Fabric Data Engineer
DP-700

Total : ***50-60*** *Questions*
(including one case study of around 10 Questions)
(~ 15-20 Questions from Each Section)

Implement and Manage an Analytics Solution

- Workspace settings
- Version Control & Deployment
- Access Control
- Data Governance
- Orchestration

Ingest and Transform Data

- Data ingestion
- Full and incremental loading
- Data stores: Lakehouse, Data
- Warehouse, Eventhouse
- Data transformation
- Streaming data
- Languages: T-SQL, KQL, PySpark

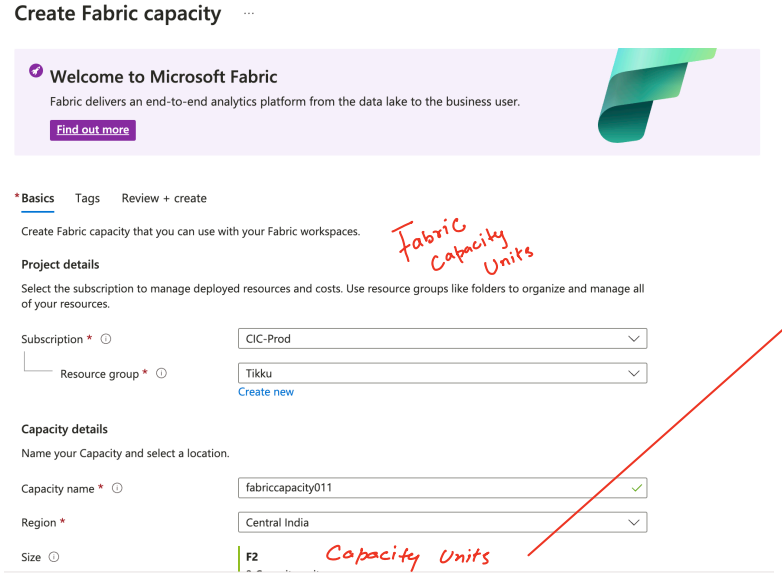
Monitor and Optimize an Analytics Solution

- Monitoring of Fabric items
- Error handling
- Optimizing solutions in Fabric

FABRIC; It weaves together Modern Data Stack

Microsoft Fabric is an end-to-end data platform by Microsoft that unifies data engineering, data science, real-time analytics, business intelligence, and data governance in one SaaS environment. Think of it as the next generation of Power BI + Azure Synapse + Data Factory + OneLake, all rolled into one seamless experience.

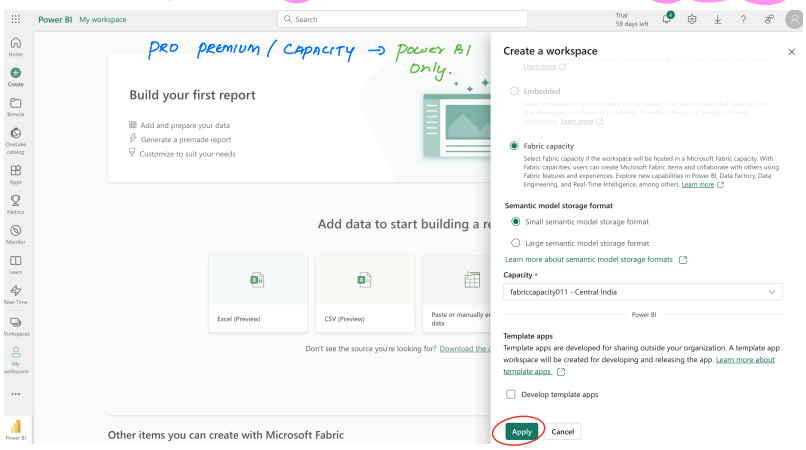
FABRIC CAPACITY; Create capacity



Plans

SKU	Capacity Units	COST (ESTIMATED/MONTH)
F2	2	₹23,136.84
F4	4	₹46,273.68
F8	8	₹92,547.37
F16	16	₹1,85,094.74
F32	32	₹3,70,189.48
F64	64	₹7,40,378.95
F128	128	₹14,80,757.90
F256	256	₹29,61,515.80
F512	512	₹59,23,031.61
F1024	1024	₹1,18,46,063.21
F2048	2048	₹2,36,92,126.43

Let's create a Fabric Workspace.



Types of LICENCES

License Model	Acronym	What You Can Use	Note
Free	-	Cannot view shared Power BI objects	Limited to personal use
Pro	-	Power BI	Can share and collaborate with other Pro users
Premium Per User	**PPU**	Power BI	Can share with other PPU users, but not Free or Pro users
Premium Per Capacity	**P**	Power BI, Fabric	Free users can view shared Power BI objects (up to June 2018) → Mostly Used
Embedded	**A / EM**	Power BI	For embedding reports in apps, requires separate setup
Fabric Capacity	**F**	Fabric	- If **F32 or below**, you need a **Power BI paid license** to view shared objects - If **F64 or above**, free users can view shared Power BI objects → Mostly Used

F-SKU

Important = Pause when Fabric is not in use

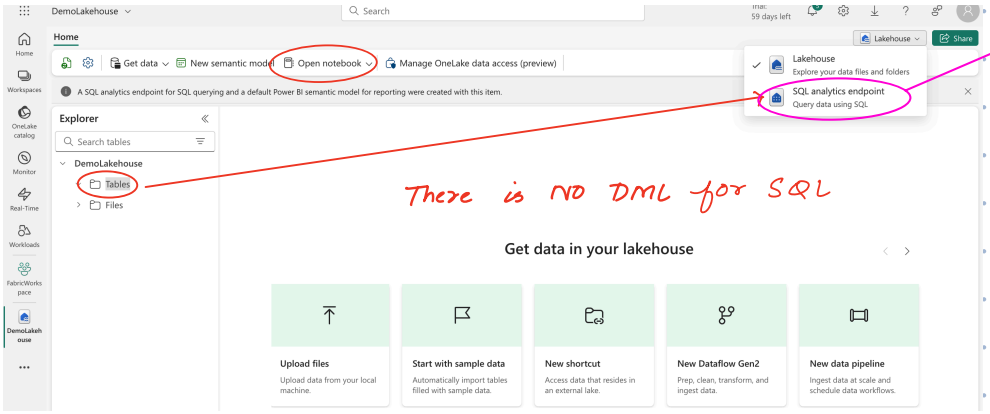
Tour to Fabric

Lakehouse; is used to store large amounts of Data

- cleaning, Querying, Reporting & Sharing

A **Data Lake** is like a giant storage space for all types of raw data—structured, semi-structured, and unstructured—without strict organization.

A **Data Lakehouse** combines the flexibility of a Data Lake with the structured querying and performance of a Data Warehouse, making it easier to analyze and process data efficiently.



Lakehouse, SQL is Read only which means DML won't work

DML works in DW

Notebook; to write code in Pyspark (Python + Spark), SQL to analyze, visualize Data. (Vs code Notebooks)

Pipeline; Set of Activities (Transform & Transport data)

Shortcut; Access Data residing in an External Lake

Data Flow Gen 2; Power Query.

Warehouse; Business Analytics on multiple Data sources

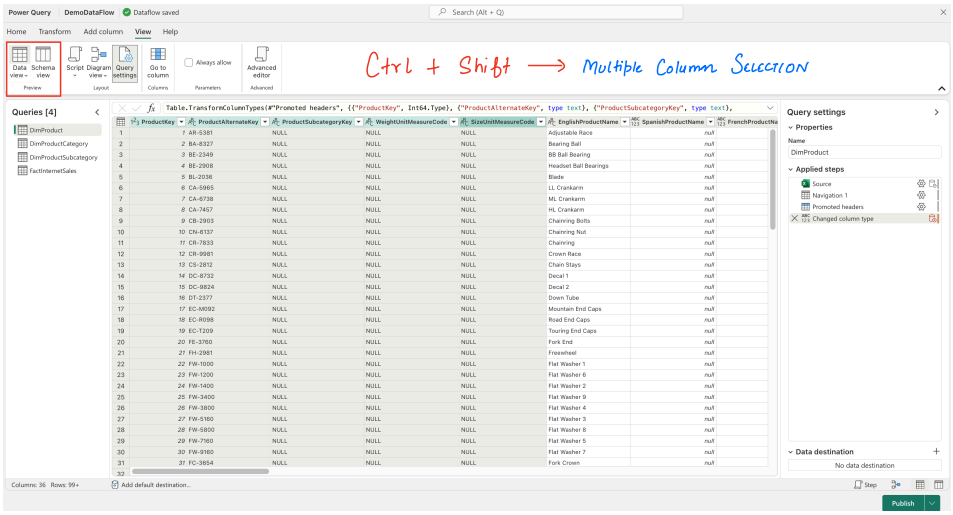
Lakehouse Vs **Warehouse**;

Feature	Data Lakehouse	Data Warehouse
Primary Skill Set	PySpark, Spark SQL	SQL
Data Structure	Folders, files, databases, tables	Databases, schemas, tables
Supported Data Types	✅ Structured, ✅ Semi-structured, ✅ Unstructured	✅ Structured only
Read Operations	Spark, T-SQL No DML	T-SQL Yes DML
Write Operations	Spark	T-SQL
Multi-Table Transactions	❌ No	✅ Yes
Security	Row-level, table-level	Object-level, column-level, row-level, DDL/DML, dynamic data masking
Access Data via Shortcuts?	✅ Yes	❌ Indirectly through Lakehouse

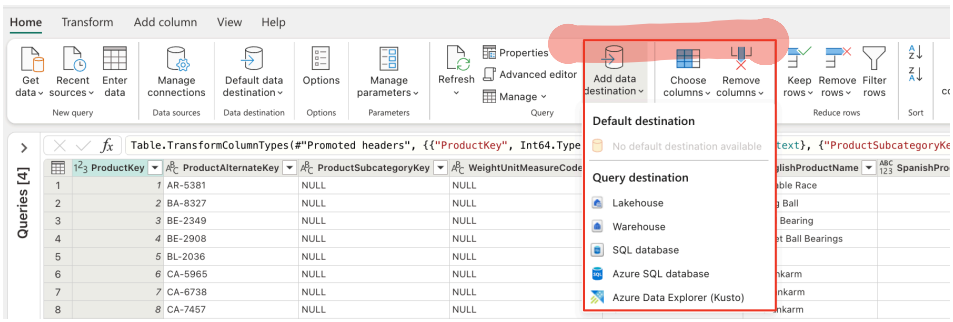
F62 to view Power BI Objects.

more security

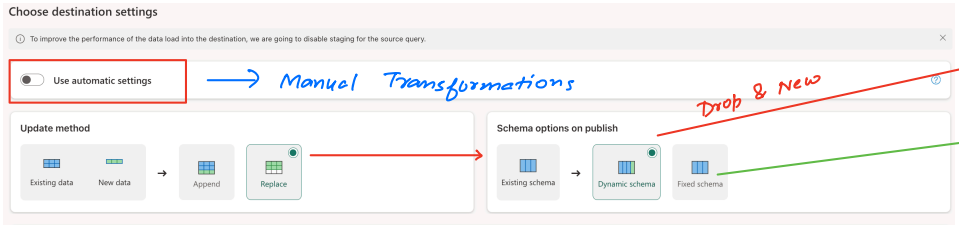
DATAFLOW GEN 2 ; Data Ingestion (100 Data Sources)



ADD DESTINATION ; Important Step



workspace
↓
Destination

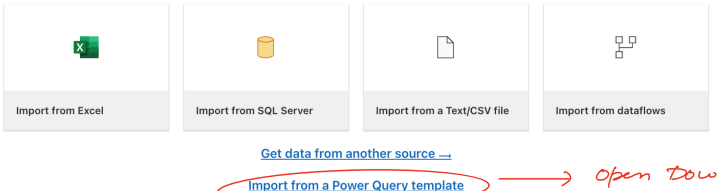


SAVE DATA FLOW AS TEMPLATE

Parquet format

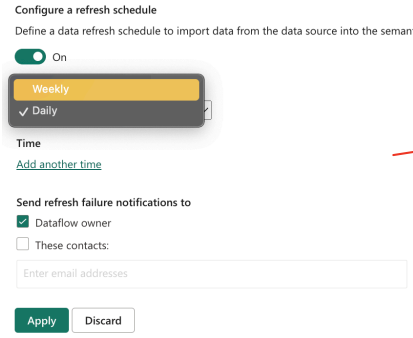
Export template → Saved as pqt → Go to Import Gen 2

Power Query Template



→ Open Downloaded pqt file

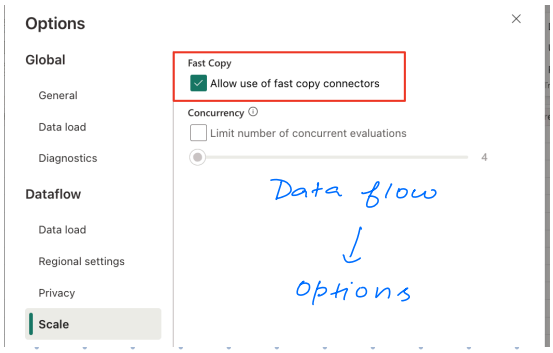
REFRESH DATAFLOW

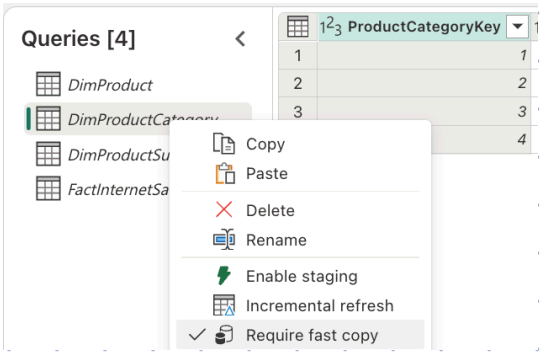


→ Every 30 Minutes

Implement using Fast Copy in Dataflows

Fastcopy; is designed to optimize the copy of large Data





Limitations:

- **Data Size:** The maximum file size supported for Fast Copy is typically **100 MB per file.**
- **Rows:** It is optimized to handle up to **5 million rows** in a single copy operation efficiently.

Monitor Data Transformations in Dataflows;

Dataflow ⟶ Go to Refresh History Details.

Optimize Dataflows; For Better performance

Technique	Description	Best Practice
Query Folding	Push computations back to the data source to leverage database optimizations.	Use native queries at the source for filtering, aggregating, etc.
Separate Load and Transform Dataflows	Split data loading and transformation into separate flows for better manageability and performance.	Run separate Dataflows: one for loading, one for transforming.
Incremental Refresh	Only refresh data that has changed, rather than the entire dataset.	Use **Change Data Capture (CDC)** or time-based filters for incremental loads.
Minimize Data Movement	Reduce unnecessary data movement by applying filters early in the flow.	Filter data at the source to move only relevant data.
Use Multiple Data Flows in Parallel	Run multiple Dataflows in parallel to process large volumes of data simultaneously. using pipelines	Break tasks into smaller, parallelized jobs to improve speed.

Transforming Data using Dataflow;

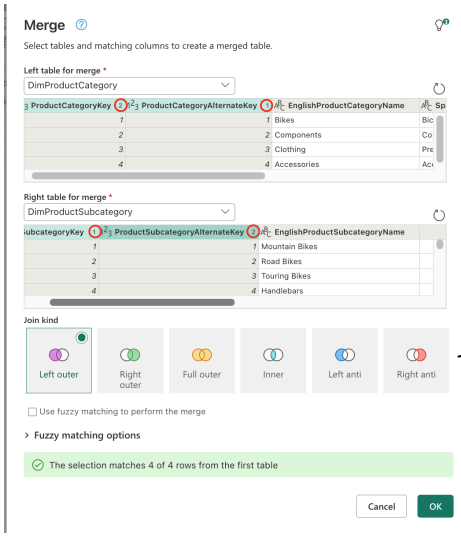
Tips; • Use Locale (Data type) in case of Error
• Ctrl + Shift + Select (Multiple columns)
↳ can be used to change DType also

Choose columns

- Explore Date time Filters
- Merge ⟶ Join ⟶ Add Columns
- Append ⟶ Union ⟶ Add Rows.

Imp: ↳ populates Rows with Same Column Names
Column Names should Match.

MERGE TWO TABLES; Imp.



Important;

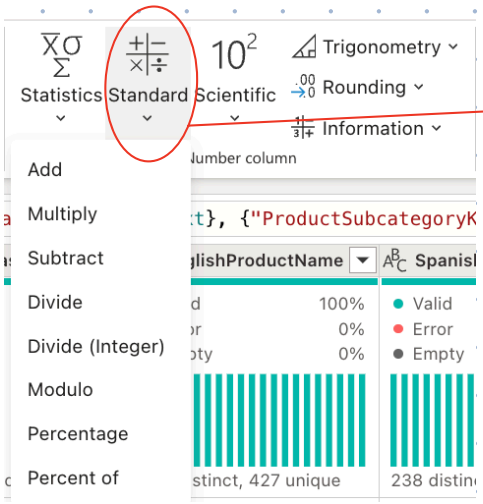
- we can merge while selecting multiple columns

Composite Key Concept

→ **Joins**

Left Anti ⟶ Exclude from R.

Right Anti ⟶ Exclude from L.



→ Add etc.

- Mark as key ⟶ Mark column as Primary Key
- Custom column ⟶ Column using M code.
- Duplicate Query ⟶ Duplicate of Query
  No changes if made in Original
- Reference Query ⟶ Shortens steps
  Changes if made in Original

DIFFERENCE BETWEEN DUPLICATE & REFERENCE

DUPLICATE = No changes to original

REFERENCE = Changes Reflects if done in parent Table.

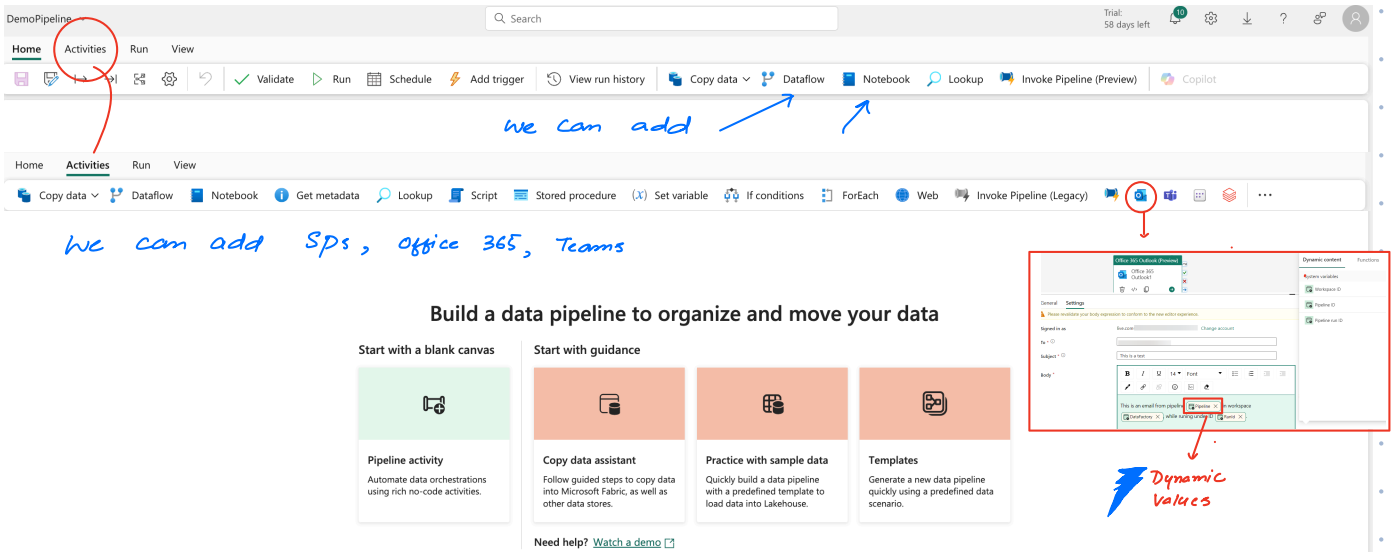
PIPELINES; Set of Activities

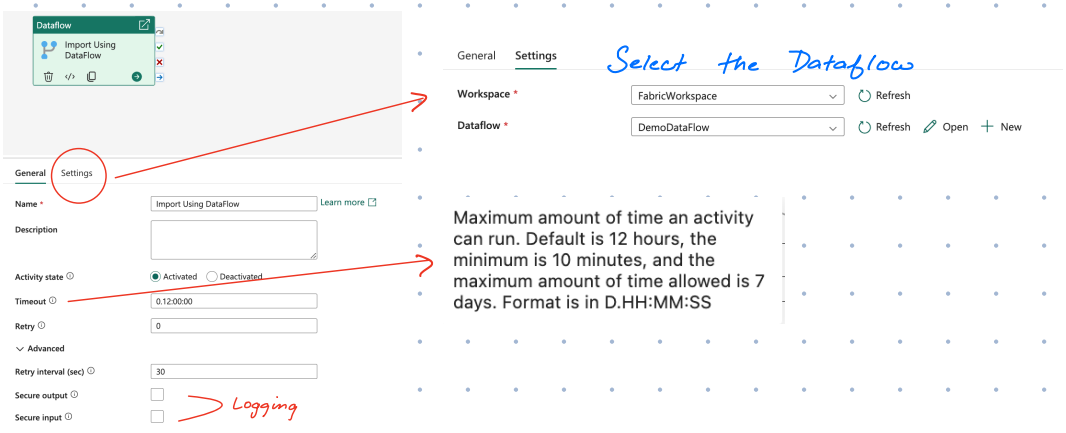
is a series of Data related tasks (activities)

- where to get the Data (Extract)
- what to do with Data (Transform)
- where to send Data (Load)

Or.

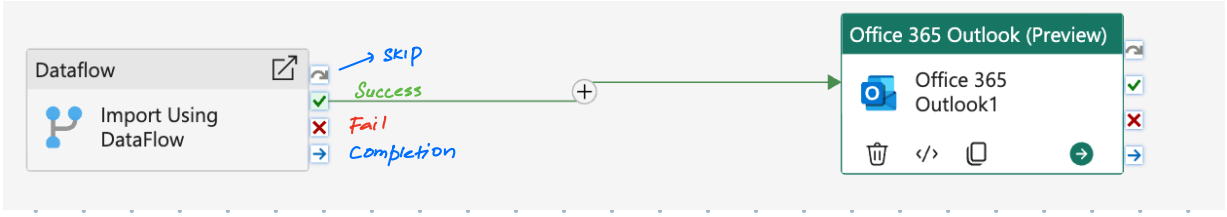
Pipeline Automates the movement and Transformation of data from one place to another





CONNECTING ACTIVITIES;

Once the Data flow is run successfully, it will then go to another activity.

