

Managing Cloudera Data Engineering jobs

Date published: 2020-07-30

Date modified: 2023-06-13



Legal Notice

© Cloudera Inc. 2025. 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

Configure users to create jobs.....	4
Configuring LDAP users.....	4
Configuring service account key tab to the machine user.....	6
Creating jobs in Cloudera Data Engineering.....	7
CDE example jobs and sample data.....	9
Using Apache Iceberg in Cloudera Data Engineering (Technical Preview).....	12
Prerequisites and limitations for using Iceberg.....	12
Accessing Iceberg tables.....	13
Editing a policy to access Iceberg files.....	14
Creating a policy to query an Iceberg table.....	16
Creating Virtual Cluster with Spark 3.....	18
Creating and running Spark jobs using Iceberg tables.....	18
Creating a new Iceberg table from Spark 3.....	19
Configuring Hive Metastore for Iceberg column changes.....	20
Importing and migrating Iceberg table in Spark 3.....	20
Importing and migrating Iceberg table format v2.....	21
Configuring Catalog.....	22
Loading data into an unpartitioned table.....	23
Querying data in an Iceberg table.....	23
Updating Iceberg table data.....	23
Iceberg library dependencies for Spark applications.....	24
Managing jobs in Cloudera Data Engineering.....	25
Running Jobs in Cloudera Data Engineering.....	26
Scheduling jobs in Cloudera Data Engineering.....	26
Creating an ad-hoc job in Cloudera Data Engineering.....	27
Deleting Jobs in Cloudera Data Engineering.....	27
Best practices for building Apache Spark applications.....	28

Configure users to create jobs

You must complete the manual steps to prepare the cluster for each user that needs to submit jobs. You can also configure a service account Kerberos key tab file to a machine user id in order to submit CDE jobs. The machine user will now have access to the hdfs storage through jobs.

**Important:**

- After you create a cluster, you must initialize the cluster, and configure users before creating jobs.
- 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 LDAP users

You must complete the following manual steps to prepare the cluster for each user that needs to submit jobs. Perform these steps for each user that needs to submit jobs to the virtual cluster.

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](#).

Procedure

1. If you already downloaded the utility script and uploaded it to an ECS or HDFS gateway cluster host as documented in [Creating virtual clusters](#), you can skip to step 8.
2. Download [cde-utils.sh](#) to your local machine.
3. Create a directory to store the files, and change to that directory:

```
mkdir -p /tmp/cde-utils && cd /tmp/cde-utils
```

4. Embedded Container Service (ECS)

Copy the extracted utility script (cde-utils.sh) to one of the Embedded Container Service (ECS) cluster hosts. To identify the ECS cluster hosts:

- a. Log in to the Cloudera Manager web interface.
- b. Click Clusters tab.
- c. Click the relevant ECS cluster from the list of the clusters displayed.
- d. Under Status, click Hosts link.
- e. Copy the script to the master host manually.

Red Hat OpenShift Container Platform (OCP)

Copy the extracted utility script (cde-utils.sh) and the OpenShift kubeconfig file to one of the HDFS service gateway hosts, and install the kubectl utility:

- a. Log in to the Cloudera Manager web interface.
- b. Go to Clusters Base Cluster HDFS Instances .
- c. Copy the script to one of the Gateway hosts manually.
- d. Copy the [OCP kubeconfig](#) file to the same host.
- e. On that host, install the kubectl utility following the [instructions](#) in the Kubernetes documentation.

5. On the cluster host that you copied the script to, set the script permissions to be executable:

```
chmod +x /PATH/TO/cde-utils.sh
```

6. Identify the virtual cluster endpoint:

- a. In the Cloudera Manager web UI, go to the Data Services page, and then click Open CDP Private Cloud Data Services.
- b. Click the Data Engineering tile.
- c. Select the CDE service containing the virtual cluster you want to activate.
- d.  Click Cluster Details.
- e. Click JOBS API URL to copy the URL to your clipboard.
- f. Paste the URL into a text editor to identify the endpoint host. For example, the URL is similar to the following:

```
https://dfdj6kgx.cde-2cdxw5x5.apps.ecs-demo.example.com/dex/api/v1
```

The endpoint host is dfdj6kgx.cde-2cdxw5x5.apps.ecs-demo.example.com.

