering API

1. Revoke the Keytab file.

```
curl -X DELETE -H "Authorization: Bearer ${CDE_TOKEN}" \
<SERVICE-URL>/user-auth/api/v1/kerberos \
```

To get the CDE_TOKEN and service-url values, see Getting a Cloudera Data Engineering API access token.

Important: To get the CDE_TOKEN value for Machine Users, see *Using CDP Access keys* tab on Getting a Cloudera Data Engineering API access token page.

2. Update the new Kerberos keytab file. For instructions about updating the new keytab file, see Configuring Hadoop Authentication.

Related Information

Creating a machine user in Cloudera

Generate Access Key

Cloudera Data Engineering example jobs and sample data

Cloudera Data Engineering provides a suite of example jobs that operate on example data to showcase its core capabilities and make the onboarding easier. The example jobs are a combination of Spark and Airflow jobs, which include scenarios such as reading and writing from object storage, running an Airflow DAG, and expanding on Python capabilities with custom virtual environments. Once loaded, these jobs can be run on demand or scheduled. The sample data will be loaded into the environment's default Data Lake location.

Before you begin

In Cloudera Data Engineering, jobs are associated with virtual clusters. Before you can create a job, you must create a virtual cluster that can run it. For more information, see Creating virtual clusters.

About this task

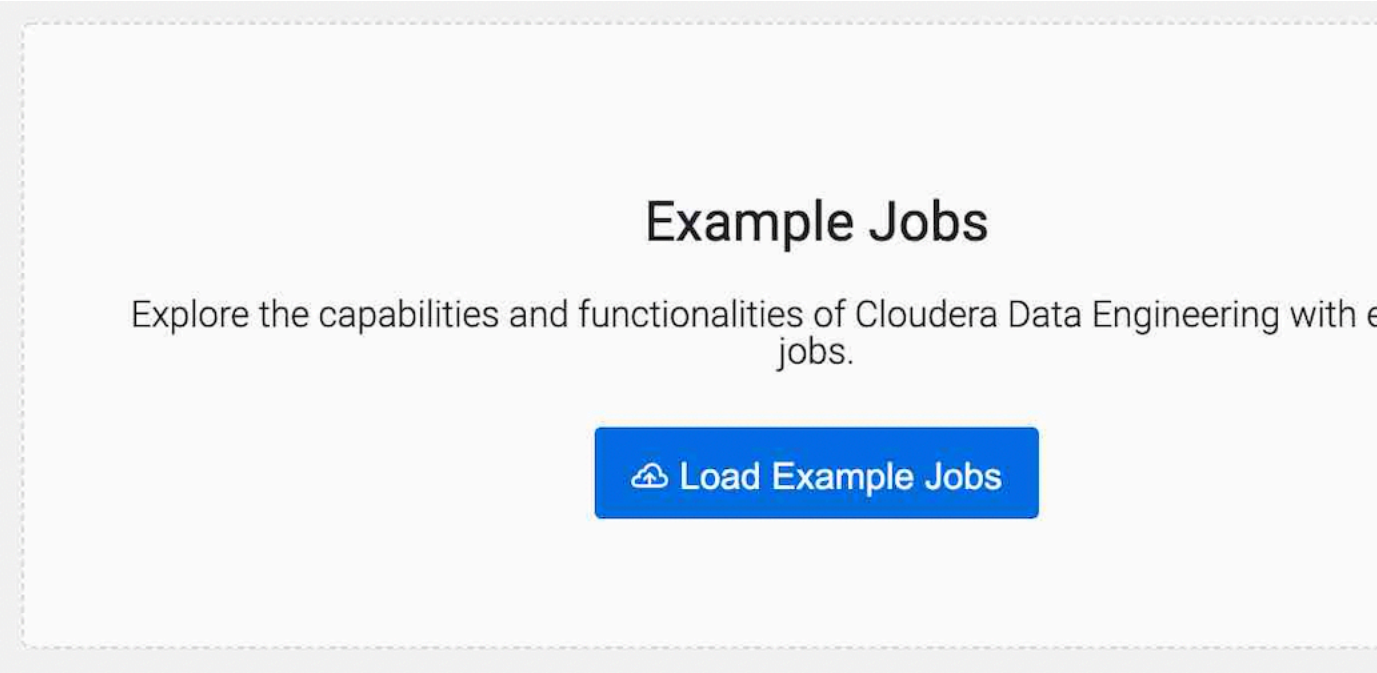
You must run the example jobs with a user who is not the Local Administrator, that is, the user must to have been granted DEUser or DEAdmin privileges in the environment associated with your DE workspace. Also ensure you have enough resources to run these example jobs. Below is the description of the different example jobs:

Table 1: Example Jobs

Job	Description
example-load-data	Loads the sample data onto the environment data lake. This job runs only once and is then deleted. **Note:** This will need to be run manually first if the sample jobs are loaded in any user defined virtual clusters. If the example-load-data job fails, contact Cloudera Support to recreate the example-load-data job.
example-virtual-env	Demonstrates Cloudera Data Engineering job configuration that utilizes Python Environment resource type to expand pyspark features via custom virtual env. This example adds pandas support. **Note:** You cannot run this job in an air-gapped environment.
example-resources	Demonstrates Cloudera Data Engineering job configuration utilizing file-based resource type. Resources are mounted on Spark driver and executor pods. This example uses an input file as a data source for a word-count Spark app. The driver stderr log contains the word count.
example-resources-schedules	Demonstrates scheduling functionality for Spark job in Cloudera Data Engineering. This example schedules a job to run at 5:04am UTC each day.
example-spark-pi	Demonstrates how to define a Cloudera Data Engineering job. It runs a SparkPi using a scala example jar located on a s3 bucket. The driver stderr log contains the value of pi. **Note:** You cannot run this job in an air-gapped environment.
example-cdeoperator	Demonstrates job orchestration using Airflow. This example uses a custom Cloudera Data Engineering Operator to run two Spark jobs in sequence, mimicking a pipeline composed of data ingestion and data processing. **Note:** You cannot run this job in an air-gapped environment.
example-object-store	Demonstrates how to access and write data to object store on different form factors: S3, ADLS, and HDFS. This example reads data already staged to object store and makes changes and then saves back the transformed data to object store. The output of the query ran on the object store table can be viewed in the driver stderr log.

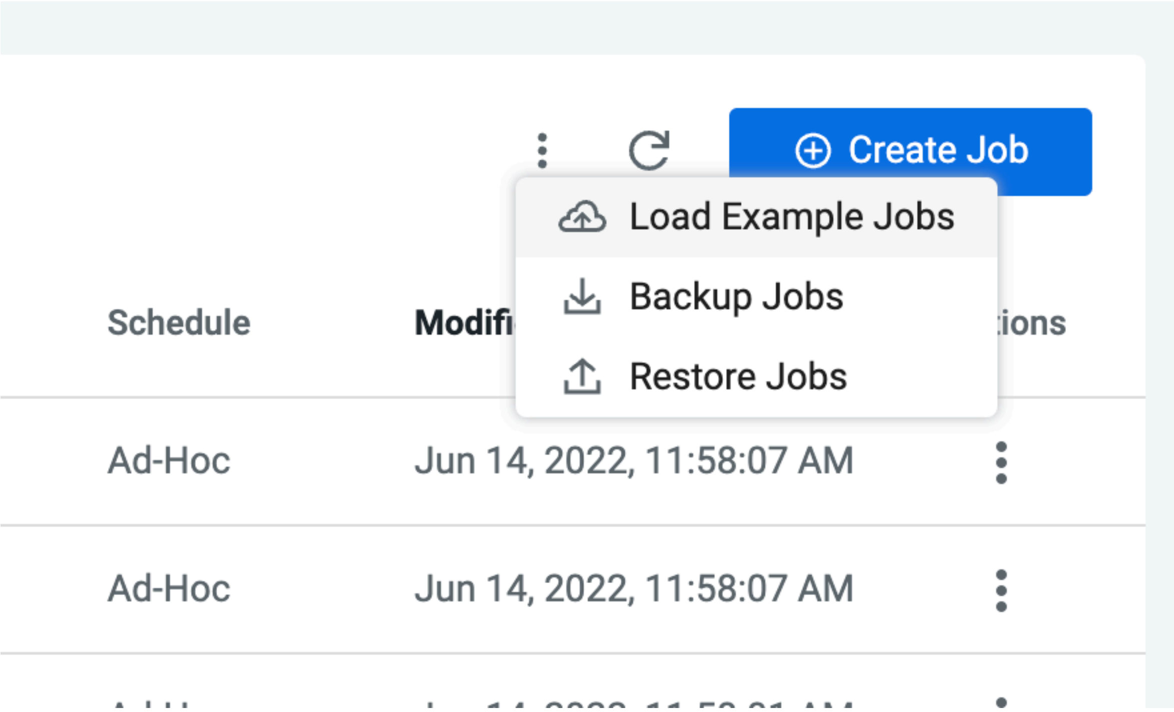
Procedure

1. In the Cloudera console, click the Data Engineering tile. The Cloudera Data Engineering Home page displays.
2. Click Jobs on the left navigation menu.
3. Select Load Example Jobs from the two options that appear.

