REPOSITOR, Vol. 6, No. 4, November 2024, Pp. 313-322
ISSN : 2714-7975
E-ISSN : 2716-1382

Otomatisasi Proses Deployment dengan Metode CI/CD Menggunakan Jenkins dan Docker Pada Web Service i-Lab

Muhammad Syauqi Amiq Amrullah*[1], Aminudin[1], Gita Indah Marthasari[1]
Universitas Muhammadiyah Malang
syauqiamiq12@webmail.umm.ac.id*

Abstrak

Proses deployment dalam pengembangan aplikasi seringkali menghadapi berbagai kendala, terutama jika dilakukan secara manual, yang dapat menimbulkan kesalahan dan memerlukan waktu yang panjang. Penelitian ini bertujuan untuk mengotomatisasi proses deployment pada web service i-Lab di Laboratorium Informatika Universitas Muhammadiyah Malang dengan menggunakan metode Continuous Integration/Continuous Deployment (CI/CD) melalui alat Jenkins dan Docker. Metode CI/CD dipilih karena kemampuannya untuk mengintegrasikan dan mengimplementasikan aplikasi secara otomatis, mengurangi intervensi manual, serta meningkatkan efisiensi dan kualitas deployment. Dalam penelitian ini, Jenkins digunakan sebagai alat utama untuk CI/CD dan Docker untuk containerization, guna memastikan lingkungan yang konsisten dan efisien. Implementasi ini bertujuan untuk mengatasi masalah deployment manual yang sering menyebabkan kesalahan, dan memungkinkan pengembang lebih fokus pada kualitas kode. Pengujian dilakukan untuk mengukur durasi dan kualitas dari otomatisasi deployment yang dirancang menggunakan Time-based Metric dan Quality-based Metric. Terdapat perbedaan waktu pada proses deployment namun tidak jauh berbeda. Hasil pengujian kualitas menunjukan tingkat Test Pass Rate dengan nilai 100%. Kesimpulan: Hasil penelitian menunjukan kualitas CI/CD yang dibuat berjalan dengan baik pada penerapan di web service i-Lab milik Laboratorium Informatika Universitas Muhammadiyah Malang, terjadi perbedaan durasi waktu proses CI/CD namun tidak jauh berbeda.

Kata Kunci: *Continuous Improvement, Continuous Deployment, Docker, Web Service, CI/CD*

Abstract

The deployment process in application development often faces various obstacles, especially if done manually, which can cause errors and take a long time. This study aims to automate the deployment process on the i-Lab web service at the Informatics Laboratory of the University of Muhammadiyah Malang using the Continuous Integration/Continuous Deployment (CI/CD) method through the Jenkins and Docker tools. The CI/CD method was chosen because of its ability to integrate and implement applications automatically, reduce manual intervention, and improve deployment efficiency and quality. In this study, Jenkins is used as the main tool for CI/CD and Docker for containerization, to ensure a consistent and efficient environment. This implementation aims to overcome the problem of manual deployment that often causes errors, and allows developers to focus more on code quality. Testing was conducted to measure the duration and quality of deployment automation designed using Time-based Metrics and Quality-based Metrics. There is a difference in time in the deployment process but not much different. The results of the quality test show a Test Pass Rate level with a value of 100%. Conclusion: The results of the study show that the quality of CI/CD created runs well in the implementation of the i-Lab web service owned by the Informatics Laboratory of the University of Muhammadiyah Malang, there is a difference in the duration of the CI/CD process but it is not much different.

Keywords: *Continuous Improvement, Continuous Deployment, Docker, Web Service, CI/CD*

1. Pendahuluan

Pengembangan aplikasi seringkali memiliki beberapa kendala dalam proses produksinya, pengembang perangkat lunak melibatkan serangkaian tahapan yang mencakup analisis, desain, pengembangan, dan tahap *deployment* yang krusial untuk mempublikasikan atau mendistribusikan aplikasi. Tahap *deployment* menentukan hasil akhir dari aplikasi tersebut. Akan tetapi proses ini seringkali terhambat oleh penggunaan sistem manual. Pengembang

memasukkan aplikasi ke *server* dan menjalankannya. Ketika terdapat pembaruan pada aplikasi, langkah-langkah tersebut harus diulang. Cara ini dianggap kurang tepat dalam pengembangan perangkat lunak, karena mengharuskan pengembang melakukan *deployment* secara berulang-ulang saat terjadi pengembangan pada aplikasi yang telah dibuat[1].

Untuk mengatasi permasalahan tersebut, maka dibutuhkan sebuah metode dalam proses *deployment* perangkat lunak. Metode yang sering digunakan adalah CI/CD[2]. CI/CD, singkatan dari *Continuous Integration/Continuous Deployment*, yang berfungsi menghubungkan tim operasional dan tim pengembang untuk otomatisasi proses *build*, pengujian, dan implementasi aplikasi. Penerapan CI/CD memungkinkan pengembang melakukan otomatisasi dan pemantauan kontinu dalam pengembangan perangkat lunak, menyederhanakan rilis dan pembaruan perangkat lunak dengan cepat, aman, dan tata kelola yang teratur[3]. *Continuous Integration* yang sukses melibatkan perubahan kode yang teratur diuji dan digabungkan ke repositori bersama. *Continuous Delivery* dan/atau *Continuous Deployment* (CD) adalah konsep terkait yang mencakup otomatisasi lebih lanjut dalam pengembangan perangkat lunak. *Continuous Delivery* mengotomatisasi pengujian perubahan aplikasi untuk melacak *bug*, lalu diunggah ke repositori, sehingga meminimalkan usaha dalam *deployment*. *Continuous Deployment*, secara otomatis merilis perubahan dari repositori ke lingkungan produksi, mengatasi masalah *deployment* manual yang dapat memperlambat kinerja aplikasi[4].