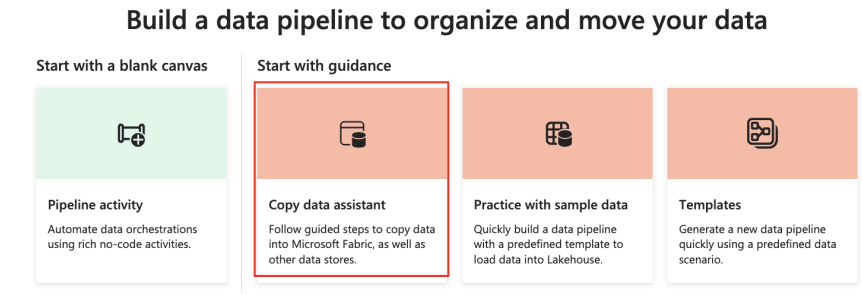


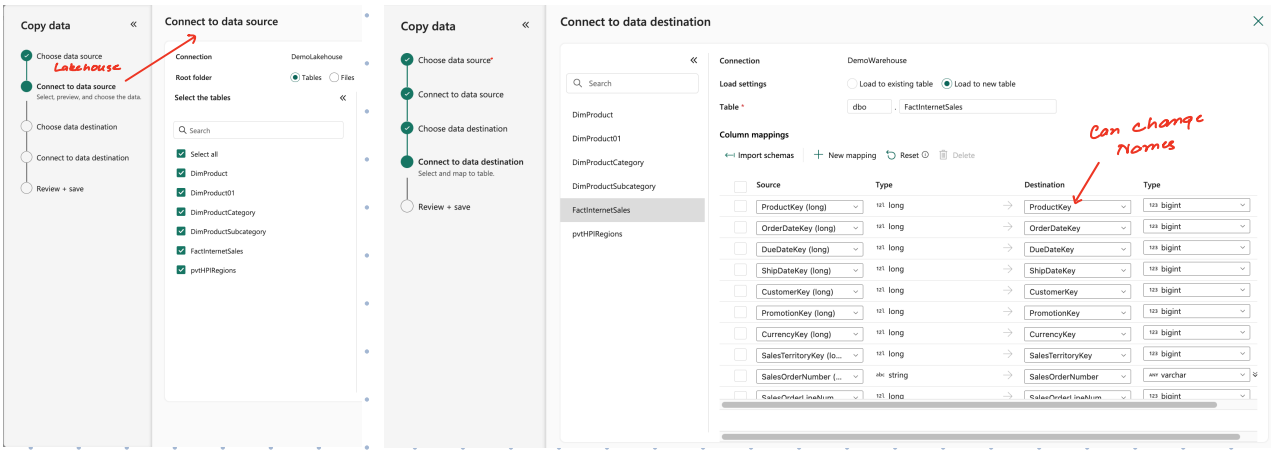
Copying Data in Pipeline;

To copy the Data, we have 2 options

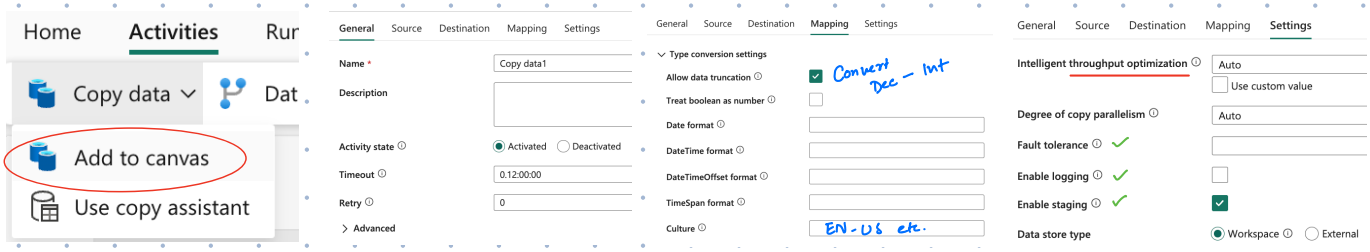
1. Using Copy Assistant (Allows Multiple Tables)
2. Using copy Activity (Allows Single Table)

1. Using Assistant





2. Using Copy Activity



What is Throughput in a Data Pipeline? Imp.

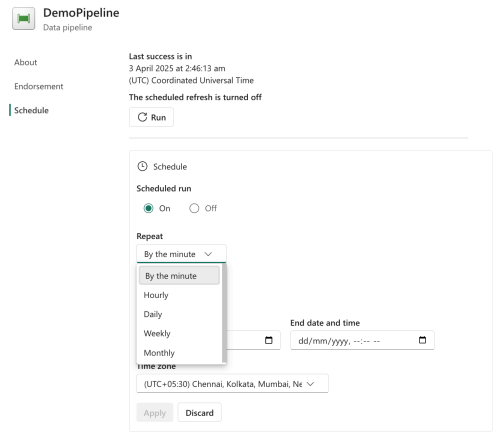
Throughput in a data pipeline refers to the amount of data that flows through the pipeline in a given time period. It is typically measured in **bytes per second (Bps), records per second, or events per second** and is a key performance metric for evaluating data pipeline efficiency.

Degree of Copy Parallelism?

In **Azure Data Factory (ADF)**, when moving data from SQL Server to Blob Storage, setting **Degree of Copy Parallelism = 4** means ADF will run **4 parallel data streams** to speed up the copy process.

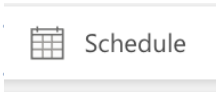
Schedule & Monitor Data Pipelines

Schedule Pipeline;

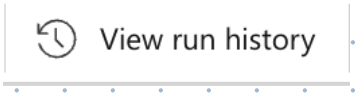


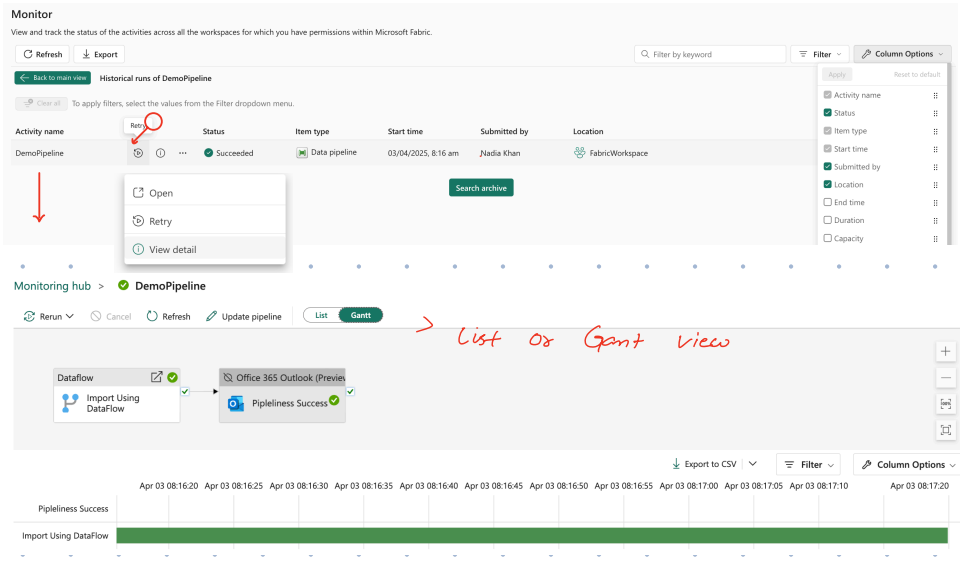
Pipelines

↓

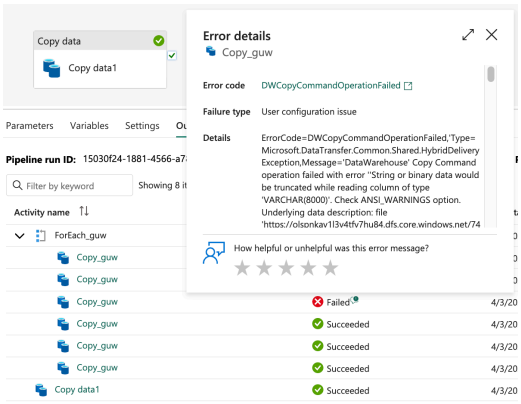


Monitor Pipeline;

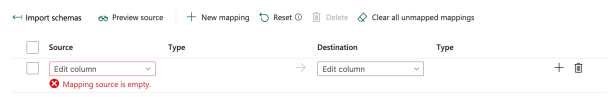




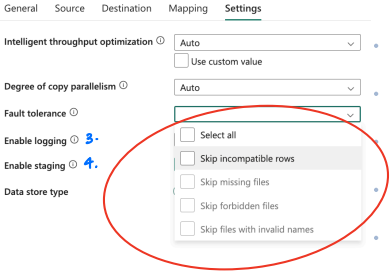
Resolve Pipeline Issues;



1. New Mapping in Mapping



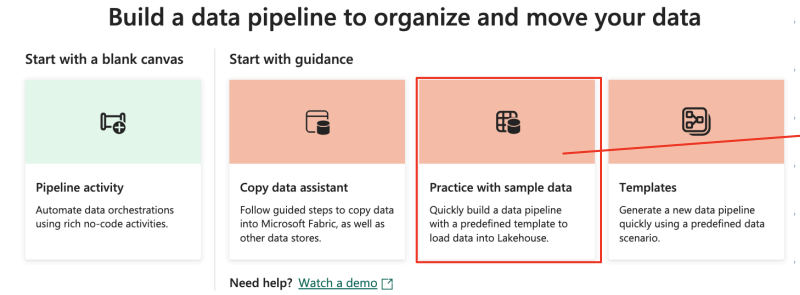
2. Fault Tolerance



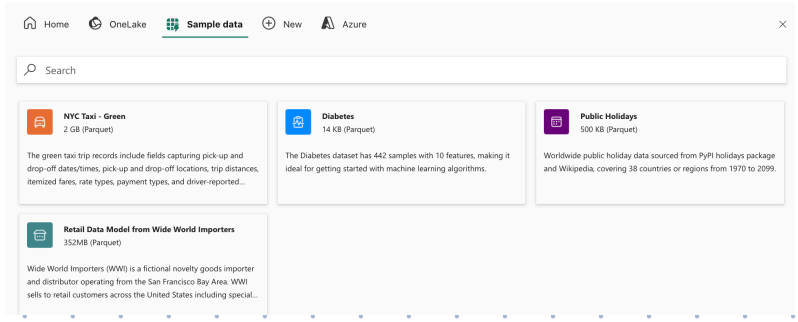
Important; warehouse doesn't support varchar(max)

Azure SQL DB / SA = port 1433

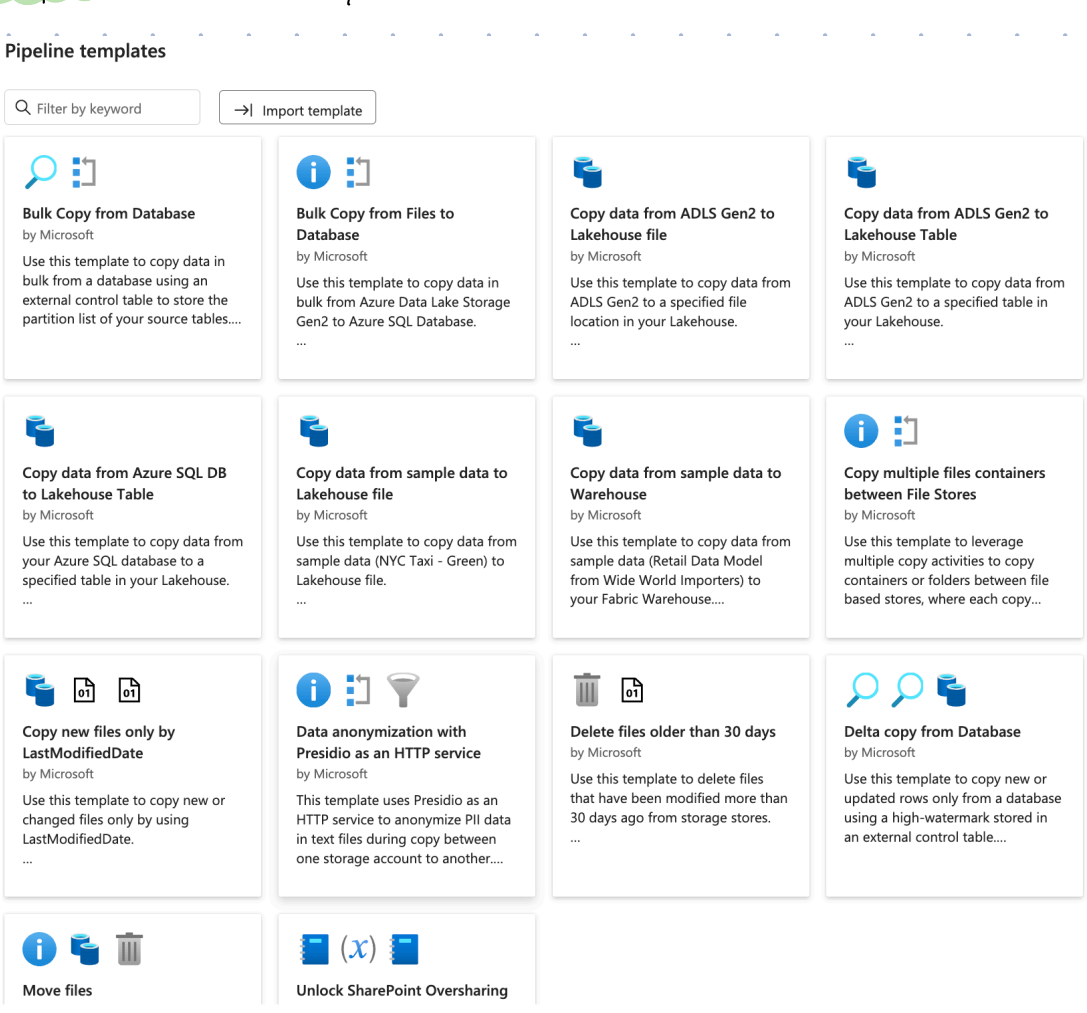
Sample Data in Pipelines:



→ Only NY Taxi Data



Pipeline Templates:



Important You cannot use MySQL Database as an incomming connector for "Fast copy"

NOTEBOOKS; is just like a Jupyter Notebook where we can run + write code but it is fully integrated with Fabric.

💬 Supported Languages:

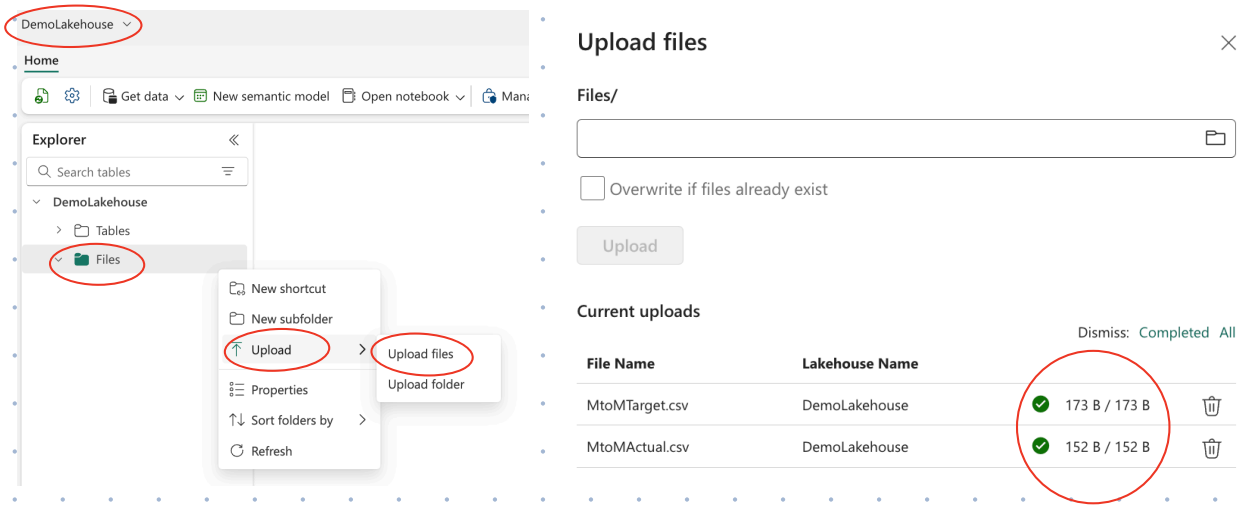
Fabric notebooks are built on **Apache Spark** and support:

- **Python** (most common for data science & ETL)
- **SQL**
- **Scala**
- **Spark R** (optional)

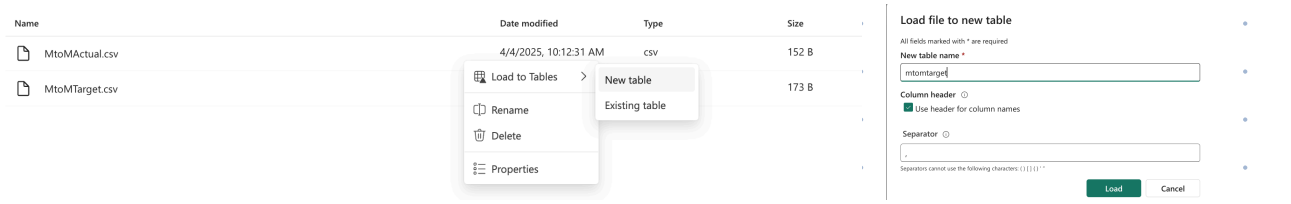
✅ What Can You Do in a Fabric Notebook?

Task	Example
🧹 Data Cleaning	Use Python (Pandas/Spark) to clean messy datasets
🔍 Data Exploration	Run queries on Lakehouse tables using SparkSQL
📊 Visualize Data	Use `matplotlib`, `seaborn`, or built-in Fabric charts
📦 ML/Modeling	Train models with `scikit-learn`, `MLlib`, etc.
🔄 Pipeline Integration	Run as part of a pipeline activity (trigger notebooks in Fabric pipelines)

INGEST DATA USING LOCAL UPLOAD

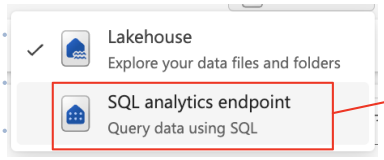


we can load these files as Tables



Important; Does it make any difference if Data sits inside Tables or Files; (In Lakehouse)

Answer Yes, because Tables are processed Data Tables will be Quicker



→ Only Tables are Accessible.

Microsoft One Lake Explorer

✓ → Success
🔄 → Sync
☁ → Online Only

→ A software that can be used locally to upload files (On windows)

Appropriate Method To Copy To LH or WH

It depends on multiple factors.

- Small files ⟶ Local Copy or One lake Explorer (Directly)
- Small files ⟶ Data flows (Gen 2) (with Transform)
- Large Data ⟶ Data pipeline (Copy Tool) (without Transformations)
- Data ⟶ Notebook Code (Complex Transform)

Ingest Data Using Notebook

Create Notebook ⟶ Add Destination.

Will focus on Pyspark → Python API for Spark

In short, use Spark using Python

✅ **Use Pandas** if you're working on local datasets (like Excel/CSV files under a few GB).
🔥 **Use Spark** if you're dealing with huge datasets (logs, transactions, sensor data, etc.) or working in a distributed environment.

ABFS ⟶ Azure Blob File System

Read CSV

Loading data from csv file into a dataframe using Spark

```
df = spark.read.option("header","true").format("csv").load("Files/MtoMActual.csv")

df2 =
spark.read.csv("abfss://FabricWorkspace@onelake.dfs.fabric.microsoft.com/DemoLakehouse.Lakehouse/Files/MtoMActual.csv")

df2 =
spark.read.load("abfss://FabricWorkspace@onelake.dfs.fabric.microsoft.com/DemoLakehouse.Lakehouse/Files/MtoMActual.csv",
format='csv', header=True)
```

Loading data from csv file into a dataframe using Pandas

```
import pandas as pd
# Load data into pandas DataFrame from "/lakehouse/default/" + "Files/MtoMActual.csv"
df = pd.read_csv("/lakehouse/default/Files/MtoMActual.csv")
display(df)
```

Saving data from a dataframe to a csv file or table

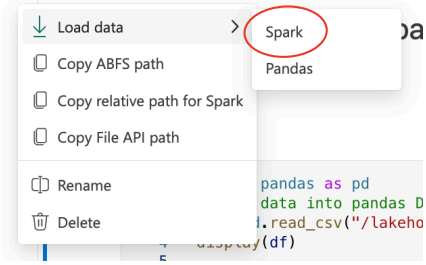
```
df.write.mode("overwrite").format("csv").save("Files/MtoMActual2.csv")

df.write.mode("overwrite").format("delta").saveAsTable("MtoMActual")
```

Loading data from a table

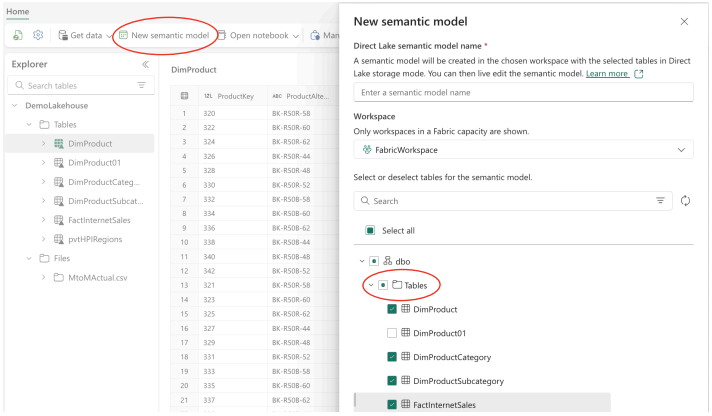
```
df = spark.read.table("Mtomactual")
display(df)

df2 = spark.read.format("delta").load("Tables/mtomactual")
```

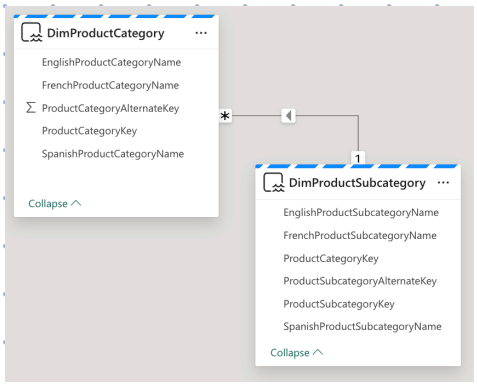
Use

display(df)



Creating Semantic Model Tables, Data model





Loading Data From Table in PySpark & SQL

In Notebooks, SQL is Read only (Lake house) for manipulations,

↳ Use Spark or pandas

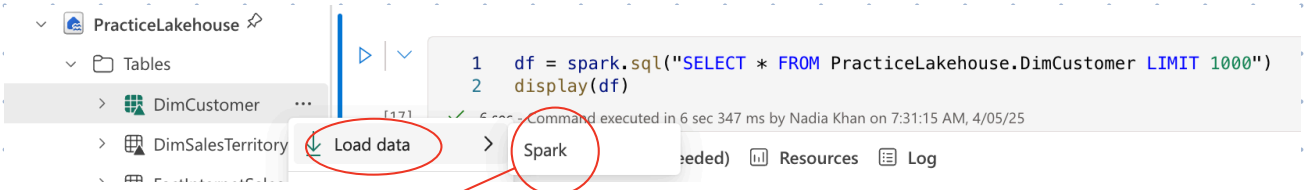
```
1  %%sql
2  SELECT * FROM mtomactual
```

Read Only.

Spark SQL

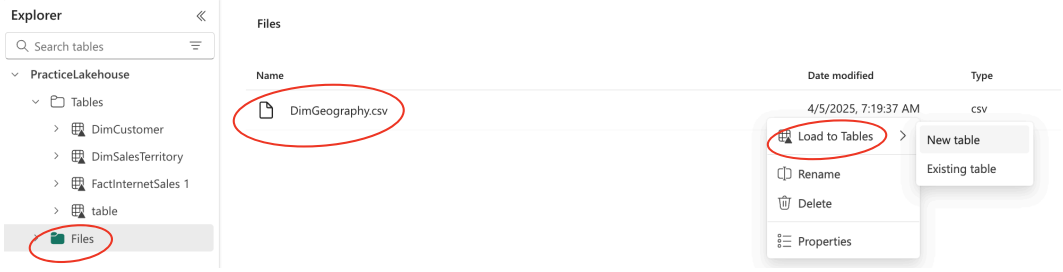
Process to Load Data into Tables

Lakehouse → Files → Upload CSV → Load as Table



From Tables, Data can be loaded by SQL Spark by default.

Tip: Always upload Tabular Files in Tables so they serve as processed data for analytics SQL Endpoint



DATETIME FORMATS IN PYSPARK.

