International Journal of Computer Science Engineering Techniques **(IJCSE)** Volume 8 Issue 1, January 2024
ISSN : 2455-135X
https://www.ijcsejournal.org/ ***Research Article***

Policy-as-Code for DevSecOps: Automating Compliance and Security Enforcement in CI/CD Workflows

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

Abstract *Policy-as-Code (PaC) has emerged as a critical discipline within the DevSecOps ecosystem, offering engineering teams a programmable, version-controlled mechanism for expressing, enforcing, and auditing security and compliance rules directly within continuous integration and continuous delivery pipelines. Rather than treating compliance as a checkpoint imposed by a separate function at the end of a release cycle, PaC embeds policy logic into the same development toolchain that produces software making violations visible and actionable before code or configuration ever reaches production infrastructure. This paper examines the adoption, implementation, and measured impact of Policy-as-Code frameworks across eleven engineering organizations observed over a sixteen-month study period from June 2022 to September 2023. We analyze how different PaC approaches spanning Open Policy Agent with Rego, Conftest-based pipeline gates, Sentinel in infrastructure-as-code workflows, and Kubernetes admission webhooks affect compliance pass rates, audit preparation effort, security gate bypass incidents, and mean time to policy violation detection. Results demonstrate that organizations with mature PaC implementations achieved compliance verification pass rates of 91% at the pipeline stage, reduced audit preparation cycles by an average of 58%, and eliminated ad-hoc security gate bypasses almost entirely. We also examine the organizational and process factors that distinguish sustainable PaC programs from those that stall after initial deployment.*

Keywords *– CI/CD security, compliance automation, DevSecOps, infrastructure-as-code, Policy-as-Code*

1. Introduction

Ask any compliance engineer who has survived a SOC 2 audit what the worst part of the experience was, and most of them will say something about spreadsheets. Spreadsheets tracking which systems have which controls. Spreadsheets documenting which code repositories enforce which policies. Spreadsheets generated from memory, cross-referenced against screenshots, submitted to auditors who send them back requesting evidence in a different format. It is a deeply human process wrapped around an inherently mechanical question: does the system conform to the rules?

Policy-as-Code is, at its core, an attempt to answer that mechanical question with mechanical tools. If compliance rules can be expressed as code and most of them can then they can be version-controlled, reviewed, tested, automatically executed, and their results logged with the same rigor applied to production software. The audit evidence questions shifts from ‘can someone produce a spreadsheet?’ to ‘here is the pipeline output from the last six months.’

The practical case for this shift has grown substantially as organizations have moved toward infrastructure-as-code, containerized workloads, and cloud-native deployment models. When a team manages hundreds of Terraform configurations, Kubernetes manifests, Helm charts, and CI/CD pipeline definitions, manually reviewing each one for policy compliance is not feasible. The configurations change daily, the environments multiply, and the gap between ‘what policy say’s’ and ‘what is actually deployed’ widens over time unless enforcement is automated.

This paper investigates how eleven organizations implementing PaC across their CI/CD pipelines fared over a sixteen-month study period. We were particularly interested in three questions. First, does automated policy enforcement in the pipeline produce measurable compliance improvement compared to human-review-based approaches? Second, how does PaC implementation maturity affect audit-related engineering overhead? Third, what distinguishes organizations that build sustainable, evolving PaC programs from those that deploy initial tooling but fail to maintain or expand it?

1.1. The Compliance Gap in Modern Pipelines

Modern CI/CD pipelines process thousands of changes per day in large engineering organizations. Each change a code commit, a configuration update, a dependency bump, an infrastructure modification is a potential compliance event. Traditional compliance programs were designed for release cycles measured in quarters, not deployments measured in minutes. The mismatch creates what practitioners sometimes

call the compliance gap: the distance between the security and compliance posture an organization believes it has and the posture it actually operates with at any given moment.

Several forces widen this gap in practice. Configuration drift where deployed resources gradually diverge from their intended specification is nearly impossible to detect manually at scale. Role-based access control policies that are too permissive accumulate over time as engineers request access for specific tasks and that access is never revoked. Secrets management rules that look clean in a policy document get quietly bypassed under delivery pressure. Network policy configurations that were correct at initial deployment become outdated as new services are added.

