Data Engineering 101 - Getting Started with Apache Airflow

BEGINNER DATA ENGINEERING PYTHON

Overview

- Understanding the need for Apache Airflow and its components
- We will create our first DAG to get live cricket scores using Apache Airflow
-

Introduction

Automation of work plays a key role in any industry and it is one of the quickest ways to reach functional efficiency. But many of us fail to understand how to automate some tasks and end in the loop of manually doing the same things again and again.

Most of us have to deal with different workflows like collecting data from multiple databases, preprocessing it, upload it, and report it. Consequently, it would be great if our daily tasks just automatically trigger on defined time, and all the processes get executed in order. Apache Airflow is one such tool that can be very helpful for you. Whether you are Data Scientist, Data Engineer, or Software Engineer you will definitely find this tool useful.

In this article, we will discuss Apache Airflow, how to install it and we will create a sample workflow and code it in Python.

Table of Contents

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.

It will provide you an amazing user interface to monitor and fix any issues that may arise.

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 AWS, 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.

Installation Steps

Let's start with the installation of the Apache Airflow. Now, if already have pip installed in your system, you can skip the first command. To install pip run the following command in the terminal.

```
sudo apt-get install python3-pip
```

Next airflow needs a home on your local system. By default ***~/airflow*** is the default location but you can change it as per your requirement.

```
export AIRFLOW_HOME=~/airflow
```

Now, install the apache airflow using the pip with the following command.

```
pip3 install apache-airflow
```

Airflow requires a database backend to run your workflows and to maintain them. Now, to initialize the database run the following command.

```
airflow initdb
```

We have already discussed that airflow has an amazing user interface. To start the webserver run the following command in the terminal. The default port is 8080 and if you are using that port for something else then you can change it.

```
airflow webserver -p 8080
```

Now, start the airflow schedular using the following command in a different terminal. It will run all the time and monitor all your workflows and triggers them as you have assigned.

```
airflow scheduler
```

Now, create a folder name dags in the airflow directory where you will define your workflows or DAGs and open the web browser and go open: http://localhost:8080/admin/ and you will see something like this:

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 is the user interface built on the Flask. It allows us to monitor the status of the DAGs and trigger them.
- **Metadata Database**: Airflow stores the status of all the 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

Now that you have installed the Airflow, let's have a quick overview of some of the components of the user interface.

DAGS VIEW

It is the default view of the user interface. This will list down all the DAGS present in your system. 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.

GRAPH VIEW

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

TREE VIEW

The tree view also represents the DAG. 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.

TASK DURATION

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

CODE

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

Define your first DAG

Let’s start and define our first DAG.

In this section, we will create a workflow in which the first step will be to print “Getting Live Cricket Scores” on the terminal, and then using an API, we will print the live scores on the terminal. Let’s test the API first and for that, you need to install the ***cricket-cli*** library using the following command.

```
sudo pip3 install cricket-cli
```

Now, run the following command and get the scores.

```
cricket scores
```

It might take a few seconds of time, based on your internet connection, and will return you the output something like this:

Importing the Libraries

Now, we will create the same workflow using Apache Airflow. The code will be completely in python to define a DAG. Let’s start with importing the libraries that we need. We will use only the BashOperator only as our workflow requires the Bash operations to run only.

```
from datetime import timedelta

# The DAG object; we'll need this to instantiate a DAG
from airflow import DAG
# Operators; we need this to operate!
from airflow.operators.bash_operator import BashOperator
from airflow.utils.dates import days_ago
```

view raw

airflow_import.py hosted with ❤ by **GitHub**

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.

```
# You can override them on a per-task basis during operator initialization
default_args = {
    'owner': 'lakshay',
    'depends_on_past': False,
    'start_date': days_ago(2),
    'email': ['airflow@example.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'
}
```

view raw

args_airflow.py hosted with ❤ by **GitHub**

Defining DAG

Now, we will create a DAG object and pass the **dag_id** which is the name of the DAG and it should be unique. Pass the arguments that we defined in the last step and add a description and **schedule_interval** which will run the DAG after the specified interval of time

```
# define the DAG
dag = DAG(
    'live_cricket_scores',
    default_args=default_args,
    description='First example to get Live Cricket Scores',
    schedule_interval=timedelta(days=1),
)
```

view raw

dag_airflow.py hosted with ❤ by **GitHub**

Defining the Tasks

We will have 2 tasks for our workflow:

1. **print**: In the first task, we will print the “Getting Live Cricket Scores!!!” on the terminal using the echo command.
2. **get_cricket_scores**: In the second task, we will print the live cricket scores using the library that we have installed.

Now, while defining the task first we need to choose the right operator for the task. Here both the commands are terminal-based so we will use the BashOperator.

We will pass the **task_id** which is a unique identifier of the task and you will see this name on the nodes of Graph View of your DAG. Pass the bash command that you want to run and finally the DAG object to which you want to link this task.

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

```
# define the first task
t1 = BashOperator(
    task_id='print',
    bash_command='echo Getting Live Cricket Scores!!!',
    dag=dag,
)


# define the second task
t2 = BashOperator(
    task_id='get_cricket_scores',
    bash_command='cricket scores',
    dag=dag,
)

# task pipeline
t1 >> t2
```

view raw

define_task.py hosted with ❤ by **GitHub**

Update the DAGS in Web UI

Now, refresh the user interface and you will see your DAG in the list. Turn on the toggle on the left of each of the DAG and then trigger the DAG.

Click on the DAG and open the graph view and you will see something like this. Each of the steps in the workflow will be in a separate box and its border will turn dark green once it is completed successfully.

Click on the node “get_cricket scores” to get more details about this step. You will see something like this.

Now, click on View Log to see the output of your code.

That’s it. You have successfully created your first DAG in the Apache Airflow.

End Notes

In this article, we have seen the features of Apache Airflow, its user interface components and we have created a simple DAG. In the upcoming article, we will discuss some more concepts like variables, branching, and will create a more complex workflow.

I recommend you go through the following data engineering resources to enhance your knowledge-

- *Getting Started with Apache Hive – A Must Know Tool For all Big Data and Data Engineering Professionals*
- *Introduction to the Hadoop Ecosystem for Big Data and Data Engineering*
- *Types of Tables in Apache Hive – A Quick Overview*

If you have any questions related to this article do let me know in the comments section below.

Article Url - https://www.analyticsvidhya.com/blog/2020/11/getting-started-with-apache-airflow/

LAKSHAY ARORA

Ideas have always excited me. The fact that we could dream of something and bring it to reality fascinates me. Computer Science provides me a window to do exactly that. I love programming and use it to solve problems and a beginner in the field of Data Science.