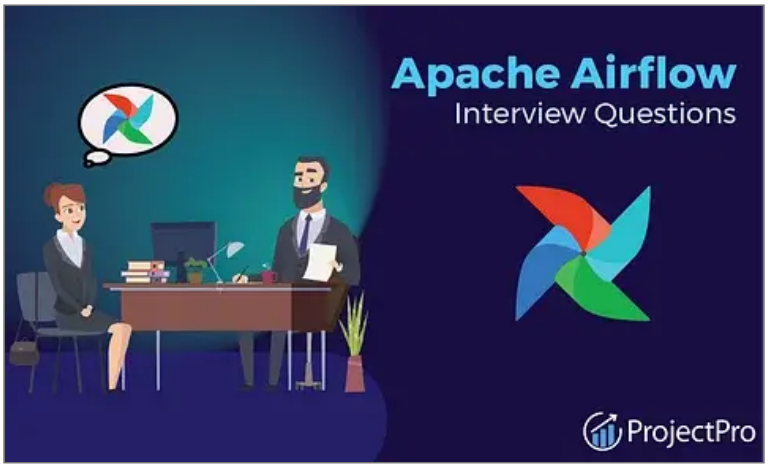
50 Apache Airflow Interview Questions and Answers

50 Good Apache Airflow Interview Questions and Answers to prepare for your next Data Engineering Job Interview | ProjectPro



Last Updated: 28 Oct 2024 | BY BADR SALAH

Data pipelines are crucial to the data infrastructure of any organization. Apache Airflow has recently garnered love from data engineers as a fantastic tool for managing their data extracts and transforms. This is one reason you are likely to be asked a few Apache Airflow Interview Questions if you appear for a data engineering job interview.

Suppose you are a Data Engineer in a retail company, your role involves creating data pipelines, which include collecting daily sales data from multiple databases, preprocessing it, uploading it, and making it ready for reporting. Now you end up manually running the same query every day, preprocessing data the same way every day, and doing it for many other similar tasks. You might do these mundane tasks for half of your time.

Isn't it great if your daily tasks automatically trigger on a defined time, and all the processes get executed in order? Then that's where Apache airflow will be helpful for you. Not just Data Engineers or Data Scientists, almost every tech job would require airflow task scheduling at some point.

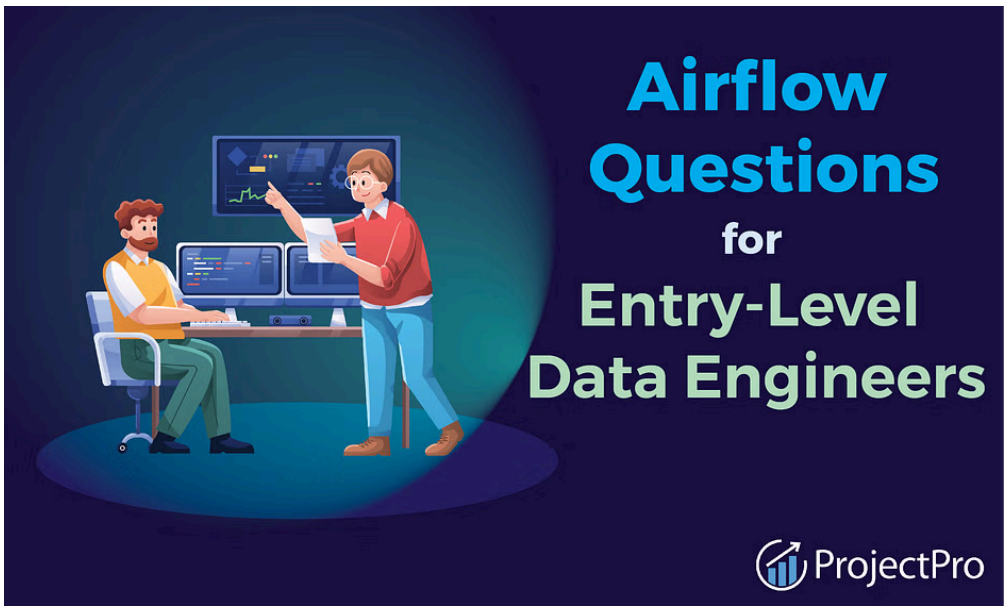
Table of Contents

50 Apache Airflow Interview Questions and Answers

So, let us discuss the top 50 Apache Airflow Interview Questions that will help you prepare for your upcoming data analytics or data engineering job interview.

Airflow Interview Questions and Answers for Freshers or Entry-Level Data

Engineers



If you're an entry-level data engineer or a fresher getting started in the data engineering domian, here are some beginner-level airflow interview questions that you must be prepared to answer.

1 . What is Apache airflow?

Airflow is an open-source workflow management tool by Apache Software Foundation (ASF), a community that has created a wide variety of software products, including

Apache Hadoop, Apache Lucene, Apache OpenOffice, Apache CloudStack, Apache Kafka, and many more.

Apache airflow helps to conduct, schedule, and monitor workflows. It helps to manage any ETL (Extract, Transform, Load) operation and data pipelines. Once we integrate airflow into our workflow (let's say an ETL task that you want to run daily at 1 pm), we can also visualize our data pipelines' dependencies, progress, logs, code, trigger tasks, and success status. And this can work simultaneously to perform multiple tasks and dependencies.

