Xebia

PYTHON PROGRAMMING WITH APACHE AIRFLOW

Eric Greene
eric@training4programmers.com

Goals for this Session

- What is Apache Airflow?
- Apache Airflow and Python
- Running Apache Airflow
- Essential Components of Apache Airflow

Xebia

What is Apache Airflow?

- **Workflow Orchestration Platform**: Apache Airflow is an open-source tool used to programmatically author, schedule, and monitor workflows as Directed Acyclic Graphs (DAGs).
- **Python-Based**: Workflows are defined in Python code, enabling developers to use standard programming constructs like loops, conditionals, and imports.
- **Scalable & Extensible**: Supports dynamic pipeline generation, integrations with many services (e.g., AWS, GCP, Docker), and custom plugin development.
- **Visual Monitoring Interface**: Comes with a rich web UI for visualizing DAG execution, tracking job status, and debugging pipeline issues.

Apache Airflow and Python

- **Python-Defined Workflows**: Airflow uses Python scripts to define DAGs, making it intuitive for developers to create complex workflows using familiar syntax.
- **Full Access to Python Ecosystem**: You can leverage Python libraries (e.g., pandas, requests, boto3) within tasks for data processing, API calls, and cloud service integration.
- **Custom Operators and Hooks**: Easily extend Airflow by writing custom Python classes to interact with external systems or encapsulate reusable logic.
- **Dynamic DAG Generation**: Python allows dynamic construction of DAGs and tasks based on variables, configurations, or external inputs, supporting flexible workflow design.

Running Apache Airflow

- **Multiple Deployment Options**: Airflow can run in Standalone Mode for quick testing (using SQLite and the Local Executor) or in Regular Mode for production (using PostgreSQL and the Local Executor or other executors).
- **Standalone for Simplicity**: The airflow standalone command launches all core components with minimal setup, ideal for learning and local development.
- **K8s and Cloud Deployments**: For production, Airflow can be deployed on Kubernetes or cloud platforms (e.g., AWS, GCP) using managed services like Amazon MWAA or Google Cloud Composer.
- **Airflow CLI for Control**: The powerful airflow command-line interface allows you to manage DAGs, trigger runs, monitor tasks, and interact with the environment programmatically.

Apache Airflow Processes

- **Webserver**: Serves the new stateless React-based front end, allowing users to visualize DAGs, inspect code, and monitor tasks by querying the metadata database (all UI code runs in the browser; server only delivers JSON/API responses).
- **Scheduler**: Continuously reads serialized DAGs from the metadata store, determines when DAG runs or task instances are ready, then enqueues them via the configured executor; supports multiple instances for high availability.

Apache Airflow Processes

- **DAG Processor**: Independently parses Python DAG files (via the airflow dag-processor process), extracts DAG definitions, and writes serialized versions into the database—isolating parsing from scheduling and improving security and scalability.
- **Triggerer**: An optional asyncio-based daemon that handles deferrable operators—listening for event-driven triggers (e.g., sensors, external conditions) and notifying the scheduler when a trigger completes so the main task can resume.

Essential Components of Apache Airflow

- **DAGs and Tasks**: DAGs (Directed Acyclic Graphs) define the workflow structure, while tasks represent individual units of work within that structure.
- **Operators and Providers**: Operators are predefined task templates (e.g., BashOperator, PythonOperator), and providers supply integrations with external systems like AWS, GCP, or Databases.
- **Variables, Connections, and Pools**: Variables store runtime configuration, Connections manage credentials to external services, and Pools limit resource usage by controlling parallel task execution.
- **XComs for Task Communication**: XComs (cross-communications) enable tasks to share data with each other, supporting more dynamic and data-driven workflows.

Python with Apache Airflow Demo

Let's Explore Python with Apache Airflow!

Python Programming with Apache Airflow Next Steps

- Review the demonstratons
- Review the Apache Airflow documentation
- Create a new Python project and incorporate Apache Airflow
- Learn more about workflows and their applications

Download the Code

github.com/cc-xebia-webinars/apache-airflow_06162025

slides and source code available

Q&A

Questions?

Thank you!

Eric Greene

eric@training4programmers.com

Xebia