IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 1, January 2026

DOI: 10.17148/IJARCCE.2026.151124

CI/CD PIPELINE AND DEPLOYMENTAUTOMATION FOR ECOMMERCE APPLICATION

M Bhuvan[1], Suma N R[2]

Department of MCA, BIT, K.R. Road, V.V. Pura, Bangalore, India[1]

Assistant Professor, Department of MCA, BIT, K.R. Road, V.V. Pura, Bangalore, India[2]

Abstract: Modern web applications demand rapid releases, consistent deployments, and strong security practices. Manual build and deployment approaches often lead to configuration drift, delayed delivery, and increased chances of human error. This paper presents the design and implementation of a CI/CD pipeline for a MERN-based e-commerce application using Jenkins and AWS, enhanced with DevSecOps practices. The proposed pipeline automates the complete workflow from source code integration to containerized deployment. It includes static code quality validation using SonarQube, vulnerability assessment using Trivy, container image packaging using Docker, private image management through AWS Elastic Container Registry (ECR), and deployment to AWS EC2. Additionally, monitoring and observability are improved using AWS CloudWatch and alerting mechanisms through Slack notifications. The implementation ensures faster and repeatable deployments, improved code reliability, and early identification of security issues, making the system suitable for real-world production workflows..

Keywords: CI/CD, DevOps, DevSecOps, Jenkins, Docker, SonarQube, Trivy, AWS EC2, AWS ECR, CloudWatch, MERN Stack, E-Commerce

I. INTRODUCTION

In recent years, the rapid growth of web-based applications and digital commerce platforms has significantly increased the demand for reliable, scalable, and continuously evolving software systems. E-commerce applications, in particular, require frequent feature updates, security patches, performance enhancements, and bug fixes to remain competitive and meet changing customer expectations. Traditional software development and deployment practices, which rely heavily on manual processes, often lead to delayed releases, configuration errors, and inconsistent application behavior across environments. These challenges highlight the need for automated and systematic approaches to software delivery.

Continuous Integration and Continuous Deployment (CI/CD) has emerged as a key practice within DevOps that addresses these limitations by automating the build, testing, security scanning, and deployment stages of the software lifecycle. CI/CD enables development teams to integrate code changes frequently, detect issues early, and deliver updates to production environments with minimal manual intervention. By reducing human dependency and standardizing workflows, CI/CD pipelines improve software quality, deployment reliability, and release velocity.This project focuses on designing and implementing a robust CI/CD pipeline for a full-stack e-commerce application built using the MERN (MongoDB, Express.js, React.js, Node.js) stack. The application simulates real-world e-commerce functionalities such as user authentication, product management, cart operations, order processing, and payment gateway integration. The CI/CD pipeline automates the complete software delivery process, starting from source code version control to production deployment on cloud infrastructure.

The proposed system integrates modern DevOps tools and practices, including GitHub for source control, Jenkins for pipeline orchestration, Docker for containerization, SonarQube for static code quality analysis, Trivy for container vulnerability scanning, AWS EC2 for hosting, and CloudWatch for monitoring and logging. Security and quality gates are enforced at multiple stages to ensure that only validated and compliant builds are deployed. Additionally, real-time notifications through Slack enhance visibility and rapid response to pipeline events.By implementing this CI/CD-driven deployment automation, the project demonstrates how e-commerce platforms can achieve faster release cycles, improved reliability, enhanced security, and better scalability. The study highlights the practical benefits of DevOps adoption in modern software engineering and provides a structured approach that can be extended to enterprise-level applications. This work serves as a reference model for integrating CI/CD pipelines into full-stack web applications, emphasizing automation, quality assurance, and continuous improvement.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 1, January 2026

