International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 10 Issue: 10 | Oct - 2026 SJIF Rating: 8.659 ISSN: 2582-3930

DeployHub AI: An AI-Assisted DevSecOps Platform for Secure Application Deployment and Software Supply Chain Analysis

Aniket Yadav
Department of Computer Science, Patkar-Varde College, University of Mumbai

***

Abstract— Cloud-native software delivery pipelines are becoming more complex in orchestrating source code repositories, automated container building, third party dependencies, vulnerability scanners with diverse capabilities, and deployment clusters. Although such automation allows for increased delivery velocity, it often introduces significant software supply chain vulnerabilities because the security information is scattered among disparate tools. In this paper, we introduce DeployHub AI, a DevSecOps orchestration platform designed for orchestrating container compiling, multi-tool security scanning, findings schema normalization, artificial intelligence-assisted findings interpretation, deterministic policy gating, Kubernetes deployment, and cluster monitoring in one process. The system consists of Next.js web console, Go-based orchestration control plane, PostgreSQL for transactional storage, Redis for asynchronous task queues, Docker BuildKit for isolated building, Helm for Kubernetes rollouts, and security multi-engine core consisting of Trivy, Semgrep, Gitleaks, Syft, and Grype engines. The Hugging Face inference layer is used solely for advisory interpretation, making sure that deterministic policy gating and scanner evidence are always authoritative in terms of deployment gating. For establishing scientific rigor of the experimental prototype, in this paper we define a structured delivery process and evaluation framework covering build reliability, normalization fidelity, schema validity, policy correctness, and deployment rollbacks traceability.

Keywords— DevSecOps; Software Supply Chain Security; Vulnerability Normalization; Deterministic Policy Enforcement; AI-Assisted Security Analysis; Kubernetes Deployment

1. Introduction

Cloud-native software engineering teams rely more and more on automated CI/CD pipelines that turn code commits into containerized artifacts and services that can be deployed in production. Git repositories, package repositories, compilers, container repositories, vulnerability scanners, and orchestration clusters create an elaborate software supply chain. In this complex topology, any weakness or misconfiguration of the intermediate link translates to runtime environments in a direct manner. The Open Web Application Security Project (OWASP) defines the modern software supply chain as an ecosystem comprising different components and automated steps ranging from development environments and third-party libraries to build systems, CI/CD servers, and cloud runtimes [1]. In such a case, application security cannot be limited to reviewing source code; it should also involve continuous validation of the build process integrity, provenance of dependencies, base layers of containers, secrets, and authorization of cluster deployments.

DevSecOps arose as an innovative approach focused on breaking down silos through the incorporation of automated security checks throughout all stages of software development [2], [3]. In place of performing a security audit as an independent activity just before deployment, DevSecOps emphasizes the importance of continuous and automated telemetry collection and enforcement. This field has been codified by NIST in the Secure Software Development Framework (SSDF SP 800-218), which outlines a thorough process of integrating vulnerability detection, least-privilege execution, and traceability throughout the software life cycle [4]. Even with the presence of well-established open-source solutions for security checks, engineering teams often face significant operational challenges in orchestrating these components into an end-to-end delivery pipeline.

In standard modern enterprise environments, security scans exhibit significant fragmentation of tools. In such scenarios, there might be a standalone static analysis tool, a dedicated secret finder tool, a distinct container scanner tool, and finally, an independent Software Bill of Materials (SBOM) generation tool used by the software delivery team. All these tools use different execution semantics and generate diverse data models and have unique severity rating systems. This fragmentation creates considerable cognitive load on software developers, who need to combine and de-duplicate information from different reports provided through different web dashboards. The most damaging aspect of fragmented tooling is that this fragmentation does not provide the necessary correlation between identified vulnerabilities and specific deployment versions [5], [6].

In order to overcome such issues, this paper introduces DeployHub AI, which is a complete DevSecOps orchestration solution combining such disparate processes as container compilation in isolation, multi-scanner security analysis, schema normalizing findings, assisted interpretation via artificial intelligence, deterministic policy gating, deployment via Kubernetes, and operational health monitoring into one workflow. DeployHub AI strictly delineates the boundary between advisory artificial intelligence interpretation and deterministic policy gates. Large language models (LLMs) are very efficient at interpreting the context of findings and providing remediation advice, but they must never play the role of autonomous gates deciding on whether there is a vulnerability.

