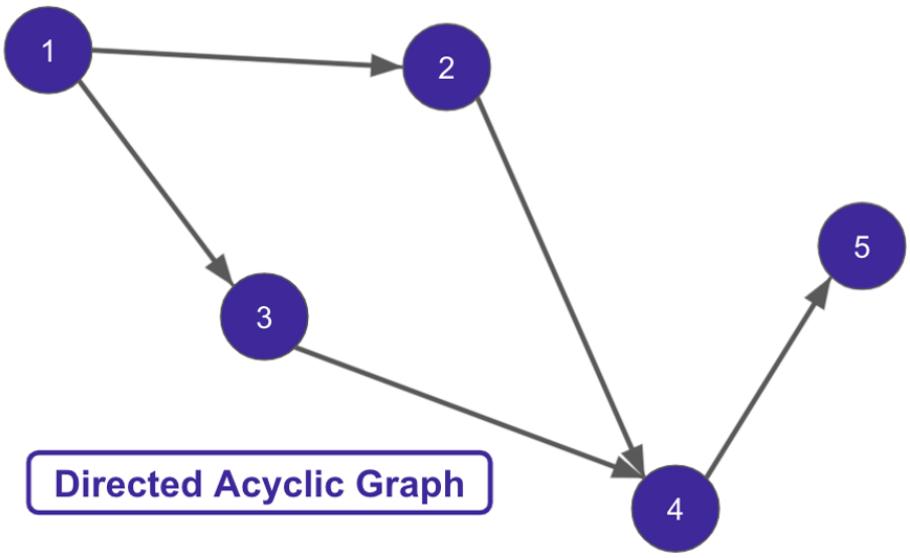
Google Cloud Composer

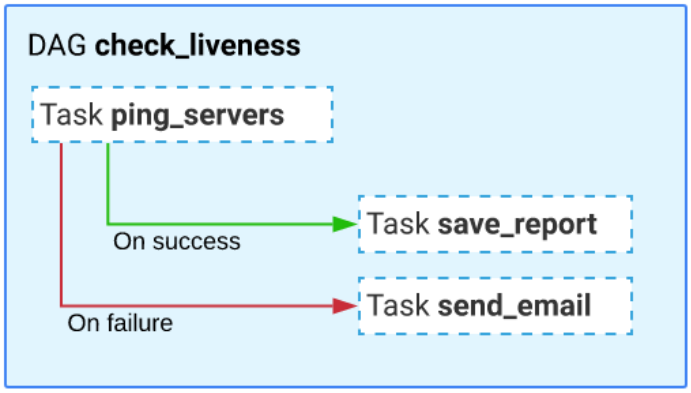
- Cloud Composer is a scalable, fully managed **workflow orchestration** service, enabling to create, schedule, monitor, and manage workflow pipelines that span across clouds, multi-cloud, hybrid and on-premises data centers.
- Cloud Composer is built on the popular Apache Airflow open source project and operates using the Python programming language.
    - Airflow is a **micro-service architected framework**.
- It integrates with several other Google products like Google BigQuery, Cloud Dataflow & Dataproc, Cloud Storage, Cloud Pub/Sub and other GCP services via well-defined APIs.
- Google Cloud Composer offers Simplicity, portability and easy deployment.
- Can create one or more Cloud Composer environments inside of a project.
- Cloud Composer automation helps you create managed Airflow environments quickly and use Airflow-native tools, such as the powerful Airflow web interface and command-line tools, so you can focus on your workflows and not your infrastructure.
- With Airflow, one can programmatically schedule and monitor workflows.
- Environments are self-contained Airflow deployments based on GKE.Create Cloud Composer environments in supported regions.
- Airflow communicates with other Google Cloud products through the products' public APIs.
- In data analytics, a **workflow** represents a series of tasks for ingesting, transforming, analyzing, or utilizing data. In Airflow, workflows are created using DAGs, or "Directed Acyclic Graphs".
    - DAG's are written in Python 3.x

Directed Acyclic Graphs

- An Apache Airflow DAG is a workflow: **a collection of tasks with additional task dependencies**.
- 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.



