Automated CI/CD Pipeline with Jenkins and DevSecOps Integration for Application Deployment

Prajwal Patelpaik[1], Smit Raut[2], Arya Kasliwal[3], Srushti Dagade[4], Prof. Shantanu Pawar[5]
[1,2,3,4,5]*Department of Information Technology, P.E.S. Modern College of Engineering, Savitribai Phule Pune University, Pune – 411005, Maharashtra, India*

***Abstract*—The increasing frequency of cyberattacks on modern enterprise applications has made security a critical requirement throughout the software development lifecycle. Traditional DevOps practices often prioritize delivery speed and automation while addressing security concerns only during later stages of development, resulting in delayed vulnerability detection, increased remediation costs, and greater production risks. This paper presents the design and implementation of a DevSecOps-driven secure software delivery pipeline for the Ekart Java-based e-commerce application deployed on AWS Elastic Kubernetes Service (EKS). The proposed solution integrates security controls directly into the CI/CD workflow by adopting a shift-left security approach. Automated static application security testing (SAST) is performed using SonarQube with mandatory quality gate enforcement, while OWASP Dependency-Check continuously scans third-party libraries for known vulnerabilities. Secure artifact management is implemented through Nexus Repository Manager, and containerized workloads are deployed on Kubernetes after passing predefined security and quality validations. Infrastructure provisioning is automated using Terraform, ensuring reproducible, secure, and policy-compliant cloud environments. By embedding security, compliance, and governance controls throughout the pipeline, the proposed DevSecOps framework prevents vulnerable code and dependencies from reaching production. Experimental evaluation demonstrates improved vulnerability detection, reduced security review effort, enhanced deployment reliability, and faster delivery of secure software releases. The proposed approach provides a scalable and practical blueprint for implementing DevSecOps in cloud-native enterprise applications.**

***Index Terms*—AWS EKS, CI/CD Pipeline, DevSecOps, Docker, Infrastructure as Code, Jenkins, Kubernetes, Nexus Repository, OWASP Dependency-Check, Shift-Left Security, SonarQube, Terraform**

I. INTRODUCTION

The rapid expansion of digital commerce has significantly increased the complexity and security requirements of modern enterprise applications. E-commerce platforms such as Ekart manage sensitive customer information, user authentication, order processing, and payment transactions, making them attractive targets for cyberattacks. As organizations accelerate software delivery to remain competitive, ensuring security throughout the software development lifecycle has become a critical challenge. Traditional development and deployment practices often treat security as a final-stage activity, resulting in delayed vulnerability detection, increased remediation costs, and a higher risk of security breaches in production environments.

DevSecOps has emerged as a transformative approach that integrates security into every phase of software development and delivery. By adopting a shift-left security model, DevSecOps enables organizations to identify and remediate vulnerabilities early through automated security testing, continuous monitoring, and policy enforcement within the CI/CD pipeline. This integration promotes collaboration among development, security, and operations teams while ensuring that security and compliance requirements are continuously validated throughout the release process.

In addition to security integration, Infrastructure as Code (IaC) technologies such as Terraform enable organizations to provision and manage cloud infrastructure in a consistent, reproducible, and secure manner. Automated infrastructure provisioning minimizes configuration drift, enforces standardized security baselines, and enhances operational reliability across environments.

This paper presents the design and implementation of a DevSecOps-driven secure software delivery pipeline for the Ekart Java-based e-commerce application deployed on AWS Elastic Kubernetes Service (EKS). The proposed framework integrates Jenkins, SonarQube, OWASP Dependency-Check, Nexus Repository Manager, Docker, and Kubernetes to automate application delivery while enforcing security and quality controls at every stage of the pipeline. Security mechanisms including static code analysis, vulnerability scanning, quality gate enforcement, secure artifact management, and automated deployment are incorporated to ensure that only validated and compliant software reaches production environments.