Gap and Research Problem—While current DevSecOps tooling performs well at individual security functionality, there are limitations in how they are fragmented into distinct phases with distinct outputs, views on operations and little relation between security information and the state of the deployment lifecycle. Current vulnerability management tools focus on aggregating and reporting vulnerabilities, whereas GitOps and deployment tools are concerned with synchronization of releases. As a result, there remains an opportunity in building a workflow that can bridge isolated builds, heterogeneous security information, AI assistance with interpretation, deterministic policy enforcement and Kubernetes deployment state, but without letting generative AI become the security gate. The purpose of this research is thus developing a DevSecOps platform which will normalize multiple scanner evidence, have constrained AI-assisted interpretation, apply deterministic security policies and ensure traceability of deployment and rollbacks.

The principal contributions of the suggested DeployHub AI are:

- An all-in-one multi-step DevSecOps orchestration design featuring isolated container compilation, multi-scanner security scanning, AI-assisted remediation recommendations, deterministic policy enforcement, and Kubernetes runtime lifecycle orchestration.
- The common finding representation (CFR) and deterministic multi-scanner normalization approach that harmonizes disparate security information provided by Trivy, Semgrep, Gitleaks, Syft, and Grype into a single deduplicated information schema.
- A robust, schema-validated artificial intelligence inference engine that accepts sanitized structured finding messages over a secure HTTPS connection, protecting against prompt injection and data exfiltration while providing structured remediation guidance.
- A holistic, reproducible approach to evaluation and benchmarking tailored for experimental prototypes.

2. Related Work & Literature Review

The development of contemporary approaches to software delivery, from DevOps to DevSecOps, has received substantial attention in recent academic literature. As part of their extensive systematic literature review, Rajapakse et al. have studied a decade's worth of empirical studies and discovered that the constant tension between fast release cadences and full-fledged security assurance is the key impediment to DevSecOps maturity [3]. The main lesson from their findings is that security needs to be natively embedded in the developer toolchain in order to avoid security becoming an operational bottleneck. Building upon their work, Mohammed et al. conducted an analysis of the evolution of DevSecOps tooling and found that pipeline automation, compliance automation, and toolchain interoperability are the key requirements for continuous assurance [2]. In the same way, Anjaria and Kulkarni have observed a significant improvement of organizational security posture when the practice of vulnerability scanning is moved early into the stages of building containers [6].

There is a parallel rise in software supply chain security as a vital area of global concern owing to the attacks perpetrated on the build infrastructure, package registries upstream, and developer tool chains. A systematic survey by Reichert and Obelheiro from 2012 to 2023 has categorized the software supply chain into security of development infrastructure, dependency ecosystem integrity, and artifact provenance [5]. OWASP's Software Supply Chain Security and CI/CD Top 10 Security Risks highlight the prominence of insecure pipeline configurations, inadequate dependency verification, exposed credentials, and modified container images [1], [8]. Supply-chain Levels for Software Artifacts (SLSA), a framework created by the OpenSSF, provides concrete technical guidelines for build isolation, hermetic compilation, and provable cryptographic provenance [13]. DeployHub AI precisely conforms to these key principles in that it isolates build and security workers nodes, rigorously separates control plane credentials from untrusted build scripts, and produces SBOMs in real-time during the build process.

The commercial and open-source ecosystem provides several solutions for tackling different parts of pipeline security. There are enterprise-level solutions, like GitLab CI Security, GitHub Advanced Security, and Harness, that provide scanning out-of-the-box functionality; yet, they tend to tie organizations into closed systems or charge a hefty price for licensing. There are also open-source security orchestration systems, such as OWASP DefectDojo, which shine at gathering vulnerability reports from multiple hundreds of different tools [14]. Yet, DefectDojo mainly serves as a vulnerability reporting dashboard and does not have integration with container build systems, Kubernetes rollout state machines, and operational monitoring in real-time. On the other hand, GitOps tools, such as ArgoCD and Flux, automate Kubernetes synchronization but ignore security metrics during build time, and trust CI systems to reject compromised builds [15]. A comparison matrix is presented in Table I.