- A DAG is a collection of tasks that you want to schedule and run, organized in a way that reflects their relationships and dependencies.
- DAGs are created in Python scripts, which define the DAG structure (tasks and their dependencies) using code.
- A DAG should not be concerned with the function of each constituent task —its purpose is to ensure that each task is executed at the right time, in the right order, or with the right issue handling.
- DAGs are composed of several components, such as DAG definition, operators, and operator relationships.
- A task in a DAG is a function to be performed, or a defined unit of work. Such functions can include monitoring an API, sending an email, or executing a pipeline.



- A task instance is the individual run of a task. It can have state information, such as “running,” “success,” “failed,” “skipped,” etc.

- Each task in a DAG can represent almost anything—for example, one task might perform any of the following functions:
    - Preparing data for ingestion
    - Monitoring an API
    - Sending an email
    - Running a pipeline

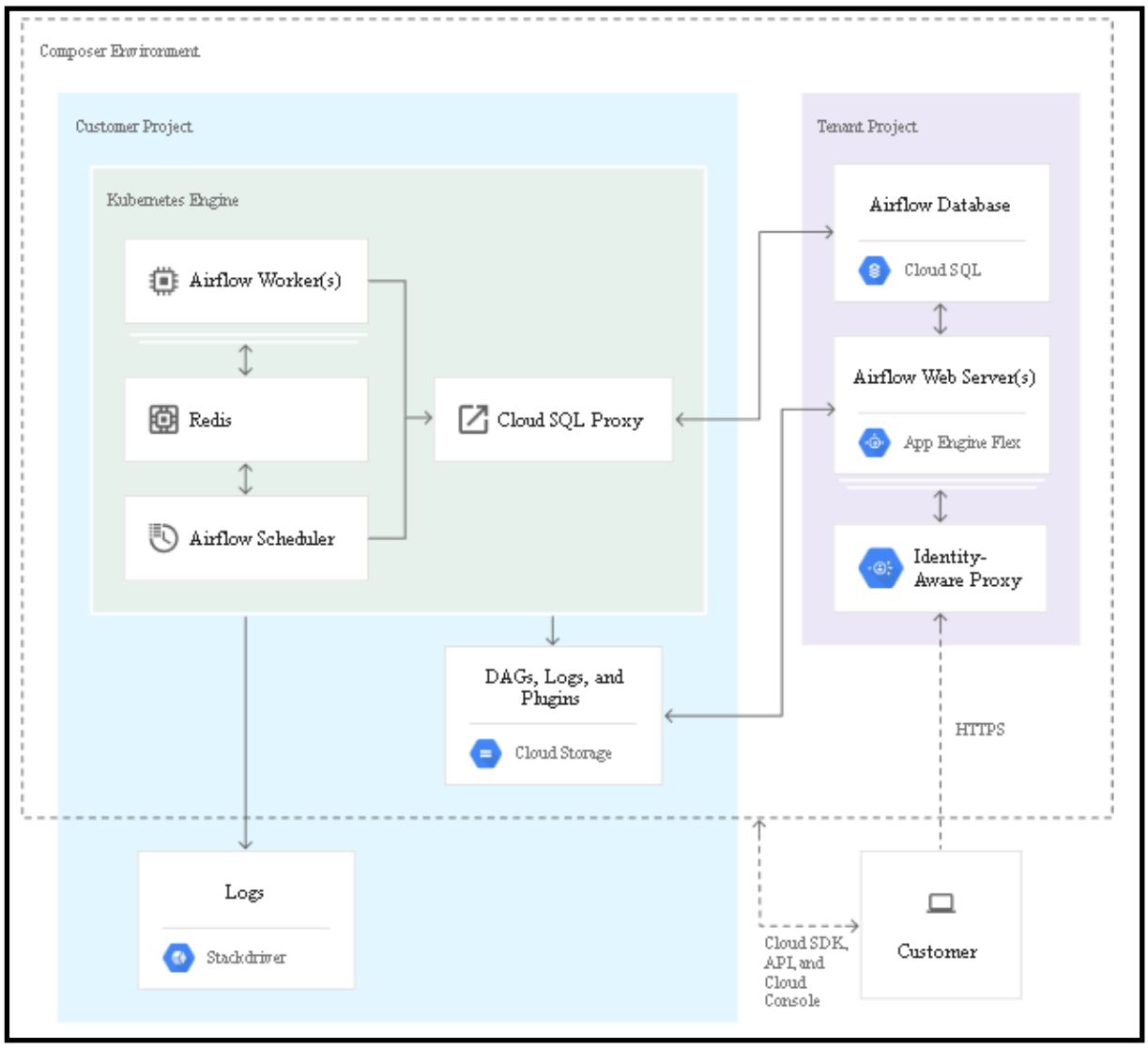
Composer Architecture

- Composer will create a Google Kubernetes Engine cluster. In this cluster, it will create deployments for
    - o Redis
    - o Airflow scheduler
    - o Airflow workers
    - o Cloud SQL proxy.
- It will also create 2 Pub/Subtopics 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 we can customize some parameters when we create the environment.

Cloud Composer creates the following components for each environment:

- **GKE cluster**
    - The Airflow schedulers, workers, and Redis Queue run as GKE workloads on a single cluster, and are responsible for processing and executing DAGs.
    - The cluster also hosts other Cloud Composer components like Composer Agent and Airflow Monitoring, which help manage the Cloud Composer environment, gather logs to store in Cloud Logging, and gather metrics to upload to Cloud Monitoring.
- **Web server**:
    - The web server runs the Apache Airflow web interface, and Identity-Aware Proxy protects the interface.

- **Cloud SQL Database**:
  - Cloud SQL stores the Airflow metadata. Composer backs up the Airflow metadata daily to minimize potential data loss.
