Final Project Submission- Docker E2 Certification (Edureka)

Spring Boot Microservice CI/CD Workflow

## Overview

This document outlines the steps to develop a Spring Boot microservice, containerize it using Docker, deploy it to a Kubernetes cluster, and set up a GitHub Actions CI/CD pipeline. The pipeline will automatically build, test, containerize, and push the Docker image to a container registry.

Scope of Work

1. Application Development

- Develop a RESTful Spring Boot application with at least one functional endpoint (e.g., `/health`, `/check`, or `/greeting`).

- Ensure the application follows clean coding and configuration practices.

2. Containerization Using Docker

- Create a Dockerfile to build the application as a lightweight, production-ready container image.

- Publish the image to a container registry such as Docker Hub or any other registry.

3. Kubernetes Deployment

- Write Kubernetes manifests for Deployment and Service.

- Ensure the application is scalable within the cluster.

4. GitHub Actions CI/CD Pipeline

- Set up a GitHub Actions pipeline that:

- Builds and tests the Spring Boot application.

- Builds and pushes the Docker image to the container registry.

- Deploys the application to the Kubernetes cluster.

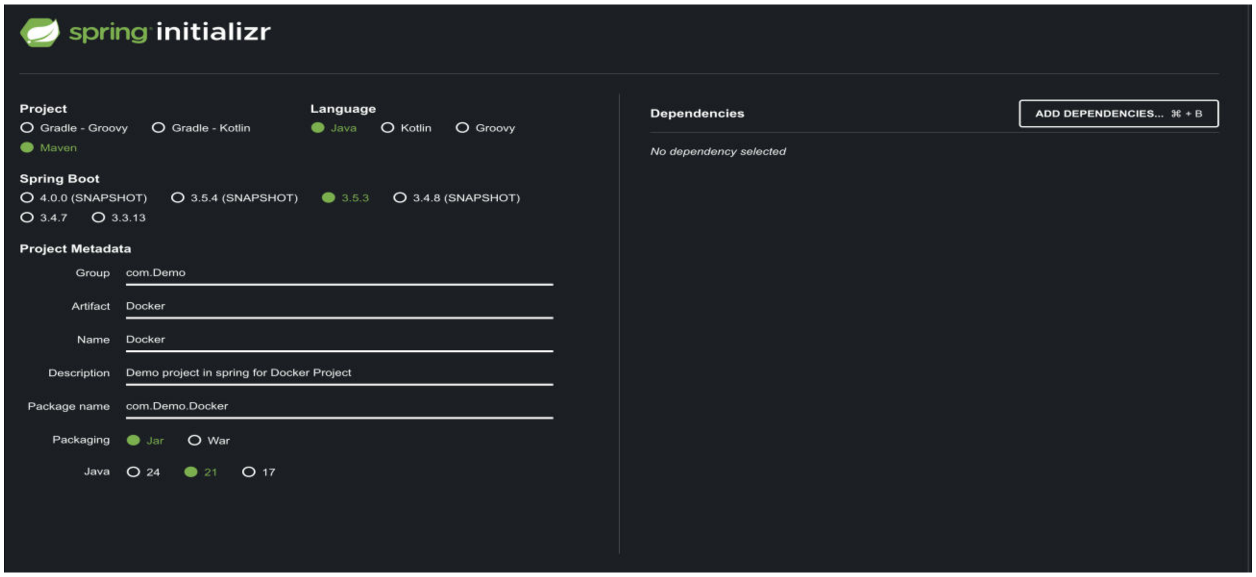
##Steps to Complete the Workflow

### 1.Application Development

- Created a Spring Boot Project:

- I used Spring Initializr (https://start.spring.io/) to generate a new Spring Boot project.

- Include dependencies such as Spring Web and Spring Boot Devtools.


spring initializr

Project
Gradle - Groovy   Gradle - Kotlin   [✓ Maven]

Language
[✓ Java]   Kotlin   Groovy

Spring Boot
4.0.0 (SNAPSHOT)   3.5.4 (SNAPSHOT)   [✓ 3.5.3]   3.4.8 (SNAPSHOT)   3.4.7   3.3.13

Project Metadata

Group	com.Demo
Artifact	Docker
Name	Docker
Description	Demo project in spring for Docker Project
Package name	com.Demo.Docker
Packaging	[✓ Jar] War
Java	24 [✓ 21] 17

