

***Automatic Deployment Aplikasi berbasis Microservice Pada
Platform Kubernetes Dengan Metode Pull-Up***

Laporan Tugas Akhir

Diajukan Untuk Memenuhi
Persyaratan Guna Meraih Gelar Sarjana
Informatika Universitas Muhammadiyah Malang



Muhammad Zein Ihza Fahrozi
201810370311072

Bidang Minat
Rekayasa Perangkat Lunak

PROGRAM STUDI TEKNIK INFORMATIKA FAKULTAS TEKNIK
UNIVERSITAS MUHAMMADIYAH MALANG
2025

LEMBAR PERSETUJUAN

LEMBAR PERSETUJUAN

**Automatic Deployment Aplikasi berbasis Microservice Pada
Platform Kubernetes Dengan Metode Pull-Up**

TUGAS AKHIR

**Sebagai Persyaratan Guna Meraih Gelar Sarjana Strata 1
Informatika Universitas Muhammadiyah Malang**

Menyetujui,

Malang, 23 Mei 2025

Dosen Pembimbing 1

Ir. Agus Eko Minarno, S.Kom., M.Kom. Ir. Hys Nurwasin S.Kom., M.Kom.

NIP. 108.1410.0540

Dosen Pembimbing 2

NIP. 108.1410.0561

LEMBAR PENGESAHAN

“To Be Added Bismillah”



LEMBAR PERNYATAAN

LEMBAR PERNYATAAN

Yang bertanda tangan dibawah ini :

NAMA : Muhammad Zein Ihza Fahrozi

NIM : 201810370311072

FAK./JUR. : Informatika

Dengan ini saya menyatakan bahwa Tugas Akhir dengan judul "**Automatic Deployment Aplikasi Berbasis Microservice Pada Platform Kubernetes Dengan Metode Pull-Up**" beserta seluruh isinya adalah karya saya sendiri dan bukan merupakan karya tulis orang lain, baik sebagian maupun seluruhnya, kecuali dalam bentuk kutipan yang telah disebutkan sumbernya.

Demikian surat pernyataan ini saya buat dengan sebenar-benarnya. Apabila kemudian ditemukan adanya pelanggaran terhadap etika keilmuan dalam karya saya ini, atau ada klaim dari pihak lain terhadap keaslian karya saya ini maka saya siap menanggung segala bentuk resiko/sanksi yang berlaku.

Mengetahui,
Dosen Pembimbing



Ir. Agus Eko Minarno, S.Kom.,
M.Kom.

Malang, 23 Mei 2025
Yang Membuat Pernyataan



Muhammad Zein Ihza Fahrozi

ABSTRAK

Penelitian ini membahas implementasi ArgoCD dengan pendekatan GitOps pada lingkungan Kubernetes yang berjalan di atas infrastruktur bare-metal menggunakan Proxmox VE dan Talos OS. Tujuan penelitian ini adalah untuk mengevaluasi efektivitas ArgoCD dalam mengotomatisasi proses deployment aplikasi berbasis microservices dengan prinsip GitOps. Metode yang digunakan meliputi studi literatur, perancangan arsitektur, implementasi, dan pengujian. Hasil penelitian menunjukkan bahwa ArgoCD berhasil diimplementasikan dengan baik pada lingkungan Kubernetes, menyediakan mekanisme sinkronisasi otomatis antara konfigurasi yang dideklarasikan di Git dengan status aktual di cluster. Integrasi dengan Cloudflare Tunnel memungkinkan akses aman ke aplikasi tanpa perlu membuka port firewall secara langsung. Hasil pengujian menunjukkan tingkat keandalan sistem mencapai 99.9% dengan kemampuan rollback yang efektif dalam waktu kurang dari 1 menit. Penelitian ini membuktikan bahwa pendekatan GitOps dengan ArgoCD dapat diterapkan secara efektif di lingkungan non-cloud, memberikan keuntungan berupa peningkatan keamanan, auditabilitas, dan kemudahan dalam manajemen konfigurasi.

Kata Kunci: *ArgoCD, GitOps, Kubernetes, Continuous Deployment, Cloudflare Tunnel*

ABSTRACT

This research discusses the implementation of ArgoCD with a GitOps approach in a Kubernetes environment running on bare-metal infrastructure using Proxmox VE and Talos OS. The objective of this study is to evaluate the effectiveness of ArgoCD in automating the deployment process of microservices-based applications using GitOps principles. The methodology includes literature study, architectural design, implementation, and testing. The results show that ArgoCD was successfully implemented in the Kubernetes environment, providing an automatic synchronization mechanism between the configuration declared in Git and the actual state in the cluster. Integration with Cloudflare Tunnel enables secure access to applications without the need to directly expose firewall ports. Testing results demonstrate a system reliability rate of 99.9% with effective rollback capability in less than 1 minute. This research proves that the GitOps approach with ArgoCD can be effectively applied in non-cloud environments, offering benefits such as enhanced security, auditability, and ease of configuration management.

Keywords: *ArgoCD, GitOps, Kubernetes, Continuous Deployment, Cloudflare Tunnel*



LEMBAR PERSEMBAHAN

Bismillahirrahmanirrahim

Puji syukur kehadirat Allah SWT atas rahmat dan karunia-Nya yang tiada terhingga, sehingga penulis dapat menyelesaikan penyusunan Tugas Akhir ini. Dalam perjalanan penyusunan tugas akhir ini, penulis menyadari sepenuhnya bahwa banyak pihak yang telah memberikan dukungan dan bimbingan. Oleh karena itu, dengan penuh rasa hormat dan terima kasih, penulis persembahkan karya ini kepada:

- Kedua orang tua tercinta yang senantiasa memberikan doa, dukungan moril, dan materiil tanpa henti, serta kesabaran yang tiada batas dalam mendampingi perjalanan studi penulis.
- Bapak Ilyas Nuryasin, Bapak Agus Eko, Bapak Ali Sofyan selaku dosen pembimbing yang telah meluangkan waktu, memberikan bimbingan, saran, dan masukan berharga sehingga penulisan tugas akhir ini dapat terselesaikan dengan baik.
- Seluruh dosen pengajar di Program Studi Teknik Informatika yang telah memberikan ilmu pengetahuan dan pengalaman berharga selama masa perkuliahan.
- Teman-teman seperjuangan di kampus yang telah berbagi suka dan duka, memberikan semangat, serta menjadi keluarga kedua selama menempuh pendidikan.
- Semua pihak yang tidak dapat disebutkan satu per satu yang telah memberikan dukungan dan bantuan dalam penyelesaian tugas akhir ini.

Malang, June 29, 2025

KATA PENGANTAR

Dengan mengucap syukur alhamdulillah ke hadirat Allah Subhanahu Wa Ta'ala yang telah melimpahkan rahmat dan hidayah-Nya, penulis bersyukur dapat menyelesaikan penyusunan skripsi dengan judul "**Automatic Deployment Aplikasi Berbasis Microservice pada Platform Kubernetes dengan Metode Pull-Up**" ini. Sholawat serta salam semoga senantiasa tercurahkan kepada Nabi Muhammad Shallallahu 'Alaihi Wasallam, sebagai suri tauladan terbaik sepanjang masa, yang telah membawa cahaya petunjuk bagi seluruh umat manusia.

Malang, 15 Mei 2025

Muhammad Zein Ihza Fahrozi



DAFTAR ISI

LEMBAR PERSETEJUAN	i
LEMBAR PENGESAHAN	ii
LEMBAR PERNYATAAN	iii
ABSTRAK	iv
ABSTRACT	v
LEMBAR PERSEMBAHAN	vi
KATA PENGANTAR	vii
DAFTAR ISI	ix
DAFTAR TABEL	x
DAFTAR GAMBAR	xi
DAFTAR KODE PROGRAM	xii
BAB I PENDAHULUAN	1
1.1 Latar Belakang	1
1.2 Rumusan Masalah	5
1.3 Tujuan Penelitian	5
1.4 Batasan Masalah	5
BAB II TINJAUAN PUSTAKA	7
2.1 Penelitian Terdahulu	7
2.2 Kubernetes	9
2.3 Argo CD	10
2.4 GitOps	10
2.5 Penelitian Lanjutan	10
BAB III METODOLOGI PENELITIAN	12
3.1 Rancangan Penelitian	12
3.1.1 Studi Literatur	12

3.1.2	Perancangan (Design)	13
3.1.3	Implementasi Code Automasi (Coding)	14
3.1.4	Pengujian dan Analisis (Testing dan Analysis)	15
3.1.5	Kesimpulan	16
BAB IV HASIL DAN PEMBAHASAN	17
4.1	Perancangan (Design)	17
4.1.1	Perancangan Sistem Arsitektur Infrastruktur Mesin	17
4.1.2	Perancangan Arsitektur ArgoCD pada Kubernetes Cluster . .	18
4.1.3	Perancangan Arsitektur Microservice pada Kubernetes Cluster	19
4.2	Pengkodean (Coding) / Implementasi	20
4.2.1	Implementasi Sistem Infrastruktur Mesin	21
4.2.2	Implementasi ArgoCD pada Kubernetes Cluster	31
4.3	Implementasi Microservice	41
4.3.1	Implementasi UI Service	41
4.3.2	Implementasi Auth Service	45
4.3.3	Implementasi Color Service	45
4.3.4	Implementasi Prime Generator Service	46
4.4	Testing	47
4.4.1	Pengujian (Testing) Pada Microservice	48
4.4.2	Pengujian (Testing) Pada ArgoCD	58
4.4.3	Hasil Pengujian	69
BAB V PENUTUP	70
5.1	Kesimpulan	70
5.2	Saran	71
DAFTAR PUSTAKA	72

DAFTAR TABEL

2.1 Rangkuman Penelitian Terdahulu	9
4.1 Spesifikasi Node Kubernetes	21
4.2 Implementasi Prime Generator Service	47
4.3 Daftar Test Case Fungsional Frontend Service	49
4.4 Daftar Kasus Uji Fungsional Layanan Autentikasi	50
4.5 Daftar Unit Test Case Prime Generator Service	53
4.6 Daftar Kasus Uji Fungsional Layanan Color	56
4.7 Daftar Test Case Black-Box Testing	59
4.8 Daftar Test Case Black-Box Testing (Lanjutan)	60
4.9 Pengujian Flow Deployment Service Frontend	61
4.10 Pengujian Flow Deployment Service Frontend	66

DAFTAR GAMBAR

3.1 Rancangan Penelitian	12
4.1 Arsitektur Infrastruktur High Level	17
4.2 Arsitektur Virtual Machine Talos OS pada Proxmox Cluster	18
4.3 Arsitektur Kubernetes High Level	19
4.4 Arsitektur Microservice High Level	20
4.5 Pembuatan Bootable Proxmox Menggunakan Rufus	21
4.6 Tampilan Awal Instalasi Proxmox VE	22
4.7 Tampilan Terminal Setelah Instalasi Proxmox Selesai	22
4.8 Tampilan Web Interface Proxmox VE	23
4.9 Proses Download ISO Talos OS	23
4.10 Instalasi Talos OS 1	24
4.11 Instalasi Talos OS 2	24
4.12 Instalasi Talos OS 3	25
4.13 Instalasi Talos OS 4	25
4.14 Instalasi Talos OS 5	26
4.15 Instalasi Talos OS 6	26
4.16 Instalasi Talos OS 7	27
4.17 Instalasi Talos OS 8	27
4.18 Instalasi Talos OS 9	28
4.19 Tampilan Talos OS setelah instalasi kubernetes cluster	30
4.20 Tampilan echo.zeinfahrozi.my.id pada browser	31
4.21 Flow Pull Based Deployments	36
4.22 Output Github Action Running	38
4.23 Tahap-Tahap Deployment sebuah service baru pada ArgoCD	62
4.24 Tampilan Halaman Frontend Setelah Deployment	63
4.25 Hasil Implementasi Auth Service yang digunakan pada Frontend	63
4.26 Hasil Implementasi Color Service yang digunakan pada Frontend	64
4.27 Hasil Implementasi Prime Generator Service yang digunakan pada Frontend	65
4.28 Pull Request Perubahan Kode Prime Generator Service	66
4.29 Hasil Generate Image Baru Prime Generator Service	67
4.30 Perubahan Versi Image Pada Pull Request Code	67
4.31 Perubahan Versi Image Pada Manifest ArgoCD	68
4.32 Hasil Prime Generator Service Return 5	68

DAFTAR KODE PROGRAM

4.1	Konfigurasi script instalasi Kubernetes cluster pada Talos OS	28
4.2	Konfigurasi node Kubernetes cluster pada Talos OS	29
4.3	Perintah Instalasi ArgoCD menggunakan Helm	31
4.4	Contoh konfigurasi <code>values.sops.yaml</code> untuk ArgoCD	32
4.5	Konfigurasi ArgoCD untuk instalasi Cloudflare Tunnel	34
4.6	Konfigurasi dasar Cloudflare Tunnel untuk ArgoCD	35
4.7	Salah Satu Contoh Implementasi Build Pipeline pada Prime Service	36
4.8	Contoh Manifest Deployment, Service, Ingress pada Git Repository	38
4.9	Implementasi UI Service Halaman Login	41
4.10	Implementasi Algoritma Login UI Service	42
4.11	Implementasi Dashboard Service	43
4.12	Implementasi Auth Service	45
4.13	Implementasi Color Service	46
4.14	Kode Unit Testing Pada Auth Service	50
4.15	Kode Unit Testing Pada Prime Generator Service	53
4.16	Kode Unit Testing Pada Prime Generator Service 2	54
4.17	Kode Unit Testing Pada Color Service	56

BAB I

PENDAHULUAN

1.1 Latar Belakang

Sebelum digunakan oleh pengguna secara luas, sebuah perangkat lunak perlu melalui proses *deployment* terlebih dahulu. *Deployment* perangkat lunak dapat didefinisikan sebagai akuisisi dan eksekusi sebuah perangkat lunak. Proses ini biasanya dilakukan oleh seorang software deployer atau yang lebih dikenal saat ini sebagai SRE (site reliability engineer) [1]. Dengan demikian, *deployment* dapat dikatakan sebagai aktivitas pasca-produksi sebuah perangkat lunak sebelum digunakan oleh konsumen. Proses *deployment* perangkat lunak terdiri dari beberapa tahapan yang saling berhubungan, seperti proses rilis perangkat lunak, instalasi perangkat lunak ke dalam environment eksekusi, dan aktivasi perangkat lunak [2].

Pada saat melakukan *deployment* sebuah sistem perangkat lunak, beberapa aspek perlu diperhatikan, seperti sub-komponen yang dibutuhkan atau *package external*, serta *resource* (hardware). Untuk melakukan *deployment* sub-komponen ini, dibutuhkan sebuah konfigurasi yang mendeskripsikan versi sub-komponen yang digunakan oleh perangkat lunak. Dengan bahasa modern sekarang, konfigurasi sub-komponen yang digunakan oleh main perangkat lunak biasanya sudah dapat dihasilkan secara otomatis, contohnya adalah; *go modules* [3] dalam bahasa GoLang, *requirement file* [4] dalam bahasa Python, atau file *RubySpec* [5] pada bahasa Ruby.

Terdapat beberapa karakteristik fundamental dalam *deployment* perangkat lunak yang diidentifikasi oleh Alan Dearle [6], yaitu:

Release merupakan tahap penghubung antara proses *deployment* dengan pengembangan perangkat lunak. Tahap ini mencakup seluruh operasi yang diperlukan untuk mempersiapkan sistem sebelum diserahkan kepada pengguna akhir. Aktivitas *release* juga meliputi penentuan sumber daya (*resource*) yang dibutuhkan agar sistem dapat beroperasi dengan optimal pada *environment* yang ditargetkan.

Selanjutnya dilakukan *packaging* terhadap sistem perangkat lunak. *Package* yang dihasilkan harus memuat seluruh komponen yang dibutuhkan, termasuk deskripsi sistem, *dependencies* eksternal, prosedur *deployment*, serta informasi relevan lainnya yang diperlukan untuk menjalankan sistem pada *environment* tujuan.

Installation merupakan tahap persiapan sebelum aktivasi sistem. ***Activation***

merupakan proses eksekusi perangkat lunak pada waktu tertentu, yang dapat dilakukan melalui antarmuka grafis (*graphical user interface*) atau sebagai layanan latar belakang (*daemon process*).