  - Only service account used to create the Composer environment can access data in the Cloud SQL database.
- **Cloud Storage bucket**:
  - Cloud Composer associates a Cloud Storage bucket with the environment. The associated bucket stores the DAGs, logs, custom plugins, and data for the environment.



- **Redis:**
  - message broker for the CeleryExecutor, runs as a StatefulSet application so that messages persist across container restarts.

- **Cloud Logging and Cloud Monitoring:**
  - Composer integrates with Cloud Logging and Cloud Monitoring, to view all Airflow service and workflow logs.

Cloud Composer Environment Component

Components for each environment:

- **Web server:**
  - The web server runs the Apache Airflow web interface, and Identity-Aware Proxy protects the interface.
- **Database:**
  - The database holds the Apache Airflow metadata.
- **Cloud Storage bucket:**
  - bucket stores the DAGs, logs, custom plugins, and data for the environment.

Airflow management:

Use following Airflow-native tools for management

- **Web interface:**
  - Access Airflow web interface from the Google Cloud Console or by direct URL.
- **Command line tools:**
  - run gcloud composer commands to issue Airflow command-line commands.
- Cloud Composer REST and RPC APIs.

Airflow configuration:

- Composer configurations provides for Apache Airflow are the same as the configurations for a locally-hosted Airflow deployment.
- Some Airflow configurations are preconfigured. Some cannot be changed.
- Other configurations, to be specified when creating or updating environment.

Airflow DAGs (workflows):

- An Apache Airflow DAG is a workflow: a collection of tasks with additional task dependencies.
- Cloud Storage used to store DAGs.
- To add or remove DAGs add or remove the DAGs from the Cloud Storage bucket
- Can schedule DAGs
- can trigger DAGs manually or in response to events

Plugins:

- can install custom plugins, into Cloud Composer environment.

Python dependencies:

- can install Python dependencies from Python Package Index.

Access control:

- manage security at the Google Cloud project level
- assign Cloud IAM roles for control.
- Without appropriate Cloud Composer IAM role, no access to any of environments.

