

Managing Cloudera Data Engineering jobs

Date published: 2020-07-30

Date modified: 2024-05-30

CLOUDERA

Legal Notice

© Cloudera Inc. 2024. 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

Creating jobs in Cloudera Data Engineering.....	4
CDE example jobs and sample data.....	7
Using Apache Iceberg in Cloudera Data Engineering.....	9
Prerequisites and limitations for using Iceberg.....	9
Accessing Iceberg tables.....	10
Editing a storage handler policy to access Iceberg files on the file system.....	11
Creating a SQL policy to query an Iceberg table.....	14
Creating Virtual Cluster with Spark 3.....	15
Creating and running Spark 3.2.1 Iceberg jobs.....	15
Creating a new Iceberg table from Spark 3.....	16
Configuring Hive Metastore for Iceberg column changes.....	17
Importing and migrating Iceberg table in Spark 3.....	17
Importing and migrating Iceberg table format v2.....	18
Configuring Catalog.....	20
Loading data into an unpartitioned table.....	20
Querying data in an Iceberg table.....	20
Updating Iceberg table data.....	21
Iceberg library dependencies for Spark applications.....	21
Creating a Git repository in Cloudera Data Engineering (Technical Preview).....	22
Managing jobs in Cloudera Data Engineering.....	25
Running Jobs in Cloudera Data Engineering.....	26
Scheduling jobs in Cloudera Data Engineering.....	26
Deleting Jobs in Cloudera Data Engineering.....	27
Configuring Spark jobs for huge shuffle data.....	27
Best practices for building Apache Spark applications.....	28

Creating jobs in Cloudera Data Engineering

A job in Cloudera Data Engineering (CDE) 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 (CDE), 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 Data Platform (CDP) console, click the Data Engineering tile. The CDE 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.

Job Details

Job Type *

☒ Spark 3.2.3 ☐ Airflow

Name *

Job Name

Select Application Files

Resource ▼

[Upload](#) or [Select from Resource](#)

Main Class

com.package.MainClass

Arguments (Optional)

Argument



Configurations (Optional)

config_key

config_value



Data Connector (Optional)

Data Connector ▼

Advanced Options

Upload additional files, customize no. of executors, driver and executor cores and memory

Schedule ☐

Turn on to schedule Job, enable catchup and jobs dependants

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 click the Add Configuration button to add multiple configuration parameters 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 CDE.

- 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.
 - 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 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](#).

8. 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](#)

CDE 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 (CDE), 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

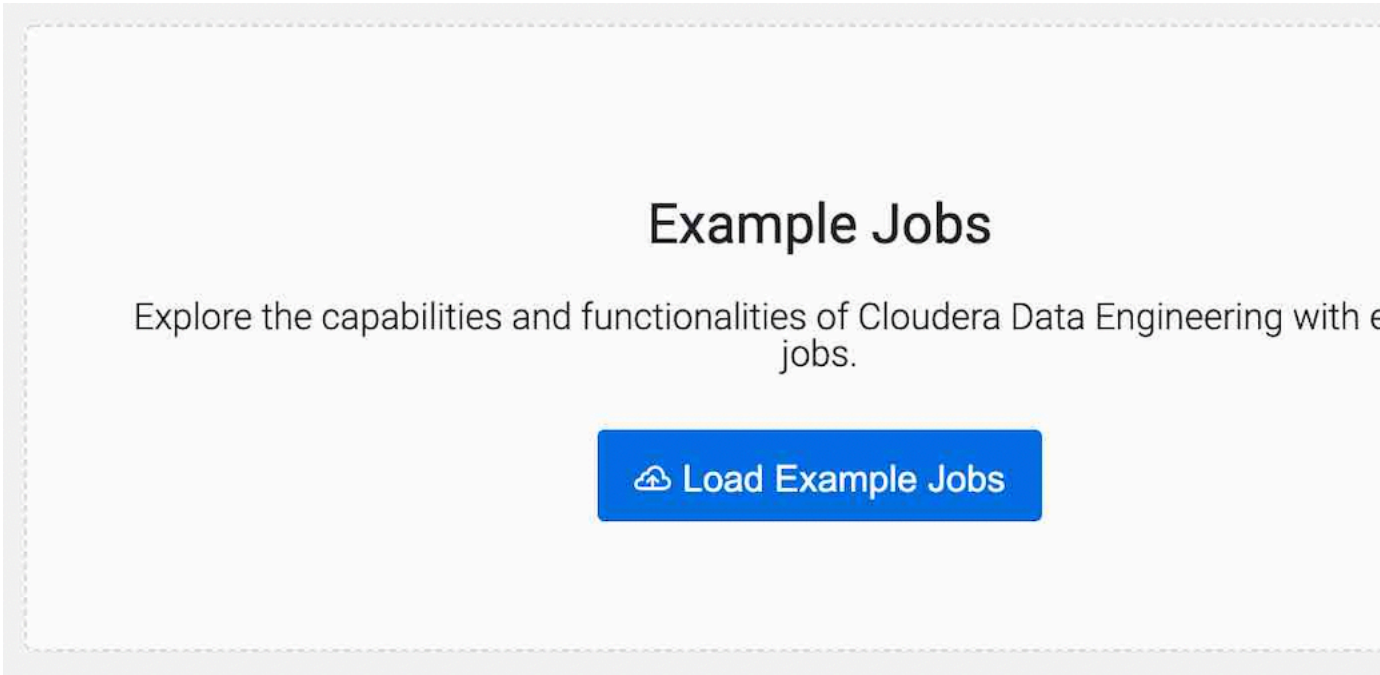
You must run the example jobs with a user who is not the Local Administrator, that is, the user must to have been granted DEUser or DEAdmin privileges in the environment associated with your DE workspace. Also ensure you have enough resources to run these example jobs. Below is the description of the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

