Databricks ODBC Driver

Installation and Configuration Guide

v0.8.0

CONTENTS

1 About this guide 1
2 About the driver 1
3 Platform support 2
4 Windows 2
5 macOS 2
6 Linux 3
7 Authentication 4
8 Using a connection string 4
9 Driver configuration options 5
10 Third-party licenses 7

The Databricks ODBC Driver provides direct SQL access to Databricks SQL warehouses, enabling Business Intelligence (BI), analytics, and reporting tools to query Databricks over the ODBC API. This is the open-source Databricks ODBC driver, built on the shared Databricks SQL kernel.

This guide explains how to install and configure the driver on Windows, macOS, and Linux, and documents the connection properties it supports. For the list of changes and known issues in this release, see `release-notes.txt` alongside this guide.

1 About this guide

Purpose. This guide explains how to install and configure the Databricks ODBC Driver and documents its connection properties.

Audience. The guide is intended for end users of the driver, as well as administrators and developers integrating it into an application.

Knowledge prerequisites. Familiarity with the platform you are installing on, the role ODBC driver managers play in connecting to a data source, and basic SQL.

Conventions. `Monospace` indicates commands, source code, connection-string keys, or file contents. **Bold** marks user-interface elements and text you type.

2 About the driver

The driver implements the ODBC 3.8 API and connects to Databricks SQL warehouses over the Databricks Statement Execution API. It translates ODBC calls into Databricks SQL, streams results (including Apache Arrow and CloudFetch result sets), and presents catalog and schema metadata to ODBC applications. It is built on the shared Databricks SQL kernel, the same Rust engine behind the Databricks Python, Go, and Node drivers.

The driver is available for Microsoft Windows, macOS, and Linux. On Windows and macOS it is distributed as a single-architecture and universal installer respectively; on Linux it ships as `.deb` and `.rpm` packages for 64-bit x86, ARM64, and 32-bit x86.

3 Platform support

For the exact operating-system versions this release supports, see `release-notes.txt` . In summary, the driver supports current 64-bit and 32-bit Windows, universal (Apple silicon and Intel) macOS 13.3 and later, and Linux distributions built against glibc 2.28 or newer (Red Hat Enterprise Linux / AlmaLinux / Rocky Linux 8+, Amazon Linux 2023, Debian 10+, Ubuntu 20.04+, SLES 15) on x86_64, ARM64, and 32-bit x86. It connects to Databricks SQL warehouses over the Statement Execution API.

4 Windows

System requirements

- Administrator rights on the machine.
- The Visual C++ Redistributable for Visual Studio 2015-2022, matching the bitness of the driver (64-bit or 32-bit).
- Roughly 100 MB of available disk space.

64-bit applications require the 64-bit driver and 32-bit applications require the 32-bit driver; both may be installed side by side.

Installing the driver on Windows

1. Depending on the bitness of your application, run `DatabricksODBC64.msi` or `DatabricksODBC32.msi` .
2. Accept the license agreement and follow the installer prompts.
3. When installation completes, the driver is registered with the Windows ODBC Data Source Administrator.

Creating a Data Source Name (DSN) on Windows

1. Open **ODBC Data Sources** from the Start menu, choosing the administrator whose bitness matches your application.
2. On the **System DSN** or **User DSN** tab, click **Add** , select the Databricks ODBC driver, and click **Finish** .
3. Enter the connection properties (see *Driver configuration options*) and save.

You can also connect without a DSN by passing a full connection string, or through a File DSN. Settings in a connection string take precedence over DSN settings.

5 macOS

System requirements

- Administrator rights on the machine.
- An ODBC driver manager. macOS ships iODBC, which the driver targets.

Installing the driver on macOS

1. Open the `.pkg` installer from the macOS distribution ZIP and follow the prompts.
2. The driver installs under `/Library/databricks/databricksodbc/` , with the driver at `/Library/databricks/databricksodbc/lib/libdatabricksodbc.dylib` .
3. Register the driver with your driver manager by adding a stanza to `odbcinst.ini` (a sample is installed under the `Setup` folder).