Logging and monitoring:

- can view Airflow logs that are associated with single DAG tasks
- View in the Airflow web interface and
- the logs folder in the associated Cloud Storage bucket.
- Streaming logs are available for Cloud Composer.
- access streaming logs in Logs Viewer in Google Cloud Console.
- Also has audit logs, such as Admin Activity audit logs, for Google Cloud projects.

Networking and security:

During environment creation, following configuration options available

- Cloud Composer environment with a route-based GKE cluster (default)
- Private IP Cloud Composer environment

- Cloud Composer environment with a VPC Native GKE cluster using alias IP addresses
- Shared VPC

Create a Project:

To create a project and enable the Cloud Composer API:

- In the Cloud Console, select or create a project.
- Make sure that billing is enabled for project.
- To activate the Cloud Composer API in a new or existing project, go to the API Overview page for Cloud Composer.
- Click Enable.

Feature of Cloud Composer

Multi-cloud	Create workflows that connect data, processing, and services across clouds, giving you a unified data environment.
Open source	Cloud Composer is built upon Apache Airflow, giving users freedom from lock-in and portability.
Hybrid	Ease your transition to the cloud or maintain a hybrid data environment by orchestrating workflows that cross between on-premises and the public cloud.
Integrated	Built-in integration with BigQuery, Dataflow, Dataproc, Datastore, Cloud Storage, Pub/Sub, AI Platform, and more, giving you the ability to orchestrate end-to-end Google Cloud workloads.
Python programming language	Leverage existing Python skills to dynamically author and schedule workflows within Cloud Composer.
Reliability	Increase reliability of your workflows through easy-to-use charts for monitoring and troubleshooting the root cause of an issue.
Fully managed	Cloud Composer's managed nature allows you to focus on authoring, scheduling, and monitoring your workflows as opposed to provisioning resources.

Networking and security	During environment creation, Cloud Composer provides the following configuration options: Cloud Composer environment with a route-based GKE cluster (default), Private IP Cloud Composer environment, Cloud Composer environment with a VPC Native GKE cluster using alias IP addresses, Shared VPC.

Difference between Composer 1 and Composer 2

Cloud Composer has two major versions:

- **Cloud Composer 1:**
  - This version has manual scaling and its environments are zonal.
- **Cloud Composer 2:**
  - This version has autoscaling environments, as well as a zonal
  - Apache Airflow metadata database and regional Airflow scheduling and execution layer.
  - Airflow schedulers, workers and web servers run in the Airflow execution layer.

Google Cloud Composer version comparison

	Cloud Composer 1	Cloud Composer 2
IMAGE VERSIONS	composer-1.x.x	composer-2.x.x
AIRFLOW VERSIONS	Airflow 1.10.* and Airflow 2	Airflow 2
PYTHON VERSIONS	3.8.6, 2.7.17	3.8.6
ENVIRONMENT'S CLUSTER	Standard mode VPC-native or routes-based Google Kubernetes Engine cluster	Autopilot mode VPC-native Google Kubernetes Engine cluster
SCALING	Manual	Automatic
PRICING MODEL	Cloud Composer 1 pricing model	Cloud Composer 2 pricing model
VERSION SUPPORT	New Cloud Composer 1 versions will be released until the end of March 2023. After that, follow the version deprecation policy.	In each release, it supports two minor versions of Airflow 2. For each minor version of Airflow 2, it supports one patch version.

The following table lists major differences between Cloud Composer 1 and Cloud Composer 2.

	Cloud Composer 1	Cloud Composer 2
Cloud Composer image versions	composer-1.x.x	composer-2.x.x
Airflow versions	Airflow 1.10.* and Airflow 2	Airflow 2
Python versions	3.8.6, 2.7.17	3.8.6
Horizontal scaling	Can adjust the number of nodes in the environment's cluster. This changes the number of Airflow workers. Can adjust the number of Airflow schedulers.	Automatically scaling number of Airflow workers, based on demand. Can set and change upper and lower limits for the number of workers. Can adjust the number of Airflow schedulers.
Vertical scaling	Can set machine types for cluster nodes, Airflow web server and database when creating an environment. Can change machine types for Airflow web server and database.	Can set and change workloads configuration: CPU, memory, and storage parameters for Airflow workers, schedulers, web server, and database.
Terraform support	Can create and update Cloud Composer 1 environments.	Can create and update Cloud Composer 2 environments.
Maintenance operations	All tasks can be impacted.	Tasks that take less than 55 minutes to execute are not impacted.
Asynchronous DAG loading	Supported in Airflow 1	Not supported

