International Journal of Computer (IJC)
ISSN 2307-4523 (Print & Online)
https://ijcjournal.org/index.php/InternationalJournalOfComputer/index

Methodological Aspects of Integrating Security Processes into Continuous Development Pipelines (CI/CD Security)

Praveen Ravula*

Software Engineer at Amazon,Arlington, VA - USA
Email:ravulapraveen31@gmail.com

Abstract

The article presents a theoretical and methodological analysis of integrating security processes into continuous integration and continuous delivery pipelines. The study consolidates DevSecOps models, software supply chain protection approaches, and multi-layer container security into a unified CI/CD Security framework. The pipeline is structured by stages of source code management, build, artifact storage, test automation, and deployment, with corresponding typical vulnerabilities and integrated security processes. This allows the pipeline to be interpreted not as a set of isolated practices but as a coherent trust chain. A multi-layer protection model is proposed, combining image control, orchestrator security, host operating system hardening, runtime protection, and artifact provenance. Within this model, the container platform and traceability mechanisms are viewed as complementary components of a single methodological foundation. The key methodological challenges of integrating security into CI/CD are identified, and it is shown that ignoring them leads to underestimated risk assessments and overstated expectations regarding adopted practices. Promising directions for advancing CI/CD Security are substantiated, including the use of intelligent telemetry analysis, self-healing mechanisms, cryptographically reinforced trust boundaries, and formalized maturity metrics. The article may be useful for architects, engineers, and researchers involved in designing secure continuous delivery pipelines in highly automated and regulated environments.

Keywords: software supply chain security; container security; provenance metadata; security automation; process maturity.

Received: *1/4/2026*
Accepted: *3/4/2026*
Published: *3/14/2026*

---

* *Corresponding author.*

1.Introduction

The transition to continuous integration and delivery has made CI/CD pipelines a key element of development. Automated builds, frequent releases, containers, and cloud platforms accelerate the delivery of functionality, but simultaneously increase the vulnerability of supply chains, build environments, and orchestration systems [3]. Attacks are shifting from finished applications to the pipelines themselves, artifacts, and utilized services.

Customer and regulatory demand is focused on process transparency, control over artifacts, and compliance. Built-in security checks with every code change, without significant loss of speed, are becoming a condition for trust, certification, and long-term cooperation.

For organizations, the risks of repository and build system compromise, exploitation of vulnerable dependencies and container images, and errors in infrastructure-as-code are particularly significant. Informal processes, manual checks, and fragmented use of security tools increase the likelihood of introducing malicious changes [5]. The consistent application of automated security tests, dependency management, artifact signing, and infrastructure change control improves pipeline resilience.

The objective of this study is to identify and theoretically substantiate mechanisms for integrating security processes into continuous software development and delivery pipelines, structure the applied approaches, and analyze the role of automation, containerization, and supply chain management as key determinants of CI/CD resilience. To achieve this goal, pipeline stages and associated vulnerabilities were systematized, DevSecOps and container security practices were classified, architectural patterns and organizational mechanisms were compared, connections between automated checks, artifact provenance management, and the pipeline operating model were defined, and criteria for the resilience and manageability of CI/CD processes were formulated.

The scientific novelty of the research lies in organizing approaches to integrating security into continuous integration and delivery processes by aligning secure development and operations practices, software supply chain protection, and container environment security. The role of the combined application of artifact provenance metadata, formalized security policies, and multi-layer container protection tools in forming a holistic model for ensuring continuous development pipeline security is demonstrated.

The hypothesis of the study is that a coherent combination of automated security testing, dependency and artifact management, container protection, and controlled access is more effective than isolated measures. Such a combination increases pipeline resilience and observability, reduces the risk of supply chain attacks, and stabilizes the development lifecycle.

The scope of the study covers organizations using CI/CD pipelines, containerization, and cloud infrastructure, where build predictability, artifact manageability, and security compliance are critical. Primary attention is paid to highly automated and regulated environments where DevOps/DevSecOps practices and specialized container protection mechanisms are applied within CI/CD.