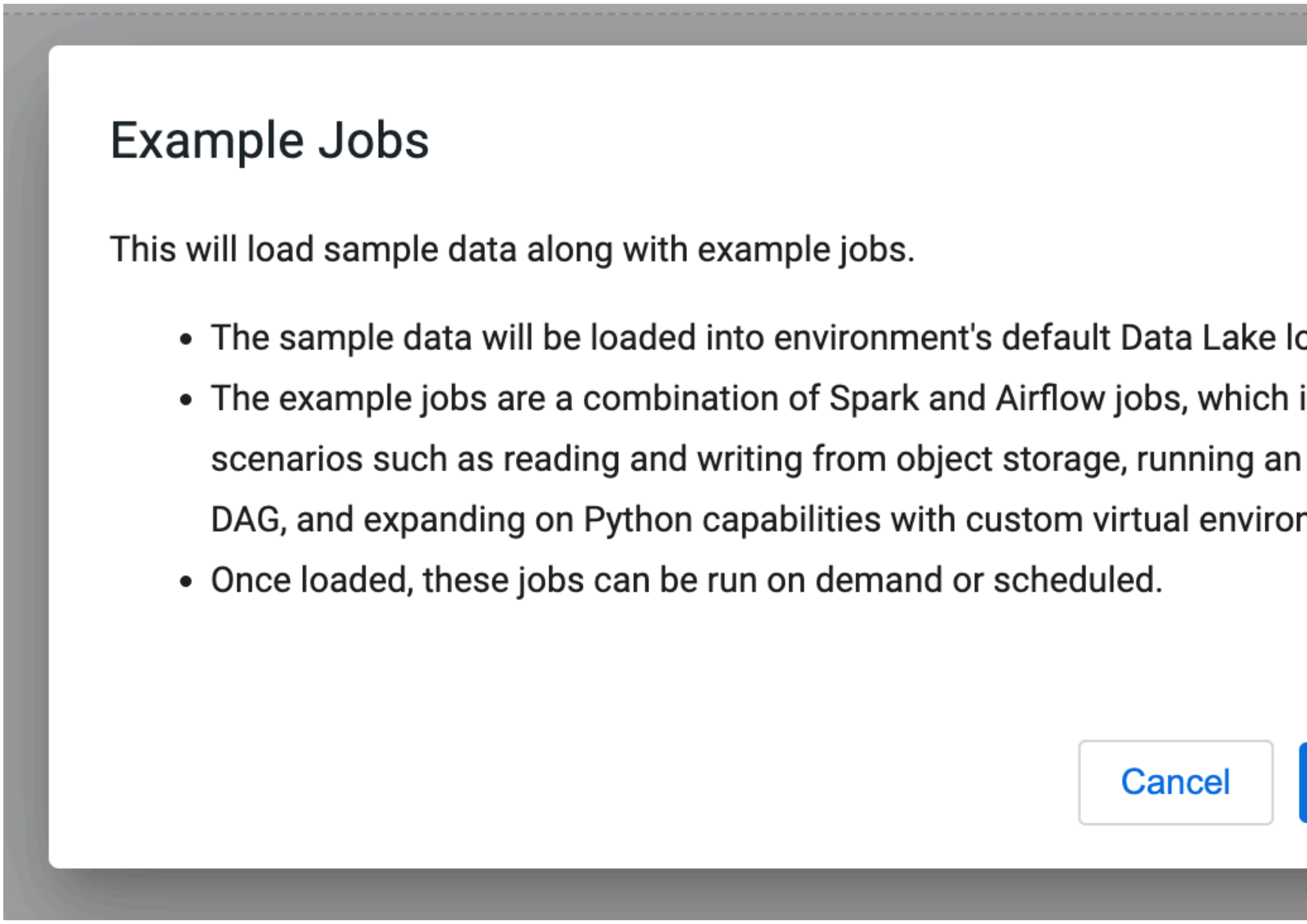


Note: You will see this window only if you have no existing jobs in the virtual cluster.

4. If you have existing jobs in the virtual cluster, click ⋮ on the jobs page to Load Example Jobs.



5. A dialog box appears explaining the example jobs and sample data. Click Confirm to load example jobs and sample data.



Results

Example jobs will be loaded in the virtual cluster and sample data will be loaded in the environment's Data Lake location.

Using Apache Iceberg in Cloudera Data Engineering

Cloudera Data Engineering supports Apache Iceberg which provides a table format for huge analytic datasets in the cloud. Iceberg enables you to work with large tables, especially on object stores, and supports concurrent reads and writes on all storage media. You can use Cloudera Data Engineering virtual clusters running Spark 3 to interact with Apache Iceberg tables.

Prerequisites and limitations for using Iceberg

To use Apache Iceberg in Cloudera Data Engineering, you'll need the following prerequisites:

- Cloudera Data Engineering Virtual Cluster with Spark 3.2.1 or higher
- Cloudera Base on premises 7.1.7 SP2, 7.1.8, or 7.1.9

Limitations

Spark streaming is not supported when using Iceberg from Spark.

Iceberg table format version 2

Iceberg table format version 2 (v2) is available starting in Iceberg 0.14. Iceberg table format v2 uses row-level UPDATE and DELETE operations that add deleted files to encoded rows that were deleted from existing data files. The DELETE, UPDATE, and MERGE operations function by writing delete files instead of rewriting the affected data files. Additionally, upon reading the data, the encoded deletes are applied to the affected rows that are read. This functionality is called merge-on-read.

To use Iceberg table format v2, you'll need the following prerequisites:

- Iceberg 0.14
- Spark 3.2 or higher

  With Iceberg table format version 1 (v1), the above-mentioned operations are only supported with copy-on-write where data files are rewritten in their entirety when rows in the files are deleted. Merge-on-read is more efficient for writes, while copy-on-write is more efficient for reads.

  Note: Unless otherwise indicated, the operations in the subsequent documentation apply to both v1 and v2 formats.

Iceberg timestamp

- Iceberg supports two timestamp types:
  - timestamp (without timezone)
  - timestamptz (with timezone)

  In Spark 3.3 and earlier, Spark SQL supports a single TIMESTAMP type, which maps to the Iceberg timestamptz type. However, Impala is unable to write to Iceberg tables with timestamptz columns. To create Iceberg tables from Spark with timestamp rather than timestamptz columns, set the following configurations to true:

  - `spark.sql.iceberg.handle-timestamp-without-timezone`
  - `spark.sql.iceberg.use-timestamp-without-timezone-in-new-tables`

  Configure these properties only on Spark 3.3 and earlier.

