Taksh Shah

# Project: Incremental Data Loading & Automated Notifications using Fabric

## Problem Statement:

In modern data ecosystems, organizations need to efficiently ingest, transform, and load data from various sources into centralized platforms for analytics, while also ensuring timely monitoring and notification upon successful data refreshes. This project addresses the challenge of incrementally loading data from on-premises sources to Microsoft Fabric Lakehouse, processing it through a structured transformation pipeline, and triggering automated notifications upon successful execution.

## Project Objective:

To build an end-to-end data pipeline on Microsoft Fabric that:

1. Ingests structured data from on-premises environments into a Fabric Lakehouse using the On-Prem Gateway

2. Utilizes the AI Bank Dataset as the source

3. Implements Dataflow Gen 1 to join tables, remove duplicates, and clean data

4. Loads the cleansed data into a Fabric Warehouse

5. Applies Slowly Changing Dimension (SCD) Type 1 logic using Fabric Notebooks and writes the results into separate warehouse tables

6. Schedules and monitors the pipeline, sending an automated email notification (via Outlook or Gmail) upon successful pipeline completion

## Tools & Technologies:

- Microsoft Fabric
- On-Premises Data Gateway
- Fabric Lakehouse and Warehouse
- Fabric Dataflow Gen 1 / Gen 2
- Fabric Notebook
- Email Notification Task (in-built)
- Azure Key Vault (optional for secure credential management)
- Draw.io / Visio for architecture diagram

**Bronze Layer:**

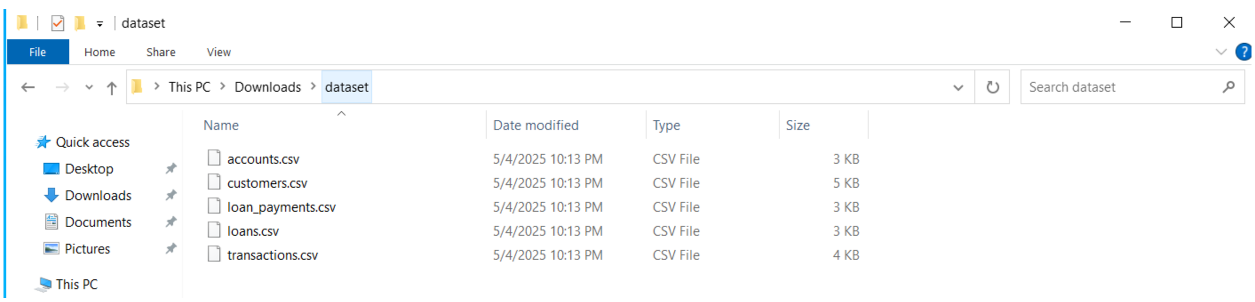
To begin with, we'll have to ingest data from on-premise environments into a Fabric Lakehouse using On-Prem Gateway.

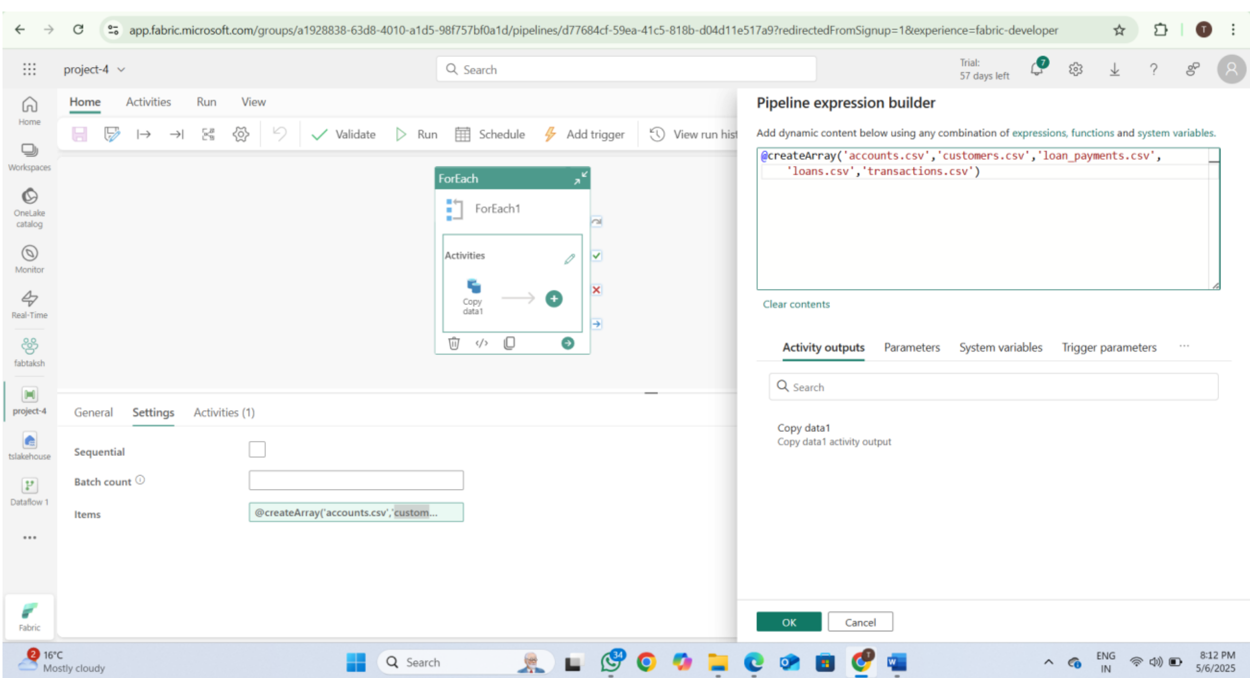
To achieve this task, let's utilize an azure virtual machine. Suppose there is some data on that virtual machine (on-prem environment) which we need to ingest into the fabric lakehouse using on-prem gateway.

In the virtual machine, download the [AI Bank Dataset] and [On-Prem Gateway] (download standard mode).

Install the on-prem gateway on the virtual machine. Once it completes, it'll ask for an email address to use the gateway with – use your fabric account email – and sign in. Consequently, it'll ask to give a name to the gateway and set a recovery key (password) for the gateway.