DOI: 10.17148/IJARCCE.2026.151124

1.1 Project Description

The project titled "CI/CD Pipeline and Deployment Automation for E-Commerce Application" focuses on developing a full-stack e-commerce web application and implementing a complete DevOps-based automation workflow for its deployment. The e-commerce application is built using the MERN stack (MongoDB, Express.js, React.js, Node.js) and includes core features such as user authentication, product listing, category filtering, cart management, order processing, admin dashboard, and payment gateway integration (Razorpay/Paytm in test mode).Along with application development, the major highlight of this project is the implementation of a Continuous Integration and Continuous Deployment (CI/CD) pipeline to automate the entire software delivery process. The pipeline integrates tools such as GitHub for version control, Jenkins for pipeline execution, Docker for containerization, SonarQube for code quality analysis, and Trivy for vulnerability scanning. The deployment is performed on cloud infrastructure such as AWS EC2, while CloudWatch monitoring is used for tracking logs, server health, and application performance. This automated workflow ensures that every code change is verified, tested, scanned, and deployed with minimal manual effort, providing a reliable and production-oriented deployment process.

1.2 Motivation

Modern software applications, especially e-commerce platforms, require frequent updates, fast releases, and high availability to meet customer expectations and remain competitive. However, in many traditional development environments, deployments are done manually, which leads to issues such as human errors, inconsistent builds, deployment failures, longer release cycles, and lack of monitoring. These challenges become more critical when the application grows and multiple developers work on the same codebase.The motivation behind this project is to solve these problems by adopting a DevOps-based CI/CD approach that automates and standardizes the deployment process. By building an automated pipeline, the project ensures faster delivery of features, early detection of bugs, improved security through scanning tools, and better reliability in production deployment. This project is also motivated by real-world industry practices, where CI/CD pipelines have become essential for maintaining software quality and enabling continuous improvements. Overall, the goal is to demonstrate how an e-commerce application can be enhanced with deployment automation to achieve efficient software releases, better scalability, and improved system stability.

II. RELATED WORK

Paper [1] discusses how Continuous Integration and Continuous Deployment (CI/CD) pipelines improve software release speed and reliability by automating build, test, and deployment stages. The study highlights that automated pipelines reduce human errors in production deployments and ensure consistent delivery across environments, especially for web applications with frequent updates.

Paper [2] presents container-based deployment practices using Docker and emphasizes how containerization ensures portability and consistent execution of applications across development, testing, and production environments. The work explains that container images help reduce "works on my machine" issues, which is a common challenge in full-stack application deployment.

Paper [3] explains the importance of DevSecOps integration in CI/CD workflows by introducing security scanning tools within the pipeline. It demonstrates how static code analysis and vulnerability scanning during the build stage can prevent insecure code and vulnerable dependencies from reaching production systems, improving overall software security posture.

Paper [4] focuses on implementing automated deployment pipelines using Jenkins and demonstrates how Jenkins pipelines support structured workflows such as source checkout, compilation, testing, artifact generation, and deployment. The study also emphasizes the role of pipeline logs and reporting in identifying failures quickly and improving traceability in software delivery.

Paper [5] highlights cloud-based deployment strategies for scalable web applications using services such as virtual machines, container registries, and monitoring tools. The paper shows how cloud platforms support scalable hosting, centralized monitoring, and controlled access management, which are essential for deploying modern e-commerce applications reliably.

III. METHODOLOGY

A. System Overview

The proposed system is a full-stack e-commerce web application developed using the MERN stack and enhanced with an automated CI/CD pipeline for continuous delivery. The application supports core e-commerce features such as user authentication, product listing and search, cart management, order placement, payment workflow, and admin-side product and order management. To reduce manual deployment effort and improve reliability, the project integrates a DevOps-

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 1, January 2026

DOI: 10.17148/IJARCCE.2026.151124

based automation workflow where every code update is validated through build, quality checks, security scans, containerization, and deployment stages. This methodology ensures that the application remains stable, secure, and consistently deployable across environments**.**

