Published by :
https://www.ijert.org/
An International Peer-Reviewed Journal

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

Design and Implementation of a DevOps Automation Platform for Continuous Integration, Continuous Delivery, and Cloud Deployment

Swapnil Mali
Department of Computer Science & Engineering
Sandip University, Nashik

Siddeshwar Hodgar
Department of Computer Science & Engineering
Sandip University, Nashik

Kalpesh Somvanshi
Department of Computer Science & Engineering
Sandip University, Nashik

Bhavesh Dalavi
Department of Computer Science & Engineering
Sandip University, Nashik

Dr. Rahis Khan
Department of Computer Science & Engineering
Sandip University, Nashik

Abstract - Modern e-commerce applications must be released frequently, remain available under unpredictable load, and recover quickly from failures. Manual build, test, and deployment practices make this difficult because they are slow, error-prone, and hard to reproduce. This project designs and implements a DevOps automation platform, built around a Jenkins pipeline, that integrates Continuous Integration (CI), Continuous Delivery (CD), and cloud deployment for an online shopping application hosted on Amazon Web Services (AWS). Every code commit automatically triggers a pipeline that builds the application, runs automated tests and code-quality checks, packages the result as a Docker container image stored in a Docker Hub registry, and deploys it to AWS infrastructure that is provisioned through Infrastructure as Code. Centralized monitoring and logging provide feedback on application health after each release. The platform reduces manual intervention, shortens the interval between a code change and its availability to users, keeps environments consistent, and makes rollback straightforward. The work demonstrates a practical, repeatable, and scalable approach to delivering e-commerce software on the cloud.

Keywords - DevOps, Jenkins, Continuous Integration, Continuous Delivery, Cloud Deployment, AWS, Docker, Infrastructure as Code, E-Commerce.

I. INTRODUCTION

Online shopping has become one of the most demanding categories of software. An e-commerce application serves catalogues, carts, payments, and order tracking to many concurrent users, and it must stay available during traffic spikes such as seasonal sales. At the same time, businesses expect new features, price changes, and bug fixes to reach customers quickly. Meeting both expectations requires a software delivery process that is fast, reliable, and repeatable.

Traditionally, development and operations were handled by separate teams. Developers wrote code and handed it over, and operations staff built, configured, and deployed it by hand. This separation produces long release cycles, configuration differences between development, testing, and production servers, and failures that are discovered late. DevOps addresses these problems by combining development and operations practices around shared responsibility, automation, and continuous feedback.

Continuous Integration (CI) is the practice of merging code changes into a shared repository frequently, with each change automatically built and tested so that defects are found early. Continuous Delivery (CD) extends CI by keeping the software in a releasable state and automating its movement through test and staging environments to production. Cloud platforms such as AWS complement these practices by offering on-demand compute, networking, databases, and managed monitoring, which can be created and destroyed through code rather than manual configuration.

This project brings these ideas together in a single automation platform. An online shopping application is used as the real-world workload. The platform automates the complete path from a developer's commit to a running application on AWS, including build, test, containerization, infrastructure provisioning, deployment, and monitoring. The result is a delivery process that is faster, more consistent, and easier to maintain than manual deployment.

II. PROBLEM STATEMENT

Deploying an e-commerce application manually involves many repeated steps: compiling code, running tests, copying files to servers, configuring the environment, and restarting services. Each step is an opportunity for human error, and differences between environments cause defects that appear only in production. Releases are infrequent and risky, rollback is slow, and scaling to meet demand requires manual effort. This project addresses the need for an automated,

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

reproducible CI/CD platform that deploys an online shopping application to the cloud reliably and with minimal manual intervention.

III. OBJECTIVE

- To design a CI/CD pipeline that automatically builds and tests an e-commerce application on every code change.
- To containerize the application so that it behaves identically across development, testing, and production environments.
- To provision and manage AWS infrastructure through Infrastructure as Code to ensure repeatability.
- To automate deployment to the cloud with support for quick rollback and minimal downtime.
- To monitor application and infrastructure health and use that feedback to improve each release.

