International Journal of Computer Techniques – IJCT Volume 11 Issue 5, September 2024
Open Access and Peer Review Journal ISSN 2394-2231 https://ijctjournal.org/
RESEARCH ARTICLE OPEN ACCESS

Shift-Left Security Practices in Kubernetes-Based DevOps Environments: Measuring Impact on Software Vulnerability Reduction

Pruthvi Raj Seknametla
Independent Researcher
Email: pruthviraj.seknametla@ieee.org, pruthviraj9369@gmail.com

----------**********----------

Abstract:

Container orchestration with Kubernetes has fundamentally changed how organizations deploy and manage software at scale. But orchestration complexity, when left unexamined from a security standpoint, creates an attack surface that grows proportionally with team velocity. This paper investigates how shifting security responsibilities to the earliest practical phases of the software development lifecycle -- commonly called shift-left security -- affects measurable vulnerability outcomes in organizations operating Kubernetes-based DevOps pipelines. Drawing on a structured study of nine mid-to-large technology organizations over a fourteen-month period (September 2022 to October 2023), we tracked vulnerability detection timing, remediation cost differentials, deployment failure rates, and mean time to remediation (MTTR) across teams that implemented shift-left practices against those following more traditional reactive models. Results indicate that teams with mature shift-left integration discovered 68% of critical and high-severity vulnerabilities before code reached staging environments, compared to 21% in control groups. Remediation costs dropped significantly when defects were caught earlier in the pipeline, and deployment rollback incidents decreased by an average of 44% across adopting teams. We also examine the tooling landscape -- static analysis, container image scanning, policy-as-code, and admission control -- and explain how each layer contributes to a measurable reduction in production-facing risk. The conclusions offer practical guidance for engineering leaders and security architects deciding where to focus investment.

Keywords — **Kubernetes, DevSecOps, shift-left security, container security, vulnerability management, CI/CD pipelines, policy-as-code, SAST, image scanning.**

----------**********----------

I. INTRODUCTION

There is a point in the lifecycle of most engineering organizations where the gap between how fast they ship software and how carefully they secure it starts to become uncomfortable. For a while, that gap can be papered over -- a penetration test here, a quick scan before release there. But Kubernetes changed the stakes. When you can spin up hundreds of pods in minutes and orchestrate microservices across multi-cloud environments, traditional bolt-on security practices start to look less like guardrails and more like suggestions.

The shift-left idea is not new. Its roots go back to Barry Boehm's 1981 work on software economics, where he documented that defect correction costs increase exponentially the later a bug is found in development. What changed is the tooling. Static

analysis tools that used to require days of manual triage now run in seconds inside a CI pipeline. Container scanning tools can compare image layers against comprehensive CVE databases before a developer's pull request is merged. Open Policy Agent and Kubernetes admission webhooks let teams encode security requirements directly into the cluster's own control plane. The environment for shift-left security matured substantially between 2019 and 2023, and the question shifted from 'can we do this?' to 'what difference does it actually make?'

This paper attempts to answer that second question with some rigor. We structured a study involving nine organizations -- all running production Kubernetes workloads -- and tracked a defined set of vulnerability and operational metrics across teams at different points of shift-left maturity. The goal was not to produce a vendor comparison or argue for a specific toolchain. The goal was to understand whether moving security earlier in the pipeline produces statistically meaningful reductions in the kinds of vulnerabilities that cause real damage.

Section 2 describes how the study was structured, what data we collected, and how we categorized shift-left maturity. Section 3 covers the analytical models we applied. Section 4 presents the results and the practical implications we drew from them. Section 5 offers conclusions and directions for further research.

A. 1.1 Problem Context

In traditional software development, security reviews happened downstream -- usually as part of QA, or worse, as a condition of release sign-off. Developers wrote code, operations teams deployed it, and a separate security function reviewed results after the fact. In that model, vulnerabilities discovered in production often required emergency patches, sometimes with significant business impact before the fix landed.

Kubernetes-based architectures have accelerated this problem in several dimensions. First, the attack surface is larger and more dynamic. A deployment might involve dozens of container images, each with its own base layer, dependencies, and configuration footprint. Second, misconfiguration has emerged as a dominant vulnerability class -- RBAC policies that are too permissive, pods running as root when they should not be, sensitive environment variables exposed through poorly scoped ConfigMaps. Third, the velocity of change is higher. Teams deploying multiple times per day in a Kubernetes environment generate far more potential vulnerability exposure events per unit time than teams shipping quarterly releases.