2.Materials and Methods

The methodological basis of the study is formed by synthesizing secure development and DevSecOps models presented in the systematic review by Mohammed and his colleagues [4]. The selection includes works from 2021–2025 dedicated to protecting continuous integration and delivery pipelines, risks of modern software delivery, and secure automation practices. The corpus of sources is supplemented by studies combining risk management, zero-trust architectures, and security automation tools into a single development framework.The study by Koneru [3] demonstrates the integration of static and dynamic analysis and dependency analysis into CI/CD pipelines, which sets the basis for automated security testing at all build stages. The work of Coston and his colleagues [1] connects DevSecOps approaches, risk management, and artificial intelligence tools to assess end-to-end lifecycle security. The P2ISE model proposed by Muñoz and his colleagues [5] is used to account for the role of trusted executors and hardware roots of trust in pipelines. Patel [8] examines the application of provenance metadata, tiered supply chain protection models, and artifact policies as a basis for traceability methods. The general framework for security control in hybrid pipelines and differences between cloud and local environments are described in the work of Obuse and his colleagues [6]. Threat analysis in open software pipelines, performed by Pan and his colleagues [7], is used to classify attack vectors and weak links in the supply chain. Solanke [9] reveals the specifics of DevSecOps in regulated industries with an emphasis on lifecycle security, compliance automation, and zero-trust models. In the study by Karanam [2], applied descriptions of risks and mitigation measures in CI/CD pipelines are provided, clarifying the practical side of implementation. The work of Ugale and Potgantwar [10] describes container security as a distinct protection layer for pipelines, while Patel [8] allows linking this layer with software component management.

Thus, the research methodology is based on comparing conceptual DevSecOps models, threat descriptions, and supply chain protection practices. Structural analysis, comparative review, and typology of security measures are applied. This approach provides a holistic basis for further describing the methodological aspects of integrating security processes into continuous development pipelines.

3.Results

The conducted analysis showed that to correctly describe security integration, it is necessary to proceed from a staged pipeline model, including source code management, automated build, artifact storage, test automation, and automated deployment, with each stage corresponding to its own vulnerability profile.

In the study by Patel [8], the initial stage is described as code management systems, where key vulnerabilities are related to credential theft, malicious commits, and storing secrets in repositories, making SCM a primary entry point for attackers. For the automated build stage, the same work indicates risks of script hijacking, exploitation of vulnerable agents, and leakage of environment variables containing sensitive data [8]. For artifact stores and image registries, Pan and his colleagues [7] identify the problem of unsigned or unscanned artifacts and repository poisoning, which directly affect supply chain integrity. The test automation stage in Koneru [3] is characterized by incomplete coverage, disabling of critical checks, and insufficient use of dependency analysis, allowing vulnerable code to pass through. At the automated deployment stage, the same author emphasizes risks of

erroneous manifests, excessive service account privileges, and weak network segmentation in orchestration systems [3].

Examining these stages in conjunction with protection measures allows the shift of security to early stages—integrating static and dynamic analysis, dependency analysis, and code change request checks into the pipeline—to be interpreted as a methodological core, representing a systemic transfer of security functions to the beginning of the software lifecycle [3]. The work of Patel [8] shows that formalized security policies and rules, described in code and implemented via tools like OPA Gatekeeper, allow blocking unsigned images and insecure configurations as early as the access control stage for repositories and clusters, turning policies into a full-fledged pipeline element.

Comparing these results shows that superimposing approaches from different authors forms a coherent scheme in which CI/CD stages and security processes create a connected structure rather than a set of isolated activities. Table 1 examines CI/CD pipeline stages, their typical vulnerabilities, and corresponding security processes.

