Self-Healing Infrastructure with Prometheus, Alertmanager & Ansible

Abstract

This project demonstrates the design and implementation of a self-healing infrastructure that ensures high availability and reliability of services. Using Prometheus for monitoring, Alertmanager for alerting, and Ansible for automation, the system automatically detects failures, triggers alerts, and recovers services without manual intervention. The solution is built around Docker containers and showcases the integration of open-source monitoring and automation tools in a modern DevOps environment.

Introduction

In production environments, service availability is crucial. Even minor downtimes can result in significant losses. This project introduces a self-healing infrastructure where failures such as Nginx downtime or high CPU usage are detected automatically, alerts are generated, and remediation is executed using Ansible. The approach minimizes downtime, reduces manual interventions, and ensures reliability.

Tools Used

- Docker & Docker Compose: Containerization and orchestration of services.
- Nginx: Web server monitored for downtime.
- Prometheus: Collects metrics and evaluates alert rules.
- Node Exporter: Exports system-level metrics.
- Alertmanager: Manages alerts generated by Prometheus.
- Webhook & Ansible: Executes self-healing automation tasks.

Steps Involved in Building the Project

1. **Environment Setup**

    - Installed Docker and Docker Compose to containerize and orchestrate services.
    - Created a structured project directory with subfolders for Nginx, Prometheus, Alertmanager, Webhook, and Ansible.

2. **Service Configuration**

    - Nginx: Configured with stub_status enabled for metrics.
    - Nginx Exporter: Deployed to expose Nginx metrics on port 9113.
    - Node Exporter: Configured to collect system-level metrics such as CPU, memory, and disk usage.
    - Prometheus: Defined scrape jobs in prometheus.yml to collect metrics from Nginx and Node Exporter.
    - Alertmanager: Configured alertmanager.yml to forward alerts to a custom webhook receiver.

3. **Defining Alerts**

    - Created alert.rules.yml with rules for:
    - NginxDown – Detects Nginx downtime.
    - HighCPUUsage – Triggers when CPU utilization crosses the threshold.
    - Integrated the rules into Prometheus for active monitoring.

4. **Webhook Integration & Self-Healing**

   - Developed a Flask-based webhook service to receive alerts from Alertmanager.
   - Configured the webhook to trigger Ansible playbooks for automated remediation.
   - Example: Automatically restart the Nginx container when the NginxDown alert is fired.

5. **Testing & Validation**

   - Simulated failures to test self-healing mechanisms:
   - Stopped the Nginx container → Verified Prometheus detected downtime, Alertmanager fired an alert, and the webhook executed Ansible playbooks to restart Nginx automatically.
   - Used the stress tool to generate high CPU load → Confirmed CPU alerts were triggered and handled.

6. **Final Outcome**

   - Delivered a fully automated self-healing monitoring system.
   - The system detects failures, generates alerts, and executes recovery actions automatically.
   - Ensures high availability, resilience, and reduced downtime through proactive monitoring and remediation.

Conclusion

The project successfully implements a self-healing infrastructure that ensures automated monitoring, alerting, and remediation. It demonstrates the power of combining monitoring tools like Prometheus with automation frameworks like Ansible to maintain system resilience. This approach reduces downtime, improves reliability, and can be extended to more complex production environments.

Author
Vaishnavi Tiwari
Github: https://github.com/VaishnaviT501/Self_healing_infra.git