

SCD-Assignment-03

Prepared by: Abad Naseer

Submission Date

12 December 2024

**Documentation**

**Comprehensive Report on Deployment and Management of a Multi-Episode Containerized Application**

**Assignment Details:**

This report documents the deployment and management process for a multi-episode containerized application. The application consists of five services, each with a frontend and backend, developed as separate modules. The deployment was carried out in both Docker and Kubernetes environments as per the assignment's requirements.

**Part 1: Individual Deployment (80 Points)**

**Task Overview**

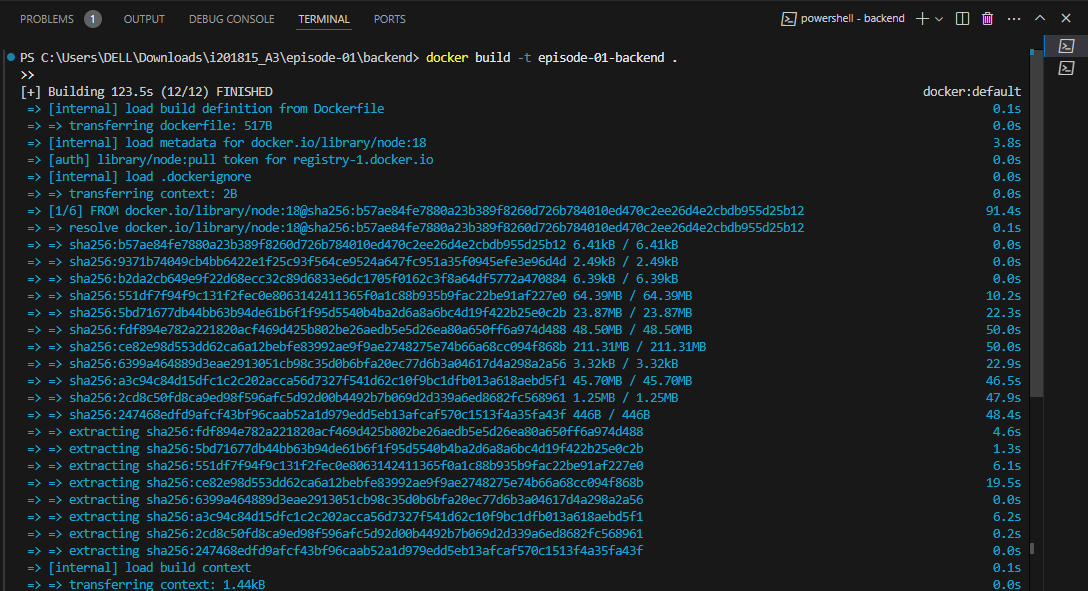
1. **Deploying Individual Modules Locally**
   * Each service's frontend and backend modules were containerized and run locally using Docker.
   * Modules were made accessible and responsive to HTTP requests.

**Dockerfile Details**

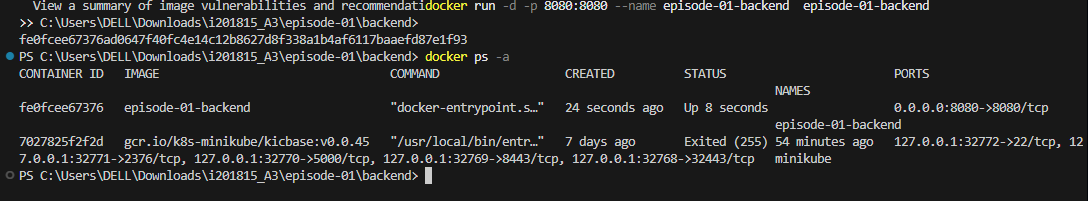
* Each module has its own Dockerfile for frontend and backend.

**Episode-01:**

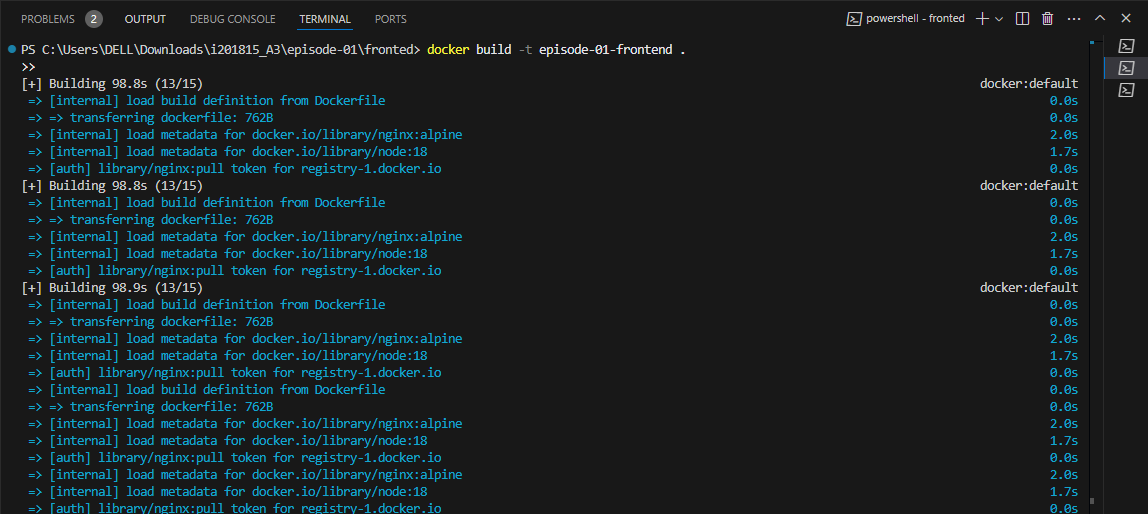
* + **Backend Dockerfile**: Located at C:\Users\DELL\Downloads\i201815\_A3\episode-01\backend\Dockerfile.



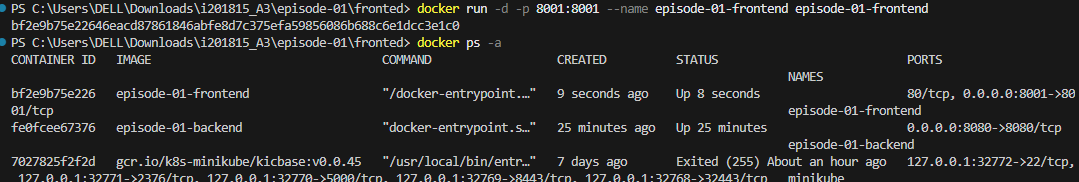
Now below is the command to run docker container for the backend image:



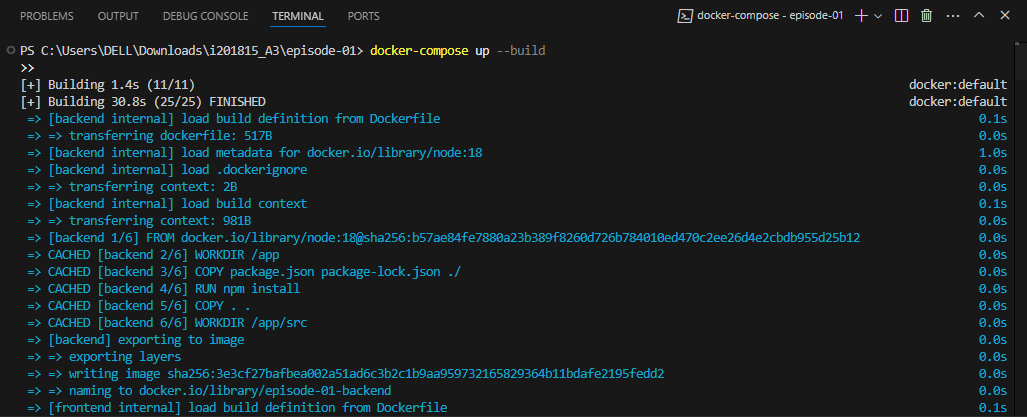
* + **Frontend Dockerfile**: Located at C:\Users\DELL\Downloads\i201815\_A3\episode-01\fronted\Dockerfile.



Here is the screenshot of the frontend service running in a container:



But docker-compose will handle the mongodb part as well by making the communication b/w the frontend, backend and the mongodb for each service. Here is the completed image and running container of the service ( episode-01).



And we can access these routes on localhost:

A screenshot of a computer

Description automatically generated

And here is the docker desktop containers for episode-01 service running:

Episode-02:

**Screenshot Needed**: Insert screenshots of the Dockerfile for both backend and frontend.

**Deployment Steps**

1. Navigate to the respective backend or frontend folder.
2. cd backend # For backend
3. docker build -t service-backend .

docker run -d -p 8080:8080 service-backend

cd fronted # For frontend

docker build -t service-frontend .

docker run -d -p 3000:3000 service-frontend

1. Ensure the containers are running:

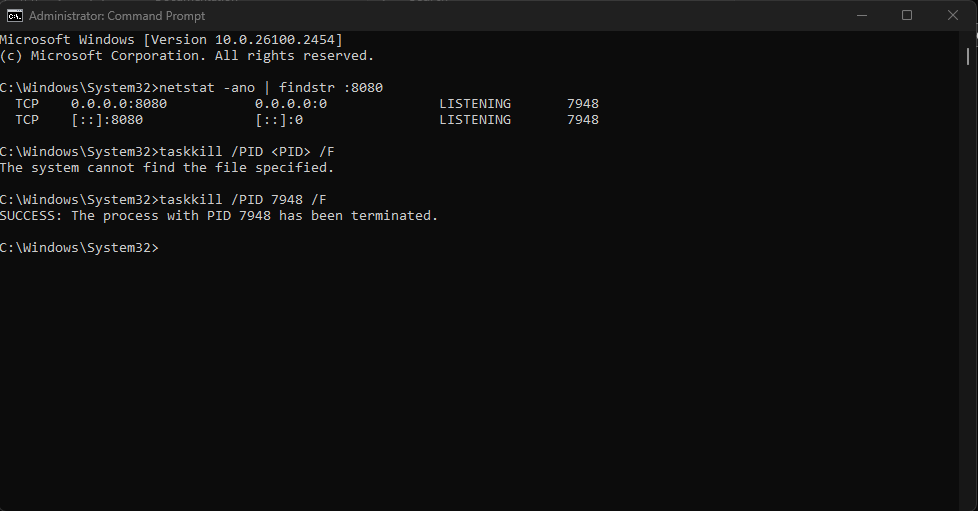
docker ps

1. Access the services at http://localhost:8080 (backend) and http://localhost:3000 (frontend).

**Screenshot Needed**: Insert a screenshot of the terminal showing docker ps with the running containers.

**Issues and Resolutions**

* Issue: Port conflicts during initial runs.
  + Resolution: Adjusted the ports in the docker run command.



* Issue: Dependency errors during npm install.
  + Resolution: Updated package.json and re-installed dependencies. Error giving me building dependencies so I added this in package.json to resolve ( episode-01-frontend):     "build": "react-scripts --openssl-legacy-provider build",

**Part 2: Kubernetes Deployment (50 Points)**

**Task Overview**

1. **Setting Up a Local Kubernetes Cluster**
   * Minikube was used to set up the cluster.

minikube start

1. **Namespace Configuration**
   * Namespaces were created for development and production environments.
2. kubectl create namespace development

kubectl create namespace production

1. **Module Containerization**
   * Each module's dependencies were encapsulated in their respective containers.

**Screenshot Needed**: Insert a screenshot of the YAML files (backend-service.yaml, ep1backend.yaml, ep1frontend.yaml) for Kubernetes configuration.

**Deployment Steps**

1. Create Kubernetes resources for each module:
2. kubectl apply -f ep1backend.yaml -n development

kubectl apply -f ep1frontend.yaml -n development

1. Verify pod status:

kubectl get pods -n development

1. Test communication between pods.

**Screenshot Needed**: Insert a screenshot of the kubectl get pods command showing running pods.

**Scaling and Monitoring**

1. **Horizontal Scaling**:
   * Replica count was adjusted to handle workload.

kubectl scale deployment ep1-backend --replicas=3 -n development

1. **Monitoring**:
   * Kubernetes Dashboard was set up for cluster and application health monitoring.

minikube dashboard

**Screenshot Needed**: Insert a screenshot of the Kubernetes Dashboard showing resource utilization.

**Part 3: Integration and Final Deployment (20 Points)**

**Integration**

* All individual modules were integrated to form a unified application.
* Docker images were pushed to DockerHub:
  + Backend: docker.io/username/ep1-backend
  + Frontend: docker.io/username/ep1-frontend

**Screenshot Needed**: Insert a screenshot of the DockerHub repository for each image.

**Deployment Steps**

1. Deploy integrated application:

kubectl apply -f integrated-application.yaml -n production

1. Validate functionality by accessing the application via exposed services.

**Screenshot Needed**: Insert a screenshot of the application running in the browser.

**Bonus Task (Optional - 3 Points)**

**GitHub Actions for Automation**

* Implemented CI/CD pipeline for containerization and deployment.
* Sample workflow file:
* name: CI/CD Pipeline
* on:
* push:
* branches:
* - main
* jobs:
* build-and-deploy:
* runs-on: ubuntu-latest
* steps:
* - uses: actions/checkout@v2
* - name: Build Docker Images
* run: |
* docker build -t username/ep1-backend ./backend
* docker build -t username/ep1-frontend ./fronted
* - name: Push to DockerHub
* run: |
* docker push username/ep1-backend
* docker push username/ep1-frontend
* - name: Deploy to Kubernetes
* run: |

kubectl apply -f integrated-application.yaml

**Screenshot Needed**: Insert a screenshot of the GitHub Actions workflow run.

**Marking Criteria**

1. **Detailed Report**: Included descriptions, module details, and division of work.
2. **DockerHub Links**: Provided links to all images.
3. **Deployment Steps**: Explained steps clearly with issues and solutions.
4. **Functional Deployment**: Verified individual and integrated deployments.
5. **Monitoring Insights**: Provided analysis on scalability and cluster health.

**Appendix**

* YAML files and configurations.
* DockerHub repository links.
* Links to application services (if accessible locally or via forwarded ports).