## Apache Airflow Overview

Welcome to Apache Airflow Overview.

After watching this video, you will be able to:

Recognize Apache Airflow as a platform to programmatically author, schedule, and monitor

workflows.

List the main features and principles of Apache Airflow.

And, list common use cases for Apache Airflow.

Apache Airflow is a great open source workflow orchestration tool that is supported by an

active community.

It is a platform that lets you build and run workflows, such as batch data pipelines.

With Apache Airflow, a workflow is represented as a DAG (a Directed Acyclic Graph), and contains

individual pieces of work called tasks, arranged with dependencies.

And, note that unlike Big Data tools such as Apache Kafka, Apache Storm, Apache Spark,

or Flink, Apache Airflow is not a data streaming solution. It is primarily a workflow manager.

Let’s take a look at a simplified overview of Apache Airflow’s basic components.

Airflow comes with a built-in Scheduler, which handles the triggering of all scheduled workflows.

The Scheduler is responsible for submitting individual tasks from each scheduled workflow

to the Executor.

The Executor handles the running of these tasks by assigning them to Workers, which

then run the tasks.

The Web Server , serves Airflow’s powerful interactive User Interface.

From this UI, you can inspect, trigger and debug any of your DAGs and their individual

tasks.

The DAG Directory contains all of your DAG files, ready to be accessed by the Scheduler,

the Executor, and each of its employed Workers.

Finally, Airflow hosts a Metadata Database, which is used by the Scheduler, Executor,

and the Web Server to store the state of each DAG and its tasks.

A DAG specifies the dependencies between tasks, and the order in which to execute them. The

tasks themselves describe what to do. In this example DAG, the tasks include data ingestion,

data analysis, saving the data, generating reports, and triggering other systems, such

as reporting any errors by email.

Let’s have a look at the life cycle of a task’s state.

In this diagram, you can see how Apache Airflow might assign states to a task during its life

cycle.

No status: This means that the task has not yet been queued for execution.

Scheduled: The scheduler has determined that the task’s dependencies are met and has

scheduled it to run.

Removed: For some reason, the task has vanished from the DAG since the run started.

Upstream failed: An upstream task has failed.

Queued: The task has been assigned to the Executor and is waiting for a worker to become

available.

Running: The task is being run by a worker.

Success: The task finished running without errors.

Failed: The task had an error during execution and failed to run, and

Up for retry: The task failed but has retry attempts left and will be rescheduled.

Ideally, a task should flow through the scheduler from ‘no status’, to ‘scheduled’,

to ‘queued’, to ‘running’, and finally to ‘success.’

Now, let’s have a look at the five main features and benefits of Apache Airflow.

Pure Python

Create your workflows using standard Python. This allows you to maintain full flexibility

when building your data pipelines.

Useful UI

Monitor, schedule, and manage your workflows via a sophisticated web app, offering you

full insight into the status of your tasks.

Integration

Apache Airflow provides many plug-and-play integrations, such as IBM Cloudant, that are

ready to execute your tasks.

Easy to Use

Anyone with Python knowledge can deploy a workflow. Airflow does not limit the scope

of your pipelines.

And finally, the open source feature.

Whenever you want to share your improvement, you can do this by opening a pull request.

Airflow has many active users who are sharing their experiences in the Apache Airflow community.

Apache Airflow pipelines are built on four main principles. They are:

Scalable: Airflow has a modular architecture and uses a message queue to orchestrate an

arbitrary number of workers. It is ready to scale to infinity.

Dynamic: Airflow pipelines are defined in Python, and allow dynamic pipeline generation.

Thus, your pipelines can contain multiple simultaneous tasks.

Extensible: You can easily define your own operators and extend libraries to suit your

environment.

And, Lean: Airflow pipelines are lean and explicit. Parameterization is built into its

core using the powerful Jinja templating engine.

Apache Airflow has supported many companies in reaching their goals. For example,

Sift used Airflow for defining and organizing Machine Learning pipeline dependencies,

SeniorLink increased the visibility of their batch processes and decoupled them,

Experity deployed Airflow as an enterprise scheduling tool, and

Onefootball used Airflow to orchestrate SQL transformations in their data warehouses,

and to send daily analytics emails.

