databricks

OAuth Integration guide for Cloud-Based Partners with Databricks

Version	V 1.2.3 (Last updated on Jan 19, 2024)

Table of Contents

Introduction

This document is a guide for OAuth Integration with ISV partners using DBSQL drivers. If you intend to use DBSQL drivers and Databricks Rest APIs you can modify your scopes to authenticate once.

There are two main scenarios: three-legged OAuth and two-legged OAuth integration.

Three-legged OAuth in cloud-based partners utilizing DBSQL drivers:

This is also known as User-to-machine (U2M) authentication. For U2M OAuth in cloud-based partners, opening a pop-up browser and redirecting due to security reasons must happen in the business logic of the partner application. This guide explains best practices how to achieve that.

- **Create an OAuth Application**

  The partner should provide the guide for registering an OAuth application to Databricks+Partner joint customers. Customers will create an OAuth application in Databricks (link)
  -
- **U2M code flow implementation:** (hosting web server, browser pop up, etc) needs to be implemented by the cloud-based partner in their business logic. Once partner business logic acquired an access-token, it should pass the token to the DBSQL driver (link)
- **Refresh token flow:** refresh token flow must be implemented in the partner business logic. Once a new token is acquired, the partner business logic must invoke the relevant DBSQL driver api for refreshing access-token (link)
- **Caching or persisting the token**: Caching/persisting token has to happen in the partner business logic, as a cloud based application may have many users, the token should be cached/persisted for each user independently in user's session (databricks workspace host, userid) (link)

We also have a sample python getting started guide for U2M OAuth desktop application for Databricks on AWS:

https://github.com/databricks/databricks-sdk-py/blob/main/examples/flask_app_with_oauth.py

Two-legged OAuth in cloud-based partners utilizing DBSQL drivers:

This is also known as Machine-to-machine (M2M) authentication.
Cloud based OAuth with M2M applications are not different from desktop application OAuth M2M, this guide covers the M2M scenario for the sake of completeness:

- **Create Service Principal**
  Joint customers of Databricks+Partner will create a service principal in their Databricks account cloud (link to the guide)
  - **Azure Databricks:** customer admin creates SP in AAD (link)
  - **Databricks in AWS**: customer admin creates SP using Databricks Admin rest api (link)
- Partner business logic passed SP (ClientId and ClientSecret) to the DBSQL driver
- **M2M code flow implementation:** (link)
  - DBSQL driver internally will have implementation for the M2M flow.
  - Open source drivers will use M2M flow implemented by the DECO team
  - Simba JDBC/ODBC drivers will use M2M flow implemented by simba

Why OAuth (OIDC)?

OAuth is preferred over personal access token (PAT)-based or username/password authentication in many situations due to several key advantages it offers. These advantages include:

- Security: OAuth provides a more secure way of granting access to resources without sharing the user's actual credentials (username and password) or PAT tokens. Instead, it utilizes access tokens, which are short-live (less than and can be limited in scope, reducing the risk of unauthorized access or data breaches.

- Standardization: OAuth is a widely-accepted industry standard for authorization, making it easier for developers to implement and maintain.

- PATs have an expiration time and must be manually rotated before they expire. In contrast, OAuth authentication doesn't have this issue, as it uses access tokens with refresh tokens to maintain continuous access.

Different types of OAuth Databricks DBSQL API Support

We support User-to-Machine and Machine-to-Machine OAuth for SqlWarehouse APIs:

- **Three-legged OAuth with Databricks DBSQL API:**

This is also known as User-to-machine (U2M) authentication.
U2M interactions in Databricks DBSQL API involve users working directly with the API to perform tasks such as executing SQL queries, managing clusters, and creating or modifying databases and tables. Users typically interact with the API through BI Tools, programming language libraries, or custom-built applications. U2M interactions are essential for data scientists, engineers, and analysts who use the Databricks platform for data processing, analysis, and machine learning tasks.

- **Two-legged OAuth with Databricks DBSQL API:**
  This is also known as Machine-to-machine (M2M) authentication.
  M2M interactions with the Databricks DBSQL API involve automated systems, services, or applications communicating with the Databricks platform without direct human intervention. This typically includes tasks like automating data ingestion, triggering data processing pipelines, or synchronizing data between systems. M2M interactions are commonly used in scenarios where multiple systems or applications need to work together to achieve a desired outcome, such as ETL pipelines, data monitoring, and orchestration of complex workflows.

Databricks Support matrix on Service side

	Azure Databricks	Databricks on AWS	Databricks on GCP
U2M OAuth	GA	GA	Not supported yet
M2M OAuth	GA	GA	Not supported yet

User-to-Machine OAuth in Cloud-based partner applications

User-to-Machine OAuth requires the application to open a browser pop up for users to interactively log in. Hence the cloud-based partner application must implement U2M flow on their end because Browser Pop up must be opened on the cloud partner application.

Registering an OAuth application for U2M in Databricks Account

Registering an OAuth application requires registering an OAuth client-id, redirect-url and optionally a client-secret. You should determine what your OAuth redirect-url is and whether it needs a client-secret or not.

