Machine Learning

Accessing Data from CML

Date published: 2020-07-16
Date modified: 2024-05-30

https://docs.cloudera.com/

Legal Notice

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

Uploading and working with local files....................4

Connecting to CDW....................4

Connecting to Hive and Impala services on CDP Private Cloud Base....................6

Accessing data with Spark....................7

Using JDBC Connection with PySpark....................8
Connecting to Iceberg tables....................8
Connecting to Hive tables via HWC....................9
Connecting to Ozone filesystem....................9

Accessing Ozone storage....................10

Creating an Ozone data connection....................10
Connecting to Ozone filesystem....................10
Accessing local files in Ozone....................11
Connecting to external Amazon S3 buckets....................11

Connect to External SQL Databases....................12

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 CML workspace. The presented code samples demonstrate how to access local data for CML workloads.

To upload the files:

- Go to the project's Overview page.
- Click Upload under the Files section.
- Select the relevant data files to be uploaded.

These files will be uploaded to an NFS share available to each project.

Note: Do not store large data files in your Project folder. Store your data files in the Data Lake.

The tips.csv dataset examples 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))
```

Connecting to CDW

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

In order to get the CDW 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 CDW 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 CDW 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
)
```

Connecting to Hive and Impala services on CDP Private Cloud Base

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

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",
        )
```

Accessing data with Spark

When you are using Cloudera Data Warehouse (CDW), 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.

Obtaining the Data Lake directory location

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

1. Select Management Console in the CDP home page.
2. Select the environment you are using in Environments.
3. Select Cloud Storage in the tabbed section.
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.

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 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. 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 CDW

Connecting to Iceberg tables

CML 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("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()
  )
```

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 Machine Learning (CML), 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
 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 Ozone storage

In Cloudera Machine Learning, (CML) you can connect CML to the Ozone object store using a script or command line commands.

Creating an Ozone data connection

Cloudera Machine Learning (CML) 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 Machine Learning (CML), 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
 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 Cloudera Machine Learning (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://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

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=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

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)
```