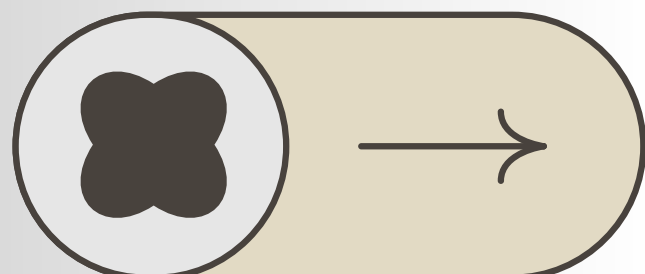


# Understand Apache Airflow



Apache  
Airflow



Apache Airflow® is an open source tool for programmatically authoring, scheduling, and monitoring data pipelines. Every month, millions of new and returning users download Airflow and it has a large, active open source community. The core principle of Airflow is to define data pipelines as code, allowing for dynamic and scalable workflows.

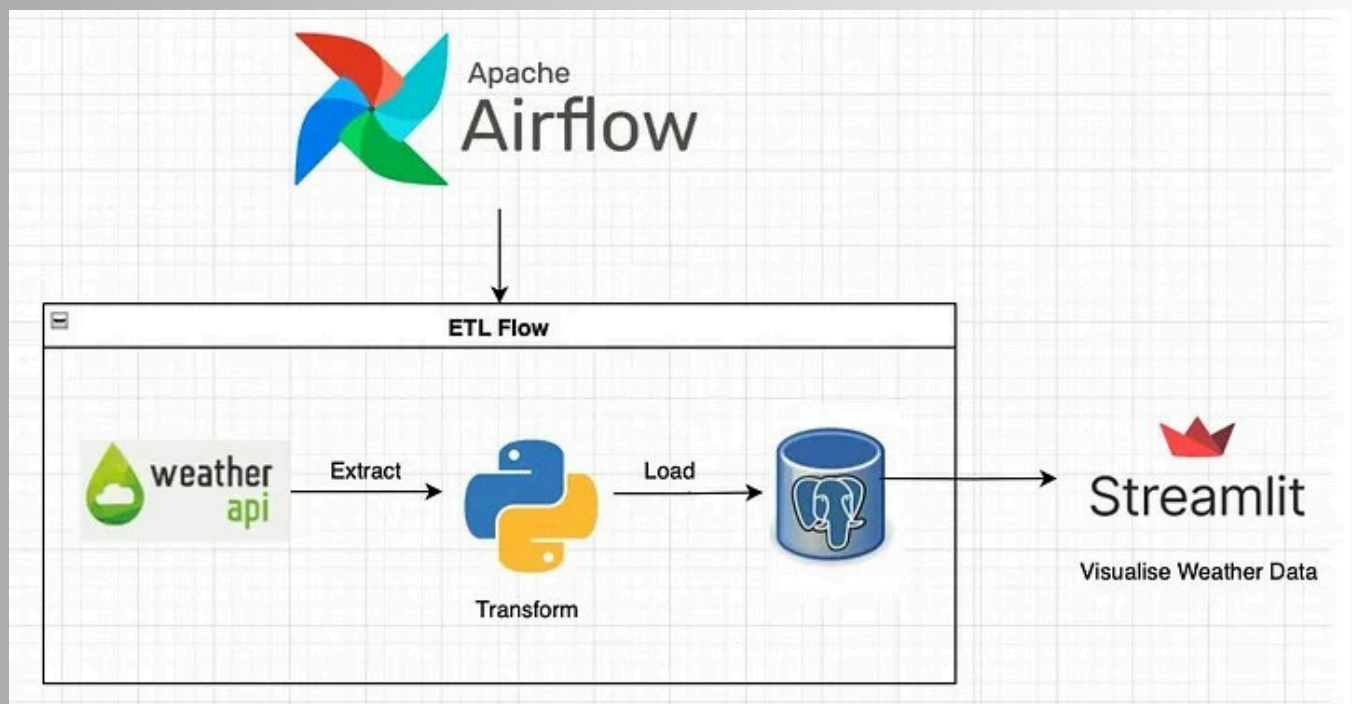
This guide offers an introduction to Apache Airflow and its core concepts.