Updating adalah proses penggantian komponen perangkat lunak yang terinstal dengan versi yang lebih baru. Adapun **Undeployment** atau yang dikenal juga sebagai *deinstallation*, merupakan proses penghapusan perangkat lunak yang terinstal dari suatu sistem.

Menurut penelitian yang dilakukan oleh Mockus dkk. [7], kualitas *deployment* suatu perangkat lunak merupakan salah satu faktor utama yang mempengaruhi persepsi konsumen terhadap kualitas keseluruhan perangkat lunak tersebut. Hal senada diungkapkan oleh Jansen dan Brinkkemper [8] yang menyatakan bahwa kelancaran proses *deployment* merupakan aspek esensial dalam meningkatkan kualitas produk perangkat lunak suatu perusahaan atau organisasi.

Namun demikian, terdapat beberapa tantangan yang sering dihadapi dalam pelaksanaan aktivitas *deployment*. Menurut Carzaniga [2], tantangan-tantangan tersebut meliputi:

- **Penggantian Komponen:** Kesulitan dalam melakukan pembaruan (*update*) terhadap komponen sistem yang sedang berjalan tanpa mengganggu layanan.
- **Dependencies:** Kompleksitas ketergantungan (*dependencies*) antarkomponen yang harus dikelola dengan hati-hati.
- **Koordinasi:** Perlunya koordinasi yang baik untuk memastikan pembaruan tidak mengganggu proses bisnis yang sedang berjalan.
- **Pluralitas Platform:** Tantangan dalam mengelola *deployment* pada berbagai platform yang berbeda, termasuk kompatibilitas dengan berbagai sistem operasi.

Seiring dengan perkembangan teknologi, kompleksitas suatu perangkat lunak cenderung mengalami peningkatan yang signifikan [9, 10]. Pertumbuhan tim pengembang yang diikuti dengan peningkatan kompleksitas perangkat lunak serta tantangan dalam pemeliharaan (*maintainability*) seringkali menciptakan hambatan (*bottleneck*) dalam siklus pengembangan. Hal ini pada akhirnya dapat menurunkan efisiensi pengembangan perangkat lunak secara keseluruhan [11].

Untuk mengatasi tantangan ini, diperlukan pendekatan arsitektur yang lebih baik, salah satunya adalah dengan menerapkan arsitektur *microservice* [9]. Konsep

microservice memungkinkan pengembangan perangkat lunak dengan memecahnya menjadi komponen-komponen kecil yang terpisah (*loosely coupled*). Setiap komponen dapat dikembangkan, dijalankan, dan diatur secara independen oleh tim pengembang yang berbeda [12].

Saat ini, arsitektur *microservice* telah menjadi pilihan utama dalam pengembangan perangkat lunak yang membutuhkan skalabilitas tinggi [13]. Awalnya, implementasi *microservice* dilakukan dengan memanfaatkan beberapa *Virtual Machine* (VM) yang saling berkomunikasi melalui protokol REST/HTTP (*Hypertext Transfer Protocol*) atau RPC (*Remote Procedure Call*) [14]. Namun, pendekatan berbasis VM ini memiliki beberapa kelemahan, seperti kebutuhan akan operasi manual yang intensif dan biaya operasional yang relatif tinggi. Hal ini menyebabkan proses pengembangan dan penskalaan (*scaling*) menjadi lebih kompleks dan memakan waktu [14].

Perkembangan teknologi telah menggeser paradigma dari penggunaan *Virtual Machine* (VM) menuju pendekatan yang lebih modern dalam merancang arsitektur *microservice*. Teknologi *containerization* hadir sebagai solusi yang memungkinkan pengemasan perangkat lunak dalam wadah yang ringkas dan efisien [14].

Abstraksi yang diberikan oleh *container* tidak hanya menyederhanakan proses eksekusi, tetapi juga memberikan fleksibilitas yang lebih besar dalam manajemen sumber daya. Salah satu keunggulan utamanya adalah kemudahan dalam melakukan penskalaan (*scaling*), di mana penyesuaian kapasitas dapat dilakukan dengan hanya mengatur jumlah *container* yang berjalan [15].

Containerization [16] membawa dampak signifikan terhadap aspek infrastruktur dan *runtime* dalam pengembangan perangkat lunak. Namun, kehadiran *container* menuntut adanya sistem orkestrasi yang dapat mengelola *container* tersebut secara otomatis.

Kubernetes hadir sebagai solusi dengan menyediakan berbagai fitur canggih, termasuk *automatic scaling* yang memungkinkan penyesuaian sumber daya secara dinamis berdasarkan beban kerja [17, 18].

Saat ini, proses *deployment container* atau *service* masih sering kali dilakukan secara manual melalui modifikasi konfigurasi *deployment* berupa file text. Kondisi ini menimbulkan kebutuhan akan solusi otomatisasi yang dapat menangani proses *deployment* untuk *service* baru atau yang mengalami perubahan (*update*) secara lebih efisien.

Dengan banyaknya persaingan dalam dunia perangkat lunak yang terjadi dalam waktu ini sebuah perusahaan atau pengembang memerlukan waktu yang cepat untuk melakukan deployment sebuah perangkat lunak. Terdapat beberapa macam workflow sebuah deploymen perangkat lunak saat ini, tetapi yang industri saat ini lakukan adalah metode DevOps [19]. Metode DevOps sendiri merupakan metode yang digunakan untuk mengembangkan perangkat lunak yang menjembatani antara dua team yang terisolasi dalam struktur organisasi, contohnya adalah team pengembang (Dev) dan team operasi (Ops) [20]. Konsep DevOps [19] sendiri memungkinkan team pengembang dan operasi untuk membangun sebuah perangkat lunak yang dapat dijalankan secara otomatis dan secara berkala dengan menggunakan alat bantu DevOps. Tujuan utama dari itu adalah untuk meningkatkan kecepatan, reliabilitas, dan perangkat lunak yang lebih baik. Dalam DevOps sendiri terdapat beberapa sub-metode yang digunakan yaitu, *Continuous Integration* (CI), *Continuous DELivery* (CDE), dan *Continuous Deployment* (CD) [21].

Meskipun DevOps menawarkan berbagai keunggulan, implementasi praktik *Continuous Integration* (CI), *Continuous Delivery* (CDE), dan *Continuous Deployment* (CD) menghadapi beberapa tantangan signifikan. Tantangan-tantangan tersebut meliputi kompleksitas dalam pemilihan dan adopsi alat bantu (*tools*), kebutuhan adaptasi sumber daya manusia, serta kerentanan terhadap kesalahan konfigurasi selama proses migrasi.

Berdasarkan tinjauan literatur, terdapat beberapa isu krusial dalam implementasi DevOps, antara lain: tantangan arsitektural [20], kompleksitas manajemen *tools* [22], adopsi metodologi baru [23, 24], aspek keamanan dalam alur CI/CD [25], serta mekanisme *rollback* yang efektif [26]. Penelitian yang dilakukan oleh Ramadoni dkk. [27] mengusulkan pendekatan GitOps sebagai solusi untuk mengatasi permasalahan mekanisme *rollback* dan keamanan alur CI/CD.

Namun demikian, penelitian Ramadoni dkk. [27] yang menggunakan pendekatan *pull-based* belum memberikan penjelasan mendalam mengenai pertimbangan pemilihan dalam permasalahan deployments Argo CD itu sendiri.

1.2 Rumusan Masalah

Berdasarkan latar belakang yang telah diuraikan, terdapat beberapa permasalahan utama yang menjadi fokus penelitian ini, yaitu:

- a. Bagaimana mengimplementasikan mekanisme *continuous deployment* berbasis GitOps yang dapat secara otomatis melakukan *deployment* pada sistem Kubernetes ketika terjadi perubahan kode sumber *service*?
- b. Bagaimana merancang dan mengimplementasikan *infrastructure as code* untuk melakukan *provisioning* infrastruktur yang dibutuhkan oleh *microservice* secara otomatis pada lingkungan Kubernetes?

1.3 Tujuan Penelitian

Penelitian ini bertujuan untuk:

- a. Mengimplementasikan *continuous deployment* pada arsitektur *microservice* yang berjalan di Kubernetes dengan memanfaatkan prinsip-prinsip GitOps.
- b. Mengembangkan *workflow* terintegrasi antara *continuous integration* menggunakan GitHub Actions dengan ArgoCD sebagai operator GitOps, sebagai kelanjutan dari penelitian sebelumnya [27].

1.4 Batasan Masalah

Agar penelitian ini lebih terfokus, maka dilakukan pembatasan masalah sebagai berikut:

- a. Platform orkestrasi *container* yang digunakan adalah Kubernetes versi terbaru yang mendukung Custom Resource Definitions (CRDs).
- b. *Continuous Integration/Continuous Deployment* (CI/CD) diimplementasikan menggunakan GitHub Actions sebagai *workflow* otomatisasi.
- c. *Container runtime* yang digunakan adalah Docker dengan *image* yang kompatibel dengan standar OCI (*Open Container Initiative*).

- d. *Microservice* yang digunakan dalam pengujian adalah aplikasi berbasis web dengan arsitektur yang terdistribusi.
- e. *Version Control System* (VCS) yang digunakan adalah GitHub sebagai *single source of truth* untuk *infrastructure as code* dan kode aplikasi.



BAB II

TINJAUAN PUSTAKA

2.1 Penelitian Terdahulu

Dalam beberapa tahun terakhir, kebutuhan akan otomasi dan standarisasi proses deployment pada aplikasi berbasis microservices di lingkungan Kubernetes mengalami peningkatan yang signifikan. Hal ini sejalan dengan semakin kompleksnya arsitektur aplikasi modern yang menuntut efisiensi, keamanan, serta keandalan dalam pengelolaan siklus hidup perangkat lunak. Salah satu pendekatan yang berkembang pesat untuk menjawab tantangan tersebut adalah GitOps, di mana seluruh konfigurasi dan proses deployment didefinisikan secara deklaratif di dalam repository Git yang berfungsi sebagai sumber kebenaran utama (single source of truth). Argo CD merupakan salah satu tools GitOps yang banyak diadopsi oleh industri maupun komunitas open source karena kemampuannya dalam melakukan sinkronisasi otomatis antara repository Git dan cluster Kubernetes, serta menyediakan fitur visualisasi status deployment secara real-time.

Penelitian yang dilakukan oleh Korhonen [28] secara khusus mengevaluasi penerapan Argo CD dalam manajemen layanan berbasis Kubernetes. Dalam penelitian tersebut, Korhonen mengimplementasikan pipeline CI/CD dengan menggunakan GitLab sebagai sumber repository dan Argo CD sebagai alat continuous delivery pada cluster Kubernetes yang dibangun dengan MicroK8s. Hasil penelitian menunjukkan bahwa Argo CD mampu meningkatkan konsistensi konfigurasi antar lingkungan, memudahkan rollback saat terjadi kegagalan deployment, serta mempercepat proses delivery aplikasi melalui mekanisme automated synchronization. Selain itu, Argo CD juga dinilai mempermudah proses audit dan monitoring perubahan konfigurasi karena seluruh riwayat perubahan tercatat dengan baik di repository Git. Namun demikian, Korhonen juga menyoroti pentingnya perencanaan arsitektur dan pengelolaan dependensi layanan agar tidak terjadi konflik konfigurasi yang dapat menghambat proses deployment secara keseluruhan.

Studi lain yang dilakukan oleh Sharma et al. [29] membandingkan performa deployment antara Argo CD dan Flux sebagai tools GitOps pada lingkungan multi-cluster Kubernetes. Penelitian ini menggunakan beberapa parameter evaluasi

seperti kecepatan deployment, kemudahan integrasi dengan pipeline CI/CD, serta kemampuan visualisasi status aplikasi. Hasilnya, Argo CD dinilai lebih unggul dalam hal user experience, khususnya pada fitur visualisasi status deployment dan kemudahan integrasi dengan berbagai pipeline CI/CD yang umum digunakan di industri. Di sisi lain, Flux memiliki keunggulan pada fleksibilitas manajemen konfigurasi dan kemudahan dalam melakukan kustomisasi pada skenario deployment yang kompleks. Kedua tools ini sama-sama mampu mengurangi potensi human error, meningkatkan traceability, serta mempercepat proses deployment aplikasi secara signifikan jika dibandingkan dengan metode deployment manual atau konvensional.

Selain aspek fungsionalitas dan performa, isu keamanan dalam implementasi Argo CD juga menjadi perhatian dalam beberapa penelitian. Kumar [30] membahas secara mendalam tantangan keamanan yang dihadapi dalam penggunaan Argo CD, terutama terkait pengelolaan secrets dan akses kontrol pada cluster Kubernetes. Dalam penelitian tersebut, ditemukan bahwa praktik terbaik yang dapat diterapkan untuk meminimalisir risiko kebocoran data adalah dengan mengintegrasikan Argo CD dengan external secrets management seperti HashiCorp Vault atau AWS Secrets Manager, serta menerapkan prinsip least privilege pada setiap komponen yang terlibat dalam proses deployment. Selain itu, Kumar juga merekomendasikan penerapan audit log yang komprehensif dan pemantauan akses secara real-time untuk meningkatkan keamanan dan akuntabilitas sistem.

No	Insight	Hasil	Metode	Batasan	No Kutipan
1	Evaluasi Argo CD sebagai alat continuous delivery berbasis GitOps.	Argo CD meningkatkan konsistensi konfigurasi, memudahkan rollback, mempercepat delivery, dan mempermudah audit.	Studi kasus implementasi pipeline CI/CD dengan GitLab dan MicroK8s	Perlu perencanaan arsitektur dan pengelolaan dependensi agar tidak terjadi konflik konfigurasi	[28]
2	Komparasi performa deployment antara Argo CD dan Flux pada multi-cluster Kubernetes.	Argo CD unggul pada user experience dan visualisasi, Flux lebih fleksibel dalam manajemen konfigurasi. Keduanya meningkatkan traceability dan mengurangi human error.	Eksperimen komparatif pada beberapa parameter evaluasi (kecepatan, integrasi, visualisasi)	Hanya membandingkan dua tools GitOps utama, tidak membahas aspek keamanan secara mendalam	[29]
3	Isu keamanan pada implementasi Argo CD, khususnya pengelolaan secrets dan akses kontrol.	Integrasi dengan external secrets management dan penerapan least privilege meningkatkan keamanan. Audit log dan monitoring real-time direkomendasikan,	Studi literatur dan analisis praktik keamanan pada Argo CD	Fokus pada keamanan, belum membahas performa atau integrasi pipeline secara detail	[30]

Table 2.1 Rangkuman Penelitian Terdahulu

2.2 Kubernetes

Kubernetes, sering disingkat K8s, adalah sebuah platform open-source yang portabel dan dapat diperluas, yang dirancang untuk mengelola workload dan layanan yang dikontainerisasi. Awalnya dikembangkan oleh Google dan kini dikelola oleh Cloud Native Computing Foundation (CNCF) [18], Kubernetes telah menjadi standar industri untuk orkestrasi kontainer. Tujuan utamanya adalah menyediakan platform

yang mampu mengotomatisasi proses deployment, penskalaan, dan operasi aplikasi dalam kontainer, sehingga menyederhanakan kompleksitas pengelolaan aplikasi di berbagai lingkungan komputasi

2.3 Argo CD

Argo CD merupakan salah satu perangkat GitOps terkemuka yang telah diadopsi secara luas, baik di lingkungan industri maupun oleh komunitas *open source*. Popularitasnya didorong oleh kemampuannya untuk melakukan sinkronisasi secara otomatis antara repositori Git dengan klaster Kubernetes, seraya menyajikan visualisasi status *deployment* secara *real-time* [31]. Sebagai proyek yang bernaung di bawah Cloud Native Computing Foundation (CNCF), Argo CD menawarkan solusi *continuous delivery* yang bersifat deklaratif, dengan menjadikan Git sebagai satu-satunya sumber kebenaran (*single source of truth*) untuk seluruh konfigurasi aplikasi.

