ETL Automation and Orchestration with Apache Airflow

Ravi Shankar Koppula

Satsyil Corp, Herndon, VA, USA
Email: *ravikoppula100[at]gmail.com*

Abstract: ***In the contemporary landscape of data engineering, ETL (Extract, Transform, Load) processes are pivotal for efficient data management and analytics. Apache Airflow has emerged as a powerful platform for orchestrating complex ETL workflows, offering robust capabilities for automation, scheduling, and monitoring. This article delves into the core functionalities and architecture of Apache Airflow, illustrating its efficacy in managing ETL pipelines. It covers the creation and management of Directed Acyclic Graphs (DAGs), task scheduling, and execution, as well as integration with various external systems. Additionally, the article highlights best practices for optimizing performance and ensuring reliability in ETL operations. Through comprehensive examples and case studies, readers will gain insights into the practical application of Apache Airflow for streamlined data workflows, ultimately enhancing data processing efficiency and accuracy.***

Keywords: ETL Automation, Data Orchestration, Apache Airflow, Directed Acyclic Graph (DAG), Task Scheduling, Workflow Management, Data Integration, Monitoring and Logging, Data Engineering, Performance Optimization.

1. Introduction to ETL Processes

ETL simply stands for extract, transform, and load. While the words themselves are explanatory, it's important to note that there are different ways to perform ETL operations.

For instance, many development teams believe in performing multiple ETL operations through scripts that are scheduled on cron jobs. As one can imagine, this approach not only has to scale facing dynamic requirements, but it becomes difficult for scheduling as well. Similarly, some of these scripts tend to perform ETL operations that are second - degree parameters of the script itself.

Another way in which ETL can be performed is setting up batch jobs. Many cloud - based services provide capabilities that allow the individual to schedule a job which involves specifying functions that perform ETL operations. While many cloud services provide predefined functions that we could take advantage of, one of the limitations of using such a service is that we are often left in isolation with certain parameters. For instance, these services provide a certain driver that is compatible with them, while all the functions should use this driver. This results in the user being trapped as far as the choice of technologies are concerned.

1.1 Definition and Importance of ETL

Modern data analytics practice almost always involves the collection, preparation, and transformation of raw data - a step commonly characterized as ETL (extract, transform, load). Decades ago, ETL was executed mainly by batch processes that moved data mostly at a particular frequency (such as weekly, daily or hourly). In the current context, data ETL typically involves the reification of data flow processes that run continuously in real time, as well as at scheduled intervals. ETL engineering frequently attempts to optimize these triple data flow problems (batch, real - time, and continuous) by choosing the right tools and languages, and the right amount of oversight, governance, and management of the data process.

Industry has produced various tools, libraries, and domain - specific languages to address these data flow problems in diverse data scenarios and environments: ETL tools such as Talend, Alteryx, Pentaho, and Informatica, as well as Unix system utilities such as cat, awk, and sed, and programming languages like R, Perl, Python, and their associated libraries, are frequently used to create complex batch - oriented transform - and - transition processes. Real - time data processing and streaming frameworks such as Apache Spark, Kafka, and Nifi, among others, facilitate the construction of resilient streaming processes. Recent software frameworks and libraries allow data practitioners to effortlessly orchestrate these transitions at scale: Apache NiFi, Pass, Workflow (percolate), Bubbles (Airflow/link), Conductor, Digdag, Ruigi, Marionette Collective, Chronos, Flink, Oozie, and Azkaban are only a few examples of such tools. Although these tools often fulfill their primary mission, no clear industry standard as of yet fully allows and simplifies organization, orchestration, and monitoring of ETL - based data portability challenges at scale and ease.

1.2 Key Components of ETL Processes

Before talking about automation, we need a clear understanding of the tasks involved with ETL. Of course, it goes without saying that we start with extraction and add a transformation and load component to our processes. Some processes require a higher level of understanding and, thus, a fourth translation step. No matter how complex or simple the ETL process, we must all adhere to the same rules: the underlying data must always be available; it is the scope of the architecture to ensure data is easily accessible.

