

Integrating Data

Date published: 2023-12-31

Date modified: 2024-04-27

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

Configure data engineering Spark to use with Cloudera Operational Database.....	4
Configure Data Engineering Cloudera Data Hub to use with Cloudera Operational Database.....	6
Running Phoenix and HBase Spark Applications using Cloudera Data Engineering.....	7

Configure data engineering Spark to use with Cloudera Operational Database

You can configure Spark in your Data Engineering cluster to interact with the Cloudera Operational Database. You can use this integration to READ and WRITE to Cloudera Operational Database from Spark on Cloudera Data Engineering using the spark-hbase connector.

About this task

If you want to leverage Phoenix instead of HBase, see *Cloudera Operational Database-Cloudera Data Engineering using Phoenix*.

Before you begin

- Cloudera Operational Database is already provisioned and the database is created. For ore information, see *Onboarding Cloudera Operational Database users*.

For this example, let us assume *TEST-COD* as the database name.

- Cloudera Data Engineering is already provisioned and the virtual cluster is already created. For more information, see *Cloudera Data Engineering service*.

Procedure

1. Download the Cloudera Operational Database client configurations.

For the Spark in Cloudera Data Engineering to connect to Cloudera Operational Database, it requires the hbase-site.xml configuration of the Cloudera Operational Database cluster. Refer to the following steps.

- a) Go to the Cloudera Operational Database UI and click on the *TEST-COD* database.
- b) Go to the **Connect HBase** tab of the Cloudera Operational Database database, and copy the command under the HBase Client Configuration URL field.

```
curl -f -o "hbase-config.zip" -u "<YOUR WORKLOAD USERNAME>" "https://cod--4wfxojpfxmwg-gateway.XXXXXXXXXX.cloudera.site/clouderamanager/api/v41/clusters/cod--4wfxojpfxmwg/services/hbase/clientConfig"
```

- Ensure to provide the workload password for the curl command.
- Explore the downloaded zip file to obtain the hbase-site.xml file.

2. Create the HBase table.

Create a new HBase table inside the Cloudera Operational Database database using the Hue link.

- a) Go to the Cloudera Operational Database UI and click on the *TEST-COD* database.
- b) Click on the Hue link under the SQL EDITOR field.
- c) On the Hue UI, click the HBase menu item on the left navigation panel. Click New Table.
- d) Choose a table name and column families, and click on the Submit button. For example, let us consider the table name *TESTTABLE* and a single column family *TESTCF*.

3. Configure the job using Cloudera Data Engineering CLI.

- a) Configure Cloudera Data Engineering CLI to point to the virtual cluster created in the previous step. For more details, see *Configuring the CLI client*.
- b) Create resources using the following command.

```
cde resource create --name cod-spark-resource
```

- c) Upload hbase-site.xml.

```
cde resource upload --name cod-spark-resource --local-path /your/path/to/hbase-site.xml --resource-path conf/hbase-site.xml
```

- d) Upload the demo app jar that was built earlier.

```
cde resource upload --name cod-spark-resource --local-path /path/to/your/spark-hbase-project.jar --resource-path spark-hbase-project.jar
```

- e) Create the Cloudera Data Engineering job using a JSON definition.

```
{
  "mounts": [
    {
      "resourceName": "cod-spark-resource"
    }
  ],
  "name": "my-cod-spark-job",
  "spark": {
    "className": "<YOUR MAIN CLASS>",
    "conf": {
      "spark.executor.extraClassPath": "/app/mount/conf",
      "spark.driver.extraClassPath": "/app/mount/conf"
    },
    "args": [ "<YOUR ARGS IF ANY>" ],
    "driverCores": 1,
    "driverMemory": "1g",
    "executorCores": 1,
    "executorMemory": "1g",
    "file": "spark-hbase-project.jar",
    "pyFiles": [],
    "files": [ "conf/hbase-site.xml" ],
    "numExecutors": 4
  }
}
```

- f) Import the job using the following command, assuming that the above JSON is saved as *MY-JOB-DEFINITION.JSON*.

```
cde job import --file my-job-definition.json
```

What to do next

The `spark.driver.extraClassPath` and `spark.executor.extraClassPath` inside the job definition points to the same path which is used to upload the `hbase-site.xml` into the Cloudera Data Engineering resource.

This way the `hbase-site.xml` is automatically loaded from the classpath and you do not need to refer to it explicitly in your Spark code. You can define as follows.

```
val conf = HBaseConfiguration.create()
val hbaseContext = new HBaseContext(spark.sparkContext, conf)
```

Related Information[Cloudera Operational Database-Cloudera Data Engineering using Phoenix](#)[Onboarding Cloudera Operational Database users](#)[Cloudera Data Engineering service](#)[Configuring the CLI client](#)

Configure Data Engineering Cloudera Data Hub to use with Cloudera Operational Database

You can configure Hive in your Cloudera Data Engineering Cloudera Data Hub cluster to interact with the Cloudera Operational Database. You can use this integration to create and modify HBase tables using Hive. You can also READ and WRITE to existing HBase tables.