Penelitian ini berfokus pada proses *deployment web service* iLab yang berada di Laboratorium Informatika Universitas Muhammadiyah Malang. Laboratorium Informatika di Universitas Muhammadiyah Malang (UMM) adalah sebuah lembaga pendidikan yang berperan sebagai pusat praktik dan percobaan yang digunakan oleh mahasiswa dan masyarakat umum untuk melakukan penelitian dan mendapatkan konsultasi di bidang teknik informatika. Di dalam laboratorium ini, terdapat beberapa bidang studi yang mencakup Program Studi Informatika, seperti Rekayasa Perangkat Lunak, Keamanan Jaringan, Analisis Data, dan Pengembangan Permainan[5]. Pada Laboratorium Informatika ini terdapat beberapa aplikasi berbasis *website* antara lain iLab, Simponia, Simutu, dan lain – lainnya.

Penelitian ini berdasarkan kegiatan *maintenance* beberapa aplikasi yang disebutkan diatas, sehingga dalam penelitian ini dapat dilakukan observasi secara langsung terhadap masalah yang terjadi dalam proses *maintenance* aplikasi – aplikasi tersebut. Masalah yang terjadi adalah seringkali terjadi *error* tanpa sepengetahuan pengembang aplikasi dan baru terdeteksi pada saat proses *deployment* di *server*. Selain itu, proses *deployment* masih dilakukan secara manual dengan cara mengetik manual sintaks linux yang diperlukan, Hal ini menyebabkan sering kali terjadinya *error* yang tidak terduga. Laboratorium Informatika UMM juga melakukan perubahan pengembang perangkat lunak yang bertugas melakukan *maintenance* setiap tahunnya. Hal ini menyebabkan proses *deployment* menjadi lebih sering terjadi *error* dikarenakan proses *Transfer of Technology* antara *programmer* lama terhadap *programmer* baru yang masih belum tepat.

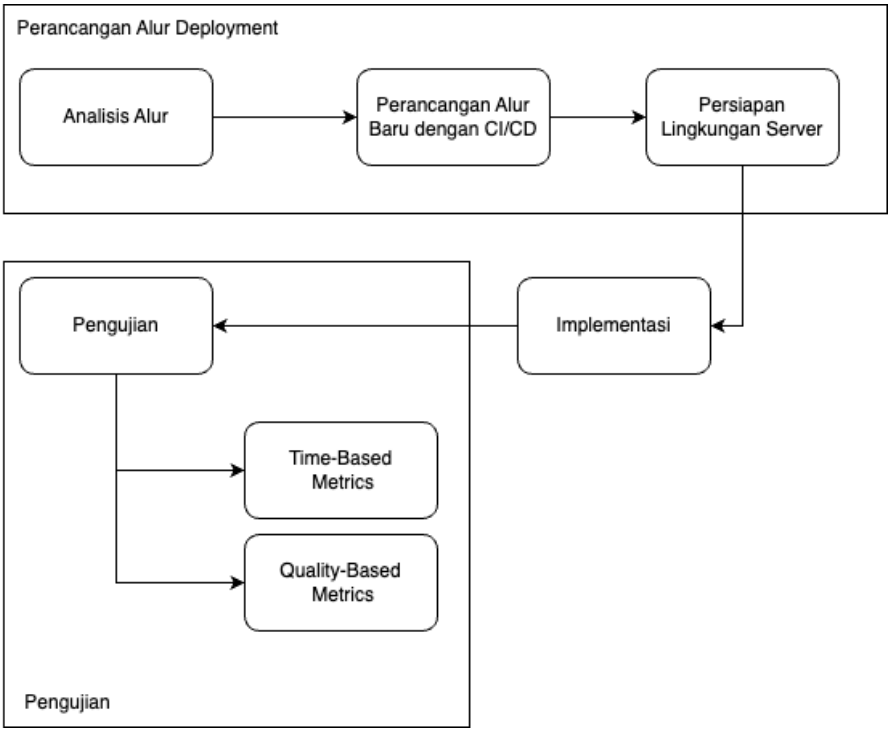
Pada proses penelitian ini, peneliti menggunakan *web service* i-Lab sebagai objek penelitian yang akan diterapkan CI/CD pada proses *deployment* nya. *Web Service* adalah aplikasi yang diakses melalui internet oleh aplikasi lain. Pengiriman pesan antar aplikasi menggunakan format pertukaran data. Fungsi utamanya adalah memberikan layanan kepada sistem lain dalam bentuk informasi atau data. *Web Service* memfasilitasi interaksi antar sistem dengan menyimpan data dalam format JSON atau XML. Kelebihannya melibatkan akses yang tidak terhalang oleh perbedaan platform, sistem operasi, atau bahasa pemrograman[6].

Proses pengembangan *Web Service* iLab yang tepat menjadi kebutuhan bagi Laboratorium Informatika UMM. Salah satu pendekatan yang dapat digunakan untuk mencapai tujuan tersebut salah satunya adalah penerapan metode CI/CD dalam proses otomatisasi *deployment web service* iLab. Banyak tools yang dapat digunakan untuk CI/CD seperti Jenkins, GitLab CI/CD, CircleCI, AWS CodePipeline, dan lain - lain. Dalam penerapan CI/CD pada *web service* iLab, peneliti memilih Jenkins, sebuah tools open source dengan kemampuan *build* dan *deployment* otomatis yang cepat dan efisien. Jenkins, yang sangat terkenal, menyediakan ratusan plugin untuk mendukung optimalisasi dalam proses CI/CD[7].

