Bootcamp Project 01 (Ramesh Allanki)

Customer Retail Sales Analysis

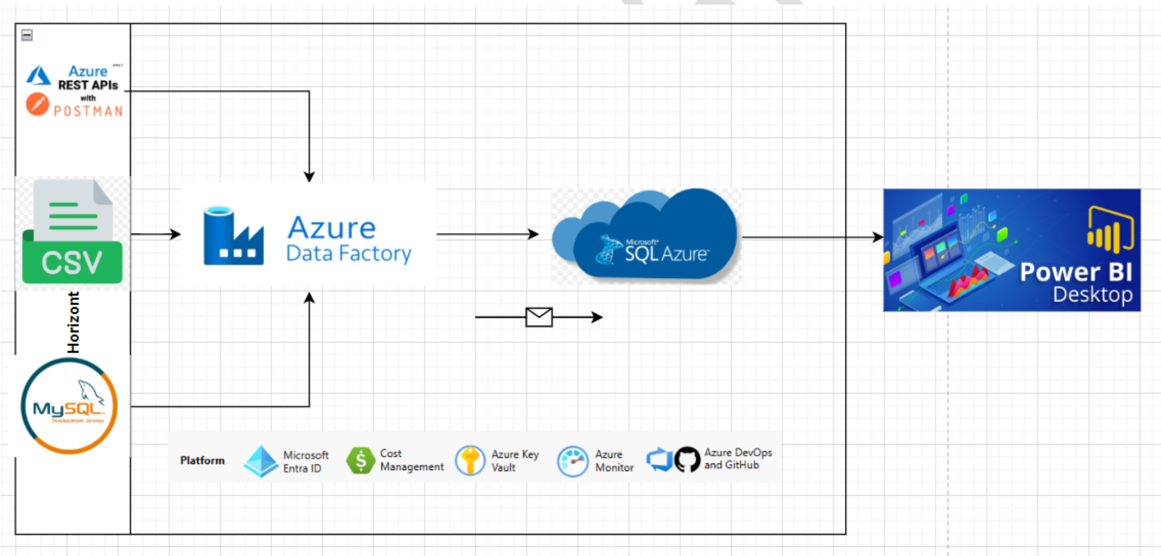
Table of Contents

Overview

The project aims to ingest, cleanse, transform, and analyze retail sales, product, and customer data from multiple sources (API, CSV, on-premise DB) into a unified data warehouse (Azure SQL), to then visualize insights via Power BI. The end goal is to enable reporting and analytics of sales trends, stock, and customer behavior in a scalable, modular pipeline.

1. Architectural Design

- I began by creating an **architecture diagram** in **draw.io** to plan the solution: how data flows, integration points, storage layers, and transformations.
    - *Why*: In real projects, having a visual architecture helps stakeholders, and reviewers understand the system design, component interactions, and responsibilities.
    - *When used*: Always used in projects before implementation — for design review, critique, scalability planning, and getting stakeholder sign-off.



Customer sales analysis Architectural Diagram

2. Data Sources & API Setup

- I collected datasets from **Kaggle(Customer, Products, Sales)**.

  https://www.kaggle.com/datasets/varunkumari/ai-shop-dataset

- I Have used **Postman** to create a mock Web API:
    - Made requests (e.g. POST) using the data
    - Created a **mock server endpoint / URL**

- Copied that URL as my API endpoint
- *Why*: Real systems often expose or consume APIs; using a mock API simulates real-time data ingestion.
- *When used*: In development, testing, or where production APIs are not yet available, mock endpoints allow you to build and test pipelines earlier.

3. Ingestion via Azure Data Factory (ADF)

Built a pipeline with:

- **Copy Data Activity** using:
  - The **API endpoint URL** (to fetch sales data)
  - CSV file(s) from local location for customer data
  - On-prem MySQL for product data
  - *Why*: ADF enables hybrid data movement (cloud, on-prem) using integration runtimes (self-hosted or Azure IR). I copied data into a unified storage (e.g. ADLS Gen2) as the **raw layer**.
    - The **Copy Activity** is optimized for data movement, with features like auto table creation and fault tolerance.
    - I have used Self-Hosted Integration Runtime(SHIR) to connect to On-Prem to ingest Database table data.
  - *Real-time usage*: Enterprises routinely integrate data from multiple sources (APIs, databases, files) into a central lake or warehouse, often using ADF.
- Stored the ingested data into **Azure Data Lake Storage Gen2** (raw zone) for staging and durability.

4. Data Cleaning & Transformation with Data Flow

- After ingesting raw data, applied **Mapping Data Flows** in ADF to clean and transform data:
  - Handle nulls or missing values (e.g., converting empty or "None" to 0 for quantity column and all other columns removed empty data)
  - Filter, join, deduplicate, type conversion, derived columns, mapping
  - *Why*: Copy alone cannot perform complex transformations (joins, aggregations, derived logic) — Data Flows apply Spark-like transformations without writing code.

- *Real-time usage*: In real ETL/ELT pipelines, the transform layer is critical: cleansing, standardization, deduplication, conforming data to canonical schema.

5. Load into Azure SQL Database (Data Warehouse)

- Moved cleaned & structured data into **Azure SQL Database** (often a "Gold" or "Curated" layer) for analytical querying and reporting.
- This forms your analytical store for dashboards and BI tools.
- *Why*: SQL DB supports relational queries, indexing, joins, performance optimizations, and is a reliable target for BI consumers.

