
Secure Software Development Lifecycle Model for Cloud Native Enterprise Platforms

Abstract

Cloud-native enterprise platforms require rapid software delivery while maintaining security across source code, dependencies, container images, infrastructure configurations, identities, and runtime workloads. This study proposes the Secure Cloud-Native Software Development Lifecycle model, called SCN-SDLC, to integrate risk-based security controls throughout planning, architecture, implementation, build, deployment, and operation. The model applies change-exposure scoring, threat analysis, normalized security findings, software bill of materials verification, artifact provenance checks, least-privilege validation, policy-controlled deployment, and runtime assurance. It was evaluated across 100 controlled releases and compared with final-stage security review, scanner-only CI/CD, and standard DevSecOps. SCN-SDLC reduced unresolved high-risk findings reaching approval to 3.4 per 100 releases, compared with 18.6, 13.2, and 8.1 for the comparison approaches. It also shortened remediation time and improved provenance completeness and runtime policy coverage. These findings demonstrate that connected lifecycle evidence can strengthen enterprise cloud security without causing excessive delivery delays, supporting traceable security decisions across complex enterprise environments.

Keywords: cloud-native security, secure SDLC, DevSecOps, software provenance, enterprise platforms

1. Introduction

Cloud-native enterprise platforms combine microservices, containers, orchestration systems, service interfaces, infrastructure-as-code templates, and managed cloud resources to deliver software at high speed. These technologies allow enterprises to scale individual services, automate deployment, and release application changes frequently. However, the same flexibility creates a broad security surface because source code, third-party packages, container images, deployment configurations, identities, secrets, and runtime workloads must remain protected. DevSecOps research identifies practices, organizational challenges, technologies, and measurement as major dimensions of secure software delivery [1]. Security must therefore be integrated across the complete software lifecycle rather than applied only before production deployment.

Conventional security reviews are often performed after development teams have completed most implementation work. This late-stage approach may detect vulnerabilities, but it increases remediation cost because insecure design decisions, dependencies, and deployment assumptions have already influenced several components. A systematic review of DevSecOps adoption found that organizations face difficulties involving people, practices, tools, infrastructure, automation, and the balance between delivery speed and security assurance [2]. Cloud-native platforms make this problem more complex because a single release may include application code, container definitions, policy files, service permissions, and infrastructure changes. A suitable lifecycle model must identify security risks before they move through the delivery pipeline.

Automated scanners are important, but a collection of disconnected tools does not create a secure lifecycle. Static analysis may identify insecure code, composition analysis can examine dependencies, and container scanning can detect known image vulnerabilities, yet the findings may be duplicated or assigned without considering business impact. A review of DevSecOps practices and tools shows that secure delivery requires coordinated activities across planning, development, integration, deployment, and operation [3]. Enterprise teams also require consistent evidence showing which controls were executed, which risks were accepted, and whether deployed workloads continue to satisfy approved policies. Security decisions should therefore be connected through a shared risk model.

Cloud-native enterprise platforms need a lifecycle that combines early threat analysis, secure implementation, verified software provenance, policy-controlled delivery, and runtime feedback. Advanced DevSecOps approaches also investigate automated threat discovery and security chaos engineering to identify weaknesses that conventional scanning may not reveal [4]. However, these capabilities must be arranged within a controlled process so that development speed is maintained without allowing high-risk findings to enter production. This study proposes a Secure Cloud-Native Software Development Lifecycle model, called SCN-SDLC, for continuously assessing security evidence and enforcing risk-based release decisions.

2. Methodology

The proposed SCN-SDLC model divided software delivery into six connected security control gates: planning, architecture, implementation, build, deployment, and runtime assurance. Every change received a Release Security Record containing the affected service, data classification, exposed interfaces, deployment environment, component dependencies, infrastructure modifications, and responsible owner. The model did not require all changes to pass through identical tests. Instead, it selected controls according to the security effect of the proposed change and retained the generated evidence throughout the release process. This structure allowed application, platform, and security teams to work from the same release context.

During planning and architecture analysis, the framework generated a Cloud Change Exposure Score. The score considered internet accessibility, data sensitivity, privilege level, dependency reach, tenancy impact, and recovery criticality. High-scoring changes received detailed threat modelling, trust-boundary examination, misuse-case generation, and architecture approval. Lower-risk internal changes followed a shorter review path while still receiving baseline controls. Software supply-chain research identifies transparency, validity, and separation as important design properties for protecting development and delivery processes [5]. These properties were included when defining repository access, approval separation, build provenance, and artifact acceptance rules.

The implementation gate analysed source changes before they were merged. It combined secure coding rules, secret detection, static application security testing, infrastructure-as-code analysis, and permission-difference checking. Findings were normalized into a common severity scale and linked to the affected code path or configuration object. Duplicate alerts produced by different tools were grouped into one Security Finding Unit. The unit stored exploitability, affected assets, exposure conditions, available mitigation, and remediation status. This reduced repeated developer notifications and allowed the model to distinguish a reachable enterprise risk from a low-impact pattern.

