IJCSE International Journal of Computer Sciences and Engineering Open Access
Research Paper Vol. 10, Issue.8, August 2022 E-ISSN: 2347-2693

Developing Pragmatic Data Pipelines using Apache Airflow on Google Cloud Platform

Sameer Shukla

Lead Software Engineer, Texas, USA

Author's Mail Id: sameer.shukla@gmail.com, Tel.: +1-480-754-9793

DOI: https://doi.org/10.26438/ijcse/v10i8.18 | Available online at: www.ijcseonline.org

Received: 22/Jul/2022, Accepted: 06/Aug/2022, Published: 31/Aug/2022

Abstract— Data Pipeline[1][2] is a series of actions which moves data from the one source to the destination, the complexity of Data Pipeline varies from use-case to use-case. The traditional data pipeline cleanups the data, aggregates the data and move it from one place to another, it sounds simple but it's very complex as the organization deals with huge and complex data and the expectation from pipeline is that it should be robust, fast, notify about the status and it should do the same task repeatedly without failing. The modern data pipelines are slightly different in nature they are supposed to deal with Petabytes of data, they stores the data in various flavors of the cloud, should provide real-time data analysis. Apache Airflow is one such tool which simplifies the entire Data Pipeline creation to a great extent and the only pre-requisite is the basic Python Knowledge. This paper focuses on the stock-exchange data pipeline creation by using the Airflow concepts such as DAGs and Operators.

Keywords—Data-Pipeline, Python, Pandas, Seaborn, Apache-Airflow, GCP, Kaggle.

I. INTRODUCTION

Pipelines can be categorized into two, ETL Pipeline[4][5] and Data Pipelines[6][7]. ETL stands for Extract Transform and Load whereas Data Pipeline is generic which is supposed to move data from various systems to another and it may or may not transform the data in between, transformation may include filtering, aggregating, cleaning and data analysis while moving data from source to destination.

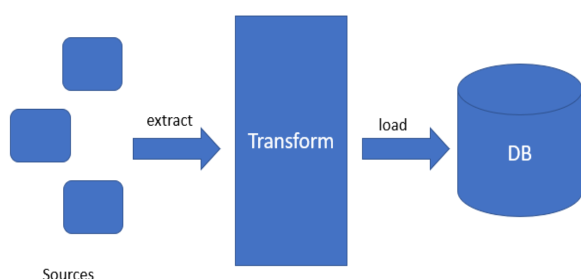

Figure 1: Description of ETL flow

There are various challenges in the ETL data flow [8] the data extraction process can be much slower depending on the data volume because if the volume is huge, it can impact the extraction process. As well as Orchestration and monitoring process can be complicated too because we need to monitor at various levels of extraction. Whereas the Data Pipelines are supposed to be Modern in nature, they should provide real-time data processing updates, they should be seamlessly deployable to any Cloud [9][10], and the architecture should be fault tolerant.

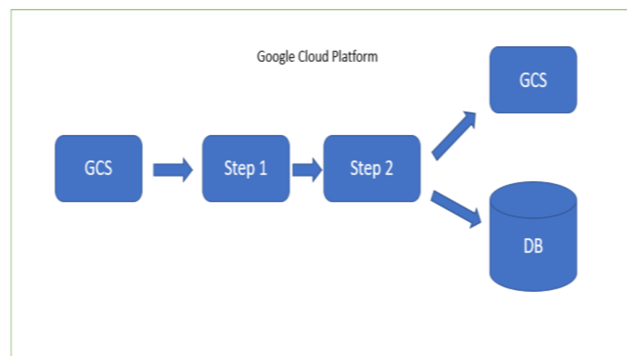

Figure 2: Description of Data Pipeline

Whether any step failed during the processing the expectation is the pipeline should provide real-time update, it should notify the users what's going on with-in the pipeline. Pipeline should process large volume of data and they should operate in self manage mode, by self-manage mode means they should trigger either automatically or on demand basis, in case of failures Pipeline should know how and when to cleanup, also every pipeline should be so modern that it has some managed services available for deployment on the cloud [10][11] for example: The managed service for Spark [12] on Google Cloud Platform is Dataproc similarly the tool we are using to create our pipelines should have some managed service on the cloud. To simplify the characteristics and the challenges we discussed in both Traditional ETL Pipeline and Modern Data Pipeline we can build our pipelines using Apache Airflow[13]

© 2022, IJCSE All Rights Reserved

II. RELATED WORK

The Mission in this paper is to create a efficient pipeline using Apache Airflow, we must understand what Airflow is and the core components of Airflow which helps in Pipeline Creation. Apache Airflow[13] is an open-source workflow management tool and it is based on Python, fundamentally the Pipeline in Airflow is represented as DAG and DAG stands for Directed Acyclic Graph.

