Boot Camp Project 4 Documentation

Incremental Data Loading and Automated Notifications using Microsoft Fabric

Problem Statement

In modern data ecosystems, organizations need to ingest, transform, and load data from various sources efficiently into centralized platforms such as Microsoft Fabric. This ensures high data quality, low latency, and visibility into the status of ingestion processes. The challenge is compounded when data originates from both on-premises sources and cloud storage, requiring automation, scheduling, and monitoring with minimal manual intervention.

This project solves that by:
- Implementing incremental data pipelines.
- Applying SCD Type 1 data warehouse techniques.
- Enabling automated email notifications on pipeline success.

Table of Contents

Project Overview

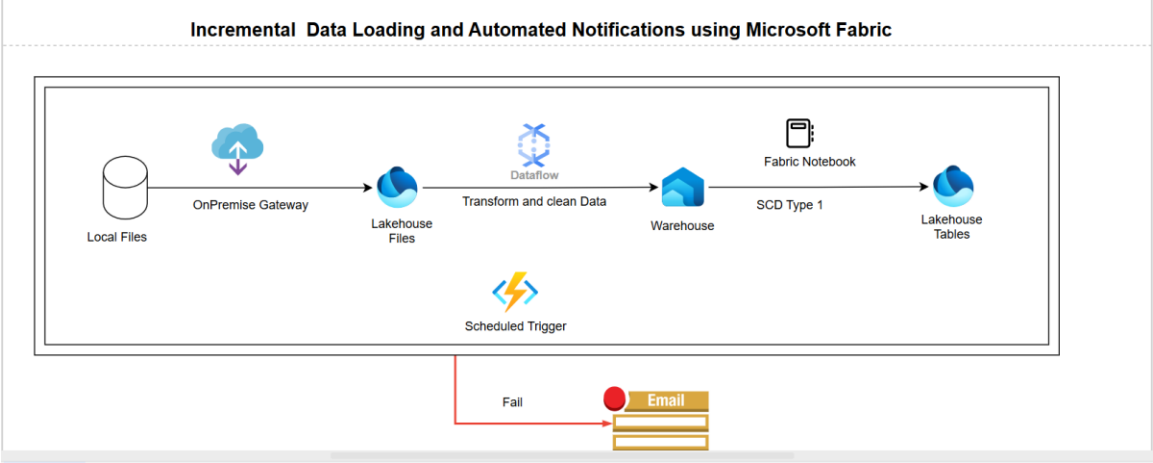
The project includes two parallel pipelines:
- Step 1: AI Bank Dataset (On-Premises via Gateway)
- Step 2–4: Sales & Sales Return Excel files (from ADLS Gen2)

Each step includes its own ingestion, transformation, loading, and notification process, with data written to Microsoft Fabric Warehouse for Power BI consumption.

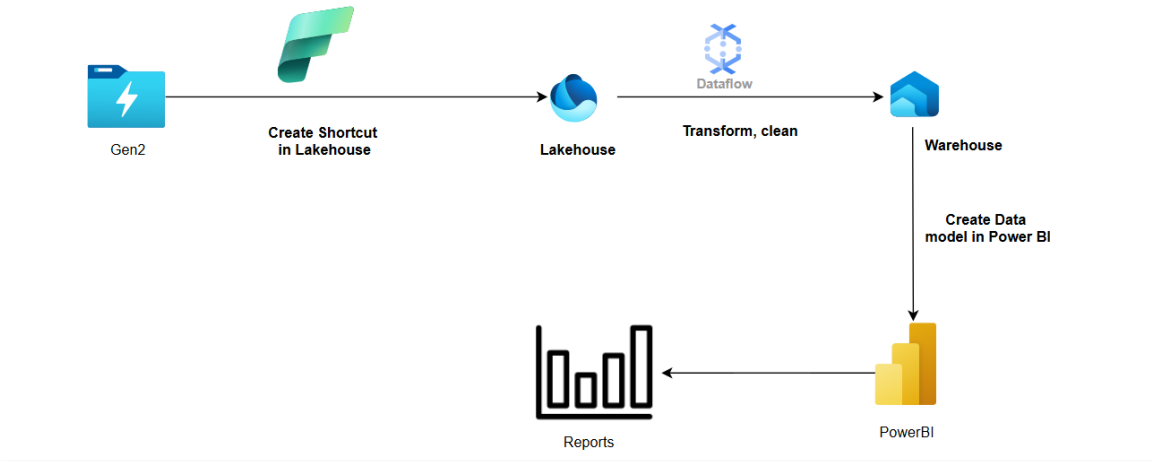
Architecture Diagram

Architecture should illustrate:

