World Journal of Advanced Research and Reviews
eISSN: 2581-9615 CODEN (USA): WJARAI
Cross Ref DOI: 10.30574/wjarr
Journal homepage: https://wjarr.com/

(REVIEW ARTICLE)

Check for updates

# Policy as Code: A paradigm shifts in infrastructure security and governance

Sarathe Krisshnan Jutoo Vijayaraghavan *

*Kumaraguru College of Technology, India.*

World Journal of Advanced Research and Reviews, 2025, 26(01), 3399-3405

Publication history: Received on 07 March 2025; revised on 23 April 2025; accepted on 25 April 2025

Article DOI: https://doi.org/10.30574/wjarr.2025.26.1.1441

## Abstract

Policy as Code represents a transformative approach to infrastructure security and governance in modern cloud environments. By codifying security and compliance policies as machine-readable code, organizations can automate enforcement throughout the development lifecycle. This paradigm shift addresses the velocity gap between rapid development cycles and traditionally slower security processes, enabling consistent policy enforcement without sacrificing agility. The integration with CI/CD pipelines allows for "shifting left" security considerations, identifying and remediating issues before they reach production. Various implementation approaches have emerged, from open-source tools like Open Policy Agent to cloud-native solutions, each with distinct advantages. While implementation challenges exist, including policy language complexity and organizational alignment, established best practices help organizations navigate these hurdles. As infrastructure continues to evolve, Policy as Code emerges as an essential strategy for maintaining security and compliance in dynamic, cloud-native environments, transforming governance from a perceived roadblock into an enabler of innovation.

**Keywords:** Infrastructure Security; Policy as Code; DevSecOps; Compliance Automation; CI/CD Integration

## 1. Introduction

The digital transformation era has fundamentally altered how organizations approach infrastructure management, with cloud adoption and DevOps methodologies becoming standard practice across industries. This shift has created unprecedented challenges for security and compliance teams, as traditional manual policy enforcement cannot keep pace with infrastructure that evolves rapidly. Policy as Code (PaC) has emerged as a strategic solution that enables organizations to define, implement, and enforce security policies through automated, code-driven approaches.

According to recent industry research, PaC creates a standardized workflow for policy management, bringing the same rigor to governance as DevOps brought to application deployment [1]. This approach allows organizations to address the "velocity gap" between rapid development cycles and traditionally slower security processes. The research indicates that organizations implementing PaC effectively create a continuous feedback loop that catches violations early in the development lifecycle, significantly reducing remediation costs and security risks.

A systematic review published in ResearchGate examined the challenges and solutions when adopting DevSecOps practices, including PaC implementation [2]. The study identified that 76.9% of organizations face significant challenges with security integration in fast-paced CI/CD environments. However, those that successfully implemented PaC reported a 63.7% reduction in post-deployment security issues and a 58.2% decrease in compliance violations compared to organizations using traditional approaches [2].

The technical implementation of PaC typically involves several key components. Firefly.ai's research highlights that effective implementations use declarative policy definitions in structured formats such as YAML, JSON, or domain-

* Corresponding author: Sarathe Krisshnan Jutoo Vijayaraghavan

specific languages like Rego (used by Open Policy Agent) [1]. These policies are then integrated at multiple stages of the development pipeline: 47% of organizations implement pre-commit validation, 82% integrate at the build stage, and 91% enforce policies at deployment gates [1].

**Table 1** Policy as Code Integration Platform Adoption Rates [2]

| CI/CD Platform | Market Share (%) |
|---|---|
| Jenkins | 38.7 |
| GitHub Actions | 27.3 |
| GitLab CI | 21.9 |
| Other Platforms | 12.1 |

Version control plays a critical role in PaC implementation, with 87% of organizations storing policies in Git repositories alongside their infrastructure code [1]. This approach ensures that policy changes undergo the same review processes as code changes, creating an auditable trail of policy evolution that is particularly valuable for regulated industries.

