The American Journal of Interdisciplinary Innovations and Research
ISSN 2642-7478 Volume 08 - 2026

# A Review Of Airflow-Based And Cloud-Native Approaches For Workflow Orchestration In Devops

**Basheer Ahmedu Basha**
Principal Build and Release Engineer
IEEE Member,USA
basheer.ahmedu@gmail.com

Received: 24 Aug 2026 | Received Revised Version: 12 Sep 2026 | Accepted: 22 Sep 2026 | Published: 15 Oct 2026

Volume 08 Issue 10 2026 | DOI:- 10.37547/tajiir/Volume08Issue10-06

## Abstract

*Workflow orchestration has become an essential component of modern DevOps environments to automate, schedule, monitor, and coordinate complex software and data workflows. To create scalable and robust operations, use cloud-native techniques and CI/CD. Apache Airflow provides a DAG-based solution to task dependency, scheduling, monitoring and distributed process execution. This analysis examines cloud-native and Airflow-based workflow orchestration methods with a focus on their design, attributes, pros, cons, scalability, fault tolerance, resource consumption and complexity. To assess the advancement of workflow orchestration and the role Airflow plays in modern DevOps workflows, researchers examine past research. Researchers review past studies to gauge the evolution of workflow orchestration and the role that Airflow plays in modern DevOps pipelines. The review also looks into new strategies for cloud-native security, automated scaling, multi-cloud orchestration, machine learning-based failure prediction, and observability. The study indicates that cloud-native orchestration provides greater scalability, automation, resilience, and flexibility in infrastructure, while Airflow provides strong dependency management capability, flexibility in scheduling, monitoring, and extensibility. There are still challenges in real-time processing, complexity in operations, cost management, multi-cloud environments, and adaptive orchestration. The highlighted research gaps highlight the need for intelligent and comprehensive orchestration frameworks that integrate cloud-native technologies, predictive analytics, and workflow management. Overall, results indicate that the cloud-native orchestration framework along with the DAG-based workflow management can provide a scalable and flexible foundation for modern workflow automation in DevOps.*

Keywords: Cloud Computing, Apache Airflow, Cloud-Native Orchestration, Workflow, DevOps, Kubernetes, DAG, CI/CD, Fault Tolerance.

**© 2026** Basheer Ahmedu Basha. This work is licensed under a Creative Commons Attribution 4.0 International License (CC BY 4.0). The authors retain copyright and allow others to share, adapt, or redistribute the work with proper attribution.

**Cite This Article:** Basha, B. A. (2026). A Review Of Airflow-Based And Cloud-Native Approaches For Workflow Orchestration In Devops . The American Journal of Interdisciplinary Innovations and Research, 8(10), 55–64. Retrieved from https://theamericanjournals.com/index.php/tajiir/article/view/8527

## 1. Introduction

In order to provide software more quickly, reliably, and automatically, modern software development is depending more and more on DevOps techniques. The incorporation of cloud computing, infrastructure automation, monitoring, continuous integration, and continuous delivery (CI/CD) has greatly increased the complexity of software and data operations. The spread of applications over microservices, containers, hybrid clouds, and multi-cloud settings makes manual task coordination ineffective and prone to errors [1]. By automatically coordinating tasks, maintaining dependencies, scheduling executions, keeping track of workflow stages, and managing failures across diverse

computing systems, workflow orchestration solves this problem [2]. A popular open-source platform for programmatically creating, scheduling, and tracking processes is Apache Airflow. Directed Acyclic Graphs (DAGs), in which distinct activities are linked by specified dependencies, are used to depict workflows. Organizations can automate complex and multi-phase workflows comprising data processing, testing, deployment, machine learning, and operational tasks thanks to this framework.

Additionally, Airflow offers a wide range of integrations with databases, cloud services, containers, Kubernetes, and other DevOps technologies [3]. It is appropriate for handling complicated processes because of its scheduling features, retry methods, logging, monitoring, and flexibility [4]. However, for real-time and highly event-driven workloads, Airflow may create constraints, operational cost, and setup complexity. Workflow orchestration has been revolutionized concurrently by cloud-native methodologies using technologies like Kubernetes, containerization, serverless computing, Infrastructure as Code (IaC), microservices, and cloud-based CI/CD systems. These methods provide a strong emphasis on resource efficiency, automation, scalability, robustness, and portability. In particular, Kubernetes makes it possible for containerized workloads to be automatically deployed, scaled, managed, and recovered. As a result, cloud-native orchestration can handle dispersed, dynamic settings where workloads are always changing based on demand. Workflow automation is supported by both Airflow-based and cloud-native techniques, although they differ in terms of scalability, execution, resource usage, monitoring, fault tolerance, and operational complexity. In order to identify the main obstacles in contemporary workflow orchestration, this research compares the designs, features, advantages, constraints, scalability, dependability, deployment efficiency, and automation capabilities of both DevOps techniques.