Step-1



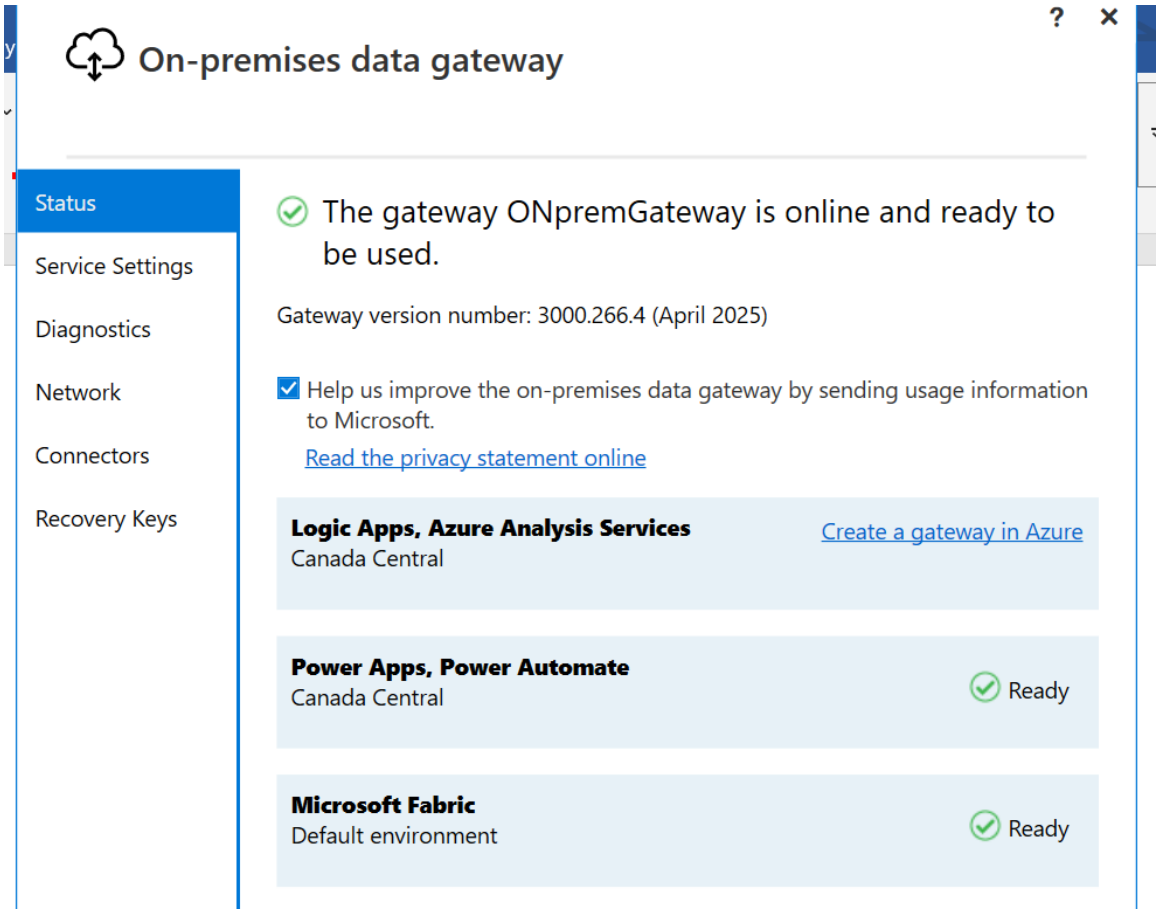
Step-2

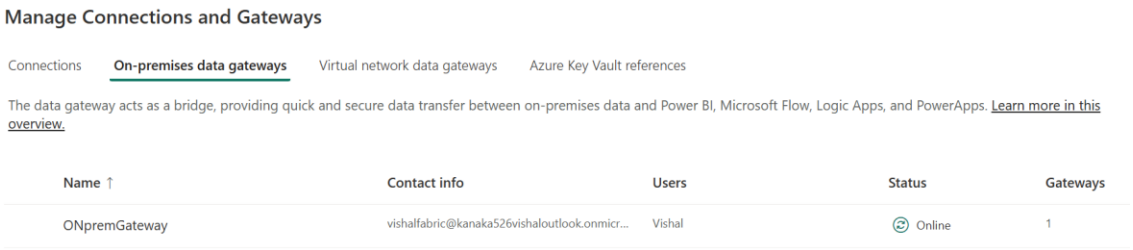


Step 1: AI Bank Dataset – On-Premise Pipeline

1.1 Ingestion

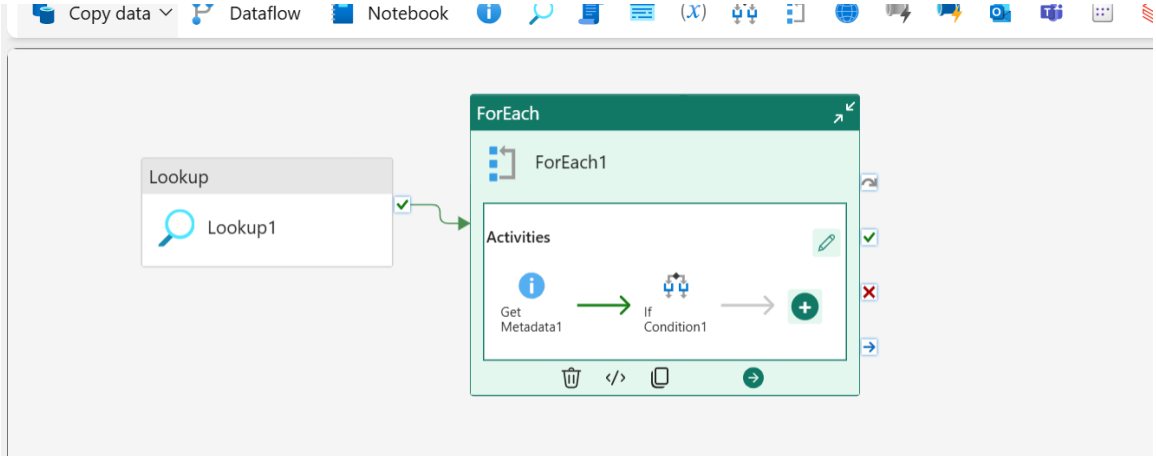
Source: AI Bank structured dataset (on-premise)
Access Method: On-Premises Data Gateway



