International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 07 Issue: 06 | June 2020 www.irjet.net p-ISSN: 2395-0072

Automation of Data Pipeline Creation in Apache Airflow

Prakarsh Anand[1], Anuj Jain[2], Suma B[3]

[1]Student, R V College of Engineering, Bengaluru, India
[2]Employee, Bounceshare Pvt Ltd, Bengaluru, India
[3]Professor, R V College of Engineering, Bengaluru, India

Abstract - *A data pipeline is a series of data processing steps. Data pipelines consist of three key elements: a source, a processing step or steps, and a destination. They are scheduled at regular intervals using job executors. They are organised as a directed acyclic graph(DAG) containing a starting point and an endpoint in the DAG. When tasks are executed in sequential, parallel or any desired manner it gives out a meaningful outcome which is used for data analysis or data sciences. Today, data pipelines are being created using programming python scripts from scratch. Airflow provides a very useful user interface for monitoring of execution of the data pipelines as per schedule and requirement.*

The paper proposes an end to end UI solution from data pipeline creation to monitoring. It proposes to provide a state of art user experience. This allows non-technical users to leverage the advantages of data orchestration tools. The paper uses Airflow, an open-source data orchestration tool and a python module named "DAG-factory" to implement the proposed solution. Approximately 65 % of all the users of Airflow are data engineers and 97% of the data pipelines executed on Airflow are used for Data Processing(ETL). Therefore, the paper focuses on data processing users as its primary users to provide the best user experience.

Key Words: ETL, Airflow, data engineering, DAG, DAG - factory, data-pipelines

1. INTRODUCTION

To deal with scheduled job execution many data pipeline orchestration tools like Airflow have been developed. Their primary work is to create robust task inter-linkages in a data-pipeline and perform scheduled runs at a regular interval. Out of several orchestration tools, Airflow provides a lot of monitoring and customisation functionalities as GUI.

Quoting Apache Airflow documentation - "Airflow is a platform created by community to programmatically author, schedule and monitor workflows". Airflow provides features to monitor workflow progress and execute DAGs. It allows visualisations to check how a DAG execution time varies with an increase in data to be processed, to check if a task execution can be performed in parallel without any data dependency. It allows back-filling of data through scheduled DAG execution using DAG execution date and not the start date of DAG. These features have undoubtedly made it massively useful worldwide.

1.1 Existing Systems

Airflow became Top-Level Apache Software Foundation project in January 2019 and has been used by a lot of companies catering their need to perform different data orchestration easily and efficiently. Airflow also supports a lot of different formats of input allowing it to be used universally to process any kind of data stored or task to be executed. Creation of tasks needs an operator in general. Airflow supports different kinds of operators, hooks or sensors where each of them can be used to create a task to be executed in a workflow. Different data pipelines are created using the customised cross-linking of different tasks which may run in parallel or sequence as per specified task dependencies created to serve the purpose of DAG creation in a meaningful manner. Airflow also supports python operator which opens a whole lot of possibility of tasks to be created ranging from machine learning module execution to simple control statement execution. The usefulness of python operator in Airflow environment is because any data pipeline created is a python based configuration file which is read by Airflow's python interpreters. Therefore, a python based configuration file can be used for any DAG creation.

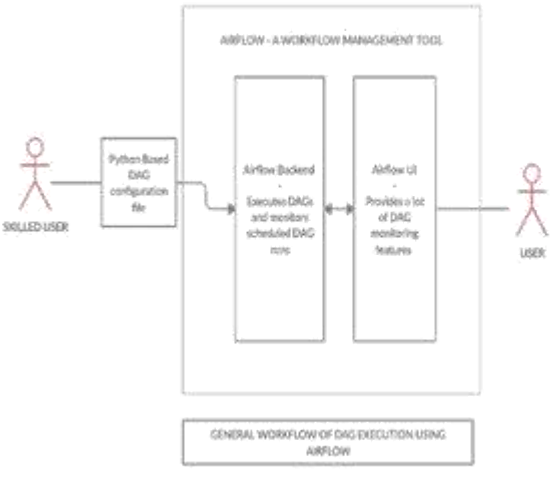

Figure 1 General Workflow of Airflow

Airflow's execution environment and the standard steps to process DAG creation and execution comprise the whole framework used to perform data orchestration. It has been widely in use by a lot of organisations but with a very

obvious lagging step of programmatical data pipeline creation. It leads to different complications like

1. One needs to know python programming to relish the full potential of the variation of data pipelines that can be created and executed.

2. The limited set of operators, sensors or hooks in contrast to rich functionalities provided by python has led to a lot of redundant and non-standardized codes across organisations. They generally are observed as different combinations of same task structures driven by mere syntax usage of operators defined by Airflow.

