A Comparative Study of Manual and Jenkins-Based CI/CD Deployment in a Cloud Environment

Miss. Shraddha Nitin Talekar
Shubhangi Bombale (Assistant Professor), Department of Master of Computer Application, JCEI's Jaihind Institute of Management and Research, Vadgaon Sahani

***Abstract*—DevOps has become an important approach to improving software development and deployment by integrating development activities with operational processes. In traditional manual deployment, several tasks depend on human intervention, while Jenkins enables teams to automate predefined stages of software building, testing, and deployment through CI/CD pipelines. This study examines the differences between manual deployment and Jenkins- supported automation in cloud computing environments through a review of five selected research publications. The reviewed studies provide insights into deployment efficiency, automation practices, continuous delivery challenges, software architecture, and the use of Infrastructure as Code. The literature indicates that automated pipelines may improve deployment consistency and reduce repetitive operational work; however, their effectiveness depends on appropriate testing procedures, system configuration, security controls, and architectural design. This paper synthesizes findings from previously published research rather than presenting results from a newly conducted experiment. Therefore, the discussion reflects evidence reported in the selected publications and does not claim independent performance measurements from an AWS deployment environment.**

***Index Terms*—DevOps, Jenkins, CI/CD, AWS EC2, Cloud Computing, Deployment Automation.**

I. INTRODUCTION

Software teams today need deployment processes that support timely releases, consistent performance, and reliable application delivery. In conventional deployment practices, developers or system administrators often perform several operations manually, including preparing application files, configuring the target environment, and restarting services. Repeating these activities for every release can increase operational effort and create opportunities for configuration mistakes. To address these challenges, many organizations adopt DevOps principles and Continuous Integration and Continuous Delivery/Deployment (CI/CD) practices.

DevOps encourages collaboration between software development and IT operations throughout the software delivery lifecycle. Continuous Integration focuses on integrating code changes regularly and checking them through automated build and testing activities. Continuous Delivery extends this workflow by keeping software in a deployable state, while Continuous Deployment can automatically release qualifying changes when the required conditions are satisfied. The implementation of these practices varies according to project requirements, testing strategies, and organizational policies.

Cloud computing has further changed how applications are hosted and maintained by providing access to flexible computing resources. Amazon Web Services (AWS), for example, offers Amazon Elastic Compute Cloud (EC2), which provides virtual computing instances for running applications. Deploying an application on an EC2 instance may require several activities, such as transferring files, installing dependencies, adjusting configuration settings, and managing application services. When these operations are performed manually for repeated releases, the process can become difficult to maintain consistently.

Jenkins is an automation server that can coordinate software build, testing, and deployment activities through configured pipelines. A Jenkins-based workflow can connect a source code repository with build tools, testing procedures, and deployment scripts. When integrated with an AWS EC2 environment, it can automate selected deployment operations and reduce the need for repeated human

intervention. Nevertheless, automation does not eliminate every deployment problem. Pipeline reliability also depends on appropriate configuration, secure credential management, sufficient testing, and ongoing maintenance.

Earlier research has examined DevOps implementation, continuous delivery practices, deployment automation, and the challenges associated with software delivery in cloud environments [1], [2]. This body of work provides a basis for examining how automation may influence deployment workflows. However, the benefits reported in one setting may not apply equally to every application because infrastructure, software architecture, testing requirements, and deployment procedures can differ.

In this paper, a literature-based comparative approach is used to examine manual deployment and Jenkins-supported CI/CD deployment in cloud environments. Five selected research publications are reviewed to identify reported benefits, limitations, and implementation challenges. The comparison considers deployment efficiency, dependence on manual operations, reliability, and the potential for configuration errors, based on evidence available in the reviewed studies. Rather than claiming results from an independently conducted AWS experiment, the study synthesizes published findings to develop a clearer understanding of the practical strengths and limitations of Jenkins-based deployment automation.

II. PROBLEM STATEMENT

Manual application deployment often depends on a series of human-performed activities, including preparing deployment packages, transferring files, configuring servers, and restarting application services. Repeating these operations across multiple releases can increase workload and create opportunities for configuration inconsistencies and deployment errors. However, their effectiveness depends on suitable pipeline design, testing practices, security measures, and ongoing maintenance.

comparisons difficult and limit the extent to which findings from one setting can be generalized to another. Therefore, there is a need to systematically examine the existing literature to understand the reported advantages, limitations, and implementation challenges of manual and Jenkins-based deployment in cloud environments. This study addresses this need by comparing findings from five selected research publications and identifying the conclusions that can reasonably be drawn from the available evidence.