B. Architecture Design

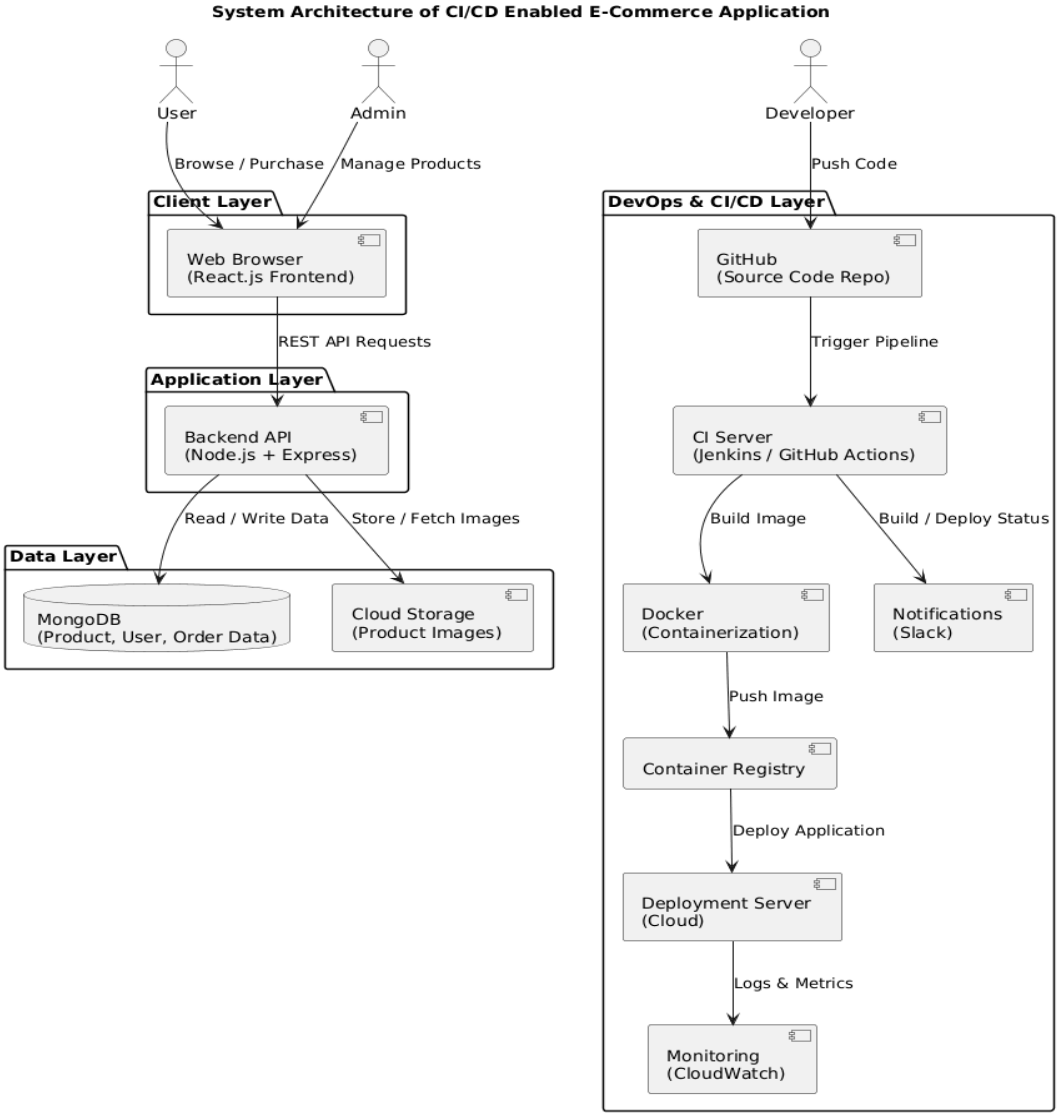
The system is designed using a modular and layered architecture to ensure separation of concerns and scalability. The major layers include:

Frontend Layer (React.js): Responsible for rendering the user interface, managing client-side routing, displaying products, and handling cart and checkout interactions.

Backend Layer (Node.js + Express.js): Provides RESTful APIs for authentication, product management, order processing, and payment handling.

Database Layer (MongoDB Atlas): Stores structured collections such as users, products, orders, reviews, and transaction data.

DevOps Automation Layer: Implements continuous integration and continuous deployment workflows with quality and security gates to automate the release process.