Shift-left security attempts to compress the feedback loop by moving detection and prevention as close to the developer's local environment as possible. The underlying logic is straightforward: a vulnerability found during code review costs almost nothing to fix. The same vulnerability found after it has been deployed to production, exploited by an attacker, and escalated into an incident cost several orders of magnitude more -- in engineering time, in potential regulatory consequences, and sometimes in direct business damage.

B. 1.2 Scope and Limitations

This study focused exclusively on organizations running Kubernetes as their primary container orchestration platform. Organizations with purely virtual machine-based deployments, or those using managed container services without Kubernetes as the underlying layer, were excluded. All participating organizations had active CI/CD pipelines -- GitHub Actions, GitLab CI, Jenkins, or CircleCI were the most common -- and had been operating their Kubernetes environments for at least eighteen months prior to the study period.

We deliberately avoided measuring business-level outcomes like breach frequency or financial losses, partly because those events are rare and do not generate statistically useful sample sizes within a fourteen-month window, and partly because the causal chain from security practice to business outcome involves too many confounding variables to attribute cleanly. Instead, we focused on engineering-level metrics: where vulnerabilities appear, when they are detected, how long they take to fix, and how often security failures contribute to deployment disruptions.

II. METHODOLOGY

Designing this kind of study requires some care around how you define 'shift-left maturity.' The term gets used loosely in the industry -- some teams call themselves shift-left because they added a dependency scanner to their build script, while others have overhauled their entire development culture to treat security as a shared engineering responsibility. For the study to produce meaningful results, we needed a consistent classification model.

C. 2.1 Participant Selection

We recruited nine organizations through a combination of professional networks, conference outreach, and referrals from engineering consultancies. Participation was voluntary and organizations were not compensated, though they were offered the aggregate findings under a data-sharing agreement. All nine consented to collection of anonymized pipeline telemetry and vulnerability data. Five of the nine had deployed some form of shift-left tooling before the study period began. The remaining four had minimal or no dedicated security integration in their pipelines when data collection started.

Organization sizes ranged from 85 to approximately 1,400 engineers. Sectors represented included financial services, healthcare technology, e-commerce, and SaaS-based enterprise software. All nine were operating in cloud-hosted environments -- predominantly AWS and GCP -- with Kubernetes cluster counts ranging from two (dev/prod) to fourteen across regions and environments.

D. 2.2 Maturity Classification Model

We adapted a four-tier model to classify shift-left maturity across participating teams:

Tier 0 -- Reactive: No deliberate security integration in CI/CD. Vulnerabilities are identified primarily through manual reviews, external penetration tests, or post-incident analysis. No automated scanning of container images or dependencies. Security team operates as a separate function consulted infrequently before production releases.

Tier 1 -- Basic Scanning: Automated dependency and container image scanning tools exist in the pipeline, but findings are treated as informational. There is no enforcement layer -- a critical CVE in a base image will generate an alert but will not block a deployment. Security findings are tracked in a separate backlog and addressed on an irregular schedule.

Tier 2 -- Enforced Gates: Scanning tools are present and their findings trigger real pipeline outcomes. Critical vulnerabilities block merges or deployments. IaC configuration is reviewed before cluster provisioning, and RBAC policies are reviewed before deployment. Developers receive security feedback within the same workflow where they write code, often via pull request annotations or pre-commit hooks.

Tier 3 -- Integrated and Measured: All of Tier 2, plus security metrics are tracked alongside engineering metrics and reviewed in the same cadence. Admission controllers enforce policy at the cluster level. Developers have access to pre-approved, hardened base images. Security champions exist within development teams. Vulnerability backlogs have defined SLAs tied to severity. Post-incident reviews routinely examine whether earlier detection would have been possible.

E. 2.3 Data Collection

Over the fourteen-month study period, we collected the following categories of data from each organization:

Vulnerability detection point: Where in the pipeline (local dev, PR review, CI build, staging, production) was each vulnerability first identified?

Severity distribution: CVSSv3 scores were used to categorize findings as Critical (9.0-10.0), High (7.0-8.9), Medium (4.0-6.9), or Low (0.1-3.9).

Time-to-remediation: Days from detection to verified fix deployment, segmented by severity.

Deployment failure rate: Percentage of deployments that required rollback or hotfix within 24 hours, where post-deployment analysis identified a security-related contributing factor.

