Guardrails as a Service

CSL7090 - Project , IIT Jodhpur

IIT Jodhpur

November 16, 2025

Members:
Pradeep Annepu (M25AI1109) Rajesh Akula (M25AI1109)
V Amarendra Chakravarthi (M25AI1109) Anirudh Reddy (M25AI1109)

Outline

Guardrails as a Service

- AI safety and policy enforcement platform
- Microservices architecture for scalability
- Real-time evaluation of AI prompts and responses
- Policy-based governance for AI applications
- Comprehensive audit logging and monitoring

Key Features

Core Capabilities:

- Policy management
- Real-time evaluation
- Lora based inference Model
- Audit logging
- Rate limiting

Technical Features:

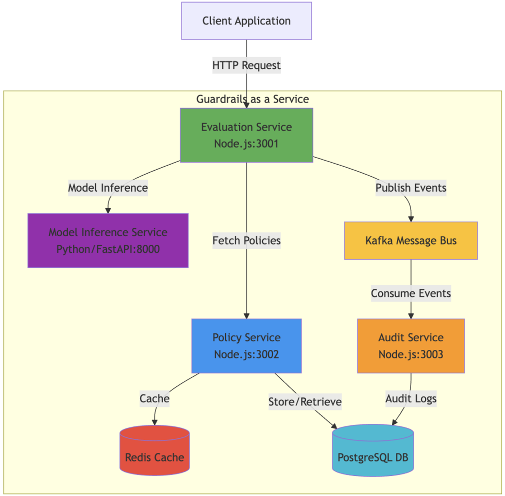
- RESTful API
- Event-driven architecture
- Caching with Redis
- PostgreSQL persistence
- Kafka message bus

Service Overview

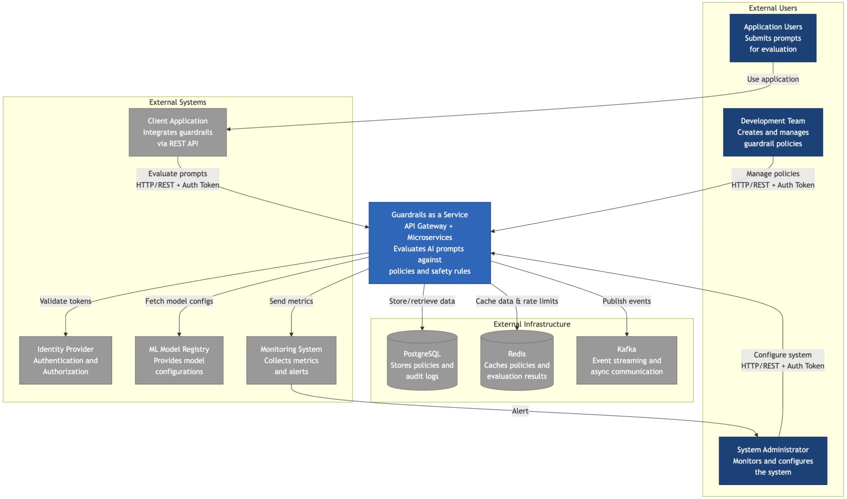
Five Microservices

1. **API Gateway** (Node.js) - Port 3000
   - Authentication & Authorization
   - Request routing
   - Rate limiting
2. **Evaluation Service** (Node.js) - Port 3001
3. **Policy Service** (Node.js) - Port 3002
4. **Audit Service** (Node.js) - Port 3003
5. **Model Inference Service** (Python/FastAPI) - Port 8000