6. Reporting & Dashboarding with Power BI

- Connected Power BI to Azure SQL, built **dashboards and visualizations** (e.g., Total Sales By Different Country and year, sales trends, Sum of quantity and total sales by category).
- *Why*: Visual insights are the end-user layer — decision makers derive insights from charts and dashboards.
- *Real-time usage*: In enterprises, data pipelines feed dashboards that are refreshed periodically (daily/hourly) for business monitoring.

7. Version Control via GitHub

- Uploaded my code, pipeline definitions, documentation, and artifacts to **GitHub**.
- *Why*: Version control ensures reproducibility, collaboration, rollback, change tracking, and transparency.
- *Real-time usage*: All professional data engineering projects maintain source control for pipelines, configurations, SQL scripts, and documentation.

Definitions and differences

Tumbling Window Trigger vs Scheduled Trigger

Feature	Tumbling Window Trigger	Scheduled Trigger
Definition	A trigger type in ADF which fires pipelines on **fixed, non-overlapping time windows**, maintaining state and enabling backfill.	A simpler trigger that fires pipelines on a fixed schedule (cron-like) without state or backfill.

Feature	Tumbling Window Trigger	Scheduled Trigger
Overlap / Windows	Windows are **non-overlapping**, contiguous blocks.	There is no concept of windows — just periodic firing.
State / Reliability	Maintains **state** per window, supports **retry** and **backfill**, ensures no gaps.	Stateless: if a run is missed (due to downtime) it is not automatically backfilled.
Use Cases	Incremental data loads (e.g. hourly, daily partitions), dependency across windows, strict coverage.	Scheduled refreshes, batch jobs that run daily/weekly without needing strict coverage.
Limitations	One-to-one mapping with a pipeline; windows cannot be extremely fine (e.g. <5 min) in some contexts.	More flexible in scheduling patterns (e.g. weekdays only), but lacks coverage guarantees.

- **Tumbling Window Trigger**: Used for API source ingestion.(API to RAW)
- **Scheduled Trigger**: Used for on-prem SQL Server ingestion.

Real-world use: For data ingestion pipelines that must process each time interval exactly once (e.g., hourly sales), a tumbling window trigger is ideal. For simpler tasks like nightly summary reports, using a scheduled trigger is usually enough.

Base Path URL vs Relative Path URL

- **Base Path URL**: The base or root part of a URL from which relative resources are resolved. E.g., https://api.example.com/v1/
- **Relative Path URL**: The portion appended or resolved relative to the base path. E.g., sales/data → full URL: https://api.example.com/v1/sales/data.

Why this matters:
When constructing API endpoints in Postman or ADF Web/Dataset connectors, We often define a base URL (host + root path) and then relative paths per request. This modular approach helps manage endpoints, versioning, and reuse.

Partitioning & Performance

- For large data volumes, partition data by time (year, month) to optimize query performance.
- Use indexes or clustered indexes in SQL target to accelerate analytic queries.

Monitoring & Alerting

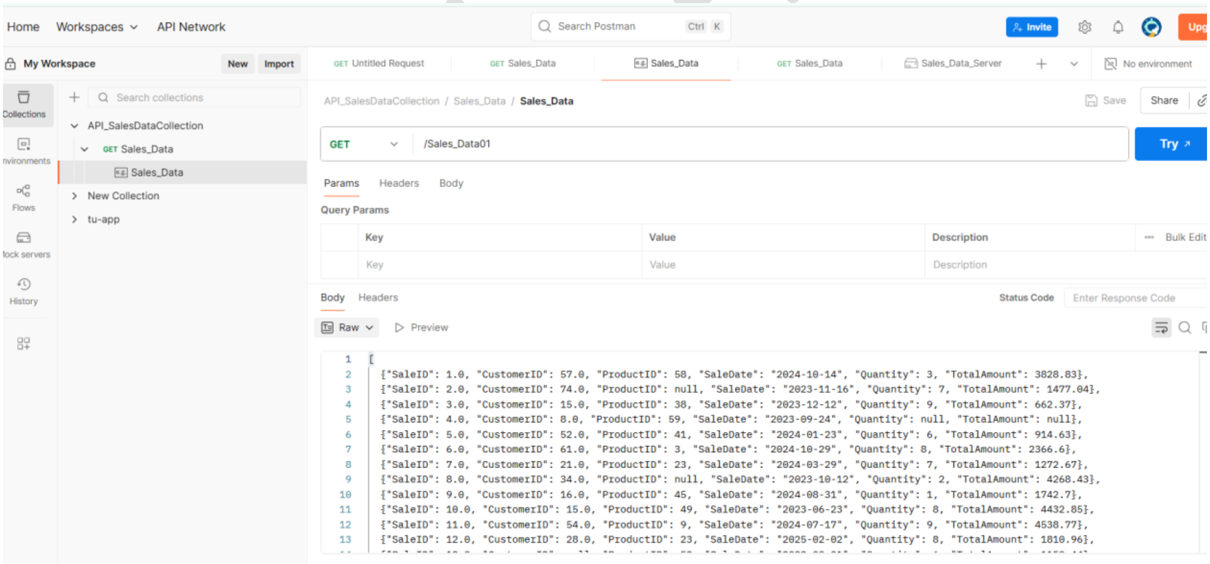
- Use ADF's monitoring features to capture pipeline success/failure statuses.
- Integrate with alerts (Azure Monitor, email, webhook) to notify you on failures.

