

Automating Docker Deployment with Jenkins and GitHub Webhooks

Name: Dhinesh kumaran S

RollNo:22CSR048

Step 1: Install Docker on Ubuntu

Run the following commands to install and start Docker:

```
sudo apt update  
sudo apt install -y docker.io  
sudo systemctl start docker  
sudo systemctl enable docker  
sudo systemctl status docker
```

Step 2: Fork a GitHub Repository

1. Go to GitHub and fork the required repository that contains necessary files.
2. This will create a copy of the repository in your GitHub account.

Take a screenshot of the forked repository for reference.

Step 3: Modify deploy.sh with Docker Hub Credentials

1. Open your forked repository.
2. Locate the deploy.sh file.
3. Update the **Docker Hub token** and **repository name** inside deploy.sh.

Save the file and commit changes.

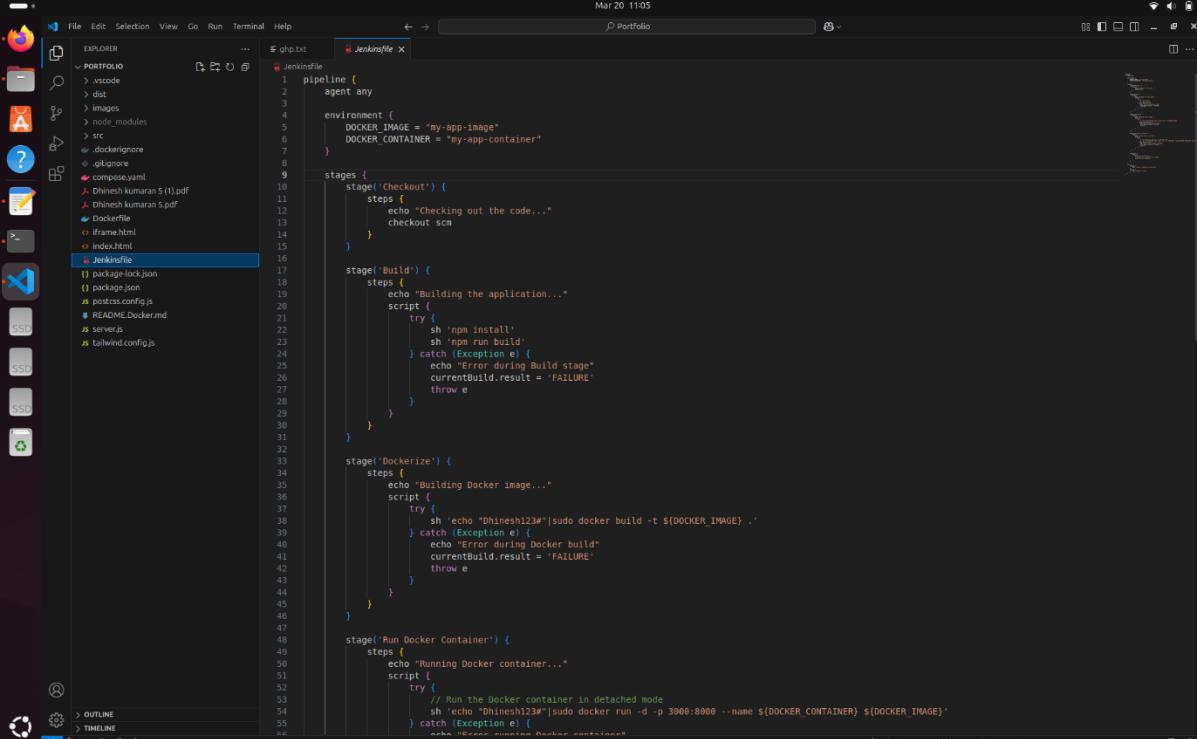
Step 4: Copy the Repository URL for Jenkins

1. Copy the **GitHub repository URL** from your forked repository.
2. This will be used in Jenkins for integration.

Step 5: Create a Pipeline Job in Jenkins

1. Open Jenkins Dashboard → Click New Item.
2. Enter a project name and select Pipeline → Click OK.
3. In the pipeline configuration:
 - o Add the GitHub repository URL.
 - o Set the correct branch.
 - o Specify the Jenkinsfile.

 Save the configuration.

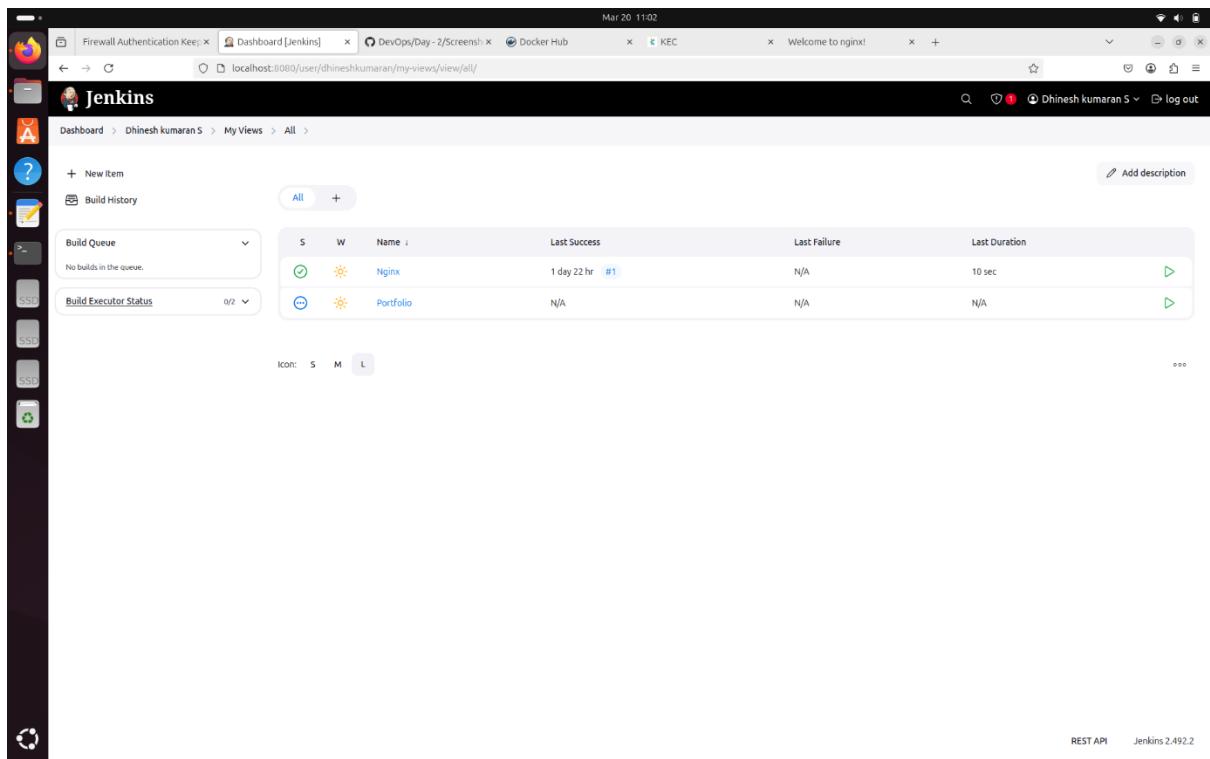


```
File Edit Selection View Go Run Terminal Help
EXPLORER PORTFOLIO Jenkinsfile
Jenkinsfile
1 pipeline {
2     agent any
3
4     environment {
5         DOCKER_IMAGE = "my-app-image"
6         DOCKER_CONTAINER = "my-app-container"
7     }
8
9     stages {
10        stage('Checkout') {
11            steps {
12                echo "Checking out the code..."
13                checkout scm
14            }
15        }
16        stage('Build') {
17            steps {
18                echo "Building the application..."
19                script {
20                    try {
21                        sh 'npm install'
22                        sh 'npm run build'
23                    } catch (Exception e) {
24                        echo "Error during Build stage"
25                        currentBuild.result = 'FAILURE'
26                        throw e
27                    }
28                }
29            }
30        }
31        stage('Dockerize') {
32            steps {
33                echo "Building Docker image..."
34                script {
35                    try {
36                        sh 'echo "Dhinesh123#|sudo docker build -t ${DOCKER_IMAGE} .'
37                    } catch (Exception e) {
38                        echo "Error during Docker build"
39                        currentBuild.result = 'FAILURE'
40                        throw e
41                    }
42                }
43            }
44        }
45        stage('Run Docker Container') {
46            steps {
47                echo "Running Docker container..."
48                script {
49                    try {
50                        // Run the Docker container in detached mode
51                        sh 'echo "Dhinesh123#|sudo docker run -d -p 3000:8000 --name ${DOCKER_CONTAINER} ${DOCKER_IMAGE}''
52                    } catch (Exception e) {
53                        echo "Error during Docker container"
54                    }
55                }
56            }
57        }
58    }
59
60 }
```

Step 6: Build the Pipeline

1. Click Build Now to execute the job.
2. The console output will show the process, and the Docker image will be created.

 Ensure the image is successfully built.



Step 7: Run the Docker Container and Host the Application

1. Open a terminal and run the following command:

```
docker run -d -p <PORT_NUMBER>:80 <IMAGE_NAME>
```

2. Open a browser and visit:

```
http://localhost:<PORT_NUMBER>
```

Your application should now be accessible.

Step 8: Use Docker Compose for Automation (Optional)

Instead of manually running the image, create a docker-compose.yml file:

```
yaml
```

```
CopyEdit
```

```
version: '3'
```

```
services:
```

```
react-capstone:
```

```
image: "test1"
```

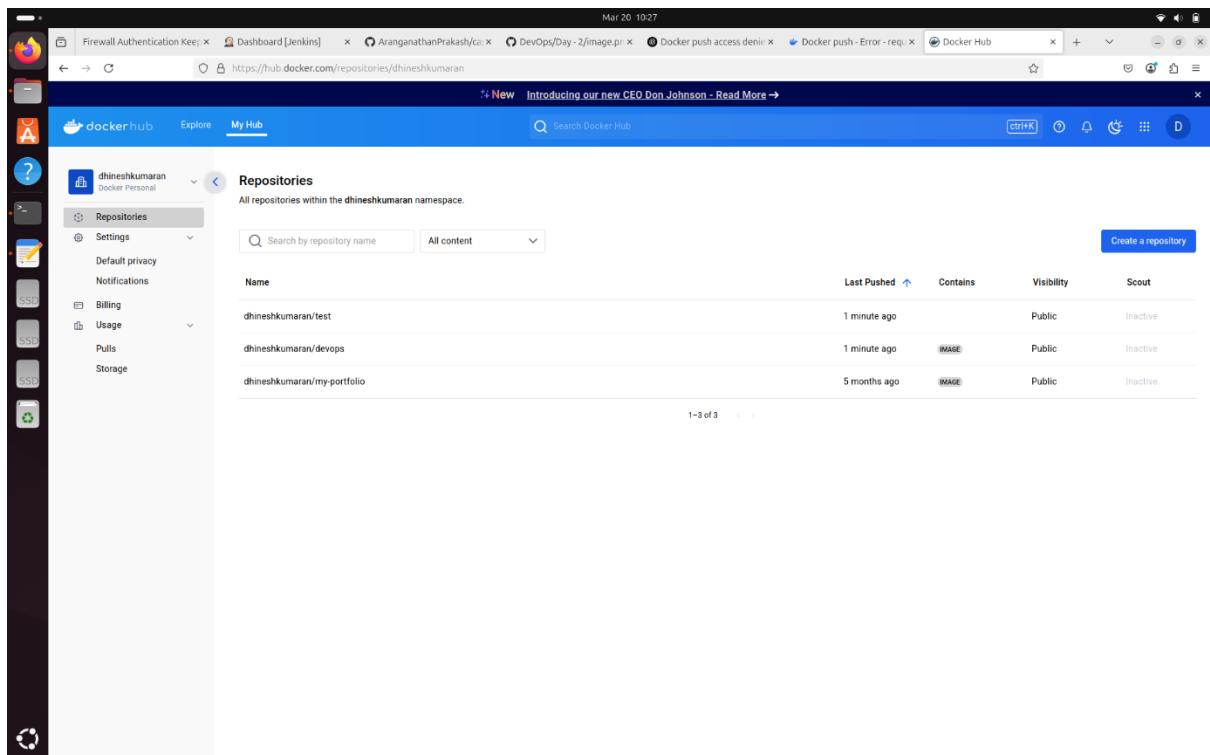
```
ports:
```

```
- "85:80"
```

Run the following command to start the container:

```
docker-compose up -d
```

 **This simplifies deployment.**



Step 9: Install Ngrok for Public Access

1. Sign in to **Ngrok** using GitHub or Google.
2. Install Ngrok:

```
sudo snap install ngrok
```

3. Add your Ngrok authentication token:

```
ngrok config add-authtoken <YOUR_AUTHTOKEN>
```

4. Start an HTTP tunnel:

```
ngrok http 8080
```

 **Copy the generated public URL for remote access.**

Step 10: Add a GitHub Webhook

1. Go to **GitHub Repository** → Click **Settings**.
2. Navigate to **Webhooks** → Click **Add Webhook**.

3. Set the **Payload URL** to:

`http://localhost:8080/github-webhook/`

4. Choose **application/json** as the Content Type.
5. Select **Just the push event** → Click **Add Webhook**.

 **This will trigger Jenkins builds on repository changes.**

Step 11: Configure Jenkins for Webhook Triggering

1. Open **Jenkins** → Click the **Pipeline Job**.
2. Click **Configure** → Go to **Build Triggers**.
3. Select **GitHub Hook Trigger for GITScm Polling**.
4. Click **Save**.

 **Jenkins will now auto-build on GitHub changes.**

Step 12: Commit Changes to Trigger Auto-Build

1. Edit any file in your GitHub repository.
2. Click **Commit Changes**.
3. This will automatically trigger a **new Jenkins build**.

 **Check Jenkins for auto-build confirmation.**

