Use case: Virtual Server Monitoring and Performance Optimization

ANJALI KRISHNA

DATA ENGINEERING INTERN

MCA

Contents

[Introduction 2](#_Toc205471312)

[Data Flow Diagram 4](#_Toc205471313)

[Cleaning and Transformation Logic 5](#_Toc205471314)

[PHASES 7](#_Toc205471315)

[Scalability of the Storage Solution 19](#_Toc205471316)

[Conclusion 19](#_Toc205471317)

# Introduction

With cloud computing and digital infrastructure, server performance monitoring plays a critical role in guaranteeing operational efficiency and reliability Project aims to establish an end-to-end data pipeline for server performance metrics by utilizing Azure Data Factory, Google Colab (Python), and Power BI. Raw Azure Blob Storage data, cleaned and processed by Python, and finally visualized in Power BI for real-time monitoring and insights. Architecture Diagram

A top-level architecture for the project is:

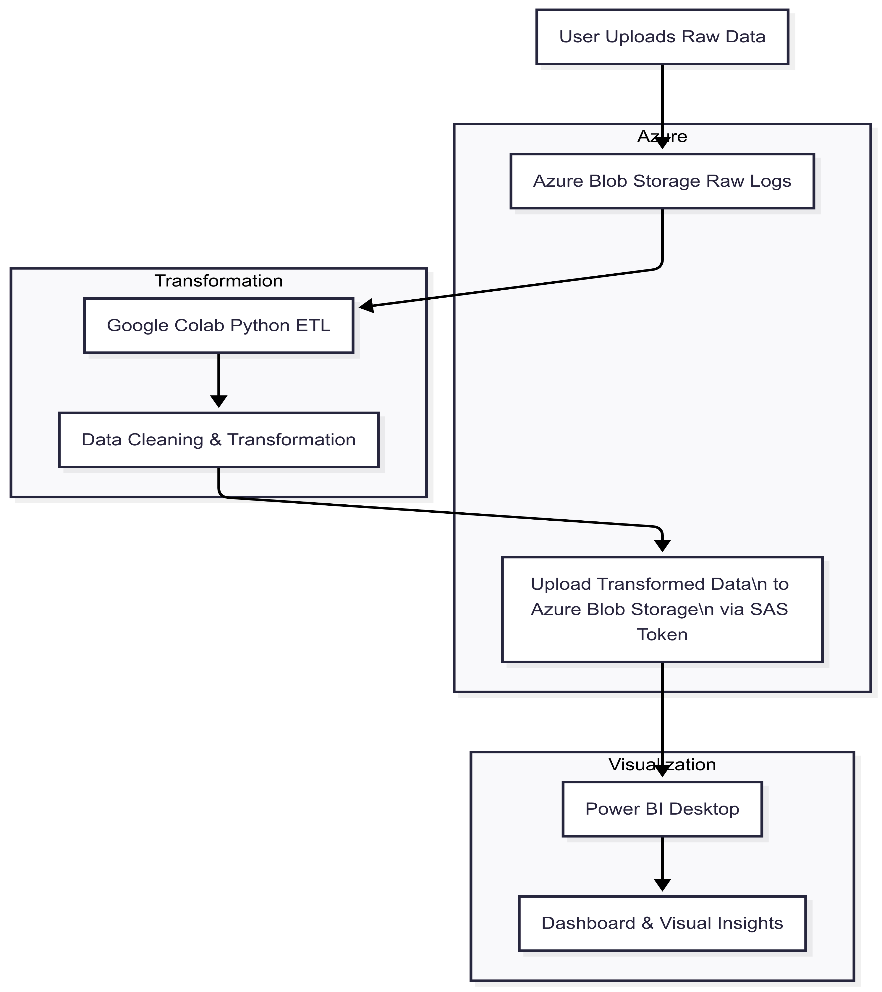
• Data Source: Raw CSV files with server performance data kept in Azure Blob Storage.

• Ingestion: Azure Data Factory pipelines to transfer data into a sink (SQL or Blob).

• Transformation: Data cleaning and enrichment done in Python (Google Colab).

• Storage: Cleaned data stored as CSVs and optionally re-uploaded to Azure Blob.

• Visualization: Power BI dashboard connected directly to cleaned data.



# Data Flow Diagram

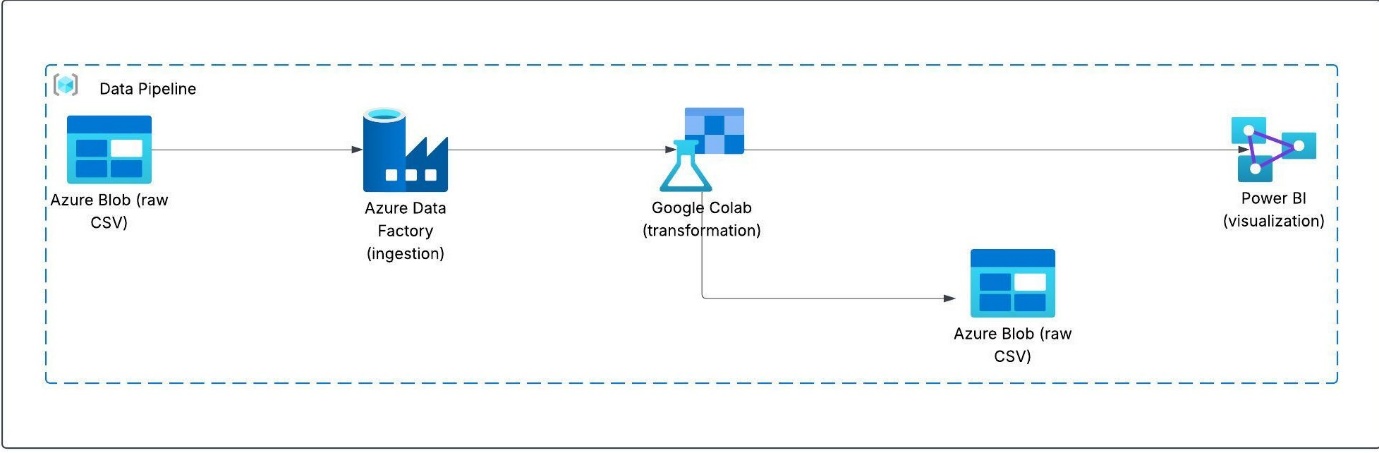
• Raw Data Upload to Azure Blob (.csv format).

• ADF ingestion (optional: sink to Azure SQL or clean Blob).

• Colab Transformation – cleaning, imputation, enrichment, station split.

• Output File Storage – local CSV and optional Azure Blob push.

• Power BI Dashboard – constructed from cleaned CSV files.



# Cleaning and Transformation Logic

Cleaning and Transformation Logic

1. Load Raw Data

* Read metadata, Station1, and Station2 CSVs from Azure Blob Storage using SAS URLs.

1. Clean Metadata

* Drop duplicate rows.
* Strip whitespace from column names.

1. Drop Irrelevant Columns

* Drop columns: Config\_Version, Last\_Patch\_Date, Deployment\_Token from both station datasets.

1. Add Station Identifier & Combine Logs

* Append a Station column to both station datasets (Station1 or Station2).
* Concatenate both station datasets into a single dataframe.

1. Handle Missing Values

* Replace missing values in numeric columns (CPU\_Utilization (%), Memory\_Usage (%), Disk\_IO (%), Network\_Traffic\_In (MB/s), Network\_Traffic\_Out (MB/s)) with the median.
* Convert Log\_Timestamp to datetime, forward-fill missing timestamps.
* Backward-fill missing Server\_ID values.

1. Merge With Metadata

* Merge the merged station logs with the metadata on Server\_ID

1. Time-Based Feature Engineering

* Extract and include columns: Date, Hour, DayOfWeek, Month, Year from Log\_Timestamp.

8. Categorize Performance Metrics

* High: ≥80%
* Medium: 50–80%
* Low: <50%

1. System Health Score & Status

Compute System\_Health\_Score as:

Assign System\_Health\_Status:

1. Excellent: ≥80
2. Good: 60–79
3. Fair: 40–59
4. Poor: <40
5. Export Cleaned Data

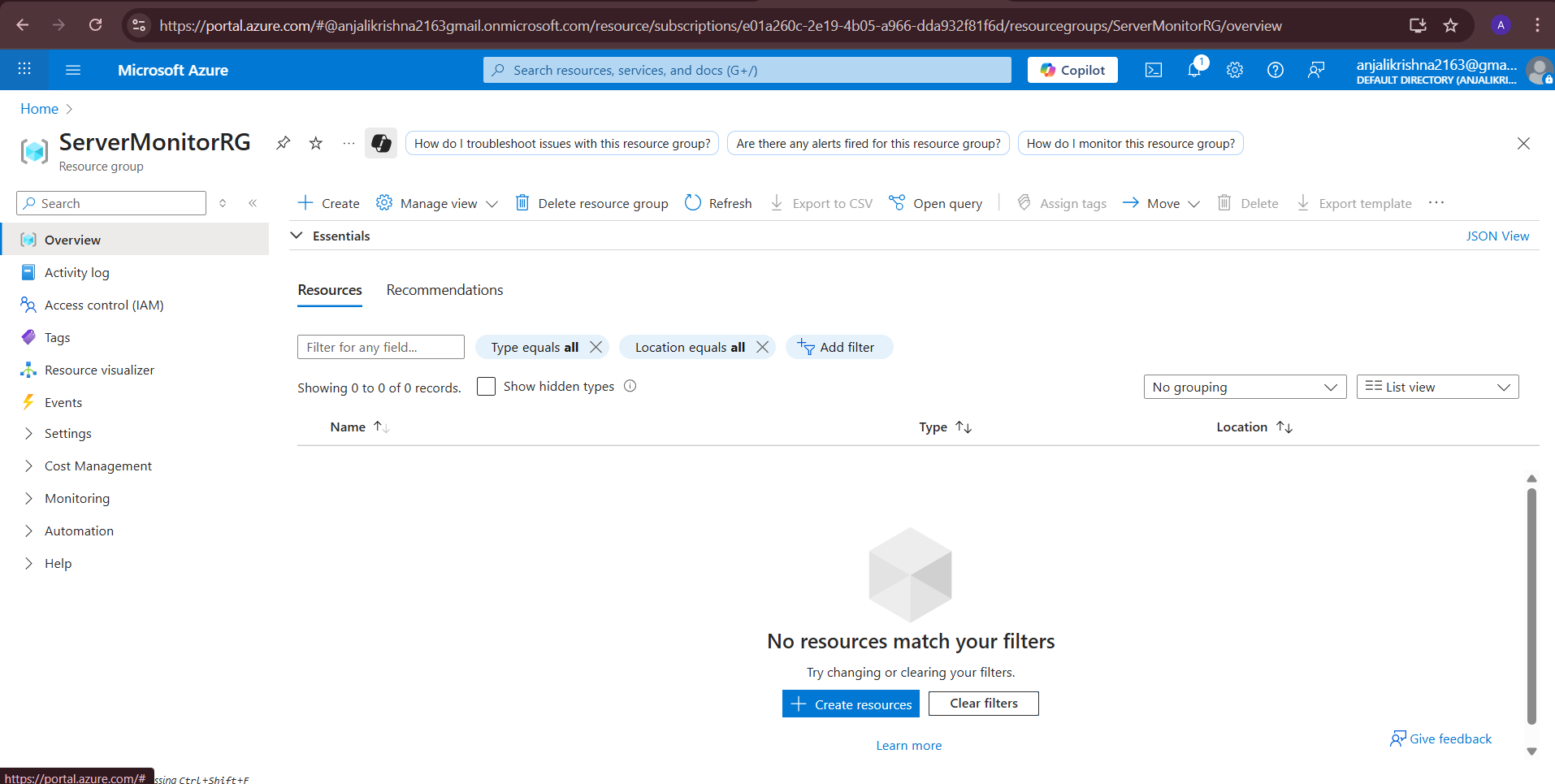
* Save final combined dataframe as transformed\_server\_data.csv.
* Save filtered data for each station as Station1\_transformed.csv and Station2\_transformed.csv.
* Save cleaned metadata as server\_metadata\_clean.csv.

# PHASES

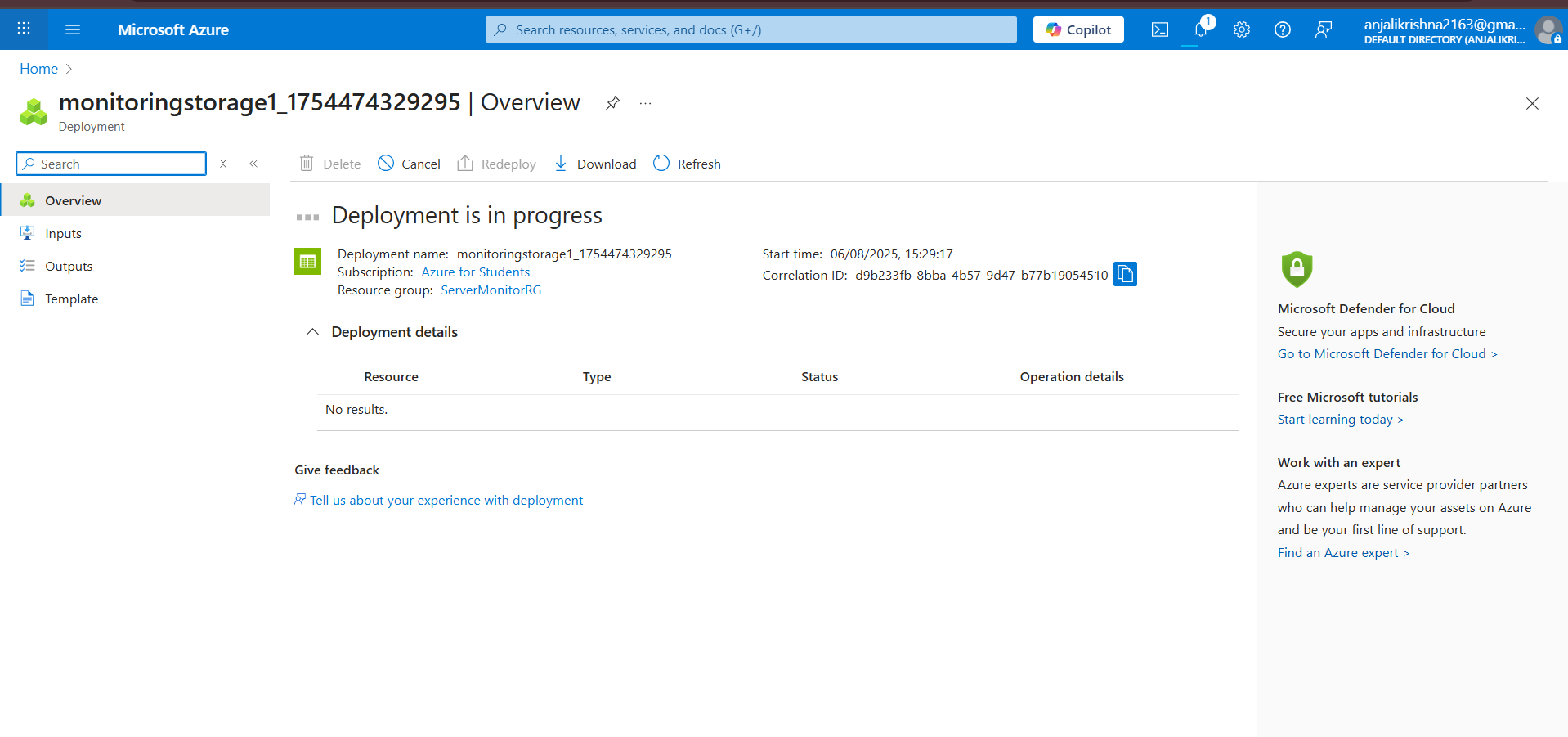
PHASE 1: Setup and Ingestion in Azure

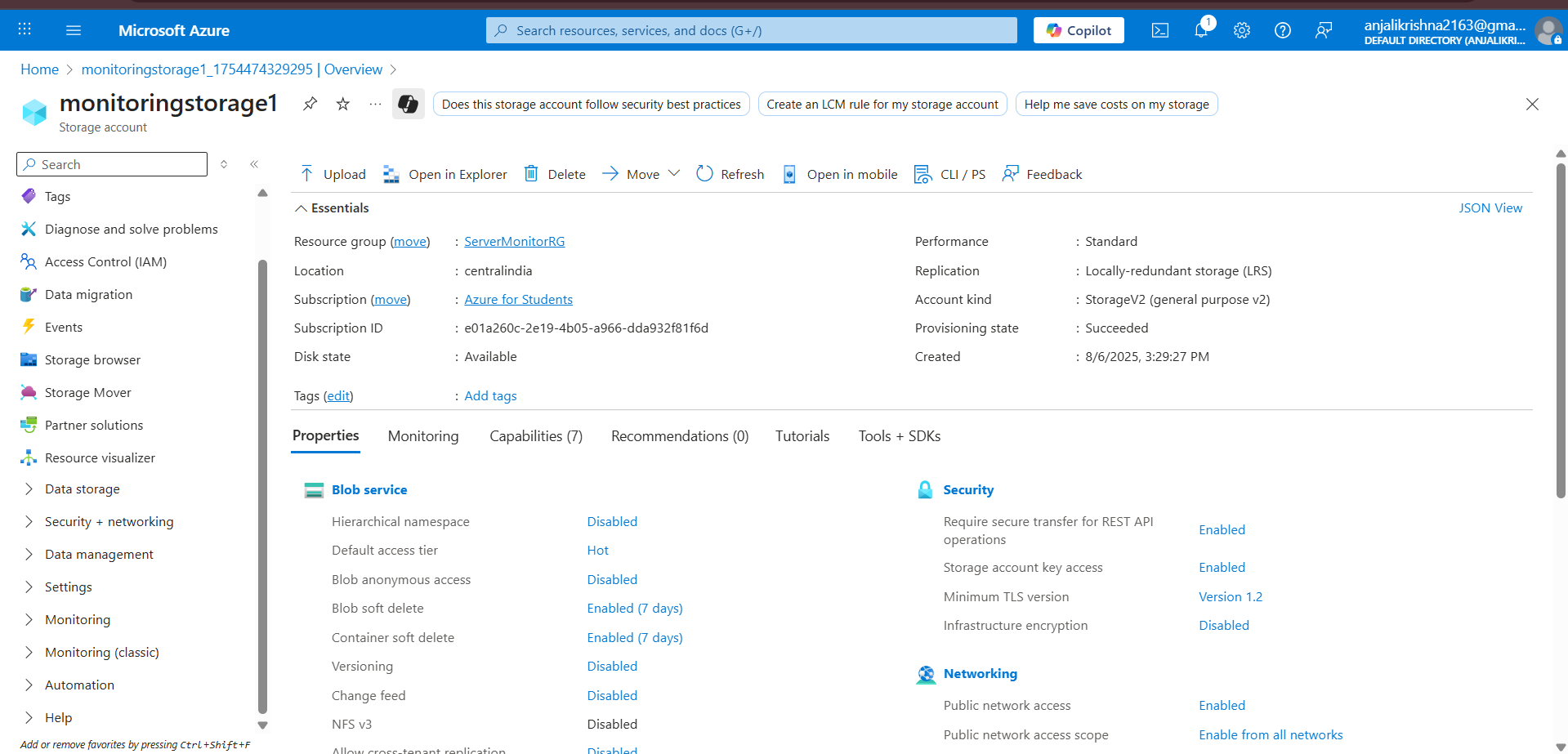
Step 1: Create Azure Resources

1.1 Create a Resource Group

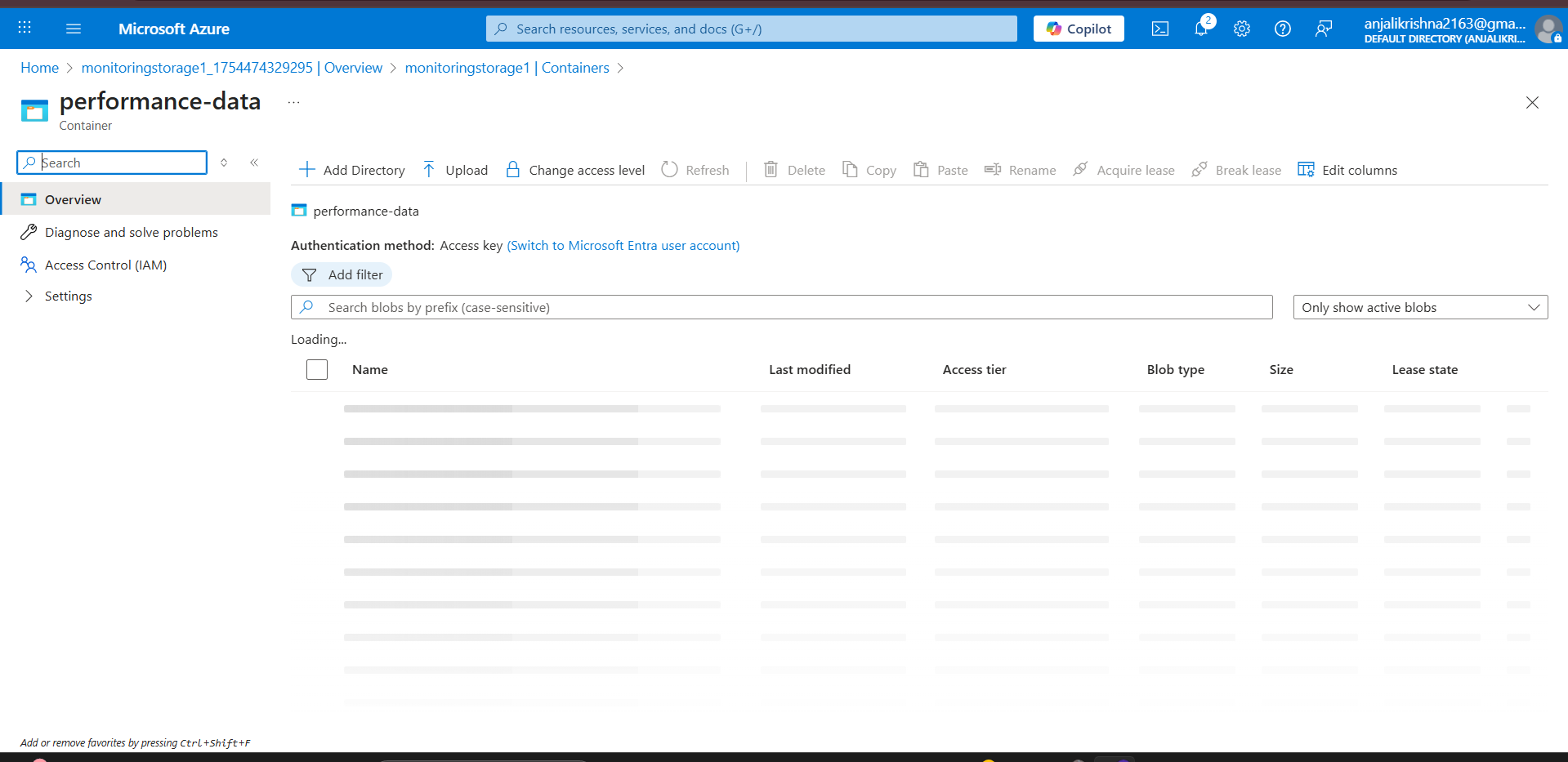


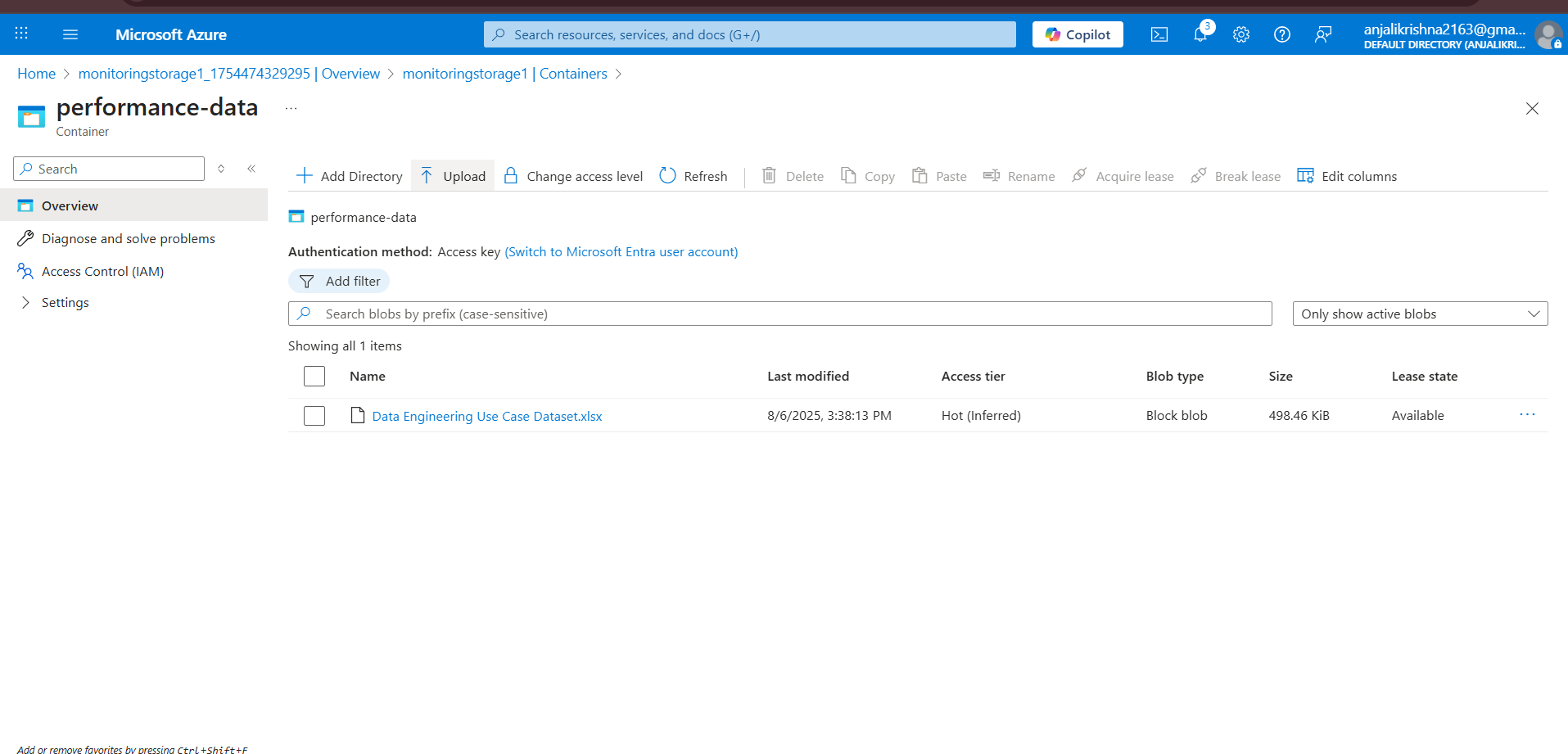
1.2 Create Storage Account (Blob)