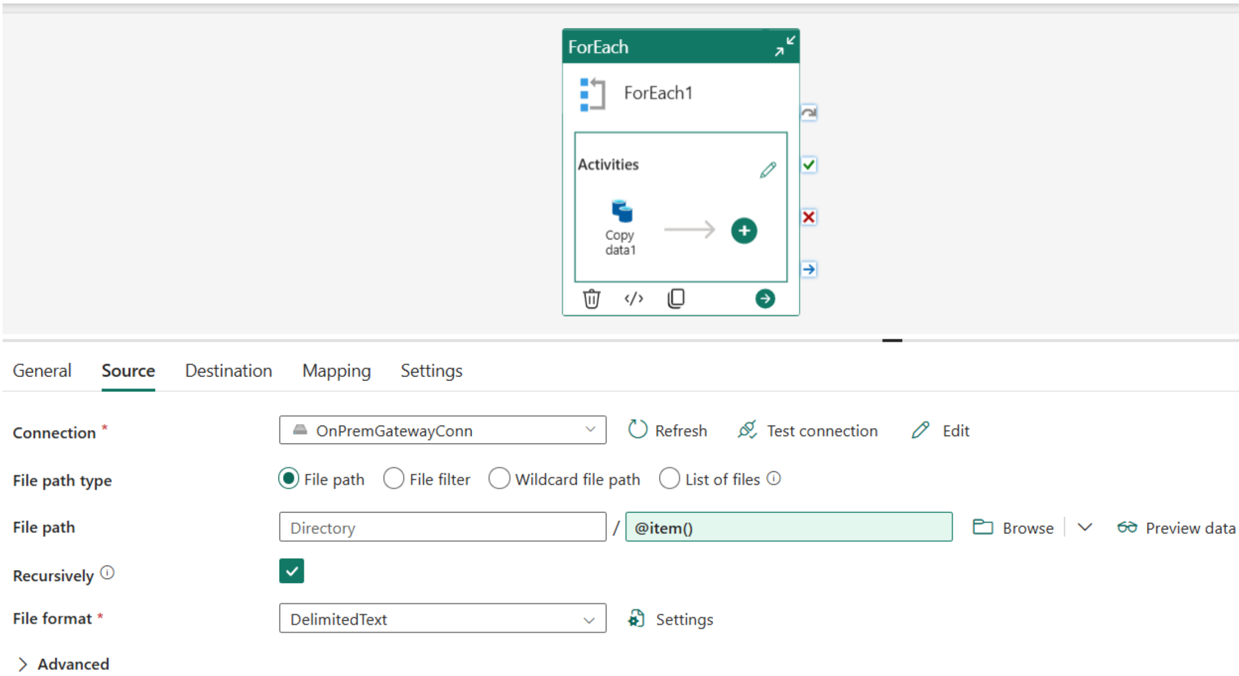
Now, we'll create a pipeline and use a copy data activity in fabric lakehouse along with a for-each loop to get all the files of the dataset from on-prem.



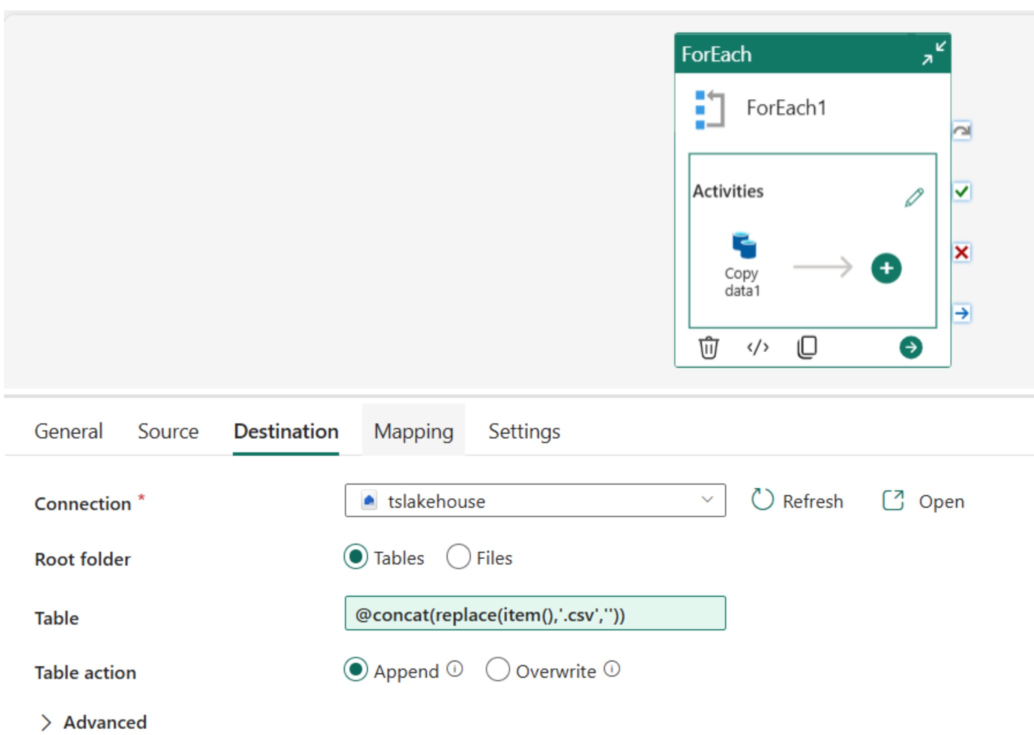



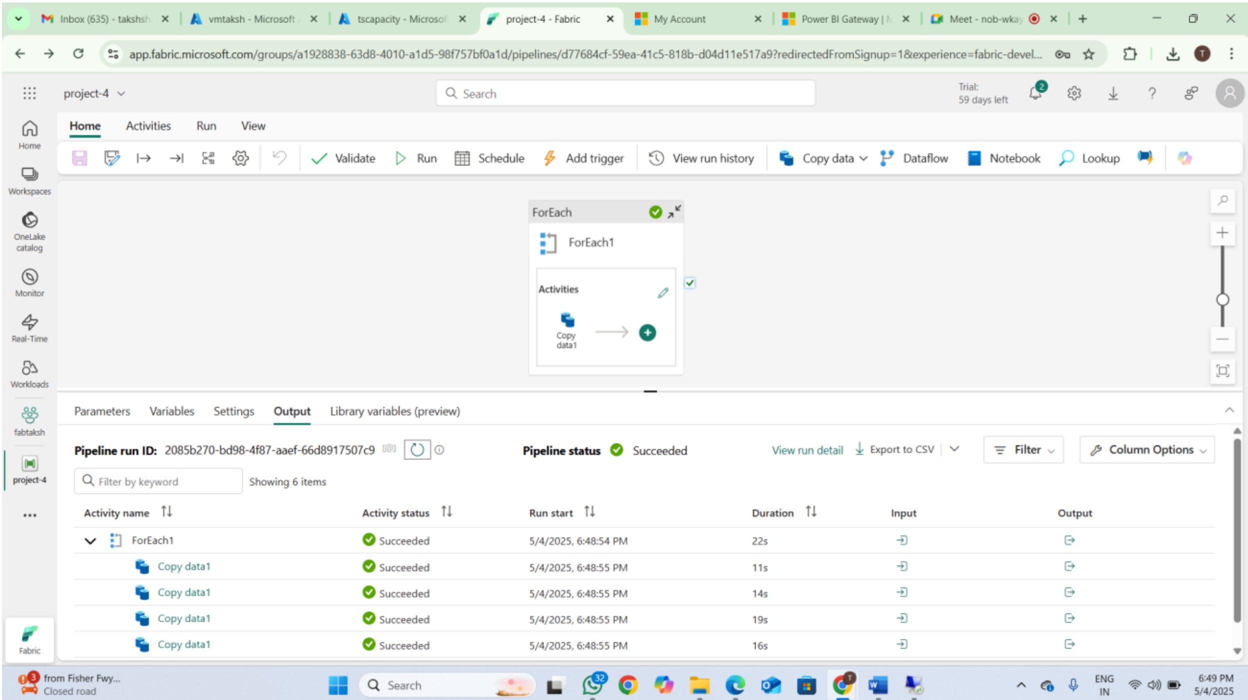
Taksh Shah

For settings section of for-each loop, under “items” we provide an array of the names of the dataset files as the loop expects an array input.




For OnPremGatewayConn connection, we have to provide the path of folder in virtual machine where the dataset is located.



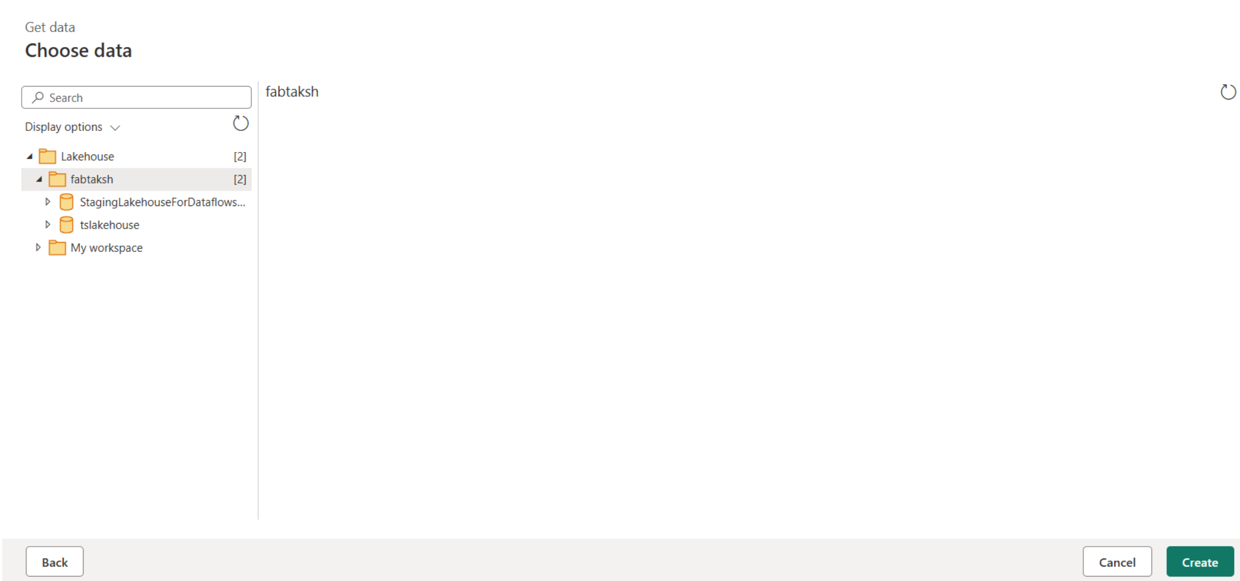



**Pipeline ran successfully**

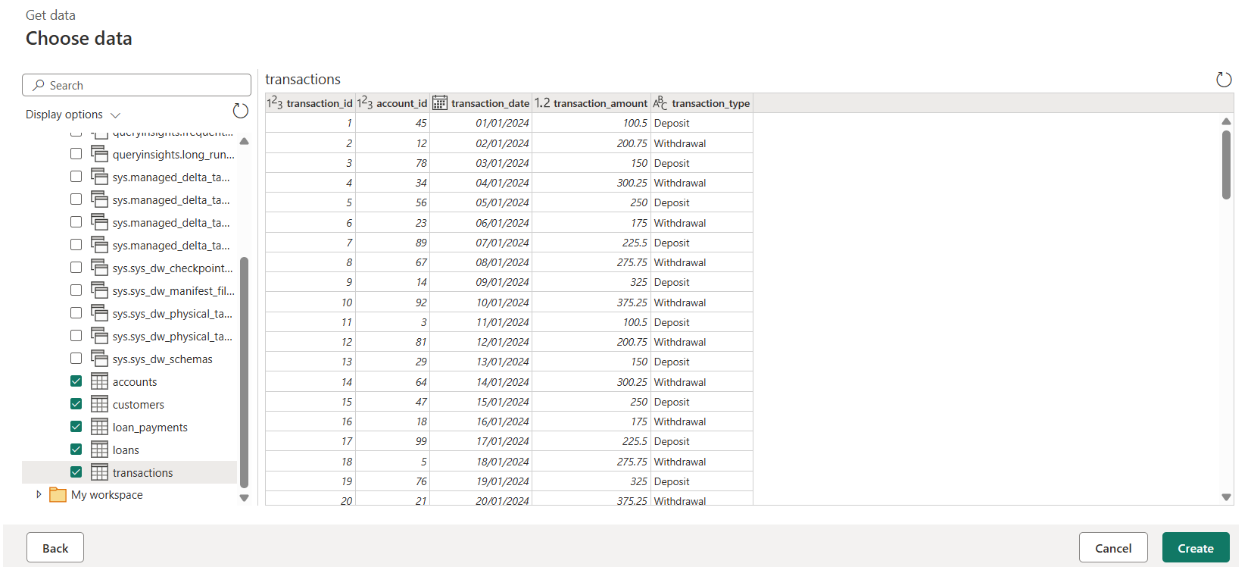
Once the pipeline runs successfully, you'll be able to see the tables loaded in fabric lakehouse.

