Use Cases, Architecture, and Best Practices

run.ai/guides/machine-learning-operations/apache-airflow

What Is Apache Airflow?

Apache Airflow is an open-source platform for authoring, scheduling and monitoring data and computing workflows. First developed by Airbnb, it is now under the Apache Software Foundation. Airflow uses Python to create workflows that can be easily scheduled and monitored. Airflow can run anything—it is completely agnostic to what you are running.

Benefits of Apache Airflow include:

- **Ease of use**—you only need a little python knowledge to get started.
- **Open-source community**—Airflow is free and has a large community of active users.
- **Integrations**—ready-to-use operators allow you to integrate Airflow with cloud platforms (Google, AWS, Azure, etc).
- **Coding with standard Python—**you can create flexible workflows using Python with no knowledge of additional technologies or frameworks.
- **Graphical UI**—monitor and manage workflows, check the status of ongoing and completed tasks.

This is part of our series of articles about machine learning operations.

In this article, you will learn:

Airflow Use Cases

Apache Airflow's versatility allows you to set up any type of workflow. Airflow can run ad hoc workloads not related to any interval or schedule. However, it is most suitable for pipelines that change slowly, are related to a specific time interval, or are pre-scheduled.

In this context, slow change means that once the pipeline is deployed, it is expected to change from time to time (once every several days or weeks, not hours or minutes). This has to do with the lack of versioning for Airflow pipelines.

Airflow is best at handling workflows that run at a specified time or every specified time interval. You can trigger the pipeline manually or using an external trigger (e.g. via REST API).

You can use Apache Airflow to schedule the following:

- ETL pipelines that extract data from multiple sources, and run Spark jobs or other data transformations
- Machine learning model training
- Automated generation of reports
- Backups and other DevOps tasks

Airflow is commonly used to automate machine learning tasks. To understand machine learning automation in more depth, **read our guides** to:

- Machine learning workflow
- Machine learning automation

Workloads

The DAG runs through a series of Tasks, which may be subclasses of Airflow's BaseOperator, including:

- **Operators**—predefined tasks that can be strung together quickly
- **Sensors**—a type of Operator that waits for external events to occur
- **TaskFlow—**a custom Python function packaged as a task, which is decorated with @tasks

Operators are the building blocks of Apache Airflow, as they define how the Tasks run and what they do. The terms Task and Operator are sometimes used interchangeably, but they should be considered separate concepts, with Operators and Sensors serving as templates for creating Tasks.

Airflow Architecture

The Airflow platform lets you build and run workflows, which are represented as Directed Acyclic Graphs (DAGs). A sample DAG is shown in the diagram below.

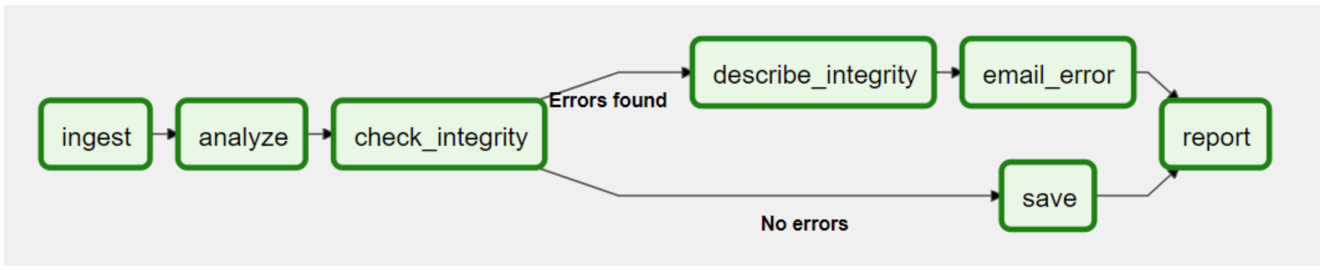


Image Source: Airflow

A DAG contains Tasks (action items) and specifies the dependencies between them and the order in which they are executed. A Scheduler handles scheduled workflows and submits Tasks to the Executor, which runs them. The Executor pushes tasks to workers.

Other typical components of an Airflow architecture include a database to store state metadata, a web server used to inspect and debug Tasks and DAGs, and a folder containing the DAG files.

