Report of Project
DevOps

Intervenant:

Efrei Paris M1 SE1 Grp4

Students:

CHOUKROUN Simon

1. Introduction:

In this assignment, we will be deploying a MERN (MongoDB, Express.js, React.js, Node.js) stack application using Docker. We'll use Vagrant to set up a virtual machine, Docker to containerize the MERN application components, and Jenkins to build a Continuous Integration/Continuous Deployment (CI/CD) pipeline to automate the deployment process from a GitHub repository.

1. Setting Up the Vagrant Virtual Machine:

Vagrant is a tool that allows you to create and manage lightweight, reproducible, and portable development environments. Using Vagrant to set up a virtual machine ensures that the development environment remains isolated from the host machine. This isolation prevents potential conflicts and ensures a consistent environment for all team members, reducing the chances of "It works on my machine" issues. In this step, we are setting up a virtual machine using Vagrant to provide an isolated and consistent environment for our project.

- Install Vagrant: Download and install Vagrant on your local machine from the official website.

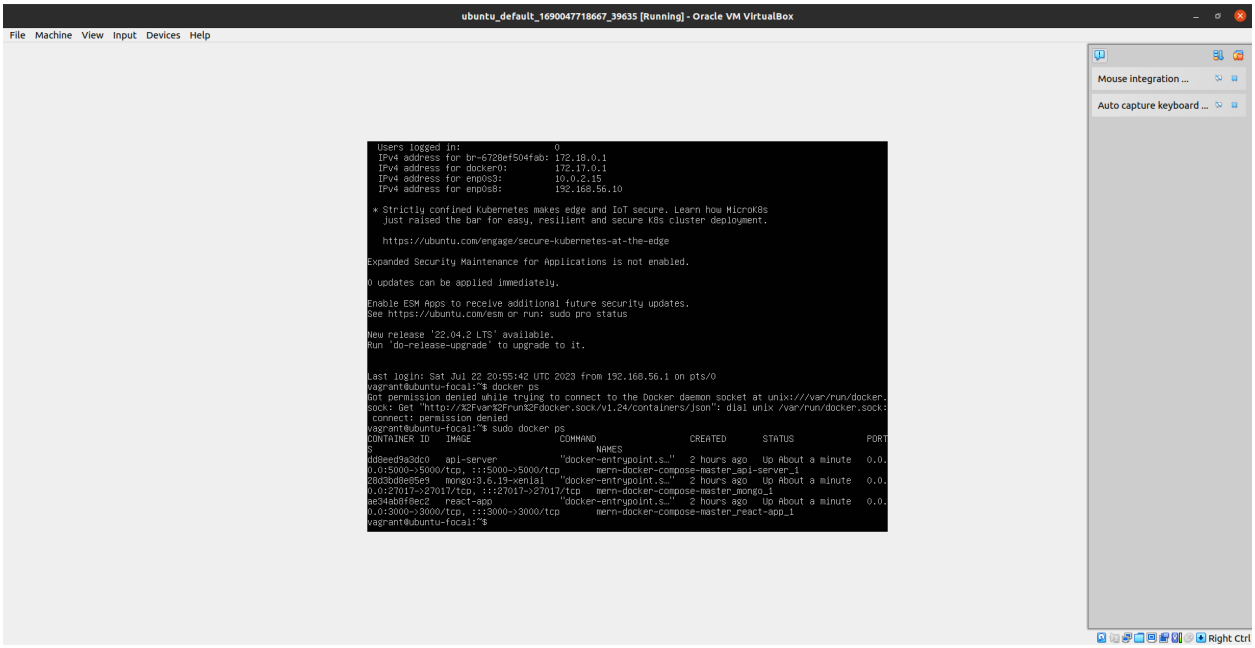
- Vagrantfile: Create a file named "Vagrantfile" in your project directory. This file contains the configuration settings for your virtual machine.

- VM Box: In the Vagrantfile, specify the base box for the VM. Here, we are using the "ubuntu/bionic64" box, which is an official Ubuntu 18.04 LTS image.

- Private Network: Set up a private network so that the VM has its IP address accessible only within the host machine. This allows you to access the VM's services from your local machine.

- Provider Configuration: In this case, we are using VirtualBox as the VM provider. We allocate resources like memory (2048 MB) and CPU cores (2) to the VM.

- Start the VM: Run vagrant up in the project directory to create and provision the VM based on the Vagrantfile.