7. On the ECS or HDFS gateway host, create a filename containing the user principal, and generate a keytab. If you do not have the ktutil utility, you might need to install the krb5-workstation package. The following example commands assume the user principal is psherman@EXAMPLE.COM

- a. Create a file named <USERNAME>.principal (for example, psherman.principal) containing the user principal:

```
psherman@EXAMPLE.COM
```

- b. Generate a keytab named <USERNAME>.keytab for the user using ktutil:

```
sudo ktutil
ktutil: addent -password -p psherman@EXAMPLE.COM -k 1 -e aes256-cts
Password for psherman@EXAMPLE.COM:
ktutil: addent -password -p psherman@EXAMPLE.COM -k 2 -e aes128-cts
Password for psherman@EXAMPLE.COM:
ktutil: wkt psherman.keytab
ktutil: q
```

8. Validate the keytab using klist and kinit:

```
klist -ekt psherman.keytab
Keytab name: FILE:psherman.keytab
KVNO Timestamp Principal
-----
1 08/01/2021 10:29:47 psherman@EXAMPLE.COM (aes256-cts-hmac-sha1-96)
1 08/01/2021 10:29:47 psherman@EXAMPLE.COM (aes128-cts-hmac-sha1-96)

kinit -kt psherman.keytab psherman@EXAMPLE.COM
```

Make sure that the keytab is valid before continuing. If the kinit command fails, the user will not be able to run jobs in the virtual cluster. After verifying that the kinit command succeeds, you can destroy the Kerberos ticket by running kdestroy.

- Use the `cde-utils.sh` script to copy the user keytab to the virtual cluster hosts:

```
./cde-utils.sh init-user-in-virtual-cluster -h <endpoint_host> -u <USER> -p <PRINCIPAL_FILE> -k <KEYTAB_FILE>
```

For example, using the `psberman` user, for the `dfdj6kgx.cde-2cdxw5x5.apps.ecs-demo.example.com` endpoint host:

```
./cde-utils.sh init-user-in-virtual-cluster -h dfdj6kgx.cde-2cdxw5x5.apps.ecs-demo.example.com -u psberman -p psberman.principal -k psberman.keytab
```

- Repeat these steps for all users that need to submit jobs to the virtual cluster.

Configuring service account key tab to the machine user

You can configure a service account Kerberos key tab file to a machine user id in order to submit CDE jobs. The machine user will now have access to the hdfs storage through jobs.

Procedure

- Create a CDP machine user. For example, `mu_cde`.
- Generate access key and secret key for the machine user and download the credential information.
- On the ECS or HDFS gateway host, create a filename containing the user principal, and generate a keytab. If you do not have the `ktutil` utility, you might need to install the `krb5-workstation` package. The following example commands assume the user principal is `mu_cde@example.com`.
 - Create a file named `<username>.principal` (for example, `MU_CDE.PRINCIPAL`) containing the user principal:

```
mu_cde@example.com
```

- Generate a key tab file for the actual service account. For example, `SVC_ACC`. Name this keytab file as `<username>.keytab`, example: `mu_cde.keytab`. The generation of keytab can be done for the user using the `ktutil` command:

```
sudo ktutil
ktutil: addent -password -p mu_cde@example.com -k 1 -e aes256-cts
Password for pmu_cde@example.com:
ktutil: addent -password -p mu_cde@example.com -k 2 -e aes128-cts
Password for mu_cde@example.com:
ktutil: wkt mu_cde.keytab

ktutil: q
```

- Validate the keytab using `klist` and `kinit` commands:

```
klist -ekt mu_cde.keytab

Keytab name: FILE:mu_cde.keytab
KVNO Timestamp Principal
-----
1 08/01/2021 10:29:47 mu_cde@example.com (aes256-cts-hmac-sha1-96)
1 08/01/2021 10:29:47 mu_cde@example.com (aes128-cts-hmac-sha1-96)
kinit -kt mu_cde.keytab mu_cde@example.com
```

Make sure that the keytab is valid before continuing. If the `kinit` command fails, the user will not be able to run jobs in the virtual cluster. After verifying that the `kinit` command succeeds, you can destroy the Kerberos ticket by running `kdestroy`.

5. Upload the keytab file for the user id as `mu_cde` and use the principal and keytab file of the actual service account (`svc_acc`).

```
./cde-utils.sh init-user-in-virtual-cluster -h <endpoint_host> -u <user> -p <principal_file> -k <keytab_file>
```

For example, using the `mu-cde` user, for the `dfdj6kgx.cde-2cdxw5x5.apps.ecs-demo.example.com` endpoint host:

```
./cde-utils.sh init-user-in-virtual-cluster -h dfdj6kgx.cde-2cdxw5x5.apps.ecs-demo.example.com -u mu-cde -p mu-cde.principal -k mu-cde.keytab
```

6. Set the *Ranger authorization policy* for the `svc_acc` account.

Related Information

[Ranger authorization policy](#)

[Creating a machine user in CDP](#)

[Generate Access Key](#)

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 CDE Home page, in Jobs, click Create New under Spark or click Jobs in the left navigation menu and then click Create Job.

Jobs / Create Job

Job Details

Job Type *

☒ Spark 3.2.3 ☐ Airflow

Name *

jobtest

Application File

☒ File ⓘ ☐ URL ⓘ

Upload or Select from Resource

Main Class

com.package.MainClass

Arguments (Optional)

Argument



Configurations (Optional)

config_key

config_value



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

Cancel

Create and Run



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.

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.
- 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 UI and the job run which uses the data connector must be only created using the UI. Triggering the jobs with the data connector using the 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.
- ### 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.

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:

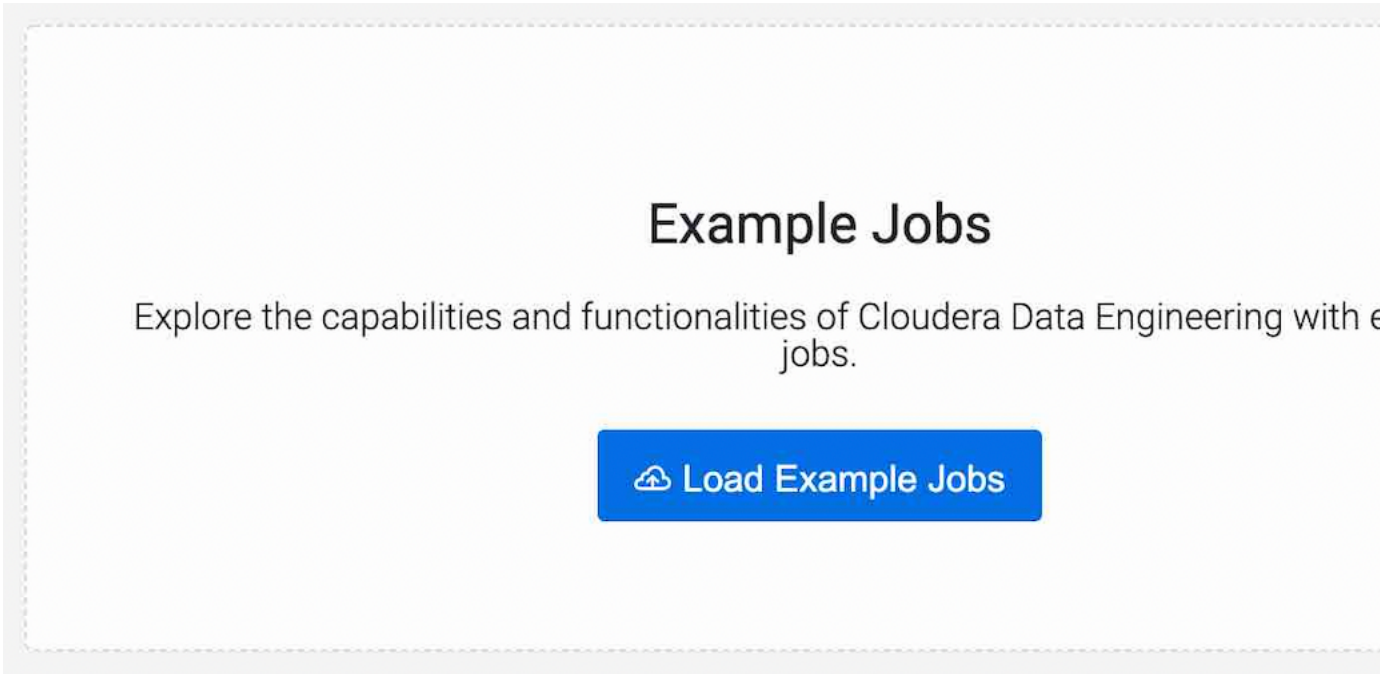
Table 1: Example Jobs

Job	Description
example-load-data	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 CDE 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 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.
example-resources-schedules	Demonstrates scheduling functionality for Spark job in CDE. This example schedules a job to run at 5:04am UTC each day.
example-spark-pi	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.  Note: You cannot run this job in an air-gapped environment.
example-cdeoperator	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.  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

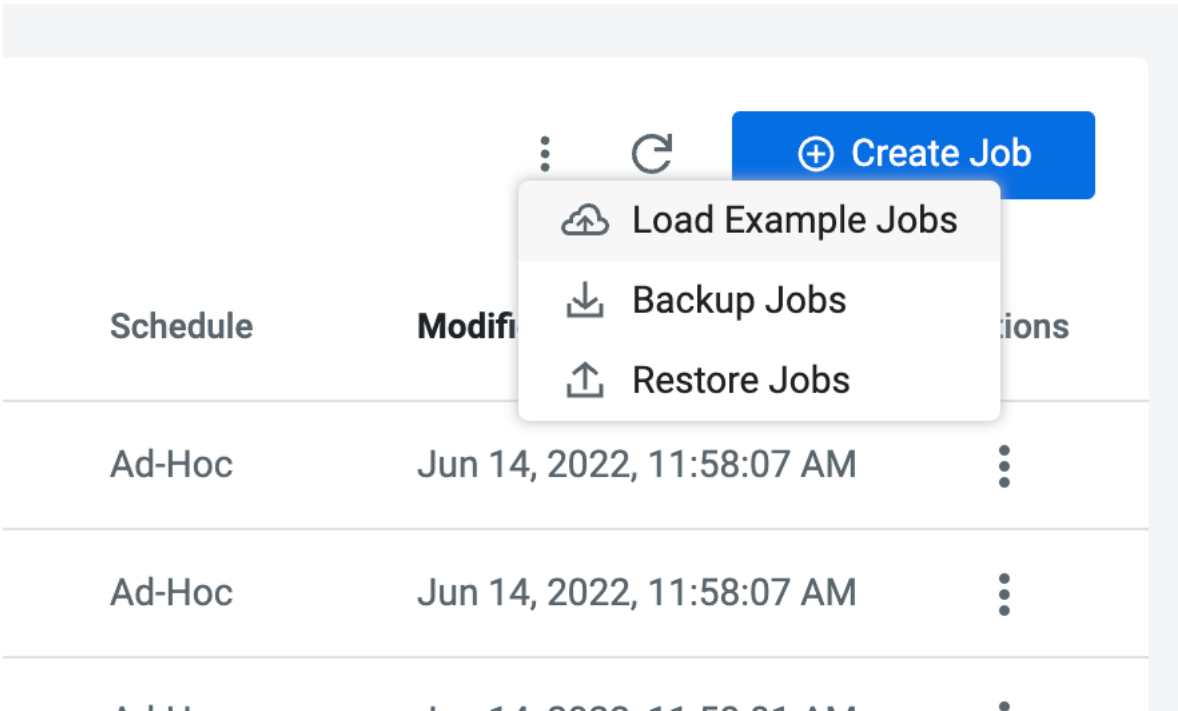
1. In the Cloudera Data Platform (CDP) console, click the Data Engineering tile. The CDE Home page displays.
2. Click Jobs on the left navigation menu.