PaC addresses the compliance gap by turning policies into executable artifacts that run continuously. A policy stating 'no container should run as root' does not live in a document or a wiki it lives as a Rego rule evaluated against every Kubernetes manifest before it is deployed. A policy stating 'all S3 buckets must have server-side encryption enabled' runs as a Conftest check against every Terraform plan before any infrastructure is provisioned. The gap narrows because enforcement is no longer dependent on human memory or manual review.

1.2. Scope and Research Context

This study focuses on PaC implementations in software delivery pipelines specifically, the enforcement of security and compliance policies at the stages of code review, infrastructure planning, and deployment admission control. We distinguish this from runtime policy enforcement (such as eBPF-based kernel-level controls or network policy enforcement by a service mesh), which operates after deployment and addresses a different threat model.

All eleven participating organizations operated cloud-hosted environments on AWS, GCP, or Azure, with active CI/CD pipelines processing multiple daily deployments. Seven of the eleven were operating containerized workloads on Kubernetes. Four were primarily deploying virtual machine-based or serverless architectures. All had some existing compliance obligation SOC 2, PCI-DSS, HIPAA, ISO 27001, or internal security baselines that motivated their interest in policy automation.

2. Methodology

2.1. Participant Selection and Onboarding

Organizations were recruited through a combination of practitioner network outreach and referrals from DevSecOps consultancies active in the study period. Participation was voluntary and uncompensated. All eleven organizations signed data-sharing agreements covering anonymized pipeline telemetry, policy evaluation logs, and audit metric data. Identifying information was removed from all datasets prior to analysis.

We required participating organizations to meet three baseline criteria: at minimum one active CI/CD pipeline processing multiple daily runs, documented compliance obligations under at least one recognized framework, and willingness to provide both automated telemetry and quarterly qualitative interviews with the engineers managing their pipeline security tooling.

Organizations ranged in engineering headcount from 60 to approximately 2,200. Industry sectors included financial services, healthcare technology, government contracting, cloud infrastructure services, and enterprise SaaS. The range of compliance frameworks was deliberately broad we wanted to understand whether PaC benefits were framework-specific or generalizable.

2.2. PaC Maturity Classification

Similar to prior work on DevSecOps maturity, we developed a four-tier classification model to group organizations by the depth and consistency of their PaC implementation:

Tier 0 - Manual Compliance: Policy review conducted manually, typically by a designated security or compliance engineer reviewing changes asynchronously. No automated policy checks in the CI/CD pipeline. Compliance evidence generated on demand for audits. Policy violations discovered primarily through periodic manual review or audit findings.

Tier 1 - Informational Automation: Automated policy checks exist in the pipeline, but results are advisory only. Violations are logged and visible in pipeline output but do not block merges or deployments. Policy rules may not be version-controlled alongside application code. Alert fatigue is common because findings have no enforcement consequence.

Tier 2 - Enforced Pipeline Gates: Policy checks run at defined pipeline stages and block progression on policy violations. Critical compliance rules are version-controlled as code and reviewed through pull requests. Policy libraries are maintained with at least quarterly reviews. Engineers receive clear violation feedback within the pipeline workflow rather than through separate reporting systems.

Tier 3 - Integrated Compliance-as-Code: All Tier 2 characteristics, plus policy code is tested with automated test suites. Policy libraries are continuously updated to reflect regulatory changes and security advisories. Compliance dashboards derive directly from pipeline telemetry rather than manual reporting. Audit evidence packages are generated automatically from policy evaluation logs. Security teams and development teams share ownership of policy repositories.

Table 1. PaC maturity tier distribution across the eleven participating organizations.

