IMPLEMENTASI CI/CD PADA MICROSERVICE KOMPRES GAMBAR MENGGUNAKAN DRONE.IO DAN ARGOCD

TUGAS AKHIR

Diajukan untuk Memenuhi Salah Satu Persyaratan Mencapai Derajat Sarjana Komputer

Dibuat Oleh:

VRIYAS HARTAMA ADESAPUTRA

180709721

PROGRAM STUDI INFORMATIKA

FAKULTAS TEKNOLOGI INDUSTRI

UNIVERSITAS ATMA JAYA YOGYAKARTA

2022

HALAMAN PENGESAHAN

Tugas Akhir Berjudul

IMPLEMENTASI CONTINUOUS INTEGRATION/CONTINUOUS DEPLOYMENT PADA MICROSERVICE KOMPRES GAMBAR MENGGUNAKAN DRONE.IO DAN AGRO CD

yang disusun oleh

Vriyas Hartama Adesaputra

180709721

dinyatakan telah memenuhi syarat pada tanggal 25 Agustus 2022

			Keterangan
Dosen Pembimbing 1	:	Joanna Ardhyanti Mita N, S.Kom., M.Kom	Telah Menyetujui
Dosen Pembimbing 2	:	Dr. Andi Wahju Rahardjo Emanuel, BSEE., MSSE	Telah Menyetujui
Tim Penguji			
Penguji 1	:	Joanna Ardhyanti Mita N, S.Kom., M.Kom	Telah Menyetujui
Penguji 2	:	Yonathan Dri Handarkho, ST., M.Eng, Ph.D.	Telah Menyetujui
Penguji 3	:	Herlina, S.Kom., M.Eng	Telah Menyetujui

Yogyakarta, 25 Agustus 2022

Universitas Atma Jaya Yogyakarta

Teknologi Industri

Dekan

ttd.

Dr. A. Teguh Siswantoro, M.Sc.

PERNYATAAN ORISINALITAS DAN PUBLIKASI ILMIAH

Saya yang bertanda tangan dibawah ini:

Nama Lengkap	:	Vriyas Hartama Adesaputra
NPM	:	180709721
Program Studi	:	Informatika
Fakultas	:	Teknologi Industri
Judul Penelitian	:	Implementasi CI/CD Pada *Microservice* Kompres Gambar Menggunakan Drone.io Dan ArgoCD

Menyatakan dengan ini:

1. Tugas Akhir ini adalah benar tidak merupakan salinan sebagian atau keseluruhan dari karya penelitian lain.
2. Memberikan kepada Universitas Atma Jaya Yogyakarta atas penelitian ini, berupa Hak untuk menyimpan, mengelola, mendistribusikan, dan menampilikan hasil penelitian selama tetap mencantumkan nama penulis.
3. Bersedia menanggung secara pribadi segala bentuk tuntutan hukum atas pelanggaran Hak Cipta dalam pembuatan Tugas Akhir ini.

Demikianlah pernyataan ini dibuat dan dapat dipergunakan sebagaimana mestinya.

Yogyakarta, 22 Agustus 2022

Yang menyatakan,

Vriyas Hartama Adesaputra

180709721

HALAMAN PERSEMBAHAN

“Talk is cheap. Show me the code.” — Linus Torvalds

KATA PENGANTAR

Puji dan syukur penulis ucapkan kepada Tuhan Yang Maha Esa atas berkat dan karunia-Nya, penulis dapat menyelesaikan penulisan tugas akhir "Implementasi CI/CD Pada *Microservice* Kompres Gambar Menggunakan Drone.io Dan ArgoCD" ini dengan baik. Penulisan tugas akhir ini bertujuan untuk memenuhi salah satu syarat untuk mencapai derajat sarjana komputer dari Program Studi Informatika, Fakultas Teknologi Industri di Universitas Atma Jaya Yogyakarta. Penyusunan laporan ini tidak terlepas dari bantuan segala pihak, baik pihak dari Universitas Atma Jaya Yogyakarta. Oleh karena itu penulis mengucapkan terima kasih kepada:

1. Tuhan Yang Maha Esa atas berkat dan karunia-Nya,
2. Orang Tua dan Keluarga yang selalu memberikan dukungan kepada penulis,
3. Ibu Pembimbing akademik: Patricia Ardanari, S.Si., M.T., selaku dosen pembimbing akademik penulis,
4. Ibu Joanna Ardhyanti Mita N, S.Kom., M.Kom., selaku dosen pembimbing I yang telah membimbing penulis selama proses penulisan tugas akhir.
5. Bapak Dr. Andi Wahju Rahardjo Emanuel, BSEE., MSSE., selaku dosen pembimbing II yang telah membimbing penulis selama proses penulisan tugas akhir.
6. Seluruh teman penulis dan mahasiswa Universitas Atma Jaya Yogyakarta.

Akhir kata, penulis menyadari dalam pelaksanaan kegiatan magang maupun proses penulisan laporan ini jauh dari kata sempurna. Tidak menutup adanya kritik dan saran yang berguna, penulis mempersilahkan untuk menyampaikan hal tersebut. Semoga dengan penulisan laporan ini, segala materi yang tertuang didalam materi ini dapat memberikan inspirasi, ide, atau manfaat bagi semua pihak.

Yogyakarta, 22 Agustus 2022

Penulis

Vriyas Hartama Adesaputra

DAFTAR ISI

LEMBAR PENGESAHAN ii
PERNYATAAN ORISINALITAS DAN PUBLIKASI ILMIAH iii
HALAMAN PERSEMBAHAN iv
KATA PENGANTAR v
DAFTAR ISI vi
DAFTAR GAMBAR vii
DAFTAR TABEL x
INTISARI xi
BAB I PENDAHULUAN 1
BAB II TINJAUAN PUSTAKA 6
BAB III LANDASAN TEORI 11
BAB IV ANALISIS DAN PERANCANGAN SISTEM 15
BAB V IMPLEMENTASI DAN PENGUJIAN SISTEM 22
BAB VI PENUTUP 59
DAFTAR PUSTAKA 61

DAFTAR GAMBAR

Gambar 3. 1 Diagram Alir Proses CI/CD Pipeline 13
Gambar 4. 1 Antar Muka Aplikasi Pengguna API Microservice Kompres Gambar 16
Gambar 4.2 Alur Kerja Pengembang 19
Gambar 4. 3 Diagram Alir Proses Perilisan Menggunakan CI/CD Pipeline 20
Gambar 4. 4 Diagram Arsitektur Continuous Integration dan Continuous Deployment 21
Gambar 5. 1 Struktur direktori berkas yang memuat konfigurasi IaC untuk host machine. 22
Gambar 5.2 Skema hasil konfigurasi yang dilakukan terraform..... 24
Gambar 5.3 Hasil eksekusi konfigurasi IaC untuk host server 25
Gambar 5.4 Struktur direktori berkas yang memuat konfigurasi IaC untuk service pendukung. 26
Gambar 5. 5 Hasil eksekusi konfigurasi IaC untuk aplikasi pendukung CI/CD pipeline 27
Gambar 5.6 Hasil deployment aplikasi DroneCI menggunakan Terraform 28
Gambar 5.7 Hasil deployment aplikasi Minio menggunakan Terraform 28
Gambar 5.8 Hasil deployment aplikasi Harbor menggunakan Terraform 29
Gambar 5.9 Hasil deployment aplikasi ArgoCD menggunakan Terraform 30
Gambar 5.10 Konfigurasi Dockerfile untuk stage pertama proses build image. .. 31
Gambar 5.11 Konfigurasi Dockerfile untuk stage kedua proses build image..... 32
Gambar 5.12 Struktur direktori berkas yang berisikan template untuk melakukan deployment aplikasi. 33
Gambar 5.13 Runner script didalam container image untuk melakukan commit otomatis kedalam repository. 34
Gambar 5.14 Menunjukkan hasil commit dari service git commiter pada repository GitOps 35
Gambar 5.15 Konfigurasi Repository Helm Chart dan GitOps di Dashboard ArgoCD 36
Gambar 5.16 Konfigurasi Projects Untuk Digunakan Oleh ArgoCD..... 37

Gambar 5. 17 Konfigurasi Pipeline Untuk Integrasi Source Code Pada Event Push dan Pull Request..... 38
Gambar 5.18 Konfigurasi Pipeline Untuk Melakukan Push Container Image Ke Container Registry 39
Gambar 5.19 Konfigurasi Pipeline Untuk Melakukan Trigger Commit Environment Staging..... 40
Gambar 5. 20 Konfigurasi Pipeline Untuk Melakukan Trigger Commit Environment Production 41
Gambar 5.21 Aplikasi kompres gambar sebelum dilakukan penambahan fitur baru 42
Gambar 5.22 Continuous Integration sedang dijalankan untuk penambahan fitur baru..... 43
Gambar 5.23 Membuat pull request untuk menggabungkan fitur baru kedalam branch staging 44
Gambar 5.24 Merge fitur baru kedalam branch staging setelah semua test berhasil 45
Gambar 5.25 DroneCI melakukan proses testing ulang, build container image dan deployment versi baru kedalam environment staging..... 46
Gambar 5.26 ArgoCD melakukan deployment otomatis setelah DroneCI mengupdate konfigurasi di dalam repository GitOps 46
Gambar 5.27 Aplikasi dengan fitur baru telah terdeploy di environment staging 47
Gambar 5.28 Aplikasi kompres gambar dalam environment production sebelum dilakukan promotion staging deployment..... 48
Gambar 5.29 Pull Request dari branch staging ke production untuk melakukan promotion deployment. 49
Gambar 5.30 DroneCI melakukan proses testing ulang, build container image dan deployment versi baru kedalam environment production..... 49
Gambar 5.31 Aplikasi dengan fitur baru telah terdeploy di environment production 50
Gambar 5.32 Tampilan history deployment yang dilakukan dalam branch production. 51

Gambar 5.33 Aplikasi di environment production setelah dilakukan rollback deployment. 52
Gambar 5.34 Dashboard DroneCI untuk repository kompres gambar. 53

DAFTAR TABEL

Tabel 2.1 Tabel Perbandingan Penelitian..... 9
Tabel 4.1 Tabel spesifikasi virtual machine yang digunakan 17
Tabel 5.1 Tabel deskripsi berkas dalam direktori konfigurasi IaC 23
Tabel 5.2 Tabel hasil waktu eksekusi untuk proses integrasi source code melalui pull request 54
Tabel 5.3 Tabel hasil waktu eksekusi untuk proses deployment aplikasi ke environment staging ataupun production. 55
Tabel 5.4 Tabel perbandingan proses pengujian hingga perilisan sebelum dan sesudah implementasi CI/CD pipeline 57

INTISARI

Dalam pengembangan sebuah perangkat lunak terdapat siklus yang bernama *Software Development Life Cycle* (SDLC). SDLC mencakup proses analisis kebutuhan, *planning*, *development, testing* dan *deployment*. Masalah umum yang terjadi sering kali berpusat dalam fase *development* seperti perubahan *requirement* atau *bug fixing.* Masalah selain dalam fase *development* adalah pada fase *deployment*. Fase *deployment* merupakan fase krusial dimana merupakan fase terakhir sebelum kembali ke awal siklus. Keterlambatan perilisan perangkat lunak sering terjadi karena ketergantungan perilisan terhadap *human input* ataupun kesulitan yang timbul dalam implementasi versi baru perangkat lunak. Melihat hal tersebut sebagai suatu permasalahan yang dapat di teliti dan di tuliskan pemecahan masalahnya. Penelitian ini menggunakan salah satu proyek pengembangan perangkat lunak yang telah selesai dikembangkan. Perangkat lunak tersebut merupakan *microservice* yang ditulis dalam bahasa pemrograman Go dengan tujuan sebagai service kompres gambar. Proyek tersebut akan di teliti dan akan mengimplementasikan proses *continuous integration/continuous deployment* (CI/CD). Hasil yang di dapatkan dari proses penelitian tersebut adalah sebuah hasil analisis, konsep dan perbandingan yang dapat di simpulkan dari keuntungan implementasi CI/CD. Berdasarkan hasil penelitian implementasi CI/CD yang dilakukan dapat diperoleh penghematan waktu pengembang sebanyak 8 menit 37.8 detik untuk proses *deployment.* Selain itu didapatkan penghematan waktu sebanyak 1 menit 43.1 detik untuk proses pengujian kode. Apabila proses tersebut diakumulasi selama satu tahun maka akan terdapat reduksi waktu sebanyak 34.52 jam untuk dialokasikan pengembang ke tugas yang lebih produktif.

Kata Kunci : *azure, ci/cd, microservice, automation*

Dosen Pembimbing I : Joanna Ardhyanti Mita N, S.Kom., M.Kom.

Dosen Pembimbing II : Dr. Andi Wahju Rahardjo Emanuel, BSEE., MSSE

BAB I
PENDAHULUAN

A. Latar Belakang

Dalam siklus pengembangan perangkat lunak, tahap terakhir yang dilakukan agar perangkat lunak dapat di gunakan oleh *user* adalah *software deployment*. [1] Menurut riset yang dilakukan oleh *DevOps Research and Assesment* (DORA) pada tahun 2018 dan 2019, *software deployment* merupakan salah satu tolak ukur pengukuran *software delivery and operational* (SDO) *performance*. SDO *performance* adalah alat ukur yang di teliti oleh DORA untuk memberikan tingkat ukur terhadap efektifitas praktek *deployment* dan *delivery*. Dalam riset tersebut, *continuous integration* (CI), *continuous deployment* (CD) dan *continuous delivery* merupakan salah satu relasi prediktif untuk hasil SDO *performance*. [2] Dalam pengembangan perangkat lunak, proses *continuous integration* (CI) merupakan proses pengabungan atau penambahan fitur atau kode kedalam *repository* utama. Sedangkan proses *continuous deployment* (CD) adalah proses yang dilakukan secara otomatis dijalankan setelah proses *build* dan *test* berhasil dalam CI. Proses tersebut akan melakukan perilisan otomatis hasil *build* kedalam lingkungan *staging* atau produksi.

Keuntungan yang di dapat dalam mengadaptasi sebuah *CI/CD pipeline* adalah proses perilisan perangkat lunak menjadi lebih efisien dan konsisten. Selain itu keuntungan lain adalah dari segi waktu pengembang dapat menjadi lebih efisien tanpa harus melakukan proses perilisan secara manual. Dengan menghemat waktu tersebut maka sumber daya manusia yang sebelumnya terpakai untuk proses manual tersebut dapat di alihkan kepada kebutuhan yang lebih penting. Kelemahan dari sistem ini mengharuskan perusahaan mengeluarkan waktu dan biaya ekstra di awal untuk melakukan integrasi otomasi tersebut.

Menurut riset yang sama pada tahun 2021, organisasi yang melakukan transformasi *DevOps* dengan mengadaptasi *continuous delivery* cenderung memiliki proses kerja yang lebih berkualitas, memiliki resiko rendah dan hemat waktu dan biaya. [3] Penghematan yang di dapat dari segi waktu adalah, waktu *developer* dapat di alihkan ke tugas yang lebih penting. Penghematan biaya didapatkan dari peningkatan efektifitas waktu *developer.* Selain dari hasil riset yang dilakukan DORA, kondisi lapangan saat ini masih banyak pihak yang melakukan proses *deployment* dengan cara manual. Cara manual tersebut antara lain: *pull repository* maupun unggah berkas *build* dari *source code.* Berdasarkan hasil riset dan pengamatan tersebut, diperlukan sebuah solusi berupa implementasi *continuous integration*, *deployment* dan *delivery* untuk sebuah proyek perangkat lunak. Untuk memenuhi tolak ukur penilaian SDO *performance* maka akan dibuat sebuah implementasi CI/CD dengan objek penelitian sebuah *microservices*. *Microservices* tersebut merupakan sebuah sistem yang bertujuan untuk melakukan kompres gambar melalui REST API. Dengan menerapkan konsep CI/CD tersebut, maka akan dibuat sistem integrasi otomatis (CI) dan *deployment* otomatis (CD) untuk menghemat waktu, biaya dan resiko dalam proses pengembangan selanjutnya.

B. Rumusan Masalah

Rumusan masalah menjadi pokok permasalahan yang akan dijawab dalam penelitian ini. Rumusan masalah ini diturunkan dari latar belakang yang telah di tulis.

1. Bagaimana implementasi proses pengembangan perangkat lunak yang terdiri dari proses integrasi *source code*, *testing*, dan *deployment* perangkat lunak secara manual menjadi otomatis?
2. Bagaimana membangun sebuah solusi CI/CD *pipeline* menggunakan perangkat sumber terbuka?
3. Bagaimana dampak otomasi yang diperoleh yang ditinjau dari segi waktu, biaya dan resiko?

C. Batasan Masalah

Agar fokus penelitian tidak menyimpang dari rumusan yang telah di buat, maka batasan masalah pada penelitian ini adalah sebagai berikut:

1. Penelitian tidak membahas proses pengembangan perangkat lunak melainkan hanya proses pengembangan dan implementasi CI/CD.
2. Penelitian hanya membahas secara garis besar mengenai fungsi dari perangkat lunak yang digunakan sebagai objek penelitian sebagai informasi konteks bagi pembaca.
3. Layanan penyedia komputasi awan pihak ketiga yang digunakan adalah Microsoft Azure dengan memanfaatkan *education license* yang didapatkan bersama email students UAJY.
4. Penelitian yang dikembangkan terbatas terhadap bahasa pemrograman dan layanan pihak ketiga seperti: Bahasa Pemrograman Go, *Code Repository* GitHub, *Continuous Integration* Drone.io dan *Continuous Deployment* Agro CD.
5. Sistem orkestrator yang digunakan adalah Kubernetes.
6. Implementasi menggunakan bahasa pemrograman dan layanan pihak ketiga lain dapat dilakukan dengan modifikasi namun tidak menjamin keberhasilannya.

D. Tujuan Penelitian

Penelitian ini dilakukan dengan tujuan sebagai berikut:

1. Memberikan rancangan perubahan proses pengembangan perangkat lunak yang tediri dari integrasi *source code*, *testing* dan *deployment* menggunakan CI/CD sehingga proses yang sebelumnya dilakukan secara manual menjadi proses otomatis.
2. Memberikan hasil implementasi berupa kode tentang penggunaan perangkat lunak sumber terbuka untuk membangun sebuah CI/CD *pipelines.*

3. Memberikan analisis dampak otomasi yang diperoleh yang ditinjau dari segi waktu, biaya dan resiko.

E. Metode Penelitian

a. Studi Literatur

Studi literatur dilakukan untuk mendapatkan referensi yang berkaitan dengan proses penelitian dan penulisan tugas akhir. Materi referensi yang dikumpulkan adalah materi yang berkaitan dengan proses *CI/CD pipelines*. Sumber referensi yang digunakan dalam melakukan studi literatur adalah: artikel, jurnal, penelitian terdahulu dan buku.

b. Analisis Kebutuhan Penelitian

Tahap ini merupakan tahapan yang dilakukan untuk menganalisis kebutuhan dasar yang harus disiapkan sebelum memulai penelitian. Analisis kebutuhan penelitian dilakukan dengan mencari kebutuhan seperti *tools* yang digunakan. Dengan adanya analisis kebutuhan proses penelitian akan memiliki kerangka sebagai acuan penelitian.

c. Pengembangan Penelitian

Setelah mengumpulkan kebutuhan penelitian, akan dilanjutkan pembuatan implementasi pengembangan CI/CD. Tahapan ini dilakukan dengan mengikuti kerangka penelitian yang telah di rumuskan.

d. Pengujian Penelitian

Tahap pengujian adalah tahapan untuk memastikan implementasi yang telah di lakukan di tahap pengembangan dapat dijalankan dengan baik. Tahapan ini akan menguji keberhasilan implemetasi dan pengujian efisiensi otomasi.

e. Evaluasi Penelitian

Tahap evaluasi adalah tahapan yang berisi *feedback* dan hasil yang didapat dari tahap pengujian. Hasil yang di dapat dalam tahap

pengujian akan di analisis. Pada tahap ini juga akan dilakukan evaluasi terkait hasil analisis yang di dapat.

F. Sistematika Penulisan

Penulisan tugas akhir ini dilakukan dengan menyusun bagian penulisan menjadi 6 (enam) bab.

BAB I PENDAHULUAN

Bab ini berisi latar belakang, rumusan masalah, tujuan penelitian, manfaat penelitian, batasan penelitian, metode penelitian dan sistematika penulisan.

BAB II TINJAUAN PUSTAKA

Bab ini berisikan penelitian-penelitian terdahulu yang menyangkut dengan penelitian yang dilakukan. Dalam bab ini terdapat juga table perbandingan antara penelitian yang dilakukan dengan penelitian-penelitian terdahulu.

BAB III LANDASAN TEORI

Bab ini berisikan penjelasan mengenai teori-teori yang menyangkut penelitian yang dilakukan.

BAB IV ANALISIS DAN PERANCANGAN SISTEM

Bab ini berisi analisis dan perancangan sistem yang di kembangkan.

BAB V IMPLEMENTASI DAN PENGUJIAN SISTEM

Bab ini berisi hasil implementasi dan pengujian dari system yang telah di rancang sebelumnya.

BAB VI PENUTUP

Bab ini berisi kesimpulan yang diambil dari penelitian yang dilakukan serta saran untuk membangun pengembangan selanjutnya.

BAB II
TINJAUAN PUSTAKA

Bab ini mengulas perbandingan antara penelitian yang dilakukan dengan penelitian yang telah dilakukan sebelumnya. Perbandingan ini dilakukan untuk memberi acuan penulisan. Selain itu perbandingan ini memberikan penjelasan tentang keunikan penelitian yang ditulis.

Penelitian pertama dilakukan oleh Ramesh Ghimire yang berjudul *Deploying Software in the Cloud with CI/CD Pipelines* pada tahun 2020. Penelitian ini bertujuan untuk memberikan kesempatan kepada pengembang kecil untuk dapat mengimplementasikan penggunaan CI/CD dalam praktek *DevOps* yang dilakukan. Penelitian tersebut dilakukan dengan objek penelitian berupa aplikasi sederhana untuk menampilkan *hello, world!* melalui web dengan Flask. CI/CD *pipelines* yang dibangun menggunakan *repository* dari Bitbucket dan proses CI menggunakan Google Cloud Platform Cloud Build. Untuk proses *deployment* dilakukan dalam bentuk *container image* dengan menggunakan AWS CodeDeploy yang telah di *install* di AWS EC2. [4]

Penelitian kedua merupakan penelitian yang dilakukan oleh Tohirin, Sri Farida Utam, Septian Rheno Widianto, dan Widhy Al Mauludyansah. Penelitian ini memiliki judul Implementasi *DevOps* Pada Pengembangan Aplikasi *e-Skrining* COVID-19. Penelitian ini dilakukan dengan tujuan untuk adopsi *DevOps* pada studi kasus pengembangan perangkat lunak aplikasi *e-Skrining* COVID-19. Dalam penelitian ini digunakan beberapa teknologi pendukung seperti *code repository* oleh GitLab. Untuk proses CI dilakukan oleh Gitlab CI yang mana juga sekaligus menjalankan proses *deployment* dengan menggunakan skrip khusus yang telah dibuat menyesuaikan kebutuhan *deployment* pada server *on premise*. [5]

Penelitian ketiga dilakukan oleh Sendy Ferdian, Tjatur Kandaga, Andreas Widjaja, Hapnes Toba, Ronaldo Joshua dan Julio Narabel pada tahun 2021. Penelitian ini dilakukan dengan tujuan untuk memberikan desain *platform* efektif untuk CI/CD yang mengakomodasi kompilasi kode, eksekusi kode dan perilisan secara otomatis. Objek penelitian yang digunakan dalam pembangunan CI/CD

pipeline adalah aplikasi proyek mahasiswa dari Universitas Kristen Maranatha. Dalam penelitian ini digunakan beberapa teknologi pendukung seperti *code repository* oleh GitHub. Untuk proses CI dilakukan oleh Jenkins dan TeamCity. Tetapi dalam penelitian ini tidak di jelaskan dengan rinci bagaimana proses perilisan tersebut ke server. [6]

Penelitan keempat berjudul Implementasi CI/CD Dalam Pengembangan Aplikasi *Web* Menggunakan *Docker* dan *Jenkins* pada tahun 2021. Penelitian ini ditulis oleh Andrian Alperi dan Muhammad Arif Fadhly Ridha. Penelitian ini memiliki tujuan mengimplemntasikan dan menganalisa kualitas, waktu yang dibutuhkan dan proses otomasi pada pengembangan aplikasi *web* dengan metode CI/CD menggunakan *Docker* dan *Jenkins*. Objek penelitian yang digunakan dalam penelitian ini adalah Aplikasi bernama Elaeis. Dalam penelitian ini digunakan teknologi *code repository* GitHub. Untuk proses CI yang dilakukan dengan bantuan Jenkins. Perilisan yang dilakukan berbentuk *container image* namun tidak di jelaskan dengan rinci bagaimana proses perilisan tersebut ke server. [7]

Penelitian kelima dilakukan oleh Akbar Dhany Widiyanto, Benediktus Anindito dan Moh. Noor Al Azam. Penelitian berjudul *Implementation of Docker and Continuous Integration/Continuous Delivery for Management Information System Development*. Penelitian ini memiliki tujuan mengintegrasikan desain sistem yang menggunakan praktik CI/CD dengan implementasi menggunakan *Docker Container Platform*. Sistem tersebut akan mengintegrasikan antara pengembang dan memungkinkan perilisan *bug fix* dan fitur kedalam perangkat lunak secara otomatis tanpa menunggu lama. Dalam penelitian ini digunakan beberapa teknologi pendukung seperti *code repository* oleh Gitea. Untuk proses CI dilakukan oleh Jenkins yang mana juga sekaligus menjalankan proses *deployment* dengan melakukan perilisan dalam bentuk *container image* yang akan di *pull* setiap kali ada perilisan baru kedalam server. [8]

Berdasarkan penelitian terdahulu maka akan dibuat implementasi untuk mengubah proses integrasi, *deployment* dan perilisan perangkat lunak secara manual menjadi otomatis. Setelah itu akan dilakukan anaisis dampak yang di peroleh dari proses otomasi ditinjau dari segi waktu, biaya dan resiko. Harapan yang

timbul dari penilitian ini adalah, diharapkan dengan implementasi tersebut waktu untuk melakukan integrasi hingga perilisan dapat di hemat dan biaya serta resiko yang menyertai proses tersebut menjadi lebih efisien dan rendah.

Tabel 2.1 Tabel Perbandingan Penelitian

Pembanding	**Ramesh Ghimire [4]**	**Tohirin, dkk [5]**	**Sendy Ferdian, dkk [6]**	**Andrian Alperi, dkk [7]**	**Akbar Dhany Widiyanto, dkk [8]**	***) Vriyas Hartama Adesaputra**
Judul Penelitian	*Deploying Software in the Cloud with CI/CD Pipelines*	Implementasi *DevOps* pada Pengembangan Aplikasi e-Skrining Covid-19	*Continuous Integration and Continuous Delivery Platform Development of Software Engineering and Software Project Management in Higher Education*	Implementasi CI/CD Dalam Pengembangan Aplikasi Web Menggunakan Docker dan Jenkins	*Implementation of Docker and Continuous Integration/Continuous Delivery for Management Information System Development.*	Implementasi *Continuous Integration/Continuous Deployment* Pada *Microservice* Kompres Gambar
Objek Penelitian	Aplikasi Web Hello World Menggunakan Flask	Aplikasi e-Skrining Covid-19	Aplikasi Proyek Mahasiswa Universitas Kristen Maranatha	Aplikasi Elaeis	Aplikasi Manajemen Sistem Informasi Universitas Narotama	Aplikasi *Microservice* Untuk Kompresi Gambar
Tujuan Perilisan Aplikasi	Server Linux AWS EC2 Instance	Server Linux *On Premise*	Tidak melakukan perilisan aplikasi	Server Linux *On Premise*	Server Linux *On Premise*	Server Linux Microsoft Azure Virtual Machine

Bentuk rilisan aplikasi	*Container Image* dengan AWS CodeDeploy untuk perilisan otomatis	*Source Code* dengan skrip khusus untuk perilisan otomatis	Tidak melakukan perilisan aplikasi	*Container Image* tanpa menjelaskan bagaimana proses perilisan otomatis	*Container Image* dengan Jenkins untuk perilisan otomatis	*Container Image* dengan ArgoCD untuk perilisan otomatis
Teknologi pendukung yang digunakan	BitBucket, GCP Cloud Build, AWS CodeDeploy, AWS dan Docker	GitLab sebagai *code repository*, Gitlab CI sebagai proses CI dan CD	GitHub, Jenkins dan TeamCity	Git, Jenkins dan Docker	Gitea, Jenkins dan Docker	GitHub, Drone.io, ArgoCD, Harbor, Minio, Kubernetes
Tujuan	Memberikan kesempatan kepada pengembang software terutama yang memiliki tim kecil untuk dapat mengimplementasi penggunaan *CI/CD Pipelines* dan praktik *DevOps*	Adopsi DevOps pada pengembangan aplikasi e-Skrining Covid-19 dengan menggunakan GitLab.	Memberikan desain platform efektif untuk *continuous integration* dan *continuous delivery* untuk mengakomodasi kompilasi kode, analisis kode, eksekusi kode dan perilisan secara otomatis.	Melakukan implementasi dan menganalisa kualitas, waktu yang dibutuhkan dan proses otomasi pada pengembangan aplikasi *web* dengan metode CI/CD menggunakan *Docker* dan *Jenkins*	Penelitian ini memiliki tujuan mengintegrasikan desain sistem yang menggunakan praktik CI/CD dengan implementasi menggunakan *Docker Container Platform*	Mengembangkan implementasi *CI/CD Pipelines* terhadap *microservice* kompres berkas menggunakan Drone CI dan ArgoCD.

BAB III
LANDASAN TEORI

Sub-bab ini berisikan tentang teori-teori yang dikemukakan oleh para ahli untuk mendukung proses penelitian yang dilakukan.

A. *Infrastructure as Code*

Infrastructure as Code (IaC) merupakan sebuah penerapan pendekatan berbasis otomasi pada pengelolaan infrastruktur dengan menggunakan praktek proses pengembangan perangkat lunak. Pendekatan ini menekankan konsistensi pada tugas berulang untuk penyediaan dan perubahan sistem atau konfigurasi infrastruktur. Praktek pengembangan perangkat lunak yang digunakan seperti penggunaan *version control system* (VCS). Ada beberapa hal yang dapat dicapai oleh sebuah organisasi melalui penerapan *Infrastructure as Code*. Infrastruktur IT menjadi penunjang atau pengerak perubahan dibanding sebagai rintangan. Staff IT memiliki waktu yang lebih berguna sesuai dengan kemampuan dibanding mengerjakan pekerjaan berulang tanpa perlu keahlian khusus. Organisasi mampu berkembang lebih cepat dengan melakukan penskalaan *resource* tanpa perlu IT staff karena telah terdefinisi di dalam IaC. Kegagalan dapat di atasi lebih cepat karena susunan yang konsisten dalam IaC daripada harus mengasumsikan kesalahan. Peningkatan IaC dapat dilakukan secara bertahap dan berkelanjutan. [9]

B. Prinsip *Infrastructure as Code*

Terdapat beberapa prinsip yang dapat diterapkan dalam *Infrastructure as Code*. Prinsip tersebut dapat mengatasi tantangan yang ada dalam IaC.

a. Sistem mampu direproduksi dengan mudah, prinsip ini memberikan kemampuan bahwa sistem yang dimiliki dapat di gandakan atau di bangun ulang dengan mudah.

b. Sistem dapat dibuang, prinsip ini memberikan kemudahan perubahan kedalam sistem seperti saat kegagalan atau perubahan infrastruktur karena sifat sistem yang dinamis.

c. Sistem menjadi konsisten, dengan menggunakan IaC sistem yang dibangun konsisten sehingga dalam proses reproduksi sistem kedua sistem yang digandakan tersebut harusnya identik.

d. Proses dapat diulang, prinsip ini menjadikan infrastruktur dapat dengan mudah di bangun ulang dalam kasus kegagalan infrastruktur atau migrasi.

C. *Orkestrasi Kontainer* Kubernetes

Kubernetes atau yang biasa disingkat K8s merupakan sistem orkestrator kontainer sumber terbuka yang bertujuan mengelola *containerized application* yang berada di berbagai *node.* Kubernetes memberikan layanan untuk otomasi perilisan perangkat lunak, penskalaan dan manajemen. Kubernetes pertama kali di kembangkan oleh Google dan saat ini berada di bawah pemeliharaan Cloud Native Computing Foundation (CNCF). [10]

D. Bahasa Pemrograman Go

Go merupakan salah satu bahasa pemrograman yang memiliki tujuan penggunaan umum dan memiliki sintaks penulisan sederhana. [11] Go dibuat oleh Google pada tahun 2007, mulai sejak itu implementasi Go dalam setiap produk Google digunakan secara masif. Go merupakan bahasa pemrograman yang harus di kompilasi sebelum dapat di gunakan. Go merupakan bahasa pemrograman *type safe* dimana setiap inisialiasi variabel memiliki satu tipe data. Saat ini penggunaan Go sangat luas hingga pengembangan *cloud and network service, command line interface (CLI), web development* dan *development operations and site reliability.*

E. *Microsevice*

Microservice adalah sebuah komponen independen yang memiliki ruang lingkup terbatas dan memiliki kemampuan interoperabilitas terhadap komponen lain melalui komunikasi berbasis pesan

.

F. Sistem Otomasi *Build* dan *Testing* Perangkat Lunak Drone CI

Drone merupakan salah satu *platform continuous integration* moderen yang dikembangkan oleh Harness. Drone membantu pengembang melakukan implementasi otomasi dalam proses pengembangan perangkat lunak. [12]

G. ArgoCD

ArgoCD merupakan sebuah *tool* perilisan berkelanjutan berbasis *Git Ops* secara deklaratif yang dikembangkan untuk digunakan di dalam sistem orkestrator Kubernetes. [13]

H. *Continuous Integration*

Gambar 3. 1 Diagram Alir Proses CI/CD Pipeline

Continuous Integration pertama kali diperkenalkan oleh Kent Beck dalam bukunya, *"Extreme Programming Explained"*, pada tahun 1999. Konsep dasar dibalik CI adalah melakukan integrasi kode kedalam basis kode setiap kali terdapat perubahan. CI membantu pengembang dalam menggabungkan kode yang ditulis kembali kedalam *repository*. Setiap perubahan yang digabungkan kedalam *repository* akan diuji dan divalidasi mengunakan instumen yang telah di definisikan. Biasanya sistem akan menjalankan *unit test*, *integration test* dan *build process* dari sebuah proyek. Hal tersebut dilakukan untuk memastikan bahwa dalam penggabungan kode tidak terdapat perubahan yang merusak perangkat lunak. Bila CI menemukan konflik atau kegagalan dalam menjalankan integrasi maka *feedback* yang diterima pengembang menjadi lebih cepat. *Feedback* tersebut membantu pengembang dalam memperbaiki *bug* lebih cepat sejak dini. [14]

I. *Continuous Delivery*

Melanjutkan konsep CI, *continuous delivery* merupakan proses otomasi perilisan dari kode yang telah diuji di dalam CI. Tujuan dari CD adalah memastikan

bahwa basis kode yang dimiliki siap untuk dilakukan *deployment* ke *production environment* kapan saja. Dalam CD, setiap langkah yang dilakukan mulai dari penggabungan kode kedalam basis kode hingga pembuatan kompilasi perangkat lunak siap rilis mencakup hingga otomasi tes dan rilis otomatis. [14]

J. *Continuous Deployment*

Langkah terakhir dalam implementasi CI/CD adalah *continuous deployment*. Langkah ini merupakan kelanjutan dari proses *continuous delivery.* Sebagai sebuah kelanjutan, langkah ini akan melakukan perilisan otomatis kode yang telah disiapkan di dalam proses CD dan mengintegrasikan dengan *production environment.* Langkah ini sangat krusial karena mengganti sebuah perangkat lunak yang berjalan di mode *production* harus dengan tingkat kepercayaan yang tinggi. Maka langkah ini akan sangat bergantung dengan kualitas implementasi CI/CD pipeline. [14]

K. Kompresi Gambar

Kompresi atau *compression* merupakan proses pemampatan ukuran sebuah data menjadi lebih kecil dari ukuran aslinya. Kompresi gambar atau kompresi citra merupakan proses mengubah data citra dengan ukuran tertentu menjadi keluaran data dengan ukuran yang lebih kecil. Tujuan dilakukan kompresi gambar adalah untuk mengurangi redundansi data-data yang terdapat di dalam cutra sehingga data dapat disimpan dan didistribusikan secara lebih efisien.

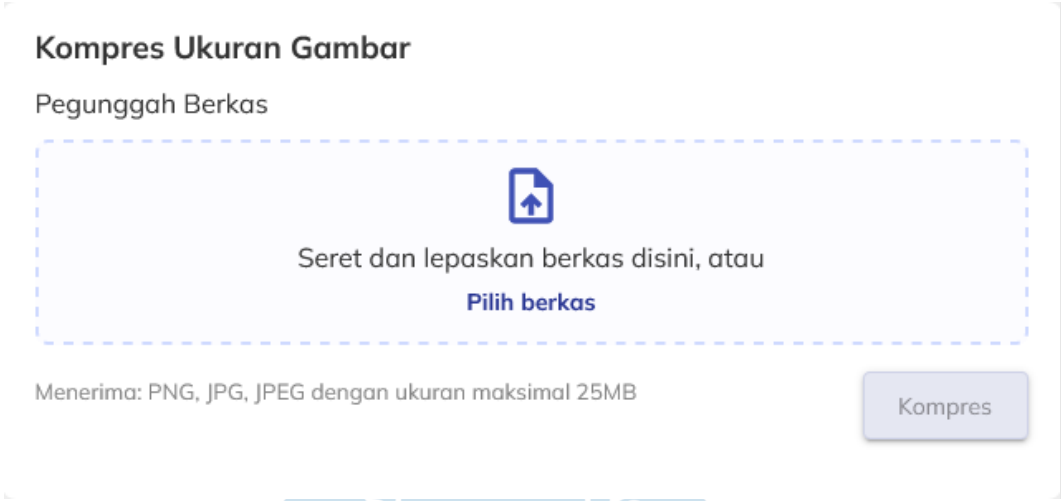
BAB IV
ANALISIS DAN PERANCANGAN SISTEM

A. Deskripsi Masalah

Dalam penelitian ini akan dilakukan perancangan dan implementasi CI/CD. Perancangan dan implementasi tersebut dilakukan untuk memberikan sebuah kerangka kerja dalam membangun CI/CD *pipeline*. Pengujian diharapkan berjalan sesuai kebutuhan dimana sebuah perangkat lunak dapat dirilis secara otomatis. Perilisan tersebut meliputi otomasi dalam proses integrasi kode, proses pengujian, proses perilisan kedalam lingkungan uji dan proses promosi perilisan menjadi perangkat lunak dalam lingkungan produksi. Selain itu proses tersebut diharapkan memberikan dampak efisiensi terhadap waktu, biaya dan pengurangan resiko kegagalan dalam perilisan perangkat lunak.

Masalah yang dihadapi adalah dalam proses perilisan perangkat lunak dari *source code* menjadi *alpha* atau *beta version* yang berada dalam lingkup *staging* masih dilakukan secara manual. Proses saat ini tersebut berimplikasi juga terhadap proses *deployment* versi dari *staging* ke *production* yang harus mengulang langkah repetitive tersebut secara berulang. Solusi yang dapat di implementasikan adalah mengintegrasikan proses *DevOps* menggunakan otomasi. Proses ini akan mencakup pembuatan *workflow* untuk *Continuous Integration* sebagai langkah awal untuk memastikan setiap integrasi kode telah memenuhi standar dan lolos uji. Langkah berikutnya adalah membangun *workflow* untuk melakukan proses otomasi perilisan perangkat lunak ke *environment staging*. Selain itu terdapat juga *workflow* untuk melakukan promosi *environment staging* yang telah lulus *quality assurance check* ke dalam *environment production.*

B. *Microservice* Kompres Gambar



Gambar 4. 1 Antar Muka Aplikasi Pengguna API Microservice Kompres Gambar

Gambar 4.1 adalah tampilan antar muka berbasis web yang menggunakan *microservice* kompres gambar untuk menjalankan fungsinya. *Microservice* yang digunakan sebagai objek penelitian adalah sebuah sistem yang telah di bangun sebelumnya. Sistem ini merupakan sebuah sistem sederhana yang memiliki *single-responsibility principle* sesuai dengan sifat *microservice.* Sistem ini bekerja untuk melakukan kompresi gambar melalui API yang dapat di gunakan oleh sistem lain.

Microservice tersebut dibangun dengan bahasa pemrograman Go. Saat ini proses perilisan *microservice* telah menggunakan Docker *Container Image* namun proses tersebut harus dilakukan secara manual. Dengan kesederhanaan aplikasi dan implementasi penggunaan *container image* yang telah teruji, yaitu sistem telah digunakan di dalam *production environment* namun proses *deployment* masih dilakukan secara manual, maka sistem ini dipilih oleh sebagai objek penelitian. Namun tidak mengurangi kemungkinan bahwa implementasi yang dilakukan akan tetap dapat diaplikasikan kedalam microservice atau sistem lain dengan mudah. Hal tersebut dikarenakan implementasi yang dilakukan bersifat *code agnostic* sehingga tidak diperlukan perubahan spesifik untuk mengakomodasi setiap sistem. Perubahan hanya dilakukan di berkas manifestasi cara *build* dan konfigurasi *perilisan.*

C. Perancangan Sistem

Sistem CI/CD yang akan dibangun terbagi menjadi dua bagian. Bagian pertama adalah bagian utama dari sistem dimana bagian ini bertangung jawab sebagai sebuah *pipeline*. Bagian kedua dari sistem adalah berkas *config* yang harus di siapkan untuk di jalankan di dalam *pipeline*. Konfigurasi tersebut terbagi menjadi dua yaitu konfigurasi untuk proses integrasi dan proses perilisan. Sistem akan dibangun menggunakan beberapa perangkat lunak pihak ketiga. Untuk memulai implementasi sistem perlu disiapkan beberapa hal. Dalam melakukan pengembangan perlu disiapkan sebuah server dengan spesifikasi teknis seperti tabel 4.1.

Tabel 4.1 Tabel spesifikasi virtual machine yang digunakan

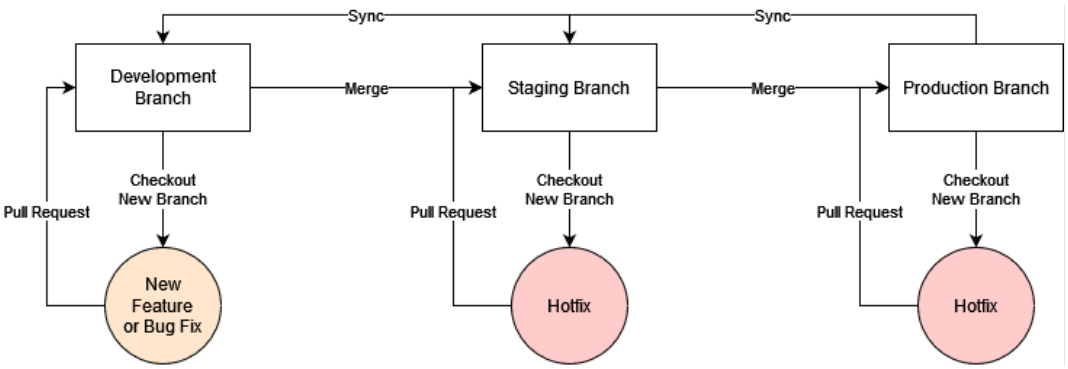
Provider	Microsoft Azure
Instance Type	D4as V5
Location	*Southeast Asia (Singapore)*
vCPU	4
RAM	16 GB
Operating System	Ubuntu 20.04

Server tersebut akan berisi semua kebutuhan perangkat lunak pendukung yang membangun CI/CD *pipeline* tersebut. Perangkat lunak tersebut terdiri dari:

1. Kubernetes sebagai sistem orkestrator semua aplikasi yang berhubungan dengan kebutuhan CI/CD *pipeline* dan *preview* atau *production release* dari aplikasi.
2. Drone CI sebagai CI server yang memiliki tujuan untuk menguji sebuah integrasi kode, melakukan *build container image* dan melakukan update konfigurasi Helm agar perubahan diketahui ArgoCD menggunakan *GitOps*.
3. Minio yaitu solusi *self-host* untuk *object storage* yang akan digunakan sebagai penyimpanan *artifacts* dan hasil *build* dari proses yang dijalankan di tahap *continuous integration.*

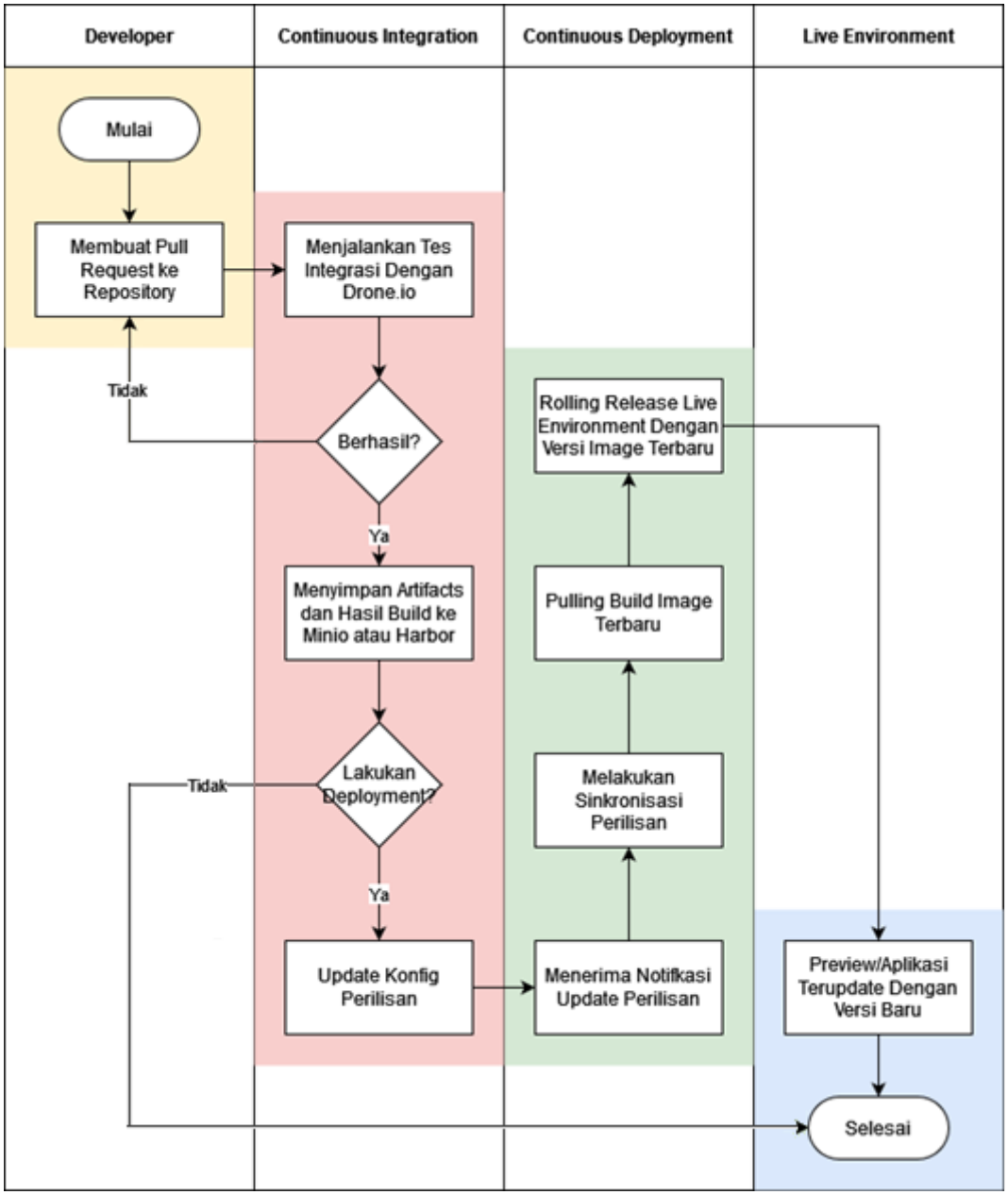
4. Harbor yaitu solusi *self-host* untuk memiliki sebuah *private container registry*. Harbor akan berintegrasi dengan Minio sebagai *storage backend* dan menyimpan *container image* yang telah di *build* pada tahap *continuous integration.*
5. Helm merupakan *tool* yang digunakan untuk memudahkan *deployment* aplikasi dalam sistem orkestrator Kubernetes.
6. ArgoCD merupakan *tool* yang akan membantu proses *continuous deployment* dari perilisan *preview*, *staging* atau *production* dari sebuah aplikasi.

Langkah selanjutnya yang dilakukan adalah melakukan *setup* Kubernetes di dalam server tersebut. Proses instalasi Kubernetes akan menggunakan k3s oleh Rancher Labs. Pertimbangan penggunaan k3s adalah kemudahan instalasi dalam *single-node* Kubernetes *cluster*. Dengan demikian apa bila akan dilakukan migrasi menggunakan *multi-node* Kubenetes *cluster* atau *cloud solution* seperti Amazon Web Service Elastic Kubernetes Service (EKS), Microsoft Azure Kubernetes Service (AKS) dan Google Cloud Platform Kubernetes Engine (GKE) konfigurasi yang digunakan dan persiapan yang dilakukan sama. Penggunaan k3s juga akan membantu proses implementasi dengan lebih mudah dimana *cost* yang harus disiapkan untuk implementasi tidak harus mengikuti implementasi *full scale.*



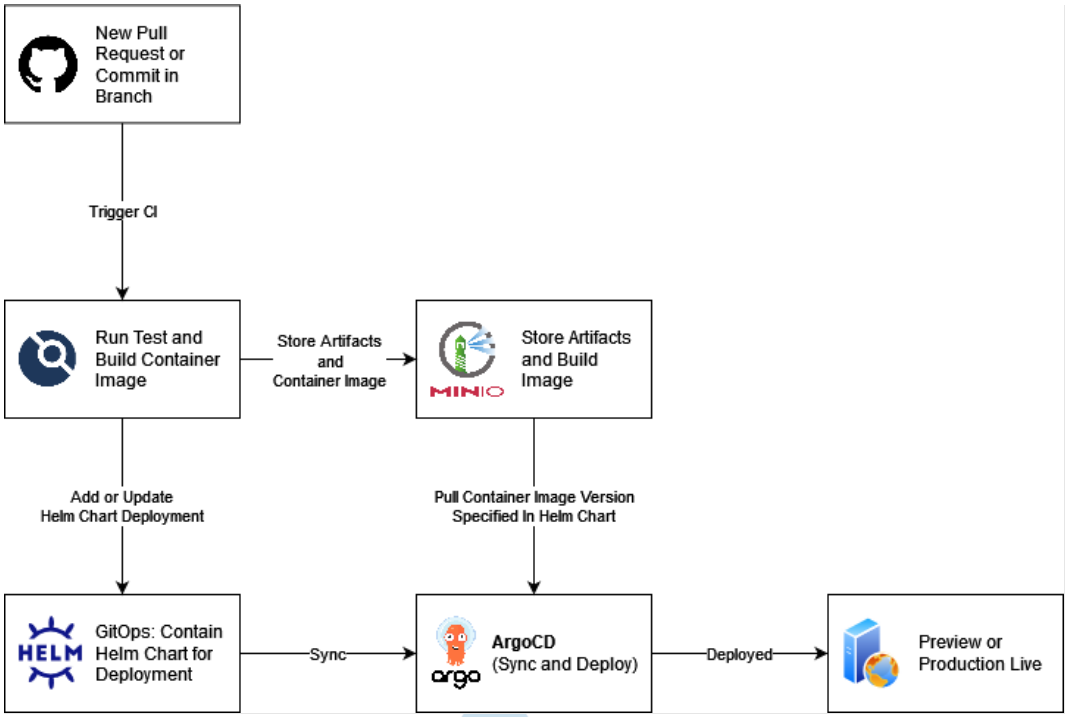
Gambar 4.2 Alur Kerja Pengembang

Gambar 4.2 adalah sebuah alur kerja yang diikuti oleh pengembang dalam melakukan perilisan fitur baru atau perbaikan *bug*. Alur kerja tersebut memiliki tiga *environment*. Setiap *environment* tersebut akan memiliki proses *continuous integration* sendiri yang memastikan bahwa setiap pengembangan yang dilakukan terintegrasi dengan benar. *Development branch* merupakan *branch* yang digunakan untuk melakukan pengembangan fitur baru atau perbaikan *bug* yang tidak mendesak. *Staging branch* merupakan *branch* yang berisi fitur-fitur baru yang telah selesai di kembangkan dan akan diuji untuk perilisan. *Production branch* adalah *branch* yang berisikan kode produksi, aplikasi yang berjalan dan digunakan oleh user berbasis pada kode di dalam *branch* ini.



Gambar 4. 3 Diagram Alir Proses Perilisan Menggunakan CI/CD Pipeline

Gambar 4.3 mengambarkan bahwa setiap *environment* yang terdapat dalam proses pengembangan perangkat lunak tersebut akan memiliki diagram alur pengembangan. Pada gambar teresbut dijelaskan alur proses dari sebuah fitur baru atau perbaikan *bug* yang dilakukan hingga menggantikan versi yang telah berjalan di *production* atau *staging*.



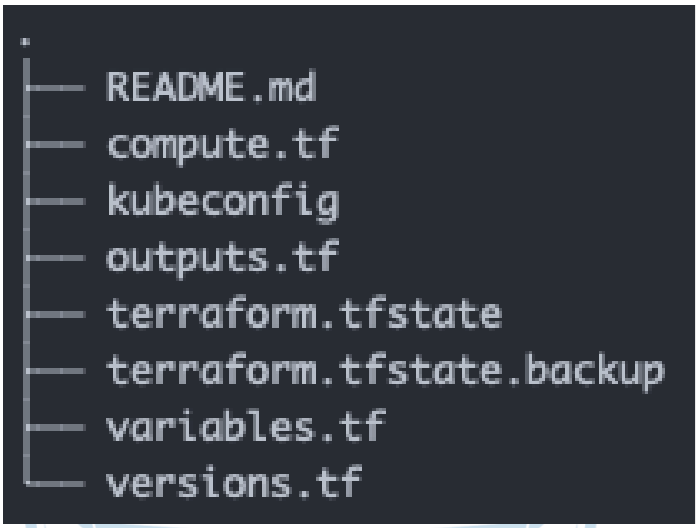
Gambar 4. 4 Diagram Arsitektur Continuous Integration dan Continuous Deployment

Arsitektur CI/CD yang digunakan dapat di lihat pada gambar 4.4. Pada gambar tersebut digambarkan sebagai sebuah alur dimana langkah dimulai saat pengembang menambahkan kode baru kedalam *repository* dalam bentuk *pull request.* Langkah terakhir berupa hasil perilisan dari kode yang berada di setiap *branch* dari masing-masing *environment*.

BAB V
IMPLEMENTASI DAN PENGUJIAN SISTEM

A. Implementasi Sistem

a. Implementasi IaC Untuk Konfigurasi Server



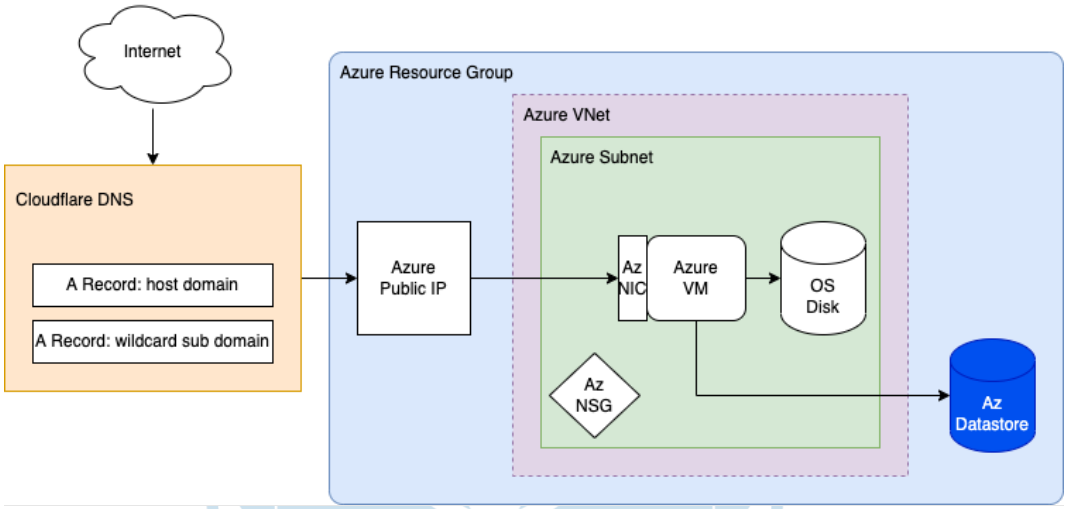
Gambar 5. 1 Struktur direktori berkas yang memuat konfigurasi IaC untuk *host machine*.

Gambar 5.1 merupakan struktur berkas yang berisi konfigurasi *Infrastructure as Code* menggunakan Terraform. Skrip tersebut memiliki tujuan untuk menyiapkan server di *cloud service provider* Microsoft Azure, melakukan instalasi Kubernetes *single cluster* dan konfigurasi DNS secara otomatis menggunakan Cloudflare.

Tabel 5.1 Tabel deskripsi berkas dalam direktori konfigurasi IaC

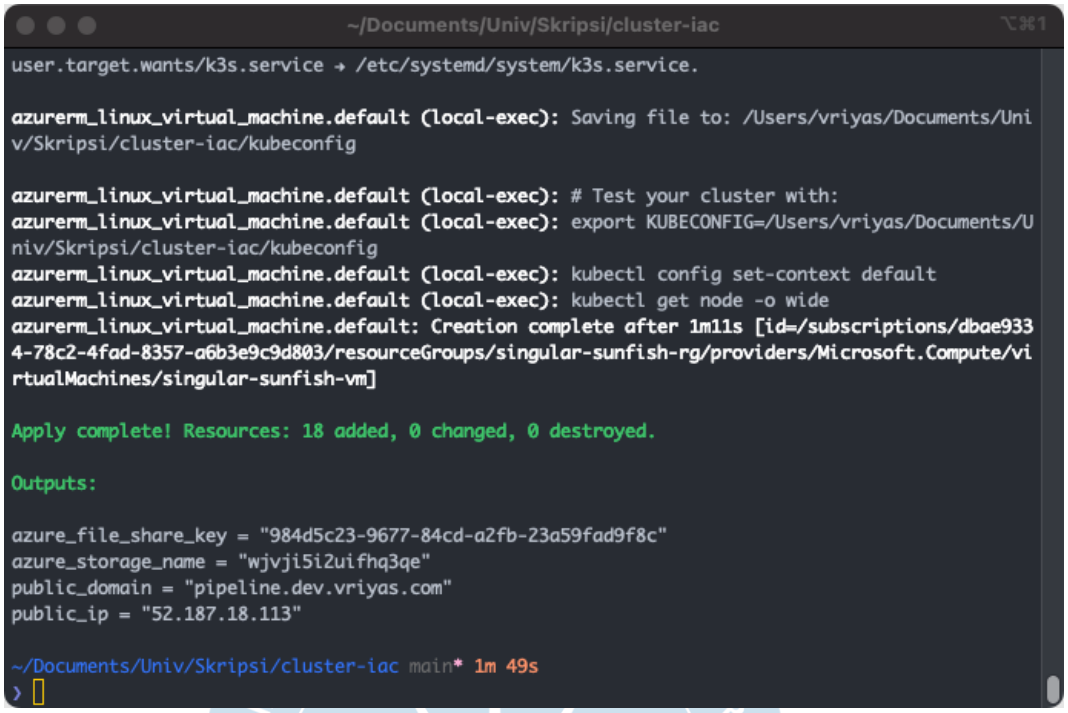
Nama Berkas	Deskripsi
`.env`	Data *key-value pairs* yang bersifat rahasia yang akan digunakan dalam proses inisialisasi.
`compute.tf`	Definisi *resource* yang akan digunakan seperti konfiguras server maupun DNS.
`kubeconfig`	Output konfig yang dihasilkan setelah berhasil menjalankan inisialisasi. Berkas ini digunakan untuk berkomunikasi dengan Kubernetes Cluster.
`outputs.tf`	Definisi output yang dikeluarkan hasil inisialisasi dalam bentuk *output print command line interface.*
`terraform.tfstate`	Hasil keluaran inisialisasi berupa sebuah file yang mengelola *current state* dari inisialisasi IaC.
`terraform.tfstate.backup`	Merupakan backup dari file utama *state* dimana file ini akan muncul setelah iterasi kedua dimana untuk menyimpan *state* iterasi pertama dan akan terus diganti sesuai iterasi.
`variables.tf`	Definisi variabel yang digunakan dalam mendefinisikan *resource.*
`versions.tf`	Definisi versi *providers* pendukung yang digunakan.

Tabel 5.1 berisi penjelasan berkas yang terdapat di dalam direktori pada gambar 5.1. Berkas-berkas tersebut merupakan komponen yang membangun IaC dengan menggunakan Terraform. Berkas tersebut terbagi menjadi beberapa bagian untuk memudahkan pengelompokan berkas sesuai dengan tujuannya.



Gambar 5.2 Skema hasil konfigurasi yang dilakukan terraform.

Gambar 5.2 menujukkan gambaran *resource* yang akan dihasilkan oleh terraform. Pada gambar tersebut dijabarkan beberapa *resource* yang akan di buat oleh IaC yang telah dibangun. Resource tersebut terdiri dari dua *service provider* yaitu Cloudflare dan Microsoft Azure. Cloudflare digunakan untuk melakukan konfigurasi DNS dari IP server yang berada di Microsoft Azure.



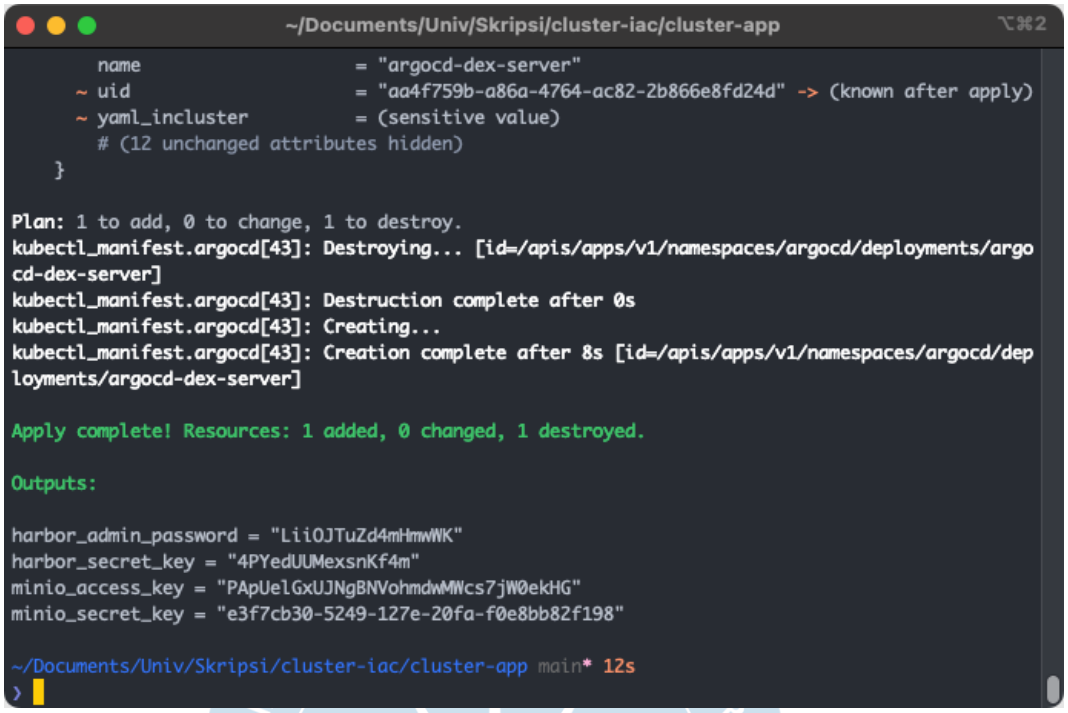
Gambar 5.3 Hasil eksekusi konfigurasi IaC untuk host server

Resource diatas akan dikonfigurasi dengan mengeksekusi skrip terraform pada gambar 5.3. Setelah konfigurasi tersebut di eksekusi Terraform akan menyimpan *state* dari konfigurasi yang dijalankan. Selain itu Terraform juga akan menampilkan *outputs* yang telah didefinisikan.*Output* tersebut antara lain adalah *output* dari alamat IP *virtual server* yang telah dibuat. Alamat DNS dari Cloudflare yang dapat digunakan untuk mengakses server. Selain *output* alamat IP dan DNS terdapat juga *output* untuk *azure_file_share_key* dan *azure_storage_name* yang akan digunakan untuk membuat *persistent volume claim* pada modul terraform untuk *service* CI/CD.

b. Implementasi IaC Pada *Service* Pendukung CI/CD Pipelines

Gambar 5.4 Struktur direktori berkas yang memuat konfigurasi IaC untuk service pendukung.

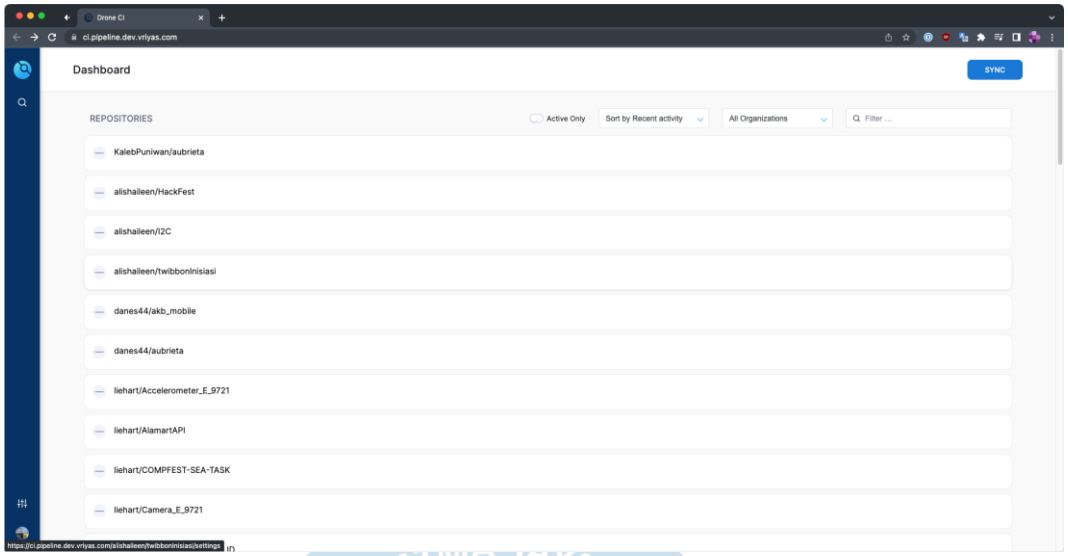
Gambar 5.4 merupakan struktur berkas yang berisi konfigurasi IaC untuk menyiapkan aplikasi pendukung di dalam server. Pemisahan konfigurasi ini dilakukan untuk menambah modularitas IaC sehingga apabila dilakukan penggantian *host machine* menggunakan *cloud service provider* lain maka tidak diperlukan konfigurasi ulang secara keseluruhan. Pada IaC *service* pendukung akan dilakukan *deployment* beberapa aplikasi seperti Traefik, Minio, Harbor, Drone CI dan ArgoCD.



Gambar 5. 5 Hasil eksekusi konfigurasi IaC untuk aplikasi pendukung CI/CD pipeline

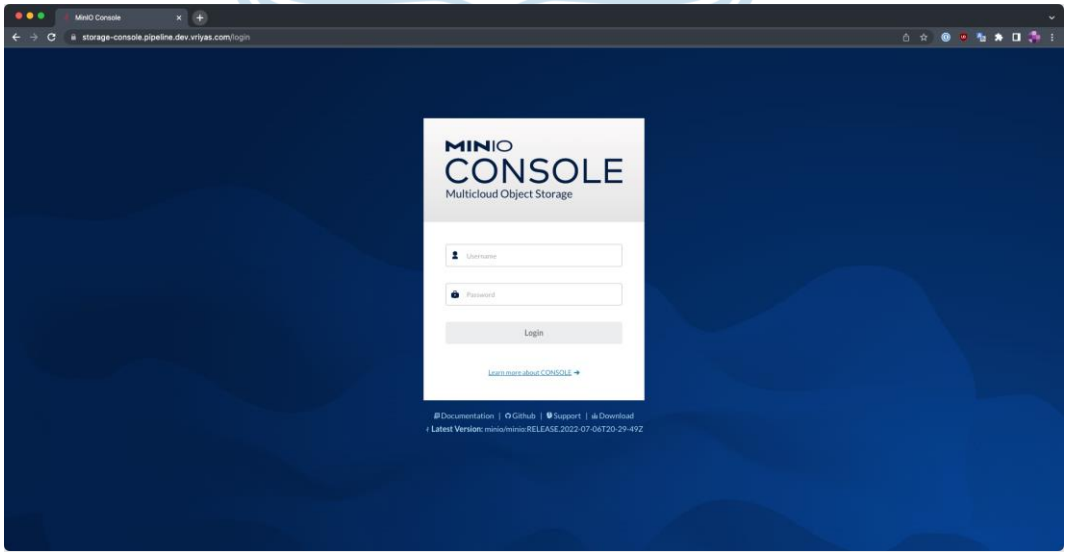
Gambar 5.5 merupakan hasil eksekusi IaC yang digunakan untuk melakukan *deployment* aplikasi pendukung CI/CD *pipeline*. Modul ini akan melakukan *deployment* beberapa *service*. *Service-service* tersebut memiliki peranan masing-masing dalam menjalankan proses CI/CD *pipeline*.

Let's Encrypt Secure Socket Layer/Transport Layer Security untuk domain utama dan sub-domain *wildcard*. SSL digunakan untuk mengenkripsi lalulintas data dari dan keluar sistem. Submodul ini akan menkonfigurasi SSL/TLS untuk domain `pipeline.dev.vriyas.com` dan `*.pipeline.dev.vriyas.com`. Traefik merupakan *ingress controller* yang digunakan *Kubernetes cluster* untuk mengelola lalulintas data serta SSL/TLS *termination.*



Gambar 5.6 Hasil deployment aplikasi DroneCI menggunakan Terraform

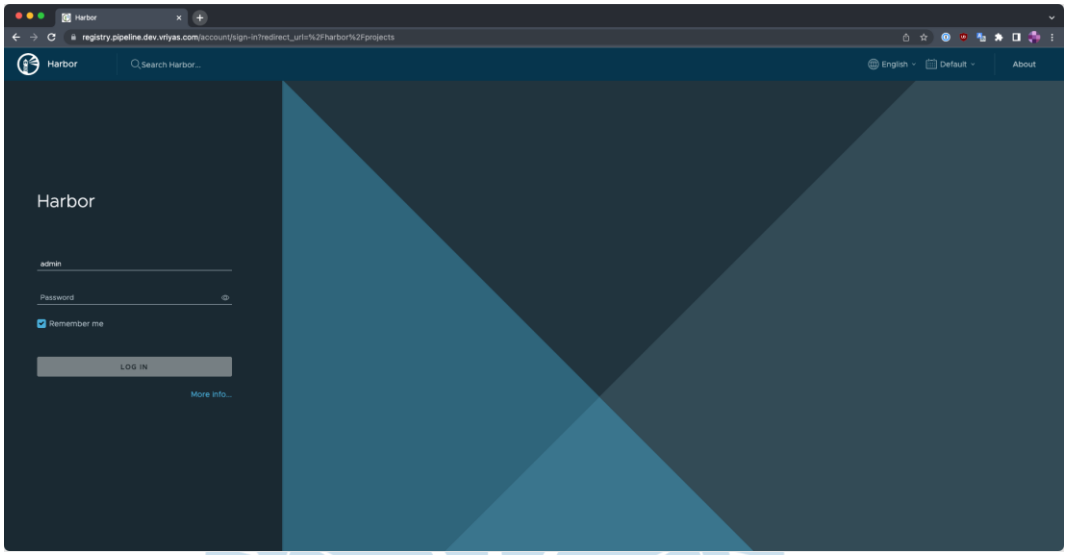
Pada gambar 5.6, DroneCI merupakan *continuous integration server* yang akan digunakan dalam membangun implementasi CI/CD *pipeline*. *Service* ini akan menjalankan sebagian besar peran sistem. Peran yang dijalankan *service* ini antara lain menjalankan konfigurasi CI yang telah dibuat didalam berkas `drone.yml`.



Gambar 5.7 Hasil deployment aplikasi Minio menggunakan Terraform

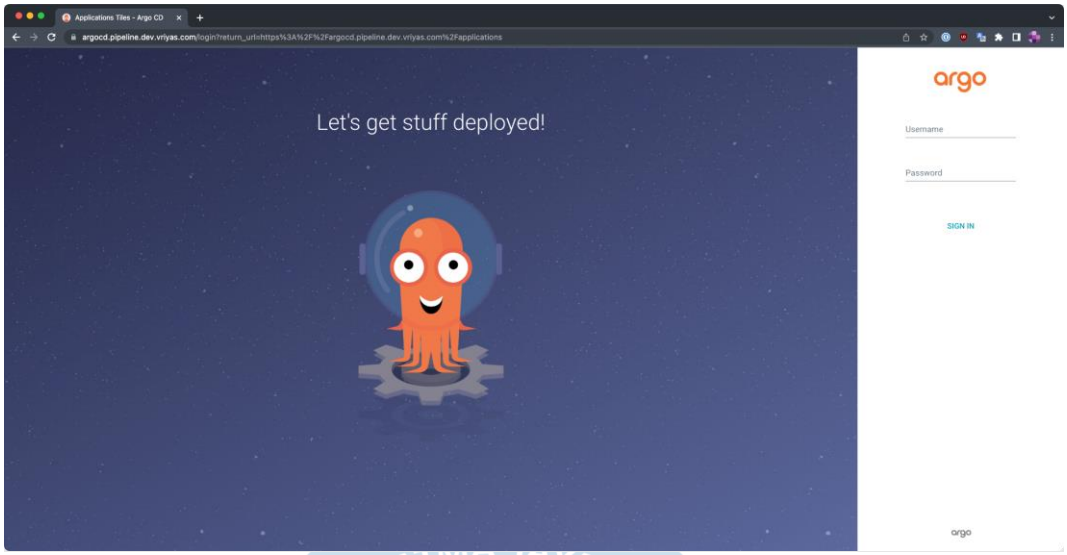
Pada gambar 5.7, MinIo yang merupakan *high performance, Kubernetes native object storage.* Secara sederhana *service* ini merupakan

service storage. Service ini akan digunakan sebagai media penyimpanan *cache* dan *build artifacts* DroneCI serta *backend storage* untuk *service container registry* Harbor.



Gambar 5.8 Hasil deployment aplikasi Harbor menggunakan Terraform

Pada gambar 5.8 Harbor merupakan *service container registry* yang digunakan untuk menyimpan semua hasil *container image* selama proses *continuous integration*. *Service* ini akan digunakan secara berkala oleh ArgoCD untuk mendapatkan *release image* yang sesuai oleh permintaan kofigurasi. Selain itu sistem ini akan menyimpan *container image* dari hasil build sebelumnya sehingga *image* tersebut tidak hilang dan dapat digunakan untuk proses *rollback deployment*.



Gambar 5.9 Hasil deployment aplikasi ArgoCD menggunakan Terraform

Pada gambar 5.9, ArgoCD merupakan *Kubernetes controller* yang memiliki peran dalam implementasi sistem sebagai *continuous deployment service.* Didalam *service* ini proses otomasi *deployment* akan dilakukan berdasarkan konfigurasi *repository* GitOps dan Helm Chart yang telah disiapkan.

c. Implementasi Dockerfile Untuk Proses *Build Container Image*

Untuk membangun sebuah *container image* yang dapat digunakan berkali-kali diperlukan sebuah proses yang bertujuan menghasilkan sebuah *image* dari aplikasi yang dibangun. Konfigurasi langkah pembangunan *image* tersebut terdapat dalam berkas bernama Dockerfile.

```
ARG GOLANG_VERSION=1.17.5
FROM golang:${GOLANG_VERSION}-buster as builder
ARG LIBVIPS_VERSION=8.12.1
RUN DEBIAN_FRONTEND=noninteractive \
    apt-get update && \
    apt-get install --no-install-recommends -y \
    build-essential pkg-config glib2.0-dev \
    libexpat1-dev libglib2.0-0 \
    libwebp-dev libjpeg62-turbo-dev libpng-dev && \
    cd /tmp && \
    curl -fsSLO
https://github.com/libvips/libvips/releases/download/v${LIBVIPS_VERS
ION}/vips-${LIBVIPS_VERSION}.tar.gz && \
    tar zvxf vips-${LIBVIPS_VERSION}.tar.gz && \
    cd /tmp/vips-${LIBVIPS_VERSION} && \
    ./configure && \
    make && \
    make install && \
    ldconfig

WORKDIR ${GOPATH}/src/github.com/liehart/file-minifier
ENV GO111MODULE=on
COPY go.mod .
COPY go.sum .
RUN go mod download
COPY . .
RUN go test -v
RUN go build -a \
    -o ${GOPATH}/bin/file-minifier
```

Gambar 5.10 Konfigurasi Dockerfile untuk stage pertama proses build image.

Pada gambar 5.10 dan gambar 5.11 merupakan isi dari berkas Dockerfile yang berisikan argumen yang harus di jalankan dalam membangun sebuah *container image*. Dalam berkas Dockerfile tersebut digunakan proses *multistage build* dimana proses dilakukan dengan membagi kedalam beberapa tahap. Tabel 5.2 merupakan proses membangun konfigurasi *build dependencies* yang digunakan aplikasi dan *compiled version* dari aplikasi. Pada tabel 5.3 merupakan proses menyatukan hasil *build* dalam tahap pertama kedalam *image* yang lebih ringan dan hanya memiliki *dependencies* minimal.

```
FROM debian:buster-slim
RUN apt-get update && apt-get install -y curl
COPY --from=builder /usr/local/lib /usr/local/lib
COPY --from=builder /go/bin/file-minifier /usr/local/bin/file-
minifier
RUN DEBIAN_FRONTEND=noninteractive \
    apt-get update && \
    apt-get install --no-install-recommends -y \
    libwebp6 libwebpmux3 libwebpdemux2 \
    libglib2.0-0 libexpat1-dev libjpeg62-turbo-dev \
    libpng-dev && \
    apt-get autoremove -y && \
    apt-get autoclean && \
    apt-get clean && \
    rm -rf /var/lib/apt/lists/* /tmp/* /var/tmp/*
ENV PORT 3000
HEALTHCHECK --interval=30s --timeout=30s --start-period=5s --
retries=3 \
    CMD curl --fail http://0.0.0.0:${PORT}/health || exit 1
ENTRYPOINT ["/usr/local/bin/file-minifier"]
EXPOSE ${PORT}
```

Gambar 5.11 Konfigurasi Dockerfile untuk stage kedua proses build image

d. Implementasi Helm Chart Untuk *Microservice* Kompres Gambar

Sebuah *repository* yang berisikan konfigurasi Helm Chart yang digunakan oleh ArgoCD dalam melakukan perilisan. *Repository* ini berisikan aplikasi apa saja yang harus di rilis secara otomatis. Setiap sub-direktori dari aplikasi yang akan dirilis berisi konfigurasi *resource definition* dari Kubernetes yang akan digunakan oleh Helm Chart. *Resource definition* tersebut terdiri dari *resource definition* untuk mengelola *deployment*, *ingress* dan *service* di dalam *cluster* Kubernetes.

Gambar 5.12 Struktur direktori berkas yang berisikan template untuk melakukan *deployment* aplikasi.

Helm Chart digunakan untuk menyimpan template konfigurasi yang harus digunakan untuk melakukan *deployment* aplikasi. Pada gambar 5.12 dapat dijabarkan bahwa dalam sebuah template Helm Chart terdapat konfigurasi YAML Kubernetes. Konfigurasi tersebut memiliki tugas masing-masing sebagaimana:

i. `_helpers.tpl` memiliki tujuan untuk menapilkan informasi saat HelmChart digunakan. Karena menggunakan ArgoCd maka penggunaan _helpers.tpl tidak relevan.

ii. `deployment.yaml` memiliki tujuan untuk melakukan *deployment* aplilkasi dengan orkestrator Kubernetes.

iii. `service.yaml` memiliki tujuan untuk mendefinisikan *resource service* di dalam *cluster*.

iv. `ingress.yaml` memiliki tujuan untuk mendefinisikan *resource ingress* dan mendefinisikan *route* yang tersedia menggunakan Traefik.

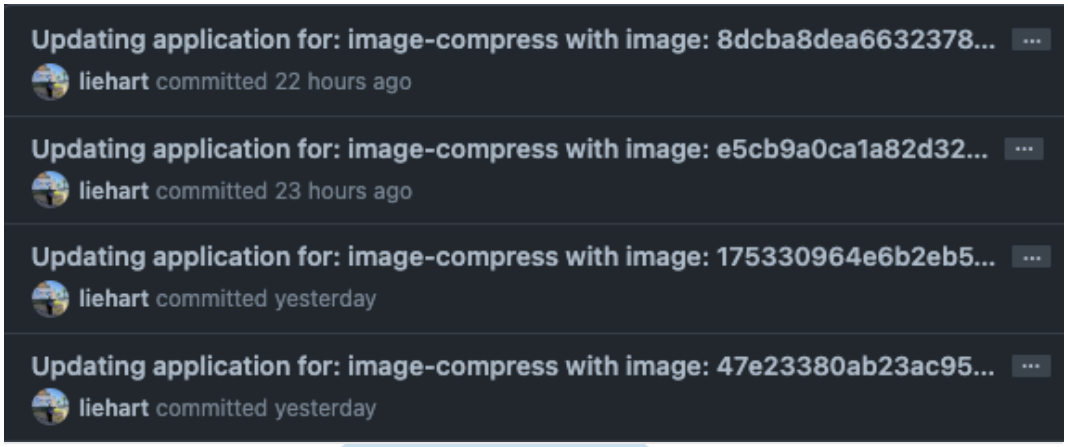
e. Implementasi *Container Image* Untuk Melakukan *Commit* Perubahan ke *Repository GitOps*

Gambar 5.13 merupakan skrip yang memiliki tujuan untuk melakukan perubahan konfigurasi secara otomatis di dalam *repository GitOps* yang telah disiapkan sebelumnya. Skrip tersebut akan dijalankan saat DroneCI sedang melakukan integrasi *commit* baru hasil *merge pull request* dari *branch* lain atau *hotfix branch*. Skrip teresbut akan melakukan *update* terutama pada `IMAGE_TAG` yang digunakan sehingga apabila terdapat versi baru pasti IMAGE_TAG berubah dan ArgoCD akan melakukan sinkronisasi konfigurasi dengan melakukan *deployment* baru ataupun lama untuk *environment* tertentu dengan IMAGE_TAG terbaru.

```
#!/bin/sh
git clone https://${GITHUB_TOKEN}@github.com/liehart/gitops.git
cd gitops
cat preview.yaml \
  | sed -e "s@{{.APP_ID}}@${APP_ID}@g" \
  | sed -e "s@{{.REPO}}@${REPO}@g" \
  | sed -e "s@{{.IMAGE_TAG}}@${IMAGE_TAG}@g" \
  | sed -e "s@{{.APP_ID}}@${APP_ID}@g" \
  | tee helm/templates/${APP_ID}.yml
git add .
git -c "user.name=${GITHUB_USERNAME}" -c
"user.email=${GITHUB_EMAIL}" commit \
  -m "Updating application for: ${REPO} with image: ${IMAGE_TAG}"
git push origin main
```

Gambar 5.13 Runner script didalam container image untuk melakukan commit otomatis kedalam repository.

f. Implementasi *Repository* GitOps Untuk Menyimpan *Deployment State*

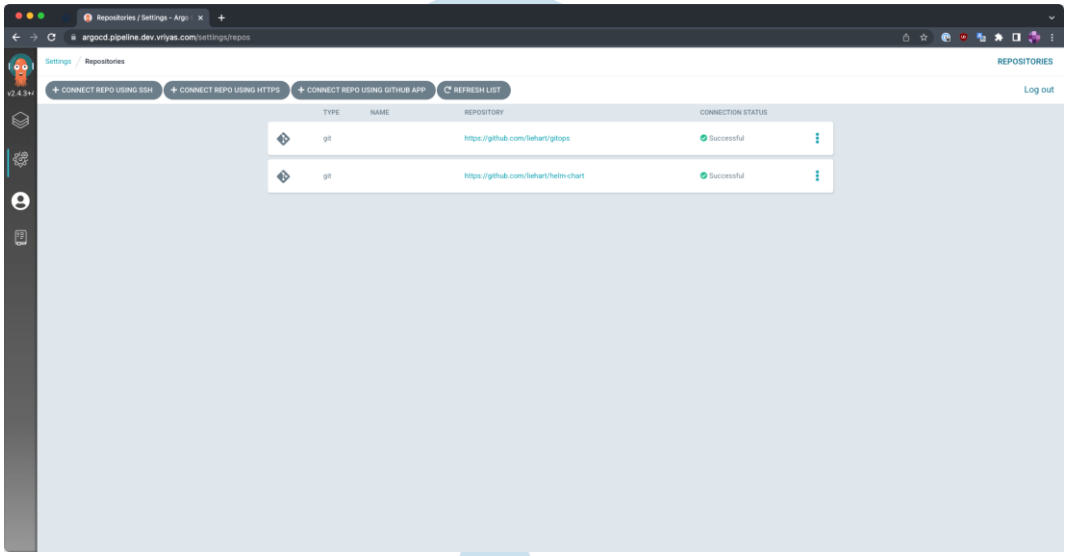


Gambar 5.14 Menunjukkan hasil commit dari service git commiter pada repository GitOps

Repository ini memiliki tujuan untuk menyimpan nilai-nilai baru yang akan digunakan *template* Helm Chart dan ArgoCD dalam melakukan sinkronsiasi konfigurasi. Perubahan data pada *repository* ini sepenuhnya dikelola secara otomatis oleh DroneCI dengan melakukan *commit* baru untuk memberitahu ArgoCD bahwa terdapat perubahan *deployment* yang harus dilakukan. Penambahan *commit* baru tersebut akan dilakukan oleh *service* bernama `git-commiter` yang sudah dikembangkan pada gambar 5.14.

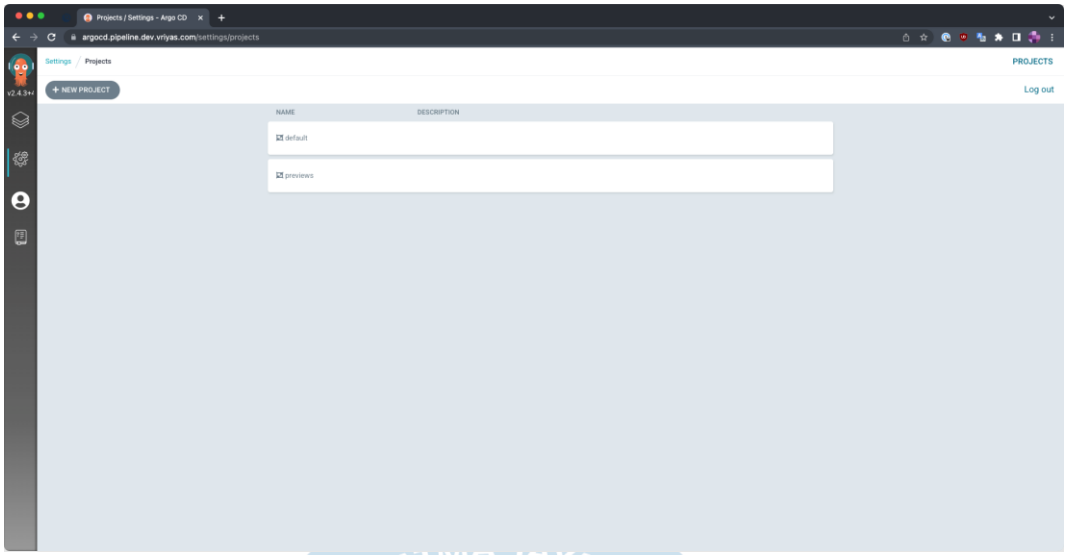
g. Konfigurasi ArgoCD Untuk Pertama Kali

Konfigurasi ini dilakukan hanya sekali pada saat berhasil melakukan *deployment* aplikasi pendukung CI/CD *pipeline*. Pada gambar 5.15 merupakan proses konfigurasi *repository* yang telah dibuat. *Repository* yang harus diberikan akses kedalam sistem ArgoCD adalah *repository* yang memuat Helm Chart untuk melakukan *deployment* aplikasi. Selain itu juga diberikan akses ke *repository* GitOps yang berisi kondisi terkini perilisan aplikasi di setiap *environment* dan parameternya.



Gambar 5.15 Konfigurasi Repository Helm Chart dan GitOps di Dashboard ArgoCD

Pada gambar 5.16 merupakan pemisahan proyek yang terdapat di dalam ArgoCD. Proyek dapat diartikan sekumpulan *deployment* yang dilakukan oleh ArgoCD. Dalam proyek *default* berisi *deployment* Helm Chart utama. Sedangkan untuk *deployment* aplikasi kompres berkas di setiap *environment* akan disimpan didalam proyek *previews*.



Gambar 5.16 Konfigurasi Projects Untuk Digunakan Oleh ArgoCD

h. Implementasi Berkas YAML Konfigurasi Drone CI

`drone.yml` merupakan sebuah berkas yang ditempatkan di *root directory* untuk repository yang akan menggunakan sistem CI dari DroneCI. Berkas tersebut berisi konfigurasi yang harus dilakukan DroneCI setiap kali mendapat *event* dari *repository*. Dalam berkas tersebut dibagi menjadi bagian-bagian yang dinamakan *pipeline*. Konfigurasi *pipeline* untuk sistem kompres berkas terbagi menjadi 4 kategori utama.

```
kind: pipeline
type: kubernetes
name: default

steps:
- name: docker
  image: plugins/docker
  settings:
    custom_dns: [ 8.8.8.8, 8.8.4.4]
    mtu: 1000
    dockerfile: Dockerfile.base
    username:
      from_secret: registry_user
    password:
      from_secret: registry_password
    repo:
registry.pipeline.dev.vriyas.com/private/${DRONE_REPO_NAME}-base
    registry: registry.pipeline.dev.vriyas.com
    tags:
      - latest
    cache_from:
      -
"registry.pipeline.dev.vriyas.com/private/${DRONE_REPO_NAME}-
base:latest"

- name: build and test
  image:
registry.pipeline.dev.vriyas.com/private/${DRONE_REPO_NAME}-
base:latest
  commands:
  - go mod download
  - go build
  - go test

trigger:
  branch:
  - master
  - staging
  - production
  event:
  - push
  - pull_request
```

Gambar 5. 17 Konfigurasi Pipeline Untuk Integrasi Source Code Pada Event Push dan Pull Request

Pada gambar 5.17, pipeline ini memiliki tujuan agar DroneCI akan melakukan build dan test untuk setiap penambahan kode baru kedalam source

code. Apabila tahap ini gagal maka tidak dapat dilakukan penggabungan kode melalui pull request.

```
kind: pipeline
type: kubernetes
name: build_push

steps:
- name: docker
  image: plugins/docker
  settings:
    custom_dns: [ 8.8.8.8, 8.8.4.4]
    mtu: 1000
    username:
      from_secret: registry_user
    password:
      from_secret: registry_password
    repo:
registry.pipeline.dev.vriyas.com/private/${DRONE_REPO_NAME}
    registry: registry.pipeline.dev.vriyas.com
    tags:
      - ${DRONE_COMMIT}
      - ${DRONE_BRANCH}
      - ${DRONE_BRANCH}-${DRONE_COMMIT}
    cache_from:
      -
"registry.pipeline.dev.vriyas.com/private/${DRONE_REPO_NAME}:master"
      -
"registry.pipeline.dev.vriyas.com/private/${DRONE_REPO_NAME}:${DRONE
_BRANCH}"

trigger:
  branch:
  - staging
  - production
  event:
  - push
```

Gambar 5.18 Konfigurasi Pipeline Untuk Melakukan Push Container Image Ke Container Registry

Pada gambar 5.18, pipeline ini memiliki tujuan untuk melakukan push container image yang telah dibangun kedalam container registry yang telah disiapkan.

```
kind: pipeline
type: kubernetes
name: update_gitops

steps:
- name: gitops commit
  pull: always
  image: registry.pipeline.dev.vriyas.com/private/git-
commiter:latest
  settings:
    custom_dns: [ 8.8.8.8, 8.8.4.4]
    mtu: 1000
  environment:
    APP_ID: ${DRONE_REPO_NAME}-${DRONE_BRANCH}
    IMAGE_TAG: ${DRONE_COMMIT_SHA}
    REPO: ${DRONE_REPO_NAME}
    GITHUB_EMAIL: ${DRONE_COMMIT_AUTHOR_NAME}
    GITHUB_USERNAME: ${DRONE_COMMIT_AUTHOR_EMAIL}
    GITHUB_TOKEN:
      from_secret: github_pull_token
  commands:
  - /app/commiter.sh
trigger:
  branch:
  - staging
  event:
  - push
depends_on:
  - build_push
```

Gambar 5.19 Konfigurasi Pipeline Untuk Melakukan Trigger Commit Environment Staging

Pada gambar 5.19, pipeline ini memiliki tujuan untuk melakukan trigger berupa penambahan commit kedalam repository GitOps. Penambahan commit tersebut akan membuat atau mengubah berkas yang sudah ada agar menggunakan IMAGE_TAG terbaru. Hal ini akan dilakukan di environment staging.

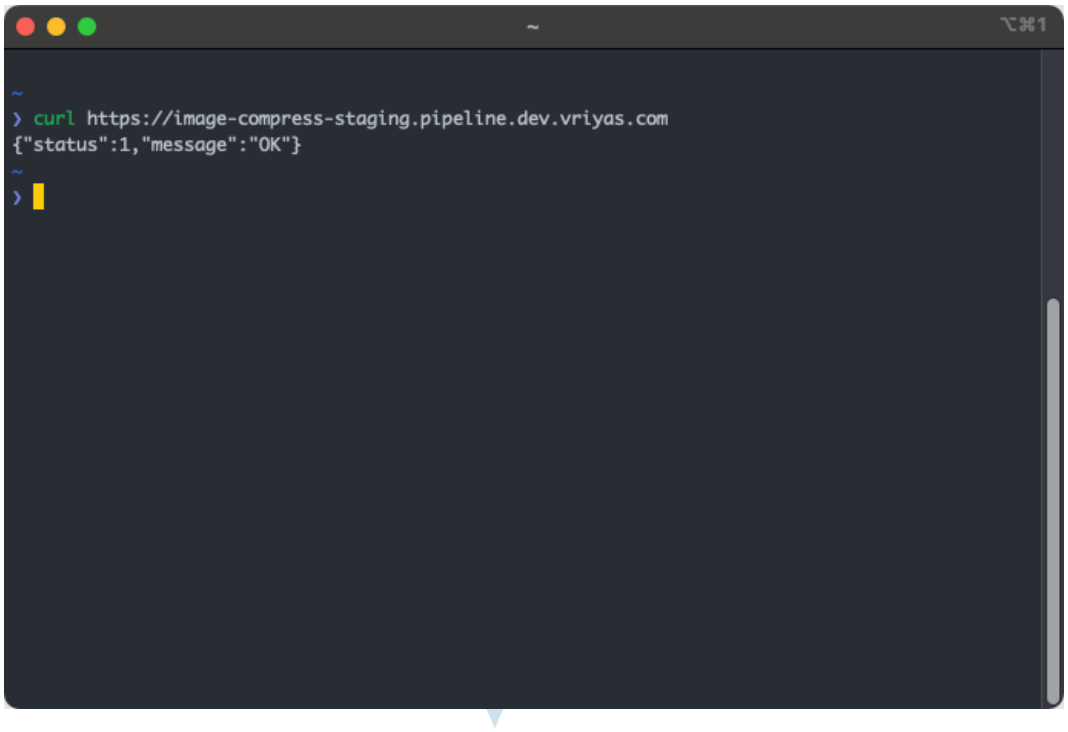
```
kind: pipeline
type: kubernetes
name: update_prod

steps:
- name: gitops commit
  pull: always
  image: registry.pipeline.dev.vriyas.com/private/git-
commiter:latest
  settings:
    custom_dns: [ 8.8.8.8, 8.8.4.4]
    mtu: 1000
  environment:
    APP_ID: ${DRONE_REPO_NAME}
    IMAGE_TAG: ${DRONE_COMMIT_SHA}
    REPO: ${DRONE_REPO_NAME}
    GITHUB_EMAIL: ${DRONE_COMMIT_AUTHOR_NAME}
    GITHUB_USERNAME: ${DRONE_COMMIT_AUTHOR_EMAIL}
    GITHUB_TOKEN:
      from_secret: github_pull_token
  commands:
  - /app/commiter.sh
trigger:
  branch:
  - production
  event:
  - push
depends_on:
  - build_push
```

Gambar 5. 20 Konfigurasi Pipeline Untuk Melakukan Trigger Commit Environment Production

Pada gambar 5.20, pipeline ini memiliki tujuan untuk melakukan trigger berupa penambahan *commit* kedalam repository GitOps. Penambahan commit tersebut akan membuat atau mengubah berkas yang sudah ada agar menggunakan `IMAGE_TAG` terbaru. Hal ini akan dilakukan di *environment production*.

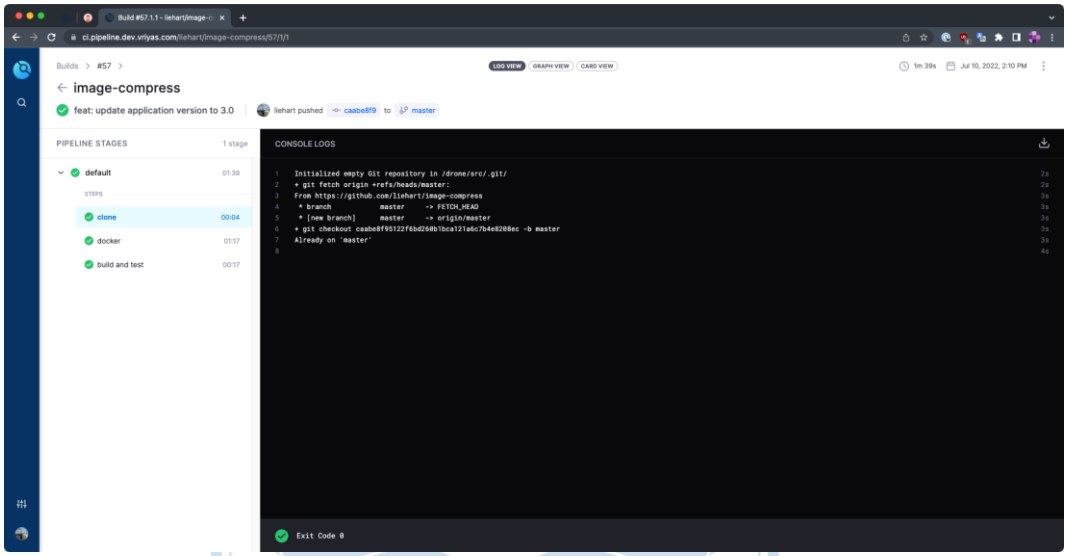
i. Uji Penambahan Fitur Baru Kedalam *Microservice*

Uji penambahan fitur dilakukan dengan membuat sebuah fitur sederhana yang akan diujikan dari proses *development* hingga masuk ke *environment production*. Pada gambar 5.21 adalah gambar kondisi sebelum dilakukan penambahan fitur. Fitur yang akan ditambahkan adalah perubahan nilai dari balikan respon API yaitu yang sebelumnya hanya "OK" menjadi "*Application running with new version. (v3.0)*".



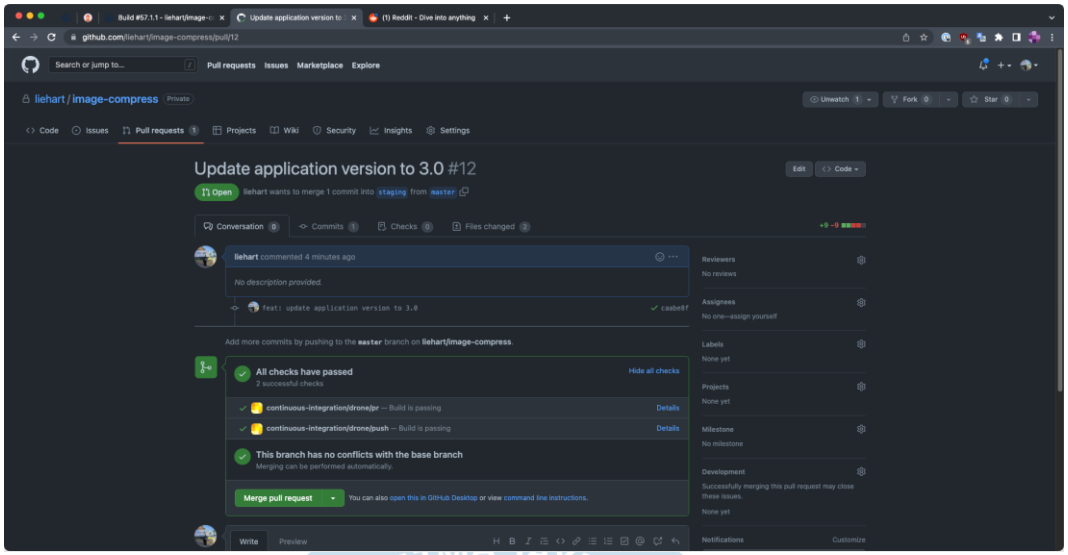
Gambar 5.21 Aplikasi kompres gambar sebelum dilakukan penambahan fitur baru

Pengujian dimulai dengan menambahkan pesan baru kedalam *source code* aplikasi. Lalu dilakukan *commit* dan *push* ke *branch master*. Di dalam *branch master* CI akan mendeteksi perubahan *commit* dan melakukan serangkaian pengujian untuk memastikan program telah sesuai standar dan berjalan dengan baik. Hasil dari eksekusi CI dapat dilihat pada gambar 5.22.



Gambar 5.22 Continuous Integration sedang dijalankan untuk penambahan fitur baru

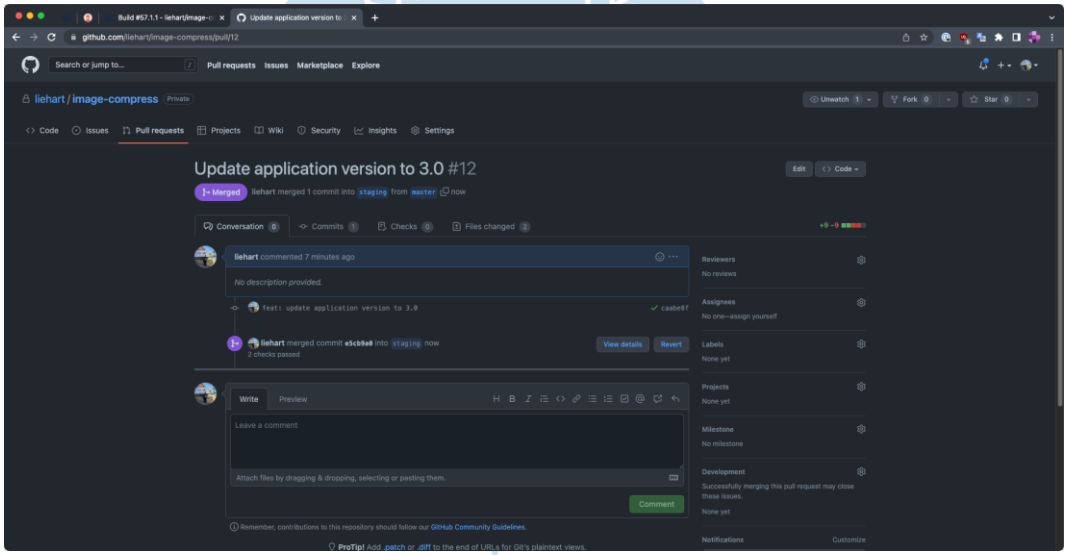
Selanjutnya dilakukan proses penggabungan *development branch* yang terdapat di *branch master* ke *branch staging*. Hal tersebut dilakukan dengan membuat sebuah *pull request* dari *branch master* ke *branch staging*. Pada gambar 5.23 dapat dilihat hasil dari pembuatan *pull request*. CI juga berhasil melakukan tugasnya untuk menguji dan memberikan *feedback* terhadap kode yang akan digabungkan.



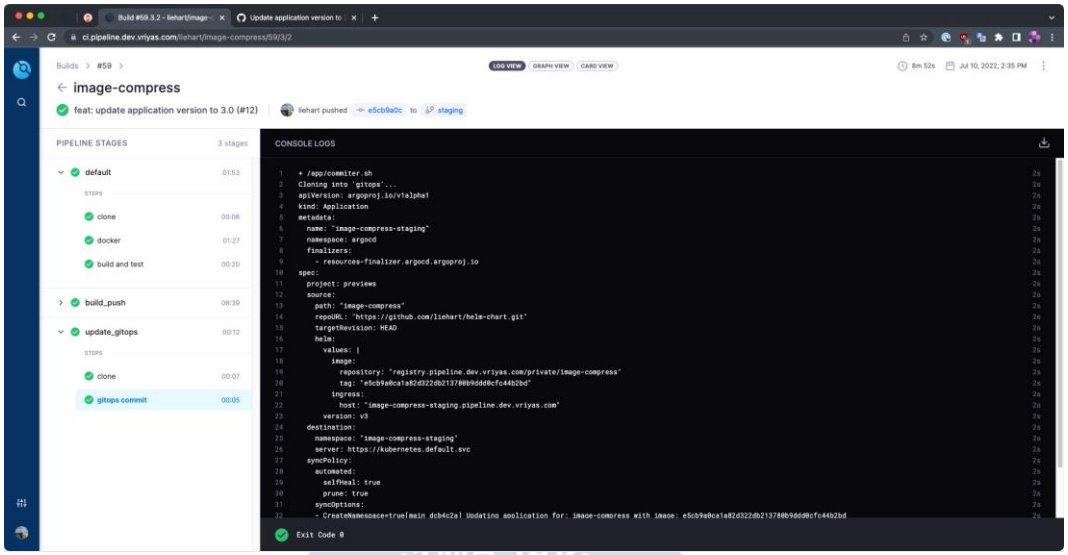
Gambar 5.23 Membuat pull request untuk menggabungkan fitur baru kedalam branch staging

j. Uji Proses CI/CD Pada *Branch Staging*

Pada uji ini proses dimulai dengan menggabungkan *pull request* yang sebelumnya telah dibuat. Pada gambar 5.24 dapat dilihat bahwa proses *merging source code* dari *branch master* ke *branch staging* berhasil dilakukan. Sesuai dengan definisi konfigurasi yang diberikan untuk DroneCI pada berkas `.drone.yml` maka setelah penggabungan berhasil DroneCI akan menjalankan pengujian ulang, membangun *container image* dan menyimpannya kedalam *container registry*. Terakhir DroneCI akan melakukan *trigger* berupa perubahan atau penambahan konfigurasi pada *repository* GitOps.

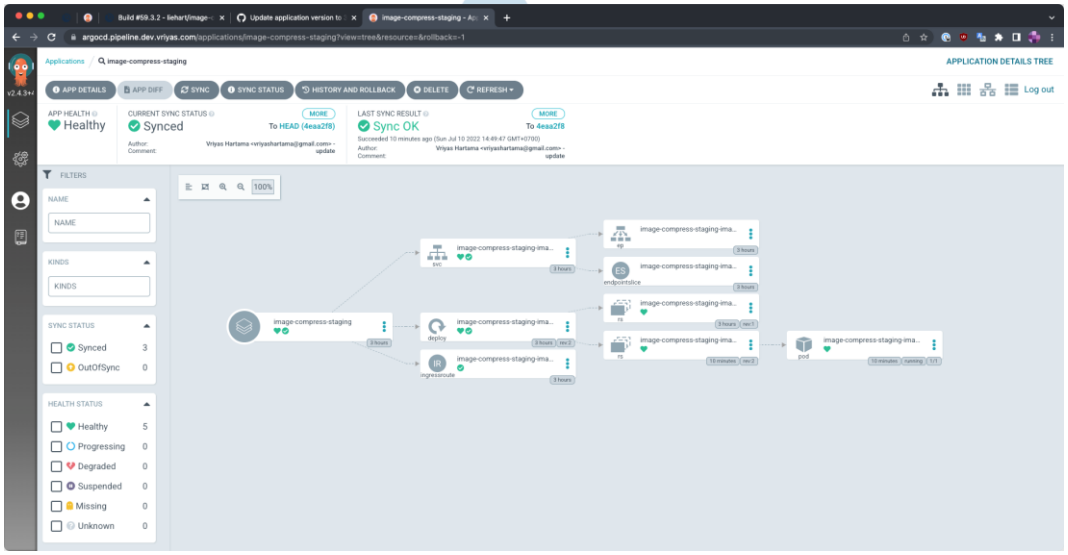


Gambar 5.24 Merge fitur baru kedalam branch staging setelah semua test berhasil



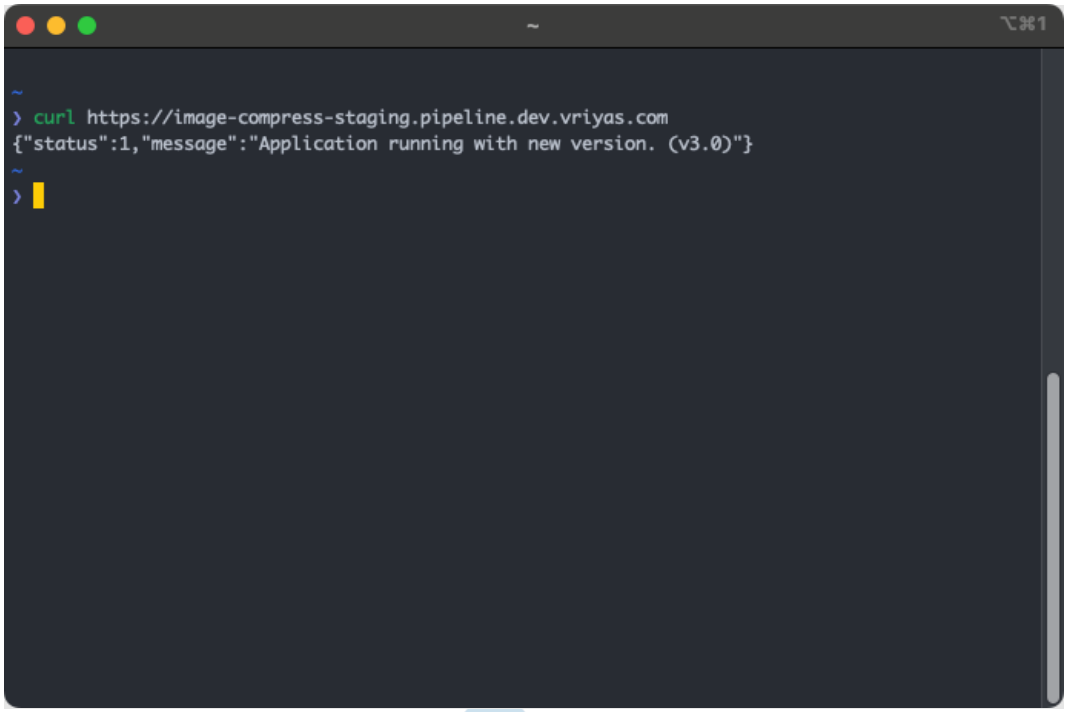
Gambar 5.25 DroneCI melakukan proses testing ulang, build container image dan deployment versi baru kedalam environment staging

Gambar 5.25 menunjukkan proses yang dilakukan CI setelah proses *merging pull request* berhasil. CI melakukan tiga tahap yang telah di definisikan dengan sukes. Langkah terahkir yang dilakukan untuk mengupdate `IMAGE_TAG` pada konfigurasi di *repository* GitOps berhasil.



Gambar 5.26 ArgoCD melakukan deployment otomatis setelah DroneCI mengupdate konfigurasi di dalam repository GitOps

Pada gambar 5.26 adalah tampilan *dashboard* dari ArgoCD. Gambar tersebut menampilkan hasil *deployment* yang telah dilakukan kedalam *environment staging*. Bagan tersebut menjelaskan *beberapa* Kubernetes *resource definition* yang telah di rilis untuk *environment staging* aplikasi kompresi gambar.



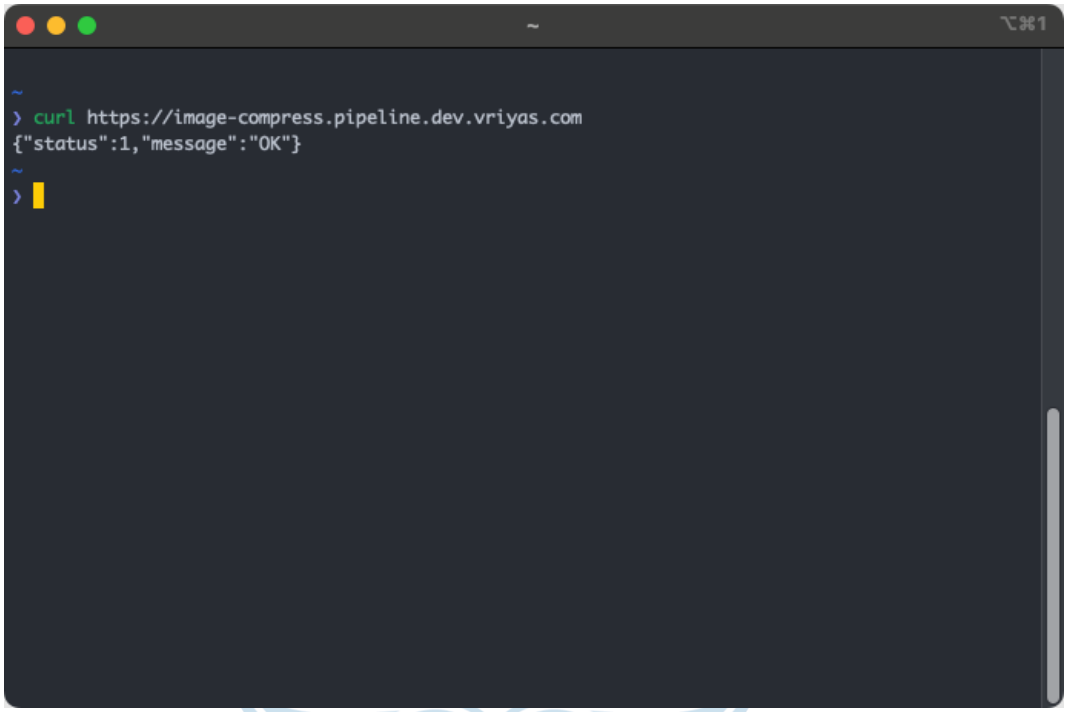
Gambar 5.27 Aplikasi dengan fitur baru telah terdeploy di environment staging

Pada gambar 5.27 adalah hasil dari *deployment* kedalam *environment staging*. Hasil eksekusi `curl` untuk URL *staging* dari aplikasi kompresi gambar telah menampilkan respon versi terbaru. Maka proses *deployment source code* dari *branch master* (*development*) ke *environment staging* berhasil.

k. *Promote Deployment* di *Environment Staging* Menjadi *Deployment* di *Environment Production*

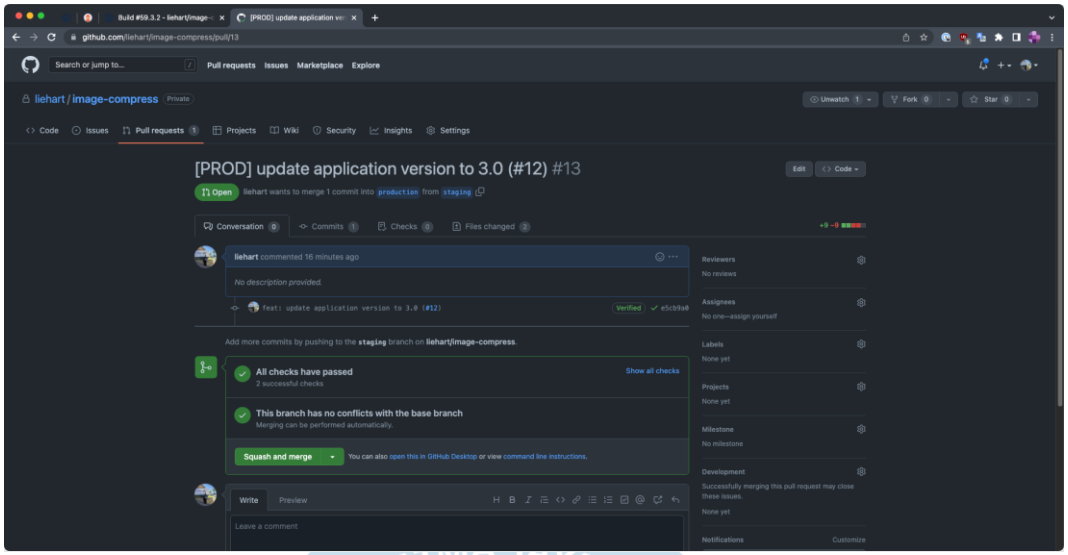
Pada bagian ini hasil dari *deployment* yang telah dilakukan di *environment* staging akan dicerminkan pada proses *deployment environment production*. Langkah yang dilakukan adalah sama dengan proses yang

dilakukan pada *environment staging.* Pada gambar 5.28 dapat dilihat bahwa status saat ini pada *environment production* masih menggunakan versi yang lama. Hal ini diketahui dengan perbedaan respon yang diberikan aplikasi pada *environment staging* dan *environment production.*

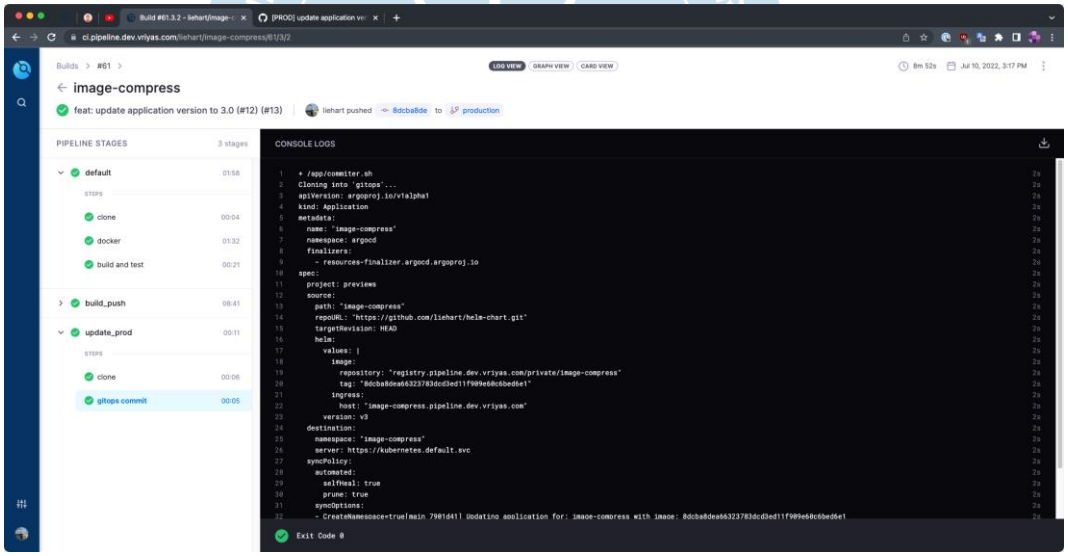


Gambar 5.28 Aplikasi kompres gambar dalam environment production sebelum dilakukan promotion staging deployment

Langkah pertama yang dilakukan untuk melakukan promosi *environment* adalah membuat *pull request. Pull request* yang dibuat berasal dari *branch staging* ke *branch production*. Penggunaan metode *pull request* menjadikan proses promosi lebih terkontrol dan memiliki pencatatan di dalam *repository* tersebut. Setelah membuat *pull request* maka DroneCI akan melakukan *testing* terhadap *pull request* untuk memastikan bahwa kode yang akan digabungkan dapat berjalan. Setelah semua tes sukses maka mengembang dapat melakuka proses *merging source code* dari *branch staging* ke *branch production. Pull request* yang dibuat untuk melakukan aksi ini dapat dilihat pada gambar 5.29.



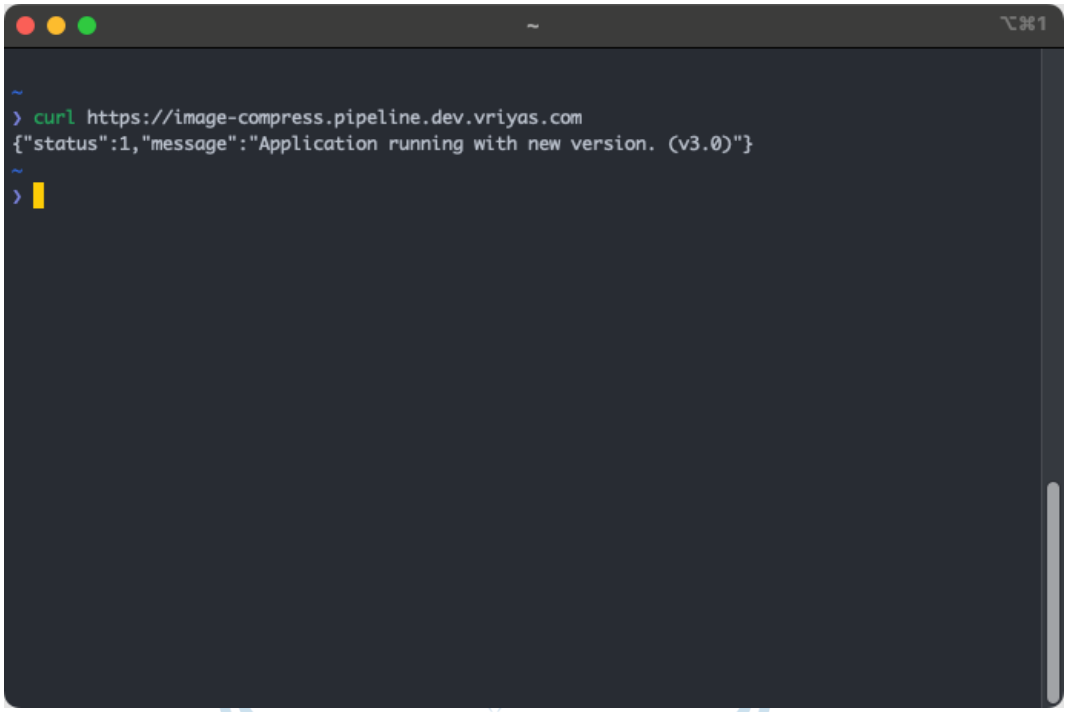
Gambar 5.29 Pull Request dari branch staging ke production untuk melakukan promotion deployment.



Gambar 5.30 DroneCI melakukan proses testing ulang, build container image dan deployment versi baru kedalam environment production

Pada gambar 5.30 adalah beberapa proses yang dilakukan oleh DroneCI setelah mendapat *webhook* dari GitHub untuk aksi *push commit* baru. Setelah branch staging digabungkan kedalam branch production dilakukan integrasi ulang berupa pengujian aplikasi apakah berjalan dengan benar. Setelah itu membangun ulang *container image* yang selanjutnya disimpan kedalam

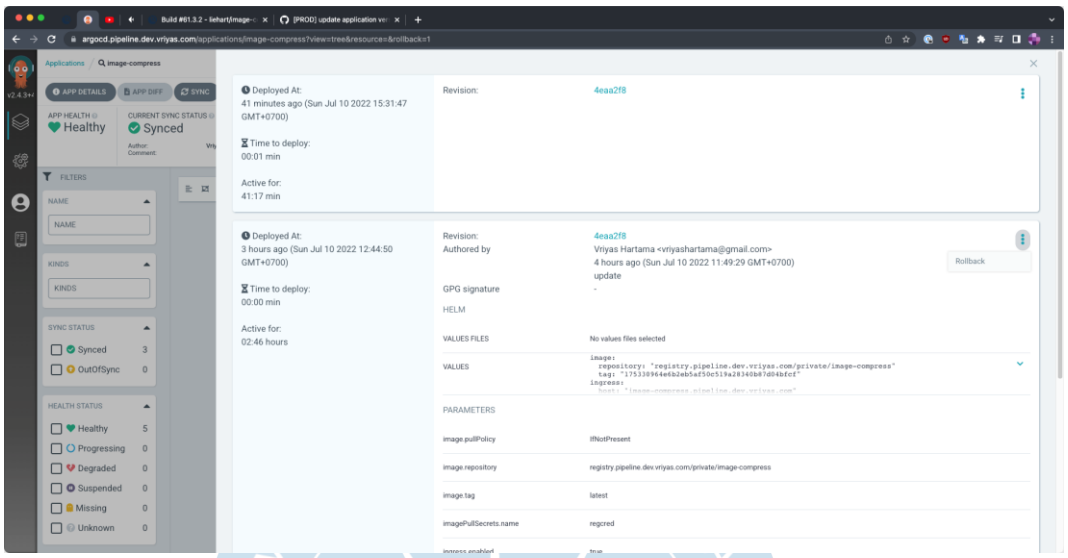
container registry Harbor. Setelah itu langkah terakhir yang dilakukan adalah membuat atau *update* konfigurasi *values* dari Helm Chart yang akan di *deploy* disimpan di dalam repository GitOps.



Gambar 5.31 Aplikasi dengan fitur baru telah terdeploy di *environment production*

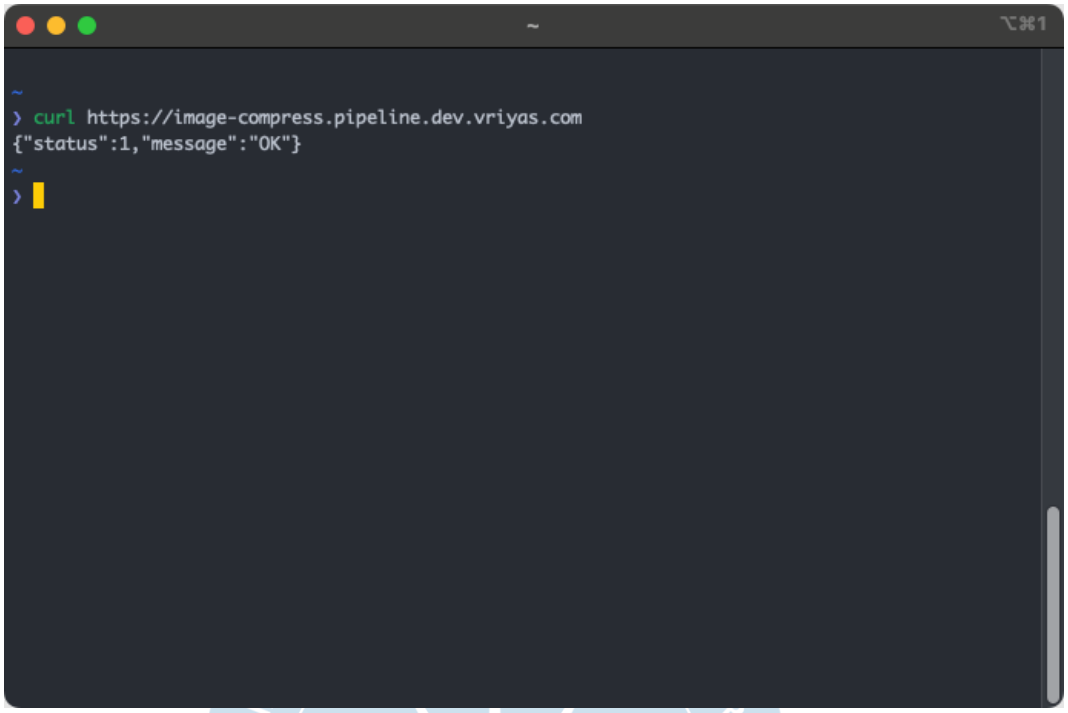
Pada gambar 5.31 adalah hasil dari *deployment* kedalam *environment production*. Hasil eksekusi `curl` untuk URL *production* dari aplikasi kompresi gambar telah menampilkan respon yang sama dengan versi terbaru yang ada di *environment staging.* Maka proses promosi hasil rilis dalam *environment staging* kedalam *environment production* berhasil dilakukan.

l. *Rollback Release Production* Menjadi Satu Versi Sebelum *Deployment* Terakhir



Gambar 5.32 Tampilan history deployment yang dilakukan dalam branch production.

Pada gambar 5.32 adalah tampilan dari *dashboard* ArgoCD dimana pengembang dapat menggunakan fitur tersebut untuk pelakukan *rollback deployment* ke versi sebelumnya atau secara spesifik ke versi lain yang terdapat di dalam history *deployment*. Untuk melakukan *rollback deployment* pengembang memilih versi *commit* yang akan dijadikan target *deployment* ulang. Pada kasus ini digunakan versi dengan *commit* `4eaa2f8`. Versi tersebut adalah satu versi sebelum versi yang telah dirilis saat ini.

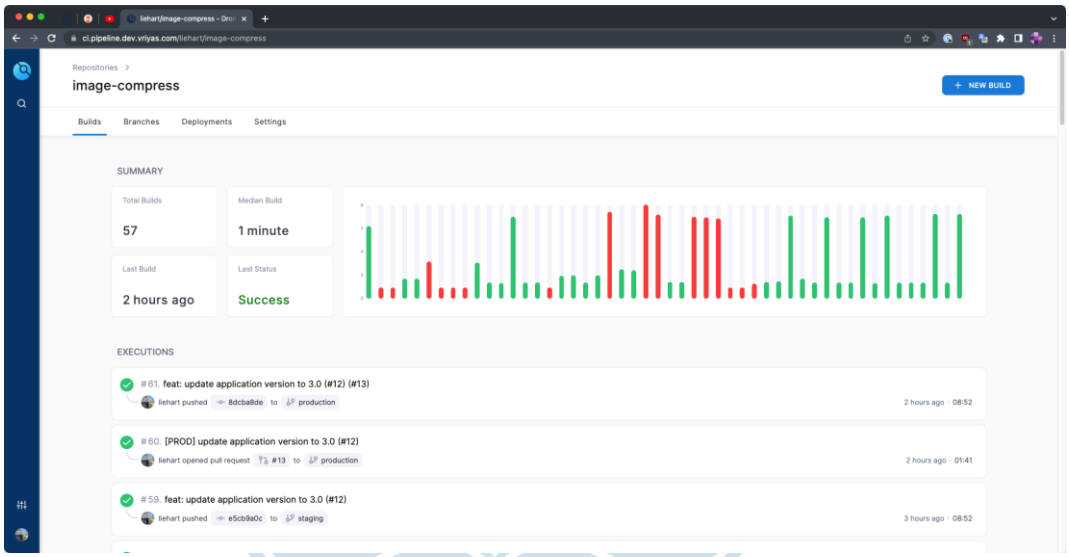


Gambar 5.33 Aplikasi di environment production setelah dilakukan rollback deployment.

Gambar 5.33 adalah hasil eksekusi `curl` untuk URL *production* dari aplikasi kompresi gambar. Dalam gambar tersebut dapat diketahui bahwa hasil respon yang diberikan aplikasi adalah hasil dari versi *deployment* sebelumnya. Dengan demikian diketahui bahwa proses *rollback deployment* ke versi sebelumnya berhasil dilakukan.

B. Hasil Pengujian

Hasil pengujian sistem diambil dari riwayat waktu eksekusi yang dilakukan DroneCI dalam melakukan *test* atau *deployment* aplikasi. Pada gambar 5.34, dari 57 eksekusi CI diambil 15 sampel berupa 10 proses integrasi *source code* melalui *pull request* dan 5 sample berupa proses *deployment* aplikasi dari *environment staging* ke *production* atau *development* ke *staging.*



Gambar 5.34 Dashboard DroneCI untuk repository kompres gambar.

Tabel 5.2 Tabel hasil waktu eksekusi untuk proses integrasi source code melalui pull request

Hasil	Deskripsi	Waktu
1	*Push commit continuous integration*	1 menit 42 detik
2	*Push commit continuous integration*	1 menit 46 detik
3	*Push commit continuous integration*	2 menit 5 detik
4	*Push commit continuous integration*	1 menit 41 detik
5	*Push commit continuous integration*	1 menit 39 detik
6	*Push commit continuous integration*	1 menit 40 detik
7	*Push commit continuous integration*	1 menit 38 detik
8	*Push commit continuous integration*	1 menit 39 detik
9	*Push commit continuous integration*	1 menit 40 detik
10	*Push commit continuous integration*	1 menit 41 detik
Rata-rata		**1 menit 43.1 detik**

Tabel 5.2 merupakan tabel hasil proses *CI* yang diperoleh dari sistem Drone CI. Hasil yang diambil berupa 10 data dari proses yang telah berjalan. Dari proses tersebut diperoleh rata-rata bahwa proses CI yang dijalankan menyangkut proses pengujian dapat dilakukan selama 1 menit 43.1 detik. Pengambilan sampel data berupa 10 data dari proses CI didasari atas keseragaman hasil yang didapat. Pada hasil ke 3 terdapat anomali bahwa proses CI yang dilakukan menjadi 2 menit 5 detik. Hal tersebut dapat terjadi dikarenakan proses CI yang melakukan pengunduhan *source code* dan *container image* dari internet sehingga penyimpangan waktu tersebut dapat terjadi bila kondisi internet yang kurang baik. Pada tabel 5.2 pengujian dilakukan dengan *microservice* kompres gambar yang memiliki *dependencies* yang cukup sedikit. Tidak menutup kemungkinan bahwa apabila sistem yang lebih besar akan memakan waktu yang lebih lama.

Tabel 5.3 Tabel hasil waktu eksekusi untuk proses deployment aplikasi ke environment staging ataupun production.

Hasil	Deskripsi	Waktu
1	*Push registry and deploy application*	8 menit 39 detik
2	*Push registry and deploy application*	8 menit 30 detik
3	*Push registry and deploy application*	8 menit 29 detik
4	*Push registry and deploy application*	8 menit 39 detik
5	*Push registry and deploy application*	8 menit 52 detik
Rata-rata		**8 menit 37.8 detik**

Tabel 5.3 merupakan tabel hasil proses *CI*/CD yang diperoleh dari sistem Drone CI dan ArgoCD. Hasil yang diambil berupa 5 data dengan skenario pengembang melakukan penggabungan *development environment* ke *staging environment* atau *production environment*. Pengambilan sampel data berupa 5 data dari proses CI/CD tersebut dikarenakan proses yang dilakukan sistem sama dan kelima data tersebut sudah dapat menggambarkan hasil data berikutnya yang tegolong tidak fluktuatif. Dalam proses tersebut diperoleh rata-rata bahwa proses otomasi tersebut menghabiskan waktu selama 8 menit 37.8 detik. Pada tabel 5.3 pengujian dilakukan dengan *microservice* kompres gambar yang memiliki *dependencies* yang cukup sedikit. Tidak menutup kemungkinan bahwa apabila sistem yang lebih besar akan memakan waktu yang lebih lama.

C. Analisis Pengujian Implementasi

Berdasarkan hasil pengujian yang dilakukan menggunakan sistem CI/CD *pipeline* yang telah di implementasikan didapatkan hasil sebagai berikut:

a. Perbandingan proses perilisan sebelum dan sesudah implementasi

Dalam rumusan masalah penelitian dilakukan kajian perbandingan mengenai proses pengujian dan perilisan sebelum dilakukan implementasi sistem CI/CD.

Tabel 5.4 Tabel perbandingan proses pengujian hingga perilisan sebelum dan sesudah implementasi CI/CD *pipeline*

Kriteria Perbandingan	**Sebelum Implementasi**	**Sesudah Implementasi**
Pengujian Perangkat Lunak	Secara manual oleh pengembang menggunakan perangkat masing-masing	Secara otomatis setiap perubahan kode di gabungkan kedalam *source code repository*
Proses Perilisan	Melakukan *pull source code* melalui *git* dan melakukan proses *build* manual di server	Menggunakan ArgoCD dengan mengganti kode manifest secara otomatisz`
Proses *Rollback*	Mengulang proses perilisan secara manual menggunakan *commit hash* atau *branch* sebelumnya	Menggunakan fitur dari ArgoCD untuk melakukan *rollback deployment* secara otomatis

Tabel 5.4 menjelaskan perbedaan proses pengujian, perilisan dan *rollback* perilisan sebelum dan sesudah implementasi. Sebelum dilakukan implementasi CI/CD proses dilakukan secara manual oleh pengembang. Sedangkan setelah dilakukan implementasi, langkah-langkah yang sebelumnya dilakukan secara manual dilakukan secara otomatis.

Terdapat beberapa perubahan yang dilakukan agar *microservice* kompres gambar dapat menggunakan implementasi CI/CD. Perubahan pertama yang harus ditambahkan adalah penggunaan berkas `drone.yml` sebagai berkas yang mendefinisikan proses apa yang harus dilakukan CI di tahap pengujian. Perlu ditambahkan juga berkas manifest untuk Helm Chart agar ArgoCD dapat melakukan proses perlilisan *service* kedalam orkestrator Kubernetes.

b. Integrasi CI/CD dalam proses *pull request*

Berdasarkan hasil pengujian *continuous integration* untuk *pull request* yang dibuat. Proses tersebut menguji bahwa kode program telah sesuai standar yang ditentukan dengan dijalankan unit test yang telah disusun. *Continuous integration* memastikan bahwa kode yang di gabungkan kedalam *branch master* pasti berjalan dengan ditandai suksesnya proses integrasi.

Berdasarkan hasil pengujian dengan mengulangi proses *pull request* fitur baru sebanyak 10 kali. Adapun hal yang didapatkan yaitu berupa hasil yang konsisten. Diketahui bahwa dalam proses integrasi waktu secara otomatis menghabiskan waktu sebanyak 1 menit 43.1 detik rata-rata.

c. Integrasi CI/CD dalam proses *deployment* ke *environment staging* atau *production*

Proses ini merupakan proses yang dijalankan secara manual. *Developer* harus melakukan *deployment* secara manual kedalam server dengan memperbaharui konfigurasi docker yang digunakan. Dengan proses ini pekerjaan yang dilakukan secara manual digantikan oleh ArgoCD. Dalam pengujian yang dilakukan dengan melakukan deployment secara otomatis didapatkan waktu salaam 8 menit 37.8 detik rata-rata. Waktu tersebut lebih rendah dari waktu *deployment* secara manual oleh *developer*.

D. Analisis Dampak Otomasi Setelah Implementasi CI/CD

a. Analisis Dampak Otomasi Terhadap Waktu Perilisan

Berdasarkan perbandingan proses perilisan sebelum dan sesudah implementasi. Diketahui bahwa Sebagian besar proses yang sebelumnya dilakukan secara manual akan dilakukan secara otomatis. Dengan demikian proses tersebut telah menghemat waktu dimana pengembang tidak perlu melakukan proses manual lagi. Jika dilakukan hipotesis bahwa dilakukan perilisan selama sekali seminggu untuk *production* dan 5 kali seminggu untuk *staging* maka dapat ditarik total

waktu sebanyak 6 kali 8 menit 37.8 detik atau 34.52 jam selama rentang waktu 1 tahun. Maka dapat ditarik kesimpulan bahwa dengan menggunakan sistem ini pengembang dapat mengalokasikan total waktu terkumpul yang sebelumnya dilakukan secara manual untuk kegiatan yang lebih produktif sebanyak 34.52 jam atau hampir 4 hari kerja. Hal tersebut terus bertambah apa bila digabungkan dengan proyek lain sehingga implementasi ini akan terus menghemat waktu.

b. Analisis Efisiensi Biaya Sebelum dan Sesudah Implementasi

Berdasarkan analisis waktu untuk proses *deployment* yang telah dilakukan diketahui penghematan selama setahun kurang lebih adalah 4 hari. Dengan demikian dapat diketahui bahawa sistem ini akan menghemat biaya yang dikeluarkan untuk sumber daya manusia melakukan tugas berulang dan dapat dialokasikan untuk mengerjakan pekerjaan yang lebih penting. Namun karena biaya yang diperlukan untuk sumber daya manusia berbeda-beda maka tidak dapat dihitung secara tepat.

c. Analisis Pengurangan Resiko Sebelum dan Sesudah Implementasi

Dengan digunakannya sistem ini proses *rollback* yang dilakukan bergantung kepada ArgoCD. ArgoCD dapat melakukan proses *rollback* secara cepat dengan menggunakan fitur *rollback* yang telah disediakan. Dengan fitur ini proses perubahan ke versi sebelumnya berjalan dengan cepat dan pasti berhasil karena versi sebelumnya akan dijalankan sama persis sebelum rilis terbaru diaplikasikan.

BAB VI
PENUTUP

A. Kesimpulan

Berdasarkan penelitian yang telah dilakukan, dapat diambil beberapa kesimpulan dari hasil analisis yang diperoleh sebagai berikut:

1. Proses otomatisasi proses pengembangan perangkat lunak menggunakan DroneCI dan ArgoCD berhasil dilakukan. Implementasi tersebut dapat digunakan untuk melakukan proses pengujian dalam integrasi *source code* dan *deployment* yang sebelumnya dilakukan manual oleh pengembang menjadi otomatis.
2. Implementasi yang diberikan dalam pengujian ini sepenuhnya dapat dilakukan dengan menggunakan perangkat sumber terbuka. Implementasi tersebut dapat dibangun menggunakan bantuan *tools* seperti ArgoCD, Terraform, DroneCI, MinIO dan Harbor.
3. Berdasarkan pengujian yang dilakukan dapat ditarik kesimpulan bahwa dengan menggunakan sistem tersebut pengembang dapat menghemat waktu secara bertahap dengan penghematan yang ditawarkan adalah 8 menit 37.8 detik untuk setiap proses *deployment* yang sebelumnya dilakukan secara manual. Penghematan bertumpuk tersebut dapat menghasilkan penghematan hingga 34.52 jam selama setahun. Perhitungan tersebut didasarkan dengan kegiatan *deployment* ke *environment staging* sebanyak 5 kali seminggu dan 1 kali seminggu untuk *environment production.*

B. Saran

Berdasarkan penelitian yang telah dilakukan, menurut kesimpulan yang diperoleh masih terdapat beberapa kekurangan. Oleh sebab itu memberikan beberapa saran yang dapat diterapkan agar kedepannya jika ada penelitian serupa dapat dilakukan dengan lebih baik. Saran yang dapat berikan antara lain:

1. Melakukan diversifikasi objek penelitian sehingga tidak hanya menjadikan satu objek sistem kompresi gambar melainkan sistem lainnya untuk mensimulasikan proses SDLC didalam perusahaan secara kompleks.
2. Implementasi yang dilakukan cukup kompleks untuk perusahaan kecil. Hal ini menjadikan implementasi yang dilakukan dicocokkan untuk perusahaan skala menengah-besar.

DAFTAR PUSTAKA

[1] R. S. Pressman and B. R. Maxim, *Software Engineering: A Practitioner's Approach*, 8th ed. New York, NY: McGraw-Hill Professional, 2014.

[2] N. Forsgren, D. Smith, J. Humble, and J. Frazelle, "2019 Accelerate State of DevOps Report," 2019. [Online]. Available: http://cloud.google.com/devops/state-of-devops/

[3] D. Smith, D. Villalba, M. Irvine, D. Stanke, and N. Harvey, "2021 Accelerate State of DevOps Report," 2021. [Online]. Available: https://cloud.google.com/devops/state-of-devops/

[4] R. Ghimire, "Deploying Software in the Cloud with CI/CD Pipelines," Haaga-Helia University of Applied Sciences, Helsinki, 2020.

[5] T. Tohirin, S. F. Utami, S. R. Widianto, and W. al Mauludyansah, "Implementasi DevOps Pada Pengembangan Aplikasi e-Skrining Covid-19," *MULTINETICS*, vol. 6, no. 1, pp. 15–20, May 2020, doi: 10.32722/multinetics.v6i1.2764.

[6] S. Ferdian, T. Kandaga, A. Widjaja, H. Toba, R. Joshua, and J. Narabel, "Continuous Integration and Continuous Delivery Platform Development of Software Engineering and Software Project Management in Higher Education," *Jurnal Teknik Informatika dan Sistem Informasi*, vol. 7, no. 1, Apr. 2021, doi: 10.28932/jutisi.v7i1.3254.

[7] A. Alpery and M. A. F. Ridha, " Implementasi CI/CD Dalam Pengembangan Aplikasi Web Menggunakan Docker dan Jenkins," *9th Applied Business and Engineering Conference* , vol. Vol. 9, pp. 287–296, 2021.

[8] A. DHANY, "Implementation of Docker and Continuous Integration / Continuous Delivery for Management Information System Development," *IJEEIT International Journal of Electrical Engineering and Information Technology*, vol. 3, no. 2, pp. 20–24, Jan. 2021, doi: 10.29138/ijeeit.v3i2.1208.

[9] M. Wittig and A. Wittig, *Amazon Web Services in Action*. 2015.

[10] "Kubernetes (K8s)." https://github.com/kubernetes/kubernetes (accessed Feb. 02, 2022).

[11] C. Doxsey, *Introducing Go: Build Reliable, Scalable Programs*, First Edition. Sebastopol, CA: O'Reilly Media, 2016.

[12] “Drone CI/CD Documentation.” https://docs.drone.io/ (accessed Feb. 02, 2022).

[13] “Argo CD - Declarative Continuous Delivery for Kubernetes.” https://github.com/argoproj/argo-cd/ (accessed Feb. 02, 2022).

[14] “What is CI/CD?,” 2018. https://www.redhat.com/en/topics/devops/what-is-ci-cd (accessed Feb. 02, 2022).