OrgGuard – Automate GitHub Policy Checks & Visualize Compliance in Grafana

by Deexith Parandaman

About Me

(Just Another Developer)

- Platform Engineer at JMAN Group
- Passionate about automation and DevOps
- Loves building scalable and efficient solutions

Motivation for OrgGuard

Why We Built This

- Managing GitHub user naming standards was difficult.
- Users had inconsistent names like `ideexith`, `i_am_deexith`, `kickbuttowski`.
- Unused invites occupied costly license spaces.
- No structured way to **audit** and **enforce policies**.
- **Solution:** Policy enforcement using `OPA`, `Webhooks`, and `Grafana`.

Tech Flow

How OrgGuard Works

1. **GitHub Webhook** in Golang captures events.
2. **Ngrok** exposes local services for testing.
3. **Webhook Route** processes GitHub events.
4. **Triggers** on new member additions or invites.
5. **Sends Data to OPA** for policy validation.
6. **Stores Violations in PostgreSQL** for tracking.
7. **Visualizes Violations in Grafana** for insights.

OrgGuard: The Setup

A Simple Yet Powerful Model

- **GitHub Webhooks** → **Golang Backend** → **OPA Policy Check**
- **PostgreSQL** for storing violations
- **Grafana Dashboard** for real-time insights
- Deployable with **Docker Compose**
- Supports **custom policies** for different organizations

DEMO

Live Walkthrough of OrgGuard

Why Self-Host OrgGuard?

Cost vs Buying GitHub Enterprise

- **GitHub Teams/Enterprise** is expensive.
- **OrgGuard** offers custom policy enforcement at a lower cost.
- **Greater flexibility** and **full control** over access rules.
- **Audit logs and monitoring** without extra costs.

Extending OrgGuard

Possible Enhancements

- **Custom Webhook Actions** beyond user management.
- **Enforce Repository Naming Conventions**.
- **Track PR/Merge Commit Policies**.
- **Implement Security Best Practices** like enforcing 2FA.

Thanks for Attending!

Reuse This Idea, Make It Better!