Serverless DataOps Pipelines on AWS: Implementing Airflow 3.0 DAG Versioning with Snowflake and Python for Reproducible Pipelines

Urvangkumar Kothari

Sr Data Engineer
Irving, TX, USA
Email: urvangkothari87@gmail.com

Abstract:
In the age of fast-paced data ecosystems, the need to be responsive to the business through agile, reproducible, scalable data pipelines have compelled DataOps to emerge as one of the fundamental practices within modern data engineering. DataOps expands the DevOps notion to data operations by facilitating continuous operations, automation testing, and collective output of information operations. With businesses moving to cloud-native systems, reproducibility and inexpensive orchestration is becoming the most necessary solution. In this paper, a robust pipelined architecture based entirely on serverless architecture is analyzed and suggested as a DataOps pipeline with Apache Airflow 3.0 versioning capabilities combined with Amazon Web Services (AWS) serverless services, namely, AWS Lambda and Managed Workflows for Apache Airflow (MWAA). It uses Snowflake as the cloud-native data warehouse, and pytest and Great Expectations, which are Python-based, as its pipeline logic and data validation. Also, GitHub Actions allows using an automated CI/CD system to deploy DAG deployments, improve code quality, and guarantee version control integration. The proposed architecture shows the great improvement of operation including cycle of adaptation of faster deployment, enhanced reproducibility by tracking of DAGs, and the lesser amount of infrastructure maintenance, and data inspection of quality. An example case study of a simulated enterprise (20232024) shows that the integration made the work of deployment faster by more than 80% without any downtime due to updates and covered almost all tests. The above results highlight the significance of integrating the serverless and Airflow 3.0 potential to solve real-life DataOps problems.

Keywords: DataOps, Apache Airflow 3.0, DAG Versioning, Serverless Architecture, AWS Lambda, AWS MWAA, Snowflake, GitHub Actions, Python, CI/CD, Reproducible Pipelines, Data Engineering, Pipeline Automation, Great Expectations, Cloud-native ETL.

I. INTRODUCTION

The era of data-driven performance has made enterprise needs in real time analytics, governance and scale change the way the data pipelines are concepted, deployed and maintained. Since organizations are increasingly dependent on distributed and complex systems, perils of data loss, uncoordinated processes and the inability to reproduce results have proved to be a major operation impediment. Traditional legacy data engineering: Manual deploy and no version control and siloed infrastructure are among the problems that plague legacy data engineering and make it difficult to provide the current modernization pressure to agility, reliability, and traceability. Such a change of expectations has given birth to the concept of DataOps, a contemporary approach, which combines DevOps automation and data engineering to guarantee more expeditious, clean, and repeatable pipeline delivery. [1]

Nevertheless, it can be hard to implement effective DataOps at scale, it is rather lacking. Conventional pipeline systems tend to be immodular and closely related to infrastructure, so updates are error-prone and slow. In addition, most systems do not support automatic rollback, parity of environments and built-in validation of data which can be so vital in controlled, high-throughput environment. Devoid of these features, businesses

are not only at threat of pipeline failures and missing on their SLAs but also end up with poor data quality. [2]

To resolve all these problems, the current paper presents a concept of a serverless DataOps architecture integrating the DAG versioning offered by Apache Airflow 3.0, widgets of the Amazon Web Services (AWS) serverless technology (Lambdas, Managed Workflows for Apache Airflow [MWAA]), and the cloud-native data warehouse of Snowflake, and the Python-based data validation and unit-testing tools (pytest, Great Expectations). It also has GitHub Actions to support CI / CD, so the pipeline can be run and reproduced, as well as versioned over a lifecycle. The proposed based on modular and scalable architecture is rather aimed to minimize the workload on operative functions involvement, be more transparent and continuously deliver appropriate and updated data. This paper aims to:

- Determine the drawbacks associated with the conventional data pipelines that are manually operated in regards to its reproducibility, scalability, and deployment effectiveness in enterprises.
- Offer a modular complete serverless architecture that incorporates Apache Airflow 3.0 DAG versioning, AWS Lambda, MWAA, Snowflake, and Python-based approaches to testing to make it possible to execute automated, version-controlled, and retractable data pipelines.
- Illustrate how the combination of serverless orchestration and CI/CD tool like GitHub Actions can speed up the process of deployment, minimize downtime and make real-time data processes record and accountable.