Table 1: Example Jobs


Job	Description
example-load-data	<p>Loads the sample data onto the environment data lake. This job runs only once and is then deleted.</p> <p> Note: This will need to be run manually first if the sample jobs are loaded in any user defined virtual clusters.</p> <p>If the example-load-data job fails, contact Cloudera Support to recreate the example-load-data job.</p>
example-virtual-env	<p>Demonstrates CDE job configuration that utilizes Python Environment resource type to expand pyspark features via custom virtual env. This example adds pandas support.</p> <p> Note: You cannot run this job in an air-gapped environment.</p>
example-resources	<p>Demonstrates CDE 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.</p>
example-resources-schedules	<p>Demonstrates scheduling functionality for Spark job in CDE. This example schedules a job to run at 5:04am UTC each day.</p>
example-spark-pi	<p>Demonstrates how to define a CDE job. It runs a SparkPi using a scala example jar located on a s3 bucket. The driver stderr log contains the value of pi.</p> <p> Note: You cannot run this job in an air-gapped environment.</p>
example-cdeoperator	<p>Demonstrates job orchestration using Airflow. This example uses a custom CDE Operator to run two Spark jobs in sequence, mimicking a pipeline composed of data ingestion and data processing.</p> <p> Note: You cannot run this job in an air-gapped environment.</p>
example-object-store	<p>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.</p>