Symbol	Meaning	Example
`yyyy`	Year	`2025`
`MM`	Month (2 digits)	`01` to `12`
`MMM`	Month (short name)	`Jan`, `Feb`
`MMMM`	Month (full name)	`January`, `February`
`dd`	Day of month (2 digits)	`01` to `31`
`E` / `EEEE`	Day of week	`Tue`, `Tuesday`
`HH`	Hour (24-hour format)	`00` to `23`
`hh`	Hour (12-hour format)	`01` to `12`
`mm`	Minutes	`00` to `59`
`ss`	Seconds	`00` to `59`
`a`	AM / PM	`AM`, `PM`
`S` / `SSS`	Fraction of second (ms)	`123` (milliseconds)
`Z`	Time zone offset	`+0530`, `-0800`
`'text'`	Escape text	`'at'` shows as `at`

Concat in Pandas / pyspark

Pandas ⟶ Doesn't need similar structure

Pyspark ⟶ Needs similar structure

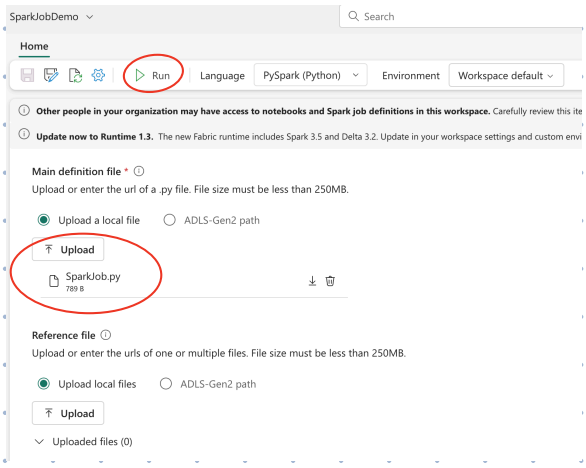
FillNa ⟶ Coalesce ⟶ IF(NA, fill) in SQL

Optimize Notebooks

. we can schedule Runs on our Notebook

Spark Job Definition Set of Tasks that we want to apply on Data

```
from pyspark.sql import SparkSession
from pyspark.sql.types import StructType, StructField, StringType
spark = SparkSession.builder.getOrCreate()

schema = StructType([StructField('Date', StringType(), True),
StructField('Subtype', StringType(), True),
StructField('PurchaseMethod', StringType(), True),
StructField('Out', StringType(), True)])

dfs = spark.readStream.option("header", "true").schema(schema).format("csv").load("Files/Shop*.csv")
deltatablepath = "Tables/Shop_table"
query = dfs.writeStream.format("delta").outputMode("append") \
        .option("checkpointLocation", deltatablepath + '/_checkpoint') \
        .option("path", deltatablepath).start()
query.awaitTermination()
```



Save this code in .py file which will later be used to create Spark Job Definition

IMPORTANT; Spark Jobs help run spark code continuously.

Choose Between Pipeline, Dataflow Gen2, Notebook

← No code / Low code → Code

Feature	Pipeline (Copy Activity)	Dataflow Gen2	Notebook
Use Case	Data ingestion, warehouse migration, basic transformation	Data lake transformation, data wrangling	Complex data transformation, profiling, machine learning, large-scale processing
Primary Developer Persona	Data Engineer, Data Integrator	Data Engineer, Business Analyst	Data Scientist, Data Engineer, Developer
	Migration	Data Transformation	Complex Transforms
Skill Set Required	ETL tools, SQL, JSON	ETL, Power Query (M), SQL	Python, Scala, Spark SQL, R
Development Interface	Visual Wizard, Canvas	No-code / Low-code Graphical Interface	Code-based Notebook Interface
Sources Supported	30+ built-in connectors	150+ built-in connectors	100s of Spark-compatible libraries
Destinations Supported	18+ connectors	Limited connectors	100s of Spark-compatible output formats
Complexity & Transformation	Low — lightweight tasks like column mapping, type conversion, flattening	Medium — 300+ built-in transformations, native & custom logic	Low to High — native Spark transformations, custom logic, advanced processing

Pipelines → simple Data movement / light Trans.

Dataflow Gen 2 → Transformation (less sources)

Notebook → Complex logic / ML etc.

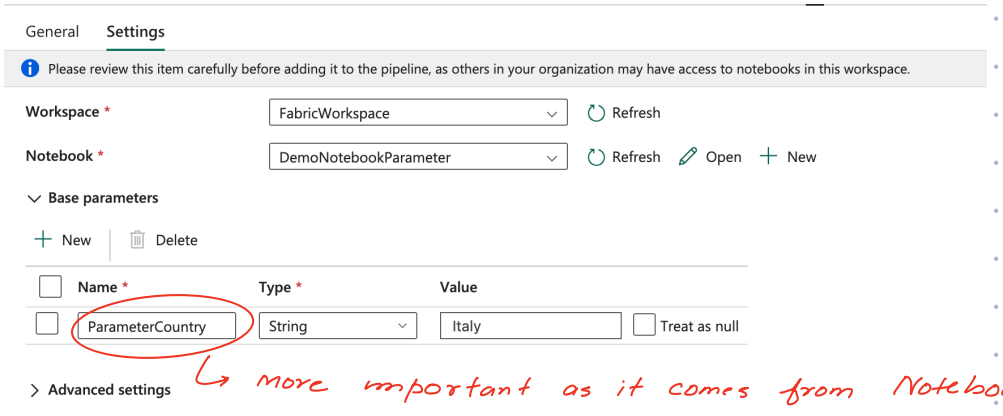
Parameters in Notebook & Pipelines

Step I; Notebook Parameter;

Check in SQL Endpoint

Step II; Pipeline Parameter

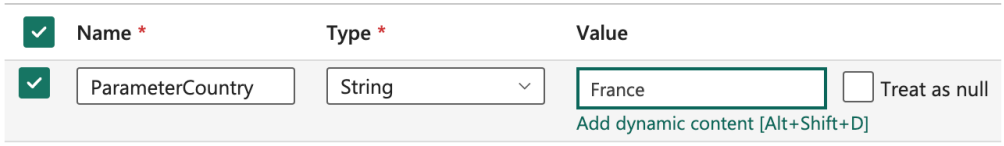
New pipeline → Notebook

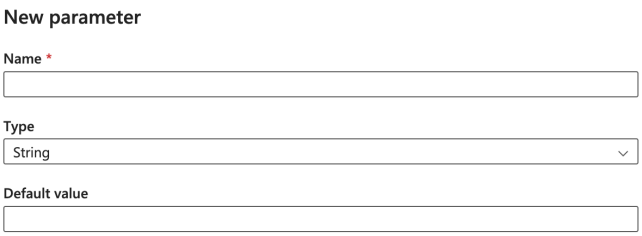


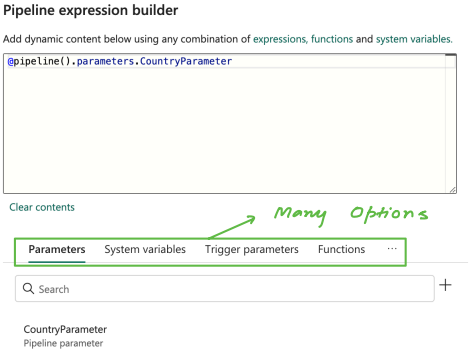
↳ more important as it comes from Notebook

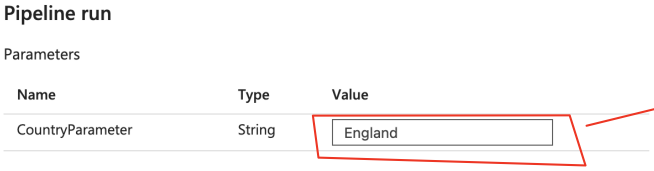
↳ we can Run and change values and it will reflect in SQL Endpoint

Using Dynamic Expressions

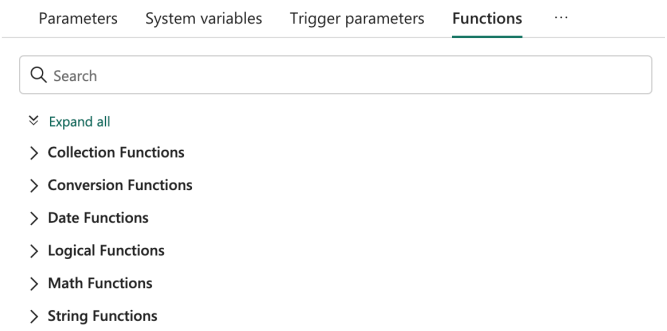








So next time I will get this prompt to enter Parameter on Run

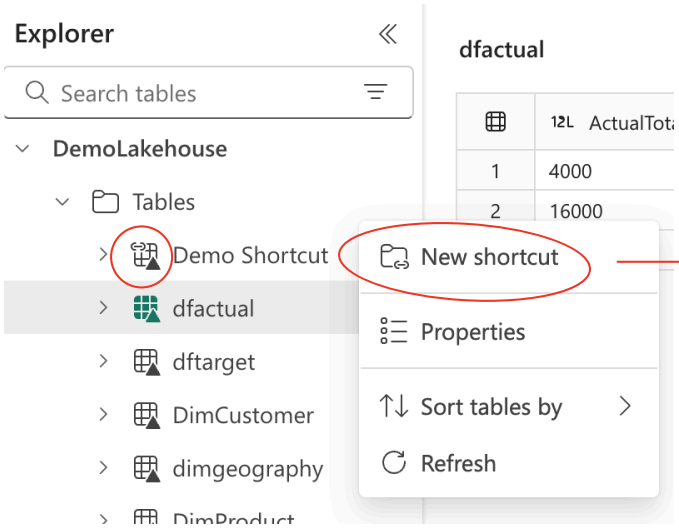


Important

Item Snapshots = It shows all the code and any output in the code

Spark Job definition; allows to Run code continuously.

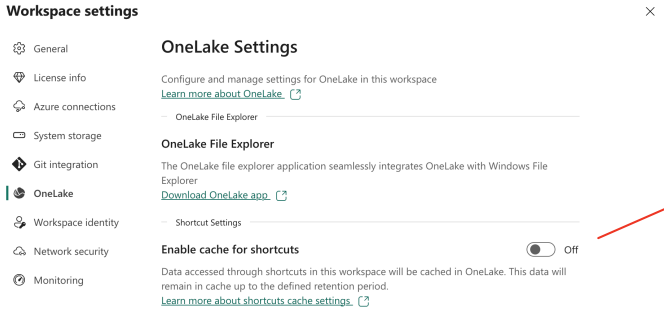
SHORTCUTS



Can create shortcuts from OneLake

Doesn't copy Data but only creates shortcut

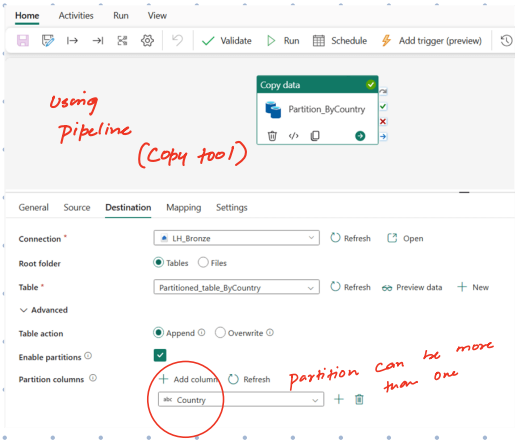
OneLake Workspace Settings



Caches Data from other sources (Amazon S3)

Cache upto 1 GB

Partition Data from DB



Note; partitioning will allow to save data in partitions (parquet)

Folders / Files / Country = India

`spark.read.parquet(path)`

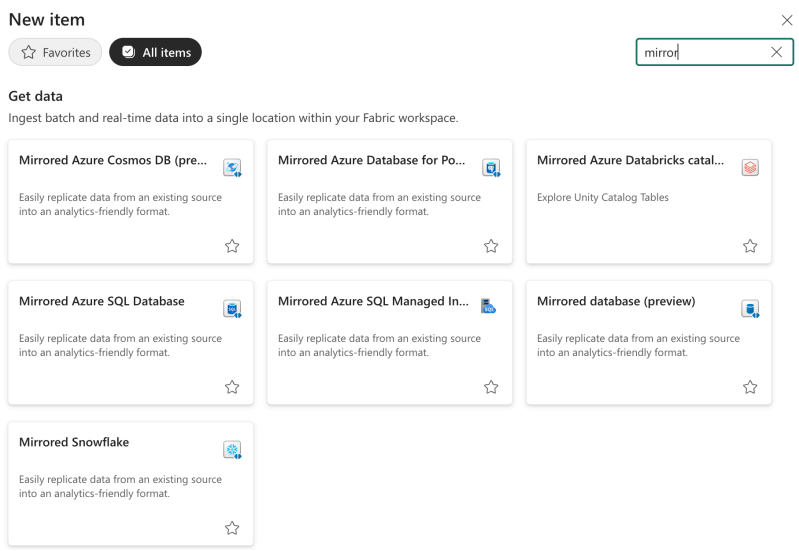
we can query whole table and also partitions

PARTITION DATA USING PYSPARK CODE IN LAKEHOUSE

```python
df.write.mode("overwrite").format("delta") \
  .partitionBy("Color", "ProductSubcategoryKey").saveAsTable("DimProductPartitioned")
```

Organisational = Entra ID

MIRROR EXTERNAL DB; Azure mirroring external DB simply helps in copying and syncing data from external data sources into Azure (SQL, Synapse etc.) for Analytics & Real time Insights

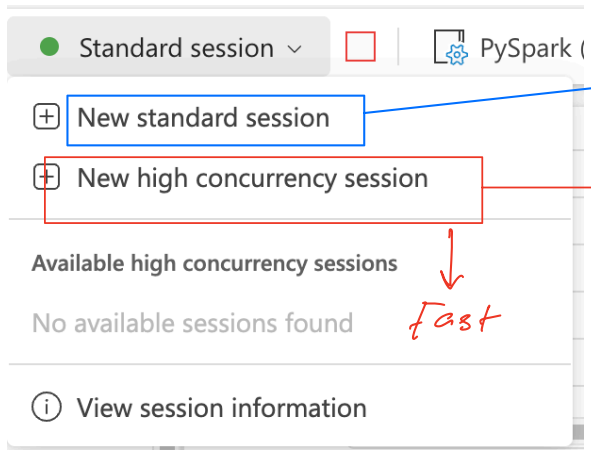


NOTE It will create
- Mirrored DB
- Semantic Model

Imp; Dropbox is not supported by shortcuts

NOTEBOOK PERFORMANCE OPTIMISATION

1. Partitioning Data before Querying
2. Same Spark Session for multiple Notebooks



For Dedicated Compute

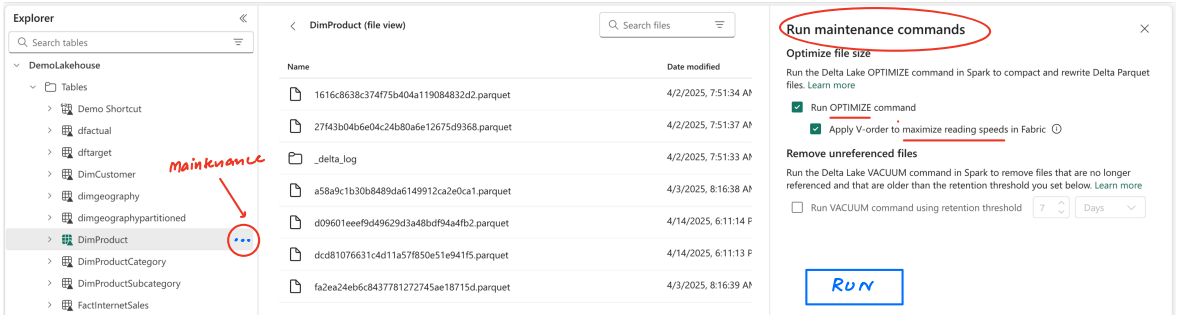
For multiple Notebooks we can use High Concurrency session for fast processing

3. Optimize write: write fewer / larger files (~128MB & 1GB)
   ↓ Optimizes file size
   Fast Reads; few small files

```python
spark.conf.set("spark.databricks.delta.optimizeWrite.enabled", "true")
```

4. VOrder; (vertical order) for column store index
   ↓ Fast Scanning / Read
   Better Compression
   Faster Scans.
   By default, NO manual work

Identify & Resolve issues with Delta Table files



Optimize Spark performance ++

1. when importing data, schema should be defined explicitly

✅ PySpark Data Types – Quick Reference Table

Data Type	PySpark Type	Description	Example
String	`StringType()`	Text values	`"Tajamul"`
Integer	`IntegerType()`	32-bit integer	`25`
Long	`LongType()`	64-bit integer	`123456789012`
Float	`FloatType()`	32-bit floating point	`3.14`
Double	`DoubleType()`	64-bit floating point	`3.1415926535`
Boolean	`BooleanType()`	True/False values	`True`, `False`
Date	`DateType()`	Date only (yyyy-mm-dd)	`2025-04-16`
Timestamp	`TimestampType()`	Date and time	`2025-04-16 14:22:00`
Decimal	`DecimalType(p, s)`	Fixed-precision decimal	`Decimal("12.34")`
Array/List	`ArrayType(elementType)`	Array of values	`[1, 2, 3]`
Map/Dict	`MapType(keyType, valueType)`	Key-value pairs	`{"a": 1, "b": 2}`
Struct (Row/Record)	`StructType([fields...])`	Nested fields/records (like JSON)	`{name: "Taj", age: 30}`

2. Avoid * and use Select specific columns needed for the Query

3. df.explain(True) ⟶ Details of Query Execution

4. Cache frequently used Tables;

   Spark.sql("CACHE TABLE mtomactual")

5. Enable Auto Tune ⟶ Automatic performance Tuning that dynamically adjusts config.
   ↳ enabled by default

Important "Enable Staging" for a query may speed up or slow down loading performance

- Optimum file size in Optimize write;

  128 MB ⟶ 1 GB

Transforming Data in Datawarehouse

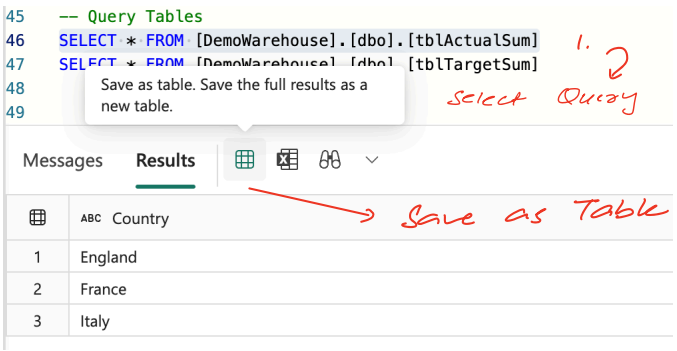
Important; we can't create a table in Lakehouse (SQL Endpoint) or Notebook because SQL is Read only

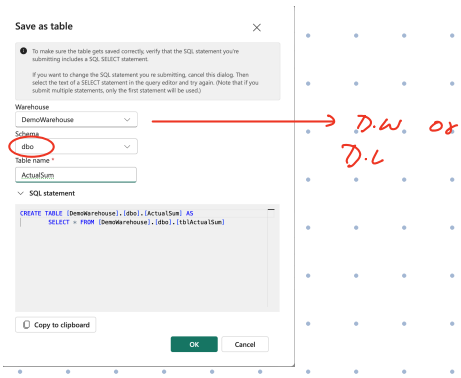
→ However Tables can be created in warehouse

Data Types in Pyspark vs SQL

Data Type Category	Range/Description	PySpark Data Type	SQL Data Type
Tiny Integer	0 to 255 (SQL unsigned) / -128 to 127 (PySpark)	`tinyint`	`tinyint` (unsigned)
Small Integer	-32,768 to +32,767	`smallint`	`smallint`
Integer	-2,147,483,648 to +2,147,483,647	`int`	`int`
Big Integer	-9,223,372,036,854,775,808 to +9,223,372,036,854,775,807	`bigint`	`bigint`
Fixed-point Numbers	Specified precision and scale	`decimal(p, s)`	`decimal(p, s)` or `numeric(p, s)`
Floating-point Numbers	Approximate real numbers	`float`, `double`	`float`, `real`
Strings	Text values	`string`, `char(n)`, `varchar(n)`	`char(n)`, `varchar(n)`
Boolean	True or False	`boolean`	`bit`
Date and Time	Date and time (with timestamp)	`timestamp`	`datetime2`
Date Only	Calendar date only	`date`	`date`
Time Only	Time without date	`time`	`time`

Saving Tables in Warehouse





Important

Select * from Table (X)
↓
Save as Demo
↓
Takes more Time

Select * from Table 1 into Table
↓
Faster and efficient

How to Choose One ?

Feature	Dataflow Gen2	Notebook	T-SQL (Warehouse)
Skill Set	Power Query (M), SQL	Spark (Python, SQL, Scala, R)	SQL
Use Case	Ingestion, transformation, profiling	Advanced transformation & processing	Reporting, transformation, analytics
User Persona	Data engineers, analysts	Data scientists, engineers	Data warehouse developers, DBAs
Interface	No-code/low-code (graphical)	Code-based (Notebook)	SQL scripts
Transformation Power	300+ built-in functions	Spark + open-source libraries	SQL-based for structured data
Connectivity	150+ connectors	100s of Spark libraries	DW, Power BI, Pipelines

Slowly Changing Dimensions

When over the months names of Products slowly changes.

Month 1 → P1

Month 2 → P1a

Month 3 → P1b

Type 0 = No Updates

Type 1 = All Updates only

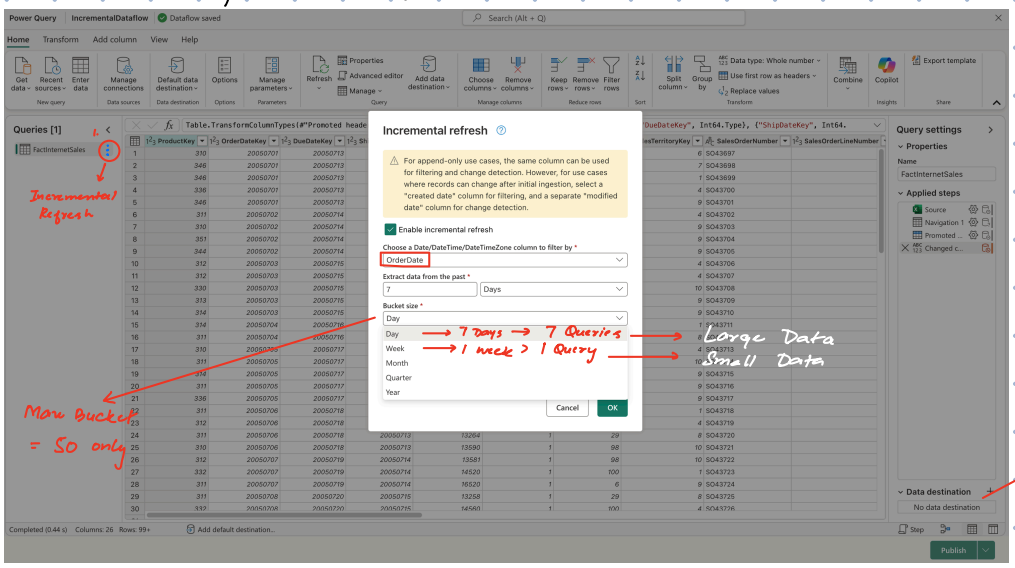
Type 2 = Allows current & Historical Data.

SCD Type	Description	Example
Type 0	No changes are tracked; data remains static	Birthdate, National ID
Type 1	Overwrites old data with new data (no history maintained)	Correcting a misspelled name
Type 2	Keeps full history by creating a new record with a new surrogate key	Changing address, new row for each change with effective date/version

INCREMENTAL DATA LOAD;

Loading only the changes.

- Using Dataflow Gen2 Not Pipeline



Manage & Optimize Data Warehouse;

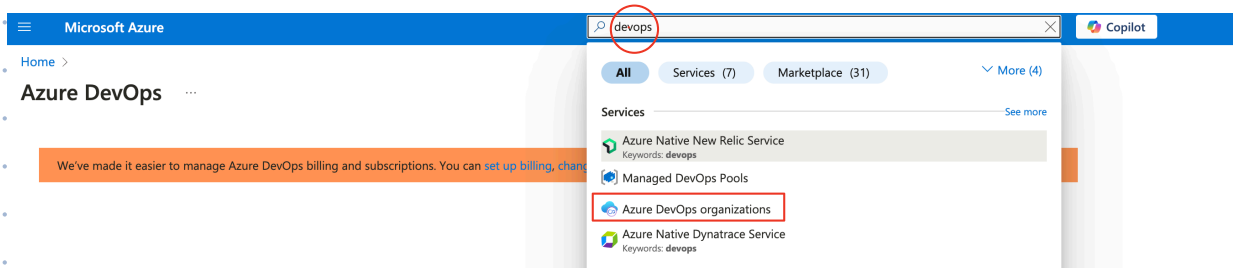
Git Integration: connecting fabric workspaces with a Git repository (Github or Azure Devops Git) to enable version control, collaborative development and tracking changes for

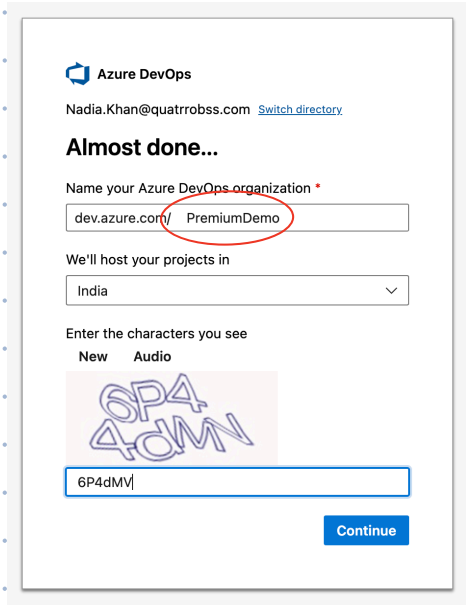
- Notebooks
- Dataflows
- pipelines
- Semantic models
- Reports & Dashboards.