Table 1 : CI/CD stages, typical vulnerabilities and integrated security processes (Compiled by the author based on sources: [3, 8, 9])

CI/CD stage	**Typical vulnerabilities**	**Integrated security processes**
Source Code Management	Credential theft, malicious commits, secrets stored in repositories	MFA, signed commits, secret management, audit logging
Automated Builds	Hijacked build scripts, vulnerable build agents, env variable leakage	Hardened build agents, SAST/DAST/SCA in pipelines, secure variables
Artifact Storage/Registry	Unsigned or unscanned artifacts, repository poisoning	Artifact signing, SBOM generation, dependency scanning, access control
Test Automation	Incomplete coverage, disabled security checks, missed vulnerable deps	Mandatory security test stages, SCA, enforced quality and security gates
Automated Deployment	Misconfigured manifests, over-privileged service accounts, weak segmentation	IaC scanning, policy-as-code (OPA), least-privilege access, centralized logging

Interpreting the data in the table shows that while maintaining the classical linear structure of CI/CD, each stage simultaneously acts as a source of risks and a platform for embedding automated security processes, and the effectiveness of the methodological model is determined not by the presence of individual practices, but by the connectivity of their application along the entire pipeline.

In the study by Ugale and Potgantwar [10], container security is described as a multi-level system including

analysis of images, the orchestration layer, the host operating system, and the runtime environment. At the image level, vulnerabilities of packages and libraries inherited from base images and public repositories are recorded, which the authors demonstrate using five popular images (Mariadb, Nginx, httpd, MySQL, Debian) scanned by two independent vulnerability analysis tools.The orchestration layer is presented as an independent risk contour. For Kubernetes-based clusters, the importance of authentication and authorization, network segmentation, namespace separation, logging, and timely component updates is emphasized [6]. It is highlighted that configuration violations at this level make attack propagation between services possible even with relatively secure images. At the host operating system level, the authors emphasize that successful node capture leads to the compromise of the entire container infrastructure, justifying the need for regular vulnerability scanning and configuration hardening according to industry recommendations [3]. A dedicated execution control service is proposed as an additional layer, analyzing network traffic and container behavior and blocking identified attacks before they develop.

Within the framework of this research, such a construction is interpreted not as a set of independent practices, but as a supporting framework for describing a secure CI/CD pipeline, where the container platform defines the boundaries of the trusted environment for all development and delivery stages. This allows matching protection levels with the concept of end-to-end integration of security processes into pipeline stages. Table 2 presents levels of container and supply chain security and corresponding measures used in building a secure pipeline.

Table 2 : Levels of container and supply chain security and corresponding measures in CI/CD (Compiled by the author based on sources: [8, 10])

Level	**Main risks**	**Security measures and processes**
Container image	Vulnerable packages and libraries; untrusted or malicious images	Minimal trusted base images; regular image vulnerability scanning (Trivy, Grype)
Orchestrator / cluster	Misconfigured Kubernetes; weak auth; flat network; missing logging	Strong authentication and authorization; network segmentation; namespaces; logging; updates
Host OS	Host-level exploits; OS misconfiguration	Host OS vulnerability scanning; hardening to security benchmarks
Runtime control	Anomalous container behaviour; resource abuse; network attacks	Runtime Security Container; behaviour and traffic monitoring; blocking malicious traffic
Supply chain and artifacts	Tampered or unsigned artifacts; missing provenance and dependency insight	SBOM (SPDX, CycloneDX); artifact signing with Cosign; Rekor logs; in-toto attestations; SLSA levels

Consequently, the comparison of levels shows that container security and software supply chain protection describe two complementary dimensions of a single methodological task: infrastructure state control and provable artifact provenance. The work of Patel [8] substantiates that recording provenance metadata for artifacts, including the author of changes, tools used, build parameters, check results, and signature facts, forms the basis for traceability and detection of substitutions in the supply chain. The study also shows that using software bills of materials, cryptographic signing, public logs, and build step attestation allows combining automated artifact verification with regulatory requirements for process control. This approach sets a norm for the CI/CD pipeline where no artifact can be deployed without a verifiable provenance trail, and each build stage is viewed as part of a single line of trust, which resonates with the concept of continuous lifecycle security.