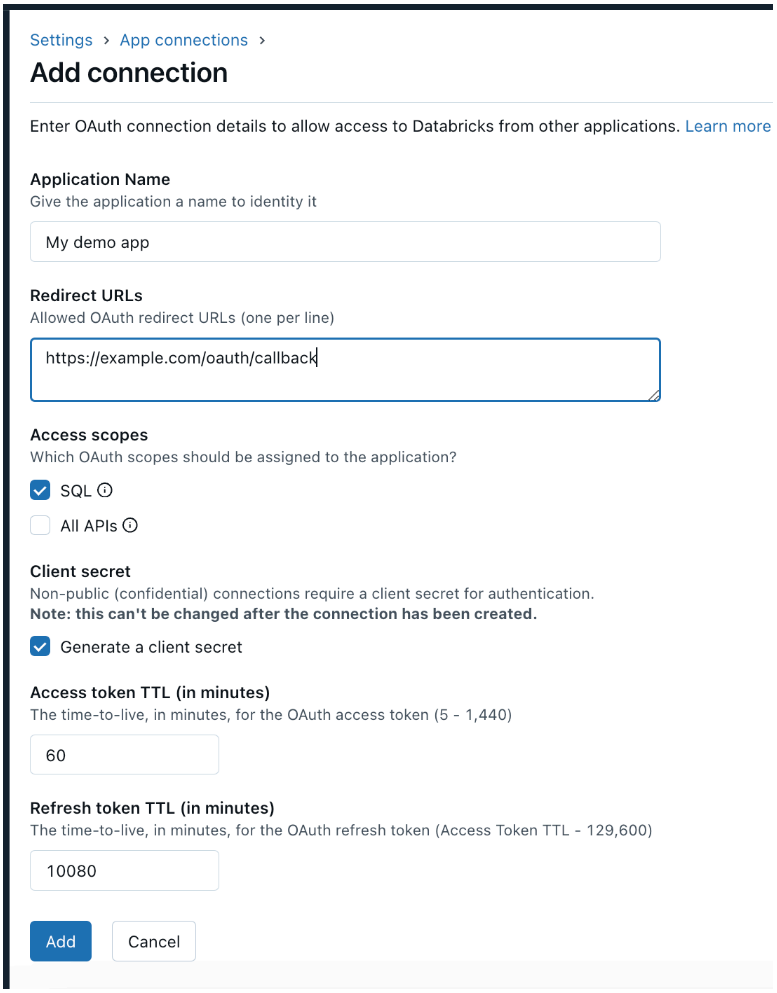
Registering an OAuth application for U2M in Databricks

Registering an OAuth application from Databricks Account Console

You can register an OAuth application for U2M in your account from **Databricks Account Console.**

Here are the steps:

1. Login to Databricks Account Console
   - AWS: https://accounts.cloud.databricks.com
   - GCP: https://accounts.gcp.databricks.com
   - Azure: https://accounts.azuredatabricks.net
2. Goto **Settings | App connections**
3. Click "**Add connection**"
4. Enter the application name and redirect URLs, and leave other fields as default
5. Click "**Add**" to create your OAuth application
6. A dialog "Connection created" will popup, please copy the "**Client ID"** and "**Client Secret**" in the dialog and store them somewhere as you won't be able to see the "**Client Secret**" again.



Settings › App connections ›

Add connection

Enter OAuth connection details to allow access to Databricks from other applications. Learn more

Application Name
Give the application a name to identity it

My demo app

Redirect URLs
Allowed OAuth redirect URLs (one per line)

https://example.com/oauth/callback

Access scopes
Which OAuth scopes should be assigned to the application?

☑ SQL ⓘ

□ All APIs ⓘ

Client secret
Non-public (confidential) connections require a client secret for authentication.
Note: this can't be changed after the connection has been created.

☑ Generate a client secret

Access token TTL (in minutes)
The time-to-live, in minutes, for the OAuth access token (5 - 1,440)

60

Refresh token TTL (in minutes)
The time-to-live, in minutes, for the OAuth refresh token (Access Token TTL - 129,600)

10080

The following scopes are automatically granted to the application.

- ***openid, email, profile:*** Required to generate the ID token.
- ***offline_access***: Required to generate refresh tokens.

Note: It only registers the application in your account. You have to ask customers to register your application in their Databrick accounts if they want to use your application.

Registering on OAuth application by REST API

We also provide Admin rest API for registering an OAuth application:

To authenticate to the Account API, you can use Databricks OAuth tokens for service principals or an account admin's username and password. Databricks strongly recommends that you use OAuth tokens for service principals. A service principal is an identity that you create in Databricks for use with automated tools, jobs, and applications. To create an OAuth token, see Authentication using OAuth tokens for service principals.

Pass the OAuth token in the header using Bearer authentication. For example:

Unset

```
export OAUTH_TOKEN=<oauth-access-token>

curl -X GET --header "Authorization: Bearer $OAUTH_TOKEN" \
'https://accounts.cloud.databricks.com/api/2.0/accounts/<accountId>/<endpoint>'
```

Run the following command (if you need an OAuth client-secret you need confidential to be set to true, otherwise false) to register the OAuth application. You need to add scope "**sql**" (required scope for DBSQL API) and "**offline_access**" (required scope for getting refresh token) to the "**scopes**" field in the request payload (see example below).