Deployment & CI/CD

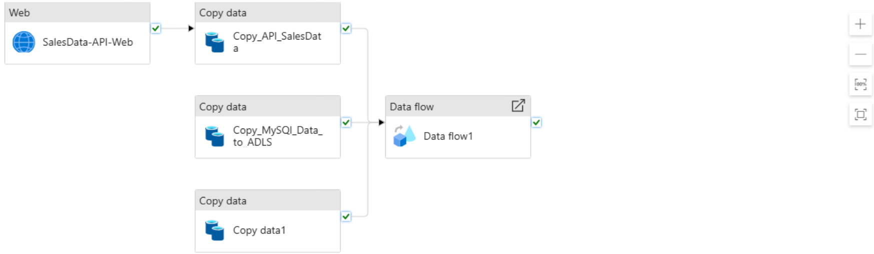
- Use ARM templates or Git integration to deploy pipelines across dev/test/prod.
- Maintain versioned releases, code reviews, and rollback options.

Here are my Screenshots of my project work:

Postman API to generate the Data and API url By creatin Mock Server:



Here is my ADF pipeline from 3 different sources:



Activity runs

Pipeline run ID efb21706-19be-405a-b959-cf44fa62045f

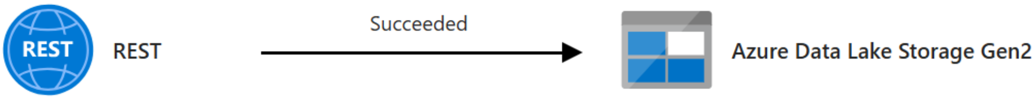
All status

Monitor in Azure Metrics | Export to CSV

Showing 1 - 5 of 5 items

Activity name	Activity status	Activity ...	Run start	Duration	Integration runtime	User proper...	Activity run ID	Log
Data flow1	Succeeded	Data flow	10/15/2025, 4:00:07 PM	1m 41s	AutoResolveIntegrationRuntime (UK South)		9a546432-468e-4d61-aa40-6fd621b6b7ca	
Copy_API_SalesData	Succeeded	Copy data	10/15/2025, 3:58:59 PM	1m 7s	AutoResolveIntegrationRuntime (UK South)		64801339-88ad-4944-8c75-f1c91df9b787	
Copy data1	Succeeded	Copy data	10/15/2025, 3:58:56 PM	1m 7s	AutoResolveIntegrationRuntime (UK South)		6f74df20-15b1-4a1c-ba4e-137da6e4e183	
Copy_MySQL_Data_to_ADLS	Succeeded	Copy data	10/15/2025, 3:58:56 PM	14s	Ram-SelfHosted-IR		e675fb96-5e8b-4527-b616-b3c5ce10e425	
SalesData-API-Web	Succeeded	Web	10/15/2025, 3:58:56 PM	3s	AutoResolveIntegrationRuntime (UK South)		ccc97be0-3c20-4076-9ca8-4c9607fb31db	

Activity run id: 2c423b62-807c-49cf-9cd7-c118f23937c9



Data read: 14.368 KB
Objects read: 90
Peak connections: 1

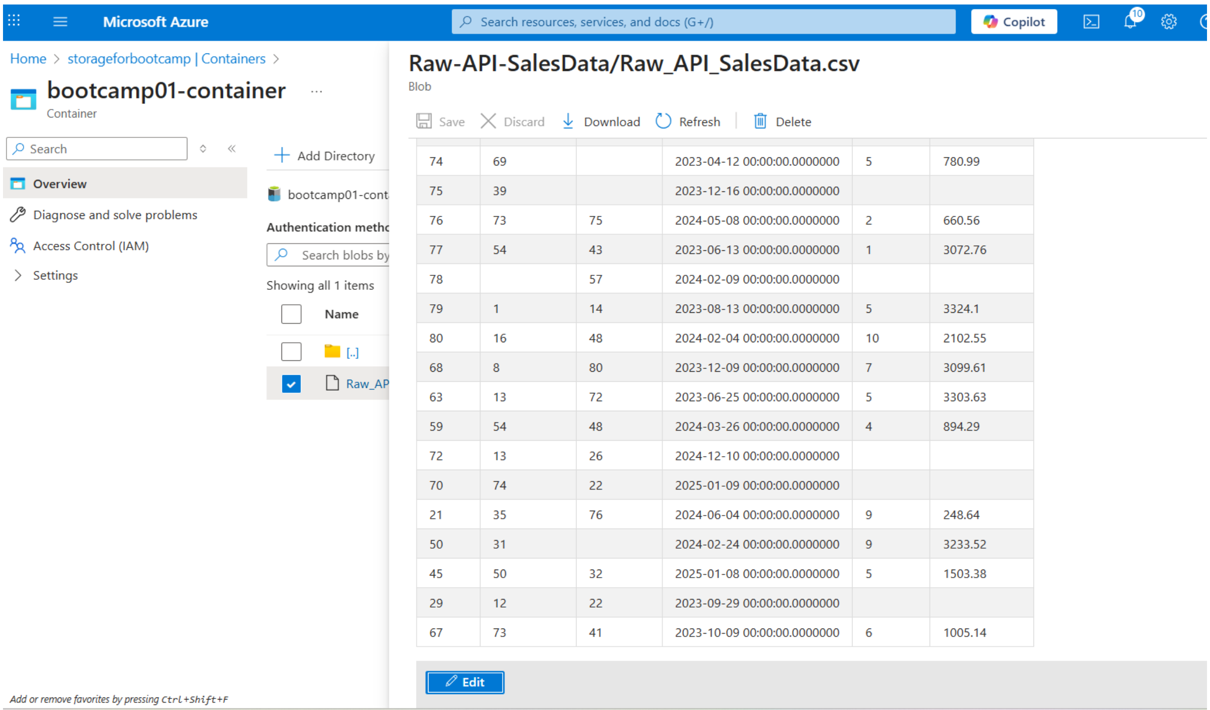
Data written: 4.133 KB
Files written: 1
Rows written: 90
Peak connections: 1