In this video, you learned that:

Apache Airflow is a platform to programmatically author, schedule, and monitor workflows.

The five main features of Airflow are its use of Python, its intuitive and useful user

interface, extensive plug-and-play integrations, ease of use, and the fact that it is open

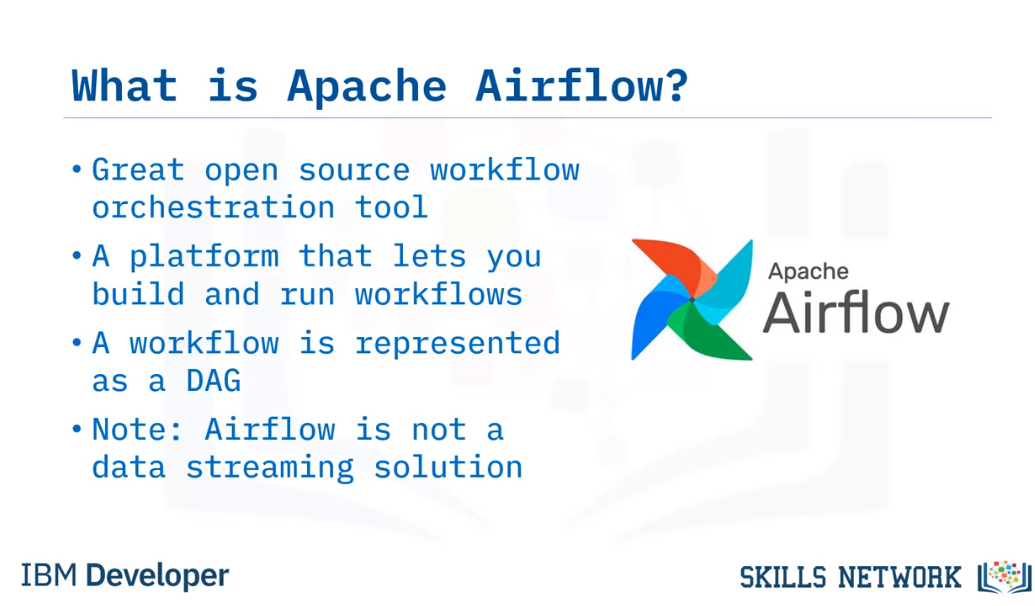
source.

You also learned that Apache Airflow is scalable, dynamic, extensible, and lean.

And finally, defining and organizing machine learning pipeline dependencies with Apache

Airflow is one of the common use cases.

Work flow orchestration tool



Airflow Architecture

A screenshot of a computer

Description automatically generatedAirflow comes with a built-in Scheduler, which handles the triggering of all scheduled workflows.

The Scheduler is responsible for submitting individual tasks from each scheduled workflow

to the Executor.

The Executor handles the running of these tasks by assigning them to Workers, which

then run the tasks.

The Web Server , serves Airflow’s powerful interactive User Interface.

From this UI, you can inspect, trigger and debug any of your DAGs and their individual

tasks.

The DAG Directory contains all of your DAG files, ready to be accessed by the Scheduler,

the Executor, and each of its employed Workers.

Finally, Airflow hosts a Metadata Database, which is used by the Scheduler, Executor,

and the Web Server to store the state of each DAG and its tasks.

A DAG specifies the dependencies between tasks, and the order in which to execute them. The

tasks themselves describe what to do.

A diagram of a computer

Description automatically generated

In this example DAG, the tasks include data ingestion,

data analysis, saving the data, generating reports, and triggering other systems, such

as reporting any errors by email.

A screenshot of a computer

Description automatically generated

et’s have a look at the life cycle of a task’s state.

In this diagram, you can see how Apache Airflow might assign states to a task during its life

cycle.

No status: This means that the task has not yet been queued for execution.

Scheduled: The scheduler has determined that the task’s dependencies are met and has

scheduled it to run.

Removed: For some reason, the task has vanished from the DAG since the run started.