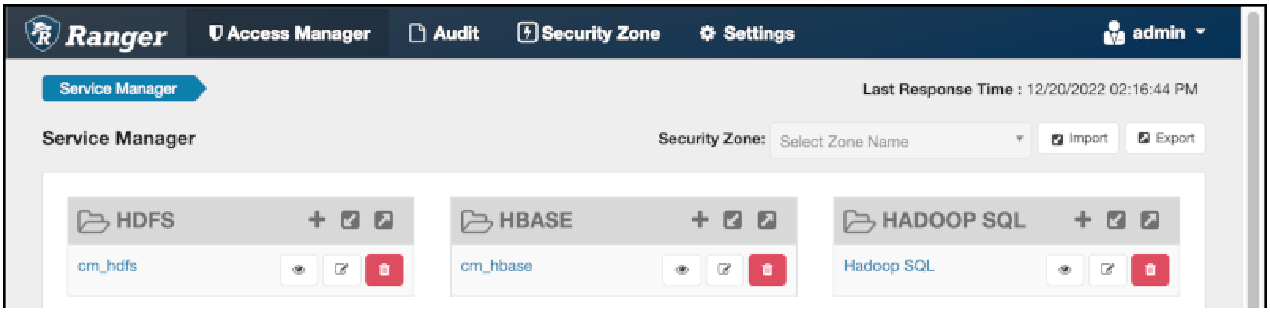
  Spark still handles the timestamp column as a timestamp with local timezone. Inconsistent results occur unless Spark is running in UTC.

Accessing Iceberg tables

Cloudera uses Apache Ranger to provide centralized security administration and management. The Ranger Admin UI is the central interface for security administration. You can use Ranger to create two policies that allow users to query Iceberg tables.

How you open the Ranger Admin UI differs from one Cloudera service to another. In Cloudera Management Console, you can select your environment, and then click Environment Details Quick Links Ranger .

You log into the Ranger Admin UI, and the Ranger Service Manager appears.



Policies for accessing tables on HDFS

The default policies that appear differ from service to service. You need to set up two Hadoop SQL policies to query Iceberg tables:

- One to authorize users to access the Iceberg files

  Follow steps in "Editing a policy to access Iceberg files" below.
- One to authorize users to query Iceberg tables

  Follow steps in "Creating a policy to query an Iceberg table" below.

Prerequisites

- Obtain the RangerAdmin role.
- Get the user name and password your Administrator set up for logging into the Ranger Admin.

  The default credentials for logging into the Ranger Admin Web UI are admin/admin123.

Editing a storage handler policy to access Iceberg files on the file system

You learn how to edit the existing default Hadoop SQL Storage Handler policy to access files. This policy is one of the two Ranger policies required to use Iceberg.

About this task

The Hadoop SQL Storage Handler policy allows references to Iceberg table storage location, which is required for creating or altering a table. You use a storage handler when you create a file stored as Iceberg on the file system or object store.

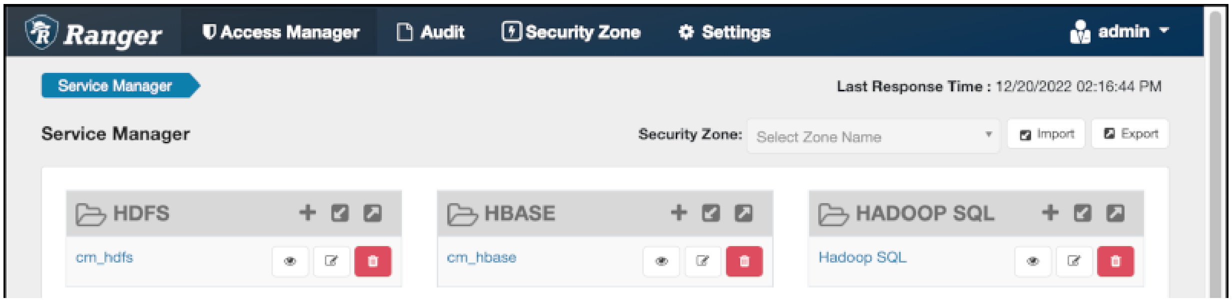
In this task, you specify Iceberg as the storage-type and allow the broadest access by setting the URL to *.

The Hadoop SQL Storage Handler policy supports only the RW Storage permission. A user having the required RW Storage permission on a resource, such as Iceberg, that you specify in the storage-type properties, is allowed only to reference the table location (for create/alter operations) in Iceberg. The RW Storage permission does not provide access to any table data. You need to create the Hadoop SQL policy described in the next topic in addition to this Hadoop SQL Storage Handler policy to access data in tables.

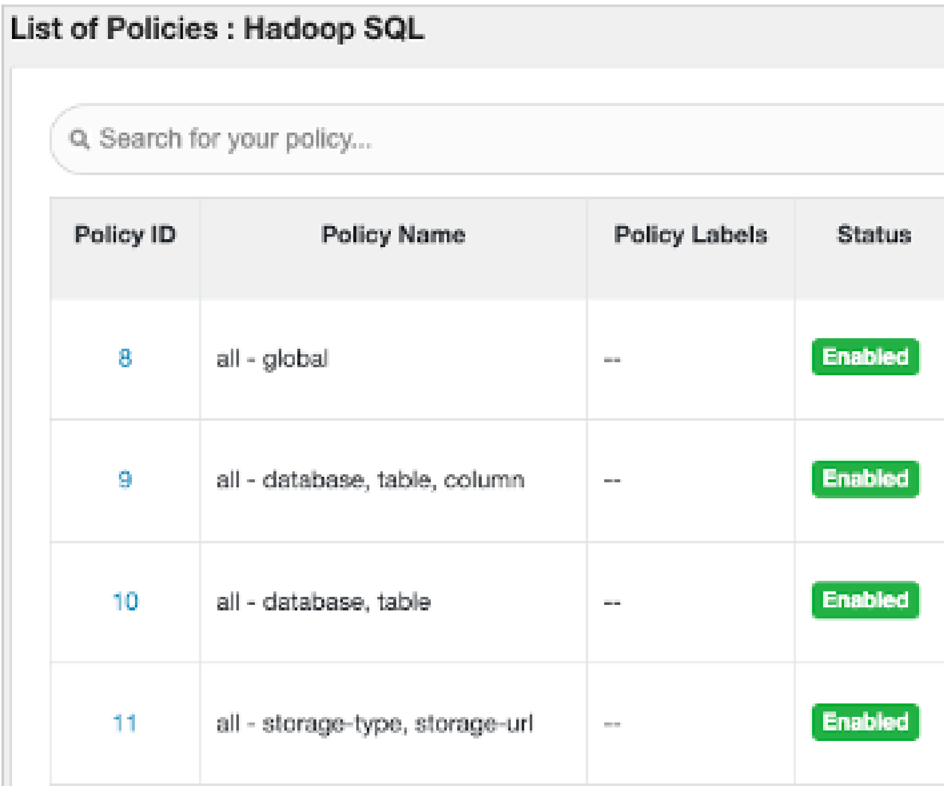
For more information about these policy settings, see Ranger Storage Handler documentation.

