Cloud Computing Lab Project
CI-CD Pipeline using GitHub and DockerHub

Md. Abu Yousuf Siam, Nilay Biswas, Momtaj Hossain Mow, MD. Rashedul Alam

Department of Computer Science and Engineering, Jagannath University, Dhaka-1100, Bangladesh

Introduction

Continuous Integration and Continuous Deployment (CI/CD) is a software development approach that aims to improve the speed, efficiency, and reliability of software delivery. It is an automated process that involves frequent code integration, automated testing, and continuous deployment of software changes to production.

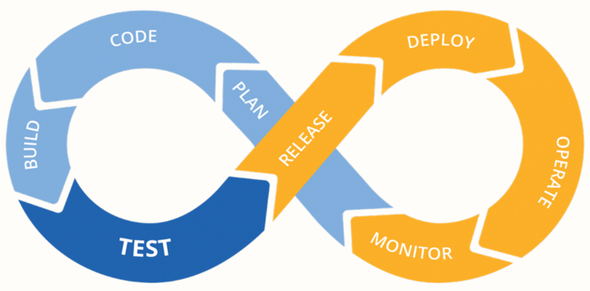

Figure 1: Continuous Integration and Continuous Deployment Cycle.

The Key Benefits of CI-CD

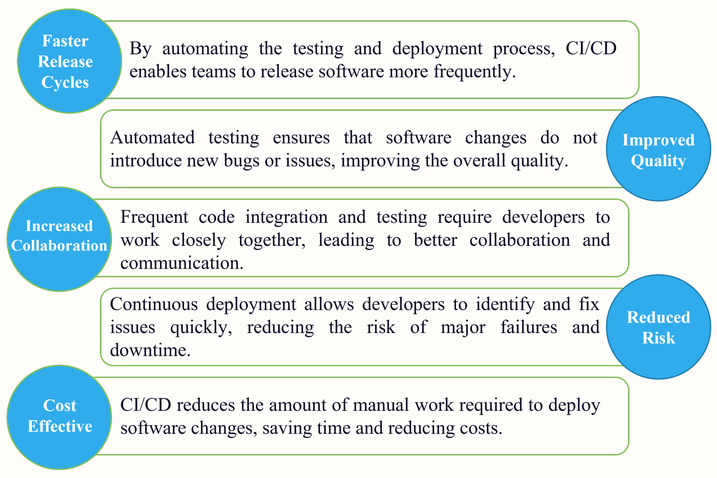

Figure 2: Benefits of CI-CD.

How CI-CD Works

Code Integration

In this step, developers commit their code changes to a remote repository (like GitHub, GitLab or BitBucket), where the code is integrated with the main codebase. This step aims to ensure that the code changes are compatible with the rest of the code-base and do not break the build.

Automated Testing

Automated testing involves running a suite of tests to ensure that the code changes are functional, meet the expected quality standards, and are free of defects.

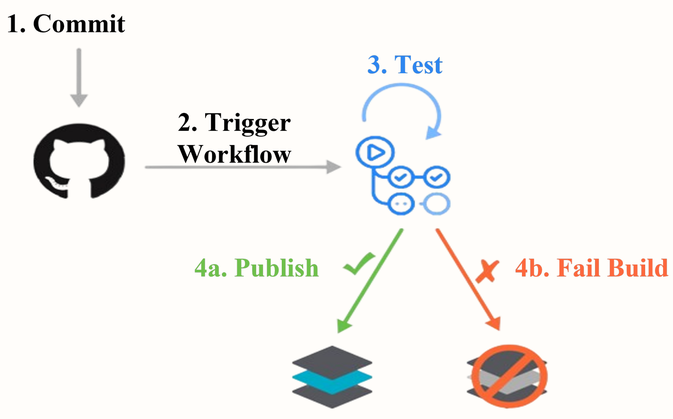

Figure 3: GitHub Actions.

Continuous Deployment

This step aims to ensure that the software is continuously updated with the latest code changes, delivering new features and functionality to users quickly and efficiently.

Production Deployment

In this step, the software changes are released to end-users. This step involves monitoring the production environment, ensuring that the software is running smoothly, and identifying and fixing any issues that arise.

The four steps of a CI/CD pipeline work together to ensure that software changes are tested, integrated, and deployed to production automatically. This automation helps to reduce errors, increase efficiency, and improve the overall quality of the software.

CI-CD Using GitHub and DockerHub

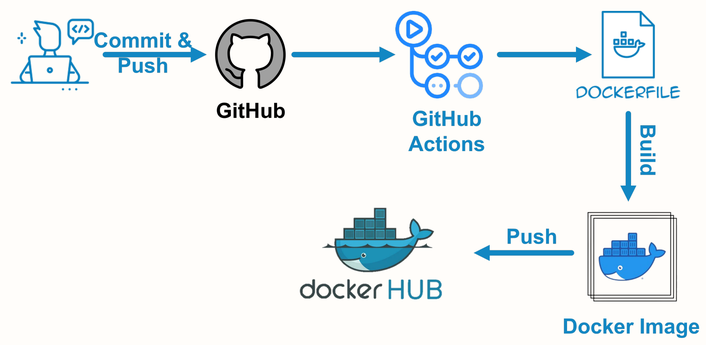

Figure 4: Workflow of CI-CD using GitHub Actions and DockerHub.

(1) GitHub Actions automate the build, test, and deployment pipeline
(2) DockerHub is used for building, annotating, and pushing images

Build and Push Docker Image to DockerHub using GitHub Actions and Docker File

```yaml
name: Build and Push "latest main" Image

on:
  push:
    branches:
      - main

jobs:
  build:

    runs-on: ubuntu-latest

    steps:
      -
        name: Checkout
        uses: actions/checkout@v3
      -
        name: Set up QEMU
        uses: docker/setup-qemu-action@v3
      -
        name: Set up Docker Buildx
        uses: docker/setup-buildx-action@v3
      -
        name: Login to Docker Hub
        uses: docker/login-action@v3
        with:
          username: ${{ secrets.DOCKERHUB_USERNAME }}
          password: ${{ secrets.DOCKERHUB_TOKEN }}
      -
        name: Build and push Docker images
        uses: docker/build-push-action@v5.1.0
        with:
          context: .
          file: Dockerfile
          push: true
          tags: ${{ secrets.DOCKERHUB_USERNAME }}/rms:main
```

Figure 5: automation.yml

```dockerfile
# IMAGE
# Dockerfile for www service
FROM php:8.2-apache

# Install mysqli extension
RUN docker-php-ext-install mysqli

# We need to create inside the container's system our workdir path
RUN mkdir -p /var/www/html

# Set the working directory to /var/www/html
WORKDIR /var/www/html

# Copy the current directory contents into the container at /var/www/html
COPY src/ /var/www/html

# Expose port 80 for Apache
EXPOSE 80

# Start Apache when the container runs
CMD ["apache2-foreground"]
```

Figure 6: Dockerfile

Pull Image from DockerHub Using Docker Compose File

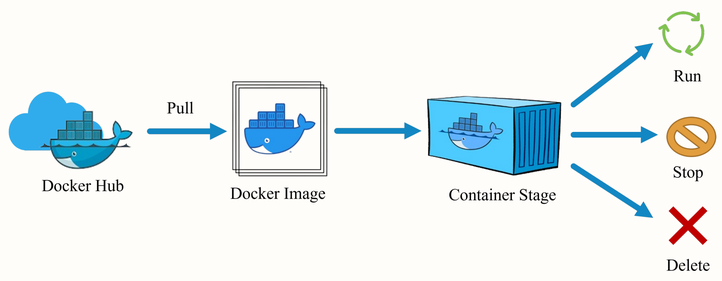

Figure 7: Pull Image from Docker-Hub.

```yaml
version: '3'

services:
  web:
    image: momtajhossainmow/rms:main
    ports:
      - "3000:80"
    links:
      - db
    networks:
      - default

  db:
       image: mysql
       ports:
           - "3306:3306"
       command: --default-authentication-plugin=mysql_native_password
       environment:
           MYSQL_DATABASE: rms
           MYSQL_ROOT_PASSWORD: test
       volumes:
      # import sql database
           - ./sql/rms.sql:/docker-entrypoint-initdb.d/rms.sql
       networks:
           - default

  phpmyadmin:
     image: phpmyadmin/phpmyadmin
     links:
        - db:db
     ports:
        - "8000:80"
     environment:
        MYSQL_USER: root
        MYSQL_PASSWORD: test
        MYSQL_ROOT_PASSWORD: test

volumes:
    persistent:
```

Figure 8: docker-compose.yml

Summary and Conclusions

GitHub Actions and DockerHub integration streamline the development process. Automating the builds and deployments pipelines creates team efficiency while reducing errors and increasing productivity.
This project has shown how to create an automated CI/CD build with GitHub and DockerHub.

References

[1] Build CI/CD Pipeline using GitHub Action's | Build and push Docker Image, https://youtu.be/euEkYEFCrI8?feature=shared

[2] https://www.freecodecamp.org/news/what-is-ci-cd/

[3] Create Automated CI/CD Builds Using GitHub Actions and DockerHub, https://earthly.dev/blog/cicd-build-github-action-dockerhub/