Rohit Jain

Apache Airflow

Quick Start Guide

Quick Start Guide On Apache Airflow

Part 1 – Foundations

Part 2 – Building Pipelines

Part 3 – Real-World Use Case

Part 4 – Deployment & Scaling

Part 5 – Beyond the Basics

Appendices

Introduction

Data engineering is no longer a niche discipline. As companies rely increasingly on automated pipelines to move, transform, and serve data, the ability to orchestrate those workflows reliably has become a core skill for engineers, analysts, and data scientists alike. This book will teach you Apache Airflow — the industry-standard tool for building, scheduling, and monitoring data pipelines — from the ground up.

What You Will Build

By the end of this book, you won't just understand Airflow conceptually — you'll have built and deployed a real, production-style data pipeline. Specifically, you will:

- Design and run your first DAG (Directed Acyclic Graph) locally
- Build a complete ETL pipeline that pulls data from an API, transforms it, and loads it into PostgreSQL
- Add production-grade features like retries, alerts, and SLA monitoring
- Deploy your pipeline to AWS MWAA (Managed Workflows for Apache Airflow)
- Explore advanced techniques including dynamic task mapping, custom operators, and CI/CD for DAGs

Who This Book Is For

This guide is written for anyone who works with data and wants to automate and orchestrate their workflows. It is most useful if you are:

- A data engineer looking to adopt Airflow as your orchestration layer
- A data analyst or scientist whose scripts and notebooks need to run on a schedule
- A software developer moving into data infrastructure for the first time
- A DevOps or cloud engineer tasked with deploying and managing Airflow in production

You do not need prior experience with Airflow. However, you will get the most out of this book if you are comfortable with the following:

- Python - Comfortable writing functions, classes, and scripts
- Command line / Terminal - Basic navigation and running commands
- SQL - Basic queries — SELECT, INSERT, WHERE
- Cloud (AWS) - Helpful but not required for Parts 1–3
- Docker - Helpful for Chapter 3's Docker setup option

If you know Python and have run a script from the terminal before, you are ready to begin.

How This Book Is Structured

The book is divided into five parts, each building on the last:

Part 1 – Foundations (Chapters 1–4) introduces Airflow's core concepts, walks you through installation, and gets your first DAG running locally.

Part 2 – Building Pipelines (Chapters 5–9) covers the tools you'll use every day — Python operators, scheduling, secrets management, monitoring, and code organization best practices.

Part 3 – Real-World Use Cases (Chapters 10–13) puts everything together in a working ETL pipeline, then layers on alerts, external integrations, and CI/CD.

Part 4 – Deployment & Scaling (Chapters 14–17) moves your pipelines to production using AWS MWAA, with guidance on debugging, pitfalls, and alternatives like Astronomer and Google Cloud Composer.

Part 5 – Beyond the Basics (Chapters 18–20) explores advanced Airflow features including custom operators, dynamic task mapping, and performance optimization.

A Note on Airflow Versions

This book is written for Apache Airflow 2.x. If you are using Airflow 1.x, some syntax and features will differ — particularly around scheduling, the TaskFlow API, and dynamic task mapping. All code examples have been written and tested against Airflow 2.6+.

How to Use This Book

If you are brand new to Airflow, read Parts 1 and 2 in order — the concepts build on each other. If you already have Airflow basics down, you can jump directly to Part 3 for the real-world pipeline, or Part 4 if your immediate need is deployment. Every chapter ends with a summary and a preview of what comes next, so you always know where you are in the journey. Let's get started.