Upstream failed: An upstream task has failed.

Queued: The task has been assigned to the Executor and is waiting for a worker to become

available.

Running: The task is being run by a worker.

Success: The task finished running without errors.

Failed: The task had an error during execution and failed to run, and

Up for retry: The task failed but has retry attempts left and will be rescheduled.

Ideally, a task should flow through the scheduler from ‘no status’, to ‘scheduled’,

to ‘queued’, to ‘running’, and finally to ‘success.’

Apache Airflow Features

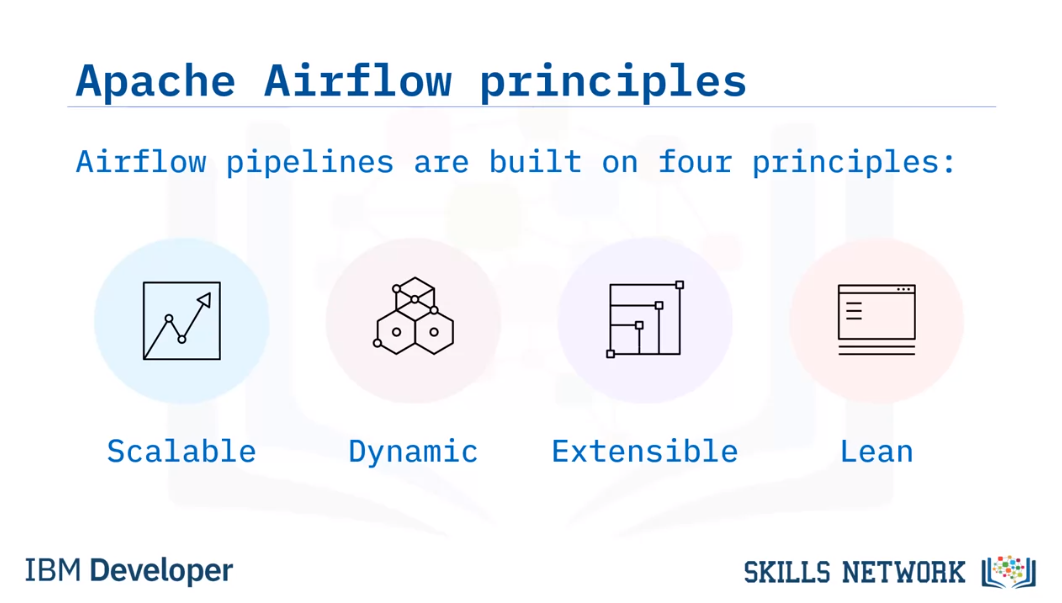
A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

Apache Airflow principles



Scalable: Airflow has a modular architecture and uses a message queue to orchestrate an

arbitrary number of workers. It is ready to scale to infinity.

Dynamic: Airflow pipelines are defined in Python, and allow dynamic pipeline generation.

Thus, your pipelines can contain multiple simultaneous tasks.

Extensible: You can easily define your own operators and extend libraries to suit your

environment.

And, Lean: Airflow pipelines are lean and explicit. Parameterization is built into its

core using the powerful Jinja templating engine.

Use Cases

A screen shot of a white background

Description automatically generated

## Advantages of Using Data Pipelines as DAGs in Apache Airflow

Welcome to

Advantages of Representing Data Pipelines as DAGs in Apache Airflow.

After watching this video, you will be able to:

Define what a Directed Acyclic Graph, or DAG, is.

Describe workflows as DAGs of tasks and dependencies.

Outline the components of a DAG definition file.

Describe how Apache Airflow Scheduler executes tasks on an array of workers, and

List key advantages of defining workflows as code.

A 'DAG' is a special kind of graph, called a directed acyclic graph.

A simple graph consists of nodes and edges, like this. Here the circles are called

nodes, and the lines connecting pairs of nodes are called edges.

A directed graph is also a graph, but it has a bit more structure. As you can see here,

each edge has a specified direction. It connects a starting node with another node.

And lastly, the acyclic part means there are no loops, or sequences of directed

edges that return to a node in the chain, such as the red cycle shown here.

