Apache Airflow

Workflow is a system for managing repetitive processes and tasks which occur in a particular order.

Workflow management refers to the identification, organization, and coordination of a particular set of tasks that produce a specific outcome.

Workflow management is all about optimizing, improving, and automating workflows wherever possible to increase output, eliminate repetition, and reduce errors.

Apache Airflow is an open-source tool to programmatically author, schedule, and monitor workflows. It is one of the most robust platforms used by Data Engineers for orchestrating workflows or pipelines. You can easily visualize your data pipelines' dependencies, progress, logs, code, trigger tasks, and success status.

With Airflow, users can author workflows as Directed Acyclic Graphs (DAGs) of tasks. Airflow's rich user interface makes it easy to visualize pipelines running in production, monitor progress, and troubleshoot issues when needed. It connects with multiple data sources and can send an alert via email or Slack when a task completes or fails.

Airflow is distributed, scalable, and flexible, making it well suited to handle the orchestration of complex business logic.

Efficient, cost-effective, and well-orchestrated data pipelines help data scientists develop better-tuned and more accurate ML models, because those models have been trained with complete data sets and not just small samples.

Airflow is natively integrated to work with big data systems such as Hive, Presto, and Spark, making it an ideal framework to orchestrate jobs running on any of these engines. Organizations are increasingly adopting Airflow to orchestrate their ETL/ELT jobs.

There are multiple ways to setup Airflow.Below are few ways.

- ✓ Basic setup using a virtualenv and pip. In this setup, we run SequentialExecutor, which is ideal for testing DAGs on a local development machine.
- ✓ Setup using Docker, in which we run CeleryExecutorusing Redis as a queue.
- ✓ Kubernetes setup using Helm, for running KubernetesExecutor.

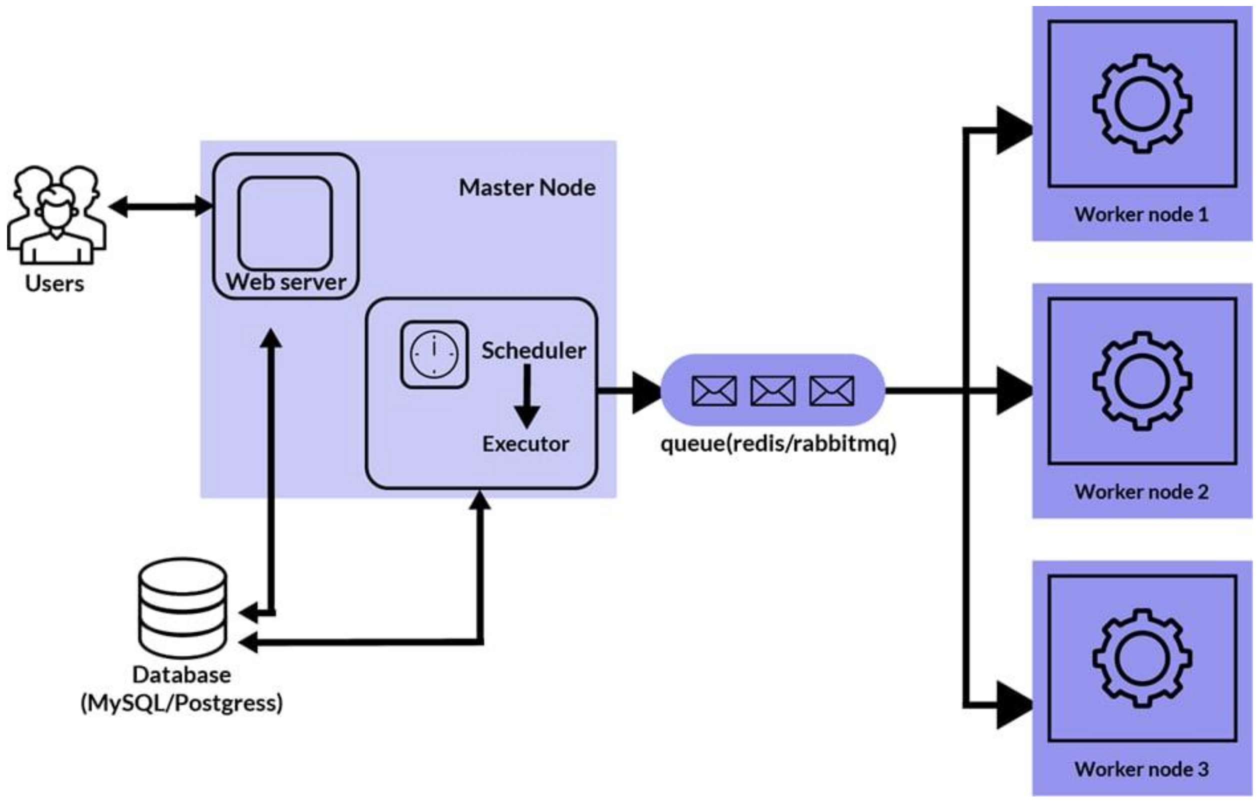
To understand how does Apache Airflow works, you must understand there are four major components that create this scalable and robust workflow scheduling platform:

Scheduler: The scheduler monitors all of the DAGs and their linked tasks. For a task, when dependencies are met, the scheduler initiates the task. It checks active tasks for initiation periodically.

- **Web Server**: This is the user interface of Airflow. It displays the status of responsibilities and lets the user interact with databases and read log files from remote file stores, such as Google Cloud Storage, S3, Microsoft Azure blobs, and more.
- **Database**: In the database, the state of the DAGs and their linked tasks are saved to make sure the schedule remembers the information of metadata. Airflow uses Object Relational Mapping (ORM) and SQLAlchemy to connect to the metadata database.
- The scheduler evaluates all of the DAGs and stores important information, such as task instances, statistics from every run, and schedule intervals.

- **Executor**: The executor zeroes down upon how the work will get done. There is a variety of executors that you can use for diverse use cases, such as SequentialExecutor, LocalExecutor, CeleryExecutor, and KubernetsExecutor.

Apache Airflow Architecture



Here's a brief introduction to Apache Airflow architecture operations:

- First, you build a DAG workflow containing the tasks, their order, and dependencies. Then, Airflow evaluates the DAG and creates a run based on the scheduling in the metadata database. At this point, the tasks are scheduled.
- Next, the scheduler questions the database, retrieves tasks, and sends them to the executor, thereby changing their status to queued. The executor then pushes the task to the workers, and the task status changes to running.
- Once a task is completed, it will be marked as either successful or failed. Finally, the scheduler will update its final status in the database backend.

What is DAG?

DAG abbreviates for **Directed Acyclic Graph.** It is the heart of the Airflow tool in Apache. It can be specifically defined as a series of tasks that you want to run as part of your workflow.

The Airflow tool might include some generic tasks like extracting out data with the SQL queries or doing some integrity calculation in Python and then fetching the result to be displayed in the form of tables. In Airflow, these generic tasks are written as individual tasks in DAG.

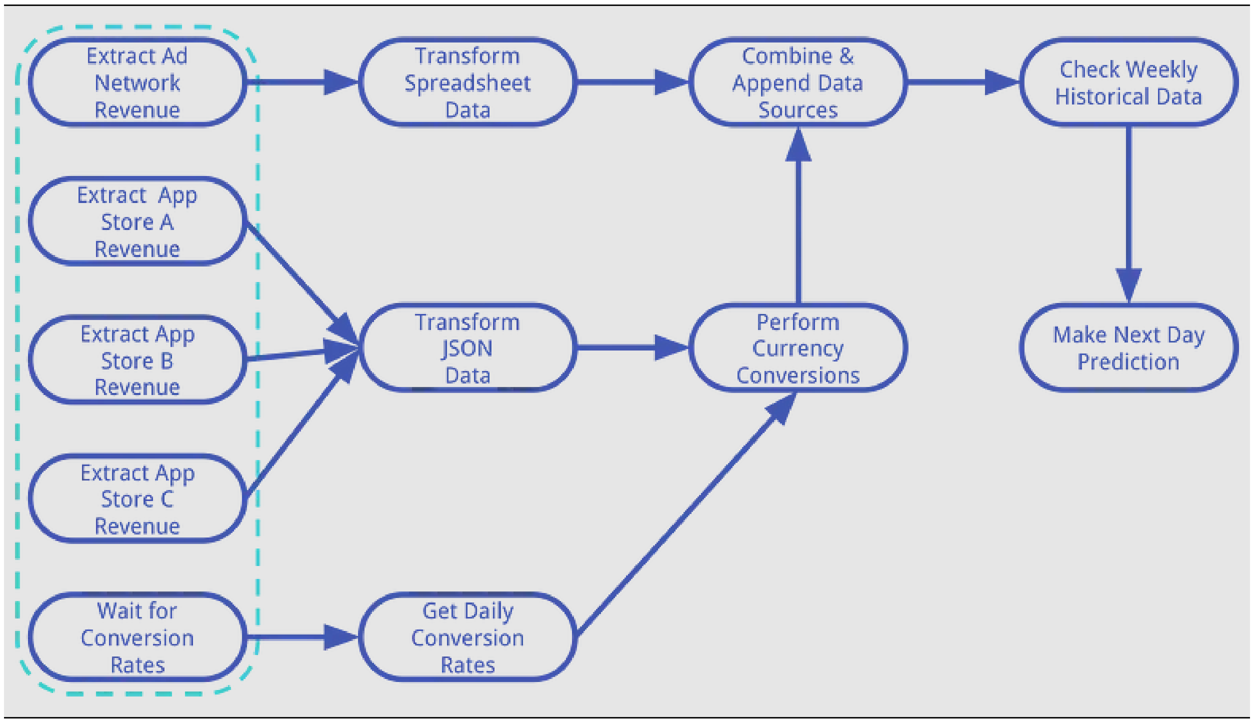
The main purpose of using Airflow is to define the relationship between the dependencies and the assigned tasks which might consist of loading data before actually executing. It might also consist of defining an order of running those scripts in a unified order.

Key Terminologies:

- **Operator:** The task in your DAG is called an operator. In airflow, the nodes of the DAG can be called an operator
- **Dependencies:** The specified relationships between your operators are known as dependencies. In airflow, the directed edges of the DAG can be called dependencies.
- **Tasks:** Tasks are units of work in Airflow. Each task can be an operator, a sensor, or a hook.
- **Task Instances:** It is a run of a task at a point in time. These are runnable entities. Task Instances belong to a DagRun.

DAG Properties

- DAGs are defined in python files inside the Airflow DAG folder.
- Transitive closure and transitive reduction are defined differently in Directed Acyclic Graphs.
- Each node receives a string of IDs to use as labels for storing the calculated value.
- *dag_id* serves as a unique ID for the DAG.
- *start_date* tells you when your DAG should start.
- *default_args* is a dictionary of variables to be used as constructor keyword parameters when initializing operators.
- max_active_tasks: The number of task instances allowed to run concurrently.
- max_active_runs: The number of active DAG runs allowed to run concurrently.
- default_view: The default view of the DAG in the Airflow UI (grid, graph, duration, gantt, or landing_times).


Extract Ad Network Revenue
Extract App Store A Revenue
Extract App Store B Revenue
Extract App Store C Revenue
Wait for Conversion Rates
Transform Spreadsheet Data
Transform JSON Data
Get Daily Conversion Rates
Combine & Append Data Sources
Perform Currency Conversions
Check Weekly Historical Data
Make Next Day Prediction

Scheduling Workflows:

The main function of Airflow is its ability to schedule workflow runs.

That is, there are 3 main ways a Airflow job can be triggered:

- ✓ manually (through the UI or CLI)
- ✓ on a scheduled interval (e.g. every 30 minutes)
- ✓ sensors (i.e. event-driven) - for example, a sensor on a file system or S3 bucket to trigger when new files are loaded.

Types of Executors in Airflow:

- ✓ **SequentialExecutor.** A sequential executor can run only one task per time, as workers and the scheduler run on the same machine.
- ✓ **LocalExecutor**: It is similar to SequentialExecutor, but it can run multiple tasks at the same time.
- ✓ **CeleryExecutor**. The CeleryExcecutor comes in handy when running distributed asynchronous Python tasks. However, it has a limited number of workers that can pick up scheduled tasks.

- ✓ **KubernetesExecutor**. This executor runs each task in a unique Kubernetes pod. As a result, it allows maximum resource distribution and usage as the KubernetesExecutor can create worker pods on demand.