The remainder of this paper is organized as follows: Section II reviews the related literature; Section III presents the problem statement; Section IV describes the proposed DevSecOps architecture; Section V details the implementation methodology; Section VI discusses the results and evaluation; Section VII outlines future enhancements; and Section VIII concludes the paper.

II. LITERATURE SURVEY

The literature consistently highlights that manual processes are inadequate for modern software delivery at scale. Table I summarizes key related works informing this research and justifying the technologies adopted.

Table I: Summary of Related Works

Reference	Focus Area	Key Contribution
Singh et al., 2023 [1]	CI/CD for web apps	Automation reduces deployment defects and accelerates release cycles.
Saleh et al., 2024 [2]	Security in CI/CD (SLR)	Identifies unauthorized access and image manipulation as top threats; supports DevSecOps.
Bavadiya, 2023 [3]	Security-as-Code	Reports 77.4% improvement in vulnerability detection with only 5% pipeline overhead.
Pahl et al., 2025 [4]	IaC/Terraform review	Confirms Terraform-based IaC guarantees reproducible, drift-free environments.
Chavan et al., 2022 [5]	DevSecOps pipeline	Jenkins + SonarQube + Docker + Kubernetes delivers secure automated delivery at scale.
Abdiukov, 2024 [6]	AI-based security testing	SAST/DAST/IAST with AI reduce false positives; introduces intelligent vulnerability prioritization.
Pando & Davila, 2023 [7]	Software testing in DevOps (SLR)	Automated tests are backbone of CI; regression testing critical for e-commerce.

III. PROBLEM STATEMENT

Modern enterprise e-commerce applications such as Ekart face increasing cybersecurity threats due to the rapid pace of software delivery. Traditional CI/CD pipelines primarily emphasize automation and deployment speed while treating security as a separate post-development activity. This approach results in vulnerabilities being detected late in the software lifecycle, increasing remediation costs and exposing production environments to security risks.

The existing software delivery process lacks integrated security controls such as automated static code analysis, dependency vulnerability scanning, container security validation, and policy enforcement within the CI/CD workflow. Additionally, manual infrastructure provisioning and deployment activities introduce configuration drift, inconsistent security policies, credential management issues, and compliance challenges.

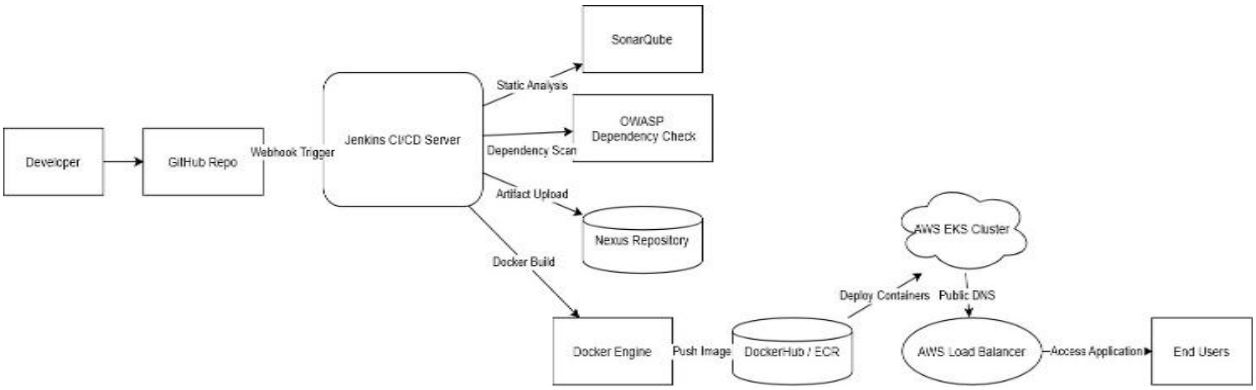
Therefore, there is a need for a DevSecOps-driven automated pipeline that embeds security testing, vulnerability assessment, compliance validation, and infrastructure automation throughout the software development lifecycle. Such a system should continuously build, test, scan, package, and deploy applications while enforcing security gates before production deployment, thereby reducing

vulnerability exposure, improving compliance, and enabling secure and reliable software delivery.