III. OBJECTIVES OF THE STUDY

This study focuses on the following objectives:

To understand how DevOps practices support coordination and improvement throughout the software delivery process.

To examine how a manually managed deployment workflow differs from a deployment process coordinated through Jenkins automation.

To investigate what previously published studies report about the contribution of deployment automation to release speed and process consistency.

To synthesize findings from five research publications covering cloud deployment, software delivery practices, system architecture, and DevOps implementation.

To assess the strengths and practical difficulties of Jenkins-based deployment by examining evidence presented in the selected literature.

To explore why reliable testing, controlled configuration, secure access, and proper operational preparation are important when implementing automated deployment workflows.

To determine which aspects of manual and automated deployment remain insufficiently examined and could be investigated through future practical experiments.

IV. LITERATURE REVIEW

The transition from traditional software releases to automated delivery has encouraged researchers and practitioners to examine how deployment activities can be organised more effectively. Several related areas contribute to this discussion, including workflow automation, continuous software validation, container-based delivery, and cloud infrastructure management. This review considers these areas to understand what existing publications can contribute to a comparison of manual and Jenkins-supported deployment.

4.1 Jenkins in Software Release Workflows

Jenkins allows a project team to describe release activities as a sequence of configured operations. Depending on the application, these operations may include retrieving source files, preparing a build,

checking the application, and transferring the release to its destination. The resulting workflow can be adapted to the requirements of a particular project.

The main distinction between manual and automated execution concerns how these operations are initiated and coordinated. A manually managed release requires an operator to carry out the necessary actions, while a Jenkins workflow can execute configured tasks according to defined conditions. Automation may reduce repeated effort, but its effectiveness depends on whether the instructions reflect the actual deployment requirements. Problems in scripts, dependencies, permissions, or server settings can still interrupt a release. Jenkins should therefore be examined as an enabling tool rather than as a guarantee of successful deployment.

4.2 Managing Software Changes Through CI/CD

CI/CD provides a way to organise software changes between development and release. Integration checks help identify problems when new code is combined with an existing application. Subsequent delivery activities prepare a validated version for release, while fully automated deployment can publish eligible changes without a separate manual release action.

The suitability of this approach depends on the quality of the checks and the rules governing release decisions. Automated execution can make a process more repeatable, but incomplete validation may allow faults to reach the target environment. Evaluating a CI/CD implementation therefore requires attention to the complete workflow, including failure handling, testing, and release controls, rather than considering automation alone.

4.3 Jenkins and Container Deployment Research

Hyun et al. (2024) published a paper titled "The Impact of an Automation System Built with Jenkins on the Efficiency of Container-Based System Deployment." Its subject is relevant to this study because it concerns the use of Jenkins in a container deployment context.

To use this paper as evidence, the original publication must be examined for its research design, deployment environment, evaluation criteria, and reported outcomes. These details determine whether its conclusions can reasonably inform a comparison with other deployment approaches. Until those details are verified, no specific performance improvement or limitation should be attributed to the study.

4.4 Cloud Resources and Infrastructure as Code

Application release procedures are influenced by the computing environment in which the software runs. Amazon EC2 provides virtual servers that can host applications, while deployment scripts and automation services can coordinate the activities required to publish a new version.

Infrastructure as Code (IaC) addresses the management of infrastructure through reusable configuration definitions. Terraform is one example of a tool used to describe and provision infrastructure resources. Coordinating infrastructure provisioning with application delivery can make environment setup more reproducible. However, the result still depends on access permissions, configuration accuracy, monitoring arrangements, credential protection, and ongoing maintenance.

4.5 Comparison of Identified Sources

Table 1. Preliminary source assessment

Source	Relevance to the research	Point requiring attention
Jenkins Documentation	Describes the configuration and execution of Jenkins workflows.	Tool documentation alone does not establish measured performance gains.
Hyun et al. (2024)	Examines Jenkins automation in container deployment.	Research methods and findings must be checked against the original publication.
Shahin et al. (2017)	Discusses challenges associated with continuous deployment.	The study context and reported challenges should be verified in the original paper.
Shahin et al. (2019)	Examines architecture for continuous delivery and deployment.	Findings should be interpreted in the context of the study's methods and environment.
Khan and Shameem (2020)	Provides a taxonomy of challenging factors in DevOps.	The taxonomy does not by itself measure deployment-time improvements.

Source Relevance to the research

Point requiring attention
Describes the configuration and execution of Jenkins workflows
Tool documentation alone does not establish measured performance gains

Jenkins Project Documentation
Examines Jenkins automation in container deployment
Research methods and findings must be checked against the original publication
Hyun et al. (2024)
Automation in Cloud- Based DevOps: A Guide to CI/CD Pipelines and Infrastructure as Code (IaC) with Terraform and Jenkins (2024)

Publication details and the nature of its supporting evidence need verification
Discusses cloud delivery and infrastructure automation
The identified sources have different purposes and should not be interpreted as interchangeable evidence. Documentation explains available functionality, whereas an empirical paper may report observations from a defined implementation. A conceptual guide may explain relationships between technologies without testing them experimentally. The final comparison should make these distinctions clear.

4.6 Research Synthesis
For this study, the important question is not simply whether automation is available, but what the available research establishes about its practical use. Relevant aspects include the work performed manually, the repeatability of release procedures, the handling of deployment failures, and the requirements for maintaining the delivery environment.
Differences in application design, infrastructure, and evaluation methods may prevent direct comparisons between published results. Findings from a container-based system, for example, should not automatically be applied to every application hosted on a virtual machine. Where the sources do not provide equivalent measurements, the comparison should describe the evidence available and clearly identify what remains uncertain.
This review establishes a basis for examining manual and Jenkins-supported deployment through published research. Completing the analysis requires five verified research publications, accurate citations, and a consistent method for comparing their findings. Any conclusions about deployment efficiency should be limited to what those sources actually support.

V. RESEARCH GAP

Previous studies have investigated Jenkins-based CI/CD automation and its impact on software deployment efficiency. Hyun et al. (2024) compared manual deployment with Jenkins-based automated deployment and reported improvements in deployment time and error rate within their experimental environment. Research on cloud-based DevOps has also discussed the integration of CI/CD pipelines and Infrastructure as Code to improve deployment consistency and operational efficiency. [1], [2] Although these studies provide useful insights into deployment automation, their findings are associated with the specific environments, configurations, and methods examined in each study. Further practical evaluation can help understand how these approaches perform under different experimental conditions. Therefore, the present study focuses on a small-scale experimental comparison of manual deployment and Jenkins-based CI/CD deployment using a selected application in a cloud environment. Deployment duration, manual intervention, deployment success, and repeatability will be recorded under comparable conditions. The purpose is to evaluate the observed differences in the selected environment and identify the practical benefits and limitations of the implemented CI/CD workflow. The contribution of this study will be based on the actual experimental setup, recorded observations, and analysis of the selected deployment parameters rather than assuming that the results of previous studies will apply universally.

VI. PROPOSED METHODOLOGY

research proposes a controlled comparison of two application release approaches: direct manual deployment and deployment through a Jenkins-supported CI/CD workflow. The planned investigation will use an application hosted on an Amazon Web Services (AWS) EC2 instance. Both approaches will be assessed using a common set of evaluation criteria so that differences in execution time, human involvement, successful completion, and deployment-

related errors can be examined. The analysis will rely on observations collected during the experiment rather than assuming that automation will necessarily produce better results.

6.1 Research Design

The investigation will follow a comparative experimental design involving two deployment procedures.

Manual deployment: The selected application will be prepared and released by an operator using the required deployment commands and procedures. Activities may include preparing the application package, transferring files to the target machine, checking configuration settings, and restarting the relevant service.

Jenkins-assisted deployment: A Jenkins pipeline will be configured to execute the deployment activities that can reasonably be automated. Depending on the application's requirements, the workflow may retrieve source code from GitHub, build the application, run available tests, transfer the required files, and perform the release on the EC2 instance.

The same application and target infrastructure will be used for both procedures wherever practical. The starting conditions, deployment completion criteria, and measurement rules will be documented before data collection. Any unavoidable differences between the two procedures will be recorded to support a fair interpretation of the results.

6.2 Tools and Technologies

The planned implementation may use the
following components, subject to the requirements of the selected application.

Amazon EC2: Provides the virtual computing environment in which the application will be hosted.

