Pipeline Code Highlights — Cloud Composer DAGs

Infosys InStep Internship · automated ETL on GCP (Apache Airflow → BigQuery)

> *Curated excerpt of the production DAG code. GCP project IDs, dataset names and bucket names are redacted. Built on a public video-game sales dataset — no client data.*

```
# ============================================================================
#  Pipeline Code Highlights — GCP Cloud Composer (Apache Airflow)
#  Infosys InStep Internship · automated ETL for a public video-game sales dataset
#
#  Curated excerpt of the production DAG code.
#  NOTE: GCP project IDs, dataset names and bucket names are REDACTED.
# ============================================================================

from datetime import datetime, timedelta
import io

import pandas as pd
from airflow import DAG
from airflow.operators.bash import BashOperator
from airflow.operators.dummy import DummyOperator
from airflow.operators.python import PythonOperator
from airflow.providers.google.cloud.transfers.gcs_to_bigquery import GCSToBigQueryOperator
from airflow.providers.google.cloud.operators.bigquery import BigQueryInsertJobOperator
from google.cloud import storage

BUCKET = class="s">"<REDACTED-BUCKET>"
PROJECT_ID = class="s">"<REDACTED-PROJECT>"
DATASET_ID = class="s">"<REDACTED-DATASET>"

# Shared defaults — retries make a transient failure self-heal instead of
# breaking the whole schedule.
default_args = {
    class="s">"owner": class="s">"Mohammad Asghar",
    class="s">"depends_on_past": False,
    class="s">"start_date": datetime(2023, 1, 1),
    class="s">"retries": 1,
    class="s">"retry_delay": timedelta(minutes=5),
}


# ----------------------------------------------------------------------------
# DAG 1 — INGESTION:  raw CSV in GCS  ->  combined dataset in GCS
# ----------------------------------------------------------------------------
def ingest_data(**kwargs):
    client = storage.Client()
```

```
    bucket = client.get_bucket(BUCKET)
    blob = bucket.blob("vgsales.csv")
    df = pd.read_csv(io.BytesIO(blob.download_as_string()))

    bucket.blob("combined_dataset.csv").upload_from_string(
        df.to_csv(index=False), "text/csv"
    )


with DAG(
    "data_ingestion_pipeline",
    default_args=default_args,
    description="Ingest raw sales data from Cloud Storage",
    schedule_interval=timedelta(days=1),
    catchup=False,
) as ingestion_dag:

    # Worker dependencies are provisioned as explicit tasks BEFORE anything
    # that needs them. This is what makes the DAG reproducible on a fresh
    # Composer environment instead of failing with import errors.
    install_pandas = BashOperator(
        task_id="install_pandas",
        bash_command="pip install pandas",
    )

    install_airflow = BashOperator(
        task_id="install_airflow",
        bash_command="pip install apache-airflow",
    )

    start_task = DummyOperator(task_id="start_task")

    data_ingestion_task = PythonOperator(
        task_id="data_ingestion",
        python_callable=ingest_data,
    )

    # Provision first, then ingest.
    install_pandas >> install_airflow >> start_task >> data_ingestion_task


# ------------------------------------------------------------------------
# DAG 2 — CLEANING:  deduplicate + drop incomplete records
# ------------------------------------------------------------------------
def clean_data(**kwargs):
    client = storage.Client()
    bucket = client.get_bucket(BUCKET)
    combined_blob = bucket.blob("combined_dataset.csv")

    try:
        data = combined_blob.download_as_string()
```

```
    except Exception as e:                      # fail loudly, not silently
        raise RuntimeError(fclass="s">"Failed to download the blob: {e}")

    df = pd.read_csv(io.BytesIO(data))
    df = df.drop_duplicates()                   # de-duplication
    df = df.dropna()                            # drop incomplete records

    bucket.blob(class="s">"processed_dataset.csv").upload_from_string(
        df.to_csv(index=False), class="s">"text/csv"
    )


with DAG(
    class="s">"data_preprocessing_pipeline",
    default_args=default_args,
    description=class="s">"Deduplicate and clean the combined dataset",
    schedule_interval=timedelta(days=1),
) as cleaning_dag:
    PythonOperator(task_id=class="s">"clean_data", python_callable=clean_data)