IV. PROPOSED SYSTEM ARCHITECTURE

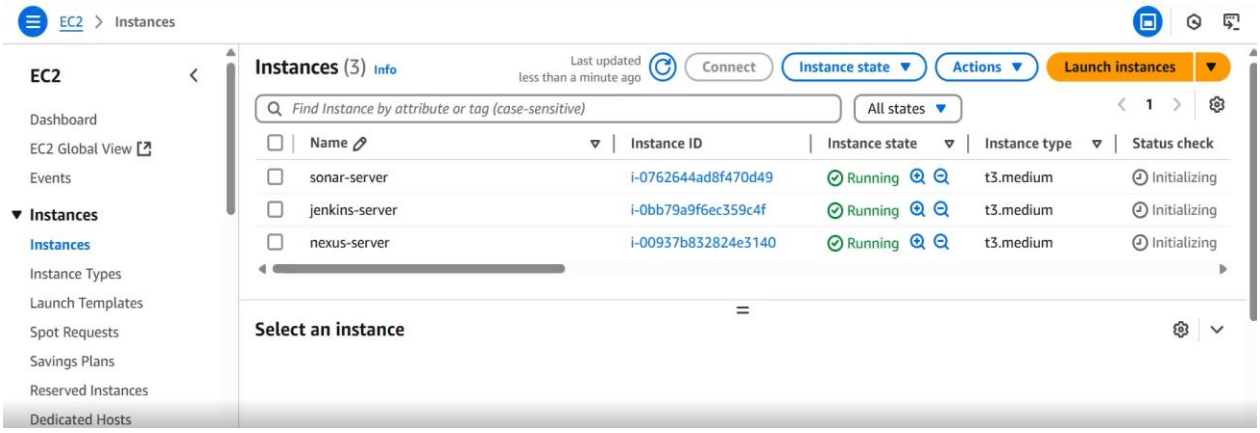
A. Architecture Overview

The proposed architecture establishes a fully automated CI/CD pipeline using Jenkins as the central orchestration engine. The end-to-end pipeline workflow operates as follows: A developer pushes source code to GitHub, automatically triggering the Jenkins pipeline via a configured webhook. Jenkins clones the repository and initiates static code analysis via SonarQube; if the quality gate fails, the pipeline halts and notifies the developer by email. Upon successful analysis, Maven compiles the Java application and executes unit tests, generating a versioned .jar artifact. The artifact is uploaded to Nexus Repository Manager for centralized, versioned storage. Jenkins builds a Docker image and pushes it to DockerHub. Jenkins applies Kubernetes deployment manifests to the AWS EKS cluster using a rolling update strategy for zero-downtime deployment. End users access the live application through an AWS Application Load Balancer on port 8070.



