Open in app ↗

Search Write

Databricks Azure Data Lake Gen2 Access

Ryan Chynoweth
7 min read · Apr 6, 2022

10

UPDATE!—This article applies to access configuration **without** Databricks Unity Catalog which is now the default data management service. The blog applies to connecting Apache Spark (Databricks or Open Source) clusters to Azure Data Lake Gen2. This is considered legacy on Databricks.

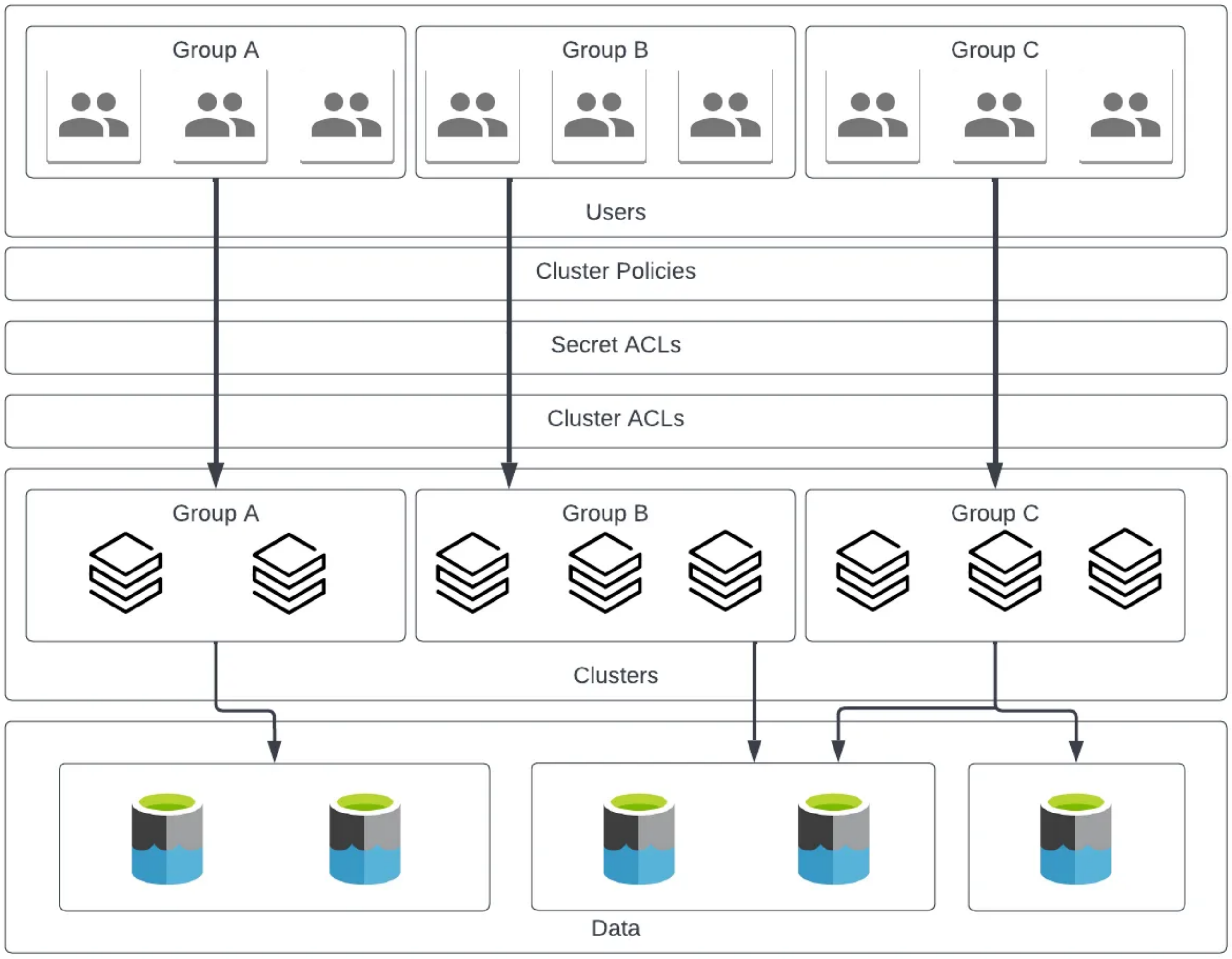
Introduction

One of our main goals at Databricks is to make working with your data as simple as possible, while providing advanced tooling to solve the worlds toughest problems. Most cloud data warehouses will own both data and compute which simplifies the data management process for their customers. It is important to note that ownership is not the same as de-coupled storage and compute, it is determined by which tenant the data resides in. At Databricks, we like our customers to "own their data", which means that you will store your data with your cloud provider in an open format so that you are not locked into a single compute engine. The process we highlight in this article is common for any database that supports external tables, as it typically requires additional configuration and governance that is not required when using a database's internal proprietary storage format. We recommend using Delta Lake, which is an open source storage layer based on Apache Parquet.

Since we believe in giving our customers the ability to own their data, it adds complexity to Databricks that many folks often have issues with. There are many ways to set up authentication between Databricks and you Azure Data Lake Gen2 storage account (ADLS). If you have not stumbled upon this Github repository, I would highly encourage you read it to understand your different options. This article will mostly focus on two of those patterns that should be considered as best practices for managing your data on Databricks: Cluster Configurations (pattern 4) for your Databricks clusters, and Table ACLs (pattern 6) for your SQL endpoints. To be fair Table ACLs are really your only option when it comes to Databricks SQL, since it is purposed for the BI and SQL personas.

Cluster Configurations and Permissions

