IJETIR
ISSN (Online) 2583-0554
International Journal of Emerging Technologies and Innovative Research (IJETIR)
Volume 6, Issue 1, January 2026
Impact Factor: 5.731

Design and Development of a Secure Devsecops Toolchain Framework for CI/CD Pipelines

[1]Ghorpade Bipin Shivaji, [2]Dr. Ayaz Ahmed Faridi, [3]Dr. Kalyan Devappa Bamane
Research Scholar, Assistant Professor, Professor
[1]Computer science engineering, [2]Department of CS & Faculty of IT, [3]Department of Computer Engineering
[1,2]Kalinga University, Nava Raipur, Chhattisgarh
[3]Indira College of Engineering and Management Parandwadi M.H
ayazahmed.faridi@kalingauniversity.ac.in[1], kalyan.bamane@indiraicem.ac.in[2]

Abstract: *Continuous Integration and Continuous Delivery (CI/CD) pipelines allow software to be released quickly, but security is often treated as a separate activity performed late in the lifecycle. DevSecOps addresses this gap by making security a shared responsibility that is automated at every stage of software delivery. This paper presents the design and development of a secure DevSecOps toolchain framework for CI/CD pipelines. The framework maps security controls to six pipeline stages—plan and code, build, test and scan, artifact repository, deploy, and operate and monitor—and combines automation tools (Jenkins, Ansible, Puppet), security testing tools (SonarQube, Snyk, Checkmarx, Burp Suite, Nessus, Qualys), and monitoring solutions (ELK stack, Splunk, New Relic). A risk-based security gate is proposed to stop builds that contain critical findings, and a reference Jenkins pipeline is provided as an implementation. The framework is evaluated analytically against the OWASP Top 10 CI/CD Security Risks, where it fully addresses six risks and partially addresses the remaining four, giving a design coverage score of 80%. The paper also compares the framework with a traditional DevOps pipeline and discusses its advantages and adoption challenges. The results indicate that integrating security tools along the pipeline improves visibility, enables earlier detection of vulnerabilities, and treats security defects as quality defects without slowing delivery.*

Keywords: DevSecOps, CI/CD Pipeline, Security Automation, Toolchain, Shift-Left Security, Security Gate.

I. INTRODUCTION

DevOps practices combine software development and IT operations to shorten release cycles through automation, collaboration, and continuous feedback [7], [8]. A CI/CD pipeline is the technical backbone of DevOps: every code change is automatically built, tested, packaged, and deployed [3]. While this greatly increases delivery speed, it also means that insecure code, vulnerable libraries, or misconfigured infrastructure can reach production within minutes if security is not part of the pipeline.

In many organizations, security testing is still performed by a separate team near the end of the release cycle. Defects found at this stage are expensive to fix and often delay releases, and developers may deprioritize security issues because they are not visible in their normal workflow [5]. DevSecOps extends DevOps by integrating security into every phase of the software lifecycle—plan, develop, build, test, release, deliver, deploy, operate, and monitor—so that security becomes a shared and automated responsibility [1], [2].

The pipeline itself has also become an attack target. Attacks on software supply chains, such as compromised dependencies and tampered build systems, are increasingly common [11], and secrets such as API keys are frequently leaked through source repositories [12]. The OWASP Top 10 CI/CD Security Risks [9] lists weaknesses such as poisoned pipeline execution, insufficient credential hygiene, and improper artifact integrity validation.

IJETIR
ISSN (Online) 2583-0554
International Journal of Emerging Technologies and Innovative Research (IJETIR)
Volume 6, Issue 1, January 2026
Impact Factor: 5.731

Although a large number of DevSecOps tools are available, practitioners often struggle to decide which tool belongs at which stage and how the tools should work together [1], [4]. The aim of this work is the design and development of a secure framework of a DevSecOps toolchain for CI/CD pipelines. The objectives are to:

- automate, monitor, and apply security controls at all phases of the software lifecycle;
- treat security vulnerabilities as quality defects, so that they are visible to developers and cannot be silently deprioritized;
- reduce the cost of compliance and enable faster, more reliable software delivery.