Before you begin

Ensure that you have a Cloudera Operational Database database instance and a Data Engineering template-based Cloudera Data Hub cluster in the same Cloudera environment.

Procedure

1. Create a Data Engineering Datahub cluster in the same Cloudera environment on which your Cloudera Operational Database is launched.
2. Download the HBase client configuration zip archive file from the Cloudera Operational Database client connectivity page. For more information see *Client connectivity information*. Then, extract hbase-site.xml from the zip file.
3. On each node in the Data Engineering Cloudera Data Hub cluster that runs HiveServer2, run the `mkdir /etc/hbase/cod-conf` command, and copy the hbase-site.xml file from Step 2 into /etc/hbase/cod-conf folder.
Do not copy all the files from client configuration zip archive. Only copy the hbase-site.xml file.
 - a) Get the value of `ssl.client.truststore.password` from the Data Engineering DataHub cluster master `ssl-client.xml` file (/etc/hadoop/conf.cloudera.hdfs/ssl-client.xml) and update the password for `hbase.zookeeper.property.ssl.trustStore.password` property in the copied hbase-site.xml file.
 - b) Create a symbolic link between /opt/cloudera/parcels/CDH/lib/hbase/conf and /etc/hbase/cod-conf in the Data Engineering Datahub cluster using the following command.


```
sudo ln -s /etc/hbase/cod-conf /opt/cloudera/parcels/CDH/lib/hbase/conf
```
4. In Cloudera Manager for the Cloudera Data Engineering Cloudera Data Hub cluster, in the Hive-on-Tez service, set the HiveServer2 Environment Advanced Configuration Snippet(hive_hs2_env_safety_valve): HADOOP_CLASSPATH to /etc/hbase/cod-conf.
 - a) Click Save Configurations.
 - b) Restart the Hive-on-Tez service.
5. Go to the Ranger service in the Data Lake.
 - a) Find HBase Ranger policy `cod_[***CLOUDERA OPERATIONAL DATABASE DATABASE NAME***]_hbase`. For example, if your Cloudera Operational Database database name is CODDB, you must modify the Ranger policy `cod_CODDB_hbase`.
 - b) Add the user hive to the existing policy. For example, all - table, column-family, column.
 - c) Wait for a few minutes because Ranger policy sync is not immediate to all resources.
6. Interact with Hive using the hive command from a node in the Data Engineering Cloudera Data Hub.

```
$ hive
hive> create table test(key int, value string) stored by 'org.apache.hadoop.hive.hbase.HBaseStorageHandler' with serdeproperties ("hbase.columns.mapping" = ":key,fl:val");
```

```
hive> insert into test values(1, 'a');
hive> select * from test;
```

7. Validate data is present in HBase as well. You can also run this from the Data Engineering Cloudera Data Hub node: the hbase executable is present to invoke. Otherwise, set up the HBase client tarball from Cloudera Operational Database client connectivity.

```
$ HBASE_CONF_DIR=/etc/hbase/cod-conf hbase shell
hbase> scan 'test'
```

Related Information

[Client connectivity information](#)

Running Phoenix and HBase Spark Applications using Cloudera Data Engineering

Cloudera Data Engineering supports running Spark applications in the Cloudera on cloud. You can use Cloudera Data Engineering to run Phoenix and HBase Spark applications against Cloudera Operational Database.

Before you begin

- Set your Cloudera workload password. For more information, see *Setting the workload password*.
- Synchronize users from the User Management Service in the Cloudera Control Plane into the environment in which your Cloudera Operational Database is running.
- Ensure Cloudera Data Engineering service is enabled and a virtual cluster is created in the Data Engineering Experience. For more information, see *Creating virtual clusters* and *Enabling a Cloudera Data Engineering service*.

Procedure

1. Set HBase and Phoenix versions in your Maven project.
 - a) Use the describe-client-connectivity command for the HBase and Phoenix version information. The following code snippet shows fetching the database connectivity information and parsing the required HBase and Phoenix information to build your application.

```
echo "HBase version"
cdp opdb describe-client-connectivity --database-name my-database --environment-name my-env | jq ".connectors[] | select(.name == \"hbase\") | .version"
echo "Phoenix Connector Version"
cdp opdb describe-client-connectivity --database-name my-database --environment-name my-env | jq ".connectors[] | select(.name == \"phoenix-thick-jdbc\") | .version"
```

```
HBase Version
2.4.6.7.2.14.0-133
Phoenix Spark Version
"6.0.0.7.2.14.0-133"
```

- b) Update the HBase and Phoenix connector versions in our Maven project or configuration.

```
<properties>
...
  <phoenix.connector.version>6.0.0.7.2.14.0-133</phoenix.connector.version>
  <hbase.version>2.4.6.7.2.14.0-133</hbase.version>
```

```
...
</properties>
```

2. Download hbase-site.xml and hbase-omid-client-config.yml configuration files.

- a) Use the describe-client-connectivity command to determine the client configuration URL.

```
cdp opdb describe-client-connectivity --database-name spark-connector --
environment-name cod-7213 | jq ".connectors[] | select(.name == \"hbase\
\") | .configuration.clientConfigurationDetails[] | select(.name == \"HBA
SE\") | .url "
```

```
"https://cod--XXXXXX-gateway0..xcu2-8y8x.dev.cldr.work/clouderamanager/a
pi/v41/clusters/XXXXX/services/hbase/clientConfig"
```

- b) Use the URL gathered from the previous command and run the curl command to download the HBase configurations.

```
curl -f -o "hbase-config.zip" -u "<csso_user>" "https://cod--XXXXXX-gate
way0.cod-7213...xcu2-8y8x.dev.cldr.work/clouderamanager/api/v41/clust
er/s/cod--XXXXX/services/hbase/clientConfig"
```

- c) Unzip hbase-config.zip and copy the hbase-site.xml and hbase-omid-client-config.yml to src/main/resources path in the Maven project.

```
unzip hbase-conf.zip
cp hbase-conf/hbase-site.xml <path to src/main/resources>
cp hbase-conf/hbase-omid-client-config.yml <path to src/main/resources>
```

3. Build the project.

```
$ mvn package
```

4. Create a Cloudera Data Engineering job.

- a) Configure Cloudera Data Engineering CLI to point to the virtual cluster. For more information, see *Downloading the Cloudera Data Engineering command line interface*.
- b) Create a resource using the following command.

```
cde resource create --name phoenix-spark-app-resource
```

- c) Upload the required jars which you downloaded while building the project.

```
cde resource upload --name spark-app-resource --local-path ./target/conn
ector-libs/hbase-shaded-mapreduce-2.4.6.7.2.14.0-133.jar --resource-path
hbase-shaded-mapreduce-2.4.6.7.2.14.0-133.jar
cde resource upload --name spark-app-resource --local-path ./target/c
onnector-libs/opentelemetry-api-0.12.0.jar --resource-path opentelemetry
-api-0.12.0.jar
cde resource upload --name spark-app-resource --local-path ./target/conn
ector-libs/opentelemetry-context-0.12.0.jar --resource-path opentelemetr
y-context-0.12.0.jar
cde resource upload --name spark-app-resource --local-path ./target/con
nector-libs/phoenix5-spark-shaded-6.0.0.7.2.14.0-133.jar --resource-path
phoenix5-spark-shaded-6.0.0.7.2.14.0-133.jar
```

- d) Upload the Spark application app jar that you had built earlier.

```
cde resource upload --name spark-app-resource --local-path ./target/phoe
nix-spark-transactions-0.1.0.jar --resource-path phoenix-spark-transacti
ons-0.1.0.jar
```


- e) Replace HBase, Phoenix, and Phoenix Spark connector versions in the spark-job.json as shown in the following sample and create a Cloudera Data Engineering job using the following JSON and import commands.

```
{
  "mounts": [
    {
      "resourceName": "phoenix-spark-app-resource"
    }
  ],
  "name": "phoenix-spark-app",
  "spark": {
    "className": "com.cloudera.cod.examples.spark.SparkApp",
    "args": [
      "{{ phoenix_jdbc_url }}"
    ],
    "driverCores": 1,
    "driverMemory": "1g",
    "executorCores": 1,
    "executorMemory": "1g",
    "file": "phoenix-spark-transactions-0.1.0.jar",
    "pyFiles": [

    ],
    "files": [
      "hbase-shaded-mapreduce-2.4.6.7.2.14.0-133.jar",
      "opentelemetry-api-0.12.0.jar",
      "opentelemetry-context-0.12.0.jar",
      "phoenix5-spark-shaded-6.0.0.7.2.14.0-133.jar",
    ],
    "numExecutors": 4
  }
}
cde job import --file spark-job.json
```

5. Run the project.

- a) Use the describe-client-connectivity command to determine the base JDBC URL to pass.

```
cdp opdb describe-client-connectivity --database-name my-database --environment-name my-env | jq ".connectors[] | select(.name == \"phoenix-thick-jdbc\") | .configuration.jdbcUrl"
```

- b) Run the job by passing the JDBC URL obtained from the previous command, as an argument to the job.

```
cde job run --name phoenix-spark-app --variable phoenix_jdbc_url=<phoenix_jdbc_url>
```

Related Information

[Enabling a Cloudera Data Engineering service](#)

[Setting the workload password](#)

[Creating virtual clusters](#)

[Downloading the Cloudera Data Engineering command line interface](#)