

Introduction To Airflow

Lab 8 - CSEN1095

Overview

What is Airflow?

- Airflow is a platform to programmatically author, schedule, and monitor workflows or data pipelines.



What is a Workflow?

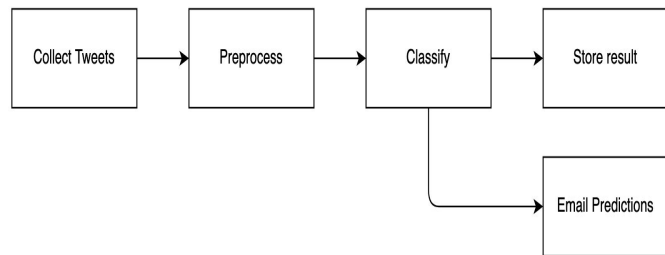
- a sequence of tasks
- started on a schedule or triggered by an event
- frequently used to handle big data processing pipelines

Why do we need Airflow?

- Data grows fast, gets more complex and harder to manage as your company scales.
- In order to provide insights, you need to have some kind of visualization to explain your findings and monitor them over time.
- For these data to be up to date, you need to extract, transform, load them into your preferred database from multiple data sources in a fixed time interval (hourly , daily, weekly, monthly).

Why do we need Airflow? - Example

- Imagine you have an ML model that does twitter sentiment analysis and you want to run that model for your favorite people on twitter for their tweets everyday.
- As you can see the data flows from one end of the pipeline to the other end.
- You can automate the process using Airflow

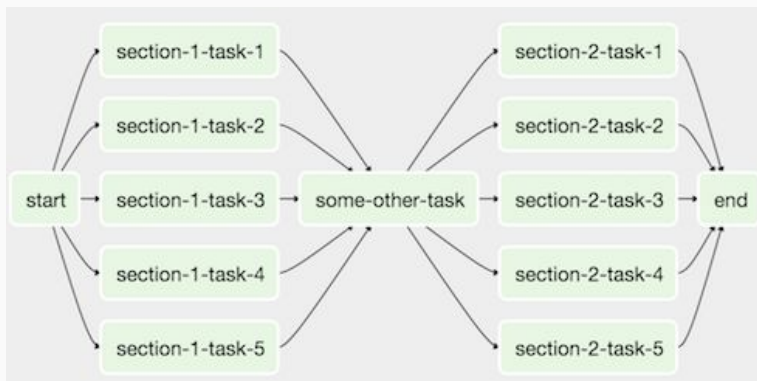


What makes Airflow great?

- Can handle upstream/downstream dependencies gracefully
- Easy to reprocess historical jobs by date, or re-run for specific intervals
- Handle errors and failures gracefully. Automatically retry when a task fails.
- Ease of deployment of workflow changes (continuous integration)
- Integrations with a lot of infrastructure (Hive, Presto, AWS, Google cloud, etc)

Airflow DAG

- Airflow provides DAG python class to create a Directed Acyclic Graph in order for you to define your workflow.
- Airflow DAGs are composed of Tasks.



Scheduling your DAGs

- There are two parameters that you can define when instantiating your DAG to specify when your DAG will be run:
 1. `start_date`
 2. `schedule_interval`: dictates how often to run the DAG
 - Can be defined using:
 - i. cron expressions
 - ii. cron presets
 - iii. `datetime.timedelta` object

Scheduling your DAGs - Cont.

preset	meaning	cron
<code>None</code>	Don't schedule, use for exclusively "externally triggered" DAGs	
<code>@once</code>	Schedule once and only once	
<code>@hourly</code>	Run once an hour at the beginning of the hour	<code>0 * * * *</code>
<code>@daily</code>	Run once a day at midnight	<code>0 0 * * *</code>
<code>@weekly</code>	Run once a week at midnight on Sunday morning	<code>0 0 * * 0</code>
<code>@monthly</code>	Run once a month at midnight of the first day of the month	<code>0 0 1 * *</code>
<code>@yearly</code>	Run once a year at midnight of January 1	<code>0 0 1 1 *</code>

Scheduling your DAGs - Cont.