Jenkins: Coordinates the configured build and deployment activities through a pipeline.

Git and GitHub: Maintain the application source code and provide repository access for deployment.

Linux: May serve as the operating system for the EC2 instance, depending on the final environment.

Application server or runtime: Executes the application using the software required by its technology stack.

Logs and monitoring facilities: Help record execution details and investigate unsuccessful deployment attempts.

The final toolchain will be documented after the environment has been configured. Only the components actually used during the experiment will be reported as implemented tools.

6.3 Experimental Procedure

The proposed experiment will follow a consistent sequence of preparation, execution, and evaluation.

Prepare the application: Select an application suitable for the chosen hosting environment and verify that its source code and runtime requirements are available.

Configure the cloud environment: Create and prepare an EC2 instance, install the required runtime components, and verify that the application can operate in the target environment.

Perform the manual release: Execute the documented deployment procedure without using Jenkins to automate the release. Record the elapsed time, human-performed actions, final application status, and deployment-related problems.

Prepare the Jenkins workflow: Configure the pipeline to perform the applicable source retrieval, build, testing, and deployment tasks. Record the pipeline settings relevant to the comparison.

Execute the automated release: Run the Jenkins pipeline and capture its execution duration, required human intervention, final application status, and any errors reported during execution.

Repeat the trials: Carry out multiple runs of both approaches under comparable conditions. Record environmental changes or other factors that could influence the observations.

Organise the observations: Store the results from each run in a structured data table. Keep unsuccessful attempts and relevant error records rather than excluding them without explanation.

Compare the approaches: Calculate the selected performance measures and examine the differences observed between manual and Jenkins-assisted deployment.

6.4 Evaluation Criteria

The following measures will be used to examine the two approaches.

Deployment duration: The elapsed time between a predefined starting event and a common completion condition. The same timing rules will be applied to both procedures. The study will also document

whether setup activities are included in or excluded from the measurement.

Human involvement: The number of deployment-related actions that require an operator to perform or approve a task. The counting rules will be established before the trials begin so that the results are interpreted consistently.

Successful completion: Whether a deployment finishes and the application passes the predefined verification checks. A successful pipeline execution alone will not be sufficient if the deployed application fails its verification checks.

Deployment-related errors: The types and numbers of relevant failures observed during the trials, including build problems, configuration mistakes, transfer failures, and unsuccessful application starts. The classification rules will be applied consistently to both methods.

Repeatability: Whether repeated runs produce the expected application state and whether the deployment procedure can be executed consistently under the documented conditions.

6.5 Data Collection and Analysis

Each deployment trial will produce a record containing the approach used, elapsed time, human-performed actions, completion status, and observed errors. Results will be organised by trial so that individual variations remain visible. The average duration and successful deployment rate will be calculated separately for each approach.

The average deployment duration will be calculated using:

[\text {Average Duration} = \frac {\text {Sum of Deployment Durations}} {\text {Number of Trials}}]

III. If the measured automated duration is lower than the manual duration, the percentage reduction will be calculated as:

[\text {Time Reduction (%)} = \ f r a c { T _ m - T _ a} { T _ m} \ t i m e s 1 0 0]

deployment duration and (T_a) represents the average Jenkins-assisted deployment duration. A positive result indicates a reduction in duration; a negative result indicates that the automated procedure took longer under the measured conditions.

The successful deployment rate will be determined using:

[\text {Success Rate (%)} = \frac {\text {Successful Deployment Attempts}} {\text {Total Deployment Attempts}} \times100]

The observations will be summarised in tables and may be represented graphically where this improves interpretation. The discussion will consider both favourable and unfavourable outcomes. Conclusions about speed, reliability, or reduced human effort will be drawn only when supported by the collected data.

6.6 Scope and Limitations

The conclusions of the experiment will apply primarily to the selected application, the documented EC2 environment, and the Jenkins pipeline used in the investigation. Differences in application architecture, available computing resources, network conditions, testing coverage, and pipeline design may produce different outcomes in other settings.

The findings will therefore be interpreted within the boundaries of the actual implementation. The study will not assume that results from one application establish the performance of every cloud deployment workflow. Any factors that could not be controlled or measured will be acknowledged when discussing the results and identifying opportunities for further research.