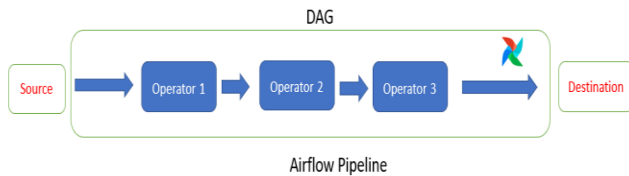

Figure 3. Airflow Pipeline Example

In the Figure above we can there is a Source and bunch of Operators which is running inside a DAG and a destination. In simple words Airflow Pipeline consists of DAG which in turn consists of Operators and in combination it formed a Modern Pipeline, as discussed in Introduction section about the characteristics of modern pipeline which consists of efficient monitoring, real-time updates about the pipeline etc, all these characteristics in a DAG will be coded as Airflow Operator, let's understand in detail about DAG and Operators.

A. Directed Acyclic Graph (DAG): DAG[14][15][16] is a kind of a grapg with nodes and edges, edges should always be directed in a DAG and node in a graph is a task, let's understand by an simple example, consider we are establishing a Pipeline for setting up dataproc on GCP using Airflow. Figure 4 represents a valid DAG where every edge is directed although it's sequential but it is correct, Step first is creating a DataProc cluster on GCP this is done via ready-

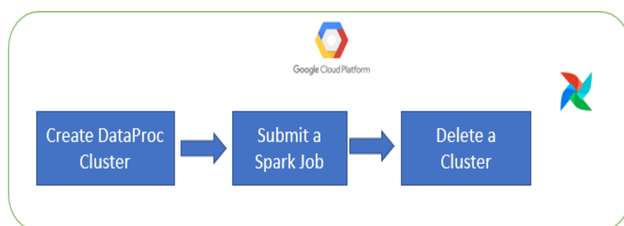

Figure 4: Pipeline Creating DataProc Cluster on GCP

made Operator called 'DataprocCreateClusterOperator', step 2 is submitting a Spark Job because Dataproc is a managed service on GCP which runs Spark Job, Submitting a Job another Node and node is Operator this step is taken care by 'DataprocSubmitJobOperator', step 3 is deleting a cluster which can be done via 'DataprocDeleteClusterOperator', so we can say that DAG in Airflow consists of Operator and it should not contain any loop, all the edges should always be directed.

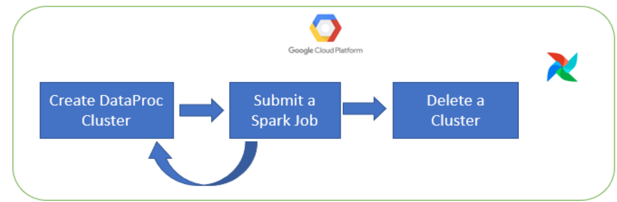

Figure 5: Example of Invalid DAG in Airflow

The Invalid DAG contains a Loop, in the above diagram after submitting a Job again a call will be made to create a Cluster, this DAG will never terminate and it's a very expensive DAG as it is stuck in a Loop and keep creating multiple Clusters on Google Cloud Platform.
Below program is the simple DAG written in Python Programming Language

```
def display():
    print("Example of DAG one")

with DAG(dag_id="dagOne",
start_date=datetime(2021,5,23),
schedule_interval="@hourly",
         catchup=False) as dag:

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

task
```

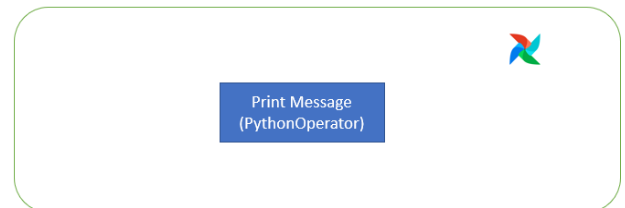

Figure 6: DAG with Single Operator

This the pictorial representation of the sample code of the DAG, it contains of only one Operator which is PythonOperator and PythonOperator is used to invoke Python Functions in a DAG.

B. Operators: Operators are tasks in Airflow, there are various ready to use Operators available in Airflow for various uses, as we have already seen four operators so far, for creating a Dataproc Cluster, Submitting a Spark Job, Deleting a Cluster and PythonOperator for executing Python functions. Airflow tasks can run in parallel as we have seen each task is a Operator but there are several Operators can run in parallel in Airflow. PythonOperator is a very powerful Operator exists in Airflow, as of now there are no ready to use operators available for Kafka Integration, say we have a csv file and we want to send each row of the csv file to Kafka as a part of Airflow Pipeline we can use PythonOperator, sample code may look like

```
def csvRowsToKafka(**context):

    filename = 'sample.csv'
    file = open(filename, 'r')
    file_reader = csv.DictReader(file)
```

© 2022, IJCSE All Rights Reserved

```
    for row in file_reader:
        """
        Send row data to Kafka
        """
    return


with
DAG(dag_id="kafkaDag",start_date=datetime(2021,5,23),
schedule_interval="@hourly", catchup=False) as dag:


  task = PythonOperator(
       task_id="csv_data_to_kafka",
       python_callable=csvRowsToKafka)
```

Airflow has a collection of lots of Operators which are ready to use for almost every cloud platform, GCP, AWS, Azure and much more.

C. Monitoring

Airflow UI is very intuitive and excellent for monitoring.

