**Apache Airflow**

**Introduction:-** Apache Airflow is an open source workflow management platform. Workflow Management is the coordination of tasks that make up the work an organization does. By 'workflow' we mean a sequence of tasks that are part of some larger task, and is sometimes synonymous with business process**.** Building on the popularity of [Python](https://en.wikipedia.org/wiki/Python_(programming_language)) as the de facto [programming language](https://en.wikipedia.org/wiki/Programming_language) for data, Airflow is written in Python and workflows are created via Python scripts. Airflow is designed under the principle of "configuration as code". While other "configuration as code" workflow platforms exist using markup languages like [XML](https://en.wikipedia.org/wiki/XML), using Python allows developers to import libraries and classes to help them create their workflows.

**Worklow management example**

Hiring a new employee is an example of work done in an organization that has a clear business goal: a new employee that has passed appropriate selection becomes a member of staff. To achieve this goal hiring managers, recruiters and HR staff work together on a number of tasks.

Hiring a job candidate requires a number of pieces of information, such as the candidate’s name and CV, and details of the job vacancy. Some tasks will gather additional information during the hiring process: who performed interviews and when, their individual evaluations, practicalities such as the candidate’s availability date, and the final decision to hire the candidate or not. This information must be captured, made available to the right people in time, and archived.

When an HR team or recruitment department is hiring multiple new employees, management will coordinate this work to achieve a number of goals. Management want to select high quality employees, at reasonable cost without avoidable delays, and to present a professional company image to all candidates.

When a candidate is eventually hired, at the end of the process, other IT systems may need to be updated. The vacancy on the company web site can be removed, and the new employee must be added to the staff database, possibly with records of the hiring process added to their personnel file.

This hiring process example includes most of the elements we need, in order to explain what workflow management is, in general.

**Features**

* Monitoring Cron jobs
* transferring data from one place to other.
* Automating your DevOps operations.
* Periodically fetching data from websites and update the database for your awesome price comparison system.
* Data processing for recommendation based systems.
* Machine Learning Pipelines.

What Is a **Cron Job**? **cron** is a Linux utility which schedules a command or script on your server to run automatically at a specified time and date. A **cron job** is the scheduled **task** itself. **Cron jobs** can be very useful to automate repetitive tasks.

**Overview**

Airflow uses directed acyclic graph (DAGs) to manage workflow orchestration. Tasks and dependencies are defined in Python and then Airflow manages the scheduling and execution. DAGs can be run either on a defined schedule (e.g. hourly or daily) or based on external event triggers (e.g. a file appearing in Hive). Previous DAG-based schedulers like Oozie and Azkaban tended to rely on multiple Configuration file and file system trees to create a DAG, whereas in Airflow, DAGs can often be written in one Python file.

**Managers Providers**

Two notable providers offer ancillary services around the core open source project. Astronomer has built a SaaS tool and Kubernetes -deployable Airflow stack that assists with monitoring, alerting, devops, and cluster management. Cloud Composer is a managed version of Airflow that runs on Google Cloud Platform (GCP) and integrates well with other GCP services.

**Installation and Setup**

Airflow is python based so, the best way to install it is via pip.

1. **pip install apache-airflow**

To verify whether it got installed, run the command: airflow version

You will need to install mysqlclient as well to incorporate MySQL in your workflows. It is optional though.

1. **pip install mysqlclient**

Before you start anything, create a folder and set it as **AIRFLOW\_HOME**. In my case it is airflow\_home. Once created you will call export command to set it in the path.

1. **export AIRFLOW\_HOME='pwd' airflow\_home**

Make sure you are a folder above of airflow\_home before running the export command. Within **airflow\_home** you will create another folder to keep DAGs. Call it dags

If you set **load\_examples=False** it will not load default examples on the Web interface.

Now you have to call airflow initdb within **airflow\_home** folder. Once it’s done it creates **airflow.cfgand unitests.cfg**

**airflow.db** is an SQLite file to store all configuration related to run workflows. **airflow.cfg** is to keep all initial settings to keep things running.

In this file, you can see **sql\_alchemy\_conn** parameter with the value **../airflow\_home/airflow.db**

You can use MySQL if you want. For now, just stick with basic settings. So far so good, now without wasting any time let’s start the web server.

1. **airflow webserver**

When starts it shows the screen like:

2020-02-05 22:36:24,943] {\_\_init\_\_.py:51} INFO - Using executor SequentialExecutor

/anaconda3/anaconda/lib/python3.6/site-packages/airflow/bin/cli.py:1595: DeprecationWarning: The celeryd\_concurrency option in [celery] has been renamed to worker\_concurrency - the old setting has been used, but please update your config.

