Published in Towards Data Science

Dhyanendra Singh Rathore Follow
Sep 25, 2020 · 7 min read · · Listen

Mounting & accessing ADLS Gen2 in Azure Databricks using Service Principal and Secret Scopes

A guide on accessing Azure Data Lake Storage Gen2 from Databricks in Python with Azure Key Vault-backed Secret Scopes and Service Principal.

Photo by Markus Winkler on Unsplash

Azure Data Lake Storage and Azure Databricks are unarguably the backbones of the Azure cloud-based data analytics systems. Azure Data Lake Storage provides scalable and cost-effective storage, whereas Azure Databricks provides the means to build analytics on that storage.

The analytics procedure begins with mounting the storage to Databricks distributed file system (DBFS). There are several ways to mount Azure Data Lake Store Gen2 to Databricks. Perhaps one of the most secure ways is to delegate the Identity and access management tasks to the Azure AD.

This article looks at how to mount Azure Data Lake Storage to Databricks authenticated by Service Principal and OAuth 2.0 with Azure Key Vault-backed Secret Scopes.

Caution: *Microsoft Azure is a paid service, and following this article can cause financial liability to you or your organization.*

> At the time of writing, Azure Key Vault-backed Secret Scopes is in 'Public Preview.' It is recommended not to use any 'Preview' feature in production or critical systems.

Prerequisites

1. An active Microsoft Azure subscription
2. Azure Data Lake Storage Gen2 account
3. Azure Databricks Workspace (Premium Pricing Tier)
4. Azure Key Vault

If you don't have prerequisites set up yet, refer to our previous article to get started:

A definitive guide to turn CSV files into Power BI visuals using Azure
A step-by-step guide to turning COVID-19 data into stunning Power BI visuals using Microsoft Azure offerings.
medium.com

. . .

To access resources secured by an Azure AD tenant (e.g., storage accounts), a security principal must represent the entity that requires access. A security principal defines the access policy and permissions for a user or an application in the Azure AD tenant. **When an application is permitted to access resources in a tenant (e.g., upon registration), a service principal object is created automatically.**

Further reading on service principals:

Apps & service principals in Azure AD — Microsoft identity platform
This article describes application registration, application objects, and service principals in Azure Active Directory...
docs.microsoft.com

Let's begin by registering an Azure AD application to create a service principal and store our application authentication key in the Azure Key Vault instance.

Register an Azure AD application

Find and select **Azure Active Directory** on the Azure Portal home page. Select **App registrations** and click **+ New registration**.

Azure Active Directory: Register a new application (Image by author)

On the **Register an application** page, enter the name *ADLSAccess*, signifying the purpose of the application, and click **Register**.

Azure Active Directory: Name and register an application (Image by author)

In the **ADLSAccess** screen, copy the **Application (client) ID** and the **Directory (tenant) ID** into notepad. Application ID refers to the app we just registered (i.e., ADLSAccess), and the Azure AD tenant our app ADLSAccess is registered to is the Directory ID.

Next, we need to generate an authentication key (aka application password or client secret or application secret) to authenticate the ADLSAccess app. Click on **Certificates and secrets**, and then click **+ New client secret**. On the **Add a client secret** blade, type a description, and expiry of one year, click **Add** when done.

Azure Active Directory: Create a client secret (Image by author)

When you click on **Add**, the client secret (authentication key) will appear, as shown in the image below. You only have one opportunity to copy this key-value into notepad. **You will not be able to retrieve it later if you perform another operation or leave this blade**.

Azure Active Directory: Client secret (Image by author)

Grant service principal access to ADLS account

Next, we need to assign an access role to our service principal (recall that a service principal is created automatically upon registering an app) to access data in our storage account. Go to the Azure portal home and open the resource group in which your storage account exists. Click **Access Control (IAM)**, on **Access Control (IAM)** page, select **+ Add** and click **Add role assignment**. On the **Add role assignment** blade, assign the **Storage Blob Data Contributor** role to our service principal (i.e., ADLSAccess), as shown below.

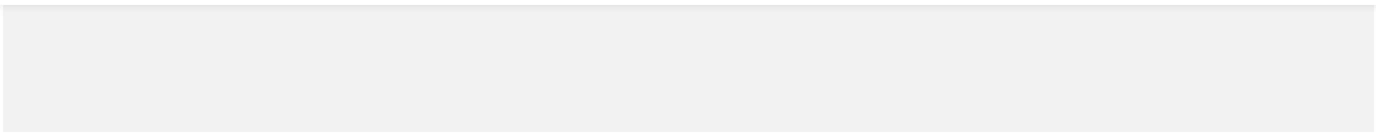
Resource group: Assign role to service principal (Image by author)

Add application secret to the Azure Key Vault

Go to the Azure portal home and open your key vault. Click **Secrets** to add a new secret; select **+ Generate/Import**. On **Create a secret** blade; give a **Name,** enter the client secret (i.e., **ADLS Access Key** we copied in the previous step) as **Value** and a **Content type** for easier readability and identification of the secret later. Repeat the creation process for the **Application (client) ID** and the **Directory (tenant) ID** we copied earlier. Your vault should have three secrets now.