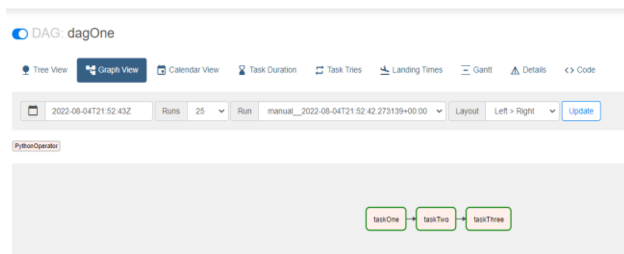

Figure 7: Airflow UI Running on Local

The above Image is a execution of below code

```
def taskOne():
    print("Task One Completed")


def taskTwo():
    print("Task Two Completed")


def taskThree():
    print("Task Three Completed")

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

    task_one = PythonOperator(
        task_id="taskOne",
        python_callable=taskOne)

    task_two = PythonOperator(
        task_id="taskTwo",
        python_callable=taskTwo)

    task_three = PythonOperator(
        task_id="taskThree",
        python_callable=taskThree)


task_two.set_upstream(task_one)

task_three.set_upstream(task_two)
```

It's a straightforward DAG with three Operators and all of them are PythonOperators which is simply executing a Python functions and functions in turn displaying messages on the console. The UI is very detailed, apart from tracking the real time progress we can also track the performance of each task, how much time each task has taken to execute

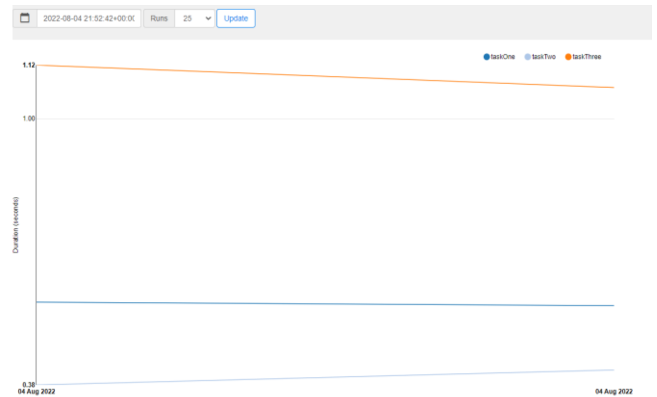

Figure 8: Performance of each task

In case if any task failed during execution, it highlights the failed task as well.

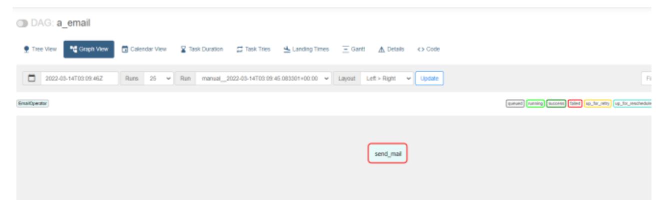

Figure 9: Failed Task

C. Running tasks in Parallel: In the above DAG all thtasks are running sequentially, Running tasks in parallel is straightforward in Airflow, all we need to do is to tweak the execution at the end and in turn it executes the tasks in Parallel. Consider a DAG with three tasks and let's airflow execute them in parallel.

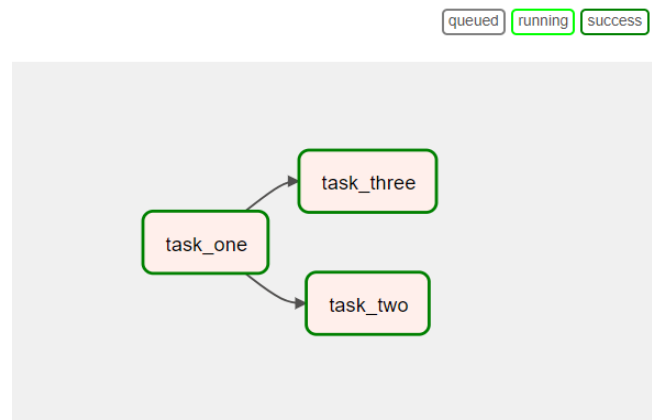

Figure 10: Task Execution in Parallel

We can see Airflow executed the task in parallel and the simple change is to put the tasks in brackets

```
def taskOne():
    print("Task One Completed")

def taskTwo():
    print("Task Two Completed")

def taskThree():
    print("Task Three Completed")

with DAG('dagOne',
```

© 2022, IJCSE All Rights Reserved

```
         start_date=datetime(2022, 1, 1),
         schedule_interval="@hourly") as dag:

    t1 = PythonOperator(
        task_id='task_one',
        python_callable=taskOne,
        email_on_failure=True,
        dag=dag)

    t2 = PythonOperator(
        task_id='task_two',
        python_callable=taskTwo,
        dag=dag)

    t3 = PythonOperator(
        task_id='task_three',
        python_callable=taskThree,
        dag=dag)

t1 >> [t2, t3]
```

t1 >> [t2, t3] will execute the tasks in Parallel. In this section we have learned the building blocks of Airflow.