Unset

```
curl -X POST -d '{ "redirect_urls" : [ "<Redirect URL>" ], "confidential" : true|false, "name" : "<Name>", "scopes": [ "<scopes for the app>" ] }' https://accounts.cloud.databricks.com/api/2.0/accounts/<AccountID>/oauth2/custom-app-integrations --header "Authorization: Bearer $OAUTH_TOKEN"
```

Example:

Unset

```
curl -X POST -d '{"redirect_urls":[
"https://example-partner.com/redirecturl1",
"https://example-partner.com/redirecturl2"], "confidential":true, "name":
"example-partner","scopes":["sql","offline_access"]}'
https://accounts.cloud.databricks.com/api/2.0/accounts/123e4567-e89b-12d3-a
456-426614174000/oauth2/custom-app-integrations --header "Authorization:
Bearer $OAUTH_TOKEN"
```

The execution of this will register the oauth-app and generate a unique OAuth client-id and in case you used `confidential=true` an OAuthclient secret will be generated for you.
Sample output:

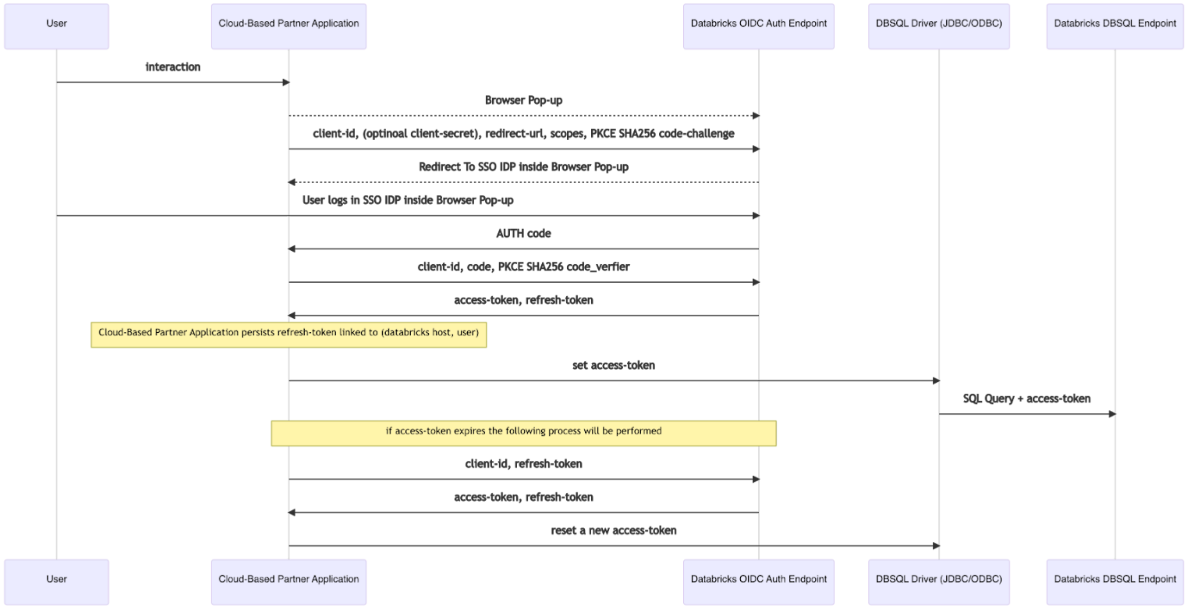
Unset

```
{"integration_id":"<Integration ID>","client_id":"<Client
ID>","client_secret":"<Client secret>"}
```

You should collect the OAuth client id and OAuth client-secret as you won't be able to see it later

Configuration and endpoints for U2M

A cloud-based partner application will need to implement the OAuth U2M flow to acquire an access token, that can then be used with the DBSQL driver (JDBC/ODBC.) If the partner application is a SaaS application, it would also need to handle multi-tenancy, such as introducing a different callback endpoint or passing in the state parameter to the OAuth flow.


User
Cloud-Based Partner Application
Databricks OIDC Auth Endpoint
DBSQL Driver (JDBC/ODBC)
Databricks DBSQL Endpoint
interaction
Browser Pop-up
client-id, (optinoal client-secret), redirect-url, scopes, PKCE SHA256 code-challenge
Redirect To SSO IDP inside Browser Pop-up
User logs in SSO IDP inside Browser Pop-up
AUTH code
client-id, code, PKCE SHA256 code_verfier
access-token, refresh-token
Cloud-Based Partner Application persists refresh-token linked to (databricks host, user)
set access-token
SQL Query + access-token
if access-token expires the following process will be performed
client-id, refresh-token
access-token, refresh-token
reset a new access-token
User
Cloud-Based Partner Application
Databricks OIDC Auth Endpoint
DBSQL Driver (JDBC/ODBC)
Databricks DBSQL Endpoint

To help implement the OAuth Code flow, follow the sample implementation: https://www.stefaanlippens.net/oauth-code-flow-pkce.html#Connect-to-authentication-provider (if you optionally have client-secret, that will be the additional parameter you need to manage)

With PKCE, even if a malicious attacker intercepts the Authorization Code, they cannot exchange it for a token without possessing the Code Verifier.

PKCE	Description	Partner Application must do
code_challenge_method	PKCE option	S256 or plain (Default is plain if it is not specified)
code_challenge	PKCE option	Generate based on PKCE S256 method
code_verifier	PKCE option	Generate based on PKCE S256 method

OAuth U2M OIDC endpoints:

You can use the .well-known endpoint, https://{databricks-host}/oidc/.well-known/openid-configuration, to get the OAuth endpoints or alternatively use the following table:

Description	Databricks in AWS	Databricks in Azure
Databricks OIDC Endpoint Prefix	https://{databricks-host}/oidc	https://{databricks-host}/oidc
Token URL	{OIDC-ENDPOINT}/v1/token	{OIDC-ENDPOINT}/oauth2/v2.0/token
Authorize URL	{OIDC-ENDPOINT}/v1/authorize	{OIDC-ENDPOINT}/oauth2/v2.0/authorize

Scopes for Request

Description	Databricks in AWS	Databricks in Azure
scopes	“sql offline_access”	“2ff814a6-3304-4ab8-85cb-cd0e6f879c1d/user_impersonation offline_access”

JDBC/ODBC Driver Integration

After the completion of the OAuth code flow, you will acquire an OAuth access-token, pass that to the JDBC/ODBC driver as following:

NOTE: JDBC and ODBC drivers are already integrated with Databricks OAuth for AWS. Partner applications are encouraged to use them.