Azure Key Vault: Add new secrets (Image by author)

Select **Properties**, copy the **Vault URI** and **Resource ID** to notepad; we will need them in the next step.

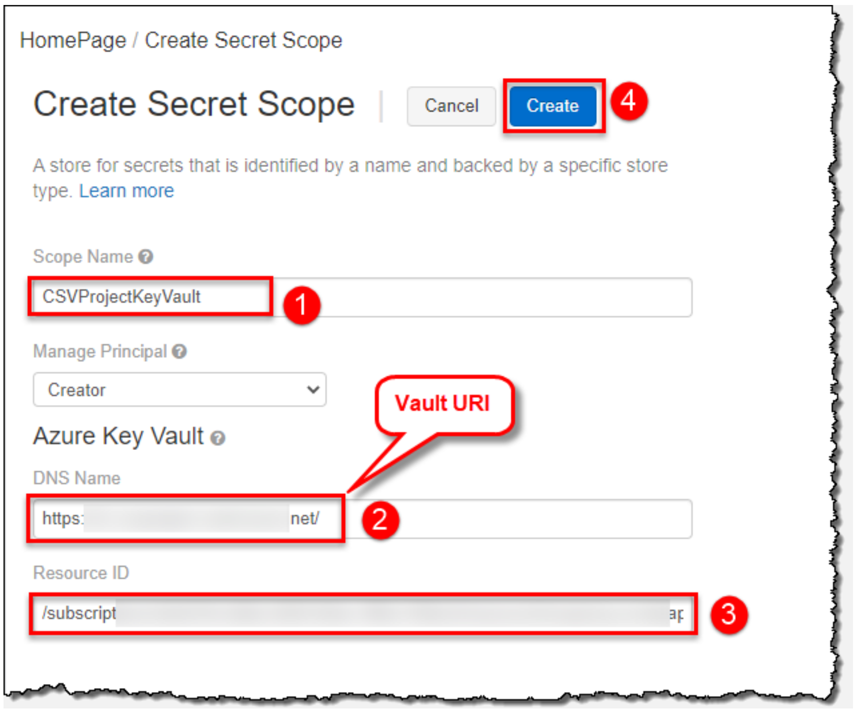

Azure Key Vault: Properties (Image by author)

Create an Azure Key Vault-backed Secret Scope in Azure Databricks

If you've followed our another article on creating a Secret Scope for Azure SQL Server credentials, you don't have to perform this step as long as your key vault and Databricks instance in question remains the same.

Go to `https://<DATABRICKS-INSTANCE>#secrets/createScope` and replace <DATABRICKS-INSTANCE> with your actual Databricks instance URL. Create a Secret Scope, as shown below.

> This URL is case sensitive.


HomePage / Create Secret Scope

Create Secret Scope

Cancel Create

A store for secrets that is identified by a name and backed by a specific store type. Learn more

Scope Name: CSVProjectKeyVault

Manage Principal: Creator

Azure Key Vault

DNS Name: https: net/

Resource ID: /subscript

Azure Databricks: Create a Secret Scope (Image by author)

Mount ADLS to Databricks using Secret Scope

Finally, it's time to mount our storage account to our Databricks cluster. Head back to your Databricks cluster and open the notebook we created earlier (or any notebook, if you are not following our entire series).

We will define some variables to generate our connection strings and fetch the secrets using Databricks utilities. You can copy-paste the below code to your notebook or type it on your own. We're using Python for this notebook. Run your code using controls given at the top-right corner of the cell. Don't forget to replace the variable assignments with your storage details and secret **Names**.

