ETL Pipeline: Batch Processing with Airflow

Inspired by: https://github.com/srbsnkr/ETL-pipeline-Batch-processing-with-Airflow

This project demonstrates the implementation of an ETL (Extract, Transform, Load) pipeline using Apache Airflow, Google Cloud Composer, Google Cloud Storage (GCS), BigQuery, and Cloud Functions. The pipeline automates the process of extracting trading data, transforming it, and loading it into BigQuery for analysis.

Project Structure:

ETL-pipeline-Batch-processing-with-Airflow/

├── dag.py (Defines the Airflow DAG that orchestrates the ETL workflow)

├── fetch_data.py (Contains the logic to fetch and prepare trading data for processing)

├── metaData/

    ├── bq.json (Defines the BigQuery table schema)

    └── udf.js (Contains JavaScript UDFs for data transformation)

├── cloud run function/ (Holds logic to pull csv from bucket via event trigger when files are uploaded)

    ├── main.py (Picks up every file uploaded by DAG and process it to create a dataflow job)

    └── requirments.txt (Contains package dependencies to run the main.py)

├── tradingData.csv (Sample trading data used for testing the pipeline)

└── README.md (Provides an overview and instructions for the project)

Workflow Overview

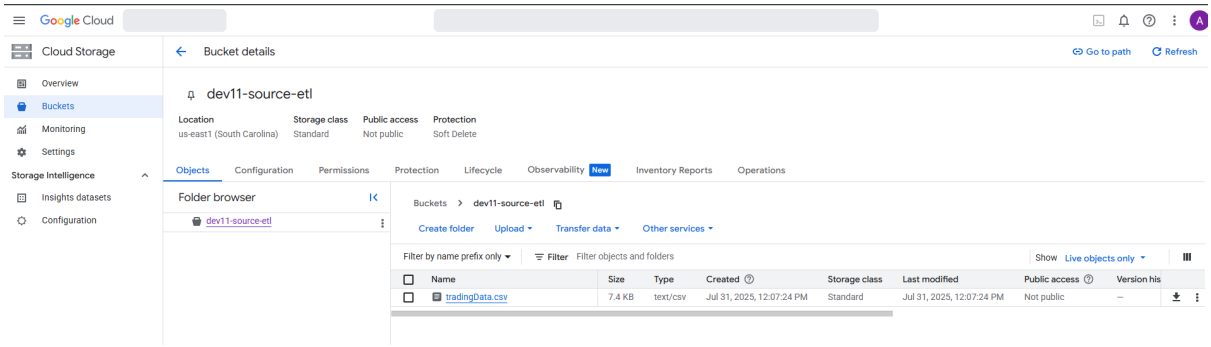
Data Ingestion: The pipeline is triggered by the presence of tradingData.csv in a designated GCS bucket.

Data Transformation: A Cloud Function is invoked to launch a Dataflow job that applies transformations using the provided UDFs.