The main contributions of the paper are: (i) a stage-wise toolchain framework that maps security, automation, and monitoring tools to each pipeline stage; (ii) a risk-based security gate that decides whether a build may be promoted; (iii) a reference implementation as a Jenkins declarative pipeline; and (iv) an analytical evaluation of the framework against the OWASP Top 10 CI/CD Security Risks.

II. LITERATURE SURVEY

Myrbakken and Colomo-Palacios [2] performed a multivocal literature review of DevSecOps and identified its core idea as integrating security practices into DevOps without losing agility. Mohan and Ben Othmane [4] mapped research on security in DevOps and found that the field lacked clear definitions and practical guidance. Rahman and Williams [5] synthesized practitioners' views and reported that automated security testing, code review, and monitoring are the most widely used practices, while organizational culture is a major barrier.

Rajapakse et al. [1] conducted a systematic review of the challenges of adopting DevSecOps and grouped them into people, practices, tools, and infrastructure. Tool-related challenges included a high number of false positives, difficulty in integrating tools into fast pipelines, and the lack of guidance on tool selection. Shahin et al. [3] reviewed CI/CD approaches and highlighted that security is one of the least addressed concerns in continuous deployment.

Individual security tools have also been studied. Static analysis tools are known to generate many false positives, which discourages developers from using them [6], and they may miss real vulnerabilities in code reviews [13]. Software composition analysis tools vary considerably in accuracy [14], and black-box web scanners miss many classes of vulnerabilities [17]. Rahman et al. [16] identified recurring security smells in infrastructure-as-code scripts such as hard-coded secrets and weak defaults. For supply-chain protection, in-toto [18] provides cryptographic guarantees that each pipeline step was performed as intended, and NIST's Secure Software Development Framework [10] recommends practices for protecting software and its build environment.

These studies show that each tool addresses only one part of the problem. What is missing is a simple, stage-wise framework that combines these tools into one toolchain, defines when a build should be stopped, and is checked against a recognized list of pipeline risks. This paper addresses that gap.

III. PROPOSED METHODOLOGY

A. Design Principles

The framework is based on four process principles: (1) create and audit secure systems—security checks are automated and every result is recorded; (2) mind data-access controls—access to code, pipelines, secrets, and environments follows least privilege; (3) examine processes—each stage has a clearly defined security gate and feedback loop; and (4) get people working together—security findings are reported in the same tools that developers and operators already use, so that development, security, and operations teams share responsibility.

B. Framework Architecture

Fig. 1 shows the proposed architecture. The left side represents the DevOps toolchain that moves code forward from source to production. The right side represents the DevSecOps security controls attached to each stage, which send findings back to developers as early as possible.

IJETIR
ISSN (Online) 2583-0554
International Journal of Emerging Technologies and Innovative Research (IJETIR)
Volume 6, Issue 1, January 2026
Impact Factor: 5.731

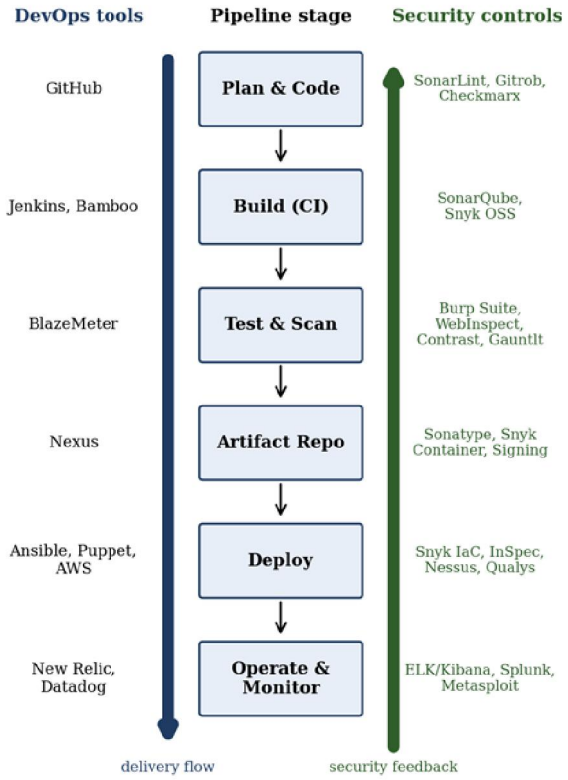


Fig. 1. Proposed DevSecOps toolchain framework with DevOps tools and security controls mapped to each CI/CD stage.

C. Stage-Wise Security Controls

Table I summarizes the tools assigned to each stage. Tools were selected from widely used open-source and commercial products so that each stage has at least one automated security control.

1) Plan and Code: Developers receive immediate feedback in the IDE through SonarLint. Repositories are scanned for leaked credentials using Gitrob or an equivalent secret scanner, and Checkmarx performs static application security testing (SAST) on pull requests. Branch protection and mandatory peer review ensure that no code reaches the main branch unreviewed.

2) Build (CI): Jenkins (or Bamboo) builds the code and runs SonarQube analysis with a quality gate. Snyk Open Source performs software composition analysis (SCA) to find known vulnerabilities in third-party dependencies.

3) Test and Scan: After functional and performance tests (e.g., BlazeMeter), dynamic application security testing (DAST) is carried out with Burp Suite or WebInspect against a test deployment. Contrast Security provides interactive testing (IAST), and Gauntlt expresses security tests as readable, repeatable scenarios.

4) Artifact Repository: Built packages and container images are stored in Nexus. Sonatype and Snyk Container scan them for vulnerable components, and artifacts are signed so that their integrity can be verified before deployment.

5) Deploy: Infrastructure is provisioned as code with Ansible or Puppet on a cloud platform such as AWS. IaC templates are scanned (Snyk IaC), InSpec checks servers against a compliance baseline, and Nessus or Qualys scans deployed hosts for vulnerabilities.

6) Operate and Monitor: Logs from the pipeline and the application are collected by the ELK stack (Elasticsearch, Logstash, Kibana, Beats) and Splunk. Performance and anomalies are monitored with New Relic, Datadog, Zabbix, or Nagios, and periodic penetration tests are performed with Metasploit.

IJETIR
ISSN (Online) 2583-0554
International Journal of Emerging Technologies and Innovative Research (IJETIR)
Volume 6, Issue 1, January 2026
Impact Factor: 5.731

TABLE I
TOOL MAPPING PER PIPELINE STAGE

Stage	Automation	Security control	Type
Plan & Code	GitHub	SonarLint, Gitrob, Checkmarx	SAST, secrets
Build	Jenkins, Bamboo	SonarQube, Snyk OSS	SAST, SCA
Test & Scan	BlazeMeter	Burp Suite, WebInspect, Contrast	DAST, IAST
Artifact	Nexus	Sonatype, Snyk Container	SCA, signing
Deploy	Ansible, Puppet, AWS	Snyk IaC, InSpec, Nessus, Qualys	IaC, compliance
Monitor	New Relic, Datadog	ELK/Kibana, Splunk, Metasploit	Logging, pentest

D. Risk-Based Security Gate

Each scanner reports findings with a severity level. To make decisions consistent across tools, all findings are converted to a common format with four severity levels $s \in \{\text{Critical, High, Medium, Low}\}$. The risk score of a build is

$$R = \Sigma s\ ws \bullet ns \quad (1)$$

where ns is the number of unresolved findings of severity s and ws is its weight. In this work the weights are set to w = (10, 5, 2, 1) for Critical, High, Medium, and Low. A build is promoted to the next stage only if

$$nCritical = 0 \text{ and } R \leqslant \theta \quad (2)$$

where θ is a threshold chosen by the organization (for example, θ = 20). Fig. 2 shows the gate logic. If the gate fails, the build is stopped and the developer is notified with the findings. A finding can only be bypassed through a documented and approved exception, which is logged for audit.

