Self-Healing Infrastructure with Prometheus, Alertmanager, and Ansible

1. Abstract

This project implements a self-healing infrastructure to automatically detect and recover from service failures using Prometheus for monitoring, Alertmanager for alerting, and Ansible for automated recovery. A sample NGINX service is monitored for uptime and CPU usage, with alerts triggering Ansible playbooks to restart the service or system when thresholds are breached (e.g., service down or CPU > 90%). The setup, deployed on an Ubuntu 22.04 VM, demonstrates real-time monitoring and recovery, enhancing system reliability.

2. Introduction

Modern systems demand high availability, where manual intervention for failures is impractical. This project addresses this by creating a self-healing system that autonomously detects issues and initiates recovery. Using Prometheus to monitor metrics, Alertmanager to handle alerts, and Ansible to execute recovery actions, the system ensures minimal downtime. The objective is to monitor an NGINX service, detect failures (e.g., service downtime or high CPU usage), and automatically recover via predefined playbooks, all within an Ubuntu environment.

3. Tools Used

- **Prometheus**: Time-series database for monitoring NGINX uptime and system CPU usage.
- **Alertmanager**: Manages alerts and triggers webhooks for notifications.
- **Ansible**: Automation tool to execute recovery playbooks (e.g., restarting NGINX).
- **NGINX**: Sample web server for monitoring and recovery demonstrations.
- **Node Exporter**: Collects system metrics (e.g., CPU usage) for Prometheus.
- **NGINX Prometheus Exporter**: Exports NGINX metrics for monitoring.
- **Flask**: Python framework for a webhook server to bridge Alertmanager and Ansible.
- **Ubuntu 22.04**: Operating system hosting all components.

4. Steps Involved in Building the Project

1. **Environment Setup**: Installed NGINX, Prometheus, Node Exporter, NGINX Prometheus Exporter, Alertmanager, Ansible, and Flask on an Ubuntu 22.04 VM. Configured systemd services for each component to ensure persistence.

2. **Prometheus Configuration**: Set up `prometheus.yml` to scrape metrics from Prometheus (port 9090), Node Exporter (9100), and NGINX Exporter (9113). Defined alert rules in `rules.yml` for NGINX downtime (`up{job="nginx"} == 0`) and high CPU usage (>90%).
3. **Alertmanager Setup**: Configured `alertmanager.yml` to send alerts to a Flask-based webhook (port 5000). Alerts trigger when conditions persist (e.g., 1 minute for NGINX down, 2 minutes for high CPU).
4. **Webhook Implementation**: Developed a Flask app (`webhook.py`) to receive Alertmanager notifications and execute an Ansible playbook.
5. **Ansible Playbook**: Created `recover.yml` to restart NGINX or, optionally, reboot the system for severe issues. The playbook runs locally with elevated privileges.
6. **Testing and Validation**: Simulated failures by stopping NGINX or stressing CPU (using `stress`). Verified alerts in Prometheus UI (port 9090), Alertmanager UI (port 9093), and recovery via logs (e.g., Ansible output, webhook logs).

5. Conclusion

The self-healing infrastructure successfully monitors NGINX and system metrics, detects failures, and automates recovery. Prometheus and Alertmanager provide robust monitoring and alerting, while Ansible ensures rapid response. The system reduces downtime and manual intervention, making it suitable for production environments. Future enhancements could include multi-node setups, advanced recovery strategies, or integration with cloud platforms.