**Paper structure**

The remainder of this review paper is organized as follows. Section II provides an overview of Apache Airflow and cloud-native workflow orchestration. Section III discusses the evolution and role of workflow orchestration in DevOps. Section IV presents the literature review and comparative analysis of existing studies. Finally, Section V concludes the review by summarizing the major findings and key observations.

**Overview of Airflow and Cloud-Native Orchestration**

The Apache Airflow platform is a workflow management tool that enables the orchestration, scheduling, and monitoring of intricate data pipelines. Data pipeline orchestration and scheduling find their home in this tool. The main task of orchestrating data pipelines is to organize and coordinate the transfer of data from different sources. Data science, ML models, and business intelligence apps that back up big data use cases get data sets from data pipelines inside an Airflow that are prepared for consumption. These parts are required for an Airflow installation:

- **Scheduler:** It is in charge of keeping track of tasks and DAGs and starts new instances of tasks when their requirements are satisfied.
- **Webserver:** Provides a graphical user interface for inspecting, debugging, and triggering tasks' and DAGs' behavior.
- **DAG file:** The fundamental idea behind Airflow is a directed acyclic graph, or DAG. It specifies the proper operation of dependencies and linkages and gathers tasks together.
- **Metadata database:** The various Airflow components save the current status of processes and tasks in this database [5].

In general, data processing operations are orchestrated using Airflow. Airflow makes it easy to define DAGs and coordinate their execution across distributed systems, allowing for the construction of sophisticated data pipelines.

**Cloud-Native Workflow Orchestration**

Automation of complicated IT activities across cloud environments is what cloud orchestration is all about. By coordinating the coordinated effort of various automated processes and resources in accordance with established protocols, the orchestrator plays the role of a conductor. Through careful management of component interactions and dependencies, this technique goes beyond basic automation to provide coherent end-to-end processes that achieve corporate goals at a higher level. Orchestration platforms also empower teams to outline their desired outcomes instead of prescribing the specific steps to implement that are most likely to meet those goals. The platform can then work out the best way to accomplish this based on the current conditions and the available resources [6]. Cloud orchestration can be conceptualized as having three methods: declarative state definitions, workflow engines, and feedback systems. The declarative approach is to identify the desired end states instead of a step-by-step list of procedures. These specifications are translated by the workflow engines into sequences of executable instructions, including variables, condition control and exception handling. Feedback systems track the execution progress and the conditions of the system, which allows the system to respond adaptively to changing

conditions [7]. This high-level coordination allows organizations to deploy complex operations such as a blue-green deployment or, for example, self-healing infrastructure, which would be difficult to achieve manually.

**Understanding Cloud-Native Development**

The term "cloud-native application development" describes the steps taken to build and run apps that make complete advantage of cloud computing.

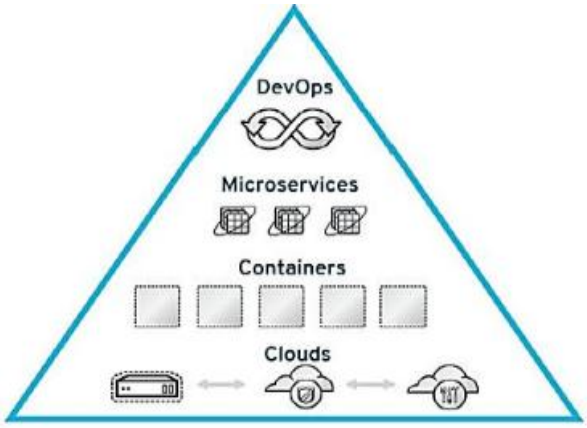



**Fig. 1. A Pyramid of Modern Cloud-Native Applications**

Cloud native development refers to the process of building apps with the intention of being used in the cloud from the beginning [5]. Container orchestrators, microservices, and other cloud technologies are used to construct cloud native applications. Figure 1 shows the following applications that were created with the intention of being used in the cloud:

**Microservices Architecture**

This form of architecture establishes loose links between several tiny services that run independently within larger applications. With the system's division into separate services, each can handle its own development, deployment, and scaling requirements. These services are tailored to meet the needs of certain business processes [8].

