Policy-as-Data for Self-Healing SaaS: A Kubernetes-Native Approach

***Amar Gurajapu, **Anurag Agarwal**
*Principal Member of Technical Staff, **Senior Software Engineer,
Network Systems, AT&T
Middletown, New Jersey, United States

DOI:10.37648/ijiest.v12i01.004

[1]*Received: 28 December 2025; Accepted: 18 January 2026; Resubmitted after correction: 25th January 2026; Published: 02 February 2026*

ABSTRACT

We propose a scalable, Kubernetes-native approach to enforce security, configuration, and regulatory policies in multi-tenant SaaS. Each policy is stored as a versioned Custom Resource (Policy CR) in a Git repo ("Policy-as-Data"), synchronized via a GitOps agent, validated on create/update through a mutating admission webhook, and reconciled continuously by a self-healing controller. This closed-loop design minimizes manual intervention, provides drift detection, and enables automated remediation across hundreds of namespaces with minimal overhead.

Keywords: *Policy-as-Data; GitOps; Kubernetes; Self-Healing; Admission Webhook; Custom Resource Definition (CRD); JSONPatch; SaaS*

1. Introduction

Today's SaaS offerings often span dozens or hundreds of Kubernetes namespaces, each with potentially hundreds of Pods. Ensuring every workload adheres to evolving security or compliance rules—such as "no root containers" or "mandatory resource limits"—is a significant operational burden:

- Manual policy checks are error-prone and slow.
- Static admission policies lack visibility into live drift.
- Custom operators frequently hard-code rules, making updates cumbersome.

We address these challenges by treating policies as data:

- Policy-as-Data - Policies are CRs in a Git repo, making them discoverable, versioned, and auditable.
- GitOps-Driven Sync - A GitOps agent (e.g. Argo CD) applies changes automatically[4]
- Admission-Time Enforcement - A mutating webhook inspects and rejects or patches new/updated Pods.
- Self-Healing Reconciliation - A controller loops periodically to detect and repair any drift in existing objects.

[1] How to cite the article: Gurajapu A., Agarwal A.; January 2026; Policy-as-Data for Self-Healing SaaS: A Kubernetes-Native Approach; *International Journal of Inventions in Engineering and Science Technology;* Vol 12 Issue 1, 28-33, DOI: http://doi.org/10.37648/*ijiest.v12i01.004*

This end-to-end loop delivers continuous compliance, integrates into existing CI/CD pipelines, and scales to large cluster footprints.

2. Materials and Methods

As Kubernetes adoption grows, enforcing cluster-wide policies consistently and at scale becomes critical. Though this is a known gap, it is often observed that DevSecOps teams rely on Manual audits, which might sometimes have serious consequences in terms of financial impact and credibility. Below we outline key pain points, survey existing approaches, and motivate our Policy-as-Data design.

A. Key Challenges

- Drift over Time - Resources may be created before a new policy is introduced. Manual fixes lag behind real-time changes, leaving windows of non-compliance.
- Scale & Multitenancy - Hundreds of namespaces and thousands of Pods demand low-latency checks. Centralized enforcement must avoid becoming a performance bottleneck.
- Evolving Rule Sets - Security and regulatory requirements change frequently. Hard-coded policy logic in controllers/operators requires rebuilds for each update.
- Audit & Traceability - Teams need a version history of policy changes. Rollback and audit trails are essential for compliance reporting.

B. Why Policy-as-a-data approach? [2]

- Decouple Rule Management - Policies live as simple YAML CRs in Git with no need for any recompilation when rules change.
- GitOps Integration - Leverage existing GitOps pipelines for automatic rollout, versioning, and audit.
- Continuous Self-Healing - Beyond admission-time checks, a controller reconciles drift, ensuring retroactive compliance.
- Lightweight & Kubernetes-Native - Builds on standard controller-runtime patterns and JSONPatch[5], avoiding heavyweight external engines.time until 50 % of new requests succeed post-failover

C. Example Use Cases

Few use cases which would justify the need emphasized in this paper.

- Security Context Defaults - Enforce `runAsNonRoot=true` on all Pods, even ones created before the policy existed.
- Mandatory Labels / Annotations - Auto-inject team or environment labels for chargeback and monitoring.
- Network Policy Enforcement - Block Pods that lack specific network segmentation rules.
- Resource Quotas - Ensure resource limits/requests meet organizational minimums.

D. Logical Components and Data Flow

Our design divides the loop into five collaborating components. Each plays a clear role, and together they form a horizontally scalable, self-healing enforcement system.

