<https://github.com/apache/airflow/tree/main/airflow>

<https://cloud.google.com/composer/docs/quickstart>

Airflow:

<https://airflow.apache.org/docs/apache-airflow-providers-google/stable/index.html>

<https://airflow.apache.org/docs/apache-airflow-providers-google/stable/operators/cloud/index.html>

<https://airflow.apache.org/docs/apache-airflow-providers-google/stable/operators/cloud/bigquery.html>

<https://airflow.apache.org/docs/apache-airflow-providers-google/stable/operators/cloud/dataflow.html>

<https://airflow.apache.org/docs/apache-airflow/stable/plugins.html>

Cloud Composer:

<https://cloud.google.com/composer/docs/concepts/airflow-configurations>

<https://cloud.google.com/composer/docs/concepts/overview>

**Cloud Composer**

* A fully managed workflow orchestration service built on Apache Airflow. (another Apache big data service adopted by Google Cloud.)
* Cloud Composer is built using Apache Airflow. This should emphasize the importance of the Apache ecosystem in the world of big data. But once again, using Apache tools is made easier by GCP offering them as a managed service.
* Airflow is a task orchestration system designed to automate complex interdependent tasks into pipelines or workflows.
* Airflow itself is written in Python and each workflow is also written in Python allowing it to immediately benefit from Python's close relationship with the big data community as well as making it highly extensible.
* Airflow provides central management and scheduling of all workflows, managing their tasks on an array of distributed workers.
* Airflow also provides an extensive CLI tool as well as a comprehensive web UI for creating and managing workflows. At the heart of an Airflow workflow is the directed acyclic graph or dag, which has its origins in math and graph theory.
* A dag is basically a graph consisting of nodes connected by edges. In this graph, you can think of the edges as how we traveled from one node to another. Because it's directed we always travel in one direction and it's acyclic. Which is just a fancy way of saying that we never circle back. It's impossible to reach the same node more than once by traversing along the edges. With these features it's possible to use dag's that represent dependencies between nodes that must be traversed in a specific order.
* Airflow uses dag's to represent a collection of all the tasks in a workflow organized in a way that shows their relationships and dependencies. And the dag itself is represented in Python code.
* The important thing about the dag is it's purely about when the tasks should be aligned to execute. It makes sure that whatever they do happens at the right time and in the right order.
* Inside the tasks of the dag we find operators which are used to actually specify what is to be done. Operators perform an action such as execute some code or read and write from storage or interact with another GCP service.
* In addition to the dag, our workflow can contain parameters about when it should run, what its dependencies are and who should be notified once it has been completed.
* Cloud Composer or Airflow under the hood manages the resources required to ensure that the

workflow completes successfully.

* Difference between Airflow and Dataflow:
  + **Dataflow** the workflow tool that we use for processing data pipelines? And it is definitely one of the tools in our tool bag. But while Dataflow is specifically about processing batch or streaming data using the features of the Apache Beam SDK.
  + **Airflow or Cloud Composer** is an orchestration tool that can use any Python code at any stage of the pipeline. You can think of it more as a scheduler than as a transformative part of the pipeline itself.
* Let's look at some simple examples of Cloud Composer workflows. In this example, a workflow runs daily, sets up a Dataproc cluster, performs some Spark analytics, writes the results to GCS and emails an administrator and then deletes the Dataproc cluster.
* In another example, a weekly scheduled workflow could query an external data source and then

use the insights to generate and send out a marketing email or newsletter again by contacting an external system such as SendGrid.

* Maybe you have an hourly schedule to check for new data dumps in GCS which can then be fed

through a data flow instance and out into Bigtable. And in fact, while Airflow is ideally suited for

data pipelines it can also be used for many other scheduled automation tasks outside of big data

like operational infrastructure automation.

* You could use Airflow to run some deployment manager templates to stand up a test GCP environment for example. Again, the limit is really your imagination as each workflow is entirely customizable with the full back catalog of supported Python modules and features.

**Cloud Composer Architecture:**

* Apache Airflow is implemented as a microservices architecture. It's deployed using numerous GCP resources grouped together into a Cloud Composer environment.
* You can have more than one Composer environment inside a GCP project, but each environment is an isolated installation of Airflow and all of its component parts.
* Some of these parts get deployed into a tenant project. This is a Google-managed project

that you can't see or control.

* In here, Composer places the Airflow database running on Cloud SQL and the Airflow web server, which provides its web UI running on App Engine Flex.
* It will also configure the Identity-Aware Proxy to control access to the web server.
* Meanwhile, in your primary project, Composer will create a Google Kubernetes Engine cluster.

In this cluster, it will create deployments for Redis, the Airflow scheduler, and the Airflow workers, along with a Cloud SQL proxy. It will also create 2 Pub/Sub topics for messaging between the various microservices and a Cloud storage bucket for logs, plugins, and most importantly, the DAGs themselves.

* Cloud Composer automatically configures parameters and environment variables for Apache Airflow, but you can also customize some parameters, although not all of them, when you create the environment.