Procedure

1. Log into Ranger Admin Web UI.
   The Ranger Service Manager appears:

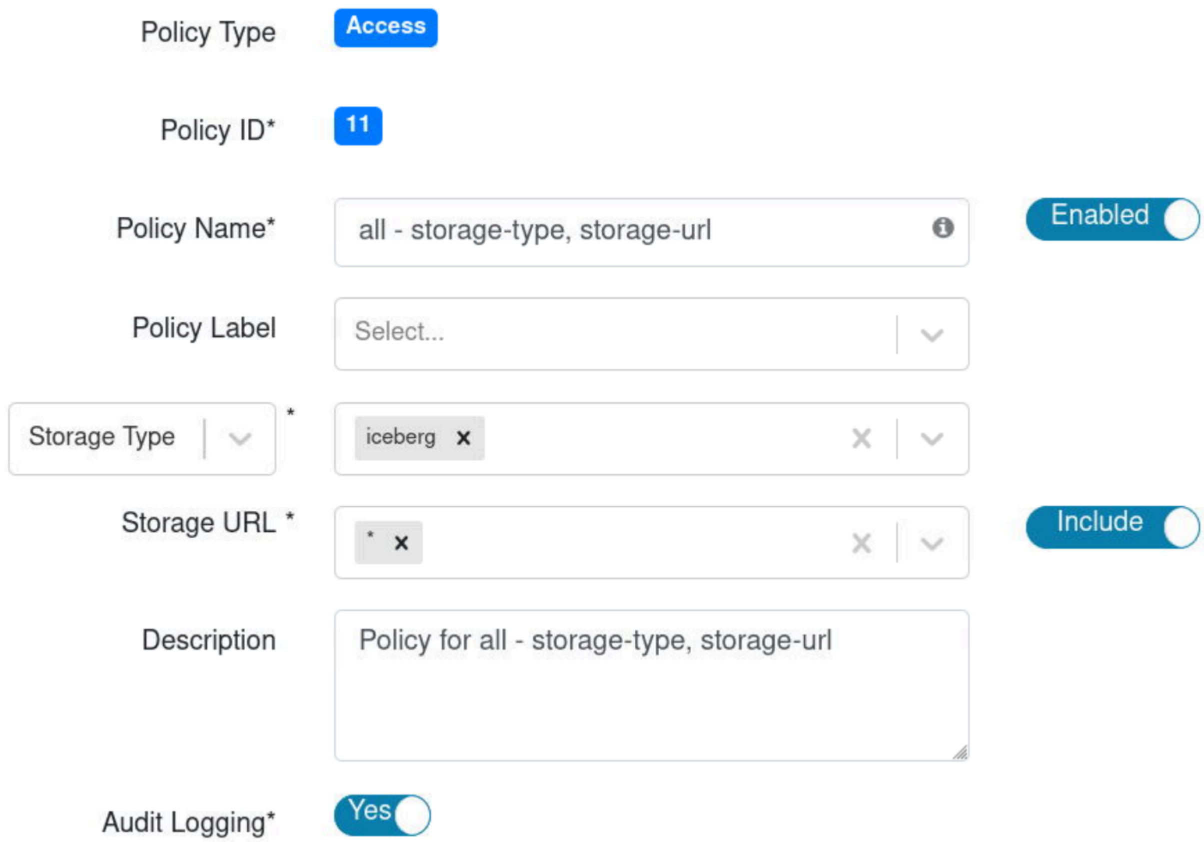


2. In Policy Name, enable the all - storage-type, storage-url policy.



3. In **Service Manager**, in Hadoop SQL, select Edit and edit the all storage-type, storage-url policy.
4. Below Policy Label, select storage-type, and enter iceberg..

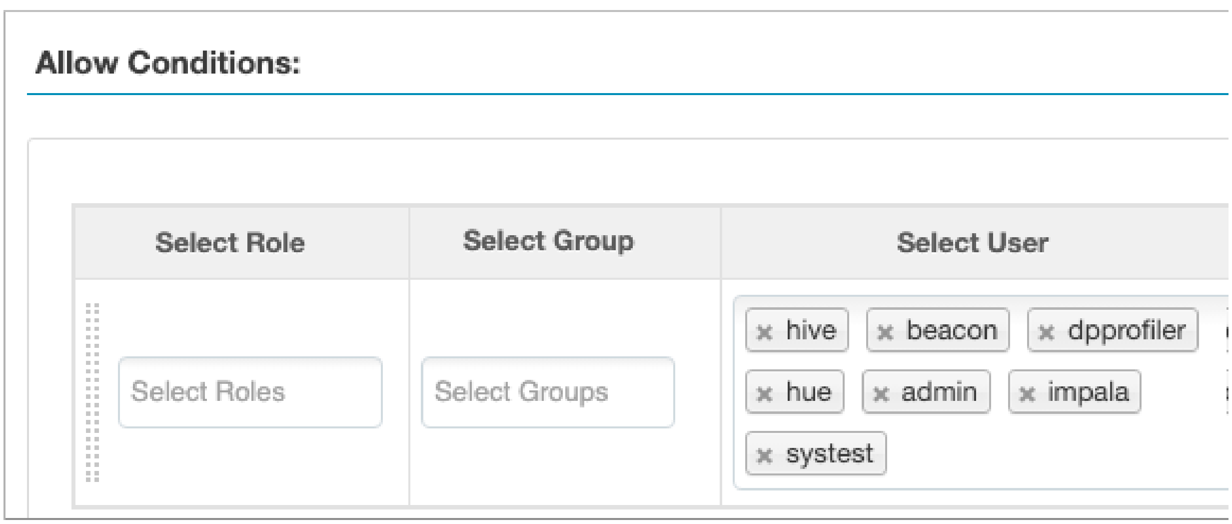
5. In Storage URL, enter the value *, enable Include.



For more information about these policy settings, see Ranger storage handler documentation.

6. In Allow Conditions, specify roles, users, or groups to whom you want to grant RW storage permissions.

You can specify PUBLIC to grant access to Iceberg tables permissions to all users. Alternatively, you can grant access to one user. For example, add the systest user to the list of users who can access Iceberg:



For more information about granting permissions, see Configure a resource-based policy: Hadoop-SQL.

7. Add the RW Storage permission to the policy.
8. Save your changes.

Creating a SQL policy to query an Iceberg table

You learn how to set up the second required policy for using Iceberg. This policy manages SQL query access to Iceberg tables.

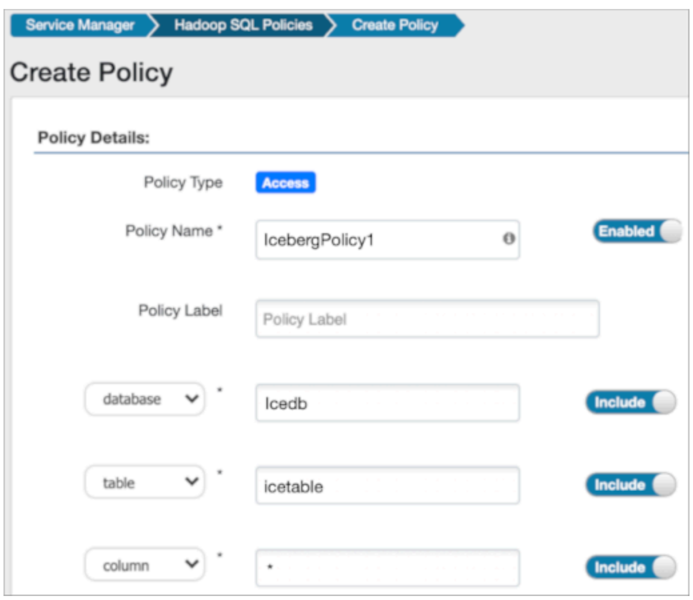
About this task

You create a Hadoop SQL policy to allow roles, groups, or users to query an Iceberg table in a database. In this task, you see an example of just one of many ways to configure the policy conditions. You grant (allow) the selected roles, groups, or users the following add or edit permissions on the table: Select, Update, Create, Drop, Alter, and All. You can also deny permissions.

For more information about creating this policy, see Ranger documentation.

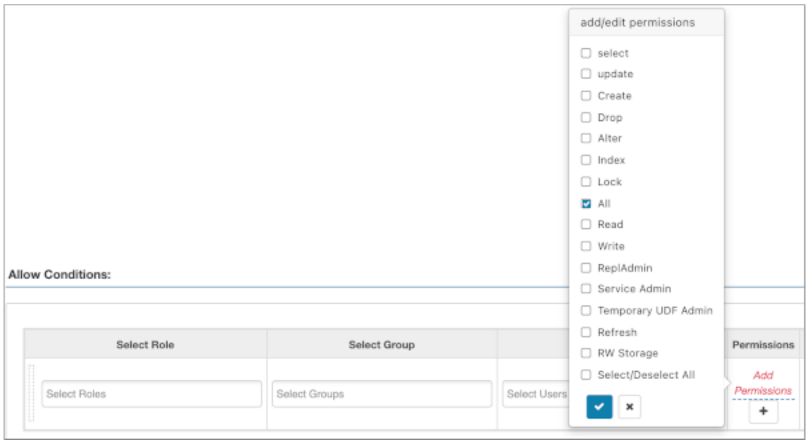
Procedure

1. Log into Ranger Admin Web UI.
   The Ranger Service Manager appears.
2. Click Add New Policy.
3. Fill in required fields.
   For example, enter the following required settings:
   - In Policy Name, enter the name of the policy, for example IcebergPolicy1.
   - In database, enter the name of the database controlled by this policy, for example icedb.
   - In table, enter the name of the table controlled by this policy, for example icetable.
   - In columns, enter the name of the column controlled by this policy, for example enter the wildcard asterisk (*) to allow access to all columns of icetable.
   - Accept defaults for other settings.



4. Scroll down to Allow Conditions, and select the roles, groups, or users you want to access the table.
   You can use Deny All Other Accesses to deny access to all other roles, groups, or users other than those specified in the allow conditions for the policy.

5. Select permissions to grant.
   For example, select Create, Select, and Alter. Alternatively, to provide the broadest permissions, select All.

   Ignore RW Storage and other permissions not named after SQL queries. These are for future implementations.
6. Click Add.

Creating Virtual Cluster with Spark 3

Create a virtual cluster with Spark 3 as the Spark version.

For more information on creating virtual clusters, see Creating virtual clusters.

Creating and running Spark jobs using Iceberg tables

Create and run a spark job which uses iceberg tables.

Before you begin

Important: If your application code directly uses Iceberg APIs, you need to build it against the Iceberg dependencies. For more information, see Iceberg library dependencies for Spark applications on page 27.

In Cloudera Data Engineering, jobs are associated with virtual clusters. Before you can create a job, you must create a virtual cluster that can run it. For more information, see Creating virtual clusters.

Procedure

1. In the Cloudera console, click the Data Engineering tile. The Cloudera Data Engineering Home page displays.
2. In the Cloudera Data Engineering Home page, in Jobs, click Create New under Spark or click Jobs in the left navigation menu and then click Create Job.
3. Provide the Job Details:
   a) Select Spark for the job type. If you are creating the job from the Home page, select the virtual cluster where you want to create the job.
   b) Specify the Name.
   c) Select File or URL for your application file, and provide or specify the file. You can upload a new file or select a file from an existing resource.
   d) If your application code is a JAR file, specify the Main Class.
   e) Click the Add Configuration button to add the following configuration parameters:

```
config_key: cde.iceberg.enabled
```

```
config_value: true
```

f) If your application code is a Python file, select the Python Version, and optionally select a Python Environment.

4. Click Advanced Configurations to display more customizations, such as additional files, initial executors, executor range, driver and executor cores, and memory.

   By default, the executor range is set to match the range of CPU cores configured for the virtual cluster. This improves resource utilization and efficiency by allowing jobs to scale up to the maximum virtual cluster resources available, without manually tuning and optimizing the number of executors per job.
5. Click Schedule to display scheduling options.

   You can schedule the application to run periodically using the Basic controls or by specifying a Cron Expression.
6. Click Alerts and provide the email id to receive alerts. Click + to add more email IDs. Optionally, you can select when you want email alerts whether for job failures or missed job service-level agreements or both.

   Note: You must configure the Configure Email Alerting option while creating a virtual cluster to send your email alerts. For more information about configuring email alerts, see Creating virtual clusters.
7. If you provided a schedule, click Schedule to create the job. If you did not specify a schedule, and you do not want the job to run immediately, click the drop-down arrow on Create and Run and select Create. Otherwise, click Create and Run to run the job immediately.

Creating a new Iceberg table from Spark 3

You can create an Iceberg table using Spark SQL.

Note: By default, Iceberg tables are created in the v1 format.

An example Spark SQL creation command to create a new Iceberg table is as follows:

```
spark.sql("""CREATE EXTERNAL TABLE ice_t (idx int, name string, state string
)
USING iceberg
PARTITIONED BY (state)""")
```

For information about creating tables, see the Iceberg documentation.

Creating an Iceberg table format v2

To use the Iceberg table format v2, set the format-version property to 2 as shown below:

```
CREATE TABLE logs (app string, lvl string, message string, event_ts timestam
p) USING iceberg TBLPROPERTIES ('format-version' = '2')
```

<delete-mode> <update-mode> and <merge-mode> can be specified during table creation for modes of the respective operation. If unspecified, they default to merge-on-read.

Unsupported Feature: CREATE TABLE … LIKE

The CREATE TABLE ... LIKE feature is not supported in Spark:

```
CREATE TABLE <target> LIKE <source> USING iceberg
```

Here, `<source>` is an existing Iceberg table. This operation may appear to succeed and does not display errors and only warnings, but the resulting table is not a usable table.

Configuring Hive Metastore for Iceberg column changes

To make schema changes to an existing column of an Iceberg table, you must configure the Hive Metastore of the Data Lake.

Procedure

1. In Cloudera Manager, select the service for the Hive Metastore.
2. Click the Configuration tab.
3. Search for safety valve and find the Hive Metastore Server Advanced Configuration Snippet (Safety Valve) for hive-site.xml safety valve.
4. Add the following property:
   - Name: hive.metastore.disallow.incompatible.col.type.changes
   - Value: false
5. Click Save Changes.
6. Restart the service to apply the configuration change.

Importing and migrating Iceberg table in Spark 3

Importing or migrating tables are supported only on existing external Hive tables. When you import a table to Iceberg, the source and destination remain intact and independent. When you migrate a table, the existing Hive table is converted into an Iceberg table. You can use Spark SQL to import or migrate a Hive table to Iceberg.

Importing

Call the snapshot procedure to import a Hive table into Iceberg using a Spark 3 application.

```
spark.sql("CALL
<catalog>.system.snapshot('<src>', '<dest>')")
```

Definitions:

- <src> is the qualified name of the Hive table
- <dest> is the qualified name of the Iceberg table to be created
- <catalog> is the name of the catalog, which you pass in a configuration file. For more information, see *Configuring Catalog*.

For example:

```
spark.sql("CALL
spark_catalog.system.snapshot('hive_db.hive_tbl',
'iceberg_db.iceberg_tbl')")
```

For information on compiling Spark 3 application with Iceberg libraries, see Iceberg library dependencies for Spark applications linked below.

Migrating

When you migrate a Hive table to Iceberg, a backup of the table, named <table_name>_backup_, is created.

Ensure that the TRANSLATED_TO_EXTERNAL property, that is located in TBLPROPERTIES, is set to false before migrating the table. This ensures that a table backup is created by renaming the table in Hive metastore (HMS) instead of moving the physical location of the table. Moving the physical location of the table would entail copying files in Amazon s3.

We recommend that you refrain from dropping the backup table, as doing so will invalidate the newly migrated table.

If you want to delete the backup table, set the following:

```
'external.table.purge'='FALSE'
```

Note: For Cloudera Data Engineering on premises 1.5.2 and above, the property will be set automatically.

Deleting the backup table in the manner above will prevent underlying data from being deleted, therefore, only the table will be deleted from the metastore.

To undo the migration, drop the migrated table and restore the Hive table from the backup table by renaming it.

Call the migrate procedure to migrate a Hive table to Iceberg.

```
spark.sql("CALL
<catalog>.system.migrate('<src>')")
```

Definitions:

- <src> is the qualified name of the Hive table
- <catalog> is the name of the catalog, which you pass in a configuration file. For more information, see Configuring Catalog linked below.

For example:

```
spark.sql("CALL
spark_catalog.system.migrate('hive_db.hive_tbl;)")
```

Related Concepts

Configuring Catalog

Related reference

Iceberg library dependencies for Spark applications

Importing and migrating Iceberg table format v2

Importing or migrating Hive tables Iceberg table formats v2 are supported only on existing external Hive tables. When you import a table to Iceberg, the source and destination remain intact and independent. When you migrate a table, the existing Hive table is converted into an Iceberg table. You can use Spark SQL to import or migrate a Hive table to Iceberg.

Importing

Call the snapshot procedure to import a Hive table into Iceberg table format v2 using a Spark 3 application.

```
spark.sql("CALL
<catalog>.system.snapshot(source_table => '<src>',
table => '<dest>',
properties => map('format-version', '2', 'write.delete.mode', '<delete-mo
de>',
'write.update.mode', '<update-mode>',
'write.merge.mode', '<merge-mode>'))")
```

Definitions:

- <src> is the qualified name of the Hive table
- <dest> is the qualified name of the Iceberg table to be created
- <catalog> is the name of the catalog, which you pass in a configuration file. For more information, see Configuring Catalog linked below.

