

Accessing Data from CML

Date published: 2020-07-16

Date modified: 2023-10-31



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

Data Access.....	4
Upload and work with local files.....	4
Create an Iceberg data connection.....	5
Create an Ozone data connection.....	5
Connect to CDW.....	5
Accessing data with Spark.....	6
Use JDBC Connection with PySpark.....	7
Connect to external Amazon S3 buckets.....	8
Connect to External SQL Databases.....	9
Accessing Ozone.....	10
Accessing Ozone from Spark.....	10
Accessing local files in Ozone.....	10

Data Access

Cloudera Machine Learning is a flexible, open platform supporting connections to many data sources.

CML supports easy, secure data access through connection snippets and the `cml.data` library. This library, implemented in Python, abstracts all of the complexity of configuring, initializing, and authenticating data connections. Users choosing to manually create and configure the data connections can follow the reference guide below.

Upload and work with local files

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)
```

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))
```

Create an Iceberg data connection

CML supports data connections to Iceberg data lakes.

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

Make sure to set the correct DATALAKE_DIRECTORY environmental variable.

```
spark = (
    SparkSession
    .builder
    .appName("Iceberg Spark")
    .config("spark.jars", "/opt/spark/optional-lib/iceberg-spark-runtime.jar
/opt/spark/optional-lib/iceberg-hive-runtime.jar")
    .config("spark.sql.extensions", "org.apache.iceberg.spark.extensions.Ice
bergSparkSessionExtensions")
    .config("spark.sql.catalog.spark_catalog.type", "hive")
    .config("spark.sql.catalog.spark_catalog", "org.apache.iceberg.spark.Sp
arkSessionCatalog")
    .config("spark.hadoop.iceberg.engine.hive.enabled", "true")
    .config("spark.yarn.access.hadoopFileSystems", <DATALAKE_DIRECTORY>)
    .getOrCreate()
)
```

Create an Ozone data connection

CML supports data connections to Ozone file systems.

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

Make sure to 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-file-system-hadoop3.jar")
    .config("spark.yarn.access.hadoopFileSystems", DATALAKE_DIRECTORY)
    .getOrCreate()
)

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

Connect to CDW

The Data Connection Snippet feature now suggests using the `cml.data` library to connect to CDW virtual warehouses - these code snippets pop up as suggestions for every new session in a project. For further information, see *Using data connection snippets*.

However, if you would still like to use raw Python code to connect, follow the below details.

You can access data stored in the data lake using a Cloudera Data Warehouse cluster from a CML workspace, using the `impyla` Python package.

Configuring the connection

The CDW connection requires a `WORKLOAD_PASSWORD` that can be configured following the steps described in *Setting the workload password*, linked below.

The `VIRTUAL_WAREHOUSE_HOSTNAME` can be extracted from the JDBC URL that can be found in CDW, by selecting the `Option menu > Copy JDBC String` on a Virtual Warehouse.

For example, if the JDBC string copied as described above is:

```
jdbc:impala//<your-vw-host-name.site>/default;transportMode=http;httpPath=cliservice;socketTimeout=60;ssl=true;auth=browser;
```

Then, the extracted hostname to assign to the `VWH_HOST` is: `<your-vw-host-name.site>`

Connection code

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

```
# This code assumes the impyla package to be installed.
# If not, please pip install impyla

from impala.dbapi import connect
import os
USERNAME=os.getenv('HADOOP_USER_NAME')
PASSWORD=os.getenv('WORKLOAD_PASSWORD')
VWH_HOST = "<<VIRTUAL_WAREHOUSE_HOSTNAME>>"
VWH_PORT = 443
conn = connect(host=VWH_HOST, port=VWH_PORT, auth_mechanism="LDAP", user=USERNAME, password=PASSWORD, use_http_transport=True, http_path="cliservice", use_ssl=True)

dbcursor = conn.cursor()
dbcursor.execute("<<INSERT SQL QUERY HERE>>")
for row in dbcursor:
    print(row)

#Sample pandas code
#from impala.util import as_pandas
#import pandas
#dbcursor = conn.cursor()
#dbcursor.execute("<<INSERT SQL QUERY HERE>>")
#tables = as_pandas(dbcursor)
#tables
#dbcursor.close()
```

Accessing data with Spark

When you are using CDW, you can use JDBC connections.

JDBC is useful in the following cases:

1. Use JDBC connections when you have fine-grained access.
2. 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.

How to obtain the Data Lake directory location

You need this location if you are using a Direct Reader connection.

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 for Hive Metastore Warehouse.
6. For external unmanaged data tables, copy the location shown for Hive Metastore External Warehouse.
7. Paste the location into the connection script in the designated position. 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.

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

PySpark can be used with JDBC connections, but it is not recommended. The recommended approach is to use Impyla for JDBC connections. For more information, see *Connect to CDW*.

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

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/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()
```

Related Information

[Connect to CDW](#)

Connect to external Amazon S3 buckets

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

To work with external S3 buckets in Python, do the following:

- 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 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=5432;DATABASE=test_db;USER=cdswuser;OPTION=3;PASSWORD=%s" % os.environ["POSTGRES_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

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("POSTGRES_PASSWORD"))
flights_table <- tbl(db, "flights")
select(flights_table, year:day, dep_delay, arr_delay)
```

Accessing Ozone

In Cloudera Machine Learning, you can connect CML to the Ozone object store using a script or command line commands. The following two articles show how to access Ozone.

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: 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
// CML 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 CML 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://omservicel/s3v/hivetest/
```

3. Download file from ozone to local:

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