Let's take a look at a few examples of DAGs.

The simplest nontrivial DAG has a single directed edge and looks like

this. It has a single root node which is connected to a single terminal node.

Here's another DAG, which we've already seen. It also has single root and terminal nodes.

Here's an example of a tree , which is a commonly used graph for representing

family trees or directory structures.

All trees are DAGs, but not all DAGs are trees. For example,

this DAG is not a tree since it has more than one root node.

A DAG doesn't impose those restrictions, so a single node can have multiple parents

and there may be multiple nodes with no parents.

DAGs are used to represent workflows or pipelines in Apache Airflow.

Each task performed by your data pipeline is represented as a node in a DAG, while

each of the dependencies between two tasks in your pipeline are represented as a directed

edge in the DAG. In other words, edges define the order in which the two tasks should run.

Thus, DAGs are used in Airflow to define what tasks should run,

and in what sequence they should run.

A DAG is defined in a Python script, which represents the DAG’s structure.

Thus, the tasks and their dependencies are defined as code.

Also, scheduling instructions are specified as code in the DAG's script.

Let's take a closer look at the nodes, or tasks in a DAG.

Just like the DAG itself, each task performed within your DAG is also written in Python.

Each task implements an operator: for example, a Python operator is used to deploy some Python

code, a SQL operator to run a SQL query, and a Bash operator can be used to run a Bash command.

Operators are used to define what each task in your DAG does.

Sensors are a class of operators which are used to poll for a certain time or condition to be met.

For example, you can use a sensor to check every 30 seconds whether a file exists,

or whether another DAG has finished running.

There are many other types of operators, including email and HTTP request operators.

Play video starting at :3:33 and follow transcript3:33

An Apache Airflow DAG is a python script consisting of the following logical blocks:

Library imports

DAG arguments

DAG definition

Task definitions

Task pipeline

Let's briefly go over an example.

The first block of your DAG definition script

is where you import any Python libraries that you require. For example, the ‘from

airflow import DAG’ command to import the DAG module from the Airflow collection.

The next block of code is for specifying default arguments for your DAG,

such as its default ‘start date’.

Next comes the DAG definition, or instantiation block for your DAG,

which specifies things like your default arguments.

Continuing along with our example DAG code, individual task definitions,

which are the nodes of the DAG, form your DAG’s next building block.

In this example we have two tasks, which happen to be Bash operators.

Finally, the task pipeline specifies the dependencies between your tasks.

Here 'task two' depends on the result of 'task

one'. And this forms the last logical block of your DAG script.

Your new DAG has been created, but it hasn't yet been deployed. To that end,

Airflow Scheduler is designed to run as a persistent service

within an Airflow production environment.

Apache Airflow Scheduler can be used to deploy your workflow on an array of workers.

It follows the tasks and dependencies that you specified in your DAG.

Once you start an Airflow Scheduler instance,

your DAGs will start running based on the 'start date' you specified as code in each of your DAGs.

After that, the Scheduler triggers each subsequent DAG run

according to the schedule interval you specified.

One of the key advantages of Apache Airflow's approach to representing data pipelines as

DAGs is the fact that they are expressed as code.

When workflows are defined as code, they become more

Maintainable: Developers can follow explicitly what

has been specified, by reading the code.

Versionable: Code revisions can easily be tracked by a version control system such as Git.

Collaborative: Teams of developers can easily collaborate

on both development and maintenance of the code for the entire workflow, and

Testable: Any revisions can be passed through

unit tests to ensure the code still works as intended.

Play video starting at :6:4 and follow transcript6:04

In this video, you learned that:

In Apache Airflow, DAGs are workflows defined as Python code.

Tasks, which are the nodes in your DAG,

are created by implementing Airflow's built-in operators.

Pipelines are specified as dependencies between tasks,

which are the directed edges between nodes in your DAG.

Airflow Scheduler schedules and deploys your DAGs.

And finally, the key advantage of Apache Airflow's approach to representing data pipelines as

DAGs is the fact that they are expressed as code.

Accordingly, it makes your data pipelines more maintainable, testable, and collaborative.