

Importing Data into Cloudera Data Science Workbench

Date published: 2020-02-28

Date modified: 2020-12-15



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

Importing Data into Cloudera Data Science Workbench.....	4
Accessing Local Data from Your Computer.....	4
Accessing Data from HDFS.....	4
Accessing Data from Apache HBase.....	6
Load Data into HBase Table.....	6
Query Data Using HappyBase.....	6
Accessing Data from Apache Hive.....	7
Accessing Data from Apache Impala.....	7
Loading CSV Data into an Impala Table.....	7
Running Queries on Impala Tables.....	8
Accessing Data in Amazon S3 Buckets.....	11
Accessing External SQL Databases.....	11
R.....	12
Python.....	12

Importing Data into Cloudera Data Science Workbench

Cloudera Data Science Workbench allows you to run analytics workloads on data imported from local files, Apache HBase, Apache Kudu, Apache Impala, Apache Hive or other external data stores such as Amazon S3.

Accessing Local Data from Your Computer

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 Data Science Workbench project.

Procedure

1. Navigate to the project's Overview page.
2. Under the Files section, click Upload and select the relevant data files to be uploaded.
3. Upload the appropriate [tips.csv](#) dataset to the data folder in your project before you run these examples.

The following sections use the [tips.csv](#) dataset to demonstrate how to work with local data stored within your project.

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.

About this task

The following code samples demonstrate how to count the number of occurrences of each word in a simple text file in HDFS.

Procedure

1. Navigate to your project and click Open Workbench.
2. Create a file called sample_text_file.txt and save it to your project in the data folder.
3. Write this file to HDFS.

You can do this in one of the following ways:

Click Terminal above the Cloudera Data Science Workbench 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 HBase

This section demonstrates how to use the HappyBase Python library to access data from HBase.

Load Data into HBase Table

This section demonstrates how to use the HappyBase Python library to access data from HBase.

About this task

For this example, we're going to import data from a CSV file into HBase using the importTsv package.

Procedure

1. Log into Cloudera Data Science Workbench and launch a Python 3 session within a new/existing project.

For this example, we will be using the following sample CSV file.

2. Create the following employees.csv file in your project.

employees.csv

```
1,Lucy,Engineering
2,Milton,Engineering
3,Edith,Support
```

3. In the workbench, click Terminal access. Perform the following steps in the Terminal:

- a) Start the HBase shell and create a new blank table called employees.

```
hbase shell
create 'employees', 'name', 'department'
exit
```

- b) Load employees.csv into HDFS.

```
hdfs dfs -put employees.csv /tmp
```

- c) Use [ImportTsv](#) to load data from HDFS (/tmp/employees.csv) into the HBase table created in the previous step.

```
hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -Dimporttsv.separator=
',' -Dimporttsv.columns=HBASE_ROW_KEY,name,department employees /tmp/emp
loyees.csv
```

- d) Go back to the HBase shell and run the following command to make sure data was loaded into the HBase table.

```
hbase shell
scan 'employees'
```

Query Data Using HappyBase

This section demonstrates how to use the HappyBase Python library to access data from HBase.

Procedure

1. Launch a Python 3 session and use the workbench command prompt to install the happybase package.

```
!pip3 install happybase
```

2. Use happybase to connect to the employees table created in the previous step.

Python

```
import happybase
connection = happybase.Connection(host='<hbase_thrift_server_hostname>', port=9090, autoconnect=True)
table = connection.table('employees')
rows = table.rows(['1', '2', '3'])
for key, data in rows:
    print(key, data)
```

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
!pip3 install impyla
!pip3 install thrift_sasl

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

# Specify HIVE_HS2_HOST host name as an environment variable in your project settings
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.

About this task

Use the following steps to save this file to a project in Cloudera Data Science Workbench, and then load it into a table in Apache Impala.

Procedure

1. Create a new Cloudera Data Science workbench project.
2. Create a folder called data and upload tips.csv to this folder.
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 Data Science Workbench 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/";
```

- a) 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)

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
```

Python 3

The specific library versions shown in this example are needed for Impyla to work correctly with Python 3.

```
# Install Libraries
!pip3 install impyla==0.13.8
!pip3 install thrift_sasl==0.2.1
!pip3 install thrift==0.9.3
!pip3 install sasl==0.2.1
# Connect to Impala
from impala.dbapi import connect

conn = connect(host='host',
               port=21050,
               auth_mechanism='GSSAPI',
               use_ssl=True,
               kerberos_service_name='impala')

# Execute Query
sql = "select * from table"
```

```

cursor = conn.cursor()
cursor.execute(sql)
results = cursor.fetchall()

```

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 Data Science Workbench has libraries available for uploading to and downloading from Amazon S3.

About this task

To work with S3:

Procedure

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/cdsd/faithful.csv')

```

Accessing External SQL Databases

Every language in Cloudera Data Science Workbench 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.

R

The examples below show how to retrieve the password from an environment variable and use it to connect.

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

Python

The examples below show how to retrieve the password from an environment variable and use it to connect.

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.getenv("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("POSTGRES_PASSWORD"))
session = sessionmaker(bind=db)
user = session.query(User).filter_by(name='ed').first()
```