Data Access 3

Integrating Apache Hive with Spark and BI

Date of Publish: 2018-07-12

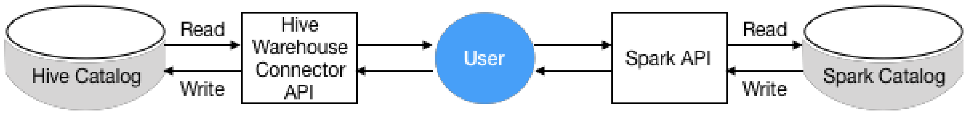
Contents

Hive Warehouse Connector for accessing Apache Spark data

The Hive Warehouse Connector (HWC) is a Spark library/plugin that is launched with the Spark app. You use the Hive Warehouse Connector API to access any managed Hive table from Spark. You must use low-latency analytical processing (LLAP) in HiveServer Interactive to read ACID, or other Hive-managed tables, from Spark.

In HDP 3.0 and later, Spark and Hive use independent catalogs for accessing SparkSQL or Hive tables on the same or different platforms. A table created by Spark resides in the Spark catalog. A table created by Hive resides in the Hive catalog. Databases fall under the catalog namespace, similar to how tables belong to a database namespace. Although independent, these tables interoperate and you can see Spark tables in the Hive catalog, but only when using the Hive Warehouse Connector.

You can use the Hive Warehouse Connector (HWC) API to access any type of table in the Hive catalog from Spark. When you use SparkSQL, standard Spark APIs access tables in the Spark catalog.



Using HWC, you can export tables and extracts from the Spark catalog to Hive and from the Hive catalog to Spark. You export tables and extracts from the Spark catalog to Hive by reading them using Spark APIs and writing them to the Hive catalog using the HWC. You must use low-latency analytical processing (LLAP) in HiveServer Interactive to read ACID, or other Hive-managed tables, from Spark. You do not need LLAP to write to ACID, or other managed tables, from Spark. You do not need HWC to access external tables from Spark.

Using the HWC, you can read and write Apache Spark DataFrames and Streaming DataFrames. Apache Ranger and the HiveWarehouseConnector library provide row and column, fine-grained access to the data.

Limitations

- HWC supports tables in ORC format only.
- The spark thrift server is not supported.
- Table stats are not generated when you write a DataFrame to Hive.

Supported applications and operations

The Hive Warehouse Connector supports the following applications:

- Spark shell
- PySpark
- The spark-submit script

The following list describes a few of the operations supported by the Hive Warehouse Connector:

- Describing a table
- Creating a table for ORC-formatted data
- Selecting Hive data and retrieving a DataFrame
- Writing a DataFrame to Hive in batch
- Executing a Hive update statement
- Reading Hive table data, transforming it in Spark, and writing it to a new Hive table
- Writing a DataFrame or Spark stream to Hive using HiveStreaming

Related Information

HiveWarehouseConnector Github project (select a feature branch)

HiveWarehouseConnector for handling Apache Spark data

Community Connection: Integrating Apache Hive with Apache Spark--Hive Warehouse Connector

Hive Warehouse Connector Use Cases

Apache Spark-Apache Hive connection configuration

You need to understand the workflow and service changes involved in accessing ACID table data from Spark. You can configure Spark properties in Ambari for using the Hive Warehouse Connector.

Prerequisites

The Hive connection string must include a user name and password; otherwise, Spark and Hive cannot connect. For example:

```
jdbc:hive2://
<host>:2181;serviceDiscoveryMode=zooKeeper;zooKeeperNamespace=hiveserver2-
interactive;user=<user name>;password=<password>
```

You need to use the following software to connect Spark and Hive using the HiveWarehouseConnector library.

- HDP 3.0 or later
- Spark2
- Hive with HiveServer Interactive (HSI)

  The Hive Warehouse Connector (HWC) and low-latency analytical processing (LLAP) are required for certain tasks, as shown in the following table:

  Table 1: Spark Compatibility

Tasks	HWC Required	LLAP Required	Other Requirement/Comments
Read Hive managed tables from Spark	Yes	Yes	Ranger ACLs enforced.
Write Hive managed tables from Spark	Yes	No	Ranger ACLs enforced. Supports ORC only.
Read Hive external tables from Spark	No	Only if HWC is used	Table must be defined in Spark catalog. Ranger ACLs not enforced.
Write Hive external tables from Spark	No	No	Ranger ACLs enforced.

  You need low-latency analytical processing (LLAP) in HSI to read ACID, or other Hive-managed tables, from Spark. You do not need LLAP to write to ACID, or other managed tables, from Spark. The HWC library internally uses the Hive Streaming API and LOAD DATA Hive commands to write the data. You do not need LLAP to access external tables from Spark with caveats shown in the table above.

