Cloudera AI

Accessing Data from Cloudera AI

Date published: 2020-07-16
Date modified: 2025-10-31

https://docs.cloudera.com/

Legal Notice

Contents

Uploading and working with local files

To work with data files (.csv, .txt, and so on) existing on your computer, upload the files directly to your project in the Cloudera AI Workbench. The presented code samples demonstrate how to access local data for Cloudera AI workloads.

If you want to work with existing data files (.csv, .txt, etc.) from your computer, you can upload these files directly to your project in the Cloudera AI Workbench. Go to the project's Overview page. Under the Files section, click Upload and select the relevant data files to be uploaded. These files will be uploaded to an NFS share available to each project.

Note: Storing large data files in your Project folder is highly discouraged. You can store your data files in the Data Lake.

The following sections use the tips.csv dataset to demonstrate how to work with local data stored in your project. Before you run these examples, create a folder called data in your project and upload the dataset file to it.

Python

```
import pandas as pd
tips = pd.read_csv('data/tips.csv')

tips \
  .query('sex == "Female"') \
  .groupby('day') \
  .agg({'tip' : 'mean'}) \
  .rename(columns={'tip': 'avg_tip_dinner'}) \
  .sort_values('avg_tip_dinner', ascending=False)
```

R

```
library(readr)
library(dplyr)

# load data from .csv file in project
tips <- read_csv("data/tips.csv")

# query using dplyr
tips %>%
  filter(sex == "Female") %>%
  group_by(day) %>%
  summarise(
    avg_tip = mean(tip, na.rm = TRUE)
  ) %>%
  arrange(desc(avg_tip))
```

Auto discovering data sources

Data connections listed in this section are automatically discovered and configured.

The following data connections are automatically discovered:

- Hive on Base
- Impala on Base

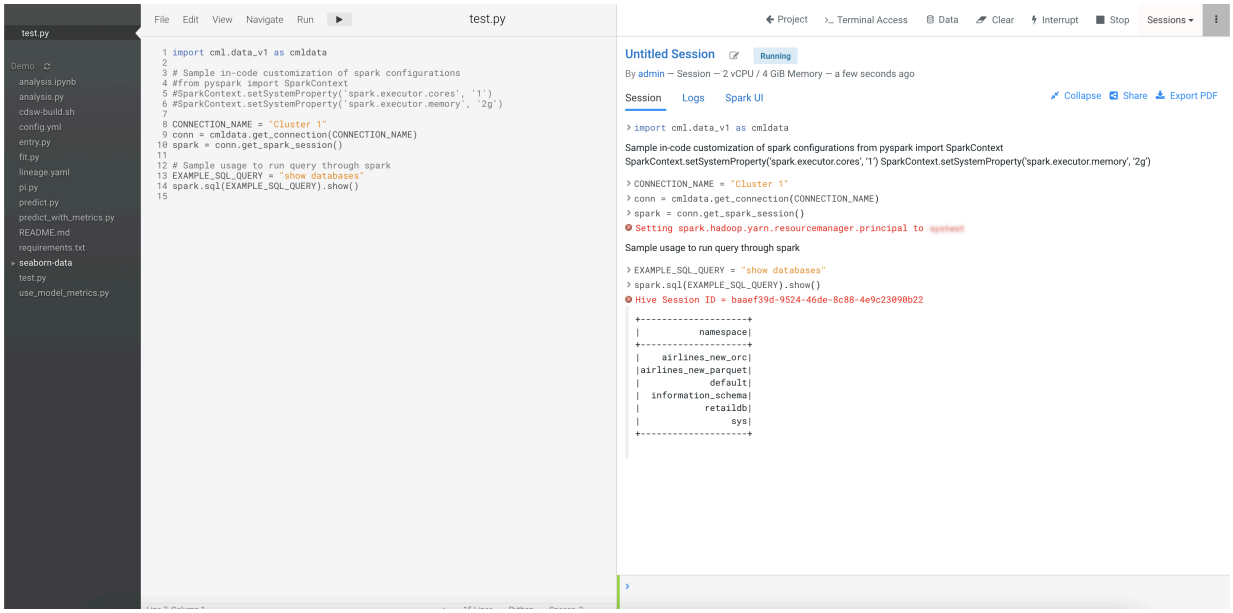
- Hive on Cloudera Data Warehouse
- Impala on Cloudera Data Warehouse
- Spark running in Cloudera AI
- Spark running in Base (when Spark pushdown is enabled)

Using data connection snippets

As a data scientist, you can connect your project to data with a data connection snippet.

Procedure

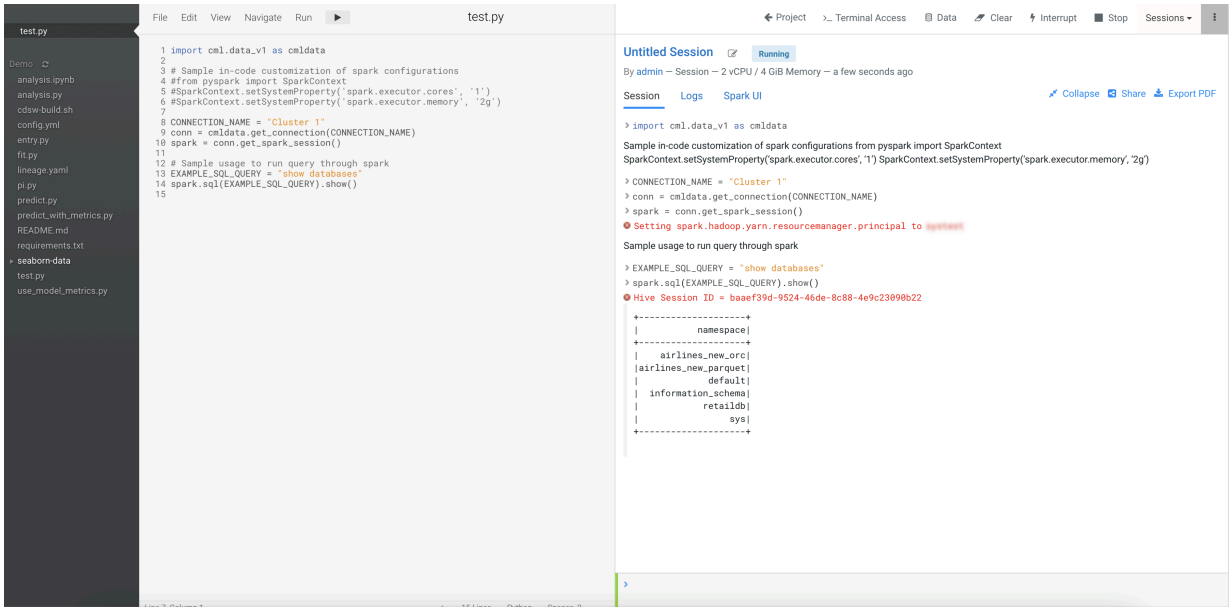
1. Select New Project.
2. Enter the project name.
3. Start a new session. To use a Spark connection, make sure to select an ML Runtimes engine.
4. In the Connection Code Snippet pane, select the data connection to use.



5. Select Copy Code.

6. Paste the snippet into your code.

Figure 1: Pasting snippet into your code



7. Enter the data connection name.
8. Uncomment and edit the SQL query.
9. Select Run.

Results

The results of the SQL query display in the command window.

What to do next

When you have finished exploring the available data sources, you can select **Don't show me this again** for this project on the Connection Code Snippet pane, and it will no longer display when you start a session in the project.

Manual data connections to connect to data sources

You can also set up data connections manually, which work across Cloudera environments. Follow the procedures to set up data connections.

Connecting to Cloudera Data Warehouse

The provided examples use Kerberos for authentication when connecting to Cloudera Data Warehouse Hive, and Impala, which requires that the Keytab is set and there are proper permissions to access Cloudera Data Warehouse.

In order to get the Cloudera Data Warehouse Hive and Impala JDBC Kerberos URLs:

1. Go to Data Warehouse Virtual Warehouses .
2. Select your Virtual Warehouse.
3. Copy the JDBC URL.

Connecting to Cloudera Data Warehouse Impala

Python

```
from impala.dbapi import connect
import os

#jdbc:impala://coordinator-cdw-impala.apps.shared-os-qe-01.kcloud.cloudera.c
om:443/default;AuthMech=1;transportMode=http;httpPath=cliservice;ssl=1;KrbHo
stFQDN=dwx-env-rhcxab-env.cdp.local.;KrbServiceName=hive

conn = connect(
            host="coordinator-cdw-impala.apps.shared-os-qe-01.kcloud.cloud
era.com", #this gets extracted from the jdbc url
            port=443, #extracted from jdbc url
            auth_mechanism="GSSAPI", #always GSSAPI for Kerberos
            use_http_transport=True, #if transportMode=http in jdbc this is
 true, otherwise false
            http_path="cliservice",#this will always be cliservice
            use_ssl=True, # if ssl=1 in jdbc set this to true, otherwise f
alse
            kerberos_service_name = "hive", #this will be KrbServiceName in
 the jdbc url
            krb_host="dwx-env-rhcxab-env.cdp.local.", #this will be the Kr
bHostFQDN in jdbc url
        )
# Execute using SQL
cursor = conn.cursor()
cursor.execute('show databases')
```

Connecting to Cloudera Data Warehouse Hive

Python

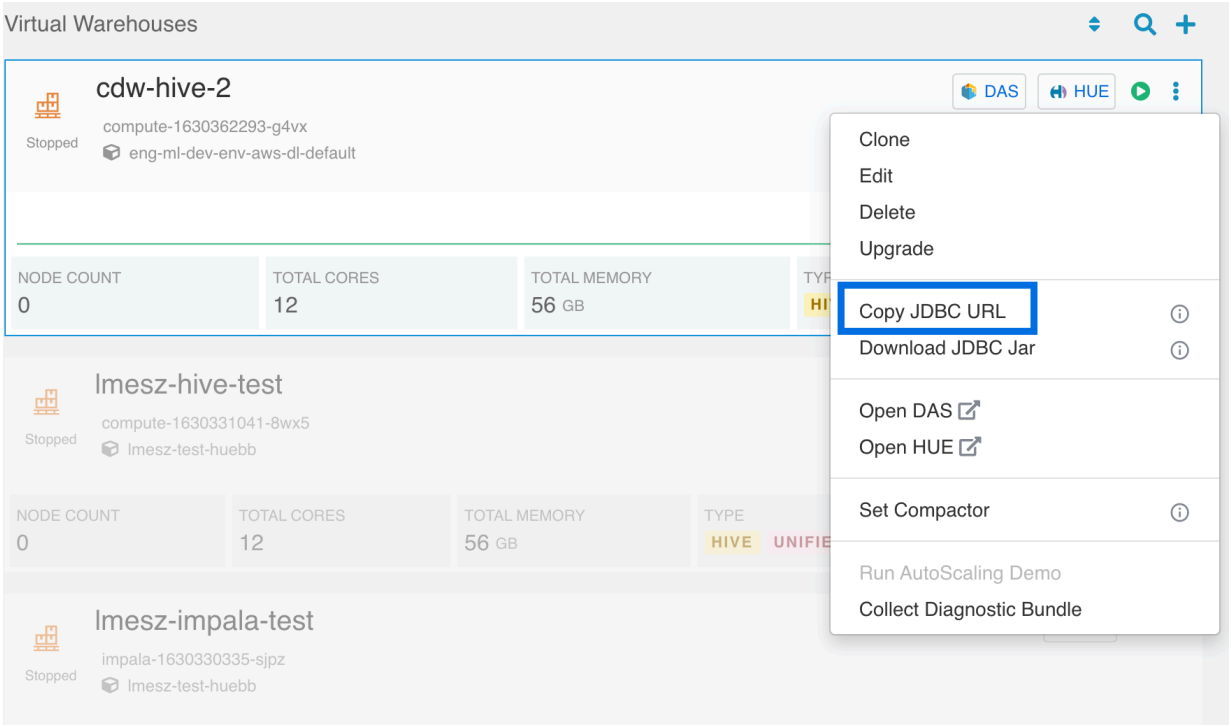
```
from impala.dbapi import connect
import os

#jdbc:hive2://hs2-cdw-hive.apps.shared-os-qe-01.kcloud.cloudera.com/default;
transportMode=http;httpPath=cliservice;socketTimeout=60;ssl=true;retries=3;k
erberosEnableCanonicalHostnameCheck=false;principal=hive/dwx-env-rhcxab-env.
cdp.local@QE-AD-1.CLOUDERA.COM

conn = connect(
    host='hs2-cdw-hive.apps.shared-os-qe-01.kcloud.cloudera.com', #copy this
 from jdbc url
    port=443, #copy this from jdbc url
    use_ssl=True, #if ssl=true in jdbc set this to True, otherwise false
    use_http_transport=True, #if transportMode=http in jdbc set this to t
rue, otherwise false
    kerberos_service_name='hive', #this is in the principal, before the / so
 in this example it's hive
    auth_mechanism='GSSAPI', #leave this as it is
    http_path="cliservice", #leave this as it is
    krb_host="dwx-env-rhcxab-env.cdp.local", #this is in the principal, the
 section after / and before @, in this example it's dwx-env-rhcxab-env.cdp
.local
)
```

