

Accessing Data from CML

Date published: 2020-07-16

Date modified: 2022-04-11

CLOUDERA

Legal Notice

© Cloudera Inc. 2023. 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

| | |
|--|-----------|
| Set up a data connection for CDP One..... | 4 |
| Set up a CDP One data connection using raw code..... | 5 |
| Accessing data with Spark..... | 6 |
| Use JDBC Connection with PySpark..... | 7 |
| Use JDBC Connection with SparklyR..... | 7 |
| Use Direct Reader Mode with PySpark..... | 8 |
| Use Direct Reader Mode with SparklyR..... | 9 |
| Accessing Local Data from Your Computer..... | 10 |
| Accessing Data from HDFS..... | 11 |
| Accessing Data from Apache Hive..... | 12 |
| Accessing Data from Apache Impala..... | 12 |
| Loading CSV Data into an Impala Table..... | 12 |
| Running Queries on Impala Tables..... | 13 |
| Accessing Data in Amazon S3 Buckets..... | 15 |
| Accessing External SQL Databases..... | 16 |
| Accessing a Data Lake from CML..... | 17 |
| Example: Connect a Spark session to Hive Metastore in a Data Lake..... | 22 |
| Accessing CDW from CML..... | 25 |
| Accessing Ozone from Spark..... | 26 |

Set up a data connection for CDP One

In CDP One, you can set up a data connection to the DataHub cluster. You can set up a connection using the New Connection dialog, or by using raw code inside your project. Both approaches are shown below.

Procedure


1. Log into the CDP One console web UI.
2. Depending on which connection you want to use, click on either Connect to Hive or Connect to Impala tile.
3. Ensure your Data Hub cluster name is correct in the popup.
4. Copy the JDBC URL string.
5. Now click on the Build a Data Science Project tile to log into the the ML workspace.
6. In Site Administration Data Connections , select New Connection.
7. Return to the Machine Learning service. In Site Administration Data Connections , select New Connection.
8. Enter the connection name. You cannot have duplicate names for data connections within a workspace or within a given project.
9. Select the connection type:
 - a. Hive Virtual Warehouse
 - b. Impala Virtual Warehouse
10. Paste the JDBC URL for the data connection.

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

New Data Connection ✕

*** Name**

*** Type** ⓘ

 Hive Virtual Warehouse ▼

*** JDBC URL** ⓘ

Virtual Warehouse Name ⓘ

☒ **Available** ⓘ

CancelCreate

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 workspace.

Set up a CDP One data connection using raw code

It is recommended to use the New Connection dialog to create a new data connection. If needed, you can also set up a data connection in your project code by using and adapting the following code snippet.

Example

```
from impala.dbapi import connect  
  
#Example connection string:
```

```
# jdbc:hive2://my-test-master0.eng-ml-i.svbr-nqvp.int.cldr.work/;ssl=true;
transportMode=http;httpPath=my-test/cdp-proxy-api/hive

conn = connect(
    host = "my-test-master0.eng-ml-i.svbr-nqvp.int.cldr.work",
    port = 443,
    auth_mechanism = "LDAP",
    use_ssl = True,
    use_http_transport = True,
    http_path = "my-test/cdp-proxy-api/hive",
    user = "csso_me",
    password = "Test@123")
cursor = conn.cursor()
cursor.execute("select * from 3yearpop")

for row in cursor:
    print(row)
cursor.close()
conn.close()
```

Accessing data with Spark

There are two general methods to access data with Spark, with various benefits and disadvantages, depending on how the data is managed. The two methods are Direct Reader and JDBC.

1. JDBC: Use in the following cases:

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

2. Direct Reader: Use in the following cases:

- a. Your data has table-level access control, and does not have row-level security or column level masking (fine-grained access.)

For both methods, add the Python or R 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. For more information, see [-link-](#).

How to obtain the Data Lake directory location

1. In the CDP home page, select Management Console.
2. In Environments, select the Environment you are using.
3. In the tabbed section, select Cloud Storage.
4. Choose the location where your data is stored.
5. For managed data tables, copy the location shown by Hive Metastore Warehouse.
6. For external unmanaged data tables, copy the location shown by Hive Metastore External Warehouse
7. Paste the location into the script in line 9. If you are using AWS, the location starts with s3:, and if you are using Azure, it starts with abfs:. If you are using a different location in the data lake, the default path is shown by Hbase Root.

Check for an updated version of the jar file

This jar file name may need to be updated with the correct version number.

1. Click Terminal Access, and run: `ls /usr/lib/hive_warehouse_connector/`
2. Check the file name and version number of the jar file.

Set 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 CDW, 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. You also have to enter your user name and password in the script. You should set up environmental variables to store these values, instead of hardcoding them in the script.

Use JDBC Connection with PySpark

Before you begin

Procedure

1. In your session, open the workbench and add the following code.
2. Obtain the JDBC connection string, as described above, 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

```
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/default;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()
```

Use JDBC Connection with SparklyR

Before you begin

Obtain the JDBC connection string, as described above, 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.