2 . Is Apache Airflow an ETL tool?

Airflow isn't an ETL tool, but it can manage, structure, and organize data transformation pipelines, Extract, Transform, and Load(ETL) pipelines, and workflows, making it a workflow orchestration tool.

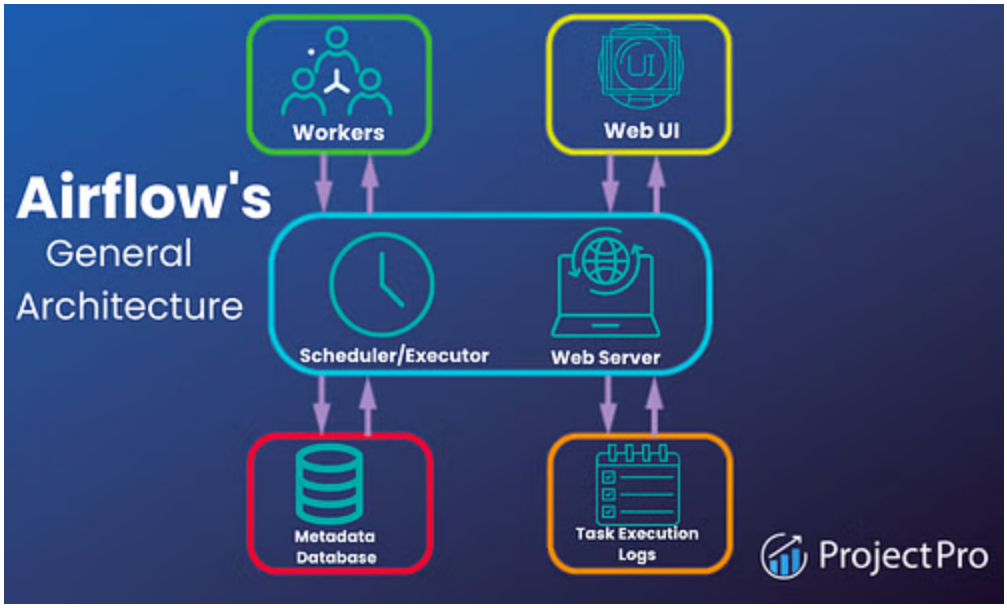
3 . How do we define workflows in Apache Airflow?

Workflows in Apache Airflow are a collection of tasks having dependencies on each other. Airflow uses directed acyclic graphs (DAGs) to represent a workflow. A task is represented as a node of a graph, and the dependencies between them are

represented as the edges of the graph. Those are "acyclic graphs" to avoid any circular dependencies causing infinite loops or deadlocks.

It defines four Tasks - A, B, C, and D - and shows the order in which they have to run and which tasks depend on which ones. It will also state how often to run the DAG (directed acyclic graph) - maybe "every day starting tomorrow at 1 pm" or "every Monday since January 1st, 2020, at 2 pm".

4 . What are the components of the Apache Airflow architecture?



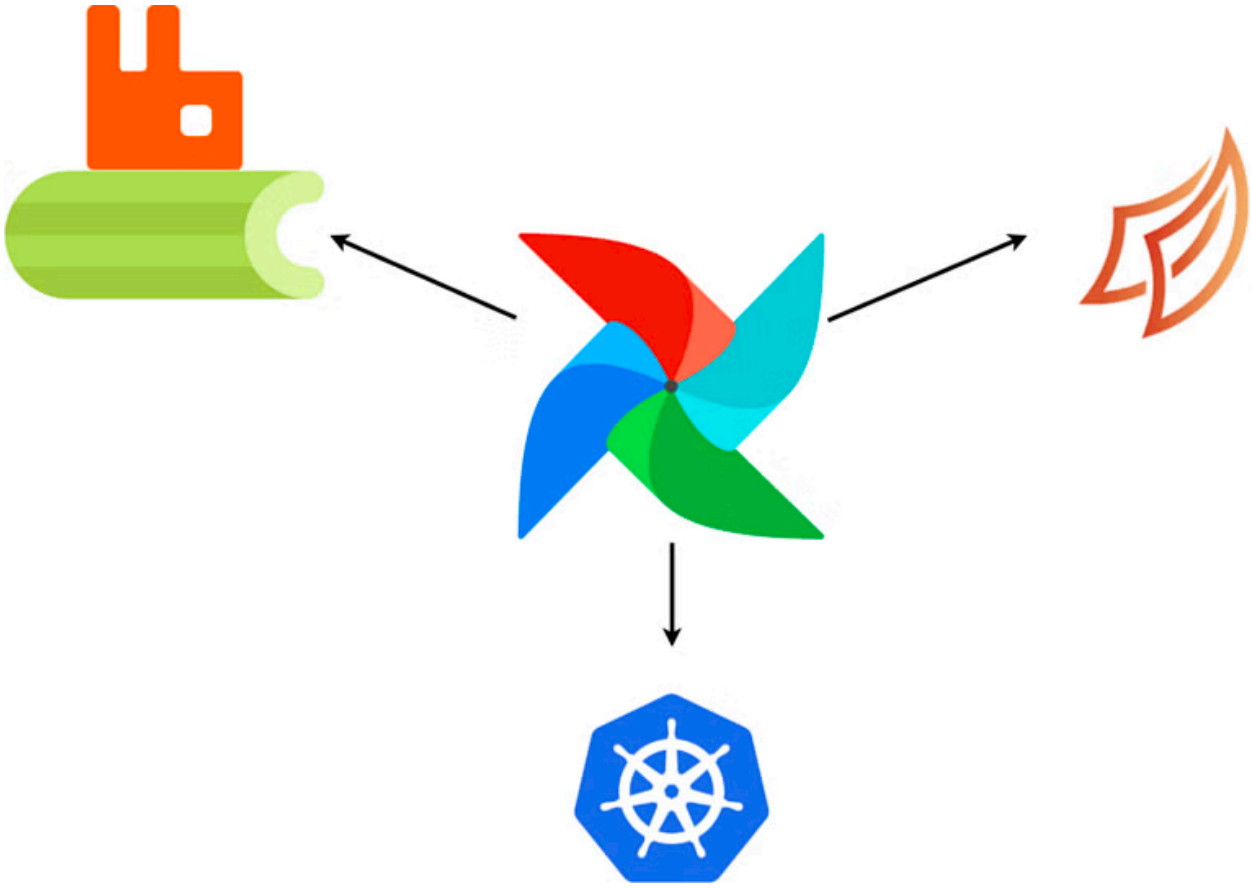
Apache Airflow architecture consists of the -