Setting up a Hive or Impala data connection manually

Data connections to Hive or Impala virtual warehouses within the same environment as the Cloudera AI Workbench are automatically discovered and configured. You can also set up a data connection manually, which works across Cloudera environments. Follow this procedure to set up a Hive or Impala data connection.

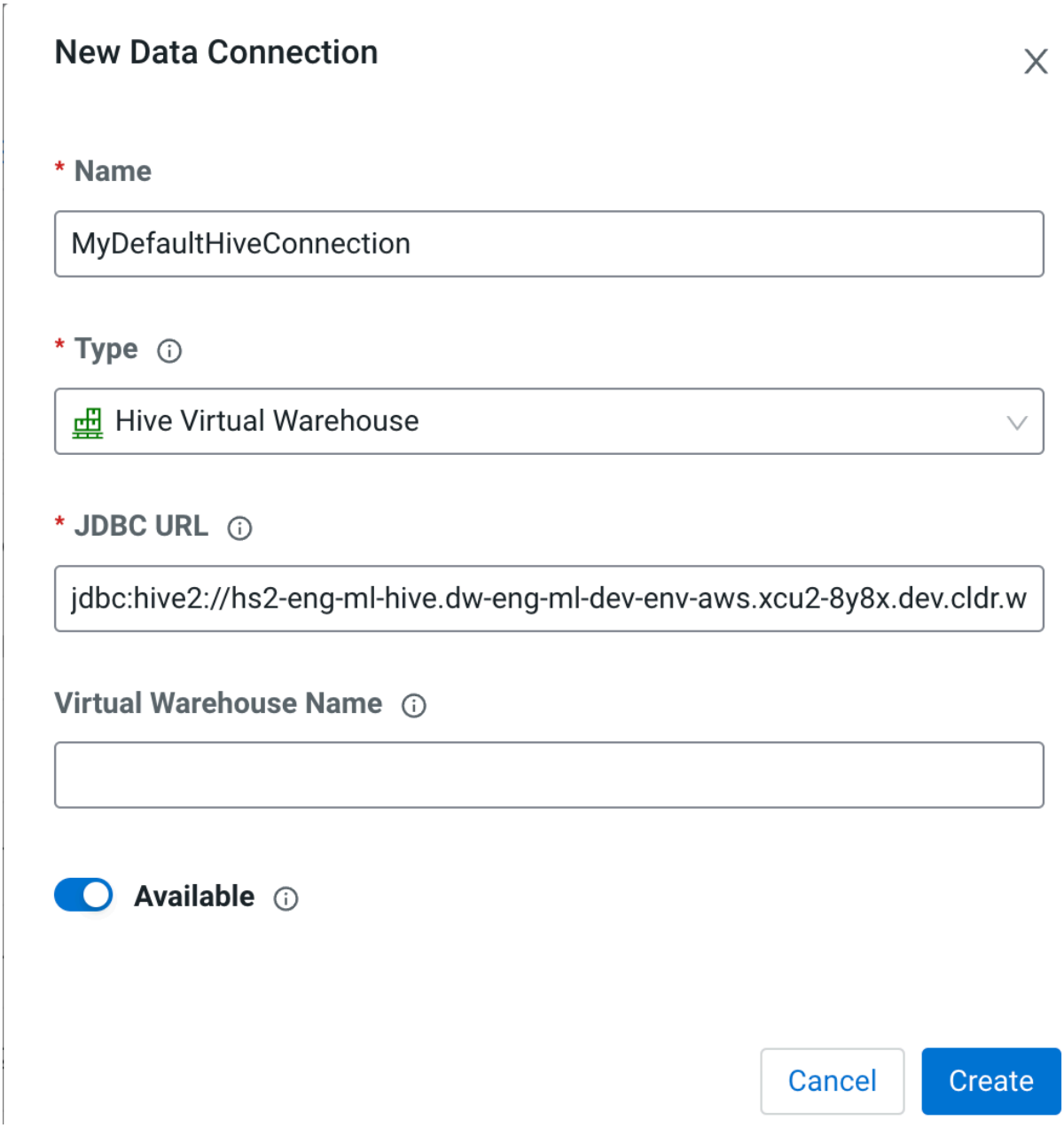
Procedure

1. Log into the Cloudera web interface and navigate to the Cloudera Data Warehouse service.
2. In the Cloudera Data Warehouse service, select Virtual Warehouses in the left navigation panel.
3. Select the options menu for the warehouse you want to access, and select Copy JDBC URL.



4. Return to the Cloudera AI service. In Site Administration Data Connections , select New Connection.
5. Enter the connection name. You cannot have duplicate names for data connections within a workbench or within a given project.
6. Select the connection type:
   a. Hive Virtual Warehouse
   b. Impala Virtual Warehouse
7. Paste the JDBC URL for the data connection.

8. (Optional) Enter the Virtual Warehouse Name. This is the name of the warehouse in Cloudera Data Warehouse.



Results

The data connection is available to users by default. To change availability, click the Available switch. This switch determines if the data connection is displayed in Projects created within the workbench.

Connecting to Hive and Impala services on Cloudera on premises Base

The provided examples use Kerberos for authentication when connecting to Cloudera Data Warehouse Hive, and Impala, which requires that the Keytab is set and there are proper permissions to access Cloudera Data Warehouse.

For details on the configuration values, referred to in the code snippets, follow the steps:

1. Go to the home page of the Cloudera Manager.

2. Find the base cluster Hive or Impala service you are interested in.
3. Click the horizontal dots icon next to the service.
4. Choose Configuration from among the available actions.