- 3. Select Load Example Jobs from the two options that appear.

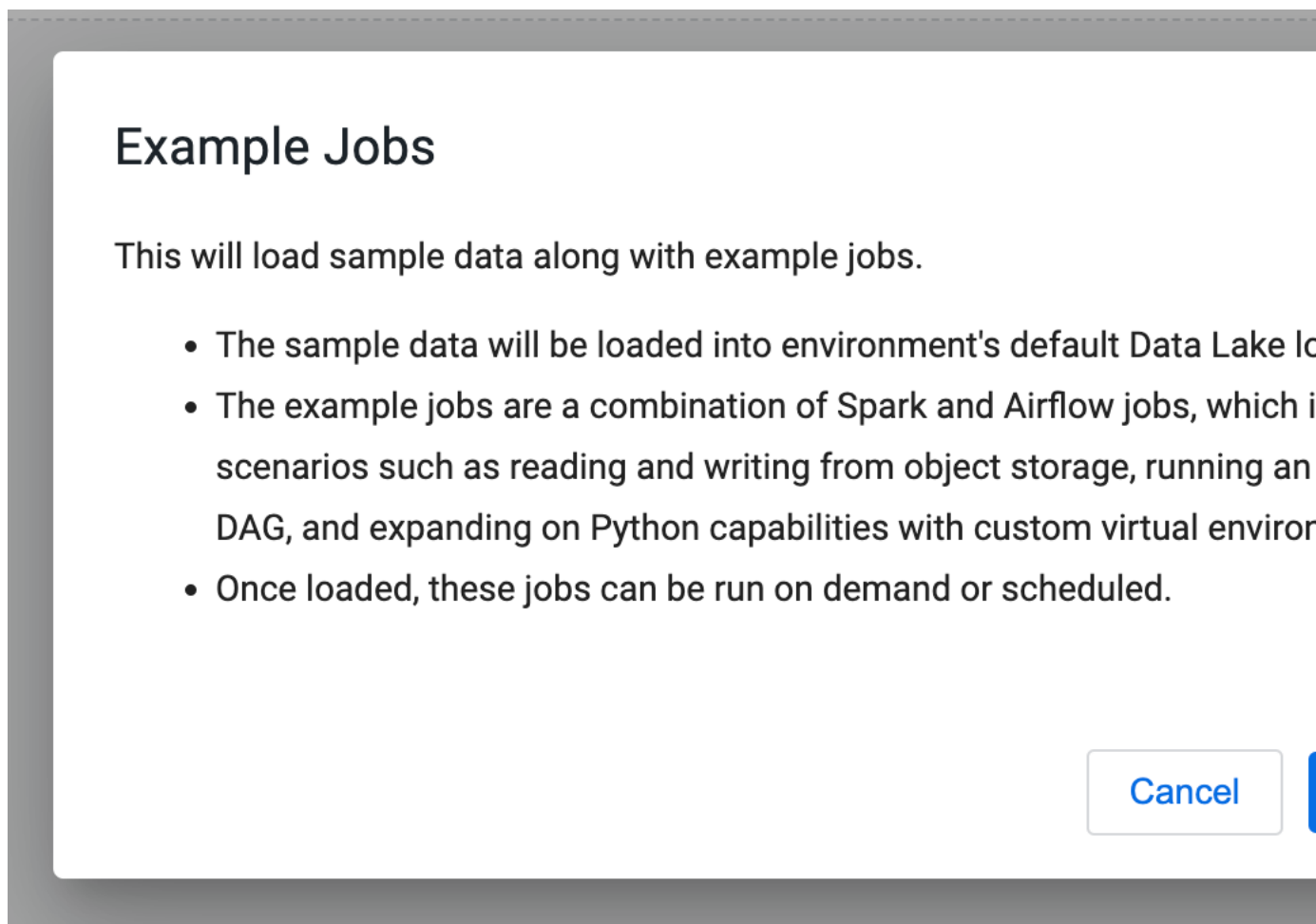


Note: You will see this window only if you have no existing jobs in the virtual cluster.

- 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 (Technical Preview)

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 or 7.1.8

Limitations

- The use of Iceberg tables as Structured Streaming sources or sinks is not supported.
- PyIceberg is not supported. Using Spark SQL to query Iceberg tables in PySpark is supported.

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.

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 .



dw-team-env

crn:cdp:environments:us-west-1:9d74eee4-1cad-45b73d:environment:6f4157a1f19ae

 US West (Oregon) - us-west-2

A shared environment for hive/impala data warehouse

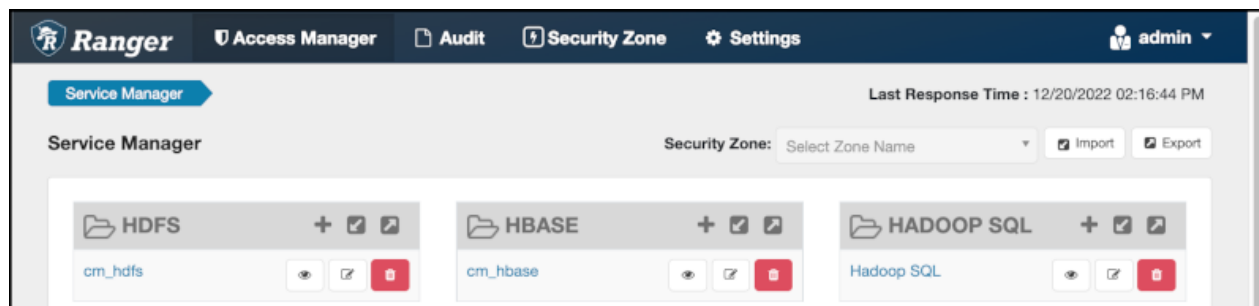
Data Lake upgrade available