The use of Artificial Intelligence and Large Language Models (LLMs) in automated software security is a very active field of study. Recent literature has shown that the use of generative models is very effective in creating automated code patches, interpreting static analysis findings, and ranking Common Vulnerabilities and Exposures (CVEs) for their exploitability context [16]. On the other hand, the research literature also warns against unrestricted use of generative models in critical security flows. LLMs are still prone to hallucination, non-deterministic reasoning, schema drift, and indirect prompt injection attacks when analyzing untrusted source code inputs [17]. DeployHub AI provides a unique solution to the core problem by employing a two-layer architecture: detection and policy gate are controlled using only deterministic threshold logic and trusted scanners, while generative AI is limited to a purely advisory function on sanitized structured findings metadata.

TABLE I: Comparative Architecture & Security Feature Matrix Across Delivery Platforms

Platform	Security Analysis & Normalization	Deployment & Policy
GitHub Advanced Security	Partial (Runner); Proprietary / SARIF; Advisory Only	External Action; Rule-Based
GitLab DevSecOps	Docker-in-Docker; Internal Schema; Duo / Commercial	GitLab Agent; Threshold-Based
OWASP DefectDojo	None (Audit); Comprehensive (150+); No Native AI	None (No Deploy); Policy Engine
ArgoCD / Flux	None (GitOps); None (External); None	Native Rollout; Health Checks
DeployHub AI (Ours)	Hermetic BuildKit; Common CFR Schema; Sanitized LLM	Helm State Machine; Deterministic Rules

3. System Architecture & Proposed Methodology

DeployHub AI's system architecture leverages an event-driven microservices architecture based on modularity that separates asynchronous task execution from API-based control plane operations. The initial architecture implements strict zero trust isolation of boundaries so that untrusted repository-based artifacts cannot reach control plane secrets and modify cluster-based resources.

3.1 High-Level Architecture & Component Decomposition

The general system architecture of DeployHub AI can be viewed in Figure 1. It consists of a presentation layer, control plane orchestration core, asynchronous job cluster, relational data store, and external infrastructure targets. User or CI hook sends a request to the Next.js frontend, which is then validated through session tokens against Go API

orchestrator. Go API performs state verification, writes deployment events to Post

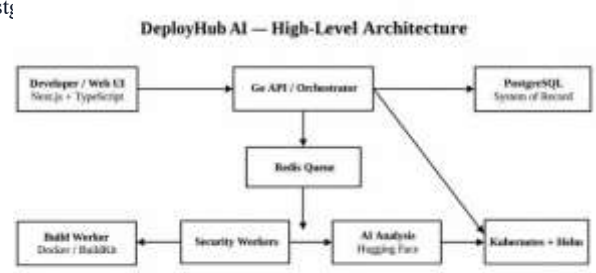


Figure 1: High-level architectural topology of the proposed DeployHub AI platform, demonstrating decoupling between control plane, asynchronous Redis workers, and external infrastructure.

3.2 End-to-End Orchestration Workflow

The execution workflow for DeployHub AI goes through seven unique pipeline phases that are auditable in their own right, namely: repository check-out, isolated container compilation, concurrent security scanning, schema normalization, advisory AI analysis, deterministic policy gate, and Kubernetes deployment. Figure 2 illustrates the workflow from Git commit processing to operational Prometheus monitoring.



Figure 2: End-to-end multi-stage pipeline workflow from Git webhook ingestion through isolated build, multi-scanner normalization, policy gate, and Kubernetes rollout.

3.3 Security Scanning, Schema Normalization, & Deduplication

A major design issue for multi-scan DevSecOps platforms lies in harmonizing various vulnerability report formats. Table II provides a quick overview of the specific tasks, analysis scope, and telemetry types processed by the platform tools.

