

# CI/CD Pipeline and Monitoring Setup for Web Application

Name: Alkaios Stelios

## Overview

This document explains the CI/CD pipeline setup for deploying a web application, including the technologies, tools, and steps taken to build, test, and deploy the application. It also details the monitoring solution used to ensure application health and performance. The pipeline uses GitHub Actions and incorporates a manual approval step before deployment to production.

## Technologies and Tools Used

### Pipeline

- GitHub Actions: Automates the CI/CD workflow.
- Docker and Docker Compose: Builds and deploys the application.
- Linode Cloud Server: Hosts the application.

### Monitoring

- Prometheus: Collects metrics from the infrastructure and application.
- Grafana: Visualizes metrics and provides dashboards.
- Node Exporter: Captures server resource metrics.
- Blackbox Exporter: Probes service endpoints for uptime and response time.

## CI/CD Pipeline

### Pipeline Workflow

#### Trigger

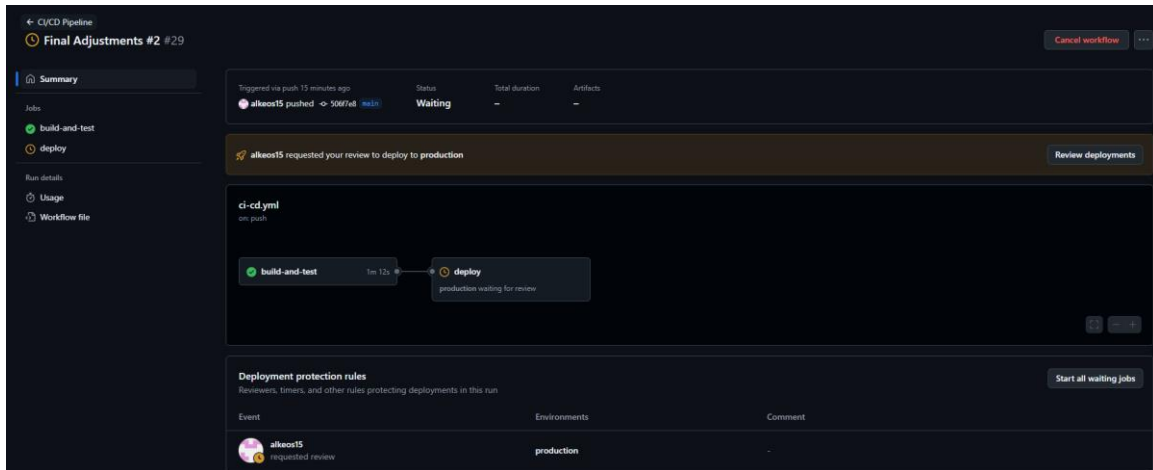
The pipeline is triggered automatically on:

- Push events to the main branch.
- Pull request creation targeting the main branch.

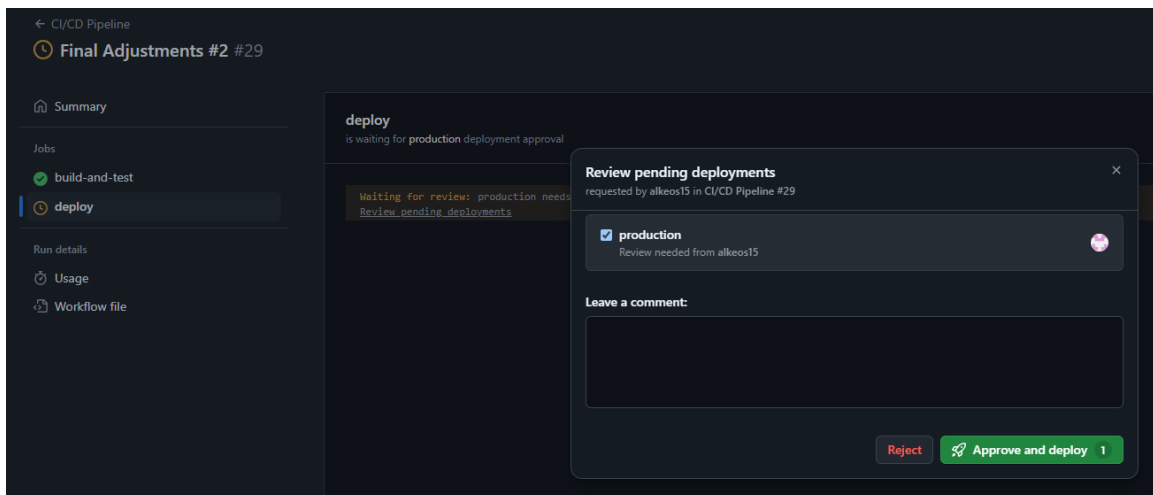
#### Workflow Stages

- Build and Test: Builds Docker images for the frontend and backend.