Tier	Description	Organizations (n=11)	Pipeline Enforcement
0	Manual Compliance	3	None
1	Informational Automation	2	Advisory only
2	Enforced Pipeline Gates	4	Blocking enforcement
3	Integrated Compliance-as-Code	2	Full integration + audit

2.3. Metrics and Data Collection

We collected data across six primary metric categories over the sixteen-month observation period:

Compliance pass rate at pipeline stage: The percentage of pipeline runs completing all policy checks without violations, measured separately for code-level checks, infrastructure plan checks, and deployment admission checks.

Policy violation detection point: Where in the delivery lifecycle compliance violation was first identified pipeline, staging review, production audit, or external audit finding.

Security gate bypass incidents: Documented cases where a policy enforcement gate was overridden or circumvented, including emergency exceptions.

Audit preparation engineering effort: Self-reported engineering hours spent preparing evidence packages for compliance audits, normalized per audit event.

Mean time to policy violation detection (MTTPVD): Time from when a non-compliant configuration was introduced to when it was identified by a policy check or review.

Policy library maintenance burden: Engineering hours per quarter spent maintaining, updating, and testing the organization's policy rule library.

Automated telemetry was collected via API integrations with CI/CD platforms (GitHub Actions, GitLab CI, Jenkins, and CircleCI were represented), OPA server logs, Conftest output captures, and Kubernetes audit logs where applicable. Qualitative data on audit effort and bypass incidents came from quarterly structured interviews with the engineers and security leads managing pipeline policy tooling.

2.4. Tooling Landscape Observed

Across the eleven organizations, four PaC tooling approaches were represented, sometimes in combination:

Open Policy Agent (OPA) with Rego: Used by seven organizations, primarily for Kubernetes admission control via Gatekeeper or custom webhooks, and for API authorization policy enforcement. Rego's general-purpose nature made it the most flexible option but also the most demanding from a learning curve perspective.

Conftest: Used by six organizations for policy evaluation of structured configuration files Terraform plans, Kubernetes manifests, Helm charts, Dockerfile linting within CI pipeline stages. Conftest uses OPA under the hood but presents a more accessible interface for pipeline integration.

HashiCorp Sentinel: Used by three organizations operating primarily on HashiCorp's infrastructure toolchain (Terraform Cloud or Terraform Enterprise). Sentinel's tight integration with Terraform plan output made it natural for infrastructure compliance gates, though its applicability outside the HashiCorp ecosystem is limited.

Cloud-native Policy Services: Two organizations used AWS Config Rules or Azure Policy in combination with pipeline-stage checks, primarily to catch drift between intended and deployed state in cloud infrastructure resources.

3. Modeling and Analysis

3.1. Compliance Pass Rate Trajectory

We modeled compliance pass rates as time-series data across the study period, calculating a rolling 30-day average for each organization and grouping by maturity tier. The core question was whether PaC maturity produced not just higher pass rates, but whether it produced sustained or improving pass rates because an organization that starts high and drifts downward as policy libraries go stale is not a success story.

Tier 0 and Tier 1 organizations showed relatively flat compliance pass rates over time, which is expected without enforcement, the metric is largely meaningless. For Tier 2 and Tier 3 organizations, we tracked pipeline-stage compliance pass rates at three layers: application code checks, infrastructure plan checks, and deployment admission control.

Table 2. Average compliance pass rates by check layer and maturity tier (16-month study period average).

Check Layer	Tier 0 (%)	Tier 1 (%)	Tier 2 (%)	Tier 3 (%)
Application code / SAST	N/A	71%	84%	92%
Infrastructure plan (IaC)	N/A	63%	88%	94%
Deployment admission control	N/A	N/A	91%	97%
Overall weighted average	~51%	~67%	~88%	~94%

The infrastructure plan layer showed the steepest improvement between Tier 1 and Tier 2. This aligns with the qualitative data from interviews: Tier 1 organizations with informational IaC checks often acknowledged violations in pipeline output and deployed anyway, particularly under time pressure. Once checks became blocking gates, the

remediation incentive was immediate, and pass rates rose significantly within the first two quarters of enforcement adoption.

3.2. Violation Detection Timing Model

Policy violation detection timing was modeled analogously to vulnerability detection stage analysis in security research earlier detection is economically and operationally preferable because the cost to remediate a violation increases with proximity to or entry into production environments.

