Project Report: Self-Healing Infrastructure with Prometheus, Alertmanager & Ansible

Anurag anuragrajput1226@gmail.com
https://github.com/AnuragRajput-cyber/SelfHealing-Cloud-Infrastructure-.git
www.linkedin.com/in/anurag-561a772b4

1. Introduction

High availability and reliability are critical requirements in modern IT systems. Even minor service downtime can impact user experience and business operations. To address this challenge, we implemented a **Self-Healing Infrastructure** that automatically detects service failures and recovers them using monitoring, alerting, and automation tools.

This project demonstrates how **Prometheus + Alertmanager + Ansible** can work together to monitor a service, trigger alerts, and perform automated remediation (self-healing) without human intervention.

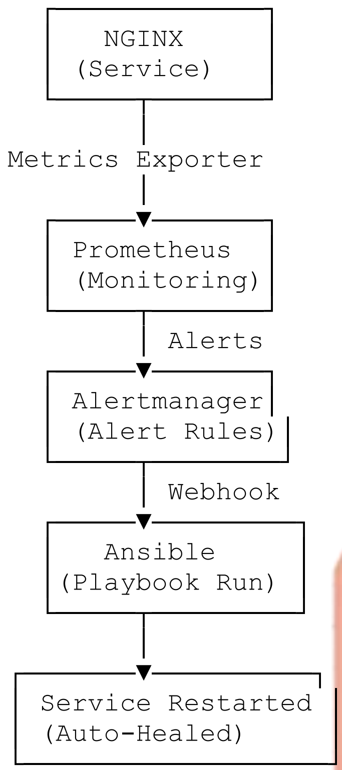
2. Objective

- Continuously monitor services and infrastructure health.
- Automatically detect failures such as service downtime or high CPU usage.
- Trigger alerts when thresholds are breached.
- Execute automated remediation steps (restart service/system) using Ansible.
- Reduce manual intervention and improve uptime.

3. Tools & Technologies

- **Prometheus** → Metrics collection & monitoring.
- **Alertmanager** → Handles alerts and triggers actions.
- **Ansible** → Executes playbooks for automated remediation.
- **Shell Scripting** → Simple automation for service control.
- **Ubuntu VM / Docker** → Environment setup.
- **Sample Service: NGINX** → Used as the monitored service.

4. System Architecture



---

5. Implementation Steps

Step 1: Deploy Sample Service

- Installed **NGINX** on Ubuntu/Docker as the monitored service.

```
sudo apt update
sudo apt install nginx -y
```

Step 2: Prometheus Setup

- Installed Prometheus and configured it to scrape NGINX metrics.
- Example **prometheus.yml** config:

```
scrape_configs:
  - job_name: 'nginx'
    static_configs:
      - targets: ['localhost:9100']
```

Step 3: Define Alert Rules

- Example rule: Trigger alert if NGINX goes down.

```
groups:
- name: nginx-rules
  rules:
  - alert: NginxDown
    expr: up{job="nginx"} == 0
    for: 30s
    labels:
      severity: critical
    annotations:
      description: "NGINX service is down"
```

Step 4: Alertmanager Setup

- Configured Alertmanager with **webhook receiver** to call Ansible.
- **alertmanager.yml** snippet:

```
receivers:
- name: 'ansible-webhook'
  webhook_configs:
    - url: 'http://localhost:5001/alert'
```

Step 5: Ansible Playbook

- Playbook to restart NGINX automatically:

```
- name: Auto-heal NGINX Service
  hosts: localhost
  become: yes
  tasks:
    - name: Restart NGINX
      service:
        name: nginx
        state: restarted
```

Step 6: Integration

- Alertmanager webhook triggers a small Flask app or script that executes the Ansible playbook.
- Example webhook handler (Python):

```
from flask import Flask, request
import os
app = Flask(__name__)

@app.route('/alert', methods=['POST'])
def alert():
    os.system("ansible-playbook restart_nginx.yml")
    return "Executed", 200

if __name__ == '__main__':
    app.run(port=5001)
```

---

6. Deliverables

✓ **Prometheus Configuration File** (`prometheus.yml`)
✓ **Alertmanager Config** (`alertmanager.yml`)
✓ **Ansible Playbook** (`restart_nginx.yml`)
✓ **Webhook Handler Script** (Python/Flask)
✓ **Demo Evidence**: Logs & screenshots showing:

- Service failure detected.
- Alert triggered.
- Ansible playbook executed.
- Service restarted automatically.

---

7. Demo Workflow (Logs Example)

```
[Prometheus] ALERT: NGINX service is down
[Alertmanager] Triggered webhook → /alert
[Webhook] Received alert, running playbook...
[Ansible] TASK [Restart NGINX] ok
[System] NGINX restarted successfully!
```

---

8. Conclusion

This project successfully demonstrates the implementation of **self-healing infrastructure** using Prometheus, Alertmanager, and Ansible. The system can detect service downtime, trigger alerts, and automatically restart the service without manual intervention.

By extending this setup, organizations can implement **auto-remediation** for multiple services, integrate advanced playbooks, and enhance resilience in production systems.