JDBC driver (2.6.22 version or above):

Unset

```
jdbc:
databricks
://example.cloud.databricks.com:443/yourDatabricksHttpPath;AuthMech=11;Auth_Flo
w=0;Auth_AccessToken=YOUR_OAUTH_ACCESS_TOKEN
```

ODBC driver:

Unset

```
Host=<server-hostname>;Port=443;HTTPPath=<http-path>;AuthMech=11;Auth_Flow=0;
Auth_AccessToken=YOUR_OAUTH_ACCESS_TOKEN
```

Refreshing Token

OAuth Access tokens are valid for a limited time (by default, 1 hour). For running new queries or for handling long running queries, the cloud based partner application must refresh the token in their business logic and set the new refreshed access token in the JDBC/ODBC driver.

The partner business logic must refresh the OAuth token and invoke the following JDBC driver API to set the new token in the JDBC driver:

Unset

```
Connection.setClientInfo("Auth_AccessToken", "YOUR_NEW_ACCESS_TOKEN")
```

Please note that as JDBC driver APIs are blocking you may need to invoke the connection#setClientInfo() API on a different thread. If the token is expected to be valid for the time t, you can use a different thread which at time t/2 sets the refreshed OAuth access token to the JDBC driver.

For the ODBC side, call SQLSetConnectAttr functions twice. The first one is to update the Auth_AccessToken, and the second one is to refresh the current connection.

Unset

```
char *credentials = "Auth_AccessToken=$(new token)"
SQLSetConnectAttr(dbc, 122, credentials, SQL_NTS); // 122 is Custom ODBC property:
SQL_ATTR_CREDENTIALS

__int32 refreshMode = -1; // Refresh now
SQLSetConnectAttr(dbc, 123, reinterpret_cast<SQLPOINTER>(refreshMode),
SQL_IS_SMALLINT); // 123 is custom ODBC property: SQL_ATTR_REFRESH_CONNECTION
```

Persistence/caching of the Tokens

The OAuth refresh token is long-lived. The user's OAuth refresh token should be persisted/cached in the business logic of the cloud-based partner application to ensure the user does not need to repeat the OAuth U2M re-login.

Cloud-based applications are typically used by multiple users at the same time. Hence the application should be able to persist OAuth refresh tokens for multiple users. For example the OAuth tokens for the user can be persisted on the cloud-based service side linked to their session.

One proposed persistence is to scope tokens such that for each (Databricks-workspace-host, user) tuple we store tokens independently.

Machine-to-Machine OAuth in Cloud-based partner applications

Registering an OAuth application for M2M in Databricks Account

Creating a Service Principal for Azure Databricks

You need an Azure Databricks account with access to its corresponding AAD tenant for creating a SP application and assigning it to your Azure Databricks workspace. Follow these steps:

1. “Add a service principal to your Azure Databricks account” as explained here https://learn.microsoft.com/en-us/azure/databricks/administration-guide/users-groups/service-principals#—add-a-service-principal-to-your-azure-databricks-account
2. “Add service principals to your account using the account console” as explained here https://learn.microsoft.com/en-us/azure/databricks/administration-guide/users-groups/service-principals#add-service-principals-to-your-account-using-the-account-console
3. “Assign a service principal to a workspace using the account console” as explained here https://learn.microsoft.com/en-us/azure/databricks/administration-guide/users-groups/service-principals#assign-a-service-principal-to-a-workspace-using-the-account-console

Creating a Service Principal for Databricks in AWS

Create a Service Principal in your Databricks account in AWS using this guide

Databricks Service Principal OAuth Token feature supports the OAuth 2.0 Client Credentials Grant and allows you to securely generate OAuth access tokens on behalf of your Databricks service principals.

You can use Databricks service principal OAuth access tokens in your backend jobs to talk to Databricks Accounts and Workspaces APIs. Those OAuth access tokens carry the identities of their respective service principals. Their access to Databricks APIs and resources is subject to service principal permission checks.

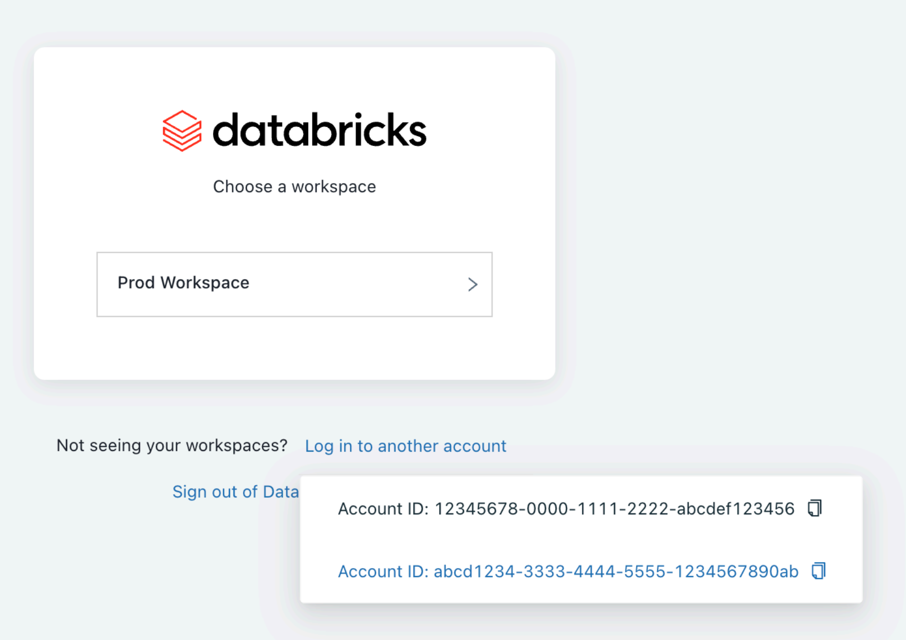
Prerequisites

- This public preview only supports Databricks on AWS.