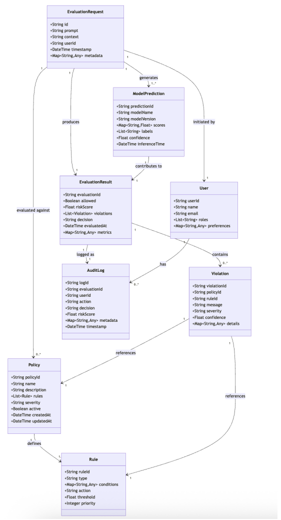
High-Level Architecture



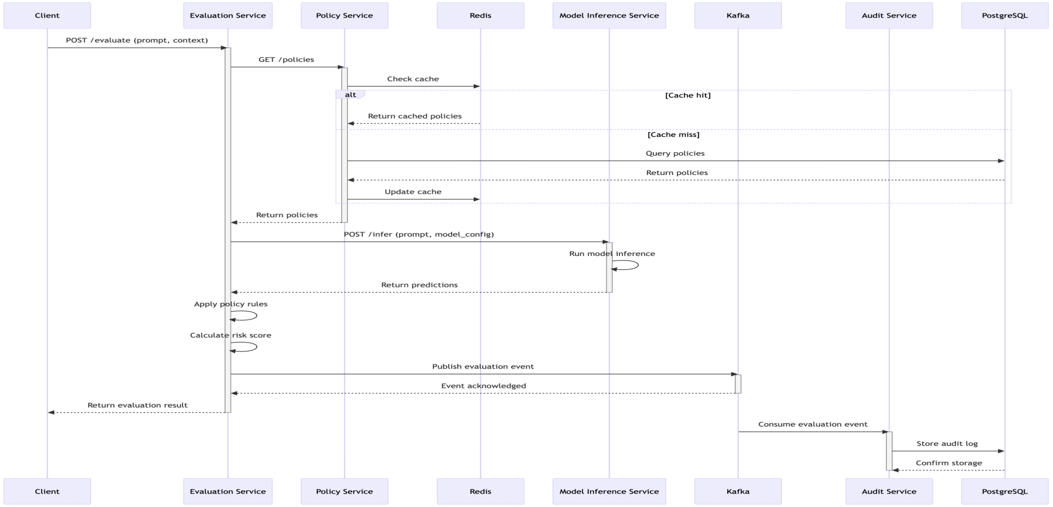
C4 System Context


External Users
Application Users Submits prompts for evaluation
Use application
Development Team Creates and manages guardrail policies
Manage policies HTTP/REST + Auth Token
External Systems
Client Application Integrates guardrails via REST API
Evaluate prompts HTTP/REST + Auth Token
Guardrails as a Service API Gateway + Microservices Evaluates AI prompts against policies and safety rules
Validate tokens
Fetch model configs
Send metrics
Store/retrieve data
Cache data & rate limits
Publish events
Identity Provider Authentication and Authorization
ML Model Registry Provides model configurations
Monitoring System Collects metrics and alerts
External Infrastructure
PostgreSQL Stores policies and audit logs
Redis Caches policies and evaluation results
Kafka Event streaming and async communication
Configure system HTTP/REST + Auth Token
Alert
System Administrator Monitors and configures the system

Core Domain Entities



Sequence Diagram: Evaluation Flow



Technology Stack

Backend Services:

- Node.js ($\geq$ 18.0.0)
- TypeScript
- Express.js
- Python ($\geq$ 3.8)
- FastAPI

Infrastructure:

- PostgreSQL (persistence)
- Redis (caching)
- Apache Kafka (messaging)

Development Tools:

- npm ($\geq$ 8.0.0)
- pip
- ESLint
- Prettier

Testing & Monitoring:

- Jest (unit testing)
- k6 (load testing)
- Monitoring integration

Load Testing with k6

Prerequisites:

```
# macOS
brew install k6

# Windows
choco install k6
```

Running Load Tests:

```
cd load_test
k6 run load_test.js

# With web dashboard
k6 run --out web-dashboard load_test.js
```

Load Test Configuration

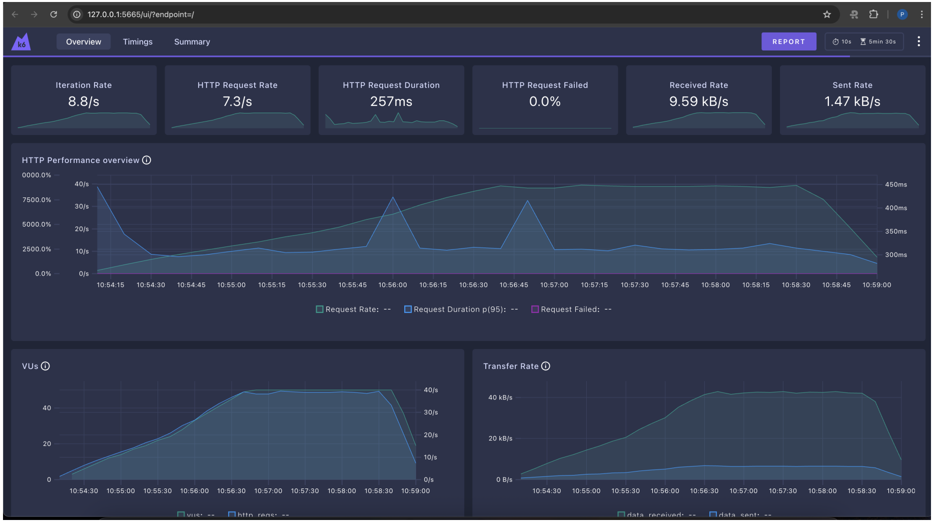
Default Load Test Pattern

- 30s: Ramp up to 10 users
- 1m: Ramp up to 25 users
- 1m: Ramp up to 50 users
- 2m: Maintain 50 users (peak load)
- 30s: Ramp down to 0 users

Configuration Notes

- Update host link to API Gateway URL
- Include authentication tokens
- Ensure all services are running before testing

Load Test Results



Iteration Rate 8.8/s

HTTP Request Rate 7.3/s

HTTP Request Duration 257ms

HTTP Request Failed 0.0%

Received Rate 9.59 kB/s

Sent Rate 1.47 kB/s

Quality attributes

1. Security
   - Centralized authentication and authorization via API Gateway
   - Rate limiting to prevent abuse
2. Scalability
   - Microservices architecture allows independent scaling
   - Redis caching to reduce latency
   - Kafka for asynchronous communication
3. Flexibility
   - Easy to add new policies and rules
   - Modular services for future enhancements

Questions?

Thank You!

GitHub: github.com/pradeepannepu/guardrails-as-a-service