Thus, combining the multi-level container model of Ugale and Potgantwar with the metadata-based approach sets the methodological foundation for integrating security processes into CI/CD pipelines. Infrastructure and artifacts are connected by a single logic of provable trust, and the assessment of pipeline security shifts from checking individual points to analyzing a continuous chain of states.

4.Discussion

The formation of a methodological model for integrating security into CI/CD pipelines inevitably encounters a conflict between the requirement for high release speed and the depth of checks, especially in regulated industries. This conflict requires additional clarification from a methodological perspective. While earlier DevSecOps studies primarily interpret security integration as a progressive extension of automation capabilities, the results of the present study indicate that automation alone does not resolve structural security risks. Instead, security effectiveness emerges from the continuity of verification across pipeline stages, where artifact provenance, policy enforcement, and infrastructure protection operate as interdependent mechanisms rather than isolated controls. The study by Solanke [9] shows that financial, medical, and government organizations are forced to simultaneously observe strict control and support continuous delivery, which creates sustained tension between development and security teams. This means that a model assuming an "ideal" pipeline without regulatory constraints inevitably underestimates real risks, and speed and security must be viewed as competing goals, not automatically aligning ones.

A significant methodological challenge is the overhead of automation and dynamic protection methods. Previous studies address this problem from different perspectives but rarely connect it to pipeline-wide methodological constraints. For example, Koneru [3] emphasizes automated testing integration, while Solanke [9] focuses on compliance-driven DevSecOps adoption, and Ugale and Potgantwar [10] analyze container protection primarily as an infrastructure-level problem. However, these approaches remain largely domain-specific. The present study extends prior research by demonstrating that performance overhead, compliance requirements, and infrastructure complexity represent interconnected limitations influencing the overall resilience of CI/CD pipelines. Muñoz and his colleagues [5] show that provenance metadata, multi-stage artifact signing, and attestations increase the load on the pipeline and complicate configuration, while Ugale and Potgantwar [10] demonstrate that multi-level container protection with regular scanning and dedicated execution control requires significant resources, is prone to false positives, and needs constant tuning. Collectively, this indicates that methodologies treating security

automation as a "free" enhancement ignore the impact of overhead on pipeline architecture and throughput.A separate group of problems is related to compliance automation and legacy infrastructure. The study by Solanke Reference [9] shows that manual audits scale poorly with frequently updated regulatory requirements and complex hybrid pipelines, while many organizations depend on legacy systems without modern access control, logging, and automated testing mechanisms that cannot be quickly replaced. From a methodological point of view, this means that CI/CD Security analysis based only on "greenfield" architectures without accounting for the legacy layer forms an unrealistic basis for conclusions. The identified methodological limitations should be interpreted not as independent implementation barriers but as interacting constraints defining the feasibility boundaries of CI/CD Security integration. Table 3 synthesizes these constraints and illustrates how mitigation strategies emerge from aligning security automation with organizational and architectural realities rather than increasing isolated control mechanisms.

Table 3: Methodological challenges of integrating security processes into CI/CD and mitigation approaches (Compiled by the author based on sources: [1, 4, 10])

Challenge	**Manifestation in studies**	**Mitigation approaches**
Balancing speed and security in DevSecOps	Tension between rapid releases and strict security checks in regulated industries	Early security integration, developer training, shared responsibility
Compliance automation	Manual audits do not scale with evolving regulations (GDPR, HIPAA, PCI-DSS, FedRAMP)	Compliance-as-code, continuous compliance monitoring
Dependency and supply chain risk management	Rapid growth of supply chain attacks and need for metadata-driven verification	SBOM, artifact signing, provenance metadata, SLSA levels
Overhead of runtime security and image scanning	Extra CPU, memory, time costs and false positives in container security tools	Multi-layer container security framework, tuning, staged rollout
Policy consistency in multi-cloud / cross-platform	Heterogeneous clouds and platforms with divergent APIs and controls	Unified security framework, central policy definition and mapping
Legacy systems integration	Lack of modern security controls in legacy applications and infrastructures	Phased modernization, containerization, segmentation

The comparison presented in Table 3 clarifies that CI/CD Security maturity depends less on the number of implemented tools and more on the consistency of policy enforcement and trust relationships across pipeline stages. This observation helps explain why organizations with extensive security tooling may still experience supply chain incidents when controls remain fragmented. Consequently, methodological effectiveness should be

evaluated through integration coherence rather than technological intensity alone.

