(BQ-BQ) Pipeline Project Documentation

Table of Contents

---

1. Overview

This project implements a **BigQuery-to-BigQuery (BQ-BQ) pipeline** designed to automate the process of transferring, transforming, and loading data from a **public BigQuery dataset** (bigquery-public-data.fhir_synthea.claim) into a **target BigQuery dataset** (synthea_ds.accident_data) inside the project **ejazgcp**.

The pipeline is designed to:

- Copy data from the source dataset.
- Transform and load it into a partitioned and clustered table for efficient querying.
- Keep the target table up-to-date using a **MERGE operation**.

It uses **Apache Airflow** for orchestration and the **BigQueryInsertJobOperator** to execute queries.

---

2. Architecture

Architecture Flow:

1. **Source Dataset** – Public BigQuery dataset (bigquery-public-data.fhir_synthea.claim).

2. **Target Dataset** – synthea_ds in project ejazgcp.
3. **Target Table** – accident_data partitioned by event_timestamp and clustered by severity, accident_type.
4. **Airflow DAG** – Executes MERGE logic for incremental updates.
5. **BigQuery** – Handles data transformations and load.

Process Flow:

mathematica

CopyEdit

Public Dataset (claim table)

↓

BigQuery Query Job (MERGE)

↓

Target Dataset: synthea_ds.accident_data

↓

Partitioned + Clustered Storage

---

3. Tools Used

- **Google BigQuery** – Data storage, querying, partitioning, clustering.
- **Apache Airflow** – Workflow orchestration.
- **BigQueryInsertJobOperator** – To submit SQL jobs to BigQuery from Airflow.
- **SQL MERGE** – For incremental upserts.
- **Python** – For defining Airflow DAG logic.

---

4. Objective

- Create a **BQ-BQ data pipeline** to copy and transform claim data.
- Ensure the target table is **optimized for query performance** (partition + clustering).
- Maintain **up-to-date data** using incremental loading (MERGE).

- Schedule and manage the pipeline with Airflow.

---

5. Implementations

Step 1 – Project and Dataset Setup

- **Project Created:** ejazgcp
- **Target Dataset:** synthea_ds
- **Target Table Schema:**

sql

CopyEdit

```
CREATE TABLE ejazgcp.synthea_ds.accident_data (
  accident_id STRING,
  patient_id STRING,
  event_timestamp TIMESTAMP,
  location_description STRING,
  accident_type STRING,
  involved_vehicles INT64,
  injured_count INT64,
  fatalities_count INT64,
  hospital_admission BOOL,
  reported_by STRING,
  created_at TIMESTAMP,
  updated_at TIMESTAMP
)
PARTITION BY DATE(event_timestamp)
CLUSTER BY severity, accident_type;
```

Step 2 – Initial Data Load

sql

CopyEdit

```
CREATE TABLE synthea_ds.accident_data AS
SELECT * FROM `bigquery-public-data.fhir_synthea.claim`;
```

Step 3 – Incremental Load Using MERGE

The MERGE operation:

- **Matches** existing rows on accident_id.
- **Updates** if found.
- **Inserts** if not found.

sql

CopyEdit

```
MERGE `ejazgcp.synthea_ds.accident_data` T
USING (
  SELECT * FROM `bigquery-public-data.fhir_synthea.claim`
) S
ON T.accident_id = S.accident_id
WHEN MATCHED THEN
 UPDATE SET
  patient_id = S.patient_id,
  event_timestamp = S.event_timestamp,
  location_description = S.location_description,
  accident_type = S.accident_type,
  involved_vehicles = S.involved_vehicles,
  injured_count = S.injured_count,
  fatalities_count = S.fatalities_count,
  hospital_admission = S.hospital_admission,
  reported_by = S.reported_by,
  updated_at = CURRENT_TIMESTAMP
```

```
WHEN NOT MATCHED THEN
  INSERT (
    accident_id, patient_id, event_timestamp, location_description,
    accident_type, involved_vehicles, injured_count, fatalities_count,
    hospital_admission, reported_by, created_at, updated_at
  )
  VALUES (
    S.accident_id, S.patient_id, S.event_timestamp, S.location_description,
    S.accident_type, S.involved_vehicles, S.injured_count, S.fatalities_count,
    S.hospital_admission, S.reported_by, CURRENT_TIMESTAMP, CURRENT_TIMESTAMP
  );
```

Step 4 – Airflow DAG Implementation

python

CopyEdit

```
from airflow import DAG
from airflow.providers.google.cloud.operators.bigquery import BigQueryInsertJobOperator
from airflow.utils.dates import days_ago

default_args = {
    'start_date': days_ago(1),
    'retries': 1,
}

with DAG(
    dag_id='bq_to_bq_pipeline',
    default_args=default_args,
    schedule_interval=None,
```

```
    catchup=False,
    tags=['bigquery', 'example'],
) as dag:

    bq_transfer = BigQueryInsertJobOperator(
        task_id='transfer_claim_to_accident',
        configuration={
            "query": {
                "query": """<MERGE SQL ABOVE>""",
                "useLegacySql": False
            }
        },
        location='US'
    )
```

---

6. Highlights

- **Partitioned & Clustered Table** – Improves query speed and reduces cost.
- **Incremental Load** – MERGE ensures no duplicate rows and keeps data fresh.
- **Airflow Automation** – Easy scheduling and monitoring.
- **Scalability** – Handles large public dataset transfers efficiently.

---

7. Future Enhancements

- Add **Data Quality Checks** before loading.
- Integrate **Dataflow** for pre-BQ transformations.
- Add **Error Logging & Alerts** in Airflow.
- Implement **Automated Scheduling** for daily refresh.

---

8. Conclusion

This BQ-to-BQ pipeline ensures:

- **Automated, efficient, and scalable** data movement.
- **Optimized** table structure for analytics.
- **Maintainable** workflow using Airflow.