Connecting to Impala service on base clusters

Python

```
from impala.dbapi import connect
import os


#host=ccycloud-3.cml-pvc-ocp.root.comops.site (Impala Daemon from config)
#port=28000 (hs2_http_port from config)
#auth_mechanism="GSSAPI" (should always be GSSAPI)
#use_http_transport=True (should always be true)
#http_path="cliservice" (should always be cliservice)
#use_ssl=True (value should be found in the client_services_ssl_enabled pr
operty on impala base cluster service config)
# kerberos_service_name=impala (kerberos_princ_name from config)

#example

conn = connect(
            host="ccycloud-3.cml-pvc-ocp.root.comops.site",
            port=28000,
            auth_mechanism="GSSAPI",
            use_http_transport=True,
            http_path="cliservice",
            use_ssl=True,
            kerberos_service_name = "impala",
        )
```

Connecting to Hive service on base clusters

Python

```
from impala.dbapi import connect
import os

#host=ccycloud-1.cml-pvc-ocp.root.comops.site (HS2_SERVER Load Balancer - h
iveserver2_load_balancer from config)
#port=10015 (HS2_SERVER port - hiveserver2_load_balancer from config)
#auth_mechanism=GSSAPI (leave this as is)
#use_http_transport=True (leave this as true)
#http_path="cliservice" (leave this as is)
#use_ssl=True (if hive.server2.use.SSL from config is checked, this is tr
ue, otherwise false)
#kerberos_service_name=hive (kerberos_princ_name from config)

conn = connect(
            host="ccycloud-1.cml-pvc-ocp.root.comops.site",
            port=10015,
            auth_mechanism="GSSAPI",
            use_http_transport=True,
            http_path="cliservice",
            use_ssl=True,
            kerberos_service_name = "hive",
        )
```

Setting up a Spark data connection

Spark data connections within the same environment as Cloudera AI are automatically discovered, but you can also set up a connection manually. Follow this procedure to set up a Spark data connection.