Comparing the identified challenges shows that the methodological model of CI/CD Security must simultaneously account for regulatory-organizational, resource-technological, and architectural-legacy constraints. Ignoring any of these groups leads to a shift in emphasis either toward idealized speed or toward an unattainable level of security.

Consequently, a grounded methodological approach to integrating security processes into continuous development pipelines should view early check shifting practices, compliance automation, multi-level container protection, and legacy management not as separate directions, but as an interconnected set of assumptions and limitations defining the boundaries of applicability for proposed solutions.

Further development of the CI/CD Security methodological model is related not so much to adding particular practices as to redefining the role of observability, trust, and measurability in continuous development pipelines. The study by Pan and his colleagues [7] shows that machine learning models trained on pipeline and operational environment telemetry can be used to identify anomalies and reduce incident detection time by analyzing logs, metrics, and events at different CI/CD stages. Ugale and Potgantwar [10] demonstrate that a specialized execution control container and analysis of behavioral characteristics of containers allow recording attacks and forming a basis for predictive analysis in large cluster environments. In this configuration, pipeline and container platform telemetry is viewed as a single observation space, and classical vulnerability scanners as a source of training data for predictive models, not the final defense line.

The work of Solanke [9] emphasizes the role of self-healing systems that automatically eliminate vulnerabilities, incorrect settings, and incidents by applying patches, changing access rights, and isolating workloads without direct human participation. Embedding such mechanisms shifts the focus from a one-time "compliance check" to a continuous cycle of "detect – analyze – automatically fix – verify," where specialist intervention becomes a calibration tool rather than the sole response method.

Building a mature CI/CD Security model is impossible without a robust measurement system. Patel [8] proposes a set of metrics including mean time to detect and resolve incidents, the share of signed artifacts, the relevance of software component lists, and the share of builds with policy violations. These indicators should be interpreted as operational benchmarks and a basis for grading maturity levels: low detection and remediation times with a high share of signed artifacts and up-to-date component lists can be interpreted as signs of stable integration of security processes into the pipeline, while an increase in policy violations and obsolescence of component lists acts as an indicator of methodological gaps.

Thus, a prospective methodological approach to CI/CD Security should combine intelligent telemetry analysis, self-healing mechanisms, cryptographic tools reinforcing trust boundaries, and formalized maturity metrics. Only by simultaneously accounting for these elements does it become possible to transition from describing particular DevSecOps practices to a holistic theoretical model in which delivery speed and resilience to attacks are viewed as jointly managed parameters of an engineering solution.

5.Limitations of the Study

This study has several limitations. First, the research is conceptual and does not include empirical validation based on industrial datasets, which limits quantitative evaluation of proposed mechanisms. Second, the analysis primarily considers cloud-native and containerized environments, while organizations relying on legacy infrastructures may face additional constraints not fully reflected in the model. Third, rapidly evolving DevSecOps tooling and regulatory standards may influence the long-term applicability of specific practices discussed in the study. Future research should therefore include empirical validation and comparative industrial case studies.

6.Conclusion

The conducted research shows that CI/CD security cannot be viewed as a set of "add-ons" to an existing pipeline. It becomes meaningful only when each stage, from source code management to deployment, is interpreted simultaneously as a source of specific risks and as a point for embedding formalized control processes. In this logic, it is not the presence of individual tools that is important, but the consistency of their application and strict rules for artifacts passing through the pipeline.