Procedure

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

Example

```
# Run this once
#install.packages("sparklyr")
library(sparklyr)
config <- spark_config()

config$spark.security.credentials.hiveserver2.enabled="false"
config$spark.datasource.hive.warehouse.read.via.llap="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/default;transportMode=http;httpPath=c
liservice;ssl=true;retries=3;user=<username>;password=<password>"

sc <- spark_connect(config = config)

ss <- spark_session(sc)
hive <- invoke_static(sc,"com.hortonworks.hwc.HiveWarehouseSession", "sessio
n",ss)%>%invoke("build")

df <- invoke(hive,"execute","select * from default.foo")
sparklyr::sdf_collect(df)

spark_disconnect(sc)
```

Use Direct Reader Mode with PySpark

Before you begin

Make sure to update the following parameters in the code sample below:

Procedure

1. spark.yarn.access.hadoopFileSystems: Enter the location where your data is stored.
2. spark.jars: Update the Hive Warehouse Connector .jar file, if necessary.

Example

```
from pyspark.sql import SparkSession

spark = SparkSession\
.builder\
.appName("CDW-CML-Spark-Direct")\
.config("spark.sql.hive.hwc.execution.mode", "spark")\
.config("spark.sql.extensions", "com.qubole.spark.hiveacid.HiveAcidAutoConv
ertExtension")\
.config("spark.kryo.registrator", "com.qubole.spark.hiveacid.util.HiveAcidKyr
oRegistrator")\
.config("spark.yarn.access.hadoopFileSystems", "s3a://demo-aws-2/")\
.config("spark.jars", "/usr/lib/hive_warehouse_connector/hive-warehouse-conn
ector-assembly-1.0.0.7.2.2.0-244.jar")\
.getOrCreate()

### The following commands test the connection
```



```
spark.sql("show databases").show()
spark.sql("describe formatted test_managed").show()
spark.sql("select * from test_managed").show()
spark.sql("describe formatted test_external").show()
spark.sql("select * from test_external").show()
```

Use Direct Reader Mode with SparklyR

Before you begin

You need to add the location of your data tables in the example below.

Procedure

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

Example

```
#Run this once
#install.packages("sparklyr")

library(sparklyr)
config <- spark_config()

config$spark.security.credentials.hiveserver2.enabled="false"
config$spark.datasource.hive.warehouse.read.via.llap="false"
config$spark.sql.hive.hwc.execution.mode="spark"
#config$spark.datasource.hive.warehouse.read.jdbc.mode="spark"

# Required setting for HWC-direct reader - the hiveacid sqlextension does
# the automatic
# switch between reading through HWC (for managed tables) or spark-native
# (for external)
# depending on table type.
config$spark.sql.extensions="com.qubole.spark.hiveacid.HiveAcidAutoConver
tExtension"
config$spark.kryo.registrator="com.qubole.spark.hiveacid.util.HiveAcidKyroRe
gistrator"

# Pick the correct jar location by using Terminal Access
# to file the exact file path/name of the hwc jar under /usr/lib/hive_ware
house_connector
config$sparklyr.jars.default <- "/usr/lib/hive_warehouse_connector/hive-ware
house-connector-assembly-1.0.0.7.2.2.0-244.jar"
# File system read access - this s3a patha is available from the environment
Cloud Storage.
# To read from this location go to Environment->Manage Access->IdBroker Map
pings
# ensure that the user (or group) has been assigned an AWS IAM role that
can
#. read this location.
#config$spark.yarn.access.hadoopFileSystems="s3a://demo-aws-2/"
config$spark.yarn.access.hadoopFileSystems="s3a://demo-aws-2/datalake/war
ehouse/tablespace/managed/hive"

sc <- spark_connect(config = config)
```

```
intDf1 <- sparklyr::spark_read_table(sc, 'foo')
sparklyr::sdf_collect(intDf1)

intDf1 <- sparklyr::spark_read_table(sc, 'foo_ext')
sparklyr::sdf_collect(intDf1)

spark_disconnect(sc)

# Optional configuration - only needed if the table is in a private Data Ca
talog of CDW
#config$spark.sql.hive.hiveserver2.jdbc.url="jdbc:hive2://hs2-aws-2-hive-vi
z.env-j2ln9x.dw.ylcu-atmi.cloudera.site/default;transportMode=http;httpPath=
cliservice;ssl=true;retries=3;user=priyankp;password=!Password1"

#ss <- spark_session(sc)
#hive <- invoke_static(sc,"com.hortonworks.hwc.HiveWarehouseSession","ses
sion",ss)%>%invoke("build")
#df <- invoke(hive,"execute","select * from default.foo limit 199")
#sparklyr::sdf_collect(df)
```

Accessing Local Data from Your Computer

This topic includes code samples that demonstrate how to access local data for CML workloads.