3. Creation of different ways of solving the problem of repetitive code-based data pipeline creation using pre-processing of different configuration files parsing. For example ".ini" extensions or ".yaml" extensions.

4. If DAGs are being created by not looking after the code repeatability issue, it allows errors like no standardisation of codebase at least across organisation. Human error during implementation increases as the dependability on the way an individual gets his/her work done creeps in every time. Machine-based task creation allows code standardisation and removes human intervention reducing the errors of DAG creation to be negligible. But this doesn't solve the code repeatability issue. This impacts the scalability of code and the organisation as a whole.

5. The steps or creativity applied using the tools available to reduce code repeatability by organisations are generally organisation specific which caters to the needs of an organisation and solid documentation is needed to scale the organisation and code base efficiently. It also means any new user of the non-standard tool created by the organisation needs to learn it from scratch using the organisation resources and it can't be used as a standard tool which doesn't serve the purpose as per the analysis of this paper.

1.2 Scope and Motivation

With every addition of module in the python library, DAG creation becomes more efficient and its userbase scales massively increasing the use of Airflow in particular multiple folds. One such python module addition is " DAG-factory " which if used in full potential will open DAG creation and execution for non-technical people. This will allow them to work with data orchestration as efficiently as people who know python programming. The proposed methodology leverages the full potential of DAG factory python module which reads a YAML based configuration file and converts it into a python-based configuration file internally. This python-based configuration file is used by the Airflow execution environment to create and execute data pipelines. Always a YAML based configuration file readability and writability are far more generalised and easier than a python-based configuration file creation. It allows non-python developers, common users of the data pipelining environment or any person who can understand configuration files to understand and write a configuration file which involves very less complexity as compared to python-based configuration files and thus has fewer chances of human error and increase in the user base of airflow as a data orchestration platform. This is the whole purpose of implementation of "DAG factory" module as an open-source module. It serves the purpose of an increase in the user base of open-source data pipeline execution environment Airflow. It also creates a potential extension of the usage of the module to completely change the way DAG creation has been looked and assumed by the world so far. It has given the potential to create an environment which may create the desired DAG on click of a button or a mere command-line execution. The paper describes an approach which may potentially change the world's notion of data pipeline creation from being a skilled person in required technology to merely using the UI to specify what you want to perform using the data pipeline and the resources to be used.

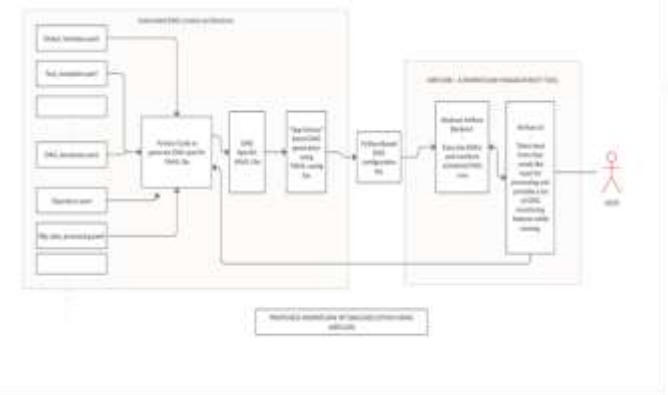

Figure 2 Proposed Airflow Architecture Diagram

2. METHODOLOGY

Airflow defines DAG as a Python script, which represents the DAGs structure (tasks and their dependencies) as code in their official documentation. This has been interpreted as skilled users may generally create a python script to create data-pipelines. Airflow uses a python-based configuration file to interpret and create data pipelines accordingly to be scheduled to execute at regular interval. These python-based configuration files can be created through coding the python configuration file manually or using the DAG-factory module which parses a configuration file and converts it into a python based configuration file to be interpreted for DAG creation. The DAG-factory module only needs the source path of the configuration file to fetch the file content and parses it to create a python based configuration file. A configuration file can contain many DAG definitions where each section in the configuration file may be fetched as a different DAG definition. Here, the configuration file is the standard content

specified in different DAG sections having DAG properties like the inter-task dependencies, different task definitions with user-defined values. The creation of these configuration files may be done manually or through a python code which generates the configuration file as per user requirement specified as system arguments while execution.