Remediation cost proxy: Engineering hours spent per vulnerability remediation, categorized by detection stage (using each organization's own time-tracking data).

Data was collected via API integrations with CI/CD platforms, vulnerability management tools (Snyk, Trivy, and Grype were the most common), and Kubernetes audit logs. Some organizations

supplemented automated data with quarterly interviews covering team process and cultural factors.

Table 1. Data collection methodology by metric category.

Metric	Collection Method	Frequency
Vulnerability detection point	CI/CD pipeline API + scanner logs	Per pipeline run
Severity distribution (CVSSv3)	Scanner output (Snyk, Trivy, Grype)	Per pipeline run
Time-to-remediation (days)	Ticket system API (Jira, Linear)	Weekly aggregation

F. 2.4 Control and Treatment Group Definition

For the purpose of comparative analysis, we designated the four organizations with Tier 0 or Tier 1 maturity at the study's outset as the control group, and the five organizations at Tier 2 or Tier 3 as the treatment group. Two of the control-group organizations implemented meaningful shift-left upgrades during the study period -- one moving from Tier 1 to Tier 2 in month six, and another reaching Tier 3 by month eleven. We retained these organizations in the control group for baseline comparison through their transition point, then tracked them separately in a longitudinal analysis to capture the before/after delta.

III. MODELING AND ANALYSIS

G. 3.1 Vulnerability Detection Distribution Model

To understand how shift-left maturity affected where vulnerabilities were caught, we modeled detection stage as a categorical outcome variable across five pipeline positions: (1) local pre-commit, (2) CI build / pull request, (3) staging environment, (4) production post-deployment, and (5) external discovery (penetration testing or incident response). For each organization and each quarter, we calculated the percentage of total vulnerabilities detected at each stage, then compared distributions across maturity tiers.

The underlying assumption is that earlier detection is better -- both economically and in terms of risk reduction. This assumption is well-supported by cost-of-quality literature and by our own remediation cost data, which we examine in Section 3.3. Moving detection left does not necessarily reduce the total number of vulnerabilities that exist in a codebase, but it dramatically changes when and where they are addressed.

H. 3.2 Remediation Time Analysis

Mean time to remediation (MTTR) was calculated per severity tier for both the control and treatment groups. We used a log-normal distribution model for MTTR because remediation times are right-skewed -- the majority of fixes happen within a few days, but a long tail of complex or deprioritized vulnerabilities can remain open for weeks or months. Log-transforming the data produced better model fit for interval estimation.

We also segmented MTTR by detection stage. A high-severity vulnerability found during a code review has a fundamentally different remediation trajectory than the same vulnerability found in production -- the production case involves triaging an active risk, potentially implementing a temporary mitigation, coordinating with operations, and then delivering a permanent fix. The staging case involves a developer fixing it before it ever reaches production infrastructure. These are not comparable remediation paths, and modeling them separately produced more useful results.

I. 3.3 Cost Model

Remediation cost was modeled using a stage-weighted multiplier approach, consistent with prior software quality economics research. We established a baseline cost of 1.0 for vulnerabilities caught at the CI/PR stage, which represents the most common entry point for shift-left tooling. Costs at other stages were estimated relative to this baseline using each organization's own engineering-hour data.

The multipliers we derived from the data are directionally consistent with published estimates but reflect our specific participant pool:

Table 2. Stage-weighted cost multipliers derived from participant data (CI/PR stage = 1.0x baseline).

Detection Stage	Relative Remediation Cost	Notes
Local pre-commit / IDE	0.6x	Developer fixes before review; lowest total cost
CI build / pull request	1.0x (baseline)	Primary target for shift-left gates
Staging environment	3.2x	Requires environment rebuild;

		wider team impact
Production (post-deploy)	8.7x	May involve mitigation, rollback, stakeholder comms
External discovery / incident	24.1x	Highest cost; may involve regulatory notification

These multipliers are averages and carry significant variance. A simple dependency upgrade caught pre-commit is genuinely cheap. A subtle authentication bypass found in production that requires a complex database migration to fix is not well-captured by an 8.7x multiplier -- it could be far worse. The model is useful for aggregate cost comparisons, not individual vulnerability costing.

J. 3.4 Deployment Failure Attribution

Attributing deployment failures to security root causes is methodologically tricky. Organizations vary in how thoroughly they conduct post-deployment retrospectives, and some security-related failures -- particularly misconfiguration-driven instability -- may be labeled as infrastructure issues rather than security events. To manage this, we applied a two-pass attribution approach: first, we accepted each organization's own classification; second, two independent analysts reviewed the incident descriptions for the subset of failures where classification was ambiguous, applying a shared rubric to determine whether a security defect was a primary or contributing cause.