III. MEHODOLOGY

In this paper we are going to build pipelines, we are going to learn by developing multiple pipelines, we have already seen one use case of building a GCP Dataproc Cluster using Airflow, the first use-case would be to building Stock Exchange Pipeline,

A. *Use-Case One*

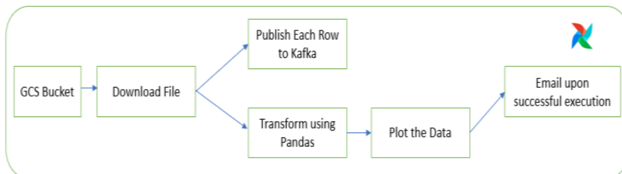

Figure 11: Stock Exchange Pipeline

In this use case we will explore the stock exchange dataset from Kaggle, imagine the file is uploaded on Google Cloud Storage the file will be downloaded first from GCS[20] then each row is parsed and published to Kafka as well as each row is transformed, here transformation means filtering, we will be filtering all the rows where value of Index is 'NYA' and the value of 'Open' is greater than 600 plus Date should be greater than 1st Jan 2020. Once the filtering is done, all the filtered rows will be written to a separate CSV and top 5 rows with highest value 'Open' will plotted and email is sent to the team stating the data pipeline execution is successful. Below is the snapshot of the data set from Kaggle.

```
Index,Date,Open,High,Low,Close,Adj Close,Volume
NYA,1965-12-31,528.690002,528.690002,528.690002,528.690002,528.690002,0
NSEI,1966-01-03,527.210022,527.210022,527.210022,527.210022,527.210022,0
HSI,1966-01-04,527.840027,527.840027,527.840027,527.840027,527.840027,0
NYA,1966-01-05,531.119995,531.119995,531.119995,531.119995,531.119995,0
NYA,1966-01-06,532.070007,532.070007,532.070007,532.070007,532.070007,0
NYA,1966-01-07,532.599976,532.599976,532.599976,532.599976,532.599976,0
```

Below Figure showcases the Stock Exchange Pipeline in execution, this pipeline is a combination of Parallel task execution and sequential task execution.

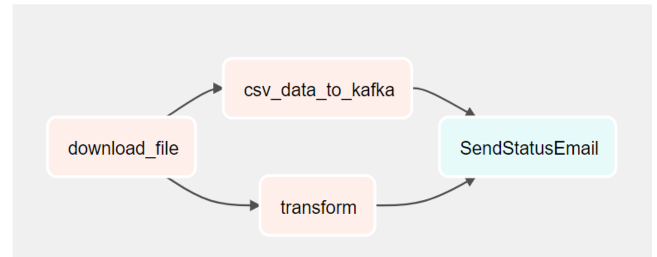

Figure 12: Stock Exchange Pipeline in Execution

Code below is the DAG, which consists of four tasks as discussed and show-cased in the Figure 12 above.

```
with DAG('data_pipeline',
         start_date=datetime(2022, 1, 1),
         schedule_interval="@hourly") as dag:

    t1 = PythonOperator(
        task_id='download_file',
        python_callable=download,
        email_on_failure=True,
        dag=dag)

    t2 = PythonOperator(
        task_id="csv_data_to_kafka",
        python_callable=csvRowsToKafka)

    t3 = PythonOperator(
        task_id='transform',
        python_callable=transformAndPlot,
        dag=dag)

    t4 = EmailOperator(
        task_id="SendStatusEmail",
        to='sameer.shukla@gmail.com',
        subject='Pipeline Status!',
        html_content='<p>Pipeline execution successful!
<p>',
        dag=dag)

chain(t1,  [t2, t3])
[t2,t3] >> t4
```

This DAG is slightly different than what we have seen so far, it is using the 'chain' function to chain the sequential and parallel task.

Let's check each function used in DAG

```
def csvRowsToKafka(**context):
    index_file = 'indexData.csv'
    file_name = open(index_file, 'r')
    file = csv.DictReader(file_name)
    for row in file:
        """
        Publish Rows To Kafka
        """
    return


def transformAndPlot():
    df = pd.read_csv('./indexData.csv',
skipinitialspace=True)

    nyadf = df[(df['Index'].str.replace(' ', '') ==
'NYA') & (df['Open'] > 600) & (df['Date'] > "2020-01-
01")]

    sortdf = nyadf.sort_values(by='Open',
ascending=False).iloc[0:5]

    sortdf['Open'].plot(kind="bar")
```

Result of transformAndPlot function discussed below

© 2022, IJCSE All Rights Reserved

	Index	Date	Open	High	Low	Close	Adj Close	Volume
13593	NYA	2020-01-02	13913.03027	14003.38965	13913.03027	14002.49023	14002.49023	3.458250e+09
13594	NYA	2020-01-03	13877.48047	13950.74023	13870.74023	13917.04981	13917.04981	3.461290e+09
13595	NYA	2020-01-06	13856.71973	13943.29981	13852.73047	13941.79981	13941.79981	3.674070e+09
13596	NYA	2020-01-07	13911.19043	13923.50977	13880.53027	13898.45020	13898.45020	3.420380e+09
13597	NYA	2020-01-08	13897.00977	13986.69043	13896.58984	13934.44043	13934.44043	3.720890e+09