The best way to control access to data is using cluster permissions. As an administrator I will set up groups and add users to those groups with the intent that the members of each group will have identical access. Once I have groups, I can then use cluster permissions to limit the ability for users to connect/use a cluster. In short, groupA will only be able to access clusterA (or a subset of clusters), then by proxy the clusters control access to data.



Users Access Clusters and Clusters Access Data

Now that I have groups, users added, and clusters created, I can then set the configuration to my cluster. Below is a sample of what it actually looks like:

```
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 {{secrets/<SCOPE NAME>/<SECRET KEY
NAME>}}
fs.azure.account.oauth2.client.secret {{secrets/<SCOPE NAME>/<SECRET
KEY NAME>}}
fs.azure.account.oauth2.client.endpoint
https://login.microsoftonline.com/<YOUR TENANT ID>/oauth2/token
```

Note that if you have multiple storage accounts and you are using different Azure service principals to access each account you may need to do something like the following. Please note that if you are using the **same** service principal for **different** accounts then the previous example will work.

```
fs.azure.account.auth.type.<STORAGE ACCOUNT
NAME>.dfs.core.windows.net OAuth
fs.azure.account.oauth.provider.type.<STORAGE ACCOUNT
NAME>.dfs.core.windows.net
org.apache.hadoop.fs.azurebfs.oauth2.ClientCredsTokenProvider
fs.azure.account.oauth2.client.id.<STORAGE ACCOUNT
NAME>.dfs.core.windows.net {{secrets/<SCOPE NAME>/<SECRET KEY NAME>}}
fs.azure.account.oauth2.client.secret.<STORAGE ACCOUNT
NAME>.dfs.core.windows.net {{secrets/<SCOPE NAME>/<SECRET KEY NAME>}}
fs.azure.account.oauth2.client.endpoint.<STORAGE ACCOUNT
NAME>.dfs.core.windows.net https://login.microsoftonline.com/<YOUR
TENANT ID>/oauth2/token
```

Note that the `{{secrets/<SCOPE NAME>/<SECRET KEY NAME>}}` is accessing secrets stored in a Databricks secret scope so that I don't have to put my sensitive information in plain text! Additionally, notice that the difference in the two config options requires the storage account information within the configuration key itself i.e. `<STORAGE ACCOUNT NAME>.dfs.core.windows.net` .

Cluster permissions need to be activated in the Admin Console of a Databricks Workspace. There are four different permissions for user access to a cluster: No Permissions, Can Attach To, Can Restart (most popular), and Can Manage.

