🛡 Self-Healing Infrastructure with Chaos Engineering

Project Goal:

This project demonstrates an industry-ready, automated Kubernetes platform that ensures application availability and resilience by:

- **Automatically recovering from failures (Self-healing infrastructure)**
- **Actively monitoring critical microservices (Prometheus/Grafana)**
- **Simulating and learning from real failures (LitmusChaos chaos engineering)**
- **Delivering observability and hands-off reliability (Zero-touch, minimal manual intervention)**

Key Components and Technologies

- **Go Microservice: Runs a sample "Voting App" tracked by real-time metrics**
- **PostgreSQL Database: Backend datastore for the application**
- **Prometheus & Grafana: For application/process health monitoring and visualization**
- **LitmusChaos: For orchestrating chaos experiments in Kubernetes to test failure recovery**
- **Kubernetes (via Minikube/Docker): Container orchestration and automation platform**
- **Automated Recovery - CronJob: For auto-restarting apps based on Prometheus alerts**

What the System Can Do

- **Detects application/database failures automatically**
- **Recovers by restarting failed pods without manual effort**
- **Provides dashboard views for real-time health and metrics**
- **Allows safe, controlled “chaos” experiments for reliability analysis**
- **Enables SRE/DevOps best practices for cloud infrastructure**

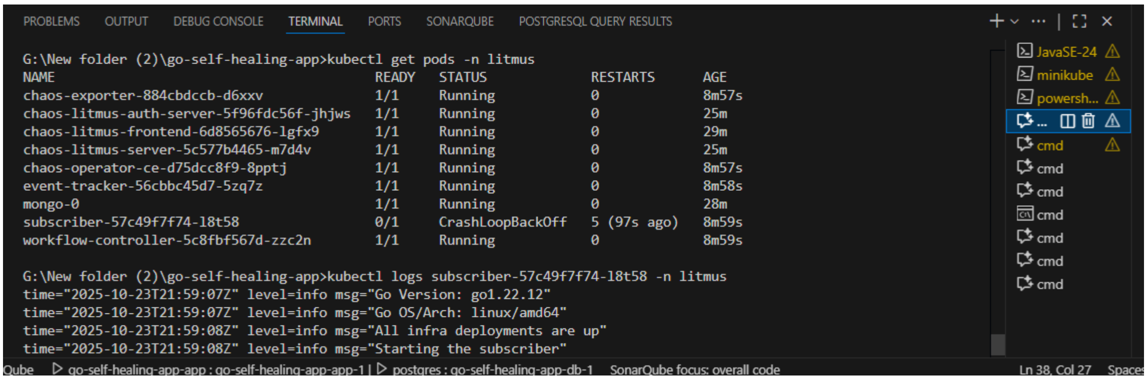
Step-by-Step Technical Flow:

1. Environment Preparation

- **Kubernetes Cluster launched with Minikube (using Docker driver)**
- **Helm 3 and kubectl installed for package and cluster management**

2. App & Database Deployment

- **Go Voting Application built and containerized (image hosted on GHCR)**
- **PostgreSQL deployed via Bitnami Helm Chart; DB secrets handled with K8s secrets**