- **Airflow Scheduler:** It takes care of both triggering workflows at the scheduled time and submitting tasks to the Executor. The Scheduler orchestrates all of them.
- **Executor:** It is an internal component of the Scheduler**.** It takes care of the running tasks. This runs everything inside the Scheduler; it works closely with the Scheduler to figure out what resources will complete those tasks as they're queued (It is an Intermediate between the Scheduler and the worker).
- **Airflow Web Server:** The Airflow UI inspects the behavior of DAGs and task dependencies.
- **Metadata Storage Databases**: It keeps records of all the tasks within a directed acyclic graph and their statuses (queued, scheduled, running, success, failed, etc.) behind the scenes.

- **The Airflow Workers:** These are the processes that execute the tasks and are determined by the Executor being used. The workers are responsible for actually 'doing the task (the work).'

Airflow Interview Questions on Executors

If you are unable to understand the complications of executors while preparing for your interview, do not worry, this section will cover some basic airflow interview questions on executors that you are likely to get asked in your interview.



5. What are Local Executors and their types in Airflow?

Local Executors run tasks locally inside the scheduler process and are of 3 types:

- The Local Executor completes tasks in parallel that run on a single machine (think: your PC, an EC2 instance, etc.). A single Local Worker picks up and runs jobs as scheduled and is fully responsible for all task execution. That means you don't need resources outside that same machine to run a DAG or multiple DAGs (even heavy workloads).
- Since you can do everything from a single machine, it's straightforward to set up. But because of the same point, it is less scalable than other executors, and if a failure occurs at any moment, all the tasks will collapse.
- The Sequential Executor is similar to the Local Executor, but it will run only one task instance at a time; it could be considered a Local Executor with limited parallelism of just one worker (only one task). It does identify a single point of failure, which makes it helpful for debugging as well. Also, this is the default executor when installing Airflow; we can switch to other airflow configurations anytime.
- The Debug Executor is a debug tool and can be used from an IDE. It is a single process executor that queues Task Instances and executes them.

6 . What is a Celery Executor?

Celery Executor is one of the ways you can scale up the number of workers. "Celery" is a python framework and way of running distributed asynchronous tasks. The Celery Executor works with a pool of a fixed number of independent workers across which it can distribute tasks via messages. Whenever a worker node goes down, the celery executor detects and assigns it to another worker.

7 . How is Kubernetes Executor different from Celery Executor?

Kubernetes Executor has its own Kubernetes Pod (A Kubernetes Pod represents a single instance of a running process in your cluster) for each task which dynamically gets delegated a task and resources. For each task that needs to run, the Executor talks to the Kubernetes API to dynamically launch Pods on demand and terminate them when that task is completed.

This means you can dynamically scale up and down on the number of pods in cases of high traffic, storage, etc., unlike Celery, where there are a fixed number of workers up and running or waiting for a task to get assigned.

8 . What are Variables (Variable Class) in Apache Airflow?

Variables are a general way to store and retrieve content or settings as a simple key-value pair within Airflow. Variables in Airflow can be listed, created, updated, and deleted from the UI. Technically, Variables are Airflow's runtime configuration concept.

Airflow Interview Questions and Answers on XComs

Airflow XComs should not sound unfamiliar to you if you are appearing for a data engineering job interview/ The below list of Apache Airflow interview questions will give you some good information on why and how Airflow XComs can be used in DAG's.

9. What is the purpose of Airflow XComs?

Whenever you write an Airflow DAG (directed acyclic graphs), it contains multiple tasks running on different machines depending on what executor you are using; so how will the tasks (the nodes in the DAG) communicate to each other (share data)? Airflow XComs kicks in for that purpose and is short for "cross-communication."