- <delete-mode> <update-mode> and <merge-mode> are the modes that shall be used to perform the respective operation. If unspecified, they default to 'merge-on-read'

For example:

```
spark.sql("CALL
spark_catalog.system.snapshot('hive_db.hive_tbl',
'iceberg_db.iceberg_tbl')")
```

For information on compiling Spark 3 application with Iceberg libraries, see Iceberg library dependencies for Spark applications linked below.

Migrating

Call the migrate procedure to migrate a Hive table to Iceberg.

```
spark.sql("CALL
<catalog>.system.migrate('<src>',
map('format-version', '2',
'write.delete.mode', '<delete-mode>',
'write.update.mode', '<update-mode>',
'write.merge.mode', '<merge-mode>'))")
```

Definitions:

- <src> is the qualified name of the Hive table
- <catalog> is the name of the catalog, which you pass in a configuration file. For more information, see Configuring Catalog linked below.
- <delete-mode> <update-mode> and <merge-mode> are the modes that shall be used to perform the respective operation. If unspecified, they default to 'merge-on-read'

For example:

```
spark.sql("CALL
spark_catalog.system.migrate('hive_db.hive_tbl',
map('format-version', '2',
'write.delete.mode', 'merge-on-read',
'write.update.mode', 'merge-on-read',
'write.merge.mode', 'merge-on-read'))")
```

Upgrading Iceberg table format v1 to v2

To upgrade an Iceberg table format from v1 to v2, run an `ALTER TABLE` command as follows:

```
spark.sql("ALTER TABLE <table_name> SET TBLPROPERTIES('merge-on-read', '
2')")
```

<delete-mode>,<update-mode>, and <merge-mode> can be specified as the modes that shall be used to perform the respective operation. If unspecified, they default to ‘merge-on-read'

Related Concepts

Configuring Catalog

Related reference

Iceberg library dependencies for Spark applications

Configuring Catalog

When using Spark SQL to query an Iceberg table from Spark, you refer to a table using the following dot notation:

```
<catalog_name>.<database_name>.<table_name>
```

The default catalog used by Spark is named spark_catalog. When referring to a table in a database known to spark_ca talog, you can omit <catalog_name>. .

Iceberg provides a SparkCatalog property that understands Iceberg tables, and a SparkSessionCatalog property that understands both Iceberg and non-Iceberg tables. The following are configured by default:

```
spark.sql.catalog.spark_catalog=org.apache.iceberg.spark.SparkSessionCatalog
spark.sql.catalog.spark_catalog.type=hive
```

This replaces Spark's default catalog by Iceberg's SparkSessionCatalog and allows you to use both Iceberg and non-Iceberg tables out of the box.

There is one caveat when using SparkSessionCatalog. Iceberg supports CREATE TABLE … AS SELECT (CTAS) and REPLACE TABLE … AS SELECT (RTAS) as atomic operations when using SparkCatalog. Whereas, the CTAS and RTAS are supported but are not atomic when using SparkSessionCatalog. As a workaround, you can configure another catalog that uses SparkCatalog. For example, to create the catalog named iceberg_catalog, set the following:

```
spark.sql.catalog.iceberg_catalog=org.apache.iceberg.spark.SparkCatalog
spark.sql.catalog.iceberg_catalog.type=hive
```

You can configure more than one catalog in the same Spark job. For more information, see the *Iceberg documentation*.

Related Information

Iceberg documentation

Loading data into an unpartitioned table

You can insert data into an unpartitioned table. The syntax to load data into an iceberg table:

```
INSERT INTO table_identifier [ ( column_list ) ]
    VALUES ( { value | NULL } [ , ... ] ) [ , ( ... ) ]
```

Or

```
INSERT INTO table_identifier [ ( column_list ) ]
    query
```

Example:

```
INSERT INTO students VALUES
    ('Amy Smith', '123 Park Ave, San Jose', 111111)
INSERT INTO students VALUES
    ('Bob Brown', '456 Taylor St, Cupertino', 222222),
    ('Cathy Johnson', '789 Race Ave, Palo Alto', 333333)
```

Querying data in an Iceberg table

To read the Iceberg table, you can use SparkSQL to query the Iceberg tables.

Example:

```
spark.sql("select * from ice_t").show(1000, false)
```

Important: When querying Iceberg tables in HDFS, Spark disables locality by default. Because enabling locality generally leads to increased Spark planning time for queries on such tables and often the increase is quite significant. If you wish to enable locality, set the spark.cloudera.iceberg.locality.enabled to true . For example, you can do it by passing --conf spark.cloudera.iceberg.locality.enabled=true to your spark-submit command.

Updating Iceberg table data

Iceberg table data can be updated using copy-on-write or merge-on-read. The table version you are using will determine how you can update the table data.

v1 format

Iceberg supports bulk updates through MERGE, by defaulting to copy-on-write deletes when using v1 table format.

v2 format

Iceberg table format v2 supports efficient row-level updates and delete operations leveraging merge-on-read.

For more details, refer to *Position Delete Files* linked below.

For updating data examples, see *Spark Writes* linked below.

Related Information

Position Delete Fies

Spark Writes

Iceberg library dependencies for Spark applications

If your Spark application only uses Spark SQL to create, read, or write Iceberg tables, and does not use any Iceberg APIs, you do not need to build it against any Iceberg dependencies. The runtime dependencies needed for Spark to use Iceberg are in the Spark classpath by default. If your code uses Iceberg APIs, then you need to build it against Iceberg dependencies.

Cloudera publishes Iceberg artifacts to a Maven repository with versions matching the Iceberg in Cloudera Data Engineering.

Cloudera publishes Iceberg artifacts to a Maven repository with versions matching the Iceberg in CDS.

Note: For CDH-7.1.x, there are no iceberg jars in the maven repository. Use 0.14.1.1.17.7215.0-27 iceberg version for compilation. The below iceberg dependencies should only be used for compilation. Including iceberg jars within a Spark application fat jar must be avoided.

Note: Use 1.3.0.7.1.9.0-387 iceberg version for compilation. The below iceberg dependencies should only be used for compilation. Including iceberg jars within a Spark application fat jar must be avoided.

```
<dependency>
     <groupId>org.apache.iceberg</groupId>
     <artifactId>iceberg-core</artifactId>
     <version>${iceberg.version}</version>
     <scope>provided</scope>
</dependency>
<!-- for org.apache.iceberg.hive.HiveCatalog -->
```

```
<dependency>
     <groupId>org.apache.iceberg</groupId>
     <artifactId>iceberg-hive-metastore</artifactId>
     <version>${iceberg.version}</version>
     <scope>provided</scope>
</dependency>
<!-- for org.apache.iceberg.spark.* classes if used -->
<dependency>
    <groupId>org.apache.iceberg</groupId>
    <artifactId>iceberg-spark</artifactId>
    <version>${iceberg.version}</version>
    <scope>provided</scope>
</dependency>
```

Alternatively, the following dependency can be used:

```
<dependency>
   <groupId>org.apache.iceberg</groupId>
   <artifactId>iceberg-spark3-runtime</artifactId>
   <version>${iceberg.version}</version>
   <scope>provided</scope>
</dependency>
```

Alternatively, the following dependency can be used:

```
<dependency>
   <groupId>org.apache.iceberg</groupId>
   <artifactId>iceberg-spark-runtime-3.3_2.12</artifactId>
   <version>${iceberg.version}</version>
   <scope>provided</scope>
</dependency>
```

The iceberg-spark3-runtime JAR contains the necessary Iceberg classes for Spark runtime support, and includes the classes from the dependencies above.

After compiling the job, you can create and run Cloudera Data Engineering jobs. For more information see, Creating Spark jobs and Running a Spark job.

Creating a repository in Cloudera Data Engineering

Repositories allow teams to collaborate, manage project artifacts, and promote applications from lower to higher environments. Cloudera currently supports Git providers such as GitHub, GitLab, and Bitbucket. Learn how to use Cloudera Data Engineering with version control service.

