

Accessing Data from Cloudera AI

Date published: 2020-07-16

Date modified: 2025-05-29

CLOUDERA

Legal Notice

© Cloudera Inc. 2025. 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**
- Connect to Cloudera Data Lake.....5**
 - Setup Data Lake Access.....5
 - Example: Connect a Spark session to the Data Lake.....9
 - Use Direct Reader Mode with PySpark.....12
 - Use Direct Reader Mode with SparklyR.....13
 - Create an Iceberg data connection.....15
 - Accessing Data from HDFS.....16
- Connecting to Cloudera Data Warehouse..... 17**
 - Accessing data with Spark..... 18
 - Use JDBC Connection with PySpark.....18
- Connect to a Cloudera Data Hub cluster.....19**
- Connecting to external Amazon S3 buckets..... 20**
- Connect to External SQL Databases.....21**
- Accessing Ozone storage..... 21**
 - Connecting to Ozone filesystem..... 21
 - Accessing local files in Ozone.....22

Data Access

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

Cloudera AI 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 shall follow the relevant instructions.

Related Information

[Using data connection snippets](#)

Upload and work with local files

This topic includes code samples that demonstrate how to access local data for Cloudera AI Workbench 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 Cloudera AI Workbench. 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))
```

Connect to Cloudera Data Lake

Cloudera AI has secure access to the data stored in the Cloudera Data Lake.

The Data Connection Snippet feature now suggests using the `cml.data` library to connect to the Cloudera Data Lake - these code snippets pop up as suggestions for every new session in a project. For more information, see *Data Exploration and Visualization*.

However, if you would still like to use raw code to connect, follow the instructions in the sections below.

Related Information

[Data Exploration and Visualization](#)

Setup Data Lake Access

Cloudera AI can access data tables stored in an AWS or Microsoft Azure Data Lake. As a Cloudera AI 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 Cloudera 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 Cloudera, 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:

The screenshot shows the 'Logs Storage and Audits' configuration page in the Cloudera Environment Setup wizard. It includes three main sections: 'Instance Profile', 'Logs Location Base', and 'Ranger Audit Role'. Each section has a text input field and a help icon (question mark in a circle).

- Instance Profile:** The dropdown menu shows 'MLX_DEV_DATA LAKE_LOG_ROLE'. A link 'Click here to refresh instance profiles from the cloud provider.' is provided.
- Logs Location Base:** The text input field shows 's3a://fooenv/logs'.
- Ranger Audit Role:** The text input field shows 'arn:aws:iam::886883559913:role/mlx-dev-prod-env_RANGER_AUD'.