This architecture supports independent development of modules and allows the deployment workflow to validate each layer without breaking system functionality.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 1, January 2026

DOI: 10.17148/IJARCCE.2026.151124

C. CI/CD Pipeline Workflow

The CI/CD pipeline is designed to automate the process of integrating, validating, and deploying changes with minimal human intervention. The workflow is triggered whenever a developer pushes updates to the GitHub repository. The pipeline includes the following stages:

1. Source Code Management (GitHub):

All source code is maintained in a centralized GitHub repository. Each commit or merge triggers the pipeline, ensuring that updates are automatically processed.

2. Continuous Integration (Jenkins Build Stage):

Jenkins pulls the latest code, installs required dependencies, and validates the build process. This confirms that the application compiles correctly and is ready for further checks.

3. Automated Testing & Build Validation:

Basic test execution and build verification steps are performed to confirm that the application remains stable after changes. This reduces the risk of deployment failures.

4. Static Code Quality Analysis (SonarQube):

SonarQube is integrated into the pipeline to perform code analysis. It identifies maintainability issues such as code smells, duplication, bugs, and reliability concerns. A quality gate is applied to stop deployment if the code fails to meet defined standards.

5. Security & Vulnerability Scanning (Trivy):

Trivy scans both application dependencies and Docker images to detect vulnerabilities. This ensures that insecure packages or high-risk dependencies are caught before deployment.

6. Containerization (Docker Build Stage):

The application is packaged into Docker images, providing environment consistency and reducing “works on my machine” issues. Containerization ensures the same build runs reliably across development and production.

7. Container Registry Integration:

The validated Docker images are pushed to a container registry. This provides versioned and reusable artifacts for deployment.

8. Continuous Deployment (AWS Deployment Stage):

The latest approved Docker image is pulled and deployed to the hosting environment. Deployment automation reduces downtime and ensures fast release cycle

D. Infrastructure & Deployment Setup

The application is deployed on cloud infrastructure to enable scalability and real-world accessibility. The deployment environment includes:

AWS EC2 Instance: Used as the compute environment for running application containers. Security Groups and Port Configuration: Ensures secure inbound access for the application and restricted administrative access .Environment Configuration Management: Sensitive values such as database credentials, JWT secret, and payment keys are stored using secure environment variables and pipeline credential storage mechanisms. Container-Based Deployment Strategy: Enables predictable releases and simplifies rollback if required. This setup ensures stable deployment with better performance and controlled access.

E. Monitoring and Notification

To maintain reliability and traceability, monitoring and notifications are integrated into the pipeline and deployment environment AWS CloudWatch Monitoring: Tracks server metrics, resource usage, and logs to identify runtime issues early.Pipeline Logs and Reports: Jenkins maintains execution logs for every build, helping in debugging and

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 1, January 2026

DOI: 10.17148/IJARCCE.2026.151124

auditing.Slack Notifications: Build, test, and deployment results are communicated instantly to the team for faster response to failures.This provides continuous observability, ensuring system health and deployment transparency.

IV. SIMULATION AND EVALUATION FRAMEWORK

To validate the effectiveness of the proposed CI/CD-enabled DevOps pipeline for the MERN-based e-commerce application, a controlled simulation was conducted using a staging-like environment. The evaluation framework focuses on measuring automation reliability, code quality improvement, security enforcement, deployment consistency, and monitoring effectiveness. The framework ensures that the system is not only functionally deployable but also adheres to modern DevSecOps standards such as quality gating, vulnerability scanning, and continuous observability.

A. Simulation Setup

The simulation environment was created to replicate a real-world development and deployment workflow where multiple code changes are committed and deployed continuously. The application source code is hosted in a GitHub repository and integrated with Jenkins for pipeline execution. Docker is used for packaging the application into portable images, enabling consistent deployments across environments. AWS EC2 is used as the deployment server and CloudWatch is used to capture runtime logs and metrics.

To represent realistic development scenarios, multiple change cycles were simulated, such as UI updates, API modifications, and configuration changes. Each change triggered an automated CI/CD pipeline run to test how the system behaves under repeated builds and deployments.

B. Pipeline Simulation Workflow

The pipeline simulation follows a sequential execution model where each stage must succeed before proceeding to the next stage. The simulated workflow includes:

1. Source Code Commit Simulation

Developers push updates to GitHub, triggering the pipeline automatically through webhook integration.

