**Project Documentation: Containerizing a Full-Stack Web Application with Docker**

**1. Project Overview**

In this project, you will containerize a full-stack web application comprising a React frontend, an Express backend, and a MongoDB database. By using Docker, Docker Compose, and Shell scripting, this project will:

* Package each component into isolated containers.
* Use Docker Compose to manage and connect all services.
* Automate the build, tagging, and deployment process using a Shell script for efficient and repeatable deployment.

**2. Aim**

The objective is to streamline deployment and improve the manageability of a full-stack application by containerizing each component. Using Docker Compose will allow simultaneous startup and teardown of all components with a single command. This approach makes the application portable and adaptable across various environments.

**3. Components**

1. **React Frontend**: Serves as the user interface of the application.
2. **Express Backend**: Provides server-side functionality and an API.
3. **MongoDB Database**: Stores the application data.

**4. Prerequisites**

* **Docker**: Ensure Docker is installed on your machine.
* **Docker Hub Account**: Sign up at [Docker Hub](https://app.docker.com/signup?) if you don’t have one.
* **GitHub Repository**: Access the code for the frontend and backend here: [GitHub Repository](https://github.com/fliprlab/devops-task).

**5. Task Breakdown**

**5.1 Create Dockerfiles**

**5.1.1 Frontend Dockerfile**

* **Location**: frontend directory.
* **Steps**:
  1. Use a Node.js base image.
  2. Set the working directory.
  3. Copy required files.
  4. Install dependencies.
  5. Build the React application.

# Use an official Node.js base image for building the React app

FROM node:16 AS build

# Set the working directory inside the container

WORKDIR /app

# Copy the package.json and package-lock.json files

COPY package\*.json ./

# Install the dependencies

RUN npm install

# Copy the rest of the application files

COPY . .

# Build the React app

RUN npm run build

CMD ["npm", "start"]

# Expose the port the app will run on

EXPOSE 3000

**5.1.2 Backend Dockerfile**

* **Location**: backend directory.
* **Steps**:
  1. Use a Node.js base image.
  2. Set the working directory.
  3. Copy necessary files.
  4. Install dependencies.
  5. Expose the appropriate port.

# Use an official Node.js base image

FROM node:16

# Set the working directory inside the container

WORKDIR /app

# Copy the package.json and package-lock.json files to install dependencies

COPY package\*.json ./

# Install dependencies

RUN npm install

# Copy the rest of the application files

COPY . .

# Expose the port the backend will run on (assuming 5000)

EXPOSE 5000

# Command to run the application

CMD ["npm", "start"]

**5.2 Create Docker Compose File**

* **Location**: Project root directory.
* **Configuration**:
  1. Define services for frontend, backend, and MongoDB.
  2. Configure port mappings.
  3. Set dependencies so that services initialize in the correct order.
  4. Establish environment variables for database connection.

version: '3.8'

services:

frontend:

image: viru786/frontend:20241112190811

container\_name: frontend # Assumes there's a Dockerfile in the frontend directory

ports:

- "3000:3000" # Maps frontend container port 3000 to localhost 3000

depends\_on:

- backend # Ensures backend service starts before frontend

backend:

image: viru786/backend:20241112190811

container\_name: backend # Assumes there's a Dockerfile in the backend directory

ports:

- "5000:5000" # Maps backend container port 5000 to localhost 5000

environment:

- MONGO\_URI=mongodb://13.127.88.246:27017/sampleApp # MongoDB connection URI

depends\_on:

- mongo # Ensures MongoDB starts before backend

mongo:

image: mongo:latest

ports:

- "27017:27017" # Maps MongoDB container port 27017 to localhost 27017

environment:

- MONGO\_INITDB\_ROOT\_USERNAME=admin # Sets MongoDB root username

- MONGO\_INITDB\_ROOT\_PASSWORD=password # Sets MongoDB root password

volumes:

- mongo\_data:/data/db # Sets up volume for MongoDB data persistence

volumes:

mongo\_data:

driver: local # Uses local driver to persist MongoDB data

* 1. Use a volume for MongoDB data persistence.