Table 3. Policy violation detection stage distribution by maturity tier.

Detection Stage	Tier 0 (%)	Tier 1 (%)	Tier 2 (%)	Tier 3 (%)
Pipeline stage	0%	18%	71%	88%
Staging review	22%	34%	19%	9%
Production drift detection	41%	31%	8%	3%
External audit finding	37%	17%	2%	<1%

The external audit finding row tells the most important story. In Tier 0 organizations, 37% of identified compliance violations were discovered by auditors meaning those violations existed in production, undetected, until a third party found them. In Tier 3 organizations, that figure dropped to under 1%.

3.3. Audit Preparation Effort Analysis

Audit preparation is a notoriously labor-intensive process in organizations without PaC. Engineers and compliance staff spend weeks before each audit gathering configuration snapshots, generating access reports, cross-referencing deployment records against policy documentation, and formatting evidence packages to match auditor expectations.

Table 4. Audit preparation engineering effort by maturity tier (SOC 2 Type II equivalent scope).

Maturity Tier	Avg. Prep Hours	Evidence Auto-Generated (%)	Primary Effort Type
Tier 0	312 hrs	0%	Manual evidence gathering
Tier 1	248 hrs	12%	Alert triage + manual export
Tier 2	141 hrs	54%	Policy review + gap remediation
Tier 3	58 hrs	89%	Policy review + auditor Q&A

The Tier 3 figure of 58 hours per audit event represents an 81% reduction compared to Tier 0. The primary driver of effort reduction was automation of evidence collection. When policy evaluation logs are structured, timestamped, and stored in a query able format, generating an audit evidence package becomes a reporting exercise rather than an archaeological dig.

3.4. Security Gate Bypass Analysis

One concern with automated enforcement gates is that they create pressure for workarounds. If a critical policy check is blocking a deployment and a team is under delivery pressure, the path of least resistance is sometimes to disable or override the check. Tracking bypass incidents provides a reality check on whether enforcement is working as designed.

Table 5. Security gate bypass incident rates per organization per quarter, Tier 2 vs Tier 3.

Bypass Type	Tier 2 (per org/qtr)	Tier 3 (per org/qtr)	Trend
Authorized emergency exception	2.4	0.8	Declining
Unauthorized override	1.1	0.1	Near-eliminated in Tier 3
Policy suppression (unreviewed)	3.7	0.4	Sharply reduced
Policy suppression (reviewed)	1.2	2.1	Increased (formalized)

The unauthorized override figure is the most significant safety indicator. Tier 3 organizations averaged 0.1 unauthorized overrides per organization per quarter effectively zero for most of the study period. The increase in reviewed policy suppressions at Tier 3 is actually a positive signal, displacing unauthorized overrides with a formalized exception process.

3.5. Mean Time to Policy Violation Detection

For organizations with runtime compliance monitoring alongside pipeline checks, we calculated mean time to policy violation detection (MTTPVD). Tier 0 organizations had a median MTTPVD of 47 days for infrastructure-level violations and 112 days for access control violations. Tier 3 organizations using continuous compliance monitoring alongside pipeline gates achieved a median MTTPVD of 4.2 hours for infrastructure violations and 8.6 hours for access control changes.

4. Results and Discussion

4.1. The Policy Code Quality Problem

One pattern that emerged clearly from the study data is that the quality of policy code matters enormously, and bad policy code can be worse than no policy at all. The most common failure mode we observed in Tier 1 organizations attempting to move to Tier 2 was what the engineers themselves called ‘false positive poisoning.’ A policy rule that is too broadly written flags compliant configurations as violations. After a few weeks, the alert-to-action ratio gets bad enough that engineers stop looking at policy output at all. The enforcement gate becomes meaningless because nobody trusts its findings.

The Tier 3 organizations had all developed explicit policy code quality practices to address this. Policy rules were tested with automated test suits that included both compliant and non-compliant fixture cases. New policies were introduced initially in informational mode, with a defined observation period to measure the false positive rate before enforcement was enabled.