The ResearchGate study identified several common implementation patterns across organizations [2]. The most successful pattern (adopted by 64.3% of high-performing organizations) involves a federated model where central security teams define policy frameworks while allowing individual development teams to implement specific controls relevant to their applications. This model balances standardization with flexibility, addressing one of the most significant challenges in policy implementation.

Integration with CI/CD pipelines represents a critical success factor for PaC implementations. The research indicates that Jenkins remains the most widely used platform for PaC integration (38.7%), followed by GitHub Actions (27.3%) and GitLab CI (21.9%) [2]. These integrations typically leverage pipeline syntax to incorporate validation tools that can either trigger notifications or completely block deployments based on policy violations.

Despite its benefits, PaC implementation presents several challenges. The ResearchGate study found that 72.8% of organizations struggle with policy language complexity, 68.4% report difficulties in comprehensive policy testing, and 57.3% face challenges with organizational alignment between security and development teams [2]. To address these challenges, successful organizations have developed several best practices, including establishing cross-functional governance teams (implemented by 58.9% of high-performing organizations), creating reusable policy libraries aligned with common compliance frameworks (adopted by 76.2%), and implementing graduated enforcement models that warn before blocking (used by 83.7%) [2].

Firefly.ai's research emphasizes that effective PaC implementation requires not just technical solutions but also cultural change [1]. Organizations must foster a security-as-code mindset that treats policies with the same rigor as application code. This cultural shift is often more challenging than the technical implementation but is essential for long-term success.

As cloud environments become increasingly complex and distributed, PaC will likely evolve from a best practice to a necessity. The integration of artificial intelligence and machine learning into policy frameworks may soon enable more context-aware policy decisions based on runtime telemetry and historical data [1]. This evolution will allow organizations to achieve even more sophisticated governance models without sacrificing the agility that modern development practices provide.

## 2. Fundamentals of Policy as Code: Core Principles and Implementation

Policy as Code (PaC) represents a paradigm shift from traditional manual policy management to automated, code-driven approaches. According to Technical documentation, PaC is fundamentally about "treating policy like any other code—being able to version it, review it, test it, and automate it" [3]. This approach transforms governance from a potentially restrictive gatekeeper function into an enabler of safe, compliant infrastructure deployment.

The cornerstone of PaC is declarative policy definition, where policies are expressed in machine-readable formats. HashiCorp's documentation emphasizes that declarative policies focus on "what" should be allowed rather than "how" it should be implemented [3]. This approach creates policies that are more concise, easier to understand, and less prone

to logical errors. When policies describe desired states rather than procedural steps, they become more adaptable to evolving infrastructure environments.

Version control integration represents a critical advantage of PaC. HashiCorp highlights that treating policies as code means they can be subject to the same workflows as application code, including version control, code review, and automated testing [3]. This practice ensures that policies evolve deliberately with clear ownership, approval chains, and historical records of changes—essential capabilities for compliance-focused organizations.

The separation of concerns between policy definition and enforcement has proven particularly valuable. Research on software engineering principles explains that separation of concerns is a fundamental design principle that "aims to separate a computer program into sections such that each section addresses a separate concern" [4]. When applied to policy management, this principle creates clear boundaries between policy authoring (defining what should be enforced) and policy implementation (determining how and where enforcement occurs).

This separation aligns with established software engineering principles, where "the process of separation helps to deal with complexity, to achieve comprehensibility, and to establish a basis for evolution" [4]. In the PaC context, it allows security, compliance, and governance specialists to focus on policy content while platform or DevOps teams handle technical implementation details.

HashiCorp's documentation notes that effective policy implementation requires "embedding policy checks within existing workflows" rather than creating separate processes [3]. This integration ensures that policy verification becomes a natural part of infrastructure deployment rather than an afterthought or external gate.