Further reading on Databricks utilities (dbutils) and accessing secrets:

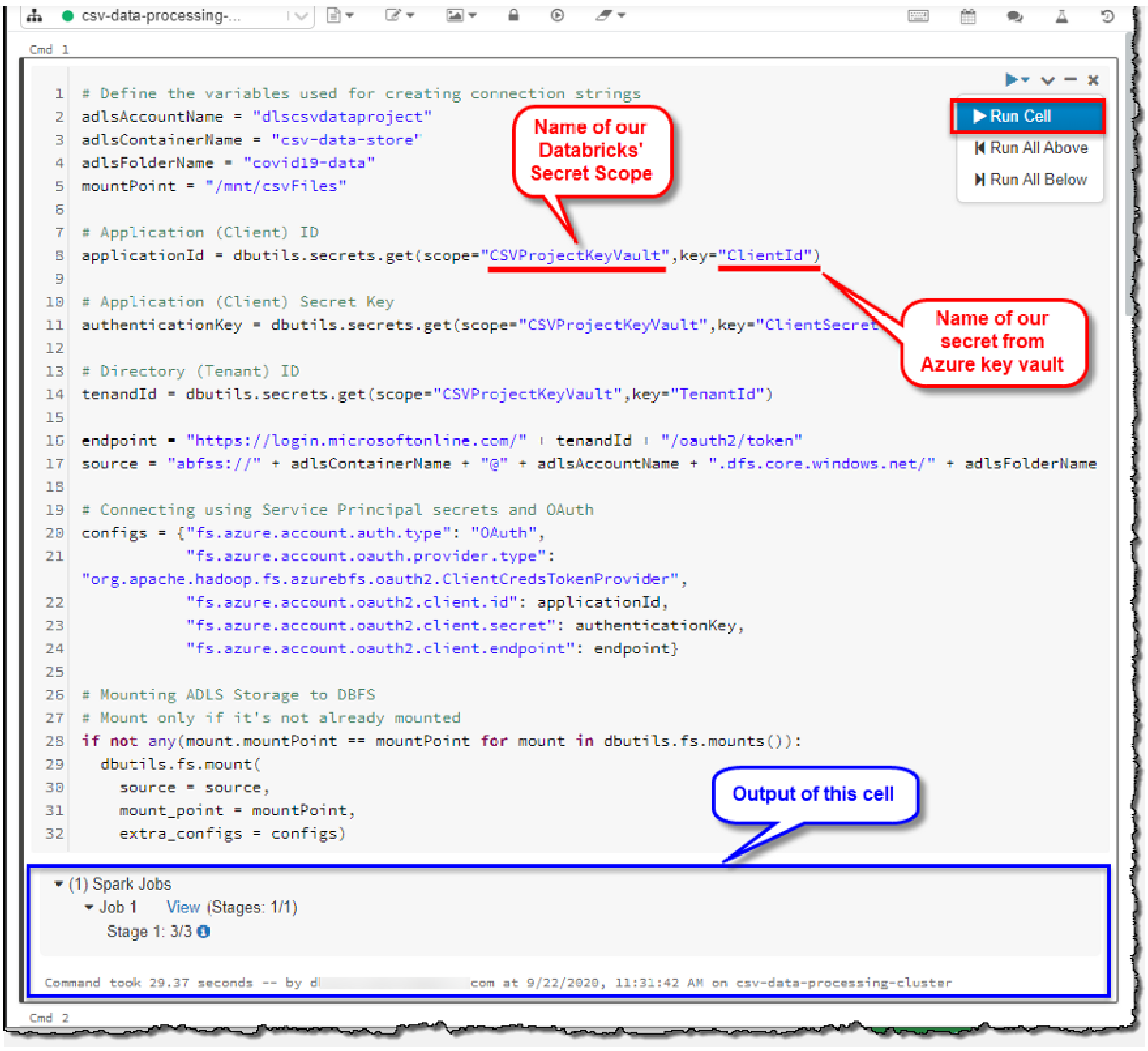
Databricks Utilities
Databricks Utilities (DBUtils) make it easy to perform powerful combinations of tasks. You can use the utilities to…
docs.databricks.com

```
# Python code to mount and access Azure Data Lake Storage Gen2 Account to Azure Databricks with Service Principal and OAuth
# Author: Dhyanendra Singh Rathore
```