Interrater agreement on the ambiguous cases was 87%, which we considered acceptable for the purposes of this study. Disagreements were resolved through discussion and a final consensus classification.

IV. RESULTS AND DISCUSSION

K. 4.1 Vulnerability Detection Stage Distribution

The most striking finding from the study was how dramatically shift-left maturity affected where vulnerabilities were detected. In the control group (Tier 0 and Tier 1 organizations), only 21% of critical and high-severity vulnerabilities were identified before code reached a staging or production environment. The distribution was heavily weighted toward staging and production discovery, with a non-trivial share of critical vulnerabilities -- roughly 9% of the control group total -- surfacing through external discovery or security incidents.

In the treatment group, the picture looked quite different. Organizations at Tier 2 detected 54% of critical and high-severity findings before staging. Tier 3 organizations pushed this to 68%. Equally significant, external or incident-driven discovery dropped to under 2% in Tier 3 organizations -- meaning the kinds of vulnerabilities that generate security incidents were overwhelmingly being caught inside the development pipeline before they ever reached production-facing infrastructure.

Table 3. Vulnerability detection stage distribution by maturity tier (Critical + High severity, averaged across study period).

Maturity Tier	Detected Pre-Staging (%)	Detected in Production (%)	External Discovery (%)
Tier 0 (Reactive)	14%	51%	12%
Tier 1 (Basic Scanning)	27%	44%	8%
Tier 2 (Enforced Gates)	54%	28%	4%
Tier 3 (Integrated & Measured)	68%	18%	2%

One nuance worth noting: the Tier 1 organizations sometimes had more total vulnerabilities logged than Tier 0 organizations, despite having some tooling in place. This is counterintuitive at first glance, but it reflects a common pattern where adding basic scanning without enforcement creates visibility into a pre-existing vulnerability backlog that was previously invisible. The scanning did not create the vulnerabilities -- it exposed them. This 'discovery surge' effect is worth anticipating when implementing initial scanning tooling, because it can generate short-term alarm before the longer-term risk reduction becomes evident.

L. 4.2 Mean Time to Remediation

MTTR results were consistent with the detection stage findings. Across all severity levels, organizations with higher shift-left maturity resolved vulnerabilities faster. But the magnitude of the difference varied by severity in ways that are practically important.

For critical vulnerabilities, Tier 3 organizations averaged 4.2 days to remediation, compared to 18.6

days in Tier 0 organizations. The most obvious explanation is that Tier 3 organizations caught critical issues earlier in the pipeline, where fixes are structurally simpler -- update a base image, bump a dependency version, adjust a policy file. Tier 0 organizations catching the same issues in production faced more complex remediation paths, including potential coordination with incident response teams and production change management processes.

Table 4. Mean time to remediation (MTTR) in days by severity level and maturity tier (averaged across study period).

Severity	Tier 0 MTTR (days)	Tier 1 MTTR (days)	Tier 2 MTTR (days)	Tier 3 MTTR (days)
Critical (9.0-10.0)	18.6	13.4	7.1	4.2
High (7.0-8.9)	24.3	19.7	11.8	8.6
Medium (4.0-6.9)	41.2	38.6	29.4	22.1
Low (0.1-3.9)	76.4	71.3	58.7	44.9

The medium and low severity numbers deserve some comment. Even in Tier 3 organizations, medium-severity vulnerabilities took over three weeks to fix on average. Low-severity findings took over six weeks. This is not a failure of the shift-left approach -- it reflects deliberate prioritization. Engineers and security teams have finite bandwidth, and a reasonable risk-based framework will always deprioritize a CVSS 5.4 finding in favor of a CVSS 9.8 finding. The MTTR data for low and medium severity items tracks more closely with team capacity and organizational risk tolerance than with pipeline security maturity.

M. 4.3 Remediation Cost Impact

Using the cost multiplier model from Section 3.3, we estimated aggregate remediation costs across the study period for each maturity tier. The differences were substantial. Tier 0 organizations, with their heavy weighting toward staging and production detection, incurred estimated remediation costs approximately 3.8 times higher per vulnerability than Tier 3 organizations when normalized for team size and vulnerability count.