10. Why don't we use Variables instead of Airflow XComs, and how are they different?

An XCom is identified by a "key," "dag id," and the "task id" it had been called from. These work just like variables but are alive for a short time while the communication is being done within a DAG. In contrast, the variables are global and can be used throughout the execution for configurations or value sharing.
There might be multiple instances when multiple tasks have multiple task dependencies; defining a variable for each instance and deleting them at quick successions would not be suitable for any process's time and space complexity.

Apache Airflow Interview Questions and Answers on Tasks and Operators

Apache Airflow DAG's consist of multiple tasks which form the basic unit of execution and are a must know concept to start building DAG's.

11 . What are the states a Task can be in? Define an ideal task flow.

Just like the state of a DAG (directed acyclic graph) being running is called a "DAG run", the tasks within that dag can have several tasks instances. they can be:

- **none:** the task is defined, but the dependencies are not met.
- **scheduled:** the task dependencies are met, has got assigned a scheduled interval, and are ready for a run.
- **queued:** the task is assigned to an executor, waiting to be picked up by a worker.
- **running:** the task is running on a worker.
- **success:** the task has finished running, and got no errors.
- **shutdown:** the task got interrupted externally to shut down while it was running.

- **restarting:** the task got interrupted externally to restart while it was running.
- **failed:** the task encountered an error.
- **skipped:** the task got skipped during a dag run due to branching (another topic for airflow interview, will cover branching some reads later)
- **upstream_failed:** An upstream task failed (the task on which this task had dependencies).
- **up_for_retry:** the task had failed but is ongoing retry attempts.
- **up_for_reschedule:** the task is waiting for its dependencies to be met (It is called the "Sensor" mode).
- **deferred:** the task has been postponed.
- **removed:** the task has been taken out from the DAG while it was running.

Ideally, the expected order of tasks should be : **none -> scheduled -> queued -> running -> success**.

12 . What is the role of Airflow Operators?

Whenever you define a DAG (directed acyclic graphs), there will be several tasks in it. Now those tasks can be written in different environments altogether, one task can be written in python code and another can be a bash script file. Now since these tasks inherit tasks dependencies within each other, they have to be operated from a single environment (which in our case would be a python file where our DAG is defined). So to solve this, airflow has its operators as python classes, where each operator can act as a wrapper around each unit of work that defines the actions that will be completed and minimizes or effort to write a lot of code.
Now, to execute the python script (Task I), we can call the PythonOperator() Class, and to execute the bash script file (Task II), we can call the BashOperator() Class.
Now if you want the airflow to send an email to you whenever the dag run or the task has been completed with its status, we also have an EmailOperator() as another DAG python class for this. Similarly many more!

13 . How does airflow communicate with a third party (S3, Postgres, MySQL)?

Airflow uses Hooks (a high-level interface) to interact with third-party systems, which enables its connection to external APIs and databases like S3, GCS, MySQL, and

Postgres. That is why Hooks are not meant to contain sensitive information like authentication credentials. It acts as an intermediate between the task Operator, and the external system/API.

Apache Airflow Interview Questions on Dynamic Acyclic Graphs (DAG's)

14 . What are the basic steps to create a DAG?

We can create a DAG flow is four major steps:

- Import the DAG class.
- Define a DAG : dag_id, start_date and schedule interval
- Define Individual tasks.
- Defining tasks relations : task_A >> [task_B >> task_D , task_C >> task_D]

15 . What is Branching in Directed Acyclic Graphs (DAGs)?

Branching tells the DAG to run all dependent tasks, but you can choose which Task to move onto based on a condition. A task_id (or list of task_ids) is given to the "BranchPythonOperator", the task_ids are followed, and all other paths are skipped. It can also be "None" to ignore all downstream tasks.
Even if tasks "branch_a" and "join" both are directly downstream to the branching operator, "join" will be executed for sure if "branch_a" will get executed, even if "join" is ruled out of the branching condition.

Explore Categories

Apache Hadoop Projects | Apache Hive Projects | Apache Hbase Projects

Apache Pig Projects | Hadoop HDFS Projects | Apache Impala Projects

Apache Flume Projects | Apache Sqoop Projects | Spark SQL Projects

Spark GraphX Projects | Spark Streaming Projects | Spark MLlib Projects

Apache Spark Projects | PySpark Projects | Apache Zepellin Projects

Apache Kafka Projects | Neo4j Projects | Microsoft Azure Projects

Google Cloud Projects GCP | AWS Projects

Apache Airflow Interview Questions and Answers for Experienced Data Engineers

Experienced data engineers are often asked question based on their previous experience working on data engineering projects using Apache Airflow. However, here is a list of some commonly asked Apache Airflow interview questions that experienced data engineers are likely to come across when appearing in interviews.

16 . What are ways to Control Airflow Workflow?

By default, a DAG will only run an airflow task when all its Task dependencies are finished and successful. However, there are several ways to modify this:

- **Branching (BranchPythonOperator):** We can apply multiple branches or conditional limits to what path the flow should go after this task.
- **Latest Only (LatestOnlyOperator):** This task will only run if the date the DAG is running is on the current data. It will help in cases when you have a few tasks which you don't want to run while backfilling historical data.
- **Depends on Past (depends_on_past = true; arg):** Will only run if this task run succeeded in the previous DAG run.
- **Trigger rules ("trigger_rule"; arg):** By default, a DAG will only run an airflow task when all of its previous tasks have succeeded, but trigger rules can help us alter those conditions. Like "trigger_rule = always" to run it anyways, irrespective of if the previous tasks succeeded or not, OR "trigger_rule = all_success" to run it only when all of its previous jobs succeed.

