Ch04

Orchestrating Batch Loads with Cloud Composer

CHAPTER ASSIGNMENT

Name _____ Class _____ No. _____ Date _____

> Answer every question in the space given. Show your working. You may use the chapter page to help you.

Questions from *Data Engineering with Google Cloud Platform*, 1st edition, Adi Wijaya (Packt, 2022), and from the Guru Jazzy chapter page, for classroom use.

Exercise from the book: Build data pipeline orchestration using Cloud Composer

L1 Level 1 DAG – Creating dummy workflows

The main goal of the Level 1 DAG is for us to understand how to create a DAG using Python, how timing in Airflow works, and how to handle task dependencies.

L2 Level 2 DAG – Scheduling a pipeline from Cloud SQL to GCS and BigQuery datasets

In this exercise, we will create a DAG that extracts data from Cloud SQL to our GCS bucket, and from the GCS bucket to BigQuery tables. We will use the bike-sharing tables similar to our exercise in Chapter 3, Building a Data Warehouse in BigQuery.

```
sql_export_task >> gcs_to_bq_example >> bq_to_bq
```

Try to construct the code as a DAG Python file and check with the full example in our GitHub repository. Deploy the DAG to Cloud Composer and check if the DAG is running successfully.

L3 Level 3 DAG – Parameterized variables

The main goal of the Level 3 DAG is for us to load the other tables for our bike-sharing tables. When loading more than one table, we might realize there are variables that we can parameterize to improve our DAG code. Also, in this exercise, we need to start paying more attention to the table time, since we will load trip data that has time information in it.

(a) Try to continue the code development yourself and check the example code in the GitHub repository to review.

(b) Lastly, try to continue the tasks to the transformation step, by creating `fact_trips_daily` and `dim_stations` tables. The explanation and the SQL queries of both tables can be found in Chapter 3, Building a Data Warehouse in BigQuery, in the Creating fact and dimension tables section. For the DAG, as we've discussed, you can use `BigQueryOperator` to handle these tasks.

(c) In the code, try to do the same thing with the `dim_stations` table. So, at the end of the DAG, we will have two checkers to make sure the table is loaded with records.

L4 Level 4 DAG – Guaranteeing task idempotency in Cloud Composer

The main goal of the Level 4 DAG is to make sure that our tasks are idempotent. More specifically, we will modify our tasks for loading trips tables, to make sure there will be no duplication when we rerun the DAG.

L5 Level 5 DAG – Handling late data using a sensor

The main goal of the Level 5 DAG is for us to understand how to handle late data. Handling late data means being able to create a flexible DAG runtime depending on the upstream data arrival.

Imagine you have two DAGs. The first DAG loads data from GCS to a BigQuery raw table, as follows:

GCS → BigQuery raw table → BigQuery DWH tables

The second DAG will load data from the BigQuery raw table to a data mart table, as follows:

BigQuery DWH tables → BigQuery data mart

Here are the two scenarios:

- The data sources from GCS are ready at 5:00 A.M.
- The requirement is to have a data mart table ready by 6:00 A.M.

(a) What can go wrong in this scenario?

(b) That's it—with these setups, your second DAG will start only if `level_5_dag` successfully runs. Please check the full code in the example code. Try to build a solution yourself first and compare it with the example code.

Sorting activities from the chapter page

S1 Backfill, rerun or catchup?

Options: **Backfill** · **Rerun** · **Catchup**

No.	Scenario	Answer
1	A pipeline has run daily for a week. The business now wants the previous month loaded too, without disturbing the running schedule.	
2	Last Tuesday's load failed. You click Clear on that run in the Airflow UI.	
3	You deploy a new DAG whose start date is two weeks ago, and Airflow immediately launches runs for every missed day.	

S2 Is this task idempotent?

Options: **Idempotent** · **Not idempotent**

No.	Scenario	Answer
1	Each daily run appends that day's events to the whole table with WRITE_APPEND. A run is repeated.	
2	Each daily run writes with WRITE_TRUNCATE to only that day's partition of a date-partitioned table.	
3	Each run truncates and reloads the entire 1 TB table from all files.	

S3 **Good practice or avoid?**

Options: **Good practice** · **Avoid**

No.	Scenario	Answer
1	Use the BigQuery native operators for every load and transform step.	
2	Download a Cloud Storage file to the worker's disk with the storage client, then process it in Python.	
3	Set `priority='BATCH'` on BigQuery query tasks in pipelines.	
4	Load a BigQuery table into a pandas DataFrame inside a task for a quick calculation.	

S4 **Where should this value live?**

Options: **DAG variable** · **Airflow variable** · **Environment variable**

No.	Scenario	Answer
1	A bucket name and dataset names shared by many DAGs, editable in the Airflow UI as one JSON value.	
2	A constant that only one DAG needs, written at the top of its script.	
3	A Cloud SQL instance name supplied when the Composer environment is created or updated.	

Quiz from the chapter page: Check yourself

1 What is a DAG in Airflow?

A. A drag-and-drop dashboard

B. A workflow: tasks chained by dependencies, with a schedule

C. A BigQuery table type

D. A Composer machine type

---

2 Where must a DAG file go for Composer to pick it up and run it?

A. The dags/ folder of the environment's Cloud Storage bucket

B. Pasted into the Airflow web UI

C. The BigQuery console

D. A Git branch named dags

---

3. Why is it a bad idea to process large data with pandas inside a DAG task?

   A. Pandas cannot read CSV files

   B. Airflow forbids Python

   C. Composer workers have limited memory and disk and are not built to process data

   D. It is always slower than SQL on small data

---

4. A run for one day fails and you rerun it. Which load design leaves the table correct?

   A. WRITE_APPEND to the whole table

   B. Manually delete duplicates afterwards

   C. WRITE_TRUNCATE to that day's partition only

   D. Skip the day

---

5. Why use a sensor between dependent DAGs instead of a later fixed schedule?

   A. Sensors are cheaper than schedules

   B. A fixed schedule cannot use cron

   C. It waits for the real condition, so a late upstream job cannot cause a blind start

   D. Sensors make tasks run faster

---

6. You want a daily-midnight DAG to run for today's date as soon as you deploy it. What start date follows Airflow's rule?

   A. Tomorrow

   B. Today

   C. Yesterday

   D. Any date; start dates are ignored

End of assignment · Check your answers against the chapter at gurujazzy.com/subjects/gcp-data-engineering/chapter-04-orchestration-with-cloud-composer/