**What is Apache Airflow?**

* Apache Airflow is a **workflow** **engine** that will **easily** **schedule** and **run** your **complex** **data** **pipelines**.
* It will make sure that **each** **task** of your **data** **pipeline** will get **executed** in the **correct** **order** and **each** **task** gets the **required** **resources**.

**Features of Apache Airflow:**

1. **Easy to Use:** If you have a bit of python knowledge, you are good to go and deploy on Airflow.
2. **Open Source:** It is free and open-source with a lot of active users.
3. **Robust Integrations:** It will give you ready to use operators so that you can work with Google Cloud Platform, Amazon Web Services, Microsoft Azure, etc.
4. **Use Standard Python to code:** You can use python to create simple to complex workflows with complete flexibility.
5. **Amazing User Interface:** You can monitor and manage your workflows. It will allow you to check the status of completed and ongoing tasks.

**Components of Apache Airflow:**

* **DAG:** It is the Directed Acyclic Graph – a **collection** of **all** the **tasks** that you want to **run** which is **organized** and **shows** the **relationship** between **different** **tasks**. It is defined in a python script.
* **Web Server:** It allows us to **monitor** the **status** of **DAGs** and **trigger** them. It is **user** **interface** built on **Flask**.
* **Metadata Database:** Airflow **stores** the **status** of **all** **tasks** in a **database** and do all **read/write** **operations** of a **workflow** from here.
* **Scheduler:** As the name suggests, this **component** is **responsible** for **scheduling** the **execution** of **DAGs**. It **retrieves** and **updates** the **status** of the **task** in the **database**.

**User Interface:**

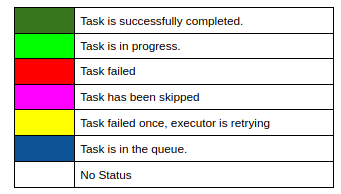
**1) DAG VIEW:**

1. It is the **default** **view** of the **user** **interface**.
2. This will **list** **down** **all** the **DAGS** **present** in **your** **system**.
3. It will **give** **you** a **summarized** **view** of the **DAGS** like **how** **many** **times** a particular DAG was **run** **successfully**, how many times it **failed**, the **last** **execution** **time**, and some other useful links.



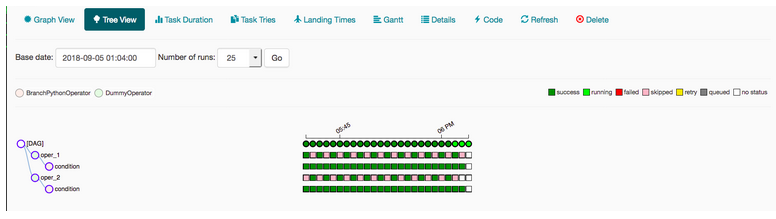
**2) GRAPH VIEW:**

1. In **graph** view, you can **visualize each** and **every** **step** of **your** **workflow** with their **dependencies** and their **current** **status**.
2. You can **check** the **current** **status** with **different** **color** **codes**.



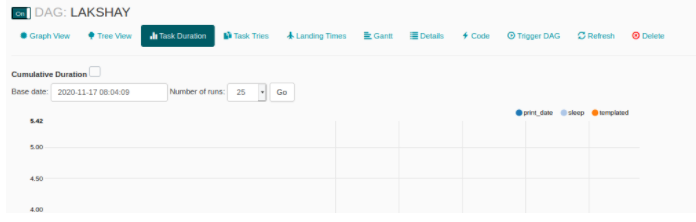
**3) TREE VIEW:**

1. The **tree** **view** also represents the **DAG**.
2. If you think your **pipeline** **took** a **longer** **time** to **execute** than **expected** then you can **check** which **part** is **taking** a **long** **time** to **execute** and **then** **you** **can** **work** on it.



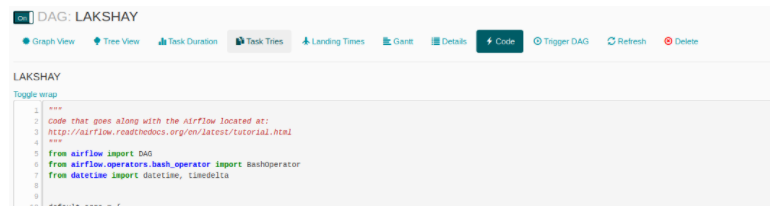
**4) TASK DURATION:**

1. In this view, you can **compare** the **duration** of your **tasks** **run** at **different** **time** **intervals**.
2. You can **optimize** your **codes** and **compare** your **performance** here.



