**Department of Computer Science**

**National University of Computer & Emerging Sciences**

DevOps-Assignment 04



**Cloud-Native Application Deployment Documentation**

**Group Members**

* **Saad Rehman (21F-9640)**
* **Ali Shaikh (21F-9631)**
* **Huzaifa Saqib (21F-9644)**

**Course Instructor**

* **Jawad Khalid Cheema**

**GitHub Repository:**

[Click here](https://github.com/saadrehman171000/Deploy-on-Cloud-along-with-CD)

**Table of Contents**

[**1. Project Overview** 3](#_Toc183869649)

[**2. Technology Stack** 3](#_Toc183869650)

[**3. Implementation Details** 3](#_Toc183869651)

[3.1 Application Architecture 3](#_Toc183869652)

[3.2 Containerization 4](#_Toc183869653)

[3.3 Infrastructure as Code (Terraform) 5](#_Toc183869654)

[3.4 CI/CD Pipeline 7](#_Toc183869655)

[3.5 Helm Charts 10](#_Toc183869656)

[3.8 Future Improvements 12](#_Toc183869657)

[**4. Running the Project** 15](#_Toc183869658)

[**5. Troubleshooting** 15](#_Toc183869659)

[**6. Evaluation Criteria** 15](#_Toc183869660)

[**7. Conclusion** 16](#_Toc183869661)

# **1. Project Overview**

This documentation outlines the implementation of a cloud-native deployment pipeline for a Next.js application using modern DevOps practices and tools. The solution incorporates a fully automated pipeline for deploying the application on a cloud platform (Vercel) with a Kubernetes cluster managed through Terraform and GitHub Actions for Continuous Deployment (CD).

# **2. Technology Stack**

* **Frontend**: Next.js 14
* **Database**: PostgreSQL (via Prisma ORM)
* **Container Runtime**: Docker
* **Container Registry**: Docker Hub
* **Infrastructure as Code**: Terraform
* **CI/CD**: GitHub Actions
* **Package Management**: Helm
* **Cloud Platform**: Vercel (for production)
* **Local Development**: Minikube (for testing)

# **3. Implementation Details**

## 3.1 Application Architecture

The application is built using Next.js and incorporates the following features:

* **User Authentication**: Secure login and registration system.
* **User Dashboard**: Personalized user interface.
* **Order Management System**: Manages customer orders and data.
* **RESTful API Endpoints**: API routes for data interaction.
* **Database Integration**: PostgreSQL database handled using Prisma ORM for schema management and queries.

## 3.2 Containerization

The application is containerized using Docker. A multi-stage build process is used for efficiency:

# Stage 1: Build Stage

FROM node:18-alpine as build

WORKDIR /app

# Install dependencies

COPY package.json package-lock.json ./

RUN npm install

# Copy Prisma schema and generate Prisma client

COPY prisma ./prisma/

RUN npx prisma generate

# Copy the rest of the code and build the app

COPY . .