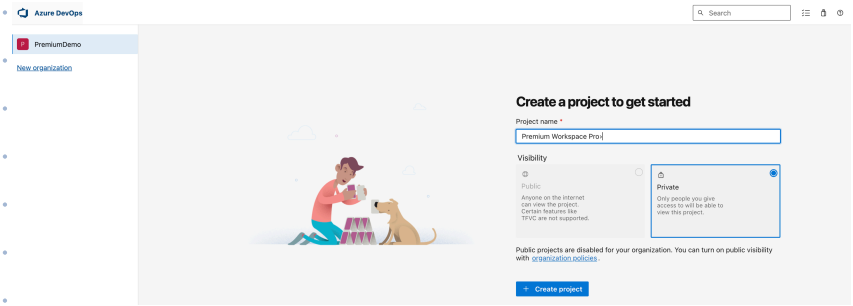
Typical Workflow:

1. **Connect your Fabric workspace to a Git repo** (Azure DevOps or GitHub).
2. **Select a branch** (usually `main` or `dev`).
3. Work on items like notebooks, reports, and pipelines.
4. **Commit changes** to the Git branch directly from Fabric.
5. Use **pull requests** for code review and collaboration.
6. Merge changes into `main` when ready for production.

Create Azure Devops Account;







Azure Devops Git is mostly preffered because it has tight integration with Azure Active Directory / Access control

Github; Open Source / personal

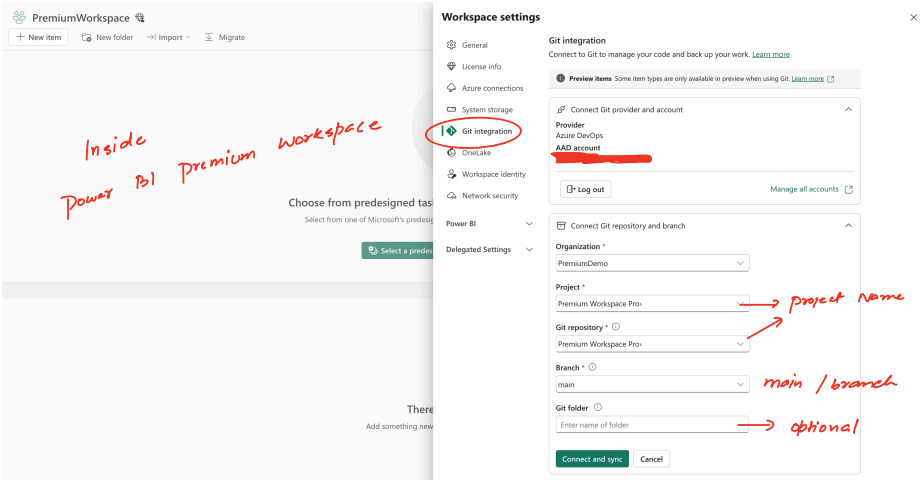
🧠 Final Take:

- Use **Azure DevOps Git** when you need a **secure, integrated enterprise DevOps platform**, especially with **Azure**.
- Use **GitHub** when you're focusing on **open-source**, community, or need **collaborative CI/CD** with GitHub Actions.

Creating Repos ; Initialize first Repository

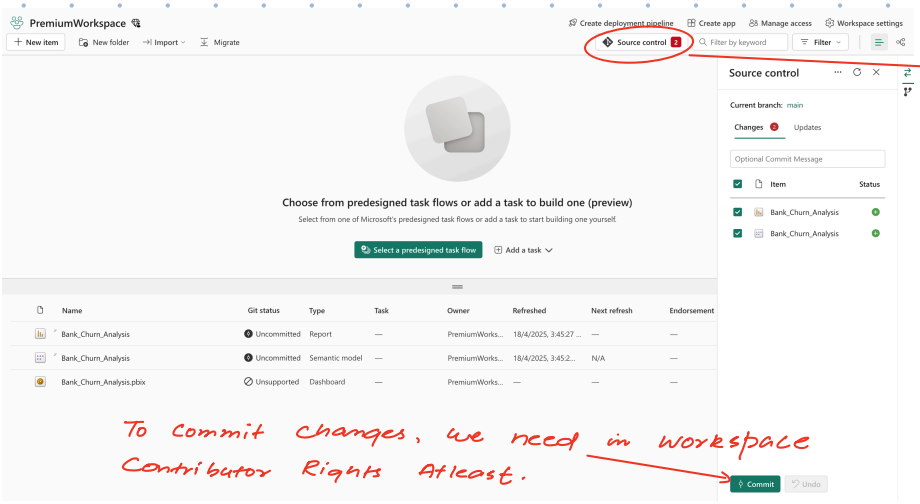
Important = Development / Testing in Branch Repo
Production in Main Repo

Git Integration



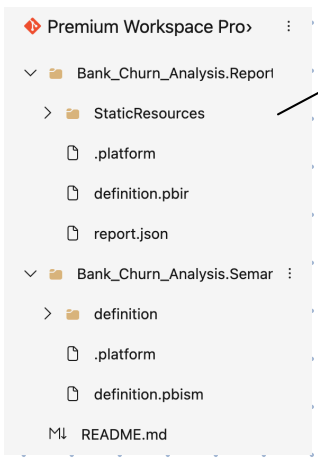
Important ; Workspace can be connected to one branch and to one folder at once

Commit in Workspace



Once version control is setup, any files added will show here as they need to be commited to show in Azure Devops Repo

Note ; Dashboards are not supported for Git Integration

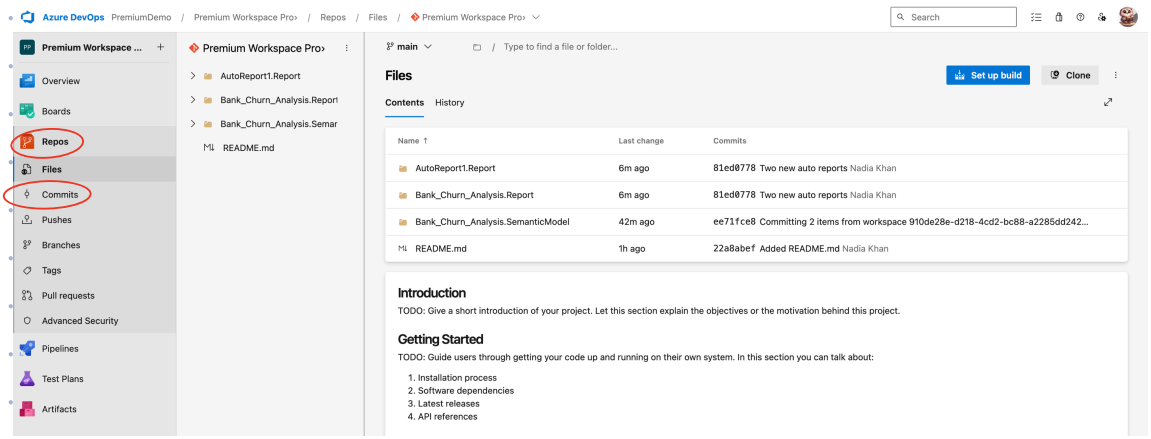


NOTE ; In Azure Devops Git, only the schema will be copied over not the Data ; which ensures Data Security

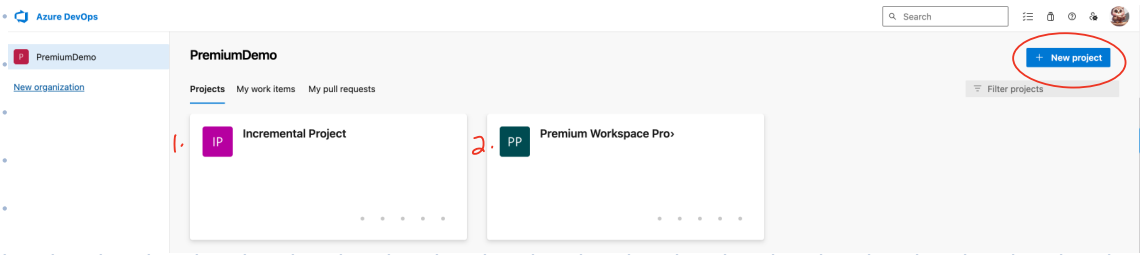
- **Changing Branch** → Needs Admin Rights
- Disconnect → Does not require

Create Quick Report; click on semantic model and Creak Quick Report.

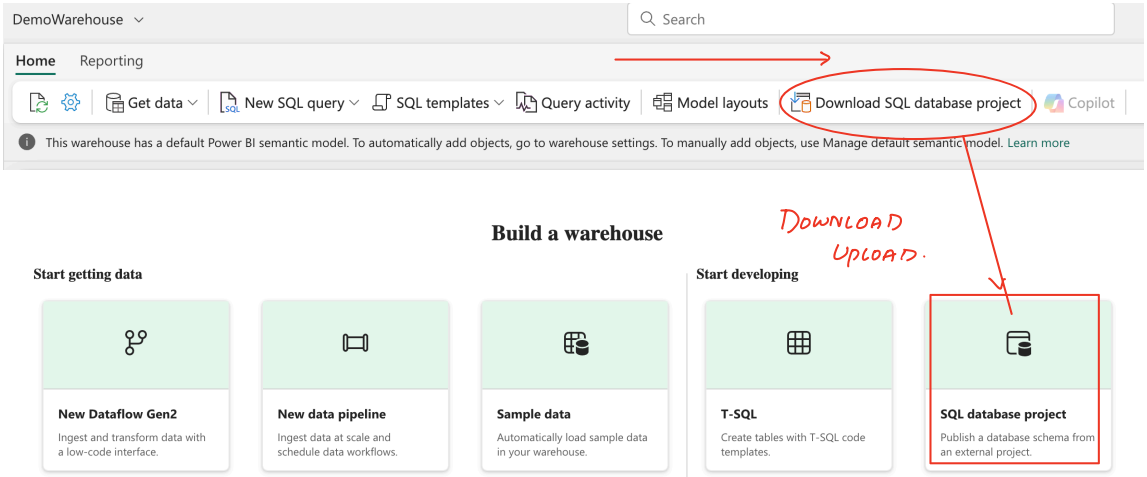
Azure Devops Git;



Add more projects;

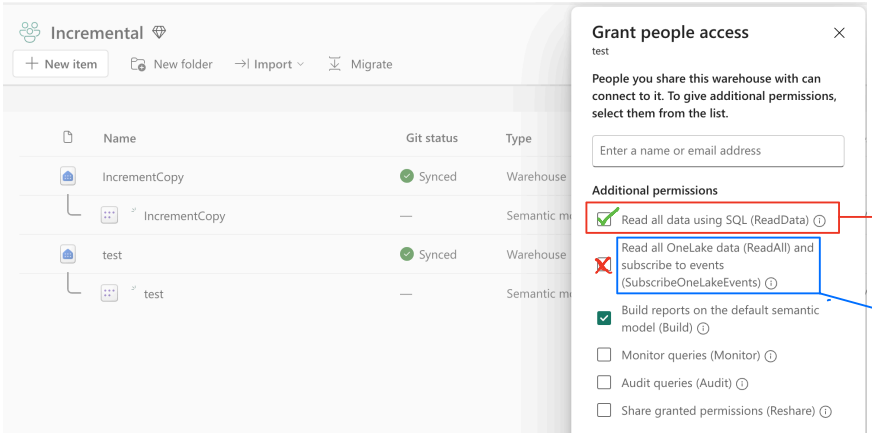


Implement DB projects; Copy schema from one DW → other



Important; Database project includes The Schema

Dynamic Data Masking in DW; Hiding Data is applicable for viewer mode



→ While sharing Masked Data, only use this one

→ Can be by-passed, so avoid using for Masked Data.

Dynamic Data Masking Code;

```
CREATE TABLE DynamicDataMasking
(ID int                 MASKED WITH (FUNCTION = 'random(1,10)'),
EmailAddress varchar(50) MASKED WITH (FUNCTION = 'email()'),
FamilyName varchar(30)   MASKED WITH (FUNCTION = 'partial(1, "XXXXXX", 2)'),
Department varchar(30)   MASKED WITH (FUNCTION = 'default()'),
DateOfBirth date         MASKED WITH (FUNCTION = 'default()'))
```

→ 11-20 ⟶ 1-10

→ khan@gmail.in ⟶ kxxx@xxx.com

→ TAJAMULKHAN ⟶ TXXXXXXXXAN

→ XXXX

→ Default → 1900/01/01

Modify Data Masking

```
GRANT SELECT ON DynamicDataMasking TO [jane@Filecats.onmicrosoft.com]

ALTER TABLE DynamicDataMasking
ALTER COLUMN ID DROP MASKED;
```

→ Grant permission

→ Drop Masking

```
ALTER TABLE DynamicDataMasking
ALTER COLUMN ID ADD MASKED WITH (FUNCTION = 'random(21, 30)')
```

→

Note; Dynamic Masking is not enough because people can still query (Select * from table where ID = 1) and it will return on ID = 1

So, we should use Dynamic Masking in conjunction with OLS and RLS.

Optimize Data Warehouse;

- Use Copy INTO then Insert (one by one)
- Star instead of Snowflake
- Reduce No. of Columns / Rows
- Use tiny/small int / varchar
- Store Data as Numbers instead of char
- Use Direct Lake Mode when possible

Note;
SQL Server shows maximum of 10k Rows, for more use SSMS

KUSTO SQL

is read only and used for Large/Big data exploration, named after deep diver "Jack Kusto"

Kusto vs Traditional SQL

iot, timeseries dat

Feature	Kusto (KQL)	Traditional SQL
Primary use case	Log & telemetry analysis	Relational data (OLTP/OLAP)
Write operations	❌ (Read-only)	✅ (SELECT, INSERT, UPDATE, DELETE)
Syntax style	SQL-like but with custom functions	ANSI SQL
Performance focus	High-speed analytics on time-series data	Transactional + Analytical
Supported in	Azure Monitor, ADX, Sentinel, etc.	SQL Server, PostgreSQL, MySQL, etc.

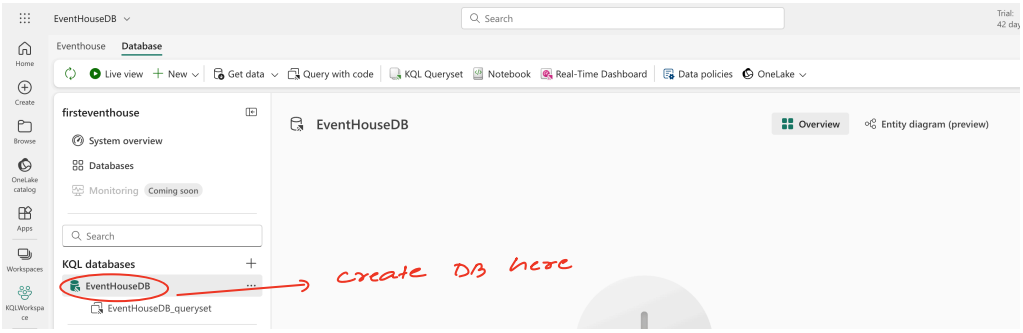
CREATING AN EVENTHOUSE ; using KQL.

Eventhouse = Fast Log Insights.

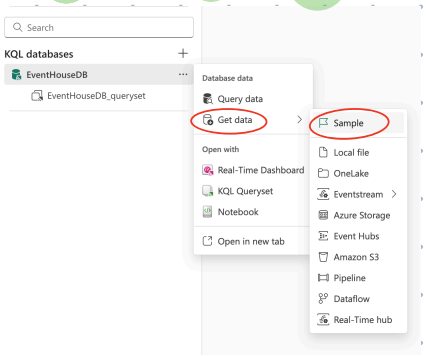
Lakehouse = Deep Data Analysis

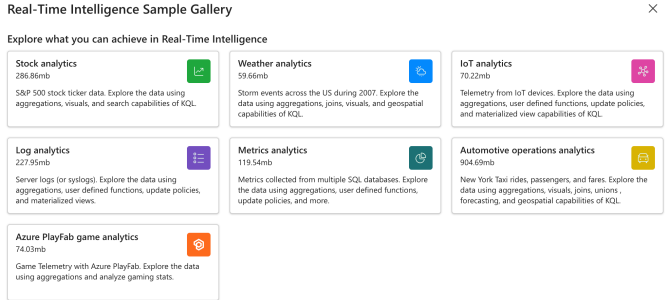
Feature	Eventhouse	Lakehouse
Use Case	Real-time log & event analytics	BI, reporting, ML on big data
Speed	Real-time (ms to seconds)	Near real-time or batch (minutes+)
Data Type	Logs, metrics, telemetry	Structured + unstructured data
Query Lang	KQL (Kusto Query Language)	SQL, Delta SQL, PySpark
Example Tools	Azure Data Explorer, Log Analytics	Databricks, Snowflake, Synapse

CREATE EVENTHOUSE DB



POPULATE WITH DATA





Note; KQL is mostly designed for Eventhouse, we can also use SQL but KQL is more preffered,

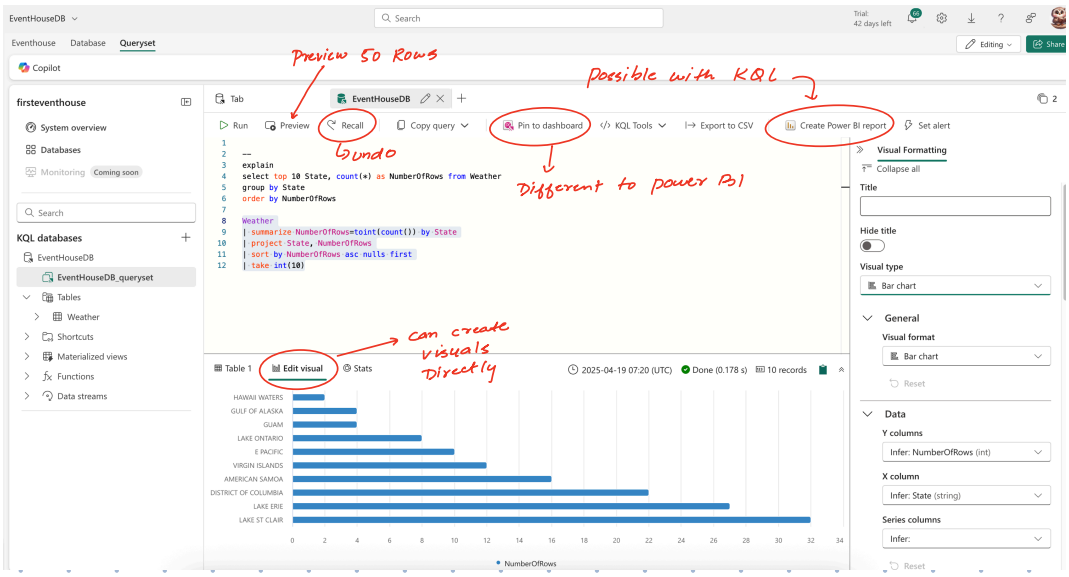
- Power BI Reports; KQL supports (✓) (Create) SQL doesn't (✗)
- KQL is way faster than SQL

```
--
explain
select top 10 State, count(*) as NumberOfRows from Weather
group by State
order by NumberOfRows
```

KQL SQL = More Time

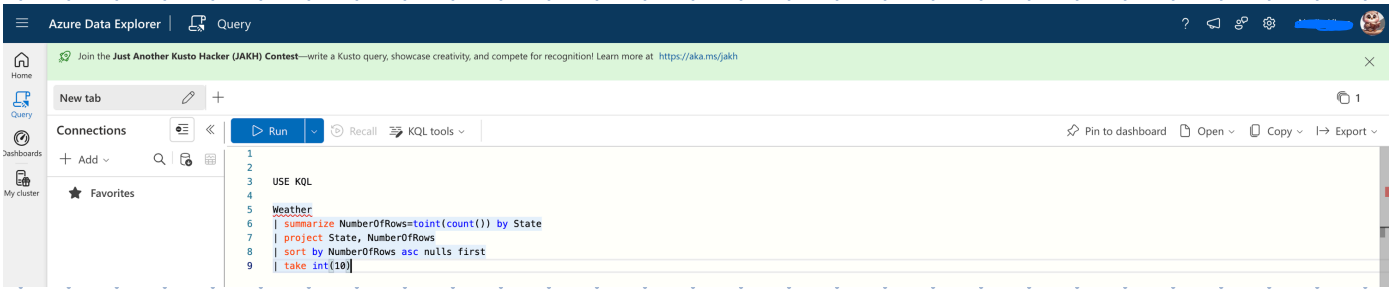
KQL = Less Time

```
Weather
| summarize NumberOfRows=toint(count()) by State
| project State, NumberOfRows
| sort by NumberOfRows asc nulls first
| take int(10)
```



ADX; AZURE DATA EXPLORER is a fast analytics service for large volumes of Data. It uses KQL. Quick insights from massive Data Streams.

- used for monitoring apps, analyzing IoT data etc.



NOTE; we can use KQL inside eventhouse & ADX.

KQL: Start Basic KQL Queries.

Note: In KQL, Order by is Desc by Default.

- Extend is used to add calculated columns

```
Personnel
| project FamilyName = LastName, GivenName
```

Correct

That's right. There would be 2 columns: FamilyName and GivenName.

Explain with AI

New — Old Name

```
Personnel | top 10
```

Correct

That's right. I could use "| top 10 by LastName", which would then order by LastName descending, and then take the first 10 rows. However, using "| top 10" by itself would result in an error.

