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

Security Methods for Agentless Infrastructure Management via Continuous Delivery Pipelines

Oleksandr Shevchenko*

TCS, SRE, Aliso Viejo, US
Email:alex.sv4k@gmail.com

Abstract

This article presents an analysis of security methods for agentless infrastructure management in the context of continuous delivery. The study is conducted as a structured analytical review and synthesis of academic publications focused on CI/CD security, software supply chain protection, infrastructure as code, policy management, and control automation. The main focus is placed on the relationship between delivery pipeline characteristics, risk distribution, threat detection mechanisms, and their impact on system resilience. Key sources of risk are examined, including code, configurations, dependencies, credentials, and the runtime environment, along with control parameters that determine the level of infrastructure security. It is established that risk in such systems is systemic in nature and emerges through transitions between stages rather than at isolated points. Existing approaches based on fragmented checks and static models are shown to lack continuity of control and to lose effectiveness in the absence of inter-stage connectivity. An original model is proposed, reflecting the transition from isolated protection mechanisms to a unified framework integrating telemetry, supply composition, behavioural data, and automated response. The article may be of interest to specialists in information security, DevSecOps, cloud technologies, and infrastructure management.

Keywords: risk management; continuous control; supply chain security; software infrastructure as code; infrastructure management.

Received: 5/21/2026
Accepted: 7/21/2026
Published: 7/31/2026

---

* *Corresponding author.*

1. Introduction

Modern infrastructure is increasingly managed without manual intervention. Changes are introduced through code and automated processes. Release cycles accelerate, and scaling becomes simpler. Risk shifts toward the change delivery pipeline, which integrates build, deployment, and environment management [2]. Pipelines grow more complex and take over control of applications, infrastructure, dependencies, and access. In cloud environments, this logic is further reinforced, as infrastructure is defined as code and executed automatically, effectively turning the pipeline into the central system control mechanism.

The nature of attacks is also changing, moving toward the level of the supply chain, as well as build and deployment processes. The compromise of a single component can affect the entire infrastructure and propagate vulnerable changes. Traditional security methods focus on individual components and predefined scenarios [5]. As a result, they fail to keep pace with the speed of change and do not cover the entire pipeline. Risk emerges from the combination of code, configurations, dependencies, and the behavior of the environment after deployment. At the same time, agentless infrastructure management is becoming more widespread, where all operations are executed through code and centralized processes.

In the context of this study, agentless infrastructure management refers to an approach in which no persistent software agents are installed on managed nodes. Instead, all configuration, provisioning, and deployment operations are executed remotely through code-driven, centralized processes, typically via SSH, APIs, or cloud-native control planes. This distinguishes it from agent-based management, where dedicated software runs continuously on each managed host to enforce state and report telemetry. The agentless model introduces a distinct security profile: the absence of local agents eliminates an entire class of agent-related vulnerabilities (update management, credential storage on nodes, lateral movement through agent compromise), but simultaneously concentrates control authority in the pipeline and its execution environment. As a result, the pipeline itself becomes the primary attack surface, and any compromise of the centralized process can propagate across the entire managed infrastructure without local safeguards.

Modern systems increasingly rely on continuous data collection and automated responses to deviations, which makes security an ongoing process [3]. Approaches to threat modeling, supply chain protection, and policy management continue to evolve separately, while a unified model for such environments is still lacking.

The aim of the study is to develop an original security model for agentless infrastructure management integrated into continuous delivery processes. To achieve this aim, the following objectives are set:

– to analyse approaches to securing build processes, deployment, and infrastructure defined as code.

– to identify the main threats to infrastructure managed through automated processes.

– to systematise protection methods in agentless management environments.

– to determine the limitations of existing security approaches.

The research hypothesis is that effective protection of infrastructure in agentless management is achieved not by increasing the number of isolated checks, but by transitioning to a continuous, risk-oriented, and automated security model embedded in the change delivery process. This hypothesis is evaluated through conceptual analysis and synthesis of existing evidence rather than through direct empirical experimentation.

The scientific novelty of the study lies in a shift in the perspective on infrastructure security. Instead of analysing individual tools, security is treated as an inherent property of the managed change process implemented through the delivery pipeline. The study demonstrates the limitations of approaches based on static checks and isolated control stages and substantiates the need to move toward a continuous, risk-oriented model. It proposes the integration of threat modelling, supply chain security, policy management, and automated response into a unified framework operating within an agentless infrastructure management environment.

2. Materials and Methods

The study is based on methods of theoretical analysis of academic publications, thematic classification of security factors in build and deployment processes, and comparative analysis of the relationships between delivery pipeline characteristics, infrastructure vulnerabilities, and applied protection measures. The focus is placed on identifying the mechanisms through which automated infrastructure management influences risk formation and system resilience. Particular attention is given to interpreting security as a managed process embedded within the change delivery pipeline, rather than as a set of isolated tools.

The study was conducted as a structured analytical review of open-access academic publications from 2022 to 2026, published in international peer-reviewed journals and academic databases. The literature search was performed in Google Scholar, ScienceDirect, SpringerLink, and MDPI using combinations of the following keywords: “CI/CD security”, “pipeline security”, “infrastructure as code security”, “agentless infrastructure management”, “software supply chain security”, “policy as code”, “continuous compliance”, “cloud security automation”, “DevOps security”, “threat modeling STRIDE”, “SLSA”, “SSDF”, and “runtime security”, with the application of AND/OR logical operators. The selection included English-language publications containing empirical or review-based evidence on the security of automated processes, infrastructure management, and development environment protection. Studies focused exclusively on application-level vulnerabilities without consideration of delivery processes and infrastructure layers were excluded.

At the initial screening stage, 36 publications were identified. After removing duplicates and reviewing titles and abstracts, studies not aligned with the research scope were excluded. Based on full-text analysis, a final sample of 15 studies was selected, reflecting key approaches to ensuring security in automated infrastructure management environments.

The analytical procedure included sequential stages of source selection, deduplication, thematic filtering, in-depth content analysis, and result systematisation. The analysis identified groups of factors, including delivery pipeline characteristics, threat types, protection methods, and limitations of existing approaches. It was established that risks emerge at the intersection of code, configurations, dependencies, and execution mechanisms. Protection

methods are distributed across process stages and do not form a unified system. Policy management and automated response evolve in parallel and are rarely integrated into a single security framework.

The limitation of the sample is associated with the focus on a specific domain that combines build processes, deployment, and infrastructure management. A significant portion of the literature addresses security either at the application level or at the level of individual tools, while issues related to the holistic organisation of security within the change delivery pipeline remain fragmented.

The results obtained were used to systematise security factors and to develop an original model of infrastructure protection embedded in the delivery process, reflecting the relationships between threats, control mechanisms, and adaptive response in the context of automated management.

3. Results and discussion

The analysis identifies trust transfer between pipeline stages as the main point at which security risk accumulates. Source code enters the build environment under an assumed author identity; the build process produces an artifact whose integrity depends on the runner, dependencies, and build configuration; deployment transfers that artefact into an environment governed by credentials and infrastructure policies. Each transition carries forward the state produced by the preceding stage. A control that validates one component cannot establish the integrity of the entire sequence when later stages accept earlier outputs without renewed verification.

This mechanism is especially consequential in agentless infrastructure management. Provisioning and configuration commands originate from a central execution environment and reach managed resources through SSH, APIs, or cloud control planes. Managed nodes do not contain persistent agents capable of independently enforcing the desired state or reporting local deviations. The pipeline therefore holds broad authority over infrastructure changes. Compromise of its execution context, credentials, or configuration can affect multiple resources through the same trusted route.

The absence of local agents removes risks associated with agent installation, patching, and credential storage on managed nodes. It also changes the distribution of control. Authentication, policy enforcement, configuration validation, and auditability become more dependent on the pipeline and the services connected to it. This concentration produces a larger blast radius when a pipeline credential or privileged runner is compromised. An agentless architecture consequently requires stronger controls over the identity of each change, the provenance of artefacts, the scope of credentials, and the relationship between deployment events and runtime behaviour.