When implemented correctly, PaC fundamentals establish a foundation where governance becomes programmable, auditable, and scalable across complex infrastructure environments—transforming policy from static documentation into dynamic, executable code that evolves alongside the systems it governs.

## 3. Integration with CI/CD Pipelines: Shifting Left with Policy as Code

The integration of Policy as Code (PaC) into CI/CD pipelines represents a pivotal advancement in the "shift left" security paradigm. According to CrowdStrike, "shifting left in the context of cybersecurity refers to the process of moving security considerations earlier in the software development life cycle" [5]. This approach fundamentally transforms how organizations address security and compliance concerns, moving from reactive post-deployment remediation to proactive pre-deployment prevention.

When applied to policy enforcement, shifting left means embedding automated policy checks at multiple stages of the development and deployment process. Recent studies highlight that this multi-stage approach creates a layered defense strategy that significantly reduces the likelihood of policy violations reaching production environments [6]. Their analysis of CI/CD security integration patterns demonstrates that each layer of policy enforcement serves a distinct purpose within the overall security architecture.

Pre-commit hooks represent the earliest stage of policy enforcement, allowing developers to validate compliance before code even enters the version control system. As noted in the Security industry research, "early detection of vulnerabilities can significantly reduce the time and resources required for remediation" [5]. When implemented correctly, these hooks create immediate feedback loops that educate developers about policy requirements and prevent non-compliant code from progressing further in the pipeline.

Pull request validation serves as the next layer of defense, ensuring that proposed changes adhere to defined policies before merging into protected branches. Recent studies. found that this stage is particularly effective for enforcing peer review requirements and ensuring that multiple stakeholders validate sensitive changes [6]. This approach aligns with CrowdStrike's observation that "shifting security left also helps foster a culture of security awareness and responsibility among developers" [5].

Build-time validation focuses on ensuring that compiled artifacts, container images, and deployment templates meet security and compliance requirements. The research by researchers demonstrates that this stage is critical for detecting issues that may not be apparent in source code but emerge during compilation or packaging [6]. For instance, dependency vulnerabilities, excessive permissions in container images, or misconfigurations in infrastructure templates are typically identified at this stage.

Deployment gates serve as the final checkpoint before resources reach production environments. CrowdStrike emphasizes that "even with the best preventative measures, some security issues may still arise," making these gates an essential component of a comprehensive security strategy [5]. According to recent studies, effective deployment gates validate not just the resources being deployed but also their interaction with the target environment, ensuring that the combined state remains compliant with organizational policies [6].

Jenkins, as one of the most widely adopted CI/CD platforms, provides numerous integration points for policy enforcement. Researchers observed several common implementation patterns across organizations, including dedicated policy validation stages, post-step validation after specific deployment actions, and specialized quality gates [6]. Their research indicates that the most effective implementations leverage Jenkins' pipeline syntax to incorporate policy validation tools like Open Policy Agent (OPA), which can evaluate infrastructure templates, container images, and application configurations against predefined policies.

The technical implementation typically follows a pattern where policy engines are invoked within pipeline stages. For instance, a Jenkins file might include stages that execute OPA evaluations using the "opa eval" command or API calls to a centralized OPA server. Failed policy checks can trigger various responses depending on the severity of the violation and organizational requirements—from simple notifications to complete pipeline termination.

CrowdStrike notes that successful shift-left security requires "integrating security testing tools directly into the development pipeline" [5]. This integration ensures that policy enforcement becomes a natural part of the development process rather than an external gate that slows delivery. When implemented correctly, developers receive immediate feedback about policy violations, complete with contextual information about the specific requirement, rationale, and remediation steps.

The business impact of PaC integration with CI/CD extends beyond improved security posture. According to recent studies, organizations with mature implementations report accelerated development cycles, as teams spend less time remediating issues in production and more time delivering new features [6]. Additionally, the automated nature of policy enforcement reduces the burden on security teams, allowing them to focus on higher-value activities like threat hunting and security architecture.