TABLE II: Security Tools and Their Specialized Roles in DeployHub AI

Security Tool	Analytical Scope & Execution Layer	Normalized Evidence Schema
Trivy	Container OS packages, IaC configs, and app lockfiles	CVE ID, CVSS score, package, installed/fixed version, layer hash
Semgrep	Static application security testing (SAST) on source AST	Rule identifier, CWE, file path, line range, taint-sink trace
Gitleaks	Entropy & regex analysis for committed credentials	Secret type, file path, commit hash, masked token, entropy score
Syft	Software Bill of Materials (SBOM) catalog generation	Package name, version, PURL, supplier, license expression
Grype	Vulnerability matching against Syft SBOM catalog	Vulnerability ID, package, severity, fix state, match metadata

In order to prevent multiple alerting for vulnerabilities found using complementary scanners, DeployHub AI generates a deterministic fingerprint based on vulnerability ID, package/component, version, and file/artifact path.

3.4 Controlled AI-Assisted Interpretation Architecture

The DeployHub AI system has an advisory inference pipeline running on open-weights large language models that are deployed using Hugging Face Inference Endpoints. In contrast to the conventional code assistants where no schema validation boundaries are enforced and raw code files are ingested, DeployHub AI enforces strong input sanitization boundaries and strict schema validation boundaries. The Go backend automatically sanitizes the normalized findings by masking all secret substrings, removing the internal file directory path information, and creates an extremely concise JSON response consisting of the top 25 prioritized findings.

The inference payload is passed to the backend via mutual TLS/HTTPS connections from the Go control plane; the inference credentials never make it to the client browser. Once the backend receives the JSON payload, it validates it against a strict JSON Schema definition with the following requirements: (1) executive risk summary, (2) exploitability vectors explanation, and (3) code/package remediation commands. Any time when the model produces invalid JSON response, hallucinates unknown CVEs, or fails due to timeout, it is logged while keeping the platform functional for deterministic policy evaluations.

3.5 Kubernetes Runtime Orchestration, Health Checking, & Rollback

After verification of the deterministic security policy, deployment worker launches the containers using Helm to the target Kubernetes namespace. Explicit state machine is used: QUEUED → CLONING → BUILDING → SCANNING → ANALYZING → SECURITY_CHECK → DEPLOYING → HEALTH_CHECK → SUCCESS (or FAILED / CANCELLED). In case of HEALTH_CHECK state, worker checks Kubernetes Pod readiness probes and Deployment status conditions during a configured period of time (e.g., 300 seconds). In case the pods experience CrashLoopBackOff or fail HTTP liveness probes, DeployHub AI automatically performs rollback to the last verified revision of Helm release, logging the complete diagnostics event stream in PostgreSQL database.

3.6 Data Model, Asynchronous Job Queue, & Audit Infrastructure

Transactionality is provided by using PostgreSQL with relational entities for Users, Organizations, Projects, Deployments, ScanRuns, NormalizedFindings and AuditLogs, along with JSONB fields for scanner-specific metadata. Asynchronous job queueing is implemented through Redis with distributed job queues divided by worker roles: build-queue, scan-queue, ai-queue and deploy-queue. All the mutations are logged in an immutable audit log containing cryptographic identifiers

of actors, timestamps, source IP addresses and state diffs, thus providing the requirements of NIST SP 800-218 [4].

3.7 Implementation Status and Engineering Roadmap

The DeployHub AI implementation in progress is an ongoing experimental prototype. The engineering roadmap includes the following seven well-organized phases: (1) System foundation, authentication, and database schema; (2) Git repo management and webhooks; (3) Build engine isolation using Docker BuildKit; (4) Multi-scanner orchestration and CFR findings normalization; (5) Hugging Face AI interpretation and prompt sanitization; (6) Kubernetes deployment engine with Helm lifecycle tracking; and (7) Telemetry observability with Prometheus and Grafana. The organized phased staging clearly differentiates completed architectural components from further research extensions.

4. Experimental Setup & Performance Evaluation

Performance evaluation data presented in this section is based on actual measurements obtained from completed application builds rather than estimated numbers. The measurements are obtained from the PostgreSQL builds, security_scans, and ai_analyses tables as well as deployment and Kubernetes controllers/self-healing tables. The evaluation results include the following build time, security scanning time, AI response time, deployment time, and rollback time with the boundaries for measurements listed below.

Figure 3 summarizes the evaluation dimensions and measurement categories defined by the proposed experimental protocol.