One of the Tier 3 organizations a financial services firm operating under PCI-DSS had an internal policy quality gate: no new enforcement rule could be enabled without evidence of at least three true-positive detections in staging and a false-positive rate below 5% in testing.

4.2. Rego Complexity and the Learning Curve

Seven of the eleven organizations used OPA with Rego as their primary policy language. Rego is powerful it can express complex, context-sensitive policies that simpler configuration-layer approaches cannot, but its learning curve is real and should not be understated in PaC adoption planning.

Rego uses a Datalog-inspired evaluation model that is unfamiliar to engineers trained primarily in procedural languages. Debugging Rego policy failures often requires running the OPA REPL with carefully crafted input documents and stepping through evaluation a process that is not hard once understood, but which creates friction for engineers encountering it for the first time under delivery pressure.

The organizations that adopted OPA successfully had generally done two things: invested in dedicated policy engineering time and provided structured onboarding for developers. Conftest, which layers an accessible CLI interface over OPA's evaluation engine, reduced the Rego exposure surface for most pipeline integration use cases.

4.3. Compliance Framework Portability

A common anxiety among organizations evaluating PaC is whether their policy library is specific to one compliance framework or whether it can serve multiple frameworks simultaneously. The Tier 3 organizations had developed policy library structures that addressed this through tagging and grouping. Rather than writing separate policy sets for each framework, they maintained a single policy library where each rule was tagged with the compliance requirements it satisfied.

This architecture requires upfront mapping work but eliminates the maintenance nightmare of separate policy libraries that inevitably diverge. When a new vulnerability class requires a new control, it is added once, tagged for all relevant frameworks, and enforced everywhere.

4.4. The Two Organizations That Did Not Sustain Progress

Of the eleven organizations, two showed initial improvements after implementing PaC tooling that subsequently degraded. Both had reached a functional Tier 2 state by month four of the study, with meaningful enforcement gates reducing compliance violations at the pipeline stage. By month ten, their metrics had regressed toward Tier 1 levels, despite the tooling still being technically present.

Post-study interviews with both organizations revealed similar root causes. In both cases, the initial PaC implementation had been driven by a small number of motivated engineers who championed the tooling and built the initial policy library. When those engineers moved to other roles or left the organization, no one owned ongoing policy maintenance. This pattern underscores that PaC requires an owner not just a tool deployment, but a person or team with defined responsibility for keeping the policy library current.

4.5. Return on Investment Estimation

While this study was not designed as a formal ROI analysis, the data collected allows a directional estimate of the value generated by PaC adoption at the Tier 3 level compared to Tier 0:

Audit preparation effort reduction: Approximately 254 engineering hours saved per audit event. At a fully loaded engineering cost of $120/hour, this represents roughly $30,500 per audit cycle.

Compliance violation remediation: Earlier detection dramatically reduces remediation cost. The shift from external audit discovery to pipeline stage detection reduces per-violation remediation cost by an estimated factor of 8 to 15 times.

Regulatory penalty avoidance: The reduction in auditor-discovered violations from 37% to under 1% materially reduces exposure to findings that could trigger regulatory scrutiny, remediation mandates, or penalties.

Engineering productivity: Organizations report reduced compliance-related interruptions for development teams after PaC adoption, as violations are caught and addressed within the development workflow rather than arriving as external demands during release cycles.

The cumulative ROI case is strong, particularly for organizations with frequent audit cycles. Initial implementation costs were reported by Tier 3 organizations at between 200 and 600 engineering hours, depending on organizational size and existing infrastructure automation maturity. This investment typically recovered within one to two audit cycles.

5. Conclusion

The study results make a clear case for Policy-as-Code as a foundational practice for organizations with meaningful compliance obligations operating modern software delivery pipelines. Compliance pass rates at the pipeline stage exceed 90% in mature implementations. Audit preparation effort drops by more than half. Auditor-discovered violations become rare events rather than routine audit findings.