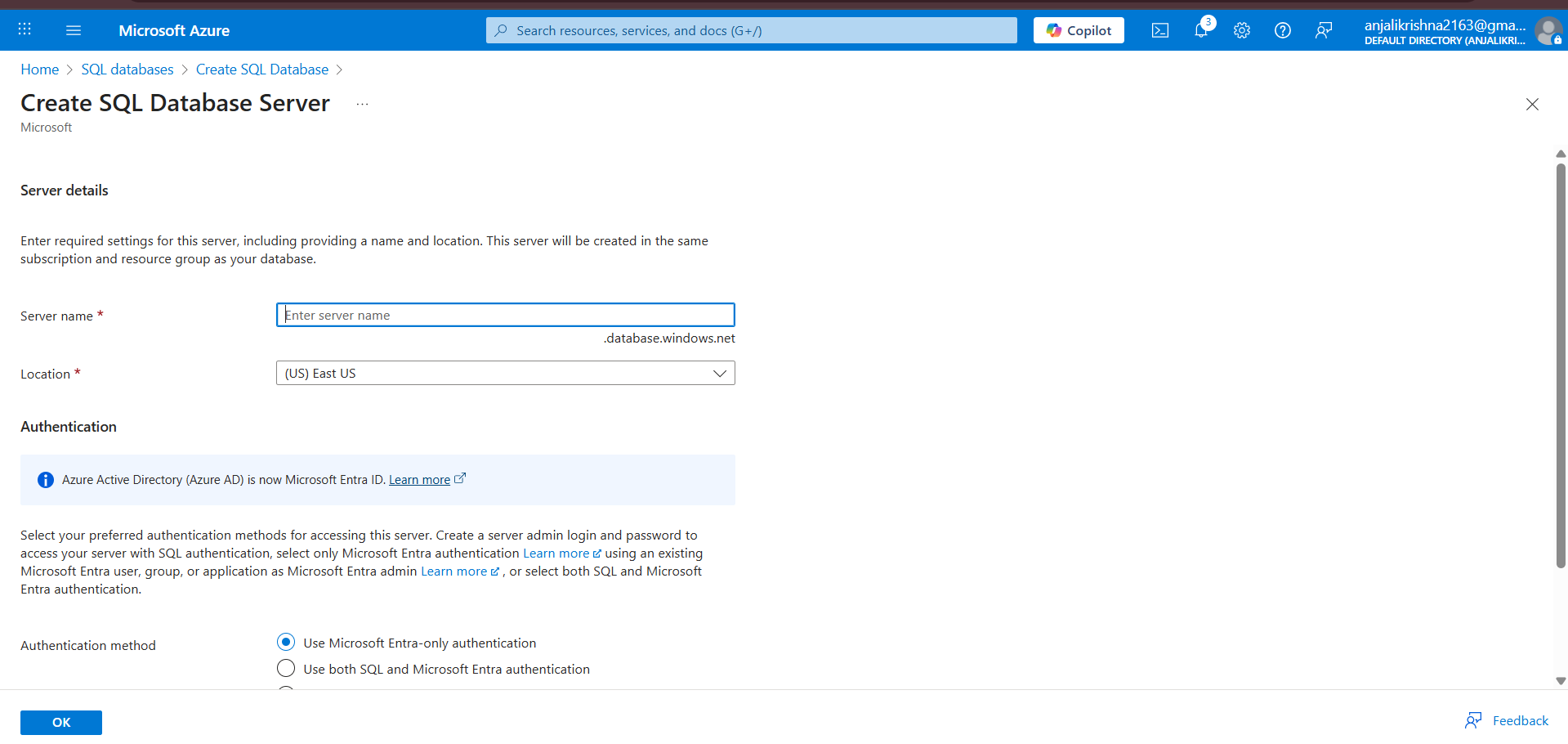
1.3 Create Blob Container and Upload Excel File



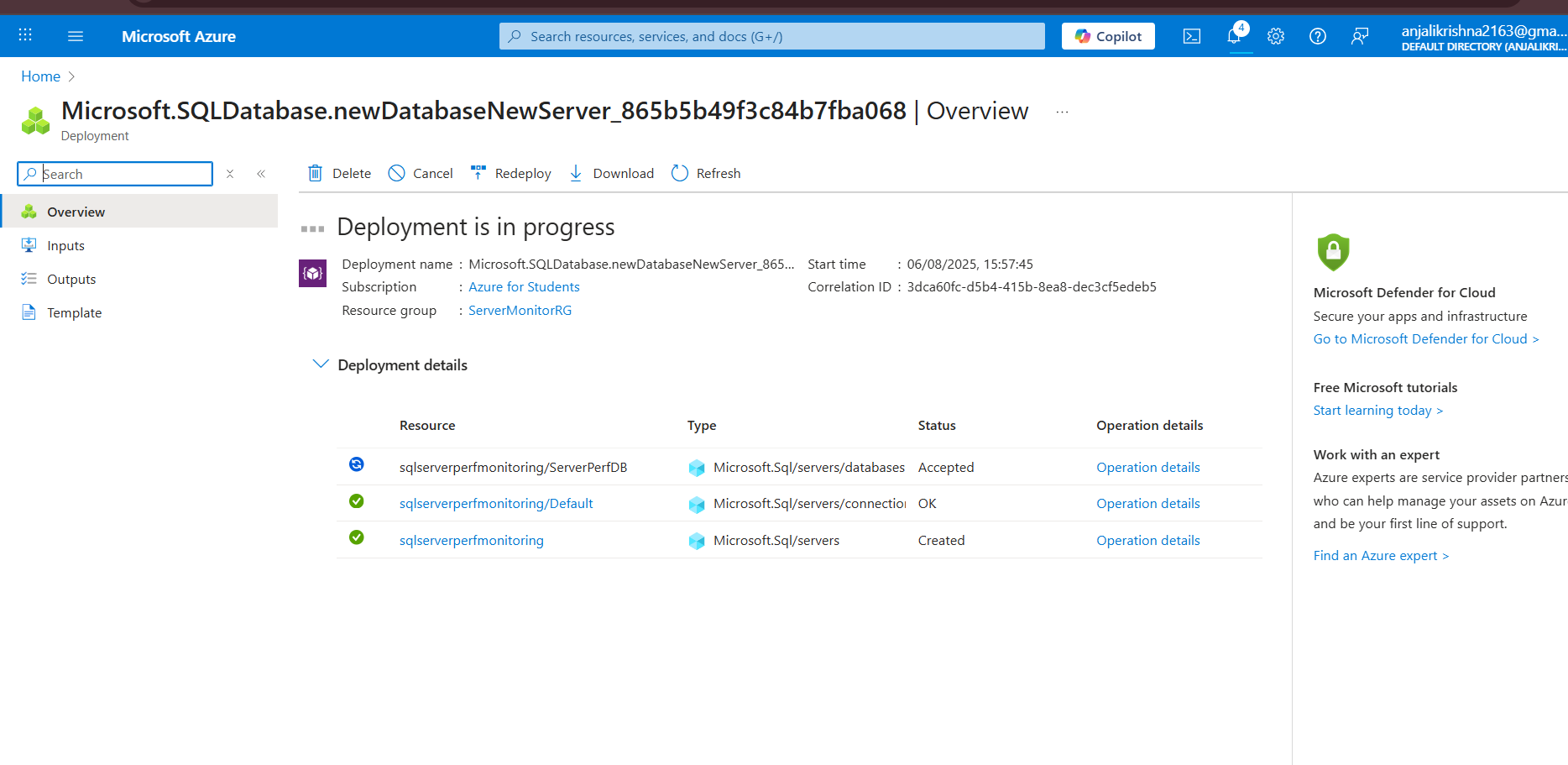


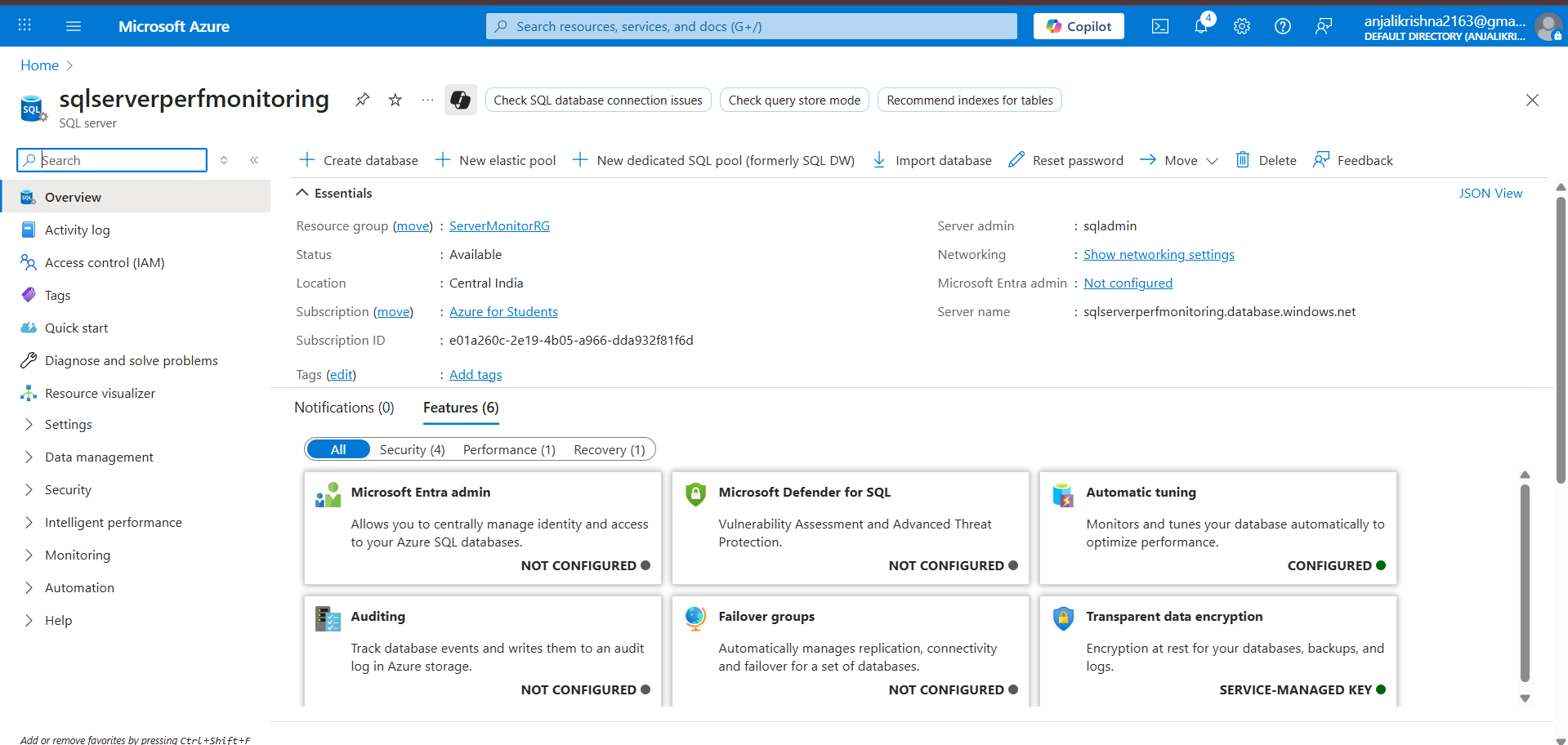
2. SET UP AZURE SQL DATABASE

2.1 Create SQL Server + Database

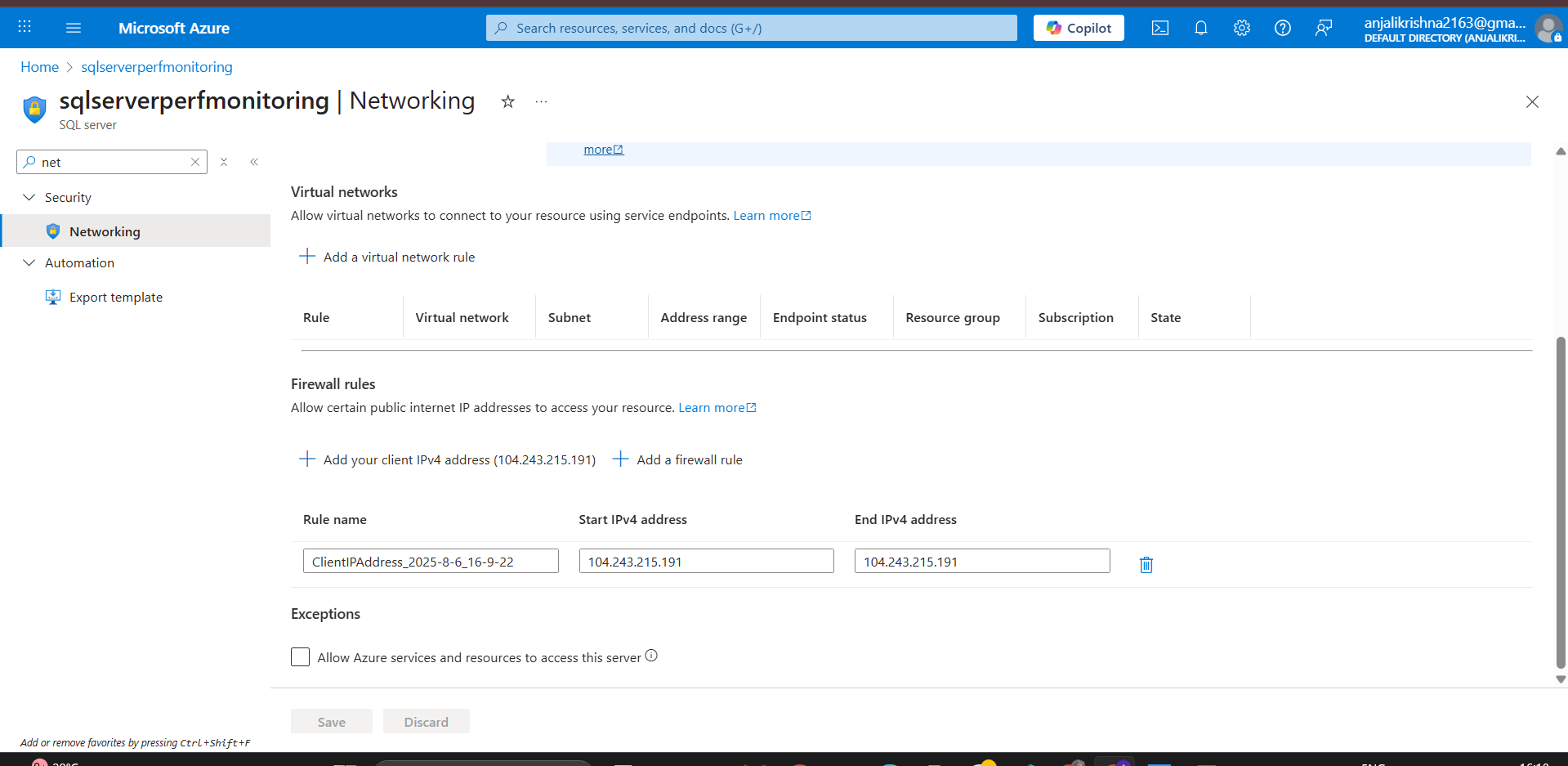


Created new sql server by SQL authentication which lets you create a SQL admin login and password to connect easily