**5.3 Automation Shell Script**

**Step 1: Clone the Provided GitHub Repository**

* Clone the frontend and backend repositories locally from [GitHub](https://github.com/fliprlab/devops-task).

**Step 2: Create Dockerfiles for the Frontend and Backend**

* **Frontend**: Write a Dockerfile that:
  + Uses a Node.js base image, builds the React app, and serves it via a lightweight server.
* **Backend**: Write a Dockerfile that:
  + Uses a Node.js base image, installs dependencies, and configures the server port.

**Step 3: Develop the Docker Compose File**

* Create a docker-compose.yml file in the project root.
* Define services for each component, ensuring correct port mappings and dependencies.

**Step 4: Automation Shell Script**

* Create a Shell script to automate building, tagging, pushing, and deploying images. The script should:
  + Build images for frontend and backend.
  + Tag each image with an appropriate version.
  + Push images to Docker Hub.
  + Update Docker Compose with the new image tags.

Deploy services by running the docker-compose command

# Variables

FRONTEND\_IMAGE="viru786/frontend1"

BACKEND\_IMAGE="viru786/backend1"

VERSION=$(date +%Y%m%d%H%M%S) # Version based on current date and time

# Step 1: Build Docker images

echo "Building Docker images..."

docker build -t ${FRONTEND\_IMAGE}:${VERSION} ./devops-task/frontend

docker build -t ${BACKEND\_IMAGE}:${VERSION} ./devops-task/backend

# Step 2: Tag images with appropriate versions

echo "Tagging images..."

docker tag ${FRONTEND\_IMAGE}:${VERSION} ${FRONTEND\_IMAGE}:latest

docker tag ${BACKEND\_IMAGE}:${VERSION} ${BACKEND\_IMAGE}:latest

# Step 3: Push images to Docker Hub

echo "Pushing images to Docker Hub..."

docker push ${FRONTEND\_IMAGE}:${VERSION}

docker push ${FRONTEND\_IMAGE}:latest

docker push ${BACKEND\_IMAGE}:${VERSION}

docker push ${BACKEND\_IMAGE}:latest

# Step 4: Update docker-compose.yml file with new image versions

echo "Updating docker-compose.yml with new image versions..."

sed -i "s|${FRONTEND\_IMAGE}:.\*|${FRONTEND\_IMAGE}:${VERSION}|g" docker-compose.yml

sed -i "s|${BACKEND\_IMAGE}:.\*|${BACKEND\_IMAGE}:${VERSION}|g" docker-compose.yml

# Step 5: Run Docker Compose to start the application

echo "Starting the application with Docker Compose..."

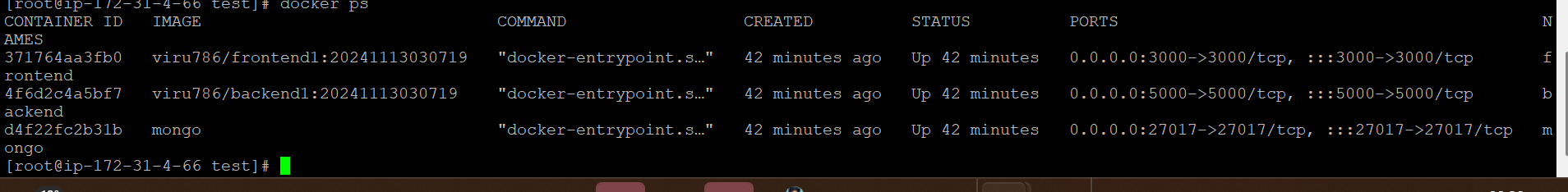
docker-compose down # Stop any existing containers

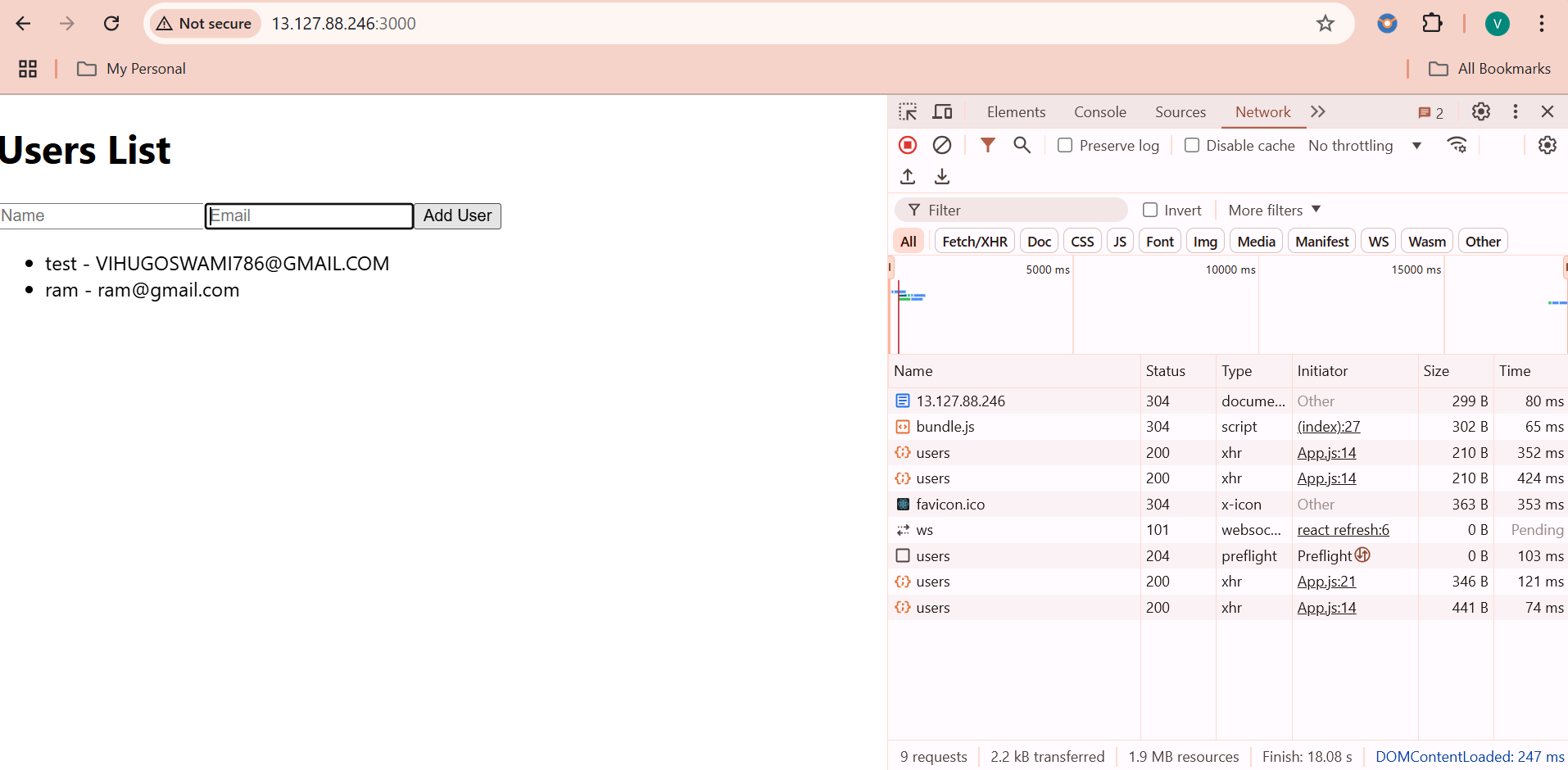
docker-compose up -d # Start new containers with latest images

echo "Application is up and running with updated images!"

**Step 5: Test the Containerized Application Locally**

* Run docker-compose up to verify that all services are correctly interacting and that the application works as expected.





**6. Conclusion**

This document provides the full outline for containerizing a full-stack application with Docker. Following the steps above should enable efficient and repeatable deployment for the application. Thanks!

Top of Form

Bottom of Form