# ***SELF-HEALING INFRASTRUCTURE***

## 1. Introduction

Modern IT systems require high availability and minimal downtime. Manual intervention for service recovery can be time-consuming and unreliable. This project demonstrates a **self-healing infrastructure** setup where service failures are **automatically detected and remediated** using monitoring, alerting, and automation tools.

---


**2. Abstract**

This project implements a **self-recovering infrastructure** for containerized services. A custom Docker container is monitored using Prometheus. Upon detecting service failures (such as the container being down), Prometheus triggers an alert via Alertmanager. The alert is forwarded to a Flask-based webhook, which in turn executes an Ansible playbook to restart the failed service, achieving **automatic recovery without manual intervention**.


---

**3. Tools Used :-**

| Tools | Purpose |
|---|---|
| **Prometheus** | Monitoring container health |
| **Alertmanager** | Handling and routing alerts |
| **Ansible** | Automating container recovery |
| **Docker** | Running services in isolated containers |
| **Flask (Python)** | Webhook listener to trigger Ansible |

**4. Steps Involved in Building the Project**

1. **Custom Docker Image Deployment**
   **A service was deployed using a custom Docker Hub image (yourusername/custom-nginx:latest) running on port 8080.**
2. **Prometheus Monitoring**
   **Prometheus was configured to scrape the health of the custom container. It uses the up metric to check if the service is responding.**
3. **Alert Rule Definition**
   **A Prometheus alert rule was defined to trigger when the container is unresponsive for more than 30 seconds.**
4. **Alertmanager Setup**
   **On alert firing, Alertmanager forwarded it to a Flask-based webhook hosted in a container.**
5. **Webhook Trigger**
   **The Flask webhook received the alert and executed an Ansible playbook using subprocess.**
6. **Ansible Execution**
   **Ansible restarted the Docker container (docker restart tindog) to restore service availability.**
7. **Docker Compose Orchestration**
   **All services (Prometheus, Alertmanager, Flask webhook, and the custom container) were orchestrated using Docker Compose for seamless deployment.**

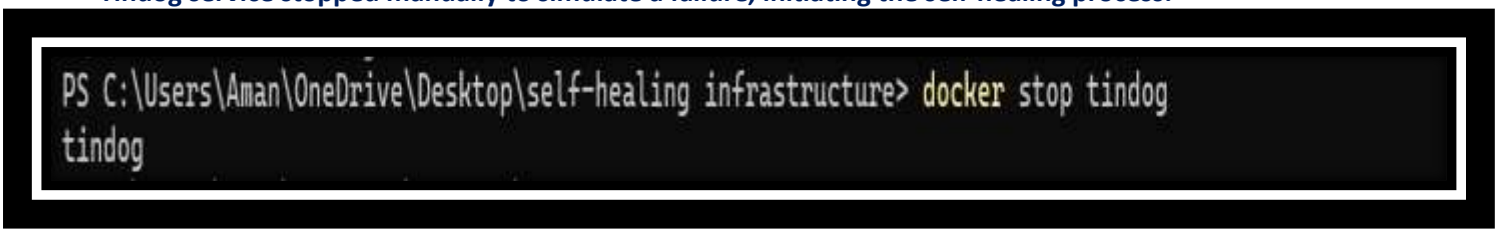
---

**5. Conclusion**

**This project successfully demonstrates a resilient self-healing infrastructure using open-source DevOps tools. The system automatically detects and resolves service downtime without requiring human intervention, enhancing system availability and reliability. The architecture is modular, scalable, and can be extended to monitor and recover various services in real-world production environments.**