The cron expression is made of five fields. Each field can have the following values.

*	*	*	*	*
minute (0-59)	hour (0 - 23)	day of the month (1 - 31)	month (1 - 12)	day of the week (0 - 6)

- You can play around with its syntax here: <https://crontab.guru/>

Operators, and Tasks

- **DAGs** do not perform any actual computation. Instead, **Operators** determine what actually gets done.
- **Task**: Once an operator is instantiated, it is referred to as a “task”. An operator describes a single task in a workflow.
- A **DAG** is a container that is used to organize tasks and set their execution context.

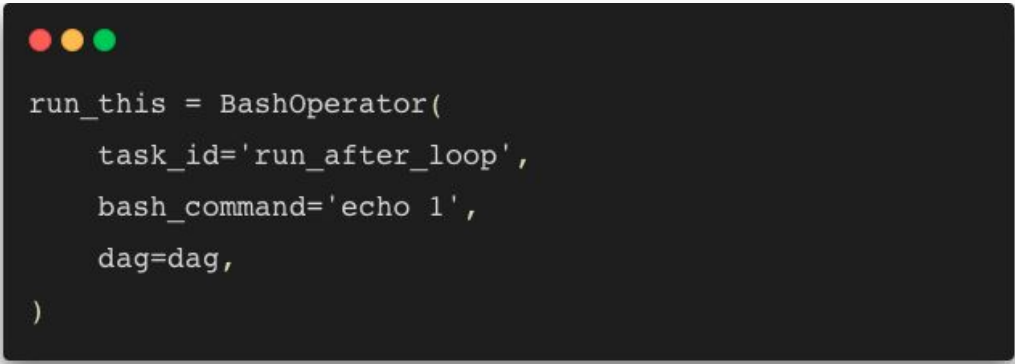
Types of Operators

- Operators define the nodes of the DAG. Each operator is an independent task.
- Examples:
 1. PythonOperator: when a PythonOperator is run, it will run the code of the python_callable function

```
def print_function():  
    print ("Hey I am a task")  
  
run_this_last = PythonOperator(  
    task_id='run_this_last',  
    dag=dag,  
    python_callable=print_function  
)
```

Types of Operators - Cont.

2. BashOperator: runs a bash command

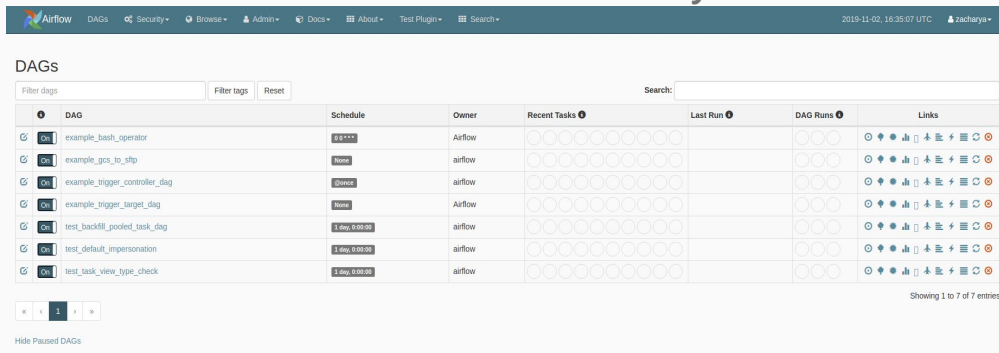
A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. It contains a Python code snippet for creating a BashOperator.

```
run_this = BashOperator(  
    task_id='run_after_loop',  
    bash_command='echo 1',  
    dag=dag,  
)
```

- There are tons of Operators that are open source that perform multiple tasks. You can even write your own operator.

UI

- Airflow also has a shiny UI that allows you to manage and monitor your workflows. Dependencies are built more easily, logs are easily accessible, code can be easily read, time spent on each task, time to finish, trigger/pause workflows with a click of a button and many more can be done with the UI.



