Implementasi *Continuous Integration* Dan *Continuous Deployment* Pada Pengembangan Aplikasi *Website* Menggunakan *Docker* Dan *Github Actions*

Zaidan Zulhakim[1], Ari Kurniawan[2]

Manajemen Informatika, Fakultas Vokasi, Universitas Negeri Surabaya

Surabaya, Indonesia

[1]zaidan.19075@mhs.unesa.ac.id

[2]arikurniawan@unesa.ac.id

Abstrak*— *Perkembangan teknologi informasi saat ini semakin tinggi, salah satunya adalah pengembangan perangkat lunak (Software Development). Saat proses penyebaran perangkat lunak (Deployment), pengembang selalu mengikuti proses konvensional. Pengembang akan terus melakukan proses yang sama, seperti menggunakan aplikasi file sharing untuk mengirimkan kode program ke masing-masing server. Proses itu akan memakan waktu lama untuk mendistribusikan kode program ke setiap server dan juga harus beradaptasi dengan aplikasi yang akan berkembang dari waktu ke waktu. Untuk mengatasi masalah tersebut, tugas akhir ini menerapkan teknologi Docker dan GitHub Actions sebagai solusi untuk CI/CD. Docker memungkinkan pembuatan, pengujian, dan pengiriman aplikasi dalam lingkungan kontainer yang terisolasi, memastikan konsistensi di berbagai tahap pengembangan. Sementara itu, GitHub Actions, sebagai platform CI/CD yang otomatis dan terintegrasi dengan repositori Git di GitHub, memberikan fleksibilitas dan kemudahan dalam mengatur alur kerja pengujian dan implementasi aplikasi secara berulang dengan cepat dan efisien. Juga membandingkan implementasi CI/CD antara Github Actions dan Jenkins. Dari hasil kajian yang telah dilakukan, terlihat bahwa penggunaan GitHub Actions dan Docker lebih cepat daripada Jenkins pada proses deployment aplikasi Website. Ini didukung oleh Quality Metrics yang diuji yaitu, waktu rata rata yang dihabiskan Github Actions dalam 5 kali deployment adalah 131.2 detik atau 2 menit 11 detik. Dan untuk rata rata waktu yang dihabiskan Jenkins dalam 5 kali deployment adalah 362.6 detik atau 6 menit 2 detik. Hal ini menunjukan bahwa waktu rata rata untuk 5 kali proses CI/CD cenderung lebih cepat Github Actions daripada Jenkins.

***Abstract*— The current advancement in information technology is rapidly growing, including software development. During the software deployment process, developers often follow conventional methods. They repeatedly go through the same procedures, such as utilizing file sharing applications to send program code to individual servers. This conventional process consumes a significant amount of time to distribute the program code to each server and also requires adapting to evolving applications over time. To address these challenges, this final project implements Docker and GitHub Actions as solutions for CI/CD. Docker enables the creation, testing, and delivery of applications within isolated container environments, ensuring consistency across various development stages. Meanwhile, GitHub Actions, as an automated CI/CD platform integrated with Git repositories on GitHub, offers flexibility and ease in managing testing and application deployment workflows repeatedly, swiftly, and efficiently. The project also compares the CI/CD implementation between GitHub Actions and Jenkins. Based on the conducted study, it is evident that the utilization of GitHub Actions and Docker is faster than Jenkins in the deployment process of the website application. This is supported by the tested Quality Metrics, where the average time spent by GitHub Actions in 5 deployment instances is 131.2 seconds or 2 minutes 11 seconds, while the average time spent by Jenkins in 5 deployment instances is 362.6 seconds or 6 minutes 2 seconds. These results demonstrate that the average time for 5 CI/CD processes tends to be faster with GitHub Actions than with Jenkins.**

Kata kunci*— *Continuous Integration (CI), Continuous Deployment (CD), Docker, Deployment, Github Actions.

I. PENDAHULUAN

Semakin pesatnya era teknologi informasi, pengembangan perangkat lunak menjadi salah satu bidang yang mengalami perkembangan pesat. Proses pengembangan perangkat lunak melibatkan tahapan-tahapan penting dalam menciptakan perangkat lunak yang berkualitas. Proses pengembangan perangkat lunak terdapat tahapan *Build* dan *Test* yang dilakukan secara terpisah oleh tim pengembang dan penguji. Proses ini memakan banyak waktu dalam pengembangan perangkat lunak menggunakan pendekatan konvensional.

Implementasi *CI/CD* pada pengembangan aplikasi website dengan menggunakan *Docker* dan *GitHub Actions* memberikan efisiensi yang lebih tinggi. *Website* yang telah melalui proses *Development*, *Test*, dan *Deploy* dapat diambil dengan mudah melalui repositori *GitHub* tanpa perlu melakukan instalasi alat-alat tambahan untuk proses *CI/CD*. Hal ini secara signifikan mempercepat proses pengembangan produk tanpa mengurangi kualitas produk

itu sendiri. Penelitian ini diharapkan dapat memberikan wawasan mendalam mengenai bagaimana penggunaan teknologi Docker dan platform GitHub Actions secara sinergis dapat mempercepat proses pengembangan, meningkatkan kualitas perangkat lunak, dan merangsang kolaborasi tim. Harapannya, penelitian ini akan memberikan panduan praktis bagi para pengembang dan organisasi dalam menerapkan CI/CD dengan efektif, sambil merespons tantangan-tantangan dan memanfaatkan peluang yang dihadapi dalam dunia perkembangan aplikasi web yang terus berubah dan dinamis.

II. LANDASAN TEORI

A. Penelitian Terdahulu

1) *Alpery A dan Ridha M. A. F (2021):* Berjudul "Implementasi CI/CD Dalam Pengembangan Aplikasi Web Menggunakan Docker Dan Jenkins". Penelitian ini bertujuan untuk menganalisa kualitas, waktu yang dibutuhkan, dan proses otomatisasi pada proses pengembangan aplikasi website mulai dari build, test, dan *deploy* dengan metode *CI/CD* menggunakan *Docker* dan *Jenkins*. Hasil dari penelitian ini adalah rata-rata waktu yang dibutuhkan *jenkins* dalam melakukan proses *deployment jenkins pipeline* adalah selama 1 menit 58 detik, tingkat keberhasilan *jenkins* dalam melakukan proses *deployment* adalah sebesar 90%, dan *jenkins* mendeteksi adanya *bugs* dengan total 78 *bugs* dalam proses *deployment* pada proses *CI/CD* dengan menggunakan *docker* dan *jenkins*.