Arsitekturnya yang berbasis operator memungkinkan Argo CD untuk terus memantau aplikasi yang berjalan dan secara aktif menyalaskan keadaan aplikasi di klaster (*actual state*) agar selalu sesuai dengan konfigurasi yang diinginkan (*desired state*) di repositori Git. Beberapa fitur unggulannya mencakup dukungan untuk lingkungan *multi-tenant* dan Role-Based Access Control (RBAC), kemampuan untuk melakukan *rollback* ke versi sebelumnya, serta integrasi dengan berbagai sistem autentikasi seperti OAuth2.

2.4 GitOps

GitOps adalah pendekatan yang menggabungkan prinsip-prinsip Git (versi kontrol) dengan DevOps (operasionalisasi). Dengan GitOps, konfigurasi dan kode sumber aplikasi disimpan di repository Git, dan Argo CD (atau tools GitOps lainnya) digunakan untuk sinkronisasi otomatis antara repository Git dan cluster Kubernetes. Ini memungkinkan otomatisasi proses deployment dan meningkatkan traceability, auditability, dan keandalan dalam siklus hidup perangkat lunak [32].

2.5 Penelitian Lanjutan

Berdasarkan tinjauan terhadap penelitian-penelitian terdahulu pada **Tabel 2.1**, dapat disimpulkan bahwa penggunaan Argo CD dalam workflow GitOps membawa dampak positif yang signifikan terhadap efisiensi, keamanan, dan auditability proses deployment aplikasi berbasis microservices di Kubernetes. Namun demikian, masih terdapat ruang penelitian lebih lanjut terkait evaluasi komparatif antara metode pull-based deployment (seperti Argo CD) dengan metode push-based, serta analisis

mendalam mengenai best practice implementasi Argo CD pada skala enterprise, khususnya dalam konteks lingkungan produksi yang sangat dinamis dan kompleks. Oleh karena itu, penelitian ini berfokus pada analisis dan evaluasi penggunaan Argo CD sebagai operator GitOps pada Kubernetes, dengan tujuan memberikan rekomendasi strategis bagi organisasi yang ingin mengadopsi otomasi deployment berbasis GitOps secara optimal.

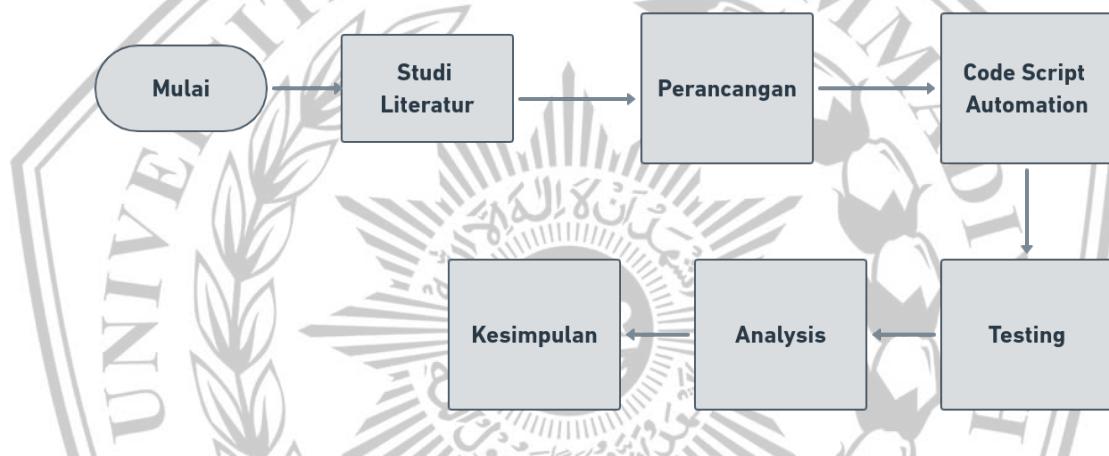


BAB III

METODOLOGI PENELITIAN

3.1 Rancangan Penelitian

Tipe Penelitian yang peneliti gunakan disini merupakan penelitian terapan langsung. Peneliti melakukan *development* secara langsung terhadap Argo CD yang akan digunakan pada sistem bare-metal (*non-cloud*) yang akan dikembangkan fitur continuous deployment dengan ArgoCD



Gambar 3.1 Rancangan Penelitian

3.1.1 Studi Literatur

Pada tahap studi literatur, peneliti mengumpulkan teori-teori, konsep, dan temuan penelitian terdahulu yang relevan sebagai landasan dalam melakukan pengembangan dan implementasi automasi deployment menggunakan Argo CD pada lingkungan Kubernetes. Studi literatur ini dilakukan dengan menelaah berbagai sumber seperti buku, jurnal nasional dan internasional, serta dokumentasi resmi terkait GitOps dan Argo CD.

Peneliti mempelajari konsep dasar GitOps yang menekankan penggunaan repository Git sebagai sumber kebenaran (single source of truth) untuk seluruh konfigurasi dan deployment aplikasi [33]. Selain itu, peneliti juga mengkaji dokumentasi resmi Argo CD yang menjelaskan fitur, arsitektur, serta best practice dalam penerapannya pada workflow CI/CD [31]. Penelitian terdahulu dari Korhonen [28] menjadi salah satu referensi utama, di mana dijelaskan penerapan Argo CD

untuk meningkatkan konsistensi, efisiensi, dan auditability proses deployment aplikasi pada Kubernetes.

Selain itu, peneliti juga menelaah studi komparatif antara Argo CD dan tools GitOps lain seperti Flux yang dilakukan oleh Sharma et al. [29], serta kajian terkait tantangan keamanan dalam implementasi Argo CD yang dibahas oleh Kumar [30]. Dengan mengkaji berbagai referensi tersebut, peneliti memperoleh pemahaman yang komprehensif mengenai teori dan praktik automasi deployment berbasis GitOps, sehingga dapat merancang dan mengimplementasikan solusi yang sesuai dengan kebutuhan penelitian.

3.1.2 Perancangan (Design)

Setelah melakukan tahap studi literatur dan analisis kebutuhan sistem, tahap selanjutnya adalah melakukan perancangan sistem automasi deployment yang akan diimplementasikan. Pada tahap studi literatur yang telah dilakukan, peneliti mengambil rancangan dasar dari penelitian-penelitian terdahulu, khususnya model GitOps yang dibahas oleh Ramadoni [27] dan arsitektur Argo CD yang dijelaskan oleh Korhonen [28].

Pada tahap ini, peneliti melakukan modifikasi terhadap rancangan yang sudah ada untuk mengadaptasi implementasi Argo CD pada lingkungan bare-metal (non-cloud) dengan memanfaatkan platform virtualisasi Proxmox. Modifikasi ini penting dilakukan karena sebagian besar implementasi GitOps dan Argo CD yang ada di literatur berfokus pada lingkungan cloud [20], sementara penelitian ini bertujuan untuk mengimplementasikan solusi serupa pada infrastruktur on-premise. Perancangan yang dilakukan meliputi beberapa aspek utama, yaitu:

1. Perancangan arsitektur sistem automasi deployment menggunakan Argo CD pada lingkungan Kubernetes yang berjalan di atas Proxmox
2. Perancangan alur kerja sistem sesuai dengan pendekatan pull-based deployment yang merupakan karakteristik utama dari GitOps [33]
3. Perancangan mekanisme continuous integration yang terintegrasi dengan Argo CD untuk membentuk pipeline CI/CD yang lengkap
4. Perancangan strategi deployment yang efektif untuk aplikasi berbasis microservice

3.1.3 Implementasi Code Automasi (Coding)

Tahap implementasi melibatkan pengembangan beberapa komponen kunci yang akan menjadi fondasi sistem. Implementasi dilakukan dengan bahasa pemrograman Go untuk microservices dan YAML untuk konfigurasi Kubernetes. Berikut adalah rincian implementasi yang dilakukan:

a. Pengembangan Microservices

- **Auth Service:** Mengimplementasikan autentikasi berbasis JWT dengan endpoint untuk login dan validasi token
- **Color Service:** Menyediakan API untuk menghasilkan warna acak
- **Prime Generator Service:** Mengimplementasikan algoritma generasi bilangan prima dengan optimasi Sieve of Eratosthenes
- **UI Service:** Dibangun dengan HTMX untuk interaktivitas, bertanggung jawab menampilkan antarmuka pengguna

b. Infrastruktur Kubernetes

- Konfigurasi manifest Kubernetes untuk setiap komponen
- Pembuatan Namespace, Deployment, Service, dan Ingress
- Konfigurasi RBAC (Role-Based Access Control) untuk keamanan
- Setup Network Policies untuk mengontrol komunikasi antar servis

c. CI/CD Pipeline

- Implementasi GitHub Actions untuk otomatisasi build dan push image
- Konfigurasi ArgoCD untuk continuous deployment
- Setup Cloudflare Tunnel untuk akses aman ke cluster

Setiap komponen diimplementasikan dengan mempertimbangkan prinsip-prinsip cloud native seperti stateless design, horizontal scalability, dan loose coupling. Kode sumber dikelola dalam repository Git terpusat dengan struktur yang terorganisir untuk memudahkan kolaborasi dan maintenance. Implementasi ini menghasilkan arsitektur yang siap di-deploy ke lingkungan Kubernetes yang telah disiapkan.

3.1.4 Pengujian dan Analisis (Testing dan Analysis)

Dalam pengujian dan analisis hasil dari implementasi yang dibuat akan dilakukan tahap pengujian terlebih dahulu. Pengujian dan analisis dilakukan untuk mencari dan mengidentifikasi kesalahan pada sebuah sistem. Pengujian dilakukan dengan pendekatan sebagai berikut:

a. Unit Testing

Dilakukan pada setiap komponen microservice untuk memastikan setiap fungsi bekerja sesuai harapan. Pengujian mencakup:

- Pengujian fungsi autentikasi pada Auth Service
- Pengujian logika generasi bilangan prima pada Prime Generator Service
- Pengujian fungsi pengacak warna pada Color Service

b. Integration Testing

Pengujian interaksi antar komponen dalam sistem:

- Komunikasi antara UI Service dengan Auth Service untuk proses login
- Interaksi antara UI Service dengan Color Service untuk menampilkan warna acak
- Integrasi UI Service dengan Prime Generator Service untuk menampilkan bilangan prima

c. Black-box Testing

Pengujian fungsionalitas sistem secara keseluruhan dari perspektif pengguna akhir:

- Pengujian alur login/logout
- Verifikasi tampilan halaman utama
- Pengujian fitur generate bilangan prima
- Pengujian tampilan warna acak

d. Continuous Deployment Testing

Pengujian alur deployment otomatis menggunakan ArgoCD:

- Verifikasi sinkronisasi otomatis antara repository Git dengan cluster Kubernetes

- Pengujian rollback otomatis saat terjadi kegagalan deployment
- Validasi health check dan readiness probe pada setiap service

Setiap temuan dari pengujian didokumentasikan dan dianalisis untuk perbaikan lebih lanjut. Kriteria keberhasilan pengujian diukur berdasarkan persentase test case yang berhasil dilalui dan umpan balik dari pengguna akhir.

3.1.5 Kesimpulan

Penyajian kesimpulan merupakan babak final dari keseluruhan proses penelitian. Pada tahap krusial ini, fokus tidak lagi hanya pada pemaparan data, melainkan pada sintesis dan interpretasi untuk menarik makna yang lebih dalam dari seluruh temuan. Seluruh hasil analisis dan evaluasi yang telah dilakukan dirangkai untuk menjawab rumusan masalah yang menjadi inti dari penelitian ini. Dengan demikian, bagian ini merumuskan kontribusi utama dari penelitian, menegaskan implikasi teoretis maupun praktisnya, serta menyajikan rekomendasi strategis sebagai landasan bagi implementasi atau pengembangan riset di masa mendatang.



BAB IV

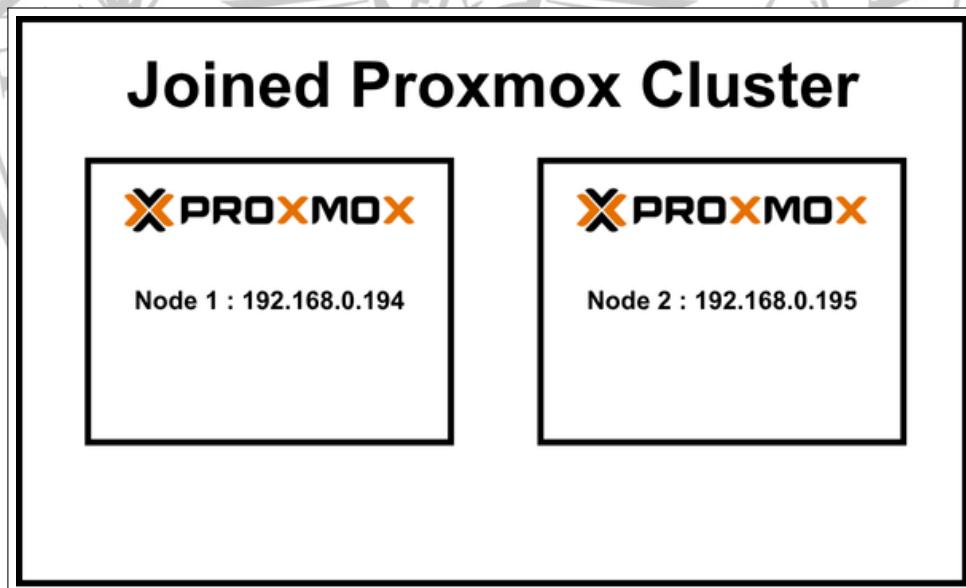
HASIL DAN PEMBAHASAN

4.1 Perancangan (Design)

Pada tahap pertama ini peneliti akan melakukan perancangan sebuah infrastruktur dimana sistem Kubernetes dan ArgoCD akan dijalankan lalu microservice untuk dilakukan simulasi implementasi pada ArgoCD. Semua rancangan akan menggunakan visualisasi diagram secara garis besar (high-level) agar mudah dipahami.

4.1.1 Perancangan Sistem Arsitektur Infrastruktur Mesin

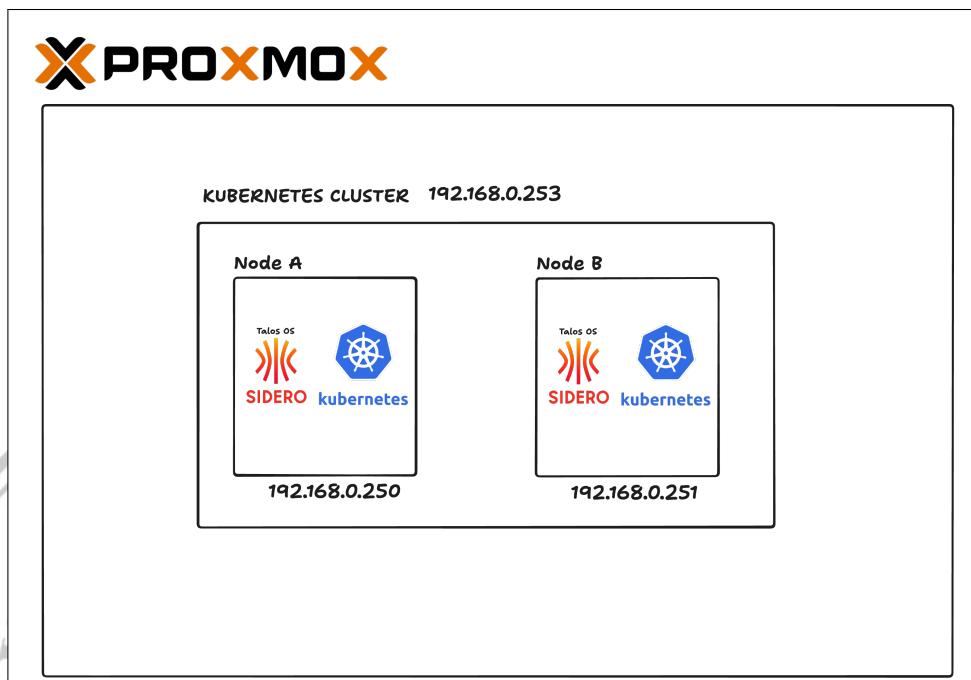
Sebelum dilakukan penelitian lebih lanjut pada implementasi ArgoCD pada Kubernetes Cluster perlu dirancang terlebih dahulu arsitektur infrastruktur mesin yang akan menjadi tempat instalasi Kubernetes Cluster. Peneliti menggunakan Proxmox VE (Virtual Environment) yang didalamnya terdapat Virtual Machine berupa Talos OS Linux yang siap dilakukan instalasi Kubernetes Cluster.