- No Permissions
- Can Attach To: can use the cluster and view logs, but cannot start or modify

- Can Restart: can restart/start clusters and view all logs, but cannot modify
- Can Manage: full permissions on the cluster

To learn more about cluster access control, check out the documentation.

With all this in place you should now be able to access your data via file path with the following (or via the Table APIs):

```
dbutils.fs.ls("abfss://<container_name>@<storage_account_name>.dfs.co
re.windows.net/path/to/table")
```

Which also means you can create all your databases with the following syntax to automatically store your tables to an external storage location.

```
CREATE DATABASE IF NOT EXISTS my_database
LOCATION
"abfss://<container_name>@<storage_account_name>.dfs.core.windows.net
/all_databases/my_database";

USE my_database;

%python
df.write.saveAsTable("my_table")
```

Cluster Policies

As I mentioned in the introduction, we want to make Databricks simple. What I described in the previous section can be a little tedious, right? I have to setup clusters for all my users and groups, which is not complex but it can be time consuming. Furthermore, cluster permissions are for existing clusters and maybe I want to allow users to create clusters but do not want to give them full permissions within the workspace.

One way to reduce the overhead of administration and give users the ability to create specific types of clusters is to use Cluster Policies. Cluster policies are a way to enforce rule sets on clusters (sizes, autoscaling, spark configs etc.) that users create, which allows the admin to give users permissions to create compute without the admin manually intervening.

With cluster policies I can set these configurations once, then apply these policies to my groups so they can only create clusters with those settings. As an example, let's say I have a group that requires five different clusters. Instead of creating each of those five clusters with specific configuration details, I can create a policy and let the users create the clusters themselves. Therefore, in order to make sure they have proper access to the data I would want to add something like the following in a cluster policy that would automatically handle data access. In the example I also include a tag on the policy so that I can either track usage at a use case or a group level which will surface in your Azure bill.

```
{
  "custom_tags.UseCase": {
    "type": "fixed",
    "value": "<<USE CASE TAG>>"
  },
  "spark_conf.fs.azure.account.oauth2.client.id": {
    "type": "fixed",
    "value": "{{secrets/<<SCOPE NAME>>/<<SECRET NAME>>}}"
  },
  "spark_conf.fs.azure.account.oauth.provider.type": {
    "type": "fixed",
    "value":
"org.apache.hadoop.fs.azurebfs.oauth2.ClientCredsTokenProvider"
  },
  "spark_conf.fs.azure.account.oauth2.client.endpoint": {
    "type": "fixed",
    "value": "https://login.microsoftonline.com/<<TENANT
ID>>/oauth2/token"
  },
  "spark_conf.fs.azure.account.auth.type": {
    "type": "fixed",
    "value": "OAuth"
  },
  "spark_conf.fs.azure.account.oauth2.client.secret": {
    "type": "fixed",
    "value": "{{secrets/<<SCOPE NAME>>/<<SECRET NAME>>}}"
  }
}
```

Secret Access Control

Since we are leveraging secrets to store our service principal id and secret, we will need to make sure that users have proper access to those secrets as well.

To create a Secret Scope and add secrets to the scope users need to use the Databricks REST API or the Databricks CLI (recommended). In this example I will use the Databricks CLI.