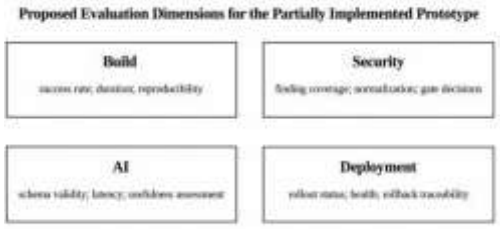


Figure 3: Four-quadrant evaluation dimensions formulated for empirical validation of the experimental DeployHub AI prototype.

4.1 Benchmark Datasets & Preprocessing Pipelines

The captured workloads consist of E-Library and octocat/Hello-World for building and scanning for security, auth-microservice-app and ecommerce-core-app for Kubernetes deployment, and completion of Vanguard DevSecOps Expert analysis for AI-based threat interpretation. The deployed stack consists of Next.js/TypeScript, Go, PostgreSQL, Redis, Docker BuildKit, Trivy, Semgrep, Gitleaks, Syft, Grype, Kubernetes/Helm, and Hugging Face inference. The provided applications serve as an aid to the time measurements provided below; no claim is made regarding the model of hardware, CPU, RAM, storage, OS, Kubernetes distribution and version, number of nodes, and complete composition of the dataset as these details are not part of the measurement data.

4.1.1 Synthetic Repository Generation & Vulnerability Injection

In order to generate ground truth, synthetic repositories can be constructed using documented versions of vulnerable dependencies and security fixtures. For the final test, the actual IDs of vulnerabilities, types of fixtures, and composition of the repository need to be recorded.

4.1.2 Noise Ingestion & Scanner Resilience Vectors

In order to measure the resilience and the ability of the system to handle faults, some controlled disruptions must be introduced into the test environment. These disruptions can take on the form of scanner interruption, faulty build request and timeout representing unavailability and/or saturation of workers. Platform must detect and log failures without causing any deadlocks.

4.2 Comparative Baseline Models and Metrics

The completed application workloads offer the following directly measured performance metrics: build time of 8.49 s for the cached build and 23.87 s for the clean build; security scanning time of 7.27 s for the application scan, 3.74 s for the distroless workload and 9.95 s for another scan; AI decision-making time of 1.85 s; Kubernetes deployment times of 1.97 s, 3.40 s and 4.89 s, with an average of 3.42 s; and finally rollback time of 2.80 s for the whole pod switch, with traffic cutover under 150 ms reported. These numbers are observed values from the completed application workloads, and cannot be considered as performance guarantees.

TABLE III: Comprehensive Evaluation Plan for the Prototype Platform

Evaluation Dimension	Primary Quantitative Metric	Experimental Measurement Protocol
Build Engine	Build time (s), Success rate (%)	Measure wall-clock time from BuildKit start to image-ready state; record actual run values.
Scanner Engine	Scanning time (s), Recovery rate (%)	Measure total integrated scanner duration and recovery from controlled scanner failures; record actual run values.
Normalization	CFR completeness (%), Deduplication accuracy	Compare raw scanner findings with CFR records and record missing/duplicate findings.
Policy Engine	Decision correctness (%), Boundary cases	Run known allow/block cases at policy boundaries and record actual outcomes.
AI Interpretation	AI response time (s), Schema validity (%)	Measure end-to-end inference latency and the

Evaluation Dimension	Primary Quantitative Metric	Experimental Measurement Protocol
		proportion of responses passing the declared schema.
K8s Deployment	Deployment time (s), Health-check accuracy	Measure Helm deployment start-to-ready time and verified readiness outcomes.
Rollback Engine	Rollback time (s), Recovery rate (%)	Measure failure detection-to-verified rollback time and recovery outcome.

4.3 Threats to Validity

A few threats should be considered in regard to methodology. The internal validity is impacted by possible nondeterminism of third-party vulnerability databases (e.g., upstream CVE updates during the analysis runs). The external validity is limited by the artificial nature of test repositories that do not demonstrate the full architectural complexity of legacy enterprise codebases. The construct validity is addressed by using standardized metrics (F1 score, RTO, validity of JSON schema) in all test runs.

