



Amazon JDBC Connector for Apache Hive

Installation and Configuration Guide

Version 2.6.26

August 2024

Copyright © 2024 Amazon Web Services, Inc. All Rights Reserved.

Information in this document is subject to change without notice. Companies, names and data used in examples herein are fictitious unless otherwise noted. No part of this publication, or the software it describes, may be reproduced, transmitted, transcribed, stored in a retrieval system, decompiled, disassembled, reverse-engineered, or translated into any language in any form by any means for any purpose without the express written permission of Amazon Web Services, Inc.

Parts of this Program and Documentation include proprietary software and content that is copyrighted and licensed by Simba Technologies Incorporated. This proprietary software and content may include one or more feature, functionality or methodology within the ODBC, JDBC, ADO.NET, OLE DB, ODBO, XMLA, SQL and/or MDX component(s).

For information about Simba's products and services, visit: www.magnitude.com.

Contact Us

For support, check the EMR Forum at <https://forums.aws.amazon.com/forum.jspa?forumID=52> or open a support case using the AWS Support Center at <https://aws.amazon.com/support>

About This Guide

The *Amazon JDBC Connector for Apache Hive Installation and Configuration Guide* explains how to install and configure the Amazon JDBC Connector for Apache Hive on all supported platforms. It also provides details about the connector's features.

The guide is intended for end users of the Amazon JDBC Connector for Apache Hive.

To use the Amazon JDBC Connector for Apache Hive, the following knowledge is helpful:

- Familiarity with the platform on which you are using the Amazon JDBC Connector for Apache Hive
- Ability to use the data store to which the Amazon JDBC Connector for Apache Hive is connecting
- An understanding of the role of JDBC technologies in connecting to a data store
- Experience creating and configuring JDBC connections
- Exposure to SQL


Document Conventions


The following conventions are used throughout this guide to emphasize important concepts:

Italics are used when referring to book and document titles.

Bold is used in procedures for graphical user interface elements that a user clicks and text that a user types.

`Monospace font` indicates commands, source code or contents of text files.

 A text box with a blue exclamation mark indicates a short note appended to a paragraph.

 A text box with a yellow exclamation mark indicates an important comment related to the preceding paragraph.

Contents

About the Amazon JDBC Connector for Apache Hive	6
System Requirements	7
Amazon JDBC Connector for Apache Hive Files	8
Installing and Using the Amazon JDBC Connector for Apache Hive	9
Referencing the JDBC Connector Libraries	9
Registering the Connector Class	10
Building the Connection URL	11
Configuring Authentication	13
Using No Authentication	13
Using Kerberos	13
Using Single Sign-On	15
Using User Name	15
Using User Name And Password (LDAP)	16
Using a Hadoop Delegation Token	17
Authentication Mechanisms	20
Configuring Kerberos Authentication for Windows	22
Kerberos Encryption Strength and the JCE Policy Files Extension	27
Configuring SSL	30
Configuring Server-Side Properties	33
Configuring Logging	34
Features	36
SQL Query versus HiveQL Query	36
Data Types	36
Catalog and Schema Support	37
Write-back	38
IHadoopStatement	38
IHadoopConnection	46
Security and Authentication	49
Connector Configuration Options	51

AllowSelfSignedCerts	51
AsyncExecPollInterval	52
AuthMech	52
BinaryColumnLength	53
CAIssuedCertsMismatch	53
CatalogSchemaSwitch	53
DecimalColumnScale	54
DefaultStringColumnLength	54
DelegationToken	54
DelegationUID	55
httpPath	55
IgnoreTransactions	56
KrbAuthType	56
KrbHostFQDN	57
KrbRealm	58
KrbServiceName	58
LogLevel	58
LogPath	60
PreparedMetaLimitZero	60
PWD	60
RowsFetchedPerBlock	61
SocketTimeout	61
SSL	61
SSLKeyStore	62
SSLKeyStorePwd	62
SSLTrustStore	63
SSLTrustStorePwd	63
SSLTrustStoreType	64
SSOWebServerTimeout	64
TransportMode	64
UID	65
UseNativeQuery	65
zk	66
Contact Us	67
Third-Party Trademarks	68

About the Amazon JDBC Connector for Apache Hive

The Amazon JDBC Connector for Apache 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 connector efficiently transforms an application's SQL query into the equivalent form in HiveQL, which is a subset of SQL-92. If an application is Hive-aware, then the connector is configurable to pass the query through to the database for processing. The connector 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, see [Features](#) on page 36.

The Amazon JDBC Connector for Apache Hive complies with the JDBC 4.1 and 4.2 data standards. 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 connector, which connects an application to the database.

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

System Requirements

Each machine where you use the Amazon JDBC Connector for Apache Hive must have Java Runtime Environment (JRE) 8.0, 11.0, installed. If you are using the connector with JDBC API version 4.2, then you must use JRE 8.0, 11.0,.

The connector supports Apache Hive versions 1.0.1 through 3.1.

Amazon JDBC Connector for Apache Hive Files

The Amazon JDBC Connector for Apache Hive is delivered in the following ZIP archives, where *[Version]* is the version number of the connector:

- `Amazon_HiveJDBC41_[Version].zip`
- `Amazon_HiveJDBC42_[Version].zip`

The archive contains the connector supporting the JDBC API version indicated in the archive name, as well as release notes and third-party license information. In addition, the required third-party libraries and dependencies are packaged and shared in the connector JAR file in the archive.

Installing and Using the Amazon JDBC Connector for Apache Hive

To install the Amazon JDBC Connector for Apache Hive on your machine, extract the files from the appropriate ZIP archive to the directory of your choice.

To access a Hive data store using the Amazon JDBC Connector for Apache Hive, you need to configure the following:

- The list of connector library files (see [Referencing the JDBC Connector Libraries](#) on page 9)
- The `Driver` or `DataSource` class (see [Registering the Connector Class](#) on page 10)
- The connection URL for the connector (see [Building the Connection URL](#) on page 11)

Referencing the JDBC Connector Libraries

Before you use the Amazon JDBC Connector for Apache Hive, the JDBC application or Java code that you are using to connect to your data must be able to access the connector JAR files. In the application or code, specify all the JAR files that you extracted from the ZIP archive.

Using the Connector in a JDBC Application

Most JDBC applications provide a set of configuration options for adding a list of connector library files. Use the provided options to include all the JAR files from the ZIP archive as part of the connector configuration in the application. For more information, see the documentation for your JDBC application.

Using the Connector in Java Code

You must include all the connector library files in the class path. This is the path that the Java Runtime Environment searches for classes and other resource files. For more information, see "Setting the Class Path" in the appropriate Java SE Documentation.

For Java SE 8:

- For Windows:
<http://docs.oracle.com/javase/8/docs/technotes/tools/windows/classpath.html>
- For Linux and Solaris:
<http://docs.oracle.com/javase/8/docs/technotes/tools/unix/classpath.html>

Registering the Connector Class

Before connecting to your data, you must register the appropriate class for your application.

The following classes are used to connect the Amazon JDBC Connector for Apache Hive to Hive data stores:

- The `Driver` classes extend `java.sql.Driver`.
- The `DataSource` classes extend `javax.sql.DataSource` and `javax.sql.ConnectionPoolDataSource`.

The connector supports the following fully-qualified class names (FQCNs) that are independent of the JDBC version:

- `com.amazon.hive.jdbc.HS2Driver`
- `com.amazon.hive.jdbc.HS2DataSource`

The following sample code shows how to use the `DriverManager` class to establish a connection for JDBC:

```
private static Connection connectViaDM() throws Exception
{
    Connection connection = null;
    connection = DriverManager.getConnection(CONNECTION_
        URL);
    return connection;
}
```

The following sample code shows how to use the `DataSource` class to establish a connection:

```
private static Connection connectViaDS() throws Exception
{
    Connection connection = null;
    DataSource ds = new
        com.amazon.hive.jdbc.HS2DataSource();
    ds.setURL(CONNECTION_URL);
    connection = ds.getConnection();
    return connection;
}
```

Building the Connection URL

Use the connection URL to supply connection information to the data source that you are accessing. The following is the format of the connection URL for the Amazon JDBC Connector for Apache Hive, where *[Subprotocol]* is **hive**, **hive2** if you are connecting to a Hive Server 2 instance, *[Host]* is the DNS or IP address of the Hive server, and *[Port]* is the number of the TCP port that the server uses to listen for client requests:

```
jdbc:[Subprotocol]://[Host]:[Port]
```

Note:

By default, Hive uses port 10000.

By default, the connector uses the schema named **default** and authenticates the connection using the user name **anonymous**.

You can specify optional settings such as the number of the schema to use or any of the connection properties supported by the connector. For a list of the properties available in the connector, see [Connector Configuration Options](#) on page 51.

Note:

If you specify a property that is not supported by the connector, then the connector attempts to apply the property as a Hive server-side property for the client session. For more information, see [Configuring Server-Side Properties](#) on page 33.

The following is the format of a connection URL that specifies some optional settings:

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

For example, to connect to port 11000 on a Hive Server 2 instance installed on the local machine, use a schema named **default2**, and authenticate the connection using a user name and password, you would use the following connection URL:

```
jdbc:hive2://localhost:11000/default2;AuthMech=3;UID=amazon;PWD=amazon
```

⚠ Important:

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

ℹ Note:

- If you specify a schema in the connection URL, you can still issue queries on other schemas by explicitly specifying the schema in the query. To inspect your databases and determine the appropriate schema to use, run the `show databases` command at the Hive command prompt.
- If you set the `transportMode` property to `http`, then the port number specified in the connection URL corresponds to the HTTP port rather than the TCP port. By default, Hive servers use 10001 as the HTTP port number.

Configuring Authentication

The Amazon JDBC Connector for Apache Hive supports the following authentication mechanisms:

- No Authentication
- Kerberos
- User Name
- User Name And Password
- Single Sign-On (SSO)
- Hadoop Delegation Token

You configure the authentication mechanism that the connector uses to connect to Hive by specifying the relevant properties in the connection URL.

For information about the properties you can use in the connection URL, see [Connector Configuration Options](#) on page 51.

Note:

In addition to authentication, you can configure the connector to connect over SSL. For more information, see [Configuring SSL](#) on page 30.

Using No Authentication

You provide this information to the connector in the connection URL. For more information about the syntax of the connection URL, see [Building the Connection URL](#) on page 11.

To configure a connection without authentication:

1. Set the `AuthMech` property to 0.
2. Set the `transportMode` property to `binary`.

For example:

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

Using Kerberos

Kerberos must be installed and configured before you can use this authentication mechanism. For information about configuring and operating Kerberos on Windows, see [Configuring Kerberos Authentication for Windows](#) on page 22. For other operating

systems, see the MIT Kerberos documentation: <http://web.mit.edu/kerberos/krb5-latest/doc/>.

You provide this information to the connector in the connection URL. For more information about the syntax of the connection URL, see [Building the Connection URL](#) on page 11.

Note:

- This authentication mechanism is available only for Hive Server 2.
- For this authentication mechanism, SASL and HTTP Thrift transport protocols are supported. If the `transportMode` property is not set, the connector defaults SASL. If the `hive.server2.transport.mode` property has been set to HTTP on the server side, set the `transportMode` property to `http`.

To configure default Kerberos authentication:

1. Set the `AuthMech` property to 1.
2. To use the default realm defined in your Kerberos setup, do not set the `KrbRealm` property.

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 `KrbRealm` property to the realm of the Hive server.

3. Set the `KrbHostFQDN` property to the fully qualified domain name of the Hive server host.
4. Optionally, specify how the connector obtains the Kerberos Subject by setting the `KrbAuthType` property as follows:
 - To configure the connector to automatically detect which method to use for obtaining the Subject, set the `KrbAuthType` property to 0. Alternatively, do not set the `KrbAuthType` property.
 - Or, to create a `LoginContext` from a JAAS configuration and then use the Subject associated with it, set the `KrbAuthType` property to 1.
 - Or, to create a `LoginContext` from a Kerberos ticket cache and then use the Subject associated with it, set the `KrbAuthType` property to 2.

For more detailed information about how the connector obtains Kerberos Subjects based on these settings, see [KrbAuthType](#) on page 56.

For example, the following connection URL connects to a Hive server with Kerberos enabled, but without SSL enabled:

```
jdbc:hive2://node1.example.com:10000;AuthMech=1;  
KrbRealm=EXAMPLE.COM;KrbHostFQDN=hs2node1.example.com;  
KrbServiceName=hive;KrbAuthType=2
```

In this example, Kerberos is enabled for JDBC connections, the Kerberos service principal name is `hive/node1.example.com@EXAMPLE.COM`, the host name for the data source is `node1.example.com`, and the server is listening on port 10000 for JDBC connections.

Using Single Sign-On

Single Sign-On (SSO) is a process that allows network users to access all authorized network resources without having to log in to each resource separately. For example, implementing SSO for users within an organization allows each user to authenticate to Hive without providing a separate set of Hive credentials.

You specify the properties in the connection URL. For more information about the syntax of the connection URL, see [Building the Connection URL](#) on page 11.

Important:

SSL is required for this authentication method. For more information, see [Configuring SSL](#) on page 30.

To configure Single Sign-On authentication:

1. Set the `AuthMech` property to 12.
2. Set the `TransportMode` property to `http`.
3. Optionally, set the `SSOWebServerTimeout` property to the number of seconds that the connector waits before timing out while waiting for a browser response.

For example:

```
jdbc:hive2://node1.example.com:10001;AuthMech=12;  
SSL=1;TransportMode=http;httpPath=cliservice;SSOWebServerTime  
out=60;
```

Using User Name

This authentication mechanism requires a user name but does not require a password. The user name labels the session, facilitating database tracking.

You provide this information to the connector in the connection URL. For more information about the syntax of the connection URL, see [Building the Connection URL](#) on page 11.

Note:

This authentication mechanism is available only for Hive Server 2. Most default configurations of Hive Server 2 require User Name authentication.

To configure User Name authentication:

1. Set the `AuthMech` property to 2.
2. Set the `transportMode` property to `sasl`.
3. Set the `UID` property to an appropriate user name for accessing the Hive server.

For example:

```
jdbc:hive2://node1.example.com:10000;AuthMech=2;  
transportMode=sasl;UID=hs2
```

Using User Name And Password (LDAP)

This authentication mechanism requires a user name and a password. It is most commonly used with LDAP authentication.

You provide this information to the connector in the connection URL. For more information about the syntax of the connection URL, see [Building the Connection URL](#) on page 11.

Note:

This authentication mechanism is available only for Hive Server 2.

To configure User Name And Password authentication:

1. Set the `AuthMech` property to 3.
2. Set the `transportMode` property to the transport protocol that you want to use in the Thrift layer.
3. If you set the `transportMode` property to `http`, then set the `httpPath` property to the partial URL corresponding to the Hive server. Otherwise, do not set the `httpPath` property.
4. Set the `UID` property to an appropriate user name for accessing the Hive server.

5. Set the `PWD` property to the password corresponding to the user name you provided.

For example, the following connection URL connects to a Hive server with LDAP authentication enabled:

```
jdbc:hive2://node1.example.com:10001;AuthMech=3;  
transportMode=http;httpPath=cliservice;UID=hs2;PWD=amazon;
```

In this example, user name and password (LDAP) authentication is enabled for JDBC connections, the LDAP user name is `hs2`, the password is `amazon`, and the server is listening on port 10001 for JDBC connections.

Using a Hadoop Delegation Token

This authentication mechanism requires a Hadoop delegation token. This token must be provided to the connector in the form of a Base64 URL-safe encoded string. It can be obtained from the connector using the `getDelegationToken()` function, or by utilizing the Hadoop distribution `.jar` files. For a code sample that demonstrates how to retrieve the token using the `getDelegationToken()` function, see [Code Samples: Retrieving a Hadoop Delegation Token](#) on page 18.

You provide this information to the connector in the connection URL. For more information about the syntax of the connection URL, see [Building the Connection URL](#) on page 11.

Note:

- This authentication mechanism is available only for Hive Server 2.
- This authentication mechanism requires that Kerberos be configured on the server.

To configure Hadoop delegation token authentication:

1. Make sure Kerberos is configured on the server.
2. Set the `AuthMech` property to 6.
3. Set the `delegationToken` property to an appropriately encoded Hadoop delegation token.

For example:

```
jdbc:  
hive  
2
```

```
://node1.example.com:  
10000;AuthMech=6;delegationToken=kP9PcyQ7prK2LwUMZMpFQ4R+5VE
```

Code Samples: Retrieving a Hadoop Delegation Token

If you are using a Hadoop delegation token for authentication, the token must be provided to the connector in the form of a Base64 URL-safe encoded string. This token can be obtained from the connector using the `getDelegationToken()` function, or by utilizing the Hadoop distribution `.jar` files.

The code samples below demonstrate the use of the `getDelegationToken()` function. For more information about this function, see [IHadoopConnection](#) on page 46.

The sample below shows how to obtain the token string with the connector using a Kerberos connection:

```
import  
com.amazon.hiveserver2.hivecommon.core.IHadoopConnection;  
public class TestDriverGetDelegationTokenClass  
{  
  
    public static void main(String[] args) throws  
        SQLException  
    {  
  
        // Create the connection object with Kerberos  
        authentication.  
        Connection kerbConnection =  
            DriverManager.getConnection(  
  
                "jdbc:hive2://localhost:10000;AuthMe  
                ch=1;KrbRealm=YourRealm;KrbHostFQDN=  
                sample.com;KrbServiceName=hive;");  
  
        // Unwrap the java.sql.Connection object to  
        an implementation of IHadoopConnection so the  
        // methods for delegation token can be  
        called.  
        String delegationToken =  
            kerbConnection.unwrap  
            (IHadoopConnection.class).getDelegationToken  
            ("owner_name", "renewer_name");  
    }  
}
```

```
        // The token can then be used to connect with
        the connector.
        String tokenConnectionString =
        "jdbc:hive2://localhost:10000;AuthMech=6;Dele
        gationToken=" + delegationToken;
        Connection tokenConnection =
        DriverManager.getConnection
        (tokenConnectionString);

    }

}
```

The sample below demonstrates how to obtain the encoded string form of the token if the delegation is saved to the UserGroupInformation. This sample requires the `hadoop-shims-common-[hadoop version].jar`, `hadoop-common-[hadoop version].jar`, and all their dependencies.

```
import org.apache.hadoop.hive.shims.Utills;
import org.apache.hive.service.auth.HiveAuthFactory;
public class TestHadoopDelegationTokenClass
{

    public static void main(String[] args) throws
    SQLException
    {

        // Obtain the delegationToken stored in the
        current UserGroupInformation.
        String delegationToken =
        Utills.getTokenStrForm(HiveAuthFactory.HS2_
        CLIENT_TOKEN);

        // The token can then be used to connect with
        the connector.
        String tokenConnectionString =
        "jdbc:hive2://localhost:10000;AuthMech=6;Dele
        gationToken=" + delegationToken;
        Connection tokenConnection =
        DriverManager.getConnection
        (tokenConnectionString);

    }

}
```

```
}  
  
}
```

Authentication Mechanisms

To connect to a Hive server, you must configure the Amazon JDBC Connector for Apache Hive to use the authentication mechanism that matches the access requirements of the server and provides the necessary credentials. To determine the authentication settings that your Hive server requires, check the server configuration and then refer to the corresponding section below.

Hive Server 2

Hive Server 2 supports the following authentication mechanisms:

- No Authentication (see [Using No Authentication](#) on page 13)
- JWT (see [Using JSON Web Token \(JWT\)](#))
- Kerberos (see [Using Kerberos](#) on page 13)
- User Name (see [Using User Name](#) on page 15)
- User Name And Password (see [Using User Name And Password \(LDAP\)](#) on page 16)
- Hadoop Delegation Token (see [Using a Hadoop Delegation Token](#) on page 17)

Most default configurations of Hive Server 2 require User Name authentication. If you are unable to connect to your Hive server using User Name authentication, then verify the authentication mechanism configured for your Hive server by examining the `hive-site.xml` file. Examine the following properties to determine which authentication mechanism your server is set to use:

- `hive.server2.authentication`: This property sets the authentication mode for Hive Server 2. The following values are available:
 - `NONE` enables plain SASL transport. This is the default value.
 - `NOSASL` disables the Simple Authentication and Security Layer (SASL).
 - `KERBEROS` enables Kerberos authentication and delegation token authentication.
 - `PLAINSASL` enables user name and password authentication using a cleartext password mechanism.
 - `LDAP` enables user name and password authentication using the Lightweight Directory Access Protocol (LDAP).
- `hive.server2.enable.doAs`: If this property is set to the default value of `TRUE`, then Hive processes queries as the user who submitted the query. If this

property is set to `FALSE`, then queries are run as the user that runs the `hiveserver2` process.

The following table lists the authentication mechanisms to configure for the connector based on the settings in the `hive-site.xml` file.

<code>hive.server2.authentication</code>	<code>hive.server2.enable.doAs</code>	Connector Authentication Mechanism
NOSASL	FALSE	No Authentication
KERBEROS	TRUE or FALSE	Kerberos
KERBEROS	TRUE	Delegation Token
NONE	TRUE or FALSE	User Name
PLAINSASL or LDAP	TRUE or FALSE	User Name And Password

Note:

It is an error to set `hive.server2.authentication` to `NOSASL` and `hive.server2.enable.doAs` to `true`. This configuration will not prevent the service from starting up, but results in an unusable service.

For more information about authentication mechanisms, refer to the documentation for your Hadoop / Hive distribution. See also "Running Hadoop in Secure Mode" in the Apache Hadoop documentation: http://hadoop.apache.org/docs/r0.23.7/hadoop-project-dist/hadoop-common/ClusterSetup.html#Running_Hadoop_in_Secure_Mode.

Using No Authentication

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

Using Kerberos

When connecting to a Hive Server 2 instance and `hive.server2.authentication` is set to `KERBEROS`, you must configure your connection to use Kerberos or Delegation Token authentication.

Using User Name

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

- If `hive.server2.enable.doAs` is set to `TRUE`, then the server attempts to map the user name provided by the connector from the connector configuration to an existing operating system user on the host running Hive Server 2. If this user name does not exist in the operating system, then the user group lookup fails and existing HDFS permissions are used. For example, if the current user group is allowed to read and write to the location in HDFS, then read and write queries are allowed.
- If `hive.server2.enable.doAs` is set to `FALSE`, then the user name in the connector configuration is ignored.

If no user name is specified in the connector configuration, then the connector defaults to using `hive` as the user name.

Using User Name And Password

When connecting to a Hive Server 2 instance and the server is configured to use the SASL-PLAIN authentication mechanism with a user name and a password, you must configure your connection to use User Name And Password authentication.

Configuring Kerberos Authentication for Windows

You can configure your Kerberos setup so that you use the MIT Kerberos Ticket Manager to get the Ticket Granting Ticket (TGT), or configure the setup so that you can use the connector to get the ticket directly from the Key Distribution Center (KDC). Also, if a client application obtains a Subject with a TGT, it is possible to use that Subject to authenticate the connection.

Downloading and Installing MIT Kerberos for Windows

To download and install MIT Kerberos for Windows 4.0.1:

1. Download the appropriate Kerberos installer:
 - For a 64-bit machine, use the following download link from the MIT Kerberos website: <http://web.mit.edu/kerberos/dist/kfw/4.0/kfw-4.0.1-amd64.msi>.
 - For a 32-bit machine, use the following download link from the MIT Kerberos website: <http://web.mit.edu/kerberos/dist/kfw/4.0/kfw-4.0.1-i386.msi>.

Note:

The 64-bit installer includes both 32-bit and 64-bit libraries. The 32-bit installer includes 32-bit libraries only.


2. To run the installer, double-click the `.msi` file that you downloaded.
3. Follow the instructions in the installer to complete the installation process.
4. When the installation completes, click **Finish**.

Using the MIT Kerberos Ticket Manager to Get Tickets

Setting the KRB5CCNAME Environment Variable


You must set the KRB5CCNAME environment variable to your credential cache file.

To set the KRB5CCNAME environment variable:

1. Click **Start** , then right-click **Computer**, and then click **Properties**.
2. Click **Advanced System Settings**.
3. In the System Properties dialog box, on the **Advanced** tab, click **Environment Variables**.
4. In the Environment Variables dialog box, under the System Variables list, click **New**.
5. In the **New System Variable** dialog box, in the Variable Name field, type **KRB5CCNAME**.
6. In the **Variable Value** field, type the path for your credential cache file. For example, type `C:\KerberosTickets.txt`.
7. Click **OK** to save the new variable.
8. Make sure that the variable appears in the System Variables list.
9. Click **OK** to close the Environment Variables dialog box, and then click **OK** to close the System Properties dialog box.
10. Restart your machine.

Getting a Kerberos Ticket

To get a Kerberos ticket:

1. Click **Start** , then click **All Programs**, and then click the **Kerberos for Windows (64-bit)** or **Kerberos for Windows (32-bit)** program group.
2. Click **MIT Kerberos Ticket Manager**.
3. In the MIT Kerberos Ticket Manager, click **Get Ticket**.

4. In the Get Ticket dialog box, type your principal name and password, and then click **OK**.

If the authentication succeeds, then your ticket information appears in the MIT Kerberos Ticket Manager.

Authenticating to the Hive Server

You provide this information to the connector in the connection URL. For more information about the syntax of the connection URL, see [Building the Connection URL](#) on page 11.

To authenticate to the Hive server:

- Use a connection URL that has the following properties defined:
 - `AuthMech`
 - `KrbHostFQDN`
 - `KrbRealm`
 - `KrbServiceName`


For detailed information about these properties, see [Connector Configuration Options](#) on page 51

Using the Connector to Get Tickets

Deleting the KRB5CCNAME Environment Variable

To enable the connector to get Ticket Granting Tickets (TGTs) directly, make sure that the KRB5CCNAME environment variable has not been set.

To delete the KRB5CCNAME environment variable:

1. Click the **Start** button , then right-click **Computer**, and then click **Properties**.
2. Click **Advanced System Settings**.
3. In the System Properties dialog box, click the **Advanced** tab and then click **Environment Variables**.
4. In the Environment Variables dialog box, check if the KRB5CCNAME variable appears in the System variables list. If the variable appears in the list, then select the variable and click **Delete**.
5. Click **OK** to close the Environment Variables dialog box, and then click **OK** to close the System Properties dialog box.

Setting Up the Kerberos Configuration File

To set up the Kerberos configuration file:

1. Create a standard `krb5.ini` file and place it in the `C:\Windows` directory.
2. Make sure that the KDC and Admin server specified in the `krb5.ini` file can be resolved from your terminal. If necessary, modify `C:\Windows\System32\drivers\etc\hosts`.

Setting Up the JAAS Login Configuration File

To set up the JAAS login configuration file:

1. Create a JAAS login configuration file that specifies a keytab file and `doNotPrompt=true`.

For example:

```
Client {  
  com.sun.security.auth.module.Krb5LoginModule required  
  useKeyTab=true  
  keyTab="PathToTheKeyTab"  
  principal="amazon@AMAZON"  
  doNotPrompt=true;  
};
```

2. Set the `java.security.auth.login.config` system property to the location of the JAAS file.

For example: `C:\KerberosLoginConfig.ini`.

Note:

JAAS configuration is disabled by default. To enable JAAS configuration, please set the `JDBC_ENABLE_JAAS` environment variable to 1.

Authenticating to the Hive Server

You provide this information to the connector in the connection URL. For more information about the syntax of the connection URL, see [Building the Connection URL](#) on page 11.

To authenticate to the Hive server:

➤ Use a connection URL that has the following properties defined:

- AuthMech
- KrbHostFQDN
- KrbRealm
- KrbServiceName

For detailed information about these properties, see [Connector Configuration Options](#) on page 51.

Using an Existing Subject to Authenticate the Connection

If the client application obtains a Subject with a TGT, then that Subject can be used to authenticate the connection to the server.

To use an existing Subject to authenticate the connection:

1. Create a PrivilegedAction for establishing the connection to the database.

For example:

```
// Contains logic to be executed as a privileged action
public class AuthenticateDriverAction
implements PrivilegedAction<Void>
{
    // The connection, which is established as a
    PrivilegedAction
    Connection con;
    // Define a string as the connection URL
    static String ConnectionURL =
    "jdbc:hive2://192.168.1.1:10000";
    /**
     * Logic executed in this method will have access to the
     * Subject that is used to "doAs". The connector will get
     * the Subject and use it for establishing a connection
     * with the server.
     */
    @Override
    public Void run()
    {
        try
        {
```

```
// Establish a connection using the connection URL
con = DriverManager.getConnection(ConnectionURL);
}
catch (SQLException e)
{
    // Handle errors that are encountered during
    // interaction with the data store
    e.printStackTrace();
}
catch (Exception e)
{
    // Handle other errors
    e.printStackTrace();
}
return null;
}
}
```

2. Run the PrivilegedAction using the existing Subject, and then use the connection.

For example:

```
// Create the action
AuthenticateDriverAction authenticateAction = new
AuthenticateDriverAction();
// Establish the connection using the Subject for
// authentication.
Subject.doAs(loginConfig.getSubject(),
authenticateAction);
// Use the established connection.
authenticateAction.con;
```

Kerberos Encryption Strength and the JCE Policy Files Extension

If the encryption being used in your Kerberos environment is too strong, you might encounter the error message "Unable to connect to server: GSS initiate failed" when trying to use the connector to connect to a Kerberos-enabled cluster. Typically, Java vendors only allow encryption strength up to 128 bits by default. If you are using greater encryption strength in your environment (for example, 256-bit encryption), then you might encounter this error.

Diagnosing the Issue

If you encounter the error message "Unable to connect to server: GSS initiate failed", confirm that it is occurring due to encryption strength by enabling Kerberos layer logging in the JVM and then checking if the log output contains the error message "KrbException: Illegal key size".

To enable Kerberos layer logging in a Sun JVM:

➤ Choose one:

- In the Java command you use to start the application, pass in the following argument:

```
-Dsun.security.krb5.debug=true
```

- Or, add the following code to the source code of your application:

```
System.setProperty  
("sun.security.krb5.debug", "true")
```

To enable Kerberos layer logging in an IBM JVM:

➤ Choose one:

- In the Java command you use to start the application, pass in the following arguments:

```
-Dcom.ibm.security.krb5.Krb5Debug=all  
-Dcom.ibm.security.jgss.debug=all
```

- Or, add the following code to the source code of your application:

```
System.setProperty  
("com.ibm.security.krb5.Krb5Debug", "all");  
System.setProperty  
("com.ibm.security.jgss.debug", "all");
```

Resolving the Issue

After you confirm that the error is occurring due to encryption strength, you can resolve the issue by downloading and installing the *Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files* extension from your Java vendor. Refer to the instructions from the vendor to install the files to the correct location.

 Important:

Consult your company's policy to make sure that you are allowed to enable encryption strengths in your environment that are greater than what the JVM allows by default.

If the issue is not resolved after you install the JCE policy files extension, then restart your machine and try your connection again. If the issue persists even after you restart your machine, then verify which directories the JVM is searching to find the JCE policy files extension. To print out the search paths that your JVM currently uses to find the JCE policy files extension, modify your Java source code to print the return value of the following call:

```
System.getProperty("java.ext.dirs")
```

Configuring SSL

Note:

In this documentation, "SSL" indicates both TLS (Transport Layer Security) and SSL (Secure Sockets Layer). The connector supports industry-standard versions of TLS/SSL.

If you are connecting to a Hive server that has Secure Sockets Layer (SSL) enabled, you can configure the connector to connect to an SSL-enabled socket. When connecting to a server over SSL, the connector uses one-way authentication to verify the identity of the server.

One-way authentication requires a signed, trusted SSL certificate for verifying the identity of the server. You can configure the connector to access a specific TrustStore or KeyStore that contains the appropriate certificate. If you do not specify a TrustStore or KeyStore, then the connector uses the default Java TrustStore named `jssecacerts`. If `jssecacerts` is not available, then the connector uses `cacerts` instead.

You provide this information to the connector in the connection URL. For more information about the syntax of the connection URL, see [Building the Connection URL](#) on page 11.

To configure SSL:

1. Set the `SSL` property to 1.
2. If you are not using one of the default Java TrustStores, then do one of the following:
 - Create a TrustStore and configure the connector to use it:
 - a. Create a TrustStore containing your signed, trusted server certificate.
 - b. Set the `SSLTrustStore` property to the full path of the TrustStore.
 - c. Set the `SSLTrustStorePwd` property to the password for accessing the TrustStore.
 - d. If the TrustStore is not a JKS TrustStore, set the `SSLTrustStoreType` property to the correct type. The supported types are:
 - i. `SSLTrustStoreType=BCFKS` (BouncyCastle FIPS Keystore)

- ii. `SSLTrustStoreType=PKCS12` (Public Key Cryptography Standards #12)

Note:

`SSLTrustStoreType=PKCS11` (Public Key Cryptography Standards #11) TrustStore type is not supported.

- e. To specify a Java Security API provider, set the `SSLTrustStoreProvider` property to the name of the provider.
- Or, create a KeyStore and configure the connector to use it:
 - a. Create a KeyStore containing your signed, trusted server certificate.
 - b. Set the `SSLKeyStore` property to the full path of the KeyStore.
 - c. Set the `SSLKeyStorePwd` property to the password for accessing the KeyStore.
 - d. If the KeyStore is not a JKS KeyStore, set the `SSLKeyStoreType` property to the correct type.
 - e. To specify a Java Security API provider, set the `SSLKeyStoreProvider` property to the name of the provider.
- 3. Optionally, to allow the SSL certificate used by the server to be self-signed, set the `AllowSelfSignedCerts` property to 1.

Important:

When the `AllowSelfSignedCerts` property is set to 1, SSL verification is disabled. The connector does not verify the server certificate against the trust store, and does not verify if the server's host name matches the common name or subject alternative names in the server certificate.

- 4. Optionally, to allow the common name of a CA-issued certificate to not match the host name of the Hive server, set the `CAIssuedCertNamesMismatch` property to 1.

For example, the following connection URL connects to a data source using username and password (LDAP) authentication, with SSL enabled:

```
jdbc:hive2://localhost:10000;AuthMech=3;SSL=1;  
SSLKeyStore=C:\\Users\\bsmith\\Desktop\\keystore.jks;SSLKeySt  
orePwd=amazonSSL123;UID=hs2;PWD=amazon123
```

i Note:

For more information about the connection properties used in SSL connections, see [Connector Configuration Options](#) on page 51.

Configuring Server-Side Properties

You can use the connector to apply configuration properties to the Hive server by setting the properties in the connection URL.

For example, to set the `mapreduce.job.queueName` property to `myQueue`, you would use a connection URL such as the following:

```
jdbc:hive://localhost:18000/default2;AuthMech=3;  
UID=amazon;PWD=amazon;mapreduce.job.queueName=myQueue
```

Note:

For a list of all Hadoop and Hive server-side properties that your implementation supports, run the `set -v` command at the Hive CLI command line or Beeline. You can also execute the `set -v` query after connecting using the connector.

Configuring Logging

To help troubleshoot issues, you can enable logging in the connector.

Important:

Only enable logging long enough to capture an issue. Logging decreases performance and can consume a large quantity of disk space.

The settings for logging apply to every connection that uses the Amazon JDBC Connector for Apache Hive, so make sure to disable the feature after you are done using it.

In the connection URL, set the `LogLevel` key to enable logging at the desired level of detail. The following table lists the logging levels provided by the Amazon JDBC Connector for Apache Hive, in order from least verbose to most verbose.

LogLevel Value	Description
0	Disable all logging.
1	Log severe error events that lead the connector to abort.
2	Log error events that might allow the connector to continue running.
3	Log events that might result in an error if action is not taken.
4	Log general information that describes the progress of the connector.
5	Log detailed information that is useful for debugging the connector.
6	Log all connector activity.

To enable logging:

1. Set the `LogLevel` property to the desired level of information to include in log files.

2. Set the `LogPath` property to the full path to the folder where you want to save log files. To make sure that the connection URL is compatible with all JDBC applications, escape the backslashes (`\`) in your file path by typing another backslash.

For example, the following connection URL enables logging level 3 and saves the log files in the `C:\temp` folder:

```
jdbc:hive://localhost:11000;LogLevel=3;LogPath=C:\\temp
```

3. To make sure that the new settings take effect, restart your JDBC application and reconnect to the server.

The Amazon JDBC Connector for Apache Hive produces the following log files in the location specified in the `LogPath` property:

- A `HiveJDBC_driver.log` file that logs connector activity that is not specific to a connection.
- A `HiveJDBC_connection_[Number].log` file for each connection made to the database, where *[Number]* is a number that identifies each log file. This file logs connector activity that is specific to the connection.

If the `LogPath` value is invalid, then the connector sends the logged information to the standard output stream (`System.out`).

To disable logging:

1. Set the `LogLevel` property to 0.
2. To make sure that the new setting takes effect, restart your JDBC application and reconnect to the server.

Features

More information is provided on the following features of the Amazon JDBC Connector for Apache Hive:

- [SQL Query versus HiveQL Query](#) on page 36
- [Data Types](#) on page 36
- [Catalog and Schema Support](#) on page 37
- [Write-back](#) on page 38
- [IHadoopStatement](#) on page 38
- [IHadoopConnection](#) on page 46
- [Security and Authentication](#) on page 49

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 Amazon JDBC Connector for Apache Hive supports many common data formats, converting between Hive, SQL, and Java data types.

The following table lists the supported data type mappings.

Hive Type	SQL Type	Java Type
BIGINT	BIGINT	java.math.BigInteger
BINARY	VARBINARY	byte[]
BOOLEAN	BOOLEAN	Boolean
CHAR (Available only in Hive 0.13.0 or later)	CHAR	String
DATE	DATE	java.sql.Date

Hive Type	SQL Type	Java Type
DECIMAL (In Hive 0.13 and later, you can specify scale and precision when creating tables using the DECIMAL data type.)	DECIMAL	java.math.BigDecimal
DOUBLE	DOUBLE	Double
FLOAT	REAL	Float
INT	INTEGER	Long
SMALLINT	SMALLINT	Integer
TIMESTAMP	TIMESTAMP	java.sql.Timestamp
TINYINT	TINYINT	Short
VARCHAR (Available only in Hive 0.12.0 or later)	VARCHAR	String

The aggregate types (ARRAY, MAP, STRUCT, and UNIONTYPE) are not yet supported. Columns of aggregate types are treated as VARCHAR columns in SQL and STRING columns in Java.

Catalog and Schema Support

The Amazon JDBC Connector for Apache Hive supports both catalogs and schemas to make it easy for the connector to work with various JDBC applications. Since Hive only organizes tables into schemas/databases, the connector provides a synthetic catalog named HIVE under which all of the schemas/databases are organized. The connector also maps the JDBC schema to the Hive schema/database.

Note:

Setting the `CatalogSchemaSwitch` connection property to 1 will cause Hive catalogs to be treated as schemas in the connector as a restriction for filtering.

Write-back

The Amazon JDBC Connector for Apache Hive supports translation for the following syntax when connecting to a Hive Server 2 instance that is running Hive 0.14 or later:

- INSERT
- UPDATE
- DELETE
- CREATE
- DROP

If the statement contains non-standard SQL-92 syntax, then the connector is unable to translate the statement to SQL and instead falls back to using HiveQL.

IHadoopStatement

IHadoopStatement is an interface implemented by the connector's statement class. It provides access to methods that allow for asynchronous execution of queries and the retrieval of the Yarn ATS GUID associated with the execution.

The IHadoopStatement interface is defined by the `IHadoopStatement.java` file. This file should look like the following example:

```
//
=====
=====
/// @file IHadoopStatement.java /// /// Exposed interface for
asynchronous query execution. /// /// Copyright (C) 2017
Simba Technologies Incorporated.
//
=====
=====
package com.amazon.hiveserver2.hivecommon.core;
import java.sql.ResultSet; import java.sql.SQLException;
import java.sql.Statement;
/**
 * An interface that extends the standard SQL Statement
 * Interface but allows for asynchronous
 * query execution.
 * The polling for query execution will occur when {@link
 * ResultSet#next()} or
 * {@link ResultSet#getMetaData()} is called.
 */ public interface IHadoopStatement extends Statement
```

```

{
    /**
     * Executes the given SQL statement
     * asynchronously.
     * <p> * Sends the query off to the server but
     * does not wait for query execution to
     * complete.
     * A ResultSet with empty columns is returned.
     * The polling for completion of query
     * execution is done when {@link ResultSet#next
     * ()} or
     * {@link ResultSet#getMetaData()} is called.
     * </p>
     *
     * @param sql
     * An SQL statement to be sent to the database,
     * typically a
     * static SQL SELECT statement.
     *
     * @return A ResultSet object that DOES NOT
     * contain the data produced by the given query;
     * never null.
     *
     * @throws SQLException If a database access
     * error occurs, or the given SQL
     * statement produces anything other than a
     * single
     * <code>ResultSet</code> object.
     */
    public ResultSet executeAsync(String sql) throws
    SQLException;
    /**
     *
     * Returns the Yarn ATS guid.
     *
     * @return String The yarn ATS guid from the
     * operation if execution has started,
     * else null.
     */
}

```

```
*/  
  
public String getYarnATSGuid(); }
```

The following methods are available for use in `IHadoopStatement`:

- `executeAsync(String sql)`

The connector sends a request to the server for statement execution and returns immediately after receiving a response from the server for the execute request without waiting for the server to complete the execution.

The connector does not wait for the server to complete query execution unless `getMetaData()` or `next()` APIs are called.

Note that this feature does not work with prepared statements.

For example:

```
import  
com.amazon.hiveserver2.hivecommon.core.IHadoopStatement;  
  
public class TestExecuteAsyncClass  
{  
  
    public static void main(String[] args) throws  
        SQLException  
    {  
  
        // Create the connection object.  
        Connection connection =  
            DriverManager.getConnection  
                ("jdbc:hive2://localhost:10000");  
  
        // Create the statement object.  
        Statement statement =  
            connection.createStatement();  
  
        // Unwrap the java.sql.Statement object to an  
        implementation of IHadoopStatement so the  
        // execution can be done asynchronously.  
  
        //
```

```
        // The connector will return from
        this call as soon as it gets a
        response from the
        // server for the execute request
        without waiting for server to
        complete query execution.

        ResultSet resultSet =

            statement.unwrap(

                IHadoopStatement.class
                ss).executeAsync(

                    "select * from example_
                    table");

        // Calling getMetaData() on the ResultSet
        here will cause the connector to wait for the
        server
        // to complete query execution before
        proceeding with the rest of the operation.
        ResultSetMetaData rsMetadata =
        resultSet.getMetaData();

        // Excluding code for work on the result set
        metadata...

        // Calling getMetaData() on the ResultSet
        here, and if getMetaData() was not call prior
        to
        // this, will cause the connector to wait for
        the server to complete query execution before
        // proceeding with the rest of the operation.
        resultSet.next();

        // Excluding code for work on the result set
        ...

    }

}
```

- `getYarnATSGuid()`

Returns the Yarn ATS GUID associated with the current execution. Returns null if the Yarn ATS GUID is not available.

For example:

```
public class TestYarnGUIDClass
{
    public static void main(String[] args) throws
        SQLException
    {
        // Create the connection object.
        Connection connection =
            DriverManager.getConnection
            ("jdbc:hive2://localhost:10000");

        // Create the statement object.
        Statement statement =
            connection.createStatement();

        // Execute a query.
        ResultSet resultSet = statement.executeQuery
            ("select * from example_table");

        // Unwrap the java.sql.Statement
        // object to an implementation of
        // IHadoopStatement to access the
        // getYarnATSGuid() API call.
        String guid = statement.unwrap(
            IHadoopStatement.class).getYarnATSGuid
            ();
    }
}
```

The connector can retrieve the query logs from the server when executing a SQL statement using the following functions:

```
getQueryLog(boolean incremental,int fetchSize)
```

The connector gets the execution logs of the given SQL statement.

```
hasMoreLogs() ;
```

The connector checks whether the query execution produces more logs to be fetched.

```
/**
 * Get the execution logs of the given SQL statement.
 *
 * @param incremental      True to indicate to get
 *                          logs incrementally,
 *                          otherwise false to indicate
 *                          to get logs from the beginning,
 * @param fetchSize        The number of lines to
 *                          fetch
 *
 * @return A list of logs. It can be empty if there
 *         are no new logs to be retrieved at that time.
 * @throws RuntimeException if an error occurs fetching
 *         logs from the server
 */
public List<String> getQueryLog(boolean incremental,
int fetchSize) throws RuntimeException;

/**
 * Check whether query execution produces more logs to
 * be fetched.
 *
 * @return true if the query execution produces more
 *         logs, false otherwise
 */
public boolean hasMoreLogs();
```

For example:

```
public class TestGetQueryLog
{
```

```
public static void main(String[] args) throws
SQLException
{

    // Create the connection object.
    Connection connection =
    DriverManager.getConnection
    ("jdbc:hive2://localhost:10000");

    // Create statement object.
    Statement statement =
    connection.createStatement();

    try
    {

        // Create and start the class that
        gets the query logs in the
        background
        // while the query is running.
        HiveLogRunnable runnable = new
        HiveLogRunnable(statement);
        Thread t1 = new Thread(runnable);
        t1.setDaemon(true);
        t1.start();

        // Execute the query.
        ResultSet resultSet =
        statement.executeQuery("select *
        from example_table");
        while (resultSet.next()) {}
    }

    finally
    {

        // Close the statement.
        if (statement != null)
        {

            statement.close();
        }
    }
}
```

```
    }  
  }  
}  
  
class HiveLogRunnable implements Runnable {  
    private Statement stmt = null;  
    public HiveLogRunnable(Statement  
        stmt) {  
        this.stmt = stmt;  
    }  
    @Override  
    public void run() {  
        try {  
            // Print the query  
            logs to the console while  
            the query has logs.  
            while (stmt.unwrap  
                (IHadoopStatement.class).has  
                MoreLogs()) {  
                List<String>  
                logLists = stmt.unwrap  
                    (IHadoopStatement.class).get  
                    QueryLog(true, 0);  
                for (String string  
                    : logLists) {  
                    System.out.pri  
ntln(string);  
                }  
                Thread.sleep(0);  
            }  
        }  
    }  
}
```

```

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```

IHadoopConnection

IHadoopConnection is an interface implemented by the connector's statement class. It provides access to methods that allow for the retrieval, deletion, and renewal of delegation tokens.

The IHadoopStatement interface is defined by the `IHadoopStatement.java` file. This file should look like the following example:

```

//
=====
package com.amazon.hiveserver2.hivecommon.core;
import java.sql.Connection;
import java.sql.SQLException;
/**
 * An interface that extends the standard SQL Connection
 * Interface but allows for the
 * retrieval/renewal/cancellation of delegation tokens.
 */
public interface IHadoopConnection extends Connection
{
    /**
     * Sends a cancel delegation token request to
     * the server.
     */
}

```

```
    * @param tokenString The token to cancel.
    * @throws SQLException If an error occurs
    while sending the request.
    */
    public void cancelDelegationToken(String
    tokenString) throws SQLException;

    /**
    * Sends a get delegation token request to the
    server and returns the token as an
    * encoded string.
    *
    * @param owner The owner of the token.
    * @param renewer The renewer of the token.
    *
    * @return The token as an encoded string.
    * @throws SQLException If an error occurs
    while getting the token.
    */
    public String getDelegationToken(String
    owner, String renewer) throws SQLException;

    /**
    * Sends a renew delegation token request to
    the sever.
    *
    * @param tokenString The token to renew.
    * @throws SQLException If an error occurs
    while sending the request.
    */
    public void renewDelegationToken(String
    tokenString) throws SQLException;

}
```

The following methods are available for use in `IHadoopConnection`:

- `getDelegationToken(String owner, String renewer)`

The connector sends a request to the server to obtain a delegation token with the given owner and renewer.

The method should be called on a Kerberos-authenticated connection.

- `cancelDelegationToken()`

The connector sends a request to the server to cancel the provided delegation token.

- `renewDelegationToken()`

The connector sends a request to the server to renew the provided delegation token.

The following is a basic code sample that demonstrates how to use the above functions:

```
public class TestDelegationTokenClass
{
    public static void main(String[] args) throws
        SQLException
    {
        // Create the connection object with Kerberos
        authentication.
        Connection kerbConnection =
            DriverManager.getConnection(
                "jdbc:hive2://localhost:10000;AuthMech=1;KrbRealm=YourRealm;KrbHostFQDN=
                sample.com;KrbServiceName=hive;");

        // Unwrap the java.sql.Connection object to
        an implementation
        // of IHadoopConnection so the methods for
        delegation token
        // can be called.
        String delegationToken =
            kerbConnection.unwrap
            (IHadoopConnection.class).getDelegationToken
            ("owner_name", "renewer_name");

        // The token can then be used to connect with
        the connector.
        String tokenConnectionString =
```

```
"jdbc:hive2://localhost:10000;AuthMech=6;DelegationToken=" + delegationToken;
Connection tokenConnection =
DriverManager.getConnection
(tokenConnectionString);

// Excluding code for work with the
tokenConnection ...

// The original token (delegationToken) can
be cancelled or renewed by unwrapping the
java.sql.Connection object again to
// an implementation of IHadoopConnection.

// Renewing the token:
kerbConnection.unwrap
(IHadoopConnection.class).renewDelegationToken(delegationToken);

// Cancelling the token:
kerbConnection.unwrap
(IHadoopConnection.class).cancelDelegationToken(delegationToken);

}

}
```

Security and Authentication

To protect data from unauthorized access, some Hive data stores require connections to be authenticated with user credentials or the SSL protocol. The Amazon JDBC Connector for Apache Hive provides full support for these authentication protocols.

Note:

In this documentation, "SSL" indicates both TLS (Transport Layer Security) and SSL (Secure Sockets Layer). The connector supports industry-standard versions of TLS/SSL.

The connector provides mechanisms that allow you to authenticate your connection using the Kerberos protocol, your Hive user name only, or your Hive user name and password. You must use the authentication mechanism that matches the security

requirements of the Hive server. For information about determining the appropriate authentication mechanism to use based on the Hive server configuration, see [Authentication Mechanisms](#) on page 20. For detailed connector configuration instructions, see [Configuring Authentication](#) on page 13.

Additionally, the connector supports SSL connections with one-way authentication. If the server has an SSL-enabled socket, then you can configure the connector to connect to it.

It is recommended that you enable SSL whenever you connect to a server that is configured to support it. SSL encryption protects data and credentials when they are transferred over the network, and provides stronger security than authentication alone. For detailed configuration instructions, see [Configuring SSL](#) on page 30.

The SSL version that the connector supports depends on the JVM version that you are using. For information about the SSL versions that are supported by each version of Java, see "Diagnosing TLS, SSL, and HTTPS" on the Java Platform Group Product Management Blog: https://blogs.oracle.com/java-platform-group/entry/diagnosing_tls_ssl_and_https.

Note:

The SSL version used for the connection is the highest version that is supported by both the connector and the server, which is determined at connection time.

Connector Configuration Options

Connector Configuration Options lists and describes the properties that you can use to configure the behavior of the Amazon JDBC Connector for Apache Hive.

You can set configuration properties using the connection URL. For more information, see [Building the Connection URL](#) on page 11.

Note:

Property names and values are case-sensitive.

AllowSelfSignedCerts

Default Value	Data Type	Required
0	Integer	No

Description

This property specifies whether the connector allows the server to use self-signed SSL certificates.

- 1: The connector allows self-signed certificates.

Important:

When this property is set to 1, SSL verification is disabled. The connector does not verify the server certificate against the trust store, and does not verify if the server's host name matches the common name in the server certificate.

- 0: The connector does not allow self-signed certificates.

Note:

This property is applicable only when SSL connections are enabled.

AsyncExecPollInterval

Default Value	Data Type	Required
	Integer	No

Description

The time in milliseconds between each poll for the asynchronous query execution status.

"Asynchronous" refers to the fact that the RPC call used to execute a query against Hive is asynchronous. It does not mean that JDBC asynchronous operations are supported.

Note:

This option is applicable only to HDInsight clusters.

AuthMech

Default Value	Data Type	Required
Depends on the <code>transportMode</code> setting. For more information, see TransportMode on page 64.	Integer	No

Description

The authentication mechanism to use. Set the property to one of the following values:

- 0 for No Authentication.
- 1 for Kerberos.
- 2 for User Name.
- 3 for User Name And Password.
- 6 for Hadoop Delegation Token.
- 12 for Single Sign-On
- 14 for JWT

BinaryColumnLength

Default Value	Data Type	Required
32767	Integer	No

Description

The maximum number of characters that can be contained in BINARY columns. The range of `BinaryColumnLength` is 0 to 32767.

By default, the columns metadata for Hive does not specify a maximum data length for BINARY columns.

CAIssuedCertsMismatch

Default Value	Data Type	Required
0	Integer	No

Description

This property specifies whether the connector requires the name of the CA-issued SSL certificate to match the host name of the Hive server.

- 0: The connector requires the names to match.
- 1: The connector allows the names to mismatch.

Note:

This property is applicable only when SSL connections are enabled.

CatalogSchemaSwitch

Default Value	Data Type	Required
0	Integer	No

Description

This property specifies whether the connector treats Hive catalogs as schemas or as catalogs.

- 1: The connector treats Hive catalogs as schemas as a restriction for filtering.
- 0: Hive catalogs are treated as catalogs, and Hive schemas are treated as schemas.

DecimalColumnScale

Default Value	Data Type	Required
10	Integer	No

Description

The maximum number of digits to the right of the decimal point for numeric data types.

DefaultStringLength

Default Value	Data Type	Required
255	Integer	No

Description

The maximum number of characters that can be contained in STRING columns. The range of `DefaultStringLength` is 0 to 32767.

By default, the columns metadata for Hive does not specify a maximum data length for STRING columns.

DelegationToken

Default Value	Data Type	Required
None	String	Yes, if <code>AuthMech</code> is set to 6 (Hadoop Delegation Token)

Description

A Hadoop delegation token for authentication.

This token must be provided to the connector in the form of a Base64 URL-safe encoded string. It can be obtained from the connector using the `getDelegationToken()` function, or by utilizing the Hadoop distribution `.jar` files.

DelegationUID

Default Value	Data Type	Required
None	String	No

Description

Use this option to delegate all operations against Hive to a user that is different than the authenticated user for the connection.

Note:

This option is applicable only when connecting to a Hive Server 2 instance that supports this feature.

httpPath

Default Value	Data Type	Required
None	String	Yes, if transportMode=http.

Description

The partial URL corresponding to the Hive server.

The connector forms the HTTP address to connect to by appending the `httpPath` value to the host and port specified in the connection URL. For example, to connect to the HTTP address `http://localhost:10002/cliservice`, you would use the following connection URL:

```
jdbc:hive2://localhost:10002;AuthMech=3;transportMode=http;  
httpPath=cliservice;UID=jsmith;PWD=amazon123;
```

Note:

By default, Hive servers use `cliservice` as the partial URL.

IgnoreTransactions

Default Value	Data Type	Required
0	Boolean	No

Description

This property specifies whether the connector ignores transaction-related operations or returns an error.

- 1: The connector ignores any transaction related operations and returns success.
- 0: The connector returns an "operation not supported" error if it attempts to run a query that contains transaction related operations.

KrbAuthType

Default Value	Data Type	Required
0	Integer	No

Description

This property specifies how the connector obtains the Subject for Kerberos authentication.

- 0: The connector automatically detects which method to use for obtaining the Subject:
 1. First, the connector tries to obtain the Subject from the current thread's inherited `AccessControlContext`. If the `AccessControlContext` contains multiple Subjects, the connector uses the most recent Subject.
 2. If the first method does not work, then the connector checks the `java.security.auth.login.config` system property for a JAAS configuration. If a JAAS configuration is specified, the connector uses that information to create a `LoginContext` and then uses the Subject associated with it.
 3. If the second method does not work, then the connector checks the `KRB5_CONFIG` and `KRB5CCNAME` system environment variables for a Kerberos ticket cache. The connector uses the information from the cache to create a `LoginContext` and then uses the Subject associated with it.

- 1: The connector checks the `java.security.auth.login.config` system property for a JAAS configuration. If a JAAS configuration is specified, the connector uses that information to create a `LoginContext` and then uses the `Subject` associated with it.
- 2: The connector checks the `KRB5_CONFIG` and `KRB5CCNAME` system environment variables for a Kerberos ticket cache. The connector uses the information from the cache to create a `LoginContext` and then uses the `Subject` associated with it.
- 3: The connector uses the native GSS-API feature in the JDK to use the Kerberos tickets in the native Windows credentials cache without the need to set the `AllowTgtSessionKey` property in the Windows registry.

Note:

- The `Native GSS-API` feature is only available in Java 11 or later. While Java 13 and later include a default `Native GSS-API` library. While a default `Native GSS-API` library might be included in a future version of Java 11, if you are using Java 11 it does not include a default `Native GSS-API` library, you may work around the issue by setting the `sun.security.jgss.lib` system property to point to a `sspi_bridge.dll` file included in Java 13 or higher.
- JAAS configuration is disabled by default. To enable JAAS configuration, please set the `JDBC_ENABLE_JAAS` environment variable to 1.

Below is an example of setting the `sun.security.jgss.lib` system property in the Java start-up command to point to the default native GSS-API library included in Java 13.

```
-Dsun.security.jgss.lib="C:\Program Files\Java\jdk-13.0.2\bin\sspi_bridge.dll"
```

KrbHostFQDN

Default Value	Data Type	Required
None	String	Yes, if <code>AuthMech=1</code> .

Description

The fully qualified domain name of the Hive Server 2 host.

KrbRealm

Default Value	Data Type	Required
Depends on your Kerberos configuration	String	No

Description

The realm of the Hive Server 2 host.

If your Kerberos configuration already defines the realm of the Hive Server 2 host as the default realm, then you do not need to configure this property.

KrbServiceName

Default Value	Data Type	Required
None	String	Yes, if <code>AuthMech=1</code> .

Description

The Kerberos service principal name of the Hive server.

LogLevel

Default Value	Data Type	Required
0	Integer	No

Description

Use this property to enable or disable logging in the connector and to specify the amount of detail included in log files.

⚠ Important:

Only enable logging long enough to capture an issue. Logging decreases performance and can consume a large quantity of disk space.

The settings for logging apply to every connection that uses the Amazon JDBC Connector for Apache Hive, so make sure to disable the feature after you are done using it.

Set the property to one of the following numbers:

- 0: Disable all logging.
- 1: Enable logging on the FATAL level, which logs very severe error events that will lead the connector to abort.
- 2: Enable logging on the ERROR level, which logs error events that might still allow the connector to continue running.
- 3: Enable logging on the WARNING level, which logs events that might result in an error if action is not taken.
- 4: Enable logging on the INFO level, which logs general information that describes the progress of the connector.
- 5: Enable logging on the DEBUG level, which logs detailed information that is useful for debugging the connector.
- 6: Enable logging on the TRACE level, which logs all connector activity.

When logging is enabled, the connector produces the following log files in the location specified in the `LogPath` property:

- A `HiveJDBC_driver.log` file that logs connector activity that is not specific to a connection.
- A `HiveJDBC_connection_[Number].log` file for each connection made to the database, where *[Number]* is a number that identifies each log file. This file logs connector activity that is specific to the connection.

If the `LogPath` value is invalid, then the connector sends the logged information to the standard output stream (`System.out`).

LogPath

Default Value	Data Type	Required
The current working directory	String	No

Description

The full path to the folder where the connector saves log files when logging is enabled.

Note:

To make sure that the connection URL is compatible with all JDBC applications, it is recommended that you escape the backslashes (\) in your file path by typing another backslash.

PreparedMetaLimitZero

Default Value	Data Type	Required
0	Integer	No

Description

This property specifies whether the `PreparedStatement.getMetadata()` call will request metadata from the server with `LIMIT 0`.

- 1: The `PreparedStatement.getMetadata()` call uses `LIMIT 0`.
- 0: The `PreparedStatement.getMetadata()` call does not use `LIMIT 0`.

PWD

Default Value	Data Type	Required
anonymous	String	Yes, if <code>AuthMech=3</code> .

Description

The password corresponding to the user name that you provided using the property [UID](#) on page 65.

⚠ Important:

If you set the `AuthMech` to 3, the default `PWD` value is not used and you must specify a password.

RowsFetchedPerBlock

Default Value	Data Type	Required
10000	Integer	No

Description

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.

SocketTimeout

Default Value	Data Type	Required
0	Integer	No

Description

The number of seconds that the TCP socket waits for a response from the server before raising an error on the request.

When this property is set to 0, the connection does not time out.

SSL

Default Value	Data Type	Required
0	Integer	No

Description

This property specifies whether the connector communicates with the Hive server through an SSL-enabled socket.

- 1: The connector connects to SSL-enabled sockets.
- 2: The connector connects to SSL-enabled sockets using two-way authentication.
- 0: The connector does not connect to SSL-enabled sockets.

Note:

SSL is configured independently of authentication. When authentication and SSL are both enabled, the connector performs the specified authentication method over an SSL connection.

SSLKeyStore

Default Value	Data Type	Required
None	String	No

Description

The full path of the Java KeyStore containing the server certificate for one-way SSL authentication.

See also the property [SSLKeyStorePwd](#) on page 62.

Note:

The Amazon JDBC Connector for Apache Hive accepts TrustStores and KeyStores for one-way SSL authentication. See also the property [SSLTrustStore](#) on page 63.

SSLKeyStorePwd

Default Value	Data Type	Required
None	Integer	Yes, if you are using a KeyStore for connecting over SSL.

Description

The password for accessing the Java KeyStore that you specified using the property [SSLKeyStore](#) on page 62.

SSLTrustStore

Default Value	Data Type	Required
<code>jssecacerts</code> , if it exists. If <code>jssecacerts</code> does not exist, then <code>cacerts</code> is used. The default location of <code>cacerts</code> is <code>jre\lib\security\.</code>	String	No

Description

The full path of the Java TrustStore containing the server certificate for one-way SSL authentication.

If the trust store requires a password, provide it using the property `SSLTrustStorePwd`. See [SSLTrustStorePwd](#) on page 63.

Note:

The Amazon JDBC Connector for Apache Hive accepts TrustStores and KeyStores for one-way SSL authentication. See also the property [SSLKeyStore](#) on page 62.

SSLTrustStorePwd

Default Value	Data Type	Required
None	String	Yes, if using a TrustStore.

Description

The password for accessing the Java TrustStore that you specified using the property [SSLTrustStore](#) on page 63.

SSLTrustStoreType

Default Value	Data Type	Required
JKS	String	No

Description

The type of Java TrustStore that is being used for one-way SSL authentication.

SSOWebServerTimeout

Default Value	Data Type	Required
120	Integer	No

Description

This property specifies the number of seconds that the connector waits before timing out while waiting for a browser response when authenticating using Single Sign-On (SSO).

If this property is set to 0, the connector will wait for an indefinite amount of time

TransportMode

Default Value	Data Type	Required
sasl	String	No

Description

The transport protocol to use in the Thrift layer.

- `binary`: The connector uses the Binary transport protocol.

When connecting to a Hive Server 1 instance, you must use this setting. If you use this setting and do not specify the `AuthMech` property, then the connector uses `AuthMech=0` by default. This setting is valid only when the `AuthMech` property is set to 0 or 3.

- `sasl`: The connector uses the SASL transport protocol.

If you use this setting but do not specify the `AuthMech` property, then the connector uses `AuthMech=2` by default. This setting is valid only when the `AuthMech` property is set to 1, 2, or 3.

- `http`: The connector uses the HTTP transport protocol.

When connecting to Hive through the Apache Knox Gateway, you must use this setting. If you use this setting but do not specify the `AuthMech` property, then the connector uses `AuthMech=3` by default. This setting is valid only when the `AuthMech` property is set to 3.

If you set this property to `http`, then the port number in the connection URL corresponds to the HTTP port rather than the TCP port, and you must specify the `httpPath` property. For more information, see [httpPath](#) on page 55.

UID

Default Value	Data Type	Required
anonymous	String	Yes, if <code>AuthMech=3</code> .

Description

The user name that you use to access the Hive server.

Important:

If you set the `AuthMech` to 3, the default `UID` value is not used and you must specify a user name.

UseNativeQuery

Default Value	Data Type	Required
	Integer	No

Description

This property specifies whether the connector transforms the queries emitted by applications.

- 0: The connector transforms the queries emitted by applications and converts them into an equivalent form in HiveQL.

- 1: The connector does not transform the queries emitted by applications, so the native query is used.

Note:

If the application is Hive-aware and already emits HiveQL, then enable this option to avoid the extra overhead of query transformation.

zk

Default Value	Data Type	Required
None	String	No

Description

The connection string to one or more ZooKeeper quorums, written in the following format where *[ZK_IP]* is the IP address, *[ZK_Port]* is the port number, and *[ZK_Namespace]* is the namespace:

```
[ZK_IP]:[ZK_Port]/[ZK_Namespace]
```

For example:

```
jdbc:hive2://zk=192.168.0.1:2181/hiveserver2
```

Use this option to enable the Dynamic Service Discovery feature, which allows you to connect to Hive servers that are registered against a ZooKeeper service by connecting to the ZooKeeper service.

You can specify multiple quorums in a comma-separated list. If connection to a quorum fails, the connector will attempt to connect to the next quorum in the list.

Contact Us

For support, check the EMR Forum at <https://forums.aws.amazon.com/forum.jspa?forumID=52> or open a support case using the AWS Support Center at <https://aws.amazon.com/support>

Third-Party Trademarks

Simba, the Simba logo, SimbaEngine, SimbaEngine C/S, SimbaExpress and SimbaLib are registered trademarks of Simba Technologies Inc.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Apache Hive, Apache, and Hive are trademarks or registered trademarks of The Apache Software Foundation or its subsidiaries in Canada, United States and/or other countries.

All other trademarks are trademarks of their respective owners.