Figure 13: Filtered Stock Exchange Pipeline in Execution

Plot

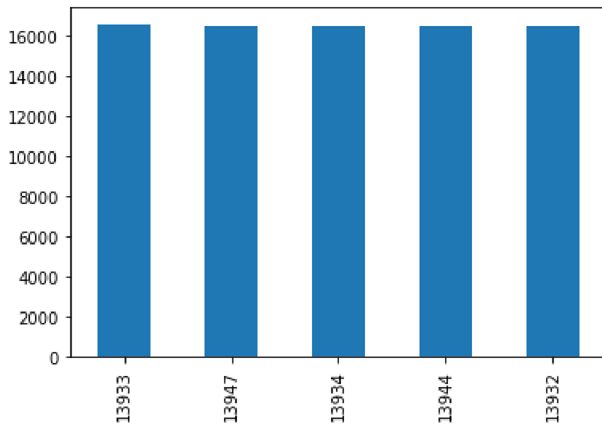

Figure 14: Top 5 Open Data

B. Use-Case Two

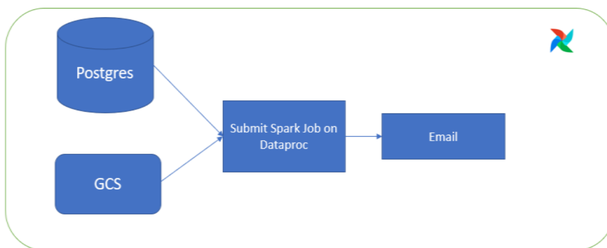

Figure 15: Sample ETL Pipeline

The above picture depicts the use-case two, which is very much like the Use-case one. The difference is, in this case data will be fetched from different data sources like Postgres DB and files from GCS Bucket, the entire data is submitted to Dataproc as Spark Job for further Data Processing and once done send an email to Users.

```
with DAG('spark_pipeline',
         start_date=datetime(2022, 1, 1),
         schedule_interval="@hourly") as dag:

    t1 = PostgresOperator(
        task_id="postgres_task",
        postgres_conn_id="postgres_default",
        sql="SELECT * FROM Table;",
        )

    t2 = PythonOperator(
        task_id="download_file_from_GCS",
        python_callable=download)

    t3 = DataprocSubmitJobOperator(
        task_id='submitSparkJob',
        python_callable=submitSparkJob,
        dag=dag)

    t4 = EmailOperator(
        task_id="SendEmail",
        to='sameer.shukla@gmail.com',
        subject='Status!',
        html_content='<p>Pipeline execution successful!
<p>',
        dag=dag)

chain(t1,  [t2, t3])
[t2,t3] >> t4
```

Above sample code represents DAG with various Operators required for the execution of the pipeline. PostgresOperator is a ready to use Operator for interacting with Postgres, DataprocSubmitJobOperator is an operator used for Submitting Spark Job again it's a ready to use operator in Apache Airflow. The image below shows the Pipeline in execution

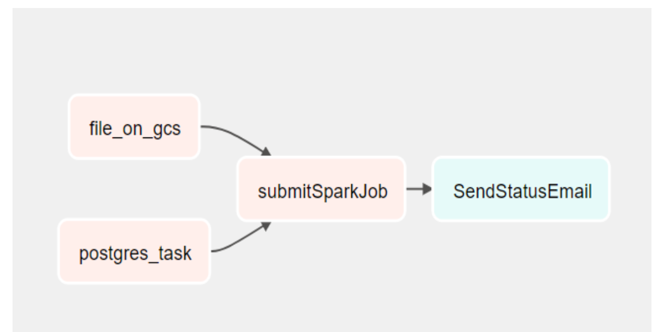

Figure 16: Spark Job Pipeline

C. Use-Case Three

This is the most interesting use-case, using Apache Airflow in Microservices environments. Imagine we have multiple REST Microservices running in the environments, and we want to create a dashboard of how many HTTP Requests each one of them is receiving. Every microservice should have an Actuator[22][23] exposed, we can leverage an Actuator endpoint called httpTrace, the httpTrace Endpoint can provide information on how many HTTP Requests received and what kind of Responses served by the service. Utilizing this Actuator endpoint, we can create a dashboard using Pandas, Matplotlib, Microservice Actuator and Airflow.

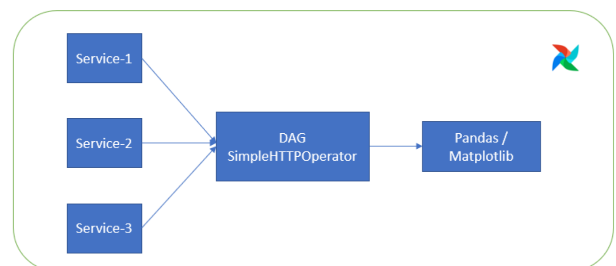

Figure 17: Airflow invoking Actuator

This use-case is not limited to /httpTrace there are various other Actuators exists one of them is /logfile to find the contents of the logs and that can be hand it over to Spark for further analysis. We have already seen how to submit the Job in Spark on GCP on Dataproc.