If you want to perform analytics operations on existing data files (.csv, .txt, etc.) from your computer, you can upload these files directly to your Cloudera Machine Learning project. Go to the project's Overview page. Under the Files section, click Upload and select the relevant data files to be uploaded.

The following sections use the [tips.csv](#) dataset to demonstrate how to work with local data stored within your project. Upload this dataset to the data folder in your project before you run these examples.

Pandas (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)
```

dplyr (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))
```

Accessing Data from HDFS

There are many ways to access HDFS data from R, Python, and Scala libraries. The following code samples demonstrate how to count the number of occurrences of each word in a simple text file in HDFS.

Navigate to your project and click Open Workbench. Create a file called `sample_text_file.txt` and save it to your project in the data folder. Now write this file to HDFS. You can do this in one of the following ways:

- Click Terminal above the Cloudera Machine Learning console and enter the following command to write the file to HDFS:

```
hdfs dfs -put data/sample_text_file.txt /tmp
```

OR

- Use the workbench command prompt:

Python Session

```
!hdfs dfs -put data/sample_text_file.txt /tmp
```

R Session

```
system("hdfs dfs -put data/tips.csv /user/hive/warehouse/tips/")
```

The following examples use Python and Scala to read `sample_text_file.txt` from HDFS (written above) and perform the count operation on it.

Python

```
from __future__ import print_function
import sys, re
from operator import add
from pyspark.sql import SparkSession

spark = SparkSession\
    .builder\
    .appName("PythonWordCount")\
    .getOrCreate()

# Access the file
lines = spark.read.text("/tmp/sample_text_file.txt").rdd.map(lambda r: r[0])
counts = lines.flatMap(lambda x: x.split(' ')) \
    .map(lambda x: (x, 1)) \
    .reduceByKey(add) \
    .sortBy(lambda x: x[1], False)
output = counts.collect()
for (word, count) in output:
    print("%s: %i" % (word, count))

spark.stop()
```

Scala

```
//count lower bound
```

```
val threshold = 2

// read the file added to hdfs
val tokenized = sc.textFile("/tmp/sample_text_file.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 Data from Apache Hive

The following code sample demonstrates how to establish a connection with the Hive metastore and access data from tables in Hive.

Python

```
import os
import pandas
from impala.dbapi import connect
from impala.util import as_pandas

# Specify HIVE_HMS_HOST as an environment variable in your project settings
HIVE_HMS_HOST = os.getenv('HIVE_HS2_HOST', '<hiveserver2_hostname>')

# This connection string depends on your cluster setup and authentication mechanism
conn = connect(host=HIVE_HS2_HOST,
               port='10000',
               auth_mechanism='GSSAPI',
               kerberos_service_name='hive')
cursor = conn.cursor()
cursor.execute('SHOW TABLES')
tables = as_pandas(cursor)
tables
```

Accessing Data from Apache Impala

In this section, we take some sample data in the form of a CSV file, save the contents of this file to a table in Impala, and then use some common Python and R libraries to run simple queries on this data.

Loading CSV Data into an Impala Table

For this demonstration, we will be using the [tips.csv](#) dataset. Use the following steps to save this file to a project in Cloudera Machine Learning, and then load it into a table in Apache Impala.

1. Create a new [Cloudera Machine Learning project](#).
2. Create a folder called data and upload tips.csv to this folder. For detailed instructions, see [Managing Project Files](#).

3. The next steps require access to services on the CDH cluster. If Kerberos has been enabled on the cluster, enter your credentials (username, password/keytab) in Cloudera Machine Learning to enable access.
4. Navigate back to the project Overview page and click Open Workbench.
5. Launch a new session (Python or R).
6. Open the Terminal.
 - a. Run the following command to create an empty table in Impala called tips. Replace `<impala_daemon_hostname>` with the hostname for your Impala daemon.

```
impala-shell -i <impala_daemon_hostname>:21000 -q '
CREATE TABLE default.tips (
  `total_bill` FLOAT,
  `tip` FLOAT,
  `sex` STRING,
  `smoker` STRING,
  `day` STRING,
  `time` STRING,
  `size` TINYINT)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ","
LOCATION "hdfs:///user/hive/warehouse/tips/";'
```

- b. Run the following command to load data from the `/data/tips.csv` file into the Impala table.

```
hdfs dfs -put data/tips.csv /user/hive/warehouse/tips/
```

Running Queries on Impala Tables

This section demonstrates how to run queries on the tips table created in the previous section using some common Python and R libraries such as Pandas, Impyla, Sparklyr and so on. All the examples in this section run the same query, but use different libraries to do so.

PySpark (Python)

```
from pyspark.sql import SparkSession

spark = SparkSession.builder.master('yarn').getOrCreate()
# load data from .csv file in HDFS
# tips = spark.read.csv("/user/hive/warehouse/tips/", header=True, inferSchema=True)

# OR load data from table in Hive metastore
tips = spark.table('tips')

from pyspark.sql.functions import col, lit, mean

# query using DataFrame API
tips \
  .filter(col('sex').like("%Female%")) \
  .groupBy('day') \
  .agg(mean('tip').alias('avg_tip')) \
  .orderBy('avg_tip', ascending=False) \
  .show()

