

Accessing Data from CML

Date published: 2020-07-16

Date modified: 2022-04-11



Legal Notice

© Cloudera Inc. 2024. All rights reserved.

The documentation is and contains Cloudera proprietary information protected by copyright and other intellectual property rights. No license under copyright or any other intellectual property right is granted herein.

Unless otherwise noted, scripts and sample code are licensed under the Apache License, Version 2.0.

Copyright information for Cloudera software may be found within the documentation accompanying each component in a particular release.

Cloudera software includes software from various open source or other third party projects, and may be released under the Apache Software License 2.0 (“ASLv2”), the Affero General Public License version 3 (AGPLv3), or other license terms. Other software included may be released under the terms of alternative open source licenses. Please review the license and notice files accompanying the software for additional licensing information.

Please visit the Cloudera software product page for more information on Cloudera software. For more information on Cloudera support services, please visit either the Support or Sales page. Feel free to contact us directly to discuss your specific needs.

Cloudera reserves the right to change any products at any time, and without notice. Cloudera assumes no responsibility nor liability arising from the use of products, except as expressly agreed to in writing by Cloudera.

Cloudera, Cloudera Altus, HUE, Impala, Cloudera Impala, and other Cloudera marks are registered or unregistered trademarks in the United States and other countries. All other trademarks are the property of their respective owners.

Disclaimer: EXCEPT AS EXPRESSLY PROVIDED IN A WRITTEN AGREEMENT WITH CLOUDERA, CLOUDERA DOES NOT MAKE NOR GIVE ANY REPRESENTATION, WARRANTY, NOR COVENANT OF ANY KIND, WHETHER EXPRESS OR IMPLIED, IN CONNECTION WITH CLOUDERA TECHNOLOGY OR RELATED SUPPORT PROVIDED IN CONNECTION THEREWITH. CLOUDERA DOES NOT WARRANT THAT CLOUDERA PRODUCTS NOR SOFTWARE WILL OPERATE UNINTERRUPTED NOR THAT IT WILL BE FREE FROM DEFECTS NOR ERRORS, THAT IT WILL PROTECT YOUR DATA FROM LOSS, CORRUPTION NOR UNAVAILABILITY, NOR THAT IT WILL MEET ALL OF CUSTOMER’S BUSINESS REQUIREMENTS. WITHOUT LIMITING THE FOREGOING, AND TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, CLOUDERA EXPRESSLY DISCLAIMS ANY AND ALL IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, QUALITY, NON-INFRINGEMENT, TITLE, AND FITNESS FOR A PARTICULAR PURPOSE AND ANY REPRESENTATION, WARRANTY, OR COVENANT BASED ON COURSE OF DEALING OR USAGE IN TRADE.

Contents

Accessing data with Spark.....	4
Use JDBC Connection with PySpark.....	4
Use JDBC Connection with SparklyR.....	5
Use Direct Reader Mode with PySpark.....	6
Use Direct Reader Mode with SparklyR.....	6
 Accessing Local Data from Your Computer.....	 8
 Accessing Data from HDFS.....	 8
 Accessing Data from Apache Hive.....	 10
 Accessing Data from Apache Impala.....	 10
Loading CSV Data into an Impala Table.....	10
Running Queries on Impala Tables.....	11
 Accessing Data in Amazon S3 Buckets.....	 13
 Accessing External SQL Databases.....	 14
 Accessing CDW from CML.....	 15
 Accessing Ozone from Spark.....	 16

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", "session", 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.HiveAcidAutoConvertExtension")\
.config("spark.kryo.registrator", "com.qubole.spark.hiveacid.util.HiveAcidKryoRegistrator")\
.config("spark.yarn.access.hadoopFileSystems", "s3a://demo-aws-2/")\
.config("spark.jars", "/usr/lib/hive_warehouse_connector/hive-warehouse-connector-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$spark.sql.extensions="com.qubole.spark.hiveacid.HiveAcidAutoConvertExtension"
config$spark.kryo.registrator="com.qubole.spark.hiveacid.util.HiveAcidKryoRegistrator"

# Pick the correct jar location by using Terminal Access
# to file the exact file path/name of the hwc jar under /usr/lib/hive_warehouse_connector
config$sparklyr.jars.default <- "/usr/lib/hive_warehouse_connector/hive-warehouse-connector-assembly-1.0.0.7.2.2.0-244.jar"
# File system read access - this s3a patha is available from the environment Cloud Storage.
# To read from this location go to Environment->Manage Access->IdBroker Mappings
# ensure that the user (or group) has been assigned an AWS IAM role that can
# read this location.
#config$spark.yarn.access.hadoopFileSystems="s3a://demo-aws-2/"
config$spark.yarn.access.hadoopFileSystems="s3a://demo-aws-2/datalake/warehouse/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 Catalog of CDW
#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=priyankp;password=!Password1"

#ss <- spark_session(sc)
#hive <- invoke_static(sc,"com.hortonworks.hwc.HiveWarehouseSession","session",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 work with existing data files (.csv, .txt, etc.) from your computer, you can upload these files directly to your project in the CML workspace. 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)
```

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',
               use_ssl='true')
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.en
viron["POSTGRESQL_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="cd
swuser", password=Sys.getenv("POSTGRESQL_PASSWORD"))
flights_table <- tbl(db, "flights")
select(flights_table, year:day, dep_delay, arr_delay)
```

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(", "))
```