About this task

Repository files can be accessed when you create a Spark or Airflow job. You can then deploy the job and use Cloudera Data Engineering's centralized monitoring and troubleshooting capabilities to tune and adjust your workloads. Cloudera Data Engineering automatically clones the project files and folders when a repository is created. Metadata such as file size and hash are also available. These files display as a read-only view in the Cloudera Data Engineering UI and users cannot delete or modify the files. This ensures a single source of truth and simplifies promotions.

Note: You can reference the same resource file at the maximum 177 times in job runs. Therefore, you can only run 177 jobs simultaneously that reference the same resource file.

Supported version control service providers: Cloudera currently supports the following version control service providers:

- GitHub
- GitLab
- Bitbucket

Before you begin

To use a non-public Git repository, you must first create repository credentials using a workload secret for Cloudera Data Engineering using the CDE CLI as follows:

cde credential create --type basic --username myuser --name my-credential

The command above prompts you for a password where you can either provide your Personal Access Token (PAT) or provide a password for your Git repository account, for example, Github.

Note: When using the password based authentication, make sure your two factor authentication (2FA) for your Github account is turned off. Cloudera recommends turning it off because the 2FA doesn't get the authentication requests when the source is an API.

Limitations

When you create a Git repository in Cloudera Data Engineering, consider the following limitations:

- **Repository creation delay**: Currently, during the repository creation, a prolonged loading screen is displayed due to a synchronous sync call.
- **Incorrect sync status**: The sync status shown is active, even if the syncronization is still in progress. Git hash and file updates for large repositories are delayed even after a sync succe ssful message.
- **Search scope limitation**: The current repository search is not recursive and it only operates within the current folder.

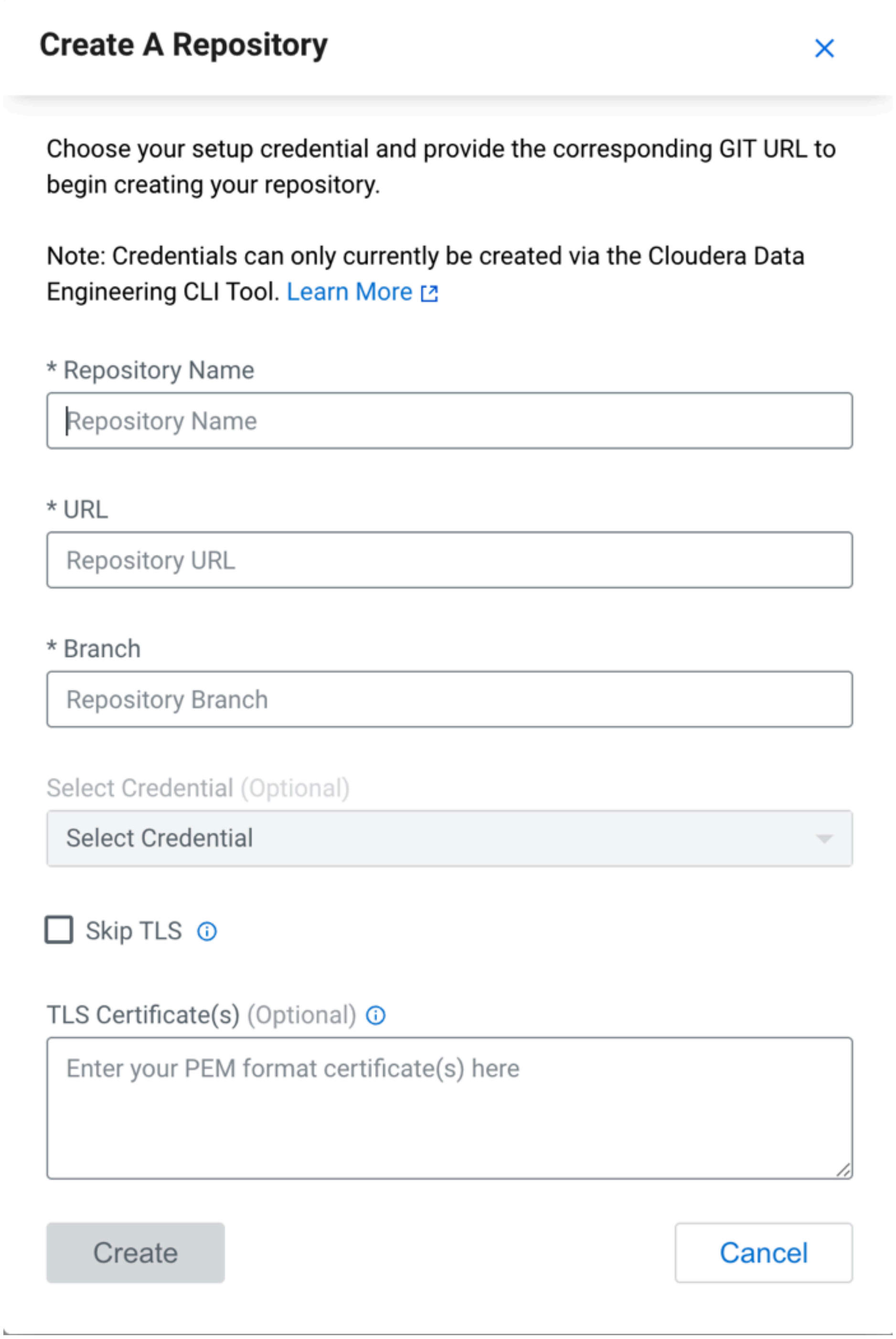
Size limitations

- The maximum size for an individual file is 100MB, which is enforced by GitHub.
- Cloudera recommends the following:
  - Store up to 1000 files in a folder.
  - Store a maximum of 10 000 files in a repository.
  - Keep the total repository size below 1GB.

Procedure

1. In the Cloudera console, click the Data Engineering tile. The Home page displays.
2. Click Repositories in the left navigation menu.The Repositories page displays.

3. Click Create Repository. The Create A Repository dialog box displays. Enter the following fields for the repository:



a) Repository Name - Enter a name for the repository.
b) URL - Enter the repository URL, https only.
c) Branch - Enter the name of the git branch.
d) (Optional) Select a credential from the Select Credential drop-down list. The credentials can be created using the Cloudera Data Engineering CLI.
e) (Optional) Select the Skip TLS checkbox. Select this option if the server uses a self-signed CA certificate that Cloudera Data Engineering does not trust. This allows Cloudera Data Engineering to skip the security check and clone the repository.
f) (Optional) In the TLS Certificate(s) text box, enter the TLS certificate in PEM format.

Note:

- This option is available only from Cloudera Data Engineering 1.5.5 SP1 release onward.
- If you select the Skip TLS checkbox, then the TLS Certificate(s) text box will be disabled.

4. Click Create.

Managing jobs in Cloudera Data Engineering

It is often necessary to modify your Cloudera Data Engineering jobs. Cloudera Data Engineering makes it easy to modify most aspects of your jobs, including replacing the application code and any supplemental files, as well as modifying configuration parameters and the schedule.