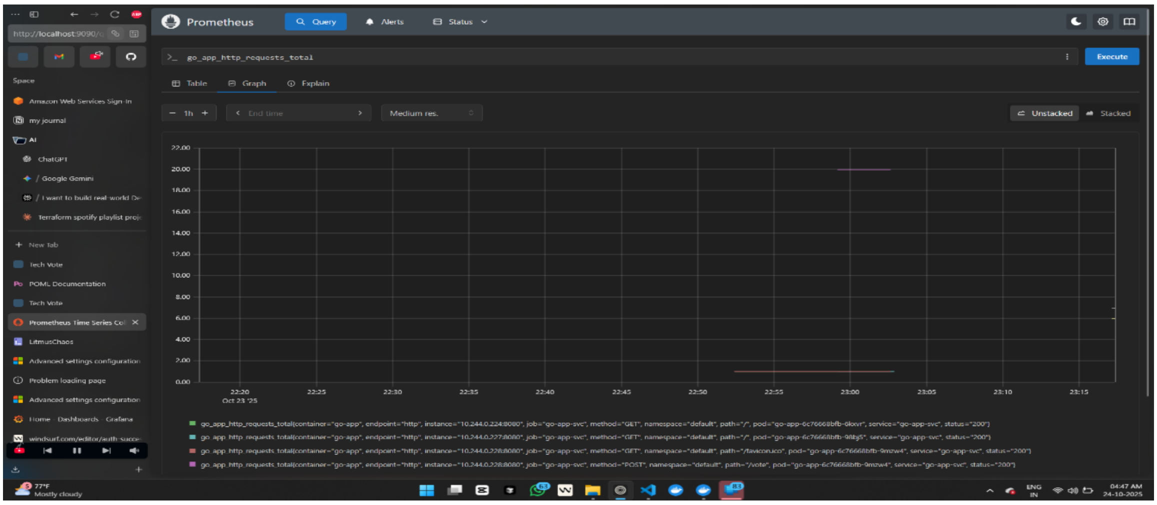
```
G:\New folder (2)\go-self-healing-app>kubectl get pods -n litmus
NAME                                     READY   STATUS             RESTARTS      AGE
chaos-exporter-884cbdccb-d6xxv           1/1     Running            0             8m57s
chaos-litmus-auth-server-5f96fdc56f-jhjws 1/1    Running            0             25m
chaos-litmus-frontend-6d8565676-lgfx9    1/1     Running            0             29m
chaos-litmus-server-5c577b4465-m7d4v     1/1     Running            0             25m
chaos-operator-ce-d75dcc8f9-8pptj        1/1     Running            0             8m57s
event-tracker-56cbbc45d7-5zq7z           1/1     Running            0             8m58s
mongo-0                                  1/1     Running            0             28m
subscriber-57c49f7f74-18t58              0/1     CrashLoopBackOff   5 (97s ago)   8m59s
workflow-controller-5c8fbf567d-zzc2n     1/1     Running            0             8m59s

G:\New folder (2)\go-self-healing-app>kubectl logs subscriber-57c49f7f74-18t58 -n litmus
time="2025-10-23T21:59:07Z" level=info msg="Go Version: go1.22.12"
time="2025-10-23T21:59:07Z" level=info msg="Go OS/Arch: linux/amd64"
time="2025-10-23T21:59:08Z" level=info msg="All infra deployments are up"
time="2025-10-23T21:59:08Z" level=info msg="Starting the subscriber"
```

Qube ▷ go-self-healing-app-app : go-self-healing-app-app-1 | ▷ postgres : go-self-healing-app-db-1 SonarQube focus: overall code

3. Service Exposure & Monitoring

- **Application exposed via Kubernetes Service (see VS Code/Minikube dashboard in screenshot)**
- **Prometheus and Grafana installed to monitor custom app metrics like go_app_http_requests_total**
- **Dashboard created for real-time requests and health status**

