Contents

[Apache Airflow Overview 1](#_Toc164382207)

[Advantages of using Data Pipelines as DAG in Apache Airflow 1](#_Toc164382208)

[Apache Airflow UI 1](#_Toc164382209)

[Reading: A note on Airflow Lab Exercises 1](#_Toc164382210)

[Hands-On – Getting started with Apache Airflow 1](#_Toc164382211)

[Build DAG Using Airflow 1](#_Toc164382212)

[Hands-on Lab: Create a DAG for Apache Airflow 1](#_Toc164382213)

[Hands-on Lab: Create and execute a Shell script from Airflow 1](#_Toc164382214)

[Airflow Monitoring and Logging 1](#_Toc164382215)

[Apache Airflow Documentation and Additional Content 1](#_Toc164382216)

**WEEK 3: USING APACHE AIRFLOW TO BUILD DATA PIPELINES**

# Apache Airflow Overview

1. **What is Apache Airflow?**

Great open source workflow orchestration tool.

A platform that lets you build and run workflows such as batch data pipelines.

With airflow, the workflow is represented as a DAG (Directed Acyclic Graph) that contains individual pieces of work called tasks arranged with dependencies.

Airflow is not a data streaming solution.

1. **Airflow Architecture:**

**A diagram of a database

Description automatically generated**

* **Scheduler:** Airflow comes with a builtin scheduler which handles the triggering of all scheduled workflows. It is responsible for submitting individual tasks from each scheduled workflow to the Executor.
* **Executor:** It handles the running of these tasks by assigning them to the workers.
* **Workers:** After executors assign work to the workers, the workers runs the tasks.
* **Web Server:** It servers the airflows interactive User Interface.
* **User Interface:** From this interface you can inspect, debug and trigger any of your DAGs and their individual tasks.
* **DAG Directory:** It contains all of your DAG files ready to be accessed by the Scheduler and Executor and each of its employed workers.
* **Metadata DB:** 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.

1. **Sample DAG illustrating the labeling of different branches:**

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

A diagram of a error

Description automatically generated

1. **Lifecycle of Apache airflow task state:**

A diagram of a work flow

Description automatically generated

1. **Apache Airflow Features:**

* Pure Python = Flexibility
* Useful UI = Full Insight
* Integration = Plug and Play
* Easy to Use = Unlimited Pipeline Scope
* Open Source = Community of Developers

1. **Apache Airflow Principles:** Airflow pipelines are built on 4 main principles:

* **Scalable**: Airflow has a modular architecture and uses a message queue to orchestrate an arbitrary number of workers. Ready to scale to infinity.
* **Dynamic**: Airflow pipelines are defined in python that 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.
* **Lean**: Airflow pipelines are Lean and explicit. Parameterization is built into its core using the powerful JINJA templating engine.



1. **Apache Airflow Use cases:**

* SIFT used airflow in defining and organizing ML pipeline dependencies.
* SENIORLINK used airflow in Increasing visibility of batch processes and decoupling them.
* EXPERITY used airflow for deploying as an enterprise scheduling tool.
* Onefootball used airflow for orchestrating SQL transformations in data warehouses.

# Advantages of using Data Pipelines as DAG in Apache Airflow

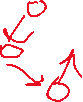
1. **What is a DAG?**



* Directed Acyclic Graph



* Graph – Nodes and Edges



* Circles -> nodes



* Directed graph – each edge has a direction
* Acyclic graph -> No loops (cycles)



Examples of DAG:

A diagram of a network

Description automatically generated

All trees are DAGs but all DAGs are not trees.

DAGs have only one root node.

1. **What is a DAG in Airflow?**

DAGs represent workflow: A blue circle with black arrows

Description automatically generated

* Nodes -> Tasks
* Edges -> Dependencies (defines the order in which two tasks should run
* Defined as code (Python)
* Scheduling instructions are specified in code in DAG script.

1. **Tasks and Operators:**

* Tasks are written in Python.
* Tasks implement operators, for example, Python, SQL, or Bash operators.
* Operators determine what each task does.
* Sensor operators poll for a certain time or condition.
* Other operators include email and HTTP request operators.

1. **DAG Definition components:**

Python script blocks:

|  |  |
| --- | --- |
| **Library imports** |  |
| **DAG arguments** |  |
| **DAG definition** |  |
| **Task definitions** |  |
| **Task pipeline** |  |

1. **Airflow Scheduler:**

Airflow Scheduler is designed to run as a persistent service within an Airflow production environment.

* It 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 according to the schedule interval you specified.

Advantages:

* Its approach to representing data pipelines as DAGs is the fact that they are expressed as code.
* **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 like GIT.
* **Collaborative**: Teams of developers can easily collaborate on both development and maintenance of the code for the entire workflow.
* **Testable**: Any revisions can be passed through unit tests to ensure the code still works as intended.