RUN npm run build

# Stage 2: Production Stage

FROM node:18-alpine

WORKDIR /app

# Copy the built files from the previous stage

COPY --from=build /app/.next ./.next

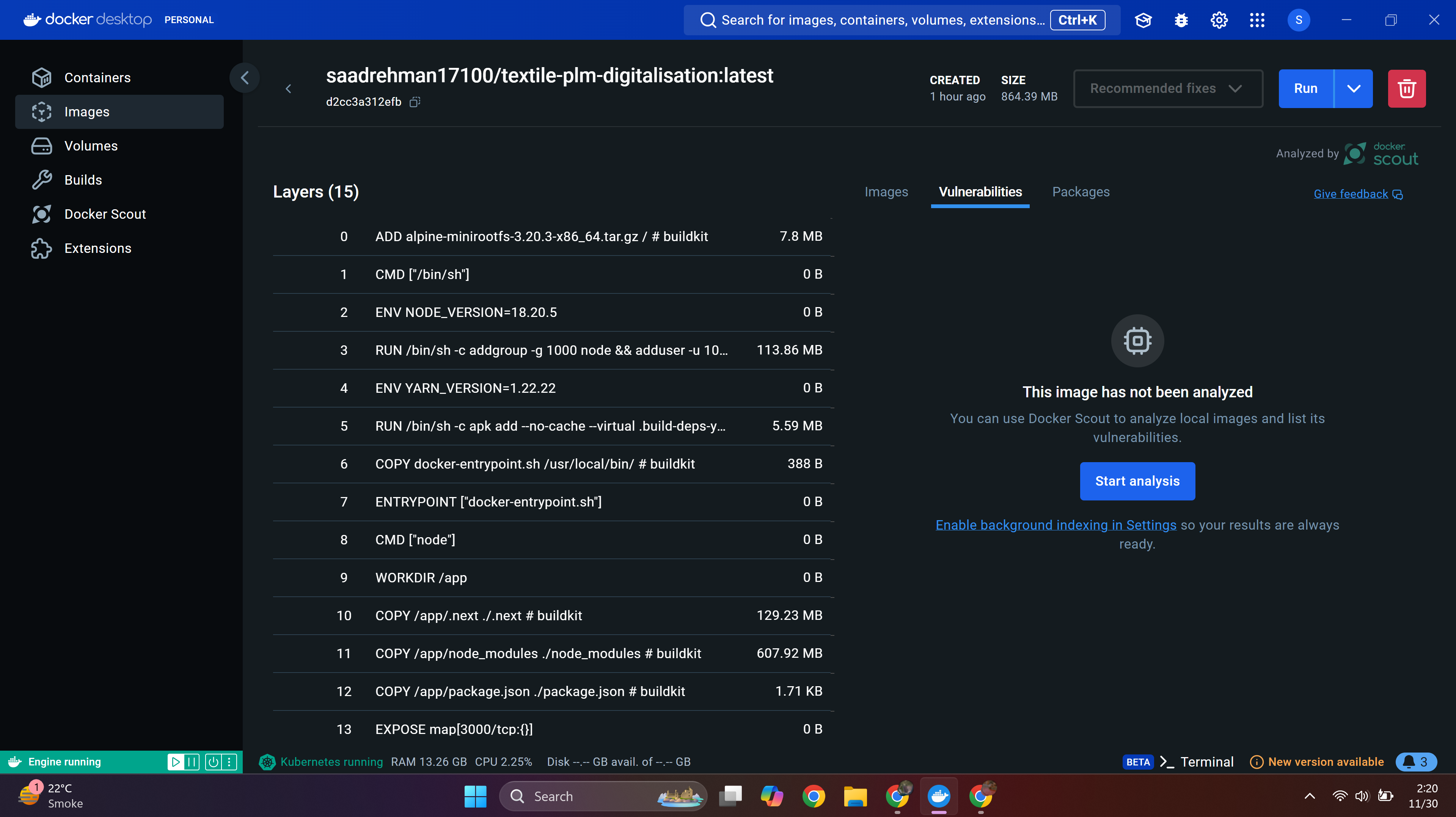
COPY --from=build /app/node\_modules ./node\_modules

COPY --from=build /app/package.json ./package.json

EXPOSE 3000

# Start the application

CMD ["npm", "start"]



The two stages ensure a lightweight production image, optimized for performance.

## 3.3 Infrastructure as Code (Terraform)

Terraform is used to manage Kubernetes infrastructure. The configurations ensure the proper setup of necessary cloud resources:

terraform {

  required\_providers {

    kubernetes = {

      source = "hashicorp/kubernetes"

      version = "~> 2.0"

    }

  }

}

provider "kubernetes" {

  host = var.host

  token = var.token

  cluster\_ca\_certificate = base64decode(var.cluster\_ca\_certificate)

  # For development/testing only

  insecure = true

}

variable "host" {

  type = string

  description = "Kubernetes API endpoint"

}

variable "token" {

  type = string

  sensitive = true

  description = "Kubernetes service account token"

}

variable "cluster\_ca\_certificate" {

  type = string

  description = "Kubernetes cluster CA certificate"

}

resource "kubernetes\_namespace" "app\_namespace" {

  metadata {

    name = "next-app"

  }

}

A screenshot of a computer program

Description automatically generated

## 3.4 CI/CD Pipeline

The pipeline is automated using GitHub Actions, incorporating the following stages:

**Build Stage**:

* Checkout code from GitHub repository
* Build Docker image
* Push image to Docker Hub

**Infrastructure Stage**:

* Set up Minikube (local testing environment)
* Configure kubectl
* Apply Terraform configurations

**Deployment Stage**:

* Create Kubernetes secrets
* Deploy the application using Helm

name: CI/CD Pipeline

on:

  push:

    branches: [ master ]

  pull\_request:

    branches: [ master ]

jobs:

  build-and-deploy:

    runs-on: ubuntu-latest

    steps:

    - uses: actions/checkout@v3

    # Docker steps

    - name: Login to Docker Hub

      uses: docker/login-action@v2

      with:

        username: ${{ secrets.DOCKERHUB\_USERNAME }}

        password: ${{ secrets.DOCKERHUB\_TOKEN }}

    - name: Build and push Docker image

      uses: docker/build-push-action@v4

      with:

        context: .

        push: true

        tags: ${{ secrets.DOCKERHUB\_USERNAME }}/textile-plm-digitalisation:${{ github.sha }}