Get unlimited access Open in app

```python
mountPoint = "/mnt/csvFiles"

# Application (Client) ID
applicationId = dbutils.secrets.get(scope="CSVProjectKeyVault",key="ClientId")

# Application (Client) Secret Key
authenticationKey = dbutils.secrets.get(scope="CSVProjectKeyVault",key="ClientSecret")

# Directory (Tenant) ID
tenandId = dbutils.secrets.get(scope="CSVProjectKeyVault",key="TenantId")

endpoint = "https://login.microsoftonline.com/" + tenandId + "/oauth2/token"
source = "abfss://" + adlsContainerName + "@" + adlsAccountName + ".dfs.core.windows.net/" + adlsFolderName

# Connecting using Service Principal secrets and OAuth
configs = {"fs.azure.account.auth.type": "OAuth",
           "fs.azure.account.oauth.provider.type": "org.apache.hadoop.fs.azurebfs.oauth2.ClientCredsTokenProvider",
           "fs.azure.account.oauth2.client.id": applicationId,
           "fs.azure.account.oauth2.client.secret": authenticationKey,
           "fs.azure.account.oauth2.client.endpoint": endpoint}

# Mounting ADLS Storage to DBFS
# Mount only if the directory is not already mounted
if not any(mount.mountPoint == mountPoint for mount in dbutils.fs.mounts()):
  dbutils.fs.mount(
    source = source,
    mount_point = mountPoint,
    extra_configs = configs)
```

azure-databricks-mount-adlsGen2.py hosted with ❤ by GitHub view raw



```
# Define the variables used for creating connection strings
adlsAccountName = "dlscsvdataproject"
adlsContainerName = "csv-data-store"
adlsFolderName = "covid19-data"
mountPoint = "/mnt/csvFiles"

# Application (Client) ID
applicationId = dbutils.secrets.get(scope="CSVProjectKeyVault",key="ClientId")

# Application (Client) Secret Key
authenticationKey = dbutils.secrets.get(scope="CSVProjectKeyVault",key="ClientSecret

# Directory (Tenant) ID
tenandId = dbutils.secrets.get(scope="CSVProjectKeyVault",key="TenantId")

endpoint = "https://login.microsoftonline.com/" + tenandId + "/oauth2/token"
source = "abfss://" + adlsContainerName + "@" + adlsAccountName + ".dfs.core.windows.net/" + adlsFolderName

# Connecting using Service Principal secrets and OAuth
configs = {"fs.azure.account.auth.type": "OAuth",
           "fs.azure.account.oauth.provider.type":
"org.apache.hadoop.fs.azurebfs.oauth2.ClientCredsTokenProvider",
           "fs.azure.account.oauth2.client.id": applicationId,
           "fs.azure.account.oauth2.client.secret": authenticationKey,
           "fs.azure.account.oauth2.client.endpoint": endpoint}

# Mounting ADLS Storage to DBFS
# Mount only if it's not already mounted
if not any(mount.mountPoint == mountPoint for mount in dbutils.fs.mounts()):
  dbutils.fs.mount(
    source = source,
    mount_point = mountPoint,
    extra_configs = configs)
```

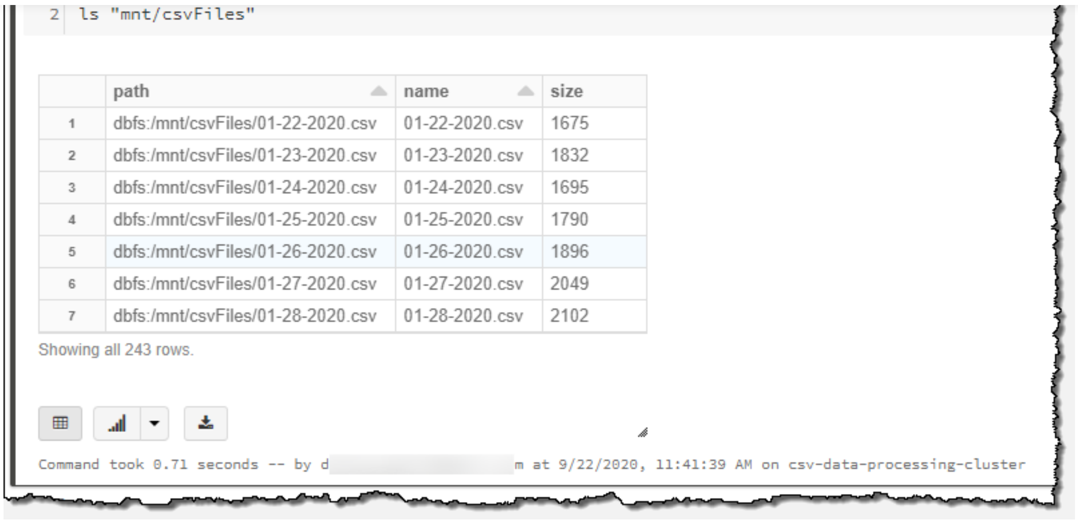
```
▾ (1) Spark Jobs
  ▾ Job 1    View (Stages: 1/1)
      Stage 1: 3/3

Command took 29.37 seconds -- by d[redacted]com at 9/22/2020, 11:31:42 AM on csv-data-processing-cluster
```

Azure Databricks: Mounting ADLS Gen2 in Python (Image by author)