# Apache Airflow UI

1. **DAGs view:**

A screenshot of a computer

Description automatically generated

Table containing about the DAG details contained in your environment.

Each row -> interactive info about a DAG in your env like:

* Toggle on/off
* DAGs name
* Run schedule -> in the above case, in crontab format
* DAGs owner
* Recent task run status
* Previous tasks run status
* Collection of quick links to drill down into more info related to the DAG.

Visualizing DAG: in **Graph** and **Tree** mode.

* By clicking DAG name.

A screenshot of a computer

Description automatically generated

View context metadata for a task: task instance context menu

View the code for your DAG

View task duration timelines

1. **Sfj**
2. **szj**

# Reading: A note on Airflow Lab Exercises

<https://www.coursera.org/professional-certificates/ibm-data-engineer>

# Hands-On – Getting started with Apache Airflow

**LAB-1 Starting Apache Airflow:**

* start\_airflow [when airflow starts successfully, you should see UI URL, uname, pwd]

A screen shot of a computer

Description automatically generated

* Copy and open the webserver UI URL in a new tab.
* You can unpause /pause a DAG using unpause/ pause toggle button.

A screenshot of a computer

Description automatically generated

* To list all DAGs: **airflow dags list**
* To list all tasks in DAG named tutorial: **airflow tasks list tutorial**
* To unpause a DAG: **airflow dags unpause tutorial**
* To pause a DAG: **airflow dags pause tutorial**

# Build DAG Using Airflow

It is a python script file that contains:

* Python library imports
* DAG arguments
* DAG definition
* Task definitions
* Task pipeline

Example: Lets create the following DAG that prints greetings, then prints date and time and schedule it to run for every 5 seconds.

A diagram of a diagram

Description automatically generated

simple\_example\_DAG.py

**Solution:**

from airflow import DAG

from airflow.operators.bash\_operator import BashOperator

import datetime as dt

default\_args = {

‘owner’ : ’me’,

‘start\_date’ : dt.datetime(2021, 7, 28),

‘retries’ : 1,

‘retry\_delay’ : dt.timedelta(minutes=5),

}

dag = DAG(‘simple\_example\_DAG’,

description=’A simple example DAG’,

default\_args=default\_args,

schedule\_interval=dt.timedelta(seconds=5)

)

task1 = BashOperator(

task\_id =’print\_hello’,

bash\_command=’echo \Greetings. The date and time are \’ ’,

dag=dag,

)

task2 = BashOperator(

task\_id=’print\_date’,

bash\_command=’date’,

dag=dag

)

task1 >> task2

**Practice Exercise:**

# import the libraries

from datetime import timedelta

# The DAG object; we'll need this to instantiate a DAG

from airflow import DAG

# Operators; we need this to write tasks!

from airflow.operators.bash\_operator import BashOperator

# This makes scheduling easy

from airflow.utils.dates import days\_ago

#defining DAG arguments

# You can override them on a per-task basis during operator initialization

default\_args = {

'owner': 'Ramesh Sannareddy',

'start\_date': days\_ago(0),

'email': ['ramesh@somemail.com'],

'email\_on\_failure': False,

'email\_on\_retry': False,

'retries': 1,

'retry\_delay': timedelta(minutes=5),

}

# defining the DAG

dag = DAG(

'dummy\_dag',

default\_args=default\_args,

description='My first DAG',

schedule\_interval=timedelta(minutes=1),

)

# define the tasks

# define the first task

task1 = BashOperator(

task\_id='task1',

bash\_command='sleep 1',

dag=dag,

)

# define the second task

task2 = BashOperator(

task\_id='task2',

bash\_command='sleep 2',

dag=dag,

)

# define the third task

task3 = BashOperator(

task\_id='task3',

bash\_command='sleep 3',

dag=dag,

)

# task pipeline

task1 >> task2 >> task3

# Hands-on Lab: Create a DAG for Apache Airflow

**LAB 2: Create a DAG for Apache Airflow:**

* start\_airflow
* **Anatomy of airflow DAG:**

**# import the libraries**

**from datetime import timedelta**

# The DAG object; we'll need this to instantiate a DAG

**from airflow import DAG**

# Operators; we need this to write tasks!

**from airflow.operators.bash\_operator import BashOperator**

# This makes scheduling easy

**from airflow.utils.dates import days\_ago**

**#DAG arguments block:**

#defining DAG arguments

# You can override them on a per-task basis during operator initialization

**default\_args = {**

**'owner': John Adams,**

**'start\_date': days\_ago(0),**

**'email': [john.a@somemail.com'],**

**'email\_on\_failure': True,**

**'email\_on\_retry': True,**

**'retries': 1,**

**'retry\_delay': timedelta(minutes=5),**

**}**

* Owner name
* When this DAG should run from: days\_age(0)🡪today
* Email address where alerts are sent
* Whether alert must be sent on failure
* Whether alert must be sent on retry
* The number of retries in case of failure
* The time delay between retries