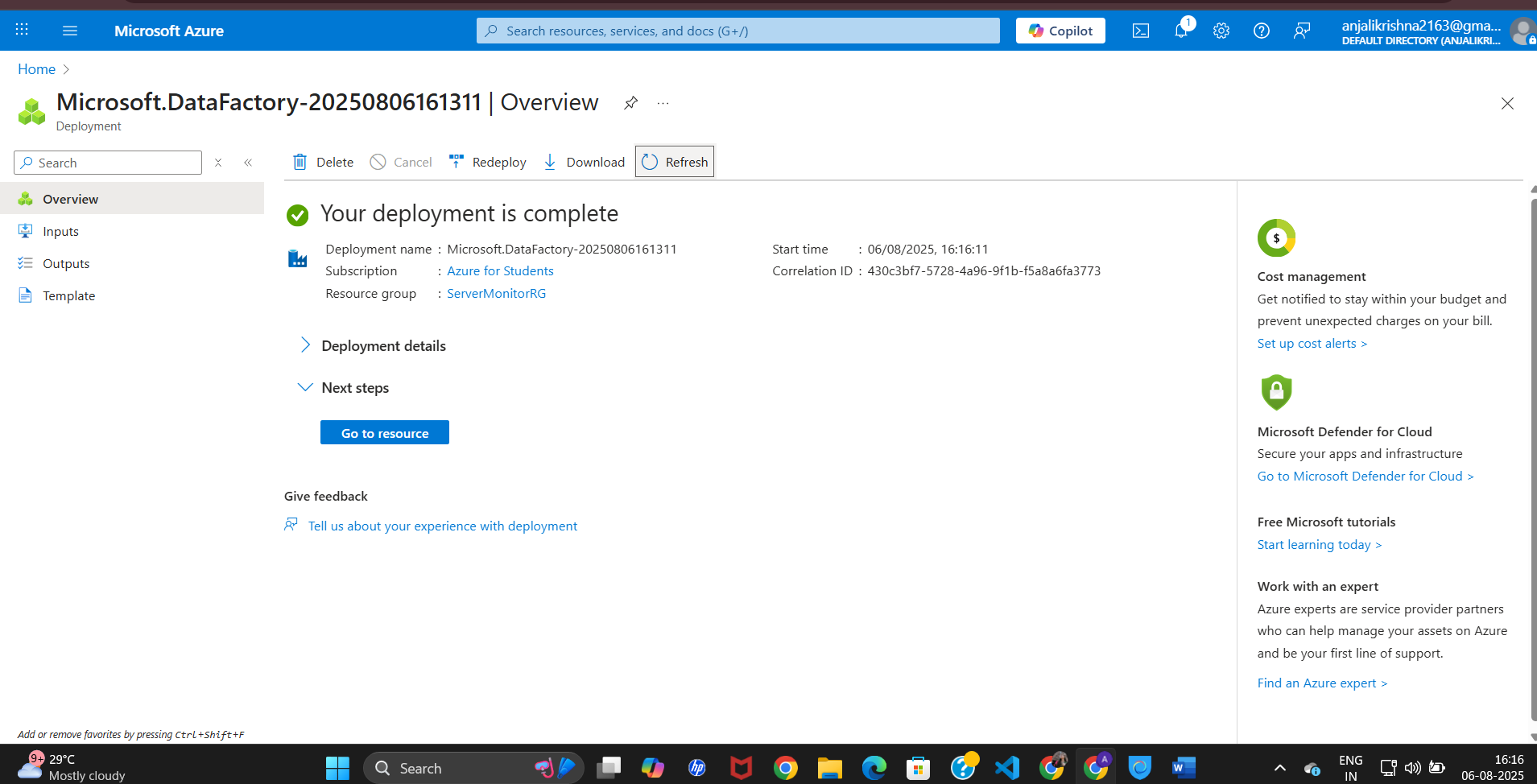


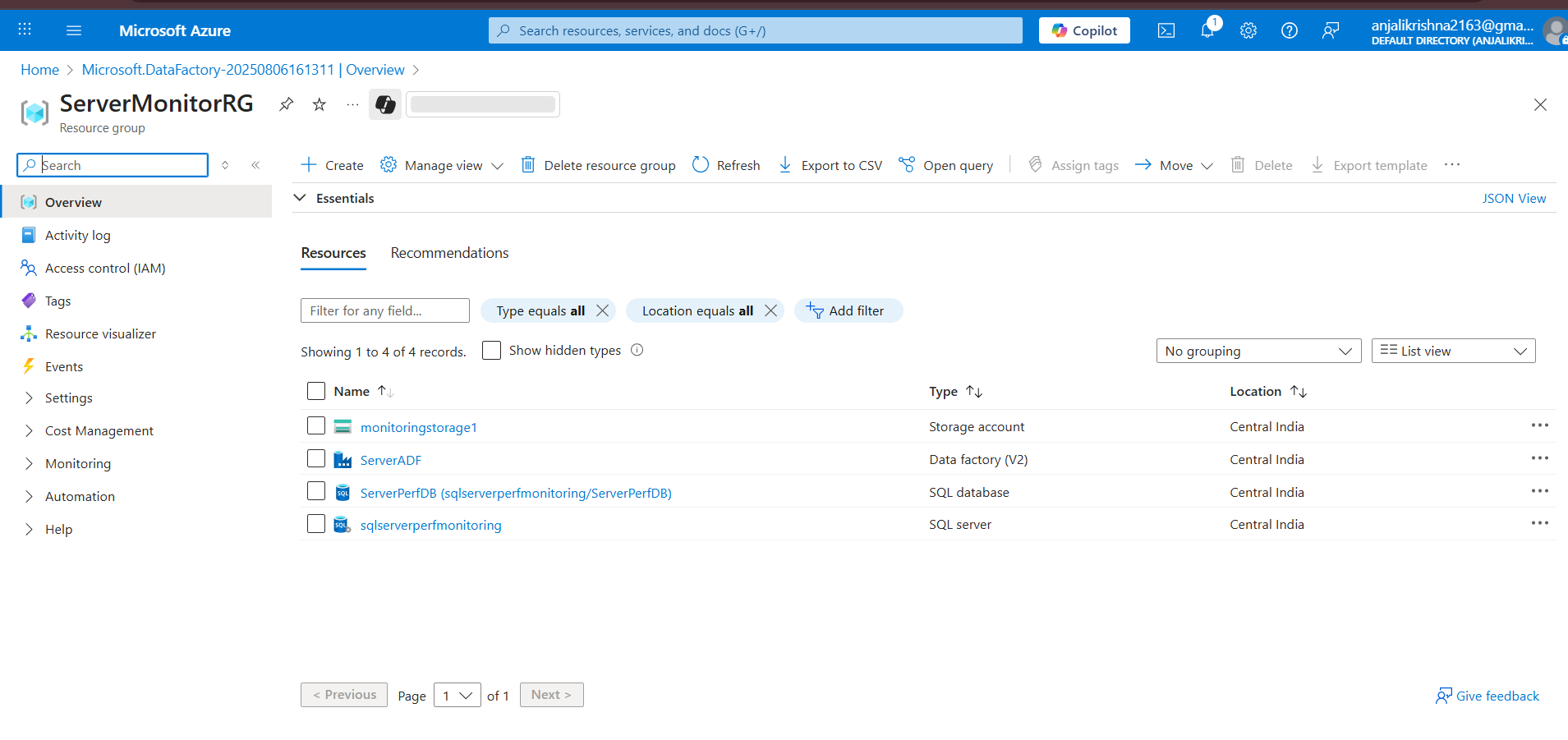
2.2 Allow Your IP

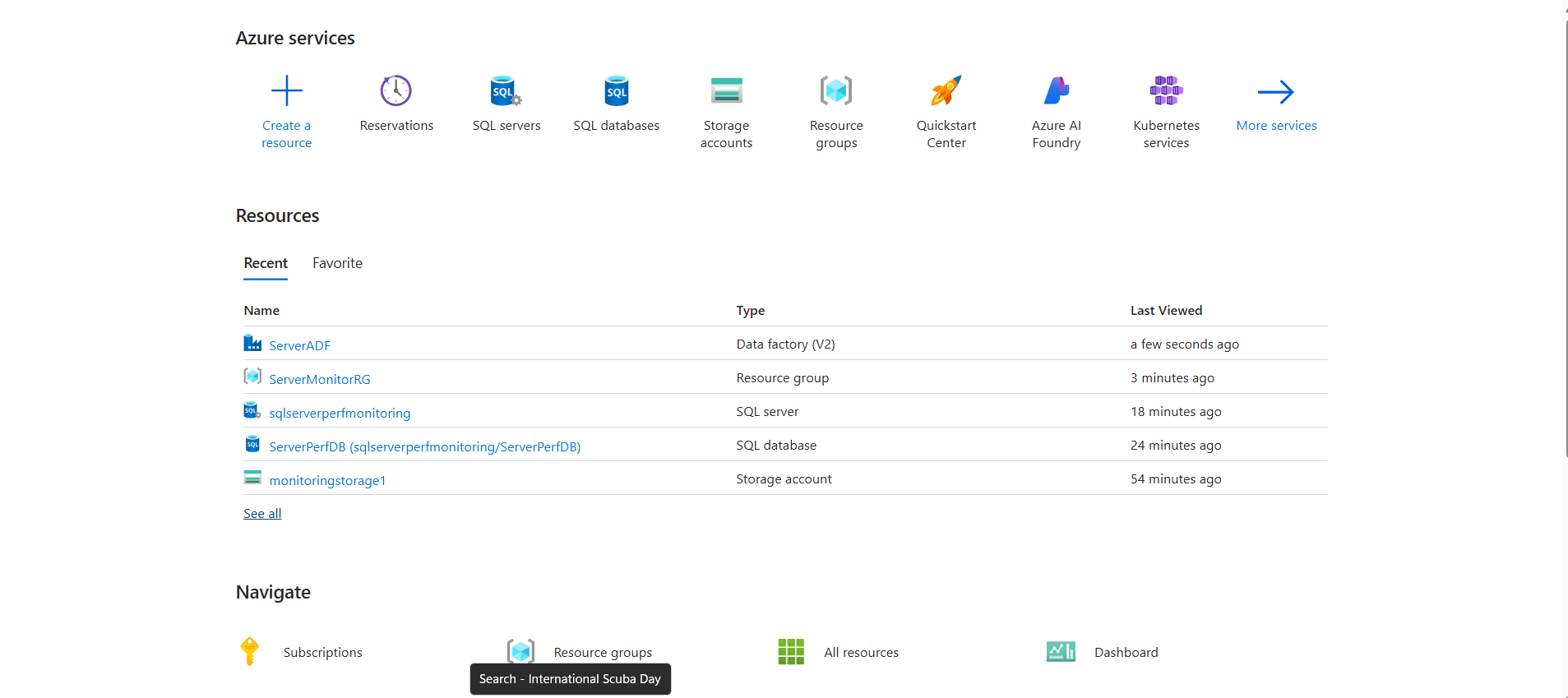


3. SET UP AZURE DATA FACTORY (ADF)

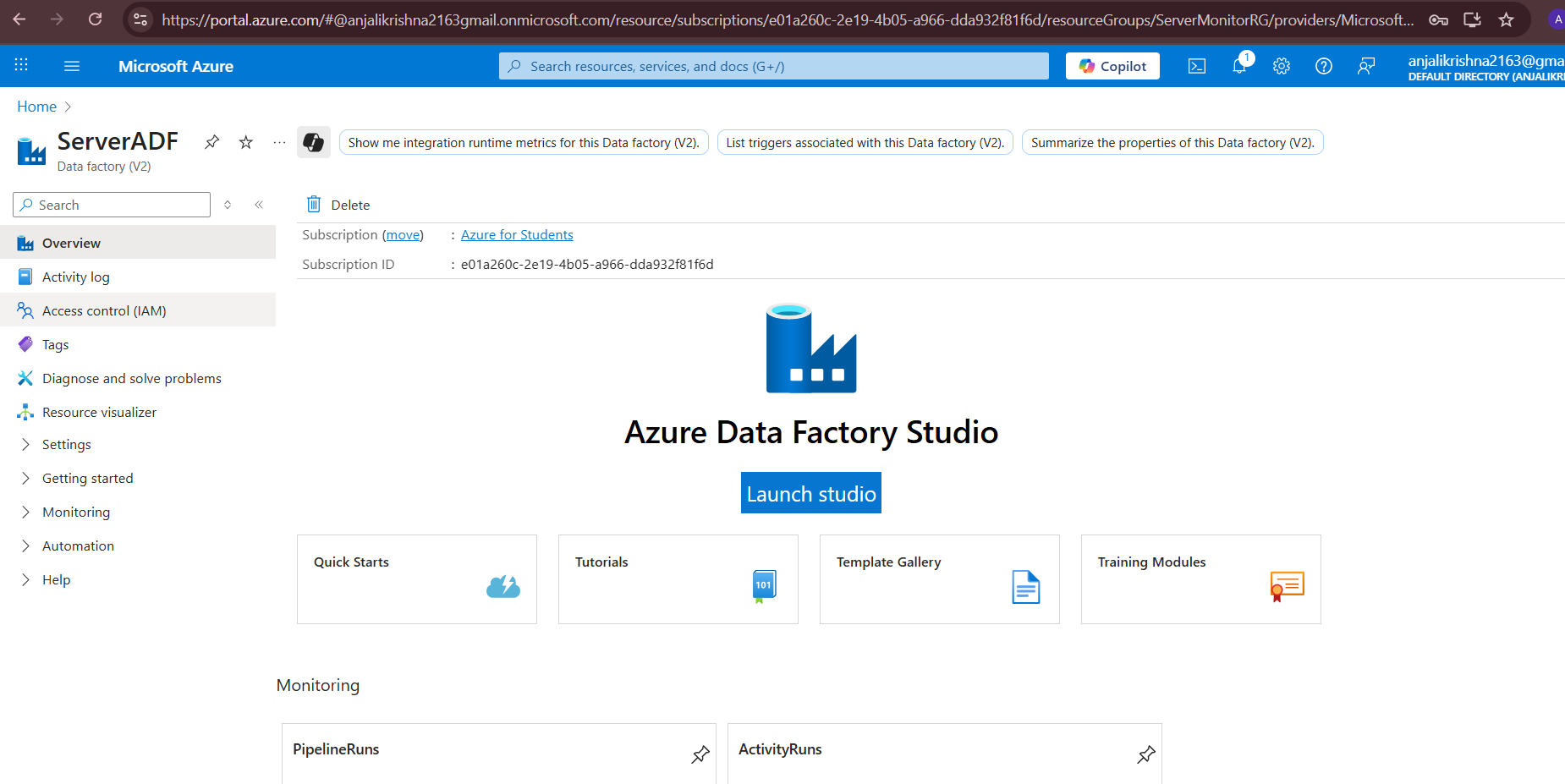
3.1 Create Data Factory

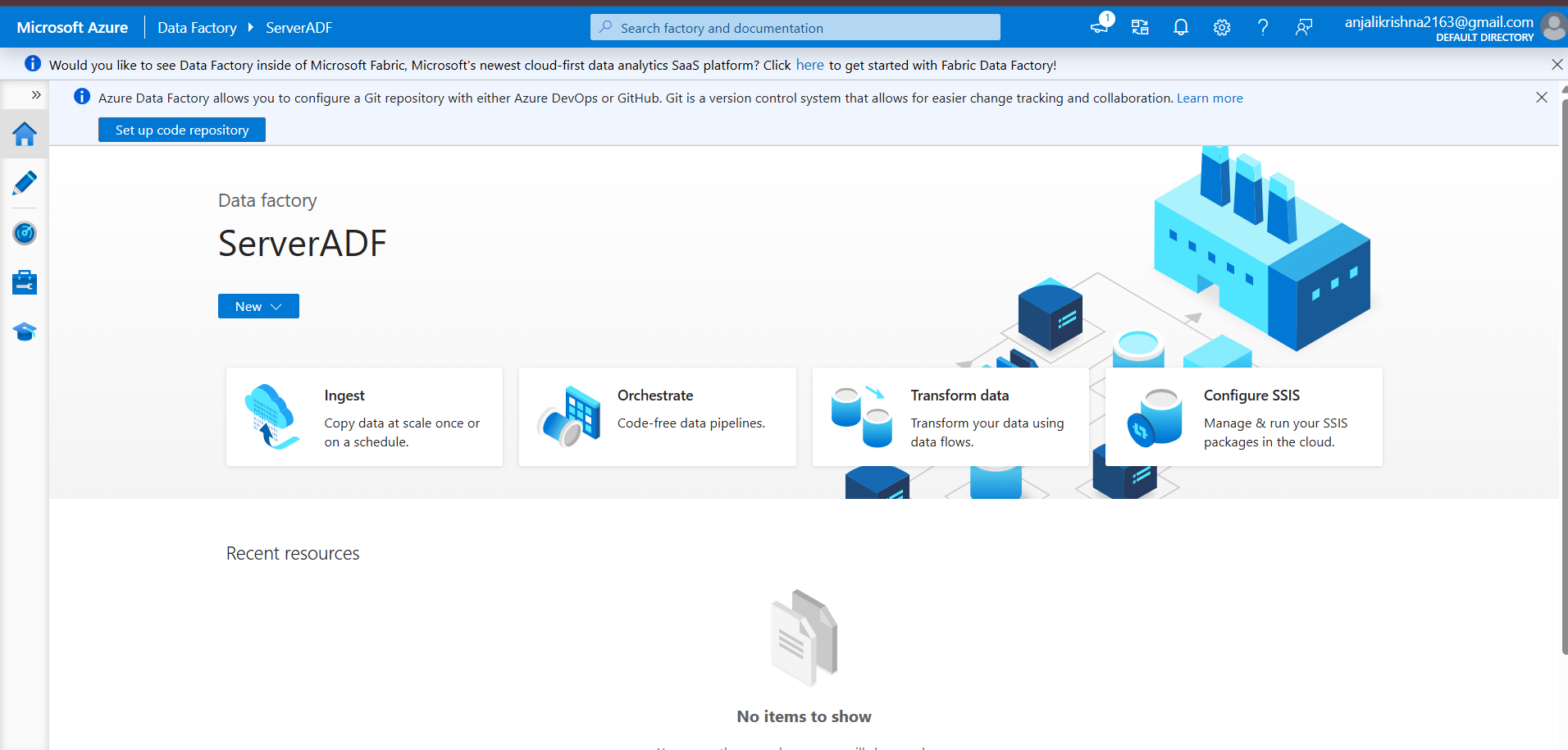






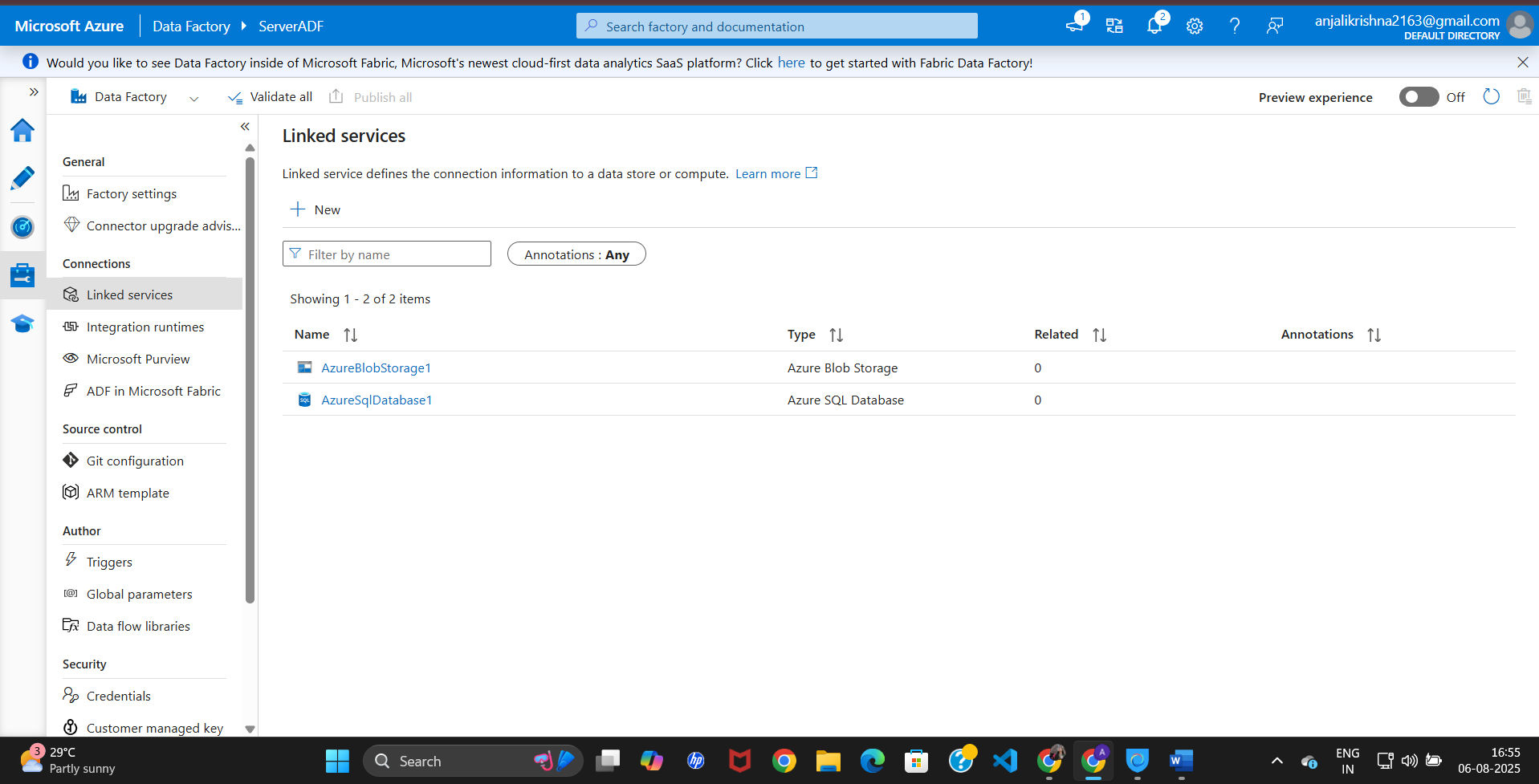
3.2 Launch ADF Studio



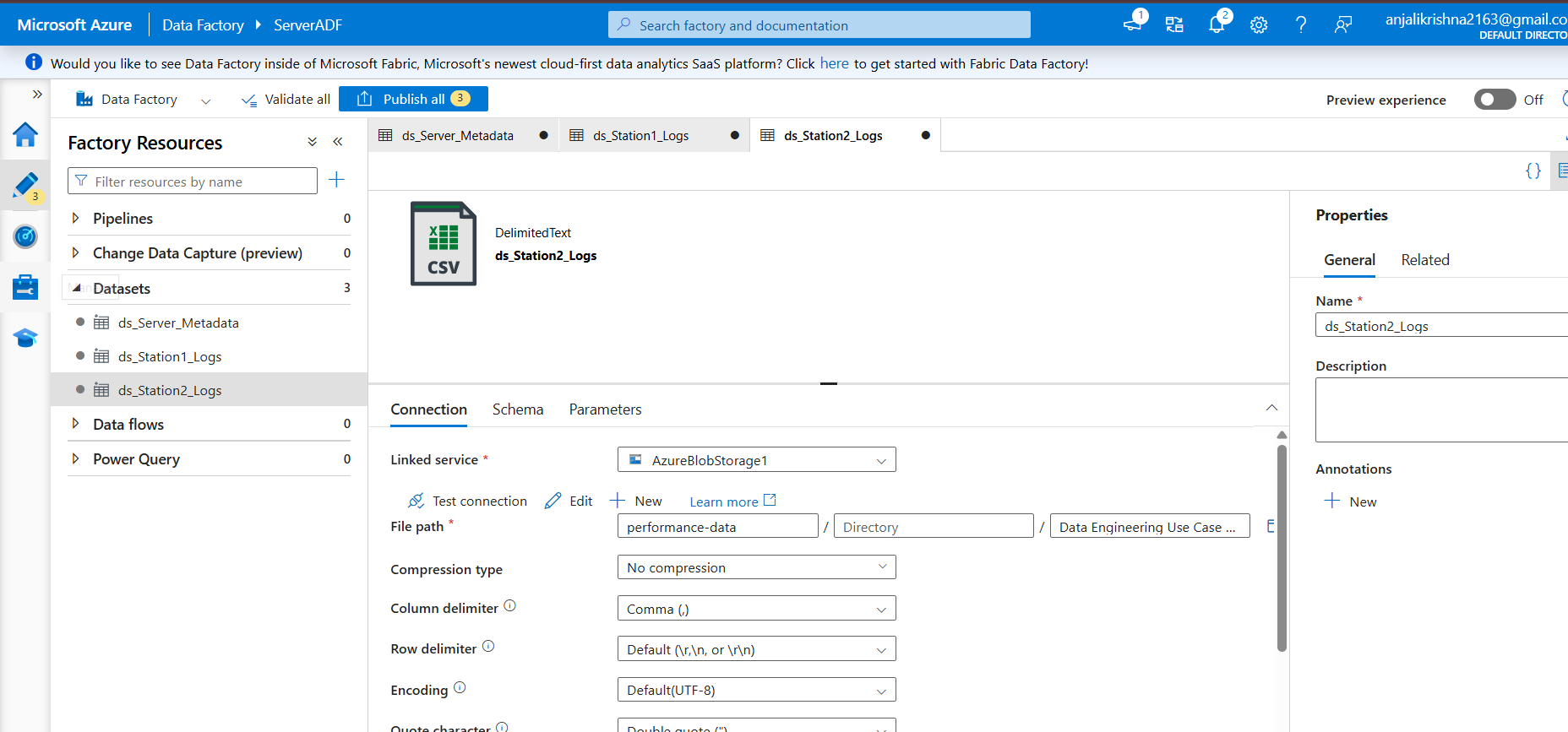


1. BUILD THE PIPELINE

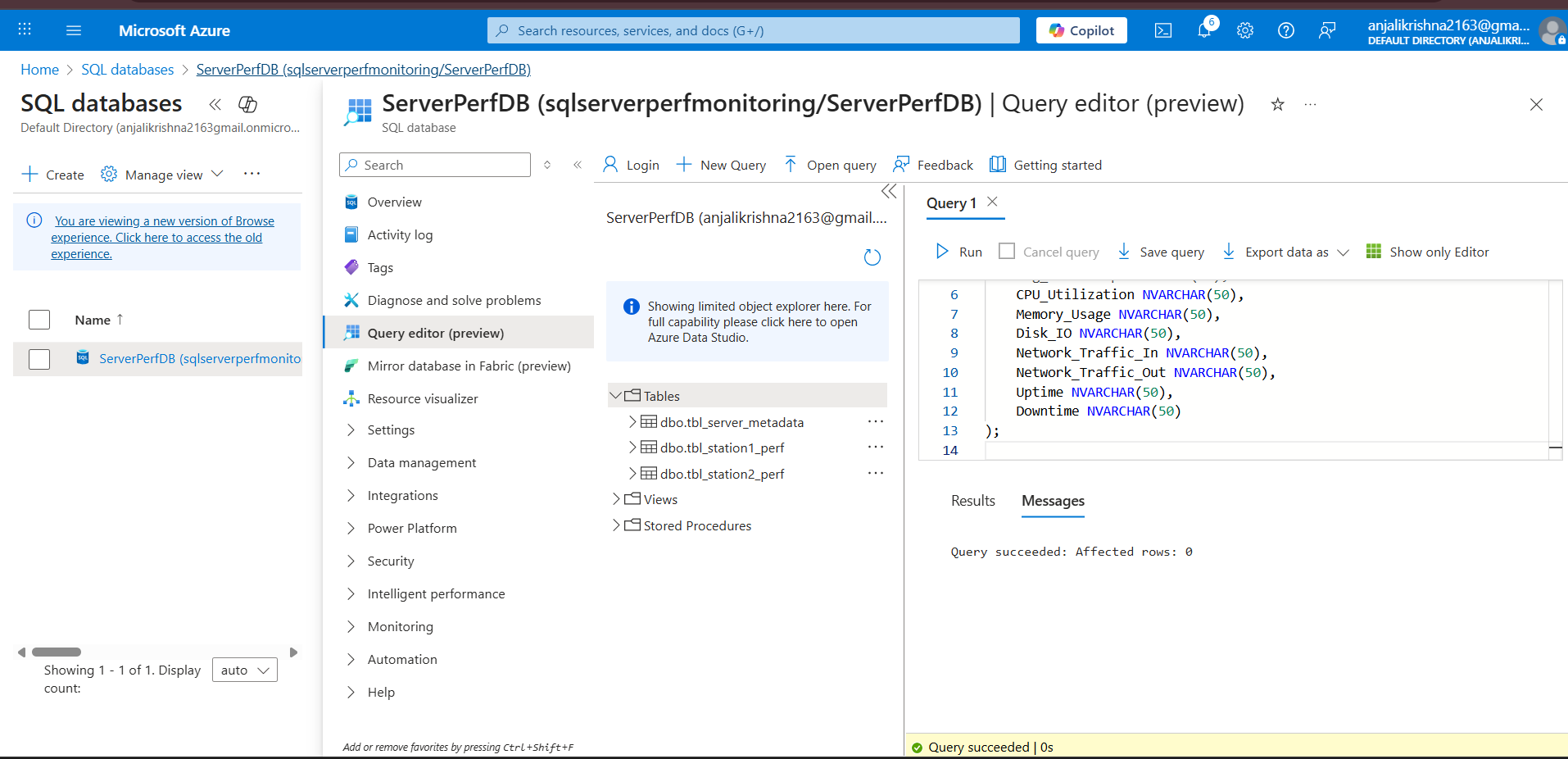
4.1 Create Linked Services