2) *Jaeni, dkk (2022):* Berjudul "Implementasi Continuous Integration / Continuous Delivery (CI/CD) Pada Performance Testing Devops". Penelitian ini bertujuan untuk meningkatkan efisiensi dalam proses pengembangan dan produktivitas dari pengembang. Dengan adanya penerapan *CI/CD* fase *test* dan *release* diterapkan dengan otomatisasi, sehingga pengembang diberikan kemudahan dan diharapkan kinerjanya akan meningkat. Hasil dari penelitian ini adalah bahwa proses *deployment* menggunakan *CI/CD* mampu mempersingkat proses dan meningkatkan kinerja. Selain itu *CI/CD* memberikan hasil yang lebih teliti dengan penemuan *bug* pada pengujian tersebut.

3) *Tohirin, dkk (2020):* Berjudul "Implementasi DevOps pada Pengembangan Aplikasi e-skrining Covid-19". Penelitian ini bertujuan untuk memberikan solusi otomatisasi *build*, *test*, dan *deploy* bagi mereka yang berkerja di lingkungan *SLDC Agile Scrum*. Hasil dari penelitian ini adalah penggabungan kode menjadi semakin mudah, *build* harian lancar dan pemeriksaan kesehatan dan kelayakan kode terjadi setiap kali ada *commit* dan *push* dari pengembang.

B. Dasar Teori

1) Continuous Integeration (CI): Fase ini merupakan tahap inti pada siklus *Development* karena adanya otomatisasi integerasi pengembangan perangkat lunak dan proses operasional. Hal ini menjadi solusi tepat untuk menciptakan sebuah produk yang belum pernah dijumpai, yakni membuat perangkat lunak dan membangun aplikasi dengan cara praktis berbasis web atau internet (Duvall, Matyas, & Glover, 2007). Karena jika menggunakan proses *Development* dengan cara konvensional maka proses akan membutuhkan waktu lebih lama dibandingkan cara otomatisasi Development yang akan penulis gunakan. Setiap kode yang di komit ke sebuah repositori, kode harus dideteksi sedini mungkin, terlebih jika ditemukan bug. Proses integerasi kode yaitu; Build, Test dan Release.

2) Continuous Deployment (CD): Pada fase ini kode disebarkan atau *Release* ke *server* produksi atau juga bisa ke lokal. Penting juga untuk memastikan bahwa kode tersebut digunakan dengan benar di semua *server*. Fase ini dilakukan setelah melewati fase *Continuous Integeration (CI)*. *Continuous Deployment (CD)* adalah pendekatan rekayasa perangkat lunak dimana tim menghasilkan perangkat lunak dalam siklus waktu yang pendek, memastikan bahwa perangkat lunak dapat dirilis dengan baik secara fleksibel (Wiedemann, 2020). Ini memungkinkan tim untuk membuat kode dengan cepat ke lingkungan produksi, sambil terus memastikan bahwa kode berfungsi seperti yang diharapkan.

3) Docker: Pengembang dan administrator sistem dapat mengemas, mengirimkan, dan mengoperasikan program yang memanfaatkan teknologi kontainerisasi dengan bantuan platform sumber terbuka yang dikenal sebagai *Docker*. *Docker* menggunakan kontainer untuk memisahkan program dan semua dependensinya dalam lingkungan yang konsisten, portabel, dan terisolasi. Tujuan mendasar dari *Docker* adalah untuk meningkatkan efektivitas dan keandalan dalam mengelola lingkungan aplikasi sekaligus membuat pengembangan, distribusi, dan operasionalisasi aplikasi menjadi lebih sederhana (Radu & Matei, 2019). Dengan menggabungkan aplikasi dan dependensinya ke dalam kontainer portabel, *Docker* dapat menyederhanakan proses pengembangan aplikasi. Pengembang dapat menghindari masalah yang sering terjadi saat aplikasi diluncurkan dalam berbagai konteks. Dengan demikan, *Docker* menjadi pilihan yang menarik bagi pengembang dan administrator sistem yang ingin meningkatkan efisiensi dan skalabilitas dalam pengelolaan lingkungan aplikasi.

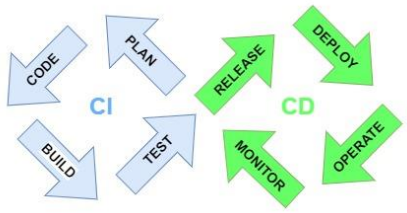
4) *Github Actions:* Sebuah platform untuk otomatisasi yang disebut *GitHub Actions* ditawarkan oleh *GitHub* untuk membantu dan mengotomatiskan proses pengembangan perangkat lunak. Pengembang dapat menggunakan *GitHub Actions* untuk merancang alur kerja yang menentukan urutan aktivitas yang harus dilakukan sebagai respons terhadap peristiwa tertentu dalam repositori. Platform ini membantu pemrogram untuk mempercepat dan meningkatkan efisiensi pengembangan perangkat lunak dengan mengotomatiskan operasi berulang seperti *Build*, *Test*, dan *Release* (Peters, 2021). Pengujian unit, integrasi berkelanjutan, dan penyebaran ke lingkungan produksi hanyalah beberapa dari proses siklus pengembangan penting yang dapat diotomatisasi

menggunakan *GitHub Actions* (Sannino & Bruneo, 2022). Menggunakan *GitHub Actions*, ditemukan dalam penelitian ini, sangat meningkatkan produktivitas tim dan membebaskan pengembang untuk berkonsentrasi pada pembuatan fitur daripada melakukan tugas administratif. Selain itu, dengan mengatur dan mengotomatiskan operasi secara terpusat dengan *GitHub Actions*, tim pengembangan dapat merilis pembaruan perangkat lunak secara teratur dan konsisten juga dengan kerumitan yang lebih sedikit.

III. METODOLOGI PENELITIAN

A. *Rancangan Penelitian*

Metodologi penelitian ini menggunakan proses Continuos Integeration (CI) dan Continuos Delivery (CD) sebagaimana dapat dilihat pada gambar 3.1 Alur proses CI/CD.



Gambar 3.1 Alur proses CI/CD

Perancangan penelitian terdiri dari proses proses seperti set up Go, Build, Test, Push. Alur kerja CI/CD menggunakan Github Actions dan Docker. Dan Aplikasi Website yang digunakan pada penelitian ini dikembangkan dengan menggunakan bahasa pemrograman Go.

B. *Implementasi Sistem*

Pada tahap proses implementasi sistem ini akan dilakukan otomatisasi pada proses *CI/CD* dalam pengembangan aplikasi *website*. Otomatisasi ini menggunakan *Github Actions* dan *Docker* berdasarkan perancangan penelitian. Pipeline men-*deploy* aplikasi *website* menggunakan *Docker images*, di mana kode akan di *push* ke repositori dan memicu *instance Cloud Run* yang menjalankan *Actions* untuk menginstal dependensi secara otomatis, menjalankan *Build and Push*, dan memublikasikan aplikasi sebagai *Artefact* yang dapat diterapkan.

Peralatan yang akan digunakan dalam penelitian ini ada dua komponen yaitu:

1. Hardware
   - Laptop *Acer Aspire 5*
   - *Intel Core i5-8265U*
   - *NVIDIA GeForce MX230 2GB*
   - *8GB RAM*
   - *512GB SSD*
2. Software
   - *Visual Studio Code*
   - *Docker Desktop*
   - *Google Chrome*
   - *Postman*

C. *Pengujian Sistem*

Skrip kode aplikasi *website* yang sudah di *push* ke *Repository Github* akan dilakukan pengujian secara otomatis dengan menjalankan proses *Build and Push* dan *Deployment* di *Github Actions* dan akan diuji berdasarkan waktu yang dibutuhkan (*Time Based Metrics*) dan kualitas (*Quality Metrics*).

IV. HASIL PENELITIAN DAN PEMBAHASAN

A. *Implementasi Penelitian*

Pada tahap ini, penulis melakukan implementasi terhadap sistem yang telah dirancang menggunakan metode Continuous Integration dan Continuous Deployment (CI/CD). Implementasi sistem ini melibatkan penggunaan dua metode CI/CD yang berbeda, yaitu GitHub Actions dan Jenkins. Berikut adalah langkah-langkah implementasi yang penulis lakukan:.

I. Implementasi menggunakan GitHub Actions

a. Konfigurasi Repositori Github Actions

```yaml
name: Go

on:
  push:
    branches: [ "main" ]
  pull_request:
    branches: [ "main" ]

jobs:

  build:
    runs-on: ubuntu-latest

    steps:
    - name: Checkout Repository
      uses: actions/checkout@v2

    - name: Set up Go
      uses: actions/setup-go@v3
      with:
        go-version: 1.18.3

    - name: Build
      run: go build -v .
```

```
    - name: Test
      run: go test -v .

  deploy:
    needs: build
    runs-on: ubuntu-latest
    steps:
      - name: Checkout repository
        uses: actions/checkout@v2

      - name: Login to Docker Hub
        uses: docker/login-action@v2
        with:
          username: ${{
secrets.DOCKERHUB_USERNAME }}
          password: ${{
secrets.DOCKERHUB_TOKEN }}

      - name: Build and push Docker Image
        uses: docker/build-push-action@v2
        with:
          context: .
          push: true
          tags: zaidanzulhakim/test:latest

      - name: Deploy to local environment
        run: |
          docker-compose up –d
```

Skrip ini mendefinisikan aliran kerja (*workflow*) dengan nama "Go". Aliran kerja ini akan diaktifkan ketika terjadi *Event* "push" atau "pull request" pada cabang "main" di repositori. Tindakan yang akan dilakukan pada aliran kerja ini adalah "build" dan "test" aplikasi Go.

Setelah proses "build" dan "test" selesai dan berhasil, langkah "deploy" akan dijalankan. Langkah ini membutuhkan hasil dari langkah "build" karena ditentukan oleh "needs: build". Selanjutnya, langkah-langkah dalam "deploy" adalah:

1. *Checkout repository*: Langkah ini mengambil kode dari repositori untuk digunakan dalam proses selanjutnya.
2. *Login to Docker Hub*: Tindakan ini akan melakukan login ke *Docker Hub* menggunakan kredensial yang tersimpan sebagai rahasia (secrets) di repositori.
3. *Build* and *push Docker Image*: Proses ini akan membangun *Docker Image* dari konteks saat ini (current directory) dan mengirimkannya ke *Docker Hub* dengan tag "zaidanzulhakim/test:latest".
4. *Deploy to local environment*: Langkah ini akan menjalankan aplikasi dalam lingkungan lokal menggunakan *Docker Compose* dengan perintah "docker-compose up -d".

b. Definisi Alur Kerja

```
PS test> git add .

PS test> git commit -m
"Changes"
[main aadf4af] Changes
 1 file changed, 1 insertion(+), 1
deletion(-)

PS test> git push origin main
Enumerating objects: 9, done.
Counting objects: 100% (9/9),
done.
Delta compression using up to 8
threads
Compressing objects: 100% (3/3),
done.
Writing objects: 100% (5/5), 377
bytes | 377.00 KiB/s, done.
Total 5 (delta 2), reused 0 (delta 0),
pack-reused 0
remote: Resolving deltas: 100%
(2/2), completed with 2 local
objects.
To
https://github.com/Zaidannzzz/test.
git
   2802625..aadf4af  main -> main
```

Pada skrip di atas, tindakan yang diambil adalah melakukan beberapa perintah *Git* untuk melakukan proses *commit* dan *push* perubahan pada repositori *GitHub*. Berikut adalah penjelasan formal dari setiap perintah:

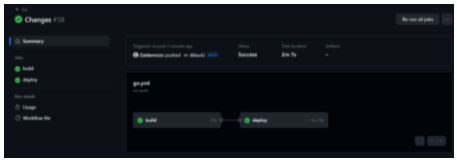
1. ***git add*** .: Perintah ini mengindeks semua perubahan yang ada dalam direktori kerja (*working directory*) ke dalam area *stage*. Dengan menggunakan tanda titik (.) setelah perintah "git add", itu berarti semua perubahan pada direktori saat ini akan diindeks.

2. ***git commit -m "Changes"***: Setelah melakukan "git add", perintah "git commit" digunakan untuk mencatat perubahan yang telah diindeks ke dalam repositori lokal dengan pesan komit yang diberikan. Dalam contoh ini, pesan komitnya adalah "Changes".

3. ***git push origin main***: Perintah "git push" digunakan untuk mengirimkan semua komit yang ada di repositori lokal ke repositori jarak jauh (remote repository), dalam hal ini ke repositori "origin" (GitHub). Kata "main" menunjukkan bahwa kita ingin mengirimkan komit dari cabang "main" lokal ke cabang "main" pada repositori jarak jauh.

   Setelah proses "git push" selesai, perubahan yang telah di-*push* akan diterapkan pada repositori *GitHub* dan dapat diakses oleh orang lain yang berkolaborasi pada proyek tersebut. Perubahan ini juga akan terlihat pada halaman repositori di *GitHub*. Setelah itu push yang sudah dilakukan akan mejadi pemicu untuk *Github Actions* yang dan akan menjalankan skrip *workflow.yml* yang sudah dibuat sebelumya.

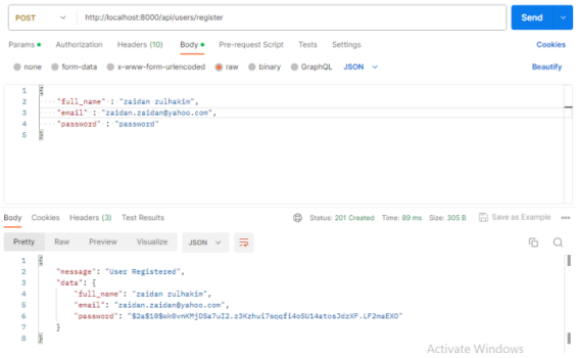
c. Uji Coba Alur Kerja



Gambar 4.1 Uji Coba Alur Kerja

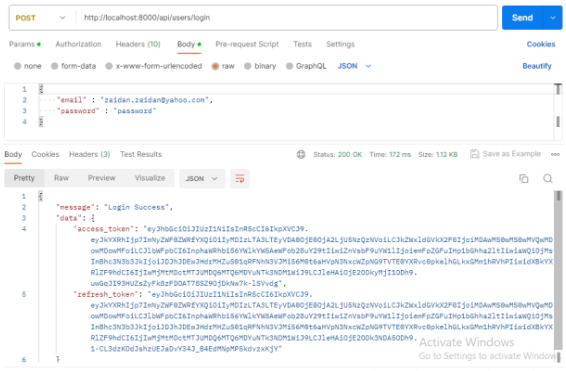
Proses CI/CD pada Github Actions telah berhasil ditandai dengan tanda ceklis hijau dimana skrip CI/CD pada file go.yml yang berisikan proses yang bernama “build” dan “deploy” telah berhasil dijalankan.

d. Pengetesan Aplikasi



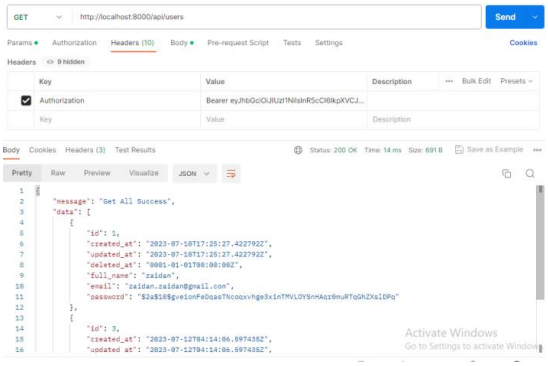
Gambar 4.2 Endpoint /users/register

Permintaan untuk membuat akun baru berhasil dilakukan. Ketika pengguna mengirimkan data melalui permintaan POST ke endpoint /users/register, server berhasil menerima data tersebut. Data yang dikirimkan melalui permintaan berhasil diproses oleh server, dan akun baru berhasil dibuat dalam sistem. Hasil permintaan ini, server mengirimkan balasan dengan status 201 Created, yang menandakan bahwa proses pendaftaran berhasil dan akun baru telah berhasil dibuat.



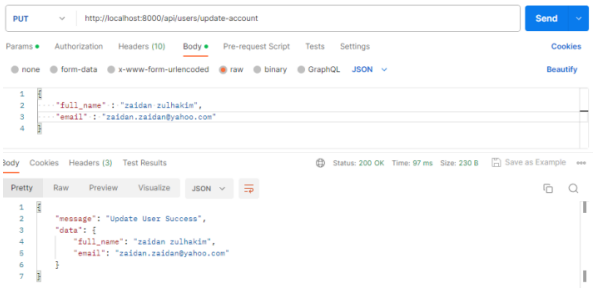
Gambar 4.3 Endpoint /users/login

Permintaan untuk masuk ke akun berhasil dieksekusi. Ketika pengguna mengirimkan data login melalui permintaan POST ke endpoint /users/login, server berhasil memeriksa keabsahan data login tersebut. Setelah data login diverifikasi, server berhasil menghasilkan token otentikasi (JWT) dan mengirimkannya sebagai respon ke pengguna. Status 200 OK ditampilkan dalam respon, menandakan bahwa proses login berhasil dan pengguna berhasil masuk ke akun mereka.



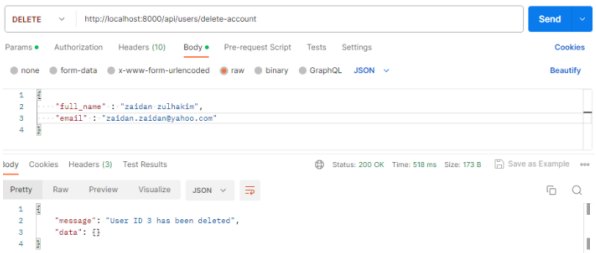
Gambar 4.4 Endpoint /users

Permintaan untuk mendapatkan daftar pengguna berhasil dilakukan. Sebelum mengizinkan akses, sistem melakukan verifikasi menggunakan middleware JwtGuard untuk memastikan token otentikasi (JWT) yang sah. Setelah otorisasi berhasil, server berhasil mengambil daftar pengguna dari database dan mengirimkan daftar tersebut sebagai respon. Respon berisi status 200 OK dan daftar pengguna yang diminta.



Gambar 4.5 Endpoint /users/update-account

Permintaan untuk memperbarui informasi akun berhasil dilakukan. Ketika pengguna mengirimkan data perubahan melalui permintaan PUT ke endpoint /update-account, server berhasil memproses data tersebut. Middleware JwtGuard juga berhasil melakukan verifikasi token otentikasi (JWT) yang sah sebelum mengizinkan akses untuk memperbarui akun. Respon yang dikirimkan oleh server berisi status 200 OK, menandakan bahwa permintaan untuk memperbarui akun telah berhasil.



Gambar 4.6 Endpoint /users/delete-account

Permintaan untuk menghapus akun berhasil dilakukan. Middleware JwtGuard memverifikasi token otentikasi (JWT) yang sah sebelum mengizinkan akses untuk menghapus akun. Setelah otorisasi berhasil, server berhasil menghapus akun pengguna dari sistem. Sebagai respon atas permintaan ini, server mengirimkan status 200 OK untuk menandakan bahwa akun pengguna telah berhasil dihapus.

II. Implementasi menggunakan Jenkins

a. Konfigurasi Jenkins

Gambar 4.7 Konfigurasi Credentials Jenkins

Di Jenkins, konfigurasi credentials menjadi hal yang sangat penting karena memungkinkan untuk menyimpan informasi rahasia secara aman, seperti token otentikasi, kata sandi, atau kunci API. Melalui fitur ini, dapat mengelola credentials di pusat Jenkins dan menggunakannya dalam berbagai pekerjaan Jenkins tanpa harus mengekspos informasi rahasia secara langsung dalam konfigurasi pekerjaan.

Setelah credentials berhasil ditambahkan, credentials tersebut dapat digunakan dalam konfigurasi pekerjaan Jenkins. Terlihat pada gambar diatas sudah di buatnya credentials dengan ID "github-zaidannzzz" dan "docker-zaidannzzz", credentials tersebut dapat digunakan sebagai bagian dari skrip build atau untuk mengakses repositori kode yang memerlukan otorisasi. Penggunaan credentials ini memberikan tingkat keamanan yang lebih tinggi, karena informasi rahasia tidak akan ditampilkan dalam konfigurasi pekerjaan secara terbuka, tetapi akan diakses melalui variabel lingkungan yang telah ditentukan sebelumnya

b. Definisi Job CI/CD

```
pipeline {
    agent any

    stages {
        stage('Checkout') {
            steps {
```

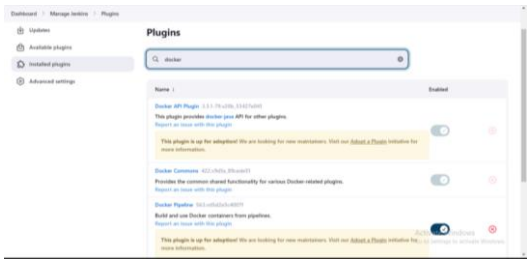
```
            // Clone repository
            git branch: 'main', credentialsId: 'github-zaidannzzz', url: 'https://github.com/Zaidannzzz/test_jenkins.git'
        }
    }

    stage('Build and Push') {
        steps {
            // Build Docker image
            script {
                if (isUnix()) {
                    sh 'docker build -t zaidanzulhakim/test_jenkins:latest .'
                } else {
                    bat 'docker build -t zaidanzulhakim/test_jenkins:latest .'
                }
            }

            // Docker login and push
            withCredentials([usernamePassword(credentialsId: 'docker-zaidannzzz', passwordVariable: 'dockerhubPassword', usernameVariable: 'dockerhubUser')]) {
                bat "docker login -u ${dockerhubUser} -p ${dockerhubPassword}"
                bat 'docker push zaidanzulhakim/test_jenkins:latest'
            }
        }
    }

    stage('Deploy') {
        steps {
            // Pull Docker image from registry
            script {
                docker.image('zaidanzulhakim/test_jenkins:latest').pull()
            }

            // Deploy using docker-compose
            bat 'docker-compose -f docker-compose.yml up -d'
        }
    }
}
```

Skrip diatas adalah bagian dari Jenkins Pipeline, yang merupakan alur kerja otomatis untuk mengotomatisasi proses build, deploy, dan pengujian aplikasi. Pada skrip ini, Jenkins akan melakukan beberapa tahap berikut:

- Checkout: Pada tahap ini, Jenkins akan melakukan proses "checkout" atau "clone" dari repositori yang ada di GitHub. Dalam langkah ini, Jenkins akan mengambil kode dari cabang "main" di repositori GitHub dengan menggunakan credentials yang telah ditentukan sebelumnya.
- Build and Push: Setelah melakukan "checkout", Jenkins akan memulai tahap "Build and Push". Di sini, Jenkins akan membangun sebuah Docker image dari kode yang telah di-"checkout". Docker image ini akan diberi tag "zaidanzulhakim/test_jenkins:latest". Selanjutnya, Jenkins akan melakukan login ke Docker Hub menggunakan credentials yang telah disimpan sebelumnya. Setelah login berhasil, Jenkins akan mendorong (push) Docker image yang telah dibuat ke Docker Hub sehingga tersedia untuk diakses oleh lingkungan produksi atau lingkungan pengujian lainnya.
- Deploy: Setelah berhasil membangun dan mendorong Docker image, Jenkins akan

melanjutkan ke tahap "Deploy". Di tahap ini, Jenkins akan mengambil Docker image terbaru dari Docker Hub dan menyimpannya dalam lingkungan Jenkins. Selanjutnya, Jenkins akan menjalankan proses deploy menggunakan docker-compose untuk memulai kontainer berdasarkan Docker image yang telah disimpan sebelumnya. Kontainer ini akan berjalan dalam mode detached (-d) untuk memastikan aplikasi tetap berjalan di latar belakang.

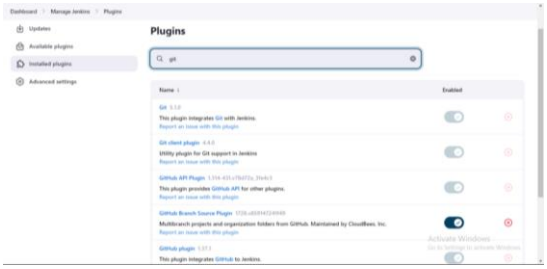
c. Konfigurasi Plugin dan Alat Pendukung



Gambar 4.8 Konfigurasi Plugin Docker

Plugin Docker di Jenkins adalah tambahan yang sangat berguna untuk mempermudah proses build, deploy, dan pengujian aplikasi dengan menggunakan Docker. Dengan mengkonfigurasi plugin Docker di Jenkins, tim pengembang dapat dengan mudah mengintegrasikan Docker ke dalam alur kerja Jenkins.

Setelah plugin Docker terpasang, skrip yang sudah di buat sebelumnya dapat dengan mudah menggunakan berbagai langkah Docker. Seperti menggunakan perintah "docker build" untuk membangun Docker image, "docker push" untuk mendorong image ke registri, dan "docker run" untuk menjalankan kontainer, juga "docker.withRegistry" untuk menghubungkan antara Jenkins dan registri Docker dengan mengakses credentials "docker-zaidannzzz" untuk login ke registri.



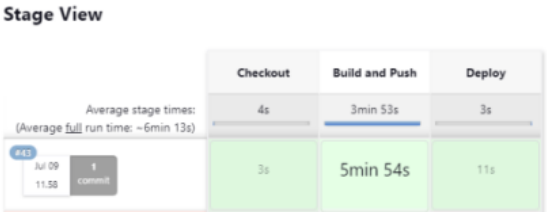
Gambar 4.9 Konfigurasi Plugin Git

Plugin Git di Jenkins merupakan alat yang sangat berguna untuk mengintegrasikan Git ke dalam alur kerja Jenkins. Dengan melakukan konfigurasi plugin Git, Anda dapat dengan mudah menghubungkan Jenkins dengan repositori Git untuk memulai proses build, deploy, dan pengujian aplikasi secara otomatis.

Setelah konfigurasi plugin Git selesai, Jenkins akan dapat secara otomatis melakukan "checkout" atau "kloning" dari repositori Git yang telah ditentukan. Hal ini memungkinkan Jenkins untuk secara teratur memonitor perubahan pada repositori dan memulai alur kerja (pipeline) sesuai dengan perubahan yang terjadi.

Konfigurasi plugin Git di Jenkins memberikan kemudahan dan kecepatan dalam bekerja dengan repositori Git, sehingga tim pengembang dapat lebih fokus pada pengembangan aplikasi tanpa harus melakukan tugas-tugas manual yang berulang. Dengan integrasi Git yang baik, Jenkins dapat secara efisien membangun dan menyampaikan aplikasi, serta memastikan bahwa kode sumber selalu terbarui dan siap untuk diuji dan diterapkan. Semua ini membantu meningkatkan produktivitas dan kualitas pengembangan perangkat lunak secara keseluruhan.

d. Uji Coba Job CI/CD



Gambar 4.10 Uji Coba Job CI/CD

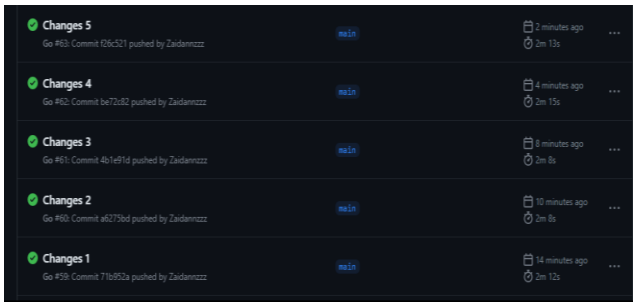
Hasil build pipeline Jenkins yang berhasil menunjukkan bahwa alur kerja (pipeline) yang telah diatur berjalan dengan sukses dan aplikasi telah berhasil dibangun tanpa ada masalah. Proses ini berlangsung secara otomatis. Proses build pipeline Jenkins yang berhasil menjamin bahwa kode yang diterapkan ke dalam aplikasi telah melalui serangkaian pengujian dan validasi sebelum di deploy ke lingkungan lokal. Oleh karena itu, proses CI/CD dapat membantu meminimalkan risiko terjadinya masalah atau bug yang dapat mempengaruhi kinerja aplikasi secara keseluruhan.

B. Pembahasan

Setelah mengimplementasikan sistem menggunakan metode CI/CD dengan GitHub Actions dan Jenkins, penulis akan membandingkan kedua metode tersebut dalam hal waktu yang dibutuhkan (Time Based Metrics) dan kualitas (Quality Metrics).

I. Time Based Metrics (Waktu yang dibutuhkan)

a. Hasil Pengujian CI/CD menggunakan Github Actions dan Docker



Gambar 4.11 Hasil Pengujian CI/CD menggunakan Github Actions dan Docker

b. Hasil Pengujian CI/CD menggunakan Jenkins dan Docker

Gambar 4.12 Hasil Pengujian CI/CD menggunakan Jenkins dan Docker

Jika kedua hasil tersebut di bandingkan dengan tabel maka akan terlihat seperti ini :

Tabel 4.1 Hasil Pengujian CI/CD Menggunakan GitHub Actions dan Jenkins

	Github Actions		*Jenkins*	
Build Number	*Build Time*	Status	*Build Time*	Status
1	*2 min 13 sec*	*PASS*	*4 min 21 sec*	*PASS*
2	*2 min 15 sec*	*PASS*	*7 min 1 sec*	*PASS*
3	*2 min 8 sec*	*PASS*	*5 min 30 sec*	*PASS*
4	*2 min 8 sec*	*PASS*	*6 min 9 sec*	*PASS*
5	*2 min 12 sec*	*PASS*	*7 min 12 sec*	*PASS*

Sumber: Diolah Peneliti, 2023.

II. Quality Metrics (Kualitas)

Sebagai suatu panduan, beberapa metrik kualitas berikut ini telah penulis pertimbangkan dengan seksama untuk menjamin bahwa proyek ini dapat berjalan dengan efisiensi dan keandalan yang maksimal:

1. Frekuensi Kegagalan (*Failure Rate*):

   Frekuensi kegagalan mengindikasikan seberapa sering aplikasi mengalami kegagalan atau failure dalam menjalankan fungsionalitasnya. Dengan memonitor dan mengurangi frekuensi kegagalan dapat meningkatkan stabilitas dan kualitas keseluruhan dari proyek ini. Bisa dilihat pada tabel 4.1 tingkat keberhasilan *Github Actions* dan *Jenkins* dalam melakukan proses *CI/CD* dalam 5 kali *deployment* adalah 100%. Hal ini membuktikan bahwa *Github Actions* dan *Jenkins* dapat menjalankan proses *CI/CD* untuk di *deploy* ke *Docker* dengan baik.
2. Waktu Rilis (*Release Time*):

   Memantau waktu rilis setiap perubahan atau fitur baru dapat membantu memastikan bahwa proses *CI/CD* berjalan dengan lancar dan tidak mengalami kendala yang berarti. Targetnya adalah mempercepat waktu rilis untuk merespons perubahan kebutuhan pasar. Jika dilihat pada tabel 4.1 waktu rata rata yang dihabiskan *Github Actions* dalam 5 kali *deployment* adalah 131.2 detik atau 2 menit 11 detik. Dan untuk rata rata waktu yang dihabiskan *Jenkins* dalam 5 kali *deployment* adalah 362.6 detik atau 6 menit 2 detik. Hal ini menunjukan bahwa waktu rata rata untuk 5 kali proses *CI/CD* cenderung lebih cepat *Github Actions* daripada *Jenkins*.

V. SIMPULAN DAN SARAN

A. Kesimpulan

Berdasarkan hasil implementasi dan pengujian pada Implementasi Continuous Integration dan Continuous Deployment pada Pengembangan Aplikasi Website

Menggunakan Docker dan GitHub Actions, dapat disimpulkan bahwa:

1. GitHub Actions membutuhkan waktu lebih cepat untuk menyelesaikan proses CI/CD daripada Jenkins. Jenkins tidak dapat menyelesaikan tugas CI/CD secepat dan seefektif GitHub Actions selama Build, Test, hingga Deploy. Hal ini terlihat dari waktu yang dibutuhkan untuk menjalankan langkah-langkah CI/CD pada kedua platform. GitHub Actions mampu mengurangi waktu pembaruan konfigurasi dan eksekusi proses CI/CD secara keseluruhan, mempercepat siklus pengembangan dan penyampaian aplikasi website.
2. GitHub Actions memberikan kemudahan dalam konfigurasi awal dan pemeliharaan sistem CI/CD. Dengan integrasi yang mudah dengan repositori GitHub, konfigurasi menjadi lebih sederhana dan terpusat. Selain itu, format konfigurasi menggunakan YAML yang mudah dipahami membantu dalam mengatur sistem CI/CD dengan lebih efisien.
3. Selain dari aspek waktu, GitHub Actions juga menawarkan fleksibilitas yang lebih tinggi dalam menyesuaikan alur kerja CI/CD. Fitur-fitur yang terus berkembang dan kemampuan untuk berintegrasi dengan berbagai alat pendukung memberikan keleluasaan dalam mengadaptasi alur kerja sesuai kebutuhan proyek.
4. Menerapkan metode Continuous Integration (CI) dan Continuous Deployment (CD) pada pengembangan aplikasi website memiliki beragam keuntungan, termasuk peningkatan kualitas perangkat lunak, efisiensi pengembangan, pengiriman lebih cepat, deteksi masalah secara real-time, meningkatkan kolaborasi tim, mengurangi risiko pengiriman besar, meningkatkan keandalan aplikasi, meningkatkan skalabilitas, dan memberikan umpan balik yang cepat dari pengguna.

B. Saran

Berdasarkan masalah yang ditemukan penulis dalam penelitian ini, maka penulis memberikan rekomendasi bagi para tim pengembang yang sedang melakukan proses CI/CD. Selain itu, kepada peneliti selanjutnya untuk bekal memperluas pengetahuan tentang implementasi proses CI/CD dengan menggunakan Github Actions atau Jenkins. Hal tersebut ditunjukan agar dapat mendorong penelitian yang serupa menjadi lebih baik.

Penelitian ini menunjukan bahwa saat menggunakan GitHub Actions lebih cepat dan dengan konfigurasi yang lebih sedikit dari Jenkins untuk Continuous Integration dan Continuous Deployment. Mengingat meningkatnya penggunaan GitHub Actions dalam proyek pengembangan aplikasi website, hasil studi ini dapat menjadi acuan untuk menentukan dengan apa proses CI/CD dapat lebih cepat. Dengan menerapkan saran ini, diharapkan proses CI/CD akan menjadi lebih efisien, inovatif, dan memberikan manfaat lebih bagi pengembang dan tim proyek.

REFERENSI

[1] Adiputra, F. (2015). Container Dan Docker: Teknik Virtualisasi Dalam Pengelolaan Banyak Aplikasi Web. Jurnal Simatec, 172.

[2] Ahmadjayadi, C. F. (2016). Melesat Atau Kandas? New Indonesia Dari Smart . Jakarta: Pt Elex Media Komputindo.

[3] Alpery, A., & Arif Fadhly, M. (2021). Implementasi Ci/Cd Dalam Pengembangan Aplikasi Web Menggunakan Docker Dan Jenkins. 9th Applied Business And Engineering Conference, 66-89.

[4] Andrian Alpery, M. A. (2021). Implementasi Ci/Cd Dalam Pengembangan Aplikasi Web Menggunakan Docker Dan Jenkins. 9th Applied Bussiness And Engineering Conderence.

[5] Brown. (2023). Agile Practices For Efficient Software Development. Journal Of Science And Technology Policy, 136-149.

[6] Cha, B. R., Kim, J. W., Moon, H. M., & Pan, S. B. (2017). Global Experimental Verification Of Docker-Based Secured Mvoip To Protect Against Eavesdropping And Dos Attacks. Cha Et Al. Eurasip Journal On Wireless Communications And Networking.

[7] Decan, A., Mens, T., Mazrae, P. R., & Golzadeh, M. (2022). On The Use Of Github Actions In Software Development Repositories.

[8] Decan, Mens, Mazraei, & Golzadeh. (2022). On The Use Of Github Actions In Software Development Repositories. Limassol, Cyprus: Ieee.

[9] Dunn, T. (2023). Automation Of Ci/Cd Pipelines With Github Actions. Github Actions, 55-67.

[10] Duvall, P. M., Matyas, S., & Glover, A. (2007). Continuous Integration: Improving Software Quality And Reducing Risk. Addison-Wesley Professional.

[11] Humble, J., & Farley, D. (2010). Continuous Delivery: Reliable Software Releases Through Build, Test, And Deployment Automation. Addison-Wesley Professional.

[12] Jaeni, Aji S, N., & Laksito, A. D. (2022). Implementasi Continuous Integration/Continuous Delivery (Ci/Cd) Pada Performance Testing Devops. Journal Of Information System Management (Joism), 62-66.

[13] Jaeni, N. A. (2022). Implementasi Continuous Integration/Continuous Delivery (Ci/Cd) Pada Performance Testing Devops. Joism : Jurnal Of Information System Management.

[14] Kumar. (2022). Efficient Software Delivery With Ci/Cd Pipelines. Ci/Cd Pipelines And Software Delivery, 167-184.

[15] Lee. (2021). Containerization For Streamlining Software Development. Atlas Software Development And Data Production, 87-101.

[16] Maryam, N. (2022). Software Development Life Cycle: A Comprehensive Review. Machine Learning, 344-354.

[17] Pachev, B., Stuart, G. K., & Dawson, C. (2022). Continuous Integration For Hpc With Github Actions And Tapis. Pearc.

[18] Pachev, Stuart, & Dawson. (2022). Continuous Integration For Hpc With Github Actions And Tapis. Pearc '22: Practice And Experience In Advanced Research Computing, 1-4.

[19] Permatasari, & Ramadhan. (2021). Pengujian Ansible Playbook Menggunakan Molecule Dan Github Actions. Seminar Nasional Sistem Informasi (Senasif), 2997 - 3004.

[20] Permatasari, D. I., & Ramadhan, A. A. (2021). Pengujian Ansible Playbook Menggunakan Molecule Dan Github Actions. Seminar Nasional Sistem Informasi . Malang.

[21] Peters, B. (2021). Automating The Ci/Cd Workflow With Github Actions. Journal Of Open Source Software, 3061.

[22] Radu, M., & Matei, I. (2019). Using Docker For Application Virtualization. Procedia Computer Science, 353-360.

[23] Sannino, L., & Bruneo, D. (2022). Improving The Reproducibility Of Continuous Integration And Continuous Deployment Pipelines With Github Actions. Journal Of Open Research Software, 33.

[24] Shama, A. M., & Chandra, D. W. (2021). Implementasi Static Application Security Testing Menggunakan Jenkins Ci/Cd

Berbasis Docker Container Pada Pt. Emporia Digital Raya. Jurnal Ilmiah Informatika - Vol. 09 No. 02.
[25] Shama, A. M., & Chandra, D. W. (2021). Implementasi Static Application Security Testing Menggunakan Jenkins Ci/Cd Berbasis Docker Container Pada Pt. Emporia Digital Raya. Jurnal Ilmiah Informatika, 95-99.
[26] Silvana Rasio, R. P. (Mei 2020). Evaluasi User Experience Sistem Informasi Akademik Mahasiswa Pada Perguruan Tinggi Menggunkan User Experience Questionnaire. Jurnal Komputer Terapan Vol. 6, No. 1, 69-78.
[27] Smith. (2021). Streamlining Software Development Process Through Continuous Integration And Deployment. Transactions On Software Engineering, 57-69.
[28] Somya Garg, S. G. (2019). Automated Cloud Infrastructure, Continuous Integration And Continuous Delivery Using Docker With Robust Container Security. Ieee Conference On Multimedia Information Processing And Retrieval (Mipr).
[29] Statista.Com. (2022, Januari). Most Popular Social Networks Worldwide As Of January 2022, Ranked By Number Of Monthly Active Users. Diambil Kembali Dari Https://Www.Statista.Com/Statistics/272014/Global-Social-Networks-Ranked-By-Number-Of-Users/
[30] Tohirin, Utam, S. F., Widianto, S. R., & Mauludyansah, W. A. (2020). Implementasi Devops Pada Pengembangan Aplikasi E-Skrining Covid-19. Jurnal Multinetics Vol. 6 No. 1.
[31] Tohirin, Utami, S. F., Widianto, S. R., & Al Mauludyansah, W. (2020). Implementasi Devops Pada Pengembangan Aplikasi E-Skrining Covid-19. Multinetics, 15-20.
[32] Wan, Guan, Wang, Bai, & Choi. (2018). Application Deployment Using Microservice And Docker Containers: Framework And Optimization. Journal Of Network And Computer Applications, 97-109.
[33] Wan, X., Guan, X., Wang, T., Bai, G., & Choi, B. Y. (2018). Application Deployment Using Microservice And Docker Containers: Framework And Optimization. Journal Of Network And Computer Applications 119.
[34] Wessel, M., Vargovich, J., Gerosa, M. A., & Treude, C. (2022). Github Actions: The Impact On The Pull Request Process.
[35] Wessel, Vargovich, Gerosa, & Treude. (2022). Github Actions: The Impact On The Pull Request Process. Software Engineering , 45-51.
[36] Wiedemann, G. (2020). Continuous Deployment Of Safety-Critical Software In Industrial Automation: A Survey. Ieee Transactions On Industrial Informatics, 3115-3126.
[37] Wilson, W. (2021). Improving Software Development Process Through Ci/Cd Pipeline. Continuous Integration And Deployment, 102-117..