Copy duration 00:01:02
Throughput: 4.789 KB/s

REST → Azure Data Lake Storage Gen2

Start time	10/13/2025, 4:56:32 PM
Used DIUs	4
Used parallel copies	1
Duration	00:01:02

Details		Working duration	Total duration
Queue			00:00:57
Transfer	Time to first byte	00:00:00	00:00:03
	Reading from source	00:00:00	
	Writing to sink	00:00:00	



74	69		2023-04-12 00:00:00.0000000	5	780.99
75	39		2023-12-16 00:00:00.0000000		
76	73	75	2024-05-08 00:00:00.0000000	2	660.56
77	54	43	2023-06-13 00:00:00.0000000	1	3072.76
78		57	2024-02-09 00:00:00.0000000		
79	1	14	2023-08-13 00:00:00.0000000	5	3324.1
80	16	48	2024-02-04 00:00:00.0000000	10	2102.55
68	8	80	2023-12-09 00:00:00.0000000	7	3099.61
63	13	72	2023-06-25 00:00:00.0000000	5	3303.63
59	54	48	2024-03-26 00:00:00.0000000	4	894.29
72	13	26	2024-12-10 00:00:00.0000000		
70	74	22	2025-01-09 00:00:00.0000000		
21	35	76	2024-06-04 00:00:00.0000000	9	248.64
50	31		2024-02-24 00:00:00.0000000	9	3233.52
45	50	32	2025-01-08 00:00:00.0000000	5	1503.38
29	12	22	2023-09-29 00:00:00.0000000		
67	73	41	2023-10-09 00:00:00.0000000	6	1005.14

Activity run id: 46eff7f3-1351-4afc-9abb-3c14b89c0fcb

Azure Data Lake Storage Gen2 —Succeeded→ Azure Data Lake Storage Gen2

Data read:	6.214 KB	**Data written:**	7.054 KB
Files read:	1	**Files written:**	1
Rows read:	90	**Rows written:**	90
Peak connections:	1	**Peak connections:**	1