2. Build and Dependency Validation

Jenkins checks out the latest code and installs required dependencies to validate build readiness.

3. Static Code Quality Analysis (SonarQube Gate)

SonarQube scans the project to detect code smells, duplicated logic, maintainability issues, and potential bugs. A quality gate threshold is applied so that deployments are blocked if minimum standards are not satisfied.

4. Security Scanning (Trivy Gate)

Trivy scans the Docker image and application dependencies to detect vulnerabilities. This ensures that insecure packages or known CVEs do not enter the deployment stage.

5. Container Build and Registry Push

A Docker image is built and pushed to the container registry (DockerHub or private registry). Version tagging is applied to maintain traceability.

6. Deployment to AWS EC2

The latest verified image is pulled on the EC2 server and deployed using container runtime commands. This stage confirms that deployment is automated and repeatable.

7. Monitoring and Notification Validation

CloudWatch logs are monitored for application health. Slack notifications confirm pipeline status (success/failure) for quick feedback.

C. Evaluation Metrics

The performance of the pipeline was evaluated using measurable parameters commonly used in DevOps and software delivery assessments:

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 1, January 2026

DOI: 10.17148/IJARCCE.2026.151124

Build Success Rate: Percentage of pipeline runs completed without build errors.

Deployment Success Rate: Percentage of successful deployments without rollback or runtime crash.

Pipeline Execution Time: Total time taken from code commit to deployment completion.

Quality Gate Pass Rate: Number of commits meeting SonarQube quality standards.

Vulnerability Detection Rate: Number of vulnerabilities detected and prevented before deployment.

Mean Time to Detect (MTTD): Time required to identify pipeline or deployment failures through monitoring.

Mean Time to Recover (MTTR): Time taken to fix and redeploy after pipeline failure

Environment Consistency: Deployment reproducibility due to containerization.

These metrics collectively demonstrate the efficiency and robustness of the proposed CI/CD framework.

D. Test Scenarios Considered

The evaluation was performed under multiple simulated scenarios to test pipeline behavior:

Scenario 1: Normal Code Update

Minor frontend or backend changes are committed and deployed successfully.

Scenario 2: Code Quality Violation

Introducing unused variables or duplicated blocks triggers SonarQube quality gate failure, blocking deployment.

Scenario 3: Security Vulnerability Injection

Adding vulnerable dependencies causes Trivy to flag the image, preventing it from being deployed.

Scenario 4: Deployment Failure Case

Incorrect environment configuration or missing variables triggers runtime errors, verified through CloudWatch logs and Slack alerts.

Scenario 5: Repeated Continuous Commits

Multiple commits were pushed in short intervals to test pipeline stability and automation consistency.

E. Result Validation Approach

The results were validated using both pipeline logs and deployment verification checks:Jenkins console logs were used to confirm each pipeline stage completion.SonarQube dashboard reports were reviewed to confirm quality gate status.Trivy scan reports were used to verify vulnerability detection and enforcement.Docker image tags and registry push history ensured artifact traceability.AWS EC2 container logs and CloudWatch metrics verified runtime health.Slack alerts confirmed real-time pipeline feedback.

This evaluation approach ensures that the system is validated not only for deployment automation but also for security and operational readiness.

F. Outcome Summary

The simulation confirms that the proposed pipeline successfully automates the build-test-scan-deploy workflow and reduces manual deployment efforts. By integrating quality and security gates, the system ensures that only verified and secure builds reach production. The monitoring and alerting layer enhances reliability by enabling faster failure detection and resolution. Overall, the evaluation demonstrates that the proposed DevSecOps-based CI/CD pipeline improves delivery speed, deployment consistency, and application trustworthiness.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 1, January 2026

