# Microsoft Fabric

**What is Microsoft Fabric?**
Microsoft Fabric is Microsoft's unified, enterprise-grade software-as-a-service (SaaS) analytics platform. It brings together the full data lifecycle—data ingestion, processing, transformation, storage, real-time analytics, and reporting—into a single, seamless environment.

**Core Capabilities**

- OneLake – A unified data lake that serves as the central storage layer across all services
- Data Workloads – Fabric offers role-specific experiences:
    - Data Engineering (Spark),
    - Data Factory (pipelines),
    - Data Warehouse (T-SQL queries, caching),
    - Data Science (notebooks, ML),
    - Real-Time Intelligence (streaming analytics),
    - Power BI (business intelligence & visualization)
- AI Integration – Native support for Copilot and Azure AI Foundry helps automate data transformation, generate insights, and simplify analysis
- Governance & Collaboration – Built-in data discovery, governance, sensitivity labeling, and seamless collaboration across teams

---

## 🌐 Why It Matters

Fabric eliminates complexities by integrating traditionally separate platforms (like Azure Data Factory, Synapse Analytics, and Power BI) into one coherent system. It helps organizations:

- Reduce tool fragmentation and integration overhead
- Accelerate insights with AI-powered data processing
- Simplify security, compliance, and governance with centralized controls

Whenever you create tenant or capacity in Fabric you get one lake.

## OneLake

https://learn.microsoft.com/en-us/fabric/onelake/onelake-overview

Microsoft Fabric’s **OneLake** is the unified data lake storage that comes with every Fabric tenant—it’s like the “OneDrive for data

**OneLake** is Microsoft's unified, organization-wide data lake solution within the Microsoft Fabric ecosystem. Often referred to as the "OneDrive for data," OneLake serves as a centralized repository for all analytics data, facilitating seamless data management and analysis across various workloads.

---

### 🔍 What Is OneLake?

OneLake is a **logical data lake** that provides a single, unified storage layer for an organization's data. It is automatically included with every Microsoft Fabric tenant and is designed to store data from various sources in a centralized manner. Built on top of **Azure Data Lake Storage Gen2 (ADLS Gen2)**, OneLake supports both structured and unstructured data formats, including Delta Lake and Parquet files.

---

### ▯ Key Features

- **Unified Storage**: Acts as a single repository for all analytics data, eliminating data silos and reducing duplication.
- **Integrated with Fabric Workloads**: Seamlessly works with various Microsoft Fabric services, such as Data Engineering, Data Factory, Synapse, and Power BI.
- **Data Shortcuts**: Allows virtual linking to external data sources, enabling access to data across different domains and clouds without physical data movement.
- **Granular Security Controls**: Supports role-based access controls, including table, column, and row-level security, ensuring data is accessible only to authorized users.
- **OneLake Catalog**: Provides a centralized hub for data discovery, management, and governance, enhancing data exploration and compliance.

---

### 🏛 Architectural Role

In the Microsoft Fabric architecture, OneLake serves as the foundational storage layer, enabling various compute engines to operate on the same data without the need for data duplication. This design supports a **lakehouse architecture**, combining the benefits of data lakes and data warehouses, and facilitates scalable data management and analysis.

---

## 🎯 Benefits of Using OneLake

- **Simplified Data Management**: Centralizes data storage, making it easier to manage and govern data across the organization.
- **Enhanced Collaboration**: By providing a single source of truth, teams can collaborate more effectively, reducing inconsistencies and errors.
- **Scalability**: Built on Azure's scalable infrastructure, OneLake can handle large volumes of data, accommodating organizational growth.
- **Improved Data Governance**: With integrated security and compliance features, organizations can ensure that data access and usage adhere to policies and regulations.