© 2022, IJCSE All Rights Reserved

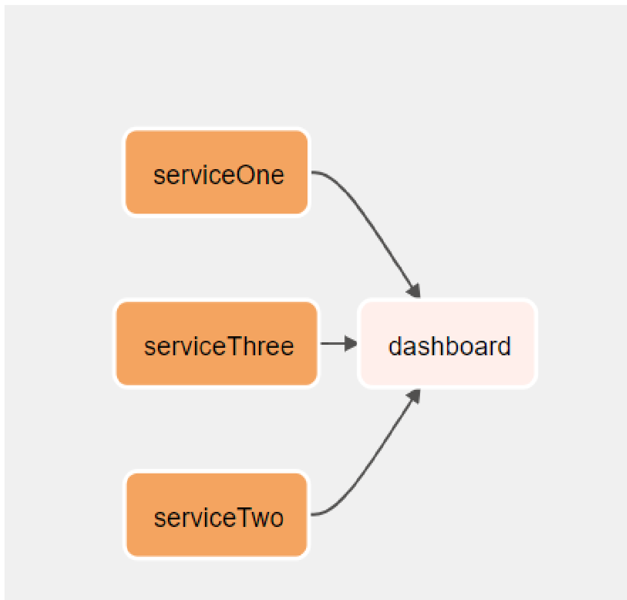

Figure 18: Airflow invoking REST APIs

For invoking REST APIs in Airflow "SimpleHTTPOperator" should be used and again it's a ready to use operator.

```
with DAG('rest_pipeline',
         start_date=datetime(2022, 1, 1),
         schedule_interval="@hourly") as dag:

    t1 = SimpleHttpOperator(
        task_id='serviceOne',
        method='GET',
        http_conn_id='call_service_one',
        headers={"Content-Type":"application/json"},
        dag=dag)

    t2 = SimpleHttpOperator(
        task_id='serviceTwo',
        method='GET',
        http_conn_id='call_service_two',
        headers={"Content-Type": "application/json"},
        dag=dag)

    t3 = SimpleHttpOperator(
        task_id='serviceThree',
        method='GET',
        http_conn_id='call_service_three',
        headers={"Content-Type": "application/json"},
        dag=dag)

    t4 = PythonOperator(
        task_id='dashboard',
        python_callable=dashboard,
        dag=dag)

chain([t1, t2, t3], t4)
```

Since the three services are invoked In Parallel that's why they are chained.

IV. RESULTS AND DISCUSSION

Apache Airflow is an excellent open-source workflow management tool which can be used to design and develop any kind of Pipeline, it doesn't matter how complex is the pipeline it is simplified by Airflow to a great extent. Apache Airflow UI provides us with all sorts of details about the DAGs like the performance of each task, Let's consider again a DAG with four tasks

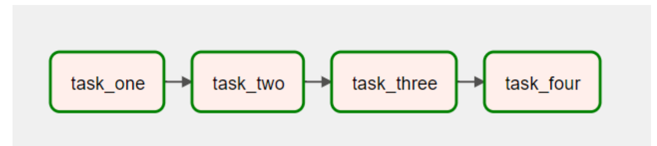

Figure 19: Sample Pipeline with Four tasks

Through UI we can monitor how much time each task has taken to complete and what's the performance in a day, from the Task Duration tab we can check the progress

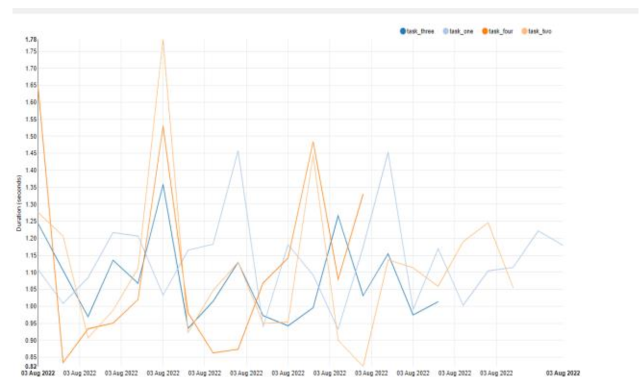

Figure 20: Metrics of all Four tasks

Also, we can check how many tries each task has performed even re-tries as well. The Monitoring tool is too intuitive and easy to debug, monitor, configure and track the pipeline.

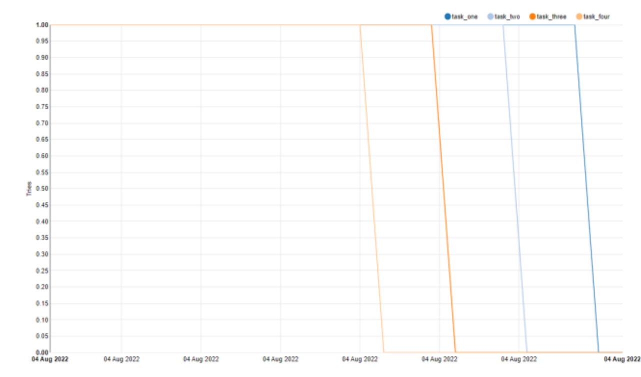

Figure 21: Tries of all Four tasks

Landing times,

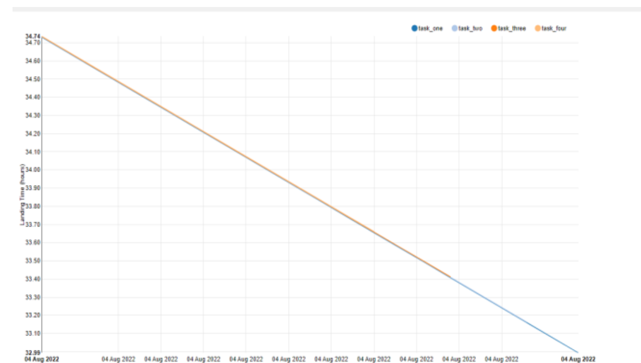

Figure 22: Landing times.