default=conf.get('celery', 'worker\_concurrency')),

\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|\_\_( )\_\_\_\_\_\_\_\_\_ \_\_/\_\_ /\_\_\_\_\_\_\_\_ \_\_\_\_\_\_ /| |\_ /\_\_ \_\_\_/\_ /\_ \_\_ /\_ \_\_ \\_ | /| / /\_\_\_ \_\_\_

| / \_ / \_ \_\_/ \_

/ / /\_/ /\_ |/ |/ /\_/\_/

|\_/\_/ /\_/ /\_/ /\_/

\\_\_\_\_/\_\_\_\_/|\_\_/

v1.10.0

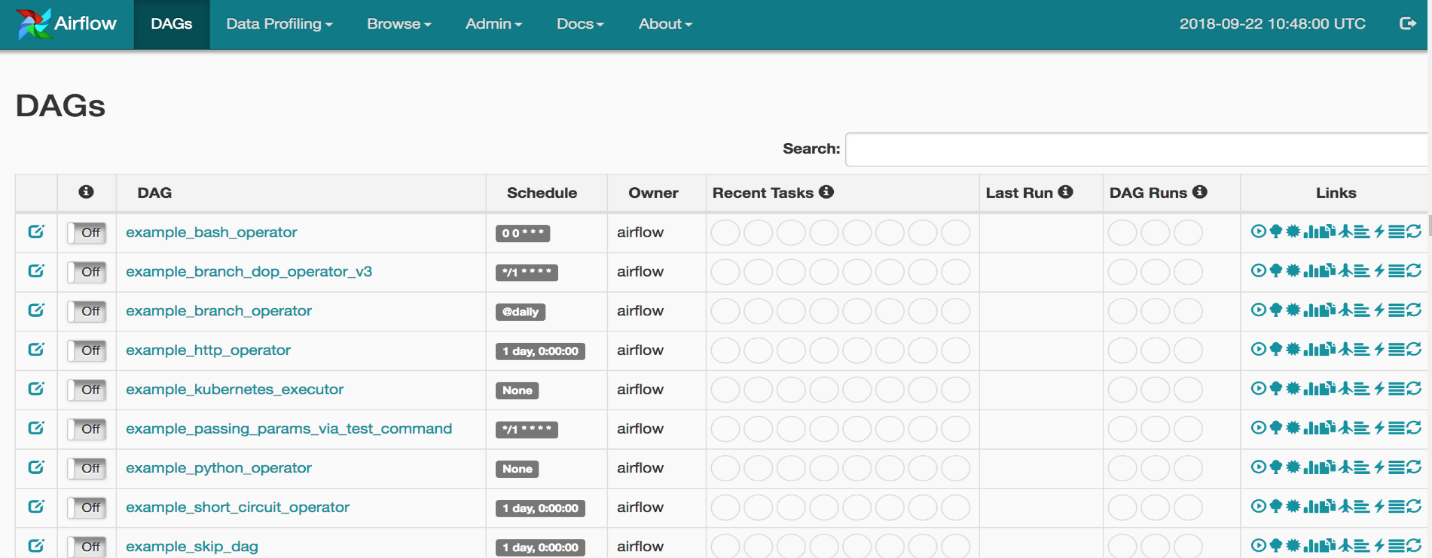
[2018-09-19 14:21:42,340] {\_\_init\_\_.py:57} INFO - Using executor SequentialExecutor

\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |\_\_( )\_\_\_\_\_\_\_\_\_ \_\_/\_\_ /\_\_\_\_\_\_\_\_ \_\_\_\_\_\_ /| |\_ /\_\_ \_\_\_/\_ /\_ \_\_ /\_ \_\_ \\_ | /| / /\_\_\_ \_\_\_ | / \_ / \_ \_\_/ \_ / / /\_/ /\_ |/ |/ /\_/\_/ |\_/\_/ /\_/ /\_/ /\_/ \\_\_\_\_/\_\_\_\_/|\_\_//

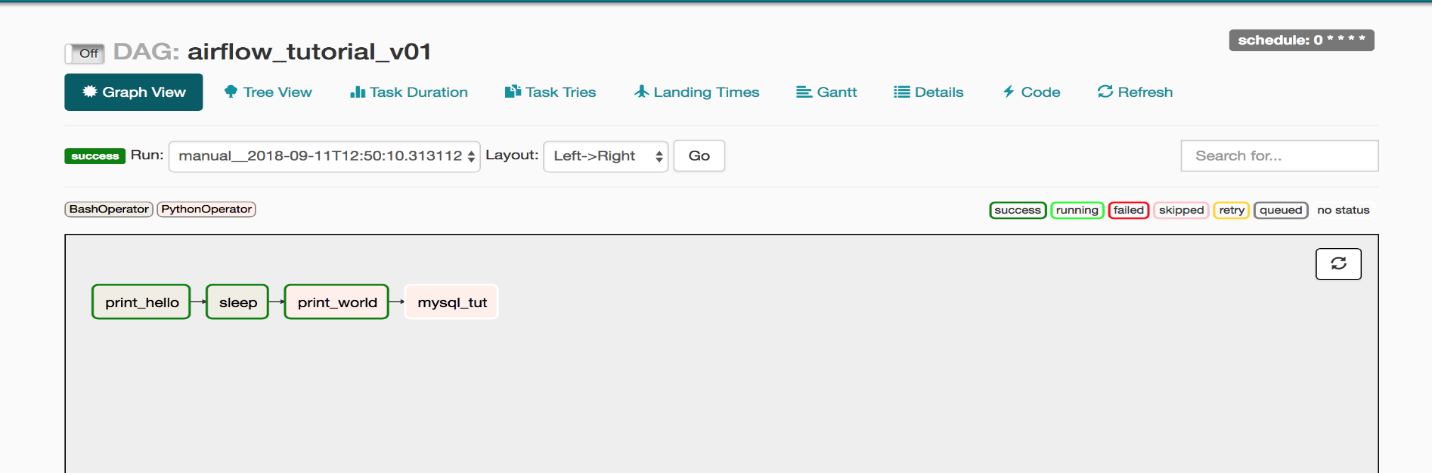
anaconda3/anaconda/lib/python3.6/site-packages/flask/exthook.py:71: ExtDeprecationWarning: Importing flask.ext.cache is deprecated, use flask\_cache instead..format(x=modname), ExtDeprecationWarning[2018-09-19 14:21:43,119] [48995]