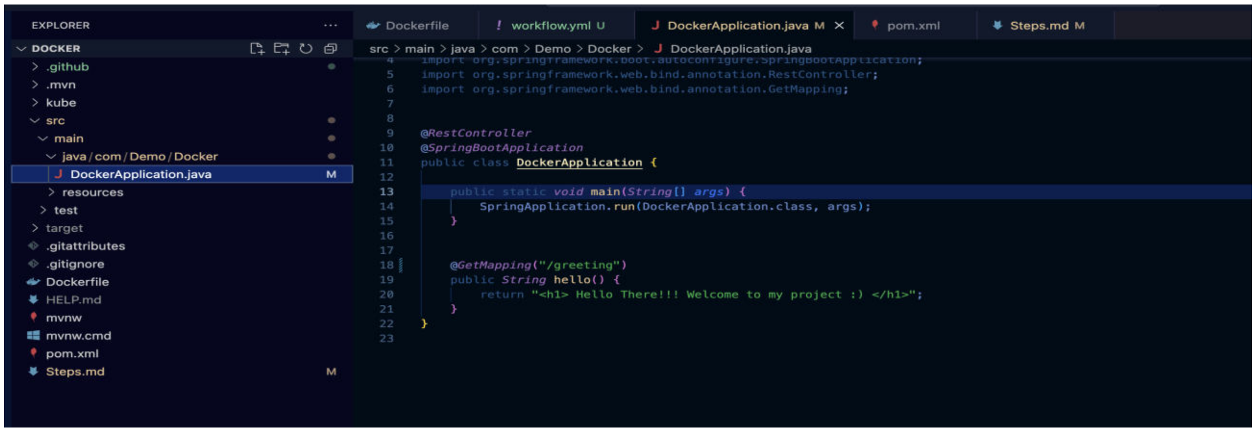
Dependencies

No dependency selected

2.Develop the RESTful Endpoint:

- Create a controller class with a simple endpoint:

added one custom get api endpoint called "/greeting" which shows a greeting message



```java
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.bind.annotation.GetMapping;


@RestController
@SpringBootApplication
public class DockerApplication {

    public static void main(String[] args) {
        SpringApplication.run(DockerApplication.class, args);
    }


    @GetMapping("/greeting")
    public String hello() {
        return "<h1> Hello There!!! Welcome to my project :) </h1>";
    }
}
```

- Clean Coding Practices:

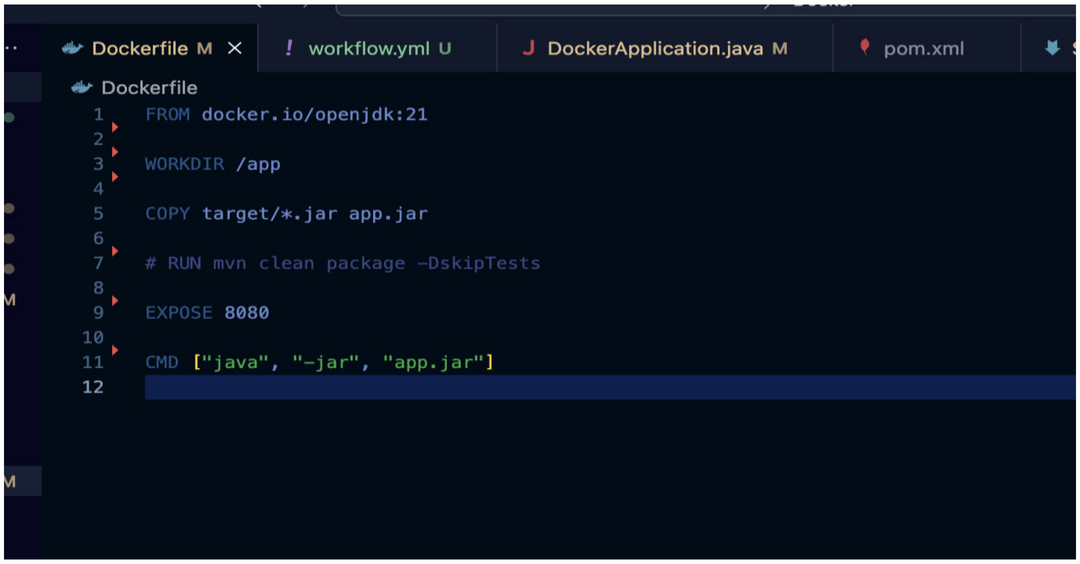
- Follow best practices for code organization, naming conventions, and documentation.

### 2. Containerization

- Create a Dockerfile:

- In the root of your project, create a `Dockerfile` with the following content:

The Dockerfile can be Found in the repo, which uses a java image and copies the build jar file and serves the file

