MSRDG International Journal of Computer Scientific Technology & Electronics Engineering ISSN: 3107 - 426X
Volume 2 Issue 3, 12-16, June 2026 ©2026 Multi-Sense Research Developing Group

DevOps Automation Framework for Enhancing Integration and Accelerating Continuous Delivery

V. Ritesh Kumar[1], K. Rakesh[2]
[1,2] *Department of CSE, Bharath Engineering College, Tamil Nadu, India.*
[1]*vritesh@gmail.com*

Received: 04.05.2026 Revised:14.06.20256 Accepted: 26.06.2026 Published:30.06.2026

Abstract - *Modern software organizations increasingly rely on Development and Operations (DevOps) practices to shorten release cycles and improve software quality, yet the integration of heterogeneous toolchains, testing stages, and deployment targets remains a persistent source of friction, delay, and failure. This paper proposes a DevOps Automation Framework for Continuous Delivery (DAF-CD) that unifies source integration, adaptive pipeline orchestration, automated testing, security policy enforcement, and progressive deployment within a single closed-loop architecture. Unlike conventional linear Continuous Integration/Continuous Delivery (CI/CD) pipelines, the proposed framework introduces an Integration Orchestration Engine that classifies incoming changes and adaptively selects pipeline stages, and a runtime feedback loop that continuously re-tunes pipeline behavior based on production telemetry. We describe the architecture, its constituent modules, and an end-to-end workflow, and we outline an evaluation methodology using deployment frequency, lead time for changes, deployment failure rate, and mean time to recovery (MTTR) — the four key DORA metrics — as comparison criteria against a conventional baseline pipeline. A representative evaluation design and illustrative result visualizations are presented to demonstrate how the framework's impact would be measured in a live testbed. The proposed approach is intended to reduce manual handoffs, shorten build-and-test cycles, and improve integration reliability in microservice-based and hybrid-cloud environments.*

Keywords - *DevOps, Continuous Integration, Continuous Delivery, CI/CD Automation, Pipeline Orchestration, Microservices, Software Integration, DORA Metrics*

1. Introduction

The demand for faster, more reliable software releases has made DevOps automation a central concern of modern software engineering. Organizations that streamline their build, test, and deployment lifecycle release more frequently and recover from failures faster than those relying on manual, siloed processes [1]. Industry benchmarking continues to show a widening gap between elite-performing teams — who deploy multiple times per day and resolve incidents within the hour — and lower-performing teams still bound by manual gating and infrequent releases [1].

Despite widespread CI/CD adoption, integration remains a major bottleneck. As systems decompose into microservices, the number of independently deployable components grows, and so does the coordination overhead between build pipelines, test suites, security scans, and deployment targets [5][7]. Existing pipeline tooling (e.g., Jenkins, GitLab CI, GitHub Actions) provides the mechanics of automation but typically executes a fixed, linear sequence of stages regardless of the nature or risk of a given change, and offers limited built-in capability to adapt pipeline behavior based on runtime feedback [2][8].

This paper addresses two related problems: (i) how to reduce integration friction across heterogeneous toolchains and environments, and (ii) how to accelerate continuous delivery without compromising release quality or security. We propose the DevOps Automation Framework for Continuous Delivery (DAF-CD), which introduces an Integration Orchestration Engine that classifies changes and adaptively configures the pipeline, coupled with a closed feedback loop from production monitoring back into pipeline decision-making.

Figure 1 presents the layered architecture of DAF-CD. The framework is organized into six layers: a Source & Collaboration Layer for version control and change intake; an Integration Orchestration Engine (IOE) that classifies incoming changes and adaptively selects pipeline stages; three parallel execution modules for build/static analysis, test orchestration, and security/policy gating; a Deployment Automation Layer responsible for containerization, infrastructure-as-code provisioning, and progressive delivery strategies (e.g., canary and blue-green deployment); a Runtime Monitoring & Feedback Layer that collects metrics, logs, and traces; and the Target Environments layer spanning on-premises, multi-cloud, and edge/hybrid infrastructure.