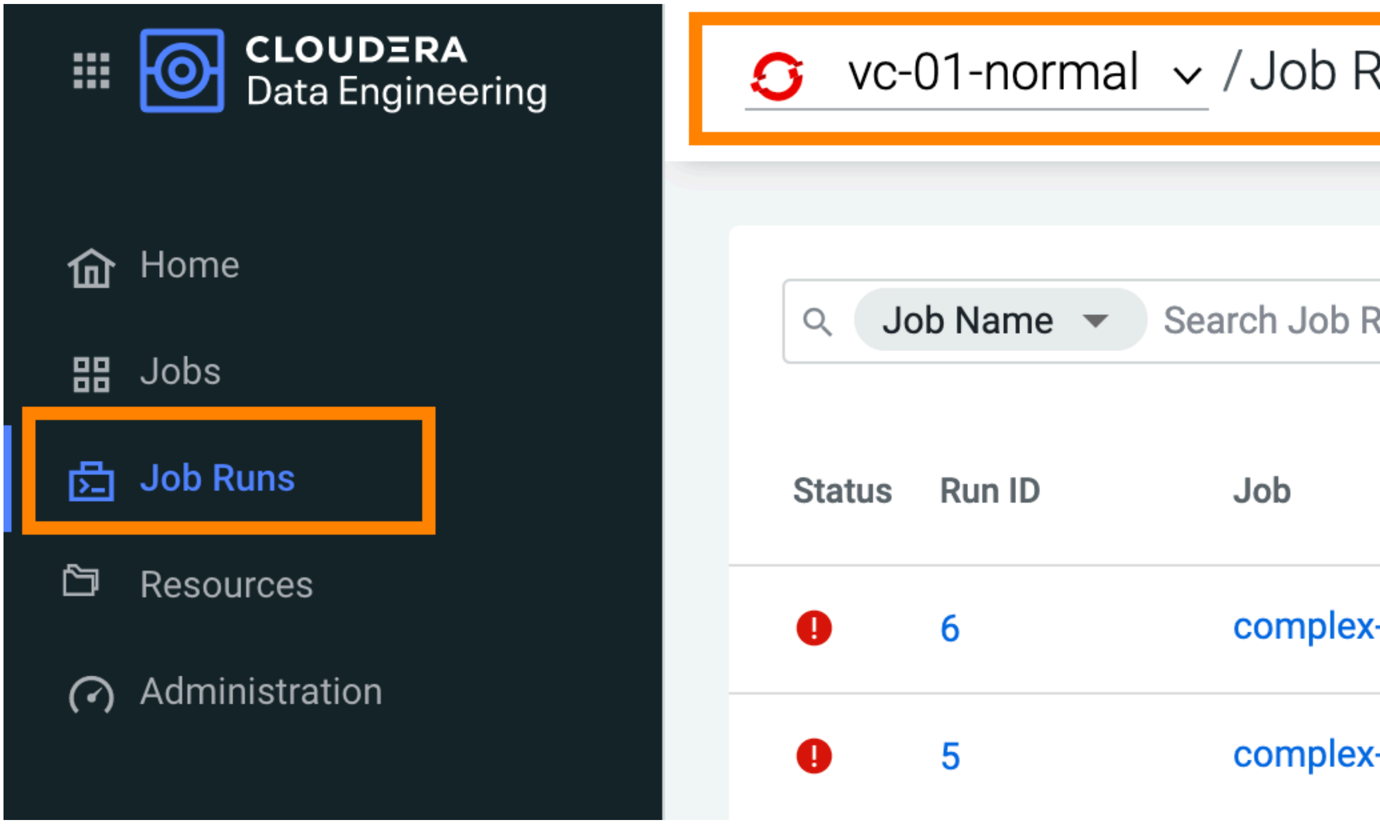
About this task

Note:

- In the OpenShift Container Platform clusters, where the User Access Management feature is not available, the job owner automatically changes to the user who has last updated the job.
- In the Embedded Container Service clusters, where the User Access Management feature is available, the job owner will not change even if a different user updates the job.

Procedure

1. In the Cloudera console, click the Data Engineering tile. The Cloudera Data Engineering Home page displays.
2. Click Job Runs on the left navigation menu.
3. Using the dropdown menu, select the virtual cluster containing the application you want to manage.



4. Click on the job that you want to modify.

5. The Run History tab lists the recent job executions for the application. Click the Configuration tab to display the job configuration.
6. Click Edit to change the application configuration.
7. Edit the configuration parameters you want to change, including uploading a modified JAR or Python file if necessary.
8. Click Update and Run to update and run the job or click the drowpdown and select Update to change configurations.

Running Jobs in Cloudera Data Engineering

Jobs in Cloudera Data Engineering can be run on demand, or scheduled to run on an ongoing basis. The following instructions demonstrate how to run a job in Cloudera Data Engineering.

Before you begin

Make sure that you have configured the Kerberos Keytab file using Hadoop Authentication. For more information about how to configure a Keytab file, see Configuring Hadoop Authentication.

Procedure

1. In the Cloudera console, click the Data Engineering tile. The Cloudera Data Engineering Home page displays.
2. Click Jobs on the left navigation menu. The Jobs page displays.
3. To run a job immediately, click ⋮ in the Actions column next to the job, and then click Run Now.

   You can cancel a running job by clicking Cancel in the same Actions menu.

   Job Run Notices:The running jobs provide notifications, in the form of a Bell icon next to the job Run ID, when certain conditions are met, without having to parse low level logs, or navigating away to a cloud provider or Kubernetes interface. This will help you to identify why certain job is running slow or stuck, and take actions to rectify this.

Scheduling jobs in Cloudera Data Engineering

Jobs in Cloudera Data Engineering can be run on demand, or scheduled to run on an ongoing basis. The following instructions demonstrate how to create or modify a schedule for an existing job.

Before you begin

Make sure that you have configured the Kerberos Keytab file using Hadoop Authentication. For more information about how to configure a Keytab file, see Configuring Hadoop Authentication.

Important: Jobs scheduled in Cloudera Data Engineering are always run as the owner of the job who created it.

Procedure

1. In the Cloudera console, click the Data Engineering tile. The Cloudera Data Engineering Home page displays.
2. Click Jobs on the left navigation menu. The Jobs page displays.
3. Click ⋮ in the Actions column next to the job, and then click Add Schedule.
4. In the Schedule tab, click Create a Schedule.

5. Set the Start time, End time, and Cron expression.

   The start and end times designate the time frame for which the schedule is active. The Cron expression uses the cron scheduling syntax to specify when the application should run within the start and end times. For information and examples of the cron syntax, see the Cron entry on Wikipedia.

   Note: Timestamps must be specified in ISO-8601 UTC format ('yyyy-MM-ddTHH:mm:ssZ'). UTC offsets are not supported.

   Note: Scheduled job runs start at the end of the first full schedule interval after the start date, at the end of the scheduled period. For example, if you schedule a job with a daily interval with a start_date of 14:00, the first scheduled run is triggered at the end of the next day, after 23:59:59. However if the start_date is set to 00:00, it is triggered at the end of the same day, after 23:59:59.
6. Select optional scheduling configurations:
   a) Select Enable Catchup to kick off job runs for any data interval that has not been run since the last data interval. If this option is not selected, only the runs that start after the time that the job was created will be included.
   b) Select Depends on Previous to ensure that each job run is preceeded by a successful job run.
7. Click Add Schedule.

Deleting Jobs in Cloudera Data Engineering

If you no longer need a job, you can delete it. Deleting a job does not delete the job run history.

Before you begin

Procedure

1. In the Cloudera console , click the Data Engineering tile. The Cloudera Data Engineering Home page displays.
2. Click Jobs on the left navigation menu. The Jobs page displays.
3. Click ⋮ in the Actions column next to the job, and then click Delete.

Best practices for building Apache Spark applications

Follow these best practices when building Apache Spark Scala and Java applications:

- Refrain from using withColumn in chain, loop, and calling it multiple times in a single query. Doing so may cause performance issues. To avoid issues, use select() with multiple columns at once. See the Apache Spark API reference linked below for more information.
- Compile your applications against the same version of Spark that you are running.
- Build a single assembly JAR ("Uber" JAR) that includes all dependencies. In Maven, add the Maven assembly plug-in to build a JAR containing all dependencies:

```
<plugin>
  <artifactId>maven-assembly-plugin</artifactId>
  <configuration>
    <descriptorRefs>
      <descriptorRef>jar-with-dependencies</descriptorRef>
    </descriptorRefs>
  </configuration>
  <executions>
    <execution>
      <id>make-assembly</id>
```

```
      <phase>package</phase>
      <goals>
        <goal>single</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

This plug-in manages the merge procedure for all available JAR files during the build. Exclude Spark, Hadoop, and Kafka classes from the assembly JAR, because they are already available on the cluster and contained in the runtime classpath. In Maven, specify Spark, Hadoop, and Kafka dependencies with scope provided. For example:

```
<dependency>
  <groupId>org.apache.spark</groupId>
  <artifactId>spark-core_2.12</artifactId>
  <version>3.4.1.7.3.1.0-SNAPSHOT</version>
  <scope>provided</scope>
</dependency>
```