Login to your Databricks account

- Login to your Databricks account https://accounts.cloud.databricks.com/login?account_id=<YOUR_ACCOUNT_ID> .

- If you have multiple accounts, use **Log in to another account** and select the right one for the private preview.



Choose a workspace

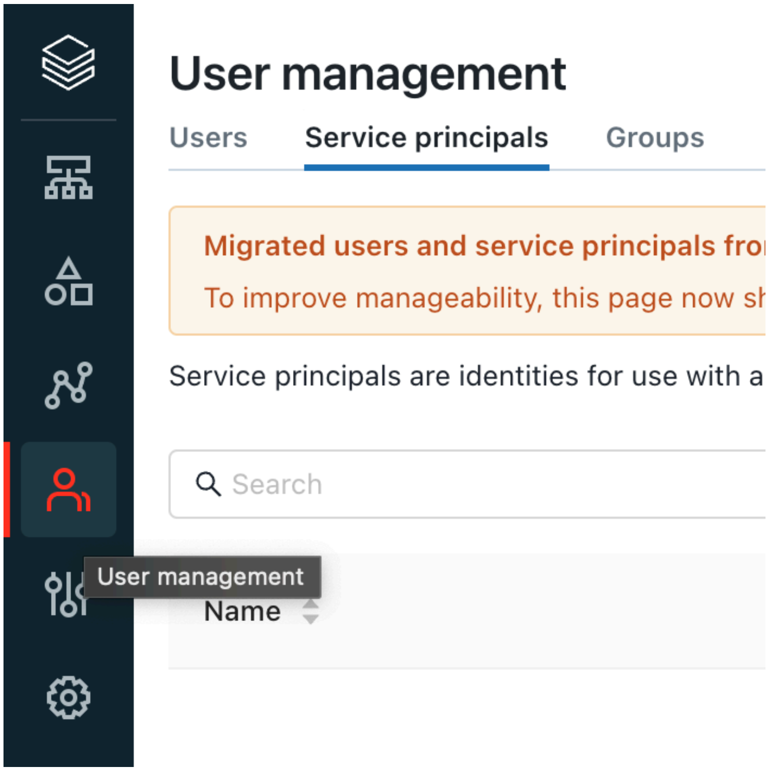
Not seeing your workspaces?

Account ID: 12345678-0000-1111-2222-abcdef123456

Account ID: abcd1234-3333-4444-5555-1234567890ab

Create a service principal

- From the Account Console, select **User Management** from the leftnav.

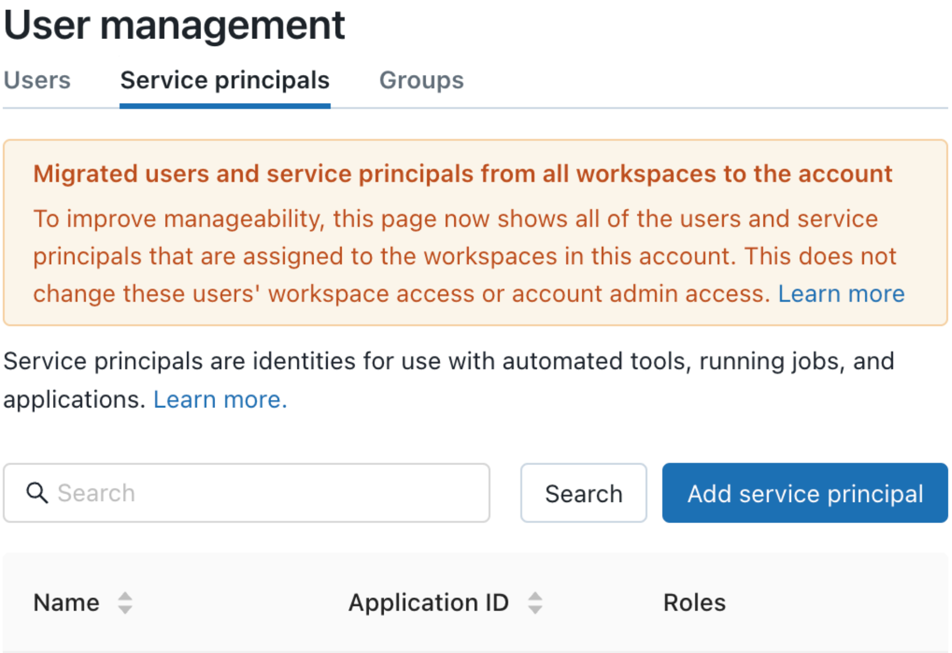


Migrated users and service principals fro

To improve manageability, this page now sh

Service principals are identities for use with a

- From the **Service Principals** tab, click on **Add service principal**.