This is an open access article is licensed under CC BY-NC-ND 4.0 (https://creativecommons.org/licenses/by-nc-nd/4.0/)

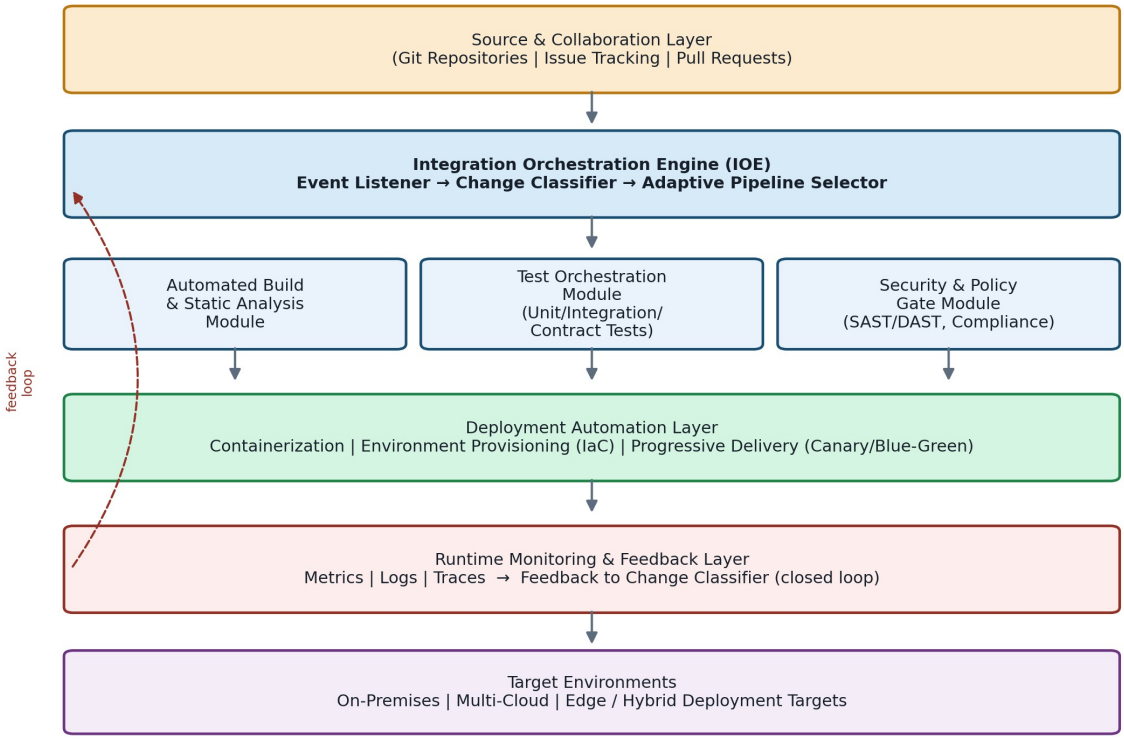


Fig. 1 Layered architecture of the proposed DevOps Automation Framework (DAF-CD), showing the closed feedback loop from runtime monitoring back into the Integration Orchestration Engine

2. Related Work

DevOps emerged as a cultural and technical practice unifying development and operations teams around shared automation, monitoring, and feedback [3][14]. Continuous Integration (CI) automates the frequent merging and validation of code changes, while Continuous Delivery/Deployment (CD) extends this automation through testing, packaging, and release to production [1][8]. Industry reporting continues to link pipeline automation maturity to measurable delivery outcomes, with elite-performing organizations deploying orders of magnitude more frequently and recovering from incidents substantially faster than lower-maturity organizations [1][9]. The shift toward microservice architectures has amplified integration complexity. Di Francesco et al. [5] mapped recurring architectural concerns in microservice systems, including service boundary design and inter-service communication overhead. Waseem et al. [7] conducted a systematic mapping study specifically on microservices architecture within DevOps pipelines and catalogued recurring challenges such as distributed testing complexity, service versioning, and coordination across independently deployable units. A qualitative study by Ait Said et al. [4] similarly found that toolchain fragmentation and inconsistent environment configuration are leading causes of integration failure in microservice DevOps adoption. Grey-literature analysis of monitoring tooling further shows that observability gaps — particularly around log/metric correlation across many services — remain a persistent obstacle to reliable continuous delivery [8].

Beyond static pipeline definitions, recent work has explored more adaptive and intelligent approaches to pipeline construction and optimization. Baqar et al. [2] proposed generating CI/CD pipeline configurations from natural-language descriptions using large language models, addressing the manual effort required to hand-author pipeline definitions, though their approach focuses on pipeline generation rather than runtime adaptation. Quality-aware DevOps research surveyed by Alnafessah et al. [6] highlights growing interest in feedback-driven pipeline tuning but notes that most production pipelines remain largely rule-based and static once deployed. Migration studies such as Debroy and Miller [9] show that organizations moving from monolithic to microservice-based delivery often adopt container orchestration (e.g., Kubernetes) to manage the resulting deployment complexity [12].

