How To - AirFlow CI/CD pipeline

Steps to write an Airflow DAG

A DAG file, which is basically just a Python script, is a configuration file specifying the DAG’s structure as code.

There are only 5 steps you need to remember to write an Airflow DAG or workflow:

- **Importing modules**
- **Default Arguments**
- **Instantiate a DAG**
- **Tasks**
- **Setting up Dependencies**

Step 1: Importing modules

Import Python dependencies needed for the workflow

```
###########################


from datetime import timedelta

import airflow
from airflow import DAG
from airflow.operators.bash_operator import BashOperator

###########################
```

We import three classes, DAG, BashOperator and PythonOperator that will define our basic setup.

Step 2: Default Arguments

Define default and DAG-specific arguments.If a dictionary of `default_args` is passed to a DAG, it will apply them to any of its operators. This makes it easy to apply a common parameter to many operators without having to type it many times.

```
########################################

default_args = {
    'owner': 'airflow',
    'start_date': airflow.utils.dates.days_ago(2),
    # 'end_date': datetime(2018, 12, 30),
    'depends_on_past': False,
    'email': ['airflow@example.com'],
    'email_on_failure': False,
    'email_on_retry': False,
    # If a task fails, retry it once after waiting
    # at least 5 minutes
    'retries': 1,
    'retry_delay': timedelta(minutes=5),
    }
########################################
```

This helps setting up default configuration that applies to the DAG.

Step 3 : Instantiate a DAG

Give the DAG name, configure the schedule, and set the DAG settings.

```
##########################################################
dag = DAG(
    dag_id='my_dag',
    default_args=default_args,
    description='A simple tutorial DAG',
    # Continue to run DAG once per day
    schedule_interval=timedelta(days=1),
)

#############################################################
```

Here is a couple of options you can use for schedule_interval.You can choose to use some preset argument or cron-like argument:

Preset	Meaning	Cron
None	Don’t schedule, use for exclusively “externally triggered” DAGs	
@once	Schedule once and only once	
@hourly	Run once an hour	0 * * * *
@daily	Run once a day	0 0 * * *
@weekly	Run once a week	0 0 * * 0

@monthly	Run once a month	0 0 1 * *
@yearly	Run once a year	0 0 1 1 *

Example:

Daily schedule:

- `schedule_interval='@daily'`
- `schedule_interval='0 0 * * *'`

Step 4: Tasks

The next step is to lay out all the tasks in the workflow.

```
# t1, t2 and t3 are examples of tasks created by instantiating operators
t1 = BashOperator(
    task_id='print_date',
    bash_command='date',
    dag=dag,
)

t2 = BashOperator(
    task_id='sleep',
    depends_on_past=False,
    bash_command='sleep 5',
    dag=dag,
)

templated_command = """
{% for i in range(5) %}
    echo "{{ ds }}"
    echo "{{ macros.ds_add(ds, 7)}}"
    echo "{{ params.my_param }}"
{% endfor %}
"""

t3 = BashOperator(
    task_id='templated',
    depends_on_past=False,
    bash_command=templated_command,
    params={'my_param': 'Parameter I passed in'},
    dag=dag,
)
```

Step 5: Setting up Dependencies

Set the dependencies or the order in which the tasks should be executed.

```
################################################
 This means that t2 will depend on t1
# running successfully to run.
t1.set_downstream(t2)

# similar to above where t3 will depend on t1
t3.set_upstream(t1)

##################################################

# The bit shift operator can also be
# used to chain operations:
t1 >> t2

# And the upstream dependency with the
# bit shift operator:
t2 << t1

####################################################

# A list of tasks can also be set as
# dependencies. These operations
# all have the same effect:
t1.set_downstream([t2, t3])
t1 >> [t2, t3]
[t2, t3] << t1

######################################################
```

Final DAG file

```
"""
Code that goes along with the Airflow tutorial located at:
https://github.com/apache/incubator-airflow/blob/master/airflow/example_
dags/tutorial.py
"""
from airflow import DAG
from airflow.operators.bash_operator import BashOperator
from datetime import datetime, timedelta


default_args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'start_date': datetime(2015, 6, 1),
    '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),
}

dag = DAG(
    'tutorial',
    default_args=default_args,
    schedule_interval=timedelta(days=1))

# t1, t2 and t3 are examples of tasks
# created by instantiating operators
t1 = BashOperator(
    task_id='print_date',
    bash_command='date',
    dag=dag)

t2 = BashOperator(
    task_id='sleep',
    bash_command='sleep 5',
    retries=3,
    dag=dag)

templated_command = """
    {% for i in range(5) %}
        echo "{{ ds }}"
        echo "{{ macros.ds_add(ds, 7)}}"
        echo "{{ params.my_param }}"
    {% endfor %}
"""

t3 = BashOperator(
    task_id='templated',
    bash_command=templated_command,
    params={'my_param': 'Parameter I passed in'},
    dag=dag)
```

```
t2.set_upstream(t1)
t3.set_upstream(t1)
```

The easiest way to work with Airflow once you define our DAG is to use the web server. Airflow internally uses a SQLite database to track active DAGs and their status. Use the following commands to start the web server and scheduler

```
> airflow webserver

> airflow scheduler
```