Penelitian ini menggunakan teknik *Containerization* dalam proses pemasangan tools yang akan digunakan dalam proses CI/CD ini. *Container* adalah bentuk virtualisasi yang lebih ringan dan merupakan alternatif bagi teknik virtualisasi berbasis *hypervisor*. Berbeda dengan pendekatan tradisional, *container* tidak memerlukan sistem operasi tersendiri untuk setiap mesin virtual. Sebaliknya, banyak *container* dapat berbagi sistem operasi yang sama, memberikan

lingkungan yang lebih efisien dan ringan[7]. Dari berbagai tools virtualisasi yang tersedia, peneliti memilih Docker sebagai alat untuk memasang *tools* dalam proses CI/CD. Docker adalah proyek open source untuk mempermudah *Software Engineer*, *DevOps Engineer*, atau *System Administrator* dalam membangun, mengemas, dan menjalankan aplikasi di dalam kontainer di mana saja. Sebagai bentuk virtualisasi, Docker dapat digunakan untuk sistem operasi, *server*, *web server*, atau *server* basis data, memungkinkan pengembang mengembangkan aplikasi sesuai dengan spesifikasi *server* yang diinginkan[8].

2. Metode Penelitian



Gambar 1. Tahapan Penelitian

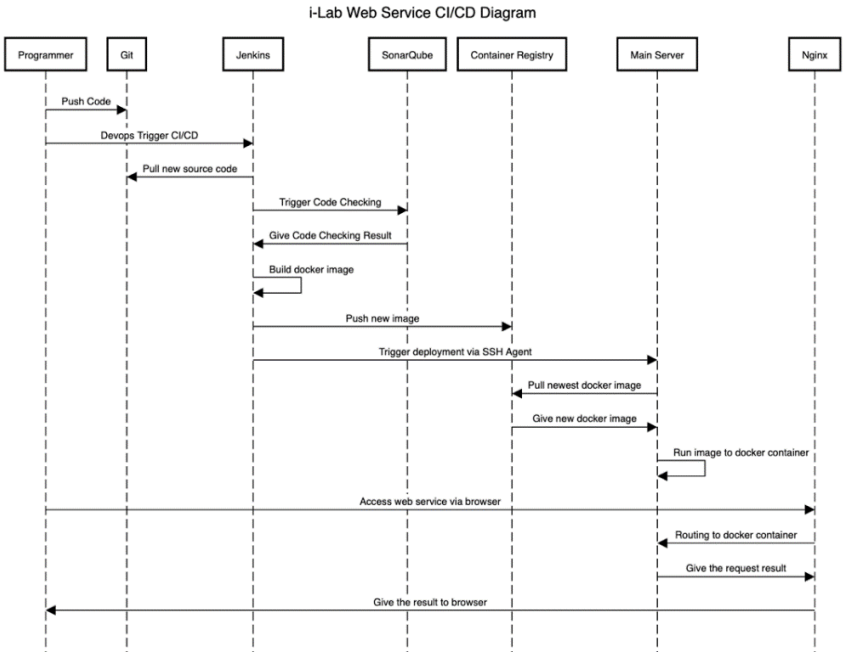
Pada Gambar 1 terlihat bahwa proses penelitian diawali dengan perancangan arsitektur *server* yang akan digunakan untuk penerapan CI/CD pada *web service* i-Lab. Proses yang kedua adalah implementasi metode CI/CD pada proses *deployment web service* i-Lab menggunakan beberapa *tools* yang sudah ditentukan. Proses yang terakhir adalah pengujian untuk mengetahui performa CI/CD yang dibuat dan seberapa efektif metode CI/CD dalam menyelesaikan masalah yang terjadi pada proses *deployment web service* i-Lab.

2.1 Perancangan Arsitektur

Proses perancangan arsitektur pada sistem CI/CD *web service* i-Lab diperlukan dalam penelitian ini. Hal ini bertujuan agar peneliti mendapatkan gambaran arsitektur sebelum menerapkan CI/CD dan sesudah menerapkan CI/CD pada sistem *web service* i-Lab pada Laboratorium Informatika Universitas Muhammadiyah Malang. Pada tahapan ini juga akan dilakukan persiapan lingkungan server yang digunakan untuk menerapkan proses CI/CD.

2.2 Rancangan Implementasi

Proses implementasi otomatisasi *deployment web service* i-Lab dengan metode CI/CD membutuhkan beberapa *tools.* Gambar dibawah ini merupakan proses yang akan digunakan pada tahapan implementasi.



Gambar 2. Sequence Diagram Tahapan Implementasi CI/CD

Pada Gambar 2 proses implementasi CI/CD *web service* i-Lab akan melibatkan Git, Jenkins, SonarQube, Main Server, dan Nginx dalam penerapannya. Penggunaan *tools* tersebut saling berkaitan satu dengan lainnya, proses akan dimulai dari menggabungkan kode pada Gitlab hingga *web service* berhasil di *deploy* pada *server* utama dan dapat diakses oleh pengguna melalui Nginx. Proses yang terjadi dimulai dari *developer* menggabungkan kode ke repositori Gitlab, lalu *developer* yang bertugas menjadi *devops* atau admin dari aplikasi Jenkins akan melakukan *trigger* CI/CD manual pada Jenkins. Selanjutnya Jenkins akan melakukan *pull code* terbaru dari repositori Gitlab dan akan melakukan *trigger* secara otomatis terhadap SonarQube sebagai *Static Application Security Testing* (SAST) yang digunakan dalam penelitian ini. Selanjutnya SonarQube akan melakukan *code quality checking* dan mengembalikan hasilnya ke Jenkins, jika kode aman dan lulus proses *code quality checking* maka Jenkins akan melakukan *build* sehingga menjadi *docker image* yang akan di upload ke *container registry*. Proses yang terakhir adalah Jenkins akan melakukan *remote deployment* dengan SSH pada *server* utama, sehingga *server* utama akan melakukan *download docker image* terbaru dari *container registry* dan menjalankannya sebagai *container docker* di server utama dan Nginx akan mengarahkan *traffic request* dari *client* ke *docker container* tersebut[9].