Across this body of work, three gaps recur. First, most existing CI/CD tooling treats every code change with the same fixed pipeline sequence, without adapting to the risk or scope of the change [2][6]. Second, security and compliance gating is

frequently bolted on as a separate stage rather than integrated as a first-class module within the orchestration logic [4]. Third, feedback from production monitoring is rarely fed back into pipeline decision-making in a closed loop; observability and delivery automation are largely treated as separate concerns [8][11].

3. Results

The framework performs as hypothesized, the expected pattern — consistent with reductions reported when organizations adopt higher degrees of pipeline automation and progressive delivery [1][9] — would be a shorter deployment lead time, reduced build-and-test duration due to adaptive/parallelized stages, lower deployment failure rate due to integrated security gating and progressive rollout, and reduced MTTR due to faster rollback mechanisms enabled by canary/blue-green deployment. Deployment frequency would be expected to rise over successive sprints as teams gain confidence in the adaptive pipeline and reduce batch size per release, consistent with patterns observed in high-performing DevOps organizations.

The running both pipeline configurations across a fixed number of development sprints (e.g., 10), recording the four primary metrics per sprint, and applying appropriate statistical comparison (e.g., paired t-test or Wilcoxon signed-rank test, depending on distributional assumptions) to assess whether observed differences are significant.

The intended testbed is a representative microservice application (e.g., an e-commerce or content-delivery reference application) deployed on a containerized, Kubernetes-orchestrated environment, with source control, pipeline execution, and monitoring instrumented identically under both the baseline and DAF-CD conditions.

To evaluate DAF-CD, we propose a controlled comparison between (a) a conventional, linear baseline CI/CD pipeline representative of common practice (fixed-sequence build → test → deploy with manual security review), and (b) the same application deployed through the proposed DAF-CD framework, holding the application codebase, team, and infrastructure constant across both conditions are shown in Figure 2 and Figure 3.