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 Cloudera 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 Cloudera 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.

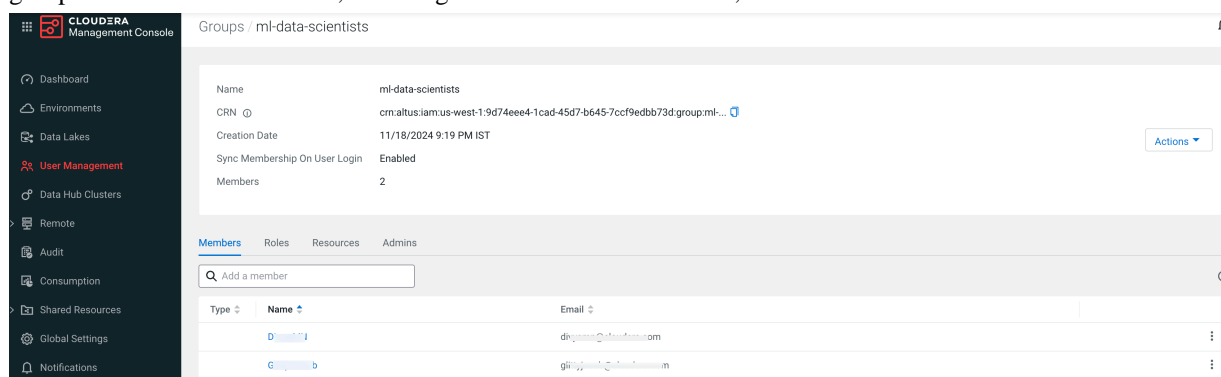


Note: There is no limit to the number of mappings that one can define but each user can only be assigned to one of the role mappings.

This completes installation of the environment.

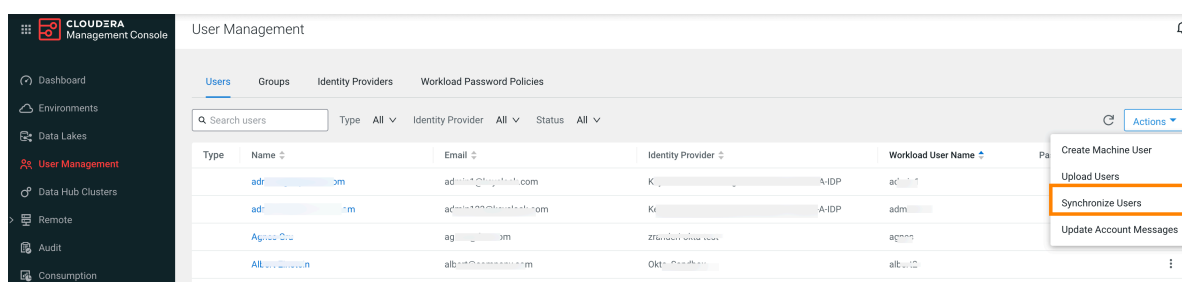
3. User Group Mappings

In Cloudera, 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. .



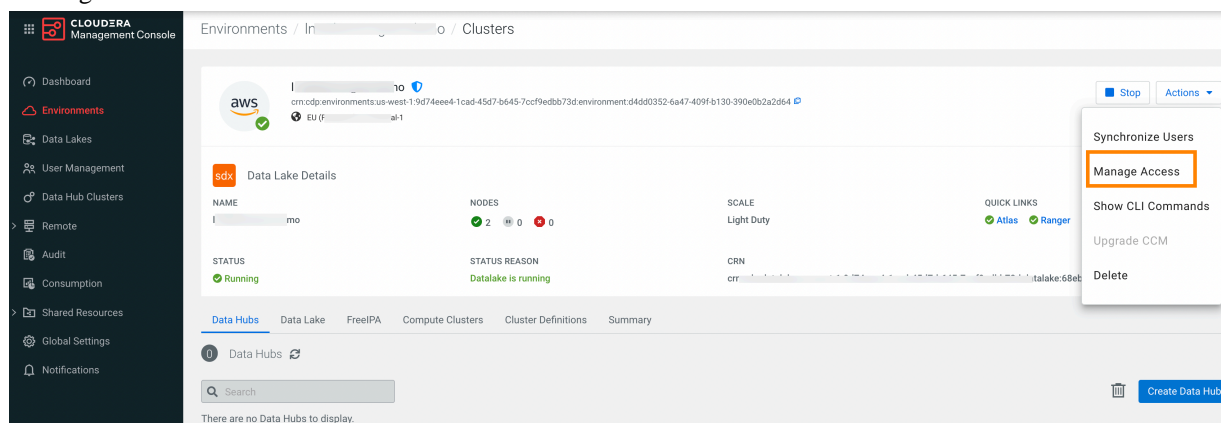
a. Sync users

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.



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, synchronize the mappings to push the settings from Cloudera to the IDBroker instance running inside the Data Lake of the environment.



At this point, Cloudera 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.

- a. Click Environments > Env > Actions > Manage Access > Add User
- b. Select EnvironmentAdmin resource role.
- c. Click Update Roles
- d. 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

- a. In Environments > Env > Data Lake Cluster, click Ranger.
- b. 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 Cloudera AI to communicate with the Data Lake.

6. Cloudera AI User Setup

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

- a. In Environments > Environment name > Actions > Manage Access > Add user, the Admin selects MLUser resource role for the user.
- b. The User logs into the workbench in Cloudera AI Workbench > Workbench name, click Launch Workbench.

The user can now access the workbench.

Related Information

[AWS environments](#)

[Azure environments](#)

Example: Connect a Spark session to the Data Lake

After the Administrator 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.

The s3 or abfs path can be retrieved from the environment's cloud storage. To read from this location, in the Cloudera AI Workbenches UI, select the name of the environment, and in the Action menu, select Manage Access IdBroker Mappings . Ensure that the user (or group) has been assigned an AWS IAM role that can read this location. For Fine Grained Access Control (Raz) enabled environments, go to Ranger to ensure that the user (or group) has access through an appropriate s3 or adls policy.

Procedure

1. Create a project in your Cloudera AI Workbench.

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 *Setup Data Lake 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 Yarlagadda — 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
+-----+
|      dbName      |
+-----+
|      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 Yarlaga](#) — 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 Yarlaga](#) — 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.

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|
+---+
```