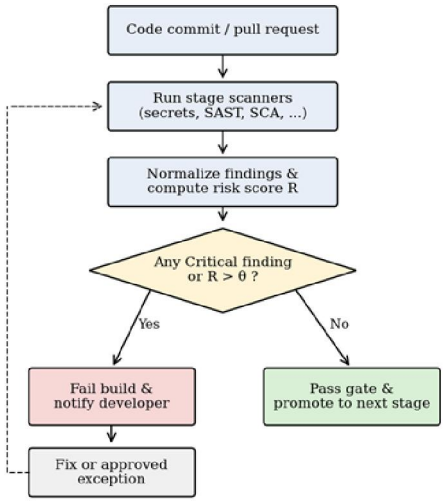


Fig. 2. Security gate decision flow applied at each stage.

E. Access Control and Secret Management

All pipeline users authenticate with multi-factor authentication and are assigned roles following least privilege. Credentials are never stored in code; they are kept in the Jenkins credentials store or a secrets vault and injected only into the stage that needs them. Build agents are isolated, and pipeline definitions are protected so that untrusted pull requests cannot modify them or access secrets.

IJETIR
ISSN (Online) 2583-0554
International Journal of Emerging Technologies and Innovative Research (IJETIR)
Volume 6, Issue 1, January 2026
Impact Factor: 5.731

IV. IMPLEMENTATION AND EVALUATION

A. Reference Implementation

The framework was implemented as a Jenkins declarative pipeline, shown in simplified form in Listing 1. Each security stage returns a non-zero exit code when its policy is violated, which stops the pipeline automatically.

Listing 1. Simplified Jenkinsfile of the proposed pipeline

```
pipeline {
 agent any
 stages {
  stage('Secret Scan') { steps { sh './scan-secrets.sh' } }
  stage('SAST') { steps {
   withSonarQubeEnv('sonar') { sh 'sonar-scanner' } } }
  stage('Quality Gate') { steps {
   waitForQualityGate abortPipeline: true } }
  stage('SCA') { steps {
   sh 'snyk test --severity-threshold=high' } }
  stage('Build Image') { steps {
   sh 'docker build -t app:${BUILD_NUMBER} .' } }
  stage('Image Scan') { steps {
   sh 'snyk container test app:${BUILD_NUMBER}' } }
  stage('IaC Scan') { steps { sh 'snyk iac test infra/' } }
  stage('Publish') { steps { sh './sign-and-push-nexus.sh' } }
  stage('Deploy') { steps { sh 'ansible-playbook deploy.yml' } }
  stage('DAST') { steps { sh './run-dast.sh staging' } }
  stage('Compliance') { steps {
   sh 'inspec exec profiles/baseline' } }
 }
 post { always { sh './ship-logs-to-elk.sh' } }
}
```

B. Coverage of OWASP Top 10 CI/CD Security Risks

To evaluate the design, each of the OWASP Top 10 CI/CD Security Risks [9] was checked against the controls of the framework. A risk is marked Full when the framework includes a direct preventive or detective control for it, and Partial when the control depends on additional organizational policy. The result is shown in Table II.