Procedure

1. In the Workbench UI, select the link environment for the workbench you are using. This takes you to the Environments UI.
2. In Environments, select Data Lake Cloud Storage tabs.
3. Select the directory path shown for Hive Metastore External Warehouse, and copy it.
4. In Project Settings > Data Connections, click New Connection.
5. Enter a name for the connection.
6. Select the type: Spark Data Lake
7. Paste the value you copied in step 3 into Datalake Hive Metastore External Warehouse Directory.
8. Click Create.

Results

The data connection is available to users by default. To change availability, click the Available toggle.

Accessing data with Spark

When you are using Cloudera Data Warehouse, you can use Java Database Connectivity (JDBC).

JDBC is useful in the following cases:

1. Use JDBC connections when you have fine-grained access.
2. Use JDBC if the scale of data sent over the wire is on the order of tens of thousands of rows of data.

Add the Python code as described below, in the session where you want to utilize the data, and update the code with the data location information.

Permissions

In addition, check with the Administrator that you have the correct permissions to access the data lake. You will need a role that has read access only.

Setting up a JDBC connection

When using a JDBC connection, you read through a virtual warehouse that has Hive or Impala installed. You need to obtain the JDBC connection string, and paste it into the script in your session.

1. In Cloudera Data Warehouse, go to the Hive database containing your data.
2. From the kebab menu, click Copy JDBC URL.
3. Paste it into the script in your session.
4. Enter your user name and password in the script. Set up environmental variables to store these values, instead of hardcoding them in the script.

Using JDBC Connection with PySpark

PySpark can be used with Java Database Connectivity (JDBC), but it is not recommended. The recommended approach is to use Impyla for JDBC connections.

Procedure

1. In your session, open the workbench and add the following code.

2. Obtain the JDBC connection string, and paste it into the script where the “jdbc” string is shown. You will also need to insert your user name and password, or create environment variables for holding those values.

Example

This example shows how to read external Hive tables using Spark and a Hive Virtual Warehouse.

```
from pyspark.sql import SparkSession
from pyspark_llap.sql.session import HiveWarehouseSession

spark = SparkSession\
.builder\
.appName("CDW-CML-JDBC-Integration")\
.config("spark.security.credentials.hiveserver2.enabled","false")\
.config("spark.datasource.hive.warehouse.read.jdbc.mode", "client")\
.config("spark.sql.hive.hiveserver2.jdbc.url",
"jdbc:hive2://hs2-aws-2-hive-viz.env-j2ln9x.dw.ylcu-atmi.cloudera.site/def
ault;\
transportMode=http;httpPath=cliservice;ssl=true;retries=3;\
user=<username>;password=<password>")\
.getOrCreate()

hive = HiveWarehouseSession.session(spark).build()
hive.showDatabases().show()
hive.setDatabase("default")
hive.showTables().show()
hive.sql("select * from foo").show()
```

Related Information

Connecting to Cloudera Data Warehouse

Connecting to Iceberg tables

Cloudera AI supports data connections to Iceberg data lakes.

You can set up a manual connection using the provided snippet example. To connect with Iceberg, you must use Spark 3.

Make sure to set the correct DATALAKE_DIRECTORY environmental variable.

```
spark = (
SparkSession.builder.appName("MyApp")
.config("spark.sql.hive.hwc.execution.mode", "spark")
.config("spark.sql.extensions","com.qubole.spark.hiveacid.HiveAcidAutoConv
ertExtension,org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensio
ns")
.config("spark.sql.catalog.spark_catalog.type", "hive")
.config("spark.sql.catalog.spark_catalog", "org.apache.iceberg.spark.SparkC
atalog")
.config("spark.kerberos.access.hadoopFileSystems","hdfs://nn1.com:8032,hdfs
://nn2.com:8032,webhdfs://nn3.com:50070")
.config("spark.hadoop.iceberg.engine.hive.enabled", "true")
.config("spark.executorEnv.HADOOP_CONF_DIR", "/home/cdsw/hadoop_config_dir")
.config("spark.sql.iceberg.handle-timestamp-without-timezone", "true")
.config("spark.jars","/opt/spark/optional-11b/iceberg-spark-runtime.jar,/o
pt/spark/optional-11b/iceberg-hive-runtime.jar")
.config("spark.driver.userClassPathFirst", "true")
.config("spark.executor.userClassPathFirst", "true")
.config("spark.yarn.user.classpath.first", "true")
.getOrCreate()
)
```