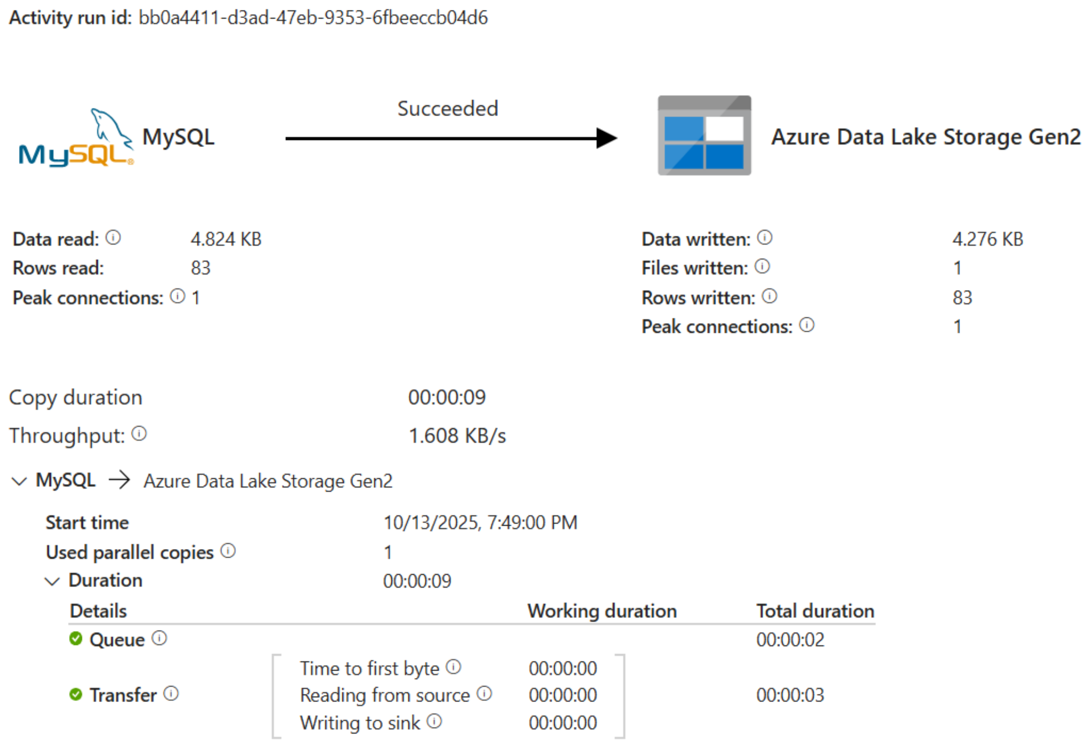
Copy duration 00:01:27

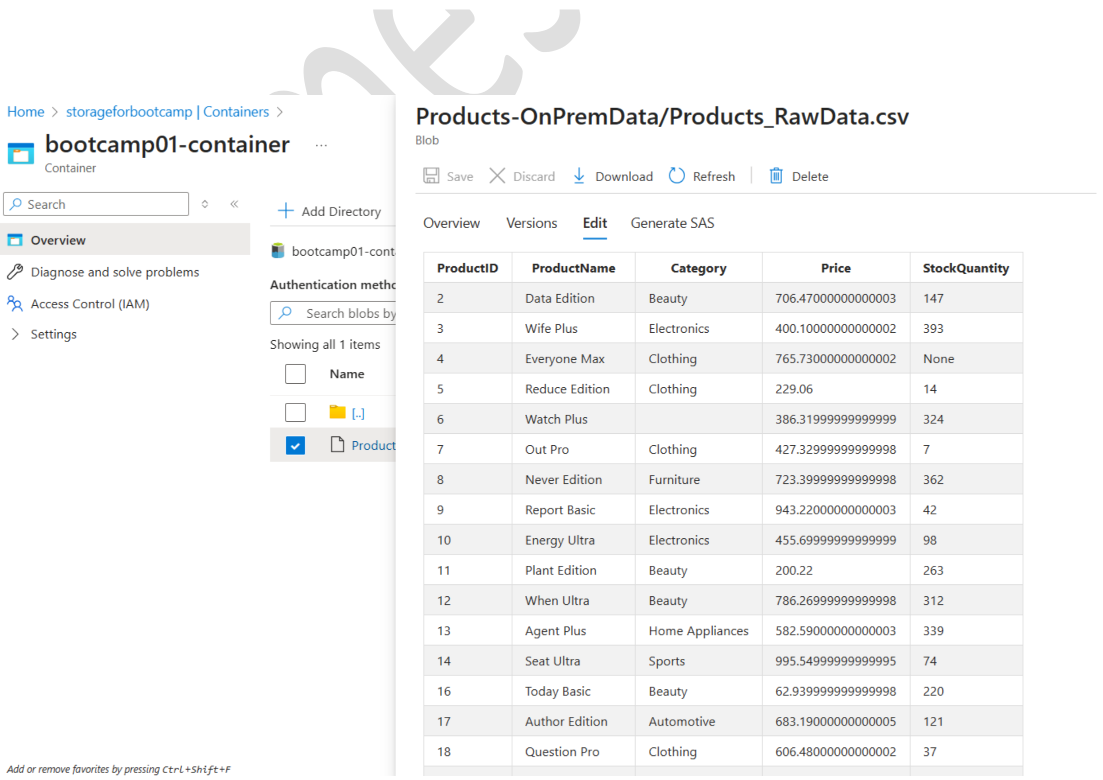
Throughput: 1.243 KB/s

Azure Data Lake Storage Gen2 → Azure Data Lake Storage Gen2

Start time	10/13/2025, 5:15:59 PM
Used DIUs	4
Used parallel copies	1
Duration	00:01:27

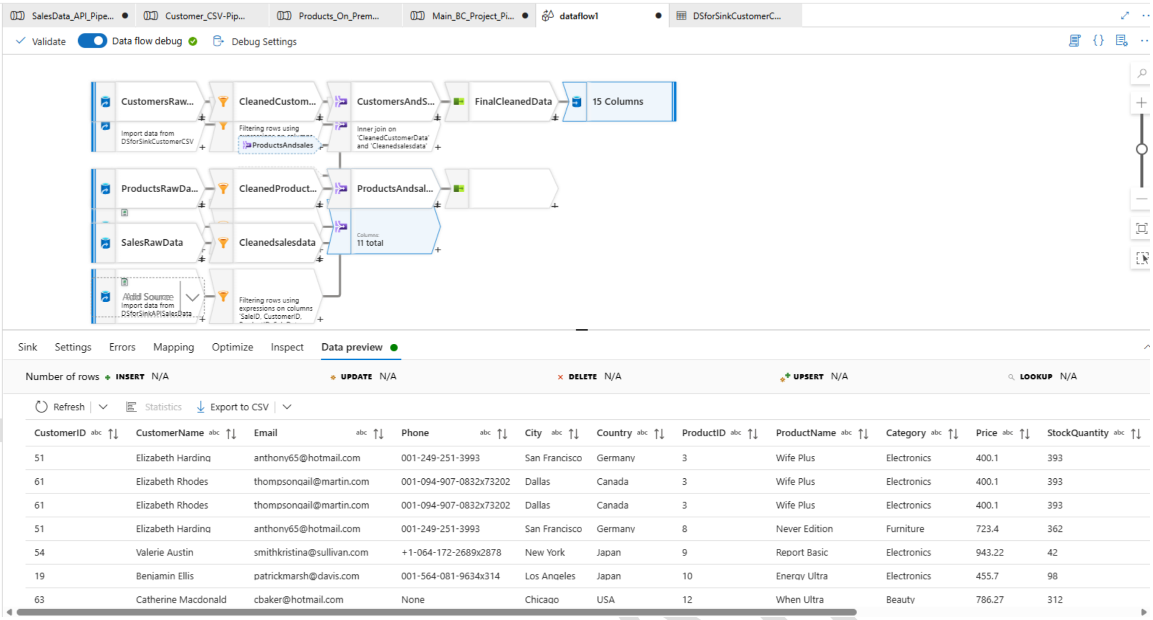
Details		Working duration	Total duration
Queue			00:01:20
Transfer	Listing source	00:00:00	00:00:05
	Reading from source	00:00:00	
	Writing to sink	00:00:00	