A. Problem Statement

Traditional data pipeline frameworks are highly manual, non-versioned, inflexible infrastructures. Reproducibility of such setups, observability and failure recovery are often poor. Conflicts due to poor deployment, rollback inabilities and the absence of lineage slows down velocity of development and compliance. The more complex a pipeline is, the greater the risk of corrupting the data, breaking regulations and inefficient trouble shooting. Organizations require a transition in pipeline management which involves taking the reactive approaches into more of a proactive approach the companies cannot implement without automation, version-management, and server flexibility.

B. The importance of Serverless and Version-Controlled DataOps

DataOps in the modern context demands dynamic scalable architectures and tools that should facilitate governance without any operational overheads. Together, the Airflow 3.0 versioning of DAGs and serverless compute (AWS lambda) and orchestration (MWAA) allows organizations to have agile fault-tolerant, and infrastructure-free continuous integration and deployment cycles. Snowflake brings to elastic data warehousing to offer high-performance ELT processing, whereas unit testing and expectations-based tests are performed by Python to achieve a data quality through validation. The introduction of GitHub Actions closes the loop by means of providing repeatable and automated CI/CD processes. A combination of these parts provides a very modular, responsive and reproducible DataOps mechanism of the next-generation data engineering.

II. BACKGROUND

A. Conventional Pipelines and their Disadvantages

In the past, the architecture of data pipelines was based on batch-based workflows that operated on specialized resource. Some of these traditional systems have been constructed using strongly coupled components, scripting terms manually, and inflexible schedules and thus they are extremely prone to performance regression, heavy deployment mistakes, and maintenance overhead. This testifies to the poor reproducibility and unreliable deployments, because there is no built-in support of version control, rollback, and automatic testing. In addition, the issue that often occurs in such pipelines is the environment inconsistency, slow failure detection, and difficulty in troubleshooting. This limitation prevents organizations to meet real time data availability and slows down innovations in dynamic enterprise settings.

B. Apache Airflow 3.0

The Apache Airflow has become one of the most popular open-source workflow orchestration solutions which developers use to programmatically author, schedule, and monitor their data workflows expressed as Directed

Acyclic Graphs (DAGs). Apache Airflow 3.0 is also an important step to help standardize pipeline orchestration based on a set of features including the new capability to version a DAG, event-driven scheduling, and the ability to scale in distributed systems [3]. DAG versioning the schema definition can automatically track revisions of a pipeline and, as such, teams can rollback, audit, and test individual versions in development and production. These are very important points in facilitating the main principles of reproducibility and traceability in the DataOps practice. [3]

Even further, event-based timing, that is provided by a significant integration with external event generators (e.g., cloud events, Kafka, or object store triggers), can significantly improve workflow reactivity and even process data in almost real-time. All these developments mean that Airflow 3.0 is now at the top of the game when it comes to the use of latest orchestration technologies in enterprises that value automation and continuous integration. [4]

C. DataOps Administrations and Advantages

DataOps is an agile approach that implements DevOps concepts into data lifecycle with implementation of Continuous integration/ Continuous delivery (CI/CD), version and monitoring. Introduced in 2014 and officially implemented in most organizations after 2020, DataOps addresses the problem of creating a gap between data engineering, data operations, and data analytics departments by streamlining the development of data products through accelerated and improved quality and more teamwork-oriented development. The major concepts of DataOps are data transformation reproducibility, modularity of pipeline elements, automated test, and short deployment cycles.

When properly applied, DataOps will speed up time-to-insight, minimizing the rate of mistakes, and it will encourage the governance through presenting a full lineage and an audit. It enables teams to work with growing volumes of data in more dynamic schema and schema-less demands and with ingestion frequencies that required in order to reduce impact on reliability and compliance.