© 2022, IJCSE All Rights Reserved

The Landing Page of the monitoring tool show cases all the DAGs deployed in the Airflow

Figure 23: All DAGs in Airflow

Each DAG can be executed, refreshed, and deleted on demand-basis. DAG will be hot deployed to Airflow we don't have to manually deploy, it's real time deployment. As soon as we make changes in the DAG (.py file) it will be refreshed and deployed automatically. Configurations of sending failure email in case the task is failed for some reason is extremely simple, by using Simple Python dictionaries we can configure the DAG below code showcases how to send email in case of failures.

```
t3 = SimpleHttpOperator(
    task_id='serviceThree',
    method='GET',
    http_conn_id='call_service_three',
    headers={"Content-Type": "application/json"},
    on_failure_callback=email,
    dag=dag)
```

The attribute responsible for sending email in case of failure is "on_failure_callback" and the email is a python callable function

```
def email(contextDict, **kwargs):
    title = "Alert: {task_name} Failed because of
reason".format(**contextDict)

    body = """
    Hi Team, <br>
    <br>
    Task ID :{task_name} Failed.<br>
    <br>
    """.format(**contextDict)

    title('abcd@gmail.com', title, body)
```

V. CONCLUSION AND FUTURE SCOPE

Airflow can work on any Cloud Platform including Google Cloud Platform, AWS, Azure, and others. Once deployed on Cloud it's easily scalable because Airflow is managed by the Kubernetes engine. In Google Cloud Platform, Cloud Composer is a managed service for Apache Airflow, Cloud Composer has a seamless integration with other GCP components such as Google Cloud Storage, Big Query, Dataproc, Dataflow etc. This has a added advantage as Airflow can communicate to these services on Google Cloud Platform. There can be a use-case such as loading a file from one bucket from GCS and after processing it move to different GCS bucket, various ready to use operators are developed by Airflow team for such operations one of them is GCSToGCSOperator and the paper already described about the Dataproc Operators by Airflow. Airflow DAG can also be triggered by Google Cloud Function and Airflow DAG is also capable of calling Cloud Function during the Pipeline execution, the cloud function can be invoked simply by SimpleHTTPOperator, below code does the same.

```
t5 = SimpleHttpOperator(
    task_id="Cloud_Function_Task",
    method='POST',
    http_conn_id='http',
    endpoint='functionaName',
    data=({"schema": "", "table": ""}),
    headers={"Content-Type": "application/json"})
```

Airflow is extremely flexible the DAG can run themselves in scheduled time and DAG can also be executed using Google Function or even through pub-sub event. Airflow UI is extremely efficient for monitoring and tracking progress of the tasks within the Airflow, the UI is so detailed that it provides logs, performance metrics using Graphs and Charts. Airflow can be setup on GCP using Cloud Composer and on local it can run on Docker, the sample docker-compose.yml is available to download and run Airflow inside Docker Container. The main components of Airflow to get started are DAG, Operator, Sensor, Executor, 3Com's, Hook etc.

ACKNOWLEDGMENT

I would like to express sincere appreciation the Apache community for launching Airflow and preparing a crisp documentation to get started on the tool, specially the "airflow.apache.org" portal. The Documentation is very detailed it explains all the operators which are ready to use and how they function. Extremely Thankful to "astronomer.io" they have solved lots of Airflow related problems where the engineers stuck, and the solution provided by them are helpful. They have covered very use-cases like usage of Airflow in Machine Learning, Data Science, Operation Analytics.

The Slack community of Airflow is extremely active and very responsive, since Airflow is open source, anybody can be the part of Slack community and take the guidance or start contributing to Airflow. Sincere appreciation team to the Cloud team for developing managed services for Airflow which makes the development extremely convenient.

REFERENCES

[1] P. Covington, J. Adams and E. Sargin, "Deep neural networks for youtube recommendations", Proceedings of the 10th ACM conference on recommender systems, **pp. 191-198, 2016.**

[2] H. H. Olsson and J. Bosch, "From opinions to data-driven software r&d: a multi-case study on how to close the'open loop'problem", 2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications, **pp. 9-16, 2014.**

[3] Panos Vassiliadis, 'A Survey of Extract-Transform-Load Technology.,' July 2009 International Journal of Data Warehousing and Mining 5:1-27

[4] Tziovara, V., Vassiliadis, P., & Simitsis, A. (2007). Deciding the Physical Implementation of ETL Work-© ows. Proceedings ACM 10th International Workshop on Data Warehousing and OLAP (DOLAP 2007), **pp. 49-56,** Lisbon, Portugal, **9 November 2007.**

© 2022, IJCSE All Rights Reserved

[5] Vassiliadis, P., & Simitsis, A. (2009). Extraction-Transformation-Loading. In Encyclopedia of Da-tabase Systems, L. Liu, T.M. Özsu (eds), Springer, **2009.**