Central to this concept is a list of key components of an ETL process. The first component refers to the data itself; any part of the ETL process can be included in generating data. This might only mean summarization, but it will also include data collected during extraction, generated and archived to extract. Other data is used only during transformation; for example, data cleaning and correction is completed. Data is also stored and required for running an ETL process, including cloning and storing. Data extraction and generating the pipeline, and log the problem are illustrated in a variety of data source or data. Great lots to partner products were location - based, but not necessarily available to support multiple steps throughout the ETL process. Consequently, users did not have the ability to support ETL without the development of custom scripts.

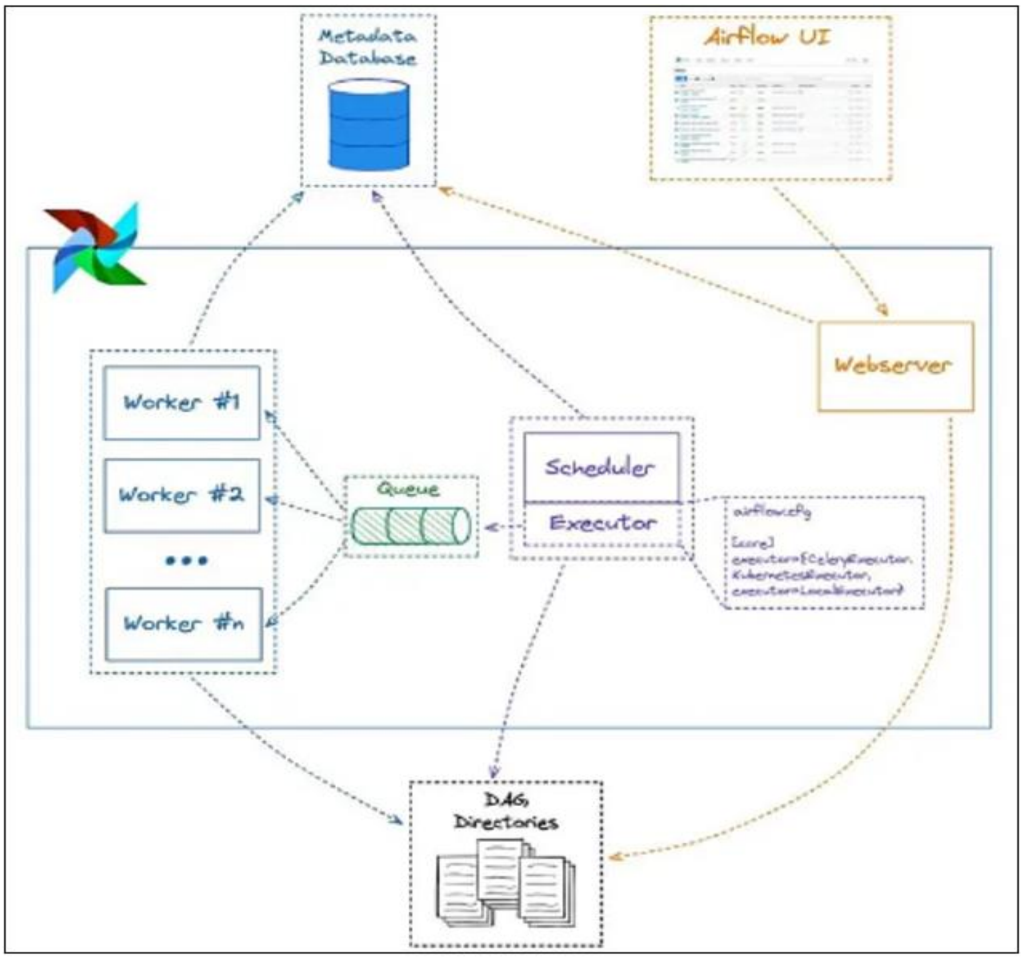
2. Overview of Apache Airflow

Developed at Airbnb, Apache Airflow is an open - source application for scheduling, monitoring, and orchestrating arbitrarily complex as well as relatively simple data - related workflows. The code base is written in Python, the hypervisor of the data science community. Airflow is particularly well - suited for ETL tasks. It allows teams to execute ETL workflows that are streaming and which are partitioned into (identical) time intervals. The latter results in data pipelines that are simple and easier to maintain because a single task can encapsulate the processing of many time periods. Airflow deals well with the rate of data growth, and it is relatively simple to horizontally scale the Airflow infrastructure. In addition, the workflows are modeled in Python, which results in data science teams being more productive, and it allows data science teams to work with technologies they already know, such as Pandas, Sqlalchemy, and/or Spark. [1]

By encapsulating tasks in Docker containers, Airflow also allows for heterogeneous processing demands, where the type of task (I/O bound, CPU bound, etc.) defines the machine on which they are run, including any required processing frameworks or libraries. The deployment of Dask, Apache Beam, Spark, Pandas, Jupyter, and Flask tasks are all possible in Kubernetes. By making heavy use of Docker, both Dask and Apache classifiers, often needed for this type of work, can be installed as part of the base image, while Spark tasks are simpler when deployed through either images or DockerHubs. Finally, Airflow allows both for on - premises and off - premises deployment, including cloud - based solutions.

2.1 Introduction to Apache Airflow

Apache Airflow is an open - source platform that facilitates the orchestration and automation of complex, primarily data - driven workflows. By using Airflow, data engineers can author, schedule, and monitor workflows programmatically from simple task chains to complex, multi - step data pipelines. Airflow was originally developed at Airbnb. Since the project's open - source release, Airflow has seen widespread adoption and a strong community of contributors. With a choice of plugins and a high degree of user customization, Airflow can represent it any way you like. It's a relevant tool for any data pipeline formed with a connection of sharing and understanding of the context and state of the automation.