The python code uses the crux of the proposed solution to convert the DAG generation process to be easy such that it can be created by a non-technical person through execution of the python code with user required arguments. The solution proposes a general configuration file which contains template based declarations of all the global variables required to create a DAG. Templates based declaration of all the operators , sensors and hooks required to create a task, different task definition templates which may be directed as per the DAG generation specified by the user. The general YAML file contains and may be appended with several DAG templates following the principle of write once and read many times. People can add their own customised DAG templates in the general configuration YAML file .The python code reads and takes out the template from the general configuration file as per the DAG to be created specified by the user . It fills in the values required to define the DAG. The DAG definition is at last dumped into the YAML file to be used by the DAG factory module to create the data pipeline.

Figure 3 Configurable DAG specific templates

This process repeats without any human intervention whenever a DAG is generated creating a drastic change from the way people are used to code a python-based configuration file to an automated configuration file generation using standard parameters already specified in the general configuration file. The python code automatically takes in the required specifications and builds out a data pipeline using the generated configuration file.

3. IMPLEMENTATION

The end goal to be achieved is to create a platform for any user to be able to create a data pipeline on click of a button or a command line execution with arguments passed as specifications. As per Figure 3 and 4, the general configuration files consists of different sections on broad level naming task templates, DAG templates, default arguments template, spark cluster templates, operator templates etc which may be changed and appended as per use and requirement. Each section consists of different types of templates containing different possibilities addressed so far and which may be used later. Every section may be appended with more templates if required and can be used as same as other templates. The sections are created distinctly keeping in mind the modular approach such that the python code which takes the templates from the general configuration file may take the required templates from different sections as per mentioned in the template of DAG to be created which the user mentions as a parameter while executing the python code. For example, any template of a specific type of DAG only contains the parameters specific to the DAG declared once as in declaration of what tasks would be in the DAG and their interdependencies, the declaration of DAG specific parameters which are used by the python code to fill in the values for the key-value pairs specified in the template. The task section contains all of the possible task definitions and is included in the DAG generation code using their declarations in the DAG template. Similarily, the default arguments which is generally passed to every data pipeline created is defined in a separate section and is used by the python code to feed into every DAG creation as per required. All the task definitions are present in the python code using jinja templating which when picked and merged in a custom manner as per requirement or interdependency of tasks as per the definition of the DAG template specified by the user creates a fully defined DAG to be dumped into a configuration file which is read by DAG factory module to create a python based configuration file to be executed by airflow. Addition of new task in the tasks section of the general configuration file also means addition of its corresponding definition in the python code.

4. INNOVATION

The existing systems doesn't allow every user to create data pipeline easily as it encourages to do so programmatically which corresponds to knowledge of the required technology stack in order to deliver the correct outcome. The proposed architecture is robust in nature and abstracts all the implementations of a data pipeline from creation to execution of data pipeline. The paper proposes an architecture which reduces the human intervention in data creation process to create the desired python-based configuration file using the below mentioned approach

1. Any completely defined data pipeline python based configuration file can be divided into modular sections like task definition, task interdependency, default arguments, operators used etc.

2. Create different template based configuration files for every section possible in a completely defined data pipeline leveraging modularity and cutting down code repeatability.

3. Develop a python script which

- takes in the specific arguments from the user
- fetches the required templates from different configuration files
- converts the templates into properly defined modular sections using jinja templates

- clubs the required modules in specific order to create a completely defined configuration file.

4. The configuration file is processed by "DAG-factory" module to convert it into a python based configuration file.

5. The python script is an executable used by Airflow interpreter to execute DAGs.

The whole process of DAG creation requires user to provide the information about what to perform using the DAG to create, knowing the resources and links to be used to accomplish use of DAG. This approach may potentially change the world's notion of data pipeline creation needs a skilled python programmer.

5. RESULTS

	Existing Implementation	Proposed Implementation
GUI based Automated DAG creation	No	Yes
Prior knowledge of any programming language	No	Yes
Knowledge of Airflow primitives	No	Yes
Duplicacy of Code	Yes	No
Human Error Possibility	Yes	No
Reduces effort required to create customised DAGs	No	Yes
Reduces time taken to create multiple DAGs	No	Yes
Basic Understanding of Airflow required to create data pipelines	Yes	Yes
Transformation from low level programming to abstracted code reusability through templates	No	Yes
Abstraction — Interoperability and platform independence	No	Yes
YAML configuration file used than a python based configuration file	No	Yes
Increases flexilbility of DAG generation types	No	Yes
Modularity - Read once write multiple times	No	Yes

Figure 5 Comparison of proposed vs existing implementations

The criteria chosen to compare the 2 implementations define the advancements possible in Airflow user experience. The system has been effectively tested in a rapidly growing software company, and showed satisfactory results. The implementation correctly ensures the reliability and robustness of the system testing process. The comparisons where done between the existing python-based DAG configuration files in the company and their equivalent conversions into the template-based implementation. The equilavent template-based implementation where used to automatically create python based DAG configuration files on issue of desired arguments to the python code generator .