The analysis of integrating container security and artifact provenance management allows for a fundamental conclusion: infrastructure protection and supply chain protection are not competing, but complementary axes of a single model. Without multi-level protection of images, orchestration, host systems, and runtime environments, any scheme remains a declaration. Without verifiable artifact provenance and managed component lists, even well-fortified infrastructure does not guarantee protection against targeted supply chain attacks. Methodologically, only their coherent combination is justified.

Identified methodological limitations show that attempting to describe "ideal" DevSecOps without accounting for regulatory pressure, automation overhead, and legacy systems leads to a systematic underestimation of real risks. Integrating security into CI/CD inevitably relies on trade-offs between check depth, pipeline throughput, and the organization's ability to maintain complex policy and technical contours. Therefore, any recommendations prove viable only insofar as they include these limitations directly in the model rather than leaving them out of the equation.

Prospective development of the approach to CI/CD Security is related to the transition from "one-time checks" to continuous, self-correcting security based on telemetry and formalized maturity metrics. Such a shift allows delivery speed and attack resilience to be viewed not as mutually exclusive goals, but as parameters that can be managed within a single engineering model, where the pipeline is designed from the outset as an object of security, not as its side effect.

References

[1]. Coston, I., Hezel, K. D., Plotnizky, E., & Nojoumian, M. (2025). Enhancing secure software development with AZTRM-D: An AI-integrated approach combining DevSecOps, risk management, and zero trust. Applied Sciences, 15(15), 8163. https://doi.org/10.3390/app15158163

[2]. Karanam, R. (2024). Securing CI/CD pipelines: Strategies for mitigating risks in modern software delivery. International Journal of Engineering and Technology Research, 9(2), 1–9. https://doi.org/10.5281/zenodo.13365012

[3]. Koneru, N. M. K. (2021). Integrating security into CI/CD pipelines: A DevSecOps approach with SAST, DAST, and SCA tools. International Journal of Science and Research Archive, 3(1), 250–265. https://doi.org/10.30574/ijsra.2021.3.1.0080

[4]. Mohammed, K. I., Shanmugam, B., & El-Den, J. (2025). Evolution of DevSecOps and its influence on application security: A systematic literature review. Technologies, 13(12), 548. https://doi.org/10.3390/technologies13120548

[5]. Muñoz, A., Farao, A., Correia, J. R. C., & Xenakis, C. (2021). P2ISE: Preserving project integrity in CI/CD based on secure elements. Information, 12(9), 357. https://doi.org/10.3390/info12090357

[6]. Obuse, E., Akindemowo, A., Ajayi, J. O., & Erigha, E. D. (2024). A conceptual framework for CI/CD pipeline security controls in hybrid application deployments. International Journal of Future Engineering Innovations, 1(2), 25–47. https://doi.org/10.54660/IJFEI.2024.1.2.25-47

[7]. Pan, Z., Shen, W., Wang, X., Yang, Y., Chang, R., Liu, Y., Liu, C., Liu, Y., & Ren, K. (2024). Ambush from all sides: Understanding security threats in open-source software CI/CD pipelines. arXiv. https://doi.org/10.48550/arXiv.2401.17606

[8]. Patel, D. G. (2025). Enhancing CI/CD security with provenance metadata and supply chain best practices. World Journal of Advanced Engineering Technology and Sciences, 16(1), 19–32. https://doi.org/10.30574/wjaets.2025.16.1.1181

[9]. Solanke, A. A. (2022). Enterprise DevSecOps: Integrating security into CI/CD pipelines for regulated industries. World Journal of Advanced Research and Reviews, 13(2), 633–648. https://doi.org/10.30574/wjarr.2022.13.2.0121

[10]. Ugale, S., & Potgantwar, A. (2023). Container security in cloud environments: A comprehensive analysis and future directions for DevSecOps. Engineering Proceedings, 59(1), 57. https://doi.org/10.3390/engproc2023059057