DOI: 10.17148/IJARCCE.2026.151124

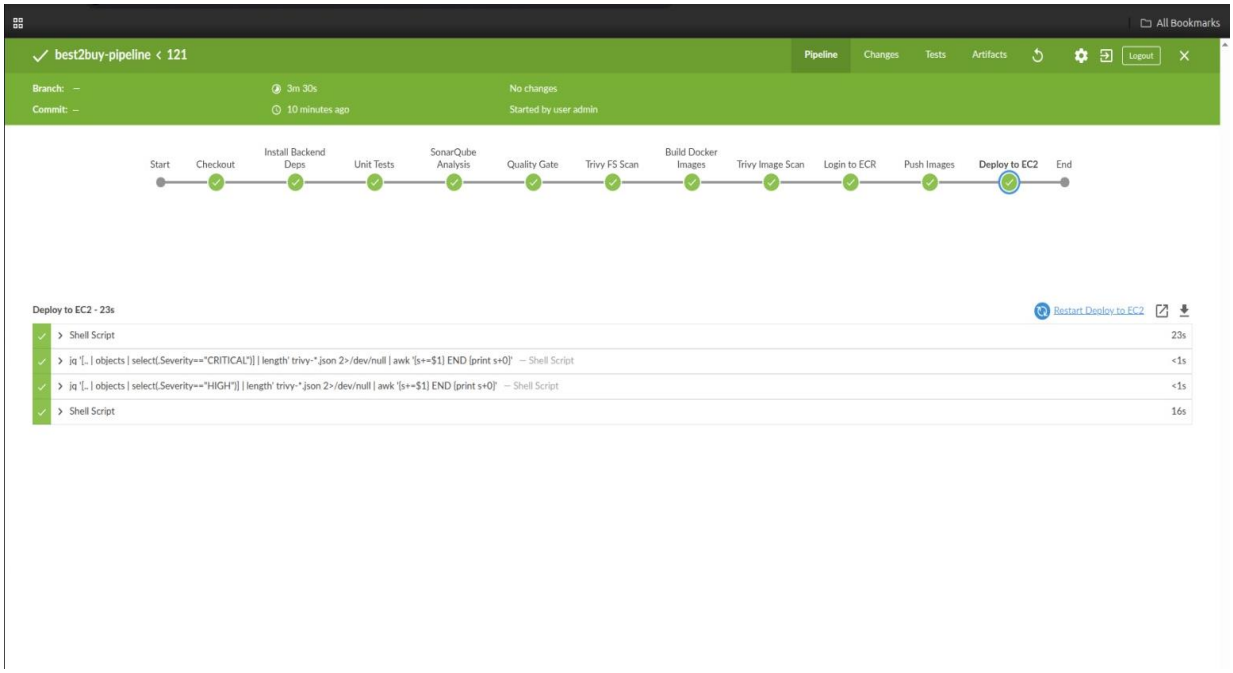


Fig 1. 6 Jenkins Blue Ocean Pipeline View illustrating end-to-end CI/CD stages.

