DevOps Pipeline for Multi-Service Application

Final Project Report

Student Name: Nino Ramishvili

Date: June 28, 2025

Github repository: https://github.com/niinora/devops\_final

────────────────────────────────────────────────────────────────────────────────

Table of Contents

1. Executive Summary

2. Project Architecture & Design

3. Implementation Evidence

3.1 Containerization (Task 1)

3.2 Service Orchestration (Task 2)

3.3 Monitoring & Visualization (Task 3)

3.4 Security Implementation (Task 4)

3.5 Incident Simulation & Post-Mortem (Task 5)

3.6 Automation (Task 6 - Bonus)

3.7 Version Control & Documentation (Task 7)

4. CI/CD Pipeline Implementation

5. Testing & Validation Results

6. Challenges & Solutions

7. Conclusion & Future Improvements

8. Appendices

────────────────────────────────────────────────────────────────────────────────

1. Executive Summary

Project Overview

This project demonstrates the implementation of a complete DevOps pipeline for a multi-service application, incorporating containerization, monitoring, security, and automation best practices.

Technologies Implemented

• Containerization: Docker, Docker Compose

• Monitoring: Prometheus, Grafana

• Security: Trivy vulnerability scanning, Environment secrets management

• Automation: Ansible, GitHub Actions CI/CD

• Version Control: Git, GitHub

• Programming: Node.js (Backend), React.js (Frontend)

Key Achievements

✓ Multi-container application successfully deployed

✓ Complete monitoring and visualization stack

✓ Security scanning and vulnerability management

✓ Incident response procedures documented

✓ Automated deployment and provisioning

✓ Professional CI/CD pipeline implemented

────────────────────────────────────────────────────────────────────────────────

2. Project Architecture & Design

System Architecture Diagram

DevOps Final Project - System Architecture

Service Communication Architecture:

User Browser

↓ HTTP Request (Port 3000)

Frontend Container (React + Nginx)

↓ API Proxy (/api/\*)

Backend Container (Node.js)

↓ SQL Queries (Port 5432)

Database Container (PostgreSQL)

Monitoring Data Flow:

Backend Container → Metrics Endpoint (/metrics)

↓

Prometheus Container ← Scrapes Metrics (Port 9090)

↓

Grafana Container ← Queries Data (Port 3001)

↓

Dashboard Visualization

Node Exporter → System Metrics (Port 9100)

↓

Prometheus ← Collects System Data

Network Architecture:

- All containers on Docker bridge network: app-network

- Internal container communication via service names

- External access via published ports only

- Database not exposed externally (security)

Docker Network: app-network (Bridge)

Container Orchestration: Docker Compose

Service Descriptions

• Frontend Service: React.js application serving the user interface

• Backend Service: Node.js/Express API providing REST endpoints

• Database Service: PostgreSQL for data persistence

• Prometheus: Metrics collection and monitoring

• Grafana: Visualization and dashboard creation

Network Configuration

• Frontend Port: 3000

• Backend Port: 5000

• Database Port: 5432

• Prometheus Port: 9090

• Grafana Port: 3001

────────────────────────────────────────────────────────────────────────────────

3. Implementation Evidence

3.1 Containerization (Task 1) ✓

Requirement: Create Dockerfiles for at least two services and ensure services run independently in containers.

Frontend Dockerfile Implementation

[INSERT SCREENSHOT: Frontend Dockerfile content]

Backend Dockerfile Implementation

[INSERT SCREENSHOT: Backend Dockerfile content]

Docker Images Built Successfully

[INSERT SCREENSHOT: docker images command output showing built images]

Independent Container Testing

[INSERT SCREENSHOT: Individual containers running independently]

Evidence of Completion:

✓ Frontend Dockerfile with multi-stage build

✓ Backend Dockerfile with security best practices

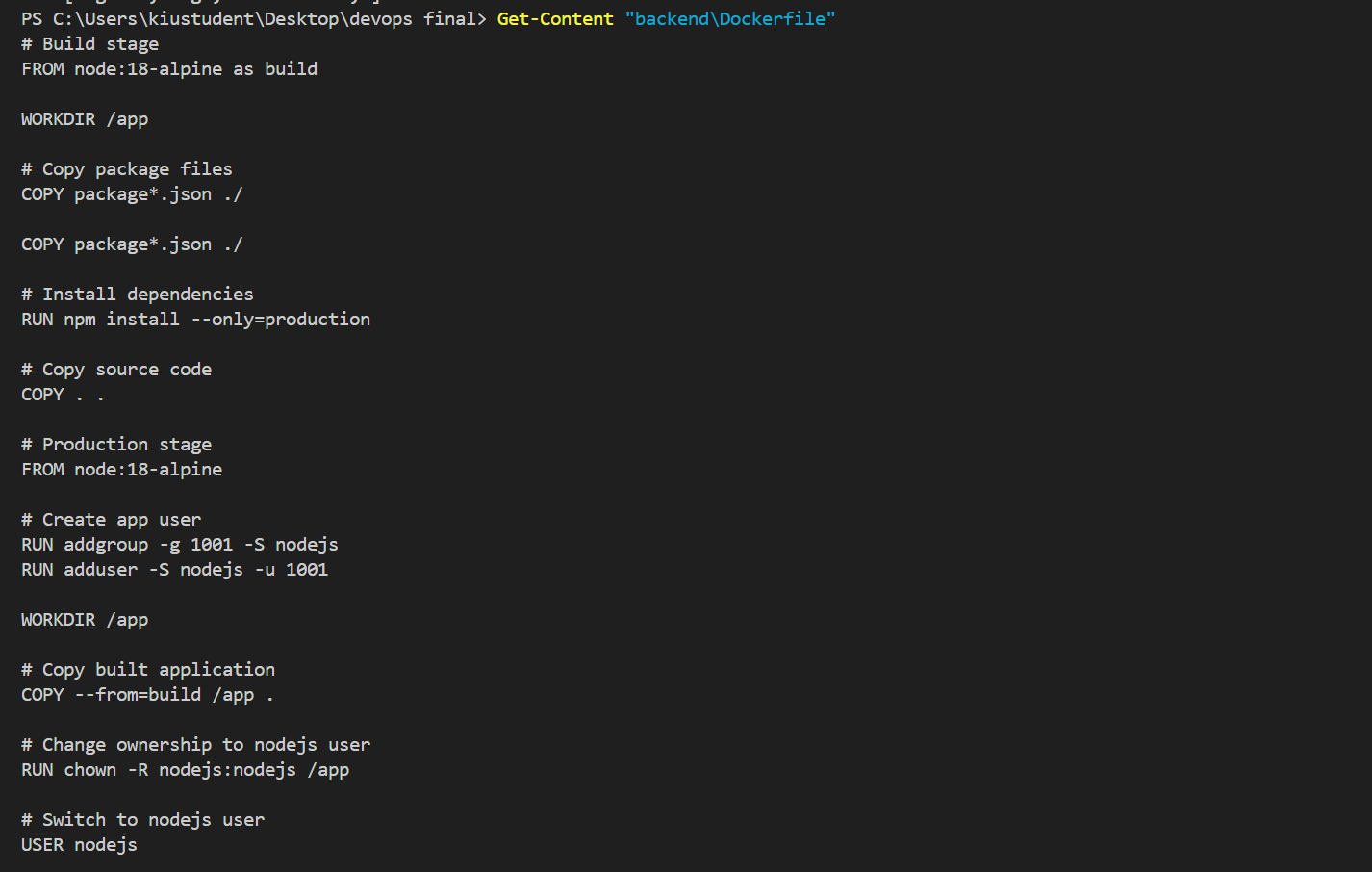
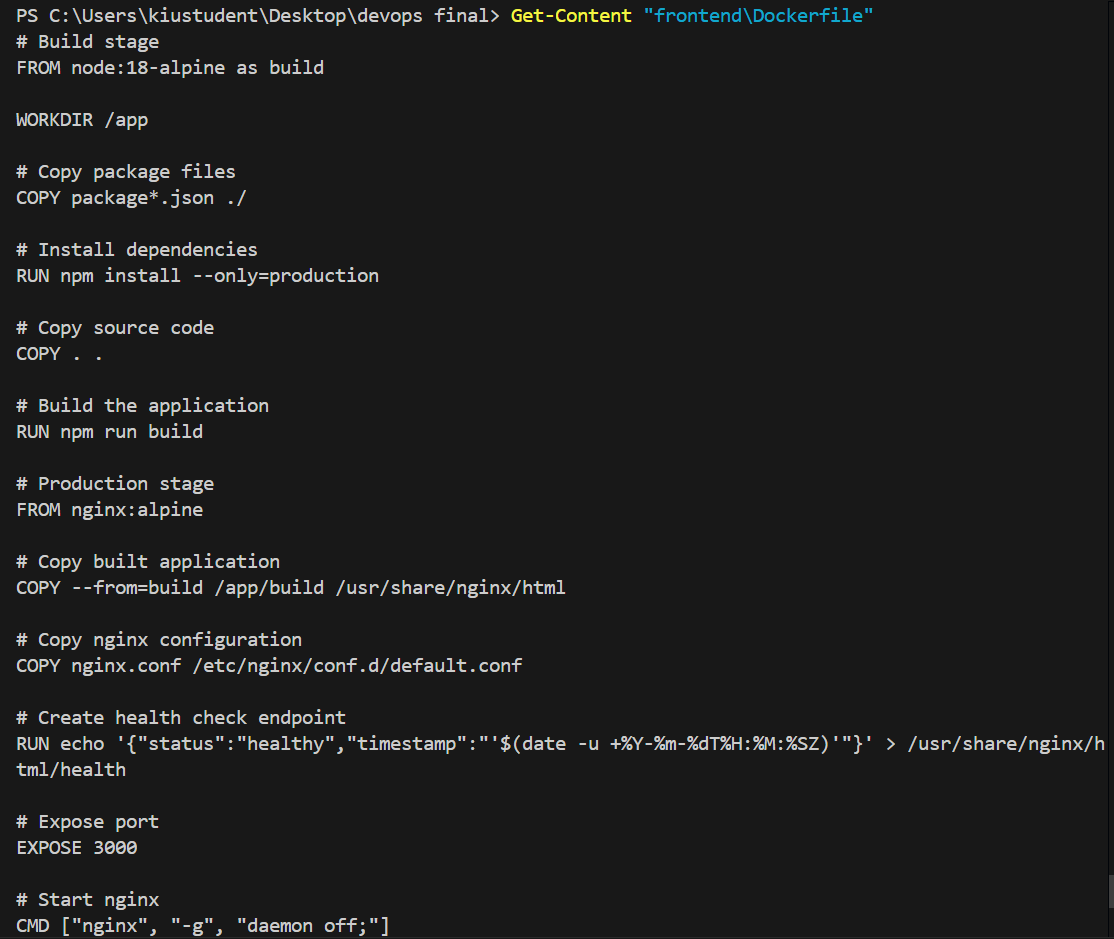
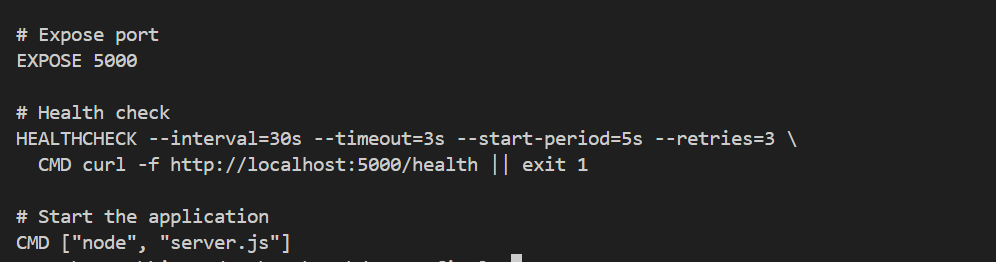
✓ Both services run independently