IV. LITERATURE SURVEY

1. **"Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices"**
   Authors: M. Shahin, M. A. Babar, L. Zhu (2017)
   Summary: This systematic review analyses the literature on CI, continuous delivery, and continuous deployment. It identifies the approaches and tools used, the benefits reported, and the challenges teams face, and discusses how these practices influence team collaboration and system architecture.
2. **"A Survey of DevOps Concepts and Challenges"**
   Authors: L. Leite, C. Rocha, F. Kon, D. Milojicic, P. Meirelles (2019)
   Summary: This survey characterizes DevOps in terms of collaboration between teams, automation of the delivery process, and monitoring. It reviews the supporting practices and tools and highlights open challenges such as organizational culture and security.
3. **"What is DevOps? A Systematic Mapping Study on Definitions and Practices"**
   Authors: R. Jabbari, N. bin Ali, K. Petersen, B. Tanveer (2016)
   Summary: This mapping study examines how DevOps is defined in the literature and finds that definitions vary. It proposes a clearer description centered on collaboration, automation, and continuous practices across development and operations.
4. **"Cloud Container Technologies: A State-of-the-Art Review"**
   Authors: C. Pahl, A. Brogi, J. Soldani, P. Jamshidi (2019)
   Summary: This review covers container technologies and orchestration, explaining how containers package applications with their dependencies for portable, lightweight deployment across cloud environments.
5. **"Borg, Omega, and Kubernetes"**
   Authors: B. Burns, B. Grant, D. Oppenheimer, E. Brewer, J. Wilkes (2016)
   Summary: This article shares lessons learned from three generations of container-management systems at Google and explains the design principles behind Kubernetes, including declarative configuration and automated management of containerized applications.
6. **"Accelerate: The Science of Lean Software and DevOps"**
   Authors: N. Forsgren, J. Humble, G. Kim (2018)
   Summary: Based on multi-year survey research, this book links software delivery performance, measured by deployment frequency, lead time for changes, time to restore service, and change failure rate, to organizational performance, and identifies the technical practices that improve it.

V. EXISTING SYSTEM

In many existing deployments of small and medium e-commerce applications, the release process is largely manual. Developers commit code to a repository, but building, testing, and deploying are carried out by hand or with loosely connected scripts. The application is typically installed directly on one or a few servers, and configuration is applied manually.

Because testing is not tied to every change, defects often reach later stages before they are noticed. Differences between the developer's machine, the test server, and the production server lead to the familiar problem of software that works in one environment but fails in another. Each release requires coordination between developers and operations staff, so releases are infrequent, and each one carries a large number of changes and therefore a higher risk.

Infrastructure in such systems is usually created and modified by hand, which makes it difficult to reproduce, document, or audit. Scaling to handle peak demand means adding and configuring servers manually, and recovering from a failed release often means redeploying the previous version by hand, causing downtime. Monitoring is frequently limited, so problems are reported by users before they are detected by the team.

Overall, existing manual approaches suffer from slow releases, inconsistent environments, limited scalability, risky rollbacks, and weak visibility. These limitations highlight the need for an automated DevOps platform that integrates CI, CD, and cloud deployment.

VI. PROPOSED SYSTEM AND METHODOLOGY

The proposed platform automates the entire software delivery lifecycle of an online shopping application on AWS, as shown in Fig. 1. The pipeline consists of the following stages.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

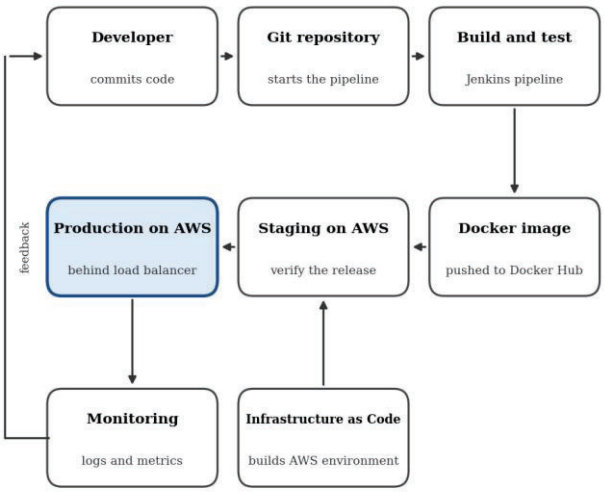

Fig.1 System Architecture of the Proposed CI/CD Platform

1) Source and Plan: Application code and infrastructure definitions are stored in a Git repository. A commit or pull request triggers the Jenkins pipeline automatically through a webhook, and the pipeline is defined as code in a Jenkinsfile stored with the application.