Fig 2. Netdata Monitoring Dashboard showing real-time system metrics

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 1, January 2026

DOI: 10.17148/IJARCCE.2026.151124

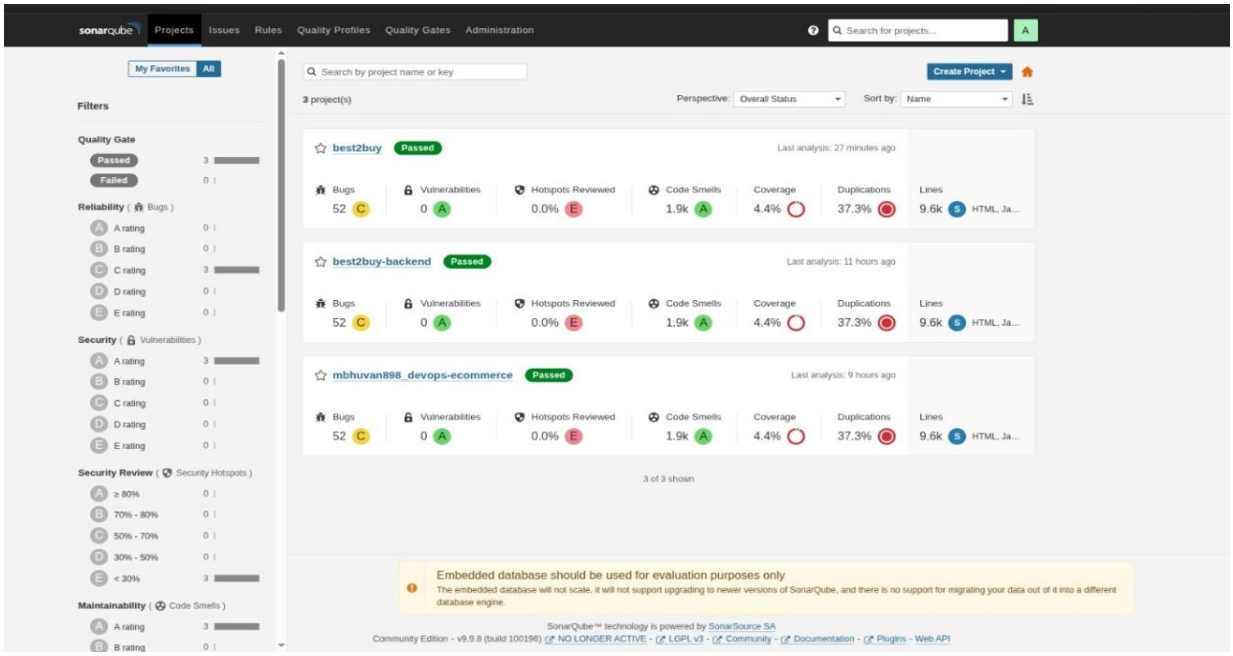


Fig 3. SonarQube Project Overview showing Quality Gate status and static code analysis results.

V.RESULTS AND DISCUSSION

The proposed CI/CD-enabled deployment automation framework was implemented and tested using a MERN-based e-commerce application as the target workload. The results indicate that the automated pipeline significantly improves release consistency by enforcing a structured build–test–scan–deploy workflow. Jenkins successfully triggered pipeline execution on every source code update, enabling continuous integration and reducing the dependency on manual deployment procedures. This ensured that application updates were delivered in a controlled and repeatable manner.

Code quality evaluation using SonarQube highlighted key maintainability aspects of the application. The quality analysis helped detect issues such as code smells, duplicated logic, and potential bug-prone segments. By applying quality gate conditions, the pipeline prevented unstable builds from proceeding further, improving overall software reliability. This step was effective in standardizing coding practices and maintaining a clean codebase over multiple iterations of development.

From a security perspective, Trivy-based scanning was integrated into the pipeline to identify vulnerabilities in dependencies and container images. The scanning results demonstrated that security checks can be automated without significantly affecting pipeline execution flow. Vulnerability detection before deployment reduced the risk of deploying insecure builds and ensured that container images remained compliant with baseline security requirements. This confirms the importance of integrating DevSecOps practices into modern CI/CD workflows.

Docker containerization improved deployment portability by packaging the application with all necessary dependencies into consistent runtime images. The container-based deployment approach minimized environment mismatch issues and supported predictable deployment outcomes. The Docker images were deployed to AWS EC2 instances, where application availability and runtime stability were observed. Monitoring through Netdata and CloudWatch provided real-time insights into system health, resource consumption, and service behavior. The monitoring outputs confirmed stable performance under normal workload conditions and enabled quick identification of operational bottlenecks.

Overall, the evaluation demonstrates that the proposed framework enhances deployment efficiency, strengthens code quality assurance, and introduces automated security validation. The results support the conclusion that CI/CD pipelines combined with quality and security gates can deliver faster and more reliable releases compared to conventional manual deployment models.

International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 1, January 2026
DOI: 10.17148/IJARCCE.2026.151124

VI.CONCLUSION

This work presented an automated CI/CD pipeline framework for a MERN-based e-commerce application, integrating continuous integration, containerized deployment, and automated quality and security checks. The implementation using Jenkins, SonarQube, Trivy, Docker, and AWS services demonstrated improved reliability and consistency in software delivery. The framework ensures that each release undergoes structured validation before deployment, reducing deployment errors and improving operational stability. The results confirm that adopting DevOps automation practices can enhance both development efficiency and production readiness in modern web applications.