But the study also reveals several things that are easy to miss when evaluating PaC from the outside. The transition from informational scanning to enforced blocking is the most important single step in the maturity journey. Organizations at Tier 1 frequently mistake tool presence for policy effectiveness they have the scanners, the dashboards, the alerts, but without enforcement gates the compliance benefit is marginal.

Policy code quality is not a secondary concern. False positive rates that seem tolerable in the first month become trust-destroying within a quarter. Investing in policy test suites, staged rollout with observation periods, and regular policy library reviews is not optional for sustainable PaC programs.

Ownership matters as much as tooling. The two organizations that regressed during the study period had equivalent tooling to the organizations that succeeded. What they lacked was sustained ownership of the policy library and the organizational structures that keep a policy program from decaying.

For organizations beginning their PaC journey, the tooling ecosystem is mature enough that technical risk is low. OPA, Conftest, and the surrounding community of policy libraries provide a solid foundation. The variable that separates successful implementations from stalled ones is almost always organizational: who owns the policies, how they are tested and updated, how bypassing incidents are handled, and whether compliance is treated as a shared engineering concern or an external imposition.

Future research directions include longitudinal study of PaC program health beyond two years, examination of PaC effectiveness in multi-cloud environments where policy portability across provider-specific tooling creates additional complexity, and deeper investigation of the relationship between policy library structure and regulatory audit outcome quality across different compliance frameworks.

6. Conflicts of Interest

The author declares that there is no conflict of interest regarding the publication of this paper.

7. Funding Statement

This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

8. Acknowledgments

The author thanks the eleven participating organizations and their engineering teams for providing access to pipeline telemetry, audit metrics, and interview time throughout the sixteen-month study period.

9. References

[1] J. Humble and D. Farley, Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Addison-Wesley Professional, 2010.

[2] Open Policy Agent Project, "OPA Documentation: Policy Language (Rego) Overview," 2024. [Online]. Available: https://www.openpolicyagent.org/docs/latest/policy-language/

[3] HashiCorp, "Sentinel Documentation: Policy as Code Framework," 2024. [Online]. Available: https://developer.hashicorp.com/sentinel

[4] Conftest Project, "Conftest: Write Tests Against Structured Configuration Data," 2024. [Online]. Available: https://www.conftest.dev/

[5] G. Kim, J. Humble, P. Debois, and J. Willis, The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations. IT Revolution Press, 2016.

[6] National Institute of Standards and Technology, "Security and Privacy Controls for Information Systems and Organizations," NIST SP 800-53 Rev. 5, 2020.

[7] Kubernetes Project, "Admission Controllers Reference," 2024. [Online]. Available: https://kubernetes.io/docs/reference/access-authn-authz/admission-controllers/

[8] Cloud Native Computing Foundation, "CNCF Policy Hub: Open Policy Agent Ecosystem," 2024. [Online]. Available: https://www.cncf.io/projects/open-policy-agent/

[9] N. Forsgren, J. Humble, and G. Kim, Accelerate: The Science of Lean Software and DevOps. IT Revolution Press, 2018.

[10] PCI Security Standards Council, "Payment Card Industry Data Security Standard (PCI-DSS) v4.0," 2022.

[11] L. Bass, I. Weber, and L. Zhu, DevOps: A Software Architect's Perspective. Addison-Wesley Professional, 2015.

[12] K. Morris, Infrastructure as Code: Managing Servers in the Cloud. O'Reilly Media, 2016.

[13] AICPA, "SOC 2 Trust Services Criteria," American Institute of Certified Public Accountants, 2022.

[14] S. Rose, O. Borchert, S. Mitchell, and S. Connelly, "Zero Trust Architecture," NIST SP 800-207, 2020.

[15] M. Souppaya, J. Morello, and K. Scarfone, "Application Container Security Guide," NIST SP 800-190, 2017.

[16] S. Garfinkel and S. Lipner, "Usable Security: Five Hard Problems," IEEE Security & Privacy, vol. 3, no. 5, pp. 33-38, 2005.

[17] M. Riley, "Compliance Automation at Scale: Patterns for Policy-as-Code in Enterprise DevSecOps," DevSecOps Days Conference Proceedings, Oct. 2023.