B. Infrastructure Provisioning via Terraform

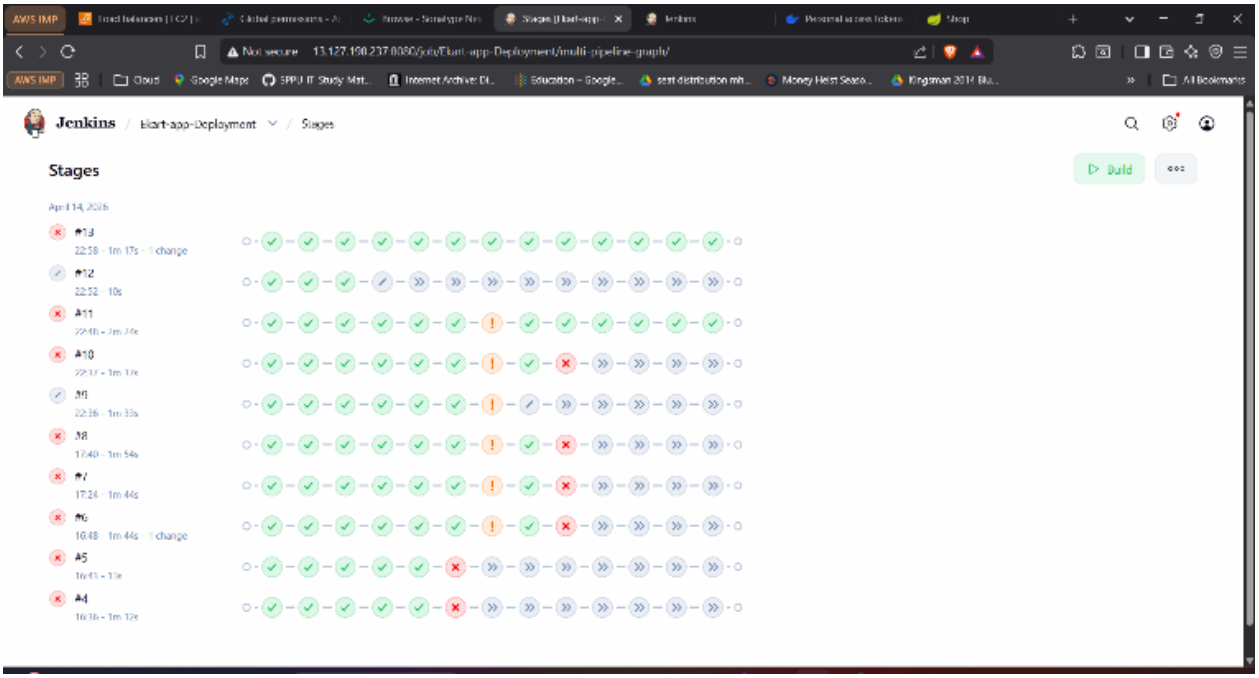
All AWS infrastructure is provisioned using Terraform scripts stored in version-controlled GitHub repositories. Two repositories manage distinct layers: Repository 1 (jenkins-sonar-nexus) provisions Jenkins (t3. medium, Ubuntu 22.04, port 8080), SonarQube and 10–15 minutes for the EKS cluster. (t3. medium, port 9000), and Nexus (t3. medium, port 8081) in the AWS ap-south-1 region. Repository 2 (eks-cluster-deployment) provisions the AWS EKS Kubernetes cluster with 2–3 worker nodes (t3. medium). Full infrastructure creation completes in approximately 20 minutes: 5 minutes for tool servers



C. DevSecOps Security Integration

Security is embedded at multiple pipeline checkpoints. SonarQube (port 9000) performs static analysis for bugs, vulnerabilities, code smells, and security hotspots; the pipeline halts on a FAILED quality gate. OWASP Dependency-Check (plugin v6.5.1) scans all

Maven dependencies against the NVD CVE database, blocking critical severity findings. Jenkins Credentials Store manages SonarQube tokens (ID: sonar-token) and DockerHub access tokens (ID: dockerhub-pwd) as Secret Text, ensuring credentials are never hardcoded in the Jenkinsfile. AWS IAM roles assign least-privilege permissions to EKS worker nodes and Jenkins EC2 instances.