The reviewed detection models differ in the unit of analysis they use. Static systems evaluate an event, file, or configuration against predefined rules. Adaptive systems interpret a signal within a changing execution context and may update the system response when new evidence appears. Table 1 presents the performance values.

Table 1: Comparison of Threat Detection Models [1]

Model	Detection Accuracy, %	False Positive Rate, %	Approach Characteristics
Static IDS	72,8	11,3	Rule- and signature-based
TADM	77,5	9,7	Static anomaly analysis
AutoGuard	95,6	6,4	Adaptive model with automated recovery

Within the reported experiment, AutoGuard exceeded the Static IDS by 22.8 percentage points in detection accuracy and reduced the false positive rate from 11.3% to 6.4%. The latter corresponds to a relative reduction of approximately 43%. Compared with TADM, the difference was 18.1 percentage points in accuracy and 3.3 percentage points in false positives. These figures describe the models tested by [1]; they do not constitute an experimental evaluation of the integrated model proposed in this article.

The performance difference can be explained by the information available at the point of classification. A signature-based system decides whether an event matches a known rule. A context-aware model can relate the same event to preceding changes, dependency information, execution conditions, and subsequent behaviour. A configuration modification that appears legitimate in isolation may warrant a different decision when it follows an unusual identity event, introduces an unverified dependency, or precedes an unexpected runtime action. Correlation changes the evidential value of each signal.

This interpretation is consistent with published findings on security integration in GitHub and Azure DevOps [6] and continuous security controls in cloud CI/CD processes [14]. These studies support embedding detection within the delivery process rather than relying entirely on periodic external checks. The available studies, however, use different tools, datasets, and evaluation procedures. Their results cannot be combined into a single performance estimate.

Static controls retain a clear function. Syntax checks, policy validation, dependency scanning, and signature-based detection can reject known defects at relatively low cost. Their limitation lies in the scope of the decision. They identify what violates a rule at a particular stage but cannot determine whether several individually permissible events form a risky sequence. Adaptive analysis addresses that problem by preserving context across stages, although it introduces dependence on telemetry quality, correlation logic, and response policies.

SLSA provides structured controls for build integrity and artifact provenance, but its protection is uneven across STRIDE categories. Table 2 reproduces the mapping presented in [3]. The table should be read as an analysis of coverage within that study rather than as an independent certification of each SLSA level.

Table 2: STRIDE Threat Coverage in SLSA [3]

STRIDE Category	SLSA Coverage by Levels	Key Gap	Practical Implication
Spoofing	L1 – none; L2 – none; L3 – yes; L4 – partial	Does not cover developer identity and tokens	Requires independent authentication and access control
Tampering	L1 – partial; L2 – yes; L3 – yes; L4 – yes	Changes before provenance verification stage	Requires source code control and configuration protection
Repudiation	L1 – partial; L2 – yes; L3 – yes; L4 – yes	Insufficient activity logging	Requires immutable logging
Information Disclosure	L1 – none; L2 – partial; L3 – yes; L4 – yes	Leakage of secrets and tokens	Requires secrets management and limited token lifetime
Denial of Service	L1–L4 – none	Not addressed	Requires separate resilience measures
Elevation of Privilege	L1 – none; L2 – none; L3 – none; L4 – partial	Insufficient access control enforcement	Requires strict access control model

The strongest coverage concerns tampering with build inputs and outputs. Provenance can establish where an artefact originated, which process produced it, and whether it remained unchanged after verification. That evidence is central to software supply chain security. It does not determine whether the identity initiating deployment should possess access to the target environment, whether a token grants excessive privileges, or whether a valid artifact behaves safely after release.

Spoofing and elevation of privilege expose a specific weakness in agentless environments. A pipeline may produce a verifiable artefact while using a compromised cloud token to deploy it beyond its intended scope. Provenance remains valid because the artefact itself was built correctly. The security failure occurs in the authorization path. Identity verification, short-lived credentials, separation of duties, and restrictions on deployment scope therefore operate alongside SLSA rather than as substitutes for it.

Denial of service presents a different boundary. Build integrity controls do not establish whether deployed infrastructure can withstand resource exhaustion or the loss of a dependency. Availability requires measures at the architecture and runtime levels, including capacity controls, recovery procedures, and restrictions on changes that can remove or overload critical resources. The reviewed mapping confirms that supply chain maturity cannot serve as a complete measure of infrastructure security. The STRIDE-based analysis complements the broader threat landscape of open-source CI/CD pipelines [13]. The latter demonstrates how such pipelines can be attacked from multiple directions, whereas the SLSA mapping clarifies which threat categories provenance-oriented controls can address. Taken together, these studies show that pipeline security extends beyond artefact integrity. The remaining exposure lies in identities, credentials, execution environments, and post-deployment behaviour.

The reviewed literature contains controls for each major pipeline asset, but those controls often produce separate findings and apply different decision criteria. Code scanners report defects in source files; dependency tools inspect packages; provenance mechanisms attest to artefacts; identity systems authorize deployment; runtime monitoring records behaviour after release. Coverage across these domains does not guarantee that their findings will be connected to the same change.

Table 3: CI/CD Protection Methods [14]

Protection Scope	Security Function	Control Mechanism	Stage Linkage	Threat Addressed
Code and IaC	Static validation	Syntax and policy verification	Design → Build	Misconfigurations, insecure code paths
Container artifacts	Integrity control	Image validation and provenance tracking	Build → Delivery	Compromised or altered artifacts
Dependencies and OSS	Integrity control	Dependency analysis and SBOM control	Build → Runtime	Vulnerable or malicious libraries
Pipeline and runtime	Behavioral monitoring	Logging, telemetry, anomaly detection	Delivery → Runtime	Hidden deviations and intrusions
Access and secrets	Identity enforcement	Authentication, authorization, secret isolation	All stages	Unauthorized access and leakage

The stage linkages in Table 3 expose the point at which fragmentation becomes operationally significant. A dependency scanner may identify a vulnerable package during the build, while runtime monitoring later detects abnormal network activity. If the two records cannot be associated with the same artefact and deployment, analysts receive separate alerts. The first lacks behavioural evidence; the second lacks supply chain context. Connecting the records changes the decision because the combined evidence supports a more specific assessment of origin, exposure, and likely impact.

The integrated model proposed in this study organizes these functions around a shared change context. It brings together telemetry, software composition, provenance, identity events, policy results, and behavioural data. Correlation depends on identifiers that remain traceable across stages: a source revision, build record, artifact digest, deployment event, target environment, and initiating identity. The model treats these records as related evidence about one change rather than independent outputs from separate tools.

Risk assessment occurs after this evidence has been associated with the relevant change and environment. A policy violation in IaC may block a build immediately because its meaning is already clear. A lower-confidence anomaly may require supporting evidence from identity or runtime records. The response can then match the observed condition: reject the change, suspend deployment, restrict a credential, initiate rollback, or refer the case for manual review. The model defines this decision structure conceptually; it does not prescribe a universal scoring

formula or threshold.

Runtime evidence completes the control cycle. If behaviour after deployment differs from the state anticipated during validation, the resulting information should influence the treatment of the artefact and subsequent changes derived from it. Without that return path, pre-deployment controls continue to evaluate new releases using assumptions that runtime evidence has already challenged. The feedback mechanism therefore links deployed behaviour to future policy decisions and incident investigation. Figure 1 represents this sequence as a continuous relationship among pipeline evidence, contextual risk assessment, policy enforcement, and response. The model requires no persistent management agent on each node. It depends instead on pipeline logs, control-plane events, artifact metadata, identity records, and available runtime telemetry. Its visibility is consequently limited by the quality and completeness of those sources.

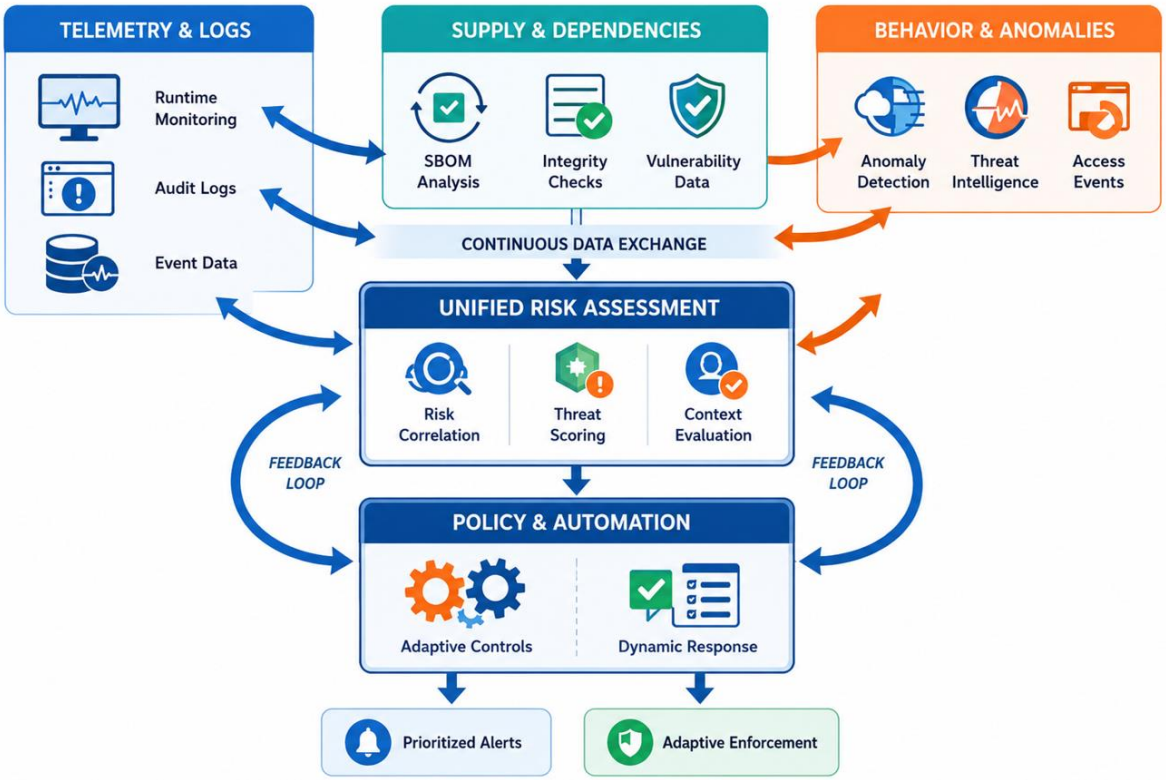

Figure 1: Integrated Security Model for Agentless Infrastructure Management (Author's own development)

The model extends existing frameworks at a different level of abstraction. NIST SSDF organizes secure development practices and defines controls that organizations can assign to development activities. SLSA concentrates on the integrity and provenance of software artefacts. Both supply essential control structures, but neither by itself specifies how evidence from identity, build, deployment, and runtime systems should be correlated for each infrastructure change. The proposed model supplies that connective function.

Persistent difficulties in integrating security into DevOps processes have been documented [5]. DevSecOps practices and tools have also been systematically reviewed [12]. These studies explain why security activities enter different parts of the delivery process, but a catalogue of practices does not resolve the loss of context

between them. The present analysis treats that loss as a source of risk in its own right. Fragmentation across software supply chain research, including dependency management, build infrastructure, and distribution, has likewise been identified [11]. This fragmentation explains why strong controls in one domain may coexist with exposure in another. The proposed model responds by maintaining an association between evidence produced in those domains. It does not replace dependency analysis, provenance, or distribution controls; it defines how their outputs can contribute to a shared assessment. Multiple pipeline types have been conceptualized through ArtifactOps and ArtifactDL [7]. This process-oriented perspective supports the view of the pipeline as an operational system whose artefacts and transitions can be described explicitly. The security model developed here applies a similar process-level perspective to trust: every transfer of code, configuration, artefact, credential, or execution authority becomes a point at which evidence must remain connected.