You'll learn about:

- Why you should use Airflow.
- Common use cases for Airflow.
- How to run Airflow.
- Important Airflow concepts.
- where to find resources to learn more about Airflow.

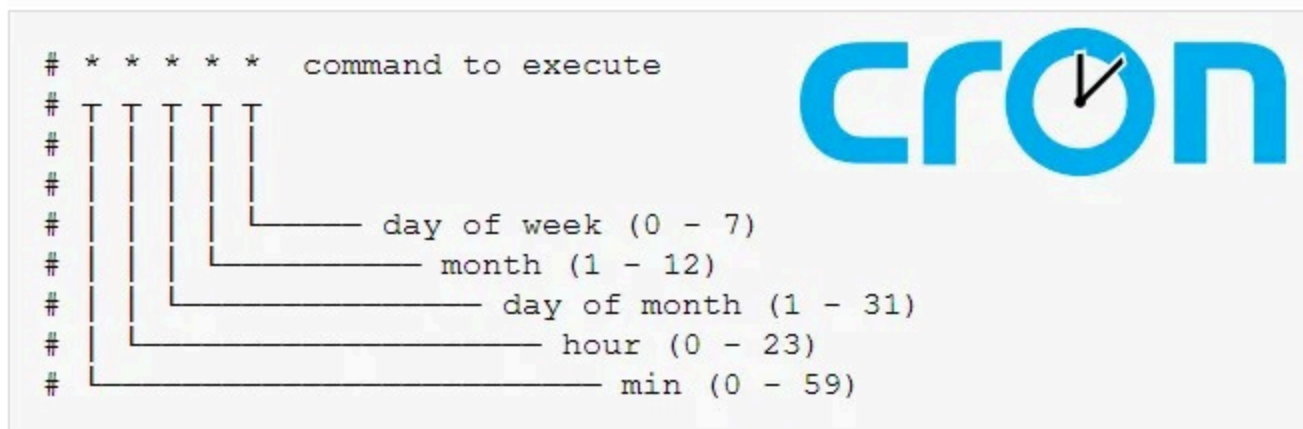
In the world of data engineering, one of the most critical tasks you'll encounter is building data pipelines.

At its core, a data pipeline involves extracting data from multiple sources, performing transformations, and loading the processed data into a target location.



While you could perform this using a simple Python script, the complexity grows when you need to manage hundreds of such pipeline

The Problem with Simple Python Scripts and Cron Jobs



Let's start with the basics. You can build a data pipeline using a simple Python script. For example:

1. Extract data from APIs or databases.
2. Transform the data (e.g., clean, aggregate, or enrich it).
3. Load the data into a target location (e.g., a data warehouse or cloud storage).

To automate this process, you can use Cron jobs to schedule your script to run at specific intervals. For instance, you might run a script daily to update your dataset.

However, this approach has limitations:

- Scalability: Managing hundreds of pipelines with Cron jobs becomes almost impossible.
- Dependencies: If one task depends on another, ensuring proper sequencing is challenging.
- Error Handling: Monitoring failures and retries manually is time-consuming.

As data grows exponentially (90% of the world's data was generated in the last two years!), businesses need robust tools to process and analyze this data efficiently.

This is where Apache Airflow comes into play.