2.3 Rancangan Pengujian

Pada penelitian ini penulis akan melakukan pengujian terhadap hasil CI/CD yang sudah diterapkan pada sistem *web service* i-Lab. Untuk mengetahui tingkat kualitas dan performa dari alur CI/CD maka dilakukan pengujian sistem menggunakan metode *Quality-Based Metrics* dan *Time-Based Metrics*[10].

3. Hasil Penelitian dan Pembahasan

Pada tahap ini merupakan elemen yang krusial dalam struktur suatu penelitian, karena disinilah hasil dari penelitian yang dilakukan dijelaskan secara rinci dan dianalisis secara mendalam. Peneliti akan secara terstruktur mengungkapkan temuan utama dari penelitian ini. Lebih dari itu, bagian ini juga akan mempersembahkan diskusi yang teliti untuk menafsirkan signifikansi temuan tersebut dalam konteks teori yang telah dikaji sebelumnya.

3.1 Implementasi

Dalam tahap ini peneliti akan menjelaskan secara rinci mengenai proses implementasi CI/CD yang telah dibuat. Peneliti akan membahas langkah-langkah yang diambil untuk menerapkan otomatisasi *deployment* pada *web service* i-Lab. Dengan menguraikan proses

implementasi ini, diharapkan pembaca akan mendapatkan pemahaman yang lebih baik tentang bagaimana proses otomatisasi dijalankan dari awal hingga akhir.

3.1.1 Kontainerisasi *Tools* CI/CD

Pada tahap ini peneliti melakukan instalasi *tools* yang akan mendukung proses berjalan nya CI/CD pada *web service* i-Lab. Proses instalasi dilakukan pada *virtual machine* khusus untuk CI/CD dan didalamnya terpasang Docker sebagai alat virtualisasi untuk membagi *resource* yang ada. Untuk memudahkan proses virtualisasi, peneliti menggunakan Portainer sebagai alat untuk mengelola Docker *container.* Portainer adalah alat bantu untuk mengelola *container* yang dirancang untuk menyederhanakan proses pengelolaan *image container* yang dilengkapi dengan antarmuka grafis (GUI), Portainer menawarkan kemudahan penggunaan dibandingkan dengan perintah berbasis teks yang rumit[11]. Dalam konteks manajemen Docker, Portainer memungkinkan pengguna untuk membuat dan mengelola Docker *image* sesuai kebutuhan mereka dengan cara yang lebih intuitif dan *user-friendly*. Pada *server* CI/CD peneliti melakukan instalasi dua *tools* utama yaitu Jenkins dan SonarQube. Jenkins akan digunakan untuk otomatisasi proses CI/CD sedangkan SonarQube akan digunakan untuk proses pengecekan kualitas kode. Untuk membuat Docker *container* yang berisi aplikasi Jenkins, maka dibutuhkan *script* Docker Compose yang akan dieksekusi oleh Portainer.

3.1.2 Implementasi *Pipeline* dengan Jenkins

Pada tahap implementasi *pipeline* pada aplikasi Jenkins dibutuhkan beberapa *file* yang akan di eksekusi oleh aplikasi Jenkins yaitu Jenkinsfile dan Dockerfile. Jenkinsfile akan digunakan untuk membuat perintah *pipeline* sedangkan Dockerfile digunakan untuk perintah yang berkaitan dengan virtualisasi Docker. *Script* Dockerfile yang digunakan untuk melakukan *build* aplikasi *web service* i-Lab hingga menjadi Docker *image.*

3.2 Pengujian

Pada tahap ini peneliti melakukan pengujian guna mengetahui performa dan kualitas dari CI/CD yang telah dibuat dan diterapkan pada *web service* i-Lab di Laboratorium Informatika Universitas Muhammadiyah Malang. Pengujian dilakukan dengan cara melakukan *trigger* proses CI/CD pada aplikasi Jenkins sebanyak 10 kali iterasi. Selanjutnya peneliti akan mengamati dan mencatat waktu dari setiap *pipeline stage* CI/CD yang berjalan. Hasil dari pengamatan dapat dilihat pada Tabel 1, Tabel 2, dan Tabel 3 berikut:

Table 1. Time-based Metric Testing Result

No	Build Number	Duration (Second)
1	Build #1	180,489
2	Build #2	190,477
3	Build #3	173,455
4	Build #4	164,506
5	Build #5	179,506
6	Build #6	184,512
7	Build #7	168,467
8	Build #8	168,449
9	Build #9	152,476
10	Build #10	180,492
Average CI/CD time in second		174,2829
Average CI/CD time in minute		2,904715

Table 2. Quality-based Metric Testing Result

No	Build Number	Status	Error
1	Build #1	SUCCESS	-
2	Build #2	SUCCESS	-
3	Build #3	SUCCESS	-
4	Build #4	SUCCESS	-
5	Build #5	SUCCESS	-
6	Build #6	SUCCESS	-

7	Build #7	SUCCESS	-
8	Build #8	SUCCESS	-
9	Build #9	SUCCESS	-
10	Build #10	SUCCESS	-

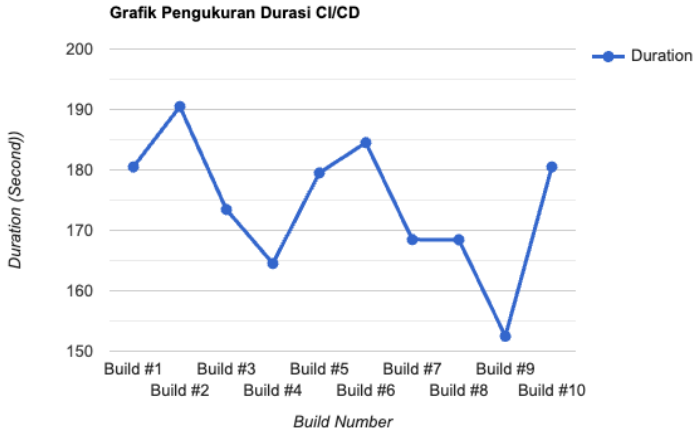
Table 3. Pipeline CI/CD Duration Result

Build Number	A	B	C	D	E	F	G
Build #1	2	10	61	50	0,489	55	2
Build #2	2	10	69	50	0,477	55	2
Build #3	1	10	56	50	0,455	54	2
Build #4	1	10	52	47	0,506	53	1
Build #5	1	10	63	49	0,506	55	1
Build #6	3	13	59	49	0,512	58	2
Build #7	2	10	62	48	0,467	45	1
Build #8	1	10	58	49	0,449	48	2
Build #9	1	9	53	48	0,476	40	1
Build #10	1	10	55	65	0,492	48	1
Average Duration in Second	1,5	10,2	58,8	50,5	0,4289	51,1	1,5
Standard Deviation	0,70	1,03	4,89	4,92	0,02	5,62	0,52
Average Standard Deviation				2,53			

3.3 Analisis Pengujian

Analisis mendalam akan dilakukan untuk mengevaluasi hasil penerapan CI/CD pada *web service* i-Lab. Peneliti juga akan menyoroti waktu dan kualitas dari *pipeline* CI/CD yang telah dibuat. Diharapkan analisis ini memberikan pemahaman yang mendalam tentang kualitas sistem dan menawarkan wawasan untuk pengembangan masa depan.

Dari hasil pengukuran durasi pada CI/CD *web service* i-Lab milik Laboratorium Informatika Universitas Muhammadiyah Malang diatas, Peneliti menemukan adanya ketidakstabilan pada durasi proses CI/CD. Hasil pengukuran durasi dalam bentuk grafik dapat dilihat pada Gambar 3 berikut:



Gambar 3. Grafik Durasi CI/CD

Pada Gambar 3 dapat dilihat bahwa terdapat perbedaan durasi dari proses CI/CD pada web service i-Lab. Berdasarkan data pada Tabel 3, peneliti mengambil data dengan Standar Deviasi diatas rata – rata yaitu pipeline stage C, D dan F untuk dianalisis lebih lanjut. Peneliti mengambil sampel dua data yang memiliki durasi terlama dan tercepat yakni data dengan Build Number Build #2 dan Build #4 untuk pipeline stage C dan D, peneliti juga mengambil sampel data Build #6 dan Build #9 untuk pipeline stage F. Selanjutnya peneliti akan melakukan analisis lebih mendalam dari segi script pada pipeline stage C, D dan F. Pada pipeline stage C peneliti menemukan perbedaan durasi pada saat eksekusi beberapa script. Gambar perbedaan durasi tersebut dapat dilihat pada Gambar 4 dan Gambar 5 dibawah ini:

Build #2

```
#5 [4/8] RUN go mod download
#5 sha256:27a232b57bc8d037fb4552bbbadad35261b17081f40727f3d53208f33bf65c4b
#5 DONE 19.1s

#12 [6/8] RUN go install github.com/swaggo/swag/cmd/swag@latest
#12 sha256:d35afd8f2f32ce87aaf12859988144ff478c0b72431c209f10e149a6838a9fd8
#12 1.166 go: downloading github.com/swaggo/swag v1.16.3
#12 1.378 go: downloading github.com/urfave/cli/v2 v2.3.0
#12 1.383 go: downloading github.com/go-openapi/spec v0.20.4
#12 1.435 go: downloading golang.org/x/tools v0.7.0
#12 1.566 go: downloading sigs.k8s.io/yaml v1.3.0
#12 1.802 go: downloading github.com/go-openapi/jsonpointer v0.19.5
#12 1.812 go: downloading github.com/go-openapi/jsonreference v0.19.6
#12 1.837 go: downloading github.com/go-openapi/swag v0.19.15
#12 2.589 go: downloading github.com/cpuguy83/go-md2man/v2 v2.0.0-20190314233015-f79a8a8ca69d
#12 2.611 go: downloading github.com/mailru/easyjson v0.7.6
#12 2.616 go: downloading github.com/PuerkitoBio/purell v1.1.1
#12 2.717 go: downloading github.com/russross/blackfriday/v2 v2.0.1
#12 2.740 go: downloading golang.org/x/net v0.17.0
#12 DONE 8.4s

[8/8] RUN go build -o output
sha256:7f17b2ac666d8988325ea15a67cc2b3ca70bf37a5fa96c0b01c9d718ea5b5f0c
DONE 14.1s

exporting to image
sha256:e7f468e650d35461438f185c2a779b8978a5d18ee35cc0da96fde1af20552db5
exporting layers
exporting layers 6.2s done
writing image sha256:85d284722f86a8d****cf45715edfbc3b590ee652bda1604c704c673c9a559d797 done
naming to ****/new-ilab-api:stage-latest done
DONE 6.2s
```

Gambar 4. Sampel Pipeline Stage C (Build Docker Image) pada Build #2

Build #4

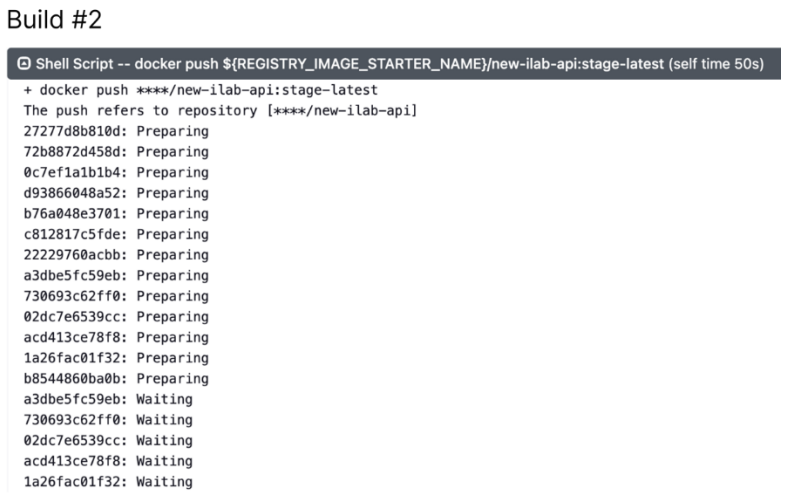
```
#4 [4/8] RUN go mod download
#4 sha256:ff68ae376a270f1371a235b45961b8a7df92aa981a1847f348e7bf0ae3e032e8
#4 DONE 13.9s

#11 [6/8] RUN go install github.com/swaggo/swag/cmd/swag@latest
#11 sha256:14ba4d6201d65ac3f17e6415be9cf8fe8c2010357a25b3b2924549e12714fdd7
#11 1.063 go: downloading github.com/swaggo/swag v1.16.3
#11 1.275 go: downloading github.com/urfave/cli/v2 v2.3.0
#11 1.278 go: downloading github.com/go-openapi/spec v0.20.4
#11 1.296 go: downloading golang.org/x/tools v0.7.0
#11 1.345 go: downloading sigs.k8s.io/yaml v1.3.0
#11 1.712 go: downloading github.com/go-openapi/swag v0.19.15
#11 1.712 go: downloading github.com/go-openapi/jsonreference v0.19.6
#11 1.713 go: downloading github.com/go-openapi/jsonpointer v0.19.5
#11 1.813 go: downloading github.com/cpuguy83/go-md2man/v2 v2.0.0-20190314233015-f79a8a8ca69d
#11 1.858 go: downloading github.com/mailru/easyjson v0.7.6
#11 1.895 go: downloading github.com/PuerkitoBio/purell v1.1.1
#11 1.959 go: downloading github.com/russross/blackfriday/v2 v2.0.1
#11 2.034 go: downloading golang.org/x/net v0.17.0
#11 DONE 7.3s

) [8/8] RUN go build -o output
) sha256:4f65e56f692e1da3ec153b468fae5c1c64695a578b26a07ba8a08ae02320bf73
) DONE 14.9s

.3 exporting to image
.3 sha256:a6b6e2fa6a28baaf07142f58f9726f28eb356f30bac64f0d801abcb3be1b1fc4
.3 exporting layers
.3 exporting layers 5.7s done
.3 writing image sha256:ef4cfd12ab7101e42522a48aff07fb56****358685b71abde9a6ce2cb9de1f8d91 don
.3 naming to ****/new-ilab-api:stage-latest done
.3 DONE 5.8s
```

Gambar 5. Sampel Pipeline Stage C (Build Docker Image) pada Build #4

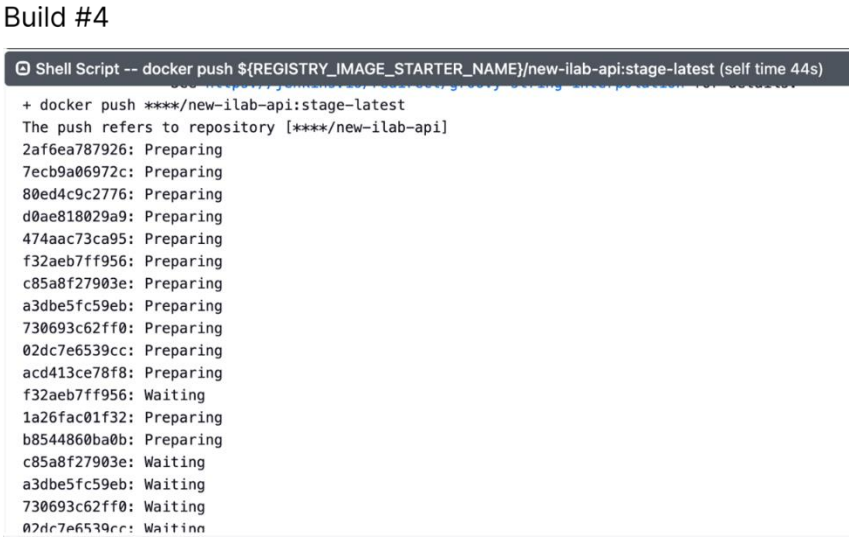
Pada Gambar 4 dan 5 terlihat bahwa dari kedua *pipeline stage* terdapat perbedaan waktu yang signifikan pada proses ke 4, 6 dan 8. Pada proses ke 4 terdapat proses eksekusi *script "go mod download"* yang berfungsi mengunduh semua *package* yang akan digunakan pada *web service* i-Lab, sedangkan pada proses ke 6 terdapat eksekusi *script "go install github.com/swaggo/swag/cmd/swag@latest"* yang berfungsi untuk mengunduh sekaligus menginstall *package* Swaggo yang digunakan untuk dokumentasi API pada *web service* i-Lab

pada Laboratorium Universitas Muhammadiyah Malang. Sedangkan pada proses ke 8 terdapat eksekusi *script "go build -o output"* yang berfungsi untuk melakukan *compile* seluruh kode menjadi sebuah aplikasi *executable* yang akan di *export* menjadi sebuah Docker *image.* Dari kedua perbandingan diatas terdapat perbedaan pada proses ke 4 sebanyak 5,2 detik, pada proses ke 6 sebanyak 1,1 detik, pada proses ke 8 sebanyak 0,8 detik dan pada proses *export* menjadi sebuah Docker *image* sebanyak 0,4 detik

Pada *pipeline stage* D, peneliti menemukan perbedaan waktu yang cukup signifikan pada saat eksekusi *script*. Gambar perbedaan waktu tersebut dapat dilihat pada Gambar 6 dan Gambar 7 dibawah ini:



Gambar 6. Sampel Pipeline Stage D (Push Image to Registry) pada Build #2



Gambar 7. Sampel Pipeline Stage D (Push Image to Registry) pada Build #4

Pada Gambar 6 dan 7 terlihat bahwa dari kedua *pipeline stage* terdapat perbedaan waktu yang signifikan pada proses *upload* Docker *image*. Pada *pipeline stage* ini hanya terdapat *script "docker push ****/new-ilab-api:stage-latest"* yang berfungsi untuk melakukan *upload* Docker *image* pada *server* CI/CD ke Gitlab *container registry*.

Pada *pipeline stage* F, peneliti menemukan perbedaan waktu yang cukup signifikan pada saat eksekusi *script*. Peneliti menggunakan *pipeline stage F* pada *Build #6 dan Build #9.* Gambar perbedaan waktu tersebut dapat dilihat pada gambar dibawah ini:

Build #6

```
Shell Script -- ssh ${DB_USERNAME}@${REMOTE_SERVER} 'docker pull ${REGISTRY_IMAGE_STARTER_NAME}/new-ilab-api:stage-latest' (self time 50s)
+ ssh ****@**** docker pull ****/new-ilab-api:stage-latest
stage-latest: Pulling from labit-riset-umm/new-ilab-api
012c0b3e998c: Already exists
00046d1e755e: Already exists
9f13f5a53d11: Already exists
190fa1651026: Already exists
0808c6468790: Already exists
5ec11cb68eac: Already exists
c11de81f2018: Pulling fs layer
e25c95a8d513: Pulling fs layer
696539185fb1: Pulling fs layer
b6a4e3d7569f: Pulling fs layer
7691c0670bb4: Pulling fs layer
00d3b8e833e7: Pulling fs layer
eeb8eb7d2d66: Pulling fs layer
b6a4e3d7569f: Waiting
7691c0670bb4: Waiting
eeb8eb7d2d66: Waiting
00d3b8e833e7: Waiting
c11de81f2018: Verifying Checksum
```

Gambar 8. Sampel Shell Script Pipeline Stage F (Deploy to Server) pada Build #6

Build #9

```
Shell Script -- ssh ${DB_USERNAME}@${REMOTE_SERVER} 'docker pull ${REGISTRY_IMAGE_STARTER_NAME}/new-ilab-api:stage-latest' (self time 33s)
+ ssh ****@**** docker pull ****/new-ilab-api:stage-latest
stage-latest: Pulling from labit-riset-umm/new-ilab-api
012c0b3e998c: Already exists
00046d1e755e: Already exists
9f13f5a53d11: Already exists
190fa1651026: Already exists
0808c6468790: Already exists
5ec11cb68eac: Already exists
cde85a888995: Pulling fs layer
51d7fc23506c: Pulling fs layer
2e2306eb9c1b: Pulling fs layer
886a37c281b6: Pulling fs layer
a5525428dca8: Pulling fs layer
be6e908e6fdf: Pulling fs layer
e3b5c0ee1228: Pulling fs layer
a5525428dca8: Waiting
be6e908e6fdf: Waiting
e3b5c0ee1228: Waiting
886a37c281b6: Waiting
51d7fc23506c: Verifying Checksum
```

Gambar 9. Sampel Shell Script Pipeline Stage F (Deploy to Server) pada Build #9

Pada Gambar 8 dan 9 terlihat bahwa dari kedua *pipeline stage* terdapat perbedaan waktu sebanyak 17 detik pada proses eksekusi *script "ssh ****@**** docker pull ****/new-ilab-api:stage-latest"* yang berfungsi untuk mengunduh Docker *image* terbaru pada Gitlab *container registry* dengan cara Jenkins melakukan eksekusi *script* tersebut pada *server* utama melalui *Secure Shell* (SSH).

Pada tahap analisis yang terakhir yaitu analisis mengenai kualitas dari CI/CD yang diterapkan pada *web service* i-Lab di Laboratorium Informatika Universitas Muhammadiyah Malang. Peneliti menggunakan *Quality-based Metric* dalam pengujian untuk mencari hasil *Test Pass Rate* dan menyimpulkan kualitas dari CI/CD yang telah dibuat. Peneliti mencari nilai *Test Pass Rate* berdasarkan Persamaan 1 dibawah ini:

$$TestPastRate = \frac{TotalSuccessBuild}{Total\ Build} x100 \quad (1)$$

Berdasarkan perhitungan dengan rumus di atas diperoleh hasil seperti berikut:

$$TestPastRate = \frac{10}{10} x100 = 100\% \quad (1)$$

Berdasarkan hasil perhitungan *Test Pass Rate* pada Persamaan 1, hasil dari *Test Pass Rate* pada pengujian CI/CD *web service* i-Lab bernilai 100%. Pengujian berlangsung sebanyak 10 kali iterasi dengan hasil keseluruhan proses bernilai "SUCCESS" yang artinya CI/CD berjalan dengan lancar dari awal hingga *web service* i-Lab dapat diakses oleh pengguna. Dengan demikian CI/CD yang diterapkan pada *web service* i-Lab mampu berjalan secara konsisten dengan kualitas baik.

Berdasarkan analisis pengujian menggunakan metode Time-based Metric dan Quality-based Metric diatas, peneliti menyimpulkan terdapat perbedaan durasi pada setiap pipeline stage yang terjadi pada proses CI/CD web service i-Lab. Berdasarkan data pada Tabel 3 dapat dilihat bahwa Standar Deviasi pada setiap pipeline stage tidak terlalu besar, hal ini membuktikan bahwa terdapat perbedaan durasi pada pipeline stage namun tidak jauh berbeda. Kualitas CI/CD juga dapat dibuktikan dengan data pada Tabel 2 yang menunjukan nilai "SUCCESS" pada seluruh percobaan deployment dan perhitungan Test Pass Rate yang menunjukan nilai 100%.

4. Kesimpulan

Metode CI/CD berhasil dirancang dan diterapkan pada *web service* i-Lab milik Laboratorium Informatika Universitas Muhammadiyah Malang guna mengotomatisasi proses *deployment* dari *web service* i-Lab yang selama ini dilakukan dengan cara manual. Proses otomatisasi diawali dengan tahapan *Continuous Improvement* (CI) yang dimulai dengan cara melakukan *trigger* satu kali klik pada Jenkins dan selanjutnya proses CI akan berjalan secara otomatis dimulai dari *pull source code* hingga Docker *image* berhasil dibuat dan diunggah ke Gitlab *container registry*. Selanjutnya Jenkins akan melakukan tahapan *Continuous Deployment* (CD) secara otomatis dengan cara mengunduh Docker *image* yang sudah tersimpan pada *container registry* dan menjalankan Docker *image* tersebut pada *server* utama melalui protokol *Secure Shell* (SSH) sehingga *web service* i-Lab dapat diakses oleh pengguna.

Peneliti berhasil mengukur kualitas dan durasi dari CI/CD yang dibuat. Pada hasil pengukuran durasi terdapat perbedaan durasi dari proses CI/CD dari awal hingga akhir tetapi tidak jauh berbeda. Hal ini disebabkan oleh pipeline stage "Build Docker Image", "Push Image to Registry" dan "Deploy to Server". Peneliti juga berhasil mengukur kualitas dari CI/CD dengan cara menjalankan proses CI/CD sebanyak 10 kali iterasi dan mendapatkan hasil keseluruhannya berstatus "SUCCESS". Sehingga dapat disimpulkan bahwa CI/CD pada web service i-Lab berjalan dengan baik.

Referensi

[1] A. Farid and I. Gita Anugrah, "Implementasi CI/CD Pipeline Pada Framework Androbase Menggunakan Jenkins (Studi Kasus: PT. Andromedia)," *J. Nas. Komputasi dan Teknol. Inf.*, vol. 4, no. 6, 2021.

[2] S. Farida Utam, S. Rheno Widianto, and W. Al Mauludyansah, "Implementasi DevOps pada Pengembangan Aplikasi e-Skrining Covid-19," 2020.

[3] Alpery Andrian and Fadhly Ridha Muhammad Arif, "Implementasi Ci/Cd Dalam Pengembangan Aplikasi Web Menggunakan Docker dan Jenkins," *9th Appl. Bus. Eng. Conf.*, 2021.

[4] R. Agung Parama, Studiawan Hudan, and Januar Akbar Rizky, "Implementasi Continuous Integration danContinuous Delivery Pada Aplikasi myITSSingle Sign On," *J. Tek. ITS*, vol. 11 No. 3, 2022.

[5] A. W. Favourite, "Implementasi Metode Prototype dalam Pengembangan Aplikasi Mobile Hybrid pada Aplikasi iLab Mobile Berbasis Framework Flutter," 2022.

[6] G. Arsya and P. Zaman, "Perancangan dan Implementasi Web Service Sebagai Media Pertukaran Data Pada Aplikasi Permainan," 2017.

[7] A. Mahandis Shama and D. W. Chandra, "Implementasi Static Application Security Implementasi Static Application Security Testing Menggunakan Jenkins CI/CD Berbasis Docker Container Pada PT. Emporia Digital Raya," 2021.

[8] R. A. Putra, "Analisa Implementasi Arsitektur Microservices Berbasis Kontainer Pada Komunitas Pengembang Perangkat Lunak Sumber Terbuka (Opendaylight Devops Community)," Teknologi Informasi dan Komputer. [Online]. Available: https://jurnal.umj.ac.id

[9] F. Apriliansyah, I. Fitri, A. Iskandar, and R. Artikel, "Implementasi Load Balancing Pada Web Server Menggunakan Nginx," *J. Teknol. dan Manaj. Inform.*, vol. 6, no. 1, 2020, [Online]. Available: http://http//jurnal.unmer.ac.id/index.php/jtmi

[10] N. S. Aji and A. L. Dwi, "Implementasi Continuous Integration/Continuous Delivery (CI/CD) Pada Performance Testing Devops."

[11] R. A. Megantara, F. Alzami, R. A. Pramunendar, and D. P. Prabowo, "Pengembangan dan Implementasi Docker Untuk Memaksimalkan Utilitas Server Universitas Pada Masa Covid-19," *Transmisi*, vol. 24, no. 2, pp. 48–54, May 2022, doi: 10.14710/transmisi.24.2.48-54.