Data pipeline orchestration tools (such as Apache Airflow) are increasingly being asked to solve problems that can only be solved by ETL tools. In some enterprises, the advent of the E in ELT (loading data into cloud platforms first and transforming in the platform) has led to useful delegation of the T function to cloud platforms. However, orchestration tools such as Apache Airflow need to continue to assume roles of crucial importance for data engineers to earn retention in their respective data engineering roles. Being able to deliver into more locations with common paradigms will help Apache Airflow in particular and orchestration tools in general to retain their place at the center of many organizations' data architectures. [2]

2.2 Key Features and Benefits

The primary role of Apache Airflow is system orchestration and task automation. It is a powerful tool with a rich ecosystem of a wide variety of providers of different services, with which to integrate. This makes Airflow suitable for developing the overall Data Lakes & Analytics platforms. [4]

We can acknowledge the main benefits: Ad Hoc parallel ETL code parallelization, with no modifications in the DAGs. Very handy Airflow UI - for developers, analysts, ETL Engineers, Data Scientists, DevOps, and Data Ops. Workflow scheduling - we don't need to manually trigger pipelines. Airflow distributes, monitors and tests the data processing. It recognizes the pipeline result, and if negative, does not trigger the next pipeline.

Airflow can monitor a wide variety of processing: ETLs, AI/ML APIs, Data Lake downloads, and loads between on - prem deployments and cloud environments. Task retries - it retries the tasks if one fails - as long as the other upstream task completes successfully. We will configure the number of retries and wait time for each of the tasks. The developer community - a large and very active audience. If we have issues we didn't solve, we can get support from Data Engineer community at Stack Overflow. With such a rich ecosystem, we could get our answer from there in less than one hour. [3]

3. Setting Up Apache Airflow

By definition, a directed acyclic graph, or a DAG, is a graph where edges have a direction (they go from parent to child), and where there are no cycles. In other words, you could start your day by going to school, then go to work, then go home. But you couldn't go to school after going to work and then going home. In programming, DAGs are important in the visualization of the steps needed to achieve a specific goal. To set them up, we need to understand the topology of the data pipeline and the dependencies amongst the different steps needed in the pipeline. Your project setup will have a bunch of DAGs, and each DAG will describe the different steps in an end - to - end ETL orchestration workflow.