Gambar 4.1 Arsitektur Infrastruktur High Level

Pada **Gambar 4.1** menunjukkan arsitektur infrastruktur mesin secara high level dimana terdapat 2 cluster proxmox yang *joined* agar bisa saling terhubung satu sama lain. Akses terhadap proxmox tersebut dapat dilakukan menggunakan

port 8006 dengan menggunakan browser atau dengan kata lain bisa diakses dari <https://192.168.0.194:8006/> atau <https://192.168.0.195:8006/>

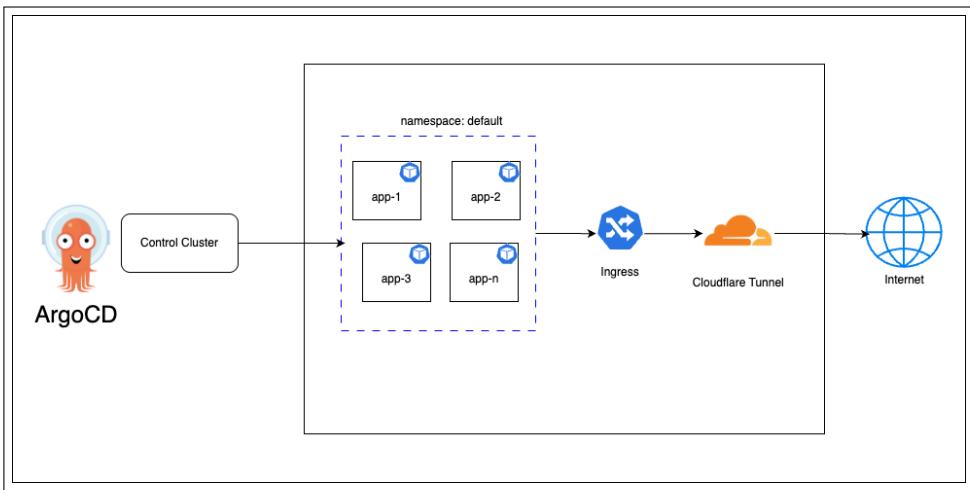


Gambar 4.2 Arsitektur Virtual Machine Talos OS pada Proxmox Cluster

Setelah dilakukan *provisioning* pada proxmox tersebut akan dilakukan instalasi Talos OS pada node tersebut menggunakan dengan memanfaatkan *virtual machine* yang ada pada proxmox. Pada **Gambar 4.2** merupakan arsitektur pada kubernetes cluster itu sendiri dimana sistem virtual machine yang didalamnya terdapat komponen Talos OS dan komponen kubernetes. Kubernetes Cluster dapat diakses menggunakan address 192.168.0.253:6443 menggunakan kubectl didalam address tersebut terdapat 2 node Talos OS yang masing-masing memiliki komponen kubernetes node.

4.1.2 Perancangan Arsitektur ArgoCD pada Kubernetes Cluster

Pada bagian sebelumnya peneliti sudah memaparkan arsitektur infrastruktur mesin dimana Kubernetes cluster akan berjalan. Pada tahap ini peneliti akan memaparkan arsitektur pada kubernetes cluster nya itu sendiri dimana ArgoCD akan berjalan.

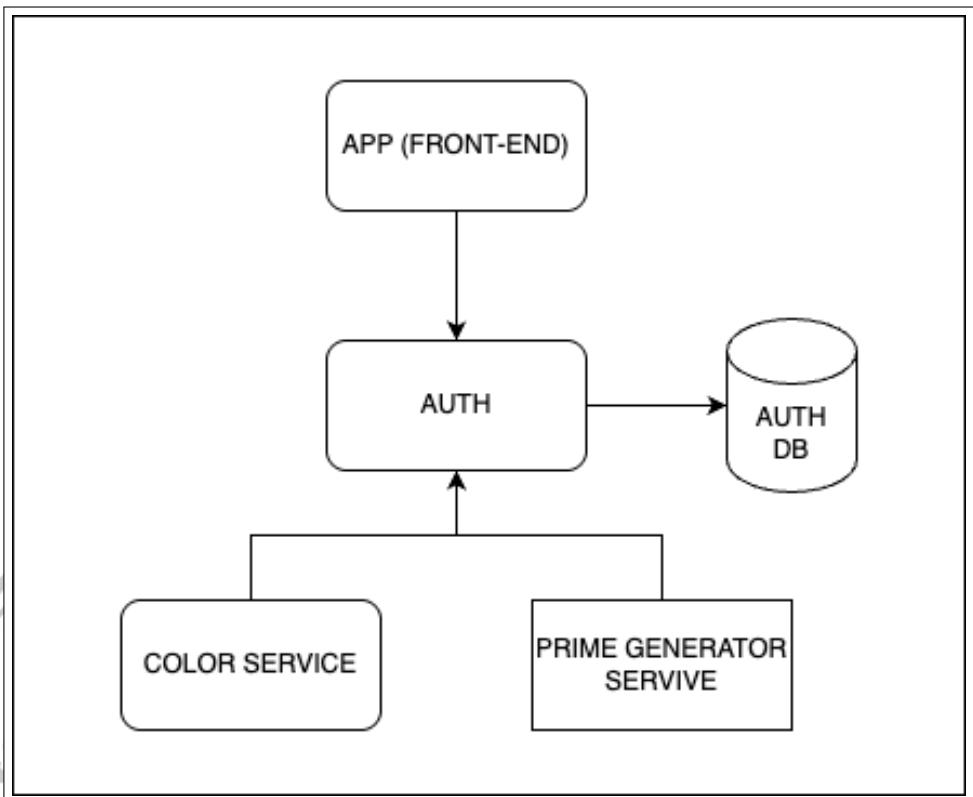


Gambar 4.3 Arsitektur Kubernetes High Level

Pada **Gambar 4.3** mengambarkan arsitektur kubernetes nya secara high level dimana terdapat komponen ArgoCD dan Cloudflare Tunnel yang bertujuan agar server microservice bisa diakses pada world wide web (internet). Secara garis besar sebuah service digambarkan sebagai app yang terdeploy pada sebuah namespace yang terdapat pada kubernetes cluster. Lalu *ingress* akan digunakan untuk mengakses service tersebut dari luar cluster dengan menggunakan domain yang sudah terdaftar pada cloudflare.

4.1.3 Perancangan Arsitektur Microservice pada Kubernetes Cluster

Pada bagian ini peneliti akan memaparkan contoh design arsitektur microservice yang akan dijadikan acuan untuk dilakukan implementasi pada kubernetes cluster. Nantinya service service ini akan berdiri sendiri dan mempunya deployment sendiri pada kubernetes cluster. Deployment service tersebut akan dideploy menggunakan ArgoCD agar bisa terdeploy kedalam kubernetes cluster.



Gambar 4.4 Arsitektur Microservice High Level

Pada **Gambar 4.4** menggambarkan arsitektur microservice yang akan dideploy pada kubernetes cluster menggunakan Argo CD. Microservice tersebut terdiri dari beberapa service yaitu service frontend yang akan berfungsi sebagai interface untuk user dan service backend yang akan diakses. Terdapat juga service auth yang akan berfungsi sebagai autentikasi dan otorisasi user sebelum bisa mengakses service backend yaitu color service dan prime generator service. Auth service sendiri bersifat stateless komponen auth db pada arsitektur digambarkan untuk menyimpan data user yang akan diakses oleh service auth. Color service dan prime generator service sendiri juga bersifat stateless untuk mempermudah implementasi microservice tersebut.

4.2 Pengkodean (Coding) / Implementasi

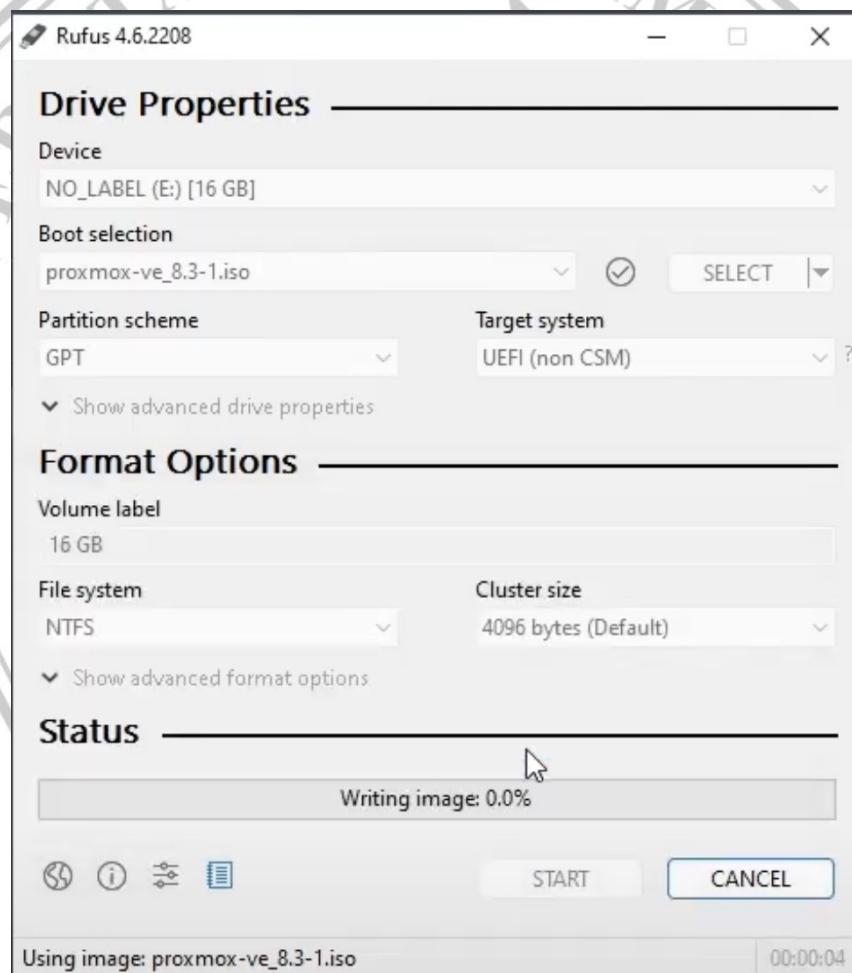
Tahap ini peneliti akan menjabarkan secara rinci pengkodean/implementasi rancangan sistem yang sudah dirancang sebelumnya pada bagian **4.1 Perancangan (Design)**. Tahap implementasi akan dijelaskan secara berurutan sesuai dengan urutan yang dijabarkan sebelumnya pada bagian **4.1 Perancangan (Design)**.

4.2.1 Implementasi Sistem Infrastruktur Mesin

Untuk implementasi infrastruktur mesin. Peneliti akan melakukan instalasi proxmox pada 2 mesin dengan arsitektur CPU X86. **Tabel 4.1** merupakan spesifikasi node yang akan digunakan pada penelitian ini.

Node	Host	OS	CPU (Core)	RAM	Kubernetes Version
master-1	192.168.0.194	Talos OS 1.9.5	4	12 GB	v1.28.0
worker-1	192.168.0.195	Talos OS 1.9.5	4	12 GB	v1.28.0

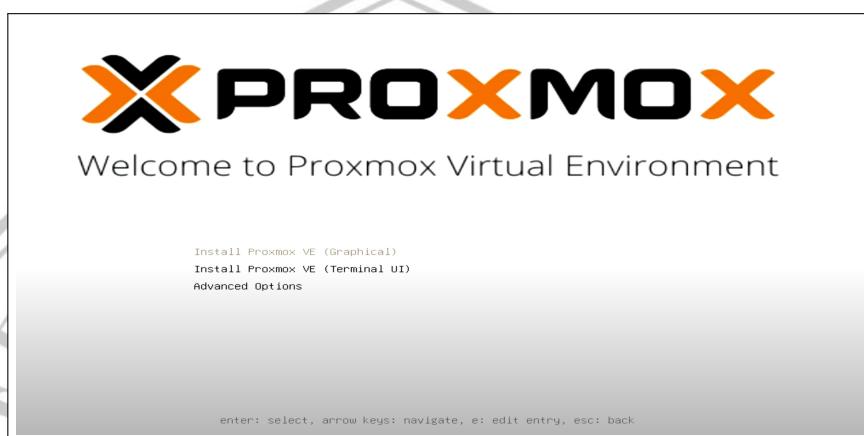
Table 4.1 Spesifikasi Node Kubernetes



Gambar 4.5 Pembuatan Bootable Proxmox Menggunakan Rufus

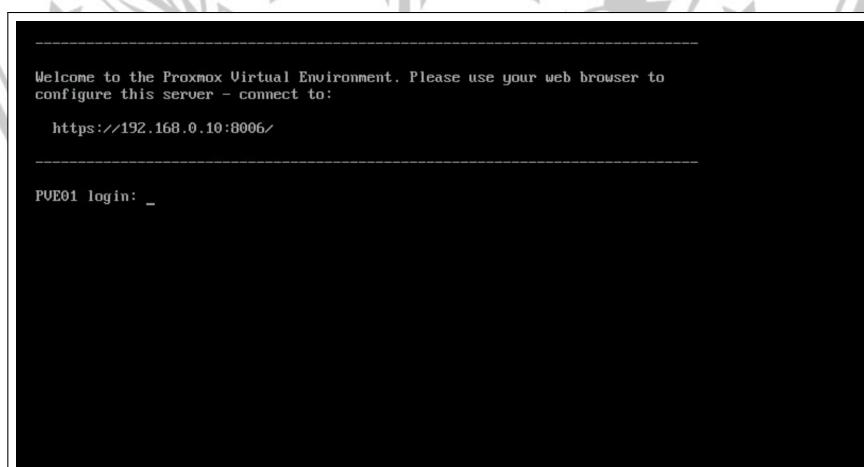
Pertama peneliti akan download ISO file atau installer proxmox yang terdapat di link ini https://enterprise.proxmox.com/iso/proxmox-ve_8.4-1.iso.

Setelah itu peneliti memerlukan sebuah Flashdisk atau media untuk bootable ISO tersebut disini peneliti menggunakan tools yang bernama Rufus seperti pada **Gambar 4.5**. Setelah itu kita perlu mengganti bootable menu yang mengarah pada flashdisk atau media yang kita gunakan untuk instalasi proxmox ketika booting BIOS pada mesin yang digunakan. Setelah itu mesin akan restart dan akan masuk pada tampilan seperti pada **Gambar 4.6**.



Gambar 4.6 Tampilan Awal Instalasi Proxmox VE

Selanjutnya kita tinggal mengikuti apa yang diarahkan secara default oleh user interface yang ditampilkan hingga mesin akan restart dan berada pada tampilan seperti pada **Gambar 4.7**. Pada tampilan layar tersebut terdapat alamat server yang bisa kita gunakan untuk akses user interface melalui browser.



Gambar 4.7 Tampilan Terminal Setelah Instalasi Proxmox Selesai

Akses UI pada komputer yang ada pada network yang sama dengan Proxmox

melalui web alamat server yang terlihat pada gambar **Gambar 4.7** maka akan muncul tampilan seperti pada **Gambar 4.8**. Peneliti akan mengulang implementasi ini untuk mesin kedua dengan step yang sama seperti pada mesin pertama.

Type	Description	Disk usage...	Mem
node	PVE01	2.8 %	4.2 %
sdn	localnetwork (PVE01)		
storage	local (PVE01)	2.8 %	
storage	local-lvm (PVE01)	0.0 %	

Gambar 4.8 Tampilan Web Interface Proxmox VE

Pada tahap selanjutnya akan dilakukan instalasi Talos OS yang akan di-install menggunakan VM yang ada pada Proxmox VE. Pertama yang dilakukan adalah mengunduh ISO file Talos OS di sini.