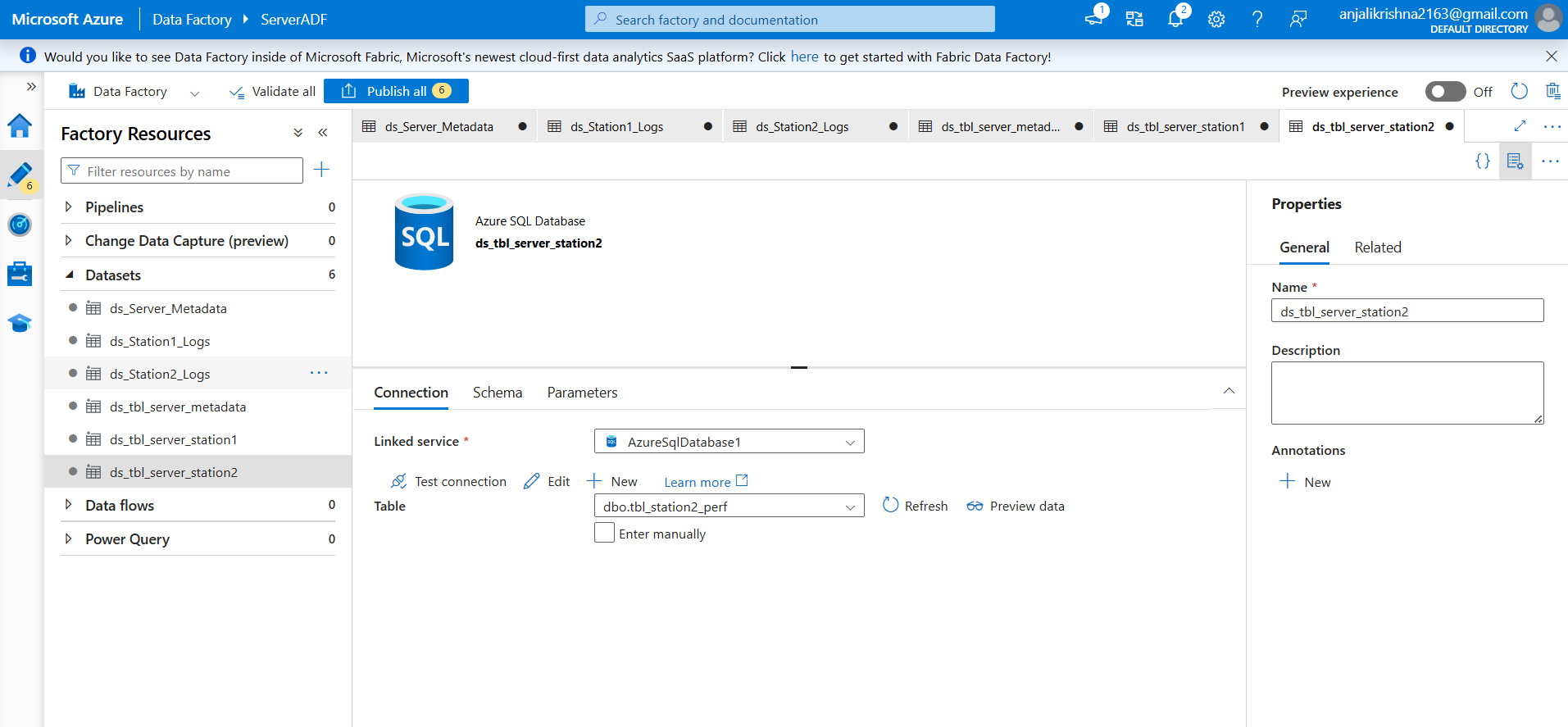


4.2 Create Dataset for Blob (Excel)

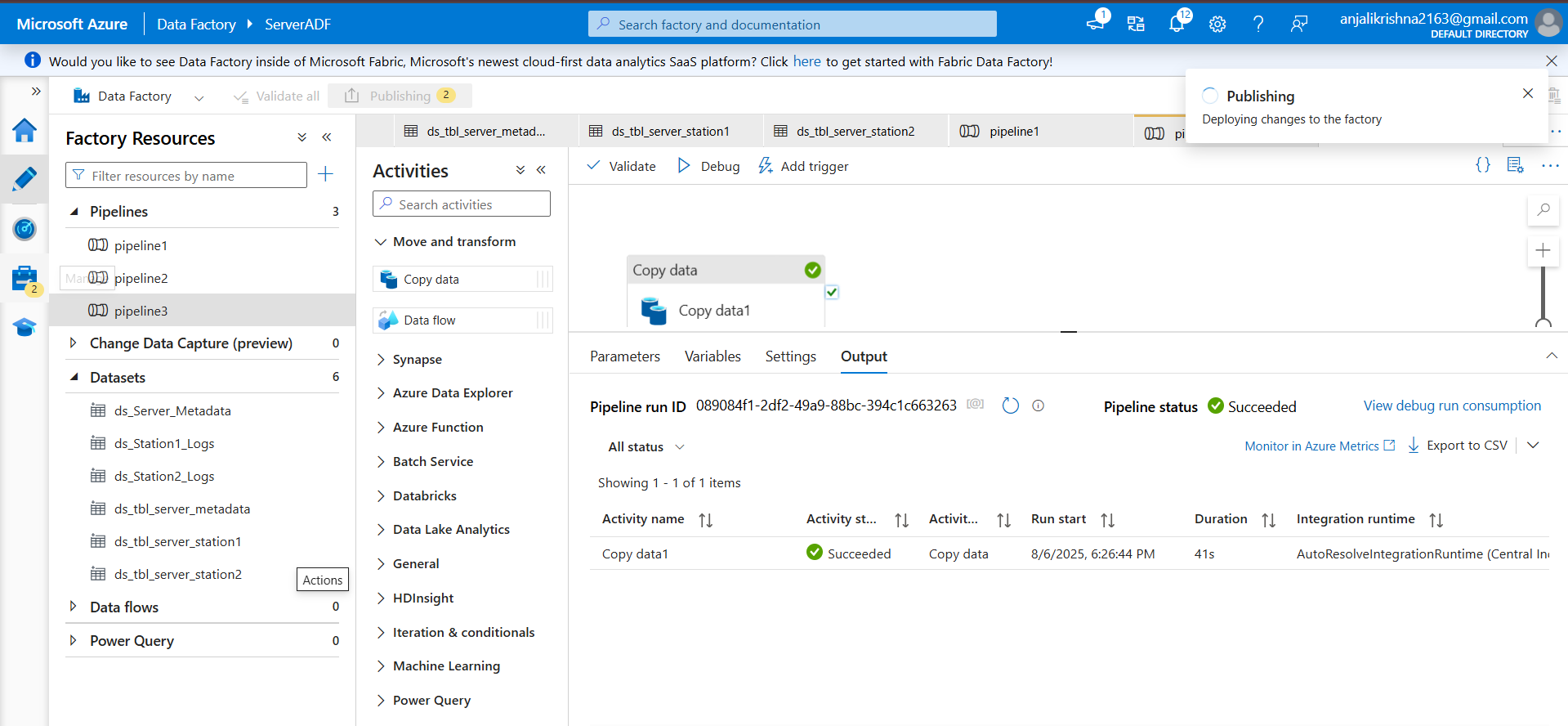


4.3 Create SQL Table Datasets

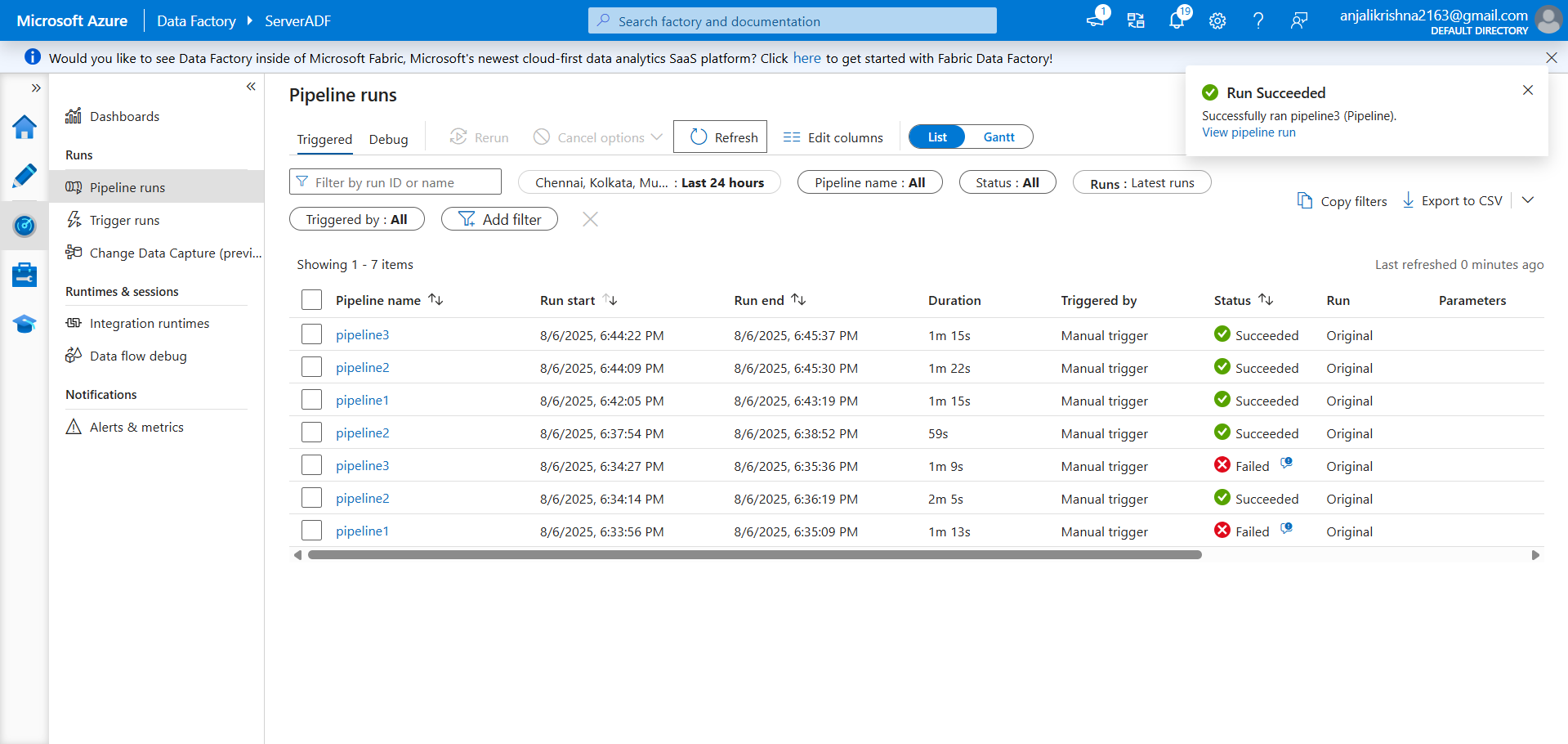




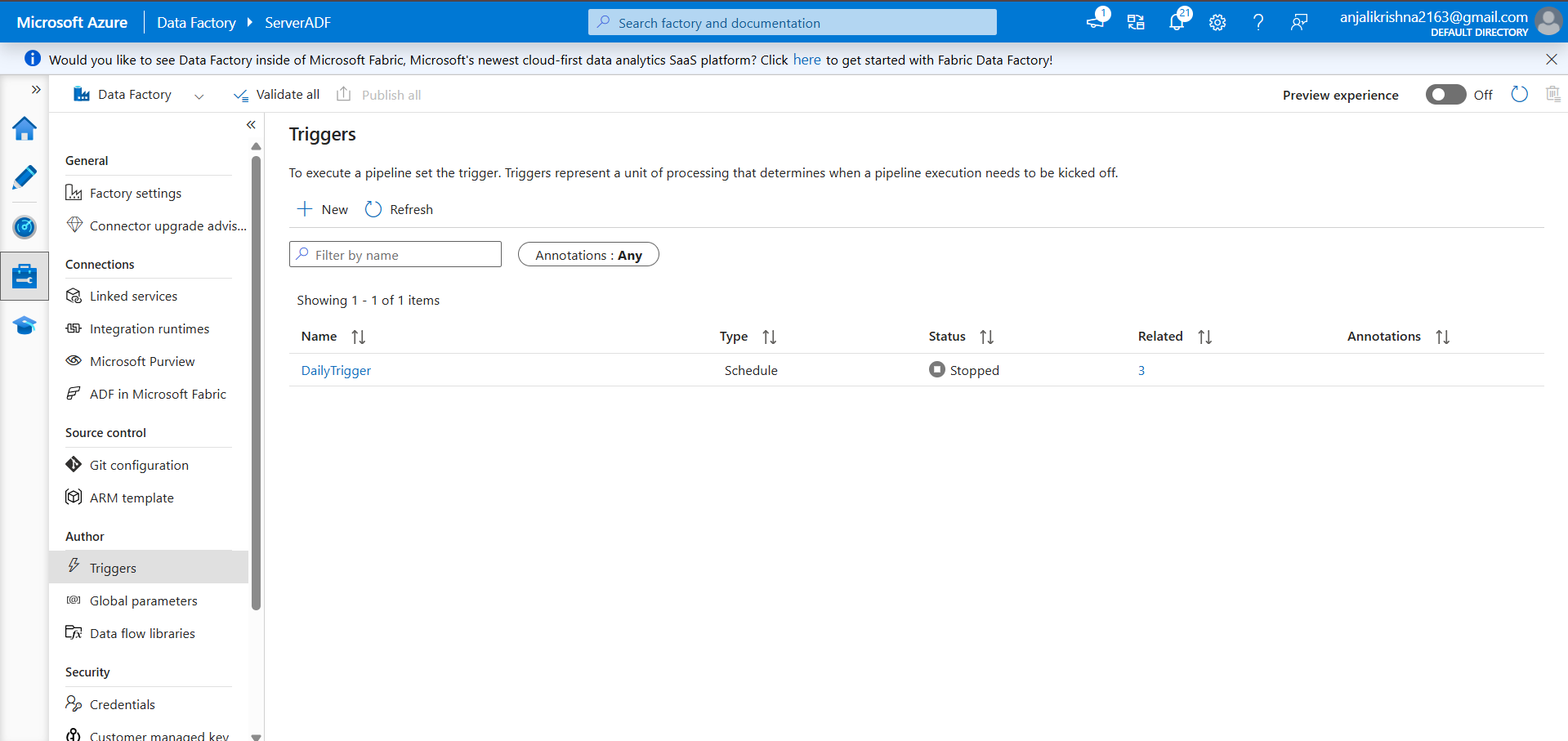