VII. RESULTS AND DISCUSSION

7.1 Findings from the Reviewed Literature

This section examines the findings of five selected research papers to understand how deployment automation influences software delivery in DevOps environments. The reviewed studies cover deployment efficiency, continuous deployment challenges, software architecture, DevOps adoption, and cloud automation practices.

Hyun et al. [1] investigated the use of Jenkins to automate container-based deployment. Their reported results show a difference in deployment time between manual and automated approaches. For example, the study reported 14 seconds for one manual deployment compared with 3 seconds for the automated approach. For 30 deployments, the reported times were 367 seconds and 10 seconds, respectively. These figures belong to the environment examined by the authors and should not be interpreted as results from the present study's AWS environment.

Shahin et al. [2] examined the practical difficulties organizations encounter when adopting continuous deployment. Their work highlights that automation alone cannot eliminate every delivery obstacle. Incomplete automated testing, manual quality checks, and organizational approval procedures can still restrict frequent releases.

The architectural perspective discussed by Shahin et al. [3] indicates that successful continuous delivery depends on software design as well as deployment tools. System structure, operational requirements, and the ability to release components independently can influence how effectively delivery practices are implemented.

Khan and Shameem [4] studied the factors that make DevOps adoption challenging. Their work organized identified challenges and applied the Analytical Hierarchy Process to prioritize them. This suggests that organizations should evaluate technical and organizational difficulties systematically instead of treating automation as a tool-only problem.

Akinbolaji et al. [5] reviewed cloud-based DevOps practices, including CI/CD pipelines and Infrastructure as Code (IaC) using tools such as Jenkins and Terraform. Their discussion emphasizes repeatable configurations and automated workflows while also recognizing concerns such as configuration complexity, security, and compliance.

7.2 Comparative Analysis of the Selected Studies

Table 1. Comparison of the Five Reviewed Research Papers

Ref.	Main research area	Key contribution	Relevance to this study
[1]	Jenkins-based deployment automation	Reports deployment-time differences between manual and automated approaches	Provides direct evidence related to deployment efficiency
[2]	Continuous deployment challenges	Identifies testing and organizational barriers to frequent releases	Explains why automation may not solve every deployment problem
[3]	Architecture for continuous delivery	Examines architectural factors that affect delivery practices	Connects software design with deployment reliability
[4]	DevOps adoption challenges	Classifies and prioritizes challenges using AHP	Supports structured evaluation of implementation barriers
[5]	Cloud CI/CD and Infrastructure as Code	Reviews automation practices using Jenkins and Terraform	Explains how cloud workflows can be made more consistent

7.3 Manual Deployment Compared with Jenkins-Based Automation

Manual deployment requires a person to carry out and verify deployment activities. Depending on the application and environment, these activities may include preparing the build, transferring files, configuring the server, and checking whether the application is running correctly. Repeating such work can increase effort and create opportunities for inconsistent steps.

A Jenkins-based pipeline can place several of these activities into a defined workflow. Once configured, the pipeline can execute the specified build and deployment stages in a repeatable manner. The findings reported by Hyun et al. [1] support the potential for reduced deployment time in the environment they evaluated.

However, the benefits depend on how the pipeline is designed and maintained. Incorrect configuration, failed builds, insufficient testing, or access-control problems may still interrupt deployment. Therefore, automation should be considered a way to standardize and execute deployment tasks, rather than a guarantee that every release will succeed.

7.4 Testing, Architecture, and Organizational Readiness

The studies by Shahin et al. [2], Shahin et al. [3], and Khan and Shameem [4] show that deployment practices are influenced by several connected factors. Automated tools are useful, but effective delivery also requires appropriate testing, suitable system architecture, and clear organizational processes.

For example, a pipeline may successfully deploy an application while an undetected software defect remains. Automated tests and appropriate quality checks are therefore important before release. Similarly, a system that is difficult to configure or maintain may require additional work before deployment can be automated reliably.

These findings indicate that deployment efficiency should not be judged only by execution time. Testing practices, operational reliability, maintainability, and organizational readiness should also be considered when comparing deployment approaches.

7.5 Relevance of Cloud Automation and Infrastructure as Code

The review by Akinbolaji et al. [5] connects CI/CD automation with cloud infrastructure management. Jenkins can coordinate defined software delivery stages, while Infrastructure as Code tools such as Terraform can help manage infrastructure through version-controlled configuration.