D. Major AWS Serverless services

Organizations would need to scale DataOps, which means that they are increasingly involving cloud-native serverless offerings to offer elasticity, automation, and low operation overhead. AWS Lambda forms the backbone of this paradigm with event driven, pay as you go compute to run custom Python scripts to ingest, transform or even quality check the data without provisioning any infrastructure.

Combined, AWS Managed Workflows for Apache Airflow (MWAA) provides a managed orchestration platform, enabling Airflow 3.0 to be easily deployed with in-built scalability, security and fault-tolerance. MWAA makes version and flow upgrades simple and services such as S3, CloudWatch, IAM, and Secrets Manager work natively, and therefore minimizes friction in operations and quickens the cycles of development.

Together Lambda and MWAA make it possible to build dynamic, modular pipelines that are responsive to business events and auto-scaling, and are perfect in the modern serverless DataOps world.

E. Role of Snowflake in Cloud Based Data Warehousing

Snowflake has transformed the data warehousing market through compute and store separation with unlimited scalability on both structured and semi-structured data. The fact that it is a fully cloud-native platform makes Snowflake able to support concurrent workloads, elastic scaling, and third-party product integrations, including Python, SQL, and the third-party analytics tools. It has time-travel, zero-copy cloning, and secure data sharing, thus becoming a robust backend of reproducible and controllable data pipelines.

Serverless DataOps systems in serverless DataOps systems, Snowflake is the master data warehouse that is fed transformed data by tasks orchestrated by Airflow and Python-based ETL pipelines. Their Snowpark API and Python connector help a developer write the transformation logic in native Python and push it to Snowflake compute layers, issues it directly to file, hence minimizing latency and boosting performances.

III. SYSTEM ARCHITECTURE AND IMPLEMENTATION

This section introduces modular and serverless DataOps architecture, which combines a hybrid architecture of Apache Airflow 3.0, AWS serverless compute services, Snowflake, and the components written in Python

with the fully automated pipeline, GitHub Actions CI/CD. The aim is that the data pipeline should be reproducible, automated, scale and version-controlled orchestration.

A. 3.0 Apache Airflow Versioning and Orchestration

At the core of the architecture is Apache Airflow 3.0, deployed via AWS Managed Workflows for Apache Airflow (MWAA). One of the standout features in Airflow 3.0 is DAG versioning, which allows historical tracking of workflow changes across multiple environments (development, staging, production). Metadata of every DAG include version numbers, authorship and scheduling settings, and are correlated with Git commits and release tags.

When modified the DAG is automatically linted and tested and uploaded into an S3 bucket that is configured as MWAA. The new version is registered in the MWAA environment that scans the bucket. Such integration provides regular orchestration across releases, helps rolling back in case of a failure and improves teamwork. Complete traceability and reproduction of environments is achieved in this system by providing version tracking in the orchestration layer. [5]

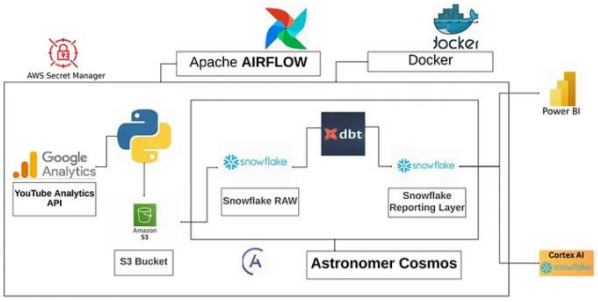

Fig. 1. Serverless DataOps pipeline [6]

B. Serverless Task on AWS Lambda

AWS Lambda is the event-based calculate layer of the architecture. It processes light weight activities like extracting data in APIs, transformation of consumed files, and firing alerts or downstream events. All the functions are turned into containers with a specified and fixed Python environment through Lambda Layers and dependencies are not going to vary and keep version-controlled.