In Apache Airflow, each DAG is a set of tasks. The ETL function for the pipeline is the task of each node in the graph. Each task is a piece of work in the workflow, and they all have to come together to complete the DAG. Task dependencies allow tasks to only start when one or more tasks are in the successful state. In this way, from the top - level, we escalated the how question to the more practical combination of what do I have to do, and in which order, to execute the steps required to achieve the final output.

3.1 Creating and Managing DAGs

The Directed Acyclic Graphs or DAGs are dataflow ones converting the high - level workflow logically to clearly defined tasks' configurations that are run by workers to ultimately perform their duties. Task metadata persists execution variables. Each DAG represents a value passed from outside, while each task - an action that relies on that parameter. DAGs themselves are valuable and can be complimented by the concept of an External Task in which a new DAG is created instead of a new Task, but all the nodes that were using the external task to signify that work is done can now reference the external DAG to query and dissect its logic.

The power of the use of graphs is beautifully illustrated by the Airflow's parent possibilities. It's easy to create parental child relations with a constraint that a child may only be executed if the parent was successful. The recursion pattern may be executed until a defined limit.

DAGs are defined in a. py file that gets executed and parsed by the scheduler at a defined run interval, which then maps out the blueprint for Worker actions. Custom operators may be created to accommodate required actions. Crucial parameter in DAG definition is the schedule_interval. Schedule interval is the periodicity of DAG runs in the format of cron expression, and DAG's can be set off by triggering the scheduling of this property, by CLI command, web interface, or by another DAG. Being launched they check if the task should run, and Task instances fire off workers that execute the developer's logic. Optionally the Unique constraint can be helpful to let Airflow know that the same task could be reused across multiple runs. In long running operations it may happen that the task is yanked in and out of the execution queue, but we are only interested in the task run with the dag_run execution.

4. Automating ETL Workflows with Apache Airflow

The main event in an ETL project is the creation of workflows. The ETL schedule is important to keep the project working efficiently and to keep the workflow on the right execution path. Workflows are sets of tasks that, in total, accomplish a particular kind of work. Complex workflows build on combinations of other workflows and data - driven dependencies. With Airflow, the workflows are the dynamically generated Directed Acyclic Graphs (DAGs) that organize and coordinate the user's tasks. Each DAG, a Python script, must define a unique dag_id. The data engineer can provide an ordered set of default args and provide metadata for each DAG. These can specify RBAC settings and start, end, and execute times as well as deschedule intervals. With DAG objects, the Airflow Pipeline objects can schedule the workflow.

