# Apache Airflow – Theoretical Assessment

## Section A – Basics

1. Apache Airflow is an open-source platform developed by Airbnb and later donated to the Apache Software Foundation. It is used to programmatically author, schedule, and monitor workflows. Airflow allows users to create Directed Acyclic Graphs (DAGs) to define workflows as code. It is primarily used in data engineering to automate data pipelines, ETL jobs, and batch processing tasks. Airflow provides scalability, flexibility, and reliability in managing complex workflows across multiple environments.

2. A DAG, or Directed Acyclic Graph, represents the structure of a workflow in Airflow. It consists of multiple tasks connected by dependencies. The term means:   
• Directed – The flow of execution moves in one specific direction.  
• Acyclic – There are no cycles, meaning tasks cannot depend on themselves directly or indirectly.  
• Graph – The workflow is represented as a collection of nodes (tasks) and edges (dependencies). DAGs help Airflow determine the order and execution logic of tasks.

3. A DAG defines the overall structure and schedule of a workflow, while a Task is a single unit of execution within that DAG. In other words, the DAG defines the 'what and when' (the orchestration), and tasks define the 'how' (the individual operations). For example, a DAG may represent an ETL workflow, and each task could represent steps like extracting data, transforming it, and loading it.

4. Workflows in Airflow must be Directed Acyclic Graphs to ensure logical and predictable execution. Cycles or loops would cause infinite execution or dependency errors. The acyclic nature guarantees that once a task is executed, Airflow will not revisit it, maintaining proper flow and consistency in workflow execution.

## Section B – Core Concepts

1. Airflow consists of several core components that work together to manage workflows:

• Webserver – The user interface of Airflow that displays DAGs, task statuses, logs, and execution history. It allows users to trigger DAGs manually and monitor their progress.

• Scheduler – Responsible for scheduling jobs according to their DAG definitions. It determines when tasks should be run based on their dependencies and intervals.

• Metadata Database – Stores metadata related to DAGs, tasks, users, and configurations. It keeps track of the state of each task instance and supports the scheduler and webserver operations.

2. The 'airflow db init' command initializes the metadata database. It creates necessary tables, default configurations, and schemas required by Airflow to store DAG runs, task states, users, and connections. It is the first step to set up Airflow after installation.

3. The 'start\_date' defines when a DAG should begin execution. It acts as the anchor for the scheduler to know the first valid date for triggering DAG runs. The 'schedule\_interval' determines how often the DAG should run (e.g., daily, hourly, or weekly). These two parameters together control the execution frequency and timing of workflows.

4. Setting 'catchup=False' prevents Airflow from executing past scheduled runs that were missed while the scheduler was inactive. For example, if a DAG with a daily schedule was paused for 7 days, setting 'catchup=False' ensures only the most recent run is executed upon resuming, reducing unnecessary computation.

## Section C – Operators & Execution

1. Operators in Airflow define the type of task that should be executed. They act as templates for defining task logic. Different operators handle different kinds of operations, such as executing a Python function, running a shell command, or interacting with databases. Examples include:  
• BashOperator – Executes bash commands or shell scripts.  
• PythonOperator – Runs Python functions.  
• EmailOperator – Sends emails as part of a workflow.  
Operators provide flexibility in automating diverse types of tasks within a workflow.

2. Airflow handles task failures by retrying them automatically based on parameters defined in the DAG or task (e.g., 'retries', 'retry\_delay'). If a task fails due to temporary issues, Airflow will attempt to rerun it after a specific delay. If it continues to fail after the maximum retries, it is marked as 'failed', and dependent tasks are skipped or retried based on the DAG configuration.

3. XCom (short for Cross-Communication) is a feature in Airflow that allows tasks to exchange small pieces of data at runtime. For instance, one task can push a value (e.g., a file path or query result) and another task can pull it later. This enables dynamic workflows where task outputs can influence downstream logic. However, XCom is intended for lightweight data exchange, not large datasets.

4. The BashOperator executes shell commands and scripts, making it useful for automation tasks that rely on bash scripting. The PythonOperator, on the other hand, allows execution of Python functions directly in Airflow, offering better flexibility and integration with Python-based data processing. While BashOperator is ideal for quick commands, PythonOperator is preferred for complex business logic or data transformations.

## Section D – Real-World Use

1. Airflow is widely used in ETL (Extract, Transform, Load) pipelines. For example, a company may use Airflow to extract data from an API, transform it using Python or Spark scripts, and load it into a data warehouse like Amazon Redshift or Google BigQuery. Airflow can automate this entire pipeline daily or hourly, ensuring reliable and repeatable data ingestion.

2. DAG scripts should be kept lightweight because the Airflow scheduler parses them continuously. Heavy computations within the DAG definition can slow down the scheduler and cause unnecessary performance overhead. Instead, complex logic should be handled within the task’s operator code or external processing functions.

3. Every DAG must have a unique 'dag\_id' because it acts as the unique identifier within Airflow’s metadata database. This ensures that workflows do not conflict with one another and helps the scheduler accurately track task runs, dependencies, and logs.

4. Airflow enforces task order and dependencies using DAG structure and operators like 'set\_upstream()' and 'set\_downstream()'. The scheduler uses this dependency graph to determine which tasks should run first and which should wait, ensuring workflows execute in the correct order every time.