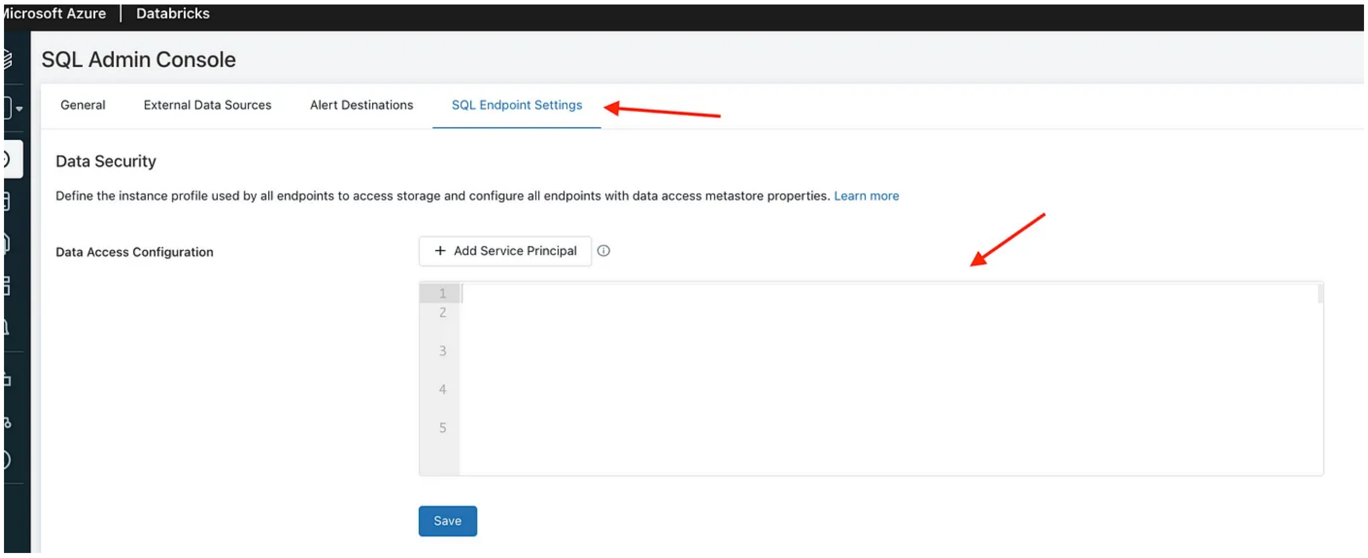
1. Create a secret scope. `databricks secrets create-scope --scope <SCOPE NAME>`
2. Add a secret to the scope. `databricks secrets put --scope <SCOPE NAME> --key <SECRET KEY NAME> --string-value <SECRET VALUE>`
3. Grant a user/group the ability to READ and LIST secrets from the scope. Possible permissions are READ, WRITE, MANAGE. `databricks secrets put-acl --scope <SCOPE NAME> --principal <user i.e. ryan.chynoweth@databricks.com OR group> --permission READ`

It is important to note that Databricks admins will have MANAGE permissions for all secret scopes in the workspace. So non-admin users will not have access to the scope unless explicitly granted. In the scenario above I would have created a scope and added a secret, then gave someone permission to READ the scope. Since we are using service principals to authenticate against ADLS Gen2, we want to ensure that only specific people have access to the credentials. It would be a best practice to use groups to manage these ACLs.

In Azure Databricks, users can also create a secret scoped backed by Azure Key Vault. Key vault backed scopes are a great option for solutions that require shared secrets outside of Databricks, as an example if you share a service principal with Azure Data Factory for job orchestration or if you want to share secrets across Databricks workspaces.

Databricks SQL

Setting up access to data from Databricks SQL is quite easy. Customers should leverage Table ACLs which are how traditional databases and data warehouses handle access to data. Data access is configured in the SQL Admin Console and users a shared service principal for all users.


Microsoft Azure | Databricks
SQL Admin Console
General
External Data Sources
Alert Destinations
SQL Endpoint Settings
Data Security
Define the instance profile used by all endpoints to access storage and configure all endpoints with data access metastore properties. Learn more
Data Access Configuration
+ Add Service Principal
Save

Databricks SQL Admin Console

The configuration here will be the same configuration that we explained at the cluster level.

```
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 {{secrets/<SCOPE NAME>/<SECRET KEY
NAME>}}
fs.azure.account.oauth2.client.secret {{secrets/<SCOPE NAME>/<SECRET
KEY NAME>}}
fs.azure.account.oauth2.client.endpoint
https://login.microsoftonline.com/<YOUR TENANT ID>/oauth2/token
```