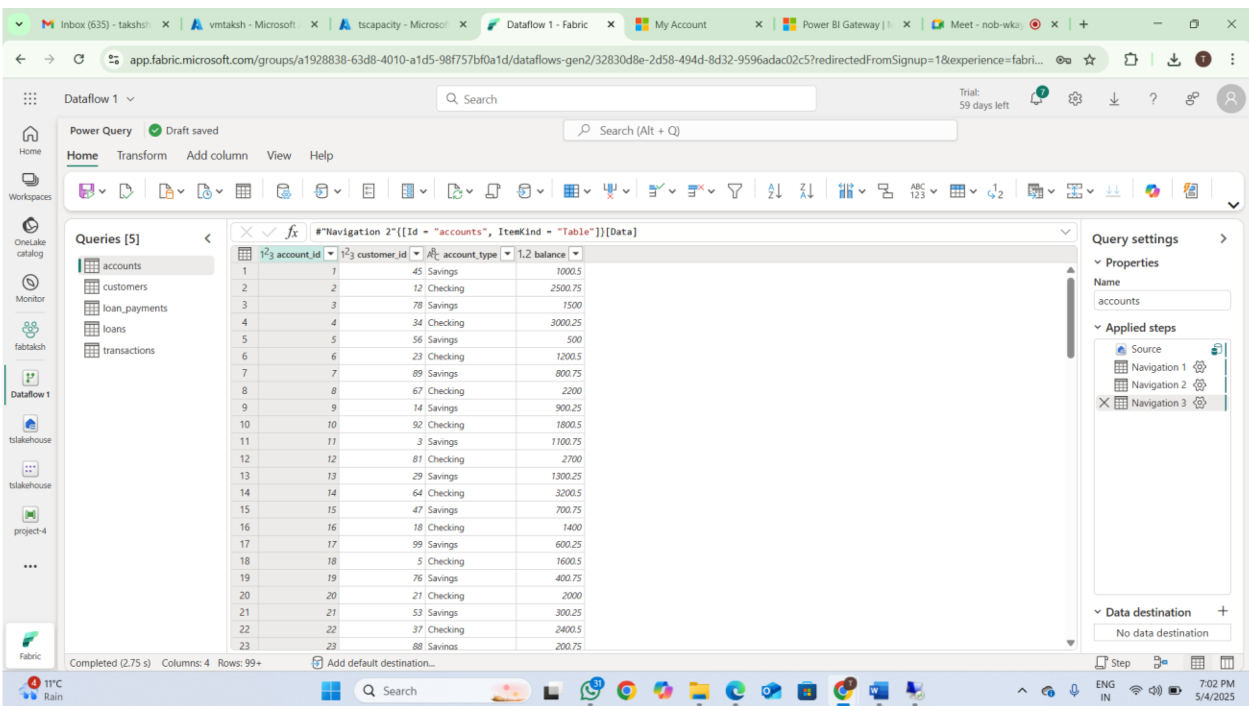
## <u>Silver Layer:</u>

We'll use dataflow gen2 to clean data (remove duplicates, replace nulls). Let's import the source tables in the dataflow. In dataflow gen2, go to **getdata->more->lakehouse** as our source will be the files which we just ingested from on-prem (virtual machine) to lakehouse.



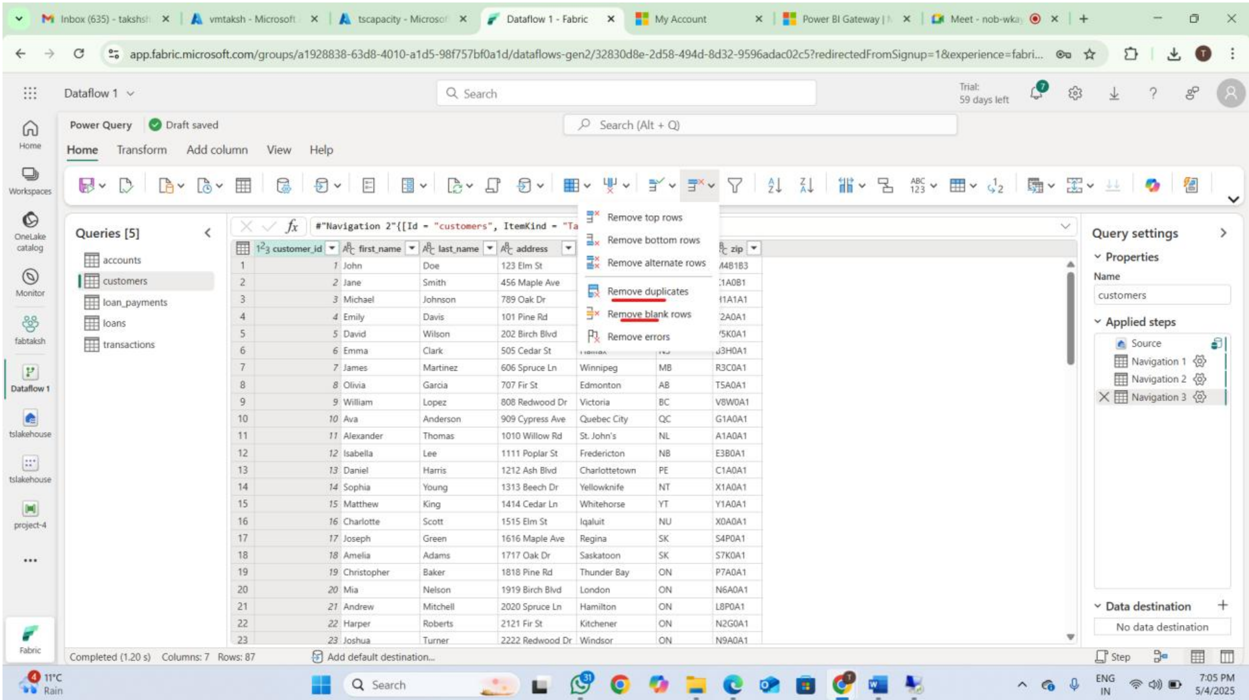
Select the tables which we created from the dataset files.