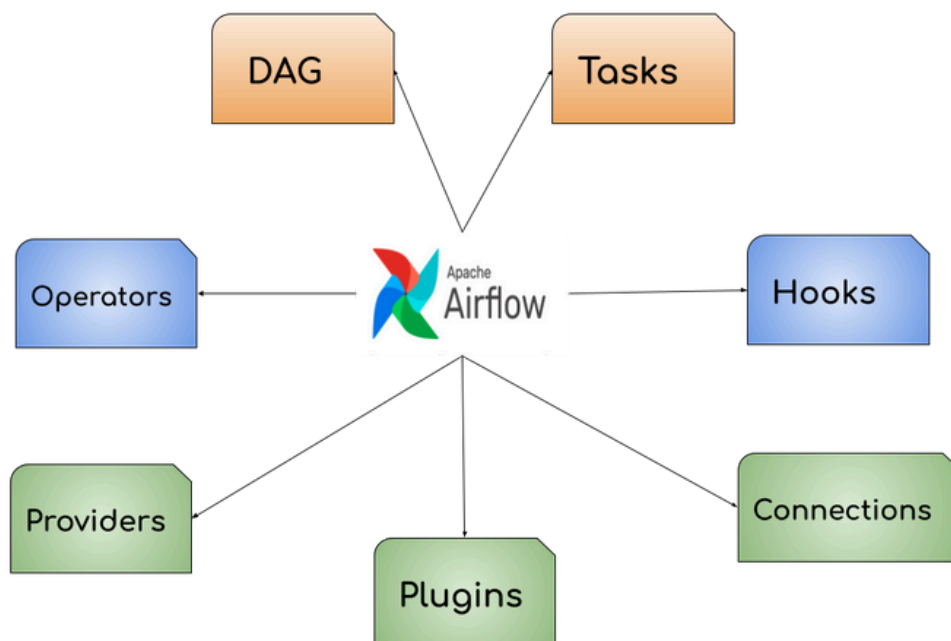


# Apache Airflow

## What is Apache Airflow?

Apache Airflow is a workflow management tool designed to handle complex task sequences. It is a platform designed to programmatically author, schedule, and monitor workflows.

It was originally developed by Airbnb in 2014 and later became part of the Apache Software Foundation in 2016.



Today, it's one of the most widely adopted tools in the data engineering community, with over 10 million monthly pip installs and 200,000 GitHub stars.

Airflow's viral adoption wasn't driven by millions in VC-funded marketing, a rich user interface, or a reputation for being easy to install and run.

What accounted for much of Airflow's initial popularity was its promise of pipelines-as-code.

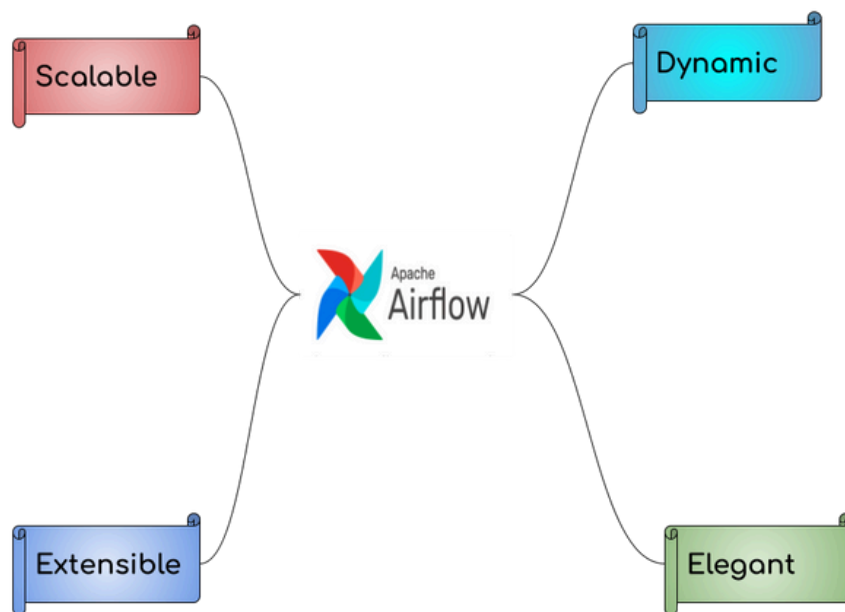
Before this, we could write Python script but it came with its problems there were some enterprise tools available such as Informatica, Alteryx but they were expensive and you couldn't customize them as per your use case

This is where Apache Airflow shined, which means anyone can use it and it gave different features for data teams to build, run, and manage data pipelines on scale.

So now we understand Why Apache Airflow and Where it fits but let's understand What is Apache Airflow and the different parts of it

Apache Airflow is a workflow management tool, a workflow is like a series of tasks that needs to be executed in a specific order

So talking about the previous example, we might have data coming from multiple sources, we want to transform this data into some specific format and then load it at the target location



This entire job is called a Workflow

The same terminology is used in Apache Airflow but it is called DAG (Directed Acyclic Graph)

