Airflow

Before airflow: Hard/Hard — cronjobs

→ Cron Tab
→ Bad Scripts
→ Python Script → Airflow replaced

→ Monitoring → Scheduler, Retries run, Job duration

→ Managing Dependencies → centralized in Airflow, Retry mechanism

→ Dependencies → infrastructure state.

→ Backfills → Historic data reprocess

→ Scalability → as centralized, ability should be there

→ Distributed → exactly 1 different server which schedule right.

→ 100 break jobs that run daily at different times

To overcome the above mentioned issues Airflow comes into picture

→ A platform that programmatically defines, schedules and monitors workflows

→ Individual homogenous and a workflow → Task

→ Task

→ Dependency → upstream data.

→ Backfills → historic data reprocess

→ Scalability → as centralized, ability should be there

→ Distributed → exactly 1 different server which schedule right.

→ 100 break jobs that run daily at different times

To overcome the above mentioned issues Airflow comes into picture

→ A platform that programmatically defines, schedules and monitors workflows

→ Individual homogenous and a workflow → Task

→ Task

The whole pipeline is the above pictures to considered as a job

→ DAG → Directed Acyclic Graph

Core Components of Airflow

Utility of Airflow:

→ Run ETL pipelines
→ Data engineering pipelines
→ ML/DL pipelines
→ Predictive data analysis
→ General purpose scheduling

Role of Core Components:

① Scheduler → Triggers the scheduled workflows and submits tasks to the executor to run.

② Executor → Handles the tasks

Executors → They will be actually the tasks

① Sequential
② Local
③ Celery
④ Kubernetes

③ Web Server → Gives a UI to inspect, trigger schedules, view logs.

④ Metadata Database → Stores state regarding the DAG's scheduled to be, process run, and task instances.

⑤ DAG folder → Place the directory where the code for the workflows will be placed and the scheduler and executor will read the DAG from here.

Types of executors:

① Sequential Executor:

→ If in the default executor. Need to set the executor manually.

→ It takes one task at a time.

→ Use cases: Local dev, testing, small pipelines.

Pros:
① Simple
② Lightweight
③ No setup required

Cons:
① Not scalable
② Runs one task at a time
③ Low throughput
④ Single point of failure
⑤ Not suitable for production.

② Local Executor:

→ Runs tasks in parallel using threads or processes.

→ Use cases: Small production setups, multi-core machines.

Pros:
① Easy to setup
② Runs on multiple cores
③ Lightweight
④ Faster than the Sequential Executor as it can handle multiple tasks at once.

Cons:
① Not scalable — single machine only
② Resource intensive
③ Single point of failure
④ Shares the same Airflow's resources

③ Celery Executor:

→ Most widely used executor in the industry

→ Distributed architecture. We will be having multiple workers for the executor to the work.

Pros:
① Highly scalable — Horizontally scalable
② Production-ready
③ Fault tolerant
④ Supports task queuing, etc.

Cons:
① Complex setup
② Extra tools & setup
③ Needs high maintenance

④ Kubernetes Executor:

→ Runs the tasks on dedicated pod

→ K8's API are used to spin up the pod.

Pros:

① It can scale for you to the required size and then may get it to zero.
② Fault tolerant
③ Scalable
④ Production-ready

Cons:

① K8's expert is req
② Cold Start Problem
To launch the pods it takes a few seconds of time.