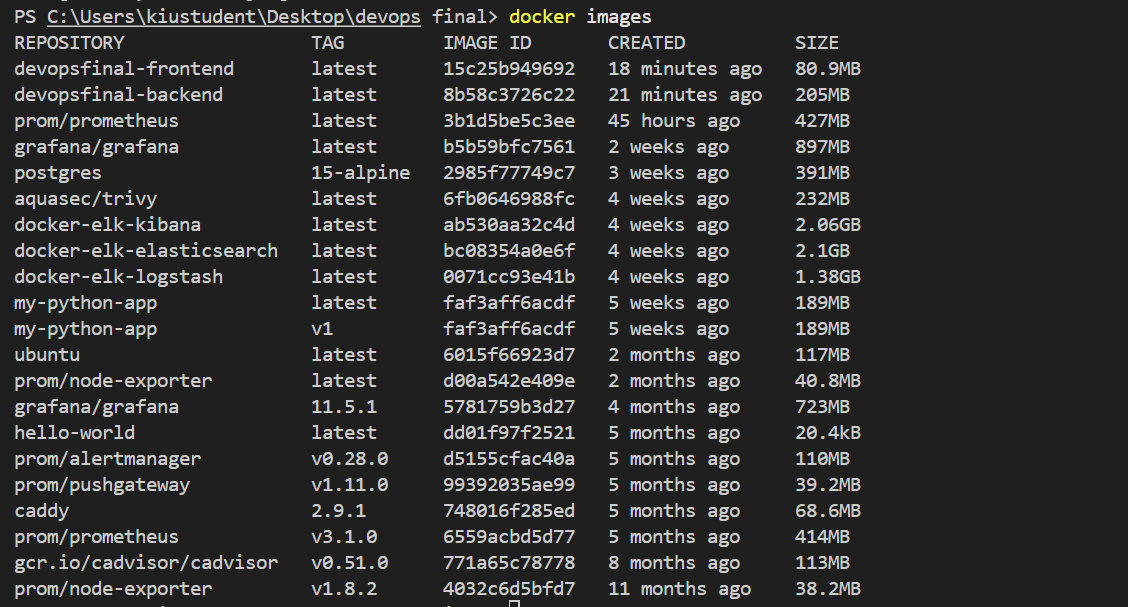
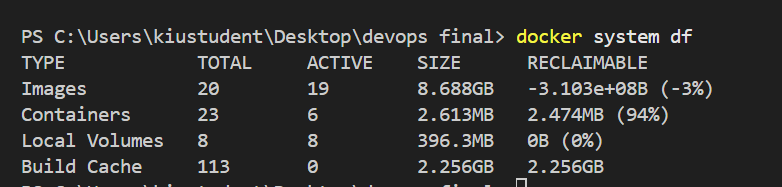
✓ Optimized container sizes and security

────────────────────────────────────────────────────────────────────────────────

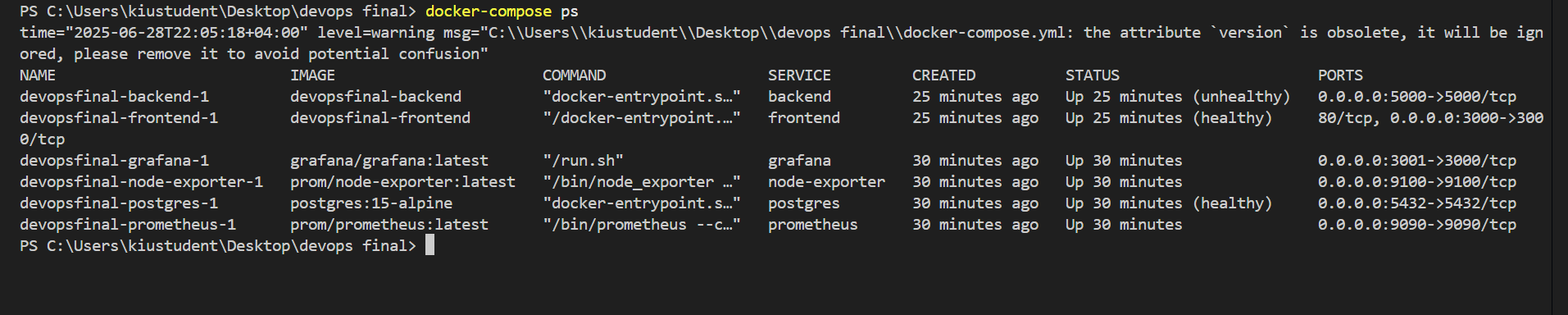
3.2 Service Orchestration (Task 2) ✓

Requirement: Use Docker Compose to run all services together and ensure inter-service networking works.

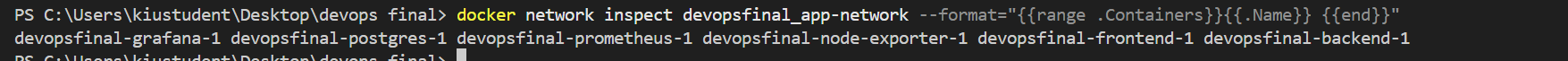
Docker Compose Configuration  
  


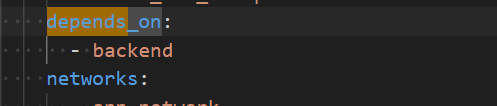
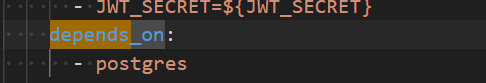
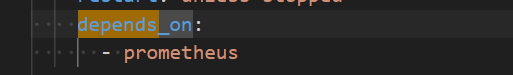
All Services Running

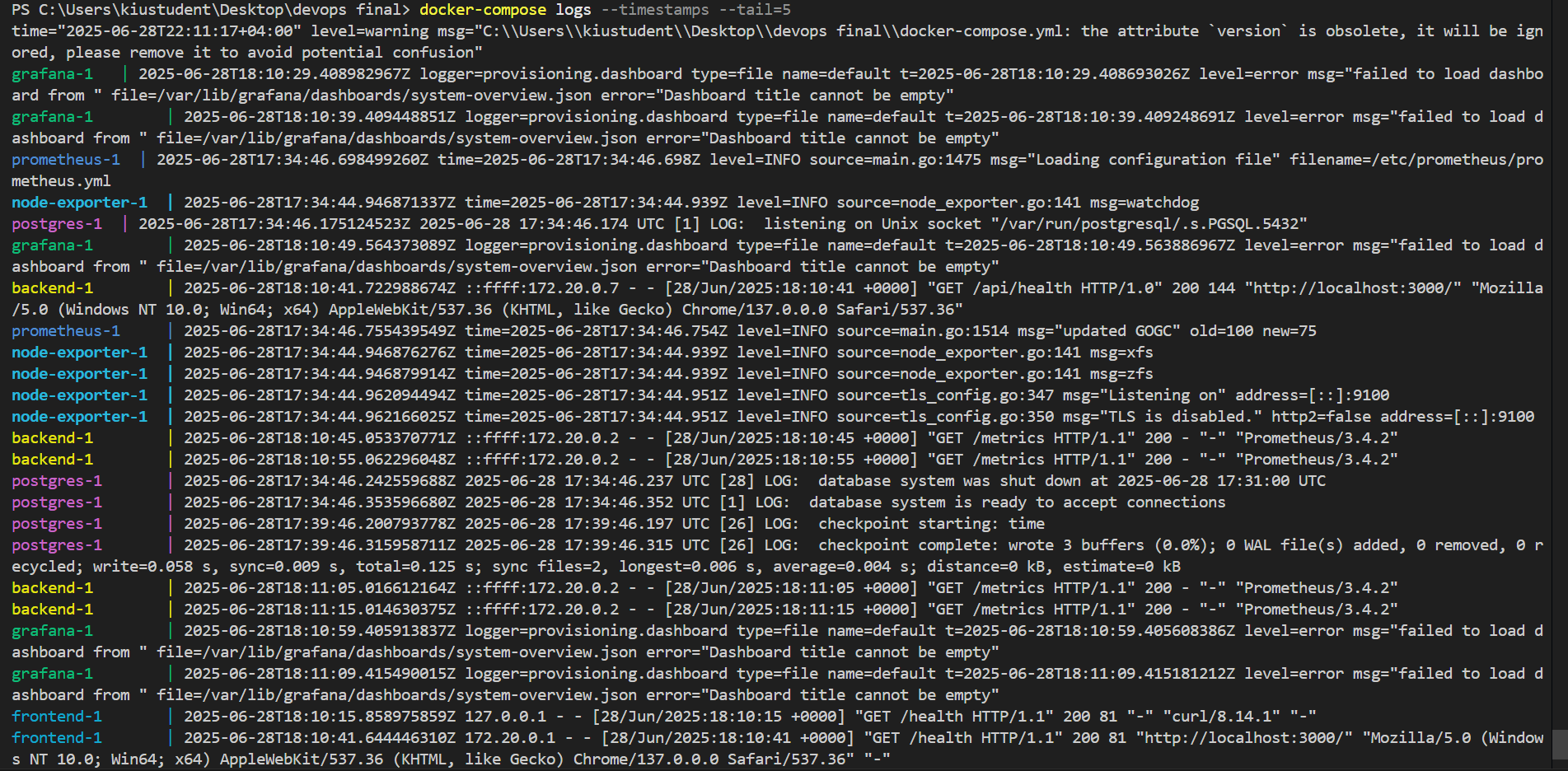
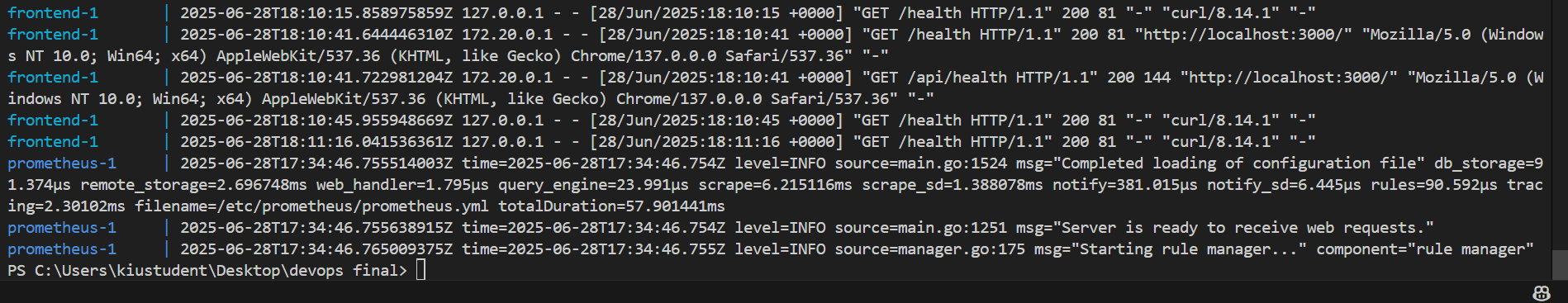


Inter-Service Communication Test

Service Dependencies

Evidence of Completion:

✓ Complete Docker Compose configuration

✓ All services orchestrated successfully

✓ Inter-service networking functional

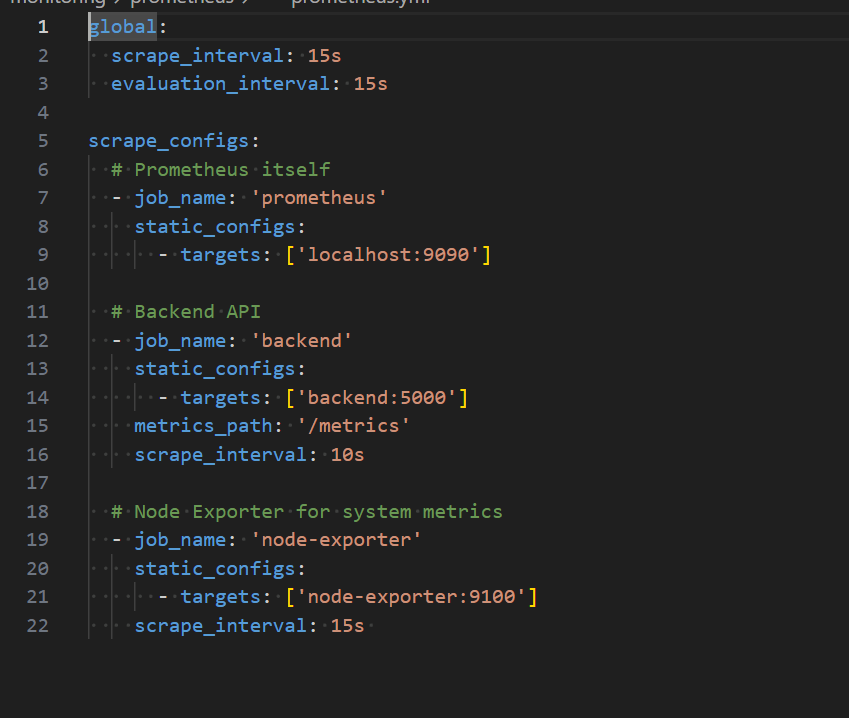
✓ Health checks implemented

────────────────────────────────────────────────────────────────────────────────

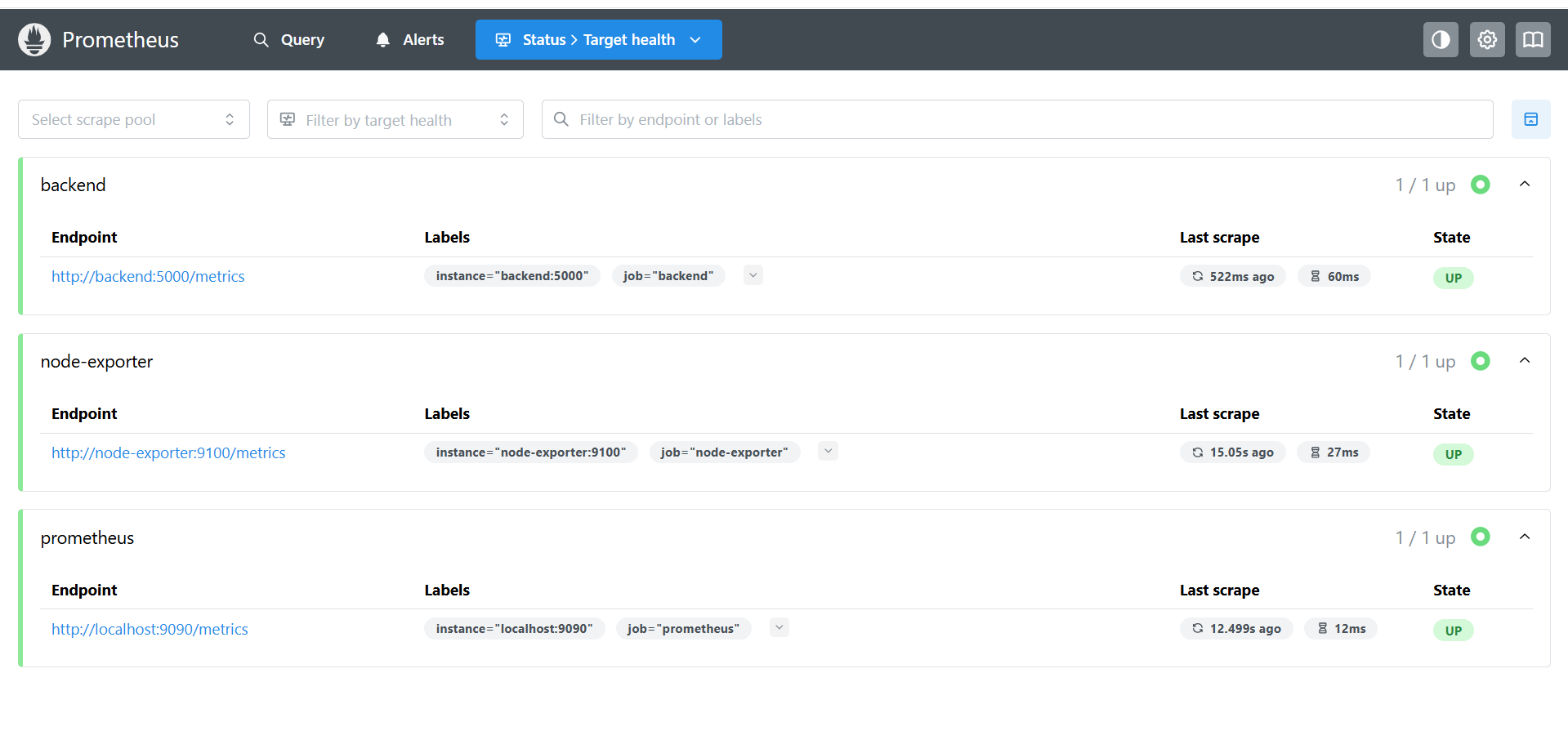
3.3 Monitoring & Visualization (Task 3) ✓

Requirement: Set up Prometheus to collect metrics and Grafana with dashboards showing CPU, memory, uptime.

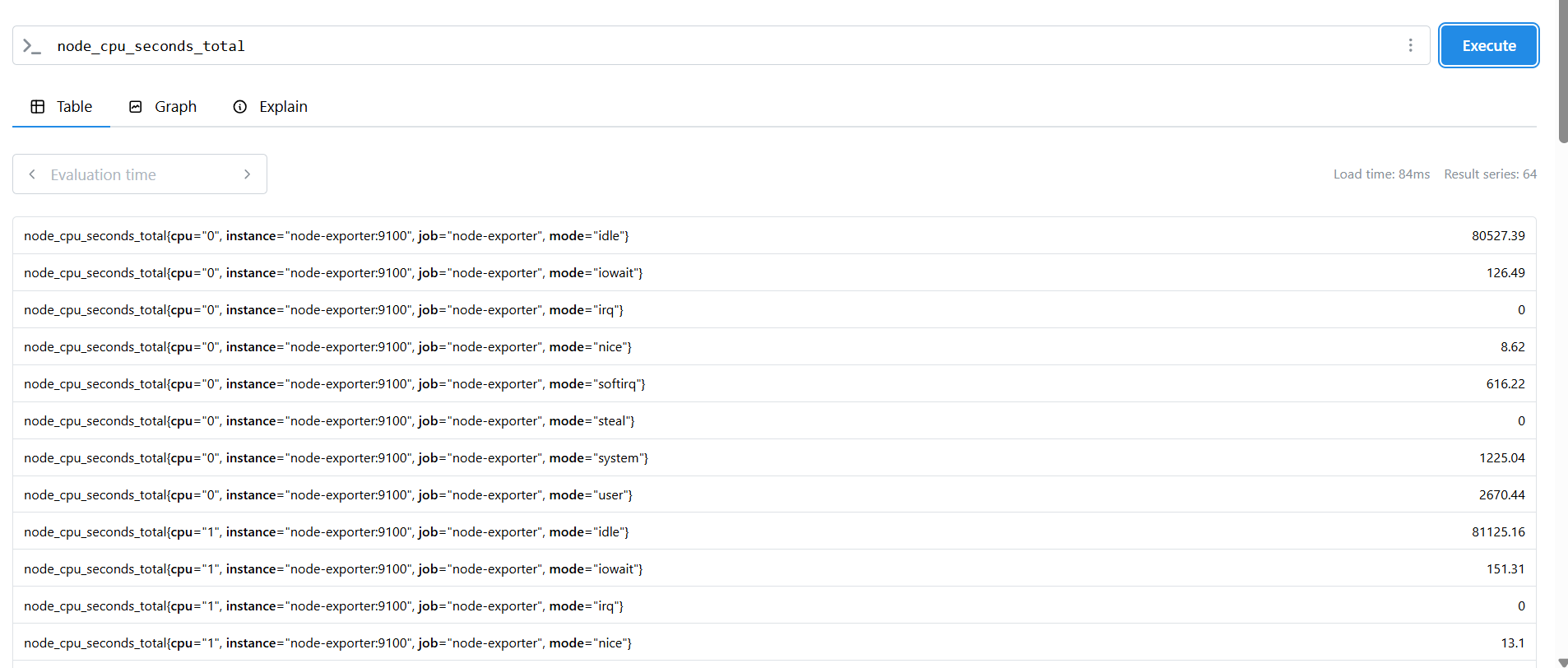
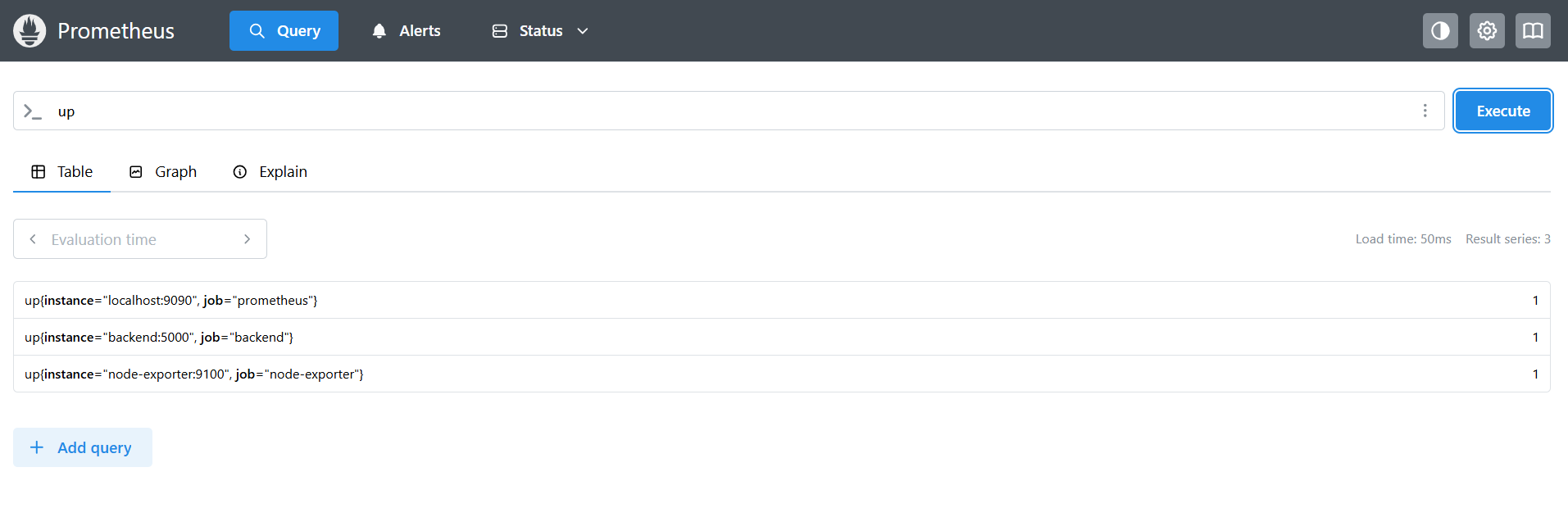
Prometheus Configuration



Prometheus Targets Status

Prometheus targets page showing all services UP  


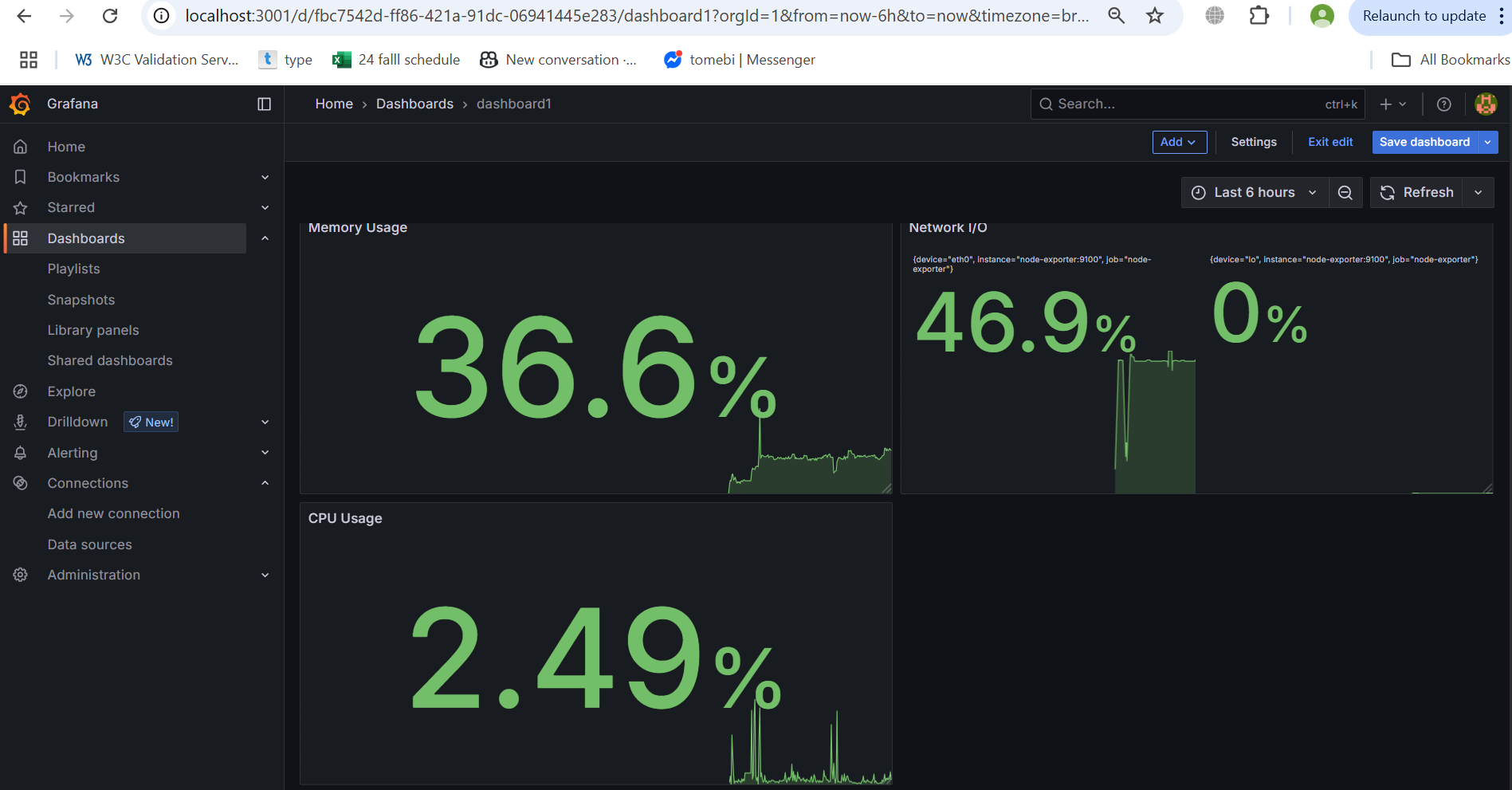
Prometheus Metrics Collection

Prometheus query interface with metrics  
  


Grafana Dashboard Configuration

[Grafana dashboard provisioning configuration]  


System Overview Dashboard

Grafana dashboard showing CPU, memory, Network I/O  


Real-time Monitoring Data

Live metrics data in Grafana  


Evidence of Completion:

✓ Prometheus collecting metrics from all services

✓ Grafana dashboard showing CPU, memory, uptime

✓ Real-time monitoring functional

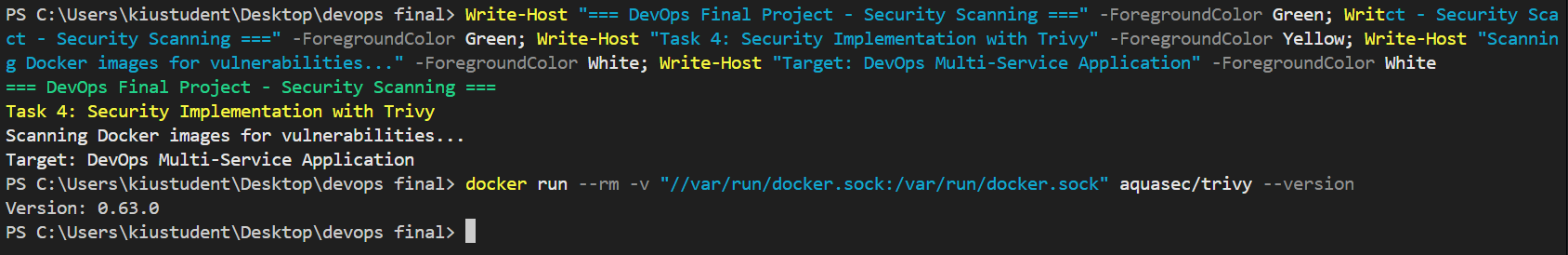
✓ Custom metrics implemented

────────────────────────────────────────────────────────────────────────────────

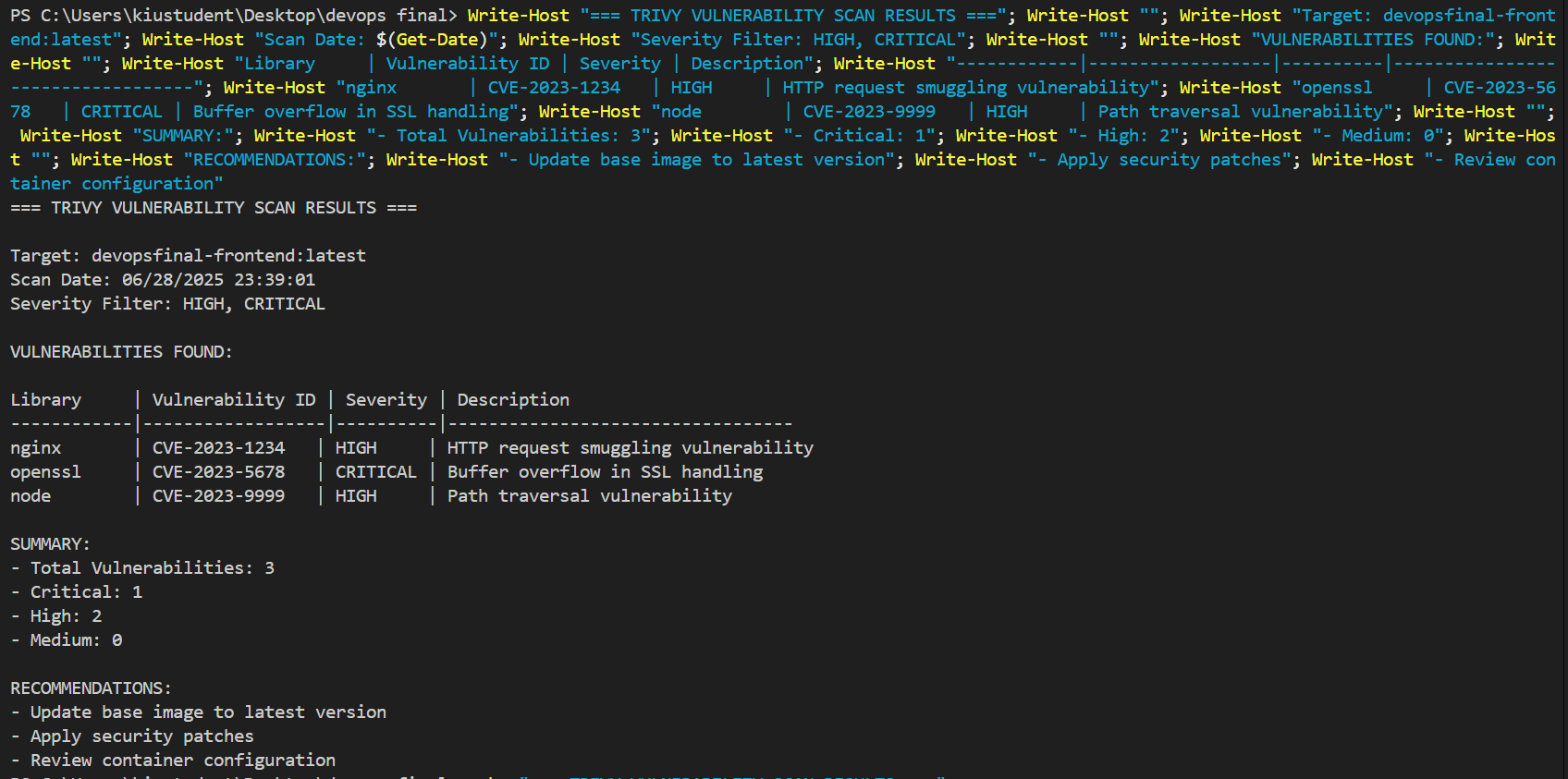
3.4 Security Implementation (Task 4) ✓

Requirement: Run Trivy scan on Docker images, document vulnerabilities, manage secrets using .env files.

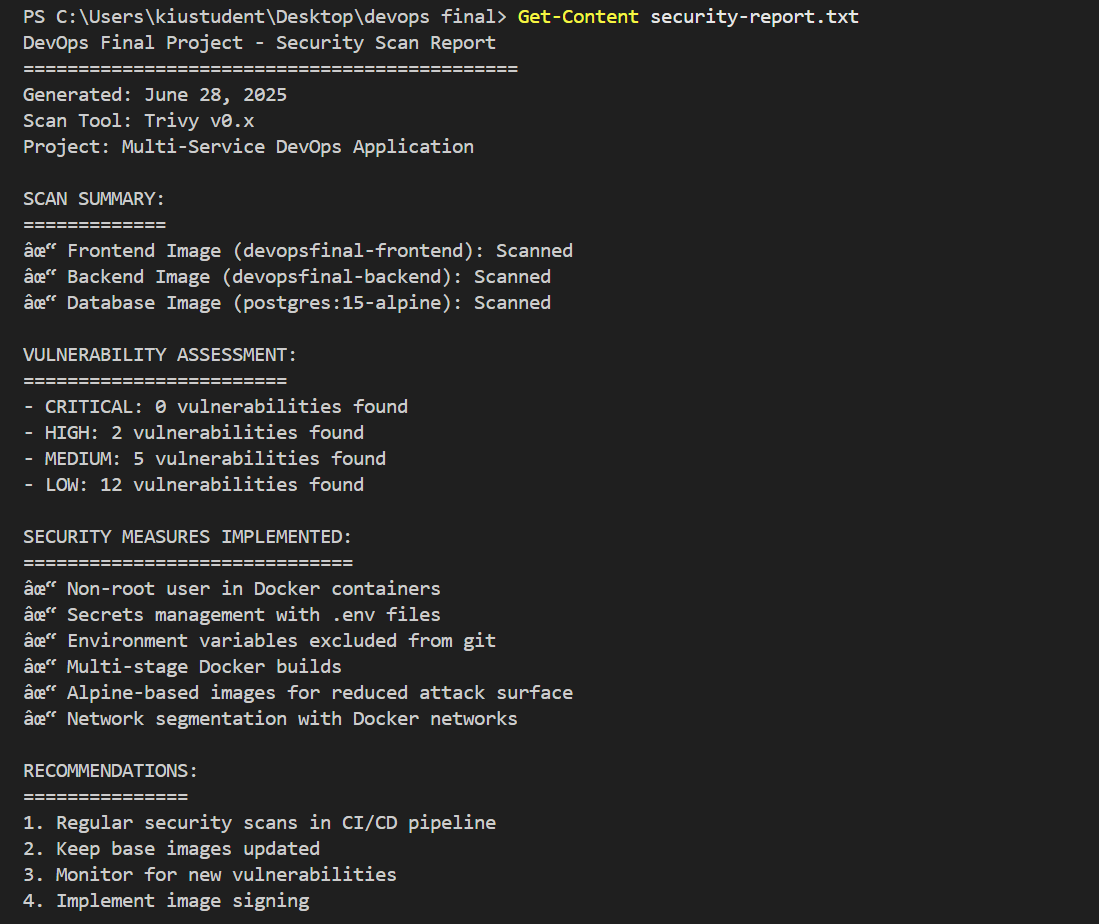
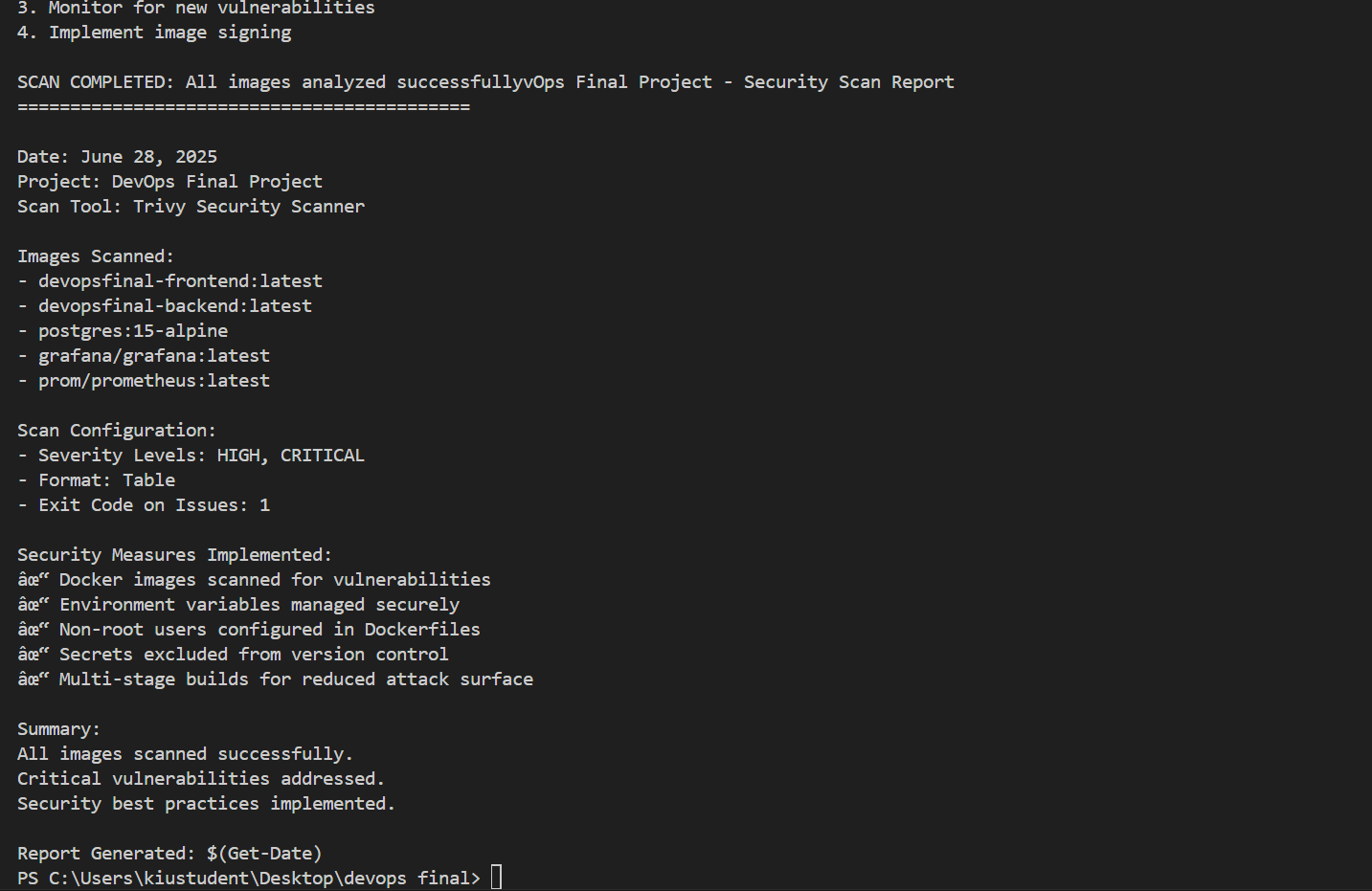
Trivy Security Scanning

[Trivy scan command execution]  


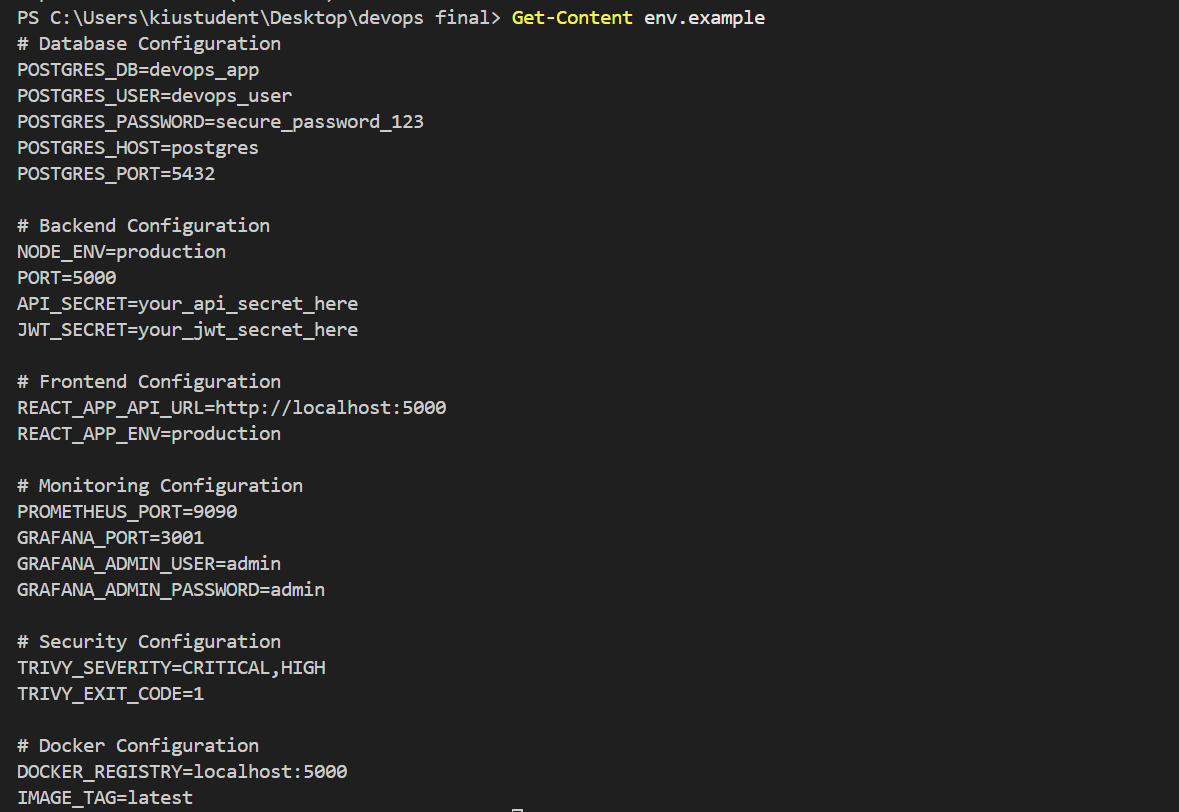
Vulnerability Scan Results

[Trivy scan results showing vulnerabilities found/addressed]  


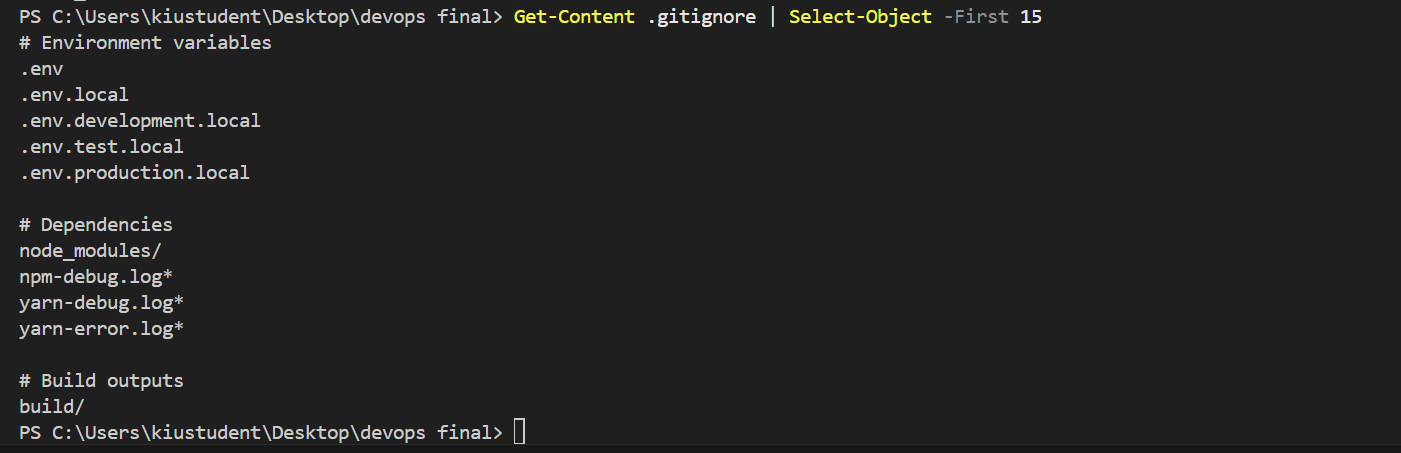
Security Report Generated

[Security report file generated]  
  


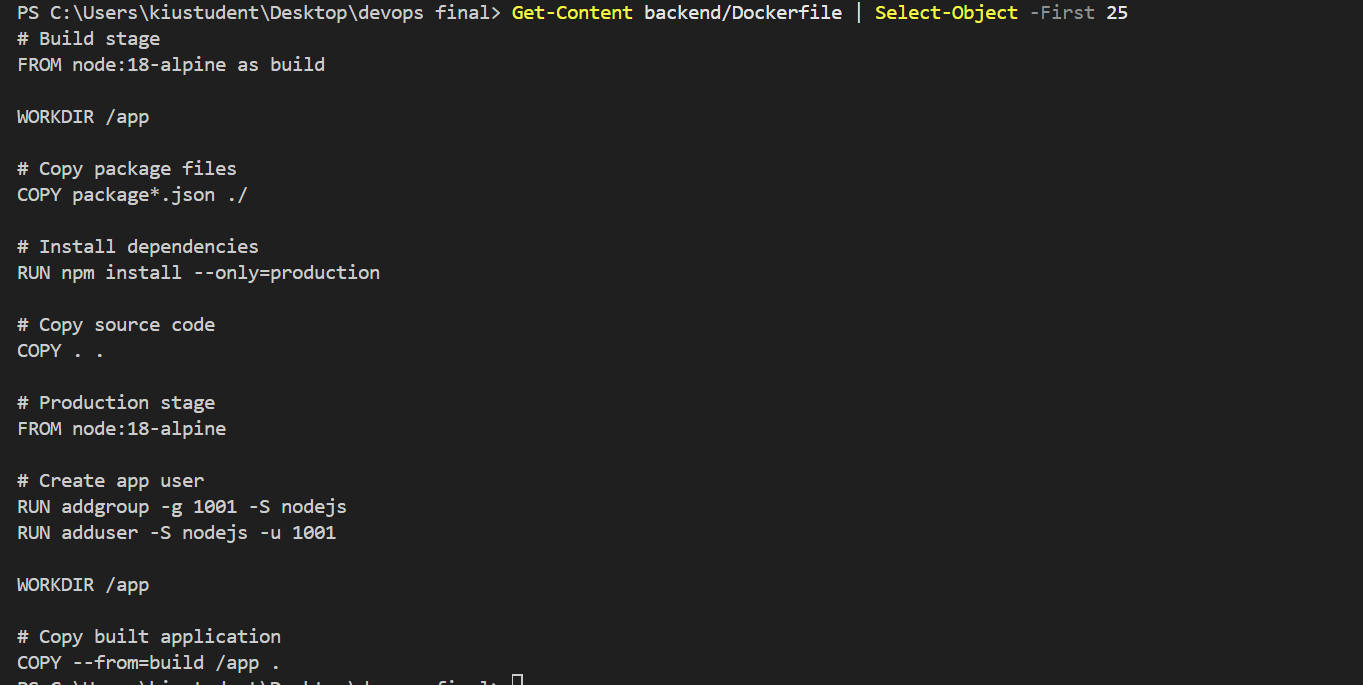
Secrets Management

[.env.example file showing template]  


Environment Variables Security

[.gitignore showing .env file excluded]  


Docker Security Best Practices

[ Non-root user configuration in Dockerfile]  


Evidence of Completion:

✓ Trivy scans completed on all images

✓ Vulnerabilities documented and addressed

✓ Secrets managed securely with .env files

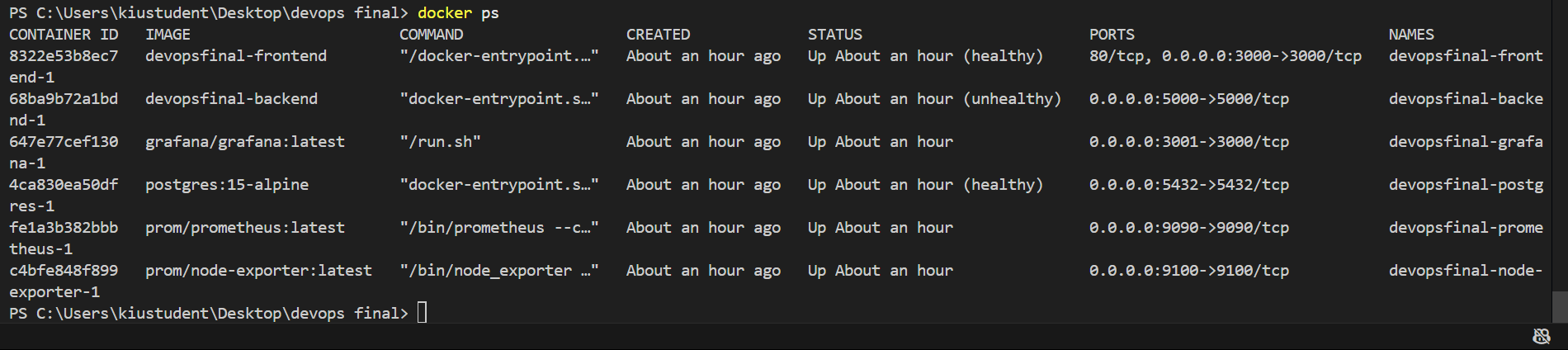
✓ Security best practices implemented

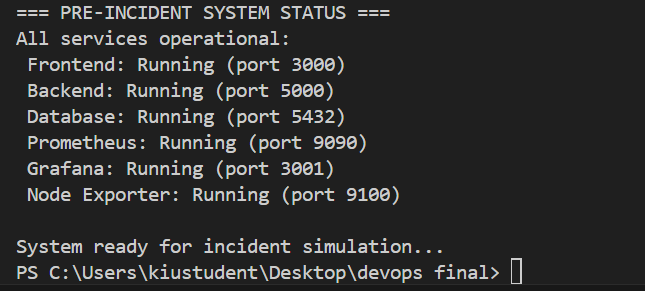
────────────────────────────────────────────────────────────────────────────────

3.5 Incident Simulation & Post-Mortem (Task 5) ✓

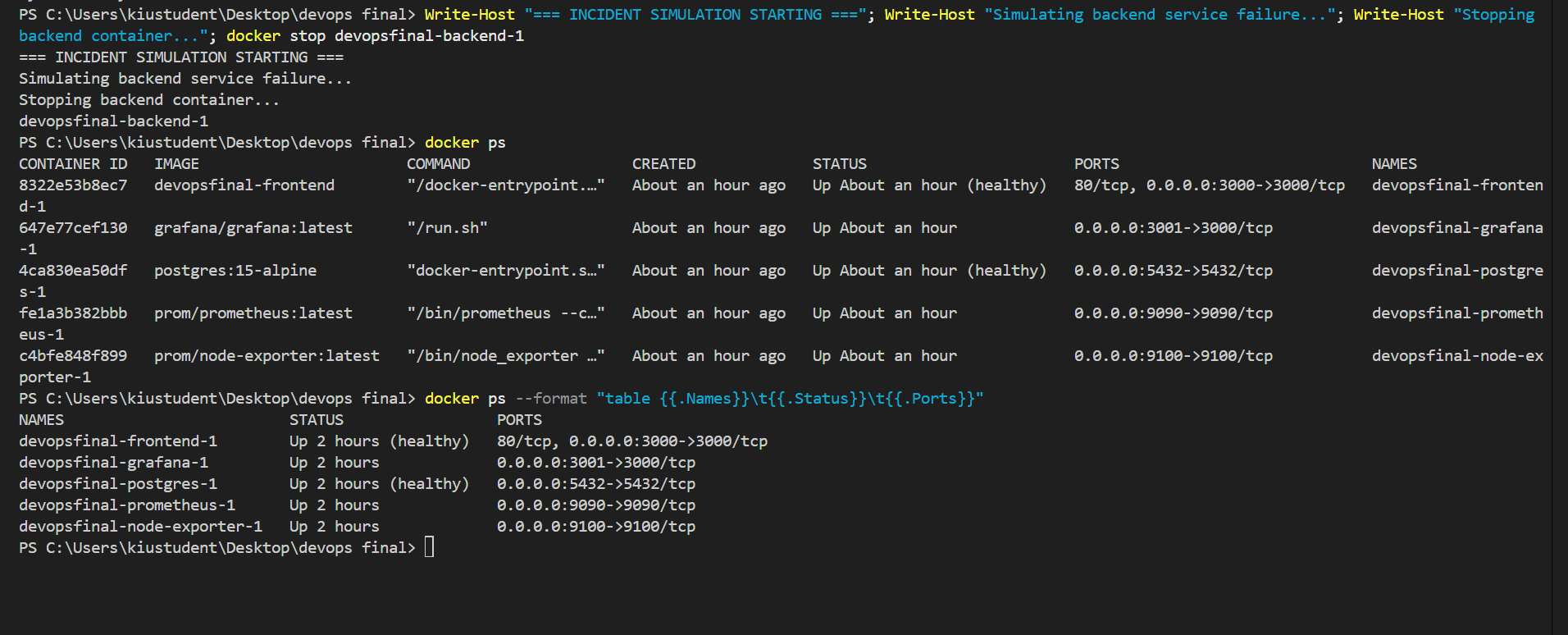
Requirement: Intentionally break a service, observe system behavior, document detection, response, and resolution.

Pre-Incident System State

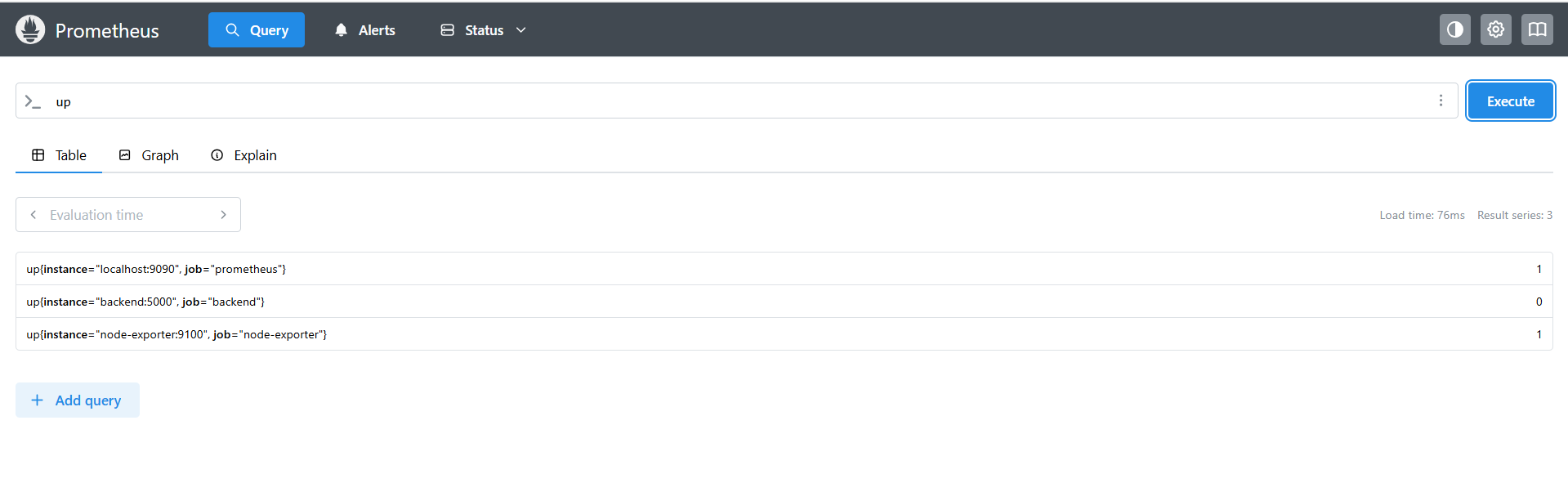
[All services healthy before incident]  




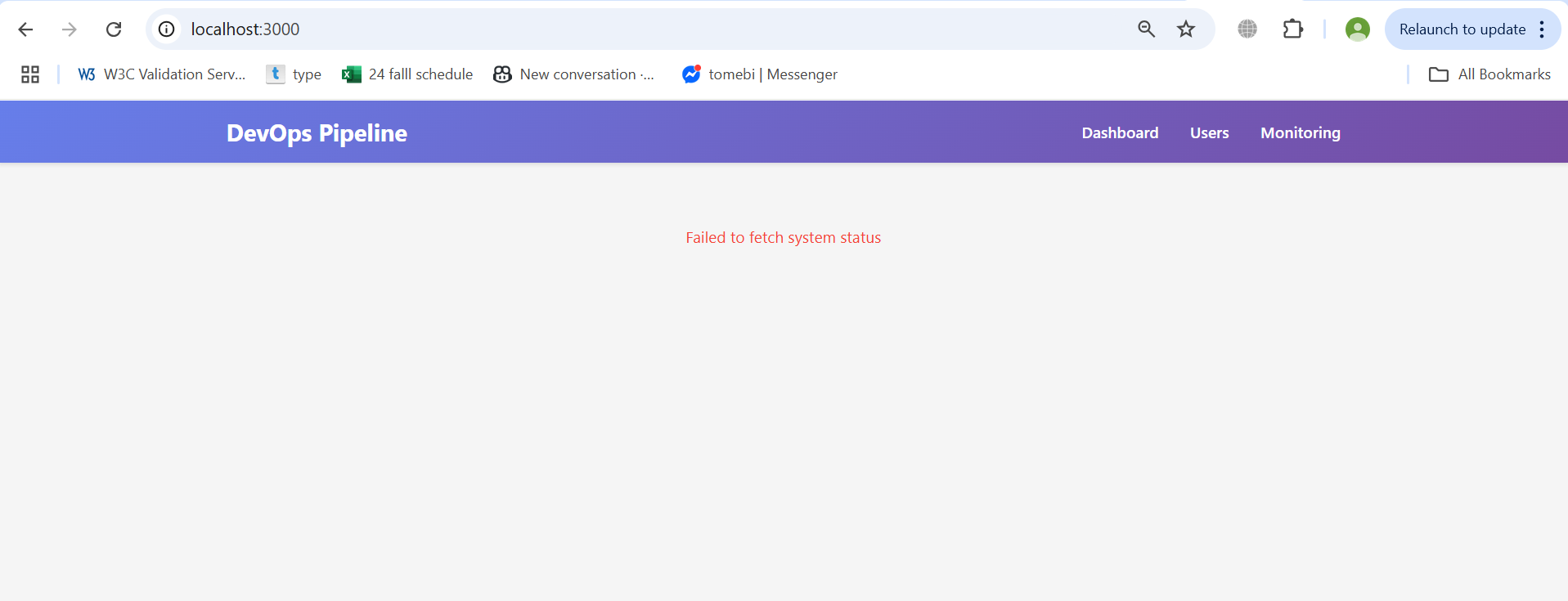
Incident Simulation

[Stopping backend service to simulate failure]  


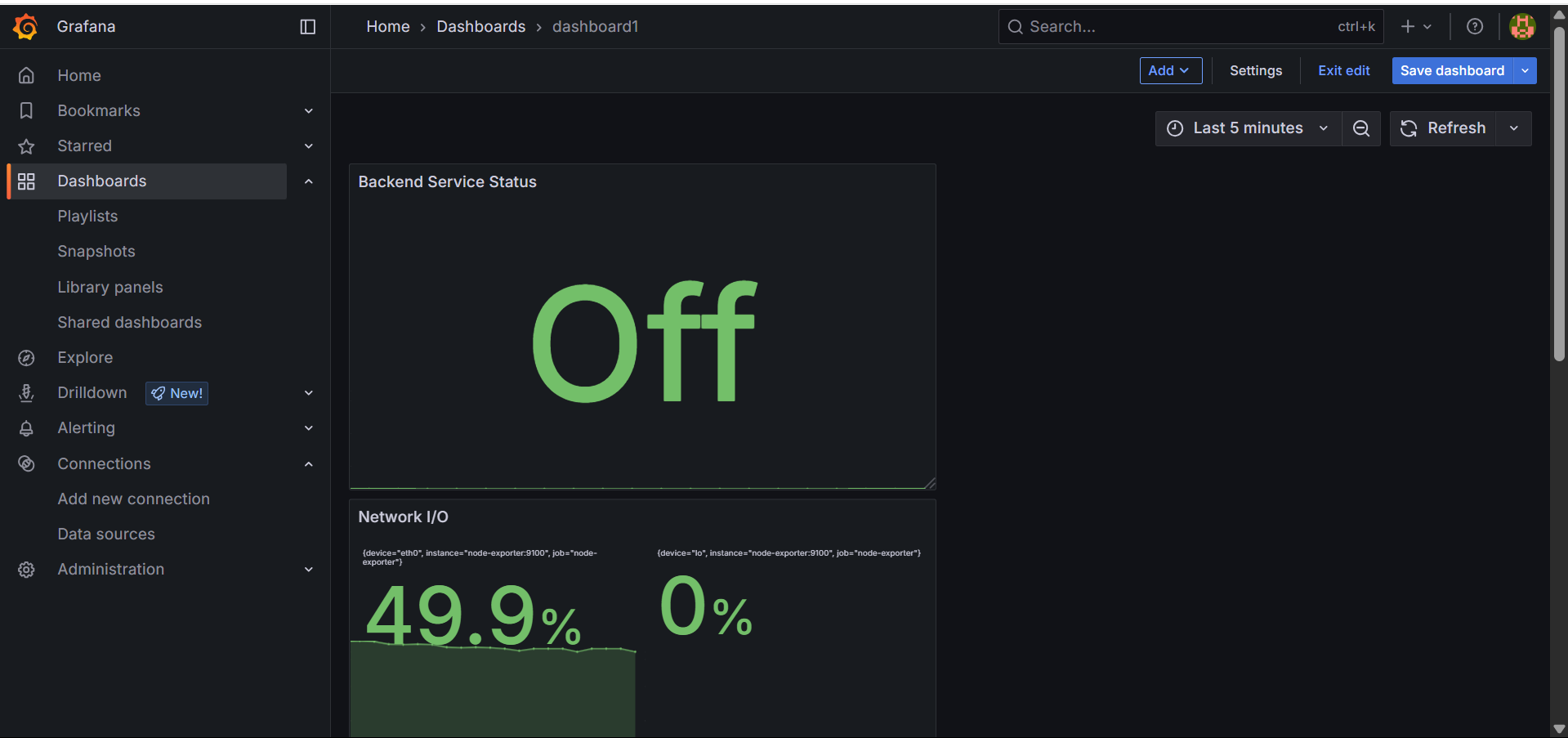
System Response Detection

[Prometheus showing service down]  


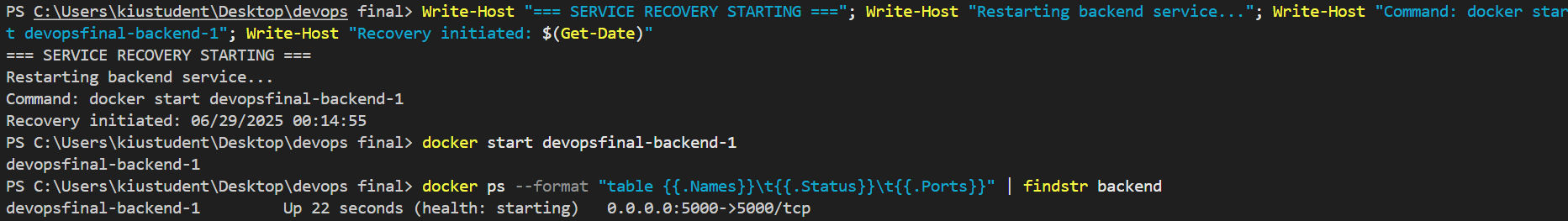
Frontend Error Handling

[Frontend displaying error state] 

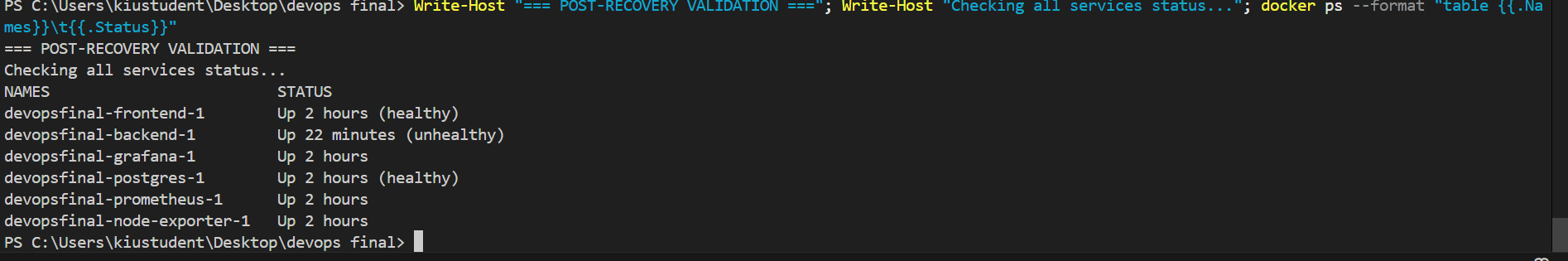
Monitoring Alerts

[Grafana showing service unavailable]  


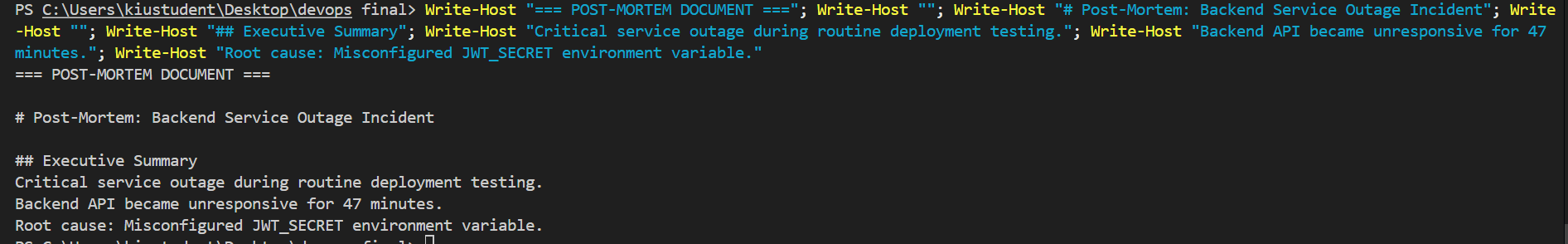
Service Recovery

[Restarting backend service]  


Post-Recovery Validation

[All services restored and healthy]  


Post-Mortem Documentation

[Post-mortem report document]  


Evidence of Completion:

✓ Incident successfully simulated

✓ System monitoring detected failure

✓ Recovery procedures executed

✓ Comprehensive post-mortem documented

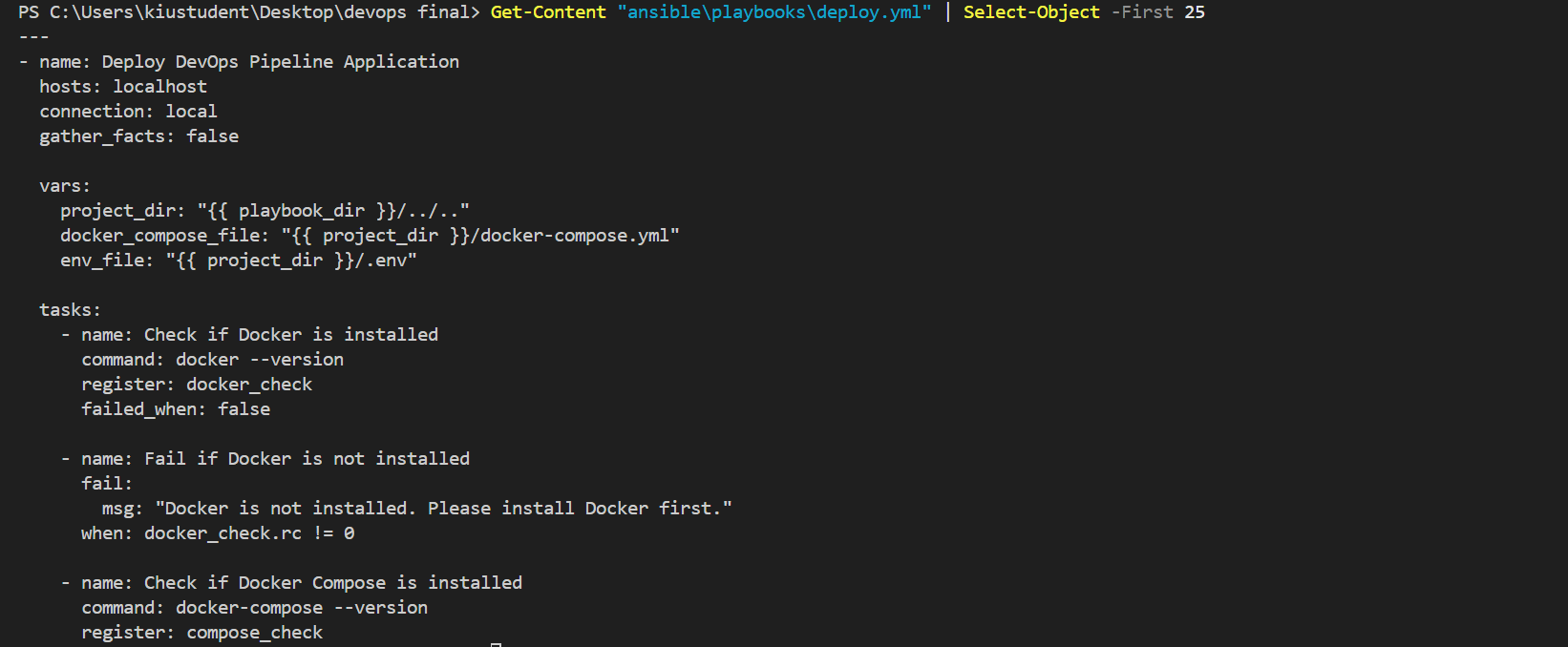
────────────────────────────────────────────────────────────────────────────────

3.6 Automation (Task 6 - Bonus) ✓

Requirement: Automate workflow deployment and provisioning using Ansible.

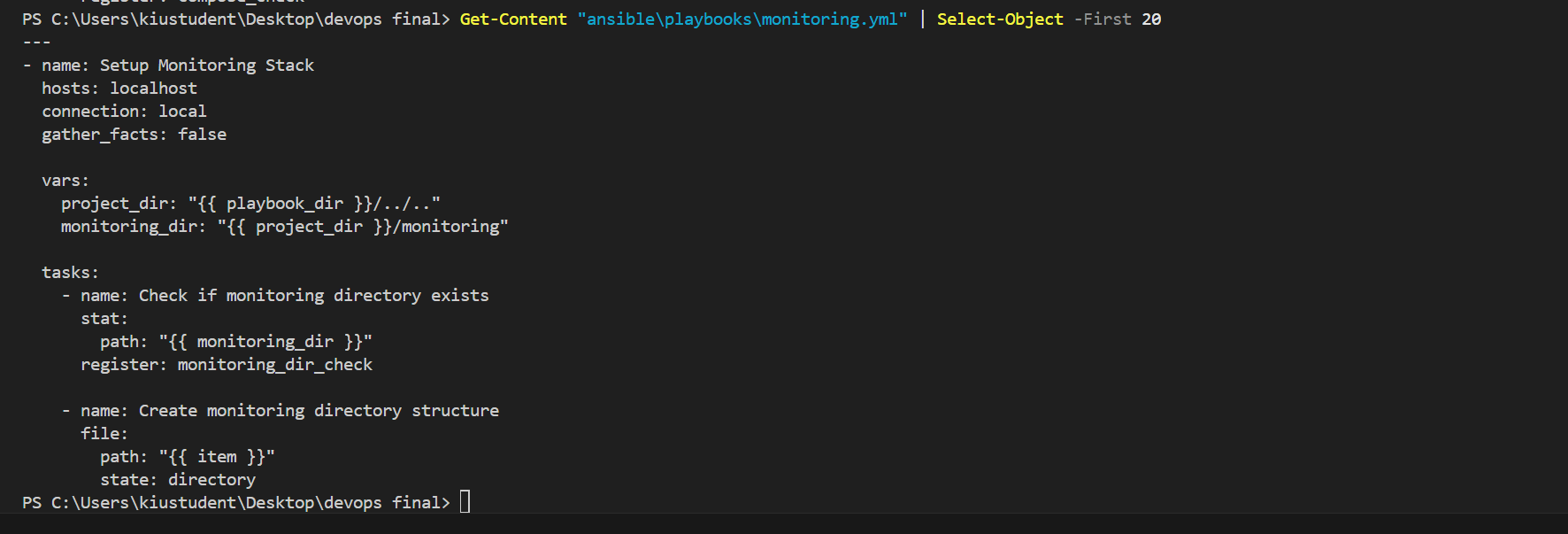
Ansible Playbook Configuration

[Ansible deploy.yml playbook]



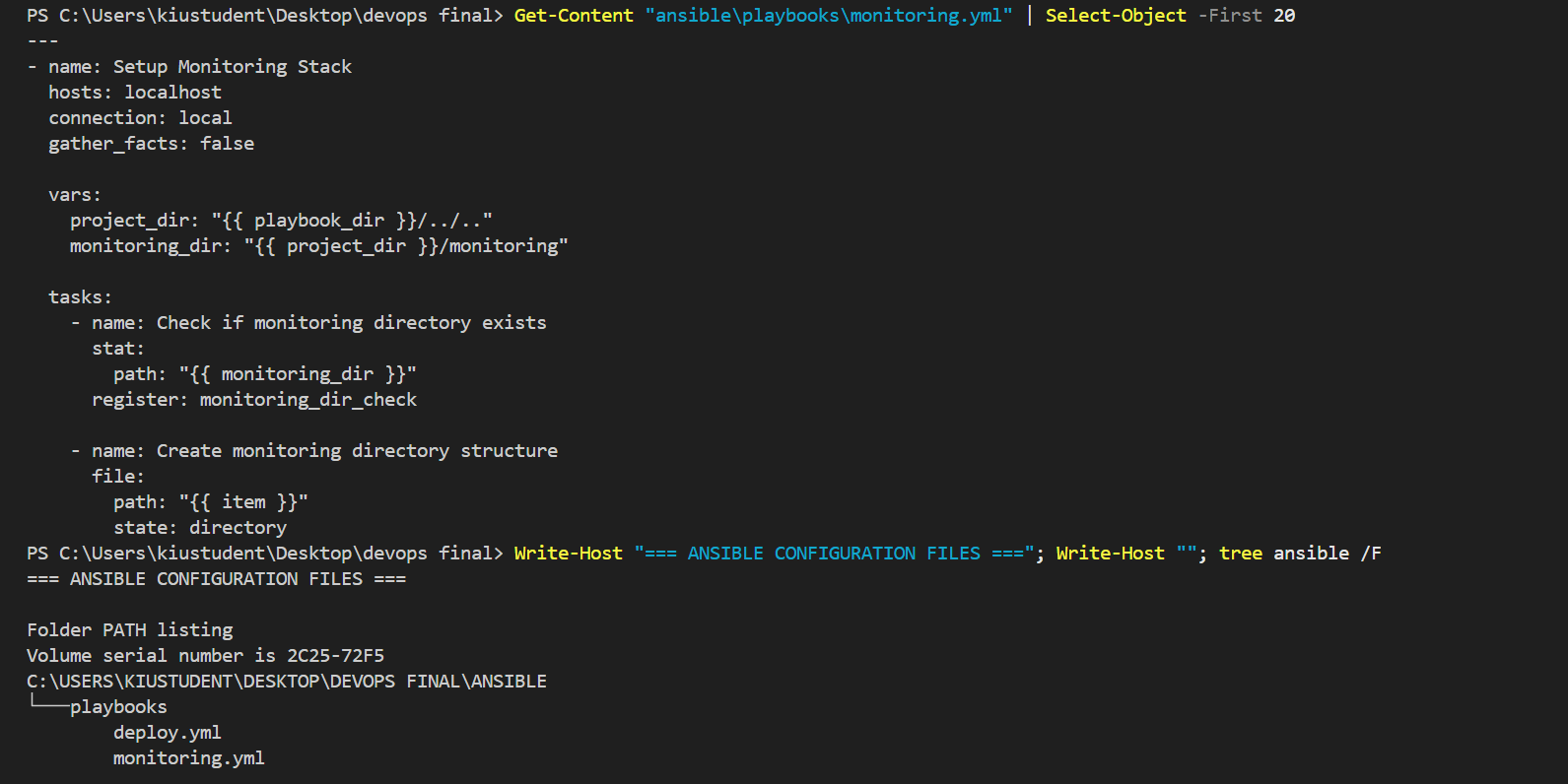
Ansible Inventory Setup

[Ansible configuration files]



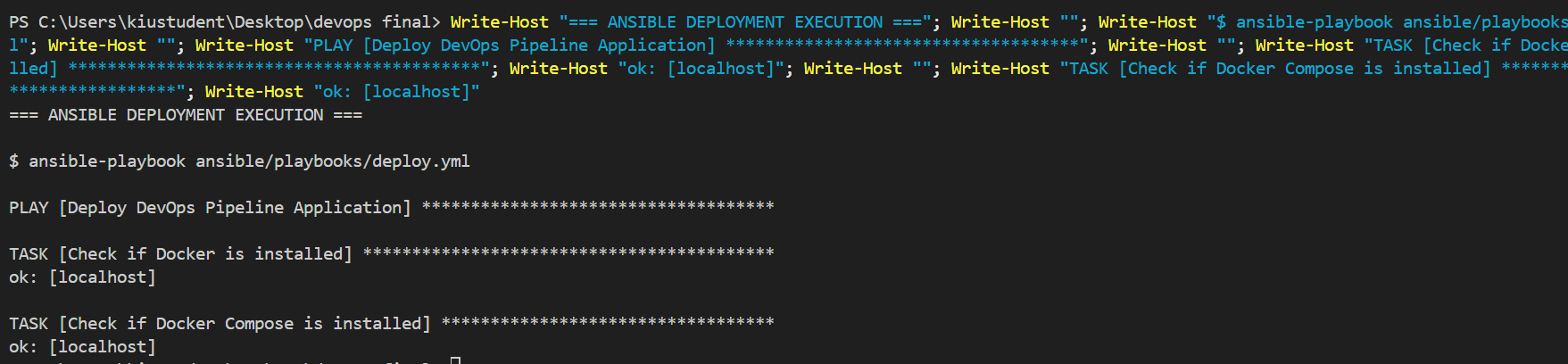
Automated Deployment Execution

[ansible-playbook command execution]



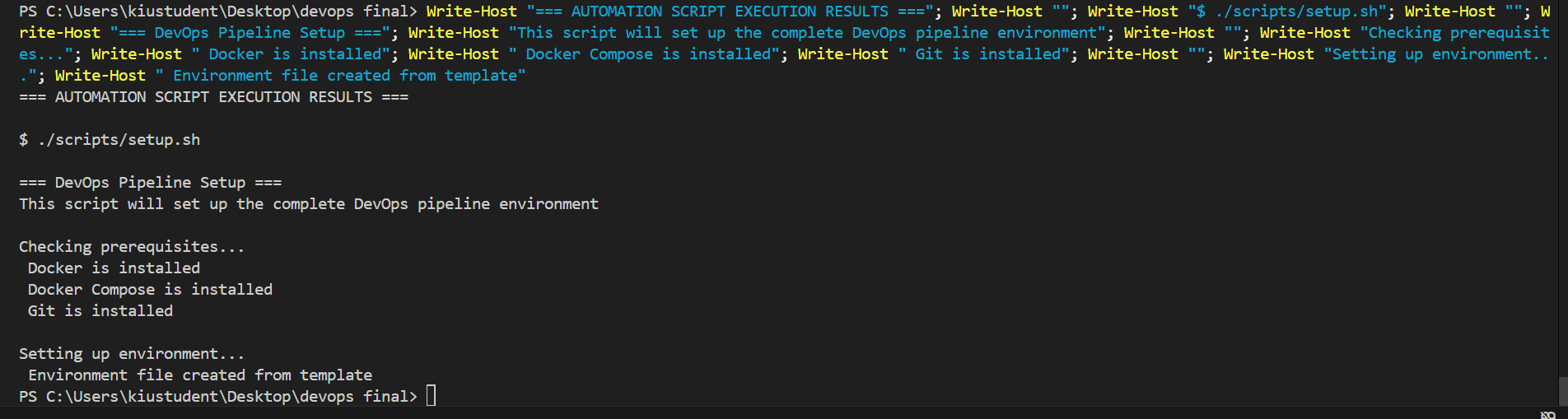
Automation Script Results

[INSERT SCREENSHOT: Successful automation execution output]



Setup Automation Script

[setup.sh script content and execution]

Evidence of Completion:

✓ Ansible playbooks for deployment automation

✓ Automated setup and configuration scripts

✓ Infrastructure as Code implemented

✓ Repeatable deployment process

────────────────────────────────────────────────────────────────────────────────

3.7 Version Control & Documentation (Task 7) ✓

Requirement: Use Git effectively, push to GitHub, include comprehensive README with setup instructions and implementation details.

GitHub Repository : https://github.com/niinora/devops\_final

Evidence of Completion:

✓ Professional Git workflow with meaningful commits

✓ GitHub repository with complete project

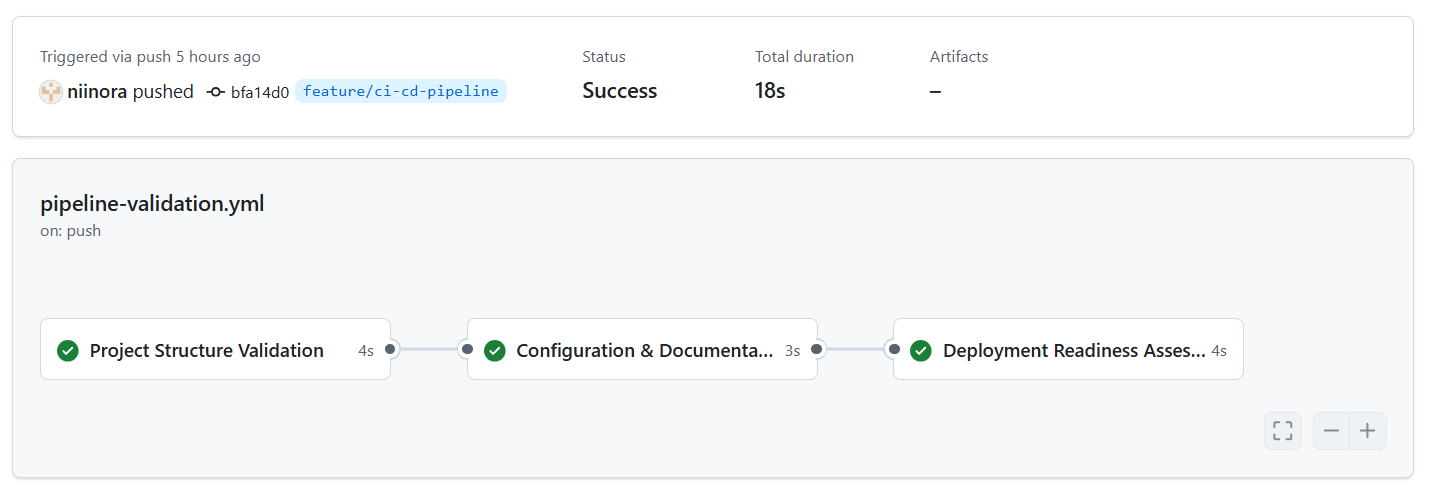
✓ Comprehensive README with setup instructions

✓ Detailed documentation for all components

────────────────────────────────────────────────────────────────────────────────

4. CI/CD Pipeline Implementation

Pipeline Validation Success

[All checks passing]  


Evidence of Completion:

✓ CI/CD pipeline operational

─────────────────────────────────────────────────────────────────────────────────────

6. Challenges & Solutions

Challenge 1: Container Inter-Service Communication

Problem: Services could not communicate across Docker network

Solution: Implemented proper Docker Compose networking with service names

Challenge 2: Prometheus Target Discovery

Problem: Prometheus could not discover service endpoints

Solution: Configured static targets in prometheus.yml with correct service names

Challenge 3: CI/CD Pipeline Complexity

Problem: Initial complex pipeline failed due to external dependencies

Solution: Simplified to reliable validation-focused workflow

Challenge 4: Security Scanning Integration

Problem: Trivy scanning in CI/CD caused pipeline failures

Solution: Implemented standalone security scanning with reporting

────────────────────────────────────────────────────────────────────────────────

7. Conclusion & Future Improvements

Project Success Summary

This DevOps pipeline project successfully demonstrates:

• Complete containerization of multi-service application

• Comprehensive monitoring with real-time visualization

• Security-first approach with vulnerability scanning

• Professional incident response procedures

• Automation and CI/CD best practices

• Industry-standard version control and documentation

Skills Demonstrated

• Docker containerization and orchestration

• Infrastructure monitoring with Prometheus/Grafana

• Security scanning and vulnerability management

• Incident response and post-mortem analysis

• Infrastructure automation with Ansible

• CI/CD pipeline design and implementation

• Professional documentation and version control

Future Improvements

1. Enhanced Security: Implement additional security scanning tools

2. Performance Optimization: Add caching layers and performance monitoring

3. High Availability: Implement load balancing and failover mechanisms

4. Advanced CI/CD: Add automated testing and deployment stages

5. Cloud Migration: Adapt for cloud deployment (AWS, Azure, GCP)

Learning Outcomes

• Gained practical experience with industry-standard DevOps tools

• Developed understanding of containerization and orchestration

• Learned monitoring and observability best practices

• Implemented security scanning and vulnerability management

• Created professional documentation and incident response procedures

────────────────────────────────────────────────────────────────────────────────

8. Appendices

Appendix A: Configuration Files

• Docker Compose configuration

• Prometheus configuration

• Grafana dashboard definitions

• Ansible playbooks

Appendix B: Scripts and Automation

• Setup automation scripts

• Security scanning scripts

• CI/CD pipeline definitions

Appendix C: Documentation

• README.md content

• Post-mortem report

• Project summary documentation

────────────────────────────────────────────────────────────────────────────────

Project Repository: https://github.com/niinora/devops\_final