- Runs automated tests with memory limits to simulate production constraints.



- Manual Approval: Before deployment, the pipeline pauses and requires manual approval to proceed.



- **Deployment:** Deploys the application to the production server using Docker Compose.

Summary

Jobs

build-and-test

deploy

Run details

Usage

Workflow file

deploy

succeeded now in 48s

```

134 out: #10 EXPORTING LOGS TO /tmp/LOGS
135 out: #18 writing image sha256:58a681ff7fb5135ae644b640ab14788b994d108c019f3a892cfa50c5c4bc9cc1
136 out: #18 writing image sha256:58a681ff7fb5135ae644b640ab14788b994d108c019f3a892cfa50c5c4bc9cc1 done
137 out: #18 naming to docker.io/library/more-project-frontend done
138 out: #18 DONE 7.1s
139 out:
140 out: #19 [frontend] resolving provenance for metadata file
141 out: #19 DONE 0.0s
142 err: Network more-project_default Creating
143 err: Network more-project_default Created
144 err: Container node-exporter Creating
145 err: Container blackbox-exporter Creating
146 err: Container more-project-backend-1 Creating
147 err: Container more-project-frontend-1 Creating
148 err: Container prometheus Creating
149 err: Container grafana Creating
150 err: Container node-exporter Created
151 err: Container blackbox-exporter Created
152 err: Container prometheus Created
153 err: Container more-project-backend-1 Created
154 err: Container more-project-frontend-1 Created
155 err: Container grafana Created
156 err: Container node-exporter Starting
157 err: Container blackbox-exporter Starting
158 err: Container more-project-backend-1 Starting
159 err: Container more-project-frontend-1 Starting
160 err: Container prometheus Starting
161 err: Container grafana Starting
162 err: Container more-project-frontend-1 Started
163 err: Container grafana Started
164 err: Container more-project-backend-1 Started
165 err: Container node-exporter Started
166 err: Container blackbox-exporter Started
167 err: Container prometheus Started
168 =====
169 [x] Successfully executed commands to all host.
170 =====

```

Complete job

## Pipeline Configuration

The CI/CD pipeline is defined in ``.github/workflows/ci-cd.yml``:

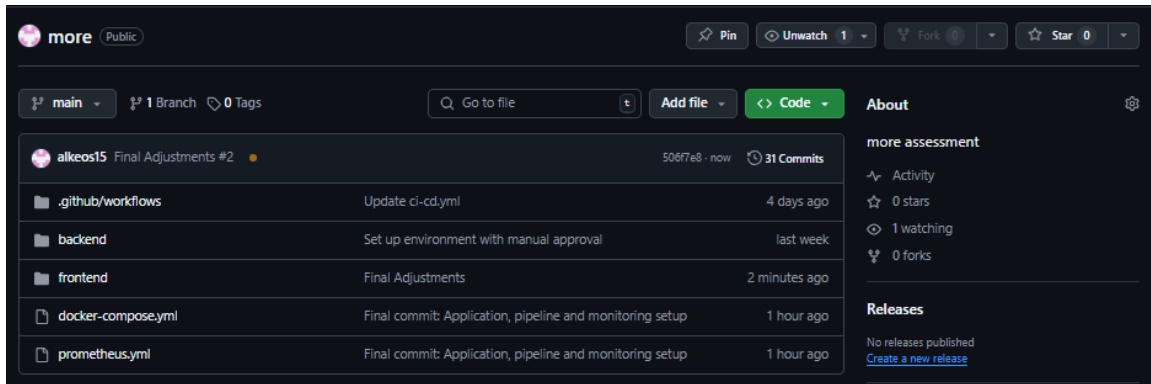
## Steps Taken to Set Up the Pipeline

- ## 1. Create the GitHub Repository

Created a new repository: <https://github.com/alkeos15/more.git>.

Added the application code (backend, frontend) and configuration files:

- docker-compose.yml
- prometheus.yml
- .github/workflows/ci-cd.yml



## 2. Define the CI/CD Pipeline

Created `.github/workflows/ci-cd.yml` to define the pipeline.