**Containerization**

The deployment element is present at the container level. Standardized portable software units are generated by Docker and comparable tools. These units consist of complete applications and their dependencies, including libraries, runtime, and configuration files. The system ensures that the testing and production environments, as well as developer workstations, are consistent, thereby preventing the occurrence of defects that are caused by environmental differences [9].

**Serverless Computing**

Serverless computing, also known as FaaS, is a promising new paradigm for deploying applications to the cloud, following the present trend in business application architectures toward microservices and containers [10]. By eliminating the need to manage servers or expand resources, serverless computing frees developers to focus on code creation. Among the many benefits of this approach are its scalability, cost-effectiveness, and reduced operational overhead [11]. An advantage of serverless architecture is its scalability.

**DevOps**

DevOps is a way of doing software development that incorporates lean and agile ideas. By using this strategy, the operations and development teams are motivated to collaborate in order to consistently provide top-notch software. Through the promotion of tight collaboration among developers, testers, and operators, the activities that comprise DevOps—i.e., continuous planning, integration, deployment, testing, and monitoring—strive to enable the fast and dependable development, testing, and deployment of software updates [12].

**Key Features of Cloud-Native Applications**

The main characteristics and functions of cloud-native apps are highlighted in Figure 2, and this section explains those elements.

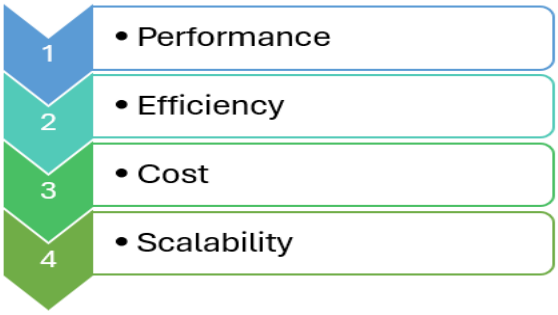



**Fig. 2. Features of Cloud-Native Applications**

- **Performance:** The public cloud's inherent capabilities, which might be more efficient than alternatives that aren't native to the cloud. For example, you may be in charge of an input/output system that can automatically scale and balance the load.
- **Efficiency**: Apps built for the cloud should leverage cloud-native features and APIs to optimize resource utilization. That leads to better performance or lower operational costs.
- **Cost:** The running cost of more efficient apps is often lower. Because cloud service providers charge you on a regular basis based on your usage, optimizing your resource use might help you save money.
- **Scalability:** Develop your apps with immediate access to the platform's autoscaling and load-balancing capabilities through the native cloud APIs [13].

**Advantages of Apache Airflow**

A popular open-source platform for creating, scheduling, and monitoring processes programmatically is Apache Airflow [14]. In DevOps, it is useful for coordinating complex, multi-step pipelines and managing task dependencies. Advantages of Airflow:

- **Workflow Automation:** Airflow automates complex workflows and reduces manual intervention in repetitive DevOps tasks.
- **Dependency Management:** DAGs clearly define relationships between tasks, ensuring that tasks execute in the required sequence.
- **Flexible Scheduling:** Airflow can handle scheduled, event-based and manually triggered workflows to meet various operational needs.
- **Extensibility:** Its large collection of operators and integrations allows Airflow to connect with databases, cloud platforms, Kubernetes, APIs, and other DevOps tools.
- **Monitoring and Logging:** The web interface provides task-status visualization, execution history, logs, and failure information, supporting effective workflow monitoring.
- **Scalability:** Airflow can be scaled by running tasks on multiple workers, which can be used for large and complex workflows.
- **Open-Source Ecosystem:** The Airflow open-source community is strong and can be tailored to meet the needs of an organization.

**Limitations of Apache Airflow**

One such limitation that should be taken into account when choosing a workflow orchestration solution is the Apache Airflow architecture.

- **Complex Setup:** Deploying and configuring Airflow can be difficult, especially for large production environments and for beginners.
- **Resource Overhead:** Scheduler, metadata database, workers and other parts can use up a significant amount of computing resources.
- **Not Ideal for Real-Time Workflows:** Airflow is not optimized for continuous, low-latency processing, but rather for batch-oriented workflows and pipelines that are scheduled to run.
- **DAG Management Complexity:** Complex workflows can lead to complex DAGs that are difficult to maintain and troubleshoot.
- **Operational Maintenance:** Production deployments involve monitoring, upgrading, database management, security setup, and infrastructure maintenance.
- **Limited Native Event Streaming:** Airflow is not primarily designed as a real-time event-stream processing platform and may require additional technologies for such workloads.

- **Cloud Dependency Challenges:** Although Airflow integrates with major cloud platforms, managing a self-hosted deployment across multi-cloud environments can increase operational complexity [15].