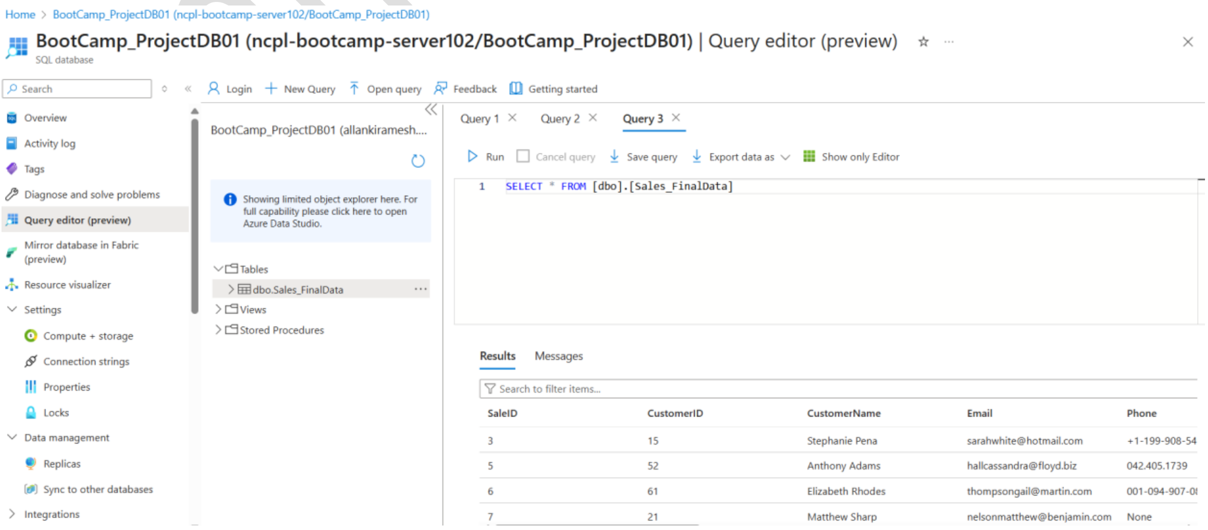


ProductID	ProductName	Category	Price	StockQuantity
2	Data Edition	Beauty	706.47000000000003	147
3	Wife Plus	Electronics	400.10000000000002	393
4	Everyone Max	Clothing	765.73000000000002	None
5	Reduce Edition	Clothing	229.06	14
6	Watch Plus		386.31999999999999	324
7	Out Pro	Clothing	427.32999999999998	7
8	Never Edition	Furniture	723.39999999999998	362
9	Report Basic	Electronics	943.22000000000003	42
10	Energy Ultra	Electronics	455.69999999999999	98
11	Plant Edition	Beauty	200.22	263
12	When Ultra	Beauty	786.26999999999998	312
13	Agent Plus	Home Appliances	582.59000000000003	339
14	Seat Ultra	Sports	995.54999999999995	74
16	Today Basic	Beauty	62.939999999999998	220
17	Author Edition	Automotive	683.19000000000005	121
18	Question Pro	Clothing	606.48000000000002	37

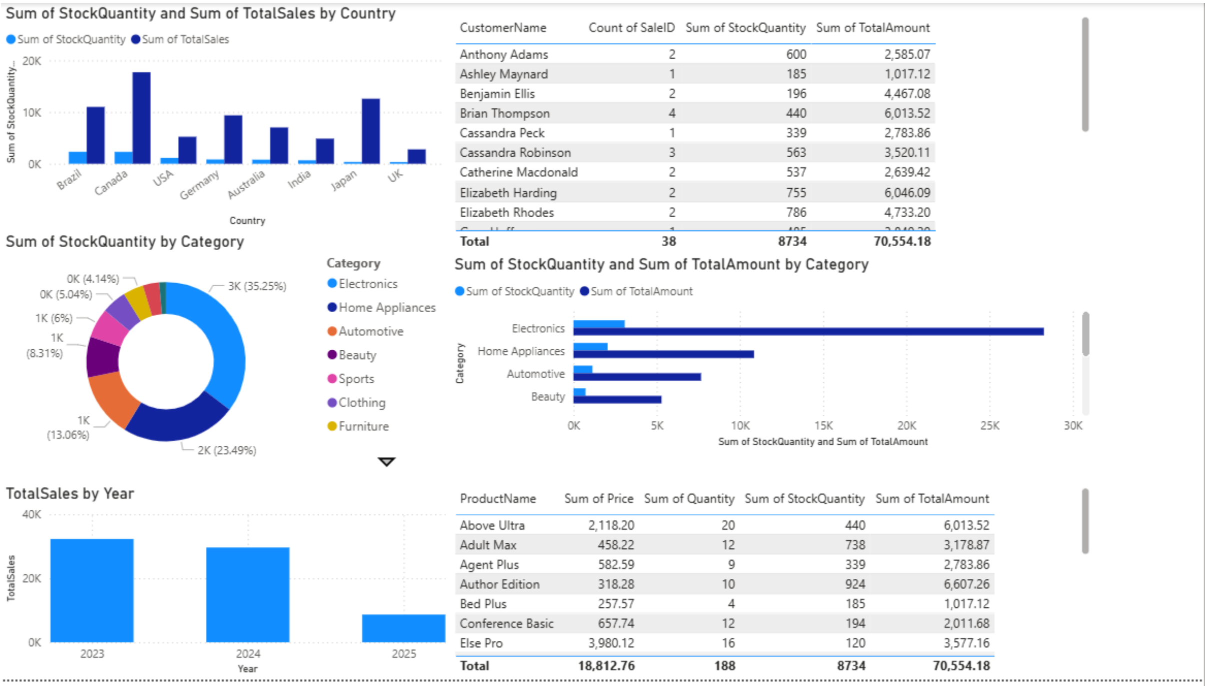
Dataflow Activity to clean and transform the Data .