[6] Florian Waa, Tobias Freudenreich, Robert Wrembel, Maik Thiele, Christian Koncilia, Pedro Furtado, 'OnDemand ELT Architecture for Right-Time BI: Extending the Vision', International Journal of Data Warehousing and Mining **9(2):21-38 · April 2013**

[7] FabianPrasser, HelmutSpengler, RaffaelBild, JohannaEicher, Klaus A.Kuhn, 'Privacy-enhancing ETLprocesses for biomedical data', International Journal of Medical Informatics, **Vol.126, pp.72- 81, June 2019.**

[8] Ibrahim Burak Ozyurt and Jeffrey S Grethe, 'Foundry: a message-oriented, horizontally scalable ETL system for scientific data integration and enhancement', Database (Oxford). 2018; 2018: bay130.C. Wohlin, P. Runeson, M. Host, M. Ohlsson, B. Regnell, ¨ and A. Wesslen. ´ Experimentation in Software Engineering. Computer Science. Springer, **2012.**

[9] Venters, W., Whitley, E.A.: A Critical Review of Cloud Computing: Researching Desires and Realities. J. Inf. Technol. **27, 179–197, 2012.**

[10] Justin, C., Ivan, B., Arvind, K. and Tom, A. "Seattle: A Platform for Educational Cloud Computing"SIGCSE09, March 37, 2009, Chattanooga, Tennessee, USA. **2009.**

[11] Google Apps Education Edition: communication, collaboration, and security in the cloud.http://www.google.com/a/edu/

[12] Matei Zaharia, Reynold S Xin, Patrick Wendell, Tathagata Das, Michael Armbrust,Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael JFranklin, et al. Apache spark: a uni ・ ed engine for big data processing. Communications of the ACM, **59(11):56–65, 2016.**

[13] Creating Data Pipelines using Apache Airflow "Sameer Shukla" Volume 9 - Issue 4 International Journal of Computer Techniques (IJCT) ,ISSN:2394-2231 , www.ijctjournal.org

[14] S. Fortune, J. Hopcroft, J. Wyllie The directed subgraph homeomorphism problem Theoret. Comput. Sci., **10, pp. 111-121, 1980.**

[15] C.L. Lucchesi, M.C.M.T. Giglio, On the irrelevance of edge orientations on the acyclic directed two disjoint paths problem, IC Technical Report DCC-92-03, Universidade Estadual de Campinas, Instituto de Computação, **1992.**

[16] Y. Perl, Y. Shiloach Finding two disjoint paths between two pairs of vertices in a graph J. ACM, **25, pp. 1-9, 1978.**

[17] R. Agrawal and R. Srikant, "Mining Sequential Patterns", Proc. Int'l Conf. Data Eng. (ICDE '95), **pp. 3-14, 1995.**

[18] J. Chen and K. Xiao, "BISC: A Binary Itemset Support Counting Approach Towards Efficient Frequent Itemset Mining", *ACM Trans. Knowledge Discovery in Data.*.

[19] Vassiliadis, P., Simitsis, A., Georgantas, P., Ter-rovitis, M., & Skiadopoulos, S. (2005). A generic and customizable framework for the design of ETL scenarios. Information Systems, 30, 7, **492-525, 2005.**

[20] P. Merle, O. Barais, J. Parpaillon, N. Plouzeau and S. Tata, "A Precise Metamodel for Open Cloud Computing Interface", the 8th International Conference on Cloud Computing (CLOUD). IEEE, **pp. 852-859, 2015.**

[21] D. C. Schmidt, "Model-Driven Engineering", COMPUTER-IEEE COMPUTER SOCIETY-, **vol. 39, no. 2, pp. 25, 2006.**

[22] Bryant, P. G. and Smith, M (1995) Practical Data Analysis: Case Studies in Business Statistics. Homewood, IL: Richard D. Irwin Publishing.

[23] Zimmermann, O. (2009). An architectural decision modeling framework for service oriented architecture design. PhD thesis, Universitat Stuttgart.

[24] Badidi, E. (2013) "A Framework for Software-As-A-Service Selection and Provisioning". In: International Journal of Computer Networks & Communications (IJCNC), **5(3): 189-200, 2013.**

[25] F. Montesi and J. Weber, "Circuit Breakers, Discovery, and API Gateways in Microservices," ArXiv160905830 Cs, Sep. 2016

[26] G. Grahne and J. Zhu, "Efficiently Using Prefix-Trees in Mining Frequent Itemsets", Proc. Workshop Frequent Itemset Mining Implementations (FIMI '03), **2003.**

[27] Z. Zhang and M. Kitsuregawa, "LAPIN-SPAM: An Improved Algorithm for Mining Sequential Pattern", Proc. Int'l Special Workshop Databases for Next Generation Researchers, **pp. 8-11, 2005.**

AUTHORS PROFILE

Sameer Shukla has done Masters in Computers from Bangalore University, India in 2004.
He is having 15 years of experience in Software Design and Development, Currently Working as a Lead Software Engineer in USA and his current expertise/interests are Distributed Computing, Data Analytics, Microservices, Functional Programming, Cloud Computing, Deep Learning, SQL, NoSQL, Big Data, Spark, Data Science, Apache Airflow.

© 2022, IJCSE All Rights Reserved