Chapter 1 – What Is Apache Airflow and Why It Matters

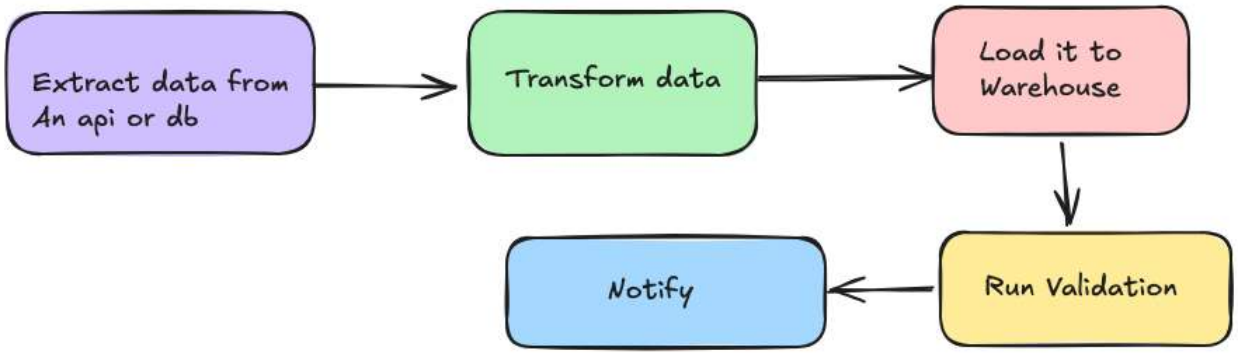
🚀 Introduction

Modern data platforms rely on pipelines that extract, transform, validate, and load data across multiple systems. These pipelines often consist of multiple dependent tasks that must execute in a specific order and on a defined schedule. Apache Airflow is a workflow orchestration framework designed to manage these pipelines reliably and at scale. This chapter introduces the core concepts behind Airflow and explains why orchestration is necessary in real-world data engineering environments.

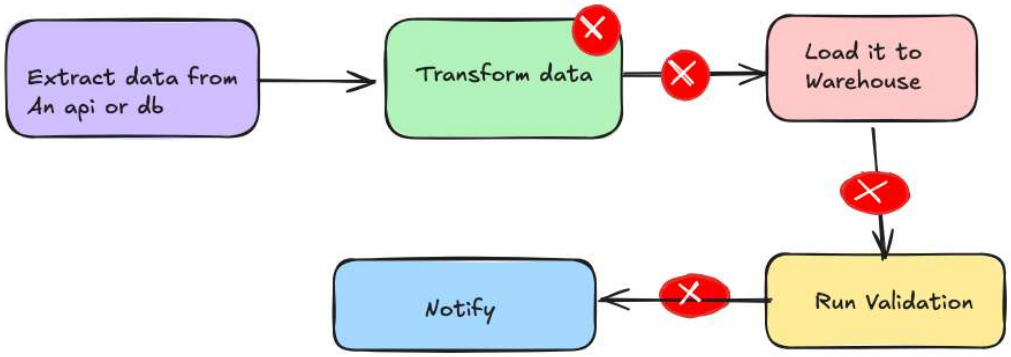
What Is Orchestration?

Orchestration is the automated coordination of multiple interdependent tasks.

In a typical data workflow:



Each task depends on the previous one. If a transformation fails, the load step should not execute. If the API is temporarily unavailable, the system should retry.



Orchestration systems provide:

1. Task dependency management
2. Scheduling
3. Retries
4. Logging
5. Monitoring
6. Failure handling

Without orchestration, these responsibilities must be implemented manually, which becomes difficult to maintain as systems grow.

Why Airflow?

Every growing data system eventually hits a wall:

cron jobs everywhere, scripts failing silently, and team members unsure what runs when.

That's the moment engineers discover **Apache Airflow** — an open-source platform that brings *order, visibility, and scalability* to workflow orchestration.

Airflow lets you define, schedule, and monitor complex workflows using simple Python code. Instead of manually managing dependencies between ETL scripts, you describe them as a **Directed Acyclic Graph (DAG)** — a fancy name for a flowchart where each node represents a task and the edges define the order of execution.

🧩 Why Airflow Exists

Before Airflow, data engineers stitched together pipelines using:

- cron jobs (crontab),
- bash scripts, or
- custom schedulers built in-house.

These solutions worked… until they didn't:

- No central monitoring or retry logic
- No dependency management between tasks
- Difficult to scale and debug

Airbnb built Apache Airflow in 2014 to solve exactly that problem.