Related Information

[Setup Data Lake Access](#)

Use Direct Reader Mode with PySpark

You can use the Direct Reader Mode when your data has table-level access control, and does not have row-level security or column level masking (fine grained access.)

Before you begin

Obtain the location of the data lake before starting this task.

Continue with the next steps to set up the connection.

Procedure

Replace `s3a://demo-aws-2/` in the code sample below with the correct S3 bucket location. This sets the value for `DATALAKE_DIRECTORY`.

The `DATALAKE_DIRECTORY` value is set for the `spark.yarn.access.hadoopFileSystems` property in the corresponding config statement.

Example



Note: The following example works for managed as well as external Hive tables, however, only for Spark 2.

```
from pyspark.sql import SparkSession
# Change to the appropriate Datalake directory location
DATALAKE_DIRECTORY = "s3a://demo-aws-2/"

spark = SparkSession\
.builder\
.appName("CDW-CML-Spark-Direct")\
.config("spark.sql.hive.hwc.execution.mode", "spark")\
.config("spark.sql.extensions", "com.qubole.spark.hiveacid.HiveAcidAutoCon\
vertExtension")\
.config("spark.kryo.registrator", "com.qubole.spark.hiveacid.util.HiveAcidKry\
roRegistrator")\
.config("spark.yarn.access.hadoopFileSystems", DATALAKE_DIRECTORY)\
.config("spark.jars", "/opt/spark/optional-lib/hive-warehouse-connector-ass\
embly.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()
```



Note: The following example works for both Spark 2 and Spark 3, but only works for external Hive tables, and does not support Hive managed tables.

```
from pyspark.sql import SparkSession
# Change to the appropriate Datalake directory location
DATALAKE_DIRECTORY = "s3a://demo-aws-2/"
spark = SparkSession\
.builder\
.appName("CDW-CML-Spark-Direct")\
.config("spark.sql.hive.hwc.execution.mode", "spark")\
.config("spark.yarn.access.hadoopFileSystems", DATALAKE_DIRECTORY)\
.getOrCreate()

spark.sql("show databases").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$sparklyr.jars.default <- "/opt/spark/optional-lib/hive-warehouse-connector-assembly.jar"
# File system read access - this s3a patha is available from the environment Cloud Storage.
# To read from this location go to Environment->Manage Access->IdBroker Mappings
# ensure that the user (or group) has been assigned an AWS IAM role that can
# read this location.
# For Fine Grained Access Control (Raz) enabled environments, go to Ranger to ensure
# that the user (or group) has access through an appropriate s3 or adls policy
config$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)
```

Create an Iceberg data connection

Cloudera AI supports data connections to Iceberg data lakes.

You can connect automatically, using the Cloudera AI-provided data connection. If necessary, you can also set up a manual connection using the snippet provided below. To connect with Iceberg, you must use Spark 3.

In your project:

1. In Project Settings, view the Data Connections tab. There should be an available data connection to a Spark Data Lake.
2. Start a New Session.
3. Select Enable Spark, and choose Spark 3 from the dropdown.
4. Select Start Session.
5. In the Connection Code Snippet UI, select the Spark Data Lake connection.
6. In the code window, select Copy Code, then Close.
7. Select FileNew File , and paste the Cloudera AI-provided code snippet into the file.
8. Select Run.