The screenshot displays the Apache Airflow web interface. At the top, there's a navigation bar with links for DAGs, Security, Browse, Admin, Docs, About, Test Plugin, and Search. The date and time are shown as 2019-11-02, 16:35:07 UTC, and the user is zacharya. Below the navigation bar, the title 'DAGs' is followed by a search bar and buttons for 'Filter dags', 'Filter tags', and 'Reset'. The main content is a table listing DAGs with columns for DAG, Schedule, Owner, Recent Tasks, Last Run, DAG Runs, and Links. The table contains seven entries, each with a status icon (On or Off), a name, a schedule, an owner, a set of task status icons, a last run timestamp, a set of DAG run status icons, and a set of action links.

DAG	Schedule	Owner	Recent Tasks	Last Run	DAG Runs	Links
example_bash_operator	18***	Airflow				
example_gcs_to_sftp	None	airflow				
example_trigger_controller_dag	cron	airflow				
example_trigger_target_dag	None	Airflow				
test_backfill_pooled_task_dag	1 day, 0:00:00	airflow				
test_default_impersonation	1 day, 0:00:00	airflow				
test_task_view_type_check	1 day, 0:00:00	airflow				

Showing 1 to 7 of 7 entries

Hide Paused DAGs

Example

Covid DAG

- Define a DAG that receives covid statistics from an API and stores the data in a csv file.

Step 1 - Import modules

- Import Python dependencies needed for the workflow

```
# step 1 - import modules
import requests
import json

from airflow import DAG
from datetime import datetime
from datetime import date
# Operators; we need this to operate!
from airflow.operators.python_operator import PythonOperator

import pandas as pd
```

Step 2 - Default Arguments

- Define default and DAG-specific arguments

```
# step 2 - define default args
# These args will get passed on to each operator
# You can override them on a per-task basis during operator initialization
default_args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'start_date': datetime(2020, 12, 13)
}
```

Step 3 - Instantiate a DAG

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

```
# step 3 - instantiate DAG
dag = DAG(
    'covid-DAG',
    default_args=default_args,
    description='Fetch covid data from API',
    schedule_interval='@once',
)
```

Step 4 - Tasks

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

```
# step 4 Define tasks
def store_data(**context):
    df = context['task_instance'].xcom_pull(task_ids='extract_data')
    df = df.set_index("date_of_interest")
    df.to_csv("data/nycovid.csv")

def extract_data(**kwargs):
    url = "https://data.cityofnewyork.us/resource/rc75-m7u3.json"
    response = requests.get(url)
    df = pd.DataFrame(json.loads(response.content))
    return df

t1 = PythonOperator(
    task_id='extract_data',
    provide_context=True,
    python_callable=extract_data,
    dag=dag,
)

t2 = PythonOperator(
    task_id='store_data',
    provide_context=True,
    python_callable=store_data,
    dag=dag,
)
```

Step 5 - Setting up Dependencies

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

```
# step 5 - define dependencies  
t1 >> t2
```

Task

Task Description

- Implement an Airflow pipeline that does the following tasks **every 5 minutes**:
 1. Read a csv file:
<https://raw.githubusercontent.com/raneemsultan/DataEngineering-CSEN1095/main/Lab8/CSV/people.csv>
 2. Drop any null values
 3. Store the csv file with the timestamp concatenated to the name

References

- <https://www.applydatascience.com/airflow/airflow-tutorial-introduction/>
- <https://towardsdatascience.com/data-engineering-basics-of-apache-airflow-build-your-first-pipeline-eefecb7f1bb9>
- <https://bhavaniravi.com/blog/apache-airflow-introduction>
- <https://morioh.com/p/643f1be319e9>
- <https://bigishdata.com/2020/04/05/apache-airflow-part-1-introduction-setup-and-writing-data-to-files/>
- <https://github.com/jduran9987/airflow-covid>
- <https://airflow.apache.org/docs/apache-airflow/1.10.1/scheduler.html>