Research on cloud-native systems adds an environmental constraint. The relationship between availability, scalability, and security during migration from container-based to cloud-native applications has also been examined [9]. These properties depend on runtime conditions that build-stage verification cannot observe. A pipeline model that ends at deployment would therefore miss part of the risk created by infrastructure changes. Runtime telemetry is included because deployment changes the state being protected.

Studies of automation and artificial intelligence address another part of the model. DevSecOps has been connected with large language models and security chaos engineering [2], while AI applications for infrastructure as code have been systematically reviewed [10]. Generative AI has also been examined as an infrastructure copilot across the DevSecOps lifecycle [4]. These works support greater automation in infrastructure analysis and change generation, but automation also increases the need for traceable decisions. A generated configuration remains subject to provenance, policy, authorization, and runtime verification. The model therefore treats AI-assisted changes as pipeline inputs governed by the same control chain rather than as a separate trust category.

Continuous compliance provides a data-oriented basis for evaluating infrastructure state throughout delivery [15]. The integrated model adds a security interpretation to that flow by connecting compliance results with supply composition, identity, and behaviour. The evolution of DevSecOps toward controls embedded across the lifecycle has likewise been described [8]. The unresolved issue is the continuity of evidence: lifecycle coverage has limited value when each stage evaluates a different representation of the same change.

The model is most applicable where infrastructure changes occur frequently and pass through centralized automation. Terraform-managed cloud resources, Kubernetes deployments orchestrated through CI/CD, and Ansible configurations executed in push mode fit this profile. In these environments, a source revision can trigger a build and deployment with little or no local intervention. Preserving the relationship among the initiating identity, configuration, artefact, target environment, and runtime outcome provides information that isolated checks cannot recover after those links have been lost.

Environments with infrequent releases and extensive manual approval gain less from continuous correlation. The cost of collecting, normalizing, and retaining evidence may exceed the benefit when changes are rare and operators can inspect each transition directly. The model also becomes harder to apply when legacy systems lack stable

identifiers, centralized logs, or reliable control-plane records. These conditions do not invalidate the security logic, but they constrain implementation.

Centralization creates its own exposure. The correlation and policy components process sensitive information and can influence deployments across many resources. An attacker who alters the correlation rules, suppresses telemetry, or changes response policies may cause the system to approve unsafe changes or block legitimate ones. The integrated control plane therefore requires restricted administrative access, immutable audit records, and independent monitoring. These protections follow from the architecture; their performance was not evaluated in the present study.

Automated response also requires calibrated authority. Rollback may be appropriate when a deployment introduces a verified policy violation or a strongly associated runtime deviation. The same action may cause disruption when evidence is incomplete or the affected resource maintains state that cannot be safely reverted. Response policies should distinguish deterministic violations from probabilistic anomalies and reserve ambiguous cases for human review. The conceptual model defines the need for this distinction but does not establish numerical thresholds.

4. Conclusion

The obtained results make it possible to consider security in agentless infrastructure management as a process shaped by the continuous movement and interpretation of signals within the change delivery pipeline. Risk is not fixed at a single point and is not eliminated by an isolated check. It persists across transitions between stages and evolves together with the system. Within this logic, security is determined not by a set of mechanisms, but by their coordinated operation. Connectivity between stages reduces uncertainty and makes risk management continuous.

The stability of security depends on the consistency of control. Even with high accuracy of individual methods, the overall outcome remains unstable if gaps occur between stages. Risk accumulates and intensifies as it moves across stages. With continuous connectivity, a unified framework is formed in which each change is evaluated in the context of the current system state.

The practical significance lies in the shift in the role of security. Control ceases to be a separate stage and becomes part of the change management process. Stability is determined by the ability to maintain connectivity between code, dependencies, and infrastructure. In highly automated, agentless environments, this becomes a critical requirement.