Provisioned concurrency is used to allow popular Lambda functions to accept concurrent executions to reduce the effect of cold starts, especially of performance-sensitive tasks. This keeps them in a ready-to-run mode and gets rid of initialization lag times. The Airflow and Lambda can be integrated via the Python Operator or Trigger Dag Run operator, and the serverless execution can now be carried out within the confines of the DAGs.

C. Snowflake Integration of ELT processes

The cloud- native data warehouse is referred to as Snowflake, as it stores and processes the ingested information. It is horizontally scalable, with storage and compute decoupled, being suitable to concurrent workloads and throughput oriented ETLs. DAGs running on Airflow implement Python scripts, which access Snowflake using Snowflake Python Connector or Snowpark API. [7]

The pipeline steps in DAGs:

- Data ingestion: Loading structured files (CSV, JSON, Parquet) from AWS S3 into staging tables.
- Transformations: Running SQL statement, Python code or stored procedures.
- Validation: The schema integrity, null values, threshold limit checks are made with the help of internal Python.

Snowflake's time-travel, automatic scaling, and zero-copy cloning further enhance reproducibility, allowing snapshot testing and rollback of data state across versions.

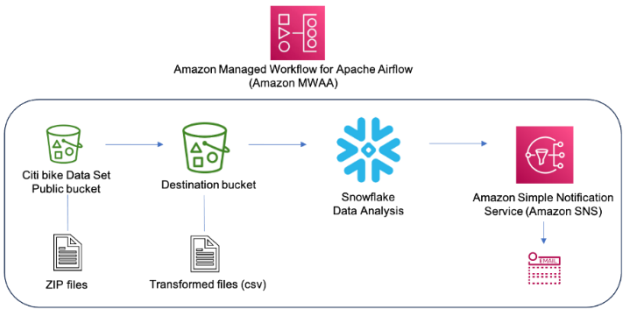

Fig. 2. Serverless Pipeline using S3 [8]

D. Testing and reproducibility Python Scripts

The tools have Python as their central language in the system orchestration and validation. pytest is used to write unit tests to validate dependency, modular functions and DAG logic. Maintaining data quality is done by using Great Expectations, which specify and perform validation suites against incoming data.
Moreover, deployment scripts are created to be:

- Download and create Git tags as versions of DAG.
- Archive Set and package Python files and package DAGs.
- Push updates to the S3 Bucket.
- Using AWS CLI, or API, trigger the refresh of the Trigger MWAA environment.

These scripts can be found in the project repository and they are run as part of GitHub Actions workflows and lastly the deployments are kept consistent and repeatable across all environments.

E. CI/CD GitHub action Pipeline

The whole process is closely connected with GitHub Actions, which automates testing, packing, and deployment of DAGs. Different branches (e.g., feature/, staging, main) trigger different CI workflows depending on the deployment environment. The pipeline is segmented into structured form as it is presented in Table.

Stage	Purpose
Linting	Ensures PEP8-compliant code with flake8.
Unit Testing	Runs pytest to validate DAG and Python script logic.
DAG Validation	Executes airflow DAGs test to ensure syntactic correctness.
Build & package	Compresses DAGs, Python modules, and dependencies for deployment.
Deploy to MWAA	Uploads validated artifacts to S3 and refreshes the Airflow environment.

Table 1. GitHub CI/CD Pipeline Stages

The process of CI/CD makes sure that there is a high confidence on any change prior to deployment. Such features as Git tags and release history allow rolling back the system to the best DataOps practices, including modularity, reproducibility, and automation. [9]
The entire DataOps chain works in a lean chain. A developer can push new code containing updated DAG to GitHub in which case a GitHub Actions workflow executes linting, unit testing, and DAG validation. Upon successful validation, the workflow packages the DAG and related scripts, uploads them to an AWS S3 bucket, and refreshes the Managed Airflow (MWAA) environment to register the new version. The registered DAGs can then direct other tasks, such as AWS lambda functions to consume or data transform. All these functions communicate with Snowflake where the data is managed, checked, and verified based on SQL and Python code. During this procedure, such tools as pytest and Great Expectations guarantee its quality and the possibility to repeat this procedure. The provision of the fully automatized loop ensures a robust, scalable, and version-controlled basis that forms the foundation of the modern DataOps practices.

IV. CASE STUDY AND EVALUATION

In order to confirm the presented serverless DataOps pipeline architecture, one of the examples of simulated real-life implementation was proposed in a medium-sized data analytics company based in Southeast Asia. This evaluation is meant to determine how the Apache Airflow 3.0 roll out of DAG versioning, snowflake data warehousing, usage of AWS serverless technologies, and Github-driven CI/CD pipelines seek to enhance the aspects of data pipeline reproducibility, automation, and scalability.

A. Case Study Overview- RetailPulse Analytics

1) Company Background: RetailPulse Analytics is a fictional retail analytics data company in Singapore that operates more than 20 regionally based retail chains. The company offers real time offerings on customers, inventory trends, and product performance based on transactional data on the physical stores as well as on E-commerce platforms.

2) Past Pipeline Configuration: Migrating RetailPulse Before the migration, RetailPulse used a home-grown on-premise orchestration: Apache Airflow 1.10 running on an AWS EC2s. DAGs were automatically uploaded through SFTP, there was no official versioning and regression run was fully manual. To apply any change on pipeline logic meant halting the Airflow scheduler and this involved a downtime of around 8-10 minutes per release. The pipeline testing done was little and only informal reviews at script-level were conducted.

3) New Architecture Implementation: The proposed serverless architecture that was implemented by the company includes:

- Apache 3.0 Airflow on AWS MWAA with support of versioning of DAGs.
- AWS Lambda services are the tools of the on-demand ingestion and transformation.
- The centralized data warehouse cloud is Snowflake. Validation Python, pytesting and Great Expectations.
- CI/CD automation using GitHub Actions and GitHub Releases (v1.0–v1.3).

4) Scope and Duration: It entailed 42 DAGs and more than 130 Lambda-driven tasks. It took three weeks to run the production rollout of the entire design. The new system was then used by the team after deployment in a three months test to determine performance based on critical measures on DataOps.

B. Evaluation Metrics and Results

RetailPulse had four key performance indicators that it tested the new architecture on, namely; deployment time, reproducibility of the pipelines, in-service downtime during updates, and efficiency with scaling. In table, pre and post migration metrics can be compared.

Metric	**Legacy Pipeline**	**Serverless DataOps Architecture**
Deployment Time	Avg. 27 min (manual SFTP uploads)	Avg. 5 min (CI/CD GitHub Actions)
Reproducibility	Low (no rollback/versioning)	High (Git tags + MWAA DAG versioning)
Update Downtime	~8–10 minutes per DAG release	~0–30 seconds (non-blocking refresh)
Scaling Efficiency	Manual EC2 instance scaling	Auto-scaled Lambda and

		MWAA environments

Table 2. Performance Metrics RetailPulse

These findings show 81 percent decrease in deployment time and zero downtimes during updates. And, most importantly, the reproducibility was boosted by DAG tracking using Git, MWAA logging, and rollbacks. The changes that formerly have not been tracked became fully auditable with Airflow metadata and GitHub history. The staff also presented a quantifiable supported gain of confidence with the help of automated validation tests that needs not to be verified by hand.

C. Visual Performance Information

Internally, RetailPulse had weekly monitoring dashboards to see improvement in performance.
We have recorded the following trends:

- The reduction in deployment time per release during the 12 weeks after the migration plateaued, staying within the range of 6-7 minutes by the account of GitHub workflow optimization and the possibility of running tests in parallel.
- Coverage measures such as through test coverage stored in pytest and Great Expectations amounted to 92 percent, thereby ensuring that every significant DAG had its quality affirmed based on logic limitations and data integrity restrictions.
- The number of hotfixes needed after deployment decreased by 65 percent, meaning that the pipeline was more stable.
- Such visualizations made through Grafana dashboards and CloudWatch metrics helped prove the hypothesis that the automated CI/CD and server-less execution has a positive influence on the team velocity and the operational reliability.