# query using SQL
spark.sql('''
SELECT day,AVG(tip) AS avg_tip \
FROM tips \
WHERE sex LIKE "%Female%" \
GROUP BY day \
''')
```

```
ORDER BY avg_tip DESC').show()

spark.stop()
```

Impyla (Python)

Due to an incompatibility with the thrift_sasl package, Impyla has been known to fail with Python 3.

Python 2

```
# (Required) Install the impyla package
# !pip install impyla
# !pip install thrift_sasl
import os
import pandas
from impala.dbapi import connect
from impala.util import as_pandas
# Connect to Impala using Impyla
# Secure clusters will require additional parameters to connect to Impala.
# Recommended: Specify IMPALA_HOST as an environment variable in your project settings
IMPALA_HOST = os.getenv('IMPALA_HOST', '<impala_daemon_hostname>')
conn = connect(host=IMPALA_HOST, port=21050)
# Execute using SQL
cursor = conn.cursor()
cursor.execute('SELECT day,AVG(tip) AS avg_tip \
               FROM tips \
               WHERE sex ILIKE "%Female%" \
               GROUP BY day \
               ORDER BY avg_tip DESC')

# Pretty output using Pandas
tables = as_pandas(cursor)
tables
```

Ibis (Python)

```
# (Required) Install the ibis-framework[impala] package
# !pip3 install ibis-framework[impala]

import ibis
import os
ibis.options.interactive = True
ibis.options.verbose = True

# Connection to Impala
# Secure clusters will require additional parameters to connect to Impala.
# Recommended: Specify IMPALA_HOST as an environment variable in your project settings
IMPALA_HOST = os.getenv('IMPALA_HOST', '<impala_daemon_hostname>')
con = ibis.impala.connect(host=IMPALA_HOST, port=21050, database='default')
con.list_tables()

tips = con.table('tips')

tips \
    .filter(tips.sex.like(['%Female%'])) \
    .group_by('day') \
    .aggregate( \
        avg_tip=tips.tip.mean() \
    ) \
```

```
.sort_by(ibis.desc('avg_tip')) \
.execute()
```

Sparklyr (R)

```
# (Required) Install the sparklyr package
# install.packages("sparklyr")

library(stringr)
library(sparklyr)
library(dplyr)
spark <- spark_connect(master = "yarn")

# load data from file in HDFS
tips <- spark_read_csv(
  sc = spark,
  name = "tips",
  path = "/user/hive/warehouse/tips/"
)

# OR load data from table
tips <- tbl(spark, "tips")

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

# query using SQL
tbl(spark, sql("
  SELECT day,AVG(tip) AS avg_tip \
  FROM tips \
  WHERE sex LIKE '%Female%' \
  GROUP BY day \
  ORDER BY avg_tip DESC"))
spark_disconnect(spark)
```

Accessing Data in Amazon S3 Buckets

Every language in Cloudera Machine Learning has libraries available for uploading to and downloading from Amazon S3.

To work with S3:

1. Add your Amazon Web Services [access keys](#) to your project's [environment variables](#) as `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`.
2. Pick your favorite language from the code samples below. Each one downloads the R 'Old Faithful' dataset from S3.

R

```
library("devtools")
install_github("armstrtw/AWS.tools")
```

```

Sys.setenv("AWSACCESSKEY"=Sys.getenv("AWS_ACCESS_KEY_ID"))
Sys.setenv("AWSSECRETKEY"=Sys.getenv("AWS_SECRET_ACCESS_KEY"))
library("AWS.tools")

s3.get("s3://sense-files/faithful.csv")

```

Python

```

# Install Boto to the project
!pip install boto
# Create the Boto S3 connection object.
from boto.s3.connection import S3Connection
aws_connection = S3Connection()

# Download the dataset to file 'faithful.csv'.
bucket = aws_connection.get_bucket('sense-files')
key = bucket.get_key('faithful.csv')
key.get_contents_to_filename('/home/cdsw/faithful.csv')

```

Accessing External SQL Databases

Every language in Cloudera Machine Learning 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 [pyodbc](#) or [SQLAlchemy](#)

```

# pyodbc lets you make direct SQL queries.
!wget https://pyodbc.googlecode.com/files/pyodbc-3.0.7.zip
!unzip pyodbc-3.0.7.zip
!cd pyodbc-3.0.7;python setup.py install --prefix /home/cdsw
import os

# See http://www.connectionstrings.com/ for information on how to construct
  ODBC connection strings.
db = pyodbc.connect("DRIVER={PostgreSQL Unicode};SERVER=localhost;PORT=54
32;DATABASE=test_db;USER=cdswuser;OPTION=3;PASSWORD=%s" % os.environ["POSTGR
ESQL_PASSWORD"])
cursor = cnxn.cursor()
cursor.execute("select user_id, user_name from users")

