Project 2

"Fix, troubleshoot, and deploy a containerized and automated CI/CD solution for a full-stack application consisting of a Next.js frontend and a Laravel backend. The solution must include containerization using Docker or Docker Compose, a CI/CD pipeline for the frontend using GitHub Actions or Jenkins, and a backend deployment pipeline using Jenkins or GitHub Actions. The project should automate the process of building, pushing, and deploying Docker images to Docker Hub or another container registry, and link the frontend and backend services in a production-ready environment. Additionally, create a script using Python or Bash to check for Docker Hub updates and ensure seamless deployment of the latest versions. The final version of this project should be able to deploy the full-stack application, with the frontend and backend communicating effectively, and include a mechanism to monitor and update the deployed containers.

You are free to explore alternative tools such as Kubernetes for orchestration, GitLab CI/CD or CircleCI for pipelines, and other container registries like GitHub Container Registry (GHCR) or AWS ECR. Modify the project while ensuring the expected output of a fully automated, containerized, and deployable full-stack application."

Milestones

🟧 🟩

Intended Outcome

Phase 1: **Containerize the Next.js frontend using Docker and Docker Compose.**

1. Set Up Docker Environment:
Install Docker and Docker Compose on local machines.
Familiarize students with basic Docker commands (docker build, docker run, docker-compose up).

2. Create a Dockerfile for Next.js:
Write a Dockerfile for the Next.js app.
Use a multi-stage build to optimize the image size (e.g., build stage with Node.js and production stage with Nginx or a lightweight server).
    Alternative: Use a pre-built Node.js image for simplicity.

Progress Criteria/ Deliverable

1. Install all dependencies and tools successfully for development based on the boilerplate code

W
e
e
k

1

Containeriz
e the
Frontend
(Next.js)

3. Create a docker-compose.yml File:

Define the Next.js service in a docker-compose.yml file.

Include environment variables (if needed) and port mappings.

Alternative: Use standalone Docker commands instead of Docker Compose for simpler setups.

4. Test the Container Locally:

Build and run the Docker container locally.

Verify that the Next.js app is accessible via a browser.

5.Push the Image to Docker Hub:

Create a Docker Hub account (if not already done).

Tag and push the Docker image to Docker Hub.

Alternative: Use another container registry like GitHub Container Registry (GHCR) or AWS ECR.

2. Configure node, composer, MySQL or PostgreSQL, Laravel, PHP

3. Setup repository, create sample architecture or theory design to be followed for the deployment

hoot boilerplate code, setup database for conn

W
e
e
k

2

Automate Frontend Builds with GitHub Actions

Phase 2: Create a CI/CD pipeline using GitHub Actions to build and push the Docker image to Docker Hub.

1. Set Up GitHub Actions:
Create a .github/workflows directory in the repository.
Write a GitHub Actions workflow YAML file (e.g., docker-build-push.yml).

2. Define the Workflow:
Trigger the workflow on push to the main branch or on a pull request.
Use actions to check out the code, log in to Docker Hub, build the Docker image, and push it to Docker Hub.
Alternative: Use GitLab CI/CD or CircleCI instead of GitHub Actions.

3. Test the Pipeline:
Push changes to the repository and verify that the workflow runs successfully.
Check Docker Hub to confirm the new image is uploaded.

4. Optional: Add Notifications:
Configure Slack or email notifications for build success/failure.

1. GitHub Action script to automate building of the container for the fron-end

2. Notifications for unsuccessful builds

W e e k 3

Automate Backend Deployment with Jenkins

Phase 3: Set up a CI/CD pipeline for the Laravel backend using Jenkins.

1. Set Up Jenkins:
Install Jenkins on a local machine or cloud server.
Install necessary plugins (e.g., Git, Docker, Pipeline).

2. Create a Jenkins Pipeline:
Write a Jenkinsfile to define the pipeline stages:
Check out the Laravel code from a specific repository.
Build the Docker image for the Laravel backend.
Push the image to Docker Hub or another registry.
Alternative: Use GitHub Actions for the backend pipeline instead of Jenkins.

3. Deploy the Backend to a Docker Container:
Write a script or use Jenkins to deploy the backend container to a server or local environment.
Use Docker Compose to manage the backend service.

4. Test the Pipeline:
Trigger the Jenkins pipeline and verify that the backend is deployed successfully.

1. Jenkinsfile for automating the backend build

2. Dockerized backend container

3. Working connection between backend and frontend

4. Trigger or workflow for the backend and front-end connection

W e e k 4	Link Frontend and Backend	**Phase 4:** Deploy and link the frontend and backend containers. Steps: **1. Create a Combined docker-compose.yml File:** Define both the Next.js frontend and Laravel backend services in a single docker-compose.yml file. Configure networking to allow communication between the containers. Alternative: Use Kubernetes for orchestration if students are advanced. **2. Deploy the Full Stack:** Run docker-compose up to start both containers. Verify that the frontend can communicate with the backend (e.g., test API endpoints). **3. Create a Script to Check for Docker Hub Updates:** Write a Python or Bash script to check Docker Hub for updates to the frontend or backend images. Use the Docker Hub API or docker pull commands to check for new tags. Alternative: Use a cron job to automate the update check. **4. Optional: Add Monitoring and Logging:** Set up monitoring tools (e.g., Prometheus, Grafana) or logging (e.g., ELK stack) for the deployed application	1. Full stack application deployment including the database, backend, and front end with their respective automation scripts 2. Login functionality