### Evolution of Workflow Orchestration in DevOps

Workflow orchestration is the glue that makes it possible to link people's goals and system activities. Manual coordination is removed and work flows predictably and efficiently. This is particularly important in today's DevOps landscape where predictable DevOps pipelines and integrated tooling are essential to keeping the velocity with which development teams can gain a competitive edge. Workflow orchestration isn't just a technical solution; it's necessary for accelerating speed, upholding reliability, and boosting the overall developer experience..

### Role of Apache Airflow in DevOps

Apache Airflow is an integral part of DevOps for automating, scheduling, monitoring and managing complex workflows. It models workflows as Directed Acyclic Graphs (DAGs) with each task in the workflow linked to other tasks in order of their dependencies. This enables DevOps teams to manage and coordinate activities like data processing, automated testing, application deployment, infrastructure operations, and model training within a single workflow. Airflow can be used to schedule tasks automatically to run on a schedule or when a certain condition is met. It has a monitoring interface that makes it easy for developers and operations teams to monitor the status of a task, detect failures, and analyze execution logs. Airflow also has the ability to support multiple executors and deployment environments, making it suitable for scalable and distributed workloads [16]. Airflow can be integrated with various tools, including Docker, Kubernetes, Git, CI/CD systems and cloud platforms, in DevOps pipelines. This integration increases workflow consistency and minimizes manual work. Additionally, mechanisms for retries, dependency management and system failure handling ensure the reliability of automated processes. As mentioned in the Figure 3, Key Roles of Airflow in DevOps are:

- **Workflow Automation:** Automates multi-step development and operational workflows.
- **Task Scheduling:** Executes tasks according to predefined schedules or triggers.
- **Dependency Management:** Ensures tasks execute in the correct order.
- **CI/CD Integration:** Supports automated testing, deployment, and release workflows.
- **Monitoring:** Provides centralized monitoring, logs, and task-status tracking.
- **Failure Handling:** Supports retries and failure notifications.
- **Scalability:** Enables distributed execution of workflows.
- **Cloud Integration:** Can coordinate workflows across cloud and containerized environments.




**Fig. 3. Roles of Airflow in DevOps**

### DAG-Based Workflow Orchestration

Data orchestration involves collecting and managing end-to-end data pipelines in a centralized way. It allows a single task (such as extract, transform or load) to be chained together into a series of logical steps. These relationships are represented by a Directed Acyclic Graph (DAG). A DAG is a graph in which the nodes are tasks, and the directed edges are dependencies that indicate which tasks must be finished

before others are started. This is the fundamental way that orchestration tools control and monitor data pipelines, without requiring you to deal with the complexity of managing dependencies in application code. The aim is to establish a dependable, repeatable and visible data workflow automation system. Effective data orchestration platforms serve as the "air traffic controller" for data pipelines (see Figure 4), delivering a number of critical features to help them run safely at scale.