# sqlalchemy is an object relational database client that lets you make data
base queries in a more Pythonic way.
!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["POSTGRES_PASSWORD"])
session = sessionmaker(bind=db)
user = session.query(User).filter_by(name='ed').first()
```

R

```
# dplyr lets you program the same way with local data frames and remote SQL databases.
install.packages("dplyr")
library("dplyr")
db <- src_postgres(dbname="test_db", host="localhost", port=5432, user="cdswuser", password=Sys.getenv("POSTGRES_PASSWORD"))
flights_table <- tbl(db, "flights")
select(flights_table, year:day, dep_delay, arr_delay)
```

Accessing a Data Lake from CML

CML can access data tables stored in an AWS or Microsoft Azure Data Lake. As a CML Admin, follow this procedure to set up the necessary permissions.

About this task

The instructions apply to Data Lakes on both AWS and Microsoft Azure. Follow the instructions that apply to your environment.

Procedure

1. Cloud Provider Setup

Make sure the prerequisites for AWS or Azure are satisfied (see the Related Topics for AWS environments and Azure environments). Then, create a CDP environment as follows.

- a) For environment logs, create an S3 bucket or ADLS Gen2 container.



Note: For ADLS Gen2, create a dedicated container for logs, and a dedicated container for data, within the same account.

- b) For environment storage, create an S3 bucket or ADLS Gen2 container.
- c) For AWS, create AWS policies for each S3 bucket, and create IAM roles (simple and instance profiles) for these policies.
- d) For Azure, create managed identities for each of the personas, and create roles to map the identities to the ADLS permissions.

For detailed information on S3 or ADLS, see Related information.

2. Environment Setup

In CDP, set up paths for logs and native data access to the S3 bucket or ADLS Gen2 container.

In the Environment Creation wizard, set the following:

a) Logs Storage and Audits


1. Instance Profile - The IAM role or Azure identity that is attached to the master node of the Data Lake cluster. The Instance Profile enables unauthenticated access to the S3 bucket or ADLS container for logs.
2. Logs Location Base - The location in S3 or ADLS where environment logs are saved.



Note: The instance profile or Azure identity must refer to the same logs location base in S3 or ADLS.

3. Ranger Audit Role - The IAM role or Azure identity that has S3 or ADLS access to write Ranger audit events. Ranger uses Hadoop authentication, therefore it uses IDBroker to access the S3 bucket or ADLS container, rather than using Instance profiles or Azure identities directly.

b) Data Access



Data Access

Provide an existing location where workload data will be stored.

Select an Instance Profile*

[Click here](#) to refresh instance profiles from the cloud provider.

mlx-dev-prod-env_IDBROKER_ROLE
?

Storage Location Base*

s3a://
fooenv/storage
?

Data Access Role*

arn:aws:iam::886883559913:role/mlx-dev-prod-env_DATA LAKE_AI
?

ID Broker Mappings

You may optionally provide mappings for users or groups.

| | | | | |
|----------------|--------------------|-----------|------------------------------|--------|
| User or Group: | ml-data-scientists | Role Arn: | arn:aws:iam::886883559913:ro | Remove |
| User or Group: | ml-data-engineers | Role Arn: | arn:aws:iam::886883559913:ro | Remove |
| User or Group: | ml-dl-admins | Role Arn: | arn:aws:iam::886883559913:ro | Remove |

Add

1. Instance Profile - The IAM role or Azure identity that is attached to the IDBroker node of the Data Lake cluster. IDBroker uses this profile to assume roles on behalf of users and get temporary credentials to access S3 buckets or ADLS containers.
2. Storage Location Base - The S3 or ADLS location where data pertaining to the environment is saved.
3. Data Access Role - The IAM role or Azure identity that has access to read or write environment data. For example, Hive creates external tables by default in the CDP environments, where metadata is stored in HMS running in the Data Lake. The data itself is stored in S3 or ADLS. As Hive uses Hadoop authentication, it uses IDBroker to access S3 or ADLS, rather than using Instance profiles or Azure identities. Hive uses the data access role for storage access.



Note: The data access role must have permissions to access the S3 or ADLS storage location.

4. ID Broker Mappings - These specify the mappings between the CDP user or groups to the AWS IAM roles or Azure roles that have appropriate S3 or ADLS access. This setting enables IDBroker to get appropriate S3 or ADLS credentials for the users based on the role mappings defined.



Note: There is no limit to the number of mappings that one can define but each user can only be assigned to one of the role mappings.

This completes installation of the environment.

3. User Group Mappings