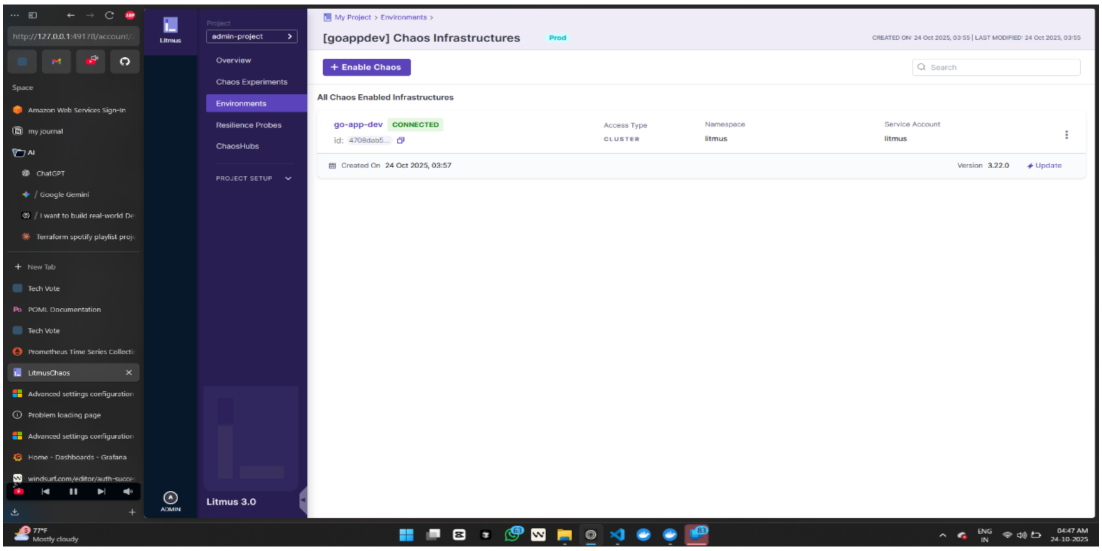


```
go_app_http_requests_total
```

- 🟩 go_app_http_requests_total{container="go-app", endpoint="http", instance="10.244.0.224:8080", job="go-app-svc", method="GET", namespace="default", path="/", pod="go-app-6c76668bfb-6kxvr", service="go-app-svc", status="200"}
- 🟦 go_app_http_requests_total{container="go-app", endpoint="http", instance="10.244.0.227:8080", job="go-app-svc", method="GET", namespace="default", path="/", pod="go-app-6c76668bfb-98bj5", service="go-app-svc", status="200"}
- 🟥 go_app_http_requests_total{container="go-app", endpoint="http", instance="10.244.0.228:8080", job="go-app-svc", method="GET", namespace="default", path="/favicon.ico", pod="go-app-6c76668bfb-9mzw4", service="go-app-svc", status="200"}
- 🟪 go_app_http_requests_total{container="go-app", endpoint="http", instance="10.244.0.228:8080", job="go-app-svc", method="POST", namespace="default", path="/vote", pod="go-app-6c76668bfb-9mzw4", service="go-app-svc", status="200"}

4. Chaos Engineering Integration

- **LitmusChaos portal and agent deployed, infrastructure registered with self-agent**



[goappdev] Chaos Infrastructures Prod

CREATED ON: 24 Oct 2025, 03:55 | LAST MODIFIED: 24 Oct 2025, 03:55

All Chaos Enabled Infrastructures

go-app-dev CONNECTED
id: 4708dab5...

Access Type: CLUSTER
Namespace: litmus
Service Account: litmus

Created On 24 Oct 2025, 03:57
Version 3.22.0 Update

- **Experiments prepared to simulate failures (pod kill, network delay, DB outages, etc.)**

5. Automated Remediation

- **CronJob implemented to auto-restart stuck apps on Prometheus alert triggers (entirely hands-off recovery)**

Key Screenshots

1. Secrets & Authentication Loops

- **Mismatched DB passwords led to app CrashLoopBackOff; fixed by aligning Kubernetes secrets with real DB creds.**
- **Litmus MongoDB pod user mismatch (admin/root) caused initialization/auth issues; solved through deployment/env patch and secret reset.**

2. Pod Failure & Log Analysis

- **Used kubectl logs and describe pod commands to diagnose container config errors, stuck initialization, and permission denials.**

3. Chaos Agent Stuck in Pending

- **Registration issues with Litmus agent (PENDING infra) resolved by proper YAML application and ensuring correct namespace/cluster context.**

4. Monitoring Integration

- **Ensured /metrics endpoint exposed by Go app and correct Prometheus service discovery config.**

5. Automated Remediation Setup

- Alertmanager and CronJob failures required proper RBAC setup and pod label targeting for reliable remediation actions.

Professional Outcomes

- Demonstrated DevOps, SRE, and Cloud-Native production-readiness.
- Automated real-time resilience for a microservice stack.
- Hands-on expertise with cloud-native tools and industry-standard incident recovery workflows.

Key Takeaway for Recruiters/HR

This project bridges theory and practice.
It not only runs a microservice reliably, but proves it will stay available with real monitoring, chaos tests, and zero-touch recovery exactly what modern software teams expect!