Apache Airflow
Automate recurring ETL Pipelines

Abstract *Apache Airflow is a workflow management tool which is widely used for data engineering pipelines. It was developed by Airbnb in 2014 and given to Apache in 2016. It solves the problem of running data pipelines on a regular basis with dependencies. Thus, it can be used to automate recurring pipelines for ETL jobs, as well as training of machine learning models. It works based on jobs described as directed acyclic graphs. It is mostly used to run recurring fixed jobs. For stream processing, other tools, like the ones described on the bottom, make more sense.*

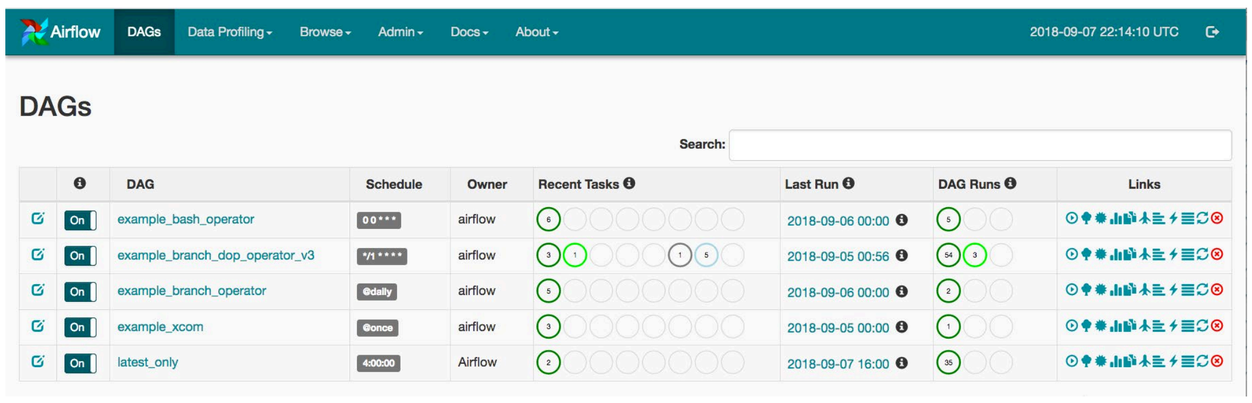
Apache Airflow [1]

How does it work?

Airflow is a workflow management service, that is based on DAGs (direct acyclic graphs). Each DAG describes how the workflow of a pipeline is run. By that, dependencies are defined and each DAG operation can only be executed when the dependencies are successfully completed. Operations define the exact command that is run in a DAG's node. It can be a bash command, a python script, database operations or setup and execution of commands within a Docker container.

Basic concepts

Besides the two basic concepts, which enable definition of complex tasks, Airflow supports data engineers and data scientists with more features in order to make handling complex tasks easy. It provides a web interface to visually describe DAG structure and their states. Error management is provided to retry failed executions.



Airflow GUI showing all DAGs with schedule information and details on last runs [1]

How to define Airflow jobs?

For regularly running DAGs, Airflow uses an included scheduler. For each DAG, execution time and frequency can be set like for cronjobs, but they run on Airflow's integrated scheduler logic, which makes it independent from system cronjobs.

Airflow is written in Python, and so are the DAG definitions. Defining a DAG can be done as shown below. Dependencies between tasks is done by defining it in the Python script as node1 >> node2.

When to use Airflow

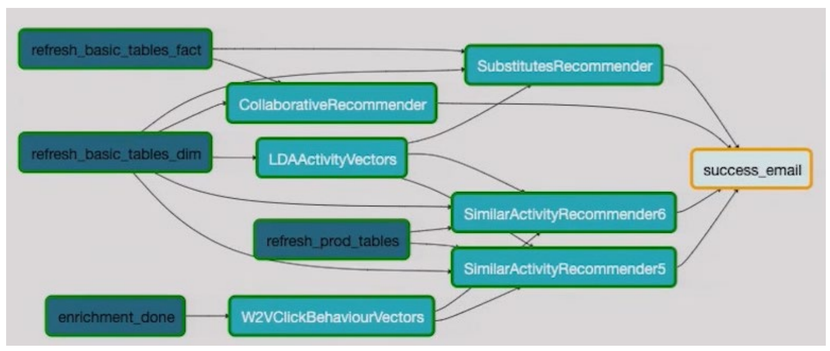
- *Need of flexible scheduling tool for recurring data engineering and data science use cases*
- *Increasing complexity of data pipelines*
- *Processing non-real time data on a regular basis*
- *Need for support of wide variety of usable operators to be called within airflow DAG nodes*

When not to use Airflow

- *Processing stream data or integration of real time updates of machine learning models and data pipelines*
- ***Reason:** Airflow works scheduler-based and is not made for working with continuous streams*

```
dag = DAG{
    'dag name',
    default_args= {
        'owner' = 'Hendrik',
        'depends_on_past' = False,
        'start_date' = datetime(2020, 1, 25),
        'email' = ['hendrik.patzlaff@student.hpi.de'],
        'email_on_failure' = True, # send email on failure
        'retries' = 2, # retry twice
        'retry_delay' = timedelta(minutes=5)
        },
        schedule_interval='0 0 * * *' # run every night midnight
    }
}
```

DAG definition Python script



DAG visualization [5]

Similar tools in comparison

As discussed above, Airflow is not made for handling data streams. As this use case is getting more and more relevant in the current times, some solutions for that will be presented below. An direct alternative to Airflow is described on the bottom left.

Advantage of stream processing over batch processing

As more and more data-driven decisions are taken, constantly analyzing data is crucial to instantly get information out of those. Differently from Airflow, there are frameworks with the focus on stream instead of batch processing.

Luigi [3] **Luigi** *(similar to Apache Airflow)*

- *Focus on **periodically running ETL jobs***
- *Disadvantage over Airflow: scheduler is based on cronjobs → not so scalable*

[2] **Apache Flink**

- *Distributed real time processing engine*
- *Support for batch processing and stream processing*
- *High level API*

APACHE STORM™ [4] **Apache Storm**

- *Real time stream processing application*
- *Used for real time analysis and processing*
- *Low level API compared to Flink*

Hendrik Patzlaff
B. Sc. Student IT-Systems Engineering

Lecture Series on Practical Data Engineering
Hasso Plattner Institute, Potsdam, Germany

hendrik.patzlaff@student.hpi.de

Sources:
[1] https://airflow.apache.org/
[2] https://flink.apache.org/
[3] https://github.com/spotify/luigi
[4] https://storm.apache.org/
Lecture Series on Practical Data Engineering, Hasso Plattner Institute 2019/20:
- [5] Recommending Tourist Activities - Data Science Challenges and the Needs for Data Pipelines – Maximilian Jenders
- Apache Flink - An Introduction and Outlook into the Future – Arvid Heise

HPI Hasso Plattner Institut