ONLINE SHOPPING PLATFORM

Team 41

NAME	ID
نور محمد محمد عبدالعزيز الحوت	20191700702
اسراء جابر هاني محمد	20191700094
محمد عبدالله الحكيم سالم	20191700835
محمد أيمن فاروق عبدالعزيز	20191700511
علي الدين عمر علي عبدالفتاح	20191700389
سيف أيمن أحمد عبدالخالق	20201700391

INTRODUCTION:

This documentation presents a detailed overview of a Dockerized E-Commerce Web Application project. The primary objective of this project is to containerize a full-stack web application using Docker containers, enhancing its portability and simplifying deployment across diverse environments. The project leverages multiple Dockerfiles, Docker Compose configurations, and Kubernetes deployment setups.

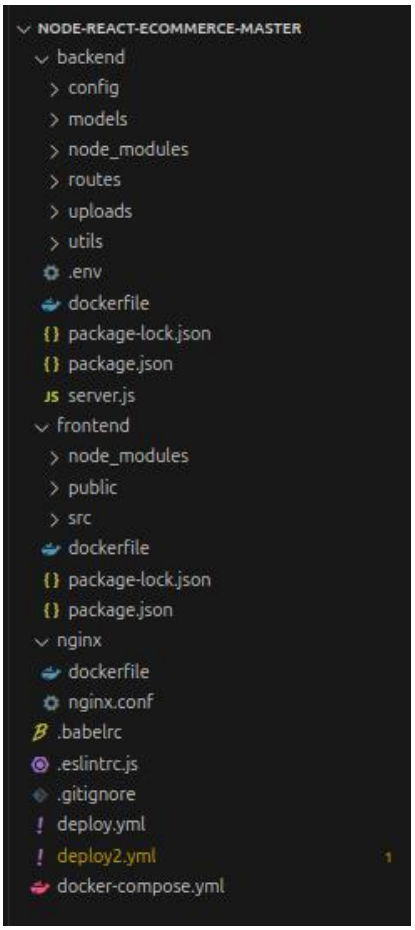
PURPOSE:

The purpose of this project is to illustrate the process of containerizing and deploying an application using Docker and Kubernetes. Containerizing the application facilitates dependency management, component scalability, and consistent deployment across various environments.

PROJECT STRUCTURE:

Our project is composed of four main points, here's an overview of each of them:

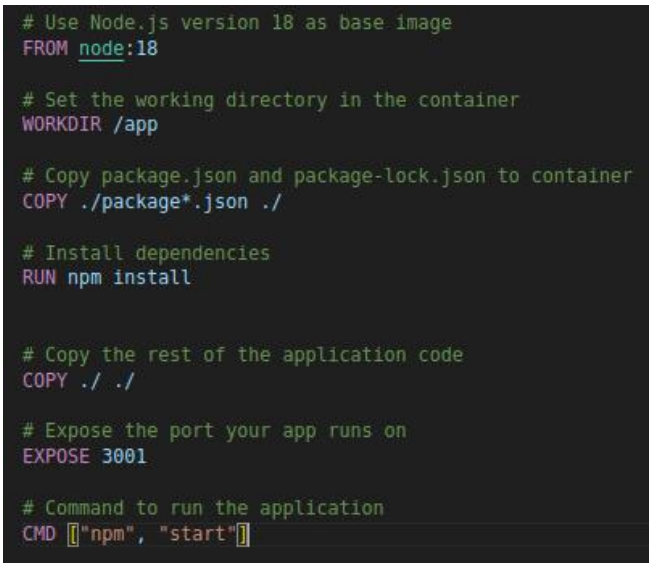
- **BACKEND:** A Node.js application handling server-side logic with its dependencies in the **package.json** file and with important environment variables to connect to MongoDB and other purposes. It can be reached through port number 3001.
- **FRONTEND:** A React.js application responsible for the client-side interface with its dependencies in **package.json**. It can be reached through port number 3000.
- **NGINX:** Acts as a reverse proxy and handles routing to the backend and frontend services.
- **CONFIGURATION FILES:**
  1. Dockerfiles for building images.
  2. Docker Compose files for orchestrating containers.
  3. Kubernetes deployment files.



DOCKERFILES CONFIGURATION:

The Dockerfile defines the configuration for building Docker images as it includes the necessary steps to install dependencies, copy source code and expose ports. Three different docker files were configured:

- **Docker File for Backend:**



```
# Use Node.js version 18 as base image
FROM node:18

# Set the working directory in the container
WORKDIR /app

# Copy package.json and package-lock.json to container
COPY ./package*.json ./

# Install dependencies
RUN npm install


# Copy the rest of the application code
COPY ./ ./

# Expose the port your app runs on
EXPOSE 3001

# Command to run the application
CMD ["npm", "start"]
```

  - **FROM node:18**
    Specifies the base image to be installed.
  - **WORKDIR /app**
    Sets the working directory inside the container as all subsequent commands will be run in this directory.
  - **COPY ./package*.json ./**
    Copies **package.json** and **package-lock.json** the local machine to the working directory in the container for dependency installation.
  - **RUN npm install**
    Installs the application's dependencies listed in package.json.
  - **COPY ./ ./**
    Copies the rest of the application code from the local machine to the working directory in the container.
  - **EXPOSE 3001**
    Informs Docker that the container will listen on port 3001, this is the port that the application will run on.
  - **CMD ["npm", "start"]**
    Runs the Node.js application.

- **Docker File for Frontend:**
  Same commands are used as the ones in the backend, except for the port EXPOSE command:

```
# Use Node.js version 14 as base image
FROM node:18

# Set the working directory in the container
WORKDIR /app

# Copy package.json and package-lock.json to container
COPY ./package*.json ./

# Install dependencies
RUN npm install

# Copy the rest of the application code
COPY ./ ./

# Expose the port your app runs on
EXPOSE 3000

# Command to run the application
CMD ["npm", "start"]
```

- **Docker File for nginx:**

```
# Use NGINX base image
FROM nginx:latest

# Remove default NGINX configuration
RUN rm /etc/nginx/conf.d/default.conf

# Copy custom NGINX configuration
COPY nginx.conf /etc/nginx/conf.d/

# Expose port 80
EXPOSE 80

# Start NGINX
CMD ["nginx", "-g", "daemon off;"]s
```

  - **FROM nginx:latest**
    Specifies the base image, it runs npm start which should start the “Node.js” application.
  - **RUN rm /etc/nginx/conf.d/default.conf**
    Removes the default NGINX configuration.
  - **COPY nginx.conf /etc/nginx/conf.d/**
    Copies the custom NGINX configuration file.
  - **EXPOSE 80**
    Informs Docker that the container will listen on port 80.
  - **CMD ["nginx”, “-g”, “daemon off;”]s**
    Runs NGINX which should start the Node.js application.

NGINX CONFIGURATION FILE:

```
upstream backend {
    server backend:3001;
}
upstream frontend {
    server frontend:3000;
}
server {
    listen 80;
    server_name localhost;

    location / {
        # Forward requests to the frontend service
        proxy_pass http://frontend;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }

    location /api {
        # Forward requests to the backend service
        proxy_pass http://backend;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }
}
```

- **Define upstream server configuration:** The upstream directives define backend and frontend server groups for load balancing.
- **Server Configuration:**
    - **listen 80;:** Listens on port 80 for incoming HTTP requests.
    - **server_name localhost;:** Defines the server name as localhost.

DOCKER COMPOSE:

The Docker Compose file orchestrates the three main services: backend, frontend and nginx.

```yaml
version: '3.8'
services:
  backend:
    build:
      context: ./backend
      dockerfile: dockerfile
    ports:
      - "3001:3001"
    volumes:
      - ./backend:/app

  frontend:
    build:
      context: ./frontend
      dockerfile: dockerfile
    ports:
      - "8080:3000"
    volumes:
      - ./frontend:/app

  nginx:
    build:
      context: ./nginx
      dockerfile: Dockerfile
    ports:
      - "80:80"
    depends_on:
      - frontend
      - backend
```

- **version:** Specifies the version of the Docker Compose file format.
- **services:** Defines the individual services (backend, frontend, nginx) that make up the application.
    - **backend:**
        - **build:** Specifies the build context and Dockerfile location.
        - **ports:** Maps port 3001 of the container to port 3001 on the host machine.
        - **volumes:** Mounts the backend directory to the /app directory inside the container.
        - **environment:** Sets the environment variables for the container.

- **frontend:**
  - **build:** Specifies the build context and Dockerfile location.
  - **ports:** Maps port 3000 of the container to port 8080 on the host machine.
  - **volumes:** Mounts the frontend directory to the /app directory inside the container.
  - **environment:** Sets the environment variables for the container.
- **nginx:**
  - **build:** Specifies the build context and Dockerfile location.
  - **ports:** Maps port 80 of the container to port 80 on the host machine.
  - **depends_on:** Ensures that the backend and frontend services are started before nginx.

KUBERNETES:

The Kubernetes configuration deploys the combined services as a single deployment with multiple containers.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: combined-deploy
spec:
  replicas: 1
  selector:
    matchLabels:
      app: combined
  strategy:
    type: RollingUpdate
  template:
    metadata:
      labels:
        app: combined
    spec:
      containers:
      - name: frontend
        image: badawyx/cloud_project:v1
        ports:
        - containerPort: 3000
        command: ["npm", "run", "start"]
      - name: backend
        image: badawyx/cloud_project:v2
        ports:
        - containerPort: 3001
        command: ["npm", "run", "start"]
      - name: nginx
        image: badawyx/my-nginx-image:latest
        ports:
        - containerPort: 80
```

- **apiVersion:** Specifies the API version of the Kubernetes object.
- **kind:** Defines the type of Kubernetes object (Deployment).
- **metadata:** Provides metadata for the deployment, such as the name.
- **spec:** Defines the specification for the deployment.
    - **replicas:** Specifies the number of pod replicas.
    - **selector:** Defines how to select pods for the deployment.
    - **strategy:** Specifies the deployment strategy (RollingUpdate).
    - **template:** Defines the pod template.

- **metadata:** Provides metadata for the pods, such as labels.
- **spec:** Defines the pod specification.
    - **containers:** Specifies the containers within the pod.
        - **name:** Defines the name of the container.
        - **image:** Specifies the Docker image for the container.
        - **ports:** Maps container ports to the host.
        - **command:** Defines the command to run in the container.

SYSTEM ARCHITECTURE DIAGRAM: