DevOps Internship Assessment

Title: Containerize and Deploy a Next.js Application using Docker, GitHub Actions, and Minikube
Objective
This assessment tests your ability to:

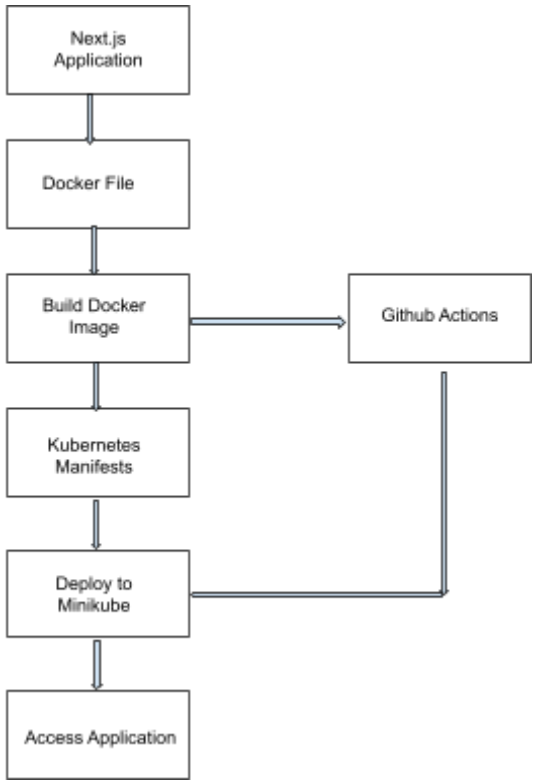
- Containerize a Next.js application using Docker
- Automate build and image push using GitHub Actions and GitHub Container Registry (GHCR)
- Deploy the containerized app to Kubernetes (Minikube) using manifests

Requirements

1. Create a Next.js application (can be a simple starter template)
2. Write a Dockerfile to containerize the application with best practices
3. Set up GitHub Actions workflow to:
    - o Build Docker image on push to main branch
    - o Push image to GitHub Container Registry (GHCR)
    - o Use proper image tagging
4. Create Kubernetes manifests in a k8s/ folder:
    - o deployment.yaml - with replicas and health checks
    - o service.yaml - to expose the application
5. Document everything in README.md with:
    - o Setup instructions
    - o Local run commands
    - o Deployment steps for Minikube
    - o How to access the deployed application

Solution

Below is the work workflow diagram for the assignment



STEPS:

1. Project Setup

I began the assignment by setting up a simple Next.js application using the official starter template.
The project was initialized using the command:

npx create-next-app@latest my-next-app

Once created, I verified that the application ran successfully in development mode by executing npm run dev and checking it locally at http://localhost:3000.

2. Dockerization of the Application

After confirming that the application worked as expected, I proceeded to containerize it using Docker.
A multi-stage Dockerfile was created to ensure the final image remained lightweight and optimized for production.

- The first stage handled dependency installation and build.
- The second stage copied only the required files and ran the app as a non-root user for security reasons.

The container image was built and tested locally with:

docker build -t my-nextjs-app .
docker run -p 3000:3000 my-nextjs-app

The application launched successfully inside the container, confirming a correct Docker setup.

3. GitHub Repository and Version Control

A public GitHub repository was created to host the project code, Dockerfile, workflow configuration, and Kubernetes manifests.
The local project was linked to the repository, and all files were committed and pushed to the main branch for version tracking.

4. GitHub Actions Workflow Configuration

To automate the build and deployment process, I configured a GitHub Actions workflow inside .github/workflows/ci.yml.
The workflow was designed to:

- Trigger automatically on every push to the main branch.
- Build the Docker image.
- Push the image to GitHub Container Registry (GHCR) using the built-in GitHub authentication token.
- Apply proper image tagging, including latest, the commit SHA, and a version number.

After committing the workflow file, I verified that the pipeline executed successfully and that the image appeared under the Packages section of the GitHub repository.

5. Kubernetes Manifests Creation

A k8s directory was created to store the Kubernetes manifests required for deployment.
Two main files were prepared:

- deployment.yaml – Defined the application deployment, including 2 replicas, container resource limits, and readiness/liveness probes for health monitoring.
- service.yaml – Configured the service to expose the application externally via a NodePort, mapping port 80 to container port 3000.

These configurations ensured scalability, fault tolerance, and proper service discovery within the Kubernetes cluster.

6. Deployment on Minikube

The deployment was tested locally using Minikube.
Minikube was started with the command:

minikube start

To test efficiently, I used two methods:

Loading the local Docker image into Minikube using:

minikube image load ghcr.io/<username>/<repository>:latest

1. Pulling directly from GHCR, where I created a Docker registry secret:

   kubectl create secret docker-registry ghcr-secret \

--docker-server=ghcr.io \
--docker-username=<username> \
--docker-password=<PAT> \
--docker-email=you@example.com

2. and referenced it in the deployment manifest.

Both approaches were verified to work successfully.

7. Verification and Testing

After applying the manifests using:

kubectl apply -f k8s/

I verified that the pods were running and healthy using:

kubectl get pods

The service details were checked with:
kubectl get svc

Finally, the deployed application was accessed through the browser using:

minikube service nextjs-service

This confirmed that the Next.js application was successfully running within the Kubernetes cluster.

8. Documentation and Repository Finalization

A detailed README.md file was written to document:

- Local setup and run instructions
- Docker build and run steps
- GitHub Actions workflow overview
- Kubernetes deployment guide for Minikube
- Application access and troubleshooting notes

All files (source code, Dockerfile, workflow, and manifests) were reviewed and committed to ensure a complete and clean repository.

9. Final Validation and Submission

As a final step, I re-ran all commands from scratch to ensure the instructions in the documentation were reproducible.
After confirming that:

- The GitHub Actions workflow completed successfully
- The GHCR image was published correctly
- The app deployed and responded on Minikube

I made the repository public and prepared the submission email containing:

- The GitHub repository link
- The GHCR image URL

This completed the assignment successfully.