Delta Sharing: An Implementation Guide for Multi-Cloud Data Sharing

Ryan Chynoweth

8 min read · 3 days ago

An open standard for secure data sharing

Delta Sharing is the industry's first open protocol for secure data sharing, making it simple to share data with other organizations regardless of which computing platforms they use.

Source: https://delta.io/sharing

Introduction

In a recent blog, Delta Sharing: Multi-Cloud Data Sharing for Architects I discuss the importance of sharing data across various environments and how to handle different scenarios at a high level. The primary focus from a technology perspective was Delta Sharing, an open-source data sharing protocol that offers unparalleled flexibility and freedom. We discussed alternative solutions depending on the technical requirements. In this blog

we will give actual examples of the various solutions provided allowing readers to gain a functional understanding of multi-cloud data sharing from an implementation standpoint.

We will cover the following examples:

- Databricks to Databricks Delta Sharing
- Databricks to Non-Databricks Delta Sharing
- Databricks to Power BI Delta Sharing
- Writing data from AWS Databricks to Azure Data Lake Gen2
  - This is a similar process between any of the three cloud storage solutions (S3, ADLS, and GCS).
- Cloning Tables from AWS Databricks to Azure Data Lake Gen2
  - This is a similar process between any of the three cloud storage solutions (S3, ADLS, and GCS).

Delta Sharing

Delta Sharing is a transformative solution to access and share data. Its support for batch, incremental, and stream processing deliver unparalleled versatility in data handling. A key feature of Delta Sharing lies in its ability to abstract data location enabling seamless access to data across different environments.

In Databricks, Delta Sharing seamlessly integrates with the Unity Catalog to adhere to enterprise governance protocols, offering centralized management, and auditing capabilities for shared data. An additional advantage of Delta Sharing is its commitment to providing users with access to the most up to date data, whether consumed through streams or batches.

This real-time accessibility empowers informed decision-making and improves operational efficiency.

There are three core components to Delta Sharing:

1. **Provider**: the individual or organization that owns the share and control access to the data objects. A provider can have many shares and many recipients.
2. **Recipient**: the individual or organization that is gaining access to a share.
3. **Share**: a read-only collection of data objects (tables, table partitions, views, etc.)

Before we dive into the examples, the following instructions are how to create a Delta Share in Databricks using SQL:

```
CREATE SHARE IF NOT EXISTS my_share COMMENT "This is a Delta share!";
```

Once the Delta Share is created then you can add a data object to the share. The following command adds a single table to the share we previously created.

```
ALTER SHARE my_share ADD TABLE my_catalog.my_schema.my_table
COMMENT "This table belongs to a share!" ;
```

We now have a share on Databricks with a single table. Please note that you can share tables, views, table history, and table partitions.

Databricks to Databricks Delta Sharing

Delta Sharing is open source, which means that it does not require the provider or the recipient to be a Databricks customer, however, sharing data on Databricks is extremely simple. To share data between Databricks Workspaces, users can simply do the following.

Obtain the recipient's metastore identifier by running the following command in the recipient's Databricks environment:

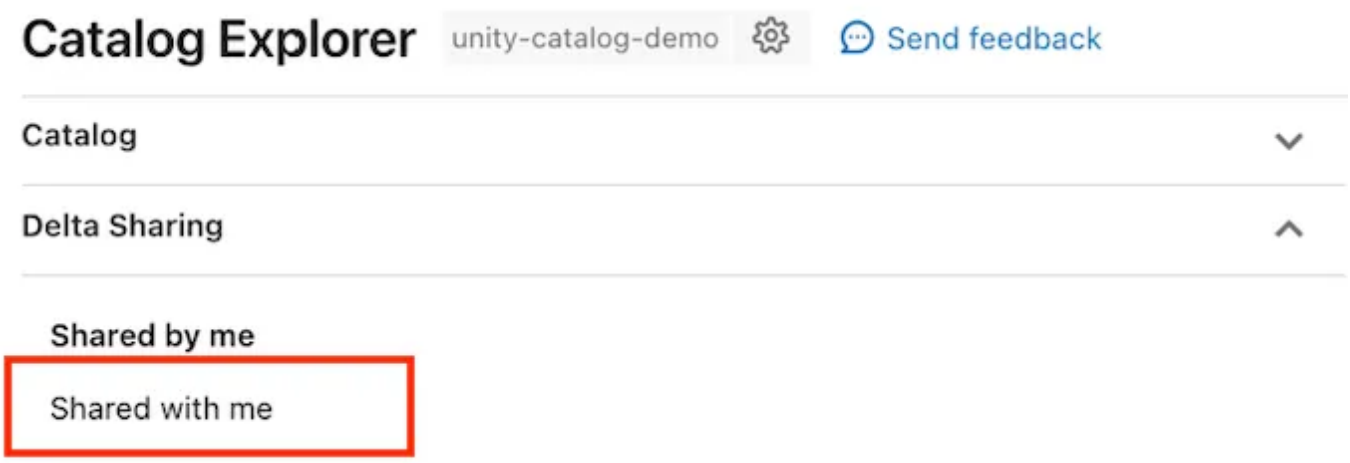
```
SELECT CURRENT_METASTORE();
```

Using the output of the previous command, one can add the recipient by running the following in the provider's environment:

```
CREATE RECIPIENT IF NOT EXISTS <recipient-name>
USING ID '<sharing-identifier>'
COMMENT "This is a new recipient!" ;

GRANT SELECT ON SHARE my_share TO RECIPIENT <recipient-name>;
```

At this point the recipient should be able to discover the share in their environment in the "Shared with me" section of the Catalog Explorer.



Recipient's Databricks User Interface Location of the Delta share

The biggest benefit of Delta Sharing between two Databricks customers is that it simply works with practically *zero* effort. The data will automatically show up in the recipient's catalog and they can begin querying data.

Databricks to Non-Databricks Delta Sharing

To share data with a non-Databricks customer, the provider will need to first add the recipient then send the connection information to the recipient.

Simply run the following command to obtain the Delta Share activation link in which you can send directly to your recipient to provide access.

```
DESCRIBE RECIPIENT <recipient-name>;
```

Once the recipient has successfully activated the delta sharing information they can use the following code to read data from the provider's Delta Share using OSS projects.

```
from delta import *
import delta_sharing
from pyspark.sql import SparkSession

profile_file = "/path/to/oss_share.share"
client = delta_sharing.SharingClient(profile=profile_file)
share_name = "share_name"
schema_name = "schema_name"
table_name = "table_name"

# Initialize Spark
spark = (SparkSession
.builder
.config('spark.jars.packages', 'org.apache.hadoop:hadoop-azure:3.3.1,io.delta:de
.config('spark.sql.extensions', 'io.delta.sql.DeltaSparkSessionExtension')
.config('spark.sql.catalog.spark_catalog', 'org.apache.spark.sql.delta.catalog.D
.getOrCreate()
)

# Read from share
table_url = f"{profile_file}#{share_name}.{schema_name}.{table_name}"

share_df = delta_sharing.load_as_spark(url=table_url)

share_df.show()

(spark.read
.format("deltaSharing")
.load(table_url)
.show()
)
```

The benefit of this approach is that the consumer does not have to be a Databricks customer. This is key, because Delta Sharing avoids a specific vendor requirement for both parties it opens up providers to many more recipients whether they are internal solutions or external organizations. While this example shows a batch solution, it is possible to perform incremental and stream processing off of a Delta Share.

Databricks to Power BI Tool Delta Sharing

Popular business intelligence tools, like Power BI, have Delta Sharing connectors which allow them to connect directly to the data without requiring data warehouse compute. This is an extreme reduction in cost for BI solutions that leverage this capability. To connect Power BI to a Delta Sharing, click on “Get Data” then search for “Delta sharing” which will open the following screen and add the required URL.

Power BI User Interface for Delta Sharing

It is important to note that Delta Sharing has less capabilities to pushdown queries when compared to connecting to a Databricks SQL warehouse because Power BI is connecting directly to the data as opposed to a warehouse. Therefore this solution is not necessarily as scalable because the Power BI server is responsible for the majority of the computation required to transform the data, however, it entirely eliminates the data warehouse cost and allows users to have more economical reporting capabilities. Essentially, when you use Delta Sharing as a data source the data modeling and transformations are almost entirely owned by the reporting tool. It may require you to import the entire dataset prior to filtering, aggregating, or applying transformations which could be a heavy load on the client server. I recommend using Delta sharing as a data source for small to medium sized

tables only. Engineers can also materialize table aggregations prior to ingesting into Power BI to reduce client server load.