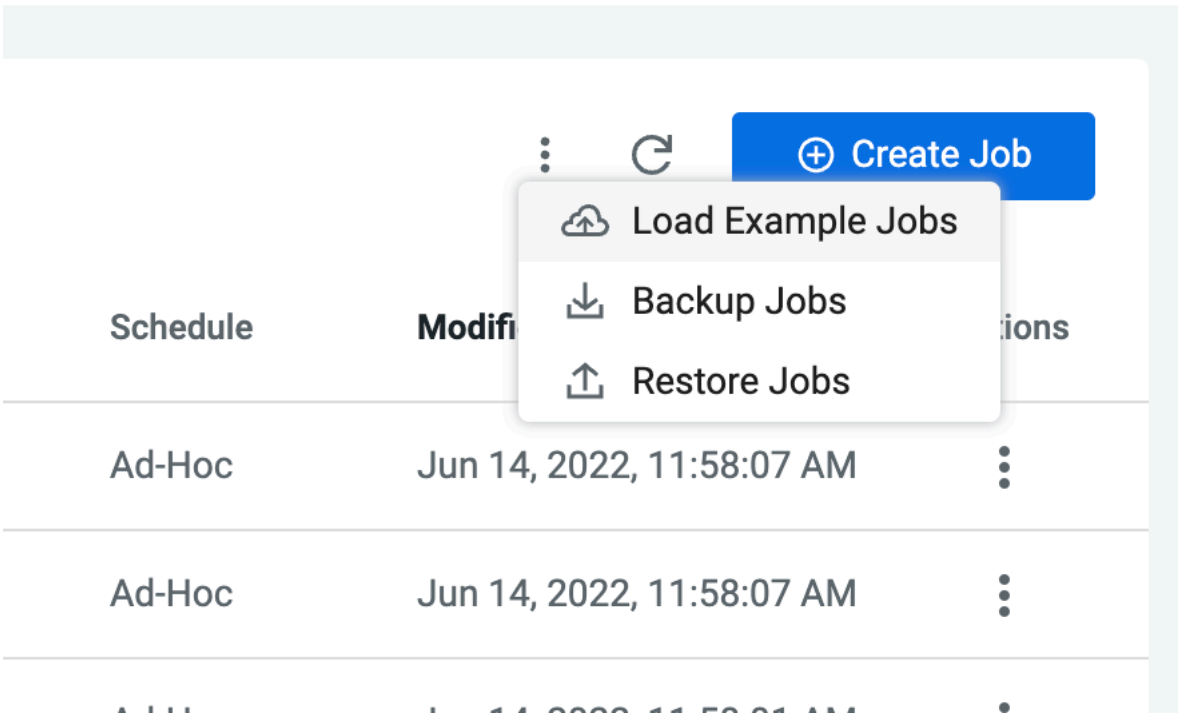
Procedure

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

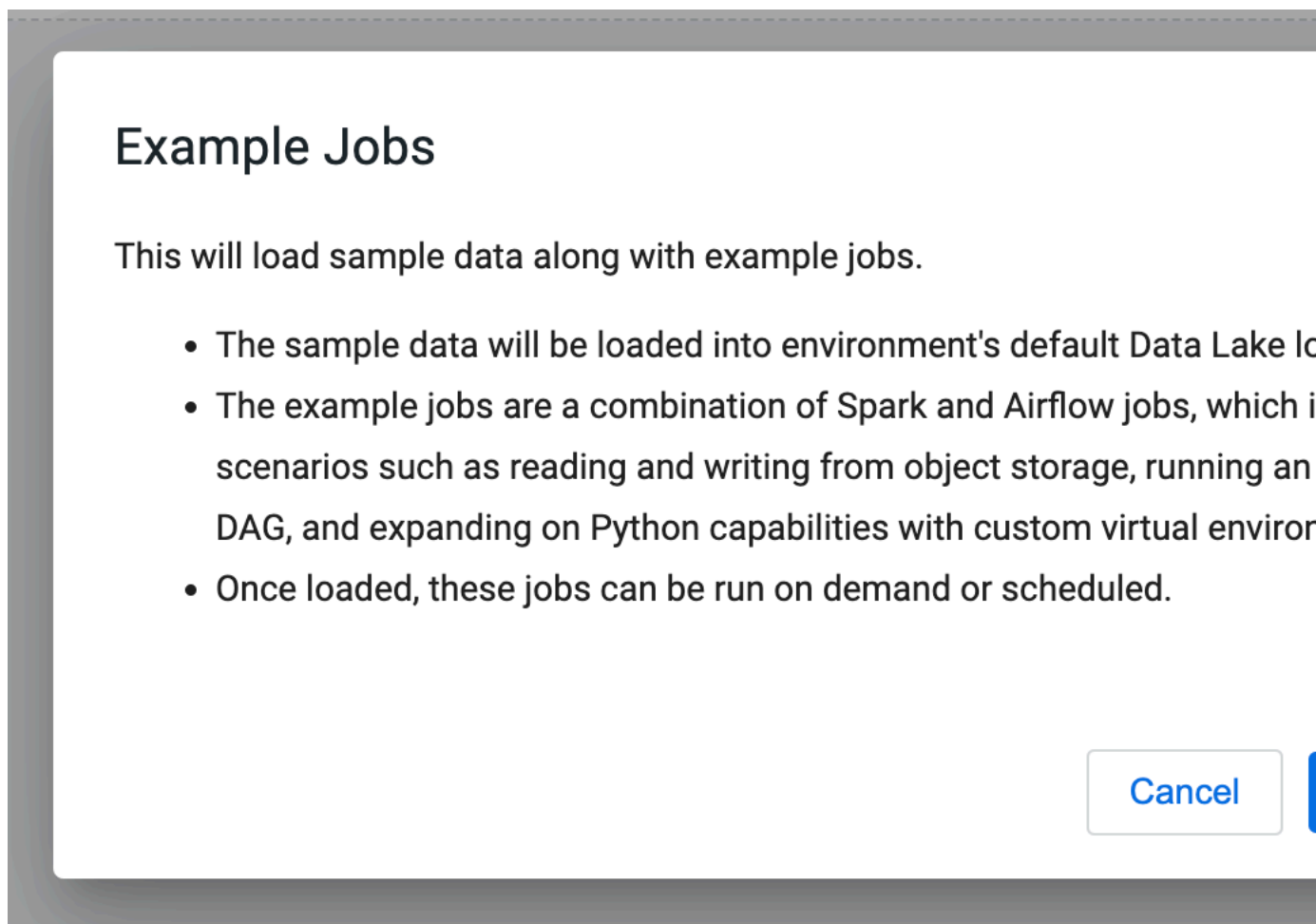


Note: You will see this window only if you have no existing jobs in the virtual cluster.

- 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 (CDE) 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 CDE, you'll need the following prerequisites:

- CDE Virtual Cluster with Spark 3.2.1 or higher
- CDP Private Cloud Base 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.

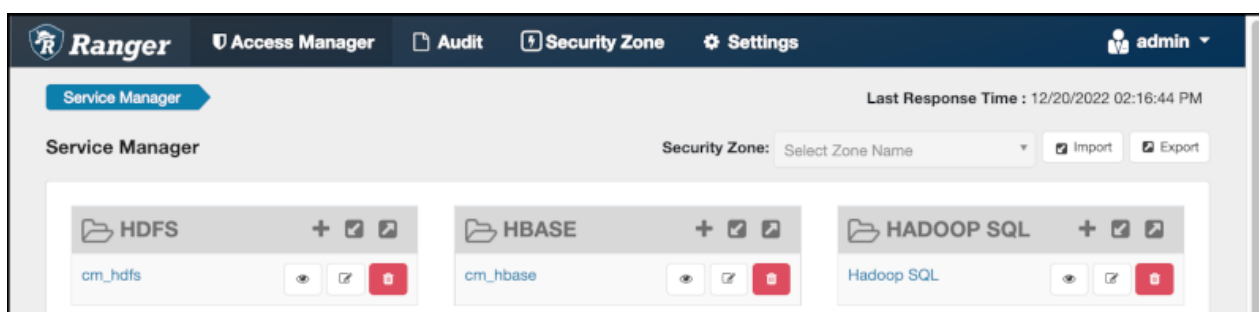
Spark still handles the timestamp column as a timestamp with local timezone. Inconsistent results occur unless Spark is running in UTC.

Accessing Iceberg tables

CDP 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 CDP service to another. In 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.

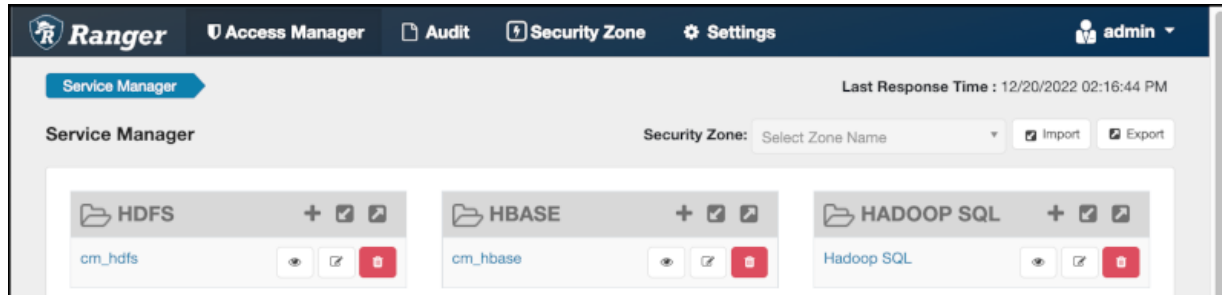
In this task, you specify Iceberg as the storage-type and allow the broadest access by setting the URL to *.

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.

List of Policies : Hadoop SQL

Search for your policy...

Policy ID	Policy Name	Policy Labels	Status
8	all - global	--	Enabled
9	all - database, table, column	--	Enabled
10	all - database, table	--	Enabled
11	all - storage-type, storage-url	--	Enabled

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.

Policy Type **Access**

Policy ID* **11**

Policy Name* **Enabled**

Policy Label

Storage Type *****

Storage URL **Include**

Description

Audit Logging* **Yes**

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:

Allow Conditions:

Select Role	Select Group	Select User
<input type="text" value="Select Roles"/>	<input type="text" value="Select Groups"/>	<input type="text" value="hive"/> <input type="text" value="beacon"/> <input type="text" value="dpprofiler"/>
		<input type="text" value="hue"/> <input type="text" value="admin"/> <input type="text" value="impala"/>
		<input type="text" value="systest"/>

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.

The screenshot shows a web interface for managing permissions. A modal window titled 'add/edit permissions' is open, displaying a list of permissions with checkboxes: select, update, Create, Drop, Alter, Index, Lock, All (checked), Read, Write, ReplAdmin, Service Admin, Temporary UDF Admin, Refresh, RW Storage, and Select/Deselect All. Below the modal, there are sections for 'Allow Conditions' and 'Permissions'. The 'Permissions' section has a button labeled 'Add Permissions'.