# ---------------------------------------------------------------------------
# DAG 3 — LOAD:  GCS -> BigQuery with an EXPLICIT typed schema.
#   Defining schema_fields (rather than autodetect) is what fixed the
#   schema-mismatch failures: every column is typed and moded up front,
#   and WRITE_TRUNCATE keeps the load idempotent on re-runs.
# ---------------------------------------------------------------------------
with DAG(
    class="s">"load_to_bigquery_pipeline",
    default_args=default_args,
    description=class="s">"Load cleaned data from GCS into BigQuery",
    schedule_interval=timedelta(days=1),
) as load_dag:
    GCSToBigQueryOperator(
        task_id=class="s">"load_to_bigquery",
        bucket=BUCKET,
        source_objects=[class="s">"processed_dataset.csv"],
        destination_project_dataset_table=fclass="s">"{PROJECT_ID}.
{DATASET_ID}.sales_cleaned_data",
        write_disposition=class="s">"WRITE_TRUNCATE",      # idempotent: never duplicates
        source_format=class="s">"CSV",
        skip_leading_rows=1,                    # header row
        schema_fields=[
            {class="s">"name": class="s">"Rank",         class="s">"type": class="s">"INTEGER",
class="s">"mode": class="s">"NULLABLE"},
            {class="s">"name": class="s">"Name",         class="s">"type": class="s">"STRING",
class="s">"mode": class="s">"NULLABLE"},
            {class="s">"name": class="s">"Platform",     class="s">"type": class="s">"STRING",
class="s">"mode": class="s">"NULLABLE"},
            {class="s">"name": class="s">"Year",         class="s">"type": class="s">"FLOAT",
class="s">"mode": class="s">"NULLABLE"},
```

```
            {class="s">"name": class="s">"Genre",        class="s">"type": class="s">"STRING",
class="s">"mode": class="s">"NULLABLE"},
            {class="s">"name": class="s">"Publisher",    class="s">"type": class="s">"STRING",
class="s">"mode": class="s">"NULLABLE"},
            {class="s">"name": class="s">"NA_Sales",     class="s">"type": class="s">"FLOAT",
class="s">"mode": class="s">"NULLABLE"},
            {class="s">"name": class="s">"EU_Sales",     class="s">"type": class="s">"FLOAT",
class="s">"mode": class="s">"NULLABLE"},
            {class="s">"name": class="s">"JP_Sales",     class="s">"type": class="s">"FLOAT",
class="s">"mode": class="s">"NULLABLE"},
            {class="s">"name": class="s">"Other_Sales",  class="s">"type": class="s">"FLOAT",
class="s">"mode": class="s">"NULLABLE"},
            {class="s">"name": class="s">"Global_Sales", class="s">"type": class="s">"FLOAT",
class="s">"mode": class="s">"NULLABLE"},
        ],
    )


# ------------------------------------------------------------------------------
# DAG 4 — AGGREGATION:  build the curated summary table INSIDE BigQuery.
#   The grouping runs where the data lives, so analysts query a small
#   pre-aggregated table instead of scanning the raw load every time.
# ------------------------------------------------------------------------------
AGGREGATION_SQL = fclass="s">"""
CREATE OR REPLACE TABLE `{PROJECT_ID}.{DATASET_ID}.summary_table` AS
SELECT
  Year,
  Genre,
  SUM(NA_Sales)     AS NA_Total_Sales,
  SUM(EU_Sales)     AS EU_Total_Sales,
  SUM(JP_Sales)     AS JP_Total_Sales,
  SUM(Other_Sales)  AS Other_Total_Sales,
  SUM(Global_Sales) AS Global_Total_Sales
FROM
  `{PROJECT_ID}.{DATASET_ID}.sales_cleaned_data`
WHERE
  Year IS NOT NULL
GROUP BY
  Year, Genre
ORDER BY
  Year, Global_Total_Sales DESC;
class="s">"""

with DAG(
    class="s">"bigquery_vgsales_aggregation_dag",
    default_args=default_args,
    description=class="s">"Aggregate sales by year and genre into a summary table",
    schedule_interval=timedelta(days=1),
) as aggregation_dag:
    BigQueryInsertJobOperator(
        task_id=class="s">"bq_aggregation_task",
```

```
        configuration={class="s">"query": {class="s">"query": AGGREGATION_SQL,
class="s">"useLegacySql": False}},
    )

# ============================================================================
#  Flow:  GCS (raw)  ->  ingest  ->  clean  ->  BigQuery (typed load)
#                                                 ->  BigQuery (summary table)
# ============================================================================
```