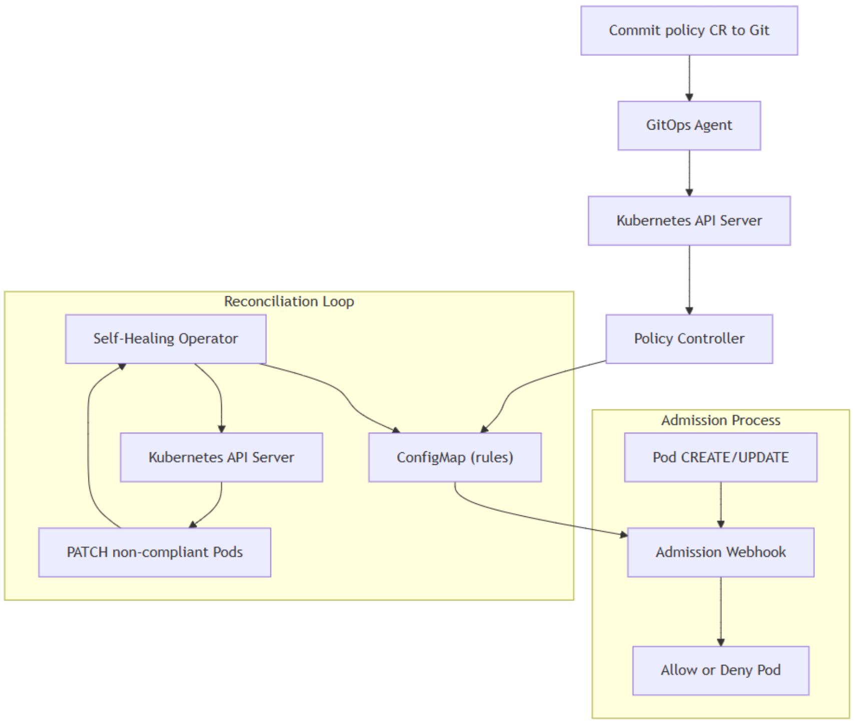

Figure 1. Data flow between modules

- Policy Git Repository - Stores one YAML file per rule (Policy CR). Branch protection ensures review before roll-out.
- GitOps Agent - Monitors the repo and applies CRs into the cluster via the Kubernetes API.
- Policy Controller - Watches for Policy CR events and aggregates active rules into a namespace ConfigMap for fast lookup.
- Admission Webhook - Intercepts Pod `CREATE` and `UPDATE` calls, reads the in-cluster ConfigMap, and allows, denies, or mutates the request.
- Self-Healing Operator - On a configurable interval (e.g. 30 sec), lists all workloads, applies JSONPatch from each policy, and issues patches to remediate any drift.

E. *Security and Resilience*

While it is important to handle the functional aspect, there is need for non-functional compliance with respect to securing data protection, high availability and disaster management.

- TLS and RBAC - Webhook server run on HTTPS with mounted certificates. RBAC roles restrict who can modify Policy CRs and ConfigMaps.
- High Availability - Deploy controller, operator with multiple replicas and leader election.
- Failure Modes - If the webhook or operator fails, policies revert to earlier commit (Git rollback) and health probes restart pods automatically.

F. *Performance and Scalability*

It is possible to choose any standard DNS mechanism used within the organization. These APIs are designed for automation, enabling seamless DevOps workflows and granular control over traffic distribution. For instance, you can rely on either Amazon Route 53 or IBM NS1 Connect (formerly NS1) which provide RESTful APIs that allow you to programmatically manage DNS records and adjust global traffic weights. compliance with respect to securing data protection, high availability and disaster management.

- ConfigMap Caching - Controllers read the small, aggregated rule set from a single ConfigMap, minimizing API calls.
- Indexing - Use field-indexed cache on Pods to avoid full cluster scans (for large clusters).
- Concurrency - Reconciliation workers process namespaces in parallel, bounded by rate limits.

G. *Evaluation*

We have basically

- Testbed - 3-node Kubernetes cluster with 500 Namespaces, each hosting 10 Pods.
- Metrics & KPIs - Block Rate (% of create/update requests denied by webhook), Patch Latency (Time between rule change / drift injection and controller patch), Controller Overhead (CPU & memory footprint at 1000 Pods).
- Procedure - Inject around 100 non-compliant Pods across namespaces and measure how many are blocked vs. patched. Finally, we record patch latency and resource usage using Prometheus.
- Scalability test - Programmatically created 500 namespaces, each with 10 Pods running without **runAsNonRoot**. After deployment of self-healing operator (one replica) and webhook, removed runAsNonRoot across all namespaces. MTTC refers to the mean time to compliance, which refers to average of all latencies across all 500 patch events.

3. Results and Discussion

We have noticed that the admission webhook provides fast, front-line enforcement, blocking 1/3 of non-compliant requests instantly. The self-healing operator repairs drift within 10 s in 95% of cases, supporting retroactive compliance. Resource overhead remains low (<0.8 cores, <200 MB) even when scaled to 500 namespaces. Finally, the Observability metrics allow fine-grained tuning of reconciliation intervals and policy complexity.

Table I. Results - Metric and Outcome

Metric Group	Metric	Value
Block Rate	Total create/update attempts	1,000
	Denied by admission webhook	320
	Block rate	32%
Patch Latency	Drift injections performed	500
	Mean Time to Compliance (MTTC)	7.2 s
	95th-percentile latency	12.5 s
	Maximum observed latency	18.0 s
Controller & Webhook Overhead	Self-Healing Operator (CPU/Mem)	0.15 cores/120 MB
	Admission Webhook (CPU/Mem)	0.05 cores (@100 req/s)/60MB
Scalability Test (500 namespaces)	Average MTTC	15 s
	Controller CPU usage	0.6 cores
	Unexpected rejections	0 (as expected)
Observability & Logging	policy_reconcile_duration_seconds	Time taken per reconciliation loop
	admission_requests_total{allowed, denied}	Counts of webhook decisions
	patch_requests_total{success,failure}	Counts of self-healing patch operations

A. *Optimization Opportunities*

- Scalability - Controller list operations can be optimized with field-indexed caches.
- Rule Conflicts - Define rule priority or merging semantics when multiple patches target the same field.
- Extensibility - Extend to other resource kinds (ConfigMap, Deployment) by adding support in webhook & reconciler.

- Security - Secure the GitOps agent and webhook server with TLS and RBAC.
- Observability - Instrument webhook and controller with metrics (Prometheus) and structured logs.
- Image Scans – The same mechanism can be extended to cover image scanning.

4. Conclusion

We have presented a fully automated loop for policy enforcements and self-healing capability. Future directions may consider

- Policy DSL - A higher-level language to author complex rules (e.g., regular expressions on labels).
- Event-Driven Healing - Leverage Kubernetes Events & informers for push-based reconciliation.
- Cross-Cluster Federation - Share policies across clusters via a centralized policy engine.
- AI-Driven Suggestions - Use ML to propose new policies based on observed drift and security findings.

5. Acknowledgment

Thanks to Tony Hansen, Swapna Anumolu for reviewing the paper and providing valuable input.

References

Argo CD - Declarative GitOps CD for Kubernetes. (n.d.). Argo CD Documentation. Retrieved January 5, 2026, from https://argo-cd.readthedocs.io/en/stable/

Bryan, P. C., & Nottingham, M. (2025). *RFC 6902: JavaScript Object Notation (JSON) Patch*. IETF. https://datatracker.ietf.org/doc/html/rfc6902

Burns, B., Grant, B., Oppenheimer, D., Brewer, E., & Wilkes, J. (2016). Borg, Omega, and Kubernetes. *Communications of the ACM, 59*(5), 50–57. https://doi.org/10.1145/2890784

Gazitt, O. (2022). *Policy-as-Code or Policy-as-Data? Why choose?* Aserto Blog. Retrieved January 5, 2026, from https://www.aserto.com/blog/policy-as-code-or-policy-as-data-why-choose

Gurajapu, A. (2024). Towards a Futuristic Security Roadmap: Advanced Strategies. *Journal of Computer Science and Technology Studies*. https://doi.org/10.13140/rg.2.2.16748.01928

Gurajapu, A. (2026a). Leveraging Artificial Intelligence to Bridge Execution Gaps in SAFe®-Scaled Agile Based Programs. *World Journal of Advanced Engineering Technology and Sciences*. https://doi.org/10.30574/wjaets.2026.18.1.1585

Gurajapu, A. (2026b). Orchestrating Adaptive Resilience and Continuity Restoration in Cloud-Native Environments. *International Journal of Inventions in Engineering & Science Technology, 12*(01). https://doi.org/10.37648/ijiest.v12i01.001

Gurajapu, A. (2026c). Shift-Left Security Validation of Containers via Kubernetes Admission Webhook. *Frontiers in Computer Science and Artificial Intelligence*. https://doi.org/10.32996/jcsts.2026.5.1.6

Gurajapu, A. (2026d). Swap Kubernetes Secrets Without Application Disruption - Comparative Study and eBPF-Powered Kernel Interception Framework. *World Journal of Advanced Engineering Technology and Sciences*. https://doi.org/10.30574/wjaets.2026.18.1.0005

Kubernetes-sigs. (2025, November 16). *GitHub - kubernetes-sigs/kubebuilder: Kubebuilder - SDK for building Kubernetes APIs using CRDs*. GitHub. Retrieved January 5, 2026, from https://github.com/kubernetes-sigs/kubebuilder

U., & _. (2019). *Kubernetes: Up and Running, 2nd Edition*. O'Reilly Online Learning. https://www.oreilly.com/library/view/kubernetes-up-and/9781492046523/

Zhang, Q., Cheng, L., & Boutaba, R. (2010). Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications, 1*(1), 7–18. https://doi.org/10.1007/s13174-010-0007-6