In CDP, you can assign users to groups to simplify permissions management. For example, you could create a group called ml-data-scientists, and assign two individual users to it, as shown here. For instructions, see link.

The screenshot shows the Cloudera Management Console interface. On the left is a sidebar with navigation links: Dashboard, Environments, Data Lakes, **User Management**, Data Hub Clusters, Data Warehouses, ML Workspaces, and Classic Clusters. The main content area is titled 'Groups / ml-data-scientists'. It displays the group's details: Name (ml-data-scientists), CRN (crm.altus:iam:us-west-1:12a0079b-1591-4ca0-b721-a446bda74e67:group:...), Creation Date (09/04/2019 11:50 AM PDT), Sync Membership On User Login (Yes), and Members (2). An 'Actions' button is visible. Below the details, there are tabs for 'Members', 'Roles', 'Resources', and 'Admins'. The 'Members' tab is active, showing a table with columns 'Type', 'Name', and 'Email'. Two members are listed: Nagaram Prasad Addepally (ram@cloudera.com) and Vamsee Yarlagaadda (vamsee@cloudera.com). An 'Add a member' button is at the top of the members list.

a. Sync users

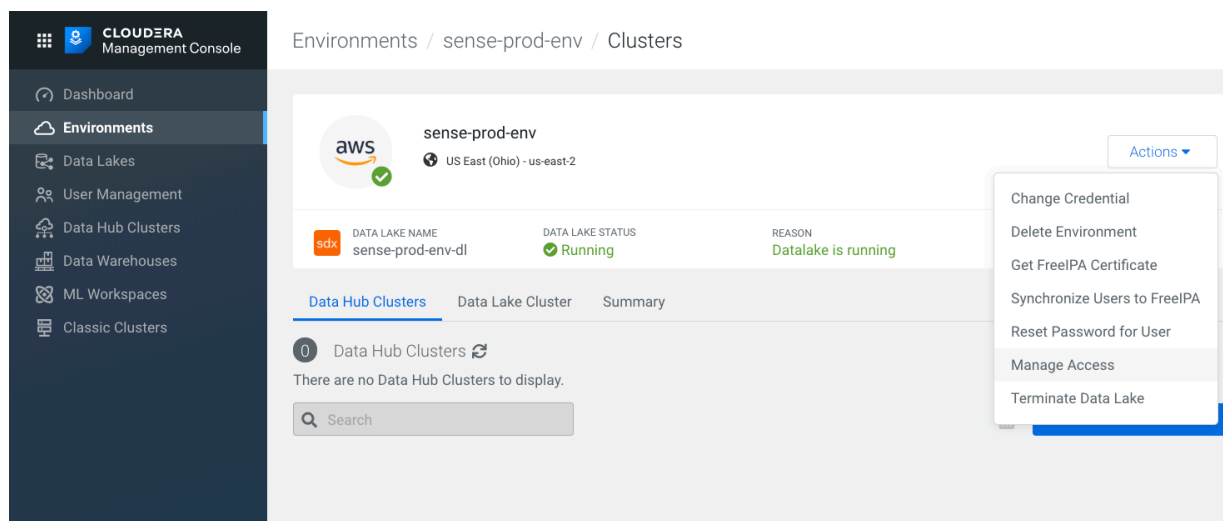
Whenever you make changes to user and group mappings, make sure to sync the mappings with the authentication layer. In User Management > Actions, click Sync Users, and select the environment.

The screenshot shows the Cloudera Management Console 'User Management' page. The sidebar is the same as in the previous screenshot. The main content area is titled 'User Management' and has tabs for 'Users', 'Groups', and 'Identity Providers'. The 'Users' tab is active. It features a search bar, filters for 'Type' (All) and 'Identity Provider' (All), and an 'Actions' button. A table lists users with columns: Type, Name, Email, Identity Provider, Workload User Name, and a status column. A dropdown menu is open from the 'Actions' button, showing options: 'Create Machine User', 'Upload Users', 'Sync Users' (highlighted), and 'Update Account Messages'. The table lists several users, including Aadarsh Jajodia, Aaditya Gururaj, Aaron Myers, Aaron T. Myers, Aaron Wiebe, Aasha Medhi, and Aasha Medhi, with their respective email addresses, identity providers, and workload user names.

4. IDBroker

IDBroker allows an authenticated and authorized user to exchange a set of credentials or a token for cloud vendor access tokens. You can also view and update the IDBroker mappings at this location. IDBroker mappings can be accessed through Environments > Manage Access. Click on the IDBroker Mappings tab. Click Edit to edit or add

mappings. When finished, sync the mappings to push the settings from CDP to the IDBroker instance running inside the Data Lake of the environment.



At this point, CDP resources can access the AWS S3 buckets or Azure ADLS storage.

5. Ranger

To get admin access to Ranger, users need the EnvironmentAdmin role, and that role must be synced with the environment.

- Click Environments > Env > Actions > Manage Access > Add User
- Select EnvironmentAdmin resource role.
- Click Update Roles
- On the Environments page for the environment, in Actions, select Synchronize Users to FreeIPA.

The permissions are now synchronized to the Data Lake, and you have admin access to Ranger.

Update permissions in Ranger

- In Environments > Env > Data Lake Cluster, click Ranger.
- Select the Hadoop SQL service, and check that the users and groups have sufficient permissions to access databases, tables, columns, and urls.

For example, a user can be part of these policies:

- all - database,table,column
- all - url

This completes all configuration needed for CML to communicate with the Data Lake.

6. CML User Setup

Now, CML is able to communicate with the Data Lake. There are two steps to get the user ready to work.

- In Environments > Environment name > Actions > Manage Access > Add user, the Admin selects MLUser resource role for the user.
- The User logs into the workspace in ML Workspaces > Workspace name, click Launch Workspace.

The user can now access the workspace.

Related Information

[AWS environments](#)

[Azure environments](#)

Example: Connect a Spark session to Hive Metastore in a Data Lake

After the Admin sets up the correct permissions, you can access the Data Lake from your project, as this example shows.

Before you begin

Make sure you have access to a Data Lake containing your data.

Procedure

1. Create a project in your ML Workspace.
2. Create a file named `spark-defaults.conf`, or update the existing file with the property:
 - For S3: `spark.yarn.access.hadoopFileSystems=s3a://STORAGE LOCATION OF ENV>`
 - For ADLS: `spark.yarn.access.hadoopFileSystems=abfs://STORAGE CONTAINER OF ENV>@STORAGE ACCOUNT OF ENV>`

Use the same location you defined in Data Access.

3. Start a session (Python or Spark) and start a Spark session.

Results

Setting up the project looks like this:

CML / SDX Interaction  Running

By Vamsee Yarlagaadda — Python 3 Session — 1 vCPU / 2 GiB Memory — 6 minutes ago

Session [Logs](#) [Spark UI](#)