## Why Introduce OneLake?

- **Break Down Silos**
  Previously, data silos existed: different teams kept their own lakes or warehouses, often duplicating data. OneLake ensures a **single, centralized repository**, managed inside one Fabric tenant and workspace hierarchy
- **Decouple Storage from Compute**
  With OneLake, **data lives independently** of computing engines. You ingest data once and multiple analytics tools can access it without needing ETL or data copies
- **Simplified Governance & Management**
  All organizational data in OneLake benefits from uniform security policies, access controls, auditing, compliance, and governance—across Delta tables *and* files
- **Flexibility via Shortcuts**
  You can reference external data (in ADLS or S3) without moving it—**zero-copy data access** across platforms

- **Support for Diverse Workloads**
  OneLake enables consistent data use across Spark (for lakehouse) and T-SQL (for warehouse) engines, plus Power BI's Direct Lake mode—on the **same**

## What is Delta Parquet Format?

**Delta Parquet format** is a hybrid data storage format that combines the **columnar efficiency of Apache Parquet** with the **transactional capabilities of Delta Lake**. It is primarily used in big data environments like **Apache Spark** or **Databricks** to manage large-scale data with **ACID transaction support**, **schema enforcement**, and **time travel**.

In short:

**Delta format = Parquet + transaction log (Delta Lake)**

---

### Underlying Storage: Apache Parquet

**What is Apache Parquet?**

- **Columnar** storage format used in big data processing.
- Supports efficient compression and encoding.
- Great for **read-heavy** workloads and **analytical queries**.

Delta Lake **uses Parquet** files under the hood to store actual data. But unlike plain Parquet, Delta adds **transactional features**.

---

### What Delta Adds to Parquet

Delta Lake enhances the Parquet format with:

#### 1. ✓ ACID Transactions

Delta ensures **atomicity**, **consistency**, **isolation**, and **durability** over distributed data:

- No partial writes.
- Read-consistent snapshots even during concurrent writes.

#### 2. Transaction Log (_delta_log)

Every Delta table contains a **_delta_log/** directory, which holds:

- **JSON log files** (recording metadata, schema, data files added/removed).
- **CHECKPOINT Parquet files** (periodic snapshots of the table state for faster reading).

**Example:**

pgsql

CopyEdit

```
/path/to/delta-table/
  ├── part-0001.parquet
  ├── part-0002.parquet
  ├── _delta_log/
      ├── 00000000000000000000.json
      ├── 00000000000000000001.json
      ├── 00000000000000000010.checkpoint.parquet
```

---

### 3. Schema Enforcement & Evolution

- You can **enforce schema** so that bad writes fail.
- Or allow **schema evolution** (add new columns on the fly).

---

### 4. Time Travel

Delta supports querying old versions of the data:

sql

CopyEdit

```
SELECT * FROM delta.`/path/to/table` VERSION AS OF 3;
```

---

### 5. Data Skipping & Z-Ordering

Delta logs statistics about columns, enabling:

- Faster queries by skipping irrelevant files.
- Optimized layout using **Z-ordering** (e.g., for faster filtering).

---

## 6. ⚙ Streaming Support

Delta tables can be **read and written by streaming jobs** (Structured Streaming in Spark).

---

## 🔍 Delta Format Structure in Detail

### ✅ Data Files

- Stored in **Parquet** format.
- Each file contains a subset of rows.
- Delta doesn't modify files; it **adds/removes** files in transaction logs.

### ✅ _delta_log Files

- JSON files record changes like "addFile", "removeFile", "metadata", etc.
- Parquet checkpoints periodically summarize the state of the table.

---

## 🔁 Delta Lifecycle Example

Let’s say you run this command:

sql

CopyEdit

```
INSERT INTO my_table VALUES (1, 'Alice');
```

Here’s what happens:

1. A **Parquet file** is written: part-0001.parquet
2. A **JSON log** is created: 00000000000000000001.json with:

json

CopyEdit

```
{
  "add": {
    "path": "part-0001.parquet",
    "size": 1024,
    ...
  }
}
```

3. Delta readers use _delta_log/ to understand which Parquet files to read.

---

## ✅ Key Benefits of Delta Parquet Format

| Feature | Apache Parquet | Delta Format (Parquet + Log) |
|---|---|---|
| Columnar Storage | ✅ | ✅ |
| ACID Transactions | ❌ | ✅ |
| Schema Evolution | ❌ | ✅ |
| Time Travel | ❌ | ✅ |
| Data Skipping | ❌ | ✅ |
| Concurrent Reads/Writes | ❌ | ✅ |
| Streaming Support | ❌ | ✅ |