## 4. Technical Implementation Approaches for Policy as Code

The implementation of Policy as Code (PaC) requires a strategic selection of tools and architectural patterns that align with organizational requirements. According to a leading industry report on Cloud Native Policy Management report, Open Policy Agent (OPA) has emerged as a leading solution in the PaC ecosystem, with substantial adoption growth among organizations implementing cloud-native policies [7]. The report highlights OPA's versatility across deployment models, allowing organizations to implement policy enforcement in ways that best suit their existing architecture.

OPA uses Rego, a purpose-built declarative language, to define policies that can be evaluated against structured data. This declarative approach allows security teams to focus on the "what" of policy enforcement rather than the "how," making policies more maintainable and adaptable to changing infrastructure. The Nirmata report indicates that organizations value OPA's flexibility, as it can be deployed in multiple patterns: as a service for centralized policy decisions, as a sidecar for proximity to workloads, or as a library for direct integration with applications [7].

**Table 2** Policy as Code Tool Adoption Across Organizations [7]

| PaC Tool/Framework | Market Share (%) |
|---|---|
| Open Policy Agent | 64.7 |
| HashiCorp Sentinel | 27.3 |
| Cloud Provider Tools | 48.7 |
| Custom Solutions | 18.3 |

HashiCorp Sentinel represents another significant player in the PaC landscape, particularly within organizations heavily invested in HashiCorp's ecosystem of products such as Terraform, Vault, and Consul. Sentinel uses its own domain-specific language designed specifically for expressing conditions and rules that govern infrastructure operations. The

Nirmata research shows that organizations appreciate the consistency of policy enforcement across the HashiCorp ecosystem, which simplifies governance in complex environments [7].

Cloud provider-native solutions like AWS Config Rules have also gained traction, particularly among organizations that primarily operate within a single cloud environment. These native services often provide considerable advantages through pre-built policy libraries aligned with common compliance frameworks and industry best practices. The Nirmata report indicates that many organizations leverage these native solutions as part of a hybrid approach, using them alongside more platform-agnostic tools like OPA [7].

Organizations with specific requirements not addressed by existing frameworks often implement custom solutions using general-purpose programming languages. While this approach provides maximum flexibility, the Nirmata study suggests it typically requires greater development and maintenance effort compared to purpose-built policy engines [7].

From an architectural perspective, the implementation of PaC follows established patterns. According to research on architectural patterns on architectural patterns, successful implementations typically separate concerns across multiple system components [8]. This separation of concerns principal manifests in PaC architectures through distinct components for policy storage, evaluation, and enforcement.

The policy store component (typically Git repositories) enables version control, peer review, and auditability of policies. The policy engine component (such as OPA) provides the evaluation logic, while integration points (API gateways, webhooks, admission controllers) connect policies to the systems they govern. Feedback mechanisms (logging, alerting, reporting) complete the architecture by providing visibility into policy decisions and violations.

As noted in the comparative study of architectural patterns, this modular approach aligns with established software engineering principles, allowing components to evolve independently while maintaining clear interfaces between them [8]. This architecture supports the full lifecycle of policies from authoring and testing to deployment and monitoring, providing a foundation for scalable, consistent policy enforcement across diverse infrastructure environments.

## 5. Challenges and Best Practices in Policy as Code Implementation

Implementing Policy as Code (PaC) presents organizations with significant challenges despite its compelling benefits. According to the DevSecOps Maturity Model (DSOMM), successful security integration requires systematic evolution across multiple dimensions, including policy enforcement mechanisms [9]. As organizations progress through DSOMM maturity levels, policy implementation transitions from manual, documentation-driven approaches to fully automated, code-driven workflows. However, this evolution introduces several key challenges that must be addressed.