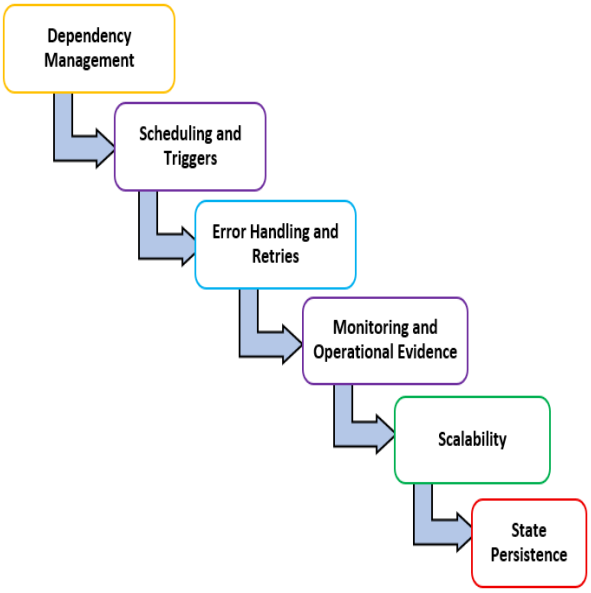



**Fig. 4. Data pipelines for DAG-Based Workflow**

- **Dependency Management:** Ensures that tasks in a DAG are executed in a proper sequence. A strong tool can prove its dependency semantics, which explains what it does when the upstream has failed (such as skip, retry or branch), without causing ambiguity.
- **Scheduling and Triggers:** Schedules and triggers tasks. This means traditional time-based scheduling (such as every hour) and event-based triggers (such as a new file) or an API callback, or the completion of an upstream workflow.
- **Error Handling and Retries:** Handles errors gracefully. This includes configurable retry mechanisms (typically exponential backoff for temporary problems), and checkpoints that enable "recover" capability without running an entire pipeline, just a single click.
- **Monitoring and Operational Evidence:** Includes visibility of the execution of workflows. This includes operational dashboards that display run statuses, task-level logs, performance data (such as step durations and retries) and data lineage to rapidly trace failures to actionable alerts.
- **Scalability:** Provides protection for the data and system. This is frequently done by scaling workers horizontally with tools such as Apache Airflow's distributed executors (such as Celery or Kubernetes Executor).
- **State Persistence:** Stores sufficient state for repeatable retries and backfills, and to survive worker restarts. A key aspect is idempotency, which means that executing a previously executed job window again returns the same target state, thus avoiding data duplication.

## Literature Review

This section covers workflow orchestration in a DevOps context, including Apache Airflow, Kubernetes, cloud-native architectures, IaC, CI/CD and ML-based approaches. The studies showcase advances in automation, scalability, reliability, resource efficiency, and deployment efficiency. The selected studies are summarized in Table I and form the basis of comparing Airflow-based and cloud-native orchestration approaches.

Avirneni and Sankiti, (2026) introduce a single multicloud orchestration approach that integrates AWS Managed Workflows for Apache Airflow. The architecture relies on the metadata model of Iceberg to ensure schema alignment and support seamless access to Iceberg data across engines, and Airflow provides cloud synchronization and incremental ingestion of data. Experimental evaluation using an 800 TB

synthetic retail dataset demonstrates a 58% reduction in pipeline latency, improved storage efficiency, and elimination of schema-related downtime. The results show that integrating open table formats with cloud native orchestration provides a scalable and cost-optimized foundation for enterprise multi-cloud data engineering [17].

Sanghi *et al.*, (2025) provide an all-inclusive framework for building systems from a financial institution's point of view, utilizing frameworks such as DevSecOps, policy-as-code, Zero Trust, and infrastructure-as-code to create safe, scalable, and compliant systems. Providing a multi-layered model and returning to 10 foundational research contributions. There were notable shifts in risk avoidance, policy adherence, and deployment frequency as a result of the tests. Trusted, cloud-native innovation is made possible by the paradigm in a data-heavy, heavily regulated business sector [18].

Poojary, Kamble and Mamatha, (2025) offer a fresh perspective that equips groups with DevOps maturity, observability, fault tolerance, security, AI-driven optimization, multi-cloud strategies, carbon-aware scheduling, regulatory compliance, cultural transformation, and automated CI/CD pipelines. We achieve scalability, robustness, and efficiency by exploiting Kubernetes' pods, services, and autoscaling. Using clear architectural insights, real-world e-commerce and IoT case studies, and unique metrics, we address problems such as network performance, security, cost, sustainability, compliance, and team collaboration to give a practical blueprint for modern DevOps practitioners [19].

Lakkireddy, (2025) introduces a scalable and modular framework, HyScaleFlow, combining real-time stream processing and machine learning–based orchestration. One of the major innovations is the FlowGuard module that provides node failure prediction and resource load forecasting using XGBoost models (classifier and regressor) and Prometheus-exported telemetry metrics. The NYC Taxi Trip dataset (more than 1.1 billion records) was evaluated on a hybrid cloud testbed, which includes both Spark at the edge and Flink in the cloud, managed through Docker/Kubernetes. The results indicate that HyScaleFlow allows for a 16.8% better completion rate for DAGs, a > 60% lower task retry rate, and a < 40% higher fault recovery time [20].

Oyekunle Claudius Oyeniran *et al.*, (2024) discuss the essential elements of cloud-native ecosystems, such as container orchestration platforms such as Kubernetes, which are used to manage and scale containerized applications automatically. This review highlights the importance of cloud-native technology for software engineers to create scalable and resilient applications. In this rapidly changing and competitive environment, organizations can leverage cloud-native principles and the right tools and practices to successfully address the demands of modern software development [21].

Supit *et al.*, (2023) propose the implementation of IaC, VCS, and CI/CD. The research results show that the application of a CNA culture and DevOps can shorten the time needed for the provisioning of the infrastructure and the deployment of the application, minimize human errors, ensure consistency of the documentation, and improve the overall reliability of the system in the company environment. The results of this study can increase the knowledge base about the synergy between CNA and DevOps. It can also be a factor to take into account for other organizations that would like to implement CNA and DevOps at the same time in their organization [22].

