Self-Healing Infrastructure with Prometheus, Alertmanager, and Ansible
===

Introduction

Modern IT systems demand high availability and resilience. Manual intervention for service failures is inefficient, prompting the need for self-healing infrastructures that automatically detect and recover from issues. This project implements a self-healing system using Prometheus for monitoring, Alertmanager for notifications, and Ansible for automated recovery. A sample NGINX service is monitored for downtime, and upon failure, the system triggers an automated restart, demonstrating a robust approach to fault tolerance. The objective is to showcase how open-source tools can create a scalable, automated recovery mechanism, reducing downtime and operational overhead.

Abstract

This project develops a self-healing infrastructure to detect and recover from service failures. An NGINX web server is deployed in a Docker environment, monitored by Prometheus for uptime and system metrics. Alertmanager handles alerts triggered by predefined thresholds (e.g., NGINX downtime or CPU usage >90%). Upon alert detection, a webhook service invokes an Ansible playbook to restart the NGINX container, ensuring rapid recovery. The system was tested by simulating NGINX failures, with logs and metrics verifying successful auto-healing. The project highlights the integration of monitoring, alerting, and automation tools to enhance system reliability.

Tools Used

- **Prometheus**: A time-series database for monitoring NGINX uptime and system metrics (e.g., CPU usage).
- **Alertmanager**: Manages alerts from Prometheus and routes them to a webhook for action.
- **Ansible**: Automates recovery by executing playbooks to restart services.
- **Docker**: Runs NGINX, Prometheus, Alertmanager, and other services in containers.
- **NGINX**: A sample web server to demonstrate failure detection and recovery.
- **Flask (Python)**: Powers the webhook service to bridge Alertmanager and Ansible.
- **Node Exporter**: Collects system metrics for Prometheus.
- **Ubuntu/Docker Desktop**: The host environment for deployment.

Steps Involved in Building the Project
----
1. **Environment Setup**:
   - Initialized a project directory and installed Docker, Ansible, and Python.
   - Created subdirectories for Prometheus, Alertmanager, Ansible, and logs.
2. **NGINX Deployment**:
   - Configured a Docker Compose file to run NGINX on port 8080 with health checks.

   - Mounted NGINX logs to a local directory for monitoring.
3. **Prometheus Configuration**:
   - Set up `prometheus.yml` to scrape NGINX and Node Exporter metrics every 15 seconds.
   - Defined alerting rules (`alert.rules.yml`) for NGINX downtime (1 minute) and high CPU usage (>90% for 2 minutes).
4. **Alertmanager Setup**:
   - Configured `alertmanager.yml` to route alerts to a webhook service.
   - Deployed Alertmanager via Docker Compose, exposing port 9093.
5. **Webhook Service**:
   - Developed a Flask-based webhook (`webhook.py`) to receive alerts and trigger Ansible.
   - Built a Docker image for the webhook with Ansible installed.
6. **Ansible Playbook**:
   - Created `restart_nginx.yml` to restart the NGINX container using the `docker_container` module.
   - Logged playbook execution to `ansible.log`.
7. **System Integration**:
   - Updated Docker Compose to include NGINX, Node Exporter, Prometheus, Alertmanager, and webhook services.
   - Ensured inter-service communication via a bridge network.
8. **Testing and Validation**:
   - Started all services with `docker-compose up -d --build`.
   - Simulated an NGINX failure by stopping its container (`docker stop self-healing-infra_nginx_1`).
   - Verified auto-healing through logs (`webhook.log`, `ansible.log`) and confirmed NGINX recovery via `curl http://localhost:8080`.
9. **Documentation**:
   - Prepared a GitHub repository with all configurations, logs, and a README.md detailing setup and usage.
   - Included example logs and optional screenshots for demonstration.

Conclusion

The self-healing infrastructure project successfully demonstrates automated failure detection and recovery using Prometheus, Alertmanager, and Ansible. By integrating these tools, the system monitors NGINX, detects downtime, and restores service without manual intervention, achieving high availability. The use of Docker ensures portability, while Ansible provides flexible automation. Testing confirmed reliable recovery within minutes of failure. Future enhancements could include additional alerts (e.g., memory usage), complex recovery playbooks, or deployment on Kubernetes for scalability. This project underscores the power of open-source tools in building resilient systems, offering a foundation for production-grade self-healing infrastructures.