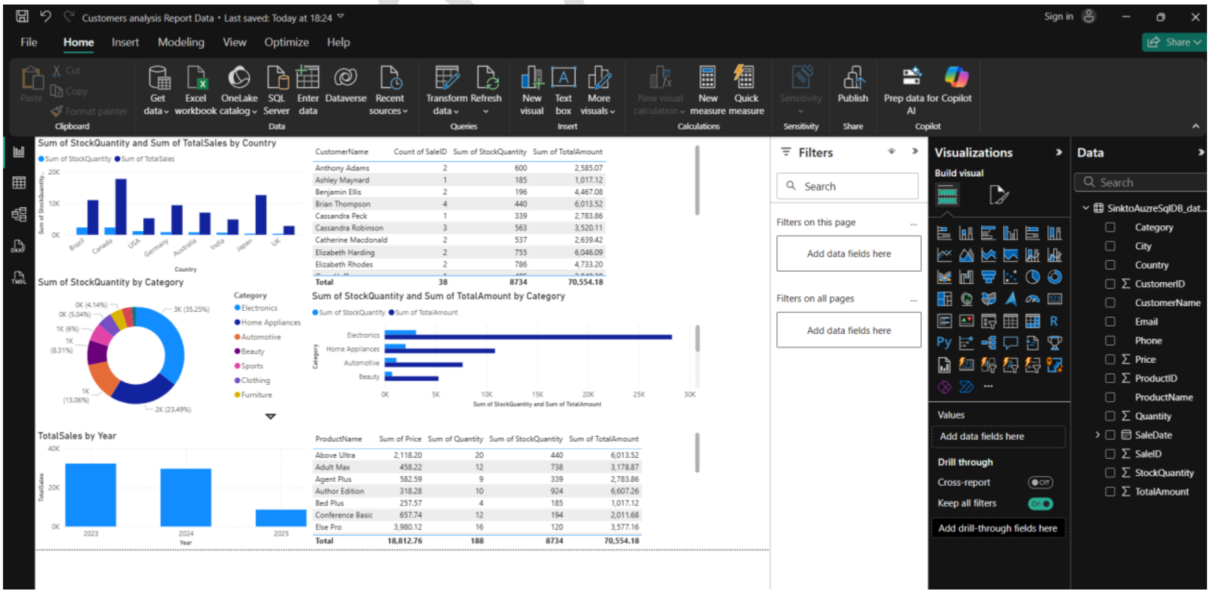


Ingested the Cleaned and transformed data into Azure Azure SQL Databse:

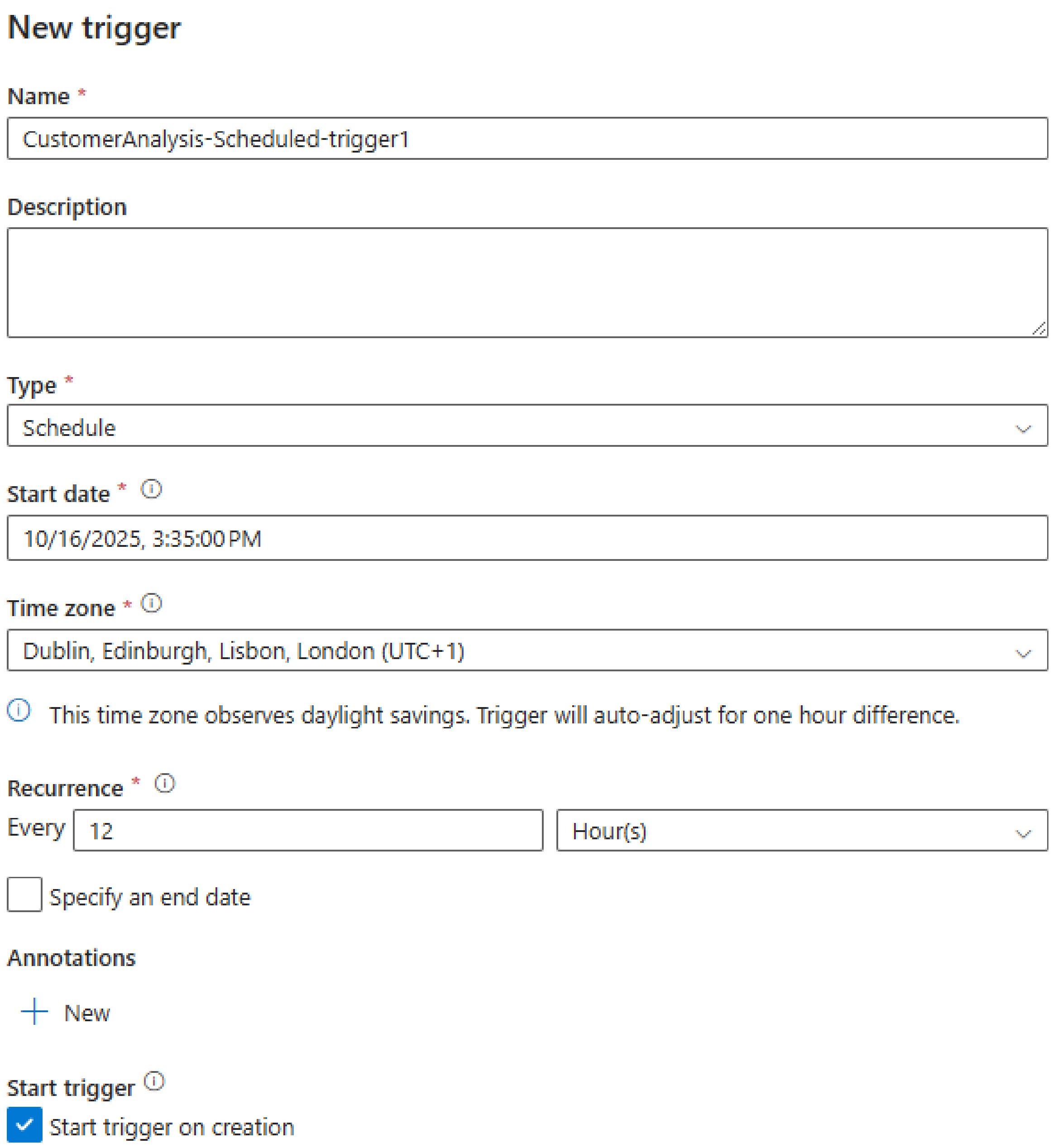


Power BI Dashbord:





Created the Triggers (Tumbling and Scheduled):



New trigger

Name *

CustomerAnalysis-Scheduled-trigger1

Description

Type *

Schedule

Start date *

10/16/2025, 3:35:00 PM

Time zone *

Dublin, Edinburgh, Lisbon, London (UTC+1)

This time zone observes daylight savings. Trigger will auto-adjust for one hour difference.