17 . Explain the External task Sensor?

An External task Sensor is used to sense the completion status of a **DAG_A from DAG_B or vice-versa**. If two tasks are in the same Airflow DAG we can simply add the line of dependencies between the two tasks. But Since these two are completely different DAGs, we cannot do this.

We can Define an ExternalTaskSensor in DAG_B if we want DAG_B to wait for the completion of DAG_A for a specific execution date.

There are six parameters to an External Task Sensor:

- **external_dag_id:** The DAG Id of the DAG, which contains the task which needs to be sensed.
- **external_task_id:** The Task Id of the task to be monitored. If set to default(None), the external task sensor waits for the entire DAG to complete.
- **allowed_states:** The task state at which it needs to be sensed. The default is "success."

- **execution_delta:** Time difference with the previous execution, which is needed to be sensed; the default is the same execution_date as the current DAG.
- **execution_date_fn:** It's a callback function that returns the desired execution dates to the query.

18 . What are the ways to monitor Apache Airflow?

- **Airflow Logs:** Logging in Airflow is implemented through Python's "logging" library. Airflow logs from the WebServer, Scheduler, and Workers performing tasks into a local system file by default.
- **DAGs View:** The DAGs View displays the list of DAGs in your environment, as well as it displays shortcuts to other built-in Airflow Monitoring tools. There we can see the names of our DAGs and the statuses of recently conducted runs and tasks.
- **Tree View:** Tree View helps us to dig deeper into the DAG; it displays the workflow as well as it displays the status of each run and each task over time.

19 . What is TaskFlow API? and how is it helpful?

We have read about Airflow XComs (cross-communication) and how it helps to transfer data/messages between tasks and fulfill data dependencies. There are two basic commands of XComs which are "xcom_pull" used to pull a list of return values from one or multiple tasks and "xcom_push" used for pushing a value to the Airflow XComs. Now, Imagine you have ten tasks, and all of them have 5-6 data dependencies on other tasks; writing an xcom_pull and x_push for passing values between tasks can get tedious. So **TaskFlow API** is an abstraction of the whole process of maintaining task relations and helps in making it easier to author DAGs without extra code, So you get a natural flow to define tasks and dependencies.
Note: *TaskFlow API was introduced in the later version of Airflow, i.e., Airflow 2.0. So can be of minor concern in airflow interview questions.*

20 . How are Connections used in Apache Airflow?

Apache Airflow is often used to pull and push data into other APIs or systems via hooks that are responsible for the connection. But since hooks are the intermediate part of the communication between the external system and our dag task, we can not use them to contain any personal information like authorization credentials, etc. Now let us assume the external system here is referred to as a MySQL database. We do need credentials to access MySQL, right? So where does the "Hook" get the credentials from?

That's the role of "Connection" in Airflow.

Airflow has a Connection concept for storing credentials that are used to talk to external systems. A Connection is a set of parameters - such as login username, password, and hostname - along with the system type it connects to and a unique id called the "conn_id".

If the connections are stored in the metadata database, metadata database airflow supports the use of "Fernet" (an encryption technique) to encrypt the password and other sensitive data.

Connections can be created in multiple ways:

- Creating them directly from the airflow UI.
- Using Environment Variables.
- Using Airflow's REST API.
- Setting it up in the airflows configuration file itself "airflow.cfg".
- Using airflow CLI (Command Line Interface).

21 . Explain Dynamic DAGs.

Dynamic-directed acyclic graphs are nothing but a way to create multiple DAGs without defining each of them explicitly. This is one of the major qualities of apache airflow, which makes it a supreme "workflow orchestration tool".

Let us say you have ten different tables to modify every day in your MySQL database, so you create ten DAG's to upload the respective data to their respective databases. Now think if the table names change, would you go to each dag and change the table

names? Or make new dags for them? Certainly not, because sometimes there can be hundreds of tables.

1. What is Apache Airflow primarily used for?

- Data storage
- Workflow orchestration
- Data cleaning
- Data visualization

Previous 1 / 7 Next

We can just treat dag creation as a python function and pass the new table name as an argument to it, and since the argument to a function can be a variable, we can change the "dag_id", and its arguments in just a single function call. Simple! Now, if we want the new dag to connect to an external API, we can just define a new connection and call it using the "conn_id," or anyways, we can use store the new arguments as the Variables and import Variable Class to call them.

22 . What are some of the most useful Airflow CLI commands?

- **Airflow dags list**: It will list all the DAGs that you currently have running.
- **Airflow dags delete :** It will delete all the data in DB related to this DAG too.
- **Airflow dags show :** It will show the structure and dependencies of the DAG.
- **Airflow DB init:** It initializes the DB.
- **Airflow DB check**: It will check the status of your database, connected, not connected, etc.
- **Airflow DB upgrade:** It will upgrade the data and dependencies of your database.
- **Airflow tasks list** : It will list down all tasks related to the mentioned DAG.

23 . How to control the parallelism or concurrency of tasks in Apache Airflow configuration?