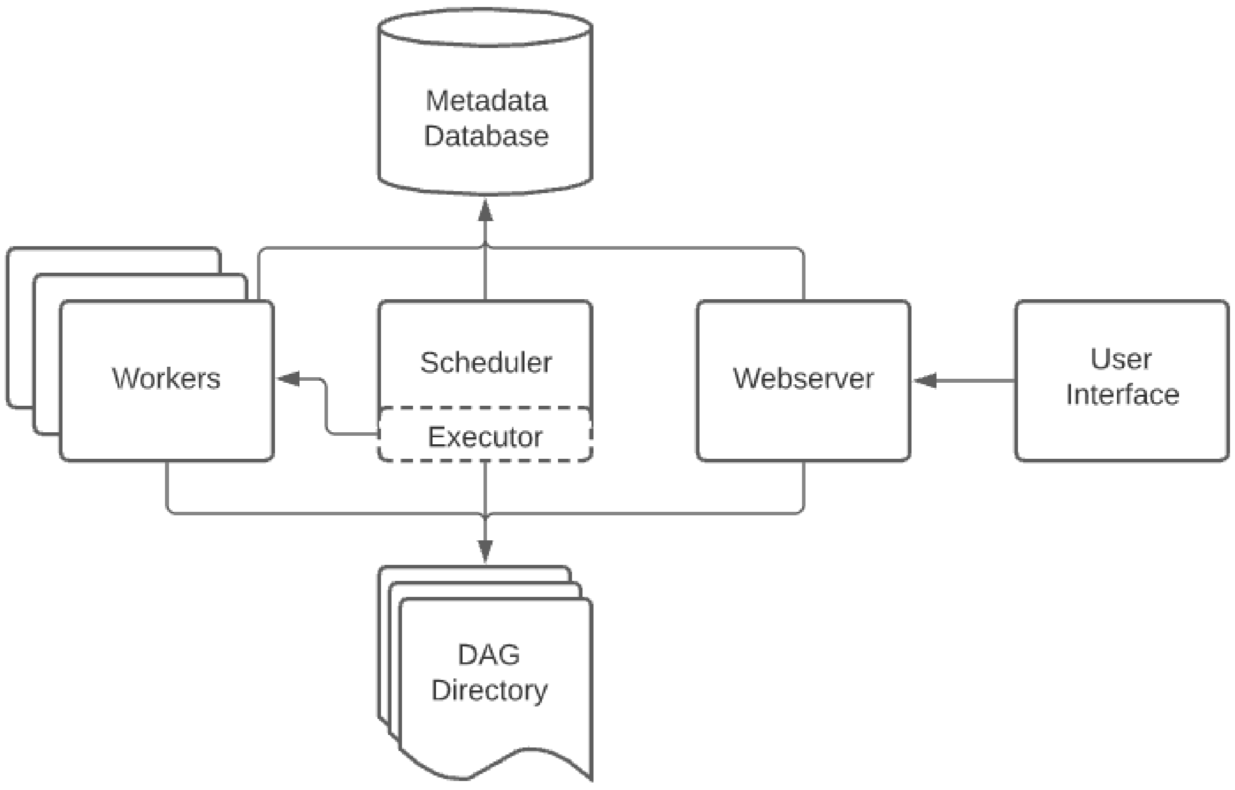


Image Source: Airflow

Control Flow

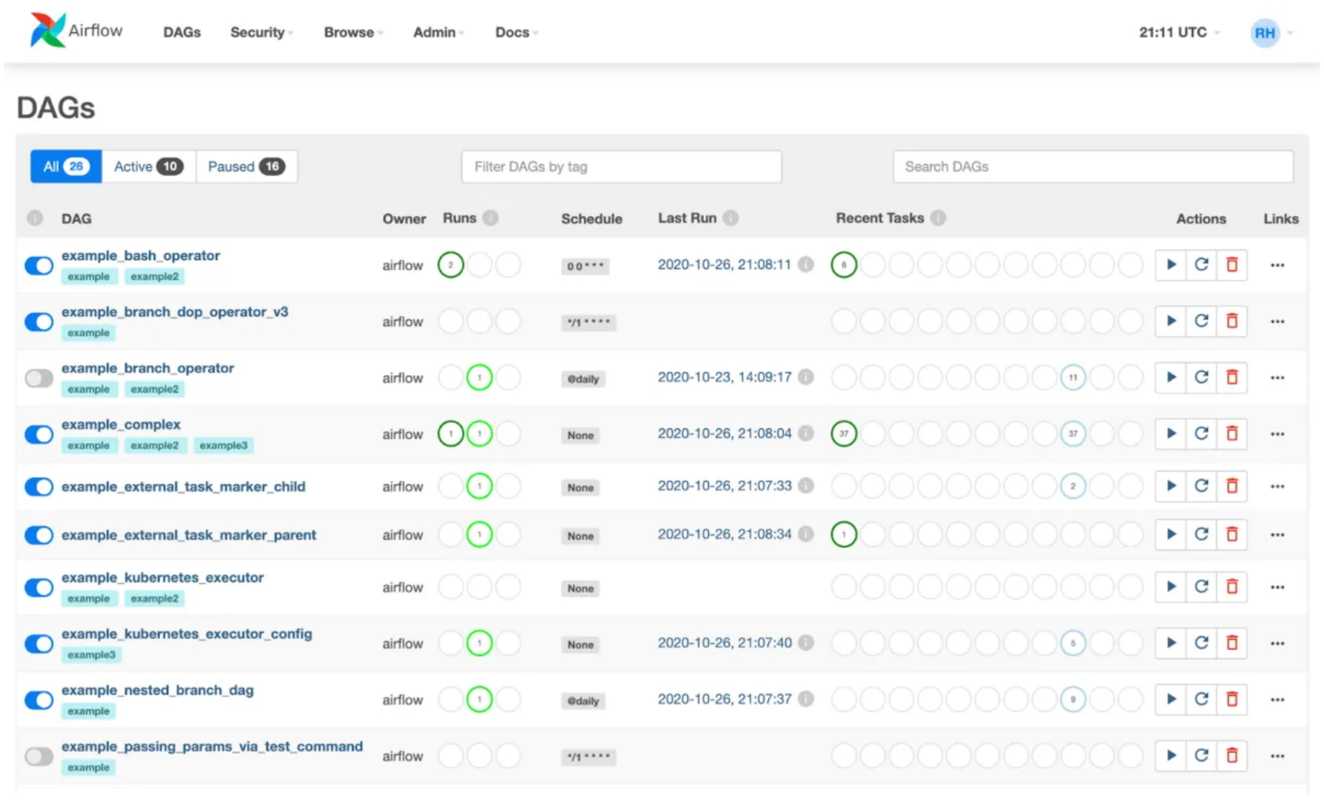
DAGs can be run multiple times, and multiple DAG runs can happen in parallel. DAGs can have multiple parameters indicating how they should operate, but all DAGs have the mandatory parameter execution_date.

You can indicate dependencies for Tasks in the DAG using the characters >> and << :

first_task >> [second_task, third_task]
third_task << fourth_task

By default, tasks have to wait for all upstream tasks to succeed before they can run, but you can customize how tasks are executed with features such as LatestOnly, Branching, and Trigger Rules.

To manage complex DAGs, you can use SubDAGs to embed "reusable" DAGs into others. You can also visually group tasks in the UI using TaskGroups.



DAG	Owner	Runs	Schedule	Last Run	Recent Tasks	Actions	Links
example_bash_operator (example, example2)	airflow	2	0 0 * * *	2020-10-26, 21:08:11	6	▶ ↻ 🗑	...
example_branch_dop_operator_v3 (example)	airflow		*/1 * * * *			▶ ↻ 🗑	...
example_branch_operator (example, example2)	airflow	1	@daily	2020-10-23, 14:09:17	11	▶ ↻ 🗑	...
example_complex (example, example2, example3)	airflow	1, 1	None	2020-10-26, 21:08:04	37, 37	▶ ↻ 🗑	...
example_external_task_marker_child	airflow	1	None	2020-10-26, 21:07:33	2	▶ ↻ 🗑	...
example_external_task_marker_parent	airflow	1	None	2020-10-26, 21:08:34	1	▶ ↻ 🗑	...
example_kubernetes_executor (example, example2)	airflow		None			▶ ↻ 🗑	...
example_kubernetes_executor_config (example3)	airflow	1	None	2020-10-26, 21:07:40	5	▶ ↻ 🗑	...
example_nested_branch_dag (example)	airflow	1	@daily	2020-10-26, 21:07:37	9	▶ ↻ 🗑	...
example_passing_params_via_test_command (example)	airflow		*/1 * * * *			▶ ↻ 🗑	...