User management

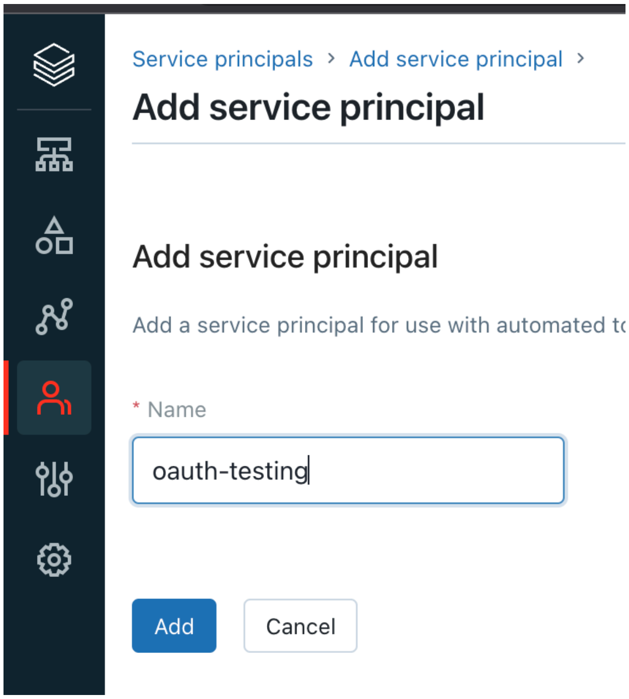
Migrated users and service principals from all workspaces to the account

To improve manageability, this page now shows all of the users and service principals that are assigned to the workspaces in this account. This does not change these users' workspace access or account admin access. Learn more

Service principals are identities for use with automated tools, running jobs, and applications. Learn more.

Name	Application ID	Roles

- Enter a name for the service principal and click on **Add**.



Add service principal

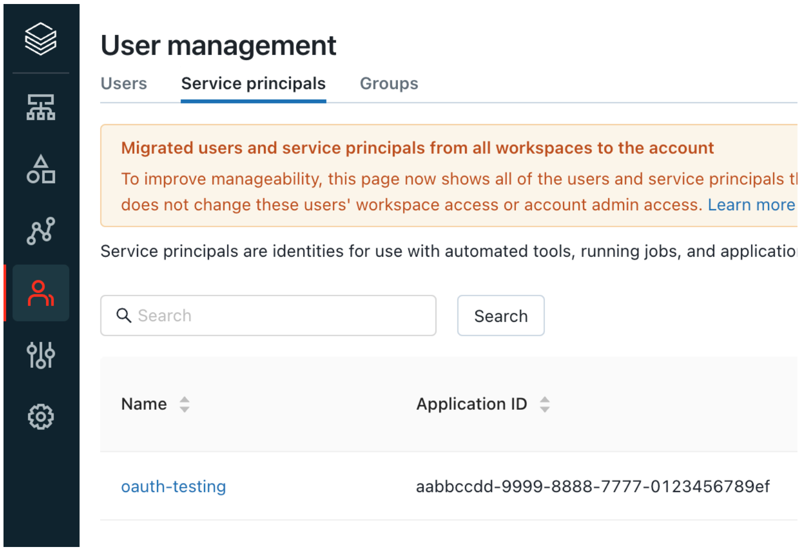
Add service principal

Add a service principal for use with automated t

* Name: oauth-testing

Create a service principal secret

- Select the service principal you just created



User management

Users | Service principals | Groups

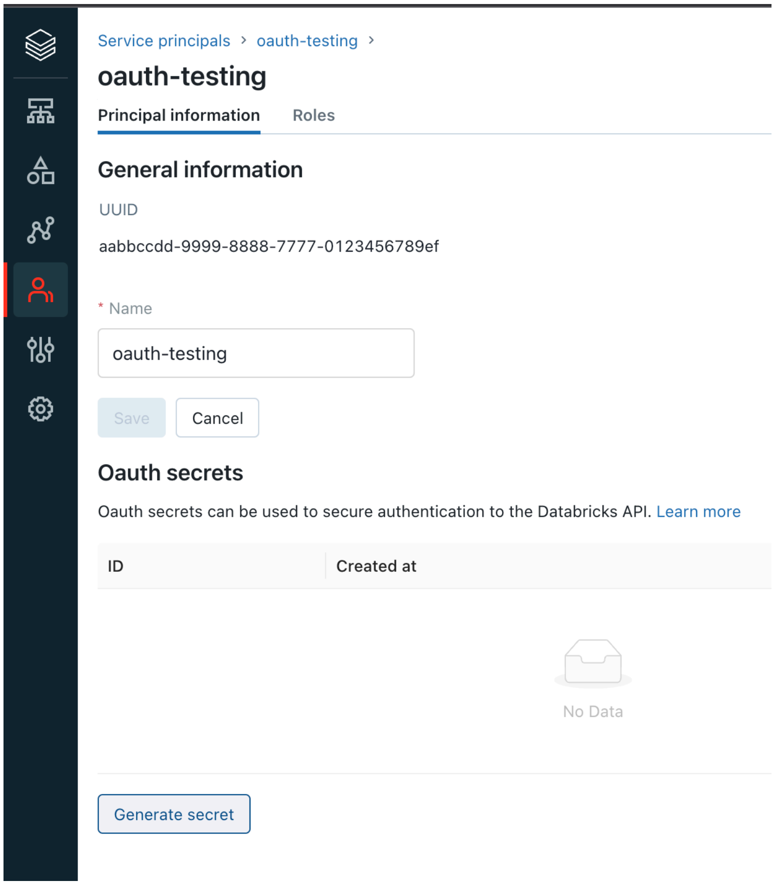
Migrated users and service principals from all workspaces to the account