**5) CODE:**

1. In this view, you can **quickly** **view** the **code** that was **used** to **generate** the **DAG**.



**Start Airflow:**

export AIRFLOW\_HOME=~/airflow

airflow webserver

airflow scheduler

**1) Python Operator in Apache Airflow:**

An **operator** describes a **single** **task** of the **workflow** and **Operators** provide us **different** **operators** for many **different** **tasks** for example **BashOperator**, **PythonOperator**, **EmailOperator**, **MySqlOperator** etc.

**a) Importing the Libraries:**

Let’s start with importing the libraries that we need. We will use the **PythonOperator** this time.

**# importing the required libraries**

from datetime import timedelta, datetime

from airflow import DAG

from airflow.operators.python\_operator import PythonOperator

from airflow.utils.dates import days\_ago

1. **Defining DAG Arguments:**

For each of the DAG, we need to **pass** **one** **argument** **dictionary**. Here is the description of some of the arguments that you can pass:

* **owner:** The **name** of the **owner** of the **workflow**, should be **alphanumeric** and can have **underscores** but should **not** contain any **spaces**.
* **depends\_on\_past:** If each time you run your workflow, the data depends upon the **past** **run** then **mark** it as **True** otherwise **mark** it as **False**.
* **start\_date:** Start date of your workflow
* **email:** Your email ID, so that you can receive an email whenever any task fails due to any reason.
* **retry\_delay:** If any task fails, then how much time it should wait to retry it.

# These args will get passed on to the python operator

default\_args = {

'owner': 'saif',

'depends\_on\_past': False,

'start\_date': datetime(2021, 9, 1),

'email': ['saifshk85@gmail.com'],

'email\_on\_failure': False,

'email\_on\_retry': False,

'retries': 1,

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

# 'queue': 'bash\_queue',

# 'pool': 'backfill',

# 'priority\_weight': 10,

# 'end\_date': datetime(2016, 1, 1),

# 'wait\_for\_downstream': False,

# 'dag': dag,

# 'sla': timedelta(hours=2),

# 'execution\_timeout': timedelta(seconds=300),

# 'on\_failure\_callback': some\_function,

# 'on\_success\_callback': some\_other\_function,

# 'on\_retry\_callback': another\_function,

# 'sla\_miss\_callback': yet\_another\_function,

# 'trigger\_rule': 'all\_success'

}

1. **Defining the Python Function:**

Now, we will define the python function that will **print** a **string** using an **argument** and this **function** will **later** be **used** by the **PythonOperator**.

**# define the python function:**

def my\_function1(x):

return x + " is a must have tool for Data Engineers."

**def my\_function2(x):**

return x + " is learning Airflow"

**d) Defining DAG:**

* Now, we will create a **DAG** **object** and **pass** the **dag\_id** which is the **name** of the **DAG** and make sure you have **not** created **any** **DAG** with **this** **name** **before**.
* **Pass** the **arguments** that we **defined** **earlier** and add a **description** and **schedule\_interval** which will **run** the **DAG** after the **specified** **interval** of **time.**

**# define the DAG**

dag = DAG(

'1\_PythonOperator',

default\_args=default\_args,

description='How to use Python Operator?',

schedule\_interval=timedelta(days=1)

)

1. **Defining the Task:**

* We have **two** **task** for our **workflow**:

1. **callf1**: In this task, we will print “Apache Airflow is a must-have tool for Data Engineers” on the terminal using the python function.
2. **callf2:** In this task, we will print “Saif is learning Airflow” on the terminal using the python function.

* We will pass the **task\_id** to the **PythonOperator** object.  You will **see** this **name** on the **nodes** of **Graph** **View** of your **DAG**.
* Pass the **python** **function** **name** to the argument **“python\_callable”** that you want to **run** and the **arguments** that your **function** is **using** to the **parameter** **“op\_kwargs”** as a **dictionary** and finally, the **DAG** **object** to which you **want** to **link** this **task**.

**# define the first task**

callf1 = PythonOperator(

task\_id='callf1',

python\_callable = my\_function1,

op\_kwargs = {"x" : "Apache Airflow"},

dag=dag,

)

callf2 = PythonOperator(

task\_id='callf2',

python\_callable = my\_function2,

op\_kwargs = {"x" : "Saif"},

dag=dag,

)

1. **Dependency or Execution Flow:**

Finally, create the pipeline by adding the **“>>”** operator between the tasks.

callf1 >> callf2

1. **Run the DAG:**

Now, when you refresh your Airflow dashboard, you will see your new DAG in the list.

**2) BashOperator:**

**Hands-On**

**3) Spark-Submit:**

**Hands-On**