Harika *et al.*, (2023) provide an in-depth look at DevOps practices and cloud-native principles, examining how they can be used to ensure smooth scalability and resilience. The complexities surrounding these concepts are unraveled, providing actionable insights for organizations aiming to optimize their software delivery processes. While ranging from automating deployment pipelines to utilizing containerization technologies, uncovering brand-new methods can change the development lifecycle [23].

Paruchuri, (2022) analyzes Airflow deployment trends on Kubernetes and compares them to those of other orchestration platforms. While the adoption of GitSync-based deployment did shorten the deploy cycle from three to five days when using manual procedures to less than five minutes when using running DAG, the increase in productivity was more than offset by operational complexity manifesting as a small set of recurrent failure modes [24].

**TABLE I. Comparative Analysis of Existing Studies on Workflow in DevOps and cloud native**

| Author | Objectives | Benefits | Key Findings | Recommendations |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| Avirneni and Sankiti (2026) | To develop a unified multi-cloud orchestration framework integrating AWS Managed Workflows for Apache Airflow and Apache Iceberg for interoperable data management. | Improved pipeline scalability, storage efficiency, schema consistency, and multi-cloud interoperability. | Achieved 58% reduction in pipeline latency, improved storage efficiency, and eliminated schema-related downtime using an 800 TB synthetic retail dataset. | Integrate Airflow with open table formats such as Iceberg for scalable and cost-efficient multi-cloud data pipelines. |
| Sanghi et al. (2025) | To develop a secure and compliant cloud-native framework for financial institutions using DevSecOps, Zero Trust, policy-as-code, and IaC. | Enhanced security, compliance, deployment speed, data sovereignty, and operational efficiency. | The proposed multi-layered model improved deployment frequency, risk avoidance, and policy adherence. | Adopt DevSecOps, Zero Trust, IaC, and policy-as-code for secure orchestration in highly regulated environments. |
| Poojary, Kamble and Mamatha (2025) | To investigate modern DevOps practices involving CI/CD, IaC, Kubernetes, observability, fault tolerance, AI optimization, multi-cloud strategies, and sustainability. | Increased scalability, resilience, automation, resource efficiency, and operational sustainability. | Kubernetes-based orchestration supports scalable and resilient applications while addressing security, cost, performance, compliance, and sustainability challenges. | Combine Kubernetes orchestration with observability, AI-based optimization, automation, and carbon-aware scheduling. |
| Lakkireddy (2025) | To develop HyScaleFlow, a modular orchestration framework using ML-based failure prediction and resource-load forecasting for stream-processing workflows. | Higher reliability, reduced retries and recovery time, improved resource utilization, and lower cloud cost. | DAG completion improved by 16.8%, task retries decreased by over 60%, recovery time improved by up to 40%, and cloud execution cost decreased by 19.5%. | Integrate predictive ML models with workflow orchestration to enable proactive scaling, task routing, and fault recovery. |
| Oyekunle Claudius Oyeniran et al. (2024) | To review cloud-native ecosystem components and practices for developing scalable and resilient applications. | Improved scalability, resilience, service communication, monitoring, and fault tolerance. | Kubernetes, service meshes, distributed tracing, circuit breaking, and chaos engineering are important for reliable cloud-native systems. | Adopt cloud-native orchestration together with observability, service meshes, tracing, and resilience engineering practices. |
| Supit et al. (2023) | To examine the integration of Infrastructure as Code, Version Control Systems, and CI/CD within cloud-native architecture and DevOps. | Faster infrastructure provisioning and deployment, fewer human errors, improved documentation, and higher reliability. | CNA and DevOps reduced infrastructure and application deployment time while improving consistency and system reliability. | Integrate IaC, VCS, and CI/CD to strengthen automation and reliability in cloud-native DevOps environments. |
| Harika et al. (2023) | To examine DevOps and cloud-native practices for improving software scalability, resilience, and delivery processes. | Improved deployment automation, scalability, resilience, and software delivery efficiency. | Containerization and automated deployment pipelines can significantly transform and optimize software development and delivery processes. | Use containerization and automated CI/CD pipelines to improve scalability and streamline software delivery. |
| Paruchuri (2022) | To examine Airflow deployment patterns on Kubernetes and compare them | Faster deployment cycles, improved maintainability, and | GitSync reduced deployment time from 3–5 days to less than 5 | Use Kubernetes-based Airflow deployment with automated GitSync while |

| | | | | |
|---|---|---|---|---|
| | with alternative orchestration approaches. | automated DAG deployment. | minutes, although operational complexity introduced recurring failure modes. | implementing stronger operational monitoring and failure management. |

**Research gap:** In order to automate workflows and DevOps operations, previous studies have shown that technologies like Kubernetes, IaC, CI/CD, and cloud-native perform well. However, rather than offering a thorough comparison of Airflow-based versus cloud-native orchestration techniques, the majority of studies concentrate on certain platforms or particular application situations. Their relative performance, scalability, fault tolerance, resource usage, cost-effectiveness, and operational complexity have received minimal attention. Furthermore, there is still much to learn about integrating machine learning for predictive and adaptive workflow orchestration. Additionally, a lot of studies use platform-specific conditions or artificial datasets. As a result, a thorough analysis is necessary to determine the advantages, disadvantages, difficulties, and potential future paths of DevOps workflow orchestration.