A limitation of this study is the absence of empirical validation; the proposed model is derived from conceptual analysis and synthesis of existing evidence rather than experimental evaluation. Further research may focus on the quantitative assessment of connectivity and on validating the model across different types of infrastructure. A promising direction is the comparison with existing approaches in controlled scenarios and the development of tools for implementing continuous risk assessment. Security in such systems should be understood as a dynamic state shaped through ongoing change. The key contribution of this study lies in redefining security as a continuous,

flow-based property of the delivery system, rather than a sequence of isolated validation steps, thereby offering a conceptual framework that may inform the design of integrated DevSecOps security architectures, subject to further empirical validation.

References

[1]. Anugula, P., Bhardwaj, A. K., Chhibber, N., Tewari, R., Khemka, S., & Ranjan, P. (2025). AutoGuard: A self-healing proactive security layer for DevSecOps pipelines using reinforcement learning. arXiv. https://doi.org/10.48550/arXiv.2512.04368

[2] Bedoya, M., Palacios, S., Díaz-López, D., et al. (2024). Enhancing DevSecOps practice with large language models and security chaos engineering. International Journal of Information Security, 23, 3765–3788. https://doi.org/10.1007/s10207-024-00909-w

[3] Dhandapani, S. (2025). Enhancing software supply chain security through STRIDE-based threat modelling of CI/CD pipelines. arXiv. https://doi.org/10.48550/arXiv.2506.06478

[4] Esposito, M., Robredo, M., Bakhtin, A., et al. (2026). Generative AI as an infrastructure copilot: Automating infrastructure-as-code across the DevSecOps lifecycle. Automated Software Engineering, 33, 58. https://doi.org/10.1007/s10515-026-00600-5

[5] Rajapakse, R. N., Zahedi, M., Babar, M. A., & Shen, H. (2022). Challenges and solutions when adopting DevSecOps: A systematic review. Information and Software Technology, 141, 106700. https://doi.org/10.1016/j.infsof.2021.106700

[6] Manolov, V., Gotseva, D., & Hinov, N. (2026). Analysis of GitHub Advanced Security: Security integration in GitHub and Azure DevOps. Future Internet, 18(2), 99. https://doi.org/10.3390/fi18020099

[7] Miñón, R., Diaz-de-Arcaya, J., Torre-Bastida, A. I., et al. (2025). ArtifactOps and ArtifactDL: A methodology and a language for conceptualizing and operationalising different types of pipelines. Journal of Cloud Computing, 14, 42. https://doi.org/10.1186/s13677-025-00761-w

[8] 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

[9] Nascimento, B., Santos, R., Henriques, J., Bernardo, M. V., & Caldeira, F. (2024). Availability, scalability, and security in the migration from container-based to cloud-native applications. Computers, 13(8), 192. https://doi.org/10.3390/computers13080192

[10] Pahl, C., Sezen, Ö. C., & Hofer, F. (2026). Artificial intelligence for infrastructure-as-code—A systematic literature review. Electronics, 15(4), 755. https://doi.org/10.3390/electronics15040755

[11] Williams, L. et al. (2025). Research directions in software supply chain security. ACM Transactions on Software Engineering and Methodology. https://doi.org/10.1145/3714464

[12] Prates, L., & Pereira, R. (2025). DevSecOps practices and tools. International Journal of Information Security, 24, 11. https://doi.org/10.1007/s10207-024-00914-z

[13] Pan, Z. et al. (2024). Ambush from all sides: Understanding security threats in open-source software CI/CD pipelines. IEEE Transactions on Dependable and Secure Computing, 21(1), 403–418. https://ieeexplore.ieee.org/document/10061526

[14] Saleh, S. M., Madhavji, N., & Steinbacher, J. (2025). A systematic literature review on continuous integration and deployment (CI/CD) for secure cloud computing. arXiv. https://doi.org/10.48550/arXiv.2506.08055

[15] Zakharchenko, A. (2026). Integrating continuous compliance into DevSecOps pipelines: A data engineering perspective. Software, 5(1), 6. https://doi.org/10.3390/software501000