V. SYSTEM IMPLEMENTATION

A. Cloud Infrastructure Configuration

Table II: Cloud Infrastructure (AWS EC2 Instances)

Component	Instance	OS	CPU	RAM	Port
Jenkins Server	t3.medium	Ubuntu 22.04	2 vCPU	4 GB	8080
SonarQube Server	t3.medium	Ubuntu 22.04	2 vCPU	4 GB	9000
Nexus Repo Server	t3.medium	Ubuntu 22.04	2 vCPU	4 GB	8081
EKS Worker Node (×3)	t3.medium	Amazon Linux 2	2 vCPU	4 GB	Dynamic

Table III: Key Software Tools and Versions

Tool / Plugin	Version	Purpose in Pipeline
Jenkins	Latest LTS	Core CI/CD automation server
Terraform	Latest stable	IaC provisioning of AWS resources
SonarQube Scanner	Auto-install	Static code analysis and quality gate
OWASP Dependency-Check	6.5.1	Library vulnerability scanning (CVE)
Maven	3.6.3	Java build, test, and packaging
JDK	17.0.11+9	Java compilation environment
Docker	Latest	Container image build and run
Nexus Repository Manager	Docker container	Maven artifact (.jar) storage
AWS EKS	Managed	Container orchestration and deployment

B. Step-by-Step Implementation

Step 1: Infrastructure Provisioning. Jenkins, SonarQube, and Nexus servers were provisioned by executing terraform init → terraform plan → terraform apply from the jenkins-sonar-nexus repository. After provisioning, Jenkins was accessible at http://<Jenkins-IP>:8080, SonarQube at http://<Sonar-IP>:9000, and Nexus at http://<Nexus-IP>:8081.

Step 2: Jenkins Plugin Installation. Essential plugins were installed including SonarQube Scanner, Nexus Artifact Uploader, Docker Pipeline, OWASP Dependency-Check, Eclipse Temurin Installer, Pipeline Maven Integration, and Config File Provider. Tools were configured under Manage Jenkins → Global Tool Configuration.

Step 3: EKS Cluster Creation. A dedicated Jenkins pipeline job (EKS-Cluster-Creation) was configured with a parameterized Choice Parameter (apply/destroy), targeting the eks-cluster-deployment Terraform repository. Execution with ACTION: apply provisioned the full EKS cluster in 10–15 minutes.

Step 4: Credential Configuration. SonarQube and DockerHub tokens were stored as Secret Text credentials in Jenkins. SonarQube server integration was configured under Manage Jenkins → Configure System, ensuring no credentials are hardcoded in the Jenkinsfile.

Step 5: Nexus Integration. The project pom.xml was updated with Nexus repository URLs for maven-releases and maven-snapshots. A Global Maven settings.xml was configured in Jenkins with Nexus admin credentials inside the <servers> block.

Step 6: Pipeline Execution. The Ekart-app-Deployment Jenkins pipeline executed the following stages sequentially. Table IV summarizes the pipeline stages and their gates.

Table IV: Jenkins Pipeline Stages – Ekart Application

Stage	Tool	Action	Gate
Git Checkout	Jenkins SCM	Clones Ekart repo on webhook trigger	Always
Code Analysis	SonarQube	Runs static analysis; evaluates quality gate	Halt if FAILED
OWASP Scan	Dep.-Check 6.5.1	Scans Maven dependencies vs. NVD CVE DB	Halt on Critical CVE
Build & Test	Maven 3.6.3 + JDK 17	mvn clean package; compiles .jar, runs JUnit	Halt if tests fail
Artifact Upload	Nexus Uploader	Uploads versioned .jar to Nexus	Continues on success
Docker Build/Push	Docker + DockerHub	Builds image; docker push to registry	Continues on success
Deploy to EKS	kubectl + AWS EKS	Applies Kubernetes manifests; rolling update	Health check post-deploy
Notify	Jenkins Email	Sends SUCCESS or FAILURE to developer	Always

VI. RESULTS AND DISCUSSION

A. Security Gate Effectiveness

The integration of SonarQube and OWASP Dependency-Check enforced security compliance before every deployment. In early pipeline runs, SonarQube successfully detected code smells, potential null pointer exceptions, and insecure coding patterns—preventing such issues from reaching production. The OWASP plugin identified outdated third-party library versions during dependency scans. Critically, no build with a failed quality gate or a critical CVE dependency could proceed to the Docker build or EKS deployment stages.