You see a list of available databases in the data lake.

Instead of using the Cloudera AI-provided data connection, you can also manually connect to a Spark Data Lake using the Spark command as shown in the snippet below.

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.sql.hive.hwc.execution.mode", "spark")
        .config("spark.sql.extensions", "com.quobole.spark.hiveacid.HiveAcidAutoConvertExtension,org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions")
        .config("spark.sql.catalog.spark_catalog.type", "hive")
        .config("spark.sql.catalog.spark_catalog", "org.apache.iceberg.spark.SparkCatalog")
        .config("spark.kerberos.access.hadoopFileSystems", "hdfs://nn1.com:8032,hdfs://nn2.com:8032,webhdfs://nn3.com:50070")
        .config("spark.hadoop.iceberg.engine.hive.enabled", "true")
        .config("spark.executorEnv.HADOOP_CONF_DIR", "/home/cdsw/hadoop_config_dir")
        .config("spark.sql.iceberg.handle-timestamp-without-timezone", "true")
        .config("spark.jars", "/opt/spark/optional-llb/iceberg-spark-runtime.jar,/opt/spark/optional-llb/iceberg-hive-runtime.jar")
        .getOrCreate()
)
spark.sql("show databases").show()
spark.sql("describe formatted <database_name>.<table_name>").show()
```

Accessing Data from HDFS

There are many ways to access HDFS data from R, Python, and Scala libraries. The following code samples assume that appropriate permissions have been set up in IDBroker or Ranger/Raz. The samples below demonstrate how to count the number of occurrences of each word in a simple text file in HDFS.

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 AI console and enter the following command to write the file to HDFS:

```
hdfs dfs -put data/sample_text_file.txt s3a://<s3_data_directory>/tmp
```

OR

- Use the workbench command prompt:

Python Session

```
!hdfs dfs -put data/sample_text_file.txt s3a://<s3_data_directory>/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("s3a://<s3_data_directory>/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("s3a://<s3_data_directory>/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(", "))
```

Connecting to Cloudera Data Warehouse

The Data Connection Snippet feature now suggests using the `cml.data` library to connect to Cloudera Data Warehouse 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 Cloudera AI Workbench, using the `impyla` Python package.

Configuring the connection

The Cloudera Data Warehouse 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 Cloudera Data Warehouse, 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(cursor)
#tables
#dbcursor.close()
```

Related Information

[Setting the workload password](#)

[Using data connection snippets](#)

Accessing data with Spark

When you are using Cloudera Data Warehouse, 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. 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. For more information, see *Setup Data Lake Access*.

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 Cloudera Data Warehouse, 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.

Related Information

[Setup Data Lake Access](#)

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

[Connecting to Cloudera Data Warehouse](#)

Connect to a Cloudera Data Hub cluster

The Data Connection Snippet feature helps simplify the user experience by abstracting the complexity of creating and configuring a data connection.

You can set up a Data Hub Impala or Hive connection by following the documentation: *Set up a data connection to Cloudera Data Hub*.

However, if you would still like to use raw Python code to connect, follow this Python 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

USERNAME=os.getenv(HADOOP_USER_NAME)
PASSWORD=os.getenv(WORKLOAD_PASSWORD)

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 = USERNAME,
    password = PASSWORD)
cursor = conn.cursor()
cursor.execute("<<INSERT SQL QUERY HERE>>")

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

```

Related Information[Set up a data connection to Cloudera DataHub](#)

Connecting to external Amazon S3 buckets

Every language in Cloudera AI 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 AI 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 [SQLAlchemy](#):

```
!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
.env["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 storage

In Cloudera AI you can connect Cloudera AI to the Ozone object store using a script or command line commands.

Connecting to Ozone filesystem

In Cloudera AI, you can connect Spark to the Ozone object store with a script.

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 Cloudera AI 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 AI 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
```