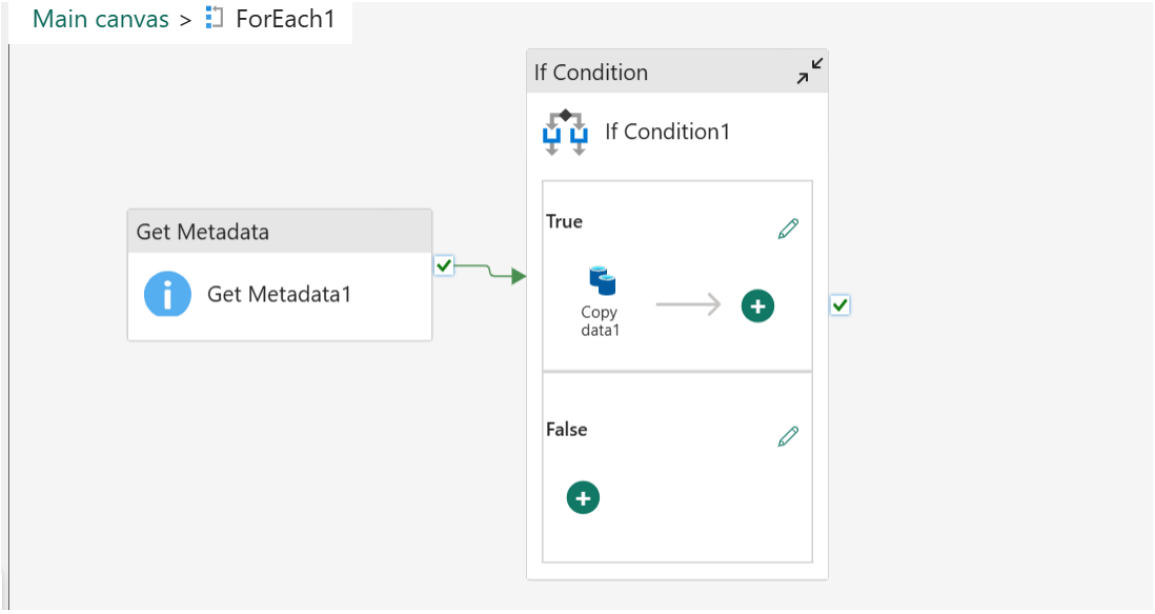


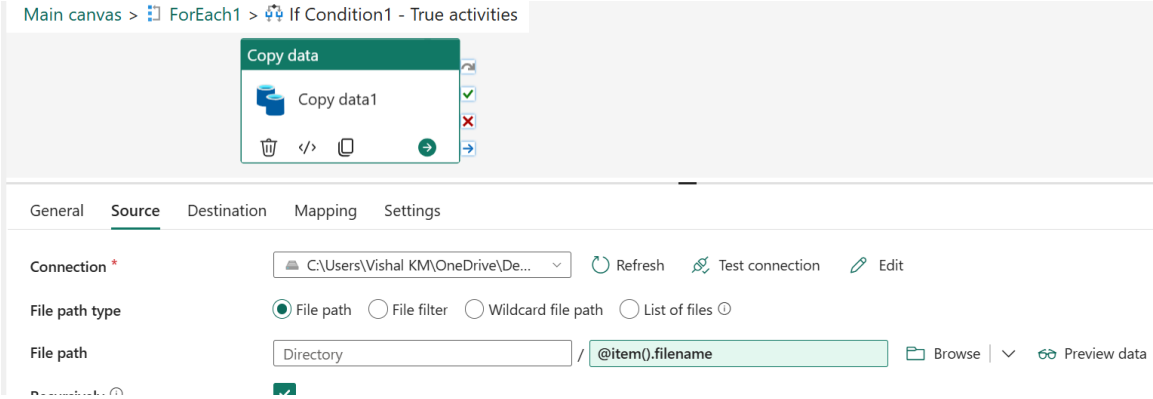
Target: Fabric Lakehouse

Pipeline

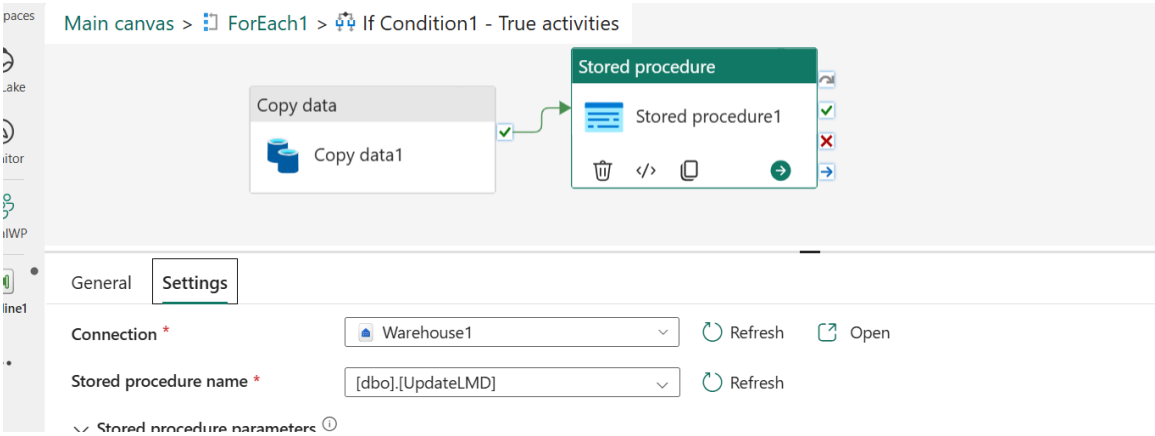


Inside forloop

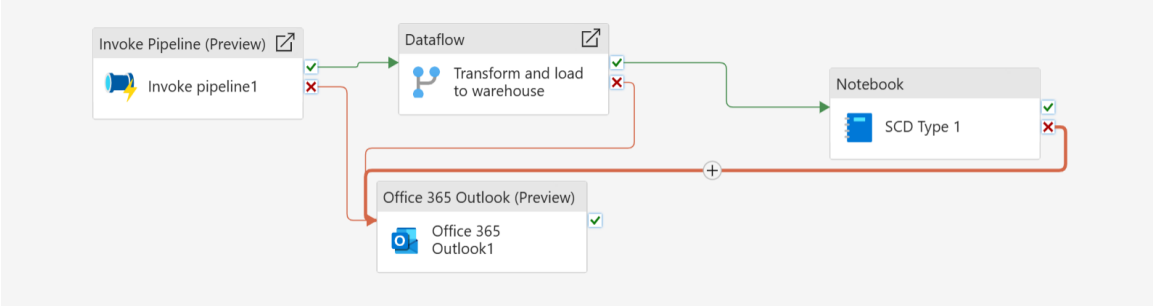




Stored procedure to update water mark table



Final Pipleine for Step-1

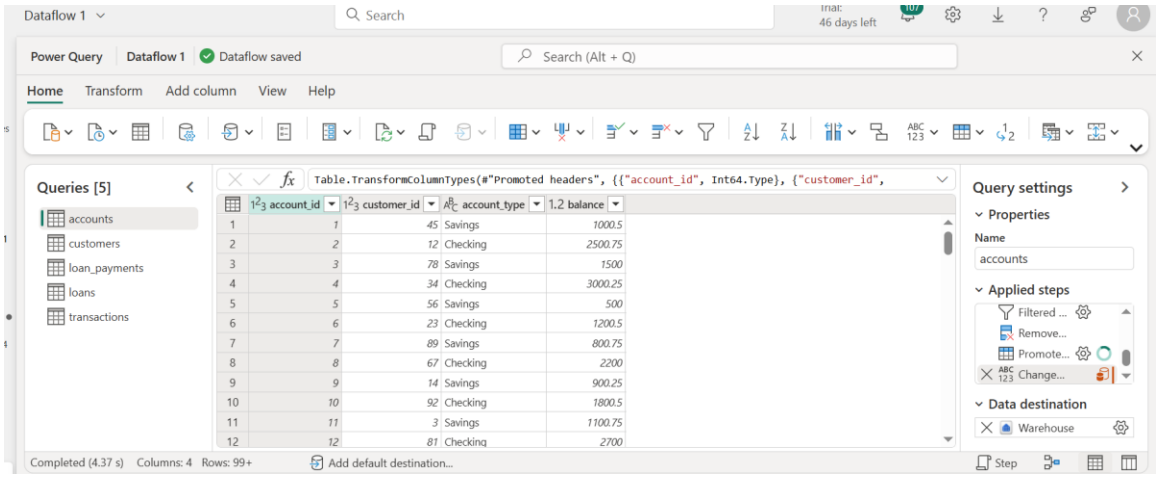


1.2 Transformation Using Dataflow Gen 2 and Store them in Warehouse tables respectively

Null filtering

Duplicate removal

Standardization and casting



1.3 SCD Type 1 Logic

Applied in Fabric Notebooks using PySpark with hashkey comparison and merge logic.

Accounts SCD type 1 Implementaion

Accounts

```
1  import com.microsoft.spark.fabric
2  from com.microsoft.spark.fabric import Constants
3  df_accounts = spark.read.table("accounts")
4  df_target_accounts = spark.read.synapsesql("Warehouse1.dbo.accounts_SCD")
5  df_accounts.show()
```

[1] ✓ - Session ready in 12 sec 66 ms. Command executed in 34 sec 774 ms by Vishal on 10:28:51 PM, 5/03/25

```
+----------+-----------+------------+-------+
|account_id|customer_id|account_type|balance|
+----------+-----------+------------+-------+
|         1|         45|     Savings| 1000.5|
|         2|         12|    Checking|2500.75|
```

Create staging delta table in lakehouse

```
1  %%sql
2  create table if not exists accounts_SCD1
3  (
4    account_id int,
5     customer_id int,
6     account_type string,
7     balance double,
8     hashkey bigint,
9     createdby string,
10    createdDate timestamp,
11    updatedby string,
12    updatedDate timestamp
13 )
14 using delta
15 location 'Tables/accounts_SCD1'
```

✓ - Command executed in 4 sec 639 ms by Vishal on 10:36:54 PM, 5/03/25

Compare for new and updated records

```
from pyspark.sql.functions import crc32, concat
df_accounts_hash=df_accounts.withColumn("hashkey",crc32(concat(*df_accounts.columns)))
```

- Command executed in 317 ms by Vishal on 10:28:57 PM, 5/03/25

```
from pyspark.sql.functions import col

df_src_accounts = df_accounts_hash.alias("src").join(
    df_target_accounts.alias("tgt"),
    (col("src.account_id") == col("tgt.account_id")) & (col("src.hashkey") == col("tgt.hashkey")),
    "anti"
).select("src.*")
```