TABLE II
COVERAGE OF OWASP TOP 10 CI/CD RISKS

Risk	Framework control	Coverage
SEC-1 Insufficient flow control	Branch protection, reviews, security gates	Full
SEC-2 Inadequate IAM	MFA, role-based least privilege	Full
SEC-3 Dependency chain abuse	Snyk OSS, Sonatype, Nexus proxy	Partial
SEC-4 Poisoned pipeline execution	Protected pipeline definitions	Partial
SEC-5 Insufficient PBAC	Isolated agents, scoped secrets	Partial
SEC-6 Credential hygiene	Secret scanning, credentials vault	Full
SEC-7 Insecure configuration	Ansible/Puppet, InSpec, Nessus	Full
SEC-8 Ungoverned 3rd-party services	Plugin and integration review	Partial
SEC-9 Artifact integrity	Artifact signing and verification	Full

IJETIR
ISSN (Online) 2583-0554
International Journal of Emerging Technologies and Innovative Research (IJETIR)
Volume 6, Issue 1, January 2026
Impact Factor: 5.731

SEC-10 Logging and visibility	ELK, Splunk, New Relic	Full

A design coverage score is computed as

$$C = (F + 0.5 \cdot P) / 10 \quad (3)$$

where F and P are the numbers of fully and partially covered risks. With F = 6 and P = 4, the framework achieves C = 0.80, or 80%. Dependency chain abuse and poisoned pipeline execution are only partially covered because known-vulnerability scanning cannot detect new malicious packages, and protecting pipeline definitions also depends on repository permissions configured by the organization.

C. Comparison with a Traditional DevOps Pipeline

Table III compares the proposed framework with a traditional DevOps pipeline in which security testing is performed separately before release.

TABLE III
TRADITIONAL DEVOPS VS. PROPOSED FRAMEWORK

Aspect	Traditional DevOps	Proposed DevSecOps
Security testing	Late, manual, before release	Automated at every stage
Feedback to developers	Days or weeks later	Within the same build
Dependencies	Rarely checked	SCA on every build
Secrets	Often in code or config	Scanned and vaulted
Release decision	Manual sign-off	Risk-based security gate
Compliance	Periodic audits	Continuous (InSpec, logs)
Responsibility	Security team only	Shared by all teams

D. Advantages

The framework offers several benefits. Security, monitoring, deployment checks, and notifications are present from the beginning of development, which reduces the cost of fixing defects and supports a faster delivery rate. Because findings are visible in the pipeline, the process is open and transparent to all teams. Security is built in by design and becomes measurable through the risk score and coverage metrics. Centralized logging enables faster recovery in the case of a security incident, and infrastructure-as-code supports immutable, consistently hardened environments.

E. Challenges and Limitations

Adopting the framework also brings challenges. Developers and security staff need retraining, and existing workflows must change. Attempting to reach zero vulnerabilities can slow developers down, so thresholds such as θ must be tuned to the organization's risk appetite. Poor communication between developers and the security team can reduce the benefit of automated feedback. Scanners also produce false positives [6], which may cause alert fatigue if they are not triaged.

This study has limitations. The evaluation is analytical: it checks the design against a risk catalogue but does not measure detection rates, pipeline execution time, or developer effort on a real project. The weights and threshold of the security gate are illustrative. Future work should deploy the framework on real applications and report quantitative results such as the number of vulnerabilities detected per stage, build-time overhead, and mean time to remediation.

V. CONCLUSION

This paper presented the design and development of a secure DevSecOps toolchain framework for CI/CD pipelines. The framework integrates continuous integration tools, continuous delivery tools, infrastructure-as-code tools, security scanning tools, and monitoring and logging tools into one stage-wise pipeline, and adds a risk-based security gate that stops insecure builds automatically. A reference Jenkins implementation shows how the framework can be put into practice. An analytical evaluation against the OWASP Top 10 CI/CD Security Risks showed full coverage of six risks and partial coverage of four, giving a design coverage score of 80%. By treating security vulnerabilities as quality defects and making them visible early, the framework helps organizations improve security, speed up development, and

encourage collaboration between teams. Future work will include an empirical evaluation on real projects and the addition of supply-chain attestation such as in-toto [18] to strengthen artifact integrity.

REFERENCES

1. R. N. Rajapakse, M. Zahedi, M. A. Babar, and H. Shen, "Challenges and solutions when adopting DevSecOps: A systematic review," Inf. Softw. Technol., vol. 141, Art. no. 106700, Jan. 2022.
2. H. Myrbakken and R. Colomo-Palacios, "DevSecOps: A multivocal literature review," in Proc. Int. Conf. Softw. Process Improv. Capability Determination (SPICE), CCIS, vol. 770, Springer, 2017, pp. 17–29.
3. M. Shahin, M. Ali Babar, and L. Zhu, "Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices," IEEE Access, vol. 5, pp. 3909–3943, 2017.
4. V. Mohan and L. Ben Othmane, "SecDevOps: Is it a marketing buzzword? Mapping research on security in DevOps," in Proc. 11th Int. Conf. Availability, Reliability and Security (ARES), 2016, pp. 542–547.
5. A. U. Rahman and L. Williams, "Software security in DevOps: Synthesizing practitioners' perceptions and practices," in Proc. IEEE/ACM Int. Workshop Continuous Softw. Evol. Del. (CSED), 2016, pp. 70–76.
6. Johnson, Y. Song, E. Murphy-Hill, and R. Bowdidge, "Why don't software developers use static analysis tools to find bugs?" in Proc. 35th Int. Conf. Softw. Eng. (ICSE), 2013, pp. 672–681.
7. J. Humble and D. Farley, Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Boston, MA, USA: Addison-Wesley, 2010.
8. G. Kim, J. Humble, P. Debois, and J. Willis, The DevOps Handbook. Portland, OR, USA: IT Revolution Press, 2016.
9. OWASP Foundation, "OWASP Top 10 CI/CD Security Risks," 2022. [Online]. Available: https://owasp.org/www-project-top-10-ci-cd-security-risks/
10. M. Souppaya, K. Scarfone, and D. Dodson, "Secure Software Development Framework (SSDF) version 1.1," NIST, Gaithersburg, MD, USA, SP 800-218, Feb. 2022.
11. P. Ladisa, H. Plate, M. Martinez, and O. Barais, "SoK: Taxonomy of attacks on open-source software supply chains," in Proc. IEEE Symp. Security Privacy (SP), 2023, pp. 1509–1526.
12. M. Meli, M. R. McNiece, and B. Reaves, "How bad can it git? Characterizing secret leakage in public GitHub repositories," in Proc. Netw. Distrib. Syst. Secur. Symp. (NDSS), 2019.
13. W. Charoenwet, P. Thongtanunam, V.-T. Pham, and C. Treude, "An empirical study of static analysis tools for secure code review," in Proc. 33rd ACM SIGSOFT Int. Symp. Softw. Testing Anal. (ISSTA), 2024, pp. 691–703.
14. L. Zhao, Y. Zhang, Y. Liu, Z. Chen, and X. Xia, "Software composition analysis for vulnerability detection: An empirical study on Java projects," in Proc. 31st ACM ESEC/FSE, 2023, pp. 960–972.
15. L. Bass, I. Weber, and L. Zhu, DevOps: A Software Architect's Perspective. Boston, MA, USA: Addison-Wesley, 2015.
16. Rahman, C. Parnin, and L. Williams, "The seven sins: Security smells in infrastructure as code scripts," in Proc. 41st Int. Conf. Softw. Eng. (ICSE), 2019, pp. 164–175.
17. Doupé, M. Cova, and G. Vigna, "Why Johnny can't pentest: An analysis of black-box web vulnerability scanners," in Proc. Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA), 2010, pp. 111–131.
18. S. Torres-Arias, H. Afzali, T. K. Kuppusamy, R. Curtmola, and J. Cappos, "in-toto: Providing farm-to-table guarantees for bits and bytes," in Proc. 28th USENIX Security Symp., 2019, pp. 1393–1410.