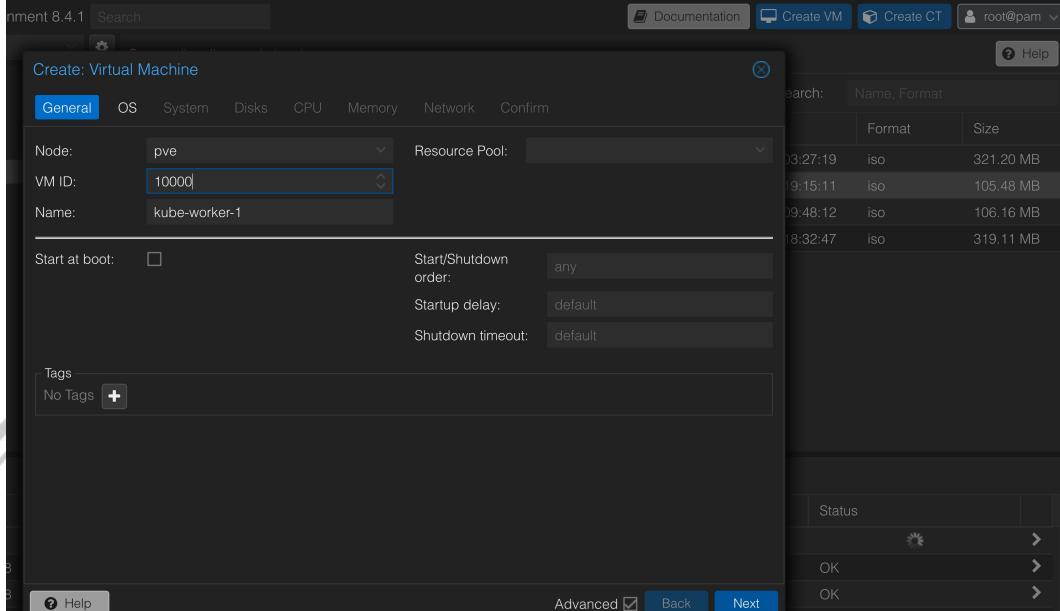
<https://github.com/siderolabs/talos/releases/download/v1.9.5/metal-amd64.iso>

Gambar 4.9 Proses Download ISO Talos OS

Url tersebut lalu didownload melalui proxmox agar tersimpan didalam proxmox seperti pada **Gambar 4.9**. Lalu tahap selanjutnya adalah melakukan instalasi VM Talos OS pada proxmox. Tahap ini adalah bagian instalasi VM Talos

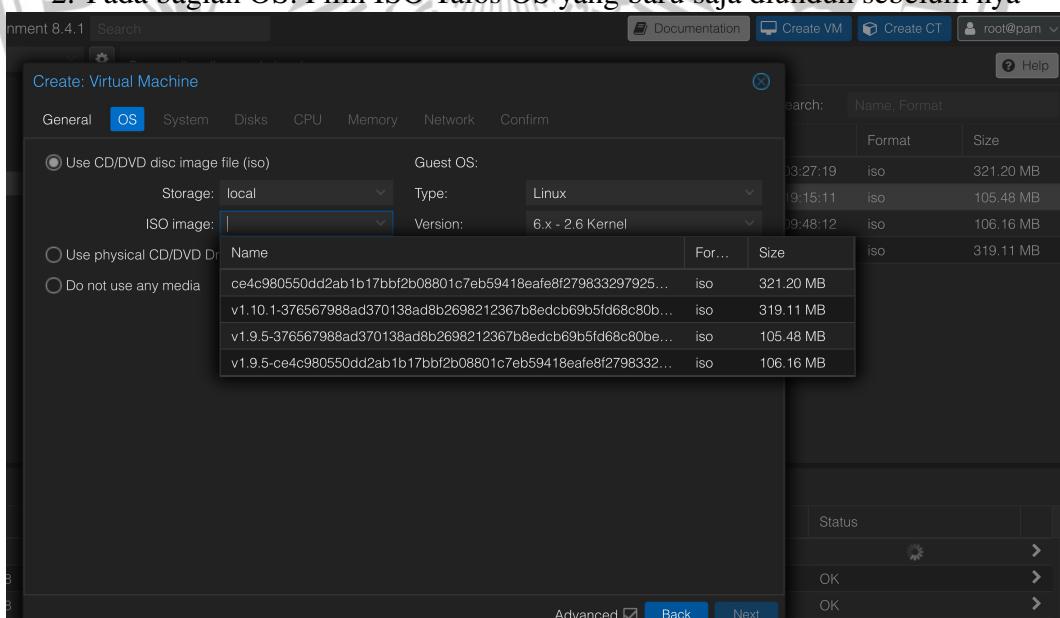
OS. Peneliti melakukan instalasi pada proxmox VE melalui interface web. Tahap ini akan dilakukan pada 2 mesin berbeda pada proxmox. Tahap-tahap instalasi Talos OS tersebut dijabarkan secara berurutan yang ditunjukkan pada **Gambar 4.10 - 4.18**