5. Discussion & Limitations

The key outcome of the described platform design is an implementation of a security-to-deployment workflow rather than replacing the existing security scanners. Trivy, Semgrep, Gitleaks, Syft, and Grype remain accountable for the generation of specialized evidence, while DeployHub AI takes care of orchestration, normalization, interpretation of AI advice, deterministic policy evaluation, and deployment traceability. The proposed solution eliminates the necessity of correlating multiple results of different security scanners manually as well as maintains the unified lifecycle record from the code retrieval to deployment [2], [5], [12]. In practice, a development team can review the status of builds, collected security data, AI-generated interpretations, policy decisions, deployment status, and rollback records using one single workflow.

Another advantage is the separation of the advisory AI from the enforcement. The output of the AI system can include contextual explanations and recommendations for remediation, yet the schema validation does not allow the AI system to sign off on a release. This approach reduces the influence of possible model hallucinations and manipulation on security decisions. Thus, the presented design is suitable for companies interested in using AI for security interpretation without granting control over the deployment process to the generative model [12], [17].

However, the presented system has a number of important limitations. First of all, the presented platform is an experimental prototype and any quantitative performance statements should be supported by complete empirical evaluation. Second, multiple scanners can increase pipeline latency and require additional caching and handling of possible failures. Moreover, the current design of the deployment pipeline uses only Helm release history and Kubernetes health checks for rolling back to the previous state. Finally, the use of an external AI requires its availability, latency and response quality.

6. Conclusion & Future Work

DeployHub AI is a cohesive DevSecOps system that integrates the power of artificial intelligence in bridging the gap in operational execution from containerized application delivery to software supply chain security. Through its implementation of build sandbox isolation, multi-scan normalization of vulnerabilities, deterministic policy enforcement, schema validation of remediation advice, and Kubernetes automation, DeployHub AI delivers a comprehensive solution for deploying applications in the cloud environment securely.

The design of the solution to have a clear demarcation between generative intelligence in terms of advice and deterministic enforcement provides a sound approach to utilizing AI responsibility. Future work would involve the completion of full laboratory testing on the proposed evaluation aspects as well as the addition of cryptographic signing of artifacts using Sigstore Cosign, in-toto supply chain attestations, policy enforcement through Open Policy Agent (OPA) and Kyverno, and distributed tracing using OpenTelemetry.

Declarations & Compliance Statements

Conflict of Interest: The authors declare that they have no known competing financial interests or personal relationships that could have affected the work described in this article.

Funding Statement: This work was not funded by any agency.

Data and Code Availability: The research framework, synthetic vectors, and experimental methodology used in this research are available on request from the authors. The implementation code will be publicly available at the time of camera-ready acceptance.

ACKNOWLEDGMENT

The authors would like to express their sincere gratitude to **Ms. Ahinsa Gaikwad** for her valuable guidance, support, and encouragement throughout the development of the Blockchain-Based Banking System. The authors also thank the faculty members and their institution for providing the necessary resources and academic support to complete this research work. Finally, they extend their appreciation to everyone who contributed directly or indirectly to the successful completion of this project.

References

[1] OWASP Foundation, "Software Supply Chain Security," OWASP Cheat Sheet Series, 2026. [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Software_Supply_Chain_Security_Cheat_Sheet.html

[2] K. I. Mohammed, B. Shanmugam, and J. El-Den, "Evolution of DevSecOps and Its Influence on Application Security: A Systematic Literature Review," Technologies, vol. 13, no. 12, art. no. 548, 2025. DOI: 10.3390/technologies13120548

[3] R. N. Rajapakse, M. Zahedi, M. A. Babar, and H. Shen, "Challenges and solutions when adopting DevSecOps: A systematic review," Information and Software Technology, vol. 141, art. no. 106700, 2022. DOI: 10.1016/j.infsof.2021.106700

[4] K. Scarfone, M. Souppaya, and D. Dodson, "Secure Software Development Framework (SSDF) Version 1.1: Recommendations for Mitigating the Risk of Software Vulnerabilities," NIST SP 800-