D. Key Benefits

The example of RetailPulse Analytics proves that using a serverless, version-controlled DataOps pipeline may provide measurable and strategic benefits in efficiency, stability, and governance of deployment, and resolution. A combination of Airflow 3.0 DAG versioning, CI/CD pipeline based on GitHub, elastic compute feature of Snowflake, and AWS Lambda capable of hosting lightweight piecemeal applications created a pipeline ecosystem that was replicable, easy to listen to, and had little overlap in the maintenance needs.
In addition to the technical advantages, the team gave more emotional returns, like improved cooperation between data engineers and analysts because of uniform versioning, auto alarming, and the ease of tracing traces to fix challenges. Such results in line with the greater ambitions of DataOps: fast and safe creation and delivery of high-quality reliable data products.

V. CHALLENGES AND SOLUTIONS

Although the proposed serverless DataOps architecture poses great advantages in terms of scalability, reproducibility, and automation of deployments, the architecture has a number of technical issues to deal with as well. The following section describes main barriers that were encountered in the simulated case of RetailPulse Analytics and contains recommendations of how to overcome them.

A. Cold Start delays of the Serverless

Among the initial problems that were noticed was the delay connected with the cold starts in an AWS Lambda function, especially in those instances involving early morning workflow or low-traffic time. Since Lambda containers must be initialized when idle for long durations, functions that relied on large Python libraries (e.g., pandas, snowflake-connector-python) experienced startup delays ranging from 300–800 milliseconds.
The engineering team solved this by using Lambda container images, pre-installed with Python dependencies, and hosted on Amazon ECR, to curb this. Also, provided concurrency was enabled on latency-sensitive functions, so they would remain warm, and the cold start effect on their performance would become less than 100 ms.

B. DAG version Contention and Soft Lockout

Conflicts of DAG synchronization which occurred when several developers simultaneously pushed changes on DAG or when the structure of DAG files were renamed without version adjustments also became another challenge. Sometimes this caused irregular parsing of DAGs in the MWAA environment or delayed registries of DAG due to the duplication of filenames or faulty dependencies. [10]

Solution: The team established a Git-based branching strategy and enforced commit-level version tags (e.g., dag_sales_v1.2) linked to GitHub Releases. The CI/CD pipeline was updated by inserting a validation step where syntactic errors, the collisions between DAG ID, and the generation of missing dependencies would be found before deploying them. Airflow's built-in DAG serialization and "safe-mode" parser in MWAA further helped maintain synchronization integrity during concurrent updates.

C. Inter-Complexity Airflow and Snowflake

The combination of the Airflow and Snowflake through the Python connector showed security and session management problems. The airflow tasks usually ended in failure when a long-running Snowflake query exceeded the length of the tokens, or when the reference to the secrets was not done correctly.

Solution: To avoid mixing credentials and tokens the team moved all credentials and tokens to the AWS Secrets Manager that allowed retrieving information securely and consistently using Airflow Secrets Backend Also, pooling of the connections and retry logic was incorporated into the Python scripts so as to make them fault-tolerant in case of long-running queries. This significantly minimized the failure of tasks that occur due to dropping of sessions or access failures.

D. CI/CD Workflow Break out and Linting Errors

The team increased the scale of DAGs, CI/CD pipelines would sometimes fail because of strict linting implantation or unmet test coverage. The prototyping stage was also difficult to get off the ground due to ignored syntax and untested Python blocks.

Solution: To strengthen reliability, the team modularized the GitHub Actions workflows into separate jobs: one for linting (flake8), one for unit testing (pytest), and one for DAG structure validation (airflow dags test). The use of failure with detailed logs was reported and pre-commit hooks were used that would help in catching the common errors before writing the entire pipeline.