---

## □ Where is Delta Parquet Used?

- **Databricks Lakehouse**
- **Azure Synapse with Delta support**
- **Apache Spark (with Delta Lake library)**
- **Microsoft Fabric (OneLake with Delta support)**

---

## 📝 Summary

The **Delta Parquet format** is **not a new file format**—it's a **storage architecture** where data is stored in **Parquet files**, and a **transaction log (_delta_log)** is used to manage metadata and transactional operations. This makes data **reliable, versioned, and performant** for big data workloads.

The **Delta Lake transaction log** (_delta_log) is the **core component** that tracks **all actions taken on the table over time**.

## What Does the Transaction Log Do?

The transaction log in the **_delta_log/** folder maintains a complete, ordered record of:

1. 📄 **New files added**
2. 🗑 **Files removed (deletions)**
3. 🏗 **Schema changes**
4. 🔢 **Table properties (e.g., createdBy, partitioning)**
5. ✅ **Checkpoints (Parquet summary of state)**

Each change to the Delta table creates a new JSON file named like 00000000000000000001.json.

---

### 📦 Example: Delta Table Update Sequence

Let's say this happens in your Delta table:

**Step 1: Initial Insert**

sql

CopyEdit

INSERT INTO my_table VALUES (1, 'Alice');

Creates:

- Parquet file: part-0001.parquet
- Log file: 00000000000000000000.json

json

CopyEdit

```json
{
  "add": {
    "path": "part-0001.parquet",
    "size": 1024,
    "modificationTime": 1720000000000
  }
}
```

---

**Step 2: Another Insert**

sql

CopyEdit

```sql
INSERT INTO my_table VALUES (2, 'Bob');
```

Creates:

- Parquet file: part-0002.parquet
- Log file: 00000000000000000001.json

json

CopyEdit

```json
{
  "add": {
    "path": "part-0002.parquet",
    "size": 1050
  }
}
```

---

**Step 3: Delete a Row**

sql

CopyEdit

```sql
DELETE FROM my_table WHERE name = 'Alice';
```

This removes the file containing Alice (e.g., part-0001.parquet) and writes a new file with remaining data.

Creates:

- New Parquet file: part-0003.parquet (without Alice)
- Log file: 00000000000000000002.json

json

CopyEdit

```json
{
  "remove": {
    "path": "part-0001.parquet",
    "deletionTimestamp": 1720000005000
  },
  "add": {
    "path": "part-0003.parquet"
  }
}
```

🔁 **Delta does not modify existing files — it adds new ones and logs which ones to ignore (remove).**

---

✅ **Why This Matters**

- **Time Travel:** You can rewind the table to any version by reading logs up to a certain point.
- **Concurrency:** Multiple users can read/write with consistency.

- **Auditing:** Every change is tracked and traceable.
- **Schema history:** You know when and how the schema evolved.

---

📁 **Folder Structure Overview**

pgsql

CopyEdit

```
/my_table/
 ├── part-0001.parquet
 ├── part-0002.parquet
 ├── part-0003.parquet
 └── _delta_log/
      ├── 00000000000000000000.json  ← initial insert
      ├── 00000000000000000001.json  ← 2nd insert
      ├── 00000000000000000002.json  ← delete
     └── 00000000000000000010.checkpoint.parquet  ← faster recovery
```

## Data Serving or Storage in Fabric

In Microsoft Fabric, storage options are centralized around a unified storage system called OneLake, which acts as the foundation for all storage in Fabric. Let's go through the available storage options, their structure, formats, and use cases in detail.

---

### 📦 1. OneLake (One Logical Lake)

🔷 OneLake = OneDrive for Data

🔷 What It Is:

- A single, unified, logical data lake for your entire organization.
- Built on top of Azure Data Lake Storage Gen2 (ADLS Gen2).

- Every Fabric item (Lakehouse, Warehouse, Dataflow Gen2, Notebook, etc.) stores its data in OneLake.