Data Lake Details

NAME	NODES	SCALE	QUICK LINKS
dw-team-env-dl	 2  0  0	Light Duty	 At Catal...

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



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 "Edit 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 policy to access Iceberg files

You learn how to edit the existing default Hadoop SQL Storage Handler policy to access files on the object store. 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 *.

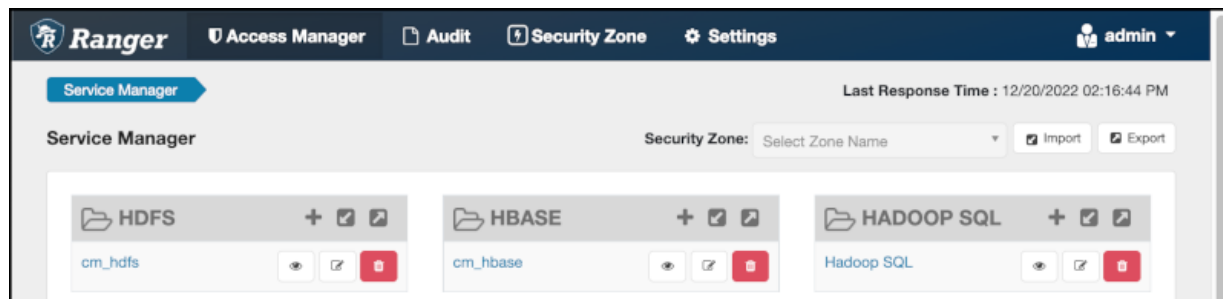
The Hadoop SQL Storage Handler policy supports only the RW Storage permission. A user having the required RW Storage permission on a resource, such as Iceberg, that you specify in the storage-type properties, is allowed only to reference the table location (for create/alter operations) in Iceberg. The 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.