To improve manageability, this page now shows all of the users and service principals tl does not change these users' workspace access or account admin access. Learn more

Service principals are identities for use with automated tools, running jobs, and applicatio

Name	Application ID
oauth-testing	aabbccdd-9999-8888-7777-0123456789ef

- Click on **Generate secret**



Service principals › oauth-testing ›

oauth-testing

Principal information | Roles

General information

UUID

aabbccdd-9999-8888-7777-0123456789ef

* Name

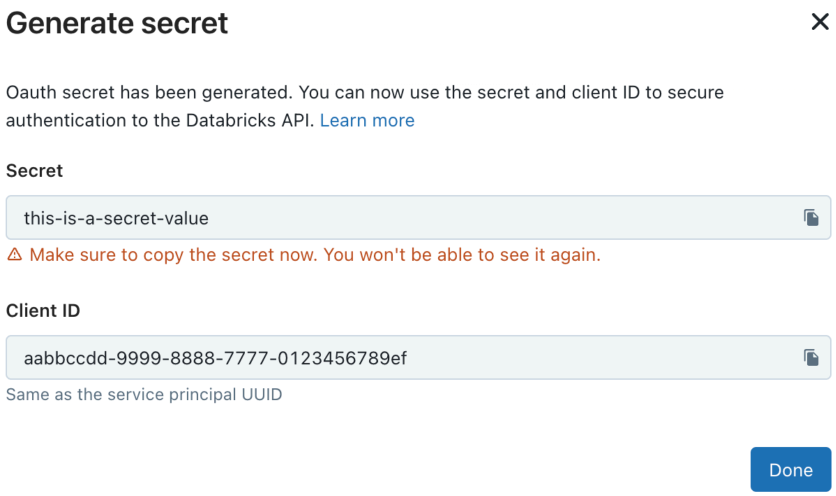
oauth-testing

Oauth secrets

Oauth secrets can be used to secure authentication to the Databricks API. Learn more

ID	Created at

- Copy the **Client ID** and **Secret** from the pop-up window. The secret will only be revealed **once** during creation.



Generate secret

Oauth secret has been generated. You can now use the secret and client ID to secure authentication to the Databricks API. Learn more

Secret: this-is-a-secret-value

⚠ Make sure to copy the secret now. You won't be able to see it again.

Client ID: aabbccdd-9999-8888-7777-0123456789ef

Same as the service principal UUID

Assign the service Principal to the workspace

- You may need to first assign the service principal to a workspace, grant permissions or assign admin roles.

Native Support for Service Principal in JDBC/ODBC drivers

The native support for Service Principal in JDBC/ODBC drivers is expected to land in 2023. This is the recommended path.

The new JDBC/ODBC drivers config for supporting Service Principal is expected to be as following

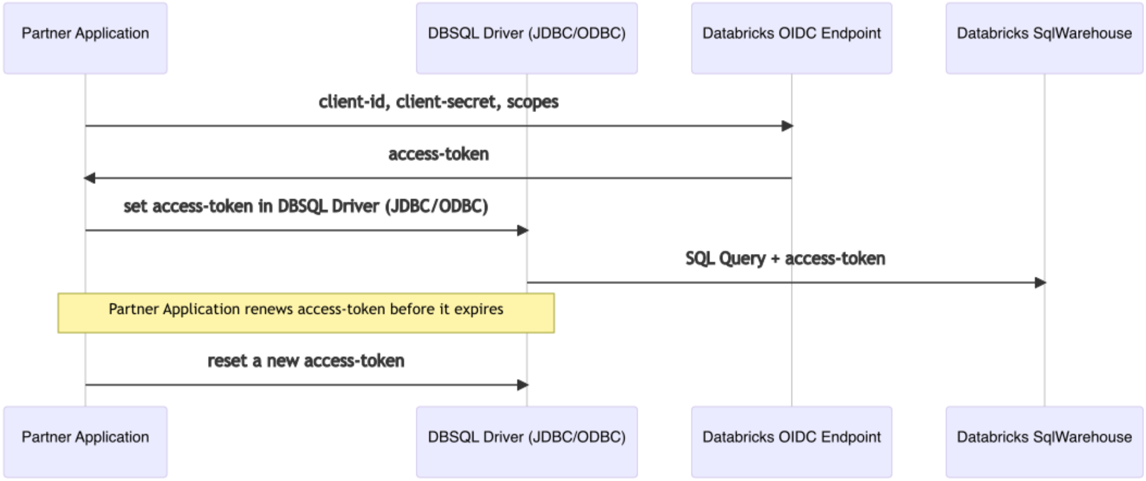
```
Unset
Host=<server-hostname>;HTTPPath=<http-path>;Auth_Client_ID=<SP-clientId>;
Auth_Client_Secret=<SP-ClientSecret>;Auth_Type=OAuth_2.0
```

Please note that with the native support for Service Principal in JDBC/ODBC drivers, the partner application can rely on the native support in JDBC/ODBC and only pass Service-Principal ClientID and ClientSecret to the JDBC/ODBC Driver.

Service Principal support without relying on Native support in JDBC/ODBC

The native support for Service Principal in JDBC/ODBC drivers is expected to land in 2023, and ideally the partner application should rely on the native support.

However if you intend to integrate immediately or not rely on the native support for Service Principal in JDBC/ODBC driver you can implement M2M flow inside your partner-application and pass the OAuth access-token to the DBSQL Driver.