```
> from pyspark.sql import SparkSession
> spark = SparkSession\
    .builder\
    .appName("PythonPi")\
    .getOrCreate()

SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/etc/spark/jars/slf4j-log4j12-1.7.16.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/hadoop/lib/slf4j-log4j12.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/hive/lib/log4j-slf4j-impl-2.10.0.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Setting spark.hadoop.yarn.resourcemanager.principal to csso_vamsee

> spark.sql("show databases").show()

Hive Session ID = 160ecf56-971b-4ebf-9a62-6b37ebc7192d
+-----+
|      databaseName|
+-----+
|      default|
| information_schema|
|      sys|
| test_alpha1|
|tpcds_bin_partiti...|
|      vamsee|
+-----+
```

Now you can run Spark SQL commands. For example, you can:

- Create a database foodb.

CML / SDX Interaction

Running

By [Vamsee Yarlagadda](#) — Python 3 Session — 1 vCPU / 2 GiB Memory — 6 minutes ago

[Session](#)
[Logs](#)
[Spark UI](#)

```
> spark.sql("create database foodb").show()
```

++

||

++

++

- List databases and tables.

CML / SDX Interaction

Running

By [Vamsee Yarlagadda](#) — Python 3 Session — 1 vCPU / 2 GiB Memory — 6 minutes ago

[Session](#)
[Logs](#)
[Spark UI](#)

```
> spark.sql("use foodb").show()
```

++

||

++

++

```
> spark.sql("show tables").show()
```

```
+-----+-----+-----+
```

```
|database|tableName|isTemporary|
```

```
+-----+-----+-----+
```

```
+-----+-----+-----+
```

- Create a table `bartable`.

CML / SDX Interaction

Running

By [Vamsee Yarlagadda](#) — Python 3 Session — 1 vCPU / 2 GiB Memory — 6 minutes ago

[Session](#)[Logs](#)[Spark UI](#)

```
> spark.sql("create table bartable(id int)").show()
```

```
++  
||  
++  
++
```

```
> spark.sql("show tables").show()
```

```
+-----+-----+-----+  
|database|tableName|isTemporary|  
+-----+-----+-----+  
|  foodb| bartable|      false|  
+-----+-----+-----+
```

- Insert data into the table.

CML / SDX Interaction

Running

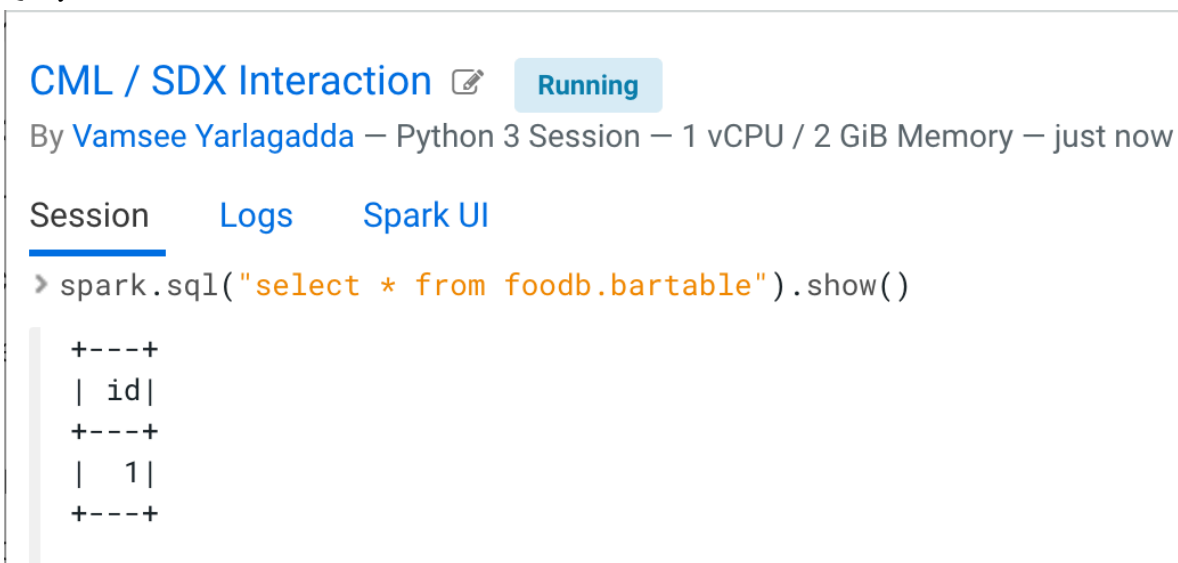
By [Vamsee Yarlagadda](#) — Python 3 Session — 1 vCPU / 2 GiB Memory — just now

[Session](#)[Logs](#)[Spark UI](#)