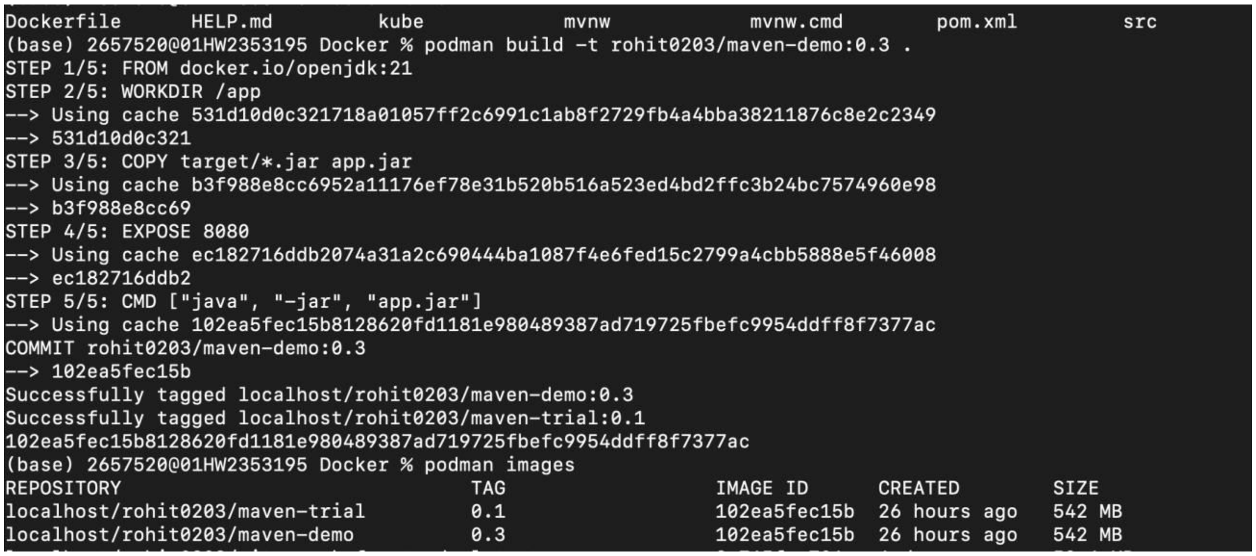


```
FROM docker.io/openjdk:21

WORKDIR /app

COPY target/*.jar app.jar

# RUN mvn clean package -DskipTests

EXPOSE 8080

CMD ["java", "-jar", "app.jar"]
```

- Build the Docker Image:

- Run the following command to build the Docker image:

```bash
docker build -t your-docker hub-username/your-image-name: latest .
```



```
Dockerfile      HELP.md         kube            mvnw            mvnw.cmd        pom.xml         src
(base) 2657520@01HW2353195 Docker % podman build -t rohit0203/maven-demo:0.3 .
STEP 1/5: FROM docker.io/openjdk:21
STEP 2/5: WORKDIR /app
--> Using cache 531d10d0c321718a01057ff2c6991c1ab8f2729fb4a4bba38211876c8e2c2349
--> 531d10d0c321
STEP 3/5: COPY target/*.jar app.jar
--> Using cache b3f988e8cc6952a11176ef78e31b520b516a523ed4bd2ffc3b24bc7574960e98
--> b3f988e8cc69
STEP 4/5: EXPOSE 8080
--> Using cache ec182716ddb2074a31a2c690444ba1087f4e6fed15c2799a4cbb5888e5f46008
--> ec182716ddb2
STEP 5/5: CMD ["java", "-jar", "app.jar"]
--> Using cache 102ea5fec15b8128620fd1181e980489387ad719725fbefc9954ddff8f7377ac
COMMIT rohit0203/maven-demo:0.3
--> 102ea5fec15b
Successfully tagged localhost/rohit0203/maven-demo:0.3
Successfully tagged localhost/rohit0203/maven-trial:0.1
102ea5fec15b8128620fd1181e980489387ad719725fbefc9954ddff8f7377ac
(base) 2657520@01HW2353195 Docker % podman images
REPOSITORY                        TAG                   IMAGE ID      CREATED       SIZE
localhost/rohit0203/maven-trial   0.1                   102ea5fec15b  26 hours ago  542 MB
localhost/rohit0203/maven-demo    0.3                   102ea5fec15b  26 hours ago  542 MB
```

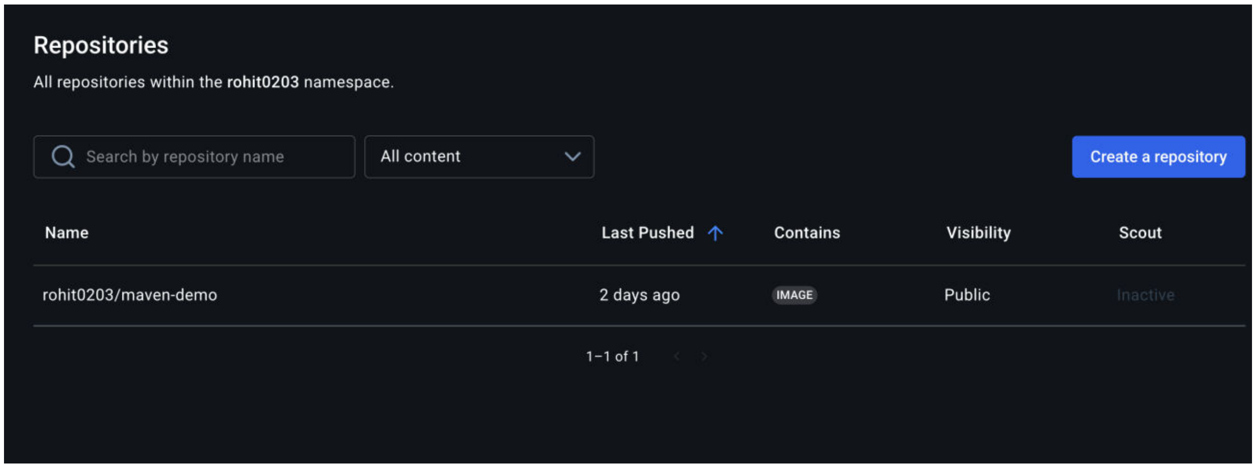
- Push the Docker Image to Docker Hub:

- Log in to Docker Hub:

```bash
docker login
```

- Push the image:

```bash
docker push your-dockerhub-username/your-image-name:latest
```



Repositories

All repositories within the rohit0203 namespace.

Name	Last Pushed	Contains	Visibility	Scout
rohit0203/maven-demo	2 days ago	IMAGE	Public	Inactive

### 3. Kubernetes Deployment

- Create Kubernetes Manifests:

- I have Creates a `kube` directory and added the following YAML files:

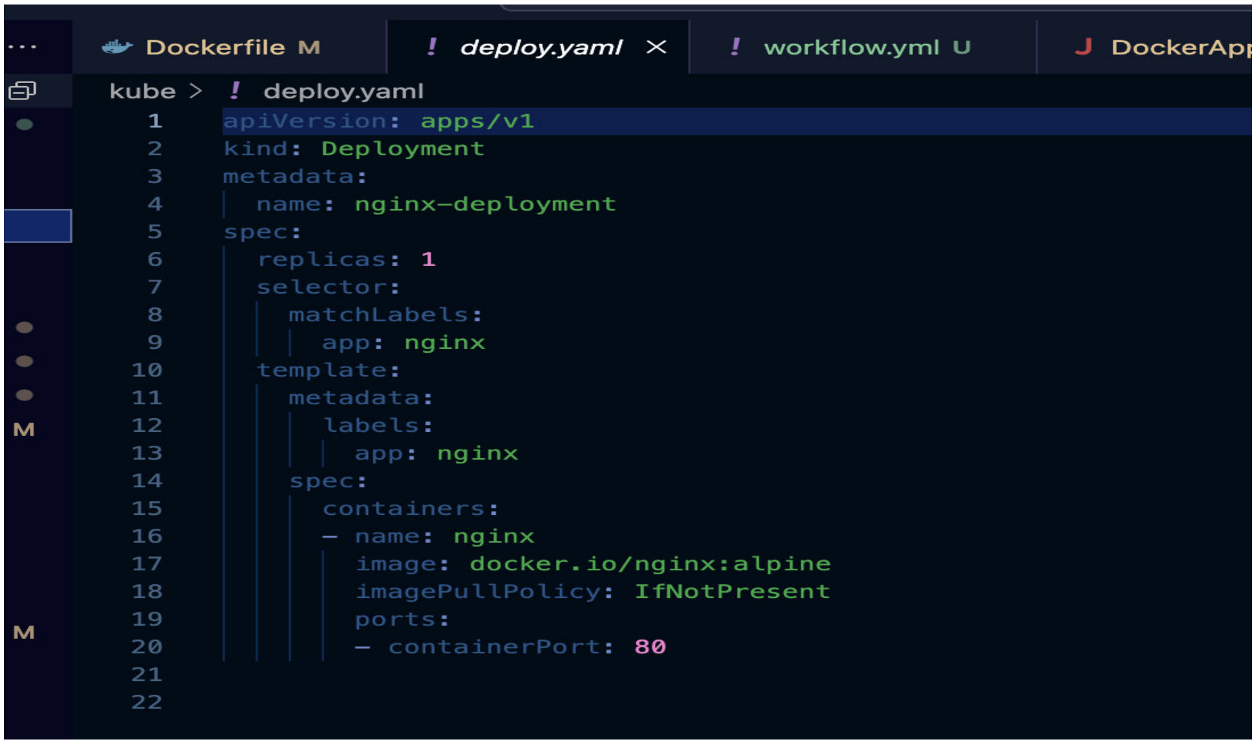
deploy.yaml -> this is a deployment file for the springboot application which runs the container of the application

service.yaml -> This is a service file to expose the application, I have used nodeport , which exposes the app on a particular ip in each node