Implementation

Identify Databricks is in which cloud

Rely on the databricks hostname to identify if it is in the AWS cloud or Azure cloud or elsewhere. If the host is not in Azure or AWS and SP is used, throw an error.

“Service Principal not supported for Databricks in this Cloud”

Azure endpoint	AWS endpoint

".azuredatabricks.net", ".databricks.azure.cn”, ".databricks.azure.us”	“.cloud.databricks.com”

Use ServicePrincipal credentials to get a token

Token generation is slightly different for Databricks in different clouds.

Token Generation for Databricks in AWS

ODBC/JDBC driver should invoke the following Https POST to get an OAuth token for Databricks in AWS cloud:

Unset

```
POST https://<databricks-host>/oidc/v1/token
headers:
'accept: application/json'
"authorization: Basic encodeBase64($CLIENT_ID:$CLIENT_SECRET)
'cache-control: no-cache'
'content-type: application/x-www-form-urlencoded'

data: 'grant_type=client_credentials&scope=all-apis'
```

Sample output:

Unset

```
{"token_type":"Bearer","expires_in":3600,"access_token":"ey....."
,"scope":"all-apis"}
```

Sample CURL equivalent for on Mac for testing

Unset

```
CLIENT_ID="REPLACEME"
CLIENT_SECRET="REPLACEME"
```

```
curl --request POST \
--url https://REPLACEME.cloud.databricks.com/oidc/v1/token \
--header 'accept: application/json' \
--header "authorization: Basic $(echo -n $CLIENT_ID:$CLIENT_SECRET |
base64)" \
--header 'cache-control: no-cache' \
--header 'content-type: application/x-www-form-urlencoded' \
--data 'grant_type=client_credentials&scope=all-apis'
```

Token Generation for Databricks in Azure

The token generation request for Azure slightly differs from the token generation request for AWS.

ODBC/JDBC driver should invoke the following Https POST to get a token.

Unset

```
POST https://<databricks-host>/oidc/oauth2/v2.0/token
headers:'Content-Type: application/x-www-form-urlencoded'

data:
"client_id=$CLIENT_ID"
'grant_type=client_credentials'
'scope=2ff814a6-3304-4ab8-85cb-cd0e6f879c1d%2F.default'
"client_secret=$CLIENT_SECRET"
```

Sample output:

Unset

```
{"token_type":"Bearer","expires_in":3599,"ext_expires_in":3599,"access_token":"eyJ0e....."}
```

Sample CURL equivalent on Mac for testing:

Unset

```
CLIENT_SECRET="REPLACEME"
CLIENT_ID="REPLACEME"

curl -X POST -H 'Content-Type: application/x-www-form-urlencoded' \
https://REPLACEME.azuredatabricks.net/oidc/oauth2/v2.0/token \
-d "client_id=$CLIENT_ID" \
-d 'grant_type=client_credentials' \
-d 'scope=2ff814a6-3304-4ab8-85cb-cd0e6f879c1d%2F.default' \
-d "client_secret=$CLIENT_SECRET"
```

Use the generated token

Pass the generated token to the JDBC/ODBC driver in the connection string:

Unset

```
Host=<server-hostname>;Port=443;HTTPPath=<http-path>;AuthMech=11;Auth_Flow=0;
Auth_AccessToken=YOUR_OAUTH_ACCESS_TOKEN
```

Refreshing the Token

The generated token has a expiration time specified in the response payload to the token generation request:

```
Unset
{"token_type":"Bearer","expires_in":3600,"access_token":"ey....."
,"scope":"all-apis"}
```

Prior to token expiration, the partner application must generate a new token (scheduled task on a different thread) and reset the new token in JDBC/ODBC driver.

The new token generation can happen on a parallel thread and must be invoked prior to the expiry time which was specified in the token generation response.

Resetting token in JDBC driver:

```
Unset

Connection.setClientInfo("Auth_AccessToken", "YOUR_NEW_ACCESS_TOKEN")
```

Resetting token in ODBC driver:

```
Unset
char *credentials = "Auth_AccessToken=$(new token)"
SQLSetConnectAttr(dbc, 122, credentials, SQL_NTS); // 122 is Custom ODBC property:
SQL_ATTR_CREDENTIALS


__int32 refreshMode = -1; // Refresh now
SQLSetConnectAttr(dbc, 123, reinterpret_cast<SQLPOINTER>(refreshMode),
SQL_IS_SMALLINT); // 123 is custom ODBC property: SQL_ATTR_REFRESH_CONNECTION
```

Appendix : OAuth Support in DBSQL Drivers for Cloud Partners.

Note that for cloud partners, native OAuth support for interactive applications will not really be used. Cloud partners will do their heavy lifting to acquire and refresh OAuth token and pass on to the drivers to connect. The following table summarizes the status as of June 2023.

Driver	Ready for Cloud Integration	Note
ODBC	Yes	Follow installation and configuration guide. AuthMech=11;Auth_Flow=0;Auth_AccessToken=<token>.
JDBC	Yes	Follow installation and configuration guide. AuthMech=11;Auth_Flow=0;Auth_AccessToken=<token>.
Python Driver	Yes	Provide a credential provider like this example: https://github.com/databricks/databricks-sql-python/blob/main/examples/custom_cred_provider.py
GoLang Driver	Yes	Implement Authenticator interface. https://github.com/databricks/databricks-sql-go/blob/main/auth/auth.go
NodeJS Driver	Yes	Implement IAuthentication interface. ./lib/connection/contracts/IAuthentication.ts