4.1 Defining ETL Tasks in DAGs

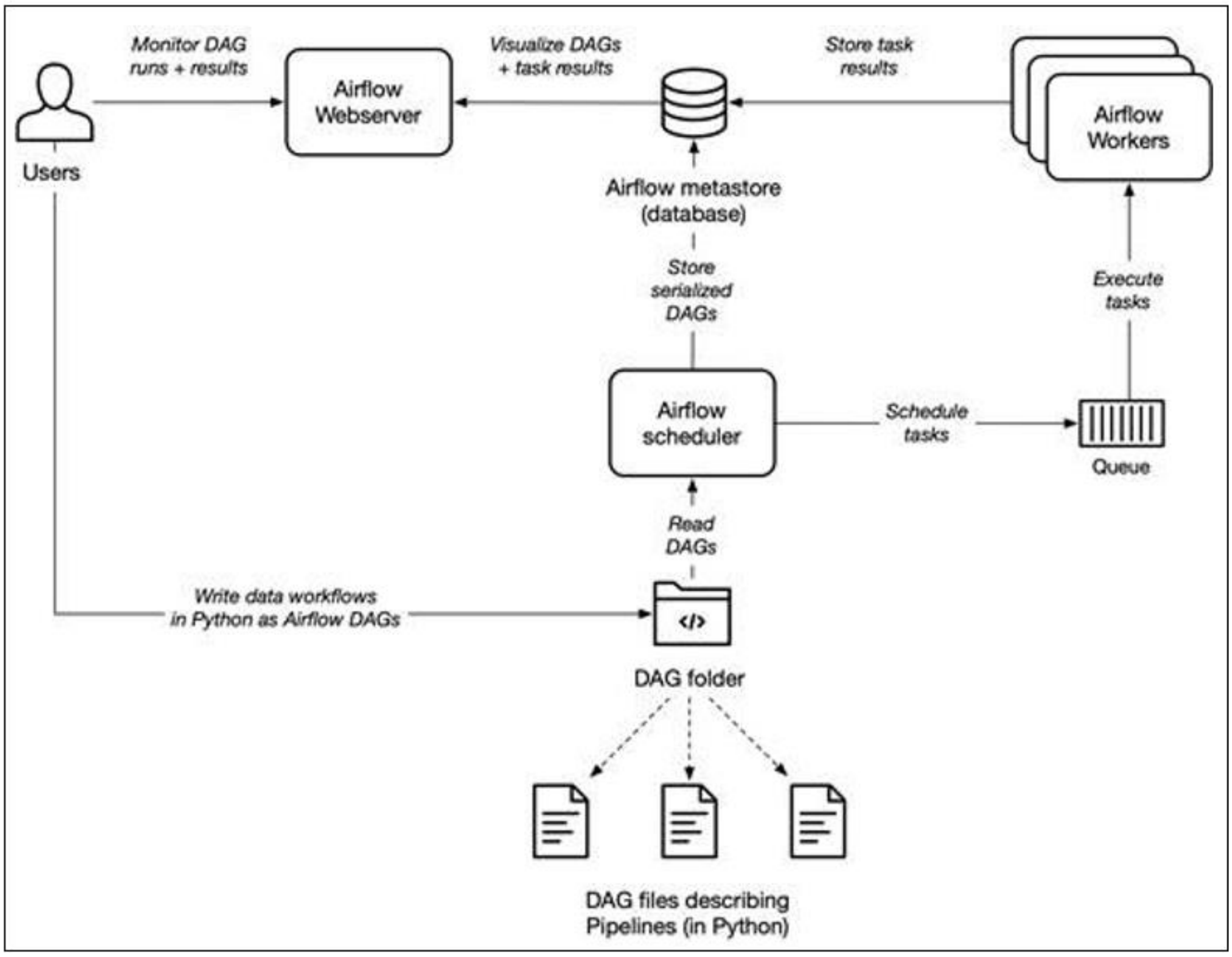
To keep our ETL process manageable, we will break down the entire process into multiple tasks in a Directed Acyclic Graph (DAG). A DAG is a model that represents a sequence of tasks where some of the tasks are dependent on the others. The sequence assures that if a task depends on other tasks, these other tasks will be executed first - and this allows us to walk the path of the graph without repeating a node.

In computerized systems, the sequential tasks that need to be executed can happen because a process is dependent on the one before to operate, as in a chain; or because there is a natural sequence of insertion/ transference/ organization/transformation, etc., on the items processed by the tasks (ETL tasks, for example).

In our case, it's also important to make sure that our ETL process continues to be a reactive process: If Team A decides to start uploading data at the same time every day, we should be able to accommodate that. Given a DAG to define, we need to lay down the edges of the graph to establish the order of the execution of our tasks - by setting up the so - called "task dependencies". One thing to bear in mind is that, at a given point in time, in our ETL process there should be multiple tasks running, each one working on a different portion of the data we want to process, in parallel with the other tasks. This will make sure that our ETL process will be able to scale with increased data loads.