Concurrency is the number of tasks allowed to run simultaneously. This can be set directly in the airflow configurations for all dags in the Airflow, or it can be set per DAG level. Below are a few ways to handle it:

In config :

- **parallelism:** maximum number of tasks that can run concurrently per scheduler across all dags.
- **max_active_tasks_per_dag:** maximum number of tasks that can be scheduled at once.
- **max_active_runs_per_dag:** . the maximum number of running tasks at once.

DAG level (as an argument to an Individual DAG) :

- **concurrency:** maximum number of tasks that can run concurrently in this dag.
- **max_active_runs:** maximum number of active runs for this DAG. The scheduler will not create new DAG runs once the limit hits.

24 . What do you understand by Jinja Templating?

Jinja Template is a web template engine in Python, which is used as a concept in Airflow. It allows us to interpolate values at runtime in static files like HTML or SQL queries. Just like we do a python string formatting using "{}, .format()", we have "{{}}" as a

placeholder. Whenever Jinja sees a "{{}}," it understands that this blank needs to be filled from an external value.

25 . What are Macros in Airflow?

Macros are functions used as variables. In Airflow, you can access macros via the "macros" library. There are pre-defined macros in Airflow that can help in calculating the time difference between two dates or more! But we can also define macros by ourselves to be used by other macros as well, like we can use a macro to dynamically generate the file path for a file. Some of the examples of pre-defined and most-used macros are:

- **Airflow.macros.datetime_diff_for_humans(dt, *since=None*):** Returns difference between two datetimes, or one and now. *(Since = None refers to "now"*)
- **airflow.macros.ds_add(*ds*, number_of_*days*) :** Add or subtract n number of days from a YYYY-MM-DD(ds), will subtract if number_of_days is negative.

26 . What are the limitations of TaskFlow API?

- Missing package dependency management, the TaskFlow abstraction can only work if everybody in the organization agrees to use the same package versions and other airflow dependencies, which makes TaskFlow not so ready for heavy production loads.
- Another limit is that TaskFlow API is built upon XComs, and XComs don't provide proper data-sharing functionality. Instead, it provides an abstraction to only share small amounts of data between tasks.

27 . How is the Executor involved in the Airflow Life cycle?

The life cycle of a task from the scheduler to the Executor includes the following steps:

- Before the scheduler sends the command on which task the Executor has to run, depending on the types of executors, the resources of executors are kept on idle or unavailable.
- Once the scheduled time hits the clock, the Airflow scheduler sends the command to the Executor.

- After receiving signals from the scheduler, the Executor starts allocating its resources and puts the tasks into the queue. Whenever work is available, it will pick up the tasks from the queue to start executing them.
- Once the tasks get finished, and the scheduler receives the "Completed**"** state from the Executor, the resources allocated for running the task get cleaned up.

28 . List the types of Trigger rules.

- **all_success:** the task gets triggered when all upstream tasks have succeeded.
- **all_failed:** the task gets triggered if all of its parent tasks have failed.
- **all_done:** the task gets triggered once all upstream tasks are done with their execution irrespective of their state, success, or failure.
- **one_failed:** the task gets triggered if any one of the upstream tasks gets failed.
- **one_success:** the task gets triggered if any one of the upstream tasks gets succeeds.
- **none_failed:** the task gets triggered if all upstream tasks have finished successfully or been skipped.
- **none_skipped:** the task gets triggered if no upstream tasks are skipped, irrespective of if they succeeded or failed.

29 . What are SLAs?

SLA stands for Service Level Agreement; this is a time by which a task or a DAG should have succeeded. If an SLA is missed, an email alert is sent out as per the system configuration, and a note is made in the log. To view the SLA misses, we can access it in the web UI.
It can be set at a task level using the "timedelta" object as an argument to the Operator, as sla *= timedelta(seconds=30)*.

30 . What is Data Lineage?

Many times, we may encounter an error while processing data. To determine the root cause of this error, we may need to track the path of the data transformation and find where the error occurred. If we have a complex data system then it would be challenging to investigate its root. Lineage allows us to track the origins of data, what happened to it, and how did it move over time, such as in S3, HDFS, MySQL or Hive, etc.

It becomes very useful when we have multiple data tasks reading and writing into storage. We need to define the input and the output data sources for each task, and a graph is created in Apache Atlas, which depicts the relationships between various data sources.

Python Airflow Interview Questions and Answers

Apache Airflow makes it easier for anyone with basic python programming knowledge to deploy a workflow without limiting the scope of the data pipeline.

31. Write a Python code to demonstrate the working of xcom_push and xcom_pull.

Below is a python script defining two functions, one to perform xcom_push and one to perform xcom_pull to get the value and perform a sum operation to the pulled value, and implementing them in DAG tasks.

32. Demonstrate the use of macros in a DAG task in python.

Below is a code for a task printing the task execution date and print it after adding 2 days to it.

```
task_A = BashOperator(
 task_id="execution_date",
 bash_command="echo 'execution date : {{ ds }} ds_add: {{ macros.ds_add(ds, 2) }}'"
)
```

33. Write a Python code to download a file from S3 to local system using airflow.

34. Define a DAG and Schedule it to run on 10:30am everyday, starting from 1st November 2022.

35. Create a Branching in a DAG with a starting task, and a 2 directional branch task.

Apache Airflow DAG and Spark Operator Interview Questions and Answers

36 . Why is Spark Integration needed in Apache airflow?