Recurrence *

Every 12 Hour(s)

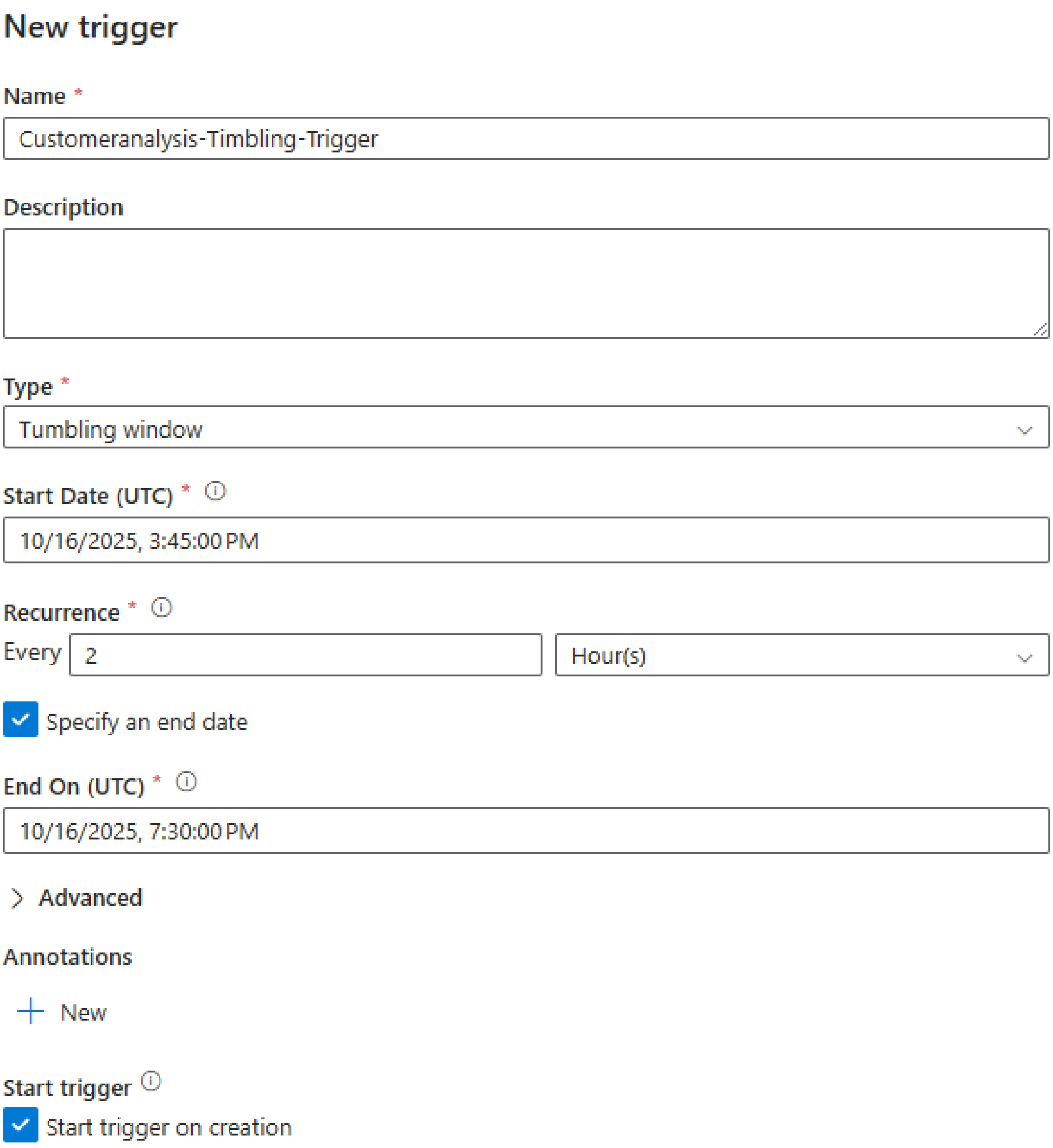
- [ ] Specify an end date

Annotations

+ New

Start trigger

- [x] Start trigger on creation



Conclusion

The **Customer Retail Sales Analysis** project demonstrates a full-scale implementation of a modern data pipeline using Azure cloud technologies and industry-standard tools. Starting from raw, unstructured, and multi-source data, the project successfully built an end-to-end pipeline that ingests, transforms, stores, and visualizes retail data for actionable insights.

My Learnings:

- Learned how to **ingest data from multiple sources** including APIs, local files, and on-prem databases using Azure Data Factory.
- Gained hands-on experience with **Data Flow transformations** for cleansing, type conversion, deduplication, and schema mapping.
- Applied **real-world practices** like using **tumbling window triggers**, mock APIs, and **version control with GitHub**.
- Built **Power BI dashboards** connected to an Azure SQL database, enabling business intelligence reporting.
- Understood the importance of designing scalable architectures, parameterizing pipelines, and applying best practices for error handling and monitoring.

Final Thought:

This project provided valuable experience in building a cloud-based data analytics pipeline from the ground up. The integration of tools like Azure Data Factory, Power BI, and Azure SQL Database reflects the **best practices followed in enterprise data engineering environments**, and this project serves as a strong foundation for more advanced data solutions in the future.