2) Continuous Integration (Build and Test): The Jenkins server checks out the code, compiles it, resolves dependencies, and runs automated unit and integration tests. Static code analysis checks code quality and common security issues. If any step fails, the pipeline stops and notifies the team.

3) Containerization: After a successful build, the application is packaged as a Docker image together with its runtime and dependencies. The image is tagged with the build version and pushed to a Docker Hub repository, which serves as the image registry.

4) Infrastructure as Code: The AWS environment, including the virtual network, subnets, security groups, compute resources, load balancer, and database, is defined in code and created through a tool such as Terraform. This ensures that development, staging, and production environments are identical and can be rebuilt on demand.

5) Continuous Delivery and Deployment: Jenkins pulls the new container image from Docker Hub and deploys it to the staging environment for verification and then to production behind a load balancer, using a rolling update so that the application remains available. If health checks fail, the pipeline rolls back to the previous image.

6) Monitoring and Feedback: Application logs and infrastructure metrics are collected in a monitoring service such as Amazon CloudWatch. Alarms notify the team of errors or abnormal load, and this feedback is used to improve subsequent releases.

The main advantages of this design are faster and more frequent releases, consistent environments, reduced human error, easy rollback, scalability through the cloud, and continuous visibility into system health.

VII. FUTURE SCOPE

The platform can be extended in several directions. Container orchestration with Kubernetes, for example through Amazon EKS, would provide automatic scaling and self-healing for larger workloads. Security can be strengthened by integrating automated vulnerability scanning of images and dependencies and secrets management, moving toward a DevSecOps model. Advanced release strategies such as blue-green and canary deployments could further reduce release risk. Monitoring can be enhanced with distributed tracing and AI-based anomaly detection, and cost optimization techniques such as autoscaling policies and spot instances can reduce cloud expenses. Finally, multi-cloud or hybrid support would reduce dependence on a single provider.

VIII. CONCLUSION

This project presented the design and implementation of a DevOps automation platform that integrates continuous integration, continuous delivery, and cloud deployment for an online shopping application on AWS. By automating build, test, containerization, infrastructure provisioning, deployment, and monitoring, the platform replaces slow and error-prone manual releases with a repeatable pipeline. The approach improves release speed, environment consistency, reliability, and scalability, and it provides a practical model for delivering e-commerce and other web applications on the cloud.

IX. REFERENCES

[1] Shahin, M., Babar, M. A., & Zhu, L. (2017). “Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices.” IEEE Access, 5, 3909-3943.

[2] Leite, L., Rocha, C., Kon, F., Milojicic, D., & Meirelles, P. (2019). “A Survey of DevOps Concepts and Challenges.” ACM Computing Surveys, 52(6), Article 127.

[3] Jabbari, R., bin Ali, N., Petersen, K., & Tanveer, B. (2016). “What is DevOps? A Systematic Mapping Study on Definitions and Practices.” Proceedings of the Scientific Workshop at XP2016, Article 12.

[4] Pahl, C., Brogi, A., Soldani, J., & Jamshidi, P. (2019). “Cloud Container Technologies: A State-of-the-Art Review.” IEEE Transactions on Cloud Computing, 7(3), 677-692.

[5] Burns, B., Grant, B., Oppenheimer, D., Brewer, E., & Wilkes, J. (2016). “Borg, Omega, and Kubernetes.” ACM Queue, 14(1), 70-93.

[6] Forsgren, N., Humble, J., & Kim, G. (2018). Accelerate: The Science of Lean Software and DevOps. IT Revolution Press.

[7] Humble, J., & Farley, D. (2010). Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Addison-Wesley.

[8] Kim, G., Humble, J., Debois, P., & Willis, J. (2016). The DevOps Handbook. IT Revolution Press.

[9] Morris, K. (2016). Infrastructure as Code: Managing Servers in the Cloud. O’Reilly Media.

[10] Fowler, M. (2006). “Continuous Integration.” martinfowler.com.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)