6. CONCLUSION

Template-based automation of DAG creation is an expiremental project work taken in order to prove that it's efficient and effective implementation will definitely cause a shift in mindset from the way, data pipelines are generally written in today's world to an automated DAG creation platform where specifying the expectation from the data pipeline to be created is enough .This template based implementation works quite well because of the simple reason that a task is the fundamental unit of any data pipeline and any task is created using an operator definition from a set of limited number of operators. Therefore, a proper definition of usage of every operator in a manner such that the definitions can be used by any task implementation is enough. Data pipeline implementation makes the proposed template-based data pipeline creation work. It also allows a very robust implementational abstraction of data pipeline creation from the end user who doesn't need to know the implementational part ensuring the benefits of abstraction to be leveraged later.

7. FUTURE WORK

It can be started as an open source project to create an ecosystem where different and new tasks created, Different and new DAGs created can be pushed in a general config file and they can be used as template to generate similar DAGs multiple times all over the world on a single search on the file with specific parameters assigned. Further, the future work for the project would be to make it robust and enough user friendly so that it can mainstreamed by the open source community to use it for data pipeline creation rather than writing repeated codes. Making the User interface as easy and user friendly as possible so that people can specify search through the DAG templates available, create a non-existing DAG template through clicks of buttons, select the template for DAG creation and specify parameter changes if required to create data pipeline. Link the DAG creation to direct execution of data pipeline on AIRFLOW. Thus completing the user experience of creation and execution of data pipelines through clicks of button than coding all of it together.

8. ACKNOWLEDGEMENT

We would like to acknowledge the support provided by employees of Data Engineering Team, Bounceshare, Bengaluru, India and teachers of Department of Computer Science & Engineering & Department of Biotechnology, RV

College of Engineering, Bengaluru, India through their assistance during the research work.

9. REFERENCES

1. H. Bal, M. Haines, "Approaches for Integrating Task and Data Parallelism", IEEE Concurrency, vol. 6, no. 3, pp. 74-84, July-Aug. 1998.
2. O. Delannoy, N. Emad, S. Petiton. Workflow Global Computing with YML. In: Proceedings of the 7th IEEE/ACM International Conference on Grid Computing, GRID'06. ACM Press, New York, 2006. 25-32.
3. F. P. Guimarães and A. C. M. Melo, "User-Defined Adaptive Fault- Tolerant Execution of Workflows in the Grid," in Proceedings of the IEEE CIT, Sep 2011, pp. 356-362.
4. L. Li, Z. Miao, L. Yuqing, Q. Liangjuan, "A Survey on Workflow Management and Scheduling in Cloud Computing", Cluster Cloud and Grid Computing (CCGrid) 2014 14th IEEE/ACM InternationalSymposium on, pp. 837-846, 2014.
5. M. Kotliar et al., "CWL-Airflow: a lightweight pipeline manager supporting common workflow language", bioRxiv, 2018.
6. M. Beauchemin, (2014) Apache Airflow Project.
7. A. Barker, J. Van Hemert, "Scientific workflow: a survey and research directions", International Conference on Parallel Processing and Applied Mathematics, pp. 746-753, 2007.
8. E. Deelman, T. Peterka et al., "The future of scientific workflows", The International Journal of High Performance Computing Applications, 2018.
9. R. Sumbaly, J. Kreps, S. Shah, "The big data ecosystem at linkedin", Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, pp. 1125-1134, 2013.
10. W. Hummer, V. Muthusamy, T. Rausch, P. Dube, K. El Maghraoui, "Modelops: Cloud-based lifecycle management for reliable and trusted ai", 2019 IEEE International Conference on Cloud Engineering (IC2E'19), Jun 2019.
11. G. Alonso, D. Agrawal, A. El Abbadi, C. Mohan, "Functionality and Limitations of Current Workflow Management Systems", IEEE Expert, vol. 1, no. 9, 1997.
12. M. Rosemann, "Evaluation of Workflow Management Systems - a Meta Model Approach", In: Proceedings of the 2nd International CAiSE/IFIP 8.1. Workshop 'Evaluation of Modeling Methods in Systems Analysis and Design, 1997.
13. M. Berger et al., An Evaluation of Workflow Management System, Austria:Institute for Applied Computer Science and Information Systems, University of Vienna, 1997.
14. Deelman, E., Vahi, K., Juve, G., Rynge, M., Callaghan, S., Maechling, P.J., Mayani,R. Chen, W., Da Silva, R.F., Livny, M., et al.: Pegasus, a workflow management system for science automation. Future Generation Computer Systems 46, 17–35 (2015)