The screenshot shows a configuration form for a policy. The fields are as follows:

- Policy Type:** Access (selected)
- Policy ID:** 11
- Policy Name *:** all - storage-type, storage-uri (with an info icon) **Enabled** (toggle)
- Policy Label:** Policy Label
- storage-type *:** iceberg (selected from a dropdown)
- Storage URL *:** * **Include** (toggle)

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:

The screenshot shows the 'Allow Conditions' section with a table for selecting roles, groups, and users.

Select Role	Select Group	Select User
Select Roles	Select Groups	× hive × beacon × dpprofiler × hue × admin × impala × 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 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.

The screenshot shows the 'Create Policy' form in the Ranger Admin Web UI. The breadcrumb navigation at the top indicates the path: Service Manager > Hadoop SQL Policies > Create Policy. The form is titled 'Create Policy' and contains a section for 'Policy Details'. The 'Policy Type' is set to 'Access'. The 'Policy Name' field contains 'IcebergPolicy1' and has an 'Enabled' toggle switch. The 'Policy Label' field contains 'Policy Label'. Below this, there are three rows for permissions, each with a dropdown menu, a text input field, and an 'Include' toggle switch. The first row is for 'database' with the value 'icedb'. The second row is for 'table' with the value 'icetable'. The third row is for 'column' with the value '*'. All three 'Include' toggle switches are turned on.

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 jobs using Iceberg tables

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 24.

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

In Cloudera Data Engineering (CDE), you can create a Spark job that creates a new Iceberg table or import an existing Hive table. Once created, the table can be used for subsequent operations.



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 CDE 1.19 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', '<de  
lete-mode>', 'write.update.mode', '<update-mode>', 'write.merge.mode', '<mer  
ge-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. In CDE, 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)
```

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 CDE 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.



Note: For CDH-7.1.x, there are no iceberg jars in the maven repository. Use 0.14.1.17.7215.0-27 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.

```
<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>
```

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](#).

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 console. On the left, the navigation menu includes 'Home', 'Jobs', 'Job Runs' (highlighted with an orange box), 'Resources', and 'Administration'. The main area displays a dropdown menu for 'vc-01-normal' and a table of job runs.

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.

Creating an ad-hoc job in Cloudera Data Engineering

Ad-hoc runs mimic the behavior of the traditional spark-submit or a single execution of an Airflow DAG, where the job runs once. These runs will not establish a permanent job definition. You can use the ad-hoc job runs for log analysis and future reference.

Before you begin

- Ensure that you have a Virtual Cluster that is ready to use.

For Spark jobs

1. In the Cloudera Data Platform (CDP) console, click the Data Engineering tile. The Home page displays.
2. In the Jobs section under Spark, click Ad-hoc Run.
3. Select a Virtual Cluster.
4. Enter a Job Name.
5. Upload an Application File or enter the Application File's External URL.
6. Enter a Main Class.
7. Enter Arguments and Configurations.
8. Select a Python Environment

Steps for advanced options

You can upload additional files, customize the number of executors, drivers, executor cores, and memory.

1. Upload files and resources.
2. Configure Compute Options.
3. Set an option for Log Level.
4. Click Create and Run.

For Airflow jobs

1. In the Cloudera Data Platform (CDP) console, click the Data Engineering tile. The Home page displays.
2. In the Jobs section under Airflow, and click Ad-hoc Run.
3. Select a Virtual Cluster.
4. Enter a Job Name.
5. Upload a DAG file.
6. Click Create and Run.

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.

Best practices for building Apache Spark applications

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

- 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>
```