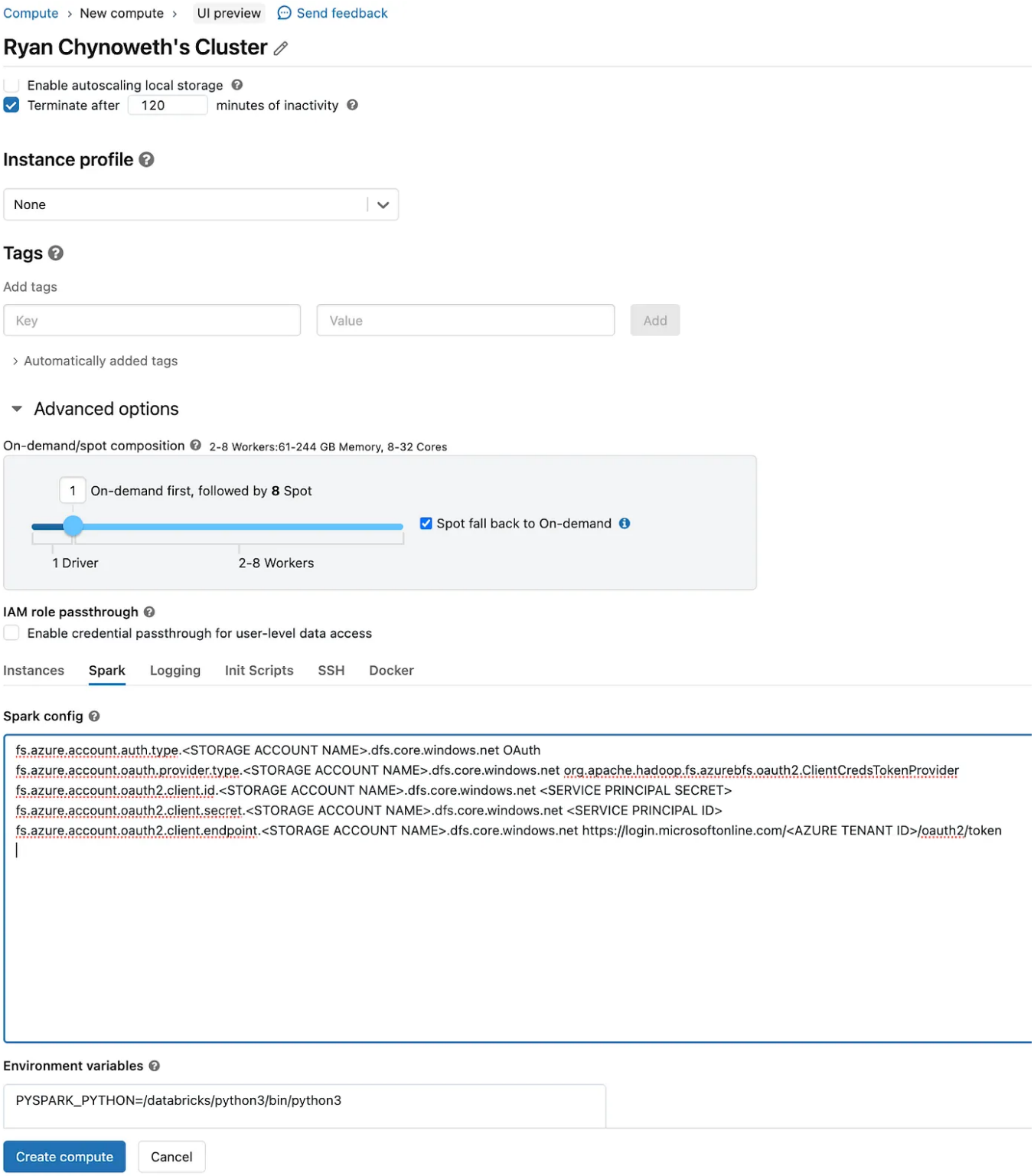
Pushing data from AWS Databricks to Azure Data Lake Gen2

To connect from AWS Databricks to Azure Data Lake Gen2 (ADLS Gen2) you can follow my previous blog in which I discuss how to set cluster settings in order to authenticate with ADLS Gen2. In short, you will need an Azure Service Principal with Storage Blob Data Contributor permissions on the container, and the Azure Tenant ID. Please note that these settings are similar for all three cloud storage solutions.