Connecting to Hive tables via HWC

To access Hive from Spark, Hive Warehouse Connector (HWC) is needed. You can use the HWC to access Hive-managed tables from Spark.

```
spark = (
            SparkSession.builder.appName(self.app_name)
            .config("spark.jars", "/opt/spark/optional-lib/hive-warehouse-
connector-assembly.jar")
            .config("spark.sql.hive.hwc.execution.mode", "spark")
            .config(
                "spark.sql.extensions",
             "com.qubole.spark.hiveacid.HiveAcidAutoConvertExtension",
            )
            .config(
                "spark.kryo.registrator",
                "com.qubole.spark.hiveacid.util.HiveAcidKyroRegistrator",
            )
            .config("spark.sql.catalog.spark_catalog.type", "hive")
             .config("spark.yarn.access.hadoopFileSystems","<DATALAKE_DIRECT
ORY>")
            .getOrCreate()
        )
```

Connecting to Ozone filesystem

In Cloudera AI, you can connect Spark to the Ozone object store with a script.

The script, in Scala, counts the number of word occurrences in a text file. The key point in this example is to use the following string to refer to the text file: ofs://omservice1/s3v/hivetest/spark/jedi_wisdom.txt

Word counting example in Scala

```
import sys.process._

// Put the input file into Ozone
//"hdfs dfs -put data/jedi_wisdom.txt ofs://omservice1/s3v/hivetest/spark" !
// Set the following spark setting in the file "spark-defaults.conf" on
 the Cloudera AI session using terminal
//spark.yarn.access.hadoopFileSystems=ofs://omservice1/s3v/hivetest
//count lower bound
val threshold = 2
// this file must already exist in hdfs, add a
// local version by dropping into the terminal.
val tokenized = sc.textFile("ofs://omservice1/s3v/hivetest/spark/jedi_wisd
om.txt").flatMap(_.split(" "))
// count the occurrence of each word
val wordCounts = tokenized.map((_ , 1)).reduceByKey(_ + _)
// filter out words with fewer than threshold occurrences
val filtered = wordCounts.filter(_._2 >= threshold)
System.out.println(filtered.collect().mkString(","))
```

Accessing Ozone storage

In Cloudera AI you can connect Cloudera AI to the Ozone object store using a script or command line commands.

Creating an Ozone data connection

Cloudera AI supports data connections to Ozone file systems.

You can set up a manual connection using the provided snippet example. To connect to Ozone, you must use Spark 3.

Set the following parameters:

- DATALAKE_DIRECTORY
- Valid database and table name in the describe formatted SQL command.

```
from pyspark.sql import SparkSession
# Change to the appropriate Datalake directory location
DATALAKE_DIRECTORY = "s3a://your-aws-demo/"

spark = (
  SparkSession.builder.appName("MyApp")
  .config("spark.jars", "/opt/ozone-addon/jar/ozone-filesystem-hadoop3.jar")
  .config("spark.yarn.access.hadoopFileSystems", DATALAKE_DIRECTORY)
  .getOrCreate()
  )

spark.sql("show databases").show()
spark.sql("describe formatted <database_name>.<table_name>").show()
```

Connecting to Ozone filesystem

In Cloudera AI, you can connect Spark to the Ozone object store with a script.

The script, in Scala, counts the number of word occurrences in a text file. The key point in this example is to use the following string to refer to the text file: ofs://omservice1/s3v/hivetest/spark/jedi_wisdom.txt

Word counting example in Scala

```
import sys.process._

// Put the input file into Ozone
//"hdfs dfs -put data/jedi_wisdom.txt ofs://omservice1/s3v/hivetest/spark" !
// Set the following spark setting in the file "spark-defaults.conf" on
 the Cloudera AI session using terminal
//spark.yarn.access.hadoopFileSystems=ofs://omservice1/s3v/hivetest
//count lower bound
val threshold = 2
// this file must already exist in hdfs, add a
// local version by dropping into the terminal.
val tokenized = sc.textFile("ofs://omservice1/s3v/hivetest/spark/jedi_wisd
om.txt").flatMap(_.split(" "))
// count the occurrence of each word
val wordCounts = tokenized.map((_ , 1)).reduceByKey(_ + _)
// filter out words with fewer than threshold occurrences
val filtered = wordCounts.filter(_._2 >= threshold)
System.out.println(filtered.collect().mkString(","))
```

Accessing local files in Ozone

You can access files in Ozone on a local file system using hdfsCLI. This method works with both legacy engines and runtime sessions.