It looks something like this

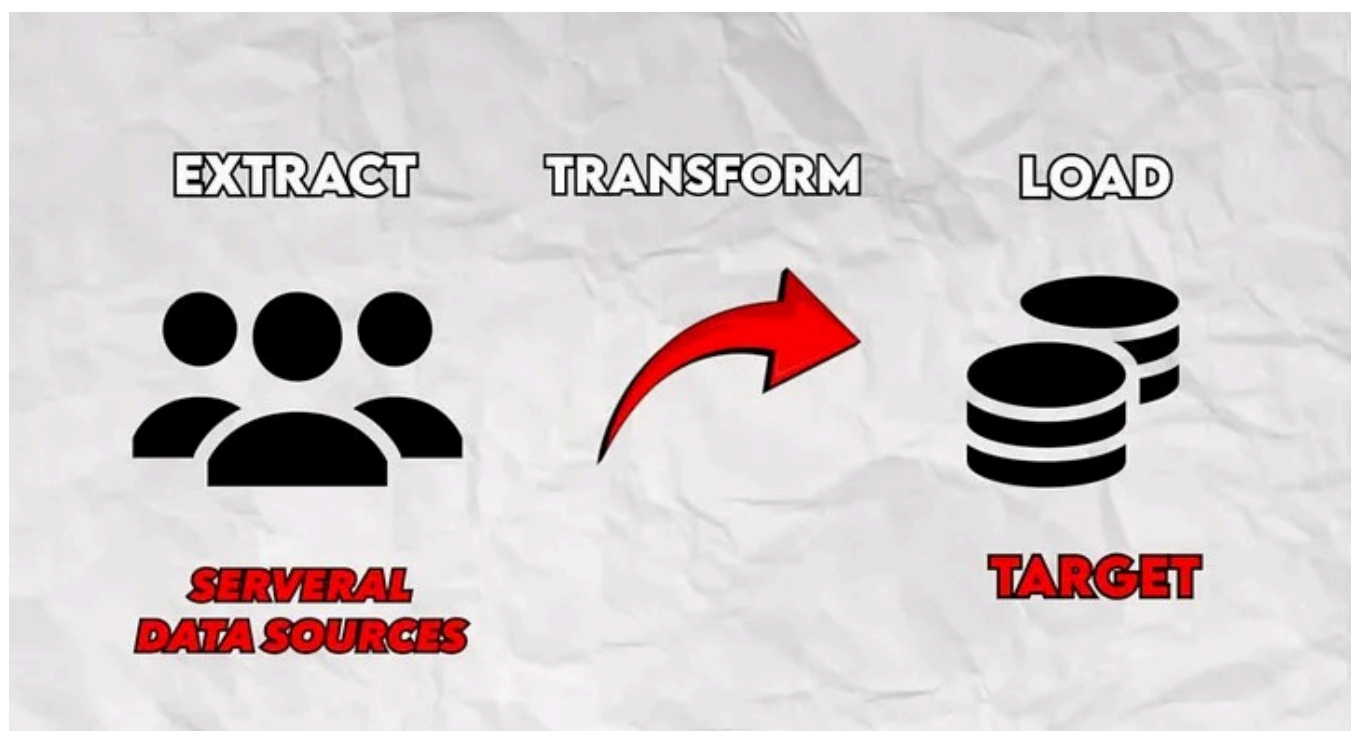
Before this, we could write Python script but it came with its problems there were some enterprise tools available such as Informatica, Alteryx but they were expensive and you couldn't customize them as per your use case

This is where Apache Airflow shined, which means anyone can use it and it gave different features for data teams to build, run, and manage data pipelines on scale.

So now we understand Why Apache Airflow and Where it fits but let's understand What is Apache Airflow and the different parts of it

Apache Airflow is a workflow management tool, a workflow is like a series of tasks that needs to be executed in a specific order