Explain with AI

SELECT DATA IN KQL

```
// Selecting Columns in KQL
// Use // for comments
// Use | to separate operators in the pipeline

Weather // Table name
| project State, EpisodeId, StartDate=StartTime, EndTime, EpisodeNarrative, EventNarrative // Select columns with optional aliasing (StartDate = StartTime)
| extend Duration = EndTime - StartDate // Add calculated column
| extend EpisodeNarrative = "hi" // Overwrite values in the column (not renaming rows)
| order by State asc nulls last // Order results; asc by default, 'nulls last' is valid
| project-rename Narrative = EpisodeNarrative // Rename column: new name = old name
| project-away EventNarrative // Remove column
```

```
// Limit Number of Rows

Weather // Table name
| distinct State
| top 10 by State asc nulls last

Weather
| sample 10
| limit 5
```

CREATE STRING LITERALS

```
// Creating string literals

print passcode = 1 // Number
print 'He is know by the name "Roy"'

// Problem with Backslash use Double Backslash
print "He is a good Husband\\Son "

// Multi line Code - Use Apstrophe
print ```He is very well know \
for his character,\
He is very Kind```
```

Multi Line String

Filtering with String

where

Note; when renaming, use only one = sign
when comparing, use == sign

```
Weather
| project Narrative=EpisodeNarrative, EventType
| where EventType == "Heavy Rain"  // case sensitive

Weather
| project Narrative=EpisodeNarrative, EventType
| where EventType =~ "heavy rain" // case insensitive

Weather
| project Narrative=EpisodeNarrative, EventType
| where EventType != "Heavy Rain"  // case sensitive
```

```
Weather
| project Narrative=EpisodeNarrative, EventType
| where EventType !~ "heavy rain" // case insensitive
```

```
Weather
| project Narrative=EpisodeNarrative, EventType
| where not (EventType == "Heavy Rain")  // case sensitive
```

```
Weather
| project Narrative=EpisodeNarrative, EventType
| where EventType == "Heavy Rain" or EventType == "Blizzard"  // case sensitive

Weather
| project Narrative=EpisodeNarrative, EventType, State
| where State == "TEXAS" and (EventType == "Heavy Rain" or EventType == "Blizzard")
// case sensitive

Weather
| project Narrative=EpisodeNarrative, EventType
| where EventType in ("Heavy Rain", "Blizzard")  // case sensitive
```

```
Weather
| project Narrative=EpisodeNarrative, EventType
| where not(EventType =~ "heavy rain" or EventType =~ "blizzard")  // case insensitive

Weather
| project Narrative=EpisodeNarrative, EventType
| where EventType !in~ ("heavy rain", "blizzard")  // case insensitive
```

Filtering for part of a String

HAS	CONTAINS
Full word	only letters

```
Weather
| project EpisodeNarrative
| where EpisodeNarrative has "county"  // case insensitive

Weather
| project EpisodeNarrative
| where EpisodeNarrative contains "county"  // case insensitive

Weather
| project EpisodeNarrative
| where EpisodeNarrative has_cs "county"  // case sensitive
```

```
Weather
| project EpisodeNarrative
| where EpisodeNarrative has "cou"

Weather
| project EpisodeNarrative
| where EpisodeNarrative contains "cou"
```

```
Weather
| project EpisodeNarrative
| where EpisodeNarrative hasprefix "cou"
| where not (EpisodeNarrative hasprefix "count")
| where EpisodeNarrative !hasprefix "couple"
```

```
Weather
| project EpisodeNarrative
| where EpisodeNarrative startswith "showers"
```

Advanced

```
Weather
| project EpisodeNarrative
| where EpisodeNarrative has "showers" or EpisodeNarrative has "county"

Weather
| project EpisodeNarrative
| where EpisodeNarrative has "showers" and EpisodeNarrative has "county"

Weather
| project EpisodeNarrative
| where EpisodeNarrative has_any ("showers", "county")

Weather
| project EpisodeNarrative
| where EpisodeNarrative has_all ("showers", "county")
```

DATA AGGREGATION

```
Weather
| summarize Calc = count() by State, EventType
```

Think of this as Group by

No having

```
Weather
| summarize Calc = countif(EventType == "Flood") by State, EventType
```

Can use multiple where

```
Weather
| summarize Calc = countif(EventType != "Flood") by State, EventType

Weather
| where State in~ ("Texas", "Kansas", "Alaska")
| summarize Calc = countif(EventType =~ "flood") by State, EventType
| where Calc >= 100
```

ADVANCE

. dcount.

```
Weather
| summarize Calc = sum(InjuriesDirect) by State

Weather
| summarize Calc = sumif(InjuriesDirect, EventType == "Flood") by State

Weather
| summarize Calc = avg(InjuriesDirect) by State

Weather
| summarize Calc = max(InjuriesDirect) by State

Weather
| summarize Calc = min(InjuriesDirect) by State

Weather
| summarize Calc = maxif(InjuriesDirect, EventType == "Flood") by State

Weather
| summarize Calc = dcount(EventType) by State
```

→ Distinct count

KQL FUNCTIONS

Empty strings (note: also can use isnotempty, isnull and isnotnull)

```
Weather
| project State, EpisodeNarrative, EventType, BeginLocation
| where isempty(BeginLocation) // or == ""
| limit 20
```

Combining strings together and trimming the result

```
Weather
| project State, EpisodeNarrative, EventType, BeginLocation
| extend Calc = strcat(State, ": ", EventType, ": ", EpisodeNarrative)
| limit 20
```

→ Simple concat

```
Weather
| project State, EpisodeNarrative, EventType, BeginLocation
| extend Calc = strcat_delim(": ", State, EventType, EpisodeNarrative)
| limit 20
```

→ Concat with Delim (Both Strings + Number)

```
Weather
| project State, EpisodeNarrative, EventType, BeginLocation, EventId
| extend EventType = toupper(EventType) // and tolower
| extend Calc = strcat_delim(": ", EventId, State, EventType, EpisodeNarrative)
| limit 20
```

→ UPPER LOWER

```
Weather
| project State, BeginLocation, EndLocation
| extend Calc = trim(' ', strcat_delim(" ", BeginLocation, State, EndLocation))
| limit 20
```

→ Trim (Remove, where)

→ we can also use trim_start or end.

MANIPULATING STRING;

```
Weather
| project EventType
| extend FindSpace = indexof(EventType, " ") // gives -1 if there is not a space
| limit 20
```

```
| extend FindSpace = indexof(strcat(EventType, " "), " ")
| extend BeforeSpace = substring(EventType, 0, FindSpace)
| project FindSpace, BeforeSpace, State, EpisodeNarrative, EventType, BeginLocation
```

Substring is used to extract part of string

Substring Syntax (Column, start index, No. of characters)

Strlen = Length of String

REPLACE STRING

```
| extend EventType = replace_string(EventType, "heavy", "huge") // does not change “Heavy”.
| extend EventType = replace_string(EventType, "Heavy", "Huge")
```

IMPORTANT

`| extend DepLoc = strcat(Department, " ", Location)` Correct

That's correct. The strcat function concatenates strings together.

Explain with AI

`print indexof("Family Name", " ")` **will return the answer 6.** Correct

That's correct. As KQL is a zero-based language, the 1st character is character number zero, and the 7th is character number 6.

Explain with AI

Important ; KQL follows BODMAS RULE

NUMBER DATA TYPES ;

- Real , Decimal = has fractions
- int , Long = do not have fractions

Example

print 1/2 = 0 ⟶ floor Division

1.0/2 or 1/2.0 = 0.5

Data Type	Example	Description
`int`	`8`	Signed 32-bit integer (e.g., whole numbers)
`long` (big int)	`1234567890123` (Default)	Signed 64-bit integer for larger whole numbers
`real`	`10.5` (Default)	64-bit double-precision floating-point number
`decimal` (exact division)	`decimal(11.99)` (slower compared to Real)	Signed 128-bit high-precision decimal number
`string`	`"hello"`	Sequence of characters (text)
`bool`	`true` / `false`	Boolean value (true or false)
`datetime`	`datetime(2024-01-01)`	Date and time value
`timespan`	`1d`, `5h`, `30m`	Time difference (duration)

whole: int, long — floats: real, decimal

1/2 = 0

1/2 = 0

1/2 = 0.50000

1/2 = 0.5

Math Functions

```
| extend Calc = round(EpisodeId - EventId/ real(5), 0) // and ,1) for 1 decimal place
| extend Calc = ceiling(EpisodeId - EventId/ real(5), 0) rounds up
```

Other math functions

```
| extend Calc2 = abs(Calc)

| extend Calc3 = sign(Calc)

| extend Calc = EpisodeId % (EventId/10) // delete Calc2 and Calc3.

| extend Calc = pow(EpisodeId, 2)

| extend Calc = sqrt(EpisodeId)

| extend Calc = rand()
| extend Calc = rand(4)
// int     - signed 32-bit integer
// long    - signed 64-bit integer
// real    - signed 64-bit double-precision, floating-point number
// decimal - signed 128-bit decimal number.
```

Date & Timestamp Datatypes

```
Weather
| distinct StartTime, EndTime
| extend Duration = EndTime - StartTime, RevisedEndTime = EndTime + 1d
// d h m s ms microsecond tick

print datetime(2030-02-03 01:23:45.6);
print todatetime("2030-02-03 01:23:45.6");
print make_datetime(2030, 2, 3, 1, 23, 45.6);

print timespan(1d);
print timespan(3);
print timespan(40 seconds);
print timespan(1.23:45:17.8);
print timespan(1.23:45:17.8) - 3h;

print timespan(1.23:45:17.8) * 3;

print timespan(1.23:45:17.8) / 3h;

print now() + 3h;
print now(3h)
print now(-3h)
print ago(3h)
```

Imp.
now ( )
ago (3h)

Date & Timestamp Functions;

```
| where StartTime between (datetime(2007-03-26) .. datetime(2007-04-15))

| extend Calc = startofday(StartTime) // and endofday

Weather
| distinct StartDate = startofday(StartTime)
| extend Calc = startofweek(StartDate) // start of the week = Sunday.
| where StartDate between (datetime(2007-03-26) .. datetime(2007-04-15))
// also startofmonth, startofyear, endofweek, endofmonth, endofyear

| extend Calc = dayofweek(StartDate)
// and also hourofday, dayofmonth, dayofyear, weekofyear, monthofyear, getyear

| extend Calc = datetime_part("week_of_year", StartDate)
// start of the week = Sunday. First week includes the first Thursday.
// you can extract Year, Quarter, Month, Day, DayOfYear, Hour, Minute, Second, Millisecond, Microsecond and Nanosecond
```

Important

Date manipulation

```
Weather
| distinct StartTime
| where StartTime between (datetime(2007-03-26) .. datetime(2007-04-15))

| extend Calc = StartTime+2d
, Calc2= datetime_add('day', 2, StartTime)
// The period can be Year, Quarter, Month, Week, Day, Hour, Minute, Second, Millisecond, Microsecond and Nanosecond

| extend Calc = EndTime-StartTime
, Calc2= datetime_diff('day', EndTime, StartTime)
// datetime_diff only uses the relevant period.
// 26th 00:00 to 26th 23:00 is 0 days
```

Important

Format Specifier	Data Type	Description
`yyyy`	datetime	4-digit year (e.g., 2025)
`MM`	datetime	Month with leading zero (01–12)
`dd`	datetime	Day of the month with leading zero (01–31)
`HH`	Both	Hour in 24-hour format (00–23)
`mm`	Both	Minutes with leading zero (00–59)
`ss`	Both	Seconds with leading zero (00–59)
`f` to `fffffff`	Both	Fractions of a second (up to 7 digits)
`tt`	datetime	AM/PM designator
`ddddddd`	timespan	Total days in duration (with padding if needed)

Converting dates and timestamps

```
Weather
| distinct StartTime, EndTime
| extend Calc = format_datetime(StartTime, 'd MM yyyy')
,      Calc2 = format_timespan(EndTime - StartTime, 'ddd.hh:mm:ss')
```

```
1 | extend EndTime = datetime_add('day', 1, StartTime)
```

Correct

That's right. 'day' needs to be quotation or speech marks, and then the number of days and the datetime.

Explain with AI

UNION THE DATA ;

One Query & second should be brackets

```
Weather
| distinct State, EventType
| union kind = outer withsource = TableSource
(Weather
| distinct State, EventNarrative)

union withsource = AdditionalColumn Wea*
```

Default union = outer

Inner = Only Columns which are common in both tables

JOIN IN KUSTO SQL

Default join = innerunique

```
Weather
| join
(Region | extend State = toupper(State))
on State

Weather
| join kind = fullouter
(Region | extend State = toupper(State))
on ($left.State == $right.State)
```

→ Can use on State and on

Remember syntax =

Imp ; Keep small table on left side for efficient join

Join Type	Description
`inner` (default)	Returns only matching rows from both tables.
`leftouter`	Returns all rows from the left table, plus matching rows from the right.
`rightouter`	Returns all rows from the right table, plus matching rows from the left.
`fullouter`	Returns all rows from both tables, with nulls where there is no match.
`leftanti`	Returns rows from the left table **that have no match** in the right table.
`rightanti`	Returns rows from the right table **that have no match** in the left table.
`innerunique`	Like `inner`, but **each match in the right table is used at most once**.

IDENTIFY NULL & DUPLICATE VALUES

```
Weather
| join kind = fullouter
(Region | extend State = toupper(State))
on ($left.State == $right.State)
| project State, Region
| summarize Count = count() by State, Region
| where Count > 1

Weather
| join kind = fullouter
(Region | extend State = toupper(State))
on ($left.State == $right.State)
//| where State1 == "" or isnull(State1)
| where coalesce(State1, "") == ""

Weather
| join kind = rightanti
(Region | extend State = toupper(State))
on ($left.State == $right.State)
```

Mostly isnull(col) or col == "" ?

Conditional Statements ;

```
| summarize NumberOfEvents = count() by State
| extend TexasOrFlorida = iif(State == "TEXAS" or State == "FLORIDA", "Texas/Florida", "Other")

Weather
| summarize NumberOfEvents = count() by State
| extend TexasOrFlorida = iif(State in ("TEXAS", "FLORIDA"), "Texas/Florida", "Other")

Weather
| summarize NumberOfEvents = count() by State
| extend TexasOrFlorida = case(State == "TEXAS", "Texas/Florida", State == "FLORIDA", "Texas/Florida", "Other")

// the Else argument is required.
```

Note iif and iff are same

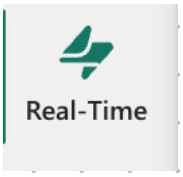
- case statement acts as Nested if

Ordered Flow of Data in Fabric

Step	Component	Purpose / Description
1	**Data Sources**	External data (APIs, databases, files, IoT, logs, etc.) collected from different systems.
2	**OneLake** (Data hub)	Centralized data lake storage for all data in Fabric (like a single data lake for all).
3	**Pipelines**	Used to orchestrate data movement and transformations (ETL/ELT).
4	**Dataflow Gen2**	No-code/low-code data prep tool for cleaning, shaping, and transforming data.
5	**Lakehouse**	Combines lake storage + structured access using Delta format (best for analytics).
6	**Eventhouse**	Specialized for ingesting and analyzing real-time, event-based data (e.g., logs/telemetry).
7	**Notebooks**	For data engineering, Python/Spark/SQL-based processing and advanced transformations.
8	**SQL Endpoints**	Query interface to access and analyze data from Lakehouse/Eventhouse.
9	**Warehouse**	Structured data warehouse optimized for BI and analytics (relational model).
10	**Power BI**	Data visualization and reporting layer to build dashboards and reports.

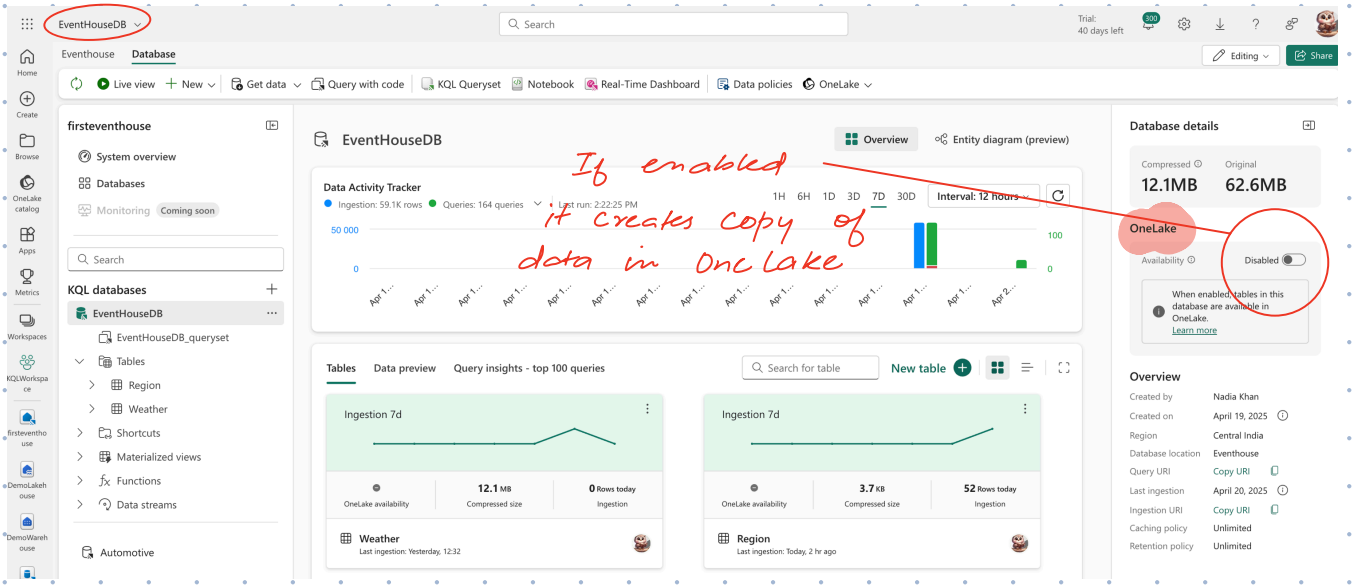
One Lake ; one source of all Data (Data hub) where we see everydata (Semantic models, eventhouse Lakehouse, DB, warehouses)

Real Time Hub Here we will see Tables from eventhouse DB's (Not DBs)

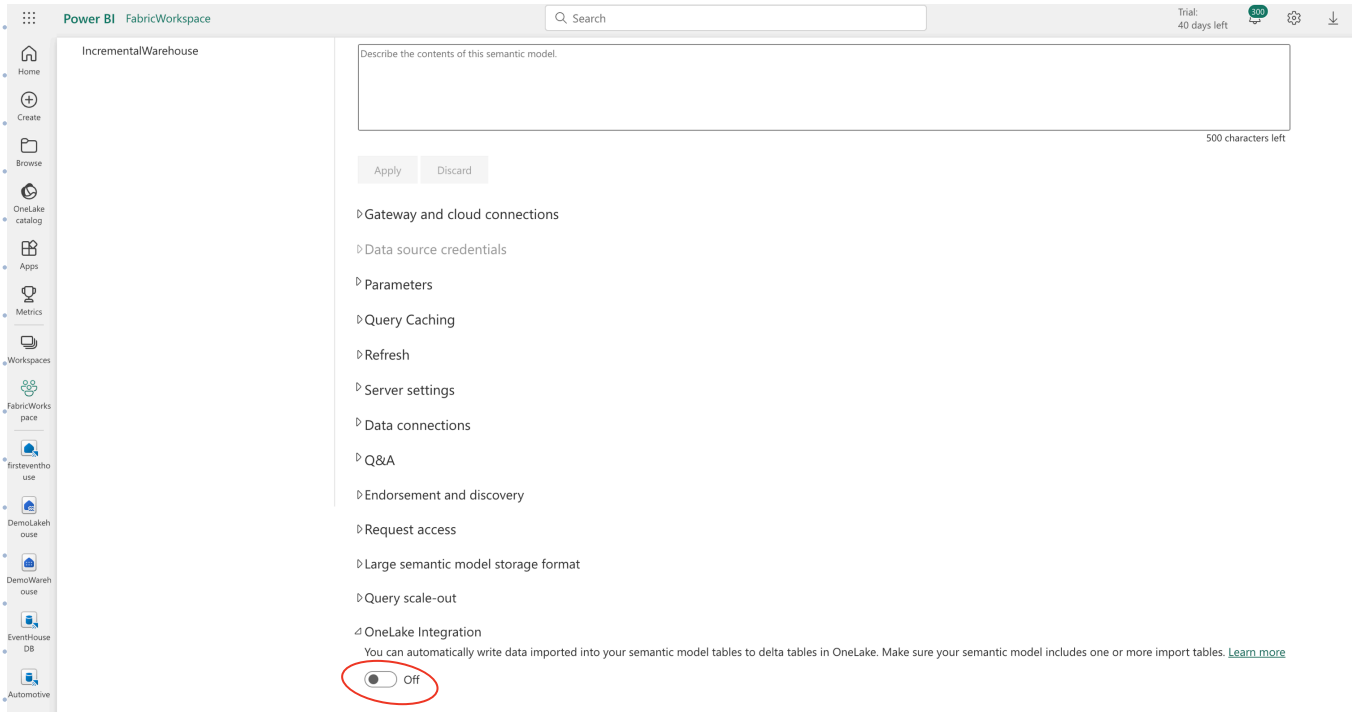


Note ; we can cacate copy of EH Tables in one lake by using option in EH

Avoid this during development as it won't let us delete the data.



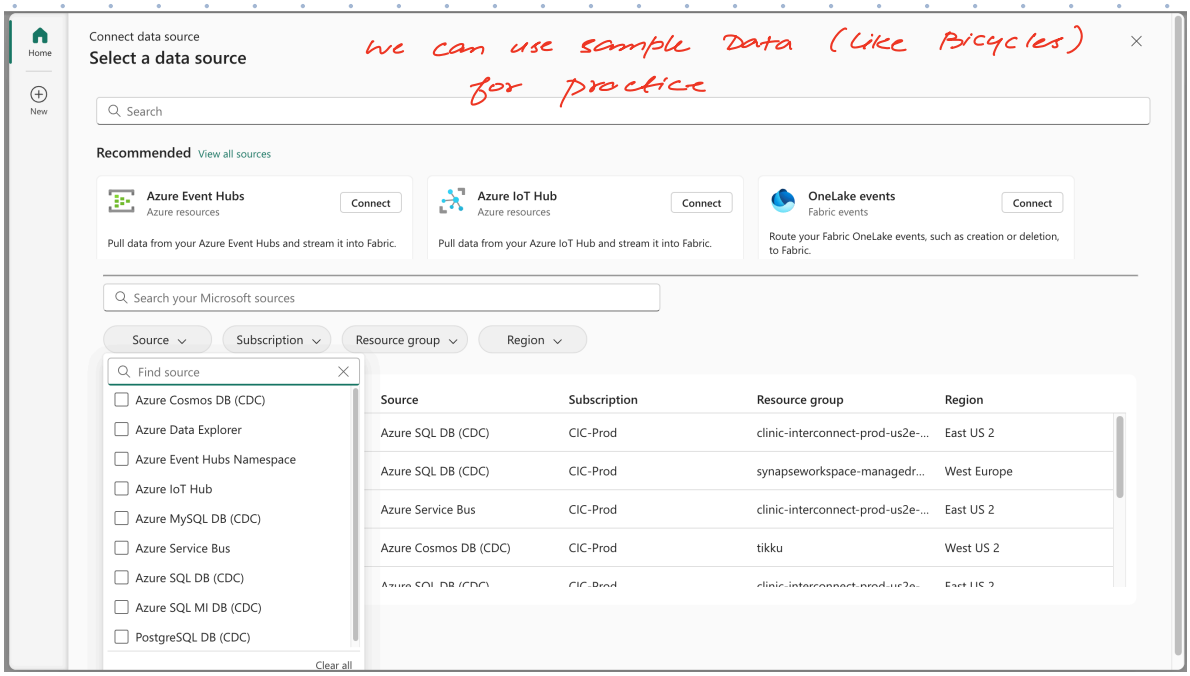
ENABLE ONE LAKE INTEGRATION FOR SEMANTIC MODELS



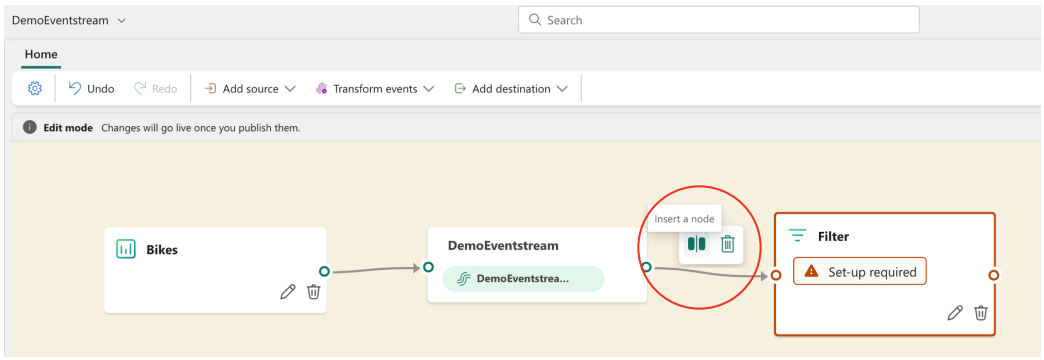
CHOOSE APPROPRIATE STREAMING

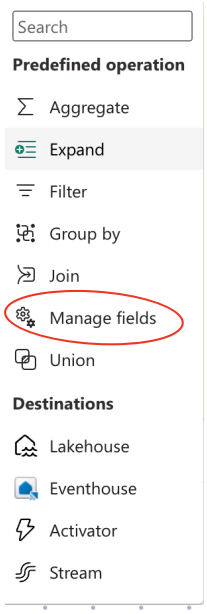
Aspect	Eventstream	KQL DB
Purpose	Ingest real-time data	Store & query streams
Processing	Capture & transform events	Complex queries, alerts
Integration	Works with Fabric sources/sinks	Tied to Eventstream & Fabric
Use Case	IoT, logs, apps	Monitoring, alerting
Query	Routing rules	KQL
Scale	Millions/sec, low latency	High concurrency