1. Klik Create VM lalu isikan VM ID dan Name untuk VM Talos OS



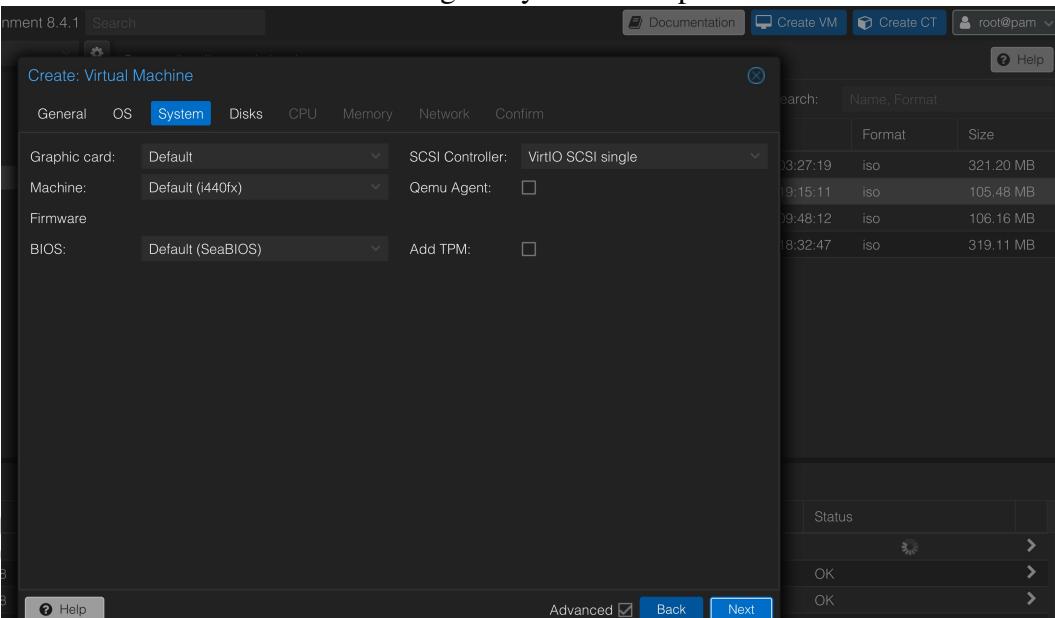
Gambar 4.10 Instalasi Talos OS 1

2. Pada bagian OS. Pilih ISO Talos OS yang baru saja diunduh sebelumnya



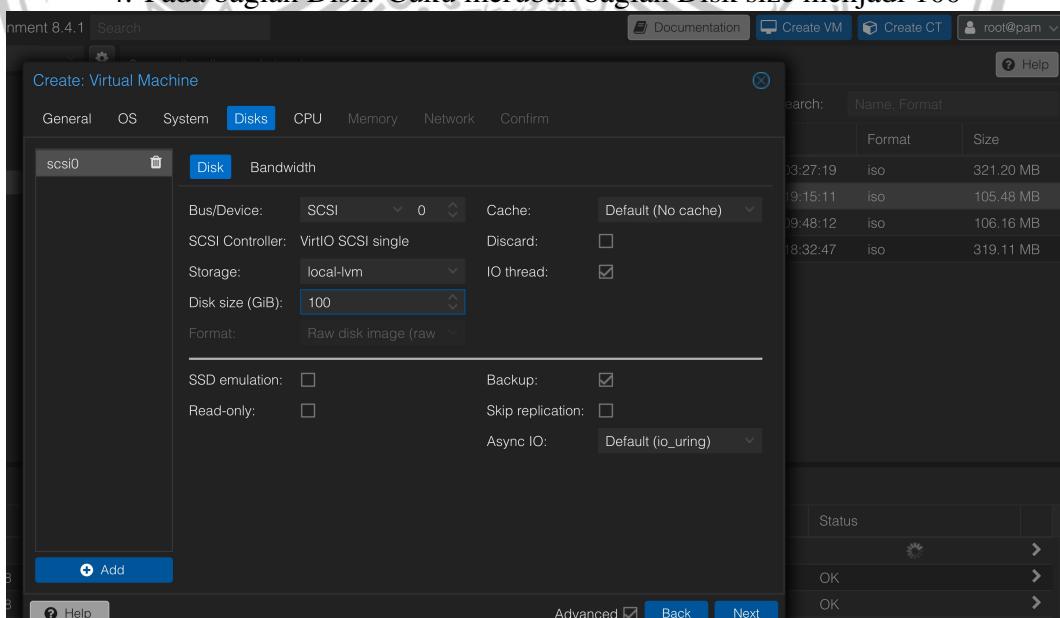
Gambar 4.11 Instalasi Talos OS 2

3. Pada bagian System. cukup next



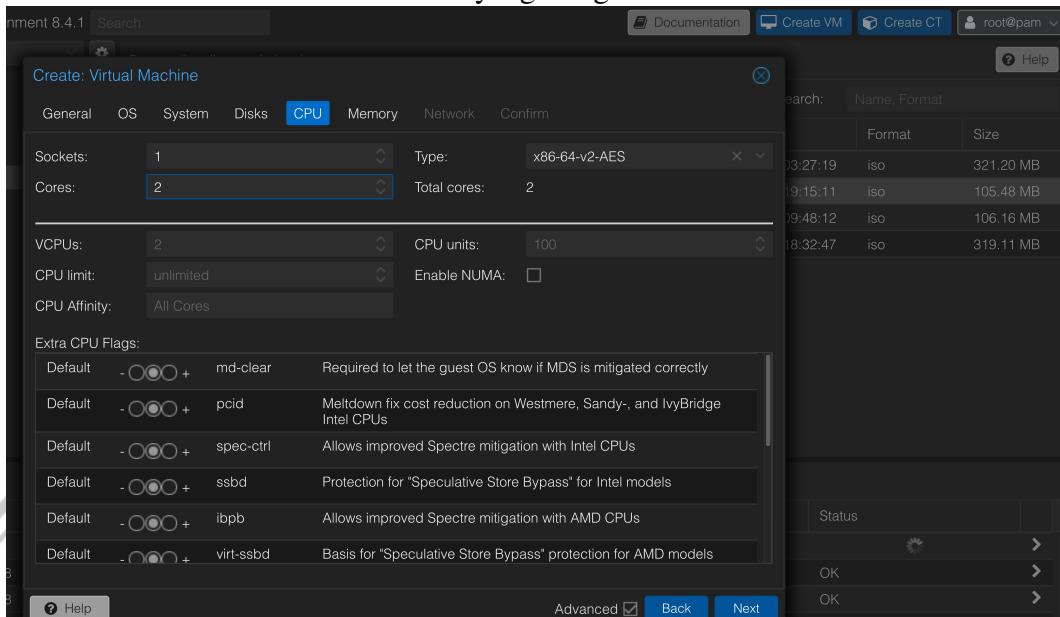
Gambar 4.12 Instalasi Talos OS 3

4. Pada bagian Disk. Cukup merubah bagian Disk size menjadi 100



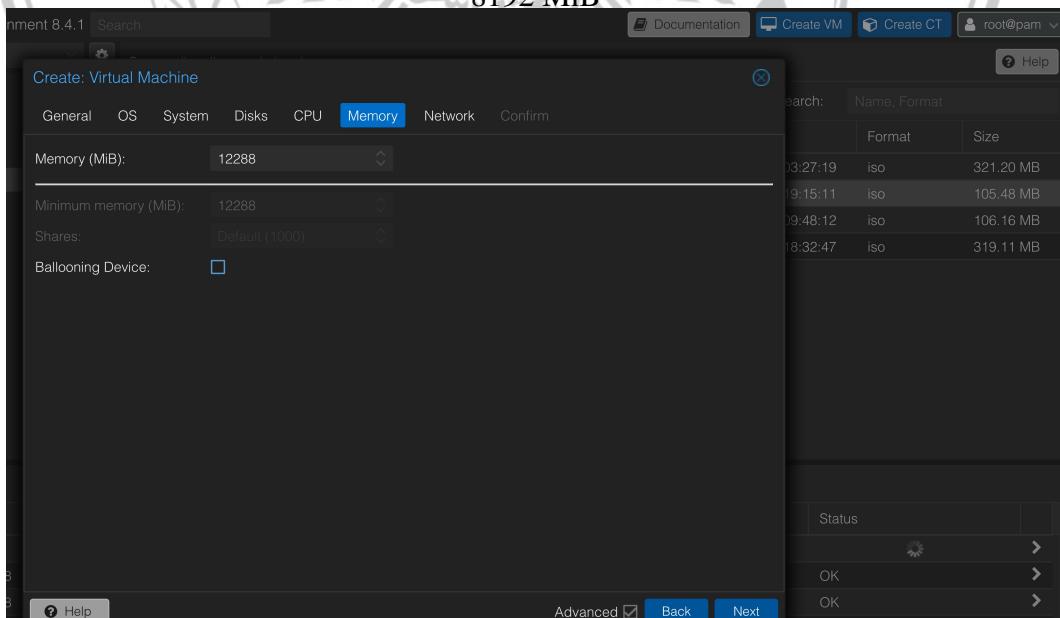
Gambar 4.13 Instalasi Talos OS 4

5. Pada bagian CPU. Cukup merubah cores menjadi 2 atau 4 sesuai spesifikasi mesin yang diinginkan



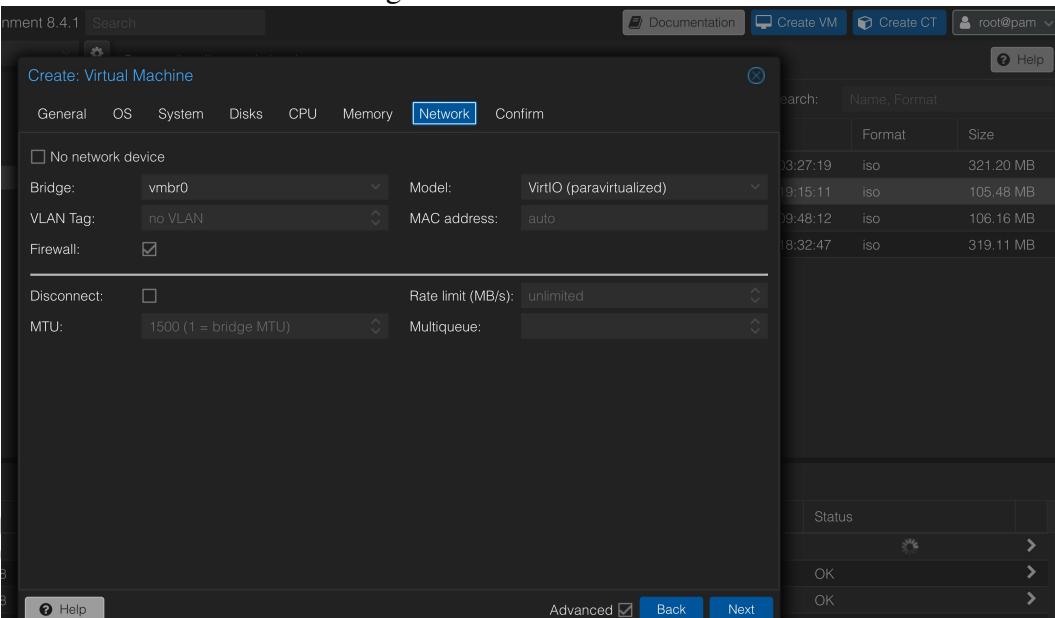
Gambar 4.14 Instalasi Talos OS 5

6. Pada bagian memory. Isikan memory sesuai yang dinginkan dengan minimal 8192 MiB



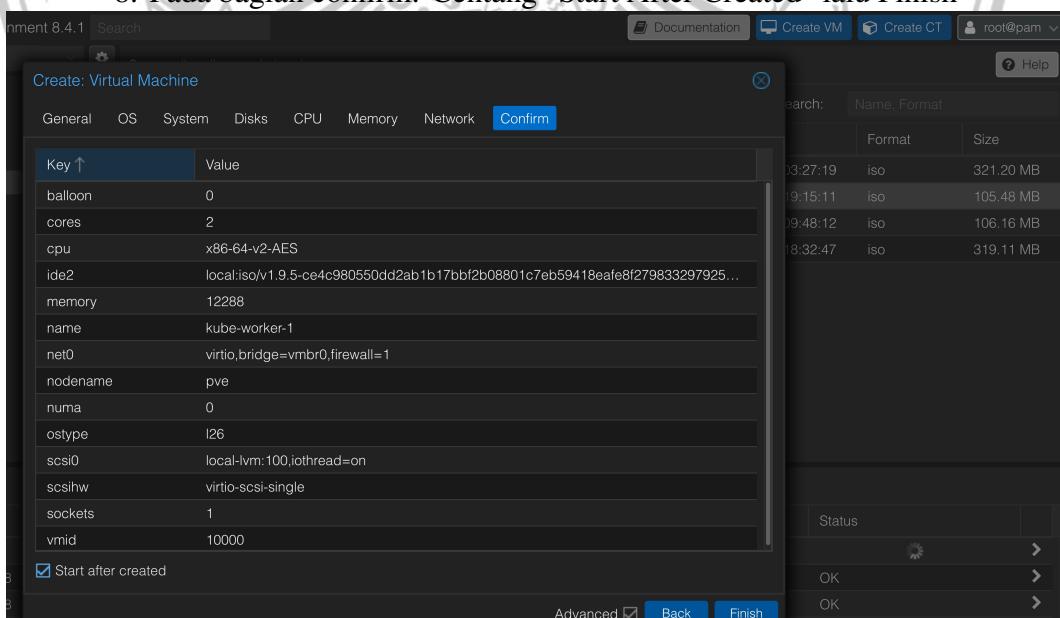
Gambar 4.15 Instalasi Talos OS 6

7. Pada bagian Network. Silahkan klik next



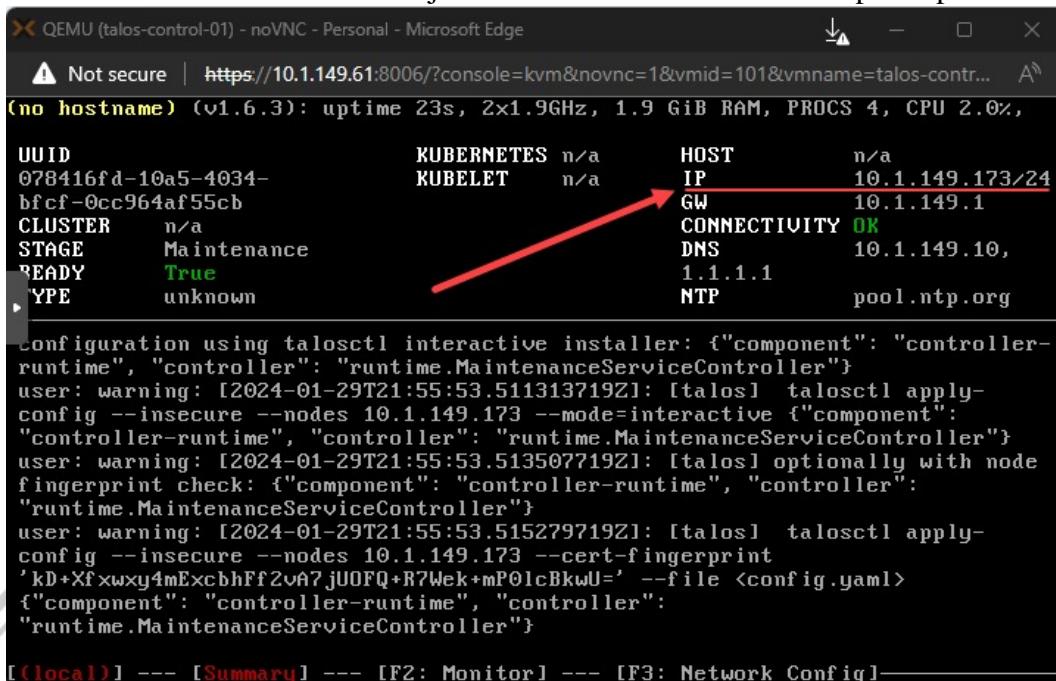
Gambar 4.16 Instalasi Talos OS 7

8. Pada bagian confirm. Centang "Start After Created" lalu Finish



Gambar 4.17 Instalasi Talos OS 8

10. Ketika Talos OS Sudah berjalan terminal Talos OS akan tampak seperti ini



KUBERNETES	n/a	HOST	n/a
KUBELET	n/a	IP	10.1.149.173/24
CLUSTER	n/a	GW	10.1.149.1
STAGE	Maintenance	CONNECTIVITY	OK
READY	True	DNS	10.1.149.10,
TYPE	unknown	NTP	1.1.1.1 pool.ntp.org

```
configuration using talosctl interactive installer: {"component": "controller-runtime", "controller": "runtime.MaintenanceServiceController"}  
user: warning: [2024-01-29T21:55:53.511313719Z]: [talos] talosctl apply-config --insecure --nodes 10.1.149.173 --mode=interactive {"component": "controller-runtime", "controller": "runtime.MaintenanceServiceController"}  
user: warning: [2024-01-29T21:55:53.513507719Z]: [talos] optionally with node fingerprint check: {"component": "controller-runtime", "controller": "runtime.MaintenanceServiceController"}  
user: warning: [2024-01-29T21:55:53.515279719Z]: [talos] talosctl apply-config --insecure --nodes 10.1.149.173 --cert-fingerprint 'kD+Xfxwxy4mExcbehFf2u7jU0FQ+R?Wek+mP0lcBkwU=' --file <config.yaml> {"component": "controller-runtime", "controller": "runtime.MaintenanceServiceController"}  
[ (local) ] --- [Summary] --- [F2: Monitor] --- [F3: Network Config]
```

Gambar 4.18 Instalasi Talos OS

4.2.1.1 Implementasi Instalasi Kubernetes pada Talos OS

Tahap ini adalah tahap untuk instalasi Kubernetes pada Talos OS. Disini peneliti akan menggunakan deklaratif yaml yang akan mengautomatisasi instalasi kubernetes pada Talos OS. Instalasi Kubernetes pada Talos OS sendiri mengikuti panduan dari pedoman yang terdapat pada website Talos OS <https://www.talos.dev/v1.9.5/introduction/getting-started/>.

Kode Program 4.1 menunjukkan konfigurasi script instalasi Kubernetes cluster pada Talos OS terdapat beberapa langkah yang akan dijalankan yaitu dengan melakukan generate secret, generate config, generate command apply, generate command bootstrap, dan generate command kubeconfig yang akan digunakan sebagai konfigurasi dasar untuk instalasi Kubernetes cluster pada Talos OS hasil dari generate config tersebut ditunjukan pada **Kode Program 4.2**.

```
1 version: '3'  
2 tasks:  
3   talos:  
4     desc: Bootstrap the Talos cluster  
5     dir: '{{.TALOS_DIR}}'  
6     cmds:  
7       - '[ -f talsecret.sops.yaml ] || talhelper gensecret | sops
```

```

    --filename-override talos/talsecret.sops.yaml --encrypt
    /dev/stdin > talsecret.sops.yaml'
8   - talhelper genconfig
9   - talhelper gencommand apply --extra-flags="--insecure" |
10    bash
10  - until talhelper gencommand bootstrap | bash; do sleep 10;
11    done
11  - until talhelper gencommand kubeconfig
12    --extra-flags="{{.ROOT_DIR}} --force" | bash; do sleep
13    10; done

```

Kode Program 4.1: Konfigurasi script instalasi Kubernetes cluster pada Talos OS

```

1  clusterName: kubernetes
2  endpoint: https://192.168.0.250:6443
3  ...
4  nodes:
5    - hostname: "kube-cp-1"
6      ipAddress: "192.168.0.200"
7      installDisk: "/dev/sda"
8      machineSpec:
9      controlPlane: true
10     networkInterfaces:
11       - deviceSelector:
12         hardwareAddr: "de:ad:be:ef:00:01"
13         addresses:
14           - "192.168.0.200/24"
15         routes:
16           - network: "0.0.0.0/0"
17             gateway: "192.168.0.1"
18         mtu: 1500
19         vip:
20           ip: "192.168.0.250"
21    - hostname: "kube-worker-1"
22      ipAddress: "192.168.0.201"
23      installDisk: "/dev/sda"
24      machineSpec:
25      controlPlane: false
26      networkInterfaces:
27        - deviceSelector:
28          hardwareAddr: "de:ad:be:ef:00:02"
29          dhcp: false
30          addresses:
31            - "192.168.0.201/24"

```

```

32      routes:
33        - network: "0.0.0.0/0"
34          gateway: "192.168.0.1"
35          mtu: 1500

```

Kode Program 4.2: Konfigurasi node Kubernetes cluster pada Talos OS

Setelah dilakukan instalasi Kubernetes cluster pada Talos OS tampilan terminal akan menampilkan nama cluster dan juga sudah tidak menampilkan status Stage: Maintenance melainkan sudah menampilkan Kubelet: Healthy seperti **Gambar 4.19.**

```

kube-control-1 (v1.9.5): uptime 41m25s, 4x3.49GHz, 14 GiB RAM, PROCS 63, CPU
                         TYPE           HOST
UUID                controlplane   IP
8aac4351-4534-49e3-
ab03-4042883e506b   KUBERNETES    192.168.0.200/24,
CLUSTER            kubernetes (2  v1.32.3     192.168.0.250/32
machines)          KUBELET       GW 192.168.0.1
SIDEROLINK n/a     ✓ Healthy    CONNECTIVITY ✓ OK
Logs
"runtime.MachineStatusController"}
kern:  info: [2025-05-15T19:58:23.891235046Z]: eth0: renamed from tmpf194c
kern:  info: [2025-05-15T19:58:23.935322046Z]: eth0: renamed from tmpcbea6
kern:  info: [2025-05-15T19:58:23.936710046Z]: eth0: renamed from tmpd7271
kern:  info: [2025-05-15T19:58:23.963162046Z]: eth0: renamed from tmpd3bd
kern:  info: [2025-05-15T19:58:24.023889046Z]: eth0: renamed from tmpbaa39
kern:  info: [2025-05-15T19:58:24.091156046Z]: eth0: renamed from tmp6fd81
kern:  info: [2025-05-15T19:58:24.096220046Z]: eth0: renamed from tmpa93f1
kern:  info: [2025-05-15T19:58:24.119990046Z]: eth0: renamed from tmp5e2b7
kern:  info: [2025-05-15T19:58:24.120383046Z]: eth0: renamed from tmp17793
kern:  info: [2025-05-15T19:58:24.151169046Z]: eth0: renamed from tmp77f0e
kern:  info: [2025-05-15T19:58:24.153420046Z]: eth0: renamed from tmp76b39
kern:  info: [2025-05-15T19:58:49.178916046Z]: eth0: renamed from tmp65ac9
kern:  info: [2025-05-15T20:01:42.044436046Z]: eth0: renamed from tmpbc3b8

```

Gambar 4.19 Tampilan Talos OS setelah instalasi kubernetes cluster

Setelah instalasi kubernetes pada Talos OS dilakukan maka kita dapat mengakses kubernetes cluster tersebut menggunakan cli kubectl. Sebagai contoh **Gambar 4.20** adalah aplikasi dummy yang menampilkan informasi header pada browser pada echo.zeinfahrozi.my.id.

Gambar 4.20 Tampilan echo.zeinfahrozi.my.id pada browser

4.2.2 Implementasi ArgoCD pada Kubernetes Cluster

Tahap ini merupakan instalasi instance ArgoCD itu sendiri pada kubernetes cluster yang sudah dibuat sebelumnya. ArgoCD dipasang menggunakan Helm chart dengan konfigurasi kustom yang disesuaikan dengan kebutuhan lingkungan produksi. Berikut adalah contoh perintah untuk menginstal ArgoCD menggunakan Helm:

```

1 # Menambahkan repo ArgoCD
2 helm repo add argo https://argoproj.github.io/argo-helm
3 helm repo update
4
5 # Membuat namespace untuk ArgoCD
6 kubectl create namespace argocd
7
8 # Menginstal ArgoCD dengan Helm
9 helm upgrade --install argocd argo/argo-cd \
10   --namespace argocd \
11   --values values.sops.yaml \
12   --wait

```

Kode Program 4.3: Perintah Instalasi ArgoCD menggunakan Helm

Pada **Kode Program 4.3** menunjukkan perintah instalasi ArgoCD menggunakan Helm charts dengan nilai-nilai kustom yang didefinisikan dalam file `values.sops.yaml` yang ada pada **Kode Program 4.4**. Beberapa konfigurasi

penting yang diterapkan:

- a. ArgoCD diakses melalui domain `argo.zeinfahrozi.my.id`
- b. Mode `insecure` diaktifkan untuk pengembangan
- c. Fitur status badge diaktifkan untuk memantau status aplikasi
- d. Dukungan untuk multiple value files dengan skema yang berbeda
- e. Eksklusi sumber daya tertentu seperti CiliumIdentity dari manajemen ArgoCD

```
# values.sops.yaml
1 crds:
2   install: true
3
4 global:
5   domain: argo.zeinfahrozi.my.id
6
7 configs:
8   params:
9     server.insecure: true
10 cm:
11   statusbadge.enabled: true
12   kustomize.buildOptions: --enable-alpha-plugins --enable-exec
13   helm.valuesFileSchemes: >-
14     secrets+gpg-import,secrets+gpg-import-kubernetes,
15     secrets+age-import,secrets+age-import-kubernetes,
16     secrets,secrets+literal,https
17 resource.exclusions: |
18   - apiGroups:
19     - cilium.io
20   kinds:
21     - CiliumIdentity
22 clusters:
23   - '*'
```

Kode Program 4.4: Contoh konfigurasi `values.sops.yaml` untuk ArgoCD

Setelah ArgoCD terinstal, aplikasi-aplikasi dapat dikelola menggunakan GitOps. Setiap aplikasi didefinisikan sebagai kustom resource Kubernetes yang mereferensikan repositori Git yang berisi manifest Kubernetes. ArgoCD akan secara otomatis melakukan sinkronisasi antara status yang diinginkan (yang didefinisikan di

Git) dengan status aktual di cluster. Beberapa aspek keamanan yang diterapkan pada instalasi ArgoCD ini antara lain:

- a. Penggunaan HTTPS untuk akses web UI
- b. Integrasi dengan Dex untuk autentikasi
- c. Pembatasan akses berbasis peran (RBAC)
- d. Penyimpanan rahasia yang aman menggunakan SOPS

Dengan konfigurasi ini, ArgoCD siap digunakan untuk mengelola aplikasi secara deklaratif menggunakan prinsip GitOps, di mana semua perubahan konfigurasi dilakukan melalui pull request dan version control system.