The following commands enable a Cloudera AI session to connect to Ozone using the OFS protocol.

1. Put the input file into Ozone:

```
hdfs dfs -put data/jedi_wisdom.txt ofs://omservice1/s3v/hivetest/spark
```

2. List the files in Ozone:

```
hdfs dfs -ls ofs://omservice1/s3v/hivetest/
```

3. Download file from ozone to local:

```
hdfs dfs -copyToLocal ofs://omservice1/s3v/hivetest/spark data/jedi_wisd
om.txt
```

Connecting to external Amazon S3 buckets

Each programming language supported by Cloudera AI includes libraries for uploading data to and downloading data from Amazon S3.

To work with external S3 buckets using Python, follow these steps:

- Add your Amazon Web Services access keys to your project's environment variables as AWS_ACCESS_KEY_ID and AWS_SECRET_ACCESS_KEY.
- Add your Ozone S3 gateway to the environment variables as OZONE_S3_GATEWAY.

Python

```
# Install Boto to the project
!pip3 install boto3

# Make sure below environment variables are set
# ozone s3 gateway : os.environ['OZONE_S3_GATEWAY']
# s3 keys from os.environ['AWS_ACCESS_KEY_ID'] and os.environ['AWS_SECRET
_ACCESS_KEY']

import os
import boto3

# Use Boto to connect to S3 and get a list of objects from a bucket
conn = boto3.session.Session()

s3g = os.environ['OZONE_S3_GATEWAY']
access_key = os.environ['AWS_ACCESS_KEY_ID']
secret_key = os.environ['AWS_SECRET_ACCESS_KEY']

s3_client = conn.client(
    service_name='s3',
    endpoint_url=s3g
)


test_bucket = 'testozones3'
s3_client.create_bucket(Bucket=test_bucket)
all_buckets = s3_client.list_buckets()
print(f"All S3 Buckets are {[i['Name'] for i in all_buckets['Buckets']]}")

s3_client.put_object(Bucket=test_bucket, Key='README.md')

all_objs = s3_client.list_objects(Bucket=test_bucket)
print(f"All keys in {bucket_name} are {[i['Key']for i in all_objs['Contents'
]]}")

s3_client.get_object(Bucket=test_bucket, Key='README.md')

ssl = "true" if s3g.startswith("https") else "false"
s3a_path = f"s3a://{test_bucket}/"
```

```
hadoop_opts = f"-Dfs.s3a.access.key='{access_key}' -Dfs.s3a.secret.key='{
secret_key}' -Dfs.s3a.endpoint='{s3g}' -Dfs.s3a.connection.ssl.enabled={ssl}
  -Dfs.s3a.path.style.access=true"

!hdfs dfs {hadoop_opts} -ls "s3a://{test_bucket}/"
```

Connect to External SQL Databases

Every language in Cloudera AI has multiple client libraries available for SQL databases.

If your database is behind a firewall or on a secure server, you can connect to it by creating an SSH tunnel to the server, then connecting to the database on localhost.

If the database is password-protected, consider storing the password in an environmental variable to avoid displaying it in your code or in consoles. The examples below show how to retrieve the password from an environment variable and use it to connect.

Python

You can access data using SQLAlchemy:

```
!pip install sqlalchemy
import os

import sqlalchemy
from sqlalchemy.orm import sessionmaker
from sqlalchemy import create_engine
db = create_engine("postgresql://cdswuser:%s@localhost:5432/test_db" % os
.environ["POSTGRESQL_PASSWORD"])
session = sessionmaker(bind=db)
user = session.query(User).filter_by(name='ed').first()
```

R

You can access remote databases with dplyr.

```
install.packages("dplyr")
library("dplyr")
db <- src_postgres(dbname="test_db", host="localhost", port=5432, user="cds
wuser", password=Sys.getenv("POSTGRESQL_PASSWORD"))
flights_table <- tbl(db, "flights")
select(flights_table, year:day, dep_delay, arr_delay)
```