EVENTSTREAM DATA SOURCES;



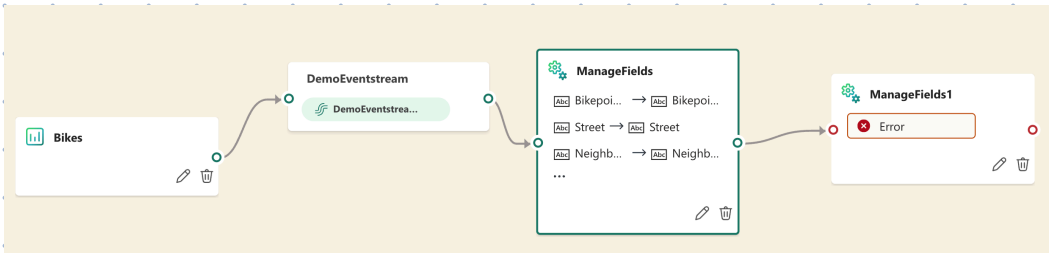
DATA PROCESSING BY EVENT STREAM

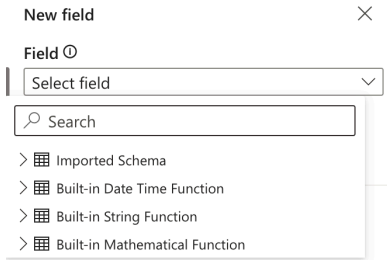




Used to Edit Fields

NOTE; Double Datatype is used to store Decimal values

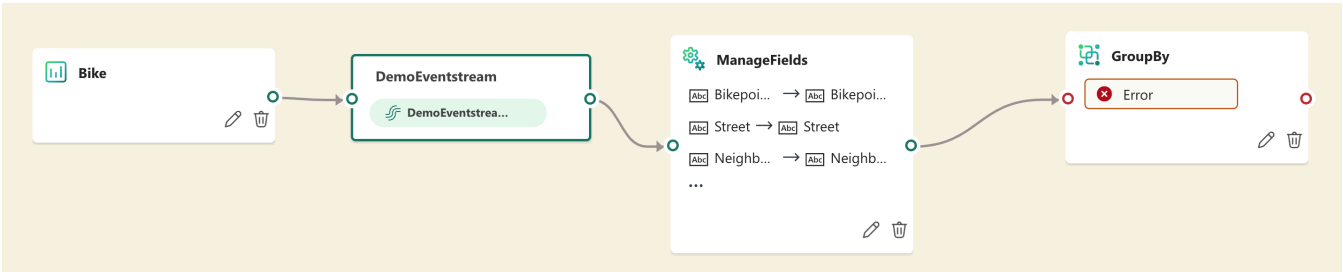


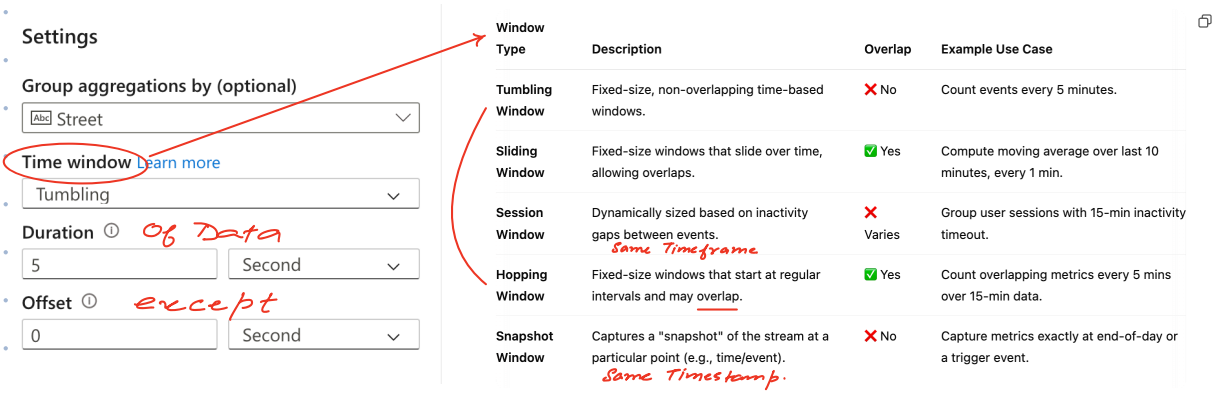


← Inbuilt functions

NOTE; Groupby is important

Group By Transform Event ;



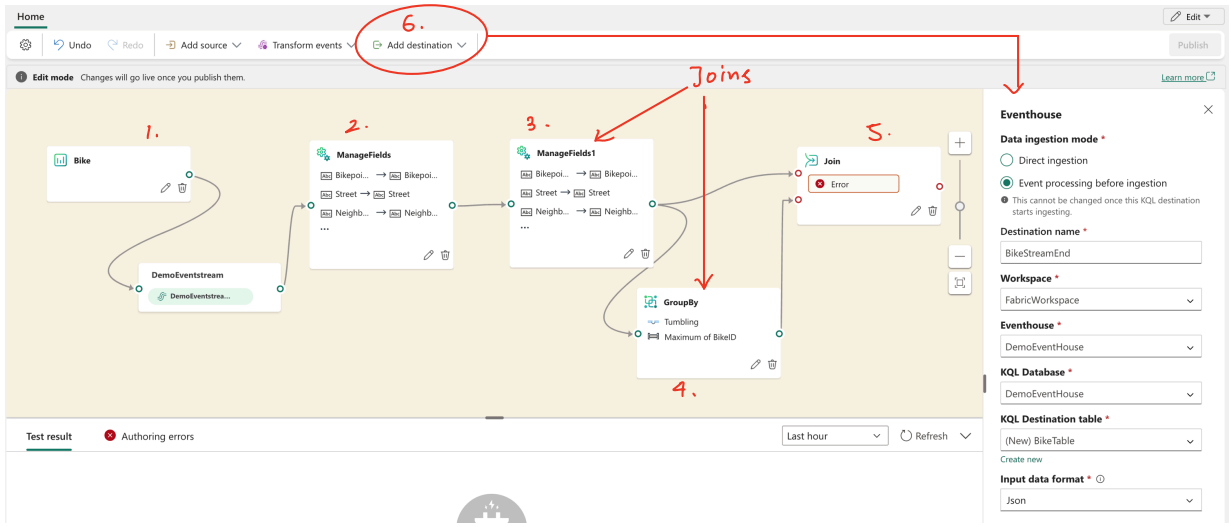


Window Type	Description	Overlap	Example Use Case
Tumbling Window	Fixed-size, non-overlapping time-based windows.	❌ No	Count events every 5 minutes.
Sliding Window	Fixed-size windows that slide over time, allowing overlaps.	✅ Yes	Compute moving average over last 10 minutes, every 1 min.
Session Window	Dynamically sized based on inactivity gaps between events.	❌ Varies	Group user sessions with 15-min inactivity timeout.
Hopping Window	Fixed-size windows that start at regular intervals and may overlap.	✅ Yes	Count overlapping metrics every 5 mins over 15-min data.
Snapshot Window	Captures a "snapshot" of the stream at a particular point (e.g., time/event).	❌ No	Capture metrics exactly at end-of-day or a trigger event.

Same Timeframe

Same Timestamp.

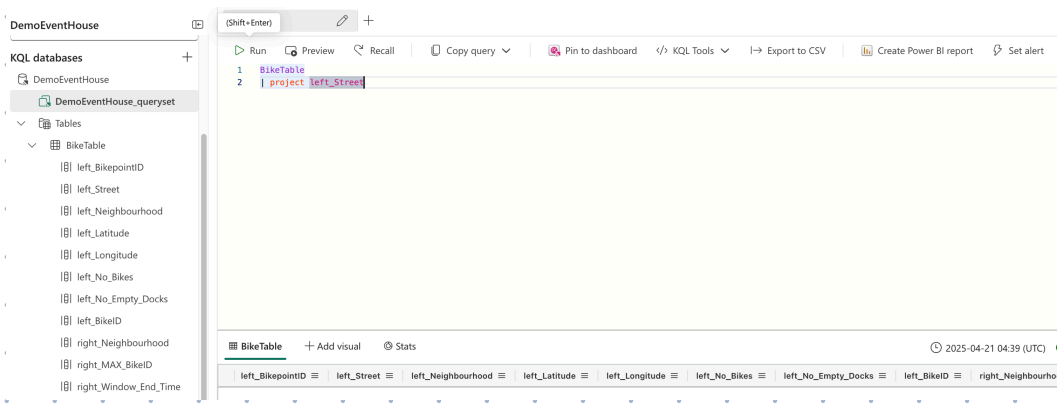
Completing Eventstream



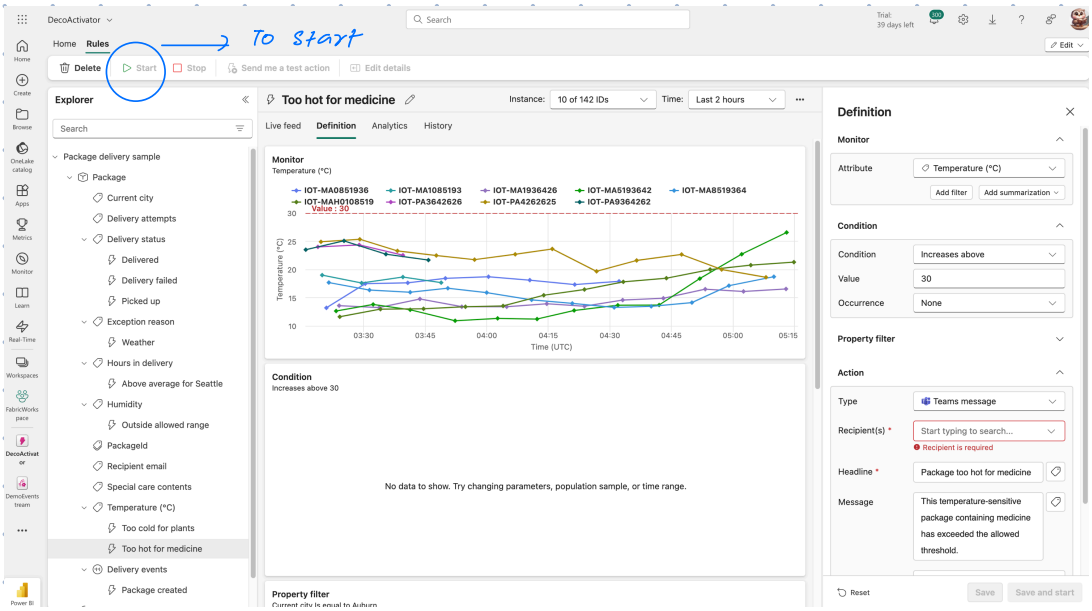
Note; once Destinations is added we can click on publish

we can Activate / Deactivate Stream.

Published Data in Eventhouse ;



ACTIVATOR Monitor Data & activate alerts & actions



ALERTS IN FABRIC

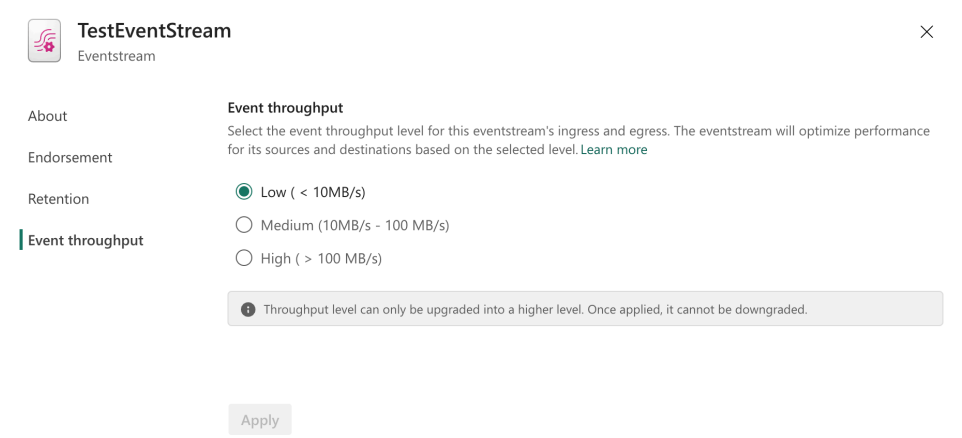
Component	Alert Capability	Trigger Type	Integration/Output
Power BI Dashboards	Tile-based alerts (KPI, card, gauge)	Value thresholds	Email, Power Automate
Eventhouse (KQL DB)	Alerts on query results using KQL & schedules	Query evaluation	Azure Monitor, Email, Logic Apps
Eventstream	Real-time conditions & routing logic	Incoming event conditions	Logic Apps, Eventhouse, Lakehouse, Functions
Data Activator	Real-time data condition monitoring	Any data source change	Email, Teams, Power Automate, Webhook

Also, we can set in KQL

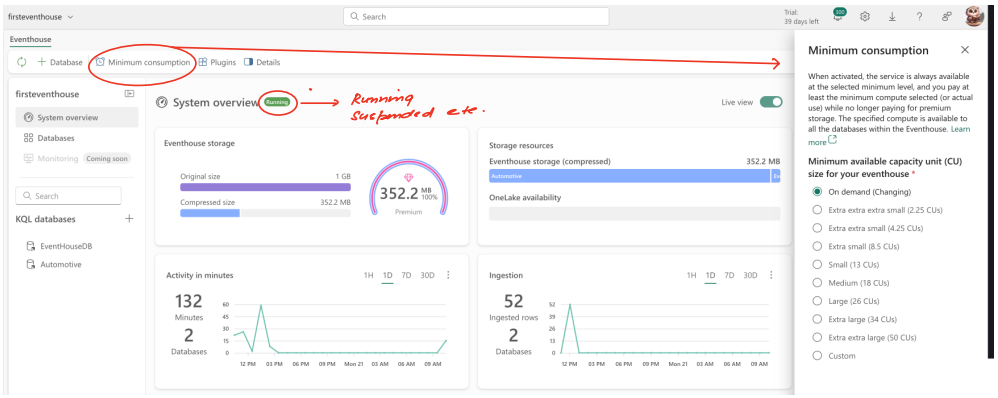
TRIGGER ON BLOB STORAGE

pipeline → Trigger → Blob Events

OPTIMIZE EVENTSTREAMS can change throughput



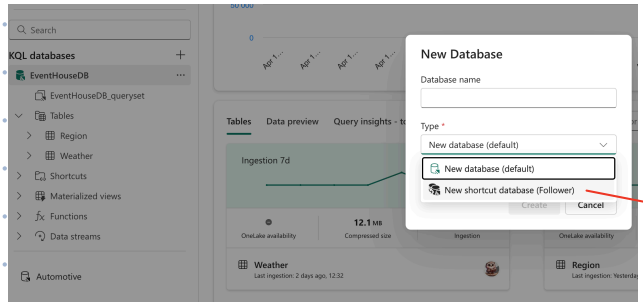
Optimize EVENTHOUSE; can change consumption

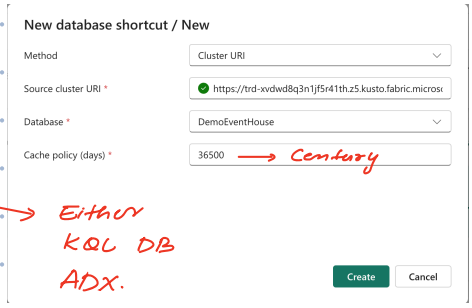


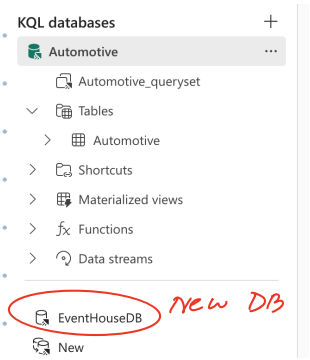
CHOOSE BETWEEN NATIVE, FOLLOWED STORAGE & SHORTCUT

For Real time Intelligence Data.

ADD OTHER INTERNAL DB (shortcut)





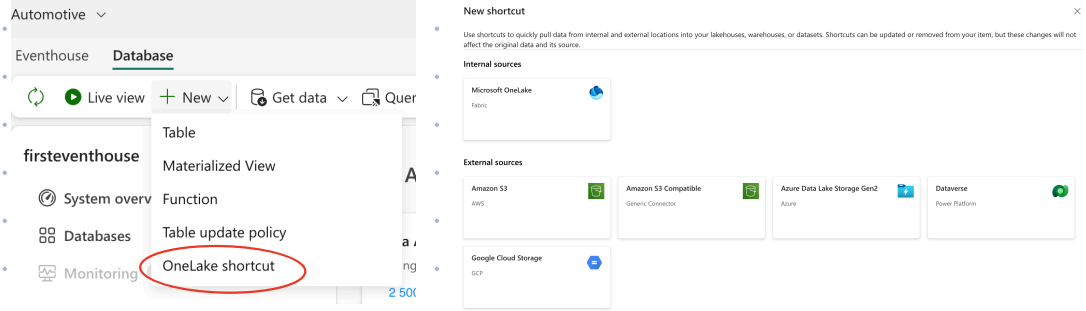


Query Both New/Old KQL DBs

```
Automotive
| distinct pickup_datetime, pickup_longitude

database('EventHouseDB').Table
| project columns
```

Important - New DB is also Read only
- Data is Referenced not copied

ADD EXTERNAL DB;



Important; External Tables will be added as Shortcut and can be KQL = external Table ("Dimproduct")

Question 1:

I want to create an Activator.

What data source can I NOT use in the Activator?

- [x] **Files being added into a SharePoint folder.** — Correct
  That's right. The data sources include various databases using CDC, Azure Blob Storage events and Fabric Events (including OneLake).
  Explain with AI

Question 3:

What is the default retention period for an eventstream?

- [ ] 1 hour
- [x] **1 day** — Correct
  That's right. You can set it anywhere between 1 and 90 days.
  Explain with AI

Question 4:

I want to adjust the estimated event throughput for an eventstream.

I click on the ... next to the eventstream in the Workspace, go to Settings, and then click on the "Event throughput" tab.

What are the options?

- [ ] Up to 10 Kb/s, 10-100 Kb/s, and over 100 Kb/s
- [ ] Up to 1 Mb/s, 1-10 Mb/s, and over 10 Mb/s
- [x] **Up to 10 Mb/s, 10-100 Mb/s, and over 100 Mb/s** — Correct
  Explain with AI

Question 2:

I am in a pipeline.

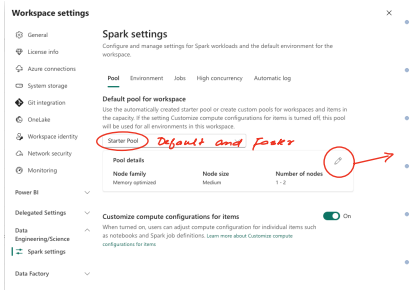
I click on Home - Add trigger.

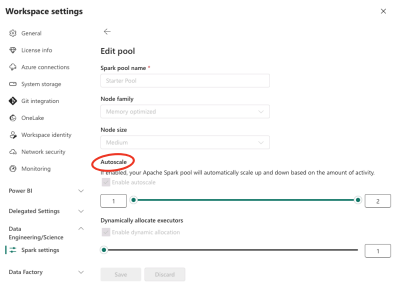
What data source do I use?

- [ ] An Azure SQL Database using Change Data Capture (CDC)
- [x] **An Azure Blob storage account** — Correct
  That's correct. Among other events, you can check whether a blob or directory has been created, deleted, renamed, or a file has been put into another tier.
  Explain with AI

WORKSPACE SETTINGS & MONITORING

Spark Workspace Settings

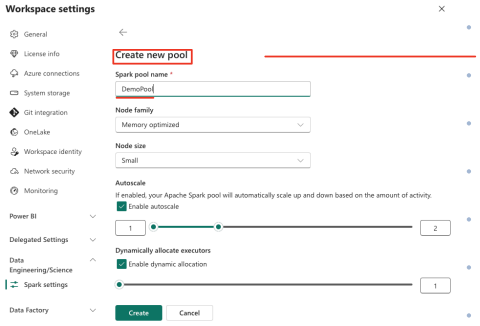


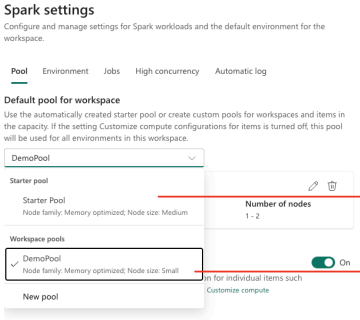


SKU Name	Default Max Nodes	Max Number of Nodes
F2	1	1
F4	1	1
F8	2	2
F16	3	4
F32	8	8
F64 or Trial	10	16
F128	10	32
F256	10	64
F512	10	128
F1024	10	200
F2048	10	200

Important: If I change settings for workspace, then these settings wont affect Active Sessions

Create New Pool





Note: Once pool is setup, we can start standard Session in PySpark

↓

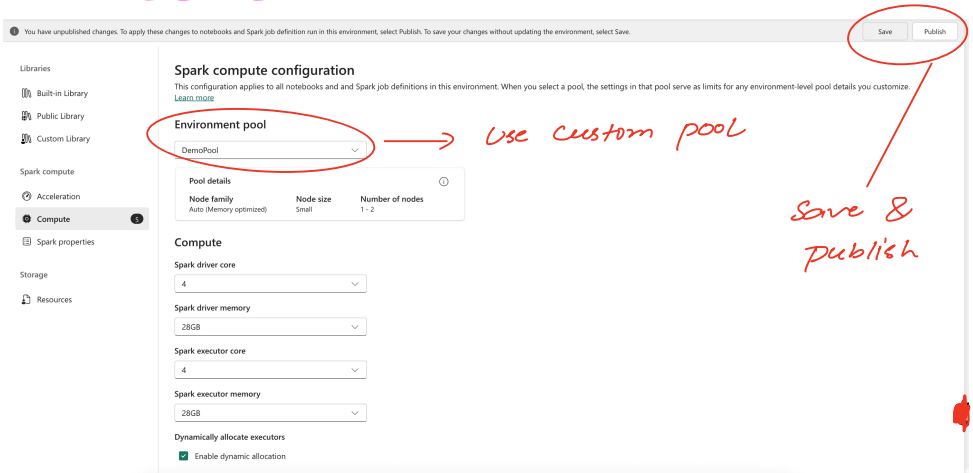
Usually takes 10 Sec

Usually takes 3 mins

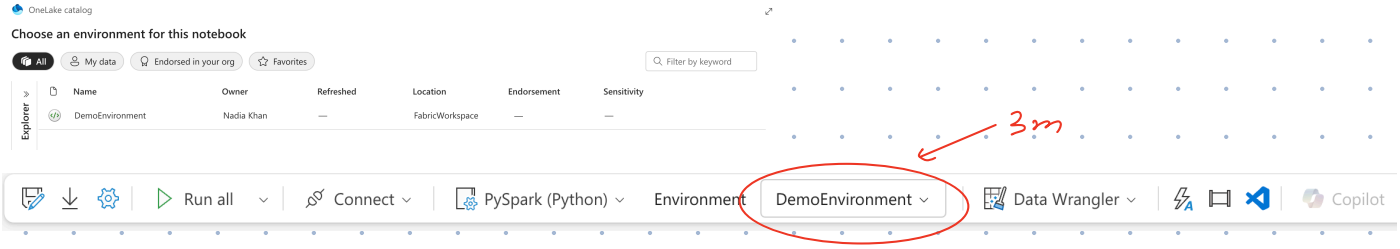
How to Allow people to change the pool;

we need to setup an environment

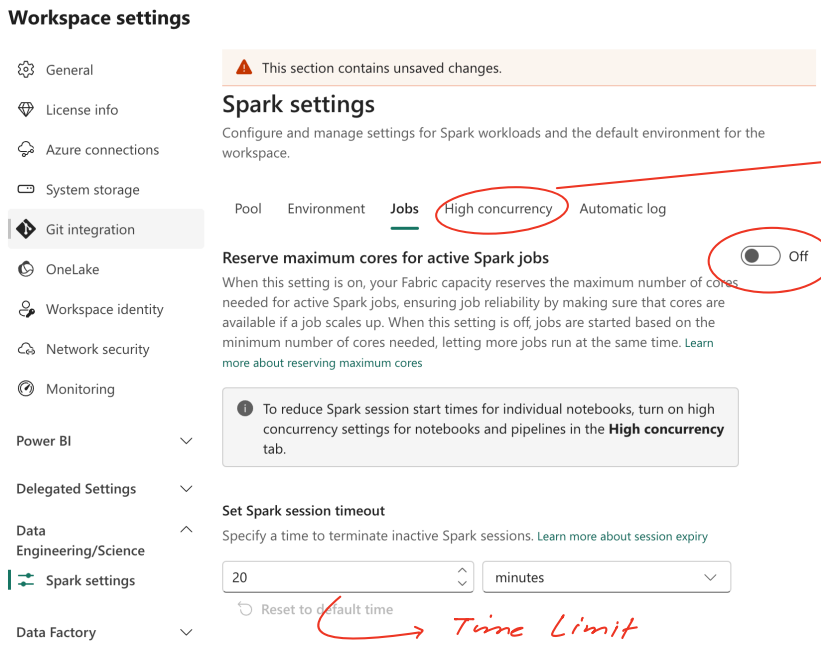
Environment;



Inside Spark, change env which uses custom pool



Other Spark Settings;



→ Time Limit

→ High concurrency means system is designed to handle multiple users or parallel notebook executions efficiently

Domain Workspace Settings; Domains are used to organize workspaces

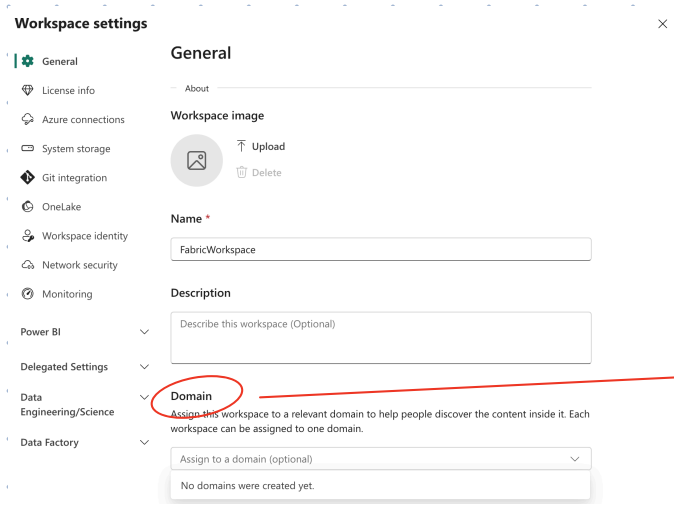
Settings → Admin Portal → Domains.