Further reading on how to use notebooks efficiently:

Use notebooks
A notebook is a collection of runnable cells (commands). When you use a notebook, you are primarily developing and…
docs.databricks.com

We can override the default language of a notebook by specifying the language magic command at the beginning of a cell. The supported magic commands are `%python`, `%r`, `%scala`, and `%sql`. Notebooks also support few additional magic commands like `%fs`, `%sh`, and `%md`. We can use `%fs ls` to list the content of our mounted store.



```
ls "mnt/csvFiles"
```

	path	name	size
1	dbfs:/mnt/csvFiles/01-22-2020.csv	01-22-2020.csv	1675
2	dbfs:/mnt/csvFiles/01-23-2020.csv	01-23-2020.csv	1832
3	dbfs:/mnt/csvFiles/01-24-2020.csv	01-24-2020.csv	1695
4	dbfs:/mnt/csvFiles/01-25-2020.csv	01-25-2020.csv	1790
5	dbfs:/mnt/csvFiles/01-26-2020.csv	01-26-2020.csv	1896
6	dbfs:/mnt/csvFiles/01-27-2020.csv	01-27-2020.csv	2049
7	dbfs:/mnt/csvFiles/01-28-2020.csv	01-28-2020.csv	2102

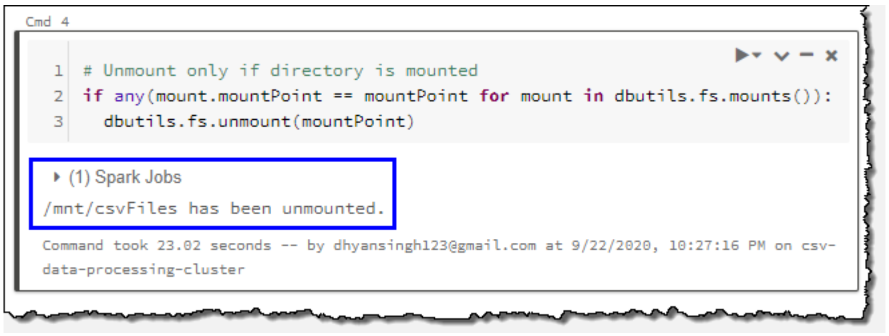
Showing all 243 rows.

```
Command took 0.71 seconds -- by d[redacted]m at 9/22/2020, 11:41:39 AM on csv-data-processing-cluster
```

Azure Databricks: Magic command (Image by author)

Don’t forget to unmount your storage when you no longer need it.

```
# Unmount only if directory is mounted
if any(mount.mountPoint == mountPoint for mount in dbutils.fs.mounts()):
  dbutils.fs.unmount(mountPoint)
```



```
# Unmount only if directory is mounted
if any(mount.mountPoint == mountPoint for mount in dbutils.fs.mounts()):
  dbutils.fs.unmount(mountPoint)
```

```
▸ (1) Spark Jobs
/mnt/csvFiles has been unmounted.
Command took 23.02 seconds -- by dhyansingh123@gmail.com at 9/22/2020, 10:27:16 PM on csv-data-processing-cluster
```

Azure Databricks: Unmounting ADLS Gen2 in Python (Image by author)

Congratulations! You’ve successfully mounted your storage account to Databricks without revealing and storing your application secrets and access keys.

Conclusion

We looked at how to register a new Azure AD application to create a service principal, assigned access roles to a service principal, and stored our secrets to Azure Key Vault. We created an Azure Key Vault-backed Secret Scope in Azure Dataricks and securely mounted and listed the files stored in our ADLS Gen2 account in Databricks.

Next Steps

If you’re curious to know how we got those CSV files in our storage, please head to our other article to set up an innovative Azure Data Factory pipeline to copy files over HTTP from a GitHub repository.

Using Azure Data Factory to incrementally copy files based on URL pattern over HTTP

An innovative Azure Data Factory pipeline to copy multiple files, incrementally, over HTTP from a third-party web…

medium.com

A credential-safe way to connect and access Azure Synapse Analytics in Azure Databricks

A guide on how to setup SQL Server firewall and connect from Databricks using Secret Scopes in PySpark

medium.com

. . .

Dhyanendra Singh Rathore is a Microsoft-certified Data, BI, and power platform professional. He is passionate about solving problems and currently gravitating towards serverless computing and AI platforms. He has a Master's degree in Computer Networking Engineering.

You can **join** him on Medium or connect with him on LinkedIn.

Got any topic-related issues you wish to discuss? Shoot an email to dhyan.singh@everydaybi.com for a private consultation.