The two organizations that transitioned maturity tiers mid-study provided the clearest evidence of cost impact, because we could compare before and after within the same organization, controlling for team size and workload changes. The organization that moved from Tier 1 to Tier 2 in month six showed a 34% reduction in estimated per-vulnerability remediation cost in the six months following the transition, compared to the six months before. The organization that reached Tier 3 by month eleven showed a 51% reduction comparing quarters on either side of the transition.

These are meaningful numbers. For an organization processing several hundred vulnerability findings per quarter across a moderately sized Kubernetes estate, a 34-51% reduction in remediation cost per finding translates to real engineering capacity that can be redirected toward product work.

N. 4.4 Deployment Failure Rate

Security-attributed deployment failures dropped significantly across increasing maturity tiers. Tier 0 organizations recorded an average of 11.3% of deployments requiring rollback or emergency hotfix with a security root cause or contributing factor. Tier 3 organizations averaged 4.2%. The absolute numbers varied considerably by organization size and deployment frequency, but the directional relationship held consistently.

Interestingly, the biggest reduction in deployment failures occurred between Tier 1 and Tier 2, rather than between Tier 0 and Tier 1. This aligns with the distinction between enforcement and observation. Tier 1 organizations had scanning but without enforcement -- critical CVEs in images might be flagged but still deployed. Tier 2 organizations had gates that actually blocked problematic deployments. The step from Tier 1 to Tier 2 added blocking logic to the feedback loop, and the failure rate improvement reflects that.

Table 5. Security-attributed deployment failure rates and quarter-over-quarter improvement trends.

Maturity Tier	Avg. Security-Attributed Failure Rate	QoQ Improvement (Study Period)
Tier 0 (Reactive)	11.3%	Negligible (-0.4%)
Tier 1 (Basic Scanning)	9.1%	Minimal (-1.2%)
Tier 2 (Enforced Gates)	6.3%	Moderate (-2.1%)
Tier 3 (Integrated & Measured)	4.2%	Sustained (-1.8% avg.)

O. 4.5 Tooling Effectiveness Observations

While this study was not designed as a tooling comparison, the participant data did reveal some consistent patterns worth noting. Static application security testing (SAST) tools integrated into pull request workflows showed the highest ratio of actionable findings to total alerts -- meaning developers were more likely to act on SAST findings than on some other alert types. Container image scanning tools generated high alert volumes but also had high false-positive rates when using default configurations, leading to alert fatigue in organizations that had not tuned their scanning policies.

Organizations using policy-as-code frameworks -- particularly Open Policy Agent with Rego policies encoded as Kubernetes admission webhooks -- reported that misconfiguration-class vulnerabilities declined fastest after implementation. This makes sense: admission control prevents non-compliant resources from being created in the cluster, regardless of what is in the container image or application code. A pod requesting host network access that violates policy simply cannot be scheduled.

The Tier 3 organizations generally combined three or four of these tooling layers rather than relying on any single approach. The layering matters because each tool catches a different class of problem. SAST catches logic-level issues in application code. Dependency scanning catches known vulnerable libraries. Image scanning catches issues at the OS and package layer. Admission control catches cluster-level misconfigurations. Organizations that only implemented one of these layers had noticeably higher rates of vulnerability class leakage -- problems that would have been caught by a missing layer still made it to production.

P. 4.6 Cultural and Organizational Factors

Tooling alone does not explain the full difference between maturity tiers. The Tier 3 organizations shared several cultural characteristics that Tier 0 and Tier 1 organizations consistently lacked. First, security was treated as a first-class engineering concern rather than a compliance function. Developers in these teams described security review as part of 'how we build things,' not as a gate imposed on them by a separate team. Second, all Tier 3 organizations had embedded security champions -- engineers with dedicated time for security advocacy within development teams, distinct from a centralized security team. Third, security metrics were reviewed in the same forum as reliability and performance metrics, meaning there was organizational visibility into the vulnerability state of the product rather than security being a quarterly report buried in a compliance deck.

These cultural factors are harder to measure than CVE counts, but they appear to be enabling conditions for tooling effectiveness. Several Tier 2 organizations had comparable tooling to Tier 3 organizations but lacked the cultural infrastructure to maintain and tune those tools over time, leading to metric decay -- improvements in the first quarter after implementation that gradually eroded as alert fatigue set in and configuration fell out of date.