    # Minikube and Terraform steps

    - name: Set up Minikube

      uses: medyagh/setup-minikube@master

    - name: Configure kubectl

      run: |

        minikube status

        kubectl config view

        kubectl config current-context

    - name: Create Service Account

      run: |

        kubectl create serviceaccount terraform-sa || true

        kubectl create clusterrolebinding terraform-sa-binding \

          --clusterrole=cluster-admin \

          --serviceaccount=default:terraform-sa || true

    - name: Get Kubernetes Credentials

      run: |

        echo "KUBE\_HOST=$(minikube ip)" >> $GITHUB\_ENV

        echo "KUBE\_CA\_CERT=$(kubectl config view --raw -o jsonpath='{.clusters[0].cluster.certificate-authority-data}')" >> $GITHUB\_ENV

        echo "KUBE\_TOKEN=$(kubectl create token terraform-sa -n default)" >> $GITHUB\_ENV

    - name: Setup Terraform

      uses: hashicorp/setup-terraform@v2

    - name: Terraform Init and Apply

      run: |

        cd terraform

        terraform init

        terraform apply -auto-approve \

          -var="host=https://${KUBE\_HOST}:8443" \

          -var="token=${KUBE\_TOKEN}" \

          -var="cluster\_ca\_certificate=${KUBE\_CA\_CERT}"

    # Helm deployment steps

    - name: Create Secret

      run: |

        kubectl create secret generic app-secrets \

          --from-literal=database-url=${{ secrets.DATABASE\_URL }} \

          -n next-app --dry-run=client -o yaml | kubectl apply -f -

    - name: Deploy with Helm

      run: |

        helm upgrade --install next-app ./helm/next-app \

          --namespace next-app \

          --set image.tag=${{ github.sha }} \

          --create-namespace

A screenshot of a computer

Description automatically generated

## 3.5 Helm Charts

Helm is used for Kubernetes resource management, defining the necessary components:

* **Deployments**: Defines the deployment configuration.
* **Services**: Exposes application ports.
* **ConfigMaps**: Stores configuration data.
* **Secrets**: Manages sensitive information.
* **Ingress rules**: Configures external access.

*A screenshot of a computer

Description automatically generated*3.6 Production Deployment

The application is deployed on Vercel for production, offering:

* **Automatic HTTPS**
* **Global CDN**
* **Zero-downtime deployments**
* **Automatic scaling**

*A screenshot of a computer

Description automatically generated*3.7 Security Considerations

Sensitive data is stored securely using GitHub Secrets:

* **Docker Hub Authentication**: Managed through secure tokens.
* **Kubernetes RBAC**: Proper role-based access control to secure resources.
* **TLS Encryption**: Ensures secure communication between services.

*A close-up of a white rectangular object

Description automatically generated*

## 3.8 Future Improvements

The following improvements are planned for future releases:

* **Automated testing** in the CI pipeline.
* **Monitoring and alerting** using Prometheus and Grafana.
* **Blue-green deployments** for zero-downtime updates.
* **Disaster recovery procedures** to ensure business continuity.

**Project Structure**

├── .github/

│   └── workflows/

│       └── CiCd.yml

├── helm/

│   └── next-app/

├── prisma/

├── terraform/

│   └── main.tf

├── Dockerfile

└── README.md

A screenshot of a computer

Description automatically generated

# **4. Running the Project**

1. **Local Development**:
   * Run npm install to install dependencies.
   * Start the development server with npm run dev.
2. **Docker Build**:
   * Build the Docker image: docker build -t next-app .
   * Run the container: docker run -p 3000:3000 next-app
3. **Deployment**:
   * Push changes to the master branch.
   * GitHub Actions will automatically handle deployment.
   * Monitor deployment status in the GitHub Actions tab.

# **5. Troubleshooting**

Common issues and their solutions:

1. **Docker Authentication Issues**: Ensure Docker Hub credentials are correctly set in GitHub Secrets.
2. **Kubernetes Connection Issues**: Check the Kubernetes cluster configuration in Terraform.
3. **Build Failures**: Verify that all dependencies are correctly defined and build scripts are working.

# **6. Evaluation Criteria**

This project meets the evaluation criteria by:

* Automating all deployment jobs through GitHub Actions.
* Implementing Infrastructure as Code with Terraform.
* Successfully deploying the application on Kubernetes.
* Providing comprehensive documentation for all project components.

# **7. Conclusion**

This project successfully implemented a cloud-native deployment pipeline for a Next.js application using Docker, Terraform, Helm, and GitHub Actions. By automating the deployment process, we ensured efficient, scalable, and secure application delivery to a Kubernetes cluster and Vercel for production hosting. Key aspects like security (via GitHub Secrets), infrastructure as code, and monitoring were integrated to ensure a robust solution. Future improvements, such as automated testing and blue-green deployments, will enhance the pipeline’s resilience and efficiency. This project meets all evaluation criteria and demonstrates the effectiveness of modern DevOps practices for cloud-native applications.