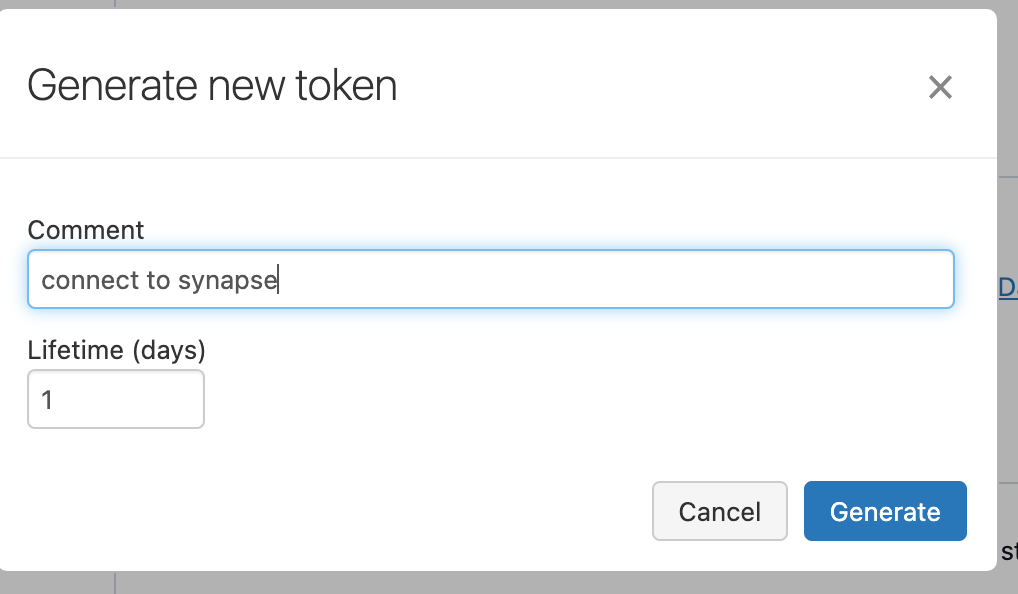
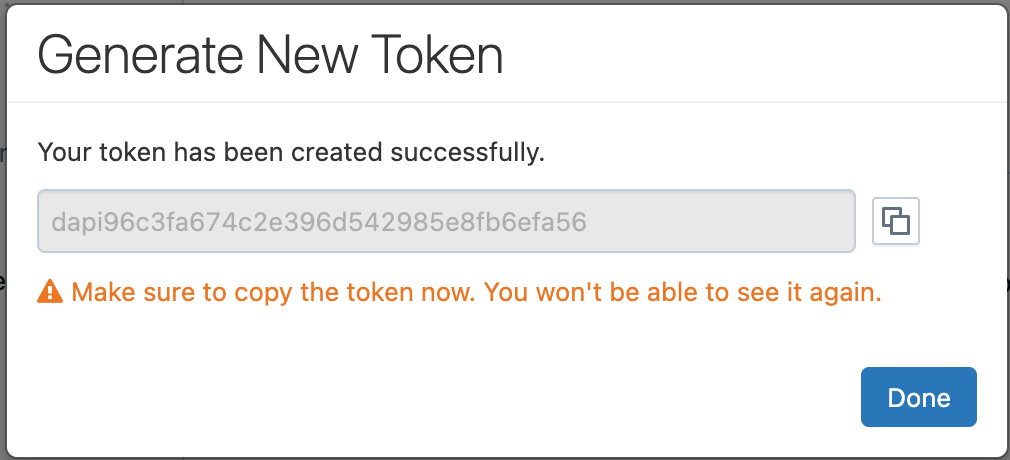
**Connect databricks notebook using synapse workspace**

Databricks 🡪setting 🡪 user🡪 developer 🡪 AccesskeyA screenshot of a computer

Description automatically generated A screenshot of a computer

Description automatically generated  Copy the token A screenshot of a computer

Description automatically generated Create a secrete in keyvault for this accesskeyA screenshot of a computer

Description automatically generated A screenshot of a computer

Description automatically generated A screenshot of a computer

Description automatically generatedNow we have to create linked services for both key vault and data bricks.