{models.py:167} INFO - Filling up the DagBag from /Development/airflow\_home/dagsRunning the Gunicorn Server with:Workers: 4 syncHost: 0.0.0.0:8080

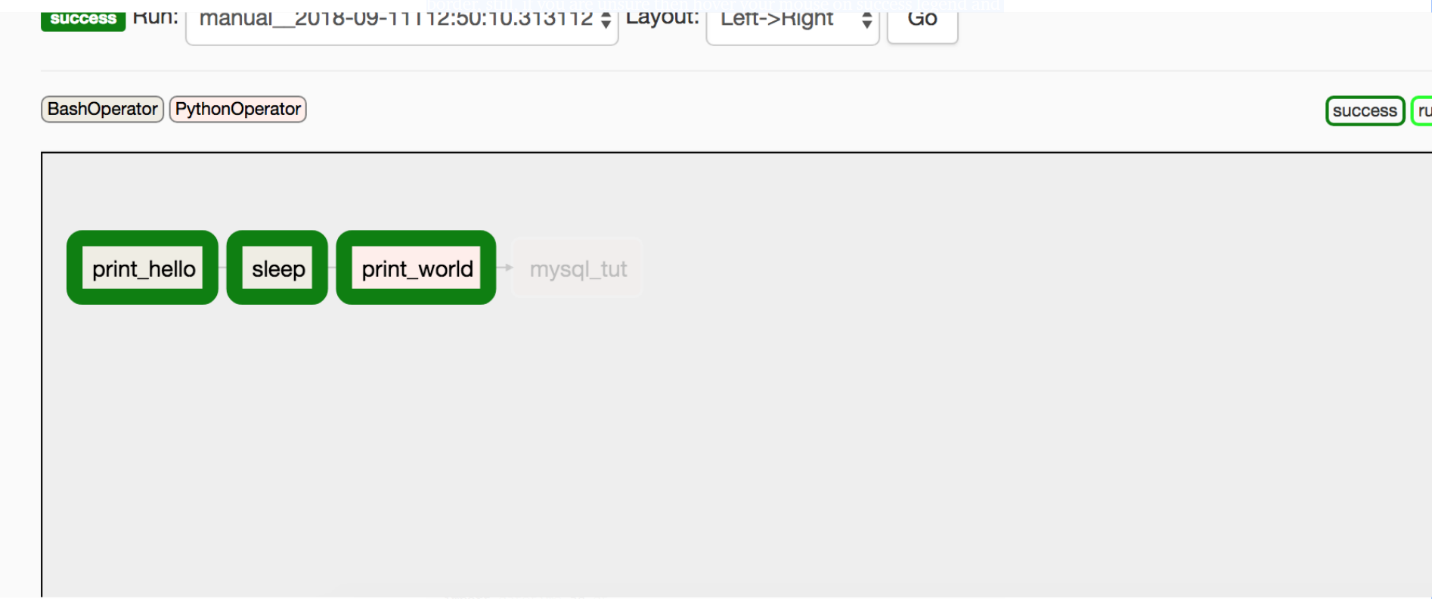
Now when you visit 0.0.0.0:8080 it shows a screen like:



You can see a bunch of entries here. These are the example shipped with the Airflow installation. You can turn them off by visiting **airflow.cfg** file and **set load\_examples** to FALSE**DAG Runs** tell how many times a certain DAG has been executed. **Recent Tasks**tells which task out of many tasks within a DAG currently running and what’s the status of it. The **Schedule** is similar to the one you would have used when scheduling a Cron, therefore, I won’t emphasize on it at the moment. The **Schedule** is responsible at what time this certain DAG should be triggered.



Here is the screenshot from a DAG I created earlier and executed. You can see rectangular boxes representing a task. You can also see different color boxes on the top right of the greyed box, named: **success**, **running**, **failed** etc. These are legends. In the picture above you can all boxes have a green border, still, if you are unsure then hover your mouse on success legend and you will see a screen like below:



You might have noticed the background/filling color of these boxes which is green and reed. On top-left of the greyed box, you can see why are they in such colors, these background color represents the different types of operators being used in this DAG. In this case, we are using **BashOperator** and **PythonOperator**.