TESIS

PIPELINE DEPLOYMENT OTOMATIS UNTUK APLIKASI BERBASIS KONTAINER MENGGUNAKAN JENKINS DAN DOCKER PADA WEB SERVICES UNTUK ALAT IOT

Disusun oleh:

Nama : **Yogi Yullanto**
NIM : **21.52.1053**
Konsentrasi : **Business Intelligence**

PROGRAM STUDI S2 TEKNIK INFORMATIKA
PROGRAM PASCASARJANA UNIVERSITAS AMIKOM YOGYAKARTA
YOGYAKARTA
2023

TESIS

PIPELINE DEPLOYMENT OTOMATIS UNTUK APLIKASI BERBASIS KONTAINER MENGGUNAKAN JENKINS DAN DOCKER PADA WEB SERVICES UNTUK ALAT IOT

AUTOMATED DEPLOYMENT PIPELINE FOR CONTAINERIZED APPLICATIONS USING JENKINS AND DOCKER ON WEB SERVICES OF IOT DEVICE

Diajukan melalui Jalur Lomba dengan Publikasi
untuk memenuhi salah satu syarat memperoleh derajat Magister

Disusun oleh:

Nama : **Yogi Yulianto**
NIM : **21.52.1053**
Konsentrasi : **Business Intelligence**

PROGRAM STUDI S2 TEKNIK INFORMATIKA
PROGRAM PASCASARJANA UNIVERSITAS AMIKOM YOGYAKARTA
YOGYAKARTA
2023

HALAMAN PENGESAHAN

PIPELINE DEPLOYMENT OTOMATIS UNTUK APLIKASI BERBASIS KONTAINER MENGGUNAKAN JENKINS DAN DOCKER PADA WEB SERVICES UNTUK ALAT IOT

AUTOMATED DEPLOYMENT PIPELINE FOR CONTAINERIZED APPLICATIONS USING JENKINS AND DOCKER ON WEB SERVICES OF IOT DEVICE

Dipersiapkan dan Disusun oleh

Yogi Yulianto

21.52.1053

Telah Diujikan dan Dipertahankan dalam Sidang Ujian Tesis
Program Studi S2 Teknik Informatika
Program Pascasarjana Universitas AMIKOM Yogyakarta
pada hari Rabu, 1 Maret 2023

Tesis ini telah diterima sebagai salah satu persyaratan
untuk memperoleh gelar Magister Komputer

Yogyakarta, 1 Maret 2023
Rektor

Prof. Dr. M. Suyanto, M.M.
NIK. 190302001

HALAMAN PERSETUJUAN

PIPELINE DEPLOYMENT OTOMATIS UNTUK APLIKASI BERBASIS KONTAINER MENGGUNAKAN JENKINS DAN DOCKER PADA WEB SERVICES UNTUK ALAT IOT

AUTOMATED DEPLOYMENT PIPELINE FOR CONTAINERIZED APPLICATIONS USING JENKINS AND DOCKER ON WEB SERVICES OF IOT DEVICE

Dipersiapkan dan Disusun oleh

Yogi Yulianto

21.52.1053

Telah Diujikan dan Dipertahankan dalam Sidang Ujian Tesis
Program Studi S2 Teknik Informatika
Program Pascasarjana Universitas AMIKOM Yogyakarta
pada hari Rabu, 1 Maret 2023

Pembimbing Utama

Prof. Dr. Ema Utami S.Si, M.Kom.
NIK. 190302037

Anggota Tim Penguji

Alva Hendi Muhammad, M.Eng, Ph.D
NIK. 0518078401

Pembimbing Pendamping

Dhani Ariatmanto, M.Kom, Ph.D.
NIK. 190302197

Kusnawi, S.Kom., M.Eng.
NIK. 190302112

Prof. Dr. Ema Utami S.Si, M.Kom.
NIK. 190302037

Tesis ini telah diterima sebagai salah satu persyaratan
untuk memperoleh gelar Magister Komputer

Yogyakarta, 1 Maret 2023
Direktur Program Pascasarjana

Prof. Dr. Kusrini, M.Kom.
NIK. 19030210

HALAMAN PERNYATAAN KEASLIAN TESIS

Yang bertandatangan di bawah ini,

Nama mahasiswa : Yogi Yulianto
NIM : 21.52.1053
Konsentrasi : Business Intelligence

Menyatakan bahwa Tesis dengan judul berikut:
Pipeline Deployment Otomatis untuk Aplikasi Berbasis Kontainer Menggunakan Jenkins dan Docker Pada Web Services untuk Alat IOT

Dosen Pembimbing Utama : Prof. Dr. Emma Utami S.Si, M.Kom
Dosen Pembimbing Pendamping : Kusnawi, S.Kom , M.Eng

1. Karya tulis ini adalah benar-benar ASLI dan BELUM PERNAH diajukan untuk mendapatkan gelar akademik, baik di Universitas AMIKOM Yogyakarta maupun di Perguruan Tinggi lainnya
2. Karya tulis ini merupakan gagasan, rumusan dan penelitian SAYA sendiri, tanpa bantuan pihak lain kecuali arahan dari Tim Dosen Pembimbing
3. Dalam karya tulis ini tidak terdapat karya atau pendapat orang lain, kecuali secara tertulis dengan jelas dicantumkan sebagai acuan dalam naskah dengan disebutkan nama pengarang dan disebutkan dalam Daftar Pustaka pada karya tulis ini
4. Perangkat lunak yang digunakan dalam penelitian ini sepenuhnya menjadi tanggung jawab SAYA, bukan tanggung jawab Universitas AMIKOM Yogyakarta
5. Pernyataan ini SAYA buat dengan sesungguhnya, apabila di kemudian hari terdapat penyimpangan dan ketidakbenaran dalam pernyataan ini, maka SAYA bersedia menerima SANKSI AKADEMIK dengan pencabutan gelar yang sudah diperoleh, serta sanksi lainnya sesuai dengan norma yang berlaku di Perguruan Tinggi

Yogyakarta, 1 Maret 2023
Yang Menyatakan,

Yogi Yulianto

HALAMAN PERSEMBAHAN

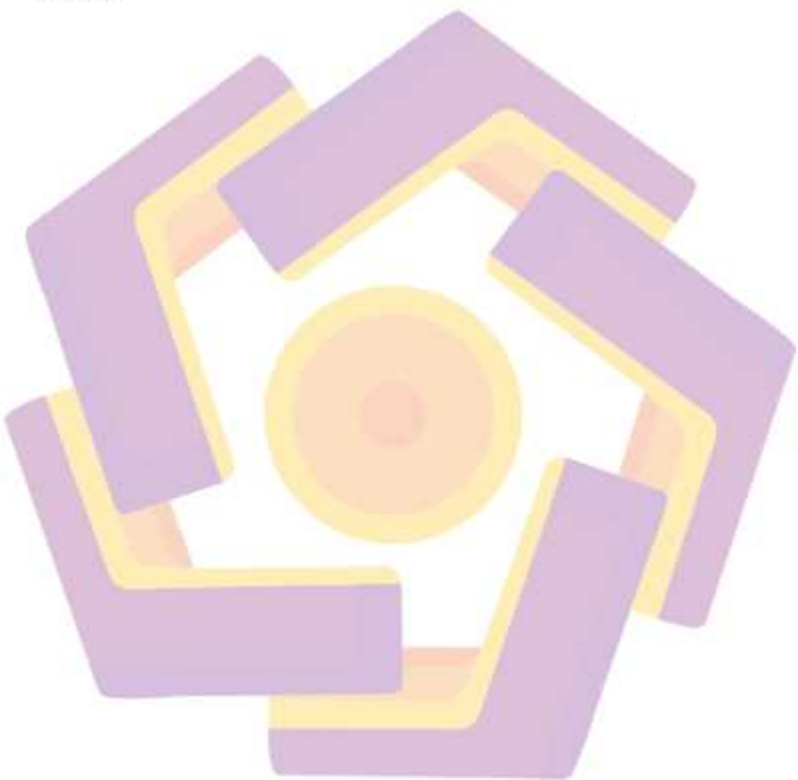
Dengan segala kerendahan hati, saya ingin menyampaikan persembahan tesis ini kepada orang tua tercinta, istri (Zari Rafida) yang selalu memberikan dukungan dan cinta sejati, dan adik (Eri Setianingsih) dan kakak (Puji Handayani Putri) dan juga Bapak (Kuat Supriyadi) mertua dan Ibu (Siti Zubaidah) mertua saya yang telah memberikan semangat, dorongan, dan doa dalam perjalanan penyelesaian tesis ini.

Bapak, dan Ibu, terima kasih atas doa dan dukungan yang tak pernah henti diberikan selama perjalanan hidup saya. Dan tak lupa terimakasih juga kepada Bapak dan Ibu Mertua yang telah mendukung dan mendorong saya untuk bisa menyelesaikan tesis ini. Segala hal yang telah kalian lakukan, mulai dari memberikan semangat, mengorbankan waktu dan tenaga, hingga memberikan saran-saran berharga, tidak akan pernah saya lupakan. Kepada istri tercinta, terima kasih atas cinta dan dukungan yang kau berikan selama proses penulisan tesis ini. Kau selalu menjadi motivator dan sumber kebahagiaan dalam hidup saya, dan tesis ini adalah bukti bahwa kita mampu mengatasi segala tantangan bersama-sama.

Serta untuk adik dan kakakku, terima kasih atas semangat dan dukungan kalian dalam menghadapi setiap perjuangan dan rintangan. Kalian adalah sosok-sosok inspiratif yang selalu mengingatkan saya untuk terus belajar dan berkembang. Saya berharap persembahan ini dapat menjadi wujud penghargaan dan rasa terima kasih saya kepada kalian.

HALAMAN MOTTO

Menuntut ilmu adalah takwa. Menyampaikan ilmu adalah ibadah. Mengulang-ulang ilmu adalah zikir. Mencari ilmu adalah jihad. (Abu Hamid Al Ghazali)



KATA PENGANTAR

Dengan mengucapkan Alhamdulillah segala puji dan syukur penulis panjatkan atas kehadirat Allah SWT, karena berkat rahmat dan hidayah-Nya penyusunan tesis yang berjudul "Pipeline Deployment Otomatis untuk Aplikasi Berbasis Kontainer Menggunakan Jenkins dan Docker Pada Web Services untuk Alat IOT" ini dapat diselesaikan guna memenuhi salah satu persyaratan dalam menyelesaikan pendidikan pada Program Studi S2 Teknik Informatika Universitas Amikom Yogyakarta. Banyak hambatan yang dihadapi dalam penyusunannya, namun berkat kehendak-Nyalah sehingga penulis berhasil menyelesaikan penulisan tesis ini. Oleh karena itu pada kesempatan ini patutlah kiranya penulis mengucapkan terima kasih kepada :

1. Prof. Dr. Emma Utami S.Si, M.Kom. Selaku Pembimbing I dan Kusnawi, S.Kom., M.Eng Selaku Pembimbing II yang telah membimbing penulis dalam penyusunan tesis ini hingga selesai
2. Tim Fishee yang telah bersama-sama berjuang pada kompetisi ITEX Malaysia dan Archimedes Russia
3. Semua pihak yang telah banyak membantu dalam penyusunan skripsi ini yang tidak bisa penulis sebutkan semuanya.

Akhir kata, penulis mengharapkan tesis ini dapat memberikan manfaat bagi penulis khususnya dan bagi pembaca pada umumnya.

Yogyakarta, 20 Februari 2023

Penulis

BAB I

DESKRIPSI KARYA LOMBA

1.1. Uraian Tentang Karya

Fishee adalah Alat monitoring pakan ikan otomatis berbasis *internet of things*, fishee dapat memberi pakan ikan secara otonom atau dengan kata lain tanpa campur tangan manusia. Dan juga dapat juga dikendalikan secara manual oleh petani melalui bot telegram dan juga website yang telah terhubung dengan alat. Alat monitoring pakan ikan otomatis berbasis *internet of things* ini dibekali sensor suhu dan sensor pH guna memeriksa kualitas air pada kolam.

1.2. Latar Belakang Pengembangan

Ikan lele (*Clarias Sp.*) merupakan salah satu ikan air tawar dengan komoditas tinggi yang ekonomis selain ikan nila, gurame dan ikan mas. Menurut KKP (Kementerian Kelautan dan Perikanan), data jumlah ikan lele di Indonesia pada tahun 2020 sebesar 347.511 ton, sedangkan untuk capaian target konsumsi ikan tahun 2020 mencapai 56,39 kg/kapita/tahun (https://kkp.go.id). Hal ini dapat menjadi prospek potensial untuk dikembangkan di Indonesia.

Budidaya ikan lele merupakan salah satu prospek di bidang perikanan yang cukup potensial dimana terbagi menjadi dua kegiatan besar yaitu proses pembibitan dan pembesaran. Kegiatan tersebut tidak dapat dipisahkan karena saling berkaitan satu sama lain. Membahas kegiatan budidaya lele sebelum proses pembesaran tentu harus melewati proses pembibitan yang juga terdiri dari lima tahap, diantaranya

pemijahan, larva (benih), benih menuju fase dewasa, benih fase dewasa, dan benih fase tua. Dalam proses pembibitan diperlukan ketelitian yang tinggi karena rentan akan tingginya angka kematian akibat stress atau kurang diperhatikannya aspek-aspek pada proses pembibitan seperti temperatur yang naik turun, kadar PH air dan cuaca yang berubah-ubah sehingga menyebabkan bibit lele menjadi stress. Sebuah sistem yang dapat mengontrol pengoptimalan kualitas media dan air budidaya pembibitan lele guna mengurangi angka kematian dan meningkatkan alat bibit lele akan diperlukan.

Selain ikan lele, ikan air tawar yang mempunyai potensi dan prospek yang cukup baik adalah Ikan nila. Ikan nila atau (*Oreochromis niloticus*) merupakan ikan air tawar yang mudah dipelihara dan gangguan penyakitnya tidak begitu banyak. Pembibitan nila cukup mudah. Dari sepasang indukan bisa dihasilkan 250-1000 butir telur. Waktu persiapan dari telur hingga menjadi benih berukuran 5-8 cm diperlukan waktu 60 hari. Nila merupakan jenis ikan air tawar yang pertumbuhannya cepat. Jenis nila unggul pertumbuhannya bisa mencapai 4,1 gram per hari. Pertumbuhan ikan jantan lebih pesat dibanding ikan betina. Dibutuhkan waktu 4-6 bulan untuk membesarkan ikan nila hingga ukuran siap konsumsi. Ikan nila juga membutuhkan kontrol media air yang digunakan seperti temperature, kadar pH air, dan cuaca sesuai dengan SNI 7550:2009.

Dari latar belakang masalah diatas maka direncanakan untuk membuat sebuah alat sensor yang mampu mengontrol pengoptimalan kualitas media dan air berdasarkan temperatur ideal, prediksi cuaca, kadar PH ideal bagi ikan lele dan ikan nila (menurut SNI atau Standar Internasional Indonesia 6484.4:2014, SNI

7550:2009) secara otomatis, yang kemudian ketika beberapa dari aspek diatas tidak sesuai dengan kadar ideal, sensor akan memberikan notifikasi kepada pembudidaya melalui *smartphone* untuk ditindak lanjuti. Melalui teknologi ini memudahkan pembudidaya dalam mengontrol kualitas media dan air secara intens tanpa harus terjaga semalaman. Penelitian ini bertujuan untuk menciptakan alat sensor otomatis dalam pengontrolan kualitas media dan air yang optimal sehingga meningkatkan produksi lele dan nila maupun ikan air tawar lain dalam skala yang besar.

1.3. Keunikan dan Value

Fishee mempunyai 4 fitur utama yaitu fishee *automatic feeder,* fishee *care,* fishee *growth,* dan *fishee SNI template*. Selain untuk pakan otomatis dan *monitoring* kolam, fishee mempunyai fitur SNI *template* yang memungkinkan petani menggunakan sistem untuk beberapa jenis ikan air tawar selain ikan lele, yaitu ikan nila dan gurami sesuai dengan SNI yang telah di konfigurasi oleh tim pengembang fishee. Kemudian keunikan dari sistem fishee adalah petani dapat memonitoring kolam hanya melalui *smartphone* yang dapat diakses baik melalui website dan juga bot telegram. Fishee memiliki beberapa competitor baik di dalam negeri maupun di luar negeri seperti yang dilampirkan pada tabel 1.1.

Tabel 1.1 Produk Sejenis

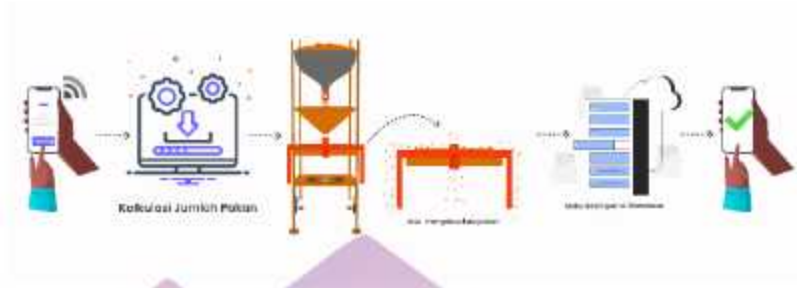
No.	Deskripsi	Produk Sejenis				Fishee
		Indonesia		Thailand	USA	
		e-Fishery	Fishgator	Minnowtech	Aquafeed	
1	Dapat digunakan pada berbagai jenis ikan	-	-	-	-	✓
2	Dukungan sensor					
	Suhu	✓	✓	✓	✓	✓
	Level Ph	✓	✓	✓	✓	✓
3	Pakan Otomatis					
	Pengaturan secara manual	✓	✓	✓	✓	✓
	Pengaturan otomatis menurut Standar Nasional Indonesia (SNI)	-	-	-	-	✓
5	Instalasi mudah	✓	✓	✓	✓	✓
6	Pengaturan jumlah pakan	-	-	-	-	✓
7	Alarm ketika internet buruk	-	-	✓	-	✓
8	Bisa digunakan melalui website dan bot telegram	-	-	-	-	✓

1.4. Fungsi, Fitur, dan Kegunaan

Fishee merupakan alat monitoring ikan berbasis *internet of things* dan dapat dikendalikan melalui website dan bot telegram. Fishee memiliki 4 fitur diantaranya yaitu

1.4.1. Fishee *Automatic Feeder*

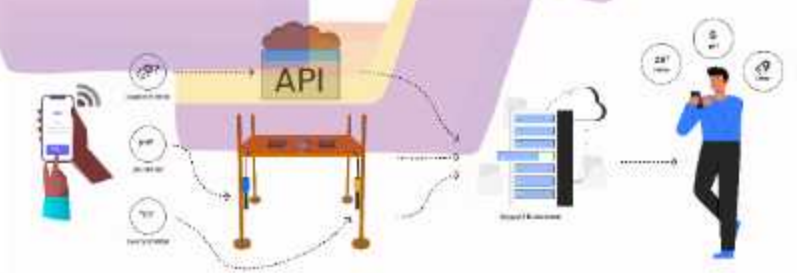
Fishee *Automatic Feeder* merupakan fitur yang memungkinkan petani untuk memberi pakan ikan lele secara otomatis maupun manual, Fitur ini dapat di atur secara manual petani dan juga bisa mengikuti *template* pemberian pakan ikan sesuai Standar Nasional Indonesia (SNI) yang telah disediakan oleh sistem Fishee.



Gambar 1.1 Alur Kerja Fishee *Automatic Feeder*

1.4.2. Fishee *Care*

Fish *Care* merupakan fitur yang bisa memantau kondisi lingkungan (kolam) bagi ikan lele. Fitur ini dapat mengecek level pH menggunakan Sensor Analog pH Meter kit. Selain itu terdapat juga sensor Suhu yang dapat mengecek tingkat suhu air dalam kolam ikan yang nantinya bisa digunakan oleh petani untuk mengambil keputusan apakah kualitas air kolam masih layak untuk ikan atau tidak. Sistem Fishee juga terintegrasi dengan API (*Application programmable Interface*) Cuaca dari BMKG (Badan Meteorologi, Klimatologi, dan Geofisika) Indonesia yang dapat memprediksi cuaca dari wilayah kolam dari petani.



Gambar 1.2 Alur Kerja Fishee *Care*

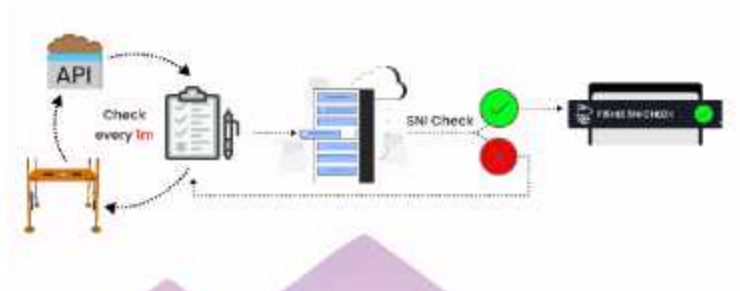
1.4.3. Fishee *Growth*

Fish *Growth* yang dapat digunakan untuk memonitoring perkembangan ikan setiap harinya sehingga dapat membantu petani dalam penentuan pakan dan panennya dalam satu siklus. Fitur ini juga dapat melihat riwayat pertumbuhan ikan pada siklus-siklus sebelumnya.

Gambar 1.3 Alur Kerja *Fishee Growth*

1.4.4. Fishee SNI *Template*

Fish SNI Template merupakan fitur yang disediakan oleh fishee untuk memonitoring apakah kondisi didalam kolam sesuai dengan Standar Nasional Indonesia atau belum. Jika belum, maka sistem akan memberikan notifikasi kepada petani bahwa kolam belum sesuai standar dan meminta petani untuk menindak lanjuti.



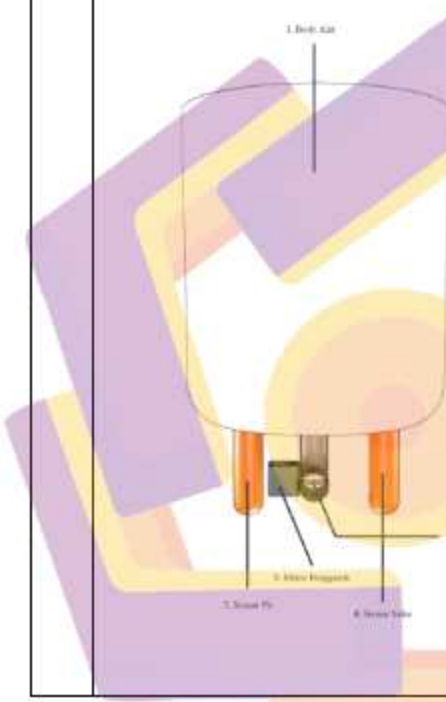
Gambar 1.4 Fishee SNI *Template*

1.5. Inovasi, dan Implementasi atau Potensi

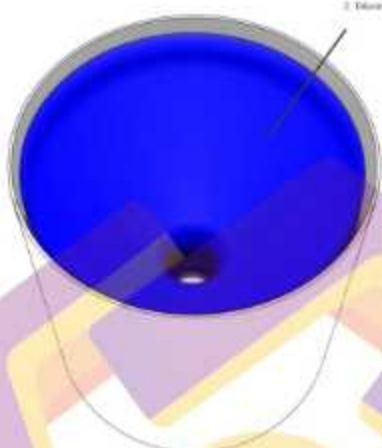
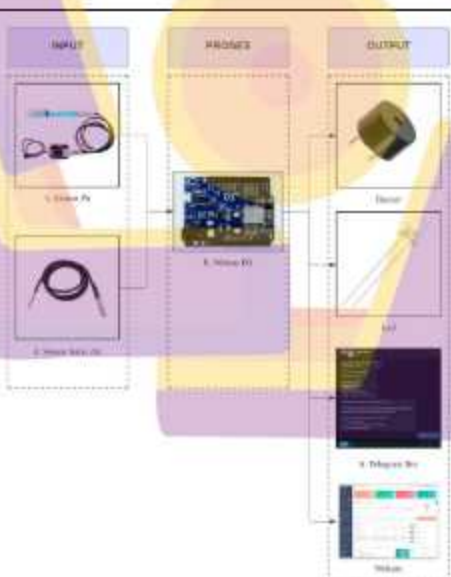
Fishee merupakan *Digital Fishery Platform* berbasiskan website dan bot telegram yang mana memiliki prinsip 3P (Pengecekan, Pemantauan & Pengelolaan) kolam ikan secara digital berbasis *Internet Of Things (IoT)* serta visualisasi Dashboard dalam bentuk website sehingga petani dapat memonitoring perkembangan produksi ikan secara *realtime* dan online. Saat ini alat Fishee sedang dikembangkan dan diuji coba pada salah satu tambak ikan lele di Kabupaten Kulonprogo, Daerah Istimewa Yogyakarta.