Go to synapse 🡪 Manage 🡪 linked service 🡪 new

A screenshot of a computer

Description automatically generated A screenshot of a computer

Description automatically generated A screenshot of a computer

Description automatically generated A screenshot of a computer

Description automatically generated A screenshot of a computer

Description automatically generatednow create a link service for databricks A screenshot of a computer

Description automatically generated A screenshot of a computer

Description automatically generated A screenshot of a computer

Description automatically generated Synapse -> integrate 🡪 new pipeline 🡪drag and drop notebook  
In azure databricks give the linked service which we created previouslyA screenshot of a computer

Description automatically generatedand in setting give the name of notebook u wanted to createA screenshot of a computer

Description automatically generated A screenshot of a computer

Description automatically generatedPipeline ran successfully

## =============================

## **Medallion Architecture & Best Way to Organize Data**

**What is Medallion Architecture?**

Medallion Architecture is a **layered approach to organizing data** in a Lakehouse or data lake using progressive levels of data refinement. The core idea is to improve **data quality, reliability, and accessibility** as it moves through layers.

**Bronze Layer – *Raw Ingestion***

* **Purpose:** Store raw, unfiltered data as-is.
* **Source:** Ingested from multiple systems like APIs, databases, logs, IoT devices, etc.
* **Format:** Often in Parquet, JSON, or CSV.
* **Use Case:** Auditing, debugging, full data lineage.

*Tip: Partition by ingestion date to optimize reads and ensure traceability.*

**Silver Layer – *Cleaned & Enriched***

* **Purpose:** Cleanse, filter, join, and validate data.
* **Examples:**
  + Null value removal
  + Applying schema
  + Joining multiple sources

*Tip: Enforce schema evolution rules and build modular transformation pipelines.*

**Gold Layer – *Curated for Business Use***

* **Purpose:** Serve high-quality, analytics-ready data.
* **Examples:**
  + Aggregated sales data
  + Metrics dashboards
  + Machine learning features

*Tip: Align structure with business domains, use business logic for final transformation.*

**Best Practices for Organizing Data:**

* Use **Delta Lake** for ACID transactions, time travel, and schema enforcement.
* Implement **data quality checks** and alerting (e.g., Great Expectations or Delta constraints).
* Automate data movement using **Databricks workflows** or **Azure Data Factory**.
* Partition smartly — by **date**, **region**, or **customer segment** depending on access pattern.
* Maintain **metadata and lineage** using Unity Catalog or Purview.

## **RDD vs Data Frames in Spark**

| **Feature** | **RDD (Resilient Distributed Dataset)** | **Data Frame** |
| --- | --- | --- |
| Abstraction | Low-level | High-level |
| API | Object-oriented (Java, Scala) | SQL-like (Data Frame API) |
| Performance | Slower due to no optimization | Faster due to Catalyst optimizer |
| Serialization | Manual control | Optimized under the hood |
| Use Case | Fine-grained transformations | ETL, aggregations, analytics |
| Schema | No schema (untyped) | Schema-based (typed) |

## **Narrow vs Wide Transformations**

### Narrow Transformations

* **Each input partition contributes to a single output partition.**
* **No shuffling** across the network.
* **Examples:** map(), filter(), union()
* **Performance:** Fast and efficient.

 Used when data doesn’t need to move between partitions.

### Wide Transformations

* **Input partitions contribute to multiple output partitions.**
* **Requires shuffling** — data is moved across nodes.
* **Examples:** groupByKey(), reduceByKey(), join()
* **Performance:** Slower due to network IO and disk spills.

Occurs when transformation needs data from multiple partitions (e.g., aggregations or joins).

### Performance Tip:

* Prefer reduceByKey() over groupByKey() (less shuffling).
* Use broadcast joins for smaller datasets to avoid full shuffles.
* Monitor with **Spark UI** to identify expensive shuffles and optimize partitions.