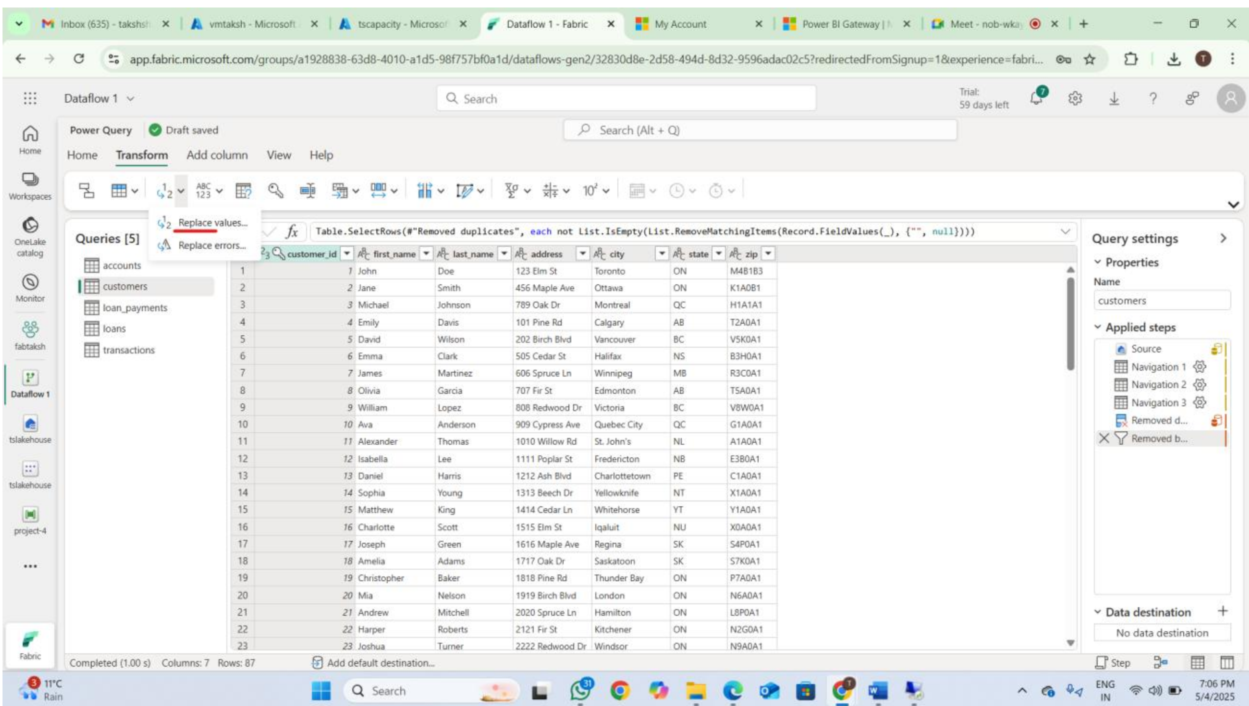




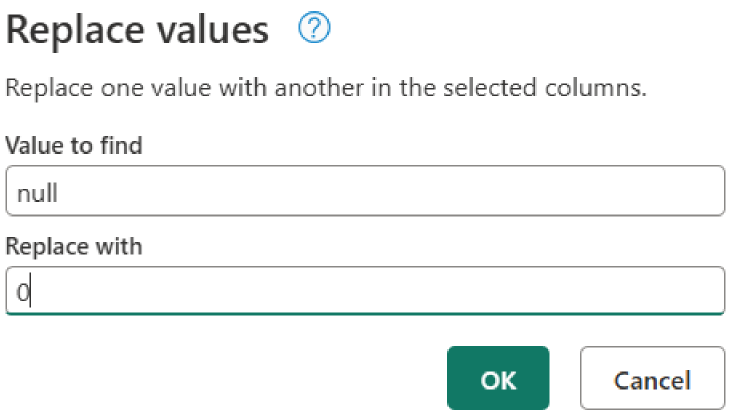
All the tables have been created in the dataflow. Follow the images provided below to clean the data for customers table and repeat the process for all the other tables.