## Conclusion and Future Work

In today's DevOps era, workflow orchestration is a key component in automating, coordinating, and monitoring complex software and data workflows. This review explored Apache Airflow and cloud-native orchestration methods and their architectures, capabilities, benefits and limitations. The results indicate that Airflow is effective in scheduling, dependency management, monitoring, retry, and integration with various DevOps and cloud technologies. The benefits of Kubernetes, containers, serverless, IaC, and CI/CD are resource flexibility, portability, resilience, and scalability. The studies reviewed also show that integration with cloud-native technologies can enhance the efficiency of deployment, scalability of the workflow, fault recovery and interoperability with multiple clouds. Airflow can be a heavy operational burden and is not ideal for real-time workloads, and cloud-native environments may come with infrastructure and management complexity. The results suggest a hybrid Airflow and cloud-native orchestration model offers a flexible, scalable, and robust platform for automated modern DevOps workflows.

## References

1. K. Petrakis et al., “Enhancing DevOps Practices in the IoT–Edge–Cloud Continuum: Architecture, Integration, and Software Orchestration Demonstrated in the COGNIFOG Framework,” Software, vol. 4, no. 2, p. 10, Apr. 2025, doi: 10.3390/software4020010.
2. Y. Wang, C. Lee, S. Ren, E. Kim, and S. Chung, “Enabling Role-Based Orchestration for Cloud Applications,” Appl. Sci., vol. 11, no. 14, p. 6656, Jul. 2021, doi: 10.3390/app11146656.
3. I. Kazlaris, G. Papadopoulos, K. Diamantaras, M. Delianidi, E. Touliou, and A. Yenitzes, “An Airflow-Orchestrated AI Pipeline for Podcast Transcription, Topic Modeling, and Recommendation System,” Multimedia, vol. 2, no. 1, p. 1, Jan. 2026, doi: 10.3390/multimedia2010001.
4. N. Karra, N. Karra, L. Echefu, and G. Bandapati, “DevOps-Enabled Cloud-Native Architectures for Scalable and Resilient IoT Applications: A Kubernetes-Based Approach,” in 2025 9th International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC), IEEE, Oct. 2025, pp. 48–53. doi: 10.1109/I-SMAC65734.2025.11597587.
5. C. Lekkala, “Implementing Cloud-Native Technologies for Big Data Processing: A Case Study with Kubernetes and Airflow,” Int. J. Sci. Res., vol. 10, no. 5, pp. 1335–1340, May 2021, doi: 10.21275/SR24430152128.
6. T. K. Chatterjee, “AI-Enabled Cloud Orchestration for Automated Workflows: A Paradigm Shift in Enterprise IT Operations,” Int. J. Sci. Res. Comput. Sci. Eng. Inf. Technol., vol. 11, no. 2, pp. 3289–3296, Apr. 2025, doi: 10.32628/CSEIT25112809.
7. Y. Muni, “A Review of Efficient Routing Architectures Using OSPF, BGP, and MPLS in Multi-Protocol Networks,” Eng. Technol. J., vol. 11, no. 01, Jan. 2026, doi: 10.47191/etj/v11i01.21.
8. S. Vijayabaskar, P. R. Rao, P. Kanchi, S. Jain, and R. Agarwal, “Integrating Cloud-Native Solutions in Financial Services for Enhanced Operational Efficiency,” Univers. Res. Reports, vol. 10, no. 4, pp. 402–419, Oct. 2023, doi: 10.36676/urr.v10.i4.1355.

**9.** J. Rahman and P. Lama, "Predicting the End-to-End Tail Latency of Containerized Microservices in the Cloud," in 2019 IEEE International Conference on Cloud Engineering (IC2E), IEEE, Jun. 2019, pp. 200–210. doi: 10.1109/IC2E.2019.00034.

**10.** A. P. Rajan, "A review on serverless architectures - function as a service (FaaS) in cloud computing," TELKOMNIKA (Telecommunication Comput. Electron. Control., vol. 18, no. 1, p. 530, Feb. 2020, doi: 10.12928/telkomnika.v18i1.12169.