- Command executed in 311 ms by Vishal on 10:29:03 PM, 5/03/25

```
from delta.tables import *
dtable_accounts = DeltaTable.forPath(spark, "Tables/accounts_SCD1")
dtable_accounts.toDF().show()
```

- Command executed in 3 sec 629 ms by Vishal on 10:44:42 PM, 5/03/25

```
+----------+-----------+------------+-------+-------+---------+-----------+---------+-----------+
|account_id|customer_id|account_type|balance|hashkey|createdby|createdDate|updatedby|updatedDate|
+----------+-----------+------------+-------+-------+---------+-----------+---------+-----------+
+----------+-----------+------------+-------+-------+---------+-----------+---------+-----------+
```

```
from pyspark.sql.functions import col

df_src_accounts = df_accounts_hash.alias("src").join(
    dtable_accounts.toDF().alias("tgt"),
    (col("src.account_id") == col("tgt.account_id")) & (col("src.hashkey") == col("tgt.hashkey")),
    "anti"
).select("src.*")
```

Perform Merge/Upsert

```

from pyspark.sql.functions import *
dtable_accounts.alias("tgt").merge(df_src_accounts.alias("src"),((col("src.account_id") == col("tgt.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.hashkey":"src.hashkey",
        "tgt.updatedDate":current_timestamp(),
        "tgt.updatedby":lit("databricks-update")
    })\
        .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.hashkey":"src.hashkey",
        "tgt.createdDate":current_timestamp(),
        "tgt.createdby":lit("databricks"),
        "tgt.updatedDate":current_timestamp(),
        "tgt.updatedBy":lit("databricks")
        }).execute()
```

Overwrite the warehouse target final table with staging delta table

```
1  df=spark.read.table("accounts_scd1")
```

✓ - Command executed in 1 sec 655 ms by Vishal on 10:46:40 PM, 5/03/25

```
1  import com.microsoft.spark.fabric
2  from com.microsoft.spark.fabric import Constants
```

✓ - Command executed in 367 ms by Vishal on 10:49:01 PM, 5/03/25

```
1  df.write.mode("overwrite").synapsesql("Warehouse1.dbo.accounts_SCD")
```

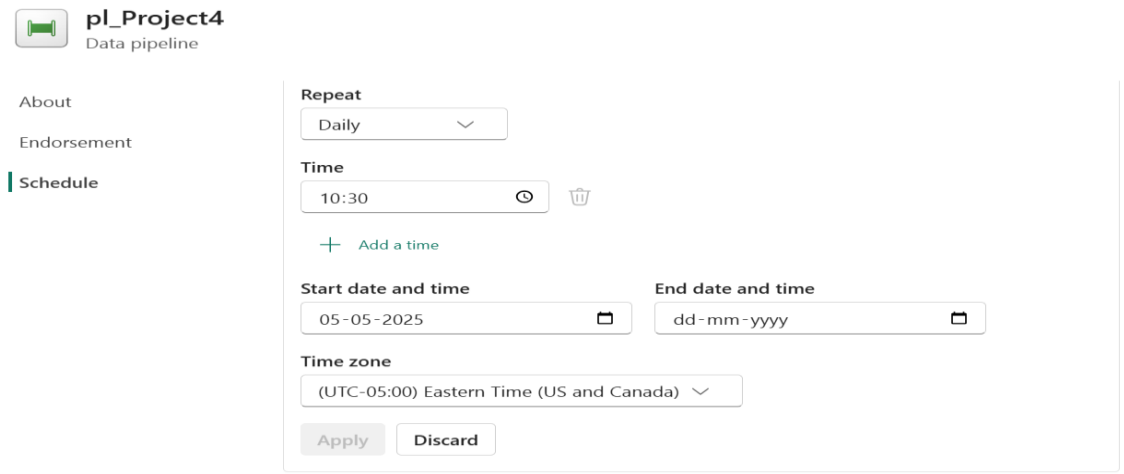
✓ - Command executed in 4 sec 882 ms by Vishal on 10:50:28 PM, 5/03/25

Similarly for all the Other Tables

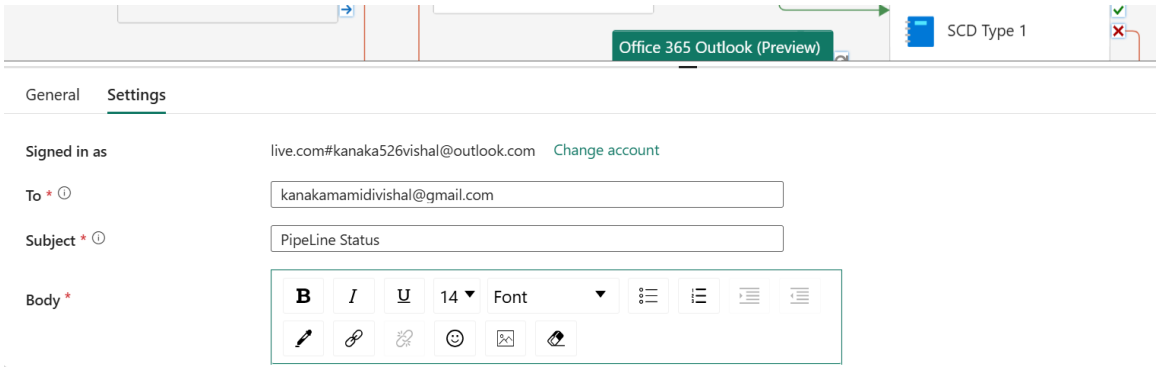
1.4 Scheduling and Notification

Fabric Pipeline triggers and sends automated email notifications via Outlook or Gmail.