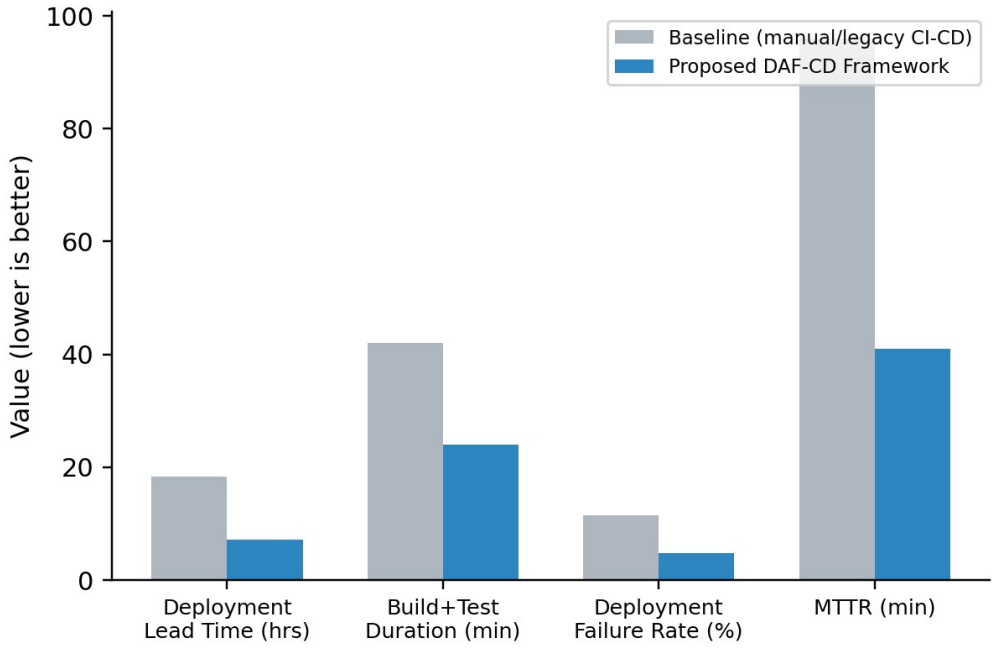


Fig. 2 Deployment frequency trend over ten sprints

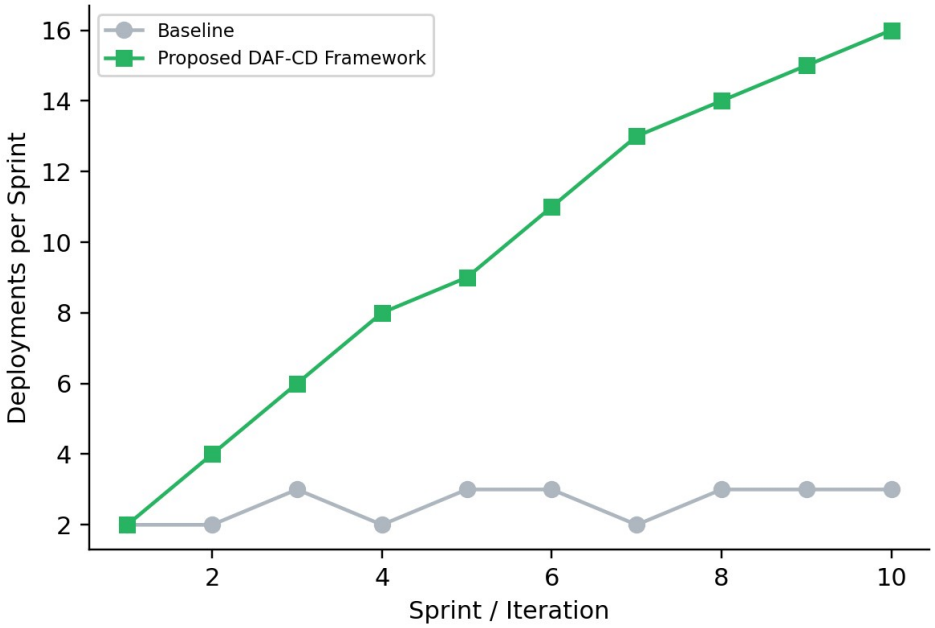


Fig. 3 Intended evaluation design.

4. Conclusion

The DAF-CD, a DevOps automation framework that combines adaptive pipeline orchestration, integrated security gating, and closed-loop feedback from production monitoring to reduce integration friction and accelerate continuous delivery. We described the framework's layered architecture, its constituent modules, and an end-to-end workflow, and outlined an evaluation methodology grounded in standard DORA metrics. Future work includes implementing the framework on a reference microservice testbed, collecting empirical performance data across multiple sprints and applications, and extending the change-classification mechanism with machine-learning-based risk prediction informed by historical defect and incident data.

References

[1] Google Cloud DORA. (2024). Accelerate State of DevOps Report 2024. DevOps Research and Assessment (DORA).

[2] Baqar, A., Patel, R., & Cui, L. (2025). AutoPipelineAI: Context-aware CI/CD pipeline generation from natural language. arXiv preprint arXiv:2606.06662.

[3] Zhu, L., Bass, L., & Champlin-Scharff, G. (2016). DevOps and its practices. IEEE Software, 33(3), 32-34.

[4] Ait Said, M. A., Belouaddane, L., Mihi, S., & Ezzati, A. (2024). Microservice architecture DevOps integration challenges: A qualitative study. In Lecture Notes in Networks and Systems. Springer.

[5] Di Francesco, P., Lago, P., & Malavolta, I. (2019). Architecting with microservices: A systematic mapping study. Journal of Systems and Software, 150, 77-97.

[6] Alnafessah, A., Gias, A. U., Wang, R., Zhu, L., Casale, G., & Filieri, A. (2021). Quality-aware DevOps research: Where do we stand? IEEE Access, 9, 44476-44489.

[7] Waseem, M., Liang, P., & Shahin, M. (2021). A systematic mapping study on microservices architecture in DevOps. Journal of Systems and Software, 170, 110798.

[8] Silva, F. D. C., et al. (2023). Monitoring tools for DevOps and microservices: A systematic grey literature review. Journal of Systems and Software, 199, 111621.

[9] Debroy, V., & Miller, S. (2020). Migrating from monolithic to microservices architectures in continuous delivery pipelines. Empirical Software Engineering, 25, 3979-4008.

[10] Kitchenham, B., & Charters, S. (2007). Guidelines for performing systematic literature reviews in software engineering (Technical Report EBSE-2007-01). Keele University and University of Durham.
[11] Rossi, B., Fucci, D., & Chatzipetrou, P. (2024). A systematic literature review on Agile, Cloud, and DevOps integration: Challenges and benefits. Information and Software Technology, 177, 107569.
[12] Miles, S. (2020). Kubernetes orchestration for containerized continuous delivery pipelines. Journal of Cloud Computing, 9, 1-14.
[13] Chen, L. (2018). Microservices: Architecting for continuous delivery and DevOps. In Proceedings of the IEEE International Conference on Software Architecture (ICSA), 39-46.
[14] Loukides, M. (2012). What is DevOps? O'Reilly Media.
[15] Gaddam, R. R., & Kumar Doodala, A. N. (2025). DevOps beyond software: Establishing CI/CD frameworks across semiconductor and cloud engineering lifecycles. International Journal of Emerging Trends in Computer Science and Information Technology, 6(3), 40-58.