Image Source: Airflow

Airflow Components

In addition to DAGs, Operators and Tasks, the Airflow offers the following components:

- **User interface**—lets you view DAGs, Tasks and logs, trigger runs and debug DAGs. This is the easiest way to keep track of your overall Airflow installation and dive into specific DAGs to check the status of tasks.
- **Hooks**—Airflow uses Hooks to interface with third-party systems, enabling connection to external APIs and databases (e.g. Hive, S3, GCS, MySQL, Postgres). Hooks should not contain sensitive information such as authentication credentials.

- **Providers**—packages containing the core Operators and Hooks for a particular service. They are maintained by the community and can be directly installed on an Airflow environment.
- **Plugins**—a variety of Hooks and Operators to help perform certain tasks, such as sending data from SalesForce to Amazon Redshift.
- **Connections**—these contain information that enable a connection to an external system. This includes authentication credentials and API tokens. You can manage connections directly from the UI, and the sensitive data will be encrypted and stored in PostgreSQL or MySQL.

Airflow Best Practices

Here are a few best practices that will help you make more effective use of Airflow.

Keep Your Workflow Files Up to Date

Since Airflow workflows are generated in Python code, you also need to find a way to keep your workflows up to date. The easiest way is to sync them with a Git repository. Airflow loads files from the DAGs folder in the Airflow directory, where you can create subfolders linked to the Git repository.

The synchronization itself can be done via a BashOperator and pull requests. Other files used in the workflow (such as training scripts for machine learning) can also be synced via the Git repository. A pull request can be made at the start of a workflow.

Define the Clear Purpose of your DAG

Before creating a DAG, you should carefully consider what you expect from it. Ask questions like what is the input of the DAG, what is its expected output, when it should trigger and at what time intervals, and which third-party tools it should interact with.

Don't overcomplicate your data pipeline. DAGs need to have a clear purpose, such as exporting data to a data warehouse or updating machine learning models. Keep DAGs as simple as possible to reduce issues and make maintenance easier.

Use Variables for More Flexibility

With Airflow, there are several ways to make DAG objects more flexible. At runtime, a context variable is passed to each workflow execution, which is quickly incorporated into an SQL statement. This includes the run ID, execution date, and last and next run times.

For example, you can adjust the data period according to a set execution interval. Airflow also offers the possibility of storing variables in a metadata database, which can be customized via web interface, API and CLI. It can be used, for example, to maintain flexibility in file paths.

Set Priorities

Temporary bottlenecks appear when multiple workflows compete for execution at the same time. The prioritization of workflows can be controlled using the priority_weight parameter. Parameters can be set for each task or passed as the default for the entire DAG. With Airflow 2.0, you can also use multiple scheduler instances to reduce workflow startup delays.

Define Service Level Agreements (SLAs)

Airflow allows you to define how quickly a task or entire workflow must be completed. If the time limit is exceeded, the person in charge is notified and the event is logged in the database. This allows you to search for anomalies that caused the delay—for example, you may be processing an unusually large amount of data.

Machine Learning Workflow Automation with Run:AI

If you are using Airflow to automate machine learning workflows, Run:AI can help automate resource management and orchestration. With Run:AI, you can automatically run as many compute intensive experiments as needed.

Here are some of the capabilities you gain when using Run:AI:

- **Advanced visibility**—create an efficient pipeline of resource sharing by pooling GPU compute resources.
- **No more bottlenecks**—you can set up guaranteed quotas of GPU resources, to avoid bottlenecks and optimize billing.
- **A higher level of control**—Run:AI enables you to dynamically change resource allocation, ensuring each job gets the resources it needs at any given time.

Run:AI simplifies machine learning infrastructure pipelines, helping data scientists accelerate their productivity and the quality of their models.

Learn more about the Run.ai GPU virtualization platform.