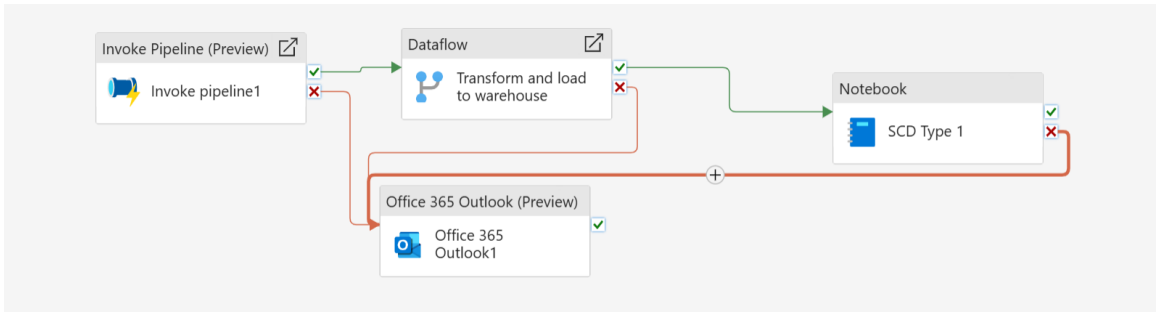
Schedule Trigger



Email Notification for Pipeline Fail



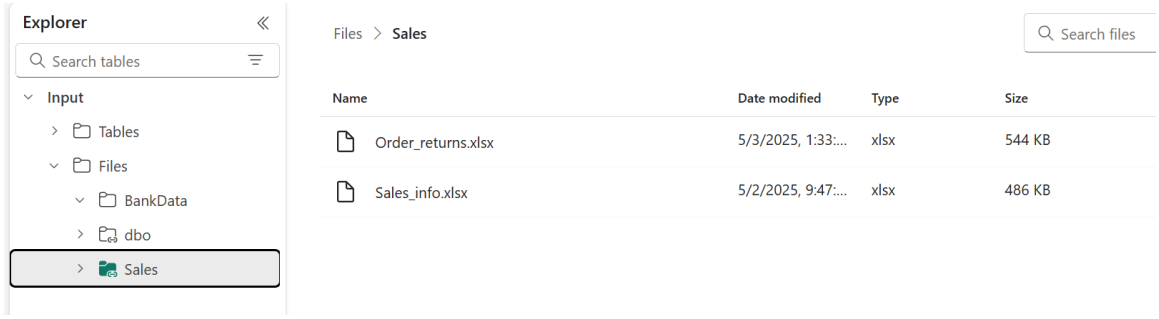
Final Pipeline



Step 2: Sales Dataset, Sales Return Dataset – ADLS Gen2

Source: Excel file from ADLS Gen2
Transformations: Deduplication, null value handling, type casting (Fabric Notebook)

Create a shortcut



Step 3: Filter Nulls generate Dimension tables – Notebook

Source: Excel file from ADLS Gen2
Transformations: Remove duplicates, filter nulls, cast columns
Loading: Single dimension table (e.g., dim_sales_return) written to Lakehouse/Warehouse.

Read raw data from excel Sales_Info

```
import pandas as pd

pdf = pd.read_excel("/lakehouse/default/Files/Sales/Sales_info.xlsx")

df_raw = spark.createDataFrame(pdf)


```

✓ - Session ready in 12 sec 118 ms. Command executed in 8 sec 383 ms by Vishal on 7:43:50 PM, 5/03/25

Split it into dimension dataframes and dropping duplicates and dropping nulls

```
from pyspark.sql.functions import monotonically_increasing_id

# dim_orders
df_orders = df_raw.select("Order_ID", "Order_Date", "Shipping_Date", "Aging", "Ship_Mode", "Order_Priority").dropDuplicates()

# dim_products
df_products = df_raw.select("Product", "Product_Category").dropDuplicates().withColumn("Product_ID", monotonically_increasing

# dim_customers
df_customers = df_raw.select("Customer_ID", "Customer_Name", "Segment", "City", "State", "Country", "Region").dropDuplicates(

```

✓ - Command executed in 325 ms by Vishal on 7:44:00 PM, 5/03/25 PySpark (Python)

Read Order_returns Excel

```
import pandas as pd

pdf1 = pd.read_excel("/lakehouse/default/Files/Sales/Order_returns.xlsx", sheet_name="Returns")
df_order_returns = spark.createDataFrame(pdf1)
df_order_returns = df_order_returns.dropDuplicates()
df_order_returns.show()
```

✓ - Command executed in 1 sec 519 ms by Vishal on 7:43:55 PM, 5/03/25

```
+----------+-----------------+------+------------+
|  Order_ID|    Customer_Name|Return|Sales_Amount|
```

Join raw and Products Dataframe to add productid column which is generated

```
df_enriched = df_raw.join(df_products, on=["Product", "Product_Category"], how="left")

```

✓ - Command executed in 325 ms by Vishal on 7:44:03 PM, 5/03/25

Now generate a Dataframe for the Fact Table

```
df_fact_sales = df_enriched.select(
    "Order_ID",
    "Customer_ID",
    "Product_ID",
    "Quantity",
    "Sales",
    "Discount",
    "Profit",
    "Shipping_Cost"
)
```