VII.FUTURE WORK

Future enhancements can extend the proposed framework by incorporating Kubernetes-based orchestration to achieve high availability, auto-scaling, and improved fault tolerance. Infrastructure provisioning can be automated using Infrastructure as Code tools such as Terraform or AWS CloudFormation to standardize environment setup. The testing phase can be expanded by integrating automated unit, integration, and UI testing to strengthen release validation. A GitOps-driven deployment approach using ArgoCD can further improve deployment traceability and version-controlled infrastructure updates. In addition, advanced observability using Prometheus and Grafana can provide deeper performance analytics and alerting capabilities, supporting large-scale production deployments.

REFERENCES

[1] Amazon Web Services, AWS CloudWatch User Guide, 2025. [Online]. Available: docs.aws.amazon.com/cloudwatch/. Accessed: 22-Dec-2025.
[2] Aqua Security, Trivy Documentation, 2025. [Online]. Available: aquasecurity.github.io/trivy/. Accessed: 22-Dec-2025.
[3] L. Bass, I. Weber, and L. Zhu, DevOps: A Software Architect's Perspective, Addison-Wesley, 2015.
[4] Docker Inc., Docker Documentation, 2025. [Online]. Available: docs.docker.com/. Accessed: 22-Dec-2025.
[5] OpenJS Foundation, Express.js Documentation, 2025. [Online]. Available: expressjs.com/. Accessed: 22-Dec-2025.
[6] M. Fowler, "Continuous Integration," MartinFowler.com, 2025. [Online]. Available: martinfowler.com/articles/continuousIntegration.html. Accessed: 22-Dec-2025.
[7] GitHub Inc., GitHub Actions Documentation, 2025. [Online]. Available: docs.github.com/en/actions. Accessed: 22-Dec-2025.
[8] Google Cloud, DevOps Practices and Principles, 2025. [Online]. Available: cloud.google.com/devops. Accessed: 22-Dec-2025.
[9] HashiCorp, "Infrastructure as Code Concepts," 2025. [Online]. Available: www.hashicorp.com/resources/what-is-infrastructure-as-code. Accessed: 22-Dec-2025.
[10] J. Humble and D. Farley, Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation, Addison-Wesley, 2011.
[11] Jenkins Project, Jenkins User Documentation, 2025. [Online]. Available: www.jenkins.io/doc/. Accessed: 22-Dec-2025.
[12] Cloud Native Computing Foundation, Kubernetes Documentation, 2025. [Online]. Available: kubernetes.io/docs/. Accessed: 22-Dec-2025.
[13] MongoDB Inc., MongoDB Manual, 2025. [Online]. Available: www.mongodb.com/docs/. Accessed: 22-Dec-2025.
[14] Mozilla Developer Network, JavaScript Documentation, 2025. [Online]. Available: developer.mozilla.org/en-US/docs/Web/JavaScript. Accessed: 22-Dec-2025.
[15] National Institute of Standards and Technology (NIST), Security and Privacy Controls for Information Systems, 2025. [Online]. Available: www.nist.gov/. Accessed: 22-Dec-2025.
[16] OpenJS Foundation, Node.js Documentation, 2025. [Online]. Available: nodejs.org/en/docs/. Accessed: 22-Dec-2025.
[17] OWASP Foundation, OWASP Top Ten Web Application Security Risks, 2025. [Online]. Available: owasp.org/www-project-top-ten/. Accessed: 22-Dec-2025.
[18] Razorpay Software Pvt. Ltd., Razorpay Developer Guide, 2025. [Online]. Available: razorpay.com/docs/. Accessed: 22-Dec-2025.
[19] Meta Platforms Inc., React Documentation, 2025. [Online]. Available: react.dev/. Accessed: 22-Dec-2025.
[20] Red Hat, "What is CI/CD?," 2025. [Online]. Available: www.redhat.com/en/topics/devops/what-is-ci-cd. Accessed: 22-Dec-2025.
[21] Slack Technologies, Slack API Documentation, 2025. [Online]. Available: api.slack.com/. Accessed: 22-Dec-2025.
[22] SonarSource SA, SonarQube Documentation, 2025. [Online]. Available: docs.sonarsource.com/sonarqube/. Accessed: 22-Dec-2025.