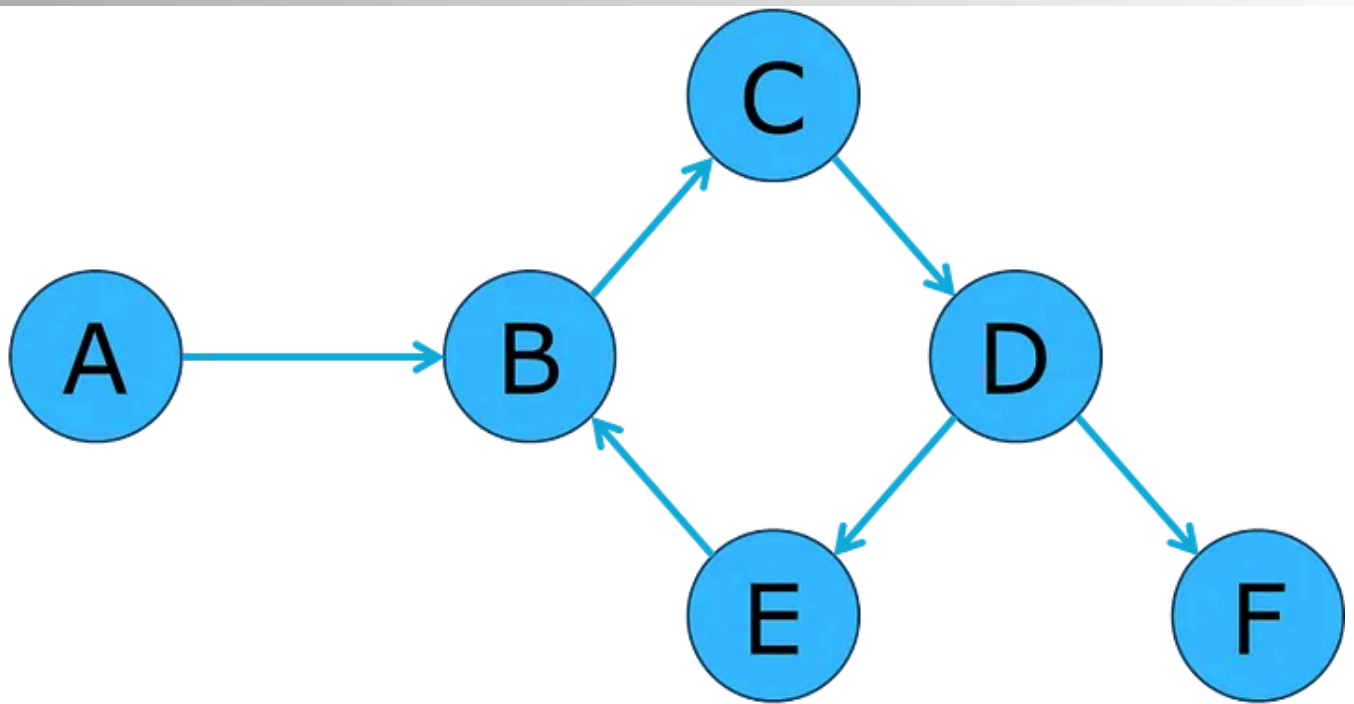
So talking about the previous example, we might have data coming from multiple sources, we want to transform this data into some specific format and then load it at the target location



This entire job is called a Workflow

The same terminology is used in Apache Airflow but it is called DAG (Directed Acyclic Graph)