B. Pipeline Execution Performance

The automated pipeline was executed across multiple build cycles during implementation and testing. The total pipeline execution time (code commit to deployment) was observed at 7–9 minutes for a typical build. Table V summarizes the key performance metrics observed.

Table V: Observed Pipeline Performance Metrics

Metric	Observed Value
Total pipeline execution (commit to deploy)	7–9 minutes (typical)
SonarQube analysis stage	1–2 minutes
OWASP Dependency-Check scan	2–3 minutes (first run longer)
Maven builds and unit tests	2–3 minutes
Docker image build and push	1–2 minutes
AWS EKS rolling deployment	1–2 minutes
Application downtime during update	0 seconds (zero-downtime)
Infrastructure provisioning (3 EC2 instances)	≈5 minutes
EKS cluster creation (Terraform)	10–15 minutes
Full environment re-creation from scratch	<20 minutes total

C. Infrastructure Reproducibility

Terraform-based provisioning proved to be one of the most impactful aspects of the implementation. The complete AWS environment—including Jenkins, SonarQube, Nexus, and the EKS cluster—could be recreated from version-controlled scripts, eliminating all manual configuration inconsistencies. Post-project, all resources were destroyed using terraform destroy, demonstrating complete infrastructure lifecycle management.

D. Deployment Reliability

The Kubernetes rolling update strategy deployed on AWS EKS achieved zero-downtime application updates across all test deployment cycles. The AWS Application Load Balancer consistently routed traffic only to healthy pods, while Kubernetes automatically replaced failed containers. The parameterized Jenkins pipeline demonstrated fully manageable deployments without manual AWS Console intervention.

E. Comparison with Traditional CI/CD Pipeline

Table VI: Traditional CI/CD Pipeline vs. DevSecOps CI/CD Pipeline

Aspect	Traditional CI/CD Pipeline	DevSecOps Pipeline (This Work)
Primary Focus	Continuous integration and rapid software delivery	Secure and automated software delivery
Security Integration	Security performed after development and testing	Security integrated throughout the software lifecycle
Code Analysis	Optional or manual code review	Automated static code analysis using SonarQube
Vulnerability Scanning	Limited or periodic security assessment	Continuous vulnerability scanning using OWASP Dependency-Check
Quality Gate Enforcement	Basic functional validation	Mandatory security and quality gate validation
Infrastructure Management	Manual or partially automated provisioning	Infrastructure as Code (IaC) using Terraform
Risk Detection	Vulnerabilities often detected late	Early vulnerability detection through Shift-Left Security
Compliance Enforcement	Manual compliance checks	Automated security and compliance validation
Team Collaboration	Collaboration between Development and Operations	Collaboration between Development, Security, and Operations

VII. CONCLUSION

This paper presented the design, implementation, and evaluation of a DevSecOps-driven secure CI/CD pipeline for the Ekart enterprise e-commerce application deployed on AWS cloud infrastructure. By integrating Jenkins, SonarQube, OWASP Dependency-Check, Nexus Repository, Docker, Terraform, and AWS EKS, the proposed framework embeds security throughout the software development lifecycle using a Shift-Left Security approach. Automated security scanning, quality gate enforcement, vulnerability assessment, and Infrastructure as Code collectively ensure that only secure and validated software artifacts are deployed to production.

The experimental results demonstrate that the proposed DevSecOps pipeline improves software quality, enhances vulnerability detection, reduces manual security review efforts, and enables secure, reliable, and reproducible deployments with minimal downtime. By integrating development, security, and operations into a unified automated workflow, the proposed framework provides a scalable, practical, and industry-ready solution for secure cloud-native application delivery. The approach can serve as a reference architecture for organizations seeking to implement DevSecOps practices and strengthen security across modern software delivery pipelines.