	Cloud Composer 1	Cloud Composer 2
DAG serialization	Always enabled in Airflow 2. Can be disabled in Airflow 1.	Always enabled in Airflow 2.
Support for Network Tags	Yes	Yes
Cluster nodes with GPUs	Yes	No

Cloud Composer pricing

- Pricing for Cloud Composer is consumption based, so you pay for what you use, as measured by vCPU/hour, GB/month, and GB transferred/month. We have multiple pricing units.
- The service charges in 10-minute intervals.

Pros and Cons of Cloud Composer

Pros

- **Tightly integrated with GCP:**
  - The biggest feature of Google Cloud Composer that sets it apart from other managed Airflow instances is its tight integration with the Google Cloud Platform.
  - For heavy users of Google's cloud products, Cloud Composer is a very attractive approach for those who require an Airflow implementation.
- **Built on Open-Source Airflow:**
  - Cloud Composer is built on Apache Airflow, a popular open source framework for workflow orchestration and management.
  - This provides community support, an extensible framework, and no lock-in to proprietary vendor tooling.
  - Updates are semi-frequent and most questions can be answered from a simple online search.
- **Python:**
  - Airflow is managed in pure Python.

    - DAGs are defined with code, bringing version control, object-oriented programming and reproducibility to workflow management.
- **Fully Managed:**
    - Cloud Composer is fully managed.
    - Users can focus on building the best orchestration and workflow possible, without worrying about resource provisioning or upkeep.

Cons

- **Ambiguous Pricing:**
    - Using Cloud Composer as a part of GCP lacks a clear pricing structure.
    - Google's pricing page is incredibly complex.
    - Furthermore, optimizing Cloud Composer costs may be an in-depth operation requiring a substantial effort. There are other data pipeline products that offer the benefit of essentially fixed pricing.
- **Requires an effective knowledge of Python:**
    - While most data/analytics engineers will have a working knowledge of Python, it's important to note that Airflow is built entirely in the Object-Oriented language.
- **No paid support:**
    - Google support is already infamous for certain aspects of GCP, but one can expect little technical support with the nuances of Airflow.
- **Requires specialized infrastructure knowledge:**
    - Airflow is a technically complex product.
    - Topics like authentication, non-standard connectors, and parallelization require the specialized knowledge typically held by a data engineer.
    - So Airflow and Cloud Composer are not a good fit for teams seeking a low-/no-code solution.
- **Not user friendly:**
    - For example, each transfer operator has a different interface, and mapping from source to destination is different for each operator.
- **Lack of integrations:**

- Data transfer operators cover a limited number of databases/ lakes/warehouses and almost no business applications, so its uses are limited.

Apache Airflow Use Cases

Apache Airflow can be used with any data pipelines and is a great tool for orchestrating jobs that have complex dependencies. It's being used in many organizations worldwide including Adobe, Paypal, Twitter, Airbnb, Square, etc.

Some of the popular use cases for Apache Airflow include the following:

- **Pipeline scheduler:** This is Airflow's support for notifications on failures, execution timeouts, triggering jobs, and retries. As a pipeline scheduler, Airflow can check the files and directories periodically and then execute bash jobs.
- **Orchestrating jobs:** Airflow can help to orchestrate jobs even when they have complex dependencies.
- **Batch processing data pipelines:** Apache Airflow helps you create and orchestrate your batch data pipelines by managing both computational workflows and data processing pipelines.
- **Track disease outbreaks:** Apache Airflow can also help you to track disease outbreaks.
- **Train models:** Apache Airflow can help you to manage all tasks in one place, including complex ML training pipelines.