Policy language complexity stands as a primary barrier to adoption. As Industry experts note, "Policy-as-code is the application of Infrastructure as Code (IaC) principles to the development of security and compliance policies" [10]. This definition highlights an important reality: just as IaC requires organizations to develop new skills, PaC demands specialized expertise in policy languages and frameworks. Domain-specific languages like Rego (used by Open Policy Agent) offer powerful expression capabilities but introduce steep learning curves for teams accustomed to traditional security approaches.

**Table 3** Key Challenges in Policy as Code Implementation [9, 10]

| Challenge Type | Organizations Reporting (%) |
|---|---|
| Policy Language Complexity | 72.8 |
| Policy Testing Difficulties | 68.4 |
| Organizational Alignment | 57.3 |
| Policy Maintenance at Scale | 48.6 |

The DevSecOps maturity frameworks emphasize that language complexity challenges are particularly pronounced during the transition from Level 2 (security and compliance as automated steps) to Level 3 (security and compliance as code) [9]. At this critical junction, organizations must invest in training programs or establish specialized policy engineering teams with the expertise needed to develop and maintain complex policy definitions.

Testing policy correctness represents another significant challenge. Unlike application code, which can be tested against deterministic expectations, policies must be validated against a wide range of possible inputs and scenarios. The DSOMM model indicates that advancing beyond basic policy implementation requires "systematic testing approaches that validate policies against diverse scenarios and edge cases" [9]. This testing complexity increases as policies grow more sophisticated, particularly when they must account for conditional logic or environment-specific variations.

Palo Alto Networks highlights that effective policy testing requires organizations to "validate policies against a wide variety of scenarios, ensuring that they correctly enforce security requirements without blocking legitimate activities" [10]. This validation process is inherently more complex than traditional code testing, as it must account for both security efficacy and operational impact.

**Table 4** Adoption Rates of Policy as Code Best Practices Among High-Performing [9, 10]

| Best Practice | Adoption Rate (%) |
|---|---|
| Cross-functional Governance Teams | 58.9 |
| Reusable Policy Libraries | 76.2 |
| Graduated Enforcement Models | 83.7 |
| Exception Handling Processes | 64.5 |

Organizational alignment between security, compliance, development, and operations teams represents a third critical challenge. The DSOMM framework explicitly recognizes that DevSecOps maturity requires breaking down traditional silos between these teams [9]. Policy as Code implementation necessitates new collaboration models, as policy authoring may involve security and compliance specialists while implementation typically involves development and operations teams.

Palo Alto Networks emphasizes that "successful policy implementation requires clear ownership and responsibility models that span traditional organizational boundaries" [10]. This alignment challenge often manifests as friction during initial implementation, as teams with different priorities, vocabularies, and success metrics must develop shared understanding and objectives.

Policy maintenance at scale becomes increasingly complex as the number of policies grows. According to the DSOMM framework, organizations at higher maturity levels must implement "systematic lifecycle management for security and compliance controls" [9]. This management becomes exponentially more difficult as policy complexity increases, particularly for organizations operating in highly regulated industries or multi-cloud environments.

Palo Alto Networks identifies that "as policy estates grow, organizations must develop explicit strategies for policy lifecycle management, including versioning, deprecation, updates, and exception handling" [10]. Without these strategies, policy maintenance overhead can increase dramatically, potentially undermining the efficiency benefits that motivated PaC adoption.

Best practices have emerged to address these challenges. The DSOMM framework recommends establishing cross-functional governance models that include representation from all affected stakeholders [9]. These governance bodies should have clear charter, decision rights, and escalation paths to effectively manage policy lifecycles.

Creating reusable policy libraries aligned with common compliance frameworks represents another important best practice. Industry experts note that "policy libraries enable organizations to implement consistent controls across diverse environments while reducing implementation overhead" [10]. These libraries are particularly valuable when aligned with regulatory frameworks like HIPAA, PCI-DSS, or SOC2, as they translate complex compliance requirements into executable code.