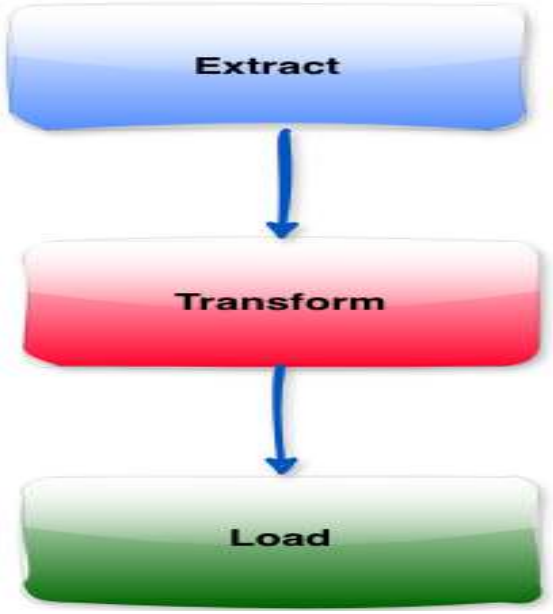
It's now used by companies like Netflix, Shopify, and NASA — and remains the standard tool for orchestrating data pipelines in modern analytics and ML workflows.

⚙️ How Airflow Works at a Glance

Conceptually, Airflow is built around five key components:

1. DAG - A Directed Acyclic Graph — the blueprint of your workflow
2. Task - A single step in the pipeline (e.g., extract data, transform, load)
3. Operator - Pre-built logic for what the task should do (PythonOperator, BashOperator, etc.)
4. Scheduler - Decides when and what to run based on your DAG's schedule
5. Executor + Workers - Run tasks in parallel across available resources

Visualization cue:



Each box is a task; arrows define dependencies. Airflow reads this DAG, schedules runs, and tracks execution through its **web UI**.

🧠 What Makes Airflow Powerful

1. **Code as Configuration** – Workflows are written in Python, not JSON files or GUIs.
2. **Modularity** – Operators, sensors, and hooks can be reused across projects.
3. **Extensibility** – Integrates with AWS, GCP, Snowflake, Databricks, Slack, and more.
4. **Visibility** – Rich UI showing DAGs, logs, and task states.
5. **Reliability** – Built-in retries, alerting, and failure handling.

Example use cases:

- Daily ETL pipelines from APIs to databases
- ML model retraining orchestration
- Report generation and email automation
- CI/CD data validation

🧰 Mini Example

Let's preview what an Airflow DAG looks like:

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

def greet():
    print("Hello from Airflow!")

with DAG(
    dag_id="hello_airflow",
    start_date=datetime(2025, 1, 1),
    schedule="@daily",
    catchup=False,
) as dag:
    greet_task = PythonOperator(
        task_id="greet_task",
        python_callable=greet,
    )
```

Save this file under /dags/hello_airflow.py, and Airflow will automatically detect it.

💬 Summary

Airflow transforms scattered scripts into well-structured, observable, and repeatable workflows.

It's not just a scheduler — it's a *platform* for orchestrating data-driven systems.

By the end of this eBook, you'll be able to:

- Design robust DAGs,
- Deploy them locally and on AWS MWAA,
- Automate real-world data pipelines end-to-end.

➡️ Next Chapter Preview

Chapter 2 — Core Concepts: DAGs, Tasks, and Operators

We'll dive deeper into Airflow's building blocks and create your first multi-task workflow.

Chapter 2 – Core Concepts: DAGs, Tasks, Operators, Schedulers, and Executor

🧩 Why These Concepts Matter When you start with Airflow, three terms show up everywhere — DAGs, Tasks, and Operators*.

Understanding how they interact is the key to designing workflows that are maintainable, debuggable, and scalable.

Think of it this way:

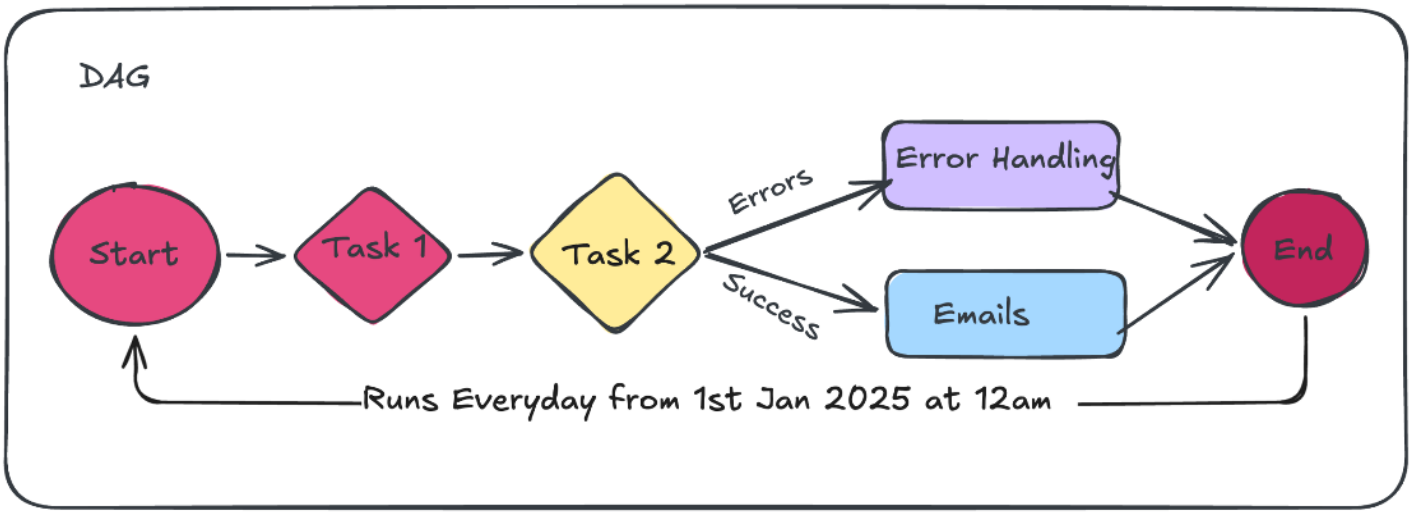
> A DAG is the map
>
> **Operators are the *building blocks***
>
> **Tasks are the *instances of those blocks on the map***

Once this mental model clicks, Airflow becomes intuitive.

🗺️ 1. DAG – The Blueprint of Your Workflow

A **DAG** (*Directed Acyclic Graph*) describes **how** and **when** your tasks run.

- **Directed** → each edge has a clear direction (Task A → Task B).
- **Acyclic** → no loops allowed; data always flows forward.
- **Graph** → a collection of nodes (tasks) and edges (dependencies).



In code:

```
from airflow import DAG
from datetime import datetime

dag = DAG(
    dag_id="daily_sales_pipeline",
    start_date=datetime(2025, 1, 1),
    schedule="@daily",
    catchup=False,
    description="ETL pipeline for daily sales data"
)
```

Here, you've told Airflow:

- "Create a pipeline named *daily_sales_pipeline*."
- "Start running from Jan 1 2025."
- "Run every day."
- "Don't backfill missed runs."

✅ **Tip:** Every DAG lives in a .py file inside the dags/ folder. Airflow automatically scans and registers it.

🧱 2. Tasks – The Work Units

Each **Task** represents a single step in your workflow — extract data, transform data, send an alert, etc.

Tasks are defined using **Operators** (coming next) and linked together to form the DAG.

Example:

```
from airflow.operators.bash import BashOperator

extract_data = BashOperator(
    task_id="extract_data",
    bash_command="python3 scripts/extract.py",
    dag=dag
)

transform_data = BashOperator(
    task_id="transform_data",
    bash_command="python3 scripts/transform.py",
    dag=dag
)

load_data = BashOperator(
    task_id="load_data",
    bash_command="python3 scripts/load.py",
    dag=dag
)
```

Now, define the order of execution:

```
extract_data >> transform_data >> load_data
```

This line literally means *"run transform after extract, then load."*

⚙️ 3. Operators – The Building Blocks

Operators define **what** kind of task to run.

Airflow provides many operator types out of the box:

#	Operator	Purpose	Example
1	PythonOperator	Run a Python function	Data transformations
2	BashOperator	Run a shell command or script	Cron replacement
3	EmailOperator	Send email notifications	Alerts
4	SimpleHttpOperator	Call APIs	Fetch or post data
5	PostgresOperator	Run SQL statements	Load data to DB Sensor
6	Sensor (e.g., S3Sensor)	Wait for a file or event	Upstream dependency

You can also create **custom operators** if you need special logic — e.g., SlackAlertOperator.

Example using a PythonOperator:

```
from airflow.operators.python import PythonOperator

def greet(name):
    print(f"Hello, {name}!")

greet_task = PythonOperator(
    task_id="greet_user",
    python_callable=greet,
    op_args=["Rohit"],
    dag=dag
)
```

🔗 4. Dependencies and Execution Order

Airflow lets you define dependencies using three main patterns:

```
# 1. Bitshift operators
task_a >> task_b      # task_a before task_b
task_a << task_b      # task_b before task_a

# 2. Lists for parallelism
task_a >> [task_b, task_c]

# 3. Explicit method
task_b.set_upstream(task_a)
task_c.set_downstream(task_b)
```

Visualize it:

```
task_a
  ├── task_b
  └── task_c
```

This makes it easy to model complex, branching workflows.

🧠 5. Putting It All Together

Let's build a mini DAG that greets users and reports success:

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

def greet():
    print("Hello from Airflow!")

def report():
    print("Workflow finished successfully!")

with DAG(
    dag_id="greet_pipeline",
    start_date=datetime(2025, 1, 1),
    schedule="@daily",
    catchup=False,
) as dag:
    greet_task = PythonOperator(task_id="greet_task", python_callable=greet)
    report_task = PythonOperator(task_id="report_task", python_callable=report)

    greet_task >> report_task
```

- Two tasks defined using PythonOperator
- Clear dependency (greet_task → report_task)
- Readable and extendable

💬 Summary

- **DAG** = overall workflow
- **Task** = unit of work inside a DAG
- **Operator** = defines how that task runs

Airflow turns this Python definition into a visual, schedulable workflow you can monitor in the web UI.

Once you grasp these three pieces, everything else — sensors, hooks, MWAA deployment — builds on top of them.

➡️ Next Chapter Preview

Chapter 3 — Setting Up Apache Airflow

You'll set up Airflow on your own machine, run the web UI.

Appendix C – Resource Links & Community References

1. **Apache Airflow Official Documentation**

   https://airflow.apache.org/docs/

2. **Airflow GitHub Repository**

   https://github.com/apache/airflow

3. **Airflow Slack Community**

   https://apache-airflow-slack.herokuapp.com/

4. **MWAA Documentation (AWS)**

   https://docs.aws.amazon.com/mwaa/latest/userguide/

5. **Astronomer Documentation**

   https://www.astronomer.io/docs/

6. **Google Cloud Composer**

   https://cloud.google.com/composer/docs

7. **Airflow Forum & Discussions**

   https://forum.astronomer.io/

8. **Helpful Tutorials**

    - Airflow ETL Tutorials: https://airflow.apache.org/docs/apache-airflow/stable/tutorial.html
    - DAG Design Patterns: https://medium.com/@marceloprates/design-patterns-for-airflow-dags-610c8f186b84

Quick Start Guide on Apache Airflow

Master the art of data orchestration with "Quick Start Guide on Apache Airflow." This comprehensive guide walks you through the essentials of Apache Airflow, from core concepts and setting up your first DAG to real-world applications and deployment strategies, including best practices for scaling and optimization. Unlock the power of automated workflows, streamline complex data processes, and ensure reliability in your production pipelines.