2. Dockerizing the MERN Stack Application:

Docker allows us to containerize our MERN stack application components, making it easier to deploy and manage. By containerizing the MERN stack application components with Docker, we achieve better portability and reproducibility. Docker containers encapsulate the application and its dependencies, making it easier to deploy the application on different environments without worrying about compatibility issues.

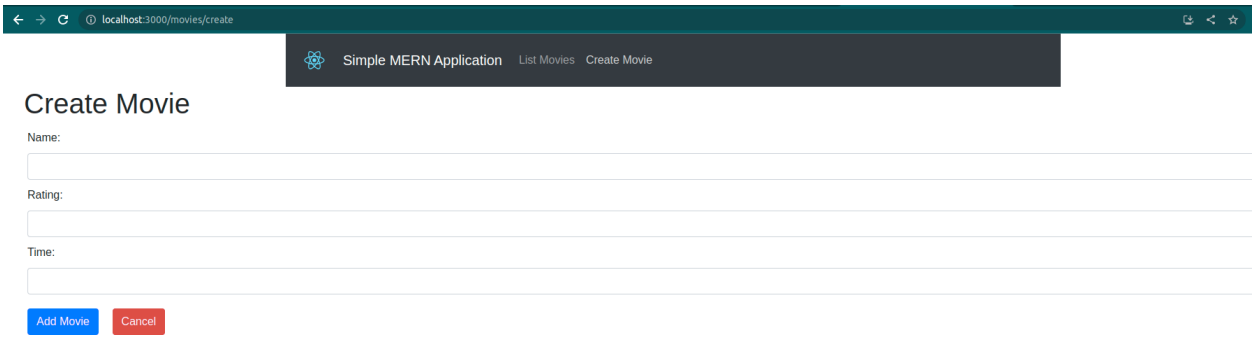
Clone the MERN Application: Clone your MERN stack application's source code from the GitHub repository inside the Vagrant VM. This source code includes the server and client components.

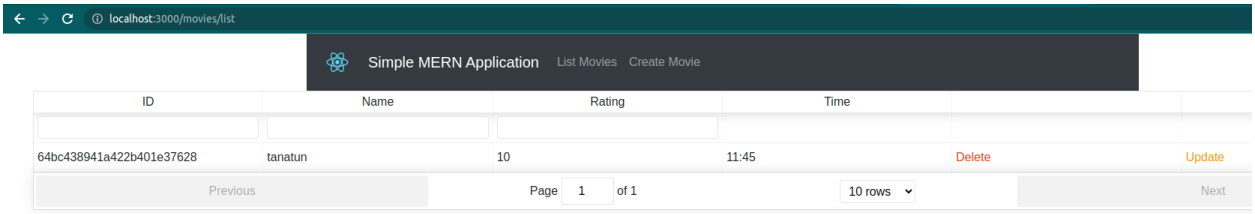
- Dockerfiles: Write separate Dockerfiles for the server and client components. Dockerfiles are used to define the steps to build a Docker image for each component.

- Docker Compose: Create a docker-compose.yml file in the root directory of your project. This file defines the services needed for the MERN application, such as MongoDB, Node.js server, and React.js client. It also specifies how these services should interact with each other.

- Make Scripts: Write make scripts to automate the build and run process for both the server and client containers. These scripts should include commands to build the Docker images and run the containers with proper configurations. The use of Docker and Makefiles simplifies the development workflow. Developers can use the provided make

scripts to build and run the Docker containers for the server and client, streamlining the process of running the application locally.





3. Implementing the CI/CD Pipeline with Jenkins:

Jenkins is an automation server that facilitates continuous integration and continuous deployment.

Install Jenkins: Install Jenkins on the Vagrant VM using the package manager appropriate for the VM's OS. Access the Jenkins web interface through your browser by navigating to http://localhost:8080 (assuming you have forwarded the VM's port 8080 to your host machine).

Install Plugins: Install Jenkins plugins as needed to integrate with Git and Docker.

Configure Jenkins: Set up Jenkins by creating an admin user and configuring basic settings.

Create Jenkins Job: Create a new Jenkins job and select "Pipeline" as the type. Configure the job to pull the MERN application's source code from the GitHub repository.

Deploy Stage: In this stage, deploy the built Docker containers to the production environment. This environment could be another Vagrant VM, a cloud server, or any other target environment where you want to run the MERN application.

The CI/CD pipeline implemented with Jenkins automates the process of building, testing, and deploying the MERN application. Whenever new changes are pushed to the GitHub repository, Jenkins automatically triggers the pipeline, ensuring that each change undergoes thorough testing and can be quickly deployed to the production environment. Automating the deployment process

through the CI/CD pipeline ensures that the latest version of the application is deployed consistently and reliably to the production environment. This reduces the risk of human errors during manual deployments.

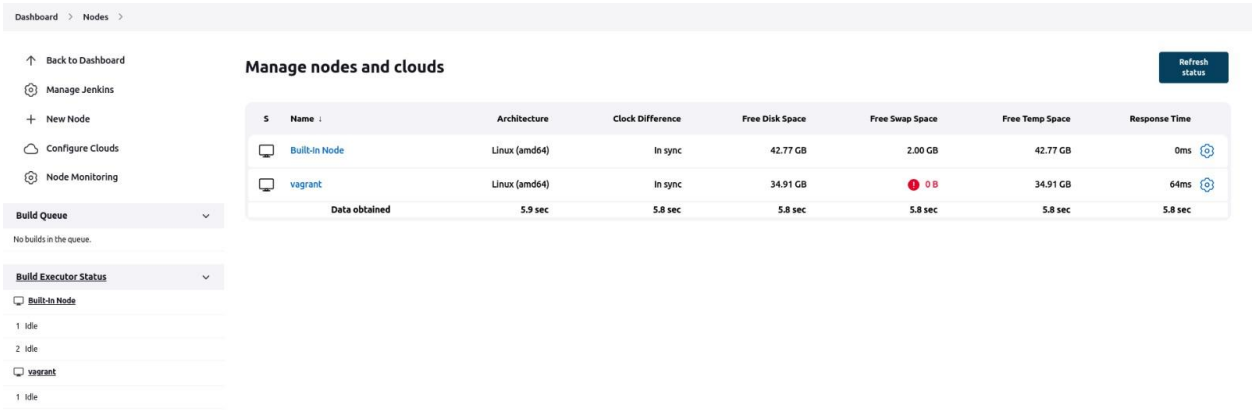
4. CI/CD Pipeline Workflow:

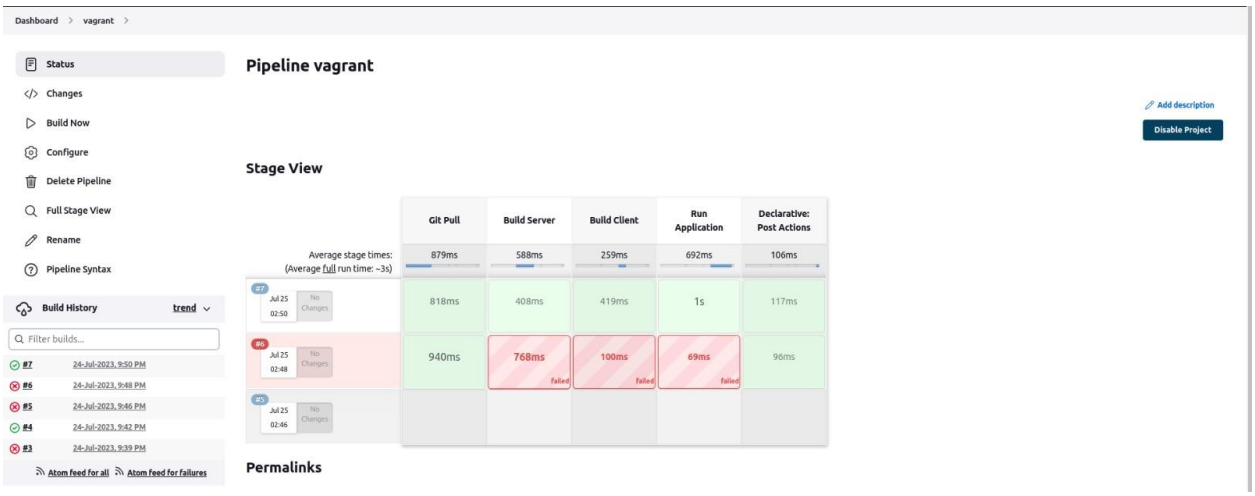
The CI/CD pipeline workflow is as follows:

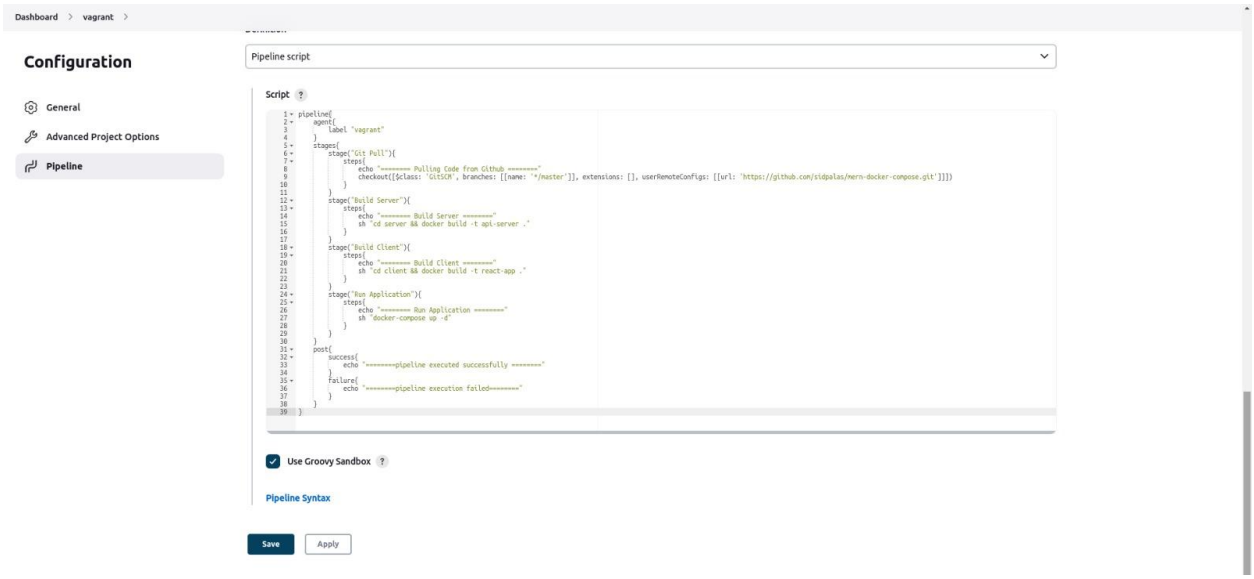
- A developer pushes code changes to the GitHub repository.

- Jenkins detects the new commits and automatically triggers the CI/CD pipeline configured for the repository.

- The Build Stage is executed first. Jenkins uses the provided make scripts or Docker commands to build the Docker images for the server and client components.







With each iteration, the development team can make incremental changes, test them, and deploy them quickly. This enables the team to respond promptly to user feedback and market demands. By using Jenkins as the automation server, the entire team gains visibility into the CI/CD pipeline's status. Developers, testers, and other stakeholders can access the Jenkins dashboard to monitor the progress of builds and deployments, fostering collaboration and transparency within the team.

Approach

By following these detailed steps, you have successfully deployed a MERN stack application using Docker and created a CI/CD pipeline with Jenkins. This approach allows for more streamlined development, testing, and deployment, resulting in a more efficient and reliable software development process.

1. Setting Up the Vagrant Virtual Machine:

a. Install Vagrant: Vagrant is a tool that allows developers to create and manage virtual machines in a consistent and reproducible manner. To install Vagrant, visit the official website (https://www.vagrantup.com/) and download the appropriate installer for your operating system. Install it like any other software on your machine.

b. Vagrantfile: A Vagrantfile is a configuration file written in Ruby that defines the settings and specifications for your virtual machine. You can create a new file named "Vagrantfile" in your project directory, and Vagrant will automatically use it when you run Vagrant commands.

c. VM Box: In the Vagrantfile, you specify the base box for your virtual machine. A box is a pre-packaged, ready-to-use virtual machine image. For example, you can use the "ubuntu/bionic64" box, which is an official Ubuntu 18.04 LTS image.

d. Private Network: By adding the following line to the Vagrantfile, you set up a private network for your virtual machine:

This allows your VM to have its IP address within the private network, enabling you to access services running inside the VM from your local machine.

e. Provider Configuration: In this step, we specify the provider (e.g., VirtualBox, VMware) and configure resource settings such as memory and CPU cores allocated to the VM. For example, using VirtualBox as the provider:

f. Start the VM: After writing the Vagrantfile, navigate to the project directory in your terminal and run the following command to create and provision the virtual machine:

vagrant up

2. Dockerizing the MERN Stack Application:

a. Clone the MERN Application: Ensure you have the source code of your MERN stack application hosted on a Git repository. Inside the Vagrant VM, clone the repository using the git clone command.

b. Dockerfiles: Dockerfiles are configuration files used to build Docker images. You need to create two Dockerfiles, one for the server and one for the client.

The server Dockerfile should start with a base image (e.g., node:14) and copy the server source code into the image. It should also install dependencies, set environment variables, and specify the command to run the server.

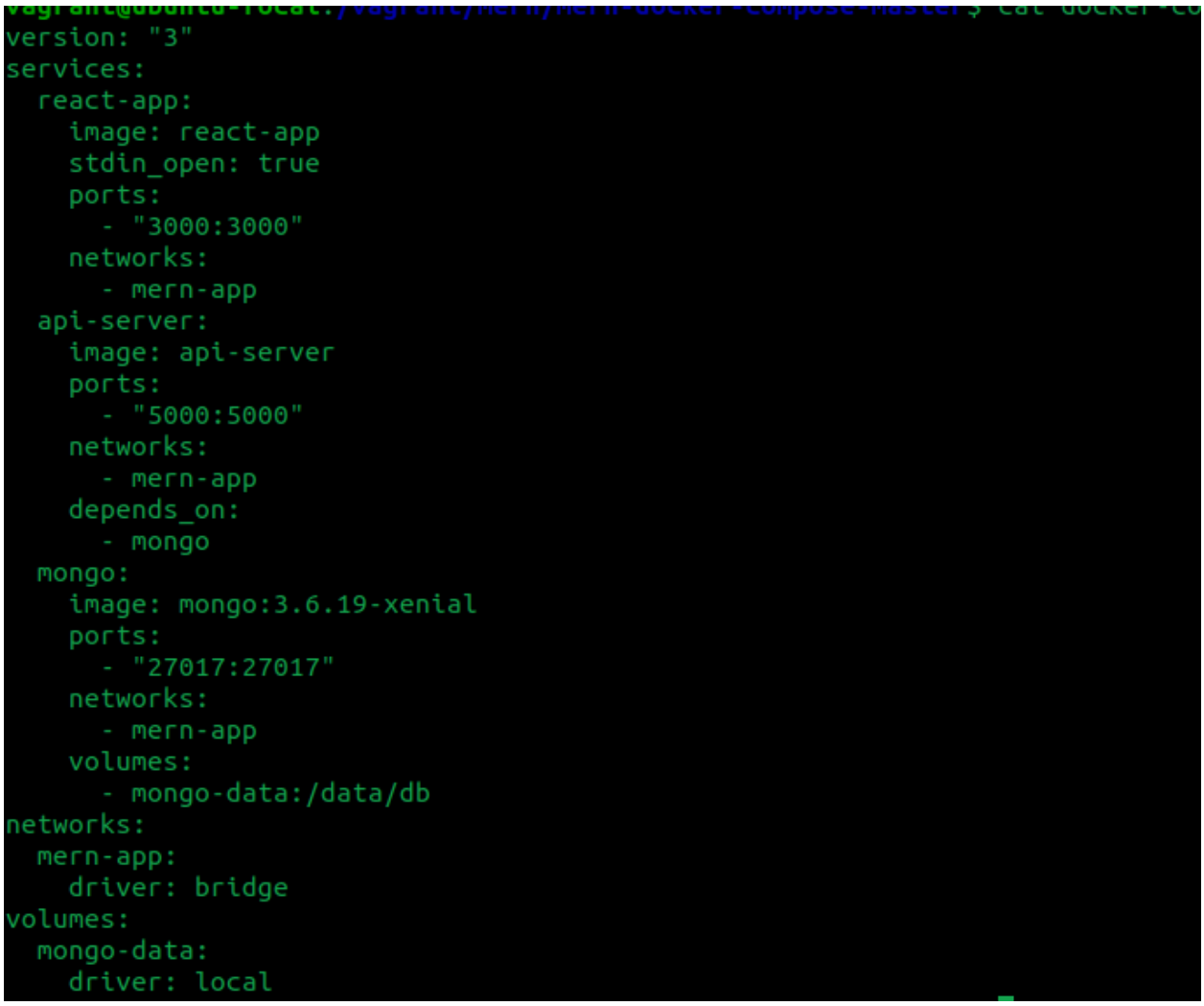
The client Dockerfile should start with a base image (e.g., node:14) and copy the client source code into the image. It should also install dependencies and specify the command to build the client application.

c. Docker Compose: Docker Compose is a tool for defining and running multi-container Docker applications. Create a file named docker-compose.yml in the root directory of your project.

Define three services in the docker-compose.yml file: one for the MongoDB database, one for the server, and one for the client. Specify their respective Dockerfile paths, ports to expose, and any other required configurations.

d. Make Scripts: A Makefile is a simple way to organize and automate tasks. Write make scripts to simplify building and running the Docker containers for the server and client.

For example, the Makefile can include targets like build-server, run-server, build-client, run-client, and a target to run the whole MERN stack application.



```
version: "3"
services:
  react-app:
    image: react-app
    stdin_open: true
    ports:
      - "3000:3000"
    networks:
      - mern-app
  api-server:
    image: api-server
    ports:
      - "5000:5000"
    networks:
      - mern-app
    depends_on:
      - mongo
  mongo:
    image: mongo:3.6.19-xenial
    ports:
      - "27017:27017"
    networks:
      - mern-app
    volumes:
      - mongo-data:/data/db
networks:
  mern-app:
    driver: bridge
volumes:
  mongo-data:
    driver: local
```

3. Implementing the CI/CD Pipeline with Jenkins:

a. Install Jenkins: Jenkins is an automation server that allows you to create pipelines for automating various stages of software development, including building, testing, and deployment. To install Jenkins, access the terminal of the Vagrant VM and follow the instructions for your OS to install Jenkins and start its service.

b. Install Plugins: Jenkins supports various plugins to extend its functionality. Install plugins that are needed for integrating with Git repositories and Docker, such as the "GitHub Integration Plugin" and the "Docker Pipeline Plugin."

c. Configure Jenkins: Access Jenkins through your browser by navigating to http://localhost:8080 (assuming port forwarding is set up for the VM's port 8080). Follow the instructions to set up Jenkins, including creating an admin user and configuring basic settings.

d. Create Jenkins Job: Create a new Jenkins job by clicking on "New Item" on the Jenkins dashboard and selecting "Pipeline" as the job type.

e. Pipeline Stages: In the job configuration, navigate to the "Pipeline" section and define the stages for your CI/CD pipeline.

```
pipeline {
        agent any

    tools {
    nodejs 'nodejs'
    }
    stages {
        stage('Cloning Git Repo') {
            steps {
                checkout([$class: 'GitSCM', branches: [[name: '*/main']], extensions: [], userRemoteConfigs: [[credentialsId: 'simon_git_credentials', url: '']
            }
        }

    stage('Building our image') {
        steps{
        script {
        sh 'cd /vagrant/ubuntu/mern/mern-docker-compose-master'
        sh 'make build'
        }
        }
        }
    stage('deploying our images') {
        steps{
        script {
        sh 'cd /vagrant/ubuntu/mern/mern-docker-compose-master'
        sh 'make run'
        }
        }
    }
    }
}
```

4. CI/CD Pipeline Workflow:

- a. A developer pushes code changes to the GitHub repository.

- b. Jenkins, through a webhook or periodic polling, detects the new commits and automatically triggers the CI/CD pipeline configured for the repository.

- c. Jenkins starts executing the pipeline stages one by one, as defined in the Jenkins job configuration.

- d. The Build Stage runs the make scripts or Docker commands to build the Docker images for the server and client components.

- e. If the Build Stage is successful, Jenkins moves on to the Test Stage, where it runs the defined tests to ensure the MERN application behaves correctly.

- f. If all tests pass, Jenkins proceeds to the Deploy Stage. In this stage, Jenkins deploys the latest Docker images to the production environment.

- g. Finally, the Notify Stage is triggered, and Jenkins sends out notifications to relevant stakeholders, informing them about the success or failure of the CI/CD pipeline run.

5. Conclusion:

By following these detailed steps, you have successfully deployed a MERN stack application using Docker and created a CI/CD pipeline with Jenkins. This approach ensures a smooth and automated development and deployment process, making it easier to maintain, test, and deploy the application with minimal manual intervention. In conclusion, this assignment has successfully demonstrated the deployment of a MERN stack application using Docker and the implementation of a Continuous Integration/Continuous Deployment (CI/CD) pipeline with Jenkins. This comprehensive approach provides several benefits and advantages for the development and deployment process.