6 Linux

System requirements

- Root (or `sudo`) access to install the package.
- A driver manager -- unixODBC (which provides `libodbcinst.so.2`).

Installing the driver on Linux

Install the package for your distribution and architecture:

```
# Debian / Ubuntu
sudo dpkg -i databricksodbc_<version>_amd64.deb

# RHEL / AlmaLinux / Rocky / Amazon Linux / SLES
sudo rpm -i databricksodbc-<version>-1.x86_64.rpm
```

The driver installs under `/opt/databricks/databricksodbc/` , with the 64-bit driver at `/opt/databricks/databricksodbc/lib/64/libdatabricksodbc64.so` (and the 32-bit driver at `lib/32/libdatabricksodbc32.so`).

Registering the driver and creating a DSN

Register the driver once with `odbcinst` , using the sample under the install `Setup` folder:

```
sudo odbcinst -i -d -f /opt/databricks/databricksodbc/Setup/odbcinst.ini
```

Then define a DSN in `~/.odbc.ini` (or the system `odbc.ini`):

```
[Databricks]
Driver      = /opt/databricks/databricksodbc/lib/64/libdatabricksodbc64.so
Host        = <workspace-hostname>
HTTPPath    = /sql/1.0/warehouses/<warehouse-id>
AuthMech    = 3
Token       = <personal-access-token>
```

Testing the connection

Use `isql` (bundled with unixODBC) to test a DSN:

```
isql -v Databricks
```

7 Authentication

Authentication is selected with the `AuthMech` property:

- `AuthMech=3` -- Personal Access Token (PAT). Set `Token` to a Databricks personal access token. This is the default when `AuthMech` is omitted.
- `AuthMech=11` -- OAuth 2.0. Select the flow with `Auth_Flow` :
  - `Auth_Flow=0` -- token pass-through (an externally obtained access token in `Auth_AccessToken`).
  - `Auth_Flow=1` -- client credentials (machine-to-machine). Set `Auth_Client_Id` and `Auth_Client_Secret` . Set `UseJWTAssertion=1` for a JWT private-key client assertion instead of a secret. For commercial Azure Entra service-principal authentication, also set `OIDCDiscoveryEndpoint` or `Azure_workspace_resource_id` .
  - `Auth_Flow=2` -- browser-based (user-to-machine). The driver opens a browser for interactive login and caches the refresh token (see `EnableTokenCache`).

For commercial Azure Entra service-principal M2M, set at least one of:

- `OIDCDiscoveryEndpoint=https://login.microsoftonline.com/<tenant>/v2.0/.well-known/openid-co` to supply the tenant explicitly. Other endpoint shapes return `HY024` .
- `Azure_workspace_resource_id=<workspace ARM resource ID>` to let the kernel discover the tenant and send the Azure management-token headers needed for an Azure-RBAC-authorized service principal without workspace membership.

Both properties may be supplied together.

8 Using a connection string

A DSN-less connection passes all properties in the connection string. For example, using a personal access token:

```
Driver=Databricks;Host=<workspace-hostname>;HTTPPath=/sql/1.0/warehouses/<id>;AuthMech=3;Token=<pat>
```

Using OAuth browser-based login:

```
Driver=Databricks;Host=<workspace-hostname>;HTTPPath=/sql/1.0/warehouses/<id>;AuthMech=11;Auth_Flow=2
```

9 Driver configuration options

The following properties may be set in a DSN or a connection string. Keys are case-insensitive.

Connection

Key	Description
`Host`	Workspace hostname (no scheme).
`Port`	Server port. Defaults to 443.
`HTTPPath`	The HTTP path of the SQL warehouse or cluster.
`Catalog`	Default catalog, applied at connect.
`Schema`	Default schema, applied at connect.
`SSL`	`1` (default) for TLS; `0` for plaintext HTTP.

Authentication

Key	Description
`AuthMech`	`3` for PAT (default), `11` for OAuth.
`Token` / `PWD`	Personal access token (with `AuthMech=3`).
`Auth_Flow`	OAuth flow: `0` pass-through, `1` client credentials, `2` browser.
`Auth_AccessToken`	Externally obtained access token (`Auth_Flow=0`).
`Auth_Client_Id` / `Auth_Client_Secret`	OAuth client credentials (`Auth_Flow=1`).
`Auth_Scope`	Requested OAuth scope(s).
`UseJWTAssertion`	`1` to use a JWT private-key client assertion (M2M).
`Auth_KID` / `Auth_JWT_Key_File` / `Auth_JWT_Passphrase`	JWT signing key material.
`OIDCDiscoveryEndpoint`	Commercial Azure Entra discovery URL; under `Auth_Flow=1`, selects Azure service-principal authentication and supplies the tenant.
`Azure_workspace_resource_id`	Optional Azure workspace ARM resource ID; also selects Azure service-principal authentication and enables Azure RBAC authorization.
`EnableTokenCache`	`1` (default) to persist the OAuth U2M refresh token; `0` to disable.
`TokenCachePassPhrase`	Passphrase for the encrypted token cache.

TLS and certificates

Key	Description
`TrustedCerts`	Path to a PEM CA bundle added on top of the system roots.
`UseSystemTrustStore`	Use the OS trust store.
`AllowSelfSignedServerCert`	Accept a self-signed server certificate.
`AllowHostNameCNMismatch`	Allow a certificate CN / hostname mismatch.
`TwoWaySSL` / `ClientCert` / `ClientPrivateKey`	Mutual-TLS client credentials.

Proxy

Key	Description
`UseProxy`	`1` to route through an HTTP proxy.
`ProxyHost` / `ProxyPort`	Proxy host and port.
`ProxyUID` / `ProxyPWD`	Proxy credentials.
`ProxyAuthType`	Proxy authentication type.
`ProxyIgnoreList`	Comma-separated hosts that bypass the proxy.

Logging

Key	Description
`LogLevel`	`0` (off) to `6` (trace).
`LogPath` / `LogFile`	Log file destination.

Timeouts and retries

Key	Description
`SocketTimeout`	Per-request HTTP timeout, in seconds.
`ConnectionTestTimeout`	Active connection-validation timeout, in seconds.
`RateLimitRetry`	Number of retry attempts on rate-limit responses.
`RateLimitRetryTimeout`	Overall retry budget, in seconds.
`RetryWaitMin` / `RetryWaitMax`	Backoff bounds between retries, in seconds.

Metadata and behavior

Key	Description
`DefaultStringColumnLength`	Reported length for string columns.

`BinaryColumnLength`	Reported length for binary columns.
`UseUnicodeSqlCharacterTypes`	Report `SQL_WVARCHAR` / `SQL_WCHAR` for string columns.
`IgnoreTransactions`	`1` (default) to no-op transaction control; `0` to enable transactions.
`EnableMetricViewMetadata`	`1` to surface Unity Catalog metric views in metadata.
`QueryTags`	Session-level query tags for cost attribution.
`MaxChunksInMemory`	CloudFetch chunks held in memory at once.
`UserAgentEntry`	Identity appended to the outgoing User-Agent.
`HTTPHeaders`	Extra HTTP headers (newline-separated `Name: Value` pairs).

Server-side session properties can be passed with the `SSP_<key>=<value>` prefix, and per-request custom HTTP headers with the `http.header.<Name>=<Value>` prefix.

10 Third-party licenses

Information about the third-party components statically linked into the driver is in the `NOTICE`, `third-party-licenses.txt`, and `THIRD_PARTY_RUST_LICENSES.txt` files packaged with the software.