It looks something like this



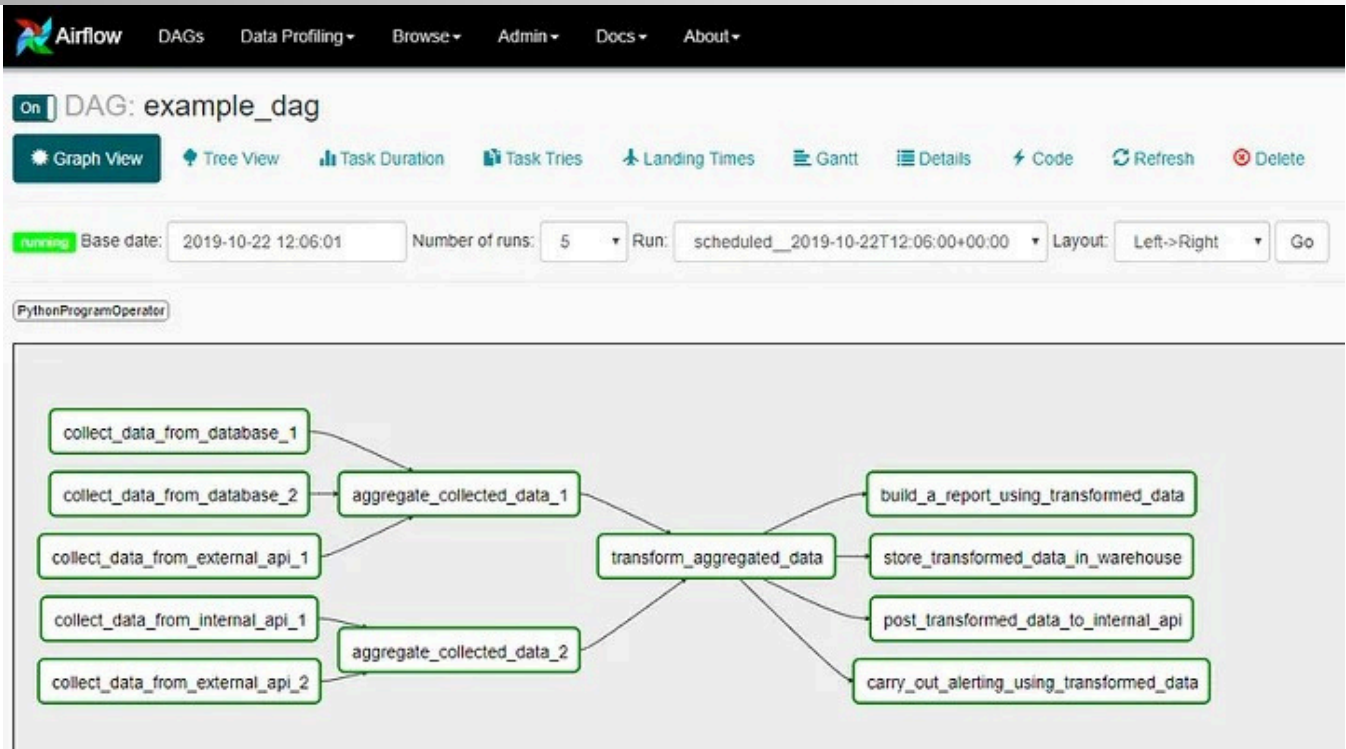
At the heart of Airflow is the DAG, which defines a collection of tasks and their dependencies in a specific order.

This is a core computer science concept.

Think of it as a blueprint of your workflow, ensuring that tasks run in the sequence.

- 👉 Directed: Tasks move in a certain direction.
- 👉 Acyclic: No loops! Tasks don't run in circles. It can only move in one direction only
- 👉 Graph: A visual representation of the tasks.





This entire flow is called DAG now individual boxes that you see are called Task

So DAG defines a complete blueprint and Tasks are your actual logic

Such as in this example Tasks are

Reading data from External Sources -> Aggregating data -> Doing transformation -> Loading it somewhere in the target location

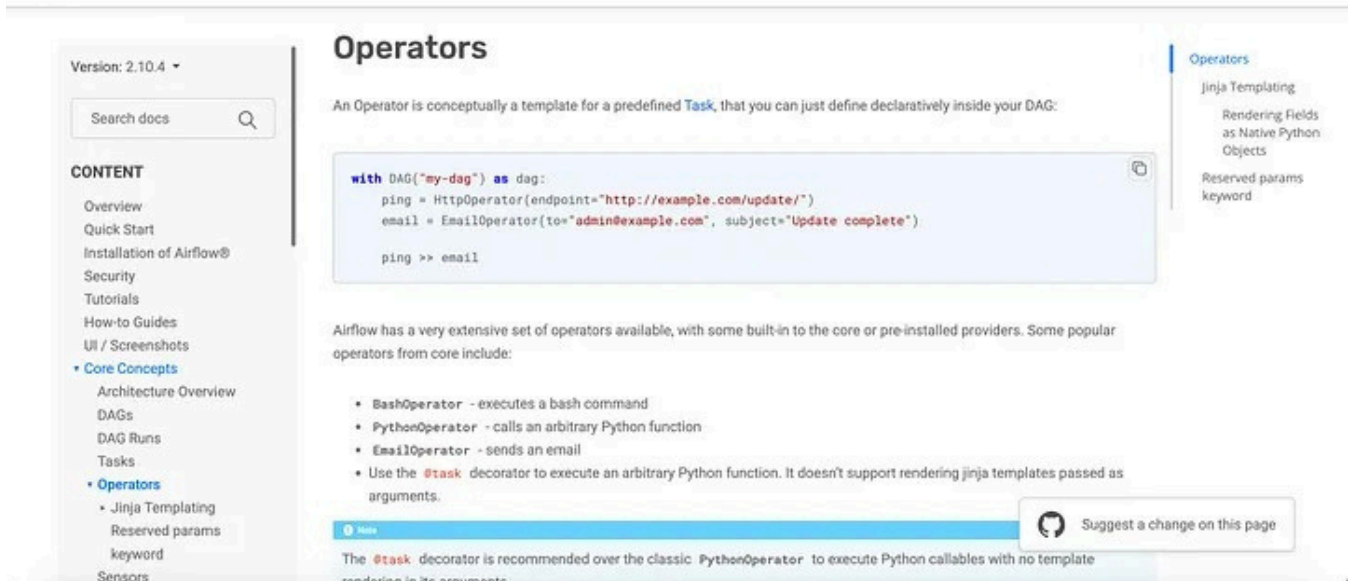
Each Task runs independently, in its process.

