RESEARCH ARTICLE OPEN ACCESS

Creating Data Pipelines using Apache Airflow

Sameer Shukla
(Lead Software Engineer, USA
Email: sameer.shukla@gmail.com)

--*************************-----------------------------------

Abstract:

This Paper addresses the use of Apache Airflow in creating Data Pipelines, the paper gives an overview of what Apache Airflow is, basic building blocks like DAGs and Operators, explains how to create a simple pipeline using a realistic ETL use-case. Paper also briefly explains about Cloud Composer the fully managed service developed by Google Cloud Platform for Apache Airflow. Results from this study on Airflow suggests that using Apache Airflow can simplifies the Data Pipeline creation process as only pre-requisite to start using Airflow is the basic Python knowledge because the Operators in Airflow should be written in Python 3.6 or above.

Keywords — **Apache Airflow, Directed Acyclic Graphs, Python, Pandas, Cloud Composer, Google Cloud Platform, Google Cloud Storage**

--*************************-----------------------------------

I. INTRODUCTION

Workflow is the series of activities required for the completion of a task, Apache Airflow is a workflow management tool that helps in authoring, scheduling, computing, and monitoring workflows. The workflows in Airflow are represented as Directed Acyclic Graphs (DAGs), or DAG is a Data Pipeline. A DAG consists of multiple tasks and task in Airflow are created as Operators and the Operators are written using Python. Important features of Airflow are

Open Source: Airflow is an Open-Source, it's free to use
Python: Basic Python knowledge is sufficient for creating workflows
Platforms: Airflow can run on various platforms like Google Cloud, Azure, AWS, and the managed service like Cloud Composer on GCP helps in default integration with other services on GCP like Dataproc, Google Cloud Storage etc.
User Interface: Airflow has a very intuitive UI for monitoring and managing workflows, user can check the status of DAG and check logs, helps in stopping or starting workflows.

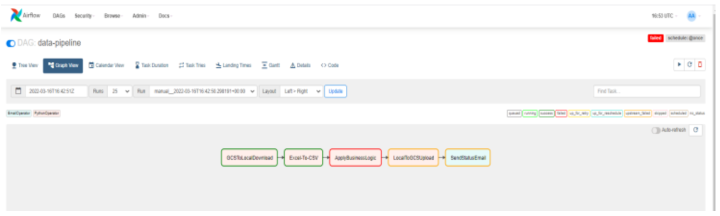


Fig. 1 Airflow User Interface

The remainder of the paper is structured as follows: The next section represents the Building Blocks of

Airflow, which is understanding in depth about DAG, the Operators and different types, Cloud Composer, understanding and visualizing use-case as DAGs and Operators

II. Directed Acyclic Graphs

DAG stands for Directed Acyclic Graph; DAG is a graph with Nodes and Edges and a valid DAG should not have any loops and it should always be directed. A Node in a DAG is an Operator which in turn is a task like "Downloading a File, Emailing, Slacking, uploading to GCS, Any DB Operator, Making an HTTP Request" etc. Example of Valid DAG is

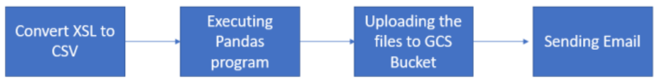


Fig. 2 Valid DAG

The Invalid DAG has a loop, meaning it will never complete the place where the loop is it always be repeating the steps and never get over.

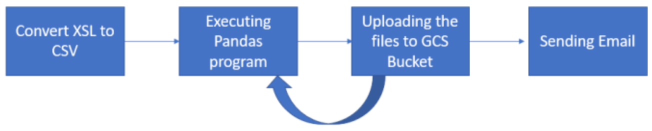
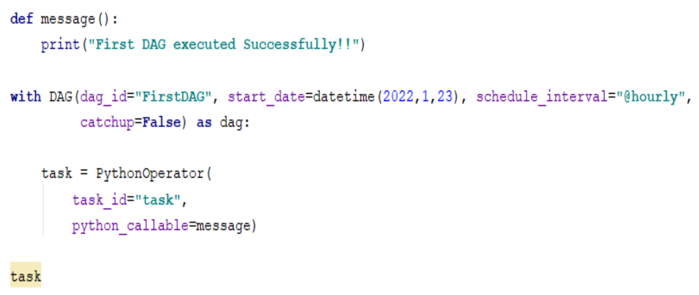


Fig. 3 Invalid DAG

Step 3 in the above DAG will keep invoking Step 2 hence it's not Directed

III. DAG Scheduling

The DAG can run at regular intervals, the property "schedule_interval" needs to be configured for running the DAGs as per the requirement. The value of 'schedule_interval' by default is 'None' meaning DAG cannot be scheduled and can be run only through Airflow UI.



```
def message():
    print("First DAG executed Successfully!!")

with DAG(dag_id="FirstDAG", start_date=datetime(2022,1,23), schedule_interval="@hourly",
         catchup=False) as dag:

    task = PythonOperator(
        task_id="task",
        python_callable=message)

task
```

Image. 1 Sample DAG

The DAGs can be configured using 'cron expressions' or through 'cron' presets like '@hourly', '@hour', '@daily', '@monthly', '@yearly' etc.