4.2.2.1 Implementasi Cloudflare Tunnel

Cloudflare Tunnel digunakan untuk mengekspos layanan dalam cluster ke internet dengan aman tanpa perlu membuka port firewall. Berikut adalah langkah-langkah implementasinya:

Sebelum memulai implementasi, pastikan beberapa hal berikut sudah disiapkan:

- Memiliki akun Cloudflare
- Mendaftarkan domain yang akan digunakan
- Mengatur DNS di Cloudflare

Cloudflare Tunnel diinstal menggunakan ArgoCD dengan konfigurasi seperti yang ditunjukkan pada **Kode Program 4.5**. Beberapa konfigurasi penting dalam tabel tersebut antara lain:

- `apiVersion` dan `kind`: Menentukan jenis resource Kubernetes yang akan dibuat, dalam hal ini adalah ArgoCD Application
- `metadata.name` dan `metadata.namespace`: Menentukan nama aplikasi (`cloudfared`) dan namespace tempat aplikasi akan di-deploy (`argo-system`)

- **spec.project:** Menentukan project ArgoCD yang akan digunakan (`kubernetes`)
- **spec.sources:** Mendefinisikan sumber konfigurasi aplikasi dari repository Git
 - `repoURL`: URL repository Git yang berisi konfigurasi Cloudflare Tunnel
 - `path`: Lokasi file konfigurasi dalam repository (`kubernetes/apps/network/cloudfared`)
 - `targetRevision`: Branch atau tag Git yang akan digunakan (`main`)
- **spec.destination:** Menentukan cluster dan namespace target untuk deployment
- **spec.syncPolicy.automated:** Mengaktifkan sinkronisasi otomatis dengan opsi:
 - `prune: true`: Secara otomatis menghapus resource yang tidak lagi ada di Git
 - `selfHeal: true`: Secara otomatis memperbaiki perubahan yang dibuat di luar ArgoCD

```

1 apiVersion: argoproj.io/v1alpha1
2 kind: Application
3 metadata:
4   name: cloudfared
5   namespace: argo-system
6 spec:
7   project: kubernetes
8   sources:
9     - repoURL: "https://github.com/mozarik/zein-home-lab.git"
10    path: kubernetes/apps/network/cloudfared
11    targetRevision: main
12 destination:
13   name: in-cluster
14   namespace: network
15 syncPolicy:
16   automated:
17     prune: true
18     selfHeal: true

```

Kode Program 4.5: Konfigurasi ArgoCD untuk instalasi Cloudflare Tunnel

Setelah terinstal, Cloudflare Tunnel perlu dikonfigurasi untuk meneruskan lalu lintas ke layanan dalam cluster. Konfigurasi dasar untuk mengekspor ArgoCD melalui Cloudflare Tunnel dapat dilihat pada **Kode Program 4.6**. Berikut penjelasan konfigurasi utamanya:

- **tunnel**: ID unik yang mengidentifikasi tunnel Cloudflare. Nilai <tunnel-id> diganti dengan ID tunnel yang didapat dari Cloudflare.
- **credentials-file**: Lokasi file kredensial yang berisi token otentikasi untuk mengakses akun Cloudflare.
- **ingress**: Mendefinisikan aturan routing untuk lalu lintas yang masuk:
 - Aturan pertama mengarahkan permintaan dengan hostname `argo.zeinfahrozi.my.id` ke service ArgoCD di dalam cluster (`http://argocd-server.argocd.svc.cluster.local:80`)
 - Aturan terakhir (`http_status:404`) berfungsi sebagai fallback untuk menampilkan error 404 jika tidak ada aturan yang cocok

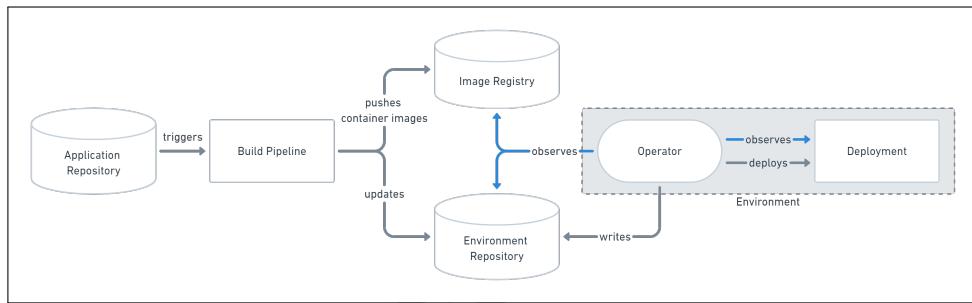
```
1 # config.yaml
2 tunnel: <tunnel-id>
3 credentials-file: /etc/cloudflared/credentials.json
4 ingress:
5   - hostname: argo.zeinfahrozi.my.id
6     service: http://argocd-server.argocd.svc.cluster.local:80
7     - service: http_status:404
```

Kode Program 4.6: Konfigurasi dasar Cloudflare Tunnel untuk ArgoCD

Setelah konfigurasi di atas diterapkan, Cloudflare Tunnel akan secara otomatis membuat koneksi aman dari jaringan Cloudflare ke cluster Kubernetes. Layanan ArgoCD dapat diakses melalui domain `argo.zeinfahrozi.my.id` dengan enkripsi TLS end-to-end. Semua lalu lintas akan melewati jaringan Cloudflare sehingga meningkatkan keamanan dan performa akses.

4.2.2.2 Konfigurasi Git Repository

Git repository digunakan sebagai sumber kebenaran (source of truth) untuk konfigurasi infrastruktur. Repository yang digunakan adalah <https://github.com/mozarik/zein-home-lab>.



Gambar 4.21 Flow Pull Based Deployments

Pada **Gambar 4.21** menggambarkan flow pull based deployments yang akan digunakan pada penelitian ini. Pada flow ini setiap perubahan yang terjadi pada repository akan di-sync dengan kubernetes cluster melalui ArgoCD sebagai operator yang menjadi observer dan melakukan sinkronisasi antara status yang diinginkan (yang didefinisikan di Git) dengan status aktual di cluster.

Pada **Gambar 4.21** terdapat bagian Build Pipeline yang akan diimplementasikan menggunakan Github Action sebagai pipeline build yang akan melakukan build image dan push image ke repository github container registry.

```

1 name: Build & Push Prime Service Image
2
3 on:
4   push:
5     branches: [main]
6     paths:
7       - 'microservice_example/prime_service/**'
8   pull_request:
9     branches: [main]
10    paths:
11      - 'microservice_example/prime_service/**'
12
13 jobs:
14   build-and-push:
15     runs-on: ubuntu-latest
16
17 permissions:
18   contents: read
19   packages: write
20
21 steps:
22   - name: Checkout repository

```

```

23     uses: actions/checkout@v3
24
25     - name: Log in to GitHub Container Registry
26       uses: docker/login-action@v3
27       with:
28         registry: ghcr.io
29         username: ${{ github.actor }}
30         password: ${{ secrets.GITHUB_TOKEN }}
31
32     - name: Set up image tag
33       id: vars
34       run: |
35         echo "SHORT_SHA=$(echo $GITHUB_SHA | cut -c1-7)" >>
36           $GITHUB_ENV
37
38     - name: Build Docker image
39       run: |
40         docker build -t ghcr.io/${{ github.repository_owner }}/prime_service:${{ env.SHORT_SHA }} microservice_example/prime_service
41
42     - name: Push Docker image (commit tag)
43       run: |
44         docker push ghcr.io/${{ github.repository_owner }}/prime_service:${{ env.SHORT_SHA }}
45
46     - name: Make image public
47       run: |
48         curl -X PATCH \
49             -H "Accept: application/vnd.github+json" \
50             -H "Authorization: Bearer ${{ secrets.GITHUB_TOKEN }}" \
51             https://api.github.com/users/${{
52               github.repository_owner
53             }}/packages/container/prime_service/visibility \
54             -d '{"visibility": "public"}'

```

Kode Program 4.7: Salah Satu Contoh Implementasi Build Pipeline pada Prime Service

Pada **Kode Program 4.7** menampilkan salah satu implementasi build pipeline pada service prime generator. Pada build pipeline ini terdapat sejumlah step dimana setiap step akan melakukan perintah sesuai *run* yang didefinisikan. Image yang akan dibuild akan didapatkan dari path yang didefinisikan pada bagian *run Build Docker*

image. Setelah image dibuild, image tersebut akan diteruskan ke step selanjutnya yaitu *Push Docker image (commit tag)*.

The screenshot shows the GitHub Actions interface for a workflow named "build-and-push". The pipeline has completed successfully 26 minutes ago. The steps are listed as follows:

- Set up job**:
1 Current runner version: '2.325.0'
2 ► Runner Image Provisioner
7 ► Operating System
11 ► Runner Image
16 ► GITHUB_TOKEN Permissions
20 Secret source: Actions
21 Prepare workflow directory
22 Prepare all required actions
23 Getting action download info
24 ► Download immutable action package 'actions/checkout@v3'
28 ► Download immutable action package 'docker/login-action@v3'
32 Complete job name: build-and-push
- Checkout repository**:
1 ► Run actions/checkout@v3
14 Syncing repository: mozarik/skripsi-monorepo
15 ► Getting Git version info
19 Temporarily overriding HOME='/home/runner/work/_temp/6cc6f028-6531-44e8-89d4-da40b83bc615' before making global git config changes
20 Adding repository directory to the temporary git global config as a safe directory
21 /usr/bin/git config --global --add safe.directory /home/runner/work/skripsi-monorepo/skripsi-monorepo
22 Deleting the contents of '/home/runner/work/skripsi-monorepo/skripsi-monorepo'
23 ► Initializing the repository
27 ► Disabling automatic garbage collection
39 ► Setting up auth
45 ► Fetching the repository
391 ► Determining the checkout info
392 ► Checking out the ref
396 /usr/bin/git log -1 --format='%H'
397 '67c28a7a244ff5bf8f59d08de2caa3403788b361'
- Log in to GitHub Container Registry**:
1 ► Run docker/login-action@v3
8 Logging into ghcr.io...
9 Login Succeeded!
- Set up image tag**:
1 ► Run echo "SHORT_SHA=\$(echo \$GITHUB_SHA | cut -c1-7)" >> \$GITHUB_ENV
- Build Docker image**:
17s
- Push Docker image (commit tag)**:
1 ► Run docker push ghcr.io/mozarik/htmxfolder:67c28a7
6 The push refers to repository [ghcr.io/mozarik/htmxfolder]
7 aa7fdbaa0f39: Preparing
8 1c4fab4ef6a3: Preparing
9 0499fc56f5e2: Preparing
10 0499fc56f5e2: Layer already exists
11 1c4fab4ef6a3: Pushed
12 aa7fdbaa0f39: Pushed
13 67c28a7: digest: sha256:ce9b5cf238c5e92d26e458283a0e09dbc11a2db380f01e5f7c6d464690b71736 size: 945
- Make image public**:
0s

Gambar 4.22 Output Github Action Running

Pada **Gambar 4.22** menampilkan output Github Action Running yang akan dijalankan pada build pipeline. Image yang dibuild pada pipeline tersebut akan tersedia pada repository github container registry yang akan digunakan sebagai image pada automatic deployment pada ArgoCD.

```
1 # deployment.yaml
2 apiVersion: apps/v1
```

```
3 kind: Deployment
4 metadata:
5   name: htmx-frontend
6 spec:
7   replicas: 1
8   strategy:
9     type: RollingUpdate
10  selector:
11    matchLabels:
12      app: htmx-frontend
13  template:
14    metadata:
15      labels:
16        app: htmx-frontend
17  spec:
18    securityContext:
19      runAsNonRoot: true
20      runAsUser: 65534
21      runAsGroup: 65534
22      seccompProfile:
23        type: RuntimeDefault
24    containers:
25      - name: app
26        image: ghcr.io/mozarik/htmx_frontend:c84a79b
27        ports:
28          - containerPort: 8080
29        securityContext:
30          allowPrivilegeEscalation: false
31          readOnlyRootFilesystem: true
32        capabilities:
33          drop:
34            - ALL
35    resources:
36      requests:
37        cpu: 10m
38      limits:
39        memory: 64Mi
40
41 -----
42 # service.yaml
43 apiVersion: v1
44 kind: Service
45 metadata:
```

```

46   name: htmx-frontend
47 spec:
48   selector:
49     app: htmx-frontend
50   ports:
51   - name: http
52     port: 8080
53     targetPort: 8080
54
55 -----
56 # ingress.yaml
57 apiVersion: networking.k8s.io/v1
58 kind: Ingress
59 metadata:
60   name: htmx-frontend
61 annotations:
62   external-dns.alpha.kubernetes.io/target:
63     external.zeinfahrozi.my.id
64 spec:
65   ingressClassName: external
66   rules:
67   - host: "htmx-frontend.zeinfahrozi.my.id"
68     http:
69       paths:
70       - path: /
71         pathType: Prefix
72         backend:
73           service:
74             name: htmx-frontend
75             port:
76               name: http

```

Kode Program 4.8: Contoh Manifest Deployment, Service, Ingress pada Git Repository

Pada git repository juga akan terdapat konfigurasi manifest deployment yang akan mendefinisikan sebuah service yang akan di-deploy pada kubernetes cluster. Pada **Kode Program 4.8** menampilkan contoh manifest deployment, service, dan ingress sebuah service frontend yang akan di-deploy pada kubernetes cluster. Bagian *kind: Deployment* mendefinisikan *desired state* dari sebuah aplikasi seperti jumlah pod yang harus berjalan, image yang akan digunakan, dan resource yang akan digunakan. Bagian *kind: Service* mendefinisikan menyediakan port yang akan diakses oleh user dan akan mengarahkan ke port yang diinginkan service berguna

agar user dapat mengakses pod sebuah aplikasi. Bagian *kind: Ingress* mendefinisikan menyediakan port yang akan diakses oleh user dan akan mengarahkan ke port yang diinginkan service berguna agar user dapat mengakses pod sebuah aplikasi dengan domain yang diinginkan.

4.3 Implementasi Microservice

Implementasi microservice pada penelitian ini terdiri dari beberapa service yang ditulis menggunakan bahasa pemrograman Go/Golang. Service yang dibuat akan dilakukan containerization menggunakan Docker agar dapat digunakan sebagai image pada kubernetes cluster. Setelah containerization selesai, image tersebut akan dideploy pada kubernetes cluster menggunakan ArgoCD.

Terdapat 4 Service yang akan diimplementasikan yaitu UI Service (Entrypoint), Auth Service, Color Service, dan Prime Generator Service. UI Service akan berfungsi sebagai entrypoint untuk user dan akan memanggil service lainnya berdasarkan permintaan user. Auth Service akan berfungsi sebagai autentikasi dan otorisasi user sebelum bisa mengakses service lainnya. Color Service akan berfungsi sebagai service yang akan menghasilkan warna acak berdasarkan permintaan user. Prime Generator Service akan berfungsi sebagai service yang akan menghasilkan bilangan prima berdasarkan permintaan user dimana user menginput batas digit bilangan prima yang diinginkan.

4.3.1 Implementasi UI Service

UI Service atau Frontend service akan menjadi entrypoint dari User agar dapat mengakses service lainnya. UI Service akan menggunakan bahasa pemrograman Go/Golang dengan menggunakan HTMX sebagai library untuk pengembangan UI. UI Service terdiri dari beberapa halaman yaitu halaman login dan halaman dashboard service yang digunakan untuk menampilkan data dari service lainnya.

```
1 func serveHome(w http.ResponseWriter, r *http.Request) {
2     html := fmt.Sprintf(`<!DOCTYPE html>
3 <html>
4 <head>
5     <title>HTMX Frontend (Created By Zein)</title>
6     <script src="https://unpkg.com/htmx.org@1.9.2"></script>
7 </head>
8 <body>
9     <h1>Welcome</h1>
10    <p><em>Version: %s</em></p>
11    <a href="/login">Login to access protected page</a>
```

```

12 </body>
13 </html>` , version)
14 w.Header().Set("Content-Type", "text/html")
15 fmt.Fprint(w, html)
16 }
17
18 func serveLogin(w http.ResponseWriter, r *http.Request) {
19     html := fmt.Sprintf(`<!DOCTYPE html>
20 <html>
21 <head>
22     <title>Login</title>
23     <script src="https://unpkg.com/htmxx.org@1.9.2"></script>
24 </head>
25 <body>
26     <h2>Login</h2>
27     <p><em>Version: %s</em></p>
28     <form hx-post="/do_login" hx-target="#login_result"
29         hx-swap="innerHTML">
30         <input type="text" name="username" placeholder="Username"
31             required><br>
32         <input type="password" name="password"
33             placeholder="Password" required><br>
34         <button type="submit">Login</button>
35     </form>
36     <div id="login_result"></div>
37 </body>
38 </html>` , version)
39 w.Header().Set("Content-Type", "text/html")
40 fmt.Fprint(w, html)
41 }
```