1.6. Screenshot

Tabel 1.2 *Screenshoot*

No.	Screenshot	Keterangan
1.		Tampilan tampak depan *prototype* alat Fishee

Tabel 1.2 *Screenshoot* (Lanjutan)

2.		Tampilan tampak atas *prototype* alat fishee
3.		Alur kerja *prototype* alat fishee.

Tabel 1.2 *Screenshoot* (Lanjutan)

4.		Tampilan utuh *prototype* alat fishee

BAB II

PUBLIKASI

2.1. Abstrak

Di era modern ini, banyak perusahaan di bidang teknologi bersaing untuk memenangkan hati pelanggan. Untuk menjaga persaingan dan memenuhi kebutuhan pelanggan, perusahaan harus mampu beradaptasi dengan perubahan dalam produk mereka. Salah satu metodologi pengembangan perangkat lunak yang berfokus pada kepuasan pelanggan adalah metodologi *agile*. Pendekatan ini populer di kalangan tim pengembangan perangkat lunak karena memungkinkan mereka untuk dengan cepat menanggapi perubahan dalam kebutuhan pelanggan. Penelitian ini bertujuan untuk menerapkan kombinasi Gitlab CI dan Jenkins untuk menerapkan pipeline CI/CD pada dua aplikasi perangkat lunak yang disebut Fishee Backend App dan Fishee Frontend App, yang menggunakan teknologi yang berbeda juga menggunakan pendekatan metodologi agile. Penelitian ini bertujuan untuk mengevaluasi kualitas *pipeline* dan waktu yang dibutuhkan untuk memberikan perubahan kepada pelanggan. Hasilnya menunjukkan bahwa Fishee Backend App membutuhkan waktu 153,7 detik dan Fishee Frontend App membutuhkan 174,2 detik. Dalam hal pekerjaan *pipeline* yang sukses, Fishee Backend App mencapai 70,45% dan Fishee Frontend App mencapai 83,3%. Berdasarkan temuan ini, penggunaan kombinasi Jenkins dan Gitlab CI dalam proses CI/CD dapat efektif dan lebih cepat daripada penelitian sebelumnya untuk mengotomatisasi *deployment* aplikasi berbasis kontainer. Namun, penelitian ini

terbatas pada implementasi di platform Digital Ocean dan tidak dibandingkan dengan platform penyedia VPS lainnya. Disarankan agar penelitian masa depan memperluas ruang lingkup studi untuk mencakup tumpukan teknologi lain dan membandingkan kinerja berbagai penyedia CI/CD pada berbagai platform penyedia VPS.

2.2. Introduction

Di era modern ini, banyak perusahaan harus dapat beradaptasi untuk mengubah produk mereka mengikuti kebutuhan pelanggan. Untuk menjamin keberhasilan proyek pengembangan perangkat lunak, metodologi pengembangan perangkat lunak adalah kunci utama untuk menang dalam persaingan. Beberapa parameter yang dapat mengukur keberhasilan pengiriman aplikasi ke pelanggan adalah akurasi yang baik, efektivitas biaya, dan pengiriman yang cepat . Metode *waterfall* tradisional memiliki beberapa masalah, seperti tidak ada pemeriksaan kesalahan selama proses pengembangan, sangat terorientasi pada dokumen, membutuhkan banyak waktu untuk perencanaan karena proses dapat kembali ke proses sebelumnya, dan perlu beradaptasi dengan perubahan . Seiring waktu, metodologi *agile* diperkenalkan sebagai metodologi pengembangan perangkat lunak yang dapat beradaptasi dengan perubahan. *Agile* memprioritaskan kebutuhan pelanggan sebagai kunci utama . Dalam siklus *agile*, ada beberapa tahap pengembangan, mulai dari perencanaan, desain, implementasi, kode, penyebaran, dan umpan balik. Dalam sebuah organisasi, ada orang yang mengumpulkan dan mengambil kode untuk didistribusikan ke server. Tanpa konsep *DevOps*, tim operasi akan secara manual mengumpulkan dan mendistribusikan kode ke server

untuk memberikan perubahan kepada pengguna. Namun, hal itu menciptakan masalah baru. Jika tim operasi tidak ada di perusahaan mereka dan klien membutuhkan kode yang didistribusikan dengan cepat, maka itu akan menyebabkan penundaan dalam proses pengiriman. *DevOps* adalah cara bagi sebuah organisasi untuk mengotomatisasi proses pengiriman perangkat lunak yang kontinyu, menjamin kualitas dan kehandalan. Untuk mengimplementasikan *DevOps* pada lingkungan pengembangan perangkat lunak Fishee, kami akan mengimplementasikan metode pengiriman otomatis dengan pendekatan metodologi agile yang disebut *Continuous Integration* and *Continuous Delivery* (CI/CD) untuk mengurangi pekerjaan manual. Studi sebelumnya telah menunjukkan bahwa pipeline CI/CD secara efisien meningkatkan kecepatan metodologi *agile* pada tahap produksi aplikasi *super app* seperti Facebook, GitHub, Rally Soft, dan Netflix. Proses otomatisasi ini ditangani oleh perangkat lunak pihak ketiga seperti Jenkins, Gitlab CI, GitHub Action, Travis CI, dan perangkat lunak CI lainnya. CI/CD seperti jembatan antara tim operasional dan pengembangan perangkat lunak dalam proses otomatisasi, seperti membangun *image* docker, pengujian, dan *deployment*. Selain itu, CI/CD dapat memantau proses pengiriman sehingga pengembang dapat memperbarui perangkat lunak dengan cepat, tepat, dan terorganisir. Docker adalah aplikasi berbasis kontainer dan open source yang memungkinkan pengembang perangkat lunak untuk membuat, menjalankan, membangun, menguji, dan merilis aplikasi dalam kontainer terisolasi.

Docker memudahkan proses *packaging* aplikasi dan komponennya dengan cepat di dalam container yang terisolasi sehingga dapat dijalankan di infrastruktur

lokal tanpa melakukan perubahan konfigurasi pada container. Penelitian sebelumnya menunjukkan bahwa container Docker memiliki performa yang lebih baik pada mesin virtual (VM) dalam hal penggunaan CPU, penggunaan memori, kecepatan penyimpanan baca/tulis, dan pengujian beban. Karena itu, penggunaan Docker dalam metodologi agile dianggap sebagai pilihan yang lebih baik daripada menggunakan VM.

Dalam penelitian ini, kami akan mengimplementasikan *pipeline* otomatisasi *deployment* untuk aplikasi berbasis container dari perangkat IoT yang disebut Fishee. Fishee adalah perangkat untuk pemberian makan ikan otomatis dan pemantauan berbasis *Internet of Things*. Namun, penelitian ini akan membahas CI/CD pada Aplikasi Frontend dan Backend Fishee, bukan pada aplikasi yang ada di dalam sistem *embed* di dalam alat Fishee.

Aplikasi Fishee akan berjalan di dalam kontainer Docker. Proses CI/CD akan di-*handle* oleh kombinasi Jenkins dan GitLab CI. Jenkins adalah perangkat lunak otomatisasi sumber terbuka yang dikembangkan menggunakan bahasa pemrograman Java dengan dukungan komunitas dan banyak plugin [12], membuat Jenkins menjadi salah satu alat CI/CD yang paling populer untuk mengimplementasikan *Continuous Integration* dan *Continuous Delivery*. GitLab CI juga merupakan salah satu alat CI/CD yang mudah digunakan karena semuanya dalam *pipeline* GitLab akan diatur dalam satu file YML. GitLab juga memiliki fitur pemantauan performa, yang menjadi keunggulan dibandingkan dengan Jenkins.

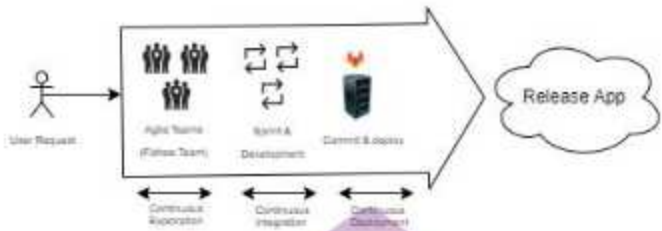
Penelitian ini berfokus pada proses CI/CD untuk aplikasi berbasis container menggunakan kombinasi Jenkins dan GitLab CI. Aplikasi yang digunakan dalam

penelitian adalah aplikasi Backend Fishee dan aplikasi Frontend Fishee, yang memiliki teknologi stack yang berbeda (PHP dan Node JS). Penelitian ini akan menganalisis waktu yang dibutuhkan dan tingkat keberhasilan dari proses *deployment* otomatis untuk kedua aplikasi tersebut.

2.3. Studi Literature

2.3.1. Otomatisasi pada Pengembangan Perangkat Lunak *Agile*

Dalam pengembangan perangkat lunak dengan pendekatan *Agile*, semua pengembangan, termasuk proses analisis, desain, pemrograman, dan pengujian, merupakan aktivitas yang iteratif. Manajer proyek dalam metode *Agile* biasanya disebut sebagai *scrum master*, yang memastikan bahwa semua aturan yang ada diterapkan dengan benar . Dalam pengembangan perangkat lunak *Agile*, proses pengembangan adalah iteratif. Perubahan relatif kecil dan akan terus berkembang. Proses *Agile* dibagi menjadi perencanaan, desain, pemrograman, implementasi, pengujian, dan evaluasi. Alur kerja *Agile* dalam tim Fishee, seperti yang ditunjukkan pada Gambar 2.1, dimulai dari menerima permintaan pelanggan, kemudian tim akan membahas permintaan tersebut. Setelah itu, sprint dan pengembangan akan dijalankan oleh tim, lalu perubahan akan dicommit ke repositori dan diimplementasikan ke server agar pelanggan dapat menggunakan perubahan pada aplikasi. Proses ini akan iteratif, dan jika proses ini ditangani secara manual oleh orang yang menyebabkan beberapa masalah, seperti implementasi yang salah dan konflik saat implementasi, maka diperlukan proses otomatisasi untuk mengurangi beban kerja tim.

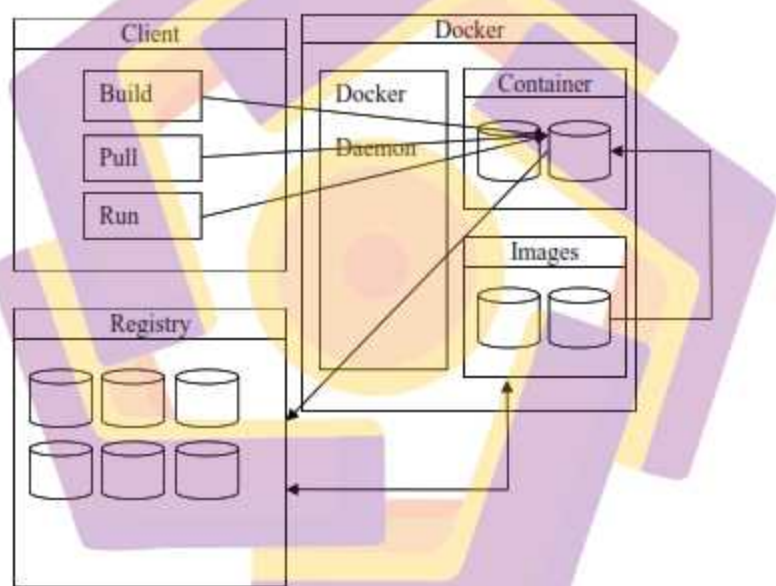


Gambar 2.1 Alur *agile* pada tim fishee

2.3.2. Aplikasi berbasis Kontainer

Sebuah container adalah lingkungan runtime yang lengkap di mana sebuah aplikasi sudah ada di dalamnya, bersama dengan dependensi, pustaka, dan konfigurasi yang diperlukan untuk menjalankan aplikasi. Dengan membuat aplikasi menjadi kontainer, lebih mudah bagi pengembang untuk men-*deploy* baik aplikasi maupun infrastruktur secara bersamaan. Salah satu aplikasi yang memudahkan pengembang perangkat lunak untuk memaketkan, membangun, menguji, dan men-*deploy* aplikasi yang siap digunakan adalah Docker. Docker adalah platform terbuka untuk mengembangkan, memindahkan, dan menjalankan aplikasi. Docker menggunakan *namespace* untuk mengisolasi ruang kerja dari sebuah *container*. Untuk *container* tertentu, Docker membuat *namespace* sehingga *container* lain tidak dapat mengakses container tersebut. Isolasi ini bertujuan untuk memastikan bahwa setiap proses terpisah dan tidak memungkinkan untuk mencari sumber daya di grup atau *container* lain. Salah satu keuntungan dari menggunakan Docker adalah bahwa aplikasi dapat di-*scaling* dengan mudah hanya dengan menambah atau menghapus sebuah container. Docker hampir mirip dengan mesin virtual. Perbedaannya adalah bahwa Docker jauh lebih ringan. Selain itu, Docker dapat

dibuat dengan sangat mudah dan cepat sehingga proses *deployment* menjadi sangat cepat. Karena proses isolasi dan keamanan Docker, pengembang dapat menjalankan beberapa *runtime* dengan beberapa container pada satu mesin. Docker menggunakan arsitektur *client-server*. Gambar 2.2 menggambarkan alur kerja Docker dan bagaimana klien Docker terkait dengan *daemon* Docker, yang dapat menarik *image* dari registry. Docker akan mengeksekusi apa yang tertulis di dalam Dockerfile.



Gambar 2.2 Diagram alur docker

2.3.3. *Continuous Integration* dan *Continuous Deployment*

Ketika sebuah organisasi ingin menerapkan *pipeline* CI/CD, hal pertama yang harus dilakukan adalah mengimplementasikan proses CI terlebih dahulu karena CD tidak dapat dimulai sebelum CI diimplementasikan. Dari *Continuous Integration*

menuju *Continuous Deployment*, proses otomatisasi ini dapat mengurangi proses yang sebelumnya ditangani oleh tim operasi secara manual dan menggantinya dengan perangkat lunak secara otomatis. CI atau *Continuous Integration* adalah metode dalam pengembangan perangkat lunak yang memungkinkan pengembang untuk mengintegrasikan pekerjaan mereka, seperti membangun dan menguji secara otomatis. Hal ini dapat membuat perangkat lunak lebih cepat diterima oleh pengguna dan membantu menemukan bug serta memperbaikinya lebih cepat. dibandingkan dengan *containerization* dan *Virtual Machine* (VM) yang berbasis pada pengembangan aplikasi *microservices*, teknologi berbasis *container* memiliki skalabilitas yang tinggi, portabilitas yang baik, dan performa yang lebih baik dalam implementasi dibandingkan dengan implementasi berbasis VM. *Continuous deployment* berarti membuat perubahan yang dilakukan di repositori dapat diimplementasikan pada tahap produksi. Menurut penelitian sebelumnya, salah satu komponen dalam variasi pengiriman yang berkesinambungan adalah pengiriman kode terus-menerus yang dikirimkan ke repositori haruslah kode yang dapat diterapkan di lingkungan produksi. Setelah melalui beberapa tahapan, seperti *build* dan pengujian otomatis, perangkat lunak akan di-*deploy* ke lingkungan produksi dengan hanya menekan tombol. Perbedaan utama antara CI dan CD adalah bahwa CI berfokus pada memastikan bahwa aplikasi dalam kondisi siap produksi setelah melalui pemeriksaan kualitas, sedangkan CD berfokus pada mengimplementasikan aplikasi ke lingkungan produksi.

2.3.4. *Software* CI/CD

Di era modern ini, banyak perusahaan yang bersaing untuk menerapkan CI/CD dalam pengembangan aplikasi mereka. Begitu banyak *software open source* yang telah dibuat untuk membantu menangani implementasi proses CI/CD seperti Jenkins, Team City, Bamboo, Travis CI, dll. Salah satu pekerjaan yang paling populer di stackoverflow untuk berbagai alat integrasi adalah Jenkins. Jenkins adalah platform CI *Open Source* (OSS) dengan tujuan utama untuk mengotomatisasi proses build dan pengujian. Jenkins dibuat dengan bahasa pemrograman Java, dan banyak *plugin* yang tersedia dapat digunakan. Selain Jenkins yang memiliki banyak *plugin*, dan dukungan dari komunitas yang besar membuat Jenkins fleksibel dan dapat disesuaikan sesuai dengan kebutuhan pengembang aplikasi. Jenkins dapat membuat proses implementasi aplikasi berbasis container melalui Job yang dieksekusi secara otomatis melalui *trigger* seperti *webhook*. Selain Jenkins, alat CI/CD yang cukup populer adalah GitLab CI. Dibandingkan dengan Jenkins, versi berbayar dari GitLab lebih baik karena, dalam penelitian , runner berbayar GitLab lebih cepat daripada Jenkins untuk melakukan CI/CD, dan GitLab memiliki fitur yang memungkinkan pengembang untuk memantau kinerja server mereka.

2.4. Metodologi

Metodologi dalam penelitian ini dibagi menjadi empat tahap. Tahap pertama adalah mendefinisikan arsitektur perangkat lunak, kemudian merumuskan infrastruktur server, dan kemudian merancang dan mengimplementasikan proses *pipeline*

implementasi otomatis dengan Jenkins dan GitLab. Yang terakhir adalah memasuki evaluasi kinerja dari *pipeline* CI/CD.

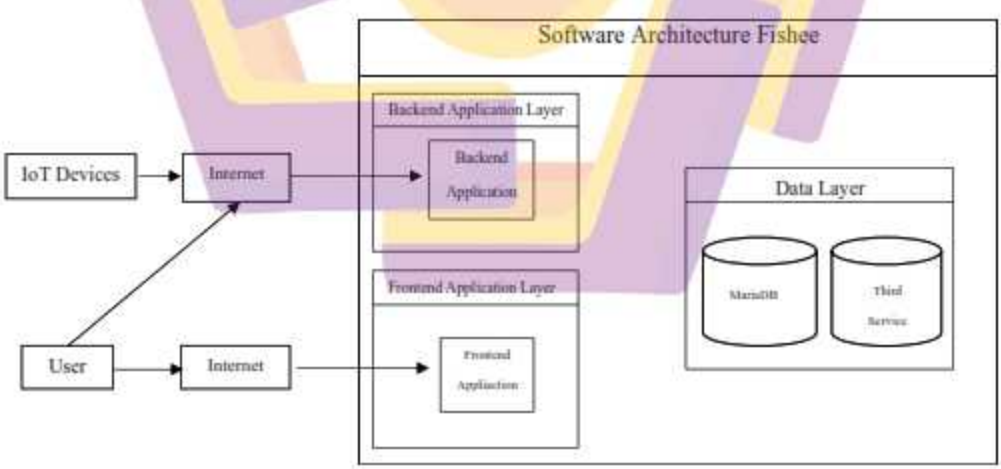
2.4.1. Arsitektur Perangkat Lunak

Perangkat lunak yang akan diterapkan sebagai arsitektur *microservices* yang terdiri dari *frontend* dan *backend*, dibuat dengan teknologi yang berbeda. Implementasi *microservices* sejalan dengan implementasi pengiriman terus-menerus. Arsitektur yang terpisah dengan baik akan mempengaruhi kemampuan membangun dan menguji yang baik. Meskipun ada manfaat dari implementasi pengiriman otomatis, pengembang harus berhati-hati bahwa konfigurasi infrastruktur untuk CI/CD akan membutuhkan banyak usaha. Membagi sebuah aplikasi menjadi *microservices* membutuhkan *pipeline* yang terpisah. Arsitektur aplikasi atau perangkat lunak dapat menjadi hambatan utama, dalam penelitian ini mengubah arsitektur *monolith* menjadi arsitektur *microservices* dapat mengurangi waktu CD dari 30 menit menjadi 3 menit.

Dalam penelitian ini, kami menggunakan arsitektur *microservices* dengan dua aplikasi yang akan diterapkan: Aplikasi Frontend Fishee untuk pelanggan mengakses dasbor mereka dan memesan perangkat IoT, dan Aplikasi Backend Fishee untuk menangani permintaan dari Aplikasi Frontend Fishee. Aplikasi Frontend Fishee dibangun dengan kerangka kerja Next JS, yang telah terbukti dapat meningkatkan kinerja situs web sebesar 14%. Next JS didukung oleh Node JS, yang merupakan salah satu kerangka kerja tercepat dibandingkan dengan aplikasi berbasis PHP dan Python. Fishee Backend adalah REST API yang akan menghubungkan perangkat IoT ke database.

REST API akan dikembangkan oleh Kerangka Kerja Codeigniter, yang dibangun dengan Bahasa Pemrograman PHP. Kerangka kerja Codeigniter adalah kerangka kerja agile dan open-source yang terbukti memberikan konsep pemisahan kode yang baik dan bersih. Codeigniter menerapkan pola desain MVC (Model-View-Controller), pendekatan perangkat lunak yang memisahkan logika aplikasi dan bagian tampilan. Server web yang digunakan adalah Apache yang dapat menjalankan aplikasi PHP di dalamnya. Untuk Database, kami akan menggunakan MariaDB. Aplikasi dibagi menjadi dua lapisan, Lapisan Aplikasi Backend dan Lapisan Data, seperti yang ditunjukkan pada Gambar 2.3.

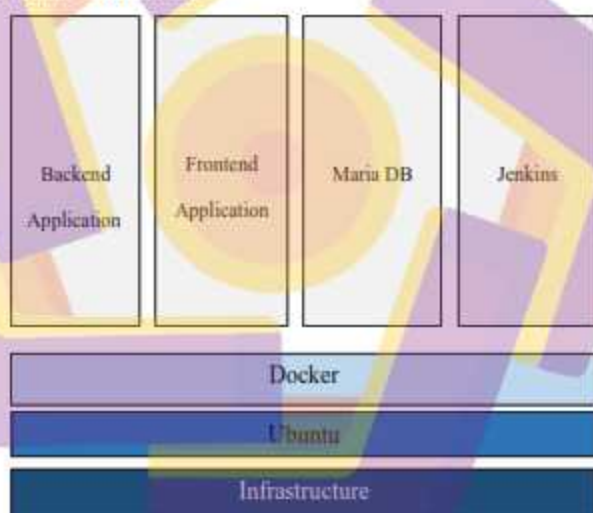
IoT akan mengakses Aplikasi Backend Fishee untuk mengirimkan data ke *Database*, dan setelah itu, REST API akan menyimpannya ke MariaDB. Data yang tersimpan dapat dilihat dalam aplikasi Frontend Fishee yang dibuat oleh Next JS. Aplikasi Frontend Fishee mengonsumsi API dari REST API. Pengguna Fishee



Gambar 2.3 Arsitektur perangkat lunak

2.4.2. Infrastruktur Server

Dalam penelitian ini, kami akan menggunakan Digital Ocean Cloud Droplets VPS dengan 2 CPU, 4 GB RAM, dan 80 GB SSD Disk. Server ini akan digunakan untuk menjalankan empat *container* terisolasi: satu untuk Jenkins, satu untuk aplikasi Fishee Backend, satu untuk aplikasi Fishee Frontend, dan satu untuk database MariaDB. *Container-container* ini akan dibuat menggunakan Dockerfile. Sistem operasi host yang digunakan adalah Ubuntu 18.04, yang menjalankan Docker dan *container-container* yang berisi aplikasi dan *database*. Jenkins diinstal di lapisan teratas dari hierarki.

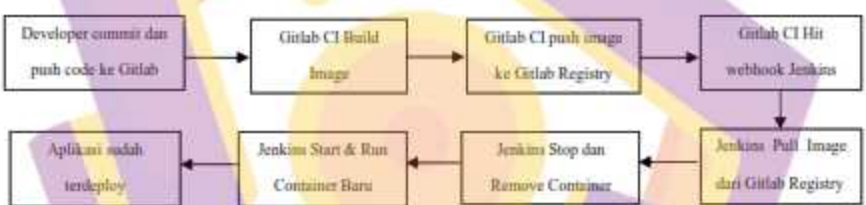


Gambar 2.4 *Architecture server*

2.4.3. *Pipeline Deployment* Otomatis

Proses penyebaran akan dibagi menjadi beberapa tahap, seperti yang dijelaskan pada Gambar 2.5, dimulai dengan pengembang melakukan commit dan push perubahan ke repositori GitLab dengan cabang utama (main) yang akan

memicu GitLab CI untuk dijalankan. Langkah kedua adalah pembuatan *image* docker oleh file docker, yang berisi aplikasi dan web server. Setelah itu, *image* akan didistribusikan ke registry GitLab. Ketika proses pembangunan dan penyebaran di GitLab selesai, Jenkins akan dipicu oleh GitLab CI untuk menjalankan *image Continuous Deployment* (CD) ke server. Pekerjaan yang dilakukan oleh Jenkins memiliki tiga tahap. Tahap pertama adalah menarik *image* dari repositori GitLab, menghentikan dan menghapus kontainer terakhir, dan menjalankan kontainer baru berdasarkan *image* yang diambil sebelumnya. Proses ini akan sama antara aplikasi Frontend dan Backend.



Gambar 2.5 *Pipeline deployment* otomatis

2.4.4. Evaluasi Kinerja

Langkah terakhir setelah penyebaran otomatis adalah mengevaluasi kinerja dan membandingkan pipeline CI/CD antara dua aplikasi yang dibangun dengan teknologi yang berbeda. Dalam fase pengujian, beberapa metrik dapat digunakan untuk mengukur kinerja pipeline, seperti frekuensi penyebaran, waktu yang dibutuhkan dari commit ke penyebaran, waktu yang dibutuhkan untuk rilis, biaya yang dibutuhkan, dan total *build* yang merujuk pada "*total quality index*". Meskipun penggunaan metrik berbeda antara satu perusahaan dan perusahaan lain

tergantung pada tujuan yang ingin dicapai, menggunakan metrik yang salah akan menyebabkan perilaku penyebaran yang merugikan. Dalam penelitian ini, penulis akan menggunakan metrik berbasis waktu yang akan mengukur waktu yang dibutuhkan dari *commit* dari repositori hingga rilis, kemudian metrik berbasis kualitas yang akan menghitung "*total quality index*".

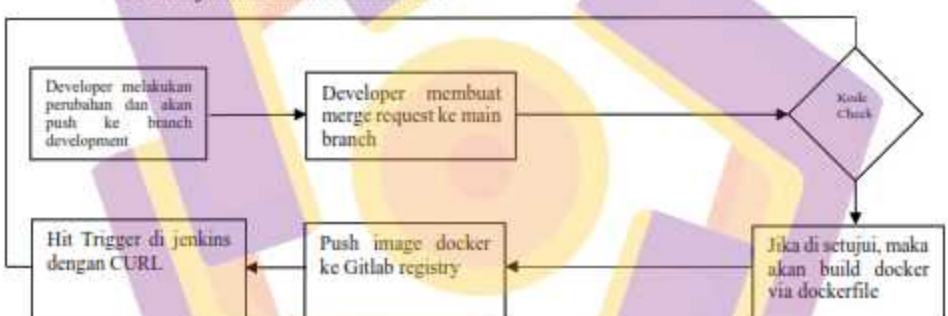
2.5. Implementasi

Bagian implementasi dibagi menjadi dua tahap. Yang pertama adalah tahap mengonfigurasi *Continuous Integration* (CI) menggunakan GitLab dan yang kedua adalah tahap pengaturan *Continuous Deployment* (CD) menggunakan Jenkins.

2.5.1. *Continuous Integration*

Gambar 2.4 menunjukkan infrastruktur aplikasi yang berjalan di atas kontainer Docker. Ada empat kontainer yang berjalan di atas Docker: satu untuk aplikasi *backend* Fishee, satu untuk aplikasi *frontend* Fishee, satu untuk *database* MariaDB, dan satu untuk Jenkins untuk otomatisasi. Keempat kontainer ini dibuat menggunakan empat Dockerfile terpisah untuk memudahkan pemeliharaan dan penyesuaian skala masing-masing kontainer. Aplikasi backend Fishee menggunakan versi PHP 8.1 dan Apache sebagai *webserver*, sehingga proses pembuatan *image* cukup sederhana: cukup menginstal PHP dan perpustakaan php mysqli, dan menggunakan perintah COPY untuk memindahkan kode sumber ke folder /var/www/html/. Aplikasi Frontend Fishee menggunakan NodeJS dan NextJS dan dimuat di Dockerfile terpisah. Dalam penelitian ini, kami menggunakan versi Node JS 16.17 dan mengekspos *port* 3000.

Proses CI di GitLab dimulai jika ada *commit* & *push/merge* oleh pengembang hanya pada cabang pengembangan. Setelah itu membuat permintaan penggabungan untuk menggabungkan *branch* pengembangan ke *branch* main. Selanjutnya, *lead developer* memeriksa perubahan apakah baik atau perlu direvisi, jika disetujui, proses dilanjutkan ke proses CI. Proses CI tidak dieksekusi jika masih berada di *branch development*. Proses CI di GitLab dibagi menjadi dua: proses pembuatan *image* dan penyebaran ke registry GitLab. Dan akhirnya, GitLab CI akan memicu Jenkins yang telah kita persiapkan sebelumnya. Diagram alir lengkap dari proses CI ditunjukkan dalam Gambar 2.6:



Gambar 2.6 Alur Proses CI

Aplikasi Fishee *Backend* akan berjalan pada *port* 80, MariaDB akan secara otomatis diekspos pada *port* 3306, dan aplikasi Fishee *Frontend* akan berjalan pada port *3000*.

2.5.2. Continuous Deployment

Tahap berikutnya adalah *Continuous Deployment* yang akan ditangani oleh Jenkins. Setelah *image* telah dibangun dan disimpan di *cloud* GitLab registry,

GitLab akan secara otomatis memicu pemicu yang sebelumnya disiapkan di Jenkins, sehingga Jenkins dapat memulai *Job* untuk menarik *image* terbaru di GitLab Registry. Jenkins akan menghapus dan menghentikan kontainer lama dan menjalankan kontainer baru. Berdasarkan pada *image* yang telah ditarik dari GitLab Registry. Alur CD pada Jenkins akan seperti yang ditunjukkan pada Gambar 2.7:

Gambar 2.7 Alur Proses CD

2.6. Hasil

2.6.1. Time Based Metric

Time Based Metric atau Metrik Berbasis Waktu akan mengukur waktu yang diperlukan oleh Jenkins dan GitLab CI untuk menyelesaikan seluruh alur dari komit

ke repositori hingga merilis aplikasi dalam mode produksi. Ada dua aplikasi yang diuji: alur aplikasi *backend* Fishee dan alur aplikasi *frontend* Fishee. Proses CI ditunjukkan dalam tabel 2.1.

Tabel 2.1 Hasil Proses CI

Pipeline	*Build*	*Push*	*Total*
Backend Application (PHP)	112.5 detik	41.2 detik	153,7 detik
Frontend Application	133,7 detik	40,5 detik	174,2 detik

Tabel 2.1 menunjukkan aplikasi Fishee Backend dari proses *Continuous Integration* (CI) dengan total waktu 153,7 detik, dengan rincian proses *build image* sebesar 112,5 detik dan proses pendaftaran dorong sebesar 41,2 detik, sementara aplikasi Fishee Frontend mendapat total waktu 174,2 detik untuk proses CI. Dengan rincian 133,7 detik untuk *build image* docker dan 40,5 detik untuk mengunggah *image* ke Gitlab registry. Untuk waktu yang diperlukan untuk *deployment*, aplikasi *backend* Fishee dengan PHP lebih baik daripada aplikasi *frontend* Fishee. Beberapa faktor memengaruhi *build image* docker aplikasi Fishee Backend lebih cepat, seperti aplikasi Fishee Backend tidak perlu menginstal banyak *library* atau paket dibandingkan dengan aplikasi Fishee Frontend saat membangun *image* Docker. Sebaliknya, aplikasi Fishee Frontend harus menginstal NodeJs dan *library* yang diperlukan. Namun, aplikasi Fishee Frontend mengunggah *image* ke registry lebih cepat daripada backend. Pada saat yang sama, faktor yang memengaruhi adalah bahwa ukuran *workspaces* Next JS lebih kecil daripada file Codeigniter. Selanjutnya, proses *Continuous Deployment* oleh Jenkins akan ditunjukkan pada

tabel 2.2 Proses deployment mencakup menarik *image* dari GitLab registry dan menghentikan dan menjalankan wadah baru.

Tabel 2.2 Hasil Proses CD

Pipeline	*Build & Push*	*Deploy*	*Total all pipeline*
Backend Application (PHP)	153,7 detik	6,1 detik	159,8 detik
Frontend Application (JS)	174,2 detik	7,8 detik	184 detik

Tabel 2.2 menunjukkan bahwa proses *deployment* untuk aplikasi Fishee Backend lebih cepat daripada aplikasi *frontend* sebesar 0,3 detik. Perbedaan ini tidak signifikan karena proses *continuous deployment (CD)* sama untuk kedua aplikasi. Dalam penelitian ini, kami menggabungkan Jenkins dan GitLab untuk menjalankan proses CI/CD. Proses *deployment* lebih cepat daripada yang dilaporkan pada penelitian sebelumnya, yang membutuhkan rata-rata 118 detik untuk proses *deployment* aplikasi berbasis *container* dengan Jenkins. Ini juga lebih cepat dari proses yang dijelaskan di penelitian sebelumnya, yang membutuhkan rata-rata 180 detik untuk proses *pipeline* CI/CD. Dalam penelitian ini, waktu rata-rata untuk proses *deployment* adalah 6,1 detik, dan proses pembangunan *image* ditangani oleh GitLab runner, yang tidak membebani server.

2.6.2. Quality Metrics

Metrik Kualitas adalah tahap untuk melihat "*total quality index*" dari sebuah pipeline. Pipa jalankan dalam rentang waktu 1-25 Oktober 2022, untuk aplikasi berikut: 44 *pipeline* aplikasi backend (PHP) dan 30 *pipeline* aplikasi frontend (JS). Metode yang digunakan dalam perhitungan adalah jumlah pipeline berhasil di bagi

dengan jumlah keseluruhan pipeline. Persentase keberhasilan *pipeline* frontend adalah 83,3%, sementara keberhasilan *pipeline backend* adalah 70,45%.

Gambar 2.8 Hasil *Pipeline Job*

Berdasarkan Gambar 2.8, GitLab CI mencatat 13 *kegagalan* pipeline untuk aplikasi backend dan lima kegagalan untuk aplikasi frontend. Kegagalan-kegagalan ini terjadi selama proses pembangunan pada proses integrasi berkelanjutan (CI) dan disebabkan oleh kesalahan pada *image* Docker. Ada beberapa faktor yang dapat menyebabkan kegagalan *build image* pada proses CI, seperti logika kode yang tidak benar, *library* yang belum lengkap, dan konfigurasi CI yang tidak tepat.

2.7. Kesimpulan

Pada penelitian ini dilakukan perbandingan kinerja antara Jenkins dan GitLab CI dalam hal kualitas dan waktu *pipeline* untuk proses *deployment* otomatis aplikasi berbasis kontainer Docker. Aplikasi backend Fishee dibangun dengan *framework* Codeigniter dan memerlukan 112,5 detik untuk *build image* Docker dan 41,2 detik untuk mengunggah *image* ke registri Gitlab, sehingga total waktu adalah 153,7 detik. Aplikasi *frontend* Fishee, yang dibangun menggunakan teknologi *stack* yang berbeda, memerlukan waktu total 184 detik untuk menyelesaikan proses CI/CD. Ini

termasuk 153,7 detik untuk membangun gambar Docker dan mengunggahnya ke registri, dan 7,8 detik untuk proses CD. Waktu ini dapat bervariasi tergantung pada persyaratan dan konfigurasi khusus dari aplikasi, tetapi mereka memberikan *benchmark* yang berguna untuk membandingkan kinerja dari berbagai alat dan pendekatan CI/CD. Dalam hal tingkat keberhasilan, aplikasi frontend Fishee memiliki tingkat keberhasilan yang lebih tinggi daripada aplikasi backend Fishee, mencapai 83,3% dibandingkan dengan 70,45%. Ini menunjukkan bahwa penggunaan GitLab dan Jenkins dalam proses CI/CD dapat efektif untuk mengotomatisasi *deployment* aplikasi berbasis kontainer yang telah ditunjukkan dalam tabel 2.2 dan gambar 2.8.

2.8. Saran

Penelitian ini terbatas pada implementasi di platform Digital Ocean dan tidak membandingkan dengan platform penyedia VPS lain seperti Amazon Web Service atau Google Cloud Platform. Penelitian masa depan dapat memperluas ruang lingkup studi untuk mencakup teknologi stack lain dan membandingkan kinerja dari berbagai alat CI/CD pada berbagai platform penyedia VPS. Ini akan memberikan pemahaman yang lebih komprehensif tentang kinerja berbagai alat CI/CD dan kesesuaian mereka untuk berbagai jenis aplikasi. Selain itu, akan menarik untuk membandingkan kinerja pipeline CI/CD untuk aplikasi dengan stack yang berbeda untuk melihat apakah ada perbedaan dalam hal waktu yang dibutuhkan dan tingkat keberhasilan proses *deployment* otomatis.

DAFTAR PUSTAKA

PUSTAKA BUKU

Badan Standardisasi Nasional. (2009). produksi ikan nila (Oreochromis niloticus Bleeker) kelas pembesaran di kolam air tenang (SNI 7550:2009). Jakarta: Badan Standardisasi Nasional.

Badan Standardisasi Nasional. (2014). Ikan lele dumbo (Clarias sp.) (SNI 6484.4:2014). Jakarta: Badan Standardisasi Nasional.

PUSTAKA MAJALAH, JURNAL ILMIAH ATAU PROSIDING

Gonen, B., & Sawant, D. (2020). Significance of agile software development and SQA powered by automation. Proceedings - 3rd International Conference on Information and Computer Technologies, ICICT 2020, 7–11.

Potdar, A. M., Narayan, D. G., Kengond, S., & Mulla, M. M. (2020). Performance Evaluation of Docker Container and Virtual Machine. Procedia Computer Science, 171, 1419–1428.

Mysari, S., & Bejgam, V. (2020). Continuous Deployment Pipeline Automation Using Jenkins Ansible. International Conference on Emerging Trends in Information Technology and Engineering, Ic-ETITE 2020, 1–4.

Arachchi, S. A. I. B. S., & Perera, I. (2018). Continuous integration and continuous delivery pipeline automation for agile software project management. MERCon 2018 - 4th International Multidisciplinary Moratuwa Engineering Research Conference, 156–161.

Alpery, A., & Ridha, M. A. F. (2021). Implementasi CI-CD dalam Pengembangan Aplikasi Web Menggunakan Docker dan Jenkins. 9th Applied Business and Engineering Conference, 287–296.

Nogueira, A. F., Ribeiro, J. C. B., Zenha-Rela, M., & Craske, A. (2018). Improving la redoute's CI/CD pipeline and devops processes by applying machine learning techniques. Proceedings - 2018 International Conference on the Quality of Information and Communications Technology, QUATIC 2018, 282–286.

Amity University, & Institute of Electrical and Electronics Engineers. (n.d.). Proceedings of the 9th International Conference On Cloud Computing, Data Science and Engineering: Confluence 2019: 10-11 January 2019, Uttar Pradesh, India.

Armenise, V. (2015). Continuous delivery with Jenkins: Jenkins solutions to implement continuous delivery. Proceedings - 3rd International Workshop on Release Engineering, RELENG 2015, 24–27.

Landicho, J. A. (2018). A web-based geographical project monitoring and information system for the road and highways. Journal of Electrical Systems and Information Technology, 5(2), 252–261.

Phie Joarno, R. J., Mohammad Fajar, & Arfan Yunus. (2022). Implementasi Progressive Web Apps Pada Website GetHelp Menggunakan Next.js. KHARISMA Tech, 17(2), 1-15.

Rizal Fathoni Aji, 2018, "Perbandingan Performa Kinerja Node.js, PHP, dan Python dalam Aplikasi REST," Cogito Smart Jurnal, vol. 4, no. 1, pp. 171-187.

C. Singh, N.S. Gaba, M. Kaur, and B. Kaur, 2019, "Comparison of Different CI/CD Tools Integrated with Cloud Platform," 2019 9th International Conference on Cloud Computing, Data Science & Engineering (Confluence), pp. 7-12.

K. Purohit, 2020, "Executing DevOps & CI/CD reduce in manual dependency," International Journal of Scientific and Development Research (IJSDR), vol. 5, no. 6, pp. 511-515.

L. Leite, C. Rocha, F. Kon, D. Milojicic, and P. Meirelles, 2020, "A Survey of DevOps Concepts and Challenges," ACM Computing Surveys, vol. 52, no. 6, pp. 1-35.

Jez Humble and David Farley, 2010, "Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation," Addison-Wesley Professional.

Ramtin Jabbari, Nauman bin Ali, Kai Petersen, Hui Kang, Michael Le, and Shu Tao, 2016, "Container and microservice driven design for cloud infrastructure DevOps," Proceedings of the IEEE International Conference on Cloud Engineering (IC2E'16), pp. 202-211.

Armin Balalaie, Abbas Heydarnoori, and Pooyan Jamshidi, 2016, "Microservices architecture enables DevOps: Migration to a cloud-native architecture," IEEE Software, vol. 33, no. 3, pp. 42-52.

Nicole Forsgren Velasquez Alanna Brown, Gene Kim, Nigel Kersten, and Jez Humble, 2016, "2016 State of DevOps Report," Puppet.

Barry Snyder and Bill Curtis, 2018, "Using analytics to guide improvement during an agile/DevOps transformation," IEEE Software, vol. 35, no. 1, pp. 78-83.

Barroso, L. A., Holzle, U. (2007). The Case for Energy-Proportional Computing. Computer Society, 40(12), 33–37.

Gallaba, K., & McIntosh, S. (2020). Use and Misuse of Continuous Integration Features: An Empirical Study of Projects That (Mis)Use Travis CI. IEEE Transactions on Software Engineering, 46(1), 33–50. https://doi.org/10.1109/TSE.2018.2838131

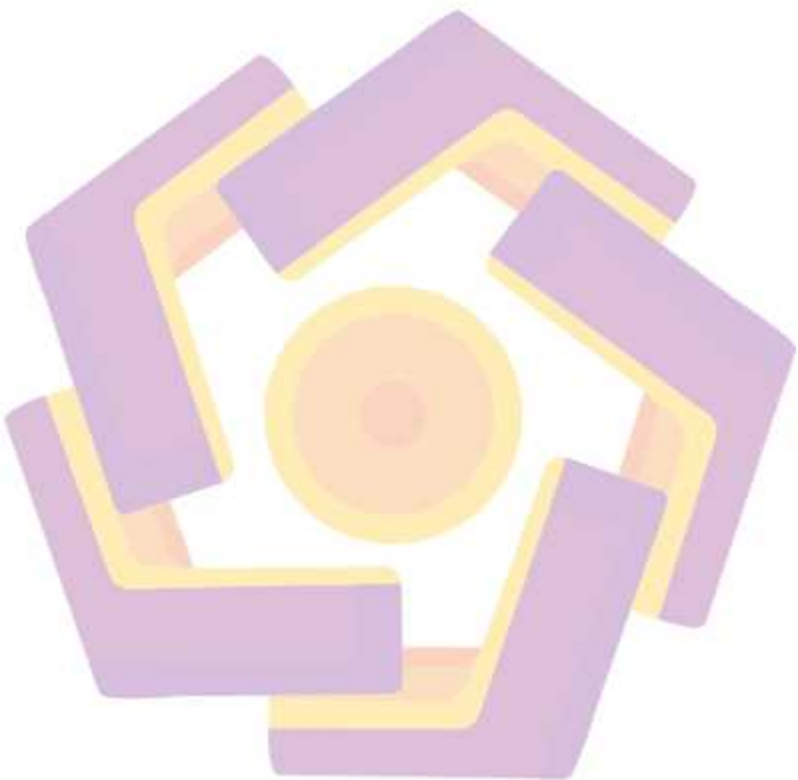
Chen, L. (2018). Microservices: Architecting for Continuous Delivery and DevOps. Proceedings - 2018 IEEE 15th International Conference on Software Architecture, ICSA 2018, 39–46. https://doi.org/10.1109/ICSA.2018.00013

Garg, S., Pundir, P., Rathee, G., Gupta, P. K., Garg, S., & Ahlawat, S. (2021). On Continuous Integration/Continuous Delivery for Automated Deployment of Machine Learning Models using MLOps. Proceedings - 2021 IEEE 4th International Conference on Artificial Intelligence and Knowledge Engineering, AIKE 2021, 25–28. https://doi.org/10.1109/AIKE52691.2021.00010

Garg, S., & Garg, S. (2019). Automated cloud infrastructure, continuous integration and continuous delivery using Docker with robust container security. In 2019 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR) (pp. 467–470). IEEE.

LAMPIRAN

Lampiran 1



Automatic Deployment Pipeline for Containerized Application of IoT Device

Yogi Yulianto
Master of Informatics Engineering
AMIKOM Yogyakarta University
Yogyakarta, Indonesia
yogi.yulianto@students.amikom.ac.id

Erna Utami
Master of Informatics Engineering
AMIKOM Yogyakarta University
Yogyakarta, Indonesia
erna.u@amikom.ac.id

Kusnawi
Master of Informatics Engineering
AMIKOM Yogyakarta University
Yogyakarta, Indonesia
khasnawi@amikom.ac.id

In this modern era, many companies in the technology field compete to win the hearts of customers. To keep up with the competition and meet customers' needs, companies must be able to adapt to changes in their products. One software development methodology that focuses on customer satisfaction is the agile methodology. This approach is popular among software development teams because it allows them to quickly respond to changes in customer needs. This study aims to apply a combination of Gitlab CI and Jenkins to implement a CI/CD pipeline on two software applications called Fishee Backend App and Fishee Frontend App, which use different technologies and also use agile methodology approach in development process. The study aims to evaluate the quality of the pipeline and the time required for the deliver changes to customer. The results show that the Fishee Backend App takes time 153.7 seconds and The Fishee Frontend App takes 174.2. In terms of successful pipeline jobs, the Fishee Backend App reached 70.45% and Fishee Frontend App reached 83,3%. Based on these findings, the use of a combination of Jenkins and Gitlab CI in the CI/CD process can be effective and more faster than previous research for automating the deployment of container-based applications. However, the study was limited to implementation on the Digital Ocean platform and did not compare with other VPS provider platforms. It is recommended that future research extend the scope of the study to include other technology stacks and compare the performance of different CI/CD tools on a range of VPS provider platforms.

Keywords—Agile, CI/CD, Docker, Automation, Software Development

I. Introduction

In this modern era, many companies must be adaptive to change their product following customers' needs. To ensure the success of the software development project, software development methodology is the primary key to winning the competition. Several parameters that can measure the success of application delivery to the customer are good accuracy, cost-effectiveness, and fast delivery [12]. The traditional waterfall method has some problems, such as no error checking during the development process, extreme orientation to documents, taking much time for planning because the process can be back to the previous process, and needing to be adaptive to changes[12]. Over time agile methodology was introduced as a software development methodology that is adaptive to changes. Agile prioritizes customer need as is primary key [3]. In the agile cycle, there are several stages of development, starting from planning, design, implementation, code, deployment, and feedback. In an organization, there is a person to collect and retrieves code to be deployed to the server. Without the concept of DevOps, the operation team will manually collect and deploy code to the server to deliver the changes to the user[13]. However, that creates a new problem. If the operation teams are not present in their companies and the client needs the code deployed quickly, it will cause a delay in the delivery process[13]. DevOps is a way for an organization to automate software's continuous delivery process, ensuring its quality and reliability [14]. To implement DevOps at the Fishee software development environment, we will implement an automatic deployment method with an agile methodology approach called Continuous Integration and Continuous Delivery (CI/CD) to reduce manual jobs. The previous study has shown that the CI/CD pipeline efficiently increases the speed of agile methodology in the production stage of super app applications such as Facebook, GitHub, Rally Soft, and Netflix[4]. This automation process is handled by third-party software such as Jenkins, Gitlab CI, GitHub Action, Travis CI, and other CI software. CI/CD is like a bridge between the operational and software development teams in the automation process, such as building docker images, testing, and deployment[5]. In addition, CI/CD can monitor the deployment process so that developers can update software quickly, precisely, and organizationally. Docker is a container-based and open-source application that allows software developers to create, run, build, test, and release, applications in an isolated container.

Docker makes the process of packaging applications and their components quickly in an isolated container so that they can be run on the local infrastructure without making configuration changes to the container. Research [2] shows that docker containers have better performance on virtual machines (VM) in terms of CPU usage, memory usage, read/write storage speed, and load tests [2]. Because of that, they were implementing docker in agile methodology as a better choice than using VM. In this study, we will implement an automatic deployment pipeline for containerized application of IoT devices. The application is called Fishee. Fishee is a device for automatic fish feeding and monitoring based on the Internet of Things. However, this study will discuss CI/CD in Fishee Frontend Application and Backend Application, not application at the embedded system inside the Fishee device.

The Fishee application will run on the docker container. The CI/CD process will be handled by a combination of Jenkins and GitLab CI. Jenkins is an open-source automation software developed using the Java programming language with community support and many plugins [12], making Jenkins one of the most popular CI/CD tools for implementing Continuous Integration and Continuous Delivery. Gitlab CI is also one of the CI/CD tools that are easy to use because everything in the Gitlab pipeline will be set in one YML file[12]. Gitlab also has a performance monitoring feature, an advantage over Jenkins[12].

This research focuses on the CI/CD process for container-based applications using a combination of Jenkins and Gitlab CI. The applications used in the study are the Fishee Backend application and the Fishee Frontend application, which have different technology stacks (PHP and Node JS). The research will analyze the time required and success rate of the automated deployment process for these applications.

II. LITERATURE STUDY

A. Automation on Agile Software Development

In software development with the Agile approach, all development, including the process of analysis, design, coding, and testing, is an iterative activity. The project manager in the agile method is usually called the scrum master, who ensures that all the existing rules are correctly implemented [1]. In agile software development, the development process is iterative. Change is relatively small and will be continuing development. The agile process is divided into planning, design, coding, deployment, testing, and evaluation. Agile workflow in the Fishee team, as shown in Figure 1, starts from receiving customer requests, then teams will discuss requests. After that, a sprint & development will be run by the team, then the changes will be committed to the repository and deployed to the server so that customers can use change in the application. This process will be iterative, and if this process is handled manually by the person that causes some trouble, such as wrong deployment and conflict when deployment, an automation process is needed to reduce the team's workload.

Figure 1 Agile Workflow in Fishee team

B. Containerization Application

A container is a complete runtime environment in which an application already exists, along with the dependencies, libraries, and configurations needed to run the application. By containerizing an application, it is easier for developers to deploy both applications and infrastructure simultaneously. One application that makes it easy for software developers to package, build, test, and deploy to applications that are ready to use is Docker. Docker is an open platform for developing, shipping, and running applications. Docker uses namespaces to isolate workspaces from a container[12]. For specific containers, Docker creates namespaces so that other containers cannot access that container. This isolation aims to ensure that each process is separated by not allowing it to find resources in other groups or containers. One of the benefits of using Docker is that applications are scalable by simply adding or removing a container[12]. Docker is almost similar to virtual machines. The difference is that Docker is much lighter[16]. Besides that, Docker can be created so easily and quickly that the deployment process is so fast. Due to Docker's isolation and security processes, developers can run multiple runtimes with multiple containers on a single machine. Docker uses a client-server architecture. Figure 2 describes the docker workflow and how the docker client relates to the docker daemon, which can pull images from the registry. Docker will execute what written inside dockerfile[25].

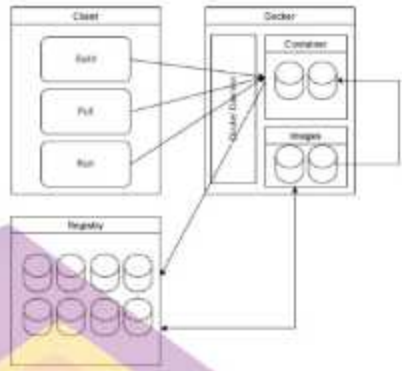


Figure 2 Docker workflow diagram

C. Continuous Integration and Continuous Deployment

When an organization wants to implement the CI/CD pipeline, the first thing to do is to implement the CI process first because CD cannot be started before CI is implemented. Continuous Integration to Continuous Deployment, this automation process can reduce processes previously handled by team operation manually and replace them with software automatically. CI or Continuous Integration is a method in software development that allows developers to integrate their work, such as build and test automatically. It can make the software more quickly accepted by the user and help find bugs and fix them faster. [2] compared to containerization and Virtual Machine (VM) based in developing microservices applications, container-based technology has high scalability, portability, and better performance in deployment than VM-based deployments. Continuous deployment means making any committed changes in the repository implementable at the production stage. According to [15], one of the components in the continuous deployment variation is the continuous delivery of code committed to the repository must be code that can be applied in the production environment. After going through several stages, such as automated build and testing, the software will be deployed to the production environment with just the press of a button. The main distinction between CI and CD is that CI focuses on ensuring that an application is in a production-ready state after undergoing quality checks, while CD focuses on actually deploying the application to a production environment[24]

D. CI/CD Tools

In the modern era, many companies compete to implement CI/CD in their application development. So many open source tools have been created to help handle the CI / CD process implementation such as Jenkins, Team City, Bambo, Travis CI[21], etc. One of the most populer jobs on stackoverflow for various integration tools is Jenkins[3]. Jenkins is an Open Source (OSS) CI platform with the primary goal of automating the build and test processes.

Jenkins is made with the Java programming language, and many available plugins can be used [8]. In addition to Jenkins is a large number of plugins, and support from a large community makes Jenkins flexible and can be customized according to the needs of application developers. Jenkins can create container-based application deployment processes through Jobs that are executed automatically via triggers such as webhooks. Besides Jenkins, a CI/CD tool that is quite popular is GitLab CI. Compared to Jenkins, paid version of GitLab is better because, in research [12], GitLab paid runner is faster than Jenkins for doing CI/CD, and GitLab has a feature that allows the developer to monitor the performance of their server.

III. METHODOLOGY

The methodology in this study is divided into four stages. The first is defining the software architecture, then formulating the server's infrastructure, and then designing and implementing the automatic deployment pipeline process with Jenkins and GitLab. The last one is entering the performance evaluation of the CI/CD pipeline.

A. Software Architecture

The software that will be deployed as a microservices architecture which is frontend and backend, is created by different technologies. Implementing microservices is in line with implementing continuous delivery. A well-separated architecture will impact good build and testing capabilities [17]. Despite the benefits of automatic deployment, developers must be careful that configuring the infrastructure for CI/CD will require a lot of effort [18]. Breaking an application into microservices requires a separate pipeline. Application's or software architecture can be a key barrier[6], in this research [6] change monolith architecture to microservice architecture can reduce CD time from 30 minutes to 3 minutes.

In this research, we use a microservice architecture with two applications that will be deployed: the Fishee Frontend App for customers to access their dashboard and ordering IoT devices, and the Fishee Backend App for handling requests from the Fishee Frontend App. The Fishee Frontend application is built with the Next JS framework, which has been shown to improve website performance by 14% [10]. Next JS is supported by Node JS, which is one of the fastest frameworks compared to PHP and Python-based applications [11]. The Fishee Backend is a REST API that will connect the IoT hardware to the database.

The REST API will be developed by the Codeigniter Framework, which is built with the PHP Programming Language. The Codeigniter framework is a proven agile and open-source framework that offers a good and clean code separation concept [9]. Codeigniter applies the MVC (Model-View-Controller) design pattern, a software approach that separates the application logic and the view part. The web server used is Apache which can run PHP applications on it. For the Database, we will use MariaDB. The application is divided into two layers, the Backend Application Layer and the Data Layer, as shown in Figure 3.

The IoT will hit the Fishee Backend Application to send data to the Database, and after that, the REST API will save to MariaDB. The stored data can be viewed in the Fishee Frontend application created by Next JS. Fishee Frontend application consumes API from REST API. The user of Fishee mainly accesses the Frontend application to view their statistic of use application and control the IoT Device.

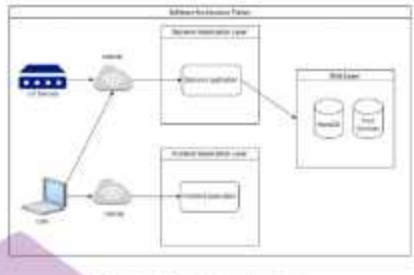


Figure 3 Software architecture

B. Server Infrastructure

In this study, we will use Digital Ocean Cloud Droplets VPS with 2 CPU, 4 GB RAM, and 80 GB SSD Disk. This server will be used to run four isolated containers: one for Jenkins, one for the Fishee Backend application, one for the Fishee Frontend application, and one for the MariaDB database. These containers will be created using Dockerfile. The host operating system used is Ubuntu 18.04, which runs Docker and the containers containing the application and database. Jenkins is installed on the top layer of the hierarchy.

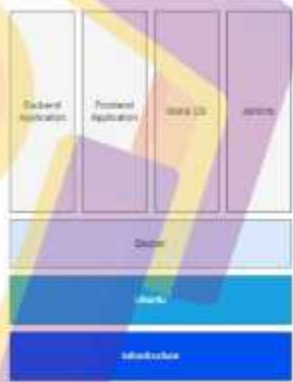


Figure 4 Server Architecture

C. Automatic Deployment Pipeline

The deployment process will be split into several stages, as described in Figure 5, starting with the developer committing and pushing changes to the GitLab repository with a branch main which will trigger GitLab CI to run. The second step is a docker image built by the docker file, which contains the application dan webserver. After that, the image will be deployed to the GitLab registry. When the build and deploy process on GitLab is complete, Jenkins will be triggered by GitLab CI to run a Continuous Deployment (CD) image to the server. The job that Jenkins does has three stages. The first is pulling the image from the Gitlab repository, stopping and removing the last container, and running a new container

based on the image that was taken previously. This process will be the same between Frontend and Backend applications.

Figure 5 Automatic Deployment Pipeline

D. Evaluation Performance

The final step after the automatic deployment is evaluating performance and comparing the CI/CD pipeline between 2 applications built with different technologies. In the testing phase, several metrics can be used to measure the performance of the pipeline, such as deployment frequency, time taken from commit to deployment [18], the time required for release, cost required, and total builds referring to the "total quality index" [19]. Although the use of metrics is different for a company and the other company depending on the goals that want to be achieved, using incorrect metrics will cause adverse deployment behavior [20]. In this study, the author will use time-based metrics that will measure the time it takes from commits from the repository to release [18, 19], then quality-based metrics that will calculate the "total quality index" [19].

IV. IMPLEMENTATION

The implementation section is split into two stages. The first is the configure Continuous Integration (CI) stage using GitLab and the second is the Continuous Deployment (CD) setting stage using Jenkins.

A. Continuous Integration

Figure 4 shows the infrastructure of the application running on top of the Docker container. There are four running containers on top of Docker: one for the Fishee backend application, one for the Fishee frontend application, one for the MariaDB database, and one for Jenkins for automation. These four containers are created using four separate Dockerfiles to make it easier to maintain and scale each container individually. The Fishee backend application uses PHP version 8.1 and Apache as its web server, so the build image process is straightforward: just install PHP and the php mysqli library, and use the COPY command to move the source code to the /var/www/html/ folder. The Fishee Frontend Application uses NodeJS and NextJS and is loaded in a separate Dockerfile. In this research, we used Node JS version 16.17 and exposed port 3000.

CI process in GitLab starts if there is a commit & push/merge by the developer only on the development branch. After that create merge request to merge branch development into branch main. Next, lead developer check the change is good or need revision, if approved the process continue to CI process, The CI process is not executed if it is still on the development branch. The CI process in GitLab is split into two: the image build process and the push to the GitLab registry. And finally GitLab CI will be hit trigger Jenkins we prepare before. The complete flowchart of CI process is shown in Figure 6:

Figure 6 Flowchart CI Process

The Fishee Backend application will run on port 80, MariaDB will be automatically exposed on port 3306, and the Fishee frontend application will run on port 3000.

B. Continuous Deployment

The next stage is Continuous Deployment which Jenkins will handle. After the image has been built and stored in the cloud in the GitLab registry, GitLab will automatically hit the trigger that was previously prepared in Jenkins, so Jenkins can start a Job to pull the latest images in the GitLab Registry. Jenkins will remove and stop the old container and run the new container. Based on the image that was pulled from the GitLab Registry. The CD flowchart on Jenkins will be as shown in Figure 7:

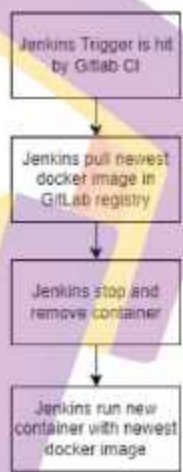


Figure 7 Jenkins CD Pipeline

V. RESULT

A. Time Based Metrics

Time-Based Metrics measure the time it takes for both Jenkins and GitLab CI to perform the entire pipeline from commits to the repository to release applications in production mode. There are two pipelines tested: Fishee backend application pipelines and Fishee frontend applications. The CI process is shown in table 1.

TABLE I. CONTINUOUS INTEGRATION

Pipeline	Build	Push	Total
Backend Application (PHP)	112.5 seconds	41.2 seconds	153,7 seconds
Frontend Application	133,7 seconds	40,5 seconds	174,2 seconds

Table 1 shows the Fishee Backend application of the Continuous Integration (CI) process with a total time of 153.7 seconds, with details of the build image process of 112.5 seconds and the push registry process of 41.2 seconds, while the Fishee Frontend application gets a total time of 174.2 seconds for the CI process. With detail 133.7 seconds to build the docker image and 40.5 seconds to upload the image to the registry Gitlab. For the time required to release, Fishee backend applications with PHP framework are better than Fishee frontend applications. Some factors affect building the docker image Fishee Backend application faster, such as the Fishee Backend application don't need to install much library or package compared with Fishee Frontend application when building the Docker image. In contrast, the Fishee Frontend application must install NodeJs and the required libraries. However, the Fishee Frontend application pushes image time to the registry faster than the backend. At the same time, the thing that affects it is that the size of the Next JS framework is smaller than the Codeigniter framework.

Next, the Continuous Deployment process by Jenkins will be shown in table 2. The deployment process includes pulling an image from the GitLab registry and stopping and running a new container.

TABLE II. CONTINUOUS DEPLOYMENT

Pipeline	Build + Push	Deploy	Total all pipeline
Backend Application (PHP)	153,7 seconds	6,1 seconds	159,8 seconds
Frontend Application	174,2 seconds	7,8 seconds	184 seconds

Table 2 shows that the deployment process for the Fishee Backend application is faster than the frontend application by 0.3 seconds. This difference is not significant because the continuous deployment (CD) process is the same for both applications. In this research, we combine Jenkins and GitLab to run the CI/CD process. The deployment process is faster than the one reported in [5], which took an average of 118 seconds for the container-based application deployment process with Jenkins. It is also faster than the process described in [23], which took an average of 180 seconds to process CI/CD pipelines. In this research, the average time for the deployment process is 6.1 seconds, and the image build process is handled by GitLab runner, which does not burden the server.

B. Quality Metrics

Quality Metrics are the stage to see a pipelne's "total quality index". The pipelines run within October 1-25, 2022, for the following applications: 44 backend application pipelines (PHP) and 30 frontend application pipelines (JS). The percentage of success of the frontend pipeline is 83.3%, while the success of the backend pipeline is 70.45%.

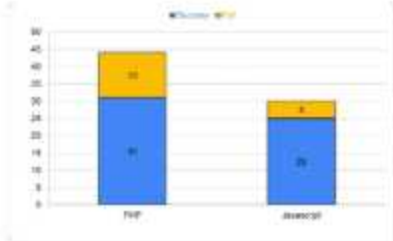


Figure 8 Pipeline Job Result

Based on Figure 8, GitLab CI recorded 13 pipeline failures for the backend application and five failures for the frontend application. These failures occurred during the build process in the continuous integration (CI) process and were caused by errors in the Docker images. There are several factors that can cause image build failures in the CI process, such as incorrect code logic, missing libraries, and improper CI configuration.

CONCLUSION

In this study, the performance of Jenkins and GitLab CI was compared in terms of pipeline quality and time for the automatic deployment process of Docker container-based applications. The Fishee backend app was built with the Codeigniter framework and required 112.5 seconds to build the Docker image and 41.2 seconds to push the image to the Gitlab registry, for a total time of 153.7 seconds. The Fishee frontend app, which was built using a different technology stack, took a total of 184 seconds to complete the CI/CD process. This included 153.7 seconds for building the Docker image and pushing it to the registry, and 7.8 seconds for the CD process. These times may vary depending on the specific requirements and configuration of the applications, but they provide a useful benchmark for comparing the performance of different CI/CD tools and approaches.

In terms of success rate, the Fishee frontend app had a higher success rate than the Fishee backend app, reaching 83.3% compared to 70.45%. This indicates that the use of GitLab and Jenkins in the CI/CD process can be effective for automating the deployment of container-based applications that has be shown table 2 and figures 10. However, this study was limited to implementation on the Digital Ocean platform and did not compare with other VPS provider platforms such as Amazon Web Service or Google Cloud Platform. Future research could extend the scope of the study to include other technology stacks and compare the performance of different CI/CD tools on a range of VPS provider platforms. This would provide a more comprehensive understanding of the performance of different CI/CD tools and their suitability for different types of applications. Additionally, it would be interesting to compare the performance of CI/CD pipelines for applications with different stack to see if there are any differences in terms of the time required and the success rate of the automated deployment process.

References

[1] Gonen, B., & Sarwani, D. (2020). Significance of agile software development and SQA powered by automation. *Proceedings - 3rd International Conference on Information and Computer Technologies, ICICT 2020*, 7–11. https://doi.org/10.1109/ICICT50521.2020.00009

[2] Pordar, A. M., Narayan, D. G., Kengond, S., & Mulla, M. M. (2020). Performance Evaluation of Docker Container and Virtual Machine. *Procedia Computer Science*, *171*, 1419–1428. https://doi.org/10.1016/j.procs.2020.04.152

[3] Mysari, S., & Bejgam, V. (2020). Continuous Integration and Continuous Deployment Pipeline Automation Using Jenkins Ansible. *International Conference on Emerging Trends in Information Technology and Engineering, Ic-ETITE 2020*, 1–4. https://doi.org/10.1109/ic-ETITE47903.2020.239

[4] Arachchi, S. A. I. B. S., & Perera, I. (2018). Continuous integration and continuous delivery pipeline automation for agile software project management. *MERCon 2018 - 4th International Multidisciplinary Moratuwa Engineering Research Conference*, 156–161. https://doi.org/10.1109/MERCon.2018.8421965

[5] Alpery, A., & Ridha, M. A. F. (2021). Implementasi CI-CD dalam Pengembangan Aplikasi Web Menggunakan Docker dan Jenkins. *9th Applied Business and Engineering Conference*, 287–296.

[6] Nogueira, A. F., Ribeiro, J. C. B., Zenha-Rela, M., & Craske, A. (2018). Improving la redoute's CI/CD pipeline and devops processes by applying machine learning techniques. *Proceedings - 2018 International Conference on the Quality of Information and Communications Technology, QUATIC 2018*, 282–286. https://doi.org/10.1109/QUATIC.2018.00050

[7] Amity University, & Institute of Electrical and Electronics Engineers. (n.d.). Proceedings of the 9th International Conference On Cloud Computing, Data Science and Engineering : Confluence 2019 : 10-11 January 2019, Uttar Pradesh, India.

[8] Armenise, V. (2015). Continuous delivery with Jenkins: Jenkins solutions to implement continuous delivery. *Proceedings - 3rd International Workshop on Release Engineering, RELENG 2015*, 24–27. https://doi.org/10.1109/RELENG.2015.19

[9] Landicho, J. A. (2018). A web-based geographical project monitoring and information system for the road and highways. *Journal of Electrical Systems and Information Technology*, *5*(2), 252–261. https://doi.org/10.1016/J.JESIT.2016.10.011

[10] Pine Jourro, R. J., Mohammad Fajar, & Arfan Yunus. (2022). Implementasi Progressive Web Apps Pada Website GoHelp Menggunakan Next.js. *KHARISMA Tech*, *17*(2), 1–15. https://doi.org/10.55645/kharismatech.v17i2.219

[11] Rizal Fathoni Aji. (2018). Perbandingan Performa Kinerja Node.js, PHP, dan Python dalam Aplikasi REST. *Cogito Smart Journal Vol. 4 No. 1 (2018): CogiTo Smart Journal*. https://doi.org/10.31154/cogito.v4i1.92.171-187

[12] C. Singh, N. S. Gaba, M. Kaur and B. Kaur, "Comparison of Different CI/CD Tools Integrated with Cloud Platform," 2019 9th International Conference on Cloud Computing, Data Science & Engineering (Confluence), 2019, pp. 7-12, doi: 10.1109/CONFLUENCE.2019.8776985

[13] Purohit, K.: Executing DevOps & CI/CD reduce in manual dependency. IJSDR 5(6), 511–515 (2020)

[14] Leite, L., Rocha, C., Kon, F., Milojicic, D., & Meirelles, P. (2020). A Survey of DevOps Concepts and Challenges. In ACM Computing Surveys (Vol. 52, Issue 6, pp. 1–35). Association for Computing Machinery (ACM). https://doi.org/10.1145/3359981

[15] Jez Humble and David Farley. 2010. Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation. Addison-Wesley Professional. [83] Ramlin Jabbari, Nauman bin Ali, Kai Petersen, and

[16] Hui Kang, Michael Le, and Shu Tao. 2016. Container and microservice driven design for cloud infrastructure DevOps. In Proceedings of the IEEE International Conference on Cloud Engineering (IC2E'16). 202–211. Code: I58.

[17] Armin Balalaie, Abbas Heydarnoori, and Pooyan Jamshidi. 2016. Microservices architecture enables DevOps: Migration to a cloud-native architecture. IEEE Softw. 33, 3 (2016), 42–52. Code: A2.

[18] Nicole Forsgren Velasquez Alanna Brown, Gene Kim, Nigel Kersten, and Jez Humble. 2016. 2016 State of DevOps Report. Retrieved from: https://puppet.com/resources/whitepaper/2016-state-of-devops-report.

[19] Barry Snyder and Bill Curtis, 2018. Using analytics to guide improvement during an agile/DevOps transformation. IEEE Softw. 35, 1 (2018), 78–83. Code: I7.

[20] Patrick Kua. 2013. An Appropriate Use of Metrics. Retrieved from: https://martinfowler.com/articles/useOfMetrics.html

[21] Barroso, L. A., Hölzle, U., 2007. The Case for Energy-Proportional Computing, Computer Society, vol. 40, no. 12, Hal 33–37.

[22] Gallaba, K., & McIntosh, S. (2020). Use and Misuse of Continuous Integration Features: An Empirical Study of Projects That (Mis)Use Travis CI. *IEEE Transactions on Software Engineering*, *46*(1), 33–50. https://doi.org/10.1109/TSE.2018.2838131

[23] Chen, L. (2018). Microservices: Architecting for Continuous Delivery and DevOps. *Proceedings - 2018 IEEE 15th International Conference on Software Architecture, ICSA 2018*, 39–46. https://doi.org/10.1109/ICSA.2018.00013

[24] Garg, S., Pundir, P., Rathee, G., Gupta, P. K., Garg, S., & Ahlawat, S. (2021). On Continuous Integration / Continuous Delivery for Automated Deployment of Machine Learning Models using MLOps. *Proceedings - 2021 IEEE 4th International Conference on Artificial Intelligence and Knowledge Engineering, AIKE 2021*, 25–28. https://doi.org/10.1109/AIKE52691.2021.00010

[25] S. Garg and S. Garg, "Automated cloud infrastructure, continuous integration and continuous delivery using docker with robust container security," in 2019 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR), IEEE, 2019, pp. 467–470.

Lampiran II

Lampiran III

Lampiran IV

Lampiran V

Lampiran VI

ICITISEE 2022

Hybrid (Online & Onsite), 13-14 December 2022

IEEE

Our Ref.: ICITISEE/LoA/[illegible]/2022/120
Date: December, 08th 2022

LETTER OF ACCEPTANCE

2022 6th INTERNATIONAL CONFERENCE ON INFORMATION TECHNOLOGY, INFORMATION SYSTEMS AND ELECTRICAL ENGINEERING (ICITISEE)

Author(s) : [illegible]
Paper Title : # [illegible] - Automatic Deployment Pipeline for Containerized Application of IoT Device

Dear Author(s),
We are pleased to inform you that your paper entitled above has been accepted to be presented in the 2022 6th International Conference on Information Technology, Information Systems and Electrical Engineering (ICITISEE) which organized by Universitas Amikom Yogyakarta with our partner [illegible] Universitas Gadjah Mada, Universitas AMIKOM Purwokerto, and IEEE. The 2022 6th ICITISEE will be conducted as a hybrid conference: Online (Virtual) and Onsite (Grand [illegible] Malioboro, Yogyakarta Indonesia) from 13th – 14th December 2022. **The accepted paper has been strictly undergone the peer-reviewed process and will be submitted for uploading to the IEEEXplore Digital Library and it will be normally indexed in SCOPUS database if you attending and presenting your paper at the 2022 6th International Conference on Information Technology, Information Systems and Electrical Engineering (ICITISEE) on 13th-14th December 2022** such as:

1) Proceedings IEEE Xplore Library (XPLORE COMPLIANT) ISBN: 978-[illegible]
2) Proceedings IEEE Xplore Library (USB) ISBN: 978-[illegible]

Please understand perfectly that the publication of papers in the IEEEXplore Digital Library and it will be normally indexed in SCOPUS database will take AT LEAST **1-4 months** from the conference date. By registering to this conference, authors and co-authors will be deemed to agree, understand, and accept all IEEE Official Conference Terms and Conditions at [illegible]

Kindly visit our website http://www.icitisee.org for more information about the conference. Congratulation on the acceptance of your paper and thank you for your interest in the 2022 6th International Conference on Information Technology, Information Systems and Electrical Engineering (ICITISEE). We look forward to seeing you at the conference soon.

Best Regards,

ICITISEE

Prof. Dr. [illegible]
General Chair of 2022 6th ICITISEE

SECRETARIAT

Universitas Amikom Yogyakarta
Jalan Ring Road Utara, Condong Catur, Sleman, D.I. Yogyakarta, Indonesia 55283

+62 851-5872-7484 (Mr. Yoga)

icitisee@amikom.ac.id

https://icitisee.org/

Lampiran VII