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 3.2.1 Iceberg jobs

Create and run a spark job which uses iceberg tables.

Before you begin



Important: You must complete the [manual steps](#) to prepare the cluster for each user who need to submit jobs.

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 21.


In Cloudera Data Engineering (CDE), 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 Data Platform (CDP) console, click the Data Engineering tile. The CDE Home page displays.
2. In the CDE 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 timestamp)
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](#) linked below.

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 PVC CDE 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-mode>',  
'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_catalog`, 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, CDS 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 `spark3-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 Files](#)

[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 CDE.

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 CDE jobs. For more information see, [Creating Spark jobs](#) and [Running a Spark job](#).

Creating a Git repository in Cloudera Data Engineering (Technical Preview)

Git 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 (CDE) 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 CDE's centralized monitoring and troubleshooting capabilities to tune and adjust your workloads. CDE 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 CDE UI and users cannot delete or modify the files. This ensures a single source of truth and simplifies promotions.

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 CDE 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.

Procedure

1. In the Cloudera Data Platform (CDP) 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:

Create A Repository



Choose your setup credential and provide the corresponding GIT URL to begin creating your repository.

Note: Credentials can only currently be created via the CDE CLI Tool. More information on that can be [found here](#)

Repository Name *

URL *

Branch *

Select Credential (Optional)

☒ Skip TLS

Create

Cancel

- 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) Select a credential from the Select Credential drop-down list. The credentials can be created using the CDE API.
 - e) Select Skip TLS. Select this option if the server uses a self-signed CA certificate that CDE does not trust. This allows CDE to skip the security check and clone the repository.
4. Click Create.

Managing jobs in Cloudera Data Engineering

It is often necessary to modify your Cloudera Data Engineering (CDE) jobs. CDE 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.

Before you begin

Procedure

1. In the Cloudera Data Platform (CDP) console, click the Data Engineering tile. The CDE 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.

The screenshot shows the Cloudera Data Engineering (CDE) console interface. On the left, a dark sidebar contains the navigation menu with options: Home, Jobs, Job Runs (highlighted with an orange box), Resources, and Administration. The main content area on the right has a header with a dropdown menu set to 'vc-01-normal' and a search bar labeled 'Job Name'. Below this is a table with columns 'Status', 'Run ID', and 'Job'. The table lists two job runs, both with a red status icon and the label 'complex-'. The first row has a Run ID of 6, and the second row has a Run ID of 5.

Status	Run ID	Job
!	6	complex-
!	5	complex-

4. Click 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 dropdown and select Update to change configurations.

Running Jobs in Cloudera Data Engineering

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

Before you begin

Procedure

1. In the Cloudera Data Platform (CDP) console, click the Data Engineering tile. The CDE 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 (CDE) 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

Procedure

1. In the Cloudera Data Platform (CDP) console, click the Data Engineering tile. The CDE 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 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 Data Platform (CDP) console, click the Data Engineering tile. The CDE 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.

Configuring Spark jobs for huge shuffle data

You can configure the Spark jobs to use the persistent volumes for shuffle data to improve the performance or handle the huge shuffle data.

About this task

By default, Cloudera Data Engineering (CDE) service does not use any PersistentVolume for shuffle data causing the shuffle data to spill over to the local disks. If the local disk space is not sufficient or local disk performance decreases, you must configure the Spark jobs to use a persistent volume for shuffle data at the job level.



Note: You can choose any storage class for the local volume depending on your requirement and the storage classes available in your Kubernetes cluster.

Before you begin

Make sure that you have a compatible storage class that supports local volumes.

Procedure

When creating a new job or editing an existing job, add the required configurations in the Job details Configurations field.

For example, If you have the local volume storage class local-path and need a volume size of maximum 10 Gi, then add the following configurations:

config_key	config_value
spark.kubernetes.executor.volumes.persistentVolumeClaim.spark-local-dir-1.mount.path	/data
spark.kubernetes.executor.volumes.persistentVolumeClaim.spark-local-dir-1.mount.readOnly	false
spark.kubernetes.executor.volumes.persistentVolumeClaim.spark-local-dir-1.options.claimName	OnDemand

config_key	config_value
spark.kubernetes.executor.volumes.persistentVolumeClaim.spark-local-dir-1.options.sizeLimit	10Gi
spark.kubernetes.executor.volumes.persistentVolumeClaim.spark-local-dir-1.options.storageClass	local-path

Related Information

[Creating jobs in Cloudera Data Engineering](#)

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_2.11</artifactId>
  <version>2.4.0.7.0.0.0</version>
  <scope>provided</scope>
</dependency>
```