Please note that these configs will apply to all SQL Endpoints in the workspace which is why we use Table ACLs to control permissions in DBSQL. Table ACLs control permissions but behind the scenes the endpoints all access the data with the same credentials. I generally recommend to use a single service principal for all your storage locations so that you can use the above configuration and only need to provide a single configuration. However, if multiple service principals are used then use the alternate settings below for each of the external locations. Keep in mind that you would be limitation and I *believe* you can use up to five service principals in DBSQL at this time[1].

```
fs.azure.account.auth.type.<STORAGE ACCOUNT
NAME>.dfs.core.windows.net OAuth
fs.azure.account.oauth.provider.type.<STORAGE ACCOUNT
NAME>.dfs.core.windows.net
org.apache.hadoop.fs.azurebfs.oauth2.ClientCredsTokenProvider
fs.azure.account.oauth2.client.id.<STORAGE ACCOUNT
NAME>.dfs.core.windows.net {{secrets/<SCOPE NAME>/<SECRET KEY NAME>}}
fs.azure.account.oauth2.client.secret.<STORAGE ACCOUNT
NAME>.dfs.core.windows.net {{secrets/<SCOPE NAME>/<SECRET KEY NAME>}}
fs.azure.account.oauth2.client.endpoint.<STORAGE ACCOUNT
```

```
NAME>.dfs.core.windows.net https://login.microsoftonline.com/<YOUR
TENANT ID>/oauth2/token
```

Conclusion

In short, setting up and managing ADLS access within Databricks is pretty straightforward. These type of configurations are common for all data warehouses that have support for **external** tables. Ultimately, configuration is slightly complex but the benefits of data ownership far outweigh the costs of storing your data in someone else's tenant and paying for permission to view your data.

Clusters control the access to the data. Admins control which users have access to those clusters.

[1] Five may not be the exact limit but please know there is a limit to the number of configs you can add to a SQL Endpoint

Disclaimer: these are my own thoughts and opinions and not a reflection of my employer

Databricks Azure Data Lake Databricks Sql

Written by Ryan Chynoweth

Edit profile

312 Followers

Senior Solutions Architect Databricks — anything shared is my own thoughts and opinions

More from Ryan Chynoweth

DELTA SHARING
en standard for secure data sh
dustry's first open protocol for secure data sharing, mak
ıer organizations regardless of which computing platfor

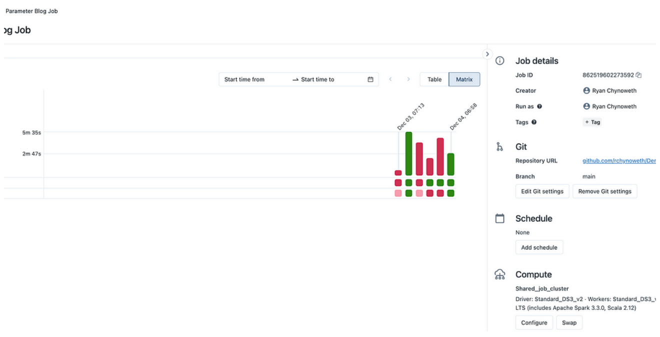
Ryan Chynoweth

Delta Sharing: An Implementation Guide for Multi-Cloud Architecture

Introduction

8 min read · 3 days ago

3 2



Job details
Job ID 862519602273592
Creator Ryan Chynoweth
Run as Ryan Chynoweth
Tags + Tag

Git
Repository URL github.com/rchynoweth/Der
Branch main

Schedule
None

Compute
Shared_job_cluster
Driver: Standard_DS3_v2 · Workers: Standard_DS3_
LTS (includes Apache Spark 3.3.0, Scala 2.12)

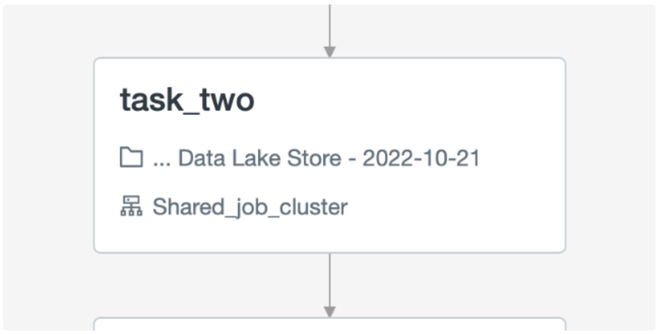
Ryan Chynoweth

Task Parameters and Values in Databricks Workflows

Databricks provides a set of powerful and dynamic orchestration capabilities that are...