On the AWS Databricks cluster you will need to set the following settings:

```
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 <SERVICE PRINCIPAL SECRET>
fs.azure.account.oauth2.client.secret.<STORAGE ACCOUNT
NAME>.dfs.core.windows.net <SERVICE PRINCIPAL ID>
fs.azure.account.oauth2.client.endpoint.<STORAGE ACCOUNT
NAME>.dfs.core.windows.net https://login.microsoftonline.com/<AZURE TENANT
ID>/oauth2/token
```

Please see below for a screenshot of the cluster configurations on an AWS Databricks Cluster.



Databricks Cluster Configuration User Interface

Now that you have set your cluster configuration you can run the following command, that reads data from an AWS Databricks environment, and streams that data to an ADLS Gen2 storage location.

```
(spark.readStream
.table("my_catalog.my_schema.my_table")
.writeStream
.format("delta")
.option("checkpointLocation", "/path/to/checkpoint")
.start("abfss://<container_name>@<storage_account_name>.dfs.core.windows.net/pat
)
```

One of the benefits of this process is that the data provider controls the cadence of data egress from one cloud environment to another, this is in contrast with Delta Sharing where the data recipient controls the egress of data. Similar to Delta Sharing, users can transform data in-flight, stream, batch, and incremental process data as needed.

Cloning Tables from AWS Databricks to Azure Data Lake Gen2

The previous section covered the required cluster configurations to authenticate AWS Databricks clusters to ADLS Gen2, which is also a requirement for table cloning. The key difference here is that the data replication is much simpler than using the native Spark read/write functions.

To clone a table from one Databricks environment to an ADLS Gen2 storage location please run the following command:

```
CREATE TABLE IF NOT EXISTS delta.`abfss://<container_name>@<storage_name>.dfs.co
CLONE my_catalog.my_schema.my_table
```

Similar to the previous section, the benefit of this process is that the data provider controls the cadence and egress of data from one cloud

environment to another, this is in contrast with Delta Sharing where the data recipient controls. However, with table clones users are unable to perform in-flight transformations and cannot stream the data. The data is able to be batch and incrementally processed as needed. I would recommend using table clones for data replication scenarios for its **simplicity** and **performance**.

Management and Automation of Delta Sharing Resources

In this section, we will discuss decisions around the automation and management of Delta Sharing resources on Databricks. Please note that this applies to scenarios where Delta Sharing is being used, not necessarily the other options in which data is written and/or cloned to a secondary location.

There are two primary options when architecting a data share for multiple consumers: single share with many recipients or many shares with isolated recipients.

- A **single share** allows providers to write a single job that processes, transforms, and partitions the data to then share that data. This allows a provider to then add many recipients to the share and isolate tables based on the partition. However, while you can isolate partitions in a table you cannot restrict access to data objects as a whole, therefore, more granular governance of the data assets is owned by the recipient.
- Having **many shares** allows providers to have strong isolation between recipient data and own the granular permissions of the shared data assets. For example, you can share a table to recipient A and not recipient B, however, with multiple shares the provider will likely require multiple jobs to process, transform, and share that data to the end consumer making it more costly.

Both scenarios work extremely well at scale, but have a few nuances that should be considered during the planning phases of the project around cost and access controls.

As for automating the management of the Delta Sharing resources, please review the APIs available where you can programatically manage your resources. For example, let's assume there is a solution where consumers can request a "subscription" to a data share. Engineers can use the following to programmatically control permissions and objects:

- Create a share.
- Update a share to add/remove data objects.
- Update permissions a share to add/remove recipients.
- Delete a share etirely.
  - Please note that deleting a share does not delete the data objects as those are persisted within the provider's Databricks environment. It simply removes the share entirely as an object.

Conclusion

In this blog, we covered the various implementation details that we discussed in my previous blog. For more information, please check out the Databricks Documentation and the Open Source Documentation. One thing to note is that I did not discuss open-to-open delta sharing, for a code example please see my GitHub repository where it requires setting up your own Delta Sharing server.

Go forth and share data!

Databricks Delta Lake Delta Sharing Multi Cloud

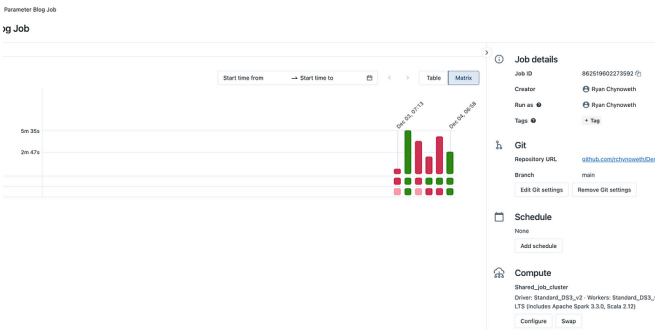
Written by Ryan Chynoweth

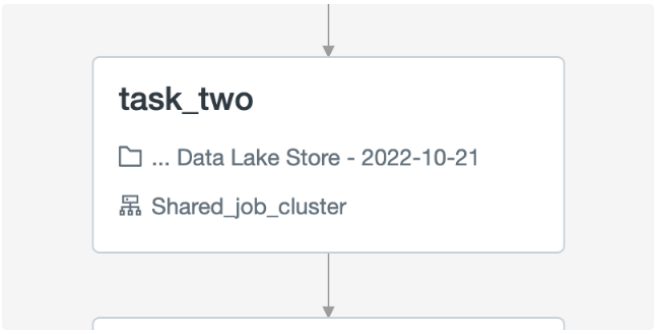
Edit profile

312 Followers

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

More from Ryan Chynoweth





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

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

50 3 116 5

Ryan Chynoweth

Recursive CTE on Databricks

Introduction

3 min read · Apr 20, 2022

33

Ryan Chynoweth

SQL Variables in Databricks

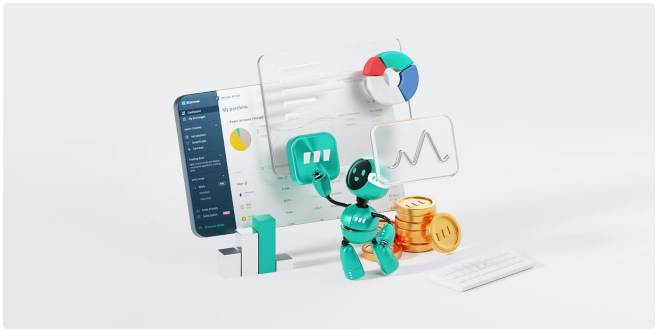
Last December we published a blog providing an overview of Converting Stored Procedure...

2 min read · Oct 6, 2023

21

See all from Ryan Chynoweth

Recommended from Medium



Arpine K in Open Data Discovery

Data Quality Dashboard

Informed decision making relies on data: a crucial asset for business, however, not all...

4 min read · Dec 21, 2023

9

Daan Rademaker

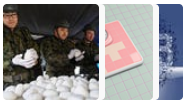
Do-it-yourself, building your own Databricks Docker Container

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

7 min read · Oct 16, 2023

26

Lists



Staff Picks

547 stories · 597 saves

Stories to Help You Level-Up at Work

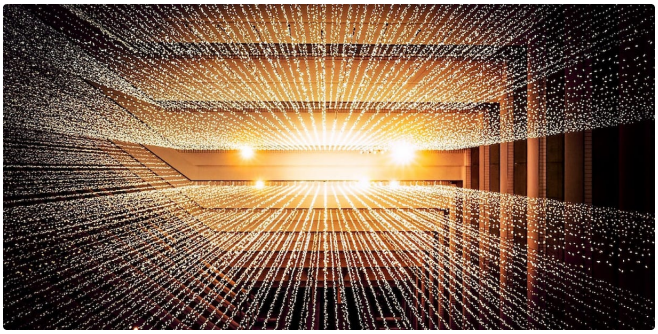
19 stories · 395 saves

Self-Improvement 101

20 stories · 1146 saves

Productivity 101

20 stories · 1047 saves



Cody Austin Davis in DBSQL SME Engineering

Why I am bullish on the Data Intelligence Platform next year

Photo by Joshua Sortino on Unsplash

6 min read · Dec 27, 2023

15

Sheila Teo in Towards Data Science

How I Won Singapore's GPT-4 Prompt Engineering Competition

A deep dive into the strategies I learned for harnessing the power of Large Language...

· 24 min read · 5 days ago

2.6K 38

Shane Hender in Zendesk Engineering

Moving from DynamoDB to tiered storage with MySQL+S3

Originally we implemented a feature to persist an event-stream into DynamoDB to allow...

8 min read · Nov 8, 2023

187 7

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

See more recommendations