cloudera
Ask Bigger Questions

Cloudera JDBC Driver for Apache Hive
Version 2.5.3

Important Notice

Cloudera, Inc.
1001 Page Mill Road, Building 2
Palo Alto, CA 94304-1008
info@cloudera.com
US: 1-888-789-1488
Intl: 1-650-843-0595
www.cloudera.com

Release Information

Version: 2.5.3

Date: September 4, 2014

Table of Contents

Introduction

Welcome to the Cloudera JDBC Driver for Hive. JDBC is one of the most established and widely supported APIs for connecting to and working with databases. At the heart of the technology is the JDBC driver, which connects an application to the database.

The Cloudera JDBC Driver for Hive is used for direct SQL and HiveQL access to Apache Hadoop / Hive distributions, enabling Business Intelligence (BI), analytics, and reporting on Hadoop / Hive-based data. The driver efficiently transforms an application's SQL query into the equivalent form in HiveQL. Hive Query Language is a subset of SQL-92. If an application is Hive-aware, then the driver is configurable to pass the query through. The driver interrogates Hive to obtain schema information to present to a SQL-based application. Queries, including joins, are translated from SQL to HiveQL. For more information about the differences between HiveQL and SQL, refer to the section *Features* on page 9.

The Cloudera JDBC Driver for Hive complies with the JDBC 3.0 and JDBC 4.0 data standards.

This guide is suitable for users who want to access data residing within Hive from their desktop environment. Application developers may also find the information helpful. Refer to your application for details on connecting via JDBC.

System Requirements

To use the Cloudera JDBC Driver for Hive with the JDBC 3.0 API, each computer where you use the driver must meet the following system requirements:

- Java Runtime Environment (JRE) version 1.4 or 5.0

To use the Cloudera JDBC Driver for Hive with the JDBC 4.0 API, each computer where you use the driver must meet the following minimum system requirements:

- Java Runtime Environment (JRE) version 6.0

The Cloudera JDBC Driver for Hive is tested using Hive 0.10, 0.11, 0.12, and 0.13.

Cloudera JDBC Driver for Hive Files

The Cloudera JDBC Driver for Hive is delivered in two ZIP archives named **Cloudera_HiveJDBC3_*<version>*.zip** and **Cloudera_HiveJDBC4_*<version>*.zip**. Each archive contains the driver supporting the JDBC API version in the archive name.

The **Cloudera_HiveJDBC3_*<version>*.zip** archive contains the following file and folder structure:

- HiveJDBC3
    - hive_metastore.jar

- hive_service.jar
- HiveJDBC3.jar
- libfb303-0.9.0.jar
- libthrift-0.9.0.jar
- log4j-1.2.14.jar
- ql.jar
- slf4j-api-1.5.8.jar
- slf4j-log4j12-1.5.8.jar
- TCLIServiceClient.jar

The **Cloudera_HiveJDBC4_*<version>*.zip** archive contains the following file and folder structure:

- HiveJDBC4
  - hive_metastore.jar
  - hive_service.jar
  - HiveJDBC4.jar
  - libfb303-0.9.0.jar
  - libthrift-0.9.0.jar
  - log4j-1.2.14.jar
  - ql.jar
  - slf4j-api-1.5.8.jar
  - slf4j-log4j12-1.5.8.jar
  - TCLIServiceClient.jar

Using the Cloudera JDBC Driver for Hive

To access a Hive data warehouse using the Cloudera JDBC Driver for Hive, you must set the following:

- Class path
- Driver class
- Connection URL

The Cloudera JDBC Driver for Hive provides read-only access to Hive data.

Setting the Class Path

The class path is the path that the Java Runtime Environment searches for classes and other resource files. For details on setting the class path, refer to http://docs.oracle.com/javase/7/docs/technotes/tools/windows/classpath.html.

To use the Cloudera JDBC Driver for Hive, you must include all the JAR files from the ZIP archive in the class path.

Cloudera JDBC Driver for Hive Classes

The Cloudera JDBC Driver for Hive makes classes with the following fully-qualified class names (FQCNs) available, supporting JDBC 3.0 and JDBC 4.0 in connections to Hive Server 1 and Hive Server 2 instances:

- com.cloudera.hive.jdbc3.HS1Driver
- com.cloudera.hive.jdbc3.HS2Driver
- com.cloudera.hive.jdbc4.HS1Driver
- com.cloudera.hive.jdbc4.HS2Driver

Initialize the appropriate class for the Hive Server instance and your application as needed prior to connecting to the Hive Server.

Hive and Hive Server Protocol Versions

The following table shows the versions of Hive Server that are supported by the driver for each version of Hive.

Hive Version	Hive Server 1	Hive Server 2
0.13	Not Currently Supported	Yes
0.12	Yes	Yes
0.11	Yes	Yes
0.10	Yes	Yes

Building the Connection URL

Using the connection URL, you supply connection information to the data source that you are accessing. The connection URL for the Cloudera JDBC Driver for Hive takes the following form:

`jdbc:`*`Subprotocol`*`://`*`Host`*`:`*`Port`*`[/`*`Schema`*`];`*`Property1`*`=`*`Value`*`;`*`Property2`*`=`*`Value`*`;…`

The placeholders in the connection URL are defined as follows:

- *Subprotocol* is the value **hive** if you are connecting to a Hive Server 1 system. If you are connecting to a Hive Server 2 system, use the value **hive2**
- *Host* is the DNS or IP address of the server hosting the Hive data warehouse.
- *Port* is the port to connect to on *Host*
- *Schema* is the name of the schema/database you want to access. Specifying a schema is optional. If you do not specify a schema, then the schema named default is used.

  Note: You can issue queries on other schemas by explicitly specifying the schema in the query. To inspect your databases and determine the appropriate database schema to use, type **show databases** at the Hive command prompt.
- You can specify one or more connection properties. For details on all available properties, see *Appendix C: Driver Configuration Options* on page 14.

  Important: Properties are case sensitive. Do not duplicate properties in the connection URL.

For example, to connect to a Hive Server 2 instance installed on the local computer by using a user name and password:

```
jdbc:hive2://localhost:10000;AuthMech=3;UID=UserName;PWD=Password
```

UserName and *Password* specify credentials for an existing user on the host that is running Hive Server 2.

Note: The UID and PWD properties apply when the authentication mechanism being used is User Name and Password or User Name and Password with Secure Sockets Layer (SSL). The UID property also applies when User Name authentication is enabled. For further details on the properties that you can use in the connection URL, see *Appendix C: Driver Configuration Options* on page 14.

Note: If you use Hive Server2 (**hive2**) and do not specify any parameters, then the UID will default to “hive” and the AuthMech will default to “2”.

Java Sample Code

The following Java code provides an example demonstrating how to use the JDBC API to:

- Register the Cloudera JDBC Driver for Hive
- Establish a connection to a Hive database
- Query the database
- Parse a result set
- Handle exceptions
- Clean up to avoid memory leakage

Important: To use the Cloudera JDBC Driver for Hive in an application, you must include all the JAR files from the ZIP archive in the class path for your Java project.

```
// java.sql packages are required
import java.sql.*;

class ClouderaJDBCHiveExample {

   // Define a string as the fully qualified class name (FQCN)
   // of the desired JDBC driver
   static String JDBCDriver =
         "com.cloudera.hive.jdbc3.HS1Driver";
   // Define a string as the connection URL
   static String ConnectionURL = "jdbc:hive://192.168.1.1:10000";

   public static void main(String[] args) {

         Connection con = null;
         Statement stmt = null;
         ResultSet rs = null;

         // Define the SQL statement for a query
         String query = "SELECT first_name, last_name, emp_id
               FROM default.emp";

         try {

               // Register the driver using the class name
               Class.forName(JDBCDriver);

               // Establish a connection using the connection URL
               con = DriverManager.getConnection(ConnectionURL);

               // Create a Statement object for sending SQL
               //  statements to the database
               stmt = con.createStatement();

               // Execute the SQL statement
               rs = stmt.executeQuery(query);

               // Display a header line for output appearing in
               // the Console View
               System.out.printf("%20s%20s%20s\r\n", "FIRST NAME",
                     "LAST NAME" , "EMPLOYEE ID");
```

```
            // Step through each row in the result set returned
            // from the database
            while(rs.next()) {
                  // Retrieve values from the row where the
                  // cursor is currently positioned using column
                  // names
                  String FirstName = rs.getString("first_name");
                  String LastName = rs.getString("last_name");
                  String EmployeeID = rs.getString("emp_id");

                  // Display values in columns 20 characters
                  // wide in the Console View using the
                  // Formatter
            System.out.printf("%20s%20s%20s\r\n", FirstName,
                  LastName, EmployeeID);
            }

      } catch (SQLException se) {
            // Handle errors encountered during interaction
            // with the data source
            se.printStackTrace();
      } catch (Exception e) {
            // Handle other errors
            e.printStackTrace();
      } finally {
            // Perform clean up
            try {
                  if (rs != null) {
                        rs.close();
                  }
            } catch (SQLException se1) {
                  // Log this
            }

            try {
                  if (stmt != null) {
                        stmt.close();
                  }
            } catch (SQLException se2) {
                  // Log this
            }

            try {
```

```
                    if (con != null) {
                        con.close();
                    }
                } catch (SQLException se3) {
                    // Log this
                    se3.printStackTrace();
                } // End try

            } // End try

        } // End main
    } // End ClouderaJDBCHiveExample
```

Configuring Authentication

When using the Cloudera JDBC Driver for Hive, you configure authentication via properties specified in the connection URL.

For details on selecting an appropriate authentication mechanism when using the Cloudera JDBC Driver for Hive, see *Appendix A: Authentication Options* on page 11.

For details on properties you can use in the connection URL, see *Appendix C: Driver Configuration Options* on page 14.

Using No Authentication

To configure Cloudera JDBC Driver for Hive to connect without authenticating:

- Set the **AuthMech** property to 0

For example:

```
jdbc:hive2://localhost:10000;AuthMech=0
```

Using Kerberos

For information on operating Kerberos, refer to the documentation for your operating system.

To configure the Cloudera JDBC Driver for Hive to use Kerberos authentication:

1. Set the **AuthMech** property to 1.
2. If your Kerberos setup does not define a default realm or if the realm of your Hive server is not the default, then set the appropriate realm using the KrbRealm property.
3. Set the **KrbHostFQDN** property to the fully qualified domain name of the Hive Server 2 host.
4. Set the **KrbServiceName** property to the service name of the Hive Server 2.

For example:

```
jdbc:hive2://localhost:10000;AuthMech=1;KrbRealm=EXAMPLE.COM;KrbH
ostFQDN=hs2.example.com;KrbServiceName=hive
```

Using User Name

To configure User Name authentication:

1. Set the **AuthMech** property to 2.
2. Set the **UID** property to a user name that is recognized by the Hive server.

For example:

```
jdbc:hive2://localhost:10000;AuthMech=2;UID=hs2
```

Using User Name and Password

To configure User Name and Password authentication:

1. Set the **AuthMech** property to 3.
2. Set the **UID** property to a user name that is recognized by the Hive server.
3. Set the **PWD** property to the password corresponding to the user name you provided in step 2.

For example:

```
jdbc:hive2://localhost:10000;AuthMech=3;UID=hs2;PWD=*****
```

Using User Name and Password with Secure Sockets Layer

To configure User Name and Password authentication using SSL:

1. Create a KeyStore containing your signed, trusted SSL certificate.
2. Set the **AuthMech** property to 4.
3. Set the **SSLKeyStore** property to the full path of the KeyStore you created in step 1, including the file name.
4. Set the **SSLKeyStorePwd** property to the password for the KeyStore you created in step 1.
5. Set the **UID** property to a user name that is recognized by the Hive server.
6. Set the **PWD** property to the password corresponding to the user name you provided in step 5.

For example:

```
jdbc:hive2://localhost:10000;AuthMech=4;SSLKeyStore=C:\\Users\\bs
mith\\Desktop\\keystore.jks;SSLKeyStorePwd=****;UID=hs2;PWD=****
```

Note: By default, the SSL certificate used by the server cannot be self-signed. Use the AllowSelfSignedCerts property to allow the SSL certificate used by the server to be self-signed. By default, the Common Name (CN) contained in a certificate signed by a Certification Authority (CA) must match the host name of the Hive server. You can allow mismatches by using the CAIssuedCertNamesMismatch property. The driver always allows the Common Name contained in self-signed certificates to mismatch the host name. See *Appendix C: Driver Configuration Options* on page 14 for more information.

Features

SQL Query versus HiveQL Query

The native query language supported by Hive is HiveQL. HiveQL is a subset of SQL-92. However, the syntax is different enough that most applications do not work with native HiveQL.

Data Types

The following data types are supported:

- TINYINT
- SMALLINT
- INT
- BIGINT
- FLOAT
- DOUBLE
- DECIMAL

  Note: In Hive 0.13 and later, users can specify scale and precision when creating tables using the DECIMAL data type.
- BOOLEAN
- STRING
- CHAR

  Note: The CHAR data type is supported in Hive 0.13 and later.
- BINARY
- TIMESTAMP
- DATE
- VARCHAR

Note: The aggregate types (ARRAY, MAP, and STRUCT) are not yet supported. Columns of aggregate types are treated as STRING columns.

Catalog and Schema Support

The Cloudera JDBC Driver for Hive supports both catalogs and schemas in order to make it easy for the driver to work with various JDBC applications. Since Hive only organizes tables into schema/database, a synthetic catalog named Hive is added, under which all of the schemas/databases are organized. The driver also maps the JDBC schema to the Hive schema/database.

Note: "CatalogSchemaSwitch" being set to 1 on the connection string will cause Hive Catalogs to be treated as schemas in the driver as a restriction for filtering.

Appendix A: Authentication Options

Hive Server 1 supports the following authentication mechanisms:

- No Authentication

Hive Server 2 supports the following authentication mechanisms:

- No Authentication
- Kerberos
- User Name
- User Name and Password
- User Name and Password with Secure Sockets Layer

To determine the authentication mechanism configured for your Hive Server 2, examine the following properties in your hive-site.xml file:

- **hive.server2.authentication**—This property sets the authentication mode for Hive Server 2. The following values are available:
  - **NOSASL** disables the Simple Authentication and Security Layer (SASL).
  - **KERBEROS** enables Kerberos authentication.
  - **NONE** enables plain SASL transport. NONE is the default value.
  - **PLAINSASL** enables user name and password based authentication using a cleartext password mechanism.
- **hive.server2.enable.doAs**—If set to the default value of TRUE, then Hive processes queries as the user submitting the query. If set to FALSE, then queries are run as the user that runs the hiveserver2 process.

For more details on Hive Server authentication mechanisms, see the documentation for your Hadoop / Hive distribution.

Table 1 lists authentication mechanisms to configure for the Cloudera JDBC Driver for Hive based on the settings of the hive.server2.authentication and hive.server2.enable.doAs properties in the hive-site.xml file.

hive.server2.authentication	hive.server2.enable.doAs	Driver Authentication Mechanism
NOSASL	FALSE	No Authentication
KERBEROS	TRUE or FALSE	Kerberos
NONE	TRUE or FALSE	User Name
PLAINSASL	TRUE or FALSE	User Name and Password

Table 1 Cloudera JDBC Driver for Hive Authentication Mechanism Configurations

More details on selecting the appropriate authentication mechanism for the Cloudera JDBC Driver for Hive are provided below.

For examples showing how to configure each authentication mechanism, see *Configuring Authentication* on page 7.

Using No Authentication

When hive.server2.authentication is set to NOSASL, you must configure your connection to use no authentication.

Note: Setting hive.server2.authentication to NOSASL and hive.server2.enable.doAs to TRUE creates an error. While the service starts, the configuration results in an unusable service.

Using Kerberos

When connecting to a Hive server of type Hive Server 2 and hive.server2.authentication is set to KERBEROS, you must configure your connection to use Kerberos.

Using User Name

When connecting to a Hive server of type Hive Server 2 and hive.server2.authentication is set to NONE, you must configure your connection to use User Name. Validation of the credentials that you include depends on hive.server2.enable.doAs:

- If hive.server2.enable.doAs is set to TRUE, then the User Name in the driver configuration must be an existing operating system user on the host running Hive Server 2.
- If hive.server2.enable.doAs is set to FALSE, then the User Name in the driver configuration is ignored.

If no User Name is specified in the driver configuration, then the driver defaults to using the user name anonymous.

Note: If you deploy Hadoop using Apache Ambari, then by default the authentication method is User Name.

Using User Name and Password

When connecting to a Hive server of type Hive Server 2 that is configured to use plain SASL authentication, you must configure your connection to include a User Name and Password.

Using User Name and Password with Secure Sockets Layer

When connecting to a Hive server of type Hive Server 2 that is configured to use plain SASL authentication and SSL, you must configure your connection to include a User Name and Password.

Appendix B: Configuring Kerberos Authentication for Windows

Note: If you do not use Kerberos authentication, then you do not need to configure Kerberos authentication.

On the Windows platform, using Kerberos authentication as part of an Active Directory implementation is recommended. To implement Active Directory, contact your system administrator.

When using Kerberos for enhanced authentication in an Active Directory implementation, users are granted a Ticket to Get Tickets (TGT) automatically when logging on to the network. The Kerberos Key Distribution Center (KDC) is installed as part of the domain controller and performs two service functions: the Authentication Service (AS) and the Ticket-Granting Service (TGS). The KDC has access to Active Directory user account information. After the Authentication Service verifies the user, the Ticket-Granting Service provides the Ticket to Get Tickets.

In an Active Directory implementation using Kerberos, Kerberos uses the Active Directory cache to store credentials.

The Cloudera JDBC Driver for Hive requests a service ticket in the process of establishing a connection to the Hive database.

Appendix C: Driver Configuration Options

Table 2 lists and describes the properties that you can use to configure the behavior of the Cloudera JDBC Driver for Hive.

Note: You can set configuration properties using the connection URL. For details on the connection URL, see *Building the Connection URL* on page 3.

Property	Default Value	Description
AllowSelfSignedCerts	0	When set to the default value of 0, the SSL certificate used by the server cannot be self-signed. When set to 1, the SSL certificate used by the server can be self-signed. **Note:** The setting is only applicable to the User Name and Password with SSL authentication mechanism, and is ignored by other authentication mechanisms. (Optional)
AuthMech	0	The authentication mechanism to use. Set the value to 0 for no authentication, 1 for Kerberos, 2 for User Name, 3 for User Name and Password, 4 for User Name and Password with SSL. (Optional)
CAIssuedCertNamesMismatch	0	The property controls whether to allow the CA issued SSL certificate name to not match the host name of the Hive server. Mismatches are not allowed when the property is set to 0. To allow mismatches, set the property to 1. **Note:** The setting is only applicable to the User Name and Password with SSL authentication mechanism, and is ignored by other authentication mechanisms. (Optional)
DecimalColumnScale	10	The maximum number of digits to the right of the decimal point for numeric data types. (Optional)

Property	Default Value	Description
DefaultStringColumnLength	255	The maximum data length for string columns. **Note:** Hive does not provide the maximum data length for String columns in the columns metadata. This option allows you to tune the maximum data length for String columns. The range of DefaultStringColumnLength is 0 to 32,767. (Optional)
DelegationUID	N/A	Used to delegate all operations against Hive to a user that is different than the authenticated user for the connection. **Note:** The setting is only applicable when connecting to a Hive Server 2 that supports the delegation feature. Otherwise, the setting has no effect. (Optional)
KrbHostFQDN		The fully qualified domain name of the Hive Server 2 host used. (Required if AuthMech is Kerberos)
KrbRealm	Depends on Kerberos configuration	If there is no default realm configured, or if the realm of the Hive Server 2 host is different from the default realm for your Kerberos setup, then define the realm of the Hive Server 2 host using this option. (Optional)
KrbServiceName		The Kerberos service principal name of the Hive Server 2. (Required if AuthMech is Kerberos)
PWD		The password set up for User Name and Password authentication mechanisms. (Required if AuthMech is User Name and Password, or User Name and Password with SSL)
RowsFetchedPerBlock	10000	The maximum number of rows that a query returns at a time. Any positive 32-bit integer is a valid value but testing has shown that performance gains are marginal beyond the default value of 10000 rows.

Property	Default Value	Description
SocketTimeout	0	The number of seconds after which Hive closes the connection with the client application if the connection is idle. The default value of 0 indicates that an idle connection is not closed.
SSLKeyStore	N/A	The full path, including the file name, of the Java KeyStore containing an SSL certificate to use during authentication. See also the SSLKeyStorePwd property. (Required if AuthMech is User Name and Password with SSL)
SSLKeyStorePwd	N/A	The password required to access the Java KeyStore specified using the SSLKeyStore property. (Required if AuthMech is User Name and Password with SSL)
UID		The user name of an existing user on the host running Hive Server 2. **Important:** You must set the hive.server2.authentication property in the hive-site.xml file for the Hive Server 2 to NONE OR You must set up the Hive Server 2 credential when using the User Name and Password authentication mechanism. (Required if AuthMech is User Name, User Name and Password, or User Name and Password with SSL)
UseNativeQuery	0	Enabling the UseNativeQuery option using a value of 1 disables the SQL Connector feature. The SQL Connector feature has been added to the driver to apply transformations to the queries emitted by an application to convert them into an equivalent form in HiveQL. If the application is Hive aware and already emits HiveQL, then turning off the SQL Connector feature avoids the extra overhead of query transformation. (Optional)

Property	Default Value	Description
CatalogSchemaSwitch	0	Enabling the CatalogSchemaSwitch option using a value of 1 on the connection string will cause Impala Catalogs to be treated as schemas in the driver as a restriction for filtering.
PreparedMetaLimitZero	0	Enabling PreparedMetaLimitZero will cause the PreparedStatement.getMetadata() call to request metadata from the server with “LIMIT 0”, increasing performance.

Table 2 Cloudera JDBC Driver for Hive Configuration Options