Guardrails-as-a-Service: Hybrid Deterministic and Semantic Policy Evaluation

Pradeep Annepu, Rajesh Akula
Email: {M25AI1109, M25AI1109}@iitj.ac.in

***Abstract*—We present the architecture and planned evaluation of a cloud-agnostic Guardrails-as-a-Service platform that integrates deterministic rule evaluation with semantic vector retrieval to enforce complex governance policies for AI-enabled systems. The design emphasizes scalability via stateless microservices and asynchronous event flow, extensibility through a handler abstraction, and audit integrity using hash-chained decision logs. We define measurable quality attributes (latency, throughput, extensibility effort, audit lag) and outline empirical validation strategies supporting operational assurance and trust.**

***Index Terms*—Policy enforcement, microservices, vector similarity, audit logging, hash chaining, observability, cloud governance.**

I. Introduction

Dynamic policy enforcement in modern distributed and AI-centric systems requires synthesizing explicit rules with semantic context. Traditional engines (e.g., Open Policy Agent (OPA) [1]) optimize declarative logic but lack native semantic retrieval. Cloud-specific governance (e.g., AWS Control Tower guardrails [2]) constrains portability. We address these gaps with a hybrid evaluation plane coupling rule logic and embedding similarity, backed by an integrity-focused audit pipeline.

II. Contributions

The work provides: (1) A hybrid evaluation architecture combining deterministic rule filtering and vector similarity; (2) An extensible handler interface enabling semantic and emerging policy types with low change cost; (3) A hash-linked audit log for tamper-evidence without blockchain overhead; (4) A metrics-driven evaluation plan aligning architectural tactics with verifiable quality attributes.

III. System Architecture

Fig. 1 System Architecture
Core components:

- API Gateway: Authentication, authorization, rate limiting.
- Policy Service: CRUD, versioning, vector indexing (pgvector [4]).
- Evaluation Plane: Cache-first retrieval, rule + semantic evaluation, decision event emission.
- Audit Logger: Consumes decision events; persists hash-chained records.
- Redis: Low-latency hot-set and embedding cache.
- Broker: Decouples evaluation from persistence; supports resilience tests.
- Observability Stack: Metrics (Prometheus [3]), logs, planned traces (OpenTelemetry [5]).

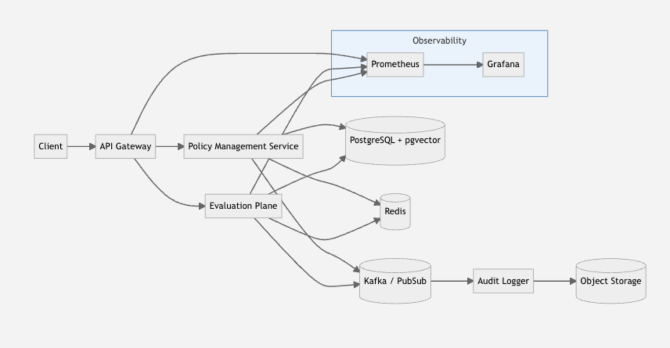

Fig. 1. System Architecture: Hybrid policy evaluation with event-driven decoupling

IV. Design Methodology

Microservice decomposition isolates policy mutation from evaluation throughput. Event-driven processing reduces synchronous coupling. Vector similarity augments metadata filtering, allowing semantically proximate policy selection. Hash chaining supplies integrity assurance with linear verification.

V. Quality Attributes

A. Scalability

Tactics: Stateless pods, horizontal autoscaling, Redis caching, asynchronous buffering. Metrics: P95 latency $<$ 150 ms under 2000 requests/sec; cache hit ratio correlated with latency reduction.

B. Extensibility

A `PolicyHandler` interface:

```
export interface PolicyHandler {
  supports(policyType: string): boolean;
  evaluate(context: any, policy: Policy): Promise;
}
```

New handlers (e.g., advanced semantic constraint) integrate via dependency injection without core modification. Verification: Diff locality; unchanged regression suite; cyclomatic complexity ¡ 10.

C. Observability & Auditability

Correlation IDs propagate through logs and metrics. Audit integrity uses chained hashes:

```
prev_hash = "GENESIS"
for event in decision_events:
    serialized = canonical_json(event)
    curr_hash = sha256(prev_hash + serialized)
    store({...event, prev_hash, curr_hash})
    prev_hash = curr_hash
```

Periodic verification recomputes sequence and alerts on mismatch; target audit lag ¡ 5 s at sustained load.

D. Maintainability

Modular boundaries and standardized evaluation signatures reduce change effort. KPIs: Mean Time To Change for routine policy extension ¡ 0.5 developer-day.

VI. Implementation Overview

Policies stored in PostgreSQL with pgvector embeddings; evaluation pipeline performs: (1) Cache lookup; (2) Vector similarity search (threshold-based); (3) Rule evaluation; (4) Decision event emission; (5) Audit service hash-link persistence.

VII. Evaluation Plan

Phases: (1) Functional tests (unit, integration); (2) Load tests (k6 ramp to 2000 req/sec); (3) Resilience (broker partition throttling). Metrics scraped every 5 s; Grafana dashboards visualize throughput, latency, audit lag. Success: No message loss; hash chain validates end-to-end; latency targets met.

VIII. Security Considerations

Current: JWT-based client auth; least-privilege DB roles; rate limiting. Planned: mTLS inter-service; policy signature verification for provenance.

IX. Related Work

OPA [1] offers strong declarative evaluation but lacks semantic retrieval integration and native hash-chained audit logging. AWS Control Tower guardrails [2] provide opinionated cloud governance with limited cross-cloud extensibility. Our approach differentiates through hybrid retrieval, event-driven decoupling, and cryptographic audit integrity.

X. Conclusion

The proposed Guardrails-as-a-Service platform aligns architectural tactics with measurable quality attributes. Hybrid policy evaluation and hash-linked auditing together enhance governance fidelity and trust. Future work targets production hardening, expanded semantic handlers, and full distributed tracing.

Acknowledgment

Omitted for anonymity.

References

[1] Open Policy Agent Documentation. [Online]. Available: https://www.openpolicyagent.org

[2] AWS Control Tower Guardrails. [Online]. Available: https://docs.aws.amazon.com/controltower

[3] Prometheus Documentation. [Online]. Available: https://prometheus.io

[4] pgvector Extension Documentation. [Online]. Available: https://github.com/pgvector/pgvector

[5] OpenTelemetry Specification. [Online]. Available: https://opentelemetry.io