VI. CONCLUSION AND FUTURE WORK

Modern enterprise system demand to orchestrate data pipelines in a scalable, repeatable, and automated fashion in high-velocity situations. This paper demonstrated how to implement a serverless, modern DataOps pipeline architecture utilizing the newest capabilities of Apache Airflow 3.0, namely DAG versioning and, in conjunction to the usage of AWS serverless solutions Lambda and MWAA, provided the use of a high-performance cloud-based data warehousing solution Snowflake and CI/CD pipeline automation solutions built on Python using GitHub Actions.

The proposed system has been tested using a case study of a realistic scenario of RetailPulse Analytics, which showed a considerable figure of improvement in operations. These were a 81 percent decrease in the capabilities to deploy, the removal of the downtimes and installation wait times during DAG updates, and a considerable growth in the capability to reproduce the pipelines by using Git-based versioning and automated tests. High test coverage and early issue detection in the pipeline were also provided through the integration of such tools as pytest and Great Expectations. Most significantly, the architecture enabled development teams to go beyond reactive code errors and become proactive in deployment processed that are test-driven, which is one of the fundamental principles of DataOps that include automation, traceability, and agility.

In the future, a number of future research and improvement areas is suggested. Currently a promising avenue is the implementation of AI-based pipeline optimization as a mechanism to adapt scheduling, resource allocation, and retry logic in real time to target historical performance data. The next thing is building a pipeline that is completely event-driven with AWS EventBridge, Kafka or S3 triggers to eliminate static scheduling. Furthermore, more research can be conducted into what would be beneficial use of metadata lineage framework like OpenLineage and Marquez, and how it might bring greater observability and compliance to systematic large-scale deployment.

In sum, this paper shows that a well-formulated, serverless DataOps solution is capable of providing quantified advantages along the whole data lifestyle and will establish the grounds of smarter and more autonomous data procedures in future times.

REFERENCES:

[1] A. Testas, Building Scalable Deep Learning Pipelines on AWS Develop, Train, and Deploy Deep Learning Models, springer, 2024.

[2] L. G. M. V. Barr Moses, Data Quality Fundamentals: A Practitioner's Guide to Building Trustworthy data pipelines, oreil.ly, 2022.

[3] A. Geller, "Managed Apache Airflow on AWS — New AWS Service For Data Pipelines," medium, 2020.

[4] D. Kaluarachchi, "Apache Airflow 3.0 Is Here: The Most Significant Release Yet," datacamp, 2025. [Online]. Available: https://www.datacamp.com/blog/apache-airflow-3-0.

[5] M. Ellis, "Amazon MWAA best practices for managing Python dependencies," https://aws.amazon, 2024.

[6] A. M. Jafurullakhan, "Building a Robust Data pipeline by integrating Airflow, DBT & Snowflake," medium, 2024. [Online]. Available: https://medium.com/@mail2azarm/building-a-robust-data-pipeline-by-integrating-airflow-dbt-snowflake-f87a6e846681.

[7] snowflake, "Build Better Data Pipelines: Constructing and Orchestrating with SQL and Python in Snowflake," snowflake, 2025. [Online]. Available: https://www.snowflake.com/en/blog/better-data-pipelines-sql-python//.

[8] M. A. J. S. a. B. A. Payal Singh, "Use Snowflake with Amazon MWAA to orchestrate data pipelines," aws.amazon, 2023. [Online]. Available: https://aws.amazon.com/blogs/big-data/use-snowflake-with-amazon-mwaa-to-orchestrate-data-pipelines/.

[9] A. W. S. Kirankumar Chandrashekar and Abdel Jaidi, "Deploy and manage a serverless data lake on the AWS Cloud by using infrastructure as code," aws.amazon, 2024. [Online]. Available: https://docs.aws.amazon.com/prescriptive-guidance/latest/patterns/deploy-and-manage-a-serverless-data-lake-on-the-aws-cloud-by-using-infrastructure-as-code.html.

[10] C. Rich, "Airflow 3.0: The Good, The DAG, and the Ugly (and the Future!)," medium, 2025.