Cloud Composer v2 - Michaël Bettan

Definition: Managed Apache Airflow workflow orchestration service. Apache Airflow is an open source platform to orchestrate data pipelines programmatically in Python.	**Availability**: Regional

Use Cases:

- Orchestration of data ingestion, including hybrid use cases
- Data processing
- Orchestration of proprietary API endpoints jobs
- Custom code tasks run directly in Airflow
- ML / AI pipelines

Billing:

- Web, Database **core hours**
- Database SQL **core hours**
- Web and Data **storage**
- **Network egress** for all
- Workers and Scheduler **GKE nodes**
- **GCS Bucket** (Dags)
- **Cloud Monitoring** logs usage

Key Capabilities

- *Author, Schedule and Monitoring within Airflow*
- *Composer v2 with Python 3 is the current standard*
- *OSS making a great fit hybrid & multi-cloud architecture*

Airflow DAGs

- ***DAG**: Directed Acyclic Graphs. Acyclic = not a circle*
- ***Workflow pipeline** and **DAG** are interchangeable*
- ***DAGs**: executed workflows*
  - *Collection of tasks with dependencies/relationships*
  - *Stored in Cloud Storage*
  - *Supports custom plugins: operators, hooks and interfaces*
  - *Python dependencies (modules)*
- *Example: Export BigQuery data → Move GCS folder → Cleanse data → Ingest in another system (e.g., Snowflake) → Send Slack Message*
- *DAGs folder is a Cloud Storage bucket where you will load your pipeline code*

Airflow Operators

- *Core building blocks of DAGs. **Airflow operators** invoke the TASKS you want to complete. Operators are usually atomic in a task (one operator per task).*
- *Represent individual tasks within a workflow.*
- *Types:*
  - ***DummyOperator**: visual clarity(start/end points).*
  - ***BashOperator**: Executes bash commands.*
  - ***PythonOperator**: Executes Python code.*
  - ***EmailOperator**: Sends emails.*
  - ***GoogleCloudStorageToBigQueryOperator**: Transfers data from Cloud Storage to BigQuery.*
  - ***BigQueryToCloudStorageOperator**: Exports data from BigQuery to GCS. Pay attention to the different export formats (Avro, Parquet) and compression options (Snappy, Deflate) for cost optimization.*
  - ***GoogleCloudStorageToGoogleCloudStorageOperator**: Copies files between GCS buckets.*
  - ***Many others**: for interacting with other services (e.g., DataProc, Dataflow, Pub/Sub).*

Architecture

- ***Environment** is the instance construct, deployed automatically with several components:*
  - *Customer project → GKE, Cloud Storage*
  - *Google-managed tenant project → Cloud SQL, Identity-Aware-Proxy, App Engine Flexible*
- ***Customer project** is where you create your environments. You can create more than one environment in a single customer project.*
- ***Google-managed tenant project** provides unified access control and an additional layer of data security for your environment. Each environment has its own tenant project.*
- ***Airflow web server** on App Eng Flex is the Airflow UI*
- ***Airflow database** on Cloud SQL: stores the metadata*
- ***Environment's bucket** on GCS to store DAGs, plugins, data dependencies, and Airflow logs.*
- ***Airflow scheduler** controls the scheduling of DAG runs and individual tasks from DAGs. Distribute tasks to Airflow workers by using a Redis queue, as a GKE deployments.*
- ***Airflow workers** execute individual tasks from DAGs by taking them from the Redis queue. Airflow workers run as GKE deployments.*
- ***Redis queue** holds a queue of individual tasks from your DAGs. Airflow schedulers fill the queue; Airflow workers take their tasks from it. Redis queue runs as a GKE StatefulSet application, so that messages persist across container restarts.*
- *For each Private IP environment, Cloud Composer creates one **VPC peering connection** for the tenant project network*

Logging & Monitoring

- *2 types of logs: operational (system) and task*
- *Operational logs (Scheduler) -- Logs Viewers in the console (from Cloud Monitoring*
- *Each DAG task has an associated log via Airflow UI, Logs folder (GCS)*
- *Monitoring natively from the Environment UI directly*

Security

- *Composer Administrator: full control of resources*
- *Environment and Storage Object Administrator*
- *Environment User and Storage Object Viewer*
- *Composer User*

Failure and Retry Mechanisms

Airflow provides robust mechanisms to handle task failures.:

- *Users can define **retry logic within their DAGs**, specifying the **number of retry attempts** and the **time interval between retries.***
- *On failure, Airflow can **trigger alerts and store logs** for debugging, enabling automated recovery and efficient troubleshooting*
- *Specific operators may also offer **customized failure handling options**.*
- *Tasks in a DAG can fail for various reasons. The **on_failure_callback parameter** is designed to handle situations where a task fails, allowing custom logic (such as sending notifications) to be executed when the task does not succeed.*

ETL: Push vs. Pull patterns

- ***Push-based ETL with Cloud Functions and Airflow:** Data ingestion is initiated by an external event (e.g., a new file in Cloud Storage, a Pub/Sub message). This event triggers a Cloud Function, which in turn triggers the execution of a specific Airflow DAG responsible for the ETL process. This approach is **reactive** and **ideal for near real-time data processing** where the ETL process is dependent on the arrival of new data. Cloud Function acts as a lightweight intermediary, decoupling the event source from the more resource-intensive Airflow DAG execution. Efficient for handling variable data volumes and maintaining responsiveness.*
- ***Pull-based ETL with Scheduled Airflow DAGs:** initiated on a **predefined schedule** (e.g., cron expression) directly within Airflow. Airflow Scheduler autonomously checks for scheduled DAG runs and executes them at the designated times. This is a proactive approach best suited for **batch processing** of data at **regular intervals** where data volume is more predictable. It's simpler to manage than push-based systems but might be less responsive to immediate data changes.*