4.2 Scheduling and Monitoring ETL Jobs

A DAG runs four times and is disabled after a successful execution. The first task plays the role of populating and preparing the target table and takes care of creating and recreating the target table. This operation may be part of a pre - ETL session, but we merge it into the airflow operation for simplicity. The other four tasks run the actual ETL operations. The first task operates on data which is less than 30 days (the number is arbitrary) old. The final three process older data and provide a 12 - month rolling total (month - to - date, previous month, and second prior month). After the last task 0528 - Airflow ETL Orchestration 165 is run, a final task performs some record count verification, and post - ETL activities may also be added. The actual DAG definition is straightforward. It consists of ten basic operations. Airflow's idea of an ETL process may be broader and more encompassing than the traditional concept; for example, basic administrative operations like table creation may be included as well as more typical ETL - type operations. Ultimately, the DAG is converted to Kubernetes pod operators on the Airflow Toaster k8s.


Monitor DAG runs + results
Airflow Webserver
Visualize DAGs + task results
Store task results
Airflow Workers
Users
Airflow metastore (database)
Store serialized DAGs
Execute tasks
Airflow scheduler
Schedule tasks
Queue
Read DAGs
Write data workflows in Python as Airflow DAGs
DAG folder
DAG files describing Pipelines (in Python)

Airflow UI: Airflow provides a very powerful and modular UI. The ETL orchestration configurations can be monitored on the UI, and any problems quickly identified. The web UI is a view of the metadata DB. A combination of RBAC (Role - Based Access Control) and a general info page (the general structure of any Airflow Job) can be created and inform the ETL orchestration functions about these issues. The metadata tables pg_public. dag_run, pg_public. dag_tag, and pg_public. task_instance contain the runtime, status, and tag information regarding the dag and should provide a starting point. The Airflow structure also provides for a logging system which automatically maintains the operational logs of every operation performed. In our case, the airflow. cfg is modified, AIRFLOW - LOGGING - TO - OUTPUT is available and pointing to STDOUT. The ETL function probjoblib. out_and_in also contains code that integrates the logs into the standard airflow log table. A monitor dashboard can be set up for real - time monitoring of the ETL process. [5] [6]

5. Best Practices and Advanced Techniques

In the context of a robust ETL system, achieving a stable operation is key. The consequent aspects and techniques focus on considerations that are necessary for the efficient handling of many levels of issues/problems that may arise in operation. The use here of complex software helps ease operations. Ensuring privacy, separate environments of all types for both testing and processes in operation (development, production), configuration driven by orchestrated and version - controlled files, logging and alerting mechanisms, use of the retry mechanism, exceptions and automatic recovery/backfill of incomplete, failed, or stalled tasks, free up resources for further operations, while automated metadata db cleaning through the use of changelog tables between big ETL pipeline runs, helps keep high performance. On the other hand, avoiding complex tasks and/or execution of several tasks concurrently, handling failures in a special way, dealing with data inconsistencies (available data problems), using highly available services for ETL and its db operations, using db connection pooling through optimized large batch inserts, alerting on db connection problems, using proven easy help thread or processes for backfilling, or fixed data availability issues often create stability and avoid failures. [7]

5.1 Performance Optimization Tips

To minimize the memory required for a task, Apache Airflow includes a configuration cmdopt flag "pickle" that determines how a task object's state is serialized to the database. You could disable pickling by setting the pickle options to False. This will make the dependencies between the tasks less informative and some of the visualization functions won't work, but you will save a lot of database space. The tasks' dependencies and parameters will still be recorded by Airflow, but you won't be able to access them easily. Therefore, this would not be a recommended option, especially for organizations that require all the dependency details to be stored.

Configure a logging level that fits your demand. It is recommended to use "logging. WARNING" for lower log levels to prevent the log from blowing up. For a single DAG, the log can be comprehensible. However, when you have hundreds of DAGs, especially running in a tight schedule, traceback actions and detailed logs can bring a large performance overhead for the Webserver. In this case, a higher logging level setting can reduce the workload significantly. Also, it is highly suggested to set the embedding of Scheduler job inside a DAG to True. This function can provide more satisfactory insights on where the scheduling errors occurred, and if the logging level is not well - organized, the display of the Scheduler job inside a DAG parameter will be removed accordingly. [8]

5.2 Error Handling and Retry Strategies

Skipping data or returning null is seldom useful whenever an error occurs. It is generally better to get the operation to perform a specific operation, allowing the end user of the data set to understand and investigate the ETL process. Both the OperatorBase and the BashOperator have an error_msg, and the BashOperator has an error_tags parameter. It is easy to use this feature with Bash scripts because they can output custom messages without emitting erroneous messages. However, the failure status of the OperatorBase will be determined based on whether the task gets a status code of 0.

