Apache Airflow Concepts & Interview Q&A; — Illustrated Guide

A Light■Theme guide covering key concepts, architecture, and 20+ interview questions with concise explanations.

■ Core Airflow Concepts

• DAG (Directed Acyclic Graph): Defines workflow and dependencies.

• Task & TaskInstance: A Task is a definition; TaskInstance is its run for a specific DAGRun.

• Operator: Defines the unit of work (PythonOperator, BashOperator, etc.).

• Scheduler: Determines when tasks/DAGs run and submits them to the executor.

• Executor & Worker: Execute tasks (LocalExecutor, CeleryExecutor, KubernetesExecutor).

• Metadata Database: Tracks DAGs, task states, variables, connections.

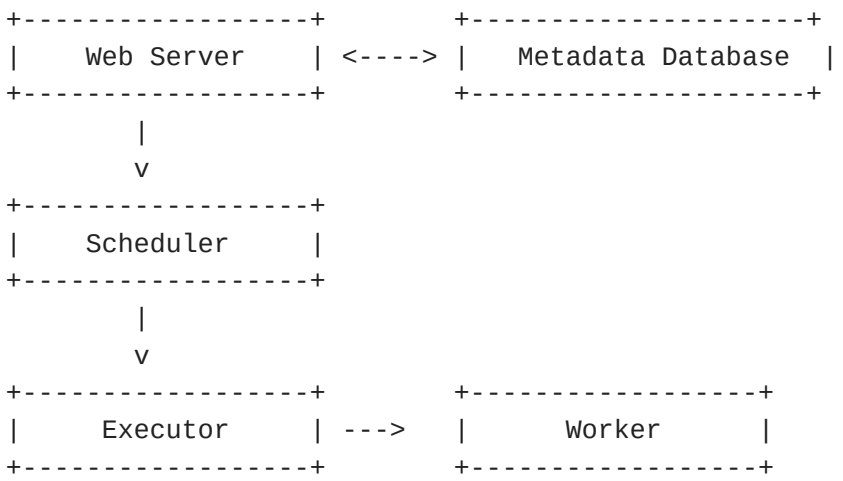
• Hooks & Connections: Interface to external systems with stored credentials.

• XCom: Mechanism to exchange data between tasks.

• Variables & Macros: For parameterization and dynamic DAG behavior.

• Idempotency: Ensuring reruns don’t produce duplicate results or side effects.

■■ Simplified Airflow Architecture



■ Interview Questions & Answers

Q: What is Airflow?

A: An open■source workflow orchestration platform to author, schedule, and monitor data pipelines.

Q: What is a DAG?

A: A Directed Acyclic Graph representing workflow tasks and their dependencies.

Q: Difference between Task and TaskInstance?

A: Task = definition, TaskInstance = specific execution of a Task for a DAGRun.

Q: How are tasks executed?

A: Scheduler creates DAGRuns, executor assigns tasks to workers, and results are stored in metadata DB.

Q: What are Operators, Sensors, and Hooks?

A: Operators define work; Sensors wait for conditions; Hooks connect to external systems.

Q: What are common Executors?

A: SequentialExecutor, LocalExecutor, CeleryExecutor, KubernetesExecutor.

Q: What is Idempotency?

A: Running the same task multiple times yields same result; essential for retries and backfills.

Q: What is Metadata Database used for?

A: Stores DAG state, task runs, variables, connections, and logs.

Q: What is XCom?

A: A cross■communication mechanism for tasks to exchange small data.

Q: What do schedule_interval, start_date, and catchup mean?

A: schedule_interval = frequency, start_date = first eligible time, catchup = run missed intervals.

Q: How to implement conditional tasks?

A: Use BranchPythonOperator, task groups, or ShortCircuitOperator.

Q: What is a TaskGroup?

A: A logical grouping of tasks for organization and visual clarity.

Q: How are retries handled?

A: Set retries, retry_delay, and callbacks for on_failure and on_success.

Q: What are DAG best practices?

A: Keep DAGs idempotent, atomic, parameterized, version-controlled, and modular.

Q: What is backfill?

A: Running missed DAG intervals between start_date and current date when catchup=True.

Q: What are DAGRun and TaskInstance states?

A: success, failed, skipped, up_for_retry, up_for_reschedule.

Q: How to handle external dependencies?

A: Use Sensors (FileSensor, ExternalTaskSensor) or TriggerDagRunOperator.

Q: What are Pools and Queues?

A: Pools limit concurrency; Queues route tasks to specific workers.

Q: depends_on_past vs wait_for_downstream?

A: depends_on_past waits for same task’s previous run; wait_for_downstream waits for all downstream tasks of previous run.

Q: What is a Sensor mode?

A: poke = check in same worker, reschedule = frees up slot and reschedules later.

■ Summary of Airflow Components

Component	Description
DAG	Defines workflow tasks and dependencies.
TaskInstance	Represents one task run for a DAGRun.
Scheduler	Determines task schedule and submits jobs to executor.
Executor	Runs tasks using processes, Celery, or Kubernetes.
Metadata DB	Tracks all DAG and task states.
Web Server	Provides UI to monitor DAGs and tasks.
XCom	Stores small data exchanged between tasks.
Variables	Store dynamic parameters used across DAGs.

■ Key Takeaways

• Airflow’s architecture: Scheduler, Executor, Workers, Webserver, Metadata DB.

• DAGs are Python-coded workflows — tasks and dependencies are defined as code.

• Tasks should be idempotent to safely handle retries and backfills.

• Use Sensors and XComs for dependency handling and data passing.

• Understand task states: success, failed, retry, reschedule.

• Apply modular design, parameterization, and version control for production-grade DAGs.