```
> spark.sql("insert into bartable values(1)").show()
```

```
++  
||  
++  
++
```


- Query the data from the table.



The screenshot shows a CML / SDX Interaction session titled "CML / SDX Interaction" with a "Running" status. It was created by "Vamsee Yarlagadda" and is a "Python 3 Session" with "1 vCPU / 2 GiB Memory". The session has tabs for "Session", "Logs", and "Spark UI". The command entered is `spark.sql("select * from foodb.bartable").show()`. The output is a table with one row and one column named "id", containing the value "1".

```

CML / SDX Interaction  Running
By Vamsee Yarlagadda — Python 3 Session — 1 vCPU / 2 GiB Memory — just now

Session  Logs  Spark UI
> spark.sql("select * from foodb.bartable").show()

+----+
| id |
+----+
|  1 |
+----+

```

Accessing CDW from CML

If you want to access data stored in a Cloudera Data Warehouse cluster from a CML workspace, you need to set up a connection. The CML and CDW instances must be within the same environment.

You need to add the following connection code to your project to establish the connection, and set the Spark properties described below. The properties can be set in the `spark=defaults.conf` file in the project, or in the Spark session itself.



Note: This connection requires engine image version 11-cml-2020.04-1 or higher.

Properties to set

Set the following properties in the following code sample to enable the connection.

- `spark.security.credentials.hiveserver2.enabled` : FALSE
- `spark.datasource.hive.warehouse.read.via.llap` : FALSE
- `spark.datasource.hive.warehouse.read.jdbc.mode` : client
- `spark.sql.hive.hiveserver2.jdbc.url` : `<JDBC_URL>;user=<username>;password=<password>`

Where:

- `JDBC_URL` : JDBC URL fetched from the CDW virtual warehouse user interface.
- `Username` : Username of the user.
- `Password` : Password of the user.

Connection code

Enter this code in your project file, and run it in a session.

```

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

spark = SparkSession \
    .builder \
    .appName("Pyspark Test") \

```

```

.config("spark.security.credentials.hiveserver2.enabled", "false")\
.config("spark.datasource.hive.warehouse.read.via.llap", "false")\
.config("spark.datasource.hive.warehouse.read.jdbc.mode", "client")\
.config("spark.sql.hive.hiveserver2.jdbc.url",
"<JDBC_URL>;user=<Username>;password=<Password>")\
.getOrCreate()

hive = HiveWarehouseSession.session(spark).build()
hive.showDatabases().show()

```

Enable the connection

Follow this procedure to enable this connection.

1. In CML workspace, create a Project.
2. In the Project, create a Python script and add the connection code.
3. In CDW, select Option menu > Copy JDBC String .
4. Paste the JDBC string into the following code. Append the user and password.



Note: The use of environmental variables is recommended for storing the user and password values.

Test the connection

Run the connection code in your session. You can then test the connection with the following commands:

- Show the available databases: `hive.showDatabases().show()`
- Set a database to use in the session: `hive.setDatabase("default")`
- List the tables in a specific database: `hive.showTables().show()`
- Run a SQL query: `hive.sql("select * from <table-name>").show()`

Accessing Ozone from Spark

In CML, you can connect Spark to the Ozone object store with a script. The following example demonstrates how to do this.

This 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: `o3fs://hivetest.s3v.o3service1/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 o3fs://hivetest.s3v.o3service1/spark"
// !
// Set the following spark setting in the file "spark-defaults.conf" on the
// CML session using terminal
// spark.yarn.access.hadoopFileSystems=o3fs://hivetest.s3v.o3service1.neptune
// e01.olympus.cloudera.com:9862

// 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("o3fs://hivetest.s3v.o3service1/spark/jedi_wisdom.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(","))
```