Configured the following stages:

- Build and test steps to ensure code correctness.
- Manual approval before deployment to production.
- Deployment using appleboy/ssh-action.

```
name: CI/CD Pipeline

on:
  push:
    branches:
      - main
  pull_request:
    branches:
      - main

jobs:
  build-and-test:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@v2

      # Step to install Docker Compose
      - name: Install Docker Compose
        run: |
          sudo curl -L "https://github.com/docker/compose/releases/latest/download/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
          sudo chmod +x /usr/local/bin/docker-compose
          docker-compose --version

      - name: Build Services
        run: docker-compose -f docker-compose.yml build

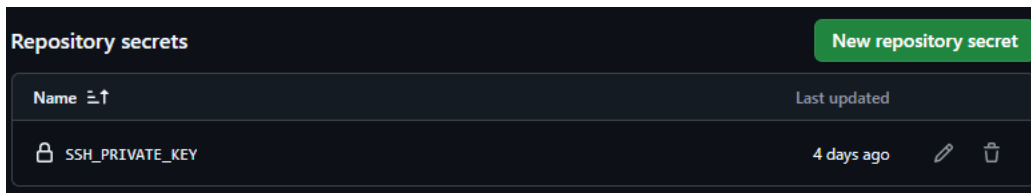
      # Run Backend tests with memory limits
      - name: Run Backend Tests
        run: docker-compose run --rm --memory=256m backend npm test || true
        # Limit memory usage for backend to 256MB

      # Run frontend tests with memory limits
      - name: Run Frontend Tests
        run: docker-compose run --rm --memory=256m frontend npm test || true
        # Limit memory usage for frontend to 256MB
    env:
      REACT_APP_BACKEND_URL: http://localhost:5000

  deploy:
    needs: build-and-test
    runs-on: ubuntu-latest
    environment: production # Specifies the production environment with manual approval
    steps:
      - name: Deploy to Production
        uses: appleboy/ssh-action@v0.1.4
        with:
          host: 172.105.77.29
          username: deployuser
          # password: ${ secrets.SSH_PASSWORD }
          key: ${ secrets.SSH_PRIVATE_KEY } # created new token
          port: 22
          script: |
            cd /home/more-project
            git pull origin main
            docker-compose down
            docker-compose up -d --build
```

### 3. Configure Secrets

Added the SSH\_PRIVATE\_KEY as a secret in GitHub to authenticate with the Linode server.



### 4. Implement the Application Deployment

Docker Compose is used to manage the deployment:

- Stops running containers with ``docker-compose down``.
- Rebuilds and starts containers with ``docker-compose up -d --build``.

```
services:
  backend:
    build: ./backend
    ports:
      - "5000:5000"
    environment:
      - PORT=5000
    deploy:
      resources:
        limits:
          memory: 256M
          cpus: '0.5'
    restart: always

  frontend:
    build: ./frontend
    ports:
      - "3000:3000"
    environment:
      - REACT_APP_BACKEND_URL=http://localhost:5000
    deploy:
      resources:
        limits:
          memory: 256M
          cpus: '0.5'
    restart: always

  prometheus:
    image: prom/prometheus:latest
    container_name: prometheus
    volumes:
      - ./prometheus.yml:/etc/prometheus/prometheus.yml
    ports:
      - "9090:9090"
    restart: always

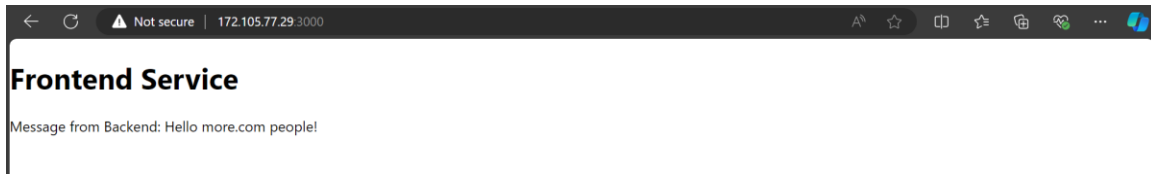
  grafana:
    image: grafana/grafana:latest
    container_name: grafana
    environment:
      - GF_SERVER_HTTP_PORT=3100
    ports:
      - "3100:3100"
    restart: always

  node-exporter:
    image: prom/node-exporter:latest
    container_name: node-exporter
    ports:
      - "9100:9100"
    restart: always

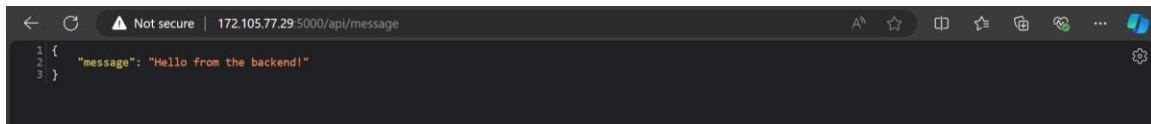
  blackbox-exporter:
    image: prom/blackbox-exporter:latest
    container_name: blackbox-exporter
    ports:
      - "9115:9115"
    restart: always
```

## 5. Validation

Accessed the deployed frontend at <http://172.105.77.29:3000>.