Taksh Shah

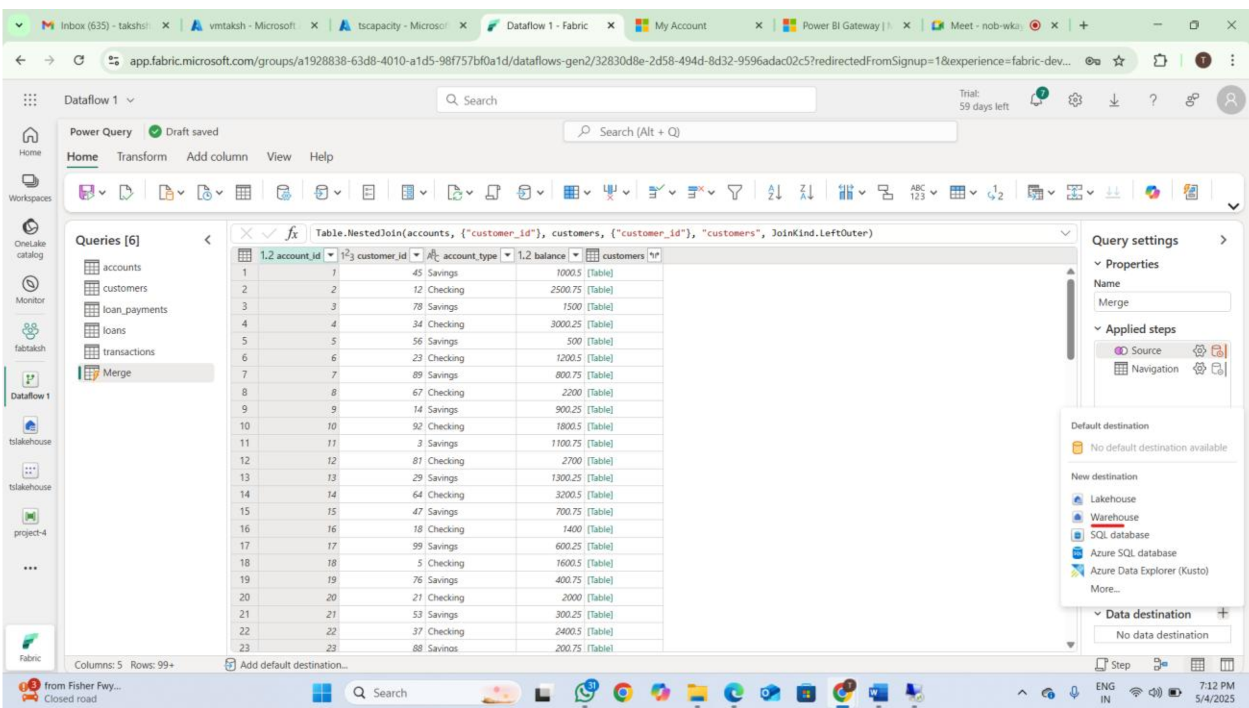


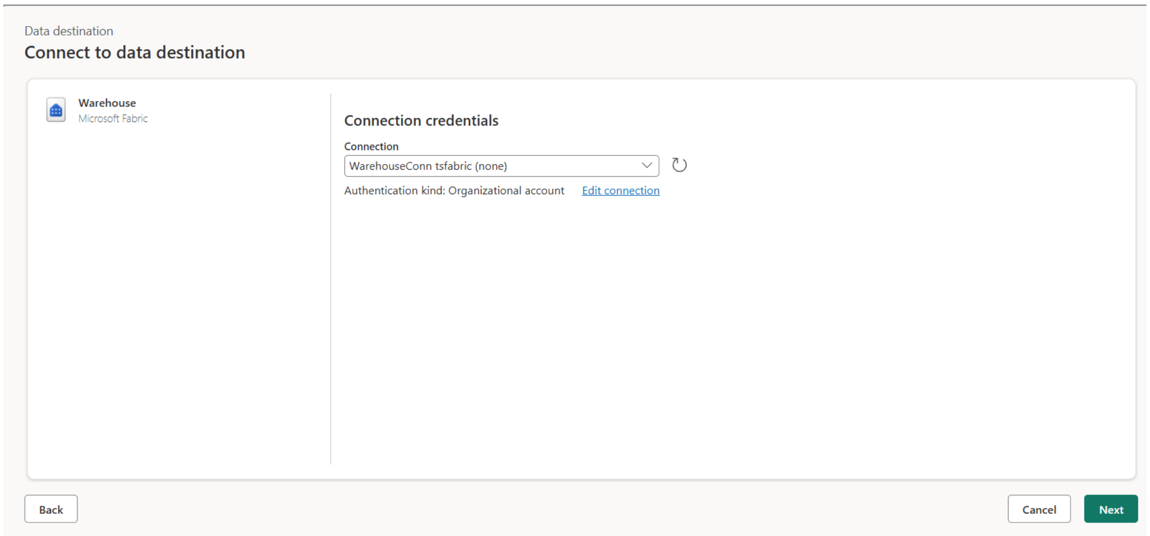


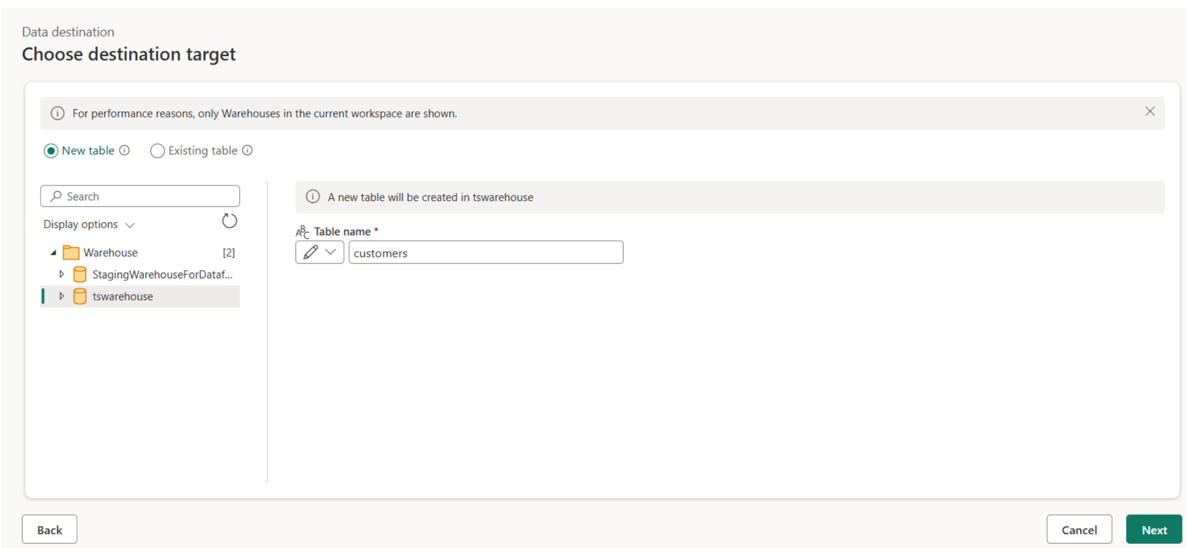
Taksh Shah

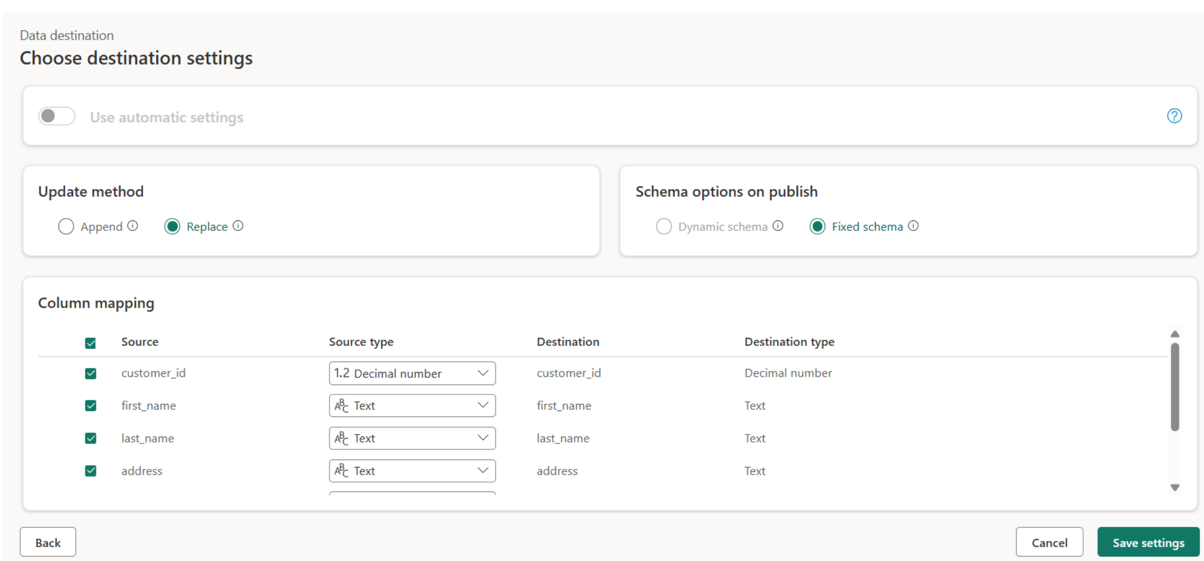



Next, we want to store the cleaned data into fabric warehouse. Follow the images provided below and do the steps for all the cleaned tables.