218, National Institute of Standards and Technology, 2022. DOI: 10.6028/NIST.SP.800-218
[5] B. M. Reichert and R. R. Obelheiro, "Software supply chain security: a systematic literature review," International Journal of Computers and Applications, vol. 46, no. 10, pp. 853–867, 2024. DOI: 10.1080/1206212X.2024.2390978
[6] D. Anjaria and M. Kulkarni, "Effective DevSecOps Implementation: A Systematic Literature Review," Cardiometry, no. 24, pp. 410–417, 2022. DOI: 10.18137/cardiometry.2022.24.410417
[7] P. Martínez-Moreno, J. A. Vergara-Camacho, V. A. Lueváno-Mondragón, and K. A. Jiménez-Martínez, "DevSecOps for Secure Software Development: A Systematic Literature Review of Practices, Benefits, and Adoption Barriers," Int. J. of Combinatorial Optimization Problems and Informatics, vol. 17, no. 4, 2026. DOI: 10.61467/2007.1558.2026.v17i4.1299
[8] OWASP Foundation, "OWASP Top 10 CI/CD Security Risks," OWASP Project, 2026. [Online]. Available: https://owasp.org/projects/top-10-cicd-security-risks
[9] OWASP Foundation, "A08:2025 Software and Data Integrity Failures," OWASP Top 10 Application Security Risks, 2025. [Online]. Available: https://top10.owasp.org/2025/A08_2025-Software_or_Data_Integrity_Failures/
[10] Kubernetes Authors, "Deployments and Workload Controllers," Kubernetes Documentation, 2026. [Online]. Available: https://kubernetes.io/docs/concepts/workloads/controllers/deployment/
[11] Kubernetes Authors, "Rollout Management and Historical Revisions via kubectl," Kubernetes Reference Guide, 2026. [Online]. Available: https://kubernetes.io/docs/reference/kubectl/generated/kubectl_rollout/
[12] Cloud Native Computing Foundation (CNCF), "Software Supply Chain Best Practices v2," CNCF TAG Security, 2024. [Online]. Available: https://tag-security.cncf.io/community/working-groups/supply-chain-security/supply-chain-security-paper-v2/
[13] OpenSSF, "Supply-chain Levels for Software Artifacts (SLSA) Specification v1.0," Open Source Security Foundation, 2024. [Online]. Available: https://slsa.dev/spec/v1.0/
[14] OWASP Foundation, "OWASP DefectDojo: Open Source Vulnerability Management and DevSecOps Orchestration," 2025. [Online]. Available: https://www.defectdojo.org
[15] Argo Project, "Argo CD: Declarative GitOps Continuous Delivery for Kubernetes," Linux Foundation, 2026. [Online]. Available: https://argo-cd.readthedocs.io
[16] M. Pearce, B. Ahmad, B. Tan, B. Dolan-Gavitt, and R. Karri, "Asleep at the keyboard? Assessing the security of GitHub Copilot's code contributions," in Proc. 43rd IEEE Symp. on Security and Privacy (S&P), 2022, pp. 754–768. DOI: 10.1109/SP46214.2022.9833571
[17] K. Greshake, S. Abdelnabi, S. Mishra, C. Endres, T. Holz, and M. Fritz, "Not What You've Signed Up For: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection," in Proc. 16th ACM Workshop on Artificial Intelligence and Security (AISec), 2023, pp. 79–90. DOI: 10.1145/3605764.3623985
[18] Aqua Security, "Trivy: Comprehensive Vulnerability and Misconfiguration Scanner for Containers and Filesystems," 2026. [Online]. Available: https://github.com/aquasecurity/trivy
[19] Semgrep Inc., "Semgrep: Lightweight Static Analysis for Many Languages," 2026. [Online]. Available: https://semgrep.dev
[20] Z. Rice, "Gitleaks: Protect and Discover Secrets in Git Repositories," 2025. [Online]. Available: https://github.com/gitleaks/gitleaks
[21] Anchore Inc., "Syft: CLI tool and library for generating a Software Bill of Materials from container images and filesystems," 2026. [Online]. Available: https://github.com/anchore/syft
[22] Anchore Inc., "Grype: An easy-to-integrate vulnerability scanner for container images and filesystems," 2026. [Online]. Available: https://github.com/anchore/grype
[23] Helm Authors, "The Package Manager for Kubernetes," CNCF Project Documentation, 2026. [Online]. Available: https://helm.sh/docs/