VIII. FUTURE SCOPE

Future work can further enhance the proposed DevSecOps framework by integrating GitOps tools such as ArgoCD or FluxCD for declarative and automated Kubernetes deployments with continuous drift detection. Advanced observability solutions, including Prometheus, Grafana, and the ELK Stack, can be incorporated to improve real-time monitoring, centralized logging, and security event analysis. Container and Kubernetes security can be strengthened by integrating image scanning tools such as Trivy or Clair, along with runtime security monitoring for detecting threats in production environments. In addition, the pipeline can be extended with Dynamic Application Security Testing (DAST), Software Composition Analysis (SCA), and Policy-as-Code frameworks such as Open Policy Agent (OPA) to automate security and compliance validation. These enhancements will further improve the scalability, resilience, and security of cloud-native DevSecOps pipelines while supporting the secure delivery of enterprise applications.

ACKNOWLEDGMENT

The authors express their sincere gratitude to the Department of Information Technology, P.E.S. Modern College of Engineering, Savitribai Phule Pune University, Pune, for providing the academic environment, technical resources, and institutional support necessary for the successful completion of this research. The authors are especially thankful to Prof. Shantanu Pawar for his valuable guidance, continuous encouragement, insightful suggestions, and constructive feedback throughout the course of this project. The support and motivation provided by the faculty and the institution were instrumental in the successful execution of this work.

REFERENCES

[1] N. Singh, D. Patel, A. Raj, S. Shubham, and S. Kour, "CI/CD Pipeline for Web Applications," *International Journal for Research in Applied Science & Engineering Technology (IJRASET)*, vol. 11, no. 5, pp. 5218–5224, May 2023.

[2] S. M. Saleh et al., "A Systematic Literature Review on Continuous Integration and Deployment (CI/CD) for Secure Cloud Computing," in Proceedings of the 20th International Conference on Web Information Systems and Technologies (WEBIST), 2024–2025.

[3] P. Bavadiya, "Security-as-Code: Integrating Automated Security Policies into DevOps Pipelines," Journal of Interdisciplinary Research (JIER), 2023.

[4] C. Pahl et al., "A Systematic Review of Infrastructure-as-Code Technologies," Springer, 2025.

[5] N. Chavan, N. Bharambe, S. Deshmukh, D. Ahire, and A. R. Jain, "Implementing DevSecOps Pipeline for an Enterprise Organization," International Journal of Advanced Research in Science, Communication and Technology (IJARSCT), vol. 2, no. 4, pp. 46–72, May 2022.

[6] T. Abdiukov, "Automated Security Testing in DevSecOps Pipelines: Integrating AI-Based Vulnerability Discovery and Compliance Validation," World Journal of Advanced Research and Reviews (WJARR), vol. 22, no. 1, pp. 2083–2093, 2024.
[7] B. Pando and A. Davila, "A Systematic Mapping Study on Software Testing in the DevOps Context," ISP RAS Proceedings, vol. 35, no. 1, pp. 21–34, 2023.
[8] D. Esther, "Automated Testing Strategies for Quality Assurance in CI/CD Pipelines," International Journal of Emerging Technologies, 2024.
[9] Amazon Web Services, "Amazon Elastic Kubernetes Service (EKS)," AWS Documentation, 2023. [Online]. Available: https://aws.amazon.com/eks/
[10] HashiCorp, "Terraform: Infrastructure as Code," Terraform Documentation, 2023. [Online]. Available: https://developer.hashicorp.com/terraform
[11] Jenkins Project, "Jenkins User Documentation," Official Documentation, 2023. [Online]. Available: https://www.jenkins.io/doc/
[12] Docker Inc., "Docker Documentation," docs.docker.com, 2023. [Online]. Available: https://docs.docker.com/
[13] OWASP Foundation, "OWASP Dependency-Check," 2023. [Online]. Available: https://owasp.org/www-project-dependency-check/
[14] Sonatype, "Nexus Repository Manager Documentation," 2023. [Online]. Available: https://help.sonatype.com/repomanager3