11 min read · Dec 7, 2022

50 3



Ryan Chynoweth

Converting Stored Procedures to Databricks

Special thanks to co-author Kyle Hale, Sr. Specialist Solutions Architect at Databricks.

14 min read · Dec 29, 2022

116 5

Ryan Chynoweth

Recursive CTE on Databricks

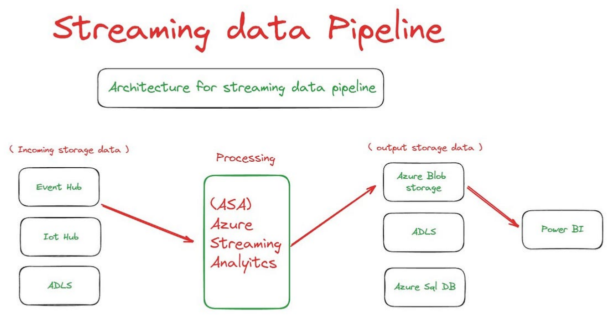
Introduction

3 min read · Apr 20, 2022

33

See all from Ryan Chynoweth

Recommended from Medium



Daan Rademaker

Do-it-yourself, building your own Databricks Docker Container

In my previous LinkedIn article, I aimed to persuade you of the numerous advantages o...

7 min read · Oct 16, 2023

26

Deepak Pandey

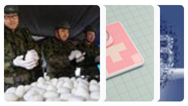
Azure Streaming Data Pipeline

Hello, My name is Deepak Pandey. I work as a data engineer, and here we are going to...

6 min read · Dec 23, 2023

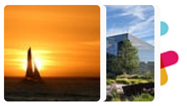
12

Lists



Staff Picks

547 stories · 597 saves



Stories to Help You Level-Up at Work

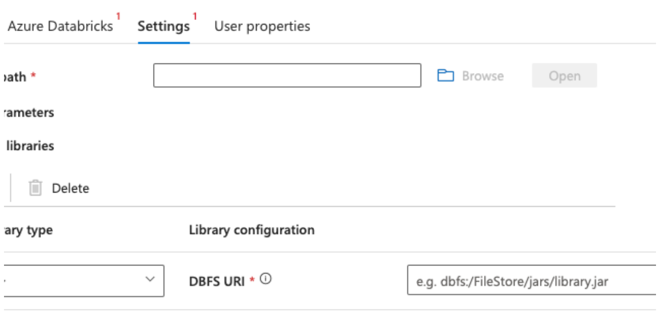
19 stories · 395 saves

Self-Improvement 101

20 stories · 1146 saves

Productivity 101

20 stories · 1047 saves

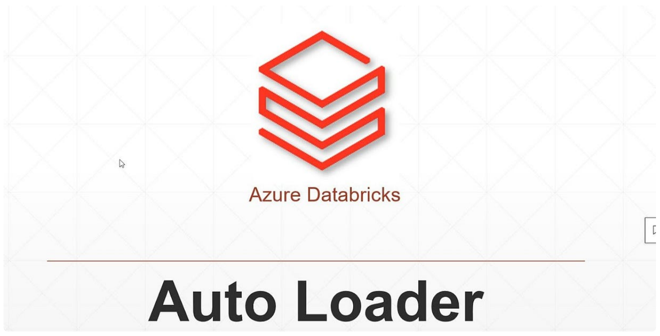


Matt Bradley

Azure Data Factory tips for running Databricks jobs

Following on from an earlier blog around using spot instances with Azure Data...

4 min read · Nov 2, 2023



SIRIGIRI HARI KRISHNA in Towards Dev

Auto Loader

Autoloader simplifies reading various data file types from popular cloud locations like...

5 min read · Dec 9, 2023

14

4 1

Manasreddy

How to pass: Databricks Data Engineer Professional Certification

Conquering the Databricks Data Engineer Professional Exam: A Definitive Guide

3 min read · Aug 24, 2023

8

Ramit Das

Azure Databricks -Deciding on Cluster Sizes

How to choose Cluster Sizes in Azure Databricks? Cluster Sizing is an important...

6 min read · Oct 10, 2023

4

See more recommendations