🔷 Key Features:

- No duplication of data across tools.
- Governed, secure, and discoverable.
- Supports shortcuts to virtualize data from other sources.

---

## ▯ 2. Lakehouse Storage

🔷 What It Is:

- A Lakehouse is a data architecture that stores data in open formats (Delta/Parquet) within OneLake.
- Combines features of data lakes (flexibility) and data warehouses (structure, performance).

🔷 Storage Format:

- Data is stored in Delta format (Parquet + transaction log).
- Ideal for big data processing, batch ingestion, and machine learning.

🔷 Storage Path:

javascript

CopyEdit

/OneLake/<WorkspaceName>/Lakehouse/<LakehouseName>/Files/

---

## 🏢 3. Data Warehouse Storage

🔷 What It Is:

- A structured, SQL-based storage option optimized for relational data and reporting.
- Built on top of OneLake but uses a relational engine with T-SQL support.

🔷 Key Features:

- Fast performance for large analytical queries.
- Table storage is abstracted from the user (you don’t see files directly).

---

## 🔁 4. Shortcuts (Data Virtualization)

◈ What It Is:

- A pointer to data in another location (OneLake, ADLS Gen2, Dataverse, Amazon S3).
- Appears as native data in Fabric without copying.

◈ Use Cases:

- Access central data sources across departments.
- Enable data mesh architecture.

Example:

- Shortcut from a Lakehouse in Finance workspace → used in another team’s Lakehouse.

---

## 🔄 5. Dataflows Gen2 Storage

◈ What It Is:

- Dataflows (ETL pipelines) store their output tables in OneLake.
- Data is stored in Delta format, and tables can be queried like any other table.

---

## ▯ 6. Notebook/ML Data Storage

- Notebooks (PySpark, SQL, Pandas) can read/write data directly to OneLake.
- Data can be stored as:
    - Delta/Parquet
    - CSV
    - JSON
- Great for machine learning, custom pipelines, or scripting.

---

## 🗂 7. KQL Database (Event/Log Analytics)

- If you're working with log or telemetry data, Fabric supports Kusto Query Language (KQL) databases.
- It is used for streaming data. Data is coming continuously
- Storage is optimized for append-only, high-ingestion workloads.

---

## 🛠 8. External Storage Connectors (Linked Services)

You can connect to and read/write from:

- Azure SQL Database
- Azure Synapse
- Azure Blob/ADLS
- Amazon S3
- Google BigQuery
- SQL Server, Oracle, SAP, etc.

These external storages are not part of OneLake but can be ingested into or linked via shortcuts.