**#DAG Definition:**

# define the DAG

**dag = DAG(**

**dag\_id='sample-etl-dag',**

**default\_args=default\_args,**

**description='Sample ETL DAG using Bash',**

**schedule\_interval=timedelta(days=1),**

**)**

Here we are creating a variable named dag to instantiate DAG class with the following params:

* sample-etl-dag: ID of the DAG. This is what you see on the web console.
* default-args: It is a dict in which all the defaults are defined.
* description: Helps us in understanding what this DAG does.
* schedule\_interval: Tells us how drequently this DAG runs. Days = 1 🡪 everyday

**#Task Definitions:**

# define the tasks

# define the first task named extract

**extract = BashOperator(**

**task\_id='extract',**

**bash\_command='echo "extract"',**

**dag=dag,**

**)**

# define the second task named transform

**transform = BashOperator(**

**task\_id='transform',**

**bash\_command='echo "transform"',**

**dag=dag,**

**)**

# define the third task named load

**load = BashOperator(**

**task\_id='load',**

**bash\_command='echo "load"',**

**dag=dag,**

**)**

Task is defined using:

* task\_id: It is a string that helps in identifying the task.
* What bash command it represents.
* Which dag this task belongs to.

**#Task Pipeline:**

**extract >> transform >> load**

Here, the task **extract** must run first, followed by **transform,** followed by the task **load.**

**Submit a DAG:**

* Submitting a DAG is as simple as copying the DAG python file into **dags** folder in the **AIRFLOW\_HOME** directory.
* Airflow searches for Python source files within the specified **DAGS\_FOLDER**. The location of **DAGS\_FOLDER** can be located in the **airflow.cfg** file, where it has been configured as **/home/project/airflow/dags.**
* Airlow will load the python source files from the designated location. It will process each file, execute its contents, and subsequently load any DAG objects present in the file.
* Therefore, while submitting a **DAG**, it is essential to position it within this directory structure. Alternaatively, **AIRFLOW\_HOME** dir, representing the structure **/home/project/airflow**, can also be utilized for DAG submission.
  + cp my\_first\_dag.py $AIRFLOW\_HOME/dags
  + airflow dags list
  + airflow dags list|grep “my\_-first-dag”
  + You should see 2 tasks in the output.

A computer screen shot of a computer program

Description automatically generated

# Hands-on Lab: Create and execute a Shell script from Airflow

# Airflow Monitoring and Logging

1. **Logging:**

Required for developers to monitor the status of tasks in DAG runs, and to diagnose and debug issues.

By default, Airflow logs are saved to local file systems as log files. This makes it convenient for developers to quickly review the log files, especially in a development environment.

For airflow production deployments, the log files can be sent to cloud storage such as IBM cloud, AWS or Azure for remote accessing.

The log files can also be sent to search engines and dashboards for further retrieval and analysis.

Airflow recommends using Elasticsearch and Splunk, which are two popular document database and search engines, to index, search and analyze log files.

By default, log files are organized by DAG IDs and Task IDs and you can find a specific log file for a task execution using the following path convention:

**logs/dag\_id/task\_id/execution\_date/try\_number.log**

For example, to find the log of the first execution try of task 1 in a dummy DAG at a specific time. You can go to the folder **logs/dummy\_dag/task1/execution\_date/1.log**

If you click the first log file, you can check a lot of useful info (running cmds, cmd results, task results etc etc).

You can quickly review the task events via UI provided by Airflow webserver.

Airflow produces three different types of metrics for checking and monitoring components health:

* **Counters:** They are metrics that will always be increasing, such as total counts of successful or failed tasks.
* **Gauges:** They are metrics that may fluctuate. Eg: no.of currently running tasks/ DAG bag sizes.
* **Timers:** They are metrics related to time duration. Eg: Time to finish a task, time for a task to reach a success or failed state.

Similar to logs, metrics produced in airflow production deployments should be sent and analyzed by dedicated repositories and tools.

Airflow recommends using **StatsD**, which is a N/W tool that can gather metrics from airflow and send them to a dedicated metrics monitoring system.

For metrics monitoring and analysis, Airflow recommends using **Prometheus** which is a popular metrics monitoring ana analysis system. Prometheus can also aggregate and visualize metrics in a dashboard for more interactive visual analytics.

# Apache Airflow Documentation and Additional Content

Refer: <https://airflow.apache.org/docs/apache-airflow/stable/index.html>