The DAG can be run at specific date and time and for that 'cron' expressions can be used

*/5 * * * * = Run every 5 minutes
0 14 * * * = every day at 14:00 PM
0 14 * 12 * = December 14:00 PM

IV. Operators

Operators are tasks in a DAG, each task is a node in the graph. Airflow has collection of lots of readymade operators which can be used.

Uploading a file from Local to GCS (Google Cloud Storage) is a task and "LocalFileSystemToGCSOperator" can be used.

Using Pandas program for file transformation is task and "PythonOperator" can be used.

Once the pipeline execution is completed, notifying the team about successful completion of the pipeline is a task and "EmailOperator" can be used.

Sr, No	**Operator**	**Description**
1	PythonOperator	Execution of Python Functions
2	SimpleHTTPOperator	For making HTTP

		Requests
3	EmailOperator	For sending emails
4	GCSToGCSOperator	Moving files from bucket to another.
5	LocalFileSystem ToGCSOperator	Uploading files from Local Filesystem to GCS
6	SlackAPIOperator	For Posting messages to Slack Channel
7	S3FileTransformOperator	For Copying data from S3 to Local Filesystem

Table. 1 Types of Operators

V. Cloud Composer

On Google Cloud Platform (GCP) a fully managed service is created for Airflow known as Cloud Composer. Running Airflow on Cloud Composer has several advantages since it's a service on GCP it has default integration with other GCP services like GCS, Cloud Dataflow, Dataproc and so on. The Airflow Pipeline on Composer can also be executed using Cloud Function as well because it has default integration. It's easy to integrate any python libraries on Composer it has a option of "PYPI" packages, the entire composer environment can be updated in few minutes with any python library required for processing. It's important to note that Composer is installed as managed Google Kubernetes Cluster engine so it's not free and there is no option of stop the process, once the processing is over, we need to fully delete the service.

VI. Use Case

ETL process can be converted to a fully automated Airflow Pipeline on the Cloud.

- Imagine a excel file is given to us for parsing and processing, the size of each file is greater than 1GB and each excel file has multiple sheets in it.
- The file arrives on the GCS bucket every day at some time, the pipeline execution should start at the same time every day.
- Since the excel file is heavy, first step would be to convert the single excel file to multiple small csv's where each sheet in excel is converted to one single csv, we should call it as Conversion Step
- Process all the CSV's and execute the business logic using Python library called Pandas, in this step filtering and transforming takes place as per the business logic, Transformation Step
- Once the Pandas processing is complete, the next step would be to upload the transformed file to another GCS bucket.
- Once the entire processing is over, send email notification to the team about the successful completion of the Pipeline.

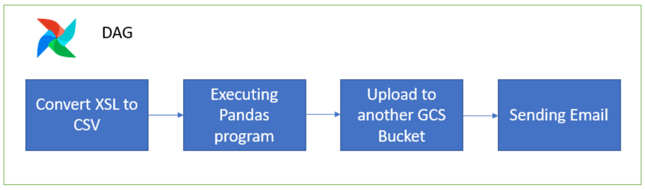


Fig. 4 Sample DAG, each step is a Task

For each task in the pipeline, following ready to use operators in Airflow should be used

- **PythonOperator**: Convert XSL to CSV
- **PythonOperator**: Executing Pandas for CSV transformation
- **GCSToGCSOperator**: Upload to another GCS Bucket
- **EmailOperator**: Sending Email

VII. CONCLUSIONS

The paper has covered automating the ETL pipeline using Airflow, but there are various other use cases on which Airflow can be used.

- Airflow can be used in automating Batch Jobs Scenario's.
- Airflow can be used for training the Machine Learning Models.
- It can also be used as a replacement of cron-jobs.
- Can be used in the Microservices architecture as well.

Apache Airflow is an easy-to-use tool, lot of help available on the Internet and community is growing very fast. Just like GCP has created fully managed service Composer for simplifying the development of Pipeline creation using Airflow, similarly other Platforms like AWS has created a service called Amazon Managed Workflows for Apache Airflow (MWAA).

VIII. ACKNOWLEDGMENT

I would like to express my deepest appreciation to all those who contributed developing the Apache Workflow Tool. Grateful for astronomer.io team for providing such valuable help and resources online which simplifies the pipeline creation on Apache Airflow.

IX. REFERENCES

[1] https://aws.amazon.com/managed-workflows-for-apache-airflow/
[2] https://www.astronomer.io/guides/intro-to-airflow/
[3] https://databand.ai/apache-airflow-monitoring-best-practices/
[4] https://airbnb.io/projects/airflow/
[5] https://docs.sentry.io/platforms/python/guides/airflow/