**11.** I. Baldini et al., "Serverless Computing: Current Trends and Open Problems," in Research Advances in Cloud Computing, Singapore: Springer Singapore, 2017, pp. 1–20. doi: 10.1007/978-981-10-5026-8_1.

**12.** M. Waseem, P. Liang, and M. Shahin, "A Systematic Mapping Study on Microservices Architecture in DevOps," J. Syst. Softw., vol. 170, p. 110798, Dec. 2020, doi: 10.1016/j.jss.2020.110798.

**13.** D. S. Linthicum, "Cloud-Native Applications and Cloud Migration: The Good, the Bad, and the Points Between," IEEE Cloud Comput., vol. 4, no. 5, pp. 12–14, Sep. 2017, doi: 10.1109/MCC.2017.4250932.

**14.** M. R. C. Mukkolakkal, "1.5 Million Messages Per Second on 3 Machines : Benchmarking and Latency Optimization of Apache Pulsar at Enterprise Scale," arXiv J., pp. 8–11, 2026, doi: 10.48550/arXiv.2603.29113.

**15.** J. Yasmin, J. A. Wang, Y. Tian, and B. Adams, "An empirical study of developers' challenges in implementing Workflows as Code: A case study on Apache Airflow," J. Syst. Softw., vol. 219, p. 112248, Jan. 2025, doi: 10.1016/j.jss.2024.112248.

**16.** S. K. Malaraju, G. C. Kakaraparthi, and R. C. Bondalapati, "Efficient Deployment of Scalable AI Models via Kubernetes Orchestration in Hybrid Cloud Setups," in 2025 International Conference on Intelligent and Secure Engineering Solutions (CISES), IEEE, Aug. 2025, pp. 1450–1454. doi: 10.1109/CISES66934.2025.11265182.

**17.** M. Avirneni and S. R. Sankiti, "Scalable Multi Cloud Data Engineering with Airflow Orchestration and Iceberg Driven Governance," 2026. doi: 10.2139/ssrn.5926642.

**18.** S. Sanghi, S. Sudhakaran, V. K. Koganti, and N. Ryali, "DevOps and Secure Cloud-Native Architectures for Finance," in 2025 International Conference on Sustainability, Innovation & Technology (ICSIT), IEEE, Aug. 2025, pp. 1–5. doi: 10.1109/ICSIT65336.2025.11294986.

**19.** S. S. Poojary, S. Kamble, and G. S. Mamatha, "A Scalable DevOps Framework Using Kubernetes-Orchestrated Microservices for Cloud-Native Infrastructure," in 2025 9th International Conference on Computational System and Information Technology for Sustainable Solutions (CSITSS), IEEE, Nov. 2025, pp. 1–6. doi: 10.1109/CSITSS67709.2025.11294211.

**20.** S. Lakkireddy, "HyScaleFlow: An ML-Driven DAG-Based Orchestration Framework for Real-Time Stream Processing in Hybrid Cloud Environments," Informatica, vol. 49, no. 9, Oct. 2025, doi: 10.31449/inf.v49i9.9498.

**21.** O. C. Oyeniran, O. T. Modupe, A. A. Otitoola, O. O. Abiona, A. O. Adewusi, and Oluwatayo Jacob Oladapo, "A comprehensive review of leveraging cloud-native technologies for scalability and resilience in software development," Int. J. Sci. Res. Arch., vol. 11, no. 2, pp. 330–337, Mar. 2024, doi: 10.30574/ijsra.2024.11.2.0432.

**22.** C. A. Supit, A. A. Pangeran, K. O. C. Laban, I. I. Gutandjala, and A. Ramadhan, "Incorporating Cloud Native Architecture and DevOps Culture to Improve Company Agility," in 2023 3rd International Conference on Intelligent Cybernetics Technology and Applications, ICICyTA 2023, 2023. doi: 10.1109/ICICyTA60173.2023.10428951.

**23.** A. Harika, P. Bhavani, P. Sriteja, S. Tajuddin, and S. S. Harsha, "Optimizing Scalability and Resilience: Strategies for Aligning DevOps and Cloud-Native Approaches," in 3rd International Conference on Innovative Mechanisms for Industry Applications, ICIMIA 2023 - Proceedings, 2023. doi: 10.1109/ICIMIA60377.2023.10426288.

**24.** J. K. Paruchuri, "Survey of Cloud-Native Workflow Orchestration with Apache Airflow," Int. J. Comput. Tech., vol. 9, no. 3, pp. 1–15, 2022.