Tested the backend API at <http://172.105.77.29:5000/api/message>.



## Monitoring Setup

To ensure continuous application monitoring, Prometheus and Grafana were set up as part of the stack.

### Prometheus Configuration

Prometheus collects metrics from Node Exporter and Blackbox Exporter. The configuration file `prometheus.yml` defines the targets to monitor.

```
---
global:
  scrape_interval: 15s

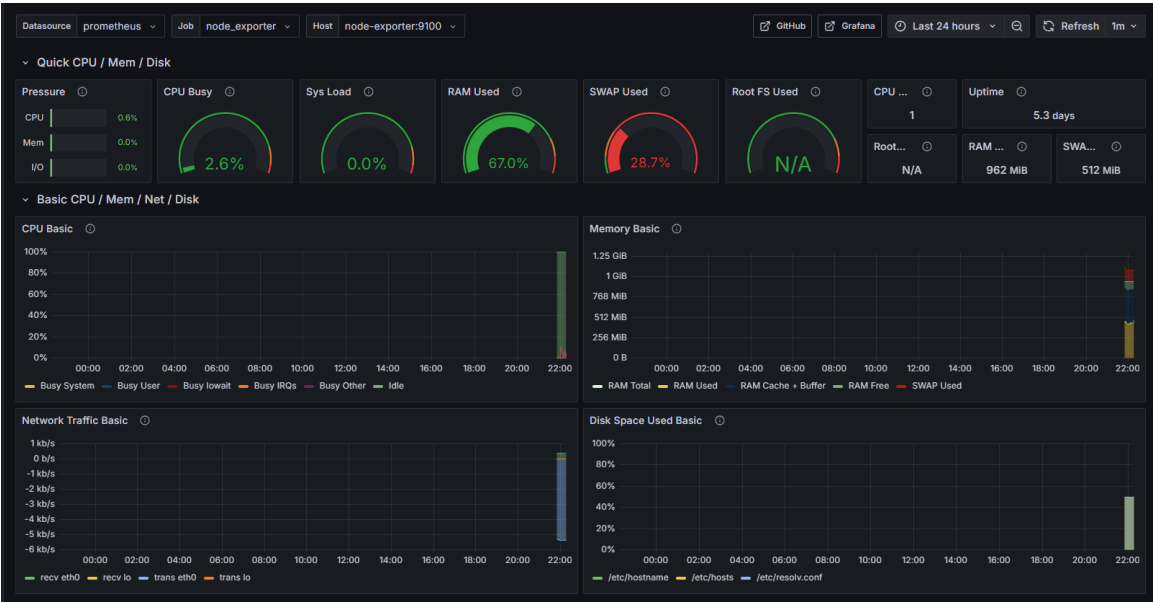
scrape_configs:
  - job_name: 'node_exporter'
    static_configs:
      - targets:
        - 'node-exporter:9100'

  - job_name: 'blackbox'
    metrics_path: /probe
    params:
      module: [http_2xx]
    static_configs:
      - targets:
        - http://frontend:3000
        - http://backend:5000/api/message
      labels:
        group: 'services'
    relabel_configs:
      - source_labels: [__address__]
        target_label: __param_target
      - source_labels: [__param_target]
        target_label: instance
      - target_label: __address__
        replacement: blackbox-exporter:9115
```

blackbox					2 / 2 up
Endpoint	Labels	Last scrape	State		
http://blackbox-exporter:9115/probe	group="services" instance="http://frontend:3000" job="blackbox"	4.705s ago	8.3ms	UP	
module: "http_2xx" target: "http://frontend:3000"					
http://blackbox-exporter:9115/probe	group="services" instance="http://backend:5000/api/message" job="blackbox"	11.112s ago	8.3ms	UP	
module: "http_2xx" target: "http://backend:5000/api/message"					
node_exporter					1 / 1 up
Endpoint	Labels	Last scrape	State		
http://node-exporter:9100/metrics	instance="node-exporter:9100" job="node_exporter"	1.926s ago	8.18ms	UP	

Grafana Configuration

Grafana visualizes the metrics collected by Prometheus. Two dashboards were set up:  
- Node Exporter Dashboard (Dashboard ID: 1860): Displays server resource metrics.



- Blackbox Exporter Dashboard (Dashboard ID: 7587): Tracks uptime and response time for service endpoints.