All of these tasks are executed in specific order only, once the first part that is the extraction of data is completed only the aggregation task will run

To create a Task we need to use Operators.

Think of Operators as different functions that provide airflow to create tasks, there are different types of operators





<https://airflow.apache.org/docs/apache-airflow/2.0.0/executor/index.html>

- BashOperator: Executes a bash command.
- PythonOperator: Calls a Python function.
- EmailOperator: Sends an email.
- PostgresOperator: Interacts with a PostgreSQL database.

For different types of work, you have operators available to make things easier.

There are many operators available:

Let's say you are working with a PostgreSQL database or S3 object storage you will have operators so that you can make connections and do your work

So Operators are different functions provided by Airflow that make up as Tasks and the collection of Tasks is called as complete DAG

Now to run this DAG we need Executors

Version: 2.0.0

Search docs

## CONTENT

- Home
- Project
- License
- Quick Start
- Basic Airflow architecture
- Installation
- Upgrading to Airflow 2.0+
- Upgrade Check Script
- Tutorial
- Tutorial on the Taskflow API
- How-to Guides
- UI / Screenshots
- Concepts
- Scheduler
- Executor**
  - Supported Backends
  - DAG Runs

Home / Executor

# Executor

Executors are the mechanism by which task instances get run.

Airflow has support for various executors. Current used is determined by the `executor` option in the `[core]` section of the configuration file. This option should contain the name executor e.g. `KubernetesExecutor` if it is a core executor. If it is to load your own executor, then you should specify the full path to the module e.g. `my_acme_company.executors.MyCustomExecutor`.

Note

For more information on setting the configuration, see [Setting Configuration Options](#).

If you want to check which executor is currently set, you can use `airflow config get-value core executor` command as in the example below.

```
$ airflow config get-value core executor
SequentialExecutor
```



Suggest a change on this page

<https://airflow.apache.org/docs/apache-airflow/2.0.0/executor/index.html>

They Determine HOW tasks are run.

There are several types:

- 👉 SequentialExecutor: Runs tasks sequentially.
- 👉 LocalExecutor: Runs tasks in parallel on a single machine.
- 👉 CeleryExecutor: Distributes tasks across multiple machines.

A Quick Example: Building a DAG

Let's walk through a simple example of defining a DAG in Airflow. Suppose we want to:

1. Extract data from an API.
2. Transform the data.
3. Load it into a database.

Here's how you can define this workflow:

```
from airflow import DAG
from airflow.operators.python_operator import PythonOperator
from datetime import datetime

# Define the DAG
dag = DAG(
    'my_data_pipeline',
```

```

description='A simple data pipeline',
schedule_interval='@daily',
start_date=datetime(2023, 10, 1),
catchup=False
)

# Define tasks
def extract_data():
    print("Extracting data from API...")

def transform_data():
    print("Transforming data...")

def load_data():
    print("Loading data into database...")

extract_task = PythonOperator(
    task_id='extract',
    python_callable=extract_data,
    dag=dag
)

transform_task = PythonOperator(
    task_id='transform',
    python_callable=transform_data,
    dag=dag
)

load_task = PythonOperator(
    task_id='load',
    python_callable=load_data,
    dag=dag
)

# Set task dependencies
extract_task >> transform_task >> load_task

```

In this example:

- We define a DAG named `my_data_pipeline` that runs daily.
- We create three tasks using the `PythonOperator`.
- We set dependencies so that tasks run in sequence: `extract` → `transform` → `load`.

## Next Steps

If you're ready to dive deeper into Airflow, here are some recommendations:

# Airflow Best-Practices



Building Airflow DAGs can be tricky. There are a few best practices to keep in mind when building data pipelines and workflows, not only with Airflow, but with other tooling.