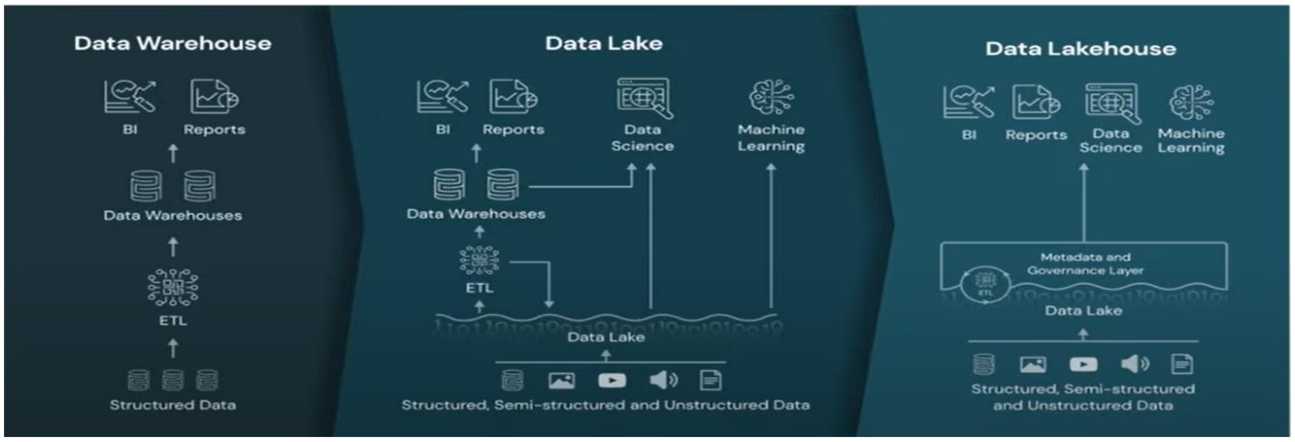
---

□

**Summary: Fabric Storage Options**

| Storage Option | Format | Backed by | Best For | Visible to Users? |
|---|---|---|---|---|
| OneLake | Delta, Parquet | ADLS Gen2 | Central data repository | Yes |
| Lakehouse | Delta (Parquet) | OneLake | Big data, ML, structured/unstructured | Yes (Files tab) |
| Data Warehouse | Relational tables | OneLake | SQL queries, reporting | Abstracted |
| Shortcuts | Virtual access | OneLake/ADLS | Data mesh, reuse | Yes |
| Dataflows Gen2 | Delta | OneLake | ETL outputs | Yes |
| Notebook Storage | Any file format | OneLake | Custom scripting, ML | Yes |
| KQL DBs | Append logs | OneLake | Log & telemetry analytics | Yes (KQL DB UI) |
| External sources | Mixed | External | Ingestion or direct access | Limited (via links) |

---



## Warehouse or lakehouse

When deciding between using a warehouse or a lakehouse, it's important to consider the specific needs and context of your data management and analytics requirements. Equally important, *this is not a one way decision*!

You always have the opportunity to add one or the other at a later point should your business needs change and regardless of where you start, both the warehouse and the lakehouse use the same powerful SQL engine for all T-SQL queries.

Here are some general guidelines to help you make the decision:

- Choose a data warehouse when you need an enterprise-scale solution with open standard format, no knobs performance, and minimal setup.  Best suited for semi-structured and structured data formats, the data warehouse is suitable for both beginner and experienced data professionals, offering simple and intuitive experiences.
- Choose a lakehouse when you need a large repository of highly unstructured data from heterogeneous sources, leveraging low-cost object storage and want to use SPARK as your primary development tool. Acting as a 'lightweight' data warehouse, you always have the option to use the SQL endpoint and T-SQL tools to deliver reporting and data intelligence scenarios in your lakehouse.

## Types of Data Ingestion in Fabric

In Microsoft Fabric, Data Ingestion is the process of bringing data into the Fabric ecosystem, typically into OneLake, where it can be transformed, modeled, and served.

Microsoft Fabric supports multiple flexible ingestion options depending on the data source, frequency (batch or streaming), and use case (analytics, ML, reporting, etc.).

---

### 🏢 Main Ways to Ingest Data into Microsoft Fabric

#### ✅ 1. Dataflows Gen2 (Low-code ETL)

- **Power Query–based visual interface (like Excel formulas).**
- **Ideal for analysts who want to ingest, transform, and load data without writing code.**

◈ **Supports Sources Like:**

- **SQL Server / Azure SQL / Oracle**
- **SharePoint / Excel / CSV**
- **Dataverse**
- **Web APIs**
- **OData, Salesforce, etc.**

◈ **Output: Delta tables stored in OneLake or Lakehouse.**

---

## ✅ 2. Data Pipelines (Code-first ETL)

- **Based on Azure Data Factory-style pipelines.**
- **Use Copy Activity, Data Flow, Notebook execution, etc.**

◈ **Supports Sources Like:**

- **Azure Blob Storage**
- **ADLS Gen2**
- **Amazon S3, Google Cloud**
- **Databases (SQL, Oracle, etc.)**

◈ **Features:**

- **Triggers and scheduling**
- **Parameterization**
- **Linked services**

---

## ✅ 3. Notebooks (Python, Spark, SQL)

- **Best for data engineers and data scientists.**

◈ **Use Cases:**

- **Custom logic ingestion**

- ML feature store preparation
- Ingest via REST APIs, SDKs, or Spark connectors

◆ **Example Code (PySpark):**

python

CopyEdit

```
df = spark.read.format("csv").load("/mnt/rawdata/sales.csv")

df.write.format("delta").save("/lakehouse/gold/sales_delta")
```

---

## ✅ 4. KQL Ingestion (For Log/Event Data)

- Use Kusto Query Language to ingest high-volume telemetry/log data.