V. CONCLUSION

The data from this study supports a straightforward claim: shifting security responsibilities earlier in the Kubernetes DevOps pipeline produces measurable, meaningful reductions in vulnerability exposure, remediation cost, and deployment disruption. The improvements are not marginal -- organizations at Tier 3 maturity detected vulnerabilities before they reached production at more than triple the rate of Tier 0 organizations, and their remediation costs per finding were substantially lower.

But the study also highlights a few things that are easy to miss in conversations about shift-left security. First, the transition from observation to enforcement is pivotal. Simply adding scanning tools without creating pipeline gates that act on their findings produces only modest improvement. Organizations looking for meaningful risk reduction need to be willing to block builds and deployments on critical findings, which requires upfront investment in reducing false positives and tuning policies so that enforcement does not become a productivity obstacle.

Second, layering matters. Container image scanning, SAST, policy-as-code, and admission control catch different problem types. An

organization that implements one layer and considers shift-left 'done' will still experience vulnerability class leakage from the other layers. Mature security posture in a Kubernetes environment requires multiple complementary controls operating in concert.

Third, cultural adoption is not separable from tooling adoption. The organizations that sustained improvement over the fourteen-month study period were those where developers had internalized security as part of engineering quality, not those where security was externally imposed through escalating friction. Building that culture takes time and intentional organizational design -- security champions, visible metrics, and genuine leadership commitment are not afterthoughts.

For engineering leaders considering investment in shift-left security, the case here is strong. The cost of implementing mature pipeline security is real but bounded. The cost of reactive security in a Kubernetes environment -- measured in incident response, production remediation cycles, and the accumulated technical debt of a growing vulnerability backlog -- compounds over time and does not get cheaper to address.

Future research should examine how AI-assisted code review tools interact with traditional SAST in shift-left pipelines, whether specific Kubernetes admission controller policy libraries produce better compliance outcomes than custom-written policies, and how shift-left maturity correlates with regulatory audit outcomes in industries with formal security compliance requirements. There is also a gap in longitudinal research beyond fourteen months -- it would be valuable to understand whether Tier 3 organizations sustain their improvement trajectory or whether plateau effects emerge as the most tractable vulnerability classes are eliminated and harder problems remain.

I. REFERENCES

[1] Boehm, B. W. (1981). Software Engineering Economics. Prentice-Hall, Englewood Cliffs, NJ.
[2] National Institute of Standards and Technology. (2022). Secure Software Development Framework (SSDF) Version 1.1: Recommendations for Mitigating the Risk of Software Vulnerabilities (NIST SP 800-218). U.S. Department of Commerce.
[3] Burns, B., Grant, B., Oppenheimer, D., Brewer, E., & Wilkes, J. (2016). Borg, Omega, and Kubernetes. ACM Queue, 14(1), 70-93.
[4] Open Policy Agent Project. (2023). OPA Documentation: Kubernetes Admission Control. https://www.openpolicyagent.org/docs/latest/kubernetes-introduction/
[5] Shostack, A. (2014). Threat Modeling: Designing for Security. Wiley, Indianapolis, IN.
[6] Kim, G., Humble, J., Debois, P., & Willis, J. (2016). The DevOps Handbook. IT Revolution Press, Portland, OR.
[7] Aqua Security. (2023). Cloud Native Security Report 2023. Aqua Security Research Team.
[8] Cloud Native Computing Foundation. (2023). CNCF Annual Survey 2023. Linux Foundation.
[9] Fitzgerald, B., & Stol, K.-J. (2017). Continuous software engineering: A roadmap and agenda. Journal of Systems and Software, 123, 176-189.
[10] MITRE Corporation. (2023). Common Vulnerabilities and Exposures (CVE) Program: 2023 Annual Metrics Report. MITRE Corporation.
[11] Humble, J., & Farley, D. (2010). Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Addison-Wesley, Upper Saddle River, NJ.
[12] Peltier, T. R. (2016). Information Security Policies, Procedures, and Standards: Guidelines for Effective Information Security Management (3rd ed.). CRC Press, Boca Raton, FL.
[13] Sysdig. (2023). 2023 Cloud-Native Security and Usage Report. Sysdig, Inc.
[14] National Vulnerability Database. (2024). NVD CVSS v3 Scoring Documentation. National Institute of Standards and Technology. https://nvd.nist.gov/vuln-metrics/cvss
[15] Forsgren, N., Humble, J., & Kim, G. (2018). Accelerate: The Science of Lean Software and DevOps. IT Revolution Press, Portland, OR.