At the build gate, SCN-SDLC created a verifiable software inventory containing direct dependencies, transitive dependencies, container base layers, build tools, package sources, and artifact hashes. The model generated a software bill of materials and checked each component against vulnerability, license, freshness, and provenance requirements. An empirically grounded reference architecture for software supply-chain metadata management demonstrates the need to coordinate heterogeneous metadata standards and tools across development environments [6]. SCN-SDLC therefore linked component information, build attestations, scan results, and release approvals through a common artifact identifier.

The deployment gate evaluated the intended production configuration rather than examining the application artifact alone. Kubernetes manifests, network policies, workload identities, secret references, storage permissions, encryption settings, and ingress rules were tested within an isolated staging namespace. Zero-trust research for cloud-native DevSecOps emphasizes continuous

verification and restricted trust relationships across identities, workloads, and services [7]. The model consequently denied implicit service access and required each deployed workload to demonstrate an approved identity, permitted communication path, and least-privilege policy. Failed controls returned the change to the responsible lifecycle stage instead of producing a general pipeline rejection.

Runtime assurance connected production observations with future development decisions. The model collected workload behaviour, unexpected service communication, privilege changes, image drift, repeated authentication failures, and policy violations. Security chaos experiments were executed in a controlled environment for critical services to examine token misuse, denied dependencies, compromised containers, and unavailable security controls. Research on security chaos engineering indicates that controlled disruption can reveal security weaknesses that remain unnoticed by standard analysis tools [8]. Runtime events were converted into new requirements, regression tests, or control adjustments for the next release cycle.

Table 1 summarizes the six control gates and the evidence used by SCN-SDLC. The model was evaluated using a simulated enterprise platform containing thirty microservices, two customer-facing portals, an identity service, an API gateway, containerized data-processing workloads, and infrastructure deployed through declarative templates. Four delivery approaches were compared across 100 controlled releases: conventional final-stage security review, scanner-only CI/CD, standard DevSecOps, and the proposed SCN-SDLC model. Each approach received the same code changes, dependency updates, configuration defects, vulnerable components, and identity-policy errors. The primary measure was unresolved high-risk security findings reaching release approval, while supporting measures included detection stage, remediation time, false-positive workload, release delay, provenance completeness, and runtime policy coverage.

Table 1. Security control gates and evidence flow within the proposed SCN-SDLC model

Lifecycle control gate	Analysed release elements	Automated security action	Required evidence	Release decision condition
Planning assurance	Business function, data class, exposure and criticality	Calculate Cloud Change Exposure Score	Approved risk profile	Security scope matches change impact
Architecture assurance	Trust boundaries, service paths and privileges	Generate threat and misuse scenarios	Reviewed architecture risk map	Critical threats have assigned controls
Implementation assurance	Source code, secrets, permissions and templates	Execute normalized code and configuration analysis	Resolved Security Finding Units	No reachable critical coding weakness
Build assurance	Packages, images, tools and artifact origin	Verify dependencies, SBOM and build provenance	Signed component and artifact record	Components satisfy risk and origin policy

Deployment assurance	Identities, network rules, manifests and cloud settings	Validate least privilege and environment policy	Staging policy verification report	Intended deployment passes mandatory controls
Runtime assurance	Workload behaviour, drift and security events	Detect violations and execute controlled resilience tests	Runtime assurance feedback record	New risks are returned to lifecycle backlog

3. Results and Discussion

The evaluation showed that security outcomes differed substantially across the four delivery approaches. Conventional final-stage review allowed 18.6 unresolved high-risk findings per 100 releases because many design, dependency, and configuration problems were discovered after the main implementation decisions had been completed. Scanner-only CI/CD reduced this value to 13.2 findings, although duplicated alerts and limited architectural context prevented consistent remediation. Standard DevSecOps achieved 8.1 findings by distributing security checks across the pipeline. SCN-SDLC produced the lowest value at 3.4 unresolved high-risk findings per 100 releases.

Figure 1 compares the number of unresolved high-risk findings that reached release approval. The proposed model reduced such findings by approximately 81.7% compared with final-stage review, 74.2% compared with scanner-only CI/CD, and 58.0% compared with standard DevSecOps. The improvement was not produced by simply adding more scanning tools. SCN-SDLC connected each finding with exposure, affected assets, provenance, deployment context, and runtime evidence. This allowed the release process to block risks that were both severe and operationally reachable while avoiding unnecessary interruption for duplicate or low-impact alerts.