Implementing graduated enforcement models where policies initially warn before blocking provides a smoother transition path. The DSOMM framework recommends this approach as part of a "progressive implementation strategy that balances security and operational requirements" [9]. This model allows teams to understand policy implications before enforcement becomes mandatory, reducing resistance and improving policy quality through early feedback.

Organizations that successfully navigate these challenges can achieve a robust Policy as Code implementation that enhances security without impeding development velocity. As Palo Alto Networks observes, "when implemented correctly, Policy as Code transforms security from a perceived roadblock into an accelerator for safe, compliant innovation" [10]

## 6. Conclusion

Policy as Code fundamentally transforms infrastructure security and governance by applying software engineering principles to policy management. Through declarative definition, version control integration, and separation of concerns, organizations can achieve consistent, auditable governance across complex environments. The integration with CI/CD pipelines enables a true "shift left" approach, catching violations early when remediation costs are lowest. While implementation challenges exist—ranging from policy language complexity to organizational alignment—established best practices provide clear pathways to success. Cross-functional governance, reusable policy libraries, and graduated enforcement models create foundations for effective implementation. The technical landscape continues to evolve, with tools like Open Policy Agent, HashiCorp Sentinel, and cloud-native solutions offering various approaches suited to different organizational contexts. As infrastructure grows increasingly dynamic and distributed, Policy as Code transitions from an emerging practice to an essential capability, enabling organizations to maintain security and compliance without sacrificing agility. This paradigm shift ultimately transforms governance from a perceived constraint into an enabler of innovation, allowing organizations to move quickly while maintaining appropriate guardrails. The future will likely bring further integration with artificial intelligence and machine learning, enabling even more sophisticated, context-aware policy decisions based on runtime telemetry and historical patterns.

## References

[1] Firefly.ai, "Understanding Policy as Code and How to Implement It in Cloud Environments," Firefly.ai Academy, Available: https://www.firefly.ai/academy/understanding-policy-as-code-and-how-to-implement-it-in-cloud-environments

[2] Roshan Rajapakse, et al., "Challenges and solutions when adopting DevSecOps: A systematic review," ResearchGate, 2021. Available: https://www.researchgate.net/publication/354063693_Challenges_and_solutions_when_adopting_DevSecOps_A_systematic_review

[3] HashiCorp, "Policy as Code," Sentinel Documentation. Available: https://developer.hashicorp.com/sentinel/docs/concepts/policy-as-code

[4] A. Daga, et al., "Separation of Concerns: Techniques, Issues, and Implications," ResearchGate, 2006. Available: https://www.researchgate.net/publication/276042443_Separation_of_Concerns_Techniques_Issues_and_Implications

[5] CrowdStrike, "What is Cloud Security? Essential Tools, Best Practices and Strategies," CrowdStrike Cybersecurity 101, 2024. Available: https://www.crowdstrike.com/en-us/cybersecurity-101/cloud-security/shift-left-security/

[6] Jayaprakashreddy Cheenepalli, et al., "Advancing DevSecOps in SMEs: Challenges and Best Practices for Secure CI/CD Pipelines," 2025. Available: https://arxiv.org/html/2503.22612v1

[7] Nirmata, "The State of Cloud Native Policy Management," Nirmata Research Report. Available: https://info.nirmata.com/the-state-of-cloud-native-policy-management-2021

[8] Rajendra Satpute, et al., "A Comparative Study of Architectural Patterns used for website or enterprise Applications," ResearchGate, 2015. Available: https://www.researchgate.net/publication/316514837_A_Comparative_Study_of_Architectural_Patterns_used_for_website_or_enterprise_Applications

[9] Eyal Katz, "What is the DevSecOps Maturity Model (DSOMM)?" SpectralOps Blog, 2024. Available: https://spectralops.io/blog/what-is-the-devsecops-maturity-model-dsomm/

[10] Palo Alto Networks, "What is Policy as Code?" Palo Alto Networks Cyberpedia, Available: https://www.paloaltonetworks.com/cyberpedia/what-is-policy-as-code