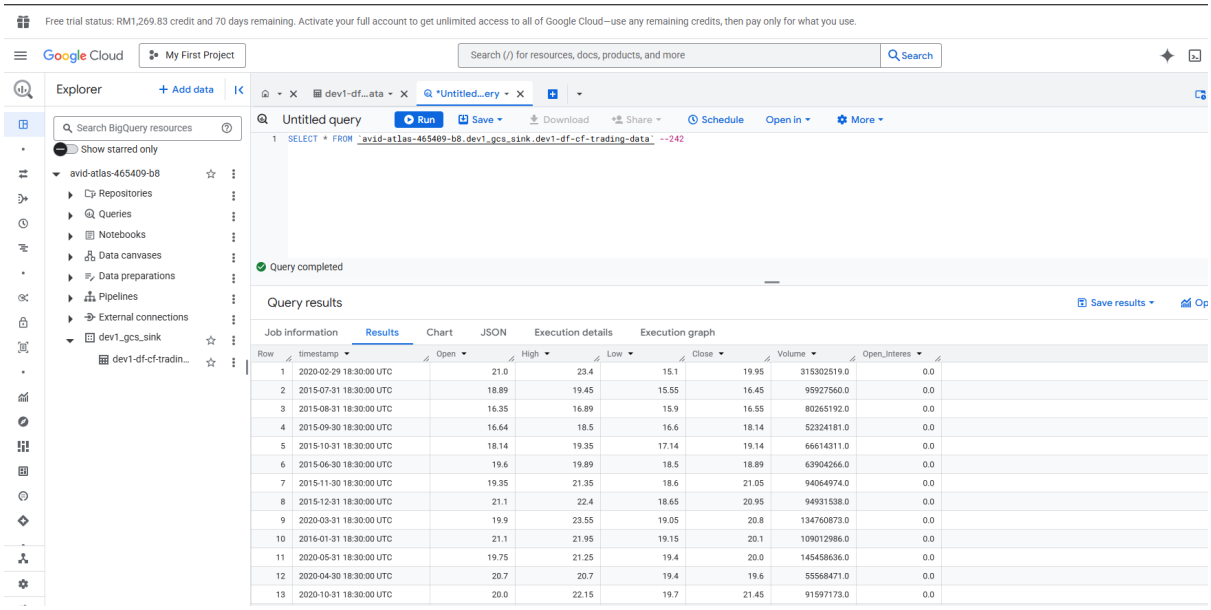
Data Loading: The transformed data is loaded into a BigQuery table as defined in bq.json.

Dashboard: The data loaded into the BigQuery table is further utilized to prepare data dashboards for different stackholders.

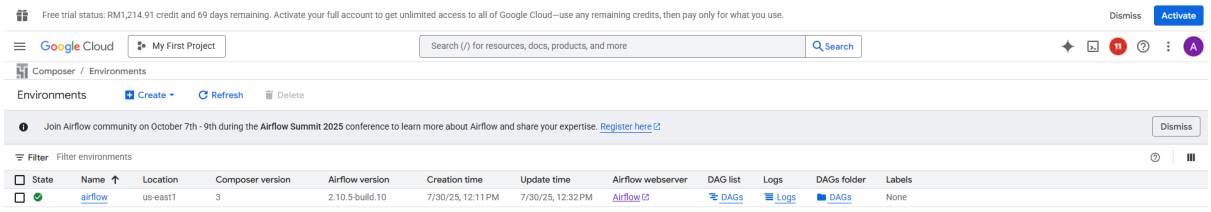
Landing Storage:



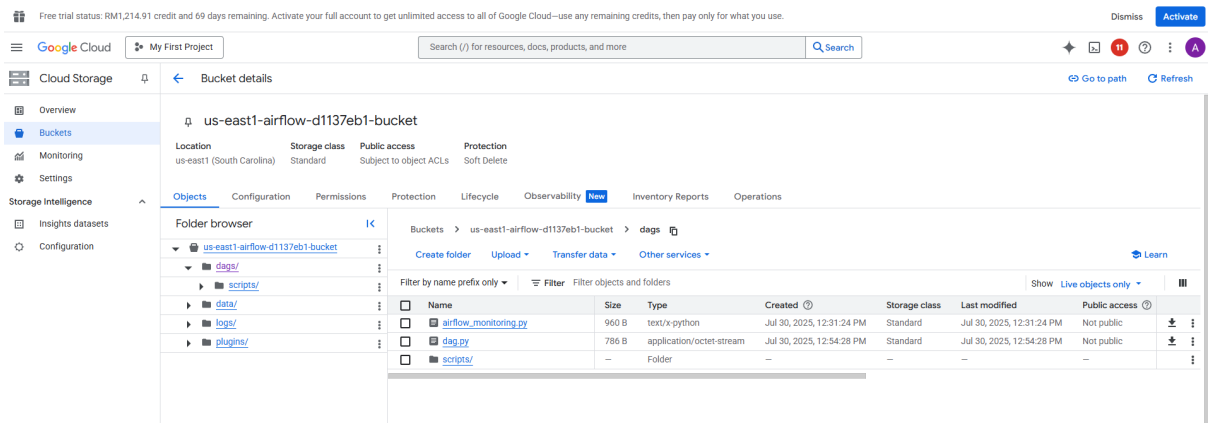
GCP Big data:



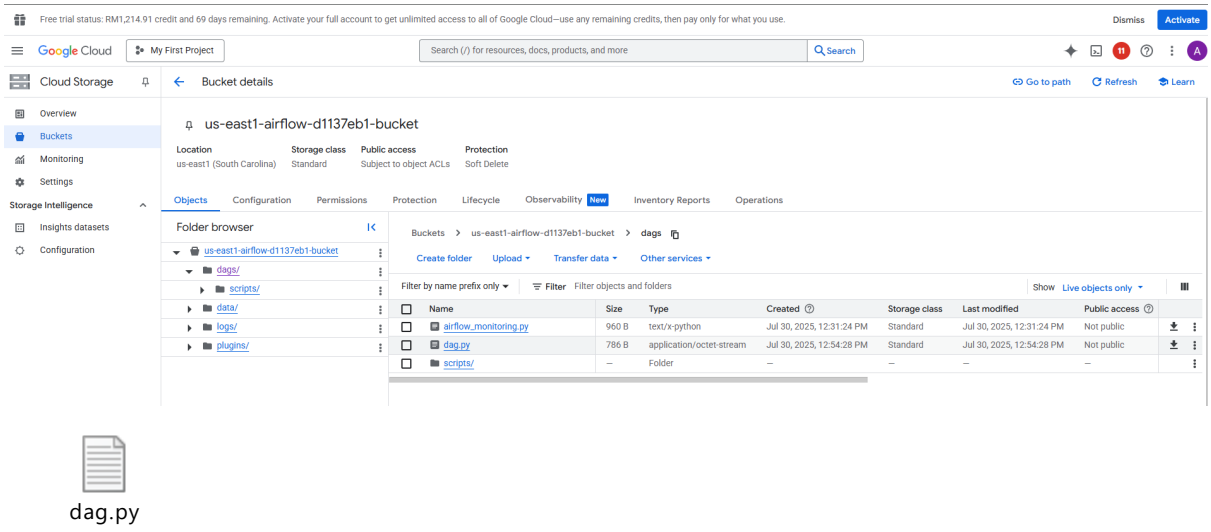
Composer:



Composer → DAG Folder:

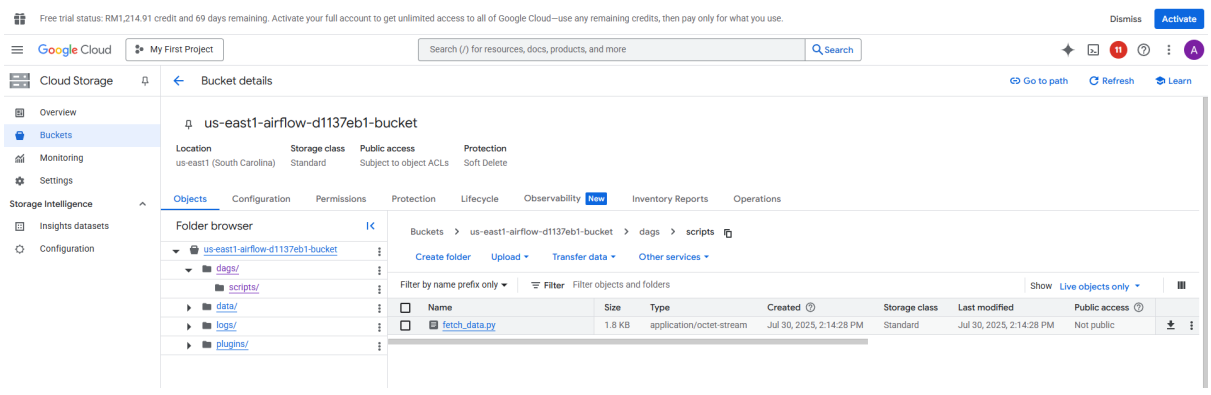


Place dag.py inside it:

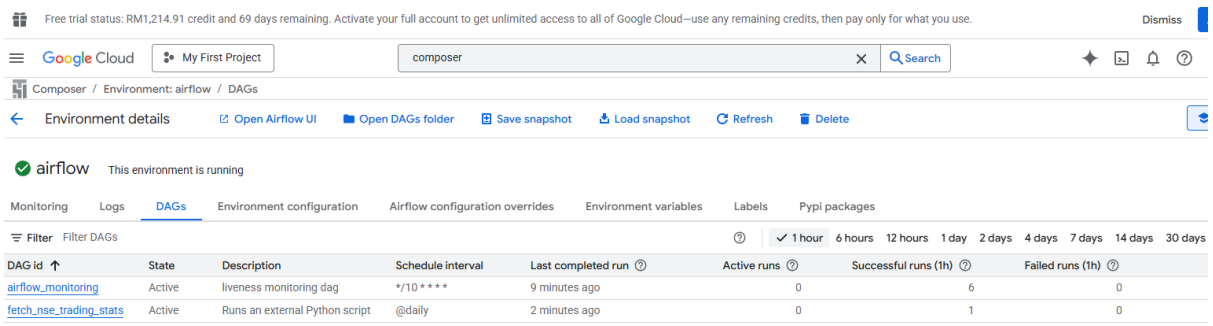


dag.py

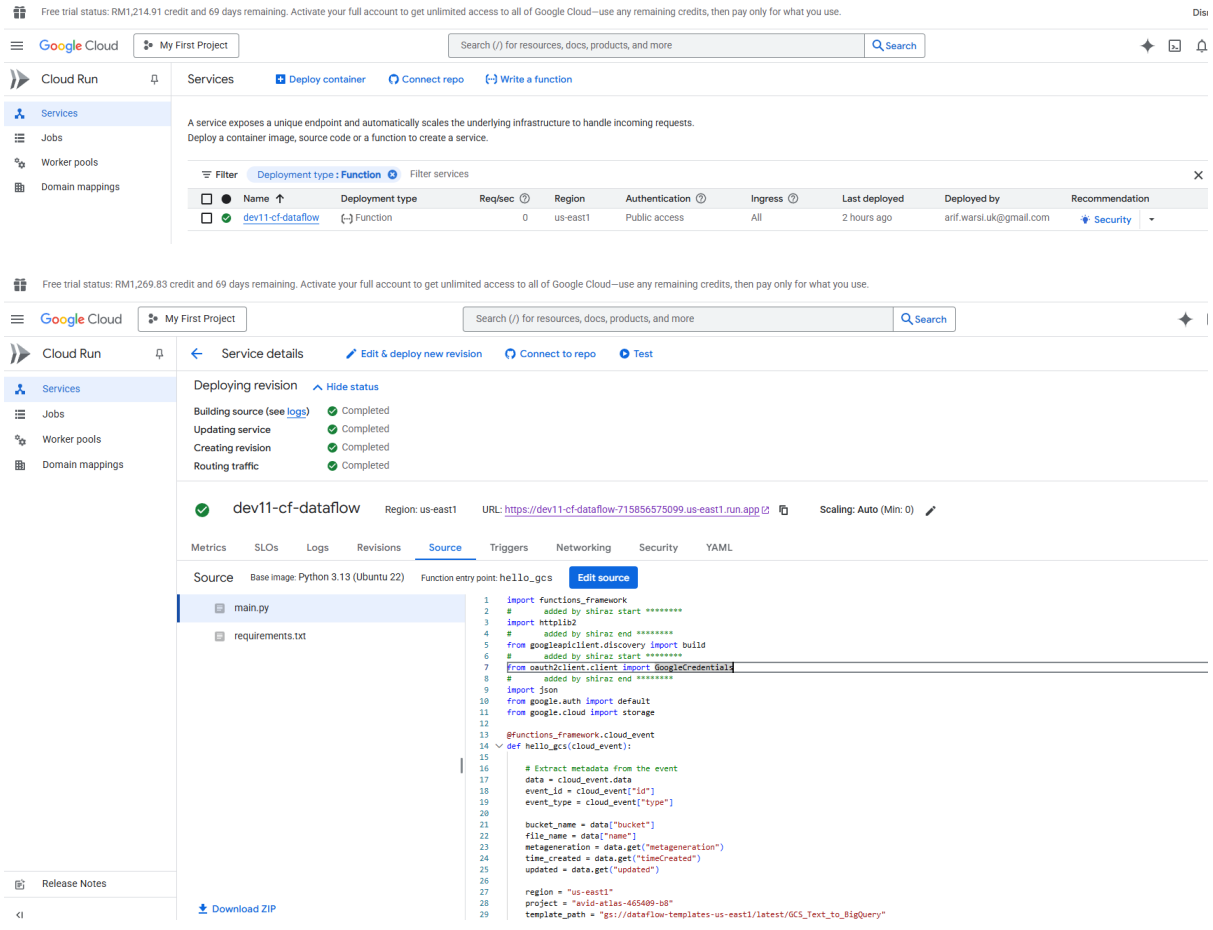
Create a new folder named scripts and place fetch_data.py inside:



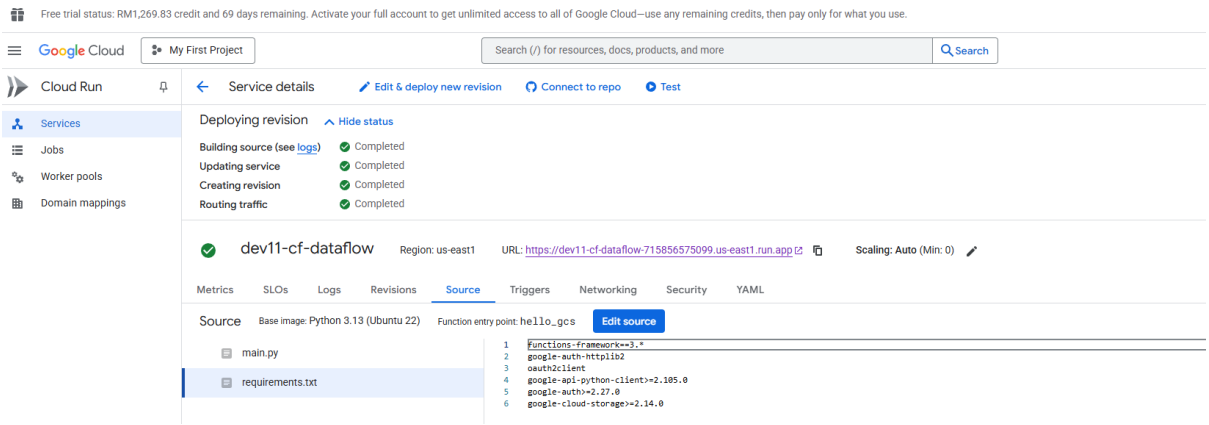
fetch_data.py



Cloud run function:

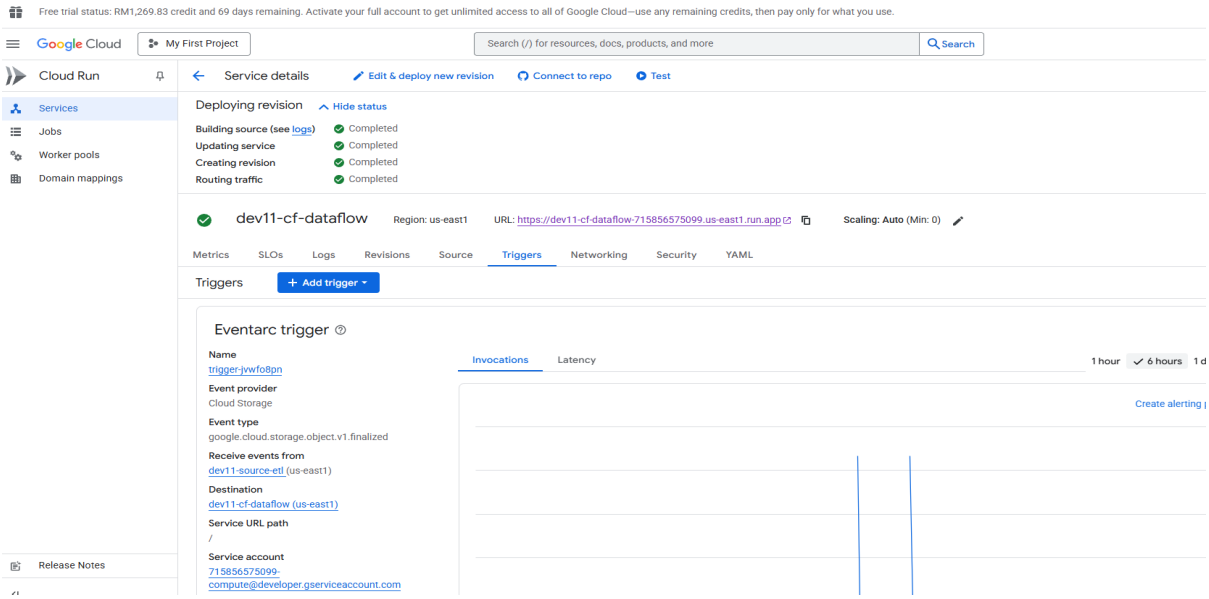


main.py

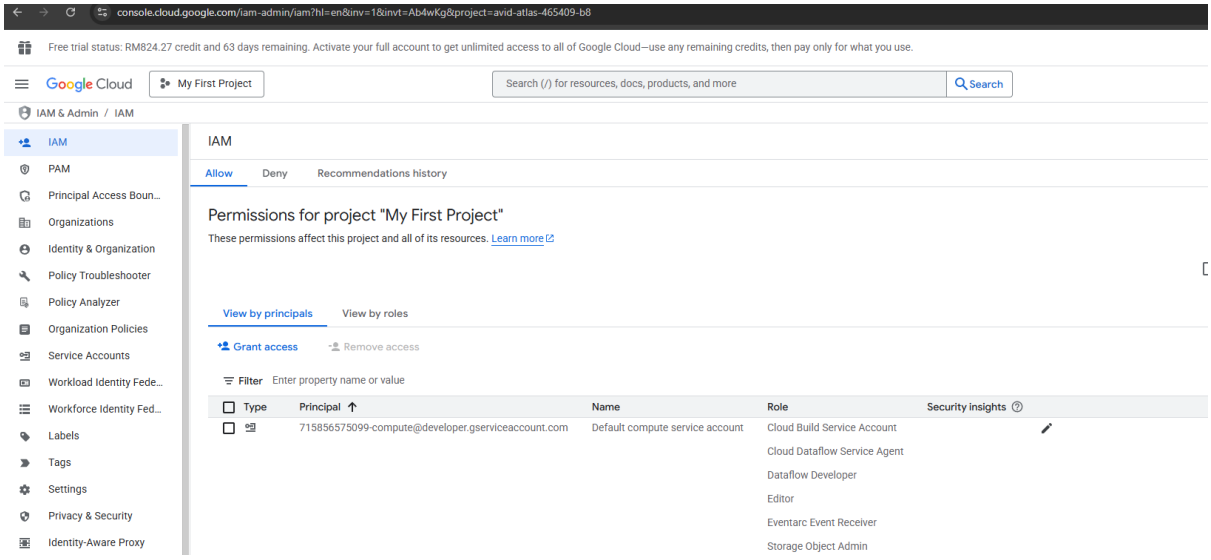


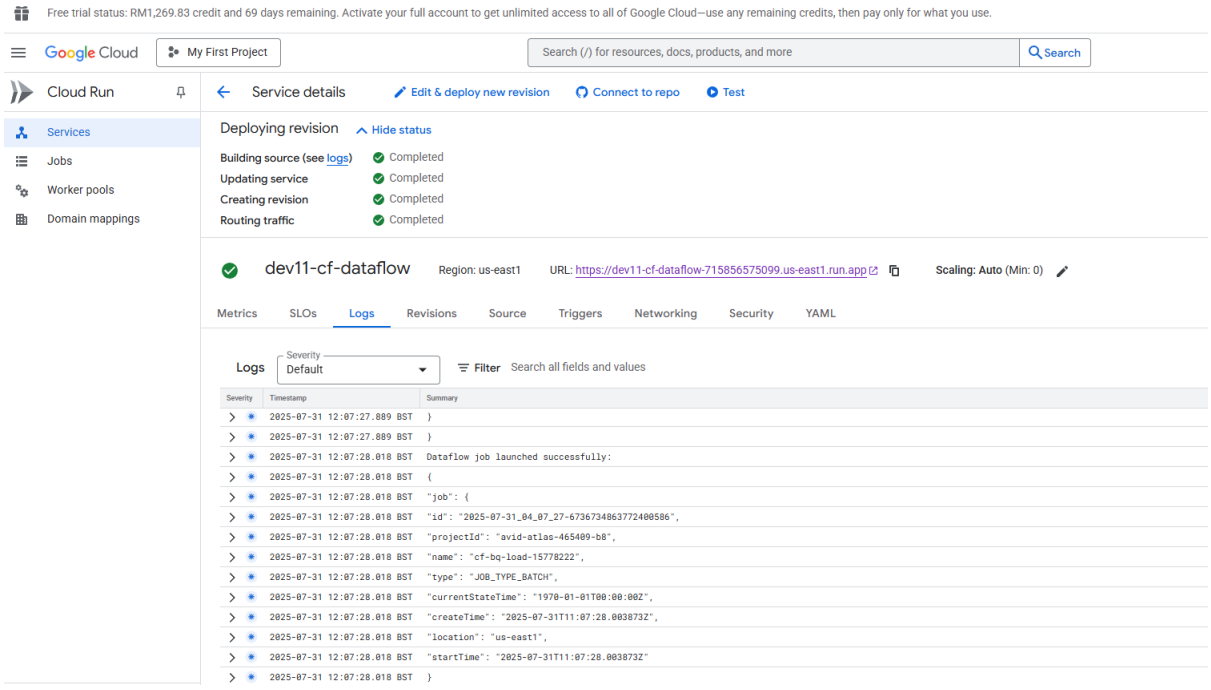
requirements.txt

Setup Trigger:



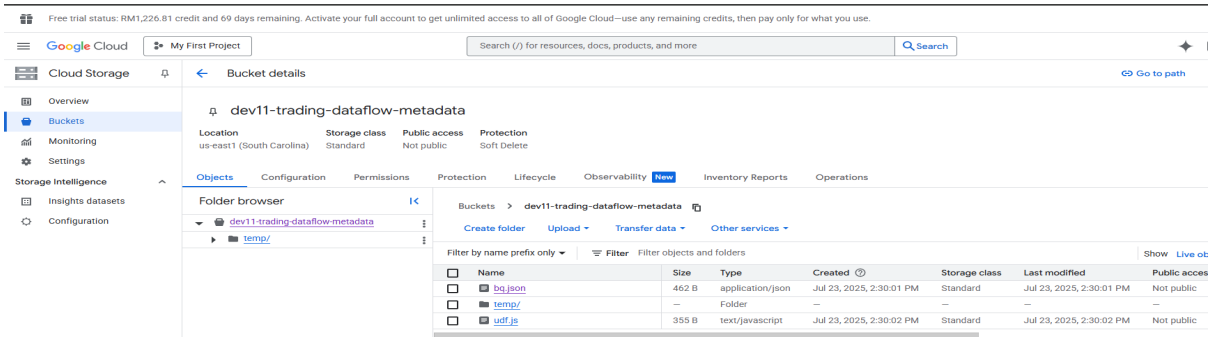
Assigning relevant roles to Service Account:





Transformation logic:

Create a bucket to hold transformation logic in files bq.json and udf.js



bq.json

udf.js

Analytics:

Display Trading data on time series graph.