Create Copy Data Pipeline



Data trigger Monitoring



Automate Pipeline Scheduling



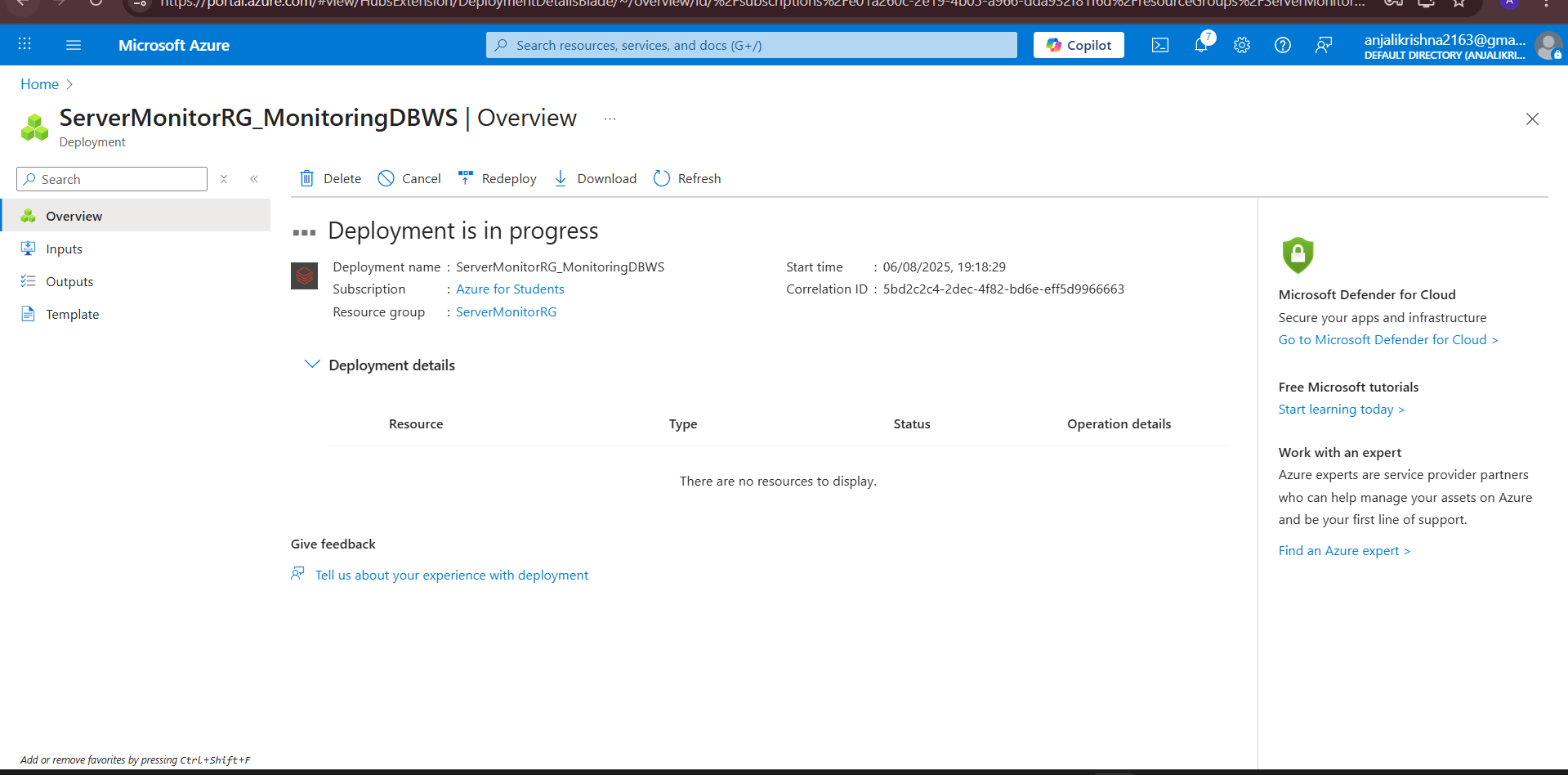
Add basic email alert for failures

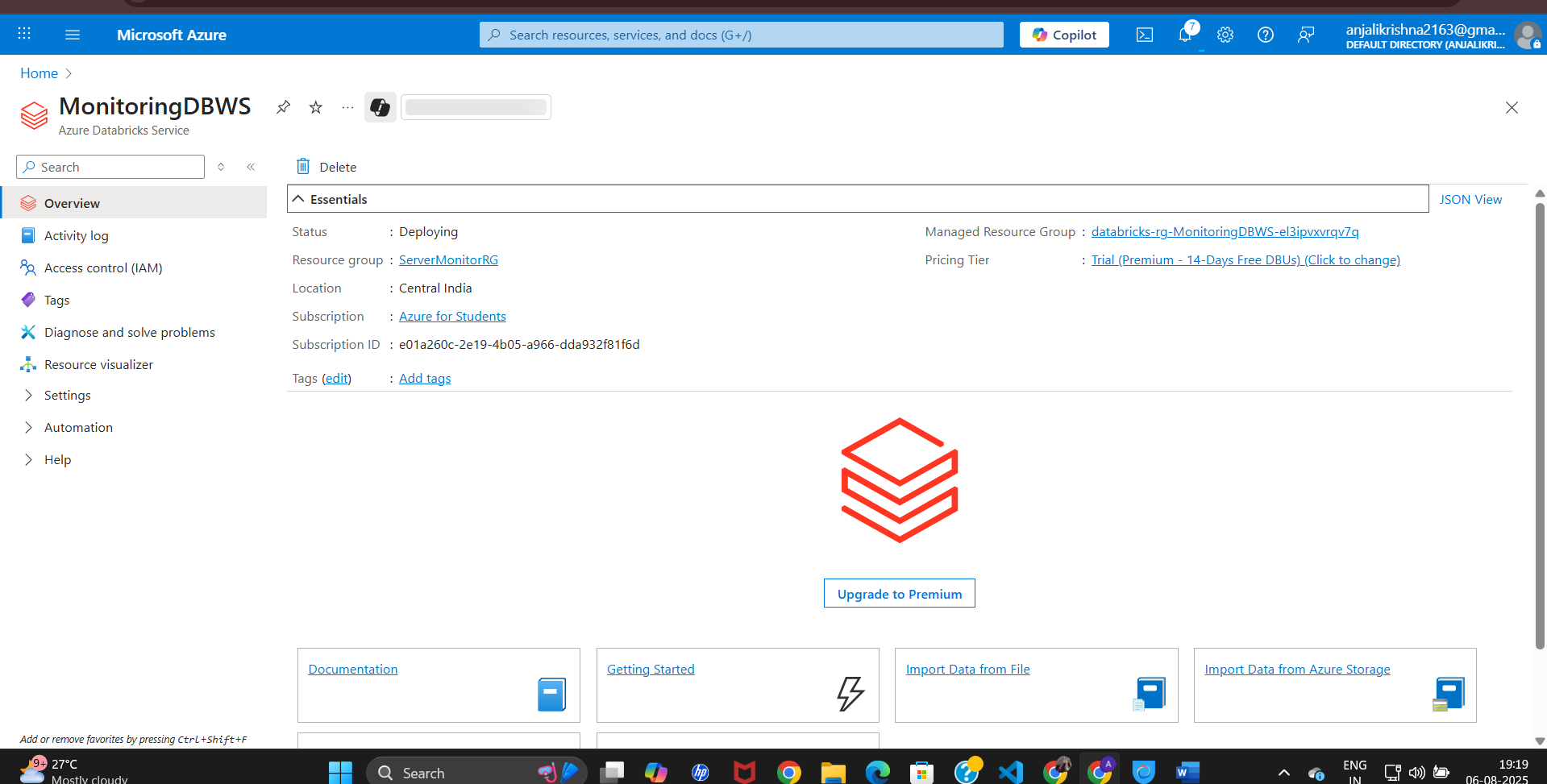
Alerts can be added via Azure Monitor later.