Required properties

You must add several Spark properties through spark-2-defaults in Ambari to use the Hive Warehouse Connector for accessing data in Hive. Alternatively, configuration can be provided for each job using hive --conf.

Property	Description	Comments
spark.sql.hive.hiveserver2.jdbc.url	URL for HiveServer2 Interactive	In Ambari, copy the value from Services > Hive > Summary > HIVESERVER2 INTERACTIVE JDBC URL.
spark.datasource.hive.warehouse.metastoreUri	URI for metastore	Copy the value from hive.metastore.uris. For example, thrift://mycluster-1.com:9083.
spark.datasource.hive.warehouse.load.staging.dir	HDFS temp directory for batch writes to Hive	For example, /tmp.
spark.hadoop.hive.llap.daemon.service.hosts	Application name for LLAP service	Copy value from Advanced hive-interactive-site > hive.llap.daemon.service.hosts.

Property	Description	Comments
spark.hadoop.hive.zookeeper.quorum	Zookeeper hosts used by LLAP	Copy value from Advanced hive-sitehive.zookeeper.quorum.

Spark on a Kerberized YARN cluster

In Spark client mode on a kerberized Yarn cluster, set the following property: spark.sql.hive.hiveserver2.jdbc.url.principal. This property must be equal to hive.server2.authentication.kerberos.principal. In Ambari, copy the value for this property from **Services** > **Hive** > **Configs** > **Advanced** > **Advanced hive-site** hive.server2.authentication.kerberos.principal.

Related Information

HiveWarehouseConnector Github project (select a feature branch)

HiveWarehouseConnector for handling Apache Spark data

Zeppelin configuration for using the Hive Warehouse Connector

You can use the Hive Warehouse Connector in Zeppelin notebooks using the spark2 interpreter by modifying or adding properties to your spark2 interpreter settings.

Interpreter properties

- spark.jars

  /usr/hdp/current/hive_warehouse_connector/hive-warehouse-connector-assembly-<version>.jar
- spark.submit.pyfiles

  usr/hdp/current/hive_warehouse_connector/pyspark_hwc-<version>.zip
- spark.hadoop.hive.llap.daemon.service.hosts

  App name for LLAP service. In Ambari, copy the value from **Services** > **Hive** > **Configs** > **Advanced hive-interactive-site** > **hive.llap.daemon.service.hosts**.
- spark.sql.hive.hiveserver2.jdbc.url

  URL for HiveServer2 Interactive. In Ambari, copy the value from **Services** > **Hive** > **Summary** > **HIVESERVER2 INTERACTIVE JDBC URL**.
- spark.yarn.security.credentials.hiveserver2.enabled

  Only enable for kerberized cluster-mode.
- spark.sql.hive.hiveserver2.jdbc.url.principal

  Kerberos principal for HiveServer2 Interactive. In Ambari, copy the value from Advanced hive-site > hive.server2.authentication.kerberos.principal.
- spark.hadoop.hive.zookeeper.quorum

  ZooKeeper hosts used by LLAP. In Ambari, copy the value from **Services** > **Hive** > **Configs** > **Advanced hive-site** > **hive.zookeeper.quorum**.

Submit a Hive Warehouse Connector Scala or Java application

You can submit an app based on the HiveWarehouseConnector library to run on Spark Shell, PySpark, and spark-submit.

Procedure

1. Locate the hive-warehouse-connector-assembly jar in /usr/hdp/current/hive_warehouse_connector/.

2. Add the connector jar to the app submission using --jars.

```
spark-shell --jars /usr/hdp/current/hive_warehouse_connector/hive-
warehouse-connector-assembly-<version>.jar
```

Related Information

HiveWarehouseConnector Github project (select a feature branch)

HiveWarehouseConnector for handling Apache Spark data

Submit a Hive Warehouse Connector Python app

You can submit a Python app based on the HiveWarehouseConnector library by following the steps to submit a Scala or Java application, and then adding a Python package.

Procedure

1. Locate the hive-warehouse-connector-assembly jar in /usr/hdp/current/hive_warehouse_connector/.
2. Add the connector jar to the app submission using --jars.

```
spark-shell --jars /usr/hdp/current/hive_warehouse_connector/hive-
warehouse-connector-assembly-<version>.jar
```

3. Locate the pyspark_hwc zip package in /usr/hdp/current/hive_warehouse_connector/.
4. Add the Python package to app submission:

```
spark-shell --jars /usr/hdp/current/hive_warehouse_connector/hive-
warehouse-connector-assembly-1.0.0.jar
```

5. Add the Python package for the connector to the app submission.

```
pyspark --jars /usr/hdp/current/hive_warehouse_connector/hive-
warehouse-connector-assembly-<version>.jar --py-files /usr/hdp/current/
hive_warehouse_connector/pyspark_hwc-<version>.zip
```

Related Information

HiveWarehouseConnector Github project (select a feature branch)

HiveWarehouseConnector for handling Apache Spark data

Hive Warehouse Connector supported types

The Hive Warehouse Connector maps most Apache Hive types to Apache Spark types and vice versa, but there are a few exceptions that you must manage.

Spark-Hive supported types mapping

The following types are supported for access through HiveWareHouseConnector library:

Spark Type	Hive Type
ByteType	TinyInt
ShortType	SmallInt
IntegerType	Integer
LongType	BigInt
FloatType	Float

Spark Type	**Hive Type**
DoubleType	Double
DecimalType	Decimal
StringType*	String, Varchar*
BinaryType	Binary
BooleanType	Boolean
TimestampType**	Timestamp**
DateType	Date
ArrayType	Array
StructType	Struct

- * StringType (Spark) and String, Varchar (Hive)

  A Hive String or Varchar column is converted to a Spark StringType column. When a Spark StringType column has maxLength metadata, it is converted to a Hive Varchar column; otherwise, it is converted to a Hive String column.
- ** Timestamp (Hive)

  The Hive Timestamp column loses submicrosecond precision when converted to a Spark TimestampType column, because a Spark TimestampType column has microsecond precision, while a Hive Timestamp column has nanosecond precision.

  Hive timestamps are interpreted to be in UTC time. When reading data from Hive, timestamps are adjusted according to the local timezone of the Spark session. For example, if Spark is running in the America/New_York timezone, a Hive timestamp 2018-06-21 09:00:00 is imported into Spark as 2018-06-21 05:00:00. This is due to the 4-hour time difference between America/New_York and UTC.

Spark-Hive unsupported types

Spark Type	**Hive Type**
CalendarIntervalType	Interval
N/A	Char
MapType	Map
N/A	Union
NullType	N/A
TimestampType	Timestamp With Timezone

Related Information

HiveWarehouseConnector Github project (select a feature branch)

HiveWarehouseConnector for handling Apache Spark data

HiveWarehouseSession API operations

HiveWarehouseSession acts as an API to bridge Spark with Hive. In your Spark source code, you create an instance of HiveWarehouseSession. You use the language-specific code to create the HiveWarehouseSession.

Import statements and variables

The following string constants are defined by the API:

- HIVE_WAREHOUSE_CONNECTOR

- DATAFRAME_TO_STREAM
- STREAM_TO_STREAM

For more information, see the Github project for the Hive Warehouse Connector.

Assuming spark is running in an existing SparkSession, use this code for imports:

- Scala

```
import com.hortonworks.hwc.HiveWarehouseSession
import com.hortonworks.hwc.HiveWarehouseSession._
val hive = HiveWarehouseSession.session(spark).build()
```

- Java

```
import com.hortonworks.hwc.HiveWarehouseSession;
import static com.hortonworks.hwc.HiveWarehouseSession.*;
HiveWarehouseSession hive = HiveWarehouseSession.session(spark).build();
```

- Python

```
from pyspark_llap import HiveWarehouseSession
hive = HiveWarehouseSession.session(spark).build()
```

Catalog operations

- Set the current database for unqualified Hive table references

  hive.setDatabase(<database>)
- Execute a catalog operation and return a DataFrame

  hive.execute("describe extended web_sales").show(100)
- Show databases

  hive.showDatabases().show(100)
- Show tables for the current database

  hive.showTables().show(100)
- Describe a table

  hive.describeTable(<table_name>).show(100)
- Create a database

  hive.createDatabase(<database_name>,<ifNotExists>)
- Create an ORC table

  ```
  hive.createTable("web_sales").ifNotExists().column("sold_time_sk",
   "bigint").column("ws_ship_date_sk", "bigint").create()
  ```

  See the CreateTableBuilder interface section below for additional table creation options. Note: You can also create tables through standard Hive using hive.executeUpdate.
- Drop a database

  hive.dropDatabase(<databaseName>, <ifExists>, <useCascade>)
- Drop a table

  hive.dropTable(<tableName>, <ifExists>, <usePurge>)

Read operations

Execute a Hive SELECT query and return a DataFrame.

hive.executeQuery("select * from web_sales")

Write operations

- Execute a Hive update statement

  hive.executeUpdate("ALTER TABLE old_name RENAME TO new_name")

  Note: You can execute CREATE, UPDATE, DELETE, INSERT, and MERGE statements in this way.
- Write a DataFrame to Hive in batch (uses LOAD DATA INTO TABLE)

  Java/Scala:

  ```
  df.write.format(HIVE_WAREHOUSE_CONNECTOR).option("table",
   <tableName>).save()
  ```

  Python:

  ```
  df.write.format(HiveWarehouseSession().HIVE_WAREHOUSE_CONNECTOR).option("table",
   &tableName>).save()
  ```
- Write a DataFrame to Hive using HiveStreaming

  Java/Scala:

  ```
  //Using dynamic partitioning
  df.write.format(DATAFRAME_TO_STREAM).option("table", <tableName>).save()

  //Or, to write to static partition
  df.write.format(DATAFRAME_TO_STREAM).option("table",
   <tableName>).option("partition", <partition>).save()
  ```

  Python:

  ```
  //Using dynamic partitioning
  df.write.format(HiveWarehouseSession().DATAFRAME_TO_STREAM).option("table",
   <tableName>).save()

  //Or, to write to static partition
  df.write.format(HiveWarehouseSession().DATAFRAME_TO_STREAM).option("table",
   <tableName>).option("partition", <partition>).save()
  ```
- Write a Spark Stream to Hive using HiveStreaming.

  Java/Scala:

  ```
  stream.writeStream.format(STREAM_TO_STREAM).option("table",
   "web_sales").start()
  ```

  Python:

  ```
  stream.writeStream.format(HiveWarehouseSession().STREAM_TO_STREAM).option("table",
   "web_sales").start()
  ```

ETL example (Scala)

Read table data from Hive, transform it in Spark, and write to a new Hive table.

```
import com.hortonworks.hwc.HiveWarehouseSession
import com.hortonworks.hwc.HiveWarehouseSession._
```

```
val hive = HiveWarehouseSession.session(spark).build()
hive.setDatabase("tpcds_bin_partitioned_orc_1000")
val df = hive.executeQuery("select * from web_sales")
df.createOrReplaceTempView("web_sales")
hive.setDatabase("testDatabase")
hive.createTable("newTable")
  .ifNotExists()
  .column("ws_sold_time_sk", "bigint")
  .column("ws_ship_date_sk", "bigint")
  .create()
sql("SELECT ws_sold_time_sk, ws_ship_date_sk FROM web_sales WHERE
 ws_sold_time_sk > 80000)
  .write.format(HIVE_WAREHOUSE_CONNECTOR)
  .option("table", "newTable")
  .save()
```

HiveWarehouseSession interface

```
package com.hortonworks.hwc;

public interface HiveWarehouseSession {

//Execute Hive SELECT query and return DataFrame
   Dataset<Row> executeQuery(String sql);

//Execute Hive update statement
   boolean executeUpdate(String sql);

//Execute Hive catalog-browsing operation and return DataFrame
   Dataset<Row> execute(String sql);

//Reference a Hive table as a DataFrame
   Dataset<Row> table(String sql);

//Return the SparkSession attached to this HiveWarehouseSession
   SparkSession session();

//Set the current database for unqualified Hive table references
   void setDatabase(String name);

/**
 * Helpers: wrapper functions over execute or executeUpdate
*/

//Helper for show databases
   Dataset<Row> showDatabases();

//Helper for show tables
   Dataset<Row> showTables();

//Helper for describeTable
   Dataset<Row> describeTable(String table);

//Helper for create database
   void createDatabase(String database, boolean ifNotExists);

//Helper for create table stored as ORC
   CreateTableBuilder createTable(String tableName);

//Helper for drop database
   void dropDatabase(String database, boolean ifExists, boolean cascade);
//Helper for drop table
   void dropTable(String table, boolean ifExists, boolean purge);
}
```

CreateTableBuilder interface

```

package com.hortonworks.hwc;

public interface CreateTableBuilder {

  //Silently skip table creation if table name exists
  CreateTableBuilder ifNotExists();

  //Add a column with the specific name and Hive type
  //Use more than once to add multiple columns
  CreateTableBuilder column(String name, String type);

  //Specific a column as table partition
  //Use more than once to specify multiple partitions
  CreateTableBuilder partition(String name, String type);

  //Add a table property
  //Use more than once to add multiple properties
  CreateTableBuilder prop(String key, String value);

  //Make table bucketed, with given number of buckets and bucket columns
  CreateTableBuilder clusterBy(long numBuckets, String ... columns);

  //Creates ORC table in Hive from builder instance
  void create();
}
```

Related Information

HiveWarehouseConnector Github project (select a feature branch)

HiveWarehouseConnector for handling Apache Spark data

Community Connection: Integrating Apache Hive with Apache Spark--Hive Warehouse Connector

Hive Warehouse Connector Use Cases

Connecting Apache Hive to BI tools

To query, analyze, and visualize data stored within the Hortonworks Data Platform using drivers provided by Hortonworks, you connect Hive to Business Intelligence (BI) tools.

About this task

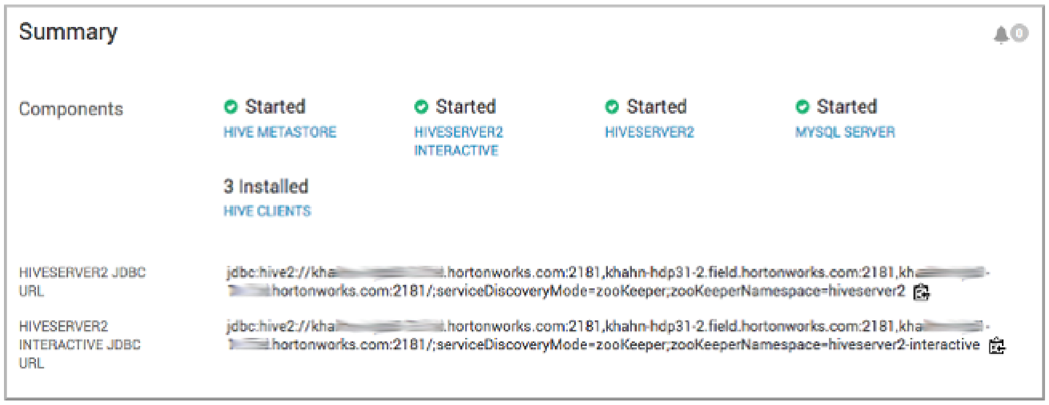
How you connect to Hive depends on a number of factors: the location of Hive inside or outside the cluster, the HiveServer deployment, the type of transport, transport-layer security, and authentication. HiveServer is the server interface that enables remote clients to execute queries against Hive and retrieve the results using a JDBC or ODBC connection. The ODBC driver from Simba and the Apache Hive JDBC driver is available for download as described in the next topic. HDP installs the Hive JDBC driver on one of the edge nodes in your cluster.

Before you begin

- You chose a Hive authorization model.
- You configured authenticated users for querying Hive through JDBC or ODBC driver by setting value of the hive.server2.enable.doAs configuration property in the hive.site.xml file.

Procedure

1. Locate the Apache Hive JDBC driver or download the ODBC driver.
2. Depending on the type of driver you obtain, proceed as follows:
    - If you use the Simba ODBC driver, follow instructions on the ODBC driver download site, and skip the rest of the steps in this procedure.
    - If you use a Apache Hive JDBC driver, specify the basic JDBC connection string as described in the following steps.
3. In Ambari, select **Services** > **Hive** > **Summary**.
4. Copy the JDBC URL for HiveServer: Click the clipboard icon.



Summary

Components

Started HIVE METASTORE

Started HIVESERVER2 INTERACTIVE

Started HIVESERVER2

Started MYSQL SERVER

3 Installed HIVE CLIENTS

HIVESERVER2 JDBC URL

```
jdbc:hive2://kha[redacted].hortonworks.com:2181,khahn-hdp31-2.field.hortonworks.com:2181,kh[redacted]-
[redacted].hortonworks.com:2181/;serviceDiscoveryMode=zooKeeper;zooKeeperNamespace=hiveserver2
```

HIVESERVER2 INTERACTIVE JDBC URL

```
jdbc:hive2://kha[redacted].hortonworks.com:2181,khahn-hdp31-2.field.hortonworks.com:2181,kha[redacted]-
[redacted].hortonworks.com:2181/;serviceDiscoveryMode=zooKeeper;zooKeeperNamespace=hiveserver2-interactive
```

5. Send the JDBC connection string to the BI tool, such as Tableau.

Related Information

HiveWarehouseConnector Github project (select a feature branch)

Locate the JDBC or ODBC driver

You download the Apache Hive JDBC driver, navigate to the installed JDBC driver, or you download the Simba ODBC driver.

About this task

Cloudera provides the Apache Hive JDBC driver as part of the HDP distribution, and provides a Simba ODBC driver as an add-on to the distribution for HDP support subscription customers.

Procedure

1. Get the driver.
    -
    - On a cluster node, navigate to /usr/hdp/current/hive-client/lib to locate the Apache Hive driver hive-jdbc.jar that HDP installed on your cluster.
    - Using the version number of the JAR in /usr/hdp/current/hive-client/lib, download the same Apache Hive JDBC driver hive-jdbc from the driver archive.
    - Download the Simba ODBC driver from the Cloudera downloads page. Skip the rest of the steps in this procedure and follow ODBC driver installation instructions.
2. Optionally, if you run a host outside of the Hadoop cluster, to use the JDBC driver in HTTP and HTTPS modes, give clients access to hive-jdbc-<version>-standalone.jar, hadoop-common.jar, and hadoop-auth.jar.

Specify the JDBC connection string

You construct a JDBC URL to connect Hive to a BI tool.

About this task

In embedded mode, HiveServer runs within the Hive client, not as a separate process. Consequently, the URL does not need a host or port number to make the JDBC connection. In remote mode, the URL must include a host and port number because HiveServer runs as a separate process on the host and port you specify. The JDBC client and HiveServer interact using remote procedure calls using the Thrift protocol. If HiveServer is configured in remote mode, the JDBC client and HiveServer can use either HTTP or TCP-based transport to exchange RPC messages.

Procedure

1. Create a minimal JDBC connection string for connecting Hive to a BI tool.
   - Embedded mode: Create the JDBC connection string for connecting to Hive in embedded mode.
   - Remote mode: Create a JDBC connection string for making an unauthenticated connection to the Hive default database on the localhost port 10000.

   Embedded mode: "jdbc:hive://"
   Remote mode: "jdbc:hive://myserver:10000/default", "", "");
2. Modify the connection string to change the transport mode from TCP (the default) to HTTP using the transportMode and httpPath session configuration variables.
   jdbc:hive2://myserver:10000/default;transportMode=http;httpPath=myendpoint.com;

   You need to specify httpPath when using the HTTP transport mode. <http_endpoint> has a corresponding HTTP endpoint configured in hive-site.xml.
3. Add parameters to the connection string for Kerberos Authentication.
   jdbc:hive2://myserver:10000/default;principal=prin.dom.com@APRINCIPAL.DOM.COM

Related Information

Hortonworks Addons

JDBC connection string syntax

The JDBC connection string for connecting to a remote Hive client requires a host, port, and Hive database name, and can optionally specify a transport type and authentication.

jdbc:hive2://<host>:<port>/<dbName>;<sessionConfs>?<hiveConfs>#<hiveVars>

Connection string parameters

The following table describes the parameters for specifying the JDBC connection.

Table 2: JDBC Connection String Parameters

JDBC Parameter	Description	Required
host	The cluster node hosting HiveServer.	yes
port	The port number to which HiveServer listens.	yes
dbName	The name of the Hive database to run the query against.	yes
sessionConfs	Optional configuration parameters for the JDBC/ODBC driver in the following format: <key1>=<value1>;<key2>=<key2>...;	no
hiveConfs	Optional configuration parameters for Hive on the server in the following format: <key1>=<value1>;<key2>=<key2>; ... The configurations last for the duration of the user session.	no

JDBC Parameter	Description	Required
hiveVars	Optional configuration parameters for Hive variables in the following format: <key1>=<value1>;<key2>=<key2>; ... The configurations last for the duration of the user session.	no

TCP and HTTP Transport

The following table shows variables for use in the connection string when you configure HiveServer in remote mode. The JDBC client and HiveServer can use either HTTP or TCP-based transport to exchange RPC messages. Because the default transport is TCP, there is no need to specify transportMode=binary if TCP transport is desired.

transportMode Variable Value	Description
http	Connect to HiveServer2 using HTTP transport.
binary	Connect to HiveServer2 using TCP transport.

The syntax for using these parameters is:

```
jdbc:hive2://<host>:<port>/
<dbName>;transportMode=http;httpPath=<http_endpoint>;<otherSessionConfs>?
<hiveConfs>#<hiveVars>
```

User Authentication

If configured in remote mode, HiveServer supports Kerberos, LDAP, Pluggable Authentication Modules (PAM), and custom plugins for authenticating the JDBC user connecting to HiveServer. The format of the JDBC connection URL for authentication with Kerberos differs from the format for other authentication models. The following table shows the variables for Kerberos authentication.

User Authentication Variable	Description
principal	A string that uniquely identifies a Kerberos user.
saslQop	Quality of protection for the SASL framework. The level of quality is negotiated between the client and server during authentication. Used by Kerberos authentication with TCP transport.
user	Username for non-Kerberos authentication model.
password	Password for non-Kerberos authentication model.

The syntax for using these parameters is:

```
jdbc:hive2://<host>:<port>/
<dbName>;principal=<HiveServer2_kerberos_principal>;<otherSessionConfs>?
<hiveConfs>#<hiveVars>
```

Transport Layer Security

HiveServer2 supports SSL and Sasl QOP for transport-layer security. The format of the JDBC connection string for SSL differs from the format used by Sasl QOP.

SSL Variable	Description
ssl	Specifies whether to use SSL

SSL Variable	Description
sslTrustStore	The path to the SSL TrustStore.
trustStorePassword	The password to the SSL TrustStore.

The syntax for using the authentication parameters is:

```
jdbc:hive2://<host>:<port>/
<dbName>;ssl=true;sslTrustStore=<ssl_truststore_path>;trustStorePassword=<truststore_pas
<hiveConfs>#<hiveVars>
```

When using TCP for transport and Kerberos for security, HiveServer2 uses Sasl QOP for encryption rather than SSL.

Sasl QOP Variable	Description
principal	A string that uniquely identifies a Kerberos user.
saslQop	The level of protection desired. For authentication, checksum, and encryption, specify auth-conf. The other valid values do not provide encryption.

```
jdbc:hive2://<host>:<port>/
<dbName>;principal=<HiveServer2_kerberos_principal>;saslQop=auth-
conf;<otherSessionConfs>?<hiveConfs>#<hiveVars>
```

Visualizing Apache Hive data using Superset

Using Apache Ambari, you can add Apache Superset to your cluster, connect to Hive, and visualize Hive data in insightful ways, such a chart or an aggregation.

About this task

Apache Superset is a technical preview in HDP 3.0 installed in Ambari by default and available as a service. Apache Superset is a data exploration platform for interactively visualizing data from diverse data sources, such as Hive and Druid. Superset supports more than 30 types of visualizations. In this task, you add Superset to a node in a cluster, start Superset, and connect Superset to Hive.

Before you begin

- You logged into Ambari and started the following components:
  - HiveServer
  - Hive Metastore
  - A database for the Superset metastore, such as the default MySQL Server
  - Hive clients
- You have a user name and password to access Hive.
- You have read, write, and execute permission to /user and /apps/hive/warehouse on HDFS.

Related Information

Apache Superset tutorial

Add the Superset service

You can add the Apache Superset service, which is installed by default with HDP 3.0, to your cluster in Apache Ambari.

About this task

In this task, you use a wizard for customizing Superset services that includes configuring a database backend that Superset uses to store metadata, such as dashboard definitions. By default, SQLite is installed for use as the metastore in nonproduction situations.

SQLite is not a client/server SQL database. For production use, you must install a suitable database. For example purposes, in this task, you accept the default SQLite database.

You configure a SECRET_KEY to encrypt user passwords. The key is stored in the Superset metastore. Do not change the key after setup.Upon completion of this task, you can connect Apache Hive to Superset.

Before you begin

You have installed a client/server database, such as MySQL or PostgreSQL, to use as the Superset database for storing metadata.

Procedure

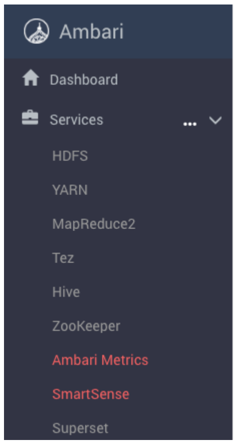
1. From the Ambari navigation pane, select Services, scroll down the list of services to Superset, and click Add Service.
2. In the Add Service wizard, scroll down to Superset, which is selected for addition, and click Next.
3. In Assign Masters, accept the single default node selected to run the Superset service, and click Next.
4. In Customize Services, configure properties:

   Superset Database password

   Superset Database Port--Enter a port number. For example, enter 8088 if you accepted the default SQLite Superset database, or 3306 if you configured MySQL as the Superset database.

   Superset SECRET_KEY--Provide any random number in SECRET_KEY, accept the other default settings, and scroll to the bottom of the page.

   Attention message--Click Show All Properties and follow prompts to configure any properties, such as providing a Superset Admin Password.
5. Click Next.
6. In Customize Services, in Advanced, enter a Superset Admin password.
7. Click Next, and then click Deploy,
8. Click Next, and in Summary, click Complete and confirm completion.
   Superset appears in the Ambari navigation pane.



Connect Apache Hive to Superset

You can connect to Hive to create a Superset visualization.

About this task

Upon completion of this task, you can create a Superset visualization.

Before you begin

You have started the Superset service in Ambari.

Procedure

1. Click Superset.
2. In the Summary portion of Quick Links, click Superset and log in using your Superset user name and password. An empty dashboard appears.
3. From Sources, select Databases.
4. In Add Filter, add a new record.
5. In Add Database, enter the name of your Hive database: for example, default.
6. Enter the SQLAlchemy URL for accessing your database.
   For example, assuming HiveServer is running on node c7402, connect the database named default to the Superset listening port 10000:

   ```
   hive://hive@c7402:10000/default
   ```

   ZooKeeper-based URL discovery is not supported.
7. Click Test Connection.
   The success message appears, and the names of any tables in the database appear at the bottom of the page.
8. Scroll to the bottom of the page, and click Save.

Configure a Superset visualization

In Apache Ambari, after connecting Apache Superset to Apache Hive, you can configure visualizations, such as aggregations, slices of data, or plotted data to better understand the data.

About this task

This task shows you how to create a simple visualization based on a table having the following schema:

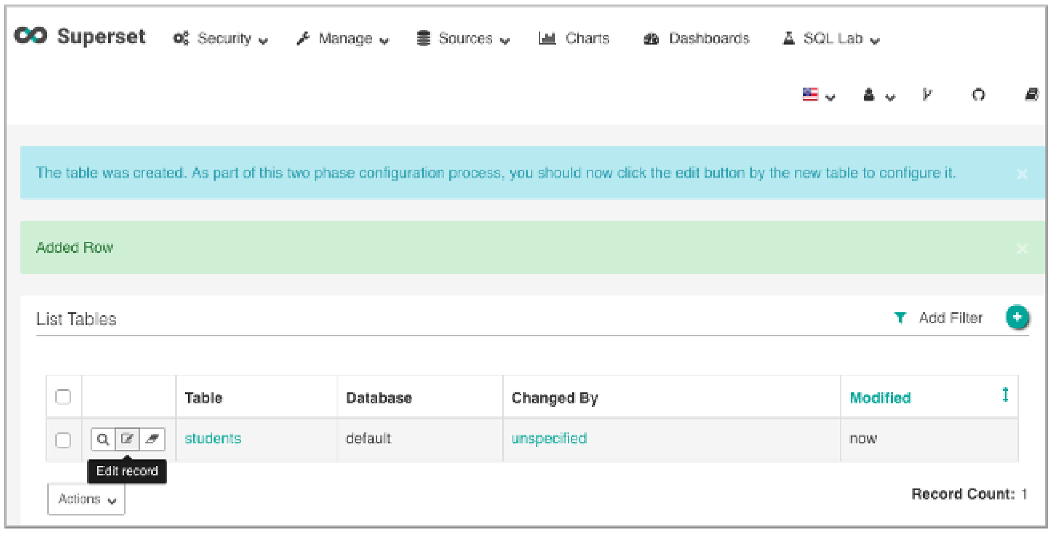
```
CREATE TABLE students (name VARCHAR(64), age INT, gpa DECIMAL(3,2));
```

Before you begin

- You created and populated a table in the Hive warehouse.

Procedure

1. Select Superset from the Ambari main menu.
2. In Summary under Quick Links, click Superset.
3. From the Sources menu, select Tables.
4. In Add Filter, add a new record.
5. On Add Table in Database, select the Hive database connected to Superset.
6. In Table Name, select a Hive table, students in the example below, and click Save.
7. On List Tables, click Edit Record:



The table was created. As part of this two phase configuration process, you should now click the edit button by the new table to configure it.

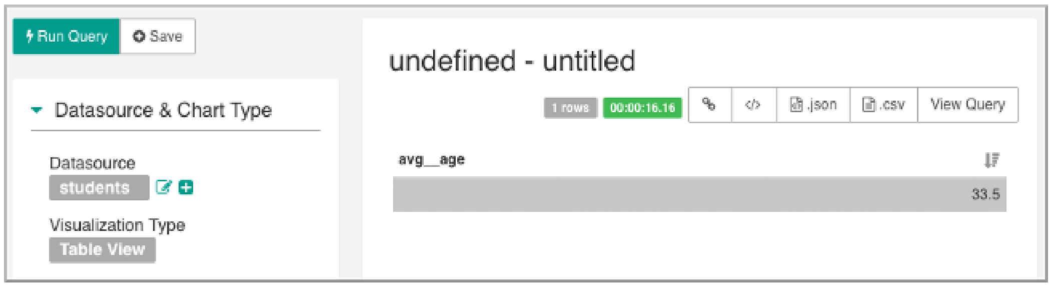
Added Row

List Tables

		Table	Database	Changed By	Modified
☐		students	default	unspecified	now

Record Count: 1

8. On the Detail tab of Edit Table, in Table Name, enter the name of a table in the Hive database.
A table visualization appears, showing an aggregation calculated automatically by Superset: average age 33.5 in this example:



avg__age
33.5