This combination can support more consistent environments and make configuration changes easier to review. Nevertheless, cloud automation introduces responsibilities involving credentials, permissions, configuration management, and compliance. Appropriate access restrictions and secure handling of secrets are therefore important parts of a reliable deployment process.

For an AWS-based application, these considerations are relevant when designing a workflow that uses cloud resources. However, the results of the reviewed papers do not establish a single performance outcome for every AWS deployment scenario.

7.6 Overall Findings and Research Implications

The comparison of the five papers suggests three main conclusions. First, Jenkins-based automation can reduce repeated manual work and deployment time in the settings studied by the researchers [1]. Second, the effectiveness of continuous delivery also depends on testing, architecture, and organizational practices [2]–[4]. Third, cloud automation and Infrastructure as Code can improve the consistency of deployment workflows, but configuration and security must be managed carefully [5].

An important limitation of this review is that the selected studies investigate different settings and research questions. Their findings cannot be combined into a single controlled measurement of manual and Jenkins-based deployment on AWS EC2. The numerical results reported in [1] should therefore remain attributed to that study.

Future research could conduct a controlled comparison in an AWS EC2 environment using clearly defined measures such as deployment duration, number of manual steps, deployment success rate, and observed errors. Until such measurements are collected, this paper's conclusions should be understood as a comparative interpretation of the published literature rather than as results from a new experiment.

VIII. CONCLUSION

This paper examines the differences between manual software deployment and Jenkins-supported CI/CD by synthesising findings from previously published research. The reviewed literature provides insights into how deployment automation can organise repetitive release activities and support a more consistent software delivery process. One of the selected studies reports improved deployment efficiency within its particular experimental setting. Such findings suggest that automation can be beneficial, although the extent of improvement depends on the environment and the method used for evaluation.

The review also indicates that introducing an automated pipeline is only one part of an effective deployment strategy. Reliable outcomes require suitable testing procedures, accurate configuration, secure access management, appropriate infrastructure, and ongoing monitoring. Problems in these areas may affect deployment results even when the workflow itself is automated. Furthermore, differences in research methods and technical environments limit the extent to which findings from individual studies can be applied universally.

As this paper synthesises published evidence rather than reporting a new deployment experiment, its conclusions are restricted to the findings that can be supported by the selected literature. Future investigations could evaluate both deployment approaches using a clearly documented application and cloud environment, such as AWS EC2. Repeated trials measuring execution time, human intervention, deployment failures, and consistency would provide additional evidence for a more direct comparison.

In conclusion, Jenkins-based CI/CD offers a practical means of coordinating and automating software release activities. Its effectiveness should be assessed in the context of the application, infrastructure, and operational controls in which it is used. A balanced implementation that combines automation with adequate testing, security, configuration management,

and monitoring is essential for dependable software delivery.

REFERENCES

[1] G. Hyun, J. Oak, D. Kim, and K. Kim, "The impact of an automation system built with Jenkins on the efficiency of container-based system deployment," *Sensors*, vol. 24, no. 18, Art. no. 6002, 2024, doi: 10.3390/s24186002.

[2] M. Shahin, M. A. Babar, M. Zahedi, and L. Zhu, "Beyond continuous delivery: An empirical investigation of continuous deployment challenges," in *Proc. 11th ACM/IEEE Int. Symp. Empirical Software Engineering and Measurement (ESEM)*, 2017, pp. 111–120, doi: 10.1109/ESEM.2017.18.

[3] M. Shahin, M. Zahedi, M. A. Babar, and L. Zhu, "An empirical study of architecting for continuous delivery and deployment," *Empirical Software Engineering*, vol. 24, no. 3, pp. 1061–1108, 2019, doi: 10.1007/s10664-018-9651-4.

[4] Khan and M. Shameem, "Multicriteria decision-making taxonomy for DevOps challenging factors using analytical hierarchy process," *Journal of Software: Evolution and Process*, vol. 32, no. 10, Art. no. e2263, 2020, doi: 10.1002/smr.2263.

[5] T. J. Akinbolaji, G. Nzeako, D. Akokodaripon, and A. V. Aderoju, "Automation in cloud-based DevOps: A guide to CI/CD pipelines and infrastructure as code (IaC) with Terraform and Jenkins," *World Journal of Advanced Engineering Technology and Sciences*, vol. 13, no. 2, pp. 90–104, 2024, doi: 10.30574/wjaets.2024.13.2.0542.