Note; These settings are specific to Fabric Admin

Important; In Domain we assign workspace or Subworkspaces to Domain

example Sales workspace → Sales Domain

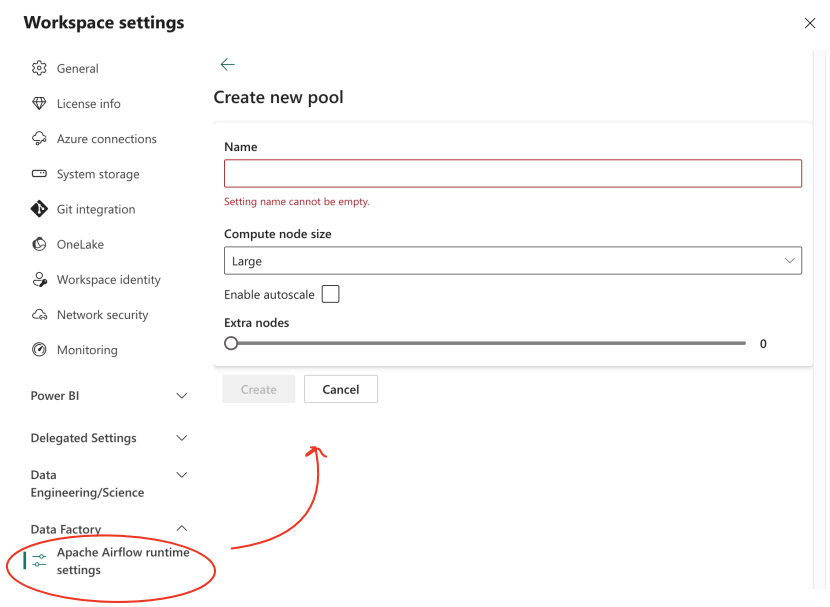
we can also use workspace setting to map our workspace to specific Domain



→ Assign workspace to Domain.

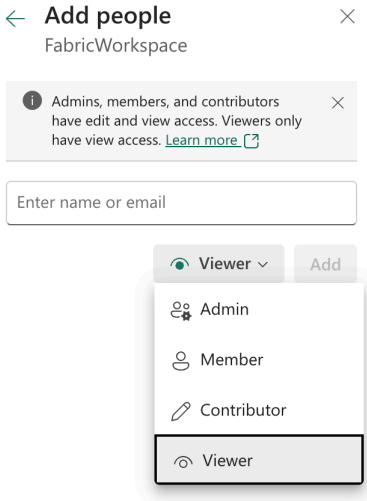
Apache Workspace Settings:

Admin → Allow Apache Airflow Jobs



Workspace Roles in Fabric

Role	Can View Content	Can Edit Content	Can Share/Publish	Can Manage Permissions	Can Delete Workspace
Viewer	✅ Yes	❌ No	❌ No	❌ No	❌ No
Contributor	✅ Yes	✅ Yes	❌ No	❌ No	❌ No
Member	✅ Yes	✅ Yes	✅ Yes	❌ No	❌ No
Admin	✅ Yes	✅ Yes	✅ Yes	✅ Yes	✅ Yes



Important Notes.

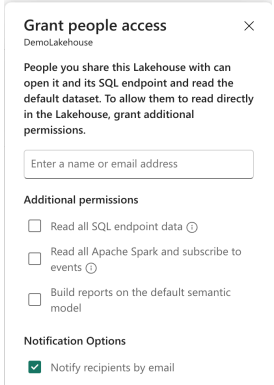
📌 Notes:

- **Viewer**: Can only view reports, dashboards, and other items.
- **Contributor**: Can edit or create items but can't publish apps or change permissions.
- **Member**: Can do everything except manage access or delete the workspace.
- **Admin**: Full control over everything in the workspace.

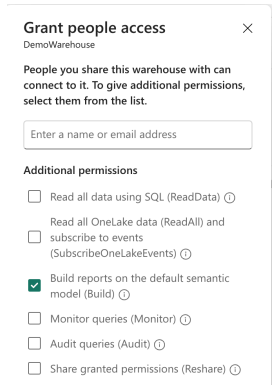
Manage Access to Fabric Items:

Lakehouse

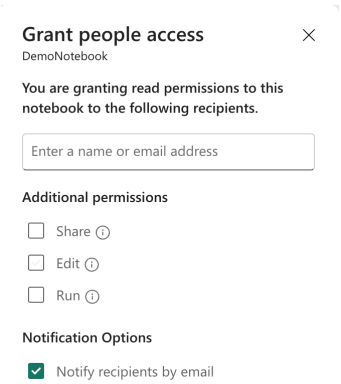
Manage Access



Warehouse



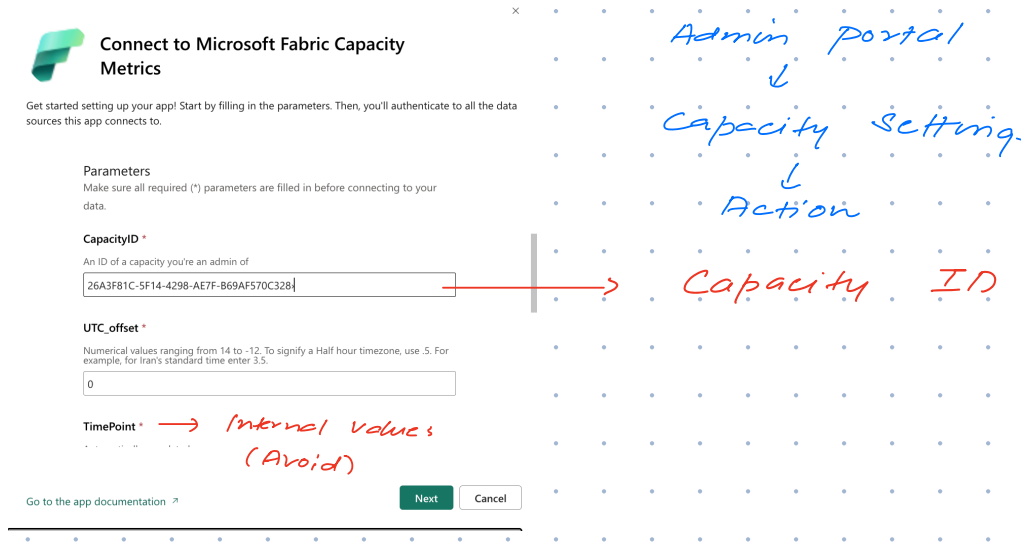
Notebook



Fabric Capacity Metrics App

Power BI App - Fabric Capacity Metrics

→ This allows to choose plan esp.

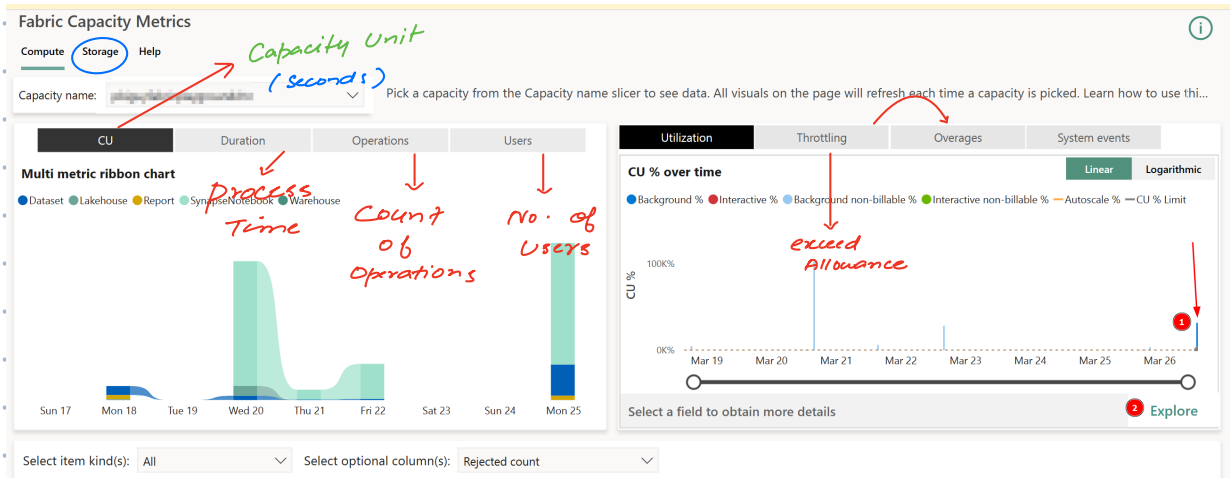


Admin portal
↓
Capacity Settings
↓
Action

→ Capacity ID

→ Internal values (Avoid)

App Interface:



Capacity Unit (seconds)

Process Time

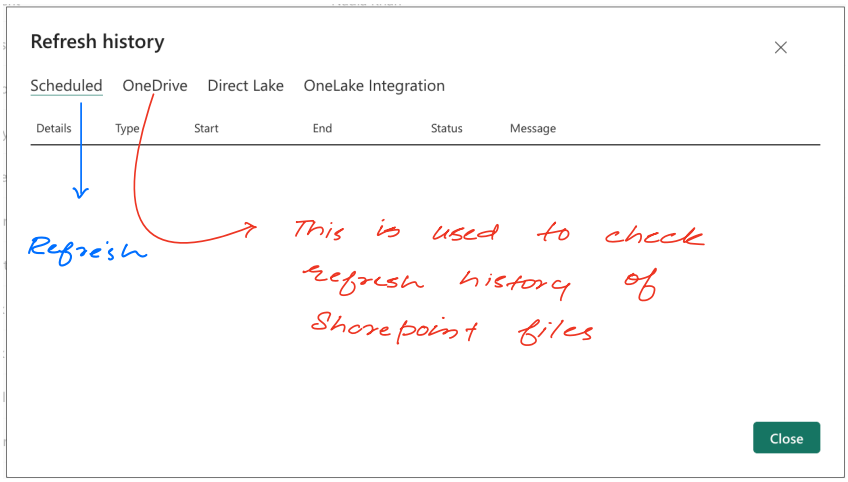
Count of Operations

No. of Users

exceed Allowance

Semantic Model Refresh

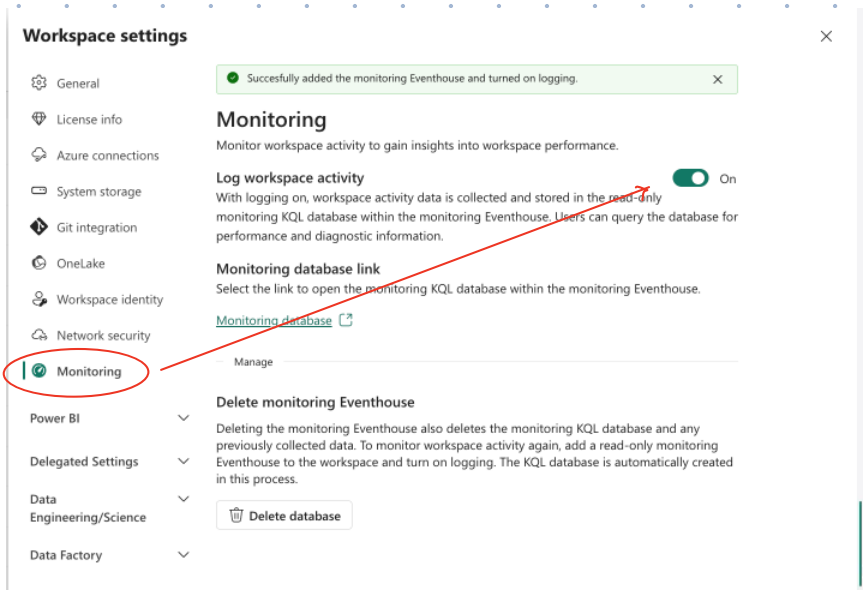
: → Semantic models → Refresh history

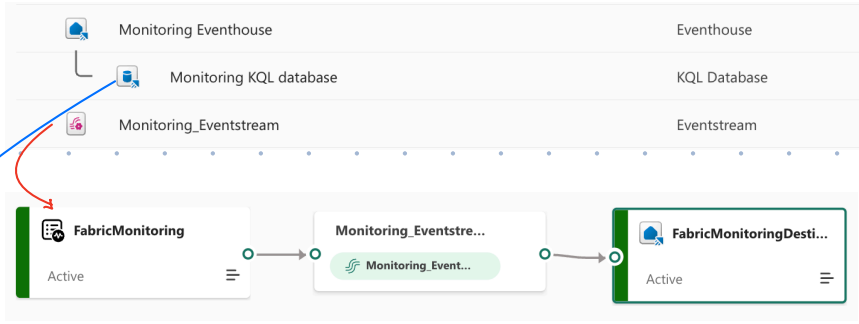


Refresh

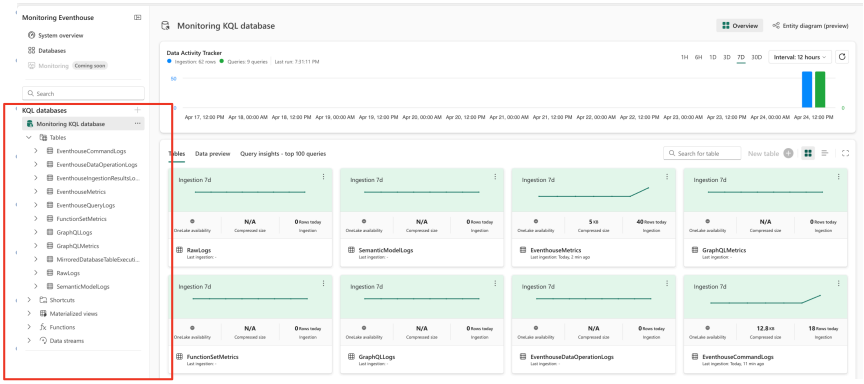
→ This is used to check refresh history of Sharepoint files

Implement Workspace Logging





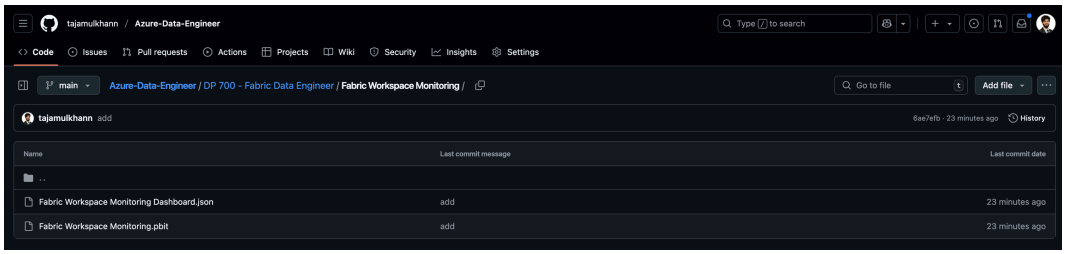
Monitor KQL Database;



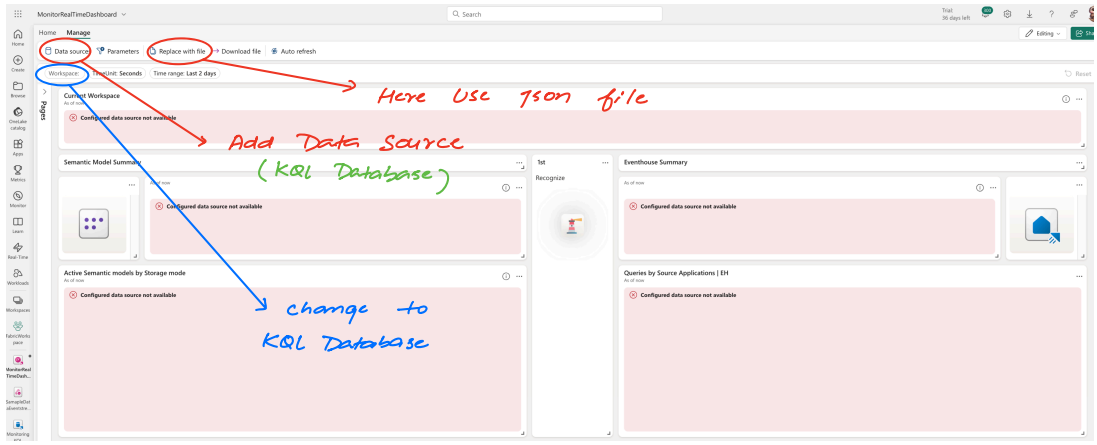
Note we can use KQL to Query workspace logs

Workspace Logging Dashboards

Download

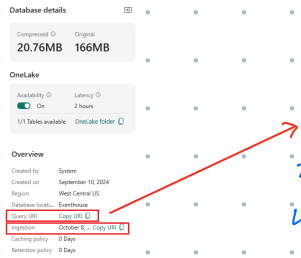


Real Time Dashboard



2nd Option

Use pbit file, and then populate with



Important

1. Workspace Retention period
   Default = 7 Days

2. Contributor Role = To Read Lakehouse
   Viewer Role = To Read SQL Endpoint

3. CU in Monitoring app = Capacity Unit

4. Workspace monitoring logs are retained for 30 Days

5. I am using a KQL Database.
   I create a database shortcut (follower) to the database `OtherDB` and the table `OtherTable`.
   What KQL code can I use to query the `OtherTable`?

   - ○ `OtherDB.OtherTable`
   - ◉ `database('OtherDB').table('OtherTable')` — Correct ✓
     That's right.
     Explain with AI

SECURITY & GOVERN

Data Governance & Security is Covered by Microsoft Purview

Apply Sensitivity Labels

By default we have, Confidential, No Sensitivity or None. No Sensitivity is default label

Label	Description	Encryption	Access Restriction
Highly Confidential	Top-level protection for critical or regulated data.	✅ Yes	✅ Strict (No external share)
Confidential	For sensitive internal data.	✅ Optional	✅ Limited (Internal only)
No Sensitivity	Default label; no special protection.	❌ No	❌ None
None	No label assigned.	❌ No	❌ None

Q1 How to add custom Sensitivity labels

Purview – Information Protection – Sensitivity Labels – Add + (Choose level of Access)

Q1 Where can we apply labels

- Power BI Desktop
- Power BI Service

Endorsement

Endorsement Type	Purpose	Visibility Boost	Who Can Set It	Usage Example
Promoted	Highlights content as useful or recommended	🔼 Yes (search ranking)	Workspace Admins, Contributors	A well-built report or reusable dataset
Certified	Officially approved content, organization-wide	🔼🔼 Highest	Admins or Designated Certifiers	Company-wide sales dashboard
None	Default state with no endorsement	❌ No boost	—	Work-in-progress or ad-hoc content

Security in Datawarehouse

Create Warehouse → Create Function

Row Level Security

```
CREATE TABLE tblActual
(
Country VARCHAR(20),
Location VARCHAR(20),
Actual INT,
UserName VARCHAR(20)
)

INSERT INTO tblActual (Country, Location, Actual, UserName) VALUES
('England', 'London', 5000, 'Susan'),
('England', 'Birmingham', 7000, 'Susan'),
('England', 'Manchester', 11000, 'Susan'),
('France', 'Paris', 4000, 'Jane'),
('Italy', 'Milan', 3000, 'Jane'),
('Italy', 'Rome', 13000, 'Thomas');

SELECT USER_NAME()

CREATE FUNCTION securityfunction(@UserName AS VARCHAR(50))
RETURNS TABLE
WITH SCHEMABINDING
AS
RETURN
SELECT 1 AS securityfunction
WHERE @UserName = LEFT(USER_NAME(), LEN(@Username)) OR LEFT(USER_NAME(), 4) = 'Jane'

CREATE SECURITY POLICY SecurityPolicy
ADD FILTER PREDICATE dbo.securityfunction(UserName)
ON dbo.tblActual
WITH (STATE = ON);

SELECT * FROM tblActual
```

Dynamic RLS

This gives logged in username

Column Level Security

```
CREATE TABLE tblActual
(
Country VARCHAR(20),
Location VARCHAR(20),
Actual INT,
UserName VARCHAR(20)
)

INSERT INTO tblActual (Country, Location, Actual, UserName) VALUES
('England', 'London', 5000, 'Susan'),
('England', 'Birmingham', 7000, 'Susan'),
('England', 'Manchester', 11000, 'Susan'),
('France', 'Paris', 4000, 'Jane'),
('Italy', 'Milan', 3000, 'Jane'),
('Italy', 'Rome', 13000, 'Thomas');

SELECT * FROM tblActual;

GRANT SELECT ON tblActual TO [jane@Filecats.onmicrosoft.com]
GRANT SELECT ON tblActual(Country, Location) TO [susan@Filecats.onmicrosoft.com]
```

In production, we will Rather Create Views CLS ✗

Object Level Security

Grant
Revoke

we can Grant Select on Tables, View, Proc etc.

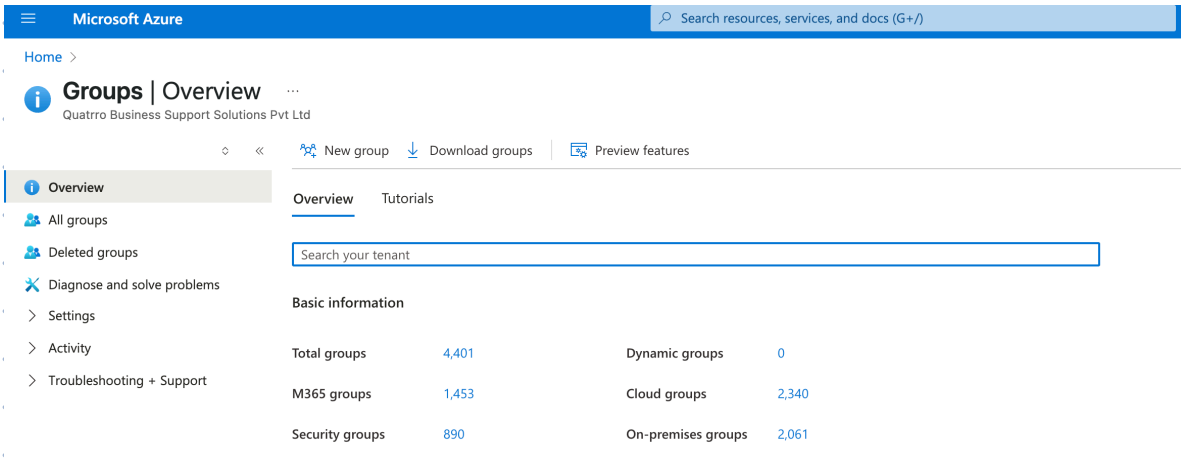
```
GRANT SELECT ON tblActual TO [jane@Filecats.onmicrosoft.com]
GRANT SELECT ON tblActual(Country, Location) TO [susan@Filecats.onmicrosoft.com]
REVOKE SELECT ON tblActual TO [susan@Filecats.onmicrosoft.com]

DENY SELECT ON tblActual TO [Microsoft@Filecats.onmicrosoft.com]

SELECT * FROM sys.database_principals

GRANT SELECT ON tblActual TO [New Group 2]
```

check Users

Important To undo Deny, use Revoke Then, Grant.

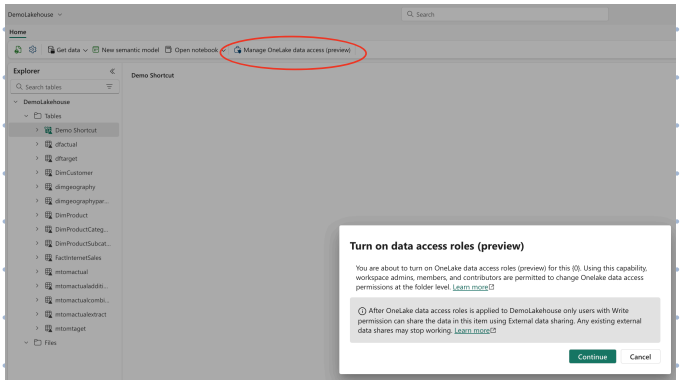
we cant Deny permissions to object owners

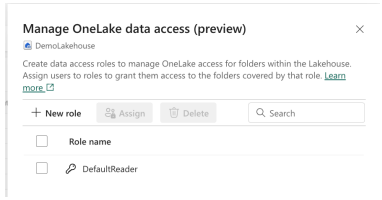
Q1 How to Add Security Groups;

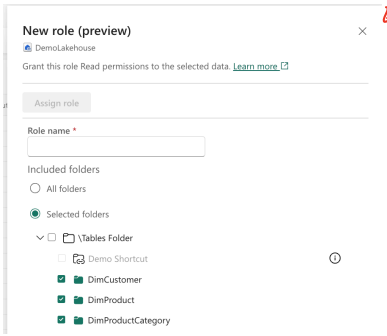


Important; we can share our objects (DW's) etc with these Groups and then use Grant, Revoke, Deny on Groups

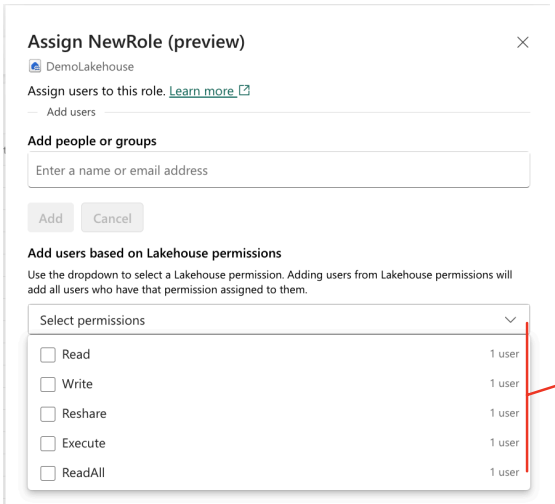
SECURITY IN LAKEHOUSE







ASSIGN ROLES ;



permissions.

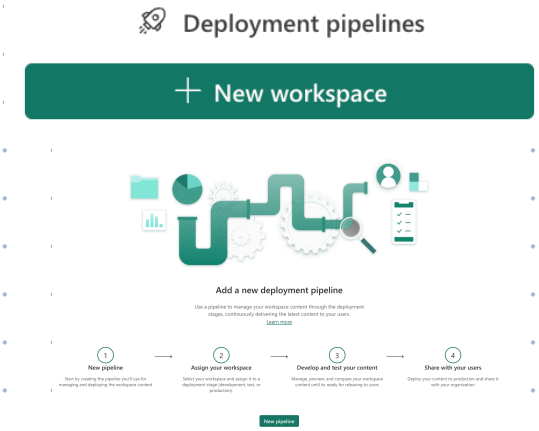
Deployment Pipelines;

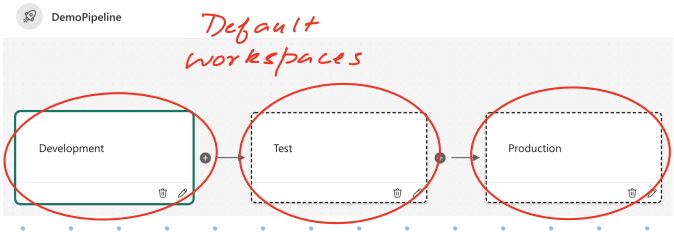
Lets you deploy your data assets (like reports, datasets, notebooks) through 3 stages

- Development
- Test
- Production

Create Deployment Pipeline;

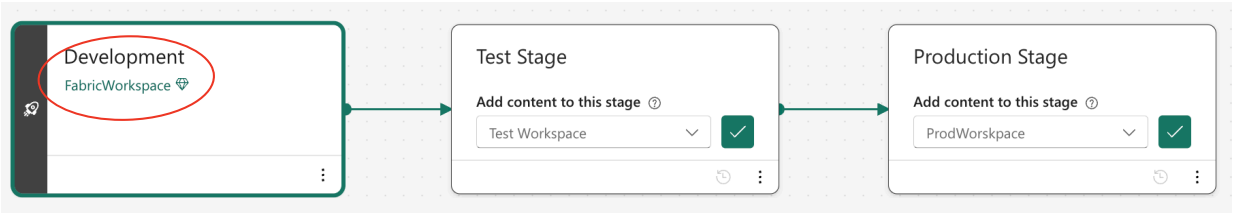
Workspaces ⟶ Deployment Pipeline





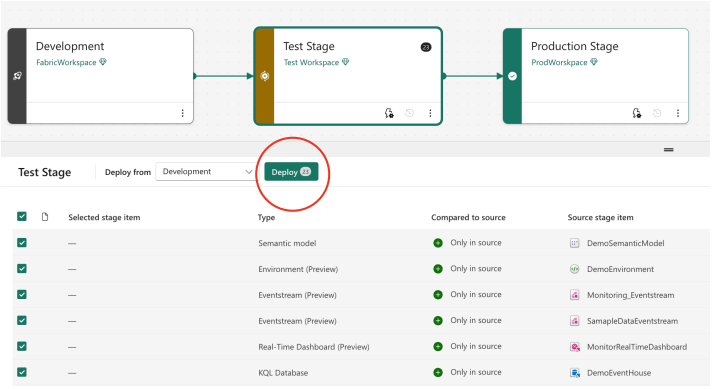
Note; we can have upto 10 workspaces and minimum of 2.

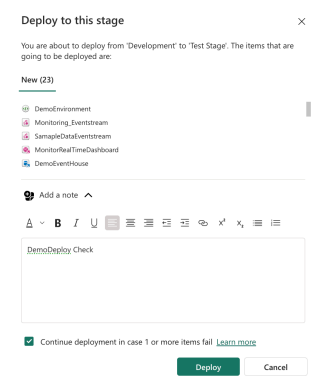
Assign Workspaces



Note; It is preffered to have multiple workspaces (Dev, Test, Prod)

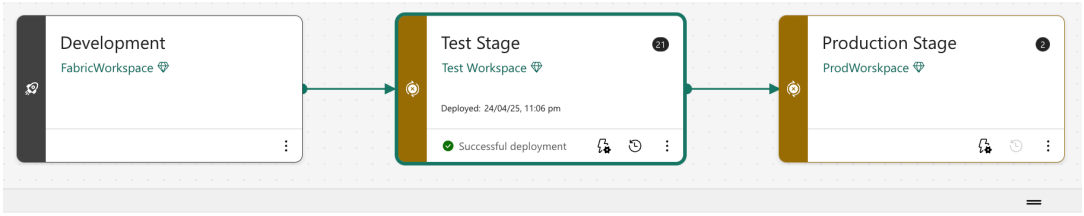
Deploy from Dev to Test Stage

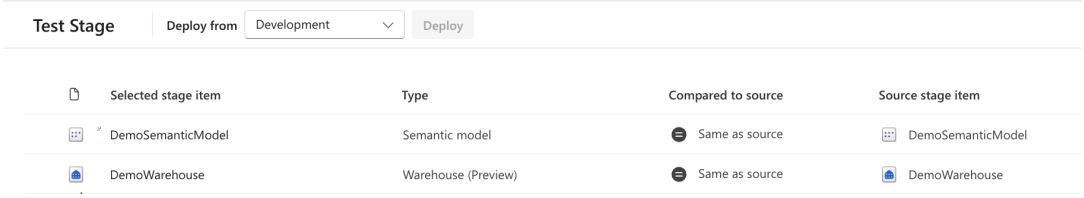




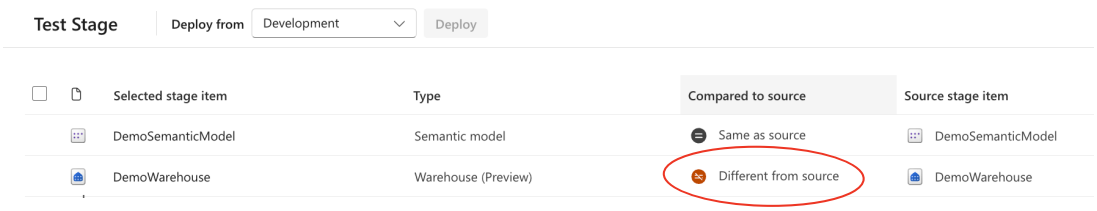
Note; Gen2 Dataflow etc is not supported

After Deployment

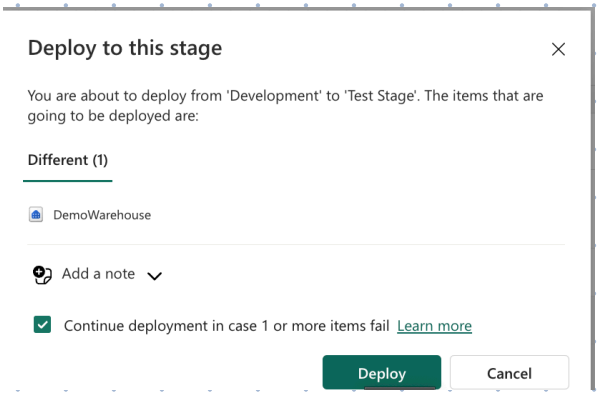




Note; If I make any changes, then



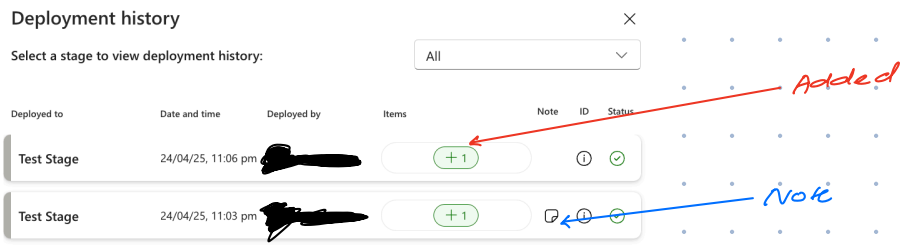
while ReDeploying only changes will be Deployed



Note; Any changes like Rename won't affect as it will be automatically mapped

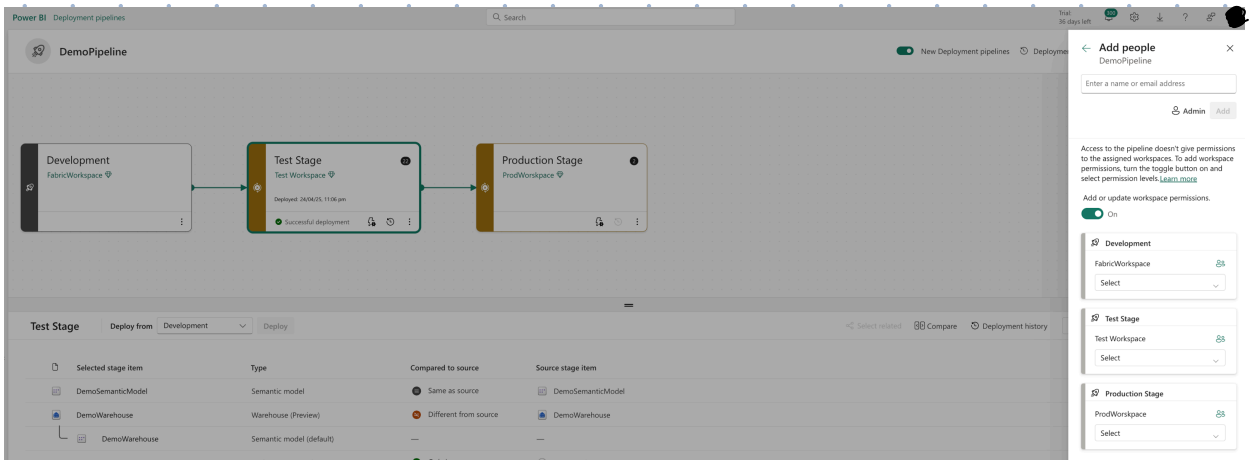
Configure Deployment Pipelines;

1. Deployment History;



Note; we can't Download microsoft .pbix file after Deployment.

Manage Access To Pipeline



Q// What can Pipeline Admin Do ?

Plan & Implement Deployment Solutions

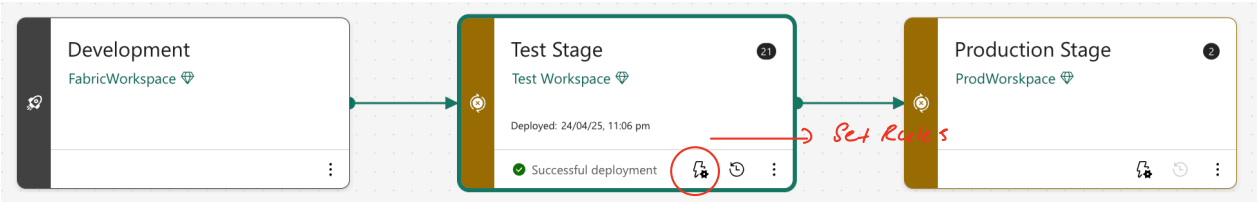
Pipeline Admin can:

- Create, view, edit, share, delete pipelines
- Assign/unassign workspaces to stages
- View deployment history & stage items
- Manage settings & users

If also a Workspace Contributor, they can:

- Compare stages
- Set rules
- Deploy to next stage (if contributor/member/admin of both workspaces)

Setup Rules



Rules can help changing Data sources

THE END

Key Concepts;

1. Warehouses can store both Relational and Standalone tables given they have fixed schema.

2.

Lakehouse	Warehouse	Eventhouse (KQL Database)
Store and analyze large, mixed data (structured + semi-structured)	Store and query pure structured, relational data	Store and query streaming, time-series, or log data
Structured + Semi-structured (e.g., CSV, Parquet, JSON)	Strictly structured (tables, columns, SQL schema)	Structured event data (logs, metrics, telemetry)
SQL (Apache Spark SQL)	T-SQL (Transact-SQL, like Azure Synapse)	KQL (Kusto Query Language)
Moderate (through Eventstreams or external tools)	Low (batch-oriented, scheduled refreshes)	High (real-time ingestion and querying)
Big Data, AI/ML, combining batch + streaming data	Reporting, BI dashboards, transactional analytics	Monitoring, alerting, real-time dashboards
Optimized for big data workloads (may have some latency)	Optimized for high concurrency, fast SQL analytics	Optimized for lightning-fast query over event data
Cheaper for massive volumes of raw data	Costlier due to high performance/compute for structured queries	Cost-effective for huge volumes of log/telemetry data

3. MASKING ; Column MASKED WITH (FUNTION = "default")

4. QUERY FOLDING;

✅ **Yes — query folding *can still happen*** even if your initial query in Power BI is just:

```sql
SELECT * FROM TableName
```

Then, if you apply filters in **Power Query** (like filtering dates or removing nulls), Power BI tries to **fold those steps** back into a single SQL query like:

```sql
SELECT * FROM TableName WHERE OrderDate >= '2024-01-01'
```

So even though your original query was `SELECT *`, Power Query is smart enough to push your filter step **back to the source** — *as long as the transformations are foldable*.

5. SQL READ OR WRITE

Lakehouse = Read only (even in notebook SQL)

Warehouse = Read + Write

6. MERGE

Dataflows Gen 2 support Merge on multi columns

7. Warehouse doesnt support Varchar (Max)

8. In Lakehouse, always upload table data to tables, for efficient retrieval and they can be accessed through SQL Endpoint

9. Shortcut;

Doesn't copy Data but only creates shortcut

10. Save Table in DW;

Select * from Table
↓
Save as Demo

(X)

Takes more Time

Select * from Table1
into Table

Faster and efficient

11. Incremental Load;

Small Data = 1 week / 1 Query

Big Data = 7 Days / 7 Queries

12. GIT INTEGRATION

Azure Devops Git = Official (more Secure)

GITHUB = Open Source

13. AZURE DEVOPS Git;

Dashboard = Not supported

Data = only Meta Data Copied.

14. KQL = orders by Desc default
Faster than SQL
Used in ADX or Eventhouse

```
// Selecting Columns in KQL
// Use // for comments
// Use | to separate operators in the pipeline

Weather // Table name
| project State, EpisodeId, StartDate=StartTime, EndTime, EpisodeNarrative, EventNarrative // Select columns with optional aliasing (StartDate = StartTime)
| extend Duration = EndTime - StartDate // Add calculated column
| extend EpisodeNarrative = "hi" // Overwrite values in the column (not renaming rows)
| order by State asc nulls last // Order results; asc by default, 'nulls last' is valid
| project-rename Narrative = EpisodeNarrative // Rename column: new name = old name
| project-away EventNarrative // Remove column
```

Limit 5 = Limits to 5 Rows

* has is used to match word & contains for string

15. Numeric D-Types in KQL

int = 1/2 = 0 (Floor Division)

Decimal = 1/2 = 0.5

16. UNION THE DATA ;-

one Query & second shoud be brackets

```
Weather
| distinct State, EventType
| union kind = outer withsource = TableSource
(Weather
| distinct State, EventNarrative)

union withsource = AdditionalColumn Wea*
```

Default union = outer

Inner = only Column which are common in both tables

17. JOIN IN KUSTO SQL

Default join = innerunique

```
Weather
| join
(Region | extend State = toupper(State))
on State

Weather
| join kind = fullouter
(Region | extend State = toupper(State))
on ($left.State == $right.State)
```

→ can use on state and on

Imp; Keep small table on left side for efficient join

18. ORDERED FLOW OF DATA IN FABRIC

Step	Component	Purpose / Description
1	**Data Sources**	External data (APIs, databases, files, IoT, logs, etc.) collected from different systems.
2	**OneLake**	Centralized data lake storage for all data in Fabric (like a single data lake for all).
3	**Pipelines** (Data hub)	Used to orchestrate data movement and transformations (ETL/ELT).
4	**Dataflow Gen2**	No-code/low-code data prep tool for cleaning, shaping, and transforming data.
5	**Lakehouse**	Combines lake storage + structured access using Delta format (best for analytics).
6	**Eventhouse**	Specialized for ingesting and analyzing real-time, event-based data (e.g., logs/telemetry).
7	**Notebooks**	For data engineering, Python/Spark/SQL-based processing and advanced transformations.
8	**SQL Endpoints**	Query interface to access and analyze data from Lakehouse/Eventhouse.
9	**Warehouse**	Structured data warehouse optimized for BI and analytics (relational model).
10	**Power BI**	Data visualization and reporting layer to build dashboards and reports.

19.

How they work together:

- **EventStream** *brings in* real-time events (clicks, purchases, actions)
- **EventDB** *stores* these events efficiently for fast querying and reporting
- **EventHouse** provides *the full infrastructure* to manage ingestion, storage, and analysis

20. Eventstream = Pipeline for Streaming Data.

21. Eventstream Double Data Type is used to store Decimal.

22. Retention period for Eventstream = 1 Day.

23. Workspace Retention period

Default = 7 Days

24. Contributor Role = To Read Lakehouse

Viewer Role = To Read SQL Endpoint

25. CU in Monitoring app = Capacity Unit

26. Workspace Monitoring Logs are retained for 30 Days

27.

I am using a KQL Database.
I create a database shortcut (follower) to the database `OtherDB` and the table `OtherTable`.
What KQL code can I use to query the `OtherTable`?

- ○ `OtherDB.OtherTable`
- ◉ `database('OtherDB').table('OtherTable')` — Correct
  That's right.
  Explain with AI

28. How to add custom Sensitivity labels

Purview - Information Protection - Sensitivity Labels - Add +

29. Endorsements ;

Promoted : Workspace Admin

Certified : Official Certifiers

30. Dataflow Gen2 is not supported in Deployment pipelines

31. Any changes like Rename won't affect as it will be automatically mapped

32. we can't Download Microsoft .pbix file after Deployment.

33. V-Order improves Read & write speeds.

34. Activator can't be created on Lakehouse.

35. Dataverse is not a destination for Dataflow Gen2

36. Delete Data Activity is Unavailable in Incremental pipeline

37. Refresh can be set Daily or weekly

38. In Maintenance, we can run VACUUM command

39. Easy way to remember windows

Type	Think of it like...
Tumbling	Tiles neatly placed on a floor
Hopping	Overlapping rugs
Sliding	A sliding glass door, moving bit by bit
Session	A casual meetup that lasts until everyone leaves

40. To start Spark Job stream

df = spark.readstream

41. Below F64 → we can Run Power BI Report

42.

Your selection is correct

☑ The quickest you can schedule a Dataflow Gen2 to run is every 30 minutes.

Your selection is correct

☑ You can schedule a Data Pipeline to run every minute.

43. To Block users from sharing access to personal workspace → Microsoft 365 Admin

43. Minimum Access level to compare 2 stages in DP Contributor

44. I want to copy Large Data without Transformation Copy Tool in Pipeline

45. Importance of Cache & Shortcuts

📰 Simple Analogy:

Imagine you're watching a YouTube video.

- Without caching: You stream it fresh every time → uses a lot of data.
- With caching: It saves the video locally once → next time, no extra internet use.

And **shortcuts** are like **bookmarks** — they point to the real video, but don't take up extra space.

🧠 Quick Summary:

- Use **Eventstream + KQL** for **real-time monitoring**.
- Use **Data Pipeline** for **general ETL** tasks (batch or near real-time).
- Use **Apache Spark** for **large data volumes**, **ML**, or **custom logic**.
- Use **Dataflow Gen2** for **drag-and-drop transformations**.
- Use **Streaming Dataflow** for **simple streaming data transformation**.
- Use **Lakehouse** for **unified analytics** storage.
- Use **Warehouse** for **Power BI + SQL analytics**.

🧠 Simplified Rules of Thumb:

- 🧱 **Dataflow Gen2** = No-code **batch** ETL
- ⚡ **Streaming Dataflow** = Low-code **real-time** stream processing
- 🔁 **Data Pipeline** = Orchestrating batch or near real-time pipelines
- 🔬 **Apache Spark** = Big data, ML, or custom logic (code-heavy)
- 📡 **Eventstream** = Real-time ingestion from external systems
- 📊 **KQL** = Real-time querying/log analytics
- 🏠 **Lakehouse** = Store everything (files + metadata)
- 🏬 **Warehouse** = SQL + BI-ready structured analytics