◆ **Source Types:**

- Azure Monitor, Application Insights
- CSV/JSON logs
- Event Hubs or IoT streams

---

## ✅ 5. Streaming Data Ingestion

Used for real-time or near-real-time ingestion.

◆ **Tools:**

- Eventstream (Fabric service)
- Event Hubs
- Kafka
- Real-time streaming with Delta Lake
- Streaming Dataflows (coming soon in Fabric)

◆ **Use Case:**

- IoT sensors

- Web logs
- App telemetry

---

## ✅ 6. Shortcuts (Data Virtualization, Not Ingestion)

- This is not traditional ingestion, but it makes external data available without copying. Shortcuts are the objects which just point to a particular location which eliminates data movement.
  In this data will not come to one lake. It supports other data sources also other than azure like Amazon S3, GCP.

🔷 Supports:

- OneLake from other workspaces
- ADLS Gen2
- Dataverse
- Amazon S3 (preview)

🔷 Benefit:

- Save storage
- Enable data mesh architecture

---

## ✅ 7. Manual Uploads

- Upload files like CSV, Excel, JSON, Parquet directly into:
  - Lakehouse (Files tab)
  - OneLake folder

🔷 Great for one-time or exploratory work

---

## ✅ 8. Power BI DirectQuery / Import

- Used mainly for BI reporting, but can also act as ingestion for semantic models.
- Import mode copies data into model memory.

- **DirectQuery queries data in real-time (doesn't copy).**

---

### 📌 Summary Table

| **Ingestion Method** | **Best For** | **Code-Free** | **Output Location** | **Format** |
|---|---|---|---|---|
| Dataflows Gen2 | Analysts, ETL from known sources | ✅ | Lakehouse / OneLake | Delta |
| Data Pipelines | Complex, scalable ingestion | ❌ / ✅ | Lakehouse / Warehouse | Delta |
| Notebooks | Custom logic, ML ingestion | ❌ | Any OneLake path | Delta/Parquet/CSV |
| KQL Ingestion | Log/event/telemetry data | ❌ | KQL Database | Kusto engine |
| EventStream (Streaming) | Real-time apps, sensors | ❌ / ✅ | Delta tables in Lakehouse | Delta |
| Shortcuts | Data mesh, virtualized access | ✅ | Virtual path in Lakehouse | As-is |
| Manual Uploads | Ad hoc files | ✅ | Files tab in Lakehouse | CSV/Excel/etc. |
| Power BI Import | Reporting data | ✅ | Power BI model memory | Tabular |

## 🔍 What Is Mirroring in Microsoft Fabric?

Mirroring is a zero-ETL (Extract-Transform-Load) feature that lets you mirror external database tables into Fabric, so they are always up to date and ready for analytics.

Mirroring in Fabric is a low-cost and low-latency solution to bring data from various systems together into a single analytics platform. You can continuously replicate your existing data estate directly into Fabric's OneLake from a variety of Azure databases and external data sources.

With the most up-to-date data in a queryable format in OneLake, you can now use all the different services in Fabric, such as running analytics with Spark, executing notebooks, data engineering, visualizing through Power BI Reports, and more.

Mirroring in Fabric allows users to enjoy a highly integrated, end-to-end, and easy-to-use product that is designed to simplify your analytics needs. Built for openness and collaboration between Microsoft, and technology solutions that can read the open-source Delta Lake table format, Mirroring is a low-cost and low-latency turnkey solution that allows you to create a replica of your data in OneLake which can be used for all your analytical needs.

The Delta tables can then be used everywhere Fabric, allowing users to accelerate their journey into Fabric.

## Why use Mirroring in Fabric?

Today many organizations have mission critical operational or analytical data sitting in silos.

Accessing and working with this data today requires complex ETL (Extract Transform Load) pipelines, business processes, and decision silos, creating:

- Restricted and limited access to important, ever changing, data
- Friction between people, process, and technology
- Long wait times to create data pipelines and processes to critically important data
- No freedom to use the tools you need to analyze and share insights comfortably
- Lack of a proper foundation for folks to share and collaborate on data
- No common, open data formats for all analytical scenarios - BI, AI, Integration, Engineering, and even Apps

Mirroring in Fabric provides an easy experience to speed the time-to-value for insights and decisions, and to break down data silos between technology solutions:

- Near real time replication of data and metadata into a SaaS data-lake, with built-in analytics built-in for BI and AI

The Microsoft Fabric platform is built on a foundation of Software as a Service (SaaS), which takes simplicity and integration to a whole new level. To learn more about Microsoft Fabric, see What is Microsoft Fabric?

Mirroring creates three items in your Fabric workspace:

- Mirroring manages the replication of data and metadata into OneLake and conversion to Parquet, in an analytics-ready format. This enables downstream scenarios like data engineering, data science, and more.
- A SQL analytics endpoint
- A Default semantic model

In addition to the SQL query editor, there's a broad ecosystem of tooling including SQL Server Management Studio (SSMS), the mssql extension with Visual Studio Code, and even GitHub Copilot.

Sharing enables ease of access control and management, to make sure you can control access to sensitive information. Sharing also enables secure and democratized decision-making across your organization.

## Types of mirroring

Fabric offers three different approaches in bringing data into OneLake through mirroring.

- **Database mirroring** – Database mirroring in Microsoft Fabric allows replication of entire databases and tables, allowing you to bring data from various systems together into a single analytics platform.
- **Metadata mirroring** – Metadata mirroring in Fabric synchronizes metadata (such as catalog names, schemas, and tables) instead of physically moving the data. This approach leverages shortcuts, ensuring the data remains in its source while still being easily accessible within Fabric.
- **Open mirroring** – Open mirroring in Fabric is designed to extend mirroring based on open Delta Lake table format. This capability enables any developer to write their application's change data directly into a mirrored database item in Microsoft Fabric, based on the open mirroring approach and public APIs.

# Creating Pipeline

Go to workspace in the left pane

✅ **Step 1: Go to Microsoft Fabric Workspace**

1. Open Microsoft Fabric

2. On the left pane, click **Workspaces**

3. Select the workspace where you want to create the pipeline

---

✅ **Step 2: Create a New Pipeline**

1. Click the **“New”** button at the top-left of the workspace

2. Select **“Data pipeline”**

🎯 If you don't see it, make sure you're in a **Fabric-enabled workspace**, not just a Power BI workspace.

3. Give your pipeline a name (e.g., SalesDataPipeline)

4. It opens the **pipeline canvas**

---

✅ **Step 3: Add Copy Activity**

1. In the pipeline canvas, go to the **Activities pane** on the left

2. Under **“Move & Transform”**, drag the **Copy data** activity into the canvas

---

✅ **Step 4: Configure the Copy Activity**

**▯ Step 4.1: Add Source**

1. Click on the Copy Activity to open the right-side configuration panel

2. Under the **Source** tab, click **“Add source”**

3. Choose a **Linked service** or click **“+ New”** to create a new one

4. Select your source type (e.g., Azure Blob Storage, SQL DB, Excel, etc.)

5. Provide connection details and credentials

6. Once connected, select the source dataset or table

---

🌐 **Step 4.2: Add Destination (Sink)**

1. Now go to the **Sink** tab

2. Click **“Add destination”**
3. Again, choose or create a **Linked service** for the destination
    - For example: Lakehouse, OneLake folder, Data Warehouse
4. Specify the destination folder or table
5. Optionally, configure file formats (e.g., Delta, CSV, Parquet)

---

## ⚙ Optional Configurations

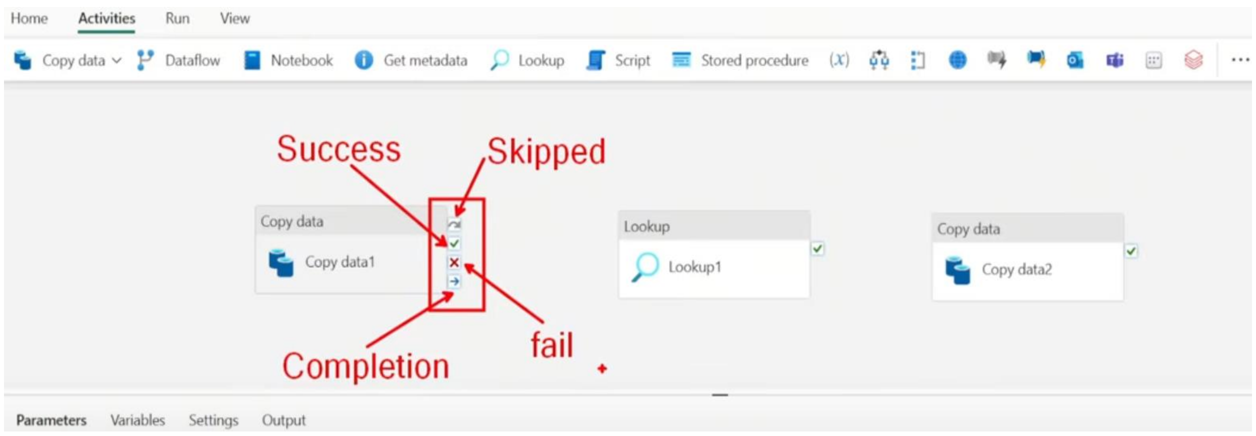
- In the **Settings** tab:
    - Set data integration units (DIUs)
    - Enable logging
    - Set up fault tolerance (skip error rows, retry count, etc.)

---

## ☑ Step 5: Trigger or Schedule the Pipeline

- Click **“Add trigger”** at the top → Choose **Manual** or **Scheduled**
- Or simply click **Run** to test the pipeline now

---



# Practical

## Scenario 1: Copy one file from github to onelake

### 🔁 Step-by-Step Process

### 🔷 Step 1: Open Microsoft Fabric and Go to Workspace

1. Navigate to Microsoft Fabric
2. Go to your desired **workspace**
3. Ensure the workspace has **Fabric enabled**

---

### 🔷 Step 2: Create a Lakehouse (if not already done)

1. Click **New** → **Lakehouse**
2. Give it a name (e.g., CustomerLakehouse)
3. Open the Lakehouse and note the **"Files"** and **"Tables"** section

---

### 🔷 Step 3: Create a New Pipeline

1. In the same workspace, click **New** → **Data pipeline**
2. Name it something like GitHubToOneLakePipeline

---

### 🔷 Step 4: Add a Copy Activity

1. In the pipeline canvas, from the left-side panel, drag **Copy Data** into the canvas
2. Click on the activity to configure it

---

### 🔷 Step 5: Configure the Source (GitHub)

1. Go to the **Source** tab
2. Click **"Add source"**

3. Choose **HTTP** as the source type
4. Create a new **Linked service**:
    - Name: GitHubCSVLink
    - Base URL: https://raw.githubusercontent.com/
5. In the dataset configuration:
    - Relative path: pragyachoukade/Fabric-Tutorial/main/Data/AdventureWorks_Customers.csv
    - File format: Delimited text (CSV)
    - Column delimiter: ,
    - First row as header: ✅

---

### 🔷 Step 6: Configure the Sink (Destination in OneLake)

1. Go to the **Sink** tab
2. Click **"Add destination"**
3. Select your **Lakehouse** (e.g., CustomerLakehouse)
4. Choose to store data as a **table** or in **Files** (we have used files)
5. Give the directory name as raw data and file name as Adventure.customer.csv
6. Choose format: Delimited text

---

### 🔷 Step 7: Validate and Run

1. Click **Validate** to make sure everything is set up correctly
2. Click **Run** → Track progress in the **Output pane**
3. Once completed, open your **Lakehouse → Files** → You'll see the Adventure.customer.csv file

**Sceanrio2: If we want to copy all files then how we can do, so we can do this with dynamic pipeline or parameterized pipeline**