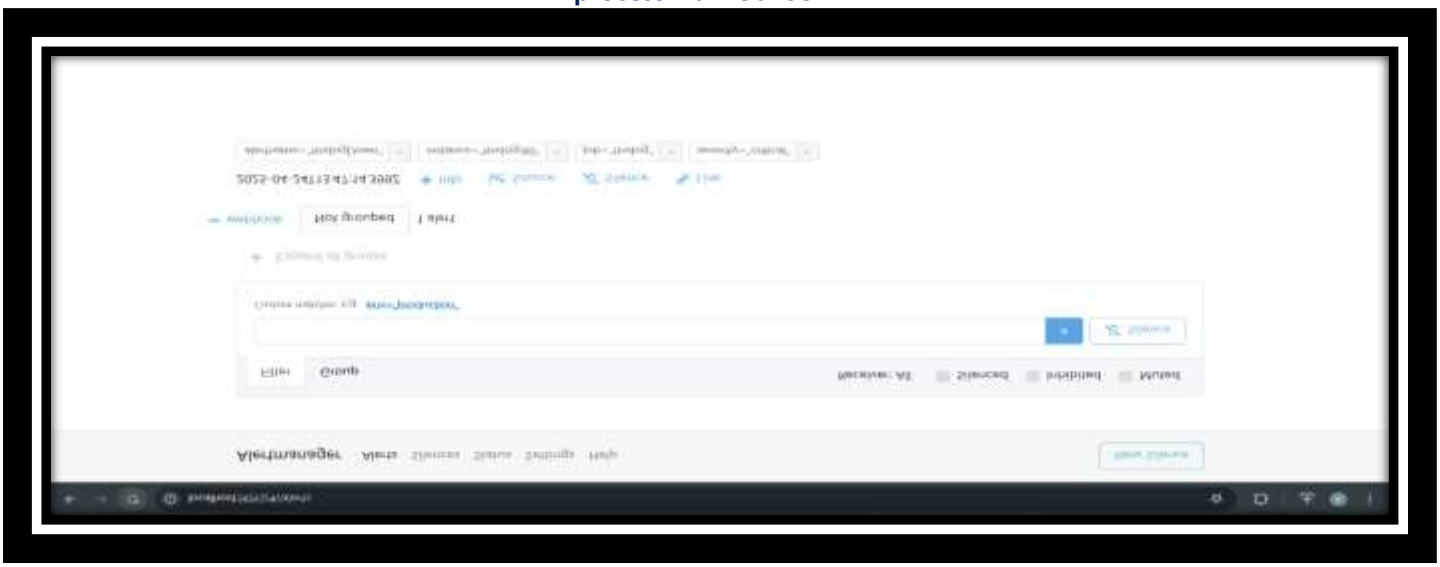
✓ **Prometheus Dashboard showing Inactive status for Tindog, indicating the service is not reporting metrics due to potential downtime or unresponsiveness**

✓ **Tindog service stopped manually to simulate a failure, initiating the self-healing process.**



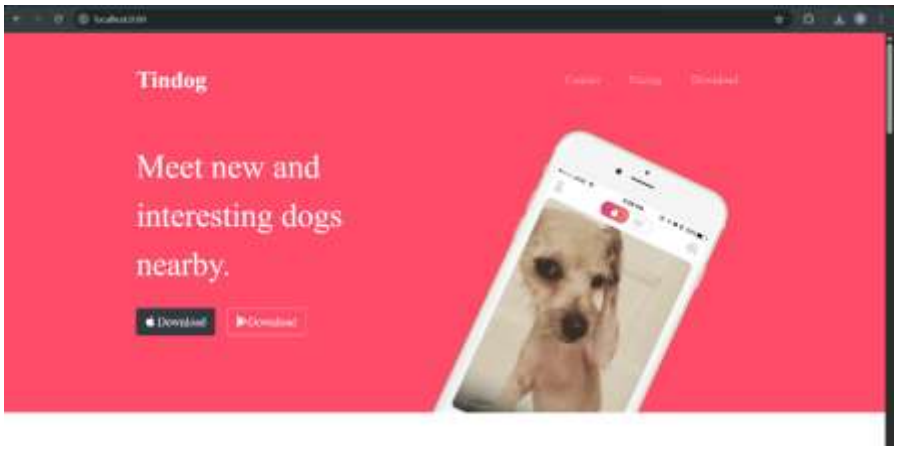

✓ **Prometheus Dashboard showing alert transition from Inactive to Firing, indicating that the Tindog service is down and triggering the alert**

✓ **Alertmanager UI displaying an active alert for NGINXDown with routing configured to trigger the recovery process via webhook**



✓ **Ansible executing the recovery task to restart the NGINX service, ensuring automatic remediation of the failure.**


✓ Tindog service successfully restarted and operational again, completing the self-healing process




***Note : This project was initially built using the official NGINX Docker image to implement and validate the self-healing workflow. Once the basic setup was tested successfully — including monitoring, alerting, and automated recovery — I created and deployed a custom Docker image . This custom image allowed for better control over the environment, such as simulating failures, customizing the index page, and integrating additional logging or status endpoints.***

***Through this project, I gained practical, hands-on experience with:***

- ***Monitoring concepts using Prometheus, including metric scraping and alert rule creation***
- ***Alert management with Alertmanager, including webhook routing and alert templating***
- ***Infrastructure automation using Ansible to trigger service-level actions based on real-time events***
- ***Docker image customization, helping me understand container behavior and build automation-ready images***
- ***Flask-based webhook integration, tying together monitoring and automation seamlessly***
- ***Most importantly, I learned how different DevOps tools can be orchestrated together to create a resilient, self-healing infrastructure — a critical skill in modern cloud-native environments.***