PHASE 2: Data Transformation Using Azure Databricks

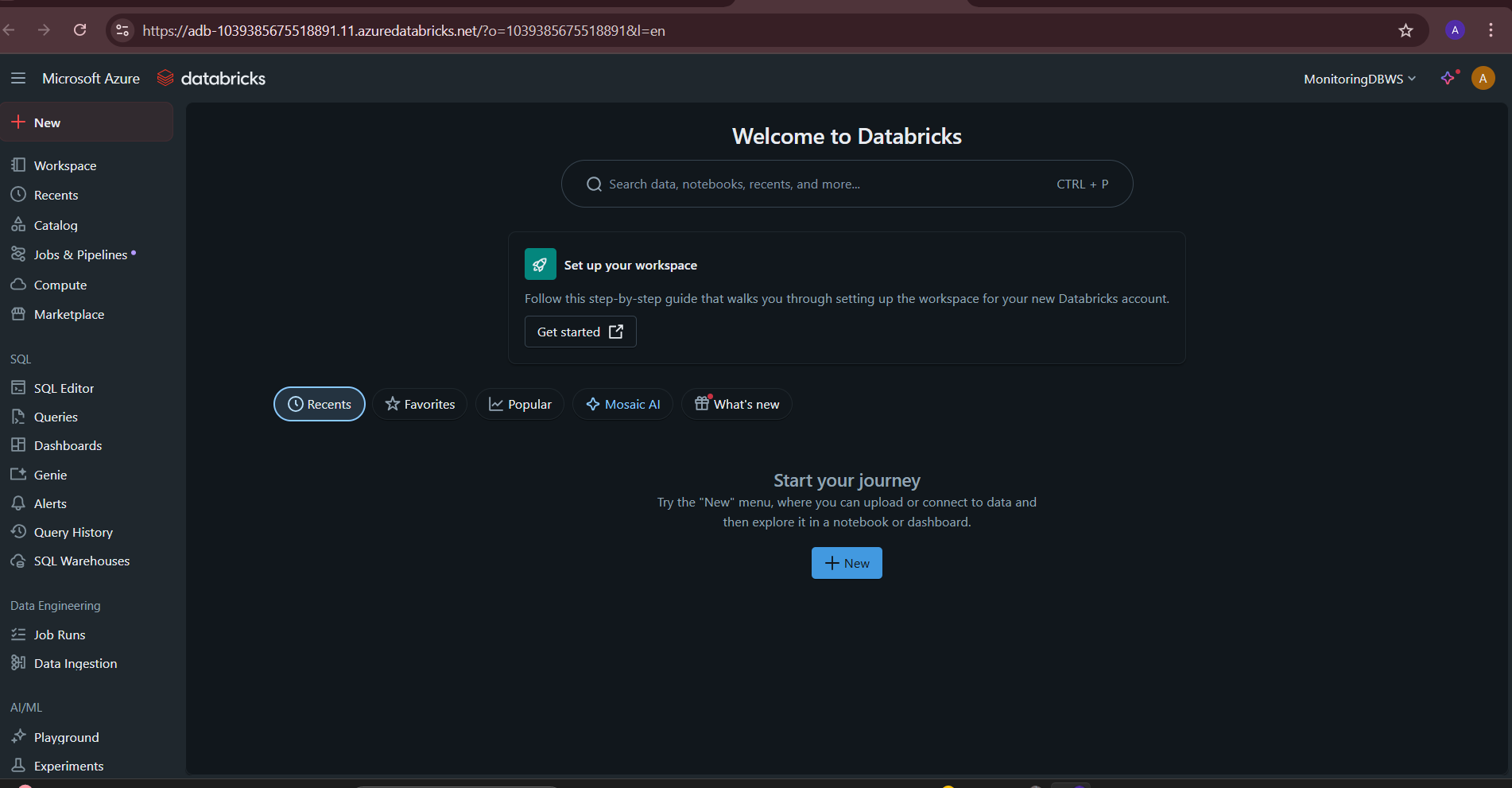
Step 1: Set Up Azure Databricks

* 1. Create Databricks Workspace

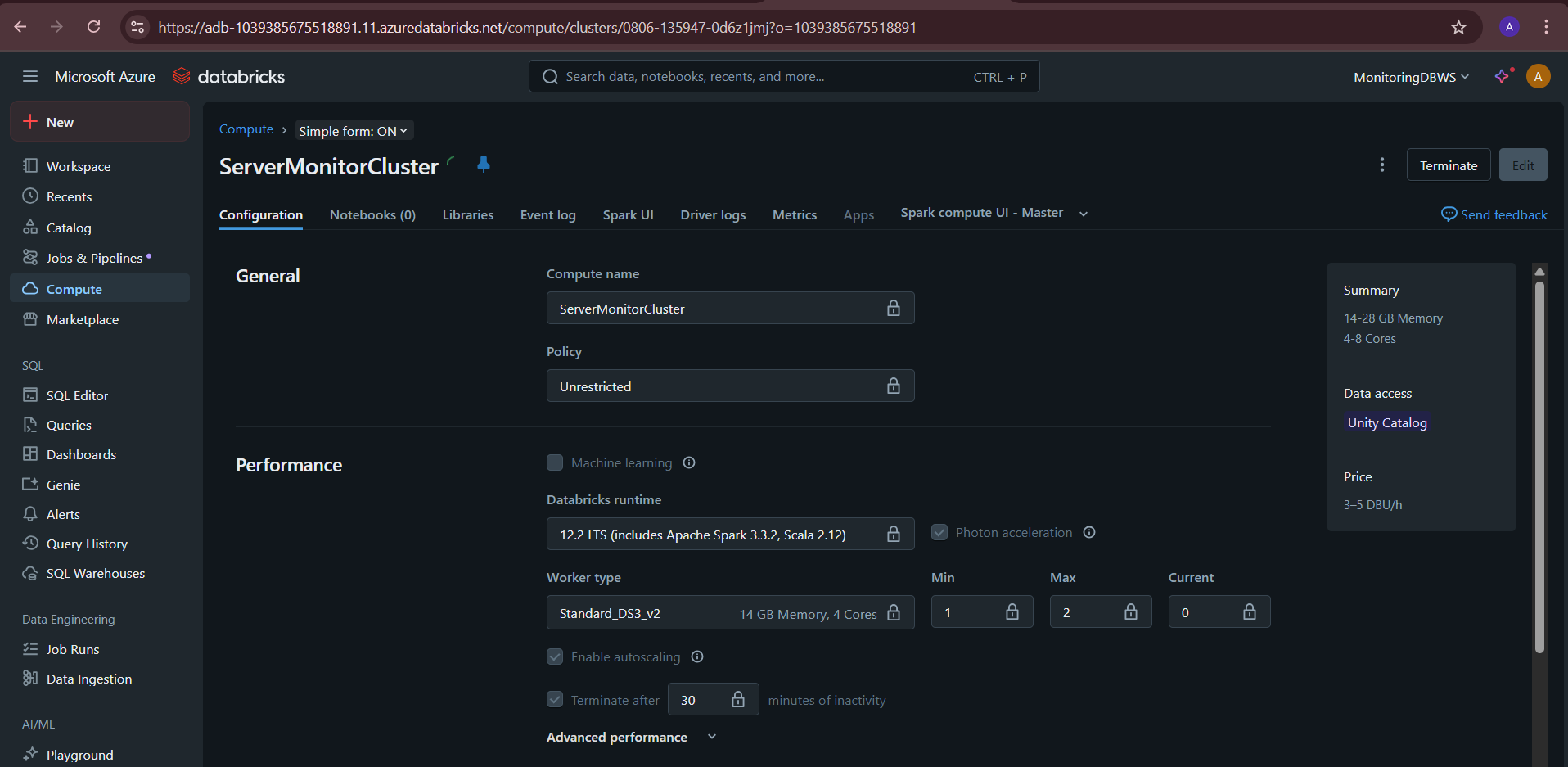




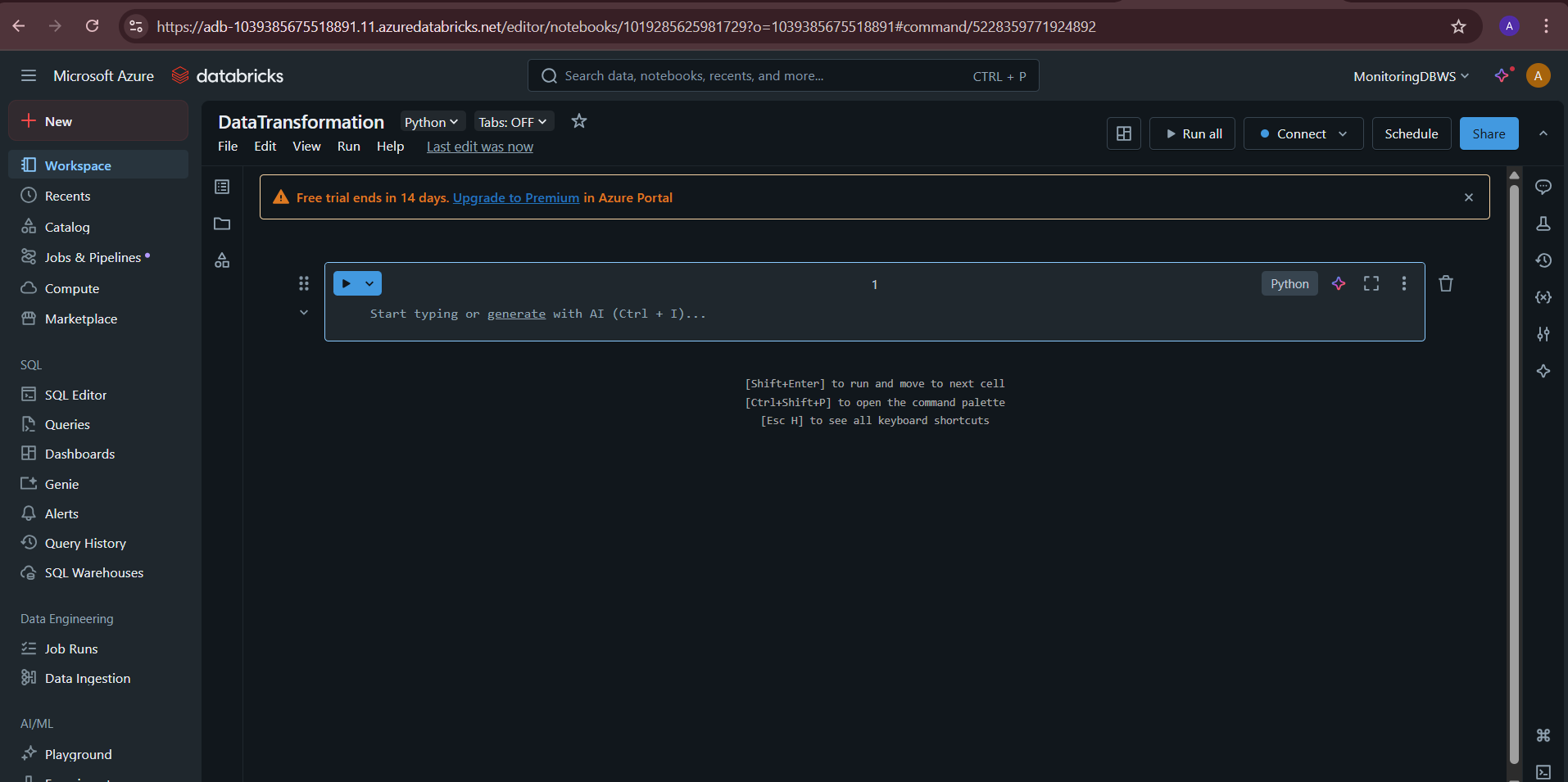
* 1. Launch the Workspace



Step 2: Create a Cluste



Step 3: Create a Notebook



Since having the student Access Encountering SKU error

Continuing with python using SAS url

STEP 1:

GET SAS URL FROM AZURE BLOB STORAGE

m=https://monitoringstorage1.blob.core.windows.net/performance-data/Data%20Engineering%20Use%20Case%20Dataset\_metadata.csv?sp=r&st=2025-08-06T16:19:25Z&se=2025-08-27T00:34:25Z&sv=2024-11-04&sr=b&sig=CJGsNFDJImwjOU02OUhSpFdfn8kkWlDXjoc7MaHZgN0%3D

s1=https://monitoringstorage1.blob.core.windows.net/performance-data/Data%20Engineering%20Use%20Case%20Dataset\_station1.csv?sp=r&st=2025-08-06T16:21:03Z&se=2025-08-27T00:36:03Z&sv=2024-11-04&sr=b&sig=2K%2FRneJyv5okyxLK33R1rtJ%2FKIzkKy8uIdCY2HIUdDU%3D

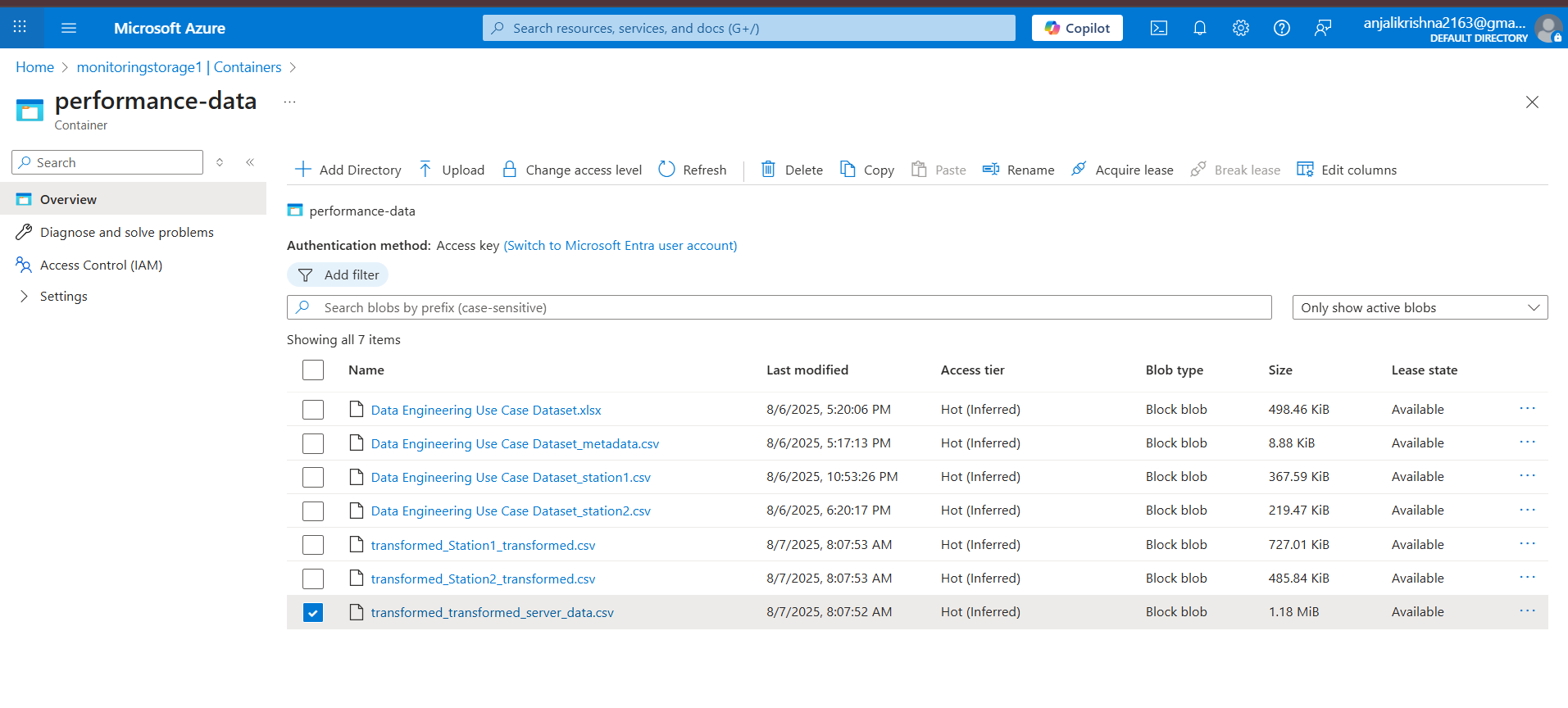
s2=https://monitoringstorage1.blob.core.windows.net/performance-data/Data%20Engineering%20Use%20Case%20Dataset\_station2.csv?sp=r&st=2025-08-06T16:22:11Z&se=2025-08-27T00:37:11Z&sv=2024-11-04&sr=b&sig=Lni0pjwFvqisPHyLdPmt0BTYsT6DcJW2hmyXNRHhSvc%3D

STEP 2: SET UP GOOGLE COLAB

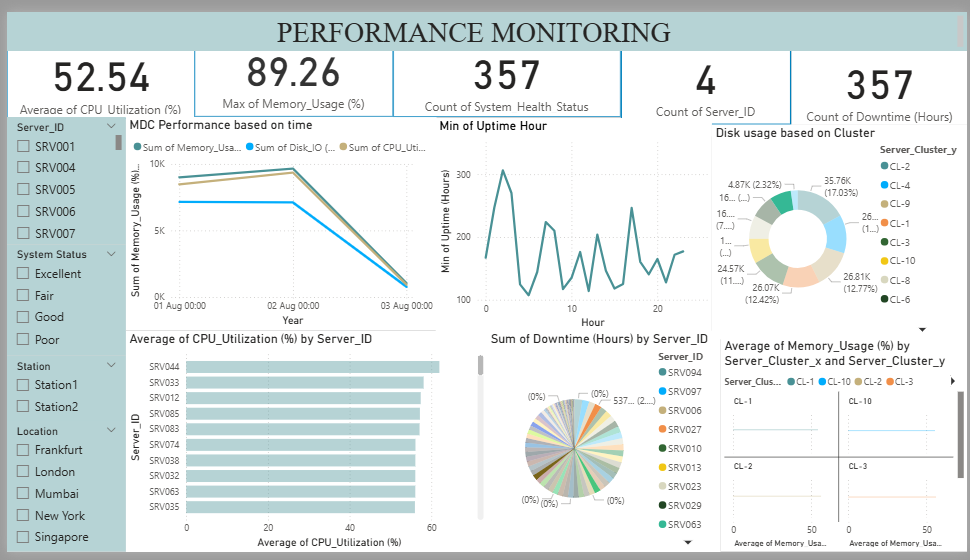
IPYNB ATTACHED FILE :

ETL PIPELINE

Etl\_pipeline.ipynb

Load the transformed data to azure  


PHASE 3:Power BI



# Scalability of the Storage Solution

For this project, I have selected Azure Blob Storage as the repo where both raw and transformed data of the server logs will be stored. This was decided on the basis of the following reasons:

* Ease of Integration: Azure Blob Storage is easy to integrate with platforms such as Azure Data Factory, Power BI, and Python scripts in Google Colab, and is thus an adaptable solution across the pipeline.
* Scalability: Being highly scalable, Blob Storage can manage future growth in volume of data — from small-scale logs to large-scale enterprise-level server monitoring data — without degrading performance.
* Security: I used a Shared Access Signature (SAS) token to provide secure, limited-access uploads of transposed data from my Python environment without the necessity for manual action or service principal setup.

# Conclusion

The end-to-end server performance monitoring solution was conceptualized and developed by utilizing Azure Data Services, data transformation via Python, and Power BI for interactive visualizations. The project effectively proved a robust pipeline from raw log ingestion to Azure Blob Storage, data cleaning, enrichment, and transformation via Google Colab. Transformed data was stored back into Azure Blob Storage or directly used for constructing meaningful Power BI dashboards.

Across this solution, data quality, transformation rules, and performance metrics like CPU usage, memory consumption, and network throughput were attended to with special care. Incorporating metadata like server region and instance type made the visualizations present richer context to the trends in performance.

The design guarantees scalability, modularity, and simplicity in future upgrades, which renders it ideal for real-time monitoring of operations and strategic analysis over the long term. This systematic design not only addresses the immediate needs of server performance monitoring but also provides a sound base for future automation, alerting, and advanced analytics integration.