Data destination
Connect to data destination
Warehouse
Microsoft Fabric
Connection credentials
Connection
WarehouseConn tsfabric (none)
Authentication kind: Organizational account
Edit connection
Back
Cancel
Next


Data destination
Choose destination target
For performance reasons, only Warehouses in the current workspace are shown.
New table
Existing table
Search
Display options
Warehouse [2]
StagingWarehouseForDataf...
tswarehouse
A new table will be created in tswarehouse
Table name *
customers
Back
Cancel
Next


Data destination
Choose destination settings
Use automatic settings
Update method
Append
Replace
Schema options on publish
Dynamic schema
Fixed schema
Column mapping
Source
Source type
Destination
Destination type
customer_id
1.2 Decimal number
customer_id
Decimal number
first_name
Text
first_name
Text
last_name
Text
last_name
Text
address
Text
address
Text
Back
Cancel
Save settings

**Gold Layer:**

Visit: [https://learn.microsoft.com/en-us/fabric/data-engineering/spark-data-warehouse-connector?tabs=pyspark](https://learn.microsoft.com/en-us/fabric/data-engineering/spark-data-warehouse-connector?tabs=pyspark)

This website provides us the code to read data in a fabric notebook from a table/view in spark dataframe.

Now, we'll use fabric notebook to implement scdtype-1 logic on 3 tables from the cleaned data. The implementation for accounts table has been shown below – same logic can be applied to the other 2 tables of choice.

```
# Welcome to your new notebook
# Type here in the cell editor to add code!
import com.microsoft.spark.fabric
from com.microsoft.spark.fabric.Constants import Constants
```

✓ - Session ready in 10 sec 723 ms. Command executed in 383 ms by Taksh-fabric-417 on 8:41:41 PM, 5/05/25

# Accounts SCD-1

```
# Step 1: Read from Fabric Warehouse table
df = spark.read.synapsesql("tswarehouse.dbo.accounts")
#df.show()
```

✓ - Command executed in 16 sec 209 ms by Taksh-fabric-417 on 8:42:22 PM, 5/05/25

```
+-----------+-------+------------+----------+
|customer_id|balance|account_type|account_id|
+-----------+-------+------------+----------+
|         45| 1000.5|     Savings|       1.0|
|         12|2500.75|    Checking|       2.0|
|         78| 1500.0|     Savings|       3.0|
|         34|3000.25|    Checking|       4.0|
|         56|  500.0|     Savings|       5.0|
```

```
create_table_query = """
CREATE TABLE if not exists Account_SCD (
    account_id int,
    customer_id int,
    account_type STRING,
    balance FLOAT,
    hash_key BIGINT,
    created_by STRING,
    created_date TIMESTAMP,
    updated_by STRING,
    updated_date TIMESTAMP
)
USING DELTA
LOCATION 'Files/Gold_layer/Account_SCD'
"""

# Execute
spark.sql(create_table_query)
```

✓ - Command executed in 13 sec 946 ms by Taksh-fabric-417 on 8:42:51 PM, 5/05/25

```
DataFrame[]
```

```
from pyspark.sql.functions import *
df_src1= df.withColumn("hash_key",crc32(concat(*df.columns)))
#display(df_src1)
```

✓ - Command executed in 2 sec 357 ms by Taksh-fabric-417 on 8:43:06 PM, 5/05/25

Table | New chart

Table view

| | 123 customer... | 1.2 balance | ABC account_... | 1.2 account_id | 12L hash_key |
|---|---|---|---|---|---|
| 1 | 45 | 1000.5 | Savings | 1.0 | 1081396551 |
| 2 | 12 | 2500.75 | Checking | 2.0 | 1322625823 |
| 3 | 78 | 1500.0 | Savings | 3.0 | 4179129744 |
| 4 | 34 | 3000.25 | Checking | 4.0 | 463029068 |
| 5 | 56 | 500.0 | Savings | 5.0 | 787317497 |
| 6 | 23 | 1200.5 | Checking | 6.0 | 2250105050 |

```
from delta.tables import DeltaTable

target_path = "Files/Gold_layer/Account_SCD"
delta_target = DeltaTable.forPath(spark, target_path)
#delta_target.toDF().show()
```

✓ - Command executed in 4 sec 838 ms by Taksh-fabric-417 on 8:44:18 PM, 5/05/25

```
+----------+-----------+------------+-------+--------+----------+------------+----------+------------+
|account_id|customer_id|account_type|balance|hash_key|created_by|created_date|updated_by|updated_date|
+----------+-----------+------------+-------+--------+----------+------------+----------+------------+
+----------+-----------+------------+-------+--------+----------+------------+----------+------------+
```

```
df_src1=df_src1.alias("src").join(delta_target.toDF().alias("tgt"),((col("src.account_id")==c
#df_src1.show()
```

✓ - Command executed in 2 sec 502 ms by Taksh-fabric-417 on 8:45:24 PM, 5/05/25

```
+-----------+-------+------------+----------+----------+
|customer_id|balance|account_type|account_id|  hash_key|
+-----------+-------+------------+----------+----------+
|         45| 1000.5|     Savings|       1.0|1081396551|
|         12|2500.75|    Checking|       2.0|1322625823|
|         78| 1500.0|     Savings|       3.0|4179129744|
|         34|3000.25|    Checking|       4.0| 463029068|
|         56|  500.0|     Savings|       5.0| 787317497|
```

df_src1=df_src1.alias("src").join(delta_target.toDF().alias("tgt"),((col("src.account_id")==col("tgt .account_id"))&(col("src.hash_key")==col("tgt.hash_key"))),"anti").select(col("src.*"))

#df_src1.show()

```
from pyspark.sql.functions import col

delta_target.alias("tgt").merge(df_src1.alias("src"),"tgt.account_id = src.account_id")\
    .whenMatchedUpdate(set={"tgt.account_id":"src.account_id","tgt.customer_id":"src.customer_id","tgt.account_ty
        .whenNotMatchedInsert(values={"tgt.account_id":"src.account_id","tgt.customer_id":"src.customer_id","tgt.

#display(spark.read.format("delta").option("header","true").load(target_path))
```

✓ - Command executed in 7 sec 989 ms by Taksh-fabric-417 on 8:46:06 PM, 5/05/25

Table + New chart 9 colum

Table view Download Sea

| | 123 account_id | 123 customer... | ABC account_... | 1.2F balance | 12L hash_key | ABC created_by | created_... | ABC updated_... | updated_... |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 45 | Savings | 1000.5 | 1081396551 | databricks | 2025-05-0... | databricks | 2025-05-0... |
| 2 | 2 | 12 | Checking | 2500.75 | 1322625823 | databricks | 2025-05-0... | databricks | 2025-05-0... |
| 3 | 3 | 78 | Savings | 1500.0 | 4179129744 | databricks | 2025-05-0... | databricks | 2025-05-0... |
| 4 | 4 | 34 | Checking | 3000.25 | 463029068 | databricks | 2025-05-0... | databricks | 2025-05-0... |
| 5 | 5 | 56 | Savings | 500.0 | 787317497 | databricks | 2025-05-0... | databricks | 2025-05-0... |

from pyspark.sql.functions import col

delta_target.alias("tgt").merge(df_src1.alias("src"),"tgt.account_id = src.account_id")\

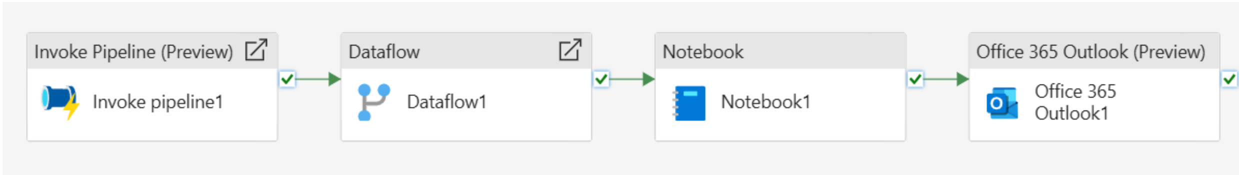
.whenMatchedUpdate(set={"tgt.account_id":"src.account_id","tgt.customer_id":"src.customer_id","tgt.account_type":"src.account_type","tgt.balance":"src.balance","tgt.hash_key":"src.hash_key","tgt.updated_date":current_timestamp(),"tgt.updated_by":lit("databricks_Updated")})\

.whenNotMatchedInsert(values={"tgt.account_id":"src.account_id","tgt.customer_id":"src.customer_id","tgt.account_type":"src.account_type","tgt.balance":"src.balance","tgt.hash_key":"src.hash_key","tgt.created_date":current_timestamp(),"tgt.created_by":lit("databricks"),"tgt.updated_date":current_timestamp(),"tgt.updated_by":lit("databricks")}).execute()

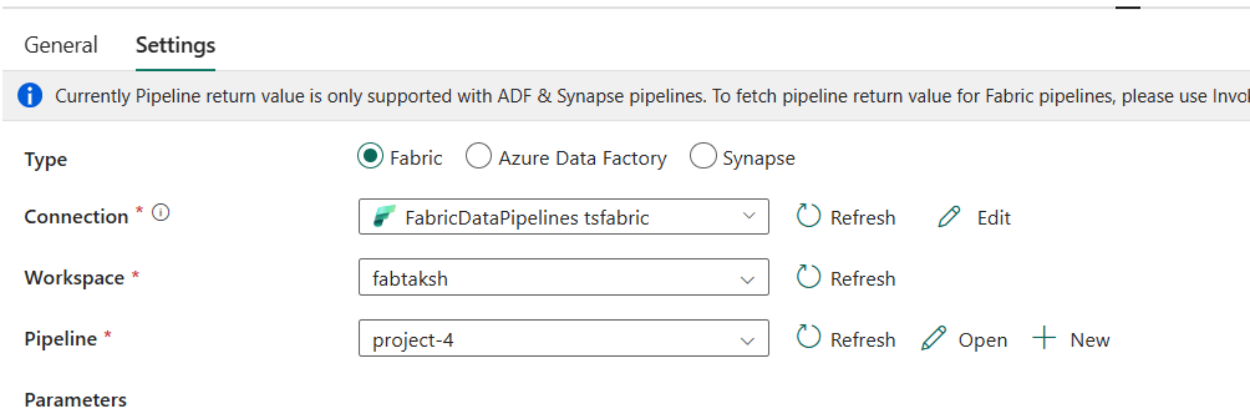
#display(spark.read.format("delta").option("header","true").load(target_path))

After coding scdtype-1 for the desired 3 cleaned data tables, test them using day1 (original dataset) and day2 (test) data.

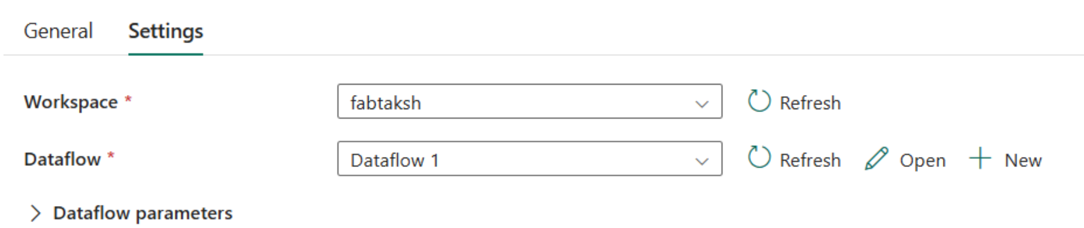
**Creating master pipeline:**

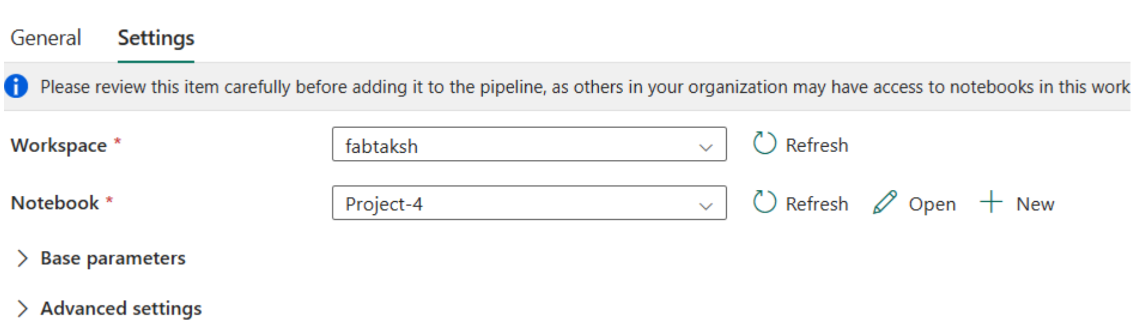



Open a new pipeline in fabric and get an invoke pipeline activity – this will invoke the pipeline which we created for bronze layer.




Similarly, dataflow activity will be used to invoke the dataflow we created for silver layer and notebook activity will invoke the notebook we created for gold layer.



Finally, as per the master pipeline image provided above, we'll connect notebook activity to an office 365 outlook (preview) activity which is basically responsible to send out email notifications upon successful runs of pipeline. First, you'll have to sign in to your outlook account and then under "settings", fill in the details as per your requirement.

Next, schedule the pipeline.