## Modularity

With tasks, Airflow helps to make modularity easier to visualize. Don't try to do too much in a single task. While an entire ETL pipeline can be built in a single task, this would make troubleshooting difficult. It would also make visualizing the performance of a DAG difficult.

When creating a task, it's important to make sure the task will only do one thing, much like functions in Python.

Take a look at the example below. Both DAGs do the same thing and fail at the same point in the code. However, in the DAG on the left, it's clear that the load logic is causing the failure, while this is not quite clear from the DAG on the right.



# Determinism



A deterministic process is one that produces the same result, given the same input. When a DAG runs for a specific interval, it should generate the same results every time. While a more complex characteristic of data pipelines, determinism is important to ensure consistent results.

With Airflow, leverage Jinja-templating to pass templated fields into Airflow operators rather than using the `datetime.now()` function to create temporal data.

Idempotency

What happens if you run a DAG for the same interval twice? How about 10 times? Will you end up with duplicate data in your target storage medium? Idempotency ensures that even if a data pipeline is executed multiple times, it was as if the pipeline was only executed once.

To make data pipelines deterministic, think about incorporating the following logic into your DAGs:

Overwrite files when DAGs are re-run, rather than creating a new file with a different name when run for the same interval

Use a delete-write pattern to push data into databases and data warehouses rather than INSERTing, which may cause duplicates.

# Orchestration is not Transformation



Airflow isn't meant to process massive amounts of data. If looking to run transformations on more than a couple gigabytes of data, Airflow is still the right tool for the job; however, Airflow should be invoking another tool, such as dbt or Databricks, to run the transformation.

Typically, tasks are executed locally on your machine or with worker nodes in production. Either way, only a few gigabytes of memory will be available for any computational work that is needed.

Focus on using Airflow for very light data transformation and as an orchestration tool when wrangling larger data.

# Apache Airflow in Industry



With Airflow's ability to define data pipelines as code and its wide variety of connectors and operators, companies across the world rely on Airflow to help power their data platforms.

In industry, a data team may work with a wide variety of tools, from SFTP sites to cloud file storage systems to a data lakehouse. To build a data platform, it's paramount for these disparate systems to be integrated.

With a vibrant open-source community, there are thousands of prebuilt connectors to help integrate your data tooling. Want to drop a file from S3 into Snowflake? Lucky for you, the `S3ToSnowflakeOperator` makes it easy to do just that! How about data quality checks with Great Expectations? That's already been built too.

If you can't find the right prebuilt tool for the job, that's okay. Airflow is extensible, making it easy for you to build your own custom tools to meet your needs.

When running Airflow in production, you'll also want to think about the tooling that you're using to manage the infrastructure. There are a number of ways to do this, with premium offerings such as Astronomer, cloud-native options like MWAA, or even a homegrown solution.

Typically, this involves a tradeoff between cost and infrastructure management; more expensive solutions may mean less to manage, while running everything on a single EC2 instance may be inexpensive but tricky to maintain.



# Conclusion



Apache Airflow is an industry-leading tool for running data pipelines in production. Providing functionality such as scheduling, extensibility, and observability while allowing data analysts, scientists, and engineers to define data pipelines as code, Airflow helps data professionals focus on making business impact.

It's easy to get started with Airflow, especially with the Astro CLI, and traditional operators and the TaskFlow API make it simple to write your first DAGs. When building data pipelines with Airflow, make sure to keep modularity, determinism, and idempotency at the forefront of your design decisions; these best practices will help you avoid headaches down the road, especially when your DAGs encounter an error.

With Airflow, there's tons to learn. For your next data analytics or data science project, give Airflow a try. Experiment with prebuilt operators, or build your own. Try sharing data between tasks with traditional operators and the TaskFlow API. Don't be afraid to push the limits. If you're ready to get started, check out DataCamp's [Introduction to Airflow in Python](#) course, which covers the basics of Airflow and explores how to implement complex data engineering pipelines in production.

# *Thanks for reading!*

*Follow my profile for more coding  
related contents*

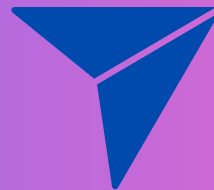
Like



Comment



Share



*@rganesh0203*