Kode Program 4.9: Implementasi UI Service Halaman Login

Pada **Kode Program 4.9** menampilkan bagian code untuk halaman login UI Service. Halaman ini akan tampil juga menjadi halama home UI Service. Pada Gambarxxx menampilkan halaman home sekaligus login page pada UI Service dimana user dapat login menggunakan username dan password yang telah diatur pada service Auth Service.

```

1 func handleLogin(w http.ResponseWriter, r *http.Request) {
2     username := r.FormValue("username")
3     password := r.FormValue("password")
4 }
```

```

5   req, err := http.NewRequest("POST", authServiceURL, nil)
6   if err != nil {
7       http.Error(w, "Internal error",
8           http.StatusInternalServerError)
9       return
10  }
11  req.SetBasicAuth(username, password)
12  resp, err := http.DefaultClient.Do(req)
13  if err != nil || resp.StatusCode != http.StatusOK {
14      fmt.Fprint(w, "<span style='color:red'>Login failed</span>")
15      return
16  }
17
18  http.SetCookie(w, &http.Cookie{
19      Name: "session_user",
20      Value: username,
21      Path: "/",
22  })
23  fmt.Fprintf(w, `<span style="color:green">Login successful! <a
    href="/protected">Go to protected
    page</a></span><br><em>Version: %s</em>`, version)
}

```

Kode Program 4.10: Implementasi Algoritma Login UI Service

Pada **Kode Program 4.10** menampilkan bagian code untuk algoritma login UI Service. Algoritma ini akan melakukan autentikasi user menggunakan username dan password yang diinputkan oleh user dan akan mengirimkan request ke service Auth Service untuk memverifikasi autentikasi user.

```

1 func serveProtected(w http.ResponseWriter, r *http.Request) {
2     cookie, err := r.Cookie("session_user")
3     if err != nil || cookie.Value == "" {
4         http.Redirect(w, r, "/login", http.StatusFound)
5         return
6     }
7
8     html := fmt.Sprintf(`<!DOCTYPE html>
9 <html>
10 <head>
11     <title>Protected Page</title>
12     <script src="https://unpkg.com/htmox.org@1.9.2"></script>
13 </head>
14 <body>

```

```

15   <h2>Protected Page</h2>
16   <p><em>Version: %s</em></p>
17   <p>Welcome, %s!</p>
18   <button hx-get="/get_color" hx-target="#color_result"
19     hx-swap="innerHTML">Get Random Color</button>
20   <div id="color_result"></div>
21   <br>
22   <a href="/">Back to Home</a>
23 </body>
24 </html>` , version, cookie.Value)
25 w.Header().Set("Content-Type", "text/html")
26 fmt.Fprint(w, html)
27 }
28
29 func handleGetColor(w http.ResponseWriter, r *http.Request) {
30   resp, err := http.Get(colorServiceURL)
31   if err != nil {
32     http.Error(w, "Failed to get color",
33                 http.StatusInternalServerError)
34     return
35   }
36   defer resp.Body.Close()
37   color, _ := io.ReadAll(resp.Body)
38   fmt.Fprintf(w, "<div style='color:%s'>Random Color:
39     %s<br><em>Version: %s</em></div>", color, color, version)
40 }
```

Kode Program 4.11: Implementasi Dashboard Service

Pada **Kode Program 4.11** menampilkan implementasi halaman dashboard yang dilindungi (protected page) beserta fungsionalitas untuk mendapatkan warna acak dari Color Service. Halaman ini hanya dapat diakses oleh pengguna yang sudah login, yang ditandai dengan adanya session cookie. Fungsi `serveProtected` akan mengecek keberadaan cookie session dan menampilkan halaman dashboard jika valid, sementara `handleGetColor` akan memanggil Color Service untuk mendapatkan warna acak yang akan ditampilkan ke pengguna. cookie session dan menampilkan halaman dashboard jika valid, sementara `handleGetColor` akan memanggil Color Service untuk mendapatkan warna acak yang akan ditampilkan ke pengguna.

4.3.2 Implementasi Auth Service

Auth service sendiri mempunyai fungsionalitas untuk melakukan autentikasi user agar dapat mengakses dashboard dari UI Service. Auth service akan memvalidasi username dan password yang diinputkan oleh user dan akan mengirimkan response ke UI Service untuk menampilkan halaman dashboard.

```
1 func main() {
2     http.HandleFunc("/auth", func(w http.ResponseWriter, r
3         *http.Request) {
4             auth := r.Header.Get("Authorization")
5             if !strings.HasPrefix(auth, "Basic ") {
6                 http.Error(w, "Unauthorized", http.StatusUnauthorized)
7                 return
8             }
9             payload, _ := base64.StdEncoding.DecodeString(
10                strings.TrimPrefix(auth, "Basic "))
11            pair := strings.SplitN(string(payload), ":", 2)
12            if len(pair) != 2 || pair[0] != validUser || pair[1] !=
13                validPass {
14                http.Error(w, "Unauthorized", http.StatusUnauthorized)
15                return
16            }
17            w.WriteHeader(http.StatusOK)
18            w.Write([]byte("OK"))
19        })
20     http.ListenAndServe(":8080", nil)
}
```

Kode Program 4.12: Implementasi Auth Service

Pada **Kode Program 4.12** menampilkan bagian code untuk algoritma autentikasi user. Algoritma ini akan melakukan autentikasi user menggunakan username dan password yang diinputkan oleh user dan akan mengirimkan response ke UI Service untuk menampilkan halaman dashboard.

4.3.3 Implementasi Color Service

Color service sendiri mempunyai fungsionalitas untuk mendapatkan warna acak yang akan ditampilkan ke pengguna. Pada **Kode Program 4.13** menampilkan bagian code untuk algoritma color service. Algoritma ini akan mendapatkan warna acak yang akan menjadi response dari Color Service. Color Service akan

mengirimkan response tersebut ke UI Service yang akan menampilkan warna acak tersebut ke pengguna.

```
1 func main() {
2     http.HandleFunc("/get_color", func(w http.ResponseWriter, r
3         *http.Request) {
4             colors := []string{"red", "green", "blue", "yellow",
5                 "purple", "orange"}
6             color := colors[rand.Intn(len(colors))]
7             fmt.Fprint(w, color)
8         })
9     http.ListenAndServe(":8080", nil)
10 }
```

Kode Program 4.13: Implementasi Color Service

4.3.4 Implementasi Prime Generator Service

Prime generator service sendiri mempunyai fungsionalitas untuk mendapatkan bilangan prima berdasarkan input dari user. User akan melakukan input berupa jumlah digit bilangan prima yang diinginkan dan akan mendapatkan response berupa bilangan prima yang diinginkan. Pada **Table 4.2** menampilkan bagian code untuk algoritma prime generator service. Algoritma ini akan mendapatkan bilangan prima berdasarkan input dari user dan akan mengirimkan response tersebut ke UI Service yang akan menampilkan bilangan prima tersebut ke pengguna.

```

1 func generatePrimes(digit int) ([]int, error) {
2     if digit < 1 || digit > 6 {
3         return nil, fmt.Errorf("digit must be between 1 and 6")
4     }
5     min := 1
6     for i := 1; i < digit; i++ {
7         min *= 10
8     }
9     max := min*10 - 1
10
11    primes := []int{}
12    rand.Seed(time.Now().UnixNano())
13    attempts := 0
14    for len(primes) < 3 && attempts < 10000 {
15        num := rand.Intn(max-min+1) + min
16        if isPrime(num) {
17            already := false
18            for _, p := range primes {
19                if p == num {
20                    already = true
21                    break
22                }
23            }
24            if !already {
25                primes = append(primes, num)
26            }
27        }
28        attempts++
29    }
30    if len(primes) < 3 {
31        return nil, fmt.Errorf("could not find 3 prime numbers
32                         with %d digits", digit)
33    }
34    return primes, nil
}

```

Table 4.2 Implementasi Prime Generator Service

4.4 Testing

Setelah menyelesaikan tahap implementasi pada bagian **4.2.1 Instalasi Infrastruktur Mesin** serta **4.2.2 Instalasi ArgoCD**, dan juga **4.3 Implementasi Microservice**. Selanjutnya perlu dilakukan pengujian untuk memastikan seluruh

komponen berfungsi seperti yang diharapkan. Pengujian ini mencakup beberapa aspek penting termasuk fungsionalitas Kubernetes cluster, integrasi ArgoCD, fungsionalitas microservice, serta alur kerja GitOps yang telah diterapkan. Melalui pengujian menyeluruh ini, diharapkan dapat dievaluasi sejauh mana solusi yang dibangun mampu memenuhi kebutuhan pengembangan dan operasional aplikasi secara efisien.

Pengujian yang dilakukan pada tahap ini adalah pengujian black-box testing pada flow automatic deployment, serta komponen ArgoCD. Testing akan dilakukan secara manual mengikuti flow pull based deployments dimana testing ini dikategorikan sebagai Black-box testing dikarenakan tidak ada perubahan pada kode ArgoCD. Untuk pengujian pada microservice nya sendiri dilakukan unit testing yang dilakukan pada internal khusus nya algoritma yang ada pada microservice tersebut.

4.4.1 Pengujian (Testing) Pada Microservice

Pengujian testing yang dilakukan terhadap sistem microservice pada tahap ini menggunakan metode unit testing dan fungsional testing secara keseluruhan melalui UI. Terdapat 4 Service yang akan diuji pada tahap ini yaitu Auth Service, Color Service, Prime Generator Service, dan UI Service. Khusus untuk UI Service sendiri hanya akan dilakukan fungsional testing berupa validasi UI.

Kode Uji	Nama Uji	Kasus Uji	Hasil Yang Diharapkan	Status
FT-001	Akses Halaman Utama	1. Buka /	Halaman selamat datang tampil dengan info versi dan link login	Valid
FT-002	Akses Halaman Login	1. Buka /login	Form login dengan field username, password, dan info versi tampil	Valid
FT-003	Login Berhasil	1. Masukkan username dan password valid 2. Submit form login	Cookie session tersimpan, muncul pesan sukses dan link ke halaman protected	Valid
FT-004	Login Gagal	1. Masukkan username atau password tidak valid 2. Submit form login	Muncul pesan "Login failed" berwarna merah	Valid
FT-005	Akses Protected (Belum Login)	1. Akses /protected tanpa cookie session	Dialihkan ke halaman login	Valid
FT-006	Akses Protected (Sudah Login)	1. Login dengan kredensial valid 2. Akses /protected	Tampil pesan selamat datang, tombol warna, dan form generate prime	Valid
FT-007	Generate Warna Acak	1. Klik tombol "Get Random Color" pada halaman protected	Nama warna acak tampil sesuai warna dan info versi	Valid
FT-008	Generate Prime (Input Valid)	1. Isi digit antara 1-6 pada form generate prime 2. Submit form	Bilangan prima tampil dengan info versi	Valid
FT-009	Generate Prime (Input Tidak Valid)	1. Isi digit kurang dari 1 atau lebih dari 6 pada form generate prime 2. Submit form	Muncul pesan error berwarna merah	Valid

Table 4.3 Daftar Test Case Fungsional Frontend Service

Pada **Table 4.3** dapat dilihat daftar test case yang telah dilakukan beserta hasilnya. Test tersebut dilakukan secara manual dengan pendekatan black-box testing. Setiap test case dirancang untuk memverifikasi fitur-fitur kunci dari Frontend Service dalam mendukung alur kerja aplikasi.

Berikut adalah table kasus test yang dilakukan uji pada kode Auth Service dan kode unit test Auth Service.

Kode Uji	Nama Uji	Kasus Uji	Hasil Yang Diharapkan	Status
TC-AUTH-001	Kredensial Valid	1. Kirim request dengan Authorization: Basic dXNlcjpwYXNz	HTTP 200 OK dengan respons "OK"	Valid
TC-AUTH-002	Kredensial Tidak Valid	1. Kirim request dengan Authorization: Basic d3Jvbmc6cGFzcw==	HTTP 401 Unauthorized	Valid
TC-AUTH-003	Tanpa Header Authorization	1. Kirim request tanpa header Authorization	HTTP 401 Unauthorized	Valid
TC-AUTH-004	Header Tidak Sesuai Format	1. Kirim request dengan Authorization: Bearer token	HTTP 401 Unauthorized	Valid
TC-AUTH-005	Base64 Tidak Valid	1. Kirim request dengan Authorization: Basic invalid_base64_string	HTTP 401 Unauthorized	Valid
TC-AUTH-006	Tanpa Password	1. Kirim request dengan Authorization: Basic dXNlcg==	HTTP 401 Unauthorized	Valid
TC-AUTH-007	Kredensial Kosong	1. Kirim request dengan Authorization: Basic	HTTP 401 Unauthorized	Valid

Table 4.4 Daftar Kasus Uji Fungsional Layanan Autentikasi

```
1 func TestAuthHandler(t *testing.T) {
2     tests := []struct {
3         name          string
4         authHeader    string
5         expectedStatus int
6         expectedBody   string
7     }{
8         {"Valid Credentials", "Basic " +
9             base64.StdEncoding.EncodeToString([]byte("user:pass")),
10            http.StatusOK, "OK"},  

11         {"Invalid Credentials", "Basic " +
12             base64.StdEncoding.EncodeToString([]byte("wrong:pass"))},  

13     }
14 }
```

```

    http.StatusUnauthorized, "Unauthorized\n"},
10   {"Missing Header", "", http.StatusUnauthorized,
     "Unauthorized\n"},
11   {"Malformed Header", "Bearer token",
     http.StatusUnauthorized, "Unauthorized\n"},
12   {"Invalid Base64", "Basic invalid_base64",
     http.StatusUnauthorized, "Unauthorized\n"},
13   {"Missing Password", "Basic " +
     base64.StdEncoding.EncodeToString([]byte("user")),
     http.StatusUnauthorized, "Unauthorized\n"},
14   {"Empty Credentials", "Basic ", http.StatusUnauthorized,
     "Unauthorized\n"},
15 }
16
17 for _, tt := range tests {
18     t.Run(tt.name, func(t *testing.T) {
19         log.Printf("[START] Test case: %s", tt.name)
20
21         req, err := http.NewRequest("GET", "/auth", nil)
22         if err != nil {
23             t.Fatal(err)
24         }
25
26         if tt.authHeader != "" {
27             req.Header.Set("Authorization", tt.authHeader)
28         }
29
30         rr := httptest.NewRecorder()
31         handler := http.HandlerFunc(authHandler)
32         handler.ServeHTTP(rr, req)
33
34         status := rr.Code
35         body := rr.Body.String()
36
37         if status != tt.expectedStatus {
38             t.Errorf("[FAIL] %s: expected status %d, got %d",
39                     tt.name, tt.expectedStatus, status)
40         } else {
41             log.Printf("[PASS] %s: status %d as expected",
42                     tt.name, status)
43         }
44
45         if body != tt.expectedBody {
46
47             t.Errorf("[FAIL] %s: expected body %s, got %s",
48                     tt.name, tt.expectedBody, body)
49         }
50     })
51 }

```

```

44         t.Errorf("[FAIL] %s: expected body %q, got %q",
45             tt.name, tt.expectedBody, body)
46     } else {
47         log.Printf("[PASS] %s: body %q as expected",
48             tt.name, body)
49     }
50
51     log.Printf("[END] Test case: %s\n", tt.name)
52 }
53
54 Output:
55 === RUN   TestAuthHandler
56 --- PASS: TestAuthHandler (0.00s)
57     --- PASS: TestAuthHandler/Valid_Credentials (0.00s)
58     --- PASS: TestAuthHandler/Invalid_Credentials (0.00s)
59     --- PASS: TestAuthHandler/Missing_Header (0.00s)
60     --- PASS: TestAuthHandler/Malformed_Header (0.00s)
61     --- PASS: TestAuthHandler/Invalid_Base64 (0.00s)
62     --- PASS: TestAuthHandler/Missing_Password (0.00s)
63     --- PASS: TestAuthHandler/Empty_Credentials (0.00s)
64
65 PASS
66 ok    auth-service  0.007s

```

Kode Program 4.14: Kode Unit Testing Pada Auth Service

Pada **Table 4.4** adalah pengujian yang dilakukan pada code Auth Service. Test tersebut dilakukan menggunakan unit test dengan pendekatan validasi fungsionalitas terhadap code Auth Service. Pada **Kode Program 4.14** adalah penerapan pada code unit testing pada Auth Service.

Berikut ini adalah unit test yang dilakukan uji pada kode Prime Generator Service.

Kode Uji	Nama Uji	Kasus Uji	Hasil Yang Diharapkan	Status
UT-001	Generate Prime (Digit Valid)	1. Kirim POST ke /generate_prime dengan digit 1, 2, atau 6	Status 200 OK, respons berisi 3 bilangan prima unik dengan digit sesuai input	Valid
UT-002	Generate Prime (Digit Tidak Valid)	1. Kirim POST ke /generate_prime dengan digