Databricks JDBC Driver

Installation and Configuration Guide

Version 2.7.1

December 2024

About This Guide

The *Databricks JDBC Driver Installation and Configuration Guide* explains how to install and configure the Databricks JDBC Driver on all supported platforms. It also provides details about the connector's features.

The guide is intended for end users of the Databricks JDBC Driver.

To use the Databricks JDBC Driver, the following knowledge is helpful:

- Familiarity with the platform on which you are using the Databricks JDBC Driver
- Ability to use the data store to which the Databricks JDBC Driver 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.

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

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

Contents

About the Databricks JDBC Driver

The Databricks JDBC Driver is used for direct SQL and HiveQL access to Apache Hadoop / Spark, enabling Business Intelligence (BI), analytics, and reporting on Hadoop / Spark-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 Spark-aware, then the connector is configurable to pass the query through to the database for processing. The connector interrogates Spark 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.

The Databricks JDBC Driver complies with the JDBC 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 Spark 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 Databricks JDBC Driver must have Java Runtime Environment (JRE) 8.0, or 11.0 installed.

The Databricks JDBC Driver supports DATABRICKS Runtime LTS releases. For more information, see https://docs.databricks.com/en/release-notes/runtime/index.html.

Additionally, the Databricks JDBC Driver supports Apache Arrow. When using a Java JVM version 11 or higher, add the line `--add-opens java.base/java.nio=ALL-UNNAMED` to the JVM arguments.

Databricks JDBC Driver Files

The Databricks JDBC Driver is delivered in the following ZIP archives, where *[Version]* is the version number of the connector:

- `DatabricksJDBC42-[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 Databricks JDBC Driver

To install the Databricks JDBC Driver on your machine, extract the files from the appropriate ZIP archive to the directory of your choice.

> **Important:**
> If you received a license file through email, then you must copy the file into the same directory as the connector JAR file before you can use the Databricks JDBC Driver.

To access a Spark data store using the Databricks JDBC Driver, you need to configure the following:

- The list of connector library files (see Referencing the JDBC Connector Libraries)
- The `Driver` or `DataSource` class (see Registering the Connector Class)
- The connection URL for the connector (see Building the Connection URL).

Referencing the JDBC Connector Libraries

Before you use the Databricks JDBC Driver, 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 7:

- For Windows: http://docs.oracle.com/javase/7/docs/technotes/tools/windows/classpath.html
- For Linux and Solaris: http://docs.oracle.com/javase/7/docs/technotes/tools/solaris/classpath.html

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 Databricks JDBC Driver to Spark 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.databricks.client.jdbc.Driver`
- `com.databricks.client.jdbc.DataSource`

To support JDBC 4.2, classes with the following fully-qualified class names (FQCNs) are available:

- `com.databricks.client.jdbc42.Driver`
- `com.databricks.client.jdbc42.DataSource`

The following sample code shows how to use the `DriverManager` class to establish a connection for JDBC 4.2:

```
private static Connection connectViaDM() throws Exception
{
        Connection connection = null;
        Class.forName(DRIVER_CLASS);
        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;
        Class.forName(DRIVER_CLASS);
        DataSource ds = new
        com.databricks.client.jdbc42.DataSource();
```

```
        ds.setURL(CONNECTION_URL);
        connection = ds.getConnection();
        return connection;
}
```

Building the Connection URL

Use the connection URL to supply connection information to the data store that you are accessing. The following is the format of the connection URL for the Databricks JDBC Driver, where *[Host]* is the DNS or IP address of the Spark server and *[Port]* is the number of the TCP port that the server uses to listen for client requests:

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

Note:
By default, Databricks uses port 443.

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

You can specify optional settings such as 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. If you specify a property that is not supported by the connector, then the connector attempts to apply the property as a Spark server-side property for the client session.

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

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

For example, to connect to port 11000 on an Spark server 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:databricks://localhost:11000/default2;AuthMech=3;UID=databricks;PWD=
databricks
```

Important:

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

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, type the `show databases` command at the Spark command prompt.

Configuring Authentication

The Databricks JDBC Driver supports the following authentication mechanisms:

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

For information about selecting an appropriate authentication mechanism when using the Databricks JDBC Driver, see Configuring Authentication.

For information about the properties you can use in the connection URL, see Connector Configuration Options.

> **Note:** In addition to authentication, you can configure the connector to connect over SSL. For more information, see Configuring SSL.

Using PAT (Personal Access Token)

This authentication mechanism uses Personal access token as password and token as UID.

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.

To configure PAT authentication:

1. Set the `AuthMech` property to `3`.
2. Set the `UID` property to token for accessing the Spark server.
3. Set the `PWD` property to personal access token.

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

jdbc:databricks://node1.example.com:443;AuthMech=3;UID=token;PWD=databricksPAT;

In this example, PAT authentication is enabled for JDBC connections, the UID is token, the PWD (personal access token here) is databricksPAT and the server is listening on port 443 for JDBC connections.

Using OAuth 2.0

Three types of authentication work flow are available when using OAuth 2.0, token pass-through, browser based authentication, and M2M authentication.

This authentication mechanism is available for Spark Thrift Server instances only. When you use OAuth 2.0 authentication, HTTP is the only Thrift transport protocol available. Browser based authentication work flow only works when SSL is enabled.

There is a discovery mode that enables the connector to auto-fill some endpoints or configurations. The endpoint discovery is enabled by default, you can disable it by setting `EnableOIDCDiscovery=0`. You can also pass the OIDC discovery endpoint by using `OIDCDiscoveryEndpoint`. The connector automatically discovers OAuth2AuthorizationEndPoint and OAuth2TokenEndPoint.

To enable SSL for OAuth endpoints, `UseServerSSLConfigsForOAuthEndPoint` can be used.

Token Pass-through

This authentication mechanism requires a valid OAuth 2.0 access token or refresh token. Be aware that access tokens typically expire after a certain amount of time, after which you must either refresh the token or obtain a new one from the server. If refresh token is provided, then the connector would use it to refresh access token when necessary.

> **Note:**
> When the access token expires, the connector will return an error with SQLState 08006. To provide the connector with a new access token, use `Connection.setClientInfo()` with "Auth_AccessToken" as the key and the new access token as the value. The connector will update the access token and use the newly provided token for any subsequent calls to the server.
>
> For more information regarding `Connection.setClientInfo()` please refer to https://docs.oracle.com/javase/8/docs/api/java/sql/Connection.html#setClientInfo-java.lang.String-java.lang.String-.

To configure OAuth 2.0 token pass-though authentication:

1. Set the `AuthMech` property to `11`.
2. Set the `Auth_Flow` property to `0`.
3. Set the `Auth_AccessToken` property to your access token.

Using Browser Based Authentication

This authentication mechanism requires SSL to be enabled.

To configure OAuth 2.0 browser based authentication:

1. Set the `AuthMech` property to `11`.
2. Set the `Auth_Flow` property to `2`.
3. Set the `PWD` property to a password of your choice. This is the key used for refresh token encryption.

Once the browser based authentication flow is done, the refresh token is cached so you don't need to do authenticate again. The token cache can be disabled by setting `EnableTokenCache=0`. The `TokenCachePassPhrase` property needs to be set when using the token cache.

Using M2M Based Authentication

This authentication mechanism is available for Spark Thrift Server instances only.

To configure OAuth 2.0 M2M based authentication:

1. Set the `AuthMech` property to `11`.
2. Set the `Auth_Flow` property to `1`.
3. Set the `OAuth2ClientId` property to client id used in specified authentication source.
4. Set the `OAuth2Secret` property to secret used in the specified authentication source.

To configure JWT assertion authentication:

1. Set the `Auth_KID` property to unique `key ID`.
2. Set the `Auth_JWT_Key_File` property to `key PEM` file.
3. Set the `Auth_JWT_Key_Passphrase` property, if private key needs to be decrypted, otherwise keep setting `unset`.
4. Set the `UseJWTAssertion` property to `1`.
5. Set the `Auth_JWT_Alg` property to use an alternate signing algorithm: `RS256, RS384, RS512, PS256, PS384, PS512, ES256, ES384, ES512`.

Note:
For JWT assertion authentication, import BouncyCastle v1.74.

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 Spark 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.

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:
     - Create a TrustStore containing your signed, trusted server certificate.
     - Set the `SSLTrustStore` property to the full path of the TrustStore.
     - Set the `SSLTrustStorePwd` property to the password for accessing the TrustStore.
     - If the TrustStore is not a JKS TrustStore, set the `SSLTrustStoreType` property to the correct type. The supported types are:
       - `SSLTrustStoreType=BCFKS` (BouncyCastle FIPS Keystore)
       - `SSLTrustStoreType=PKCS12` (Public Key Cryptography Standards #12)

       Note: `SSLTrustStoreType=PKCS11` (Public Key Cryptography Standards #11) TrustStore type is not supported.
     - 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:
  - Create a KeyStore containing your signed, trusted server certificate.
  - Set the `SSLKeyStore` property to the full path of the KeyStore.
  - Set the `SSLKeyStorePwd` property to the password for accessing the KeyStore.
  - If the KeyStore is not a JKS KeyStore, set the `SSLKeyStoreType` property to the correct type.
  - 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 Spark server, set the `CAIssuedCertNamesMismatch` property to `1`.
5. If you want the connector to use the subject alternative names of the certificate instead of the common name when checking if the certificate name matches the host name of the Spark server, then set the following properties:
   a. Make sure that the `AllowSelfSignedCerts` property is set to `0`.
   b. Set the `CAIssuedCertNamesMismatch` property to `1` if you have not already done so.
   c. Set the `SubjectAlternativeNamesHostNames` property to `1`.

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

```
jdbc:databricks://localhost:443;AuthMech=3;SSL=1;
SSLKeyStore=C:\\Users\\bsmith\\Desktop\\keystore.jks;SSLKeyStorePwd=databric
ksSSL123;UID=token;PWD=databricksPAT;
```

Note:
For more information about the connection properties used in SSL connections, see Connector Configuration Options.

Configuring Virtual Private Cloud (VPC) Services

You can configure the connector to connect to Spark using a Virtual Private Cloud (VPC) service. To do this, use the following connection properties:

- `SocketFactory`
- `SocketFactoryArg`
- `DnsResolver`
- `DnsResolverArg`

The `SocketFactory` property extends `javax.net.SocketFactory`, and the `DnsResolver` property implements `com.interfaces.networking.CustomDnsResolver`. The `SocketFactoryArg` and `DnsResolverArg` properties pass any required arguments to these services.

The properties in the following instructions are all optional, and only need to be used if you are connecting through the relevant service.

To configure the connector to use a VPC:

1. If necessary, set the `SocketFactory` property to the fully-qualified class path for a class that extends `javax.net.SocketFactory` and provides the socket factory implementation.
2. If necessary, set the `SocketFactoryArg` to a string argument to pass to the constructor of the class indicated by the `SocketFactory` property.
3. If necessary, set the `DnsResolver` property to the fully-qualified class path for a class that extends the `DnsResolver` interface for the connector, `com.interfaces.networking.CustomDnsResolver`.
4. If necessary, set the `DnsResolverArg` to a string argument to pass to the constructor of the class indicated by the `DnsResolver` property.

For example, the following connection URLs show how to connect to data sources using the supported VPCs.

Using SocketFactory:

```
jdbc:databricks
://localhost:443;UID=jsmith;SocketFactory=com..databricks.junit.jdbc.utils.CustomSocketFact
ory;SocketFactoryArg=Args;
```

Using DnsResolver:

```
jdbc:databricks
://localhost:443;UID=jsmith;DnsResolver=com..databricks.junit.jdbc.utils.TestDnsResolver;D
nsResolverArg=agrs;
```

Using both SocketFactory and DnsResolver:

```
jdbc:databricks
://localhost:443;UID=jsmith;DnsResolver=com..junit.jdbc.utils.TestDnsResolver;DnsResolver
Arg=Agrs;SocketFactory=com..databricks.junit.jdbc.utils.CustomSocketFactory;SocketFactor
yArg=Args;
```

Configuring Proxy Connections

You can configure the connector to connect through a proxy server instead of connecting directly to the Spark service. When connecting through a proxy server, the connector supports basic authentication.

Note:
The connector currently only supports SOCKS proxies.

You provide the configuration information to the connector in the connection URL. For more information about the syntax of the connection URL, see Building the Connection URL .

To configure a proxy connection:

1. Set the `UseProxy` property to `1`.
2. Set the `ProxyHost` property to the IP address or host name of your proxy server.
3. Set the `ProxyPort` property to the port that the proxy server uses to listen for client connections.
4. Set the `ProxyIgnoreList` property to a comma separated hostname.
5. For authentication with the proxy server:
   a. Set the `ProxyAuth` property to `1`.
   b. Set the `ProxyUID` property to your user name for accessing the server.
   c. Set the `ProxyPWD` property to your password for accessing the server.

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 Databricks JDBC Driver, 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 Databricks JDBC Driver, 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:databricks.databricks://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 Databricks JDBC Driver produces the following log files in the location specified in the `LogPath` property:

   - A `SparkJDBC_driver.log` file that logs connector activity that is not specific to a connection.

- A `SparkJDBC_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 Databricks JDBC Driver:

- SQL Query versus HiveQL Query
- Data Types
- Catalog and Schema Support
- Write-back

SQL Query versus HiveQL Query

The native query language supported by Spark 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 Databricks JDBC Driver supports many common data formats, converting between Spark, SQL, and Java data types.

The following table lists the supported data type mappings.

Spark Type	SQL Type	Java Type
BIGINT	BIGINT	java.math.BigInteger
BINARY	VARBINARY	byte[]
BOOLEAN	BOOLEAN	Boolean
DATE	DATE	java.sql.Date
DECIMAL	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	VARCHAR	String

Catalog and Schema Support

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

Note:

- The synthetic SPARK catalog only applies to servers that do not support multiple catalogs. For servers that do, the connector returns their catalogs as is.
- Setting the `CatalogSchemaSwitch` connection property to `1` will cause Spark catalogs to be treated as schemas in the connector as a restriction for filtering.

Write-back

The Databricks JDBC Driver supports translation for the following syntax when connecting to a Spark Thrift Server instance that is running Spark 1.3 or later:

- INSERT
- CREATE
- DROP

Spark does not support UPDATE or DELETE syntax.

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.

Connector Configuration Options

Connector Configuration Options lists and describes the properties that you can use to configure the behavior of the Databricks JDBC Driver.

You can set configuration properties using the connection URL. For more information, see Building the Connection URL.

Note:
Property names and values are case-sensitive.

AllowSelfSignedCerts

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 or subject alternative 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.

Default Value	Data Type	Required
`0`	Integer	No

AsyncExecPollInterval

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 Spark is asynchronous. It does not mean that JDBC asynchronous operations are supported.

Default Value	Data Type	Required
`10`1	Integer	No

AuthMech

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

- `3` for PAT (Personal Access Token).
- `11` for OAuth 2.0.

Default Value	Data Type	Required
`3`	Integer	No

Auth_AccessToken

The OAuth 2.0 access token used for connecting to a server.

Default Value	Data Type	Required
None	String	Yes, if AuthMech=11.

Auth_Flow

This option specifies the type of OAuth authentication flow that the connector uses when the `AuthMech` property is set to `11`.

When this option is set to Token Passthrough (`0`), the connector uses the access token specified by the Auth_Access Token (`Auth_AccessToken`) or the refresh token specified by `Auth_RefreshToken` option to authenticate the connection to the server. For more information, see Auth_AccessToken and Auth_RefreshToken.

Default Value	Data Type	Required
0 (Token passthrough)	Integer	No

Auth_JWT_Key_File

This property specifies the key file path for JWT OAuth 2.0 authentication.

Default Value	Data Type	Required
None	String	No

Auth_JWT_Key_Passphrase

This property specifies the password for the encrypted private key file for JWT OAuth 2.0 authentication.

Default Value	Data Type	Required
None	String	No

Auth_KID

This property specifies the key ID for JWT OAuth 2.0 authentication.

Default Value	Data Type	Required
None	Integer	No

Auth_RefreshToken

The OAuth 2.0 refresh token used for connecting to a server.

Default Value	Data Type	Required
None	String	No

Auth_Token_Expiry_Buffer

If the lifetime (in seconds) for the current access token is less than Auth_Token_Expiry_Buffer, the connector should try to refresh access token using refresh token passed to connector.

Default Value	Data Type	Required
60	Integer	No

CAIssuedCertsMismatch

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

Default Value	Data Type	Required
0	Integer	No

CatalogSchemaSwitch

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

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

Default Value	Data Type	Required
0	Integer	No

DecimalColumnScale

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

Default Value	Data Type	Required
10	Integer	No

DefaultStringColumnLength

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

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

Default Value	Data Type	Required
255	Integer	No

DnsResolver

The fully-qualified class path for a class that extends the DnsResolver interface provided by the connector, com.interfaces.networking.CustomDnsResolver. Using a custom DnsResolver enables you to provide your own resolver logic.

Default Value	Data Type	Required
None	String	No

DnsResolverArg

A string argument to pass to the constructor of the class indicated by the `DnsResolver` property.

Default Value	Data Type	Required
None	String	No

EnableAsyncModeForMetadataOperation

This property specifies whether to enable Async mode for metadata operation.

Default Value	Data Type	Required
`1`	Integer	No

EnableOIDCDiscovery

This property specifies whether to enable the OIDC discovery.

- `1`: Enables OIDC discovery.
- `0`: Disables OIDC discovery.

Default Value	Data Type	Required
`True`	String	No

EnableTokenCache

This property specifies whether the connector enables the token cache for OAuth.

- `1`: The connector enables the token cache for OAuth.
- `0`: The connector disables the token cache for OAuth.

Default Value	Data Type	Required
`1`	Integer	No

http.header.

Set a custom HTTP header by using the following syntax, where *[HeaderKey]* is the name of the header to set and *[HeaderValue]* is the value to assign to the header:

```
http.header.[HeaderKey]=[HeaderValue]
```

For example:

```
http.header.AUTHENTICATED_USER=john
```

After the connector applies the header, the http.header. prefix is removed from the DSN entry, leaving an entry of *[HeaderKey]=[HeaderValue]*

The example above would create the following custom HTTP header:

```
AUTHENTICATED_USER: john
```

Note:

- The `http.header.` prefix is case-sensitive.
- This option is applicable only when you are using HTTP as the Thrift transport protocol. For more information, see TransportMode.

Default Value	Data Type	Required
None	String	No

httpPath

The partial URL corresponding to the Spark 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:spark://localhost:10002;AuthMech=3;transportMode=http;
httpPath=cliservice;UID=jsmith;PWD=databricks123;
```

Note:
By default, Spark servers use `cliservice` as the partial URL.

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

IgnoreTransactions

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.

Default Value	Data Type	Required
`0`	Boolean	No

LogLevel

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 Databricks JDBC Driver, 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 `SparkDatabricksJDBC_driver.log` file that logs connector activity that is not specific to a connection.
- A `SparkDatabricksJDBC_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`).

Default Value	Data Type	Required
`0`	Integer	No

LogPath

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.

Default Value	Data Type	Required
The current working directory	String	No

NonRowcountQueryPrefixes

A comma-separated list of queries used to return a regular result set, rather than a row count.

For example:

```
NonRowcountQueryPrefixes=INSERT,UPDATE,DELETE;
```

Default Value	Data Type	Required
None	String	No

OAuth2ClientId

This property specifies the Client Id for OAuth 2.0 authentication.

Default Value	Data Type	Required
databricks-sql-jdbc	String	No

OAuth2ConnAuthAuthscopeKey

This property specifies the scope for OAuth 2.0 authentication.

Default Value	Data Type	Required
sql offline_access	String	No

OAuthEnabledIPAddressRanges

List of IP ranges that bypass OAuth AuthorizationURL check. Items in the list are separated by comma (","), and list is defined by dash("-").

```
Example : "10.0.0.0-10.255.255.255,192.54.200.198".
```

Default Value	Data Type	Required
Null	Integer	No

OAuthHTTPRetryTimeout

This option specifies whether the connector determines the retry timeout for OAuth access and refresh token HTTP requests in minutes.

Default Value	Data Type	Required
15	Integer	No

OAuth2ConnAuthAuthorizeEndpoint

This property specifies the authorization endpoint for OAuth2.0 connection.

Default Value	Data Type	Required
$(host) +/oidc/oauth2/v2.0/authorize	String	No

OAuth2ConnAuthTokenEndpoint

This property specifies the authorization endpoint for OAuth2.0 connection.

Default Value	Data Type	Required
`$(host)+/oidc/oauth2/v2.0/token`	String	No

OAuth2ConnAuthCodeChallengeMethodKey

This property specifies the code challenge method for OAuth2.0 connection.

Default Value	Data Type	Required
`S256`	String	No

OAuth2RedirectUrlPort

The number of the redirect port that the connector uses for OAuth authentication.

When not specified, the connector uses `8020` as the OAuth redirect port. If `8020` is not available, the connector tries `8021`, and then `8022`, until the login timeout.

When assigned a single port, the connector uses it as the starting port and tries port+1, port+2, until the login timeout.

When assigned a comma separated port list, the connector tries all the ports in the list.

Default Value	Data Type	Required
`8020`	Integer	No

OAuth2Secret

This property specifies the client secret for OAuth2.0 connection.

Default Value	Data Type	Required
None	String	No

OAuthRetriableHttpCode

This option specifies whether the connector determines the retriable Http codes for OAuth access and refresh token HTTP requests.

Default Value	Data Type	Required
`408,502,503,504`	Integer	No

OAuthWebServerTimeout

The length of time, in seconds, for which the connector waits for a browser response during OAuth2 authentication before timing out. If set to `0`, the connector waits for an indefinite amount of time.

Note:
If SQL_ATTR_LOGIN_TIMEOUT is set, SQL_ATTR_LOGIN_TIMEOUT takes precedence. The connector honors SQL_ATTR_LOGIN_TIMEOUT when using the OAuth2 browser based authentication workflow.

Default Value	Data Type	Required
`120`	Integer	No

OIDC Discovery Endpoint

The OIDC discovery endpoint.

Note:
This option is applicable only when `Use OIDC Discovery Endpoint` is enabled.

Default Value	Data Type	Required
`Null`	String	No

OCIConfigFile

The absolute path to the OCI configuration file to use for the connection.

Note:
If this property is specified, the connector ignores the OCI_CLI_CONFIG_FILE environment variable when attempting to locate the configuration file.

Default Value	Data Type	Required
None	String	No

OCIProfile

The name of the OCI profile to use for the connection. The connector retrieves the named profile from the configuration file, and uses its credentials for the connection.

If the named profile cannot be opened, the connector switches to token-based authentication.

Default Value	Data Type	Required
DEFAULT	String	No

ProxyAuth

This option specifies whether the proxy server that you are connecting to requires authentication.

- `1`: The proxy server that you are connecting to requires authentication.
- `0`: The proxy server that you are connecting to does not require authentication.

Default Value	Data Type	Required
`0`	Integer	No

ProxyHost

The IP address or host name of your proxy server.

Default Value	Data Type	Required
None	String	Yes, if connecting through a proxy server.

Proxy Ignore List

This option specifies whether the connector allows the hosts or domains that do not use proxy. For example: `ProxyIgnoreList=localhost,127.0.0.1`

Default Value	Data Type	Required
`Empty string`	String	No

ProxyPort

The listening port of your proxy server.

Default Value	Data Type	Required
None	Integer	Yes, if connecting through a proxy server.

ProxyPWD

The password that you use to access the proxy server.

Default Value	Data Type	Required
None	String	Yes, if connecting through a proxy server that requires authentication.

ProxyUID

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

Default Value	Data Type	Required
None	String	Yes, if connecting through a proxy server that requires authentication.

PWD

The password corresponding to the user name that you provided using the property UID.

If `UID` is either set to `token` or left at the default value (which is also `token`), then set this `PWD` property to the Databricks token value that you want to use for authentication.

Default Value	Data Type	Required
None	String	Yes, if `AuthMech=3`.

RateLimitRetry

This option specifies whether the connector retries operations that receive HTTP 429 responses if the server response is returned with `Retry-After` headers.

- `1`: The connector retries the operation until the time limit specified by `RateLimitRetryTimeout` is exceeded. For more information, see RateLimitRetryTimeout.
- `0`: The connector does not retry the operation, and returns an error message.

Default Value	Data Type	Required
`1`	Integer	No

RateLimitRetryTimeout

The number of seconds that the connector waits before stopping an attempt to retry an operation when the operation receives an HTTP 429 response with `Retry-After` headers.

Default Value	Data Type	Required
`120`	Integer	No

RowsFetchedPerBlock

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.

Default Value	Data Type	Required
10000	Integer	No

SocketFactory

The fully-qualified class path for a class that extends `javax.net.SocketFactory` and provides the socket factory implementation. Using a custom SocketFactory enables you to customize the socket that the connector uses.

Default Value	Data Type	Required
None	String	No

SocketFactoryArg

A string argument to pass to the constructor of the class indicated by the `SocketFactory` property.

Default Value	Data Type	Required
None	String	No

SocketTimeout

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.

Default Value	Data Type	Required
0	Integer	No

SparkServerType

The type of cluster that the connector connects to:

- `Other`: The connection is for a regular Spark cluster. When this value is specified, the authentication method is specified using the AuthMech configuration property. For more information, see AuthMech.

Default Value	Data Type	Required
None	String	No

SSL

This property specifies whether the connector communicates with the Spark 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.

Default Value	Data Type	Required
`0 1`	Integer	No

SSLKeyStore

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

See also the property SSLKeyStorePwd.

Note:
The Databricks JDBC Driver accepts TrustStores and KeyStores for one-way SSL authentication. See also the property SSLTrustStore.

Default Value	Data Type	Required
None	String	No

SSLKeyStoreProvider

The provider of the Java Security API for the KeyStore that is being used for one-way SSL authentication.

Default Value	Data Type	Required
None	String	No

SSLKeyStorePwd

The password for accessing the Java KeyStore that you specified using the property SSLKeyStore.

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

SSLKeyStoreType

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

Default Value	Data Type	Required
`JKS`	String	No

SSLTrustStore

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.

Note:
The Databricks JDBC Driver accepts TrustStores and KeyStores for one-way SSL authentication. See also the property SSLKeyStore.

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

SSLTrustStoreProvider

The provider of the Java Security API for the TrustStore that is being used for one-way SSL authentication.

Default Value	Data Type	Required
None	String	No

SSLTrustStorePwd

The password for accessing the Java TrustStore that you specified using the property SSLTrustStore.

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

SSLTrustStoreType

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

Default Value	Data Type	Required
`JKS`	String	No

StagingAllowedLocalPaths

The comma separated list of allowed local paths for downloading and uploading of UC Volume Ingestion files.

Default Value	Data Type	Required
" "	String	No

StripCatalogName

This property specifies whether the connector removes catalog names from query statements, if translation fails .

- `1`: If query translation fails, then the connector removes catalog names from the query statement.
- `0`: The connector does not remove catalog names from query statements.

Default Value	Data Type	Required
`1`	Integer	No

SubjectAlternativeNamesHostNames

This property specifies whether the connector uses the subject alternative names of the SSL certificate instead of the common name when checking if the certificate name matches the host name of the Spark server.

- `0`: The connector checks if the common name of the certificate matches the host name of the server.

- `1`: The connector checks if the subject alternative names of the certificate match the host name of the server.

Note:
This property is applicable only when SSL with one-way authentication is enabled, and the `CAIssuedCertsMismatch` property is also enabled.

Default Value	Data Type	Required
`0`	Integer	No

TemporarilyUnavailableRetry

This option specifies whether the connector retries operations that receive HTTP 503 responses if the server response is returned with `Retry-After` headers.

- `1`: The connector retries the operation until the time limit specified by `TemporarilyUnavailableRetryTimeout` is exceeded. For more information, see TemporarilyUnavailableRetryTimeout.
- `0`: The connector does not retry the operation, and returns an error message.

Default Value	Data Type	Required
`1`	Integer	No

TemporarilyUnavailableRetryTimeout

The number of seconds that the connector waits before stopping an attempt to retry an operation when the operation receives an HTTP 503 response with `Retry-After` headers.

Default Value	Data Type	Required
`900`	Integer	No

TokenCachePassPhrase

Set this property to a passphrase for token cache encryption.

Default Value	Data Type	Required
None	String	Yes

TransportMode

The transport protocol to use in the Thrift layer.

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

  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.

  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, seehttpPath.

Default Value	Data Type	Required
`http`	String	No

UCIngestionRetriableHttpCode

The HTTP code to configure for retry mechanism of UC Ingestion commands.

Default Value	Data Type	Required
`408,502,503,504`	Integer	No

UCIngestionRetryTimeout

The retry timeout in minutes for UCIngestion HTTP requests.

Default Value	Data Type	Required
`15`	Integer	No

UID

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

Default Value	Data Type	Required
`token`, if `AuthMech=3`	String	Yes, if `AuthMech=3`.

UseJWTAssertion

This option specifies whether the connector uses JWT for OAuth 2.0 authentication.

- Enabled (`True`): The connector uses JWT for OAuth 2.0 authentication
- Disabled (`False`): The connector does not use JWT for OAuth 2.0 authentication

Default Value	Data Type	Required
`false`	Integer	No

UseNativeQuery

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.
- `2`: The connector automatically sets this property to either `0` or `1`, depending on the server capabilities.

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

Default Value	Data Type	Required
`2`	Integer	No

UserAgentEntry

The User-Agent entry to be included in the HTTP request. This value is in the following format:

[ProductName]/[ProductVersion] [Comment]

Where:

- `[ProductName]` is the name of the application, with no spaces.
- `[ProductVersion]` is the version number of the application.
- `[Comment]` is an optional comment. Nested comments are not supported.

Only one User-Agent entry may be included.

Default Value	Data Type	Required
None	String	No

UseProxy

This option specifies whether the connector connects through a proxy server.

- `1`: The connector connects to a proxy server based on the information provided in `ProxyAuth`, `ProxyHost`, `ProxyPort`, `ProxyUID`, and `ProxyPWD` keys.
- `0`: The connector connects to the Spark server directly.

Note:
The connector currently only supports SOCKS proxies.

Default Value	Data Type	Required
0	Integer	No

UseServerSSLConfigsForOAuthEndPoint

This option specifies whether to use SSL settings for the OAuth Endpoint.

`1`: The connector shares SSL settings with the OAuth endpoints.

`0`: the Oauth Endpoint will have SSL disabled.

Default Value	Data Type	Required
0	Integer	No

UseSystemTrustStore

This option specifies whether to use the systemtruststore for JVM.

`1`: The connector uses the system's truststore. Only on Windows

`0`: The connector will use either declared SSL truststore or will not use truststore.

Default Value	Data Type	Required
0	Integer	No

Third-Party Trademarks

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

Apache Spark, Apache, and Spark 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.