If you have conditional logic that must pass to avoid errors, the only way to handle this condition for a task is to build that logic into workflow code such as if you're programming it. The TaskGroup/Chain can be used to create simplified branching logic or common sub - graph definitions, a little like functions or macros.

The common structure describes a series of tasks that should implement the exact same operational parameters, such as the same parameters, the same retries strategy, etc., but they may have different business logic. You can delegate this logic to a separate evolutionary operator instead of using the PythonOperator for your logic if it becomes important. At each stage, a task of different operators meant to call all tasks is established via the remaining set of operator services. In general, there is no consideration of renaming or saved naming of particular kinds of managers and, for the purpose of groups and their points, task settings apply only once.

6. Conclusion

Apache Airflow stands out as a robust platform for orchestrating ETL processes, providing unparalleled capabilities in automation, scheduling, and monitoring. By leveraging its Directed Acyclic Graph (DAG) structure, users can design complex workflows with ease, ensuring that tasks are executed in a precise and orderly manner. The integration with various external systems allows for seamless data movement and transformation, catering to diverse data engineering needs. Throughout this article, we have explored the fundamental architecture of Apache Airflow, highlighting its modular components that contribute to its flexibility and scalability. The detailed examination of task scheduling and execution mechanisms underscores the platform's ability to handle complex dependencies and large - scale data processing. Additionally, the discussion on monitoring and logging emphasizes the importance of transparency and control in maintaining reliable ETL pipelines.

Best practices for optimizing performance and ensuring reliability have been presented, offering practical guidance for data engineers to maximize the potential of Apache Airflow. The inclusion of real - world examples and case studies provides a concrete understanding of how these concepts are applied in practice, demonstrating the platform's efficacy in various scenarios. In summary, Apache Airflow is an indispensable tool for modern data engineering, enabling the automation and orchestration of intricate ETL workflows with efficiency and precision. Its comprehensive features and adaptability make it a preferred choice for organizations aiming to streamline their data processing operations. By adopting the strategies and insights discussed in this article, data engineers can harness the full power of Apache Airflow, driving enhanced performance and accuracy in their ETL endeavors.

References

[1] M. Zaharia et al., "Apache Spark: A Unified Engine for Big Data Processing, " Communications of the ACM, vol.59, no.11, pp.56 - 65, Nov.2016. doi: 10.1145/2934664.
[2] M. B. Juric, "Business Process Execution Language for Web Services BPEL and BPEL4WS 2nd Ed.: Business Process Management with Web Services, " Packt Publishing Ltd, 2006.
[3] J. Caserta and V. L. Henderson, "ETL: Extract, Transform, Load, " in The Data Warehouse ETL Toolkit, Wiley, 2004, pp.25 - 47
[4] A. B. Wilkinson, "Data Management and Transformation in the Cloud, " in Cloud Computing and Big Data: Technologies, Applications, and Security, Springer, 2014, pp.91 - 110.
[5] T. Lorentz, "Effective Data Pipeline Orchestration with Apache Airflow, " O'Reilly Media, 2020.
[6] M. Brown and S. Zhang, "Orchestration of Data Pipelines Using Apache Airflow, " *International Journal of Data Engineering*, vol.12, no.4, pp.45 - 53, 2019.
[7] K. Patel and R. Lee, "Automating Data Pipelines with Apache Airflow: Best Practices, " *IEEE Transactions on Big Data*, vol.8, no.1, pp.37 - 44, 2020.
[8] C. White, "Data Engineering with Apache Airflow, " *Journal of Cloud Computing*, vol.9, no.2, pp.13 - 20, 2020.