ingress.yaml -> also one ingress service to expose the app in a hostname

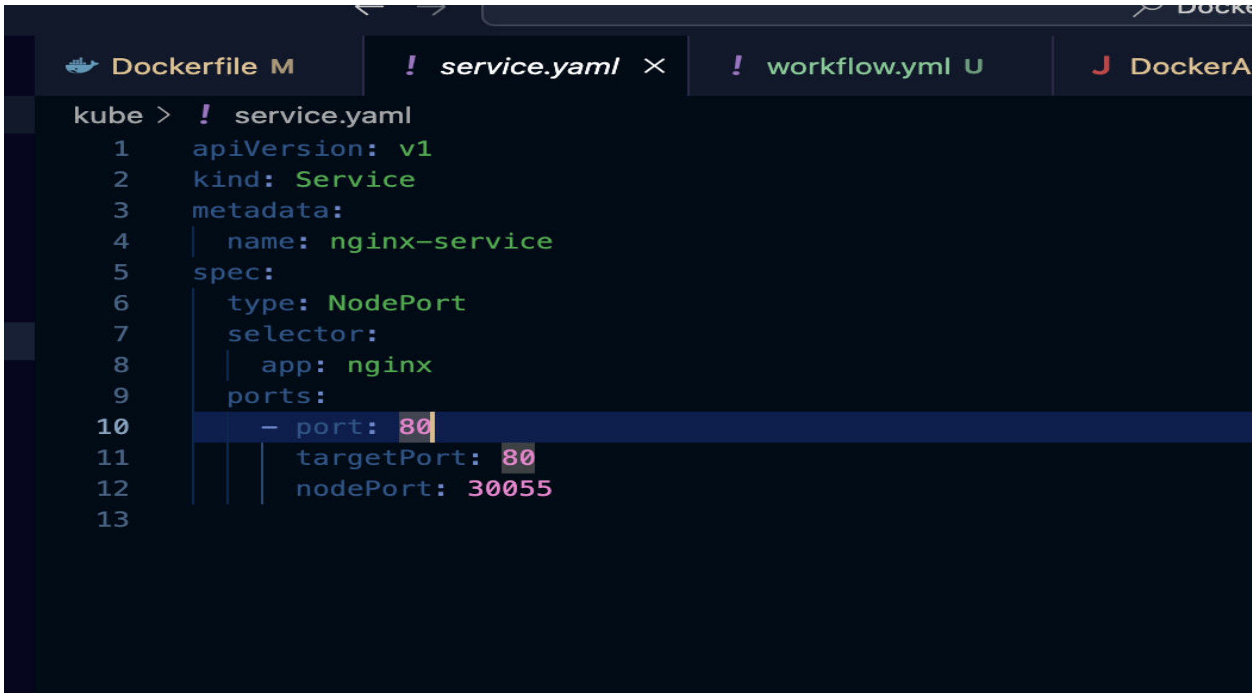
I have used minikube for this development

Deployment File



```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: docker.io/nginx:alpine
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 80


```

Service File

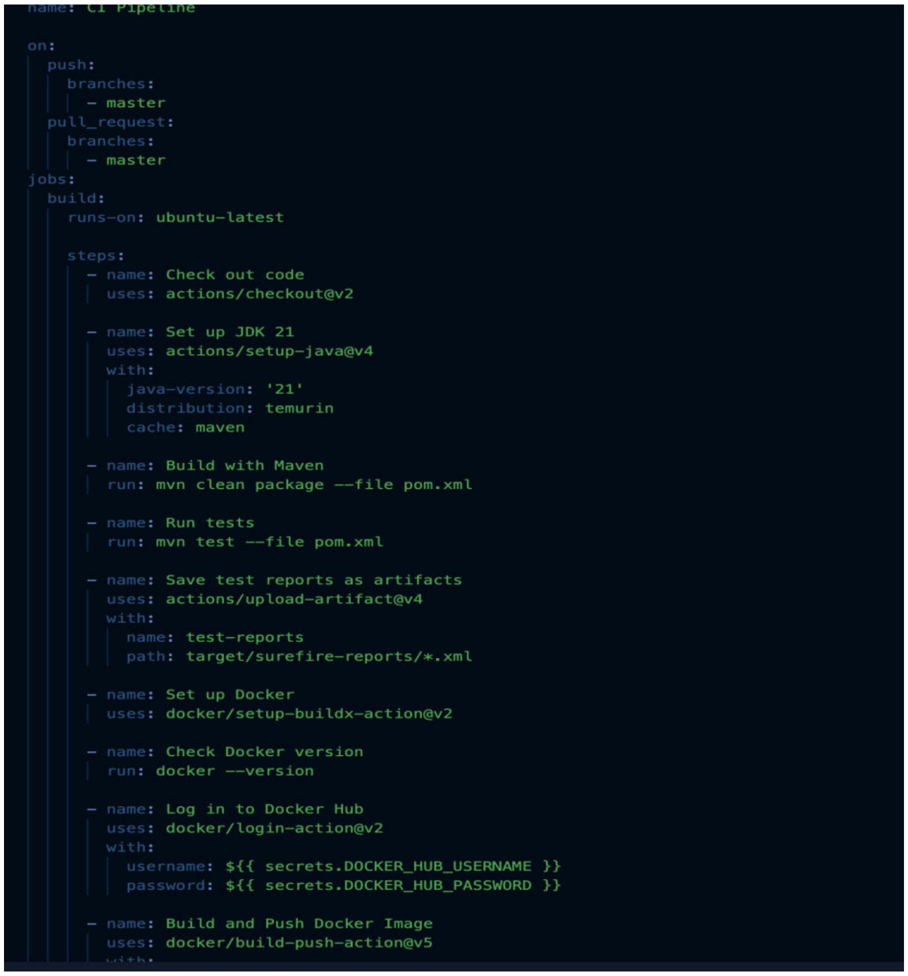


```yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
  type: NodePort
  selector:
    app: nginx
  ports:
    - port: 80
      targetPort: 80
      nodePort: 30055

```

### 4. GitHub Actions CI/CD Pipeline

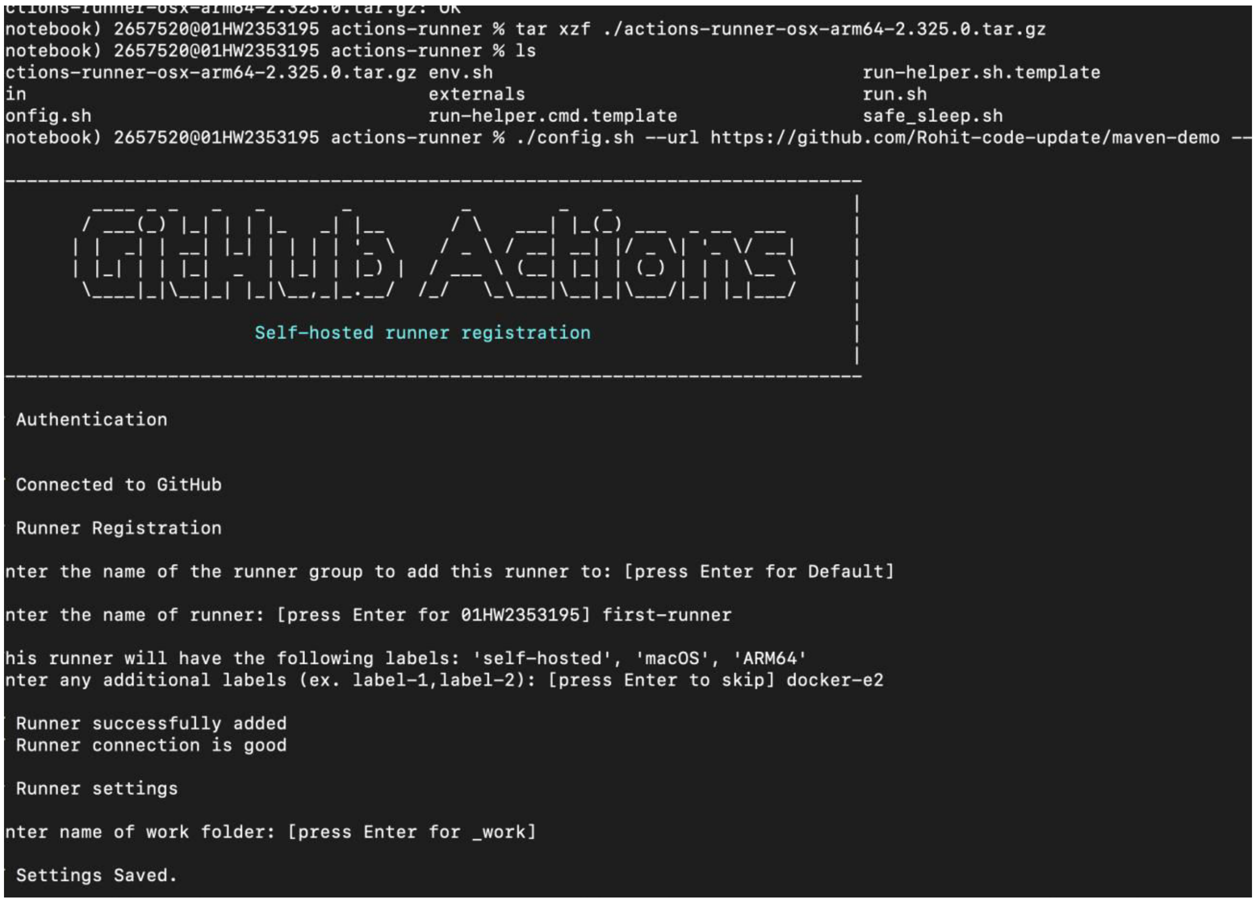
- Create GitHub Actions Workflow:

- In my project, created a directory `.github/workflows` and add a file named `workflow.yml` with the following content:added all the steps needed to create the app and building the image and pushing it to docker hub. I have tried both self-hosted and GitHub runners.



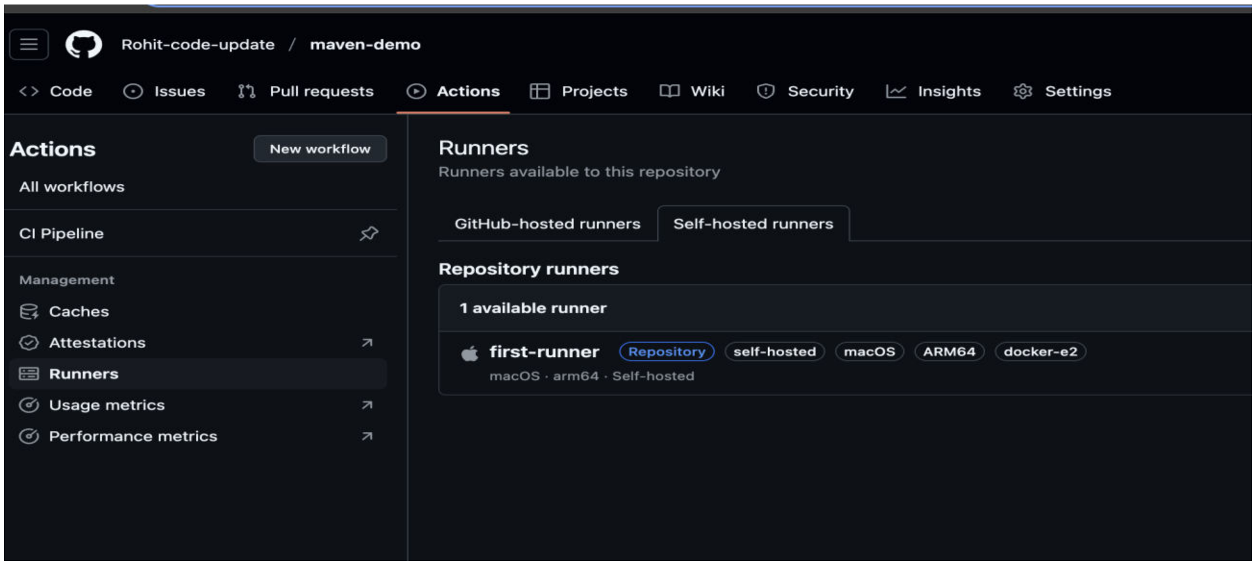
```yaml
name: CI Pipeline

on:
  push:
    branches:
      - master
  pull_request:
    branches:
      - master
jobs:
  build:
    runs-on: ubuntu-latest

    steps:
      - name: Check out code
        uses: actions/checkout@v2

      - name: Set up JDK 21
        uses: actions/setup-java@v4
        with:
          java-version: '21'
          distribution: temurin
          cache: maven

      - name: Build with Maven
        run: mvn clean package --file pom.xml

      - name: Run tests
        run: mvn test --file pom.xml

      - name: Save test reports as artifacts
        uses: actions/upload-artifact@v4
        with:
          name: test-reports
          path: target/surefire-reports/*.xml

      - name: Set up Docker
        uses: docker/setup-buildx-action@v2

      - name: Check Docker version
        run: docker --version

      - name: Log in to Docker Hub
        uses: docker/login-action@v2
        with:
          username: ${{ secrets.DOCKER_HUB_USERNAME }}
          password: ${{ secrets.DOCKER_HUB_PASSWORD }}

      - name: Build and Push Docker Image
        uses: docker/build-push-action@v5
        with:
```

Here are the screenshots for the self-hosted runners:



```
ctions-runner-osx-arm64-2.325.0.tar.gz: OK
notebook) 2657520@01HW2353195 actions-runner % tar xzf ./actions-runner-osx-arm64-2.325.0.tar.gz
notebook) 2657520@01HW2353195 actions-runner % ls
ctions-runner-osx-arm64-2.325.0.tar.gz env.sh                                 run-helper.sh.template
in                                     externals                              run.sh
onfig.sh                               run-helper.cmd.template                safe_sleep.sh
notebook) 2657520@01HW2353195 actions-runner % ./config.sh --url https://github.com/Rohit-code-update/maven-demo --

--------------------------------------------------------------------------------
|        ____ _ _   _   _       _          _        _   _                      |
|       / ___(_) |_| | | |_   _| |__      / \   ___| |_(_) ___  _ __  ___      |
|      | |  _| | __| |_| | | | | '_ \    / _ \ / __| __| |/ _ \| '_ \/ __|     |
|      | |_| | | |_|  _  | |_| | |_) |  / ___ \ (__| |_| | (_) | | | \__ \     |
|       \____|_|\__|_| |_|\__,_|_.__/  /_/   \_\___|\__|_|\___/|_| |_|___/     |
|                                                                              |
|                       Self-hosted runner registration                        |
|                                                                              |
--------------------------------------------------------------------------------

 Authentication


 Connected to GitHub

 Runner Registration

nter the name of the runner group to add this runner to: [press Enter for Default]

nter the name of runner: [press Enter for 01HW2353195] first-runner

his runner will have the following labels: 'self-hosted', 'macOS', 'ARM64'
nter any additional labels (ex. label-1,label-2): [press Enter to skip] docker-e2

 Runner successfully added
 Runner connection is good

 Runner settings

nter name of work folder: [press Enter for _work]

 Settings Saved.
```

Self-hosted Runner in GitHub:



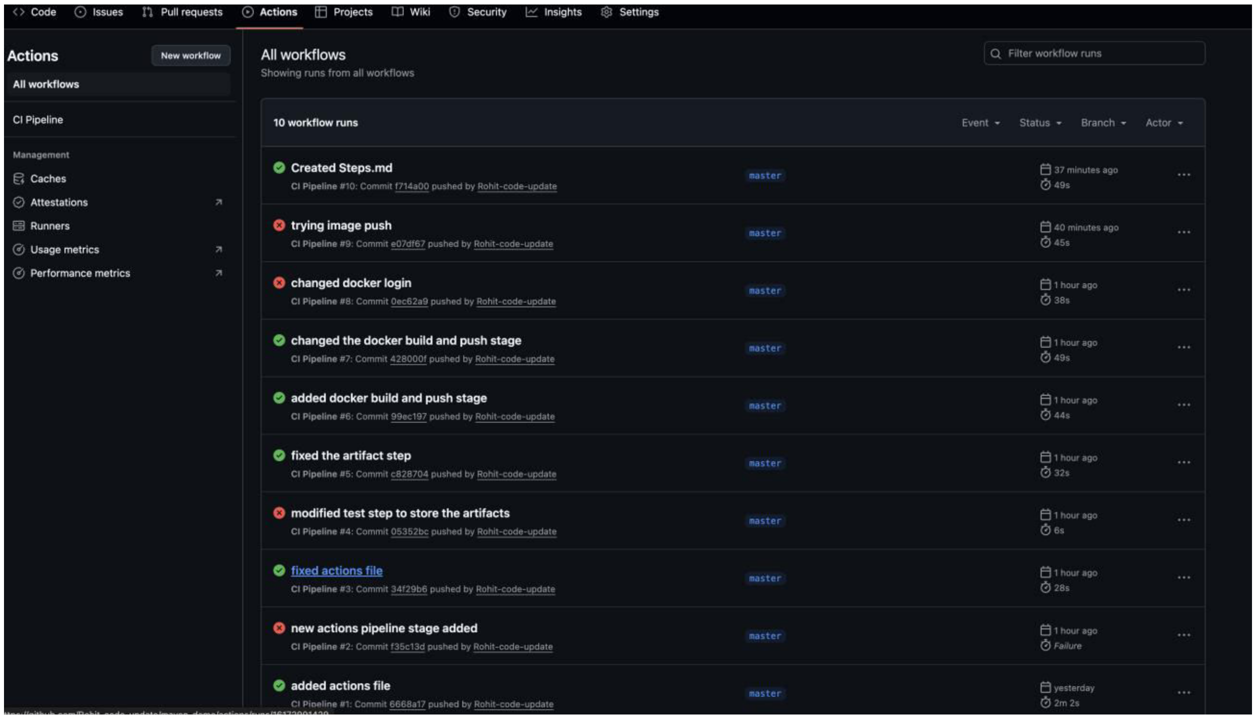
Runners
Runners available to this repository

Repository runners

1 available runner

first-runner Repository self-hosted macOS ARM64 docker-e2
macOS · arm64 · Self-hosted

The Logs of Runners:



All workflows
Showing runs from all workflows

10 workflow runs

✓ Created Steps.md CI Pipeline #10: Commit f714a00 pushed by Rohit-code-update	master	37 minutes ago 49s	
✗ trying image push CI Pipeline #9: Commit e07df67 pushed by Rohit-code-update	master	40 minutes ago 45s	
✗ changed docker login CI Pipeline #8: Commit 0ec62a9 pushed by Rohit-code-update	master	1 hour ago 38s	
✓ changed the docker build and push stage CI Pipeline #7: Commit 428000f pushed by Rohit-code-update	master	1 hour ago 49s	
✓ added docker build and push stage CI Pipeline #6: Commit 99ec197 pushed by Rohit-code-update	master	1 hour ago 44s	
✓ fixed the artifact step CI Pipeline #5: Commit c828704 pushed by Rohit-code-update	master	1 hour ago 32s	
✗ modified test step to store the artifacts CI Pipeline #4: Commit 05352bc pushed by Rohit-code-update	master	1 hour ago 6s	
✓ fixed actions file CI Pipeline #3: Commit 34f29b6 pushed by Rohit-code-update	master	1 hour ago 28s	
✗ new actions pipeline stage added CI Pipeline #2: Commit f35c13d pushed by Rohit-code-update	master	1 hour ago Failure	
✓ added actions file CI Pipeline #1: Commit 6668a17 pushed by Rohit-code-update	master	yesterday 2m 2s	

Thank You.