- Fast Speed: Spark can satisfy the organization's need to process big data at high speeds. And the tool depends on Resilient Distributed Dataset (RDD), in which the data is stored transparently in the memory, and read/write operations are carried out from there. Due to this reading and writing, time is reduced.
- Supports Advanced Analytics: Apart from the map and reduce operations, Apache Spark also supports SQL queries, advanced analytics, and data stream. Its extensions, such as Spark Streaming, MLib, and Spark SQL, make it possible.
- Real-Time Data Streaming: Spark can handle real-time data streaming; it can recover lost work and deliver other high-level functionalities without needing to write extra code.
- Scalable: Spark allows us to process humongous data sizes and write scalable applications in multiple languages, including Python.

37 . What is a Spark Submit Operator?

SparkSubmitOperator executes the "spark-submit" command through Apache Airflow. SparkSubmitOperatoraccepts takes all the desired arguments and assembles them to a bash command "spark-submit" which takes care of setting up Spark and its dependencies and can support different cluster managers and deploy modes of Spark, which is then executed by the BashOperator.

- from Airflow. operators import SparkSubmitOperator

The few important parameters of this operator are :

- application (str): Path to a bundled jar, including your application and all the dependencies.
- conf (dict[str, Any] | None) : Spark configuration property in key=value format (wrap “key=value” in quotes for configs with spaces).
- spark_conn_id: The Spark's conn_id is configured in Airflow administration.
- master (str): The master value for the cluster.
- main_class (str): The entry point for our application.
- deploy_mode (str): Whether to deploy the Driver on the worker nodes (cluster) or locally as an external client (client), the default is set to "client."
- application_args (str): If any arguments are needed to be passed to the main method of your main class.

38 . What is a Spark JDBC Operator?

This operator allows SparkSubmitOperator specifically for performing data transfers from JDBC-based databases such as PostgreSQL, MySQL, Oracle 11, etc., with Apache Spark.

A few important parameters of this operator are:

- spark_app_name: Name of the spark job (default: "airflow-spark-jdbc")
- spark_conn_id: The Spark's conn_id is configured in Airflow administration.
- spark_py_files: Additional python files used.
- spark_files: Additional files to upload to the container.
- spark_jars: Additional jars to upload and add to the classpath.
- jdbc_table: JDBC table name.
- jdbc_driver: Name of the JDBC driver to use for the connection (e.g., org.PostgreSQL. Driver)
- jdbc_conn_id : conn_id used for the JDBC database
- metastore_table: Metastore table's name.

- save_mode: overwrite/append.
- save_format : only for jdbc_to_sparkm, eg. JSON.

39 . What is the SparkSQL operator?

It executes Spark SQL queries. This operator runs the SQL query on Spark Hive metastore service. The SQL query can either be templated or used as .sql or .hql files, given that the spark SQL script is in the PATH.
The operator takes "SQL" for templated SQL query or "template_ext" as a list of .sql or .hql to execute along with the spark job name and connection id.

40 . Difference between Client mode and Cluster mode while deploying to a Spark Job.

In Client deploy mode, the spark driver component runs on the machine node from where the spark job is submitted.
In the cluster mode, the Spark driver will get started in any of the worker machines. So, the client who is applying (us, the airflow environment) can apply and then can go away to continue with some other job.
Whenever an airflow task is involved with much more than just the spark operation or the job is long enough, cluster mode is preferred because the client node (our local machine) will be needed to stay online till the spark job finishes in the client mode. Also, Cluster mode involves multiple workers, whereas client mode will fail instantly with the client node fails.

Scenario-Based Apache Airflow Interview Questions and Answers

41 . How would you approach if you wanted to queue up multiple dags with order dependencies?

We can use External task sensors for such a case. Let us say we have 3 DAGs A, B, and C with the sequence A->B->C. We can assign an external task senor in DAG B, with dag_id = 'A' and the corresponding task_id. And similarly, we can assign an external task sensor in DAG C, with dag_id = 'B' and the corresponding task_id.

42 . What if your Apache Airflow DAG failed for the last ten days, and now you want to backfill those last ten days' data, but you don't need to run all the tasks of the dag to backfill the data?

We can use the Latest Only (LatestOnlyOperator) for such a case. While defining a task, we can set the latest_only to True for those tasks, which we do not need to use for backfilling the previous ten days' data.

43 . What will happen if you set 'catchup=False' in the dag and 'latest_only = True' for some of the dag tasks?

Since in the dag definition, we have set catchup to False, the dag will only run for the current date, irrespective of whether latest_only is set to True or False in any one or all the tasks of the dag. 'catchup = False' will just ensure you do not need to set latest_only to True for all the tasks.

44 . What if you need to use a set of functions to be used in a directed acyclic graph?

We can go with using user-defined macros; macros are functions used as a variable. Since Macros are preprocessed, which means that all the macros would be processed before our program compiles. However, functions are not preprocessed but are compiled; it makes more sense to use macros instead of functions. But the catch is, we don't have to rely on pre-defined macros for that; we can define our own macros and pass them as "user_defined_marcos = {'macro_name': }" in our dag arguments.

45 . How would you handle a task which has no dependencies on any other tasks?

We can set "trigger_rules = 'always'" in a task, which will make sure the task will run irrespective of if the previous tasks have succeeded or not.

46 . How can you use a set or a subset of parameters in some of the dags tasks without explicitly defining them in each task?

We can use the "params" argument. It is a dictionary of DAG-level parameters that are made accessible in jinja templates. These "params" can be used at the task level. We can pass "params" as a parameter to our dag as a dictionary of parameters such as

{"param1": "value1", "param2": "value2"}. And these can be used as "echo {{params.param1}}" in a bash operator.

47 . Is there any way to restrict the number of variables to be used in your directed acyclic graph, and why would we need to do that?

Airflow Variables are stored in the Metadata Database, so any call to a variable would mean a connection to the database. Since our DAG files are parsed every X seconds, using a large number of variables in our DAG might end up saturating the number of allowed connections to our database. To tackle that, we can just use a single Airflow variable as a JSON, as an Airflow variable can contain JSON values such as {"var1": "value1", "var2": "value2"}.

48 . What would you do if you wanted to create multiple dags with similar functionalities but with different arguments?

We can use the concept of Dynamic DAGs generation. We can define a create_dag method which can take a fixed number of arguments, but the arguments will be dynamic. The dynamic arguments can be passed to the create_dag method through Variables, Connections, Config Files, or just passing a hard-coded value to the method.

49 . If we want to exchange large amounts of data, what is the solution to the limitation of XComs?

Since Airflow is an orchestrator tool and not a data processing framework, if we want to process large gigabytes of data with Airflow, we use Spark (which is an open-source distributed system for large-scale data processing) along with the Airflow DAGs because of all the optimizations that It brings to the table.

50 . What Executor will you use to test multiple jobs at a low scale?

Local Executor is ideal for testing multiple jobs in parallel for performing tasks for a small-scale production environment. The Local Executor runs the tasks on the same node as the scheduler but on different processors. There are other executors as well who use this style while distributing the work. Like, Kubernetes Executor would also use Local Executor within each pod to run the task.

Get Your Hands-Dirty with Apache Airflow to Prepare For Your Next Data Engineer Job Interview

So, it must be pretty evident that Apache airflow is a vast topic. Out of all the 50 questions covered above, the theoretical points are very few because the application of Airflow is filled with its practices. Hardly you will find a person (even an experienced professional) who has got his/her hands dirty on all of these topics as all these concepts are used in different scenarios and how rare (and unlucky, maybe :P) it would be to face all of these questions by working on hands-on Apache Airflow projects. Anyways, it is always advised to create your own basic DAGs and play with them to get a much better understanding.

FAQs on Apache Airflow

1 . What are some of the Alternatives to Apache Airflow?

- Luigi - A python package used to build Hadoop Jobs.
- Kedro - Used for creating easy-to-maintain and reproducible modular data science codes.

- Pinball - Open Source workflow manager built by Pinterest.
- AWS Step Functions - It is a fully managed, serverless, and low-code visual workflow service used to prepare data for machine learning, build serverless applications, automate ETL processes and orchestrate microservices.

2 . Why is Apache Airflow better for Data Engineers?

- Lower cost, innovation, and community support come up from open-source.
- It can be used with the Big 3 cloud providers - AWS, Azure, and GCP.
- Airflow UI allows us to monitor and troubleshoot the pipelines with ease.
- We can approach it programmatically through python.
- Many data pipelines have to customize for retries; Airflow has that built-in.

3 . What is the purpose of Apache Airflow?

The purpose of Airflow is to orchestrate pipelines or workflows, which refers to sequencing, coordinating, scheduling, and managing complex data pipelines from multiple sources.

PREVIOUS NEXT

About the Author

BadrSalah

A computer science graduate with over four years of writing experience in various fields. His passion for technology and knack for clear communication enables him to simplify compl...

Meet The Author >

Trending Blog Categories

- Artificial Intelligence Blogs
- AWS Blogs
- Azure Blogs
- Data Science Blogs
- Machine Learning Blogs
- Data Engineering Blogs

- Big Data Blogs

Project Categories

Machine Learning Projects

Data Science Projects

Deep Learning Projects

Big Data Projects

Apache Hadoop Projects

Apache Spark Projects

Show more

Projects

Walmart Sales Forecasting Data Science Project

BigMart Sales Prediction ML Project

Music Recommender System Project

Credit Card Fraud Detection Using Machine Learning

Resume Parser Python Project for Data Science

Time Series Forecasting Projects

Show more

Blogs

Machine Learning Projects for Beginners with Source Code

Data Science Projects for Beginners with Source Code

Big Data Projects for Beginners with Source Code

IoT Projects for Beginners with Source Code

Data Analyst vs Data Scientist

Data Science Interview Questions and Answers

Show more

Certification Courses

Practical MLOps Course

Data Engineering Course

AWS Data Engineering Course

Azure Data Engineering Course

GCP Data Engineering Course

PySpark Course

Snowflake Course

Show more

Tutorials

PCA in Machine Learning Tutorial

PySpark Tutorial

Hive Commands Tutorial

MapReduce in Hadoop Tutorial

Apache Hive Tutorial -Tables

Linear Regression Tutorial

Show more

ProjectPro

About us

Contact us

Privacy policy

User policy

Write for ProjectPro