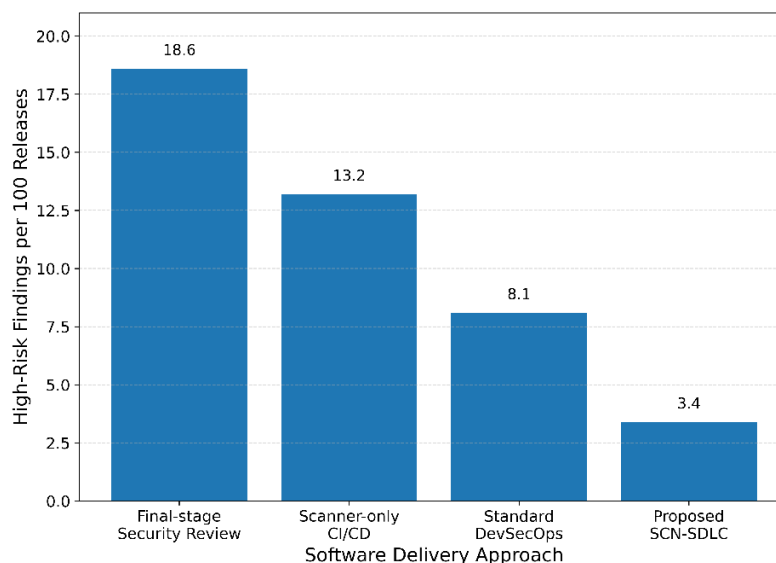


Figure 1. Unresolved high-risk security findings reaching approval across 100 cloud-native releases

Earlier detection also improved remediation efficiency. Under final-stage review, the mean time required to resolve a high-risk finding was 4.8 working days because development and deployment decisions had to be reconsidered. Scanner-only CI/CD reduced the value to 3.6 days, standard DevSecOps required 2.7 days, and SCN-SDLC required 1.6 days. The proposed model returned each failed control to the lifecycle stage where the risk originated. Architecture weaknesses were therefore handled during design, insecure permissions during implementation, and environment-policy violations during deployment validation.

SCN-SDLC achieved 97% provenance completeness and 94% runtime policy coverage across the evaluated releases. It also reduced duplicated security notifications by grouping related scanner outputs into Security Finding Units. The additional lifecycle controls increased the median pipeline duration by 6.4%, but risk-based test selection limited the delay for low-exposure changes. These findings indicate that secure cloud-native delivery benefits from connected evidence and context-aware control gates rather than uniform scanning at every stage. The model provides a balanced approach in which stronger assurance is applied to high-impact enterprise changes without creating the same security workload for every release.

4. Conclusion

This study introduced SCN-SDLC, a secure software development lifecycle model for cloud-native enterprise platforms. The model organized security activities into planning, architecture, implementation, build, deployment, and runtime control gates. It combined change-exposure scoring, threat analysis, normalized security findings, dependency verification, software bills of materials, artifact provenance, identity validation, policy-controlled deployment, and runtime assurance. A shared Release Security Record connected the evidence produced across these stages and supported risk-based approval decisions.

The evaluation showed that SCN-SDLC reduced unresolved high-risk findings reaching release approval and shortened the time required for remediation. Its main practical contribution is the connection of security controls with the specific code, component, configuration, identity, and environment affected by each change. The model can support enterprise platforms that require rapid cloud-native delivery while maintaining traceable security and compliance evidence. Further studies should evaluate the model using production release histories, multiple cloud providers, larger service environments, and longer operational monitoring periods.

References

1. Zhao, X., Clear, T., & Lal, R. (2024). Identifying the primary dimensions of DevSecOps: A multi-vocal literature review. *Journal of Systems and Software*, 214, 112063.
2. 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.
3. Prates, L., & Pereira, R. (2025). DevSecOps practices and tools. *International Journal of Information Security*, 24(1), 11.
4. Bedoya, M., Palacios, S., Díaz-López, D., Laverde, E., & Nespoli, P. (2024). Enhancing devsecops practice with large language models and security chaos engineering. *International Journal of Information Security*, 23(6), 3765-3788.
5. Okafor, C., Schorlemmer, T. R., Torres-Arias, S., & Davis, J. C. (2022, November). Sok: Analysis of software supply chain security by establishing secure design properties. In *Proceedings of the 2022 ACM Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses* (pp. 15-24).
6. Tran, N. K., Pallewatta, S., & Babar, M. A. (2024, June). An Empirically Grounded Reference Architecture for Software Supply Chain Metadata Management. In *Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering* (pp. 38-47).
7. Shin, D., Kim, J., Pawana, I. W. A. J., & You, I. (2025). Enhancing cloud-native DevSecOps: A Zero Trust approach for the financial sector. *Computer Standards & Interfaces*, 93, 103975.
8. Alghawli, A. S. A., & Radivilova, T. (2024). Resilient cloud cluster with DevSecOps security model, automates a data analysis, vulnerability search and risk calculation. *Alexandria Engineering Journal*, 107, 136-149.
9. Sinan, M., Shahin, M., & Gondal, I. (2025). Integrating security controls in DevSecOps: Challenges, solutions, and future research directions. *Journal of Software: Evolution and Process*, 37(6), e70029.