✓ - Command executed in 294 ms by Vishal on 7:44:06 PM, 5/03/25

Write data into respective Warehouse tables using Data-Frames

```
import com.microsoft.spark.fabric
from com.microsoft.spark.fabric import Constants
df_products.write.mode("overwrite").synapsesql("Warehouse1.dbo.dim_Products")
```

```
df_orders.write.mode("overwrite").synapsesql("Warehouse1.dbo.dim_Orders")
```

✓ - Command executed in 11 sec 772 ms by Vishal on 7:44:36 PM, 5/03/25

```
df_customers.write.mode("overwrite").synapsesql("Warehouse1.dbo.dim_Customers")
```

✓ - Command executed in 4 sec 787 ms by Vishal on 7:44:44 PM, 5/03/25

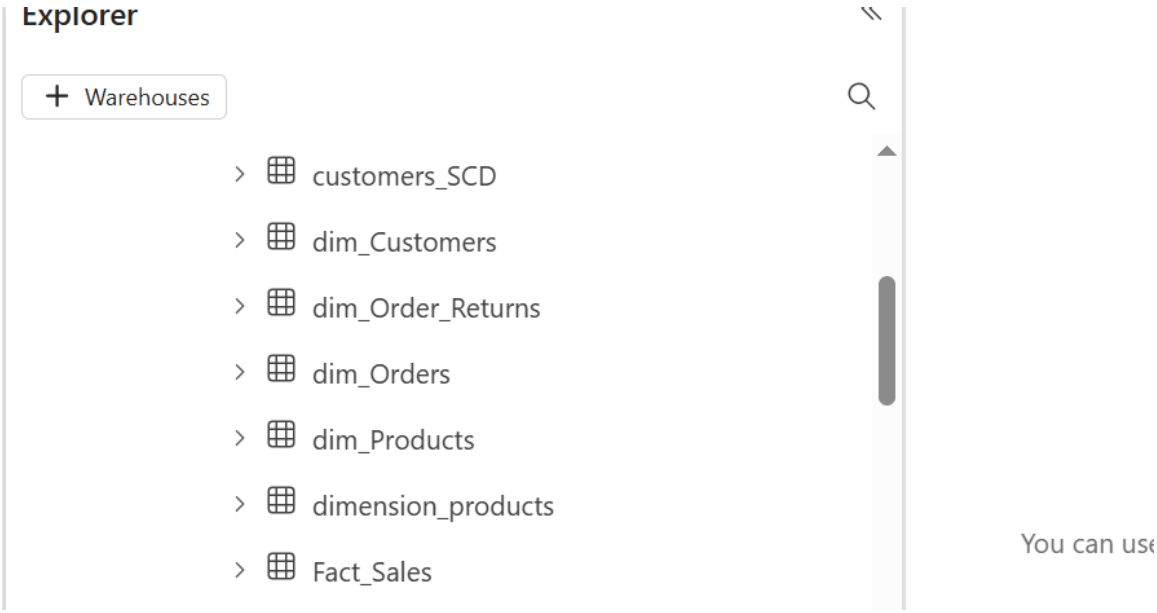
```
df_order_returns.write.mode("overwrite").synapsesql("Warehouse1.dbo.dim_Order_Returns")
```

✓ - Command executed in 4 sec 746 ms by Vishal on 7:44:53 PM, 5/03/25

```
df_fact_sales.write.mode("overwrite").synapsesql("Warehouse1.dbo.Fact_Sales")
```

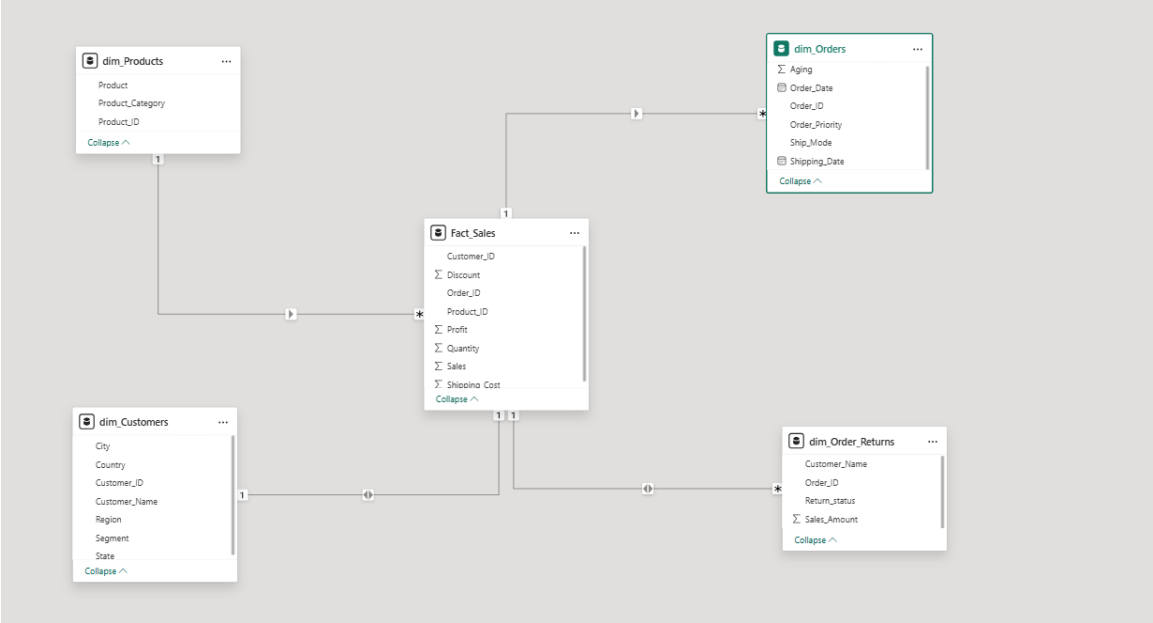
✓ - Command executed in 6 sec 187 ms by Vishal on 7:45:02 PM, 5/03/25

Now we have 5 tables 4 dimension and 1 fact table in warehouse

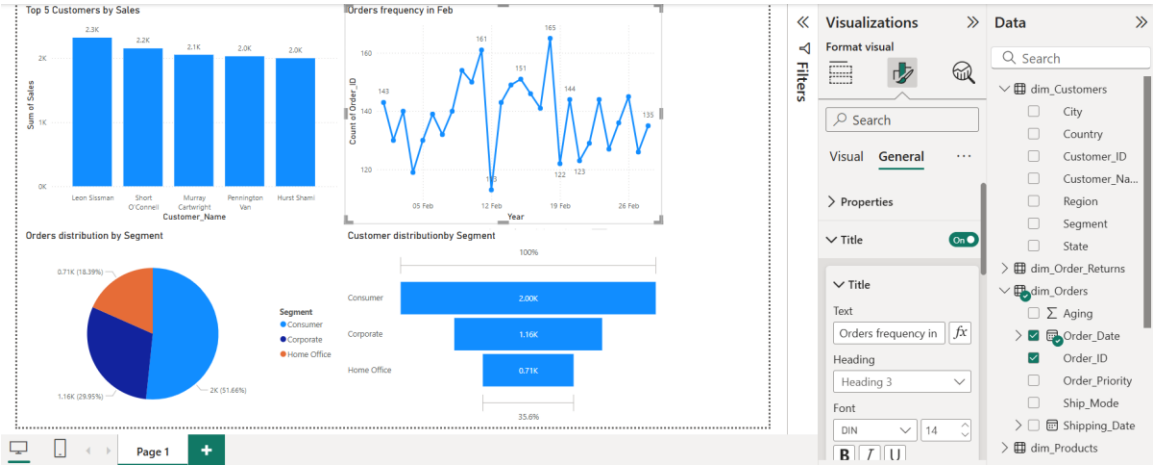


Step 4: Star Schema Data Model Generation & Power BI Integration

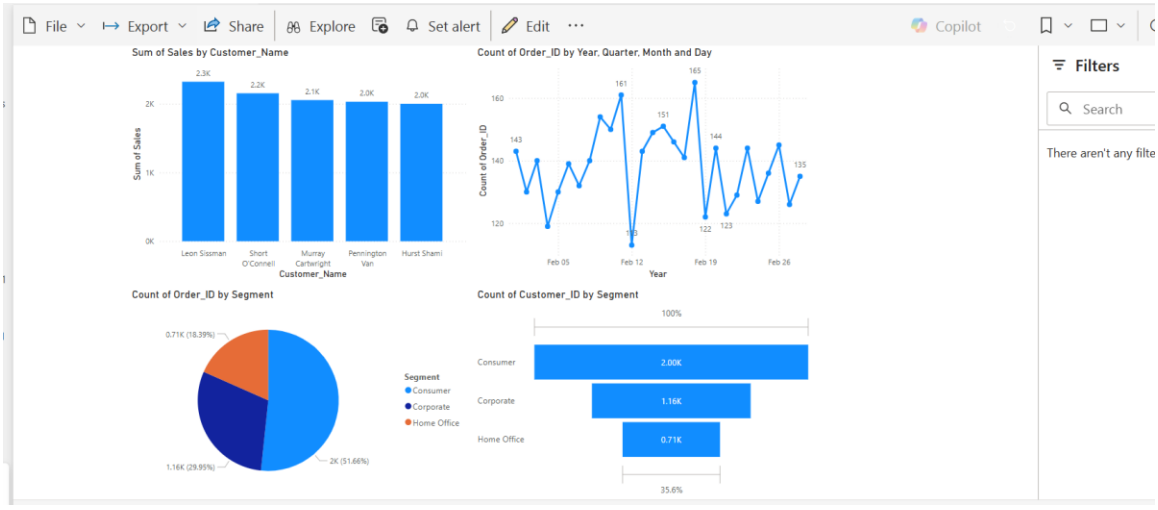
Joins and Aggregation: Dimension tables are joined to sales/sales return data using Fabric Notebooks.
Fact Table: Contains foreign keys and numeric measures.
Power BI: Power BI dashboard.



Dashboard



Publish to Fabric Workspace



Tools & Technologies

Ingestion: On-Premises Gateway, ADLS Gen2

Processing: Fabric Notebooks, Dataflow Gen 1

Storage: Fabric Lakehouse, Fabric Warehouse

Transformation: PySpark, SQL

BI & Visualization: Power BI

Documentation/Diagram: Draw.io

Deliverables

- 📄 Documentation: End-to-end explanation (this file)
- 🧠 Architecture Diagram: Visual overview of the pipeline
- 📂 GitHub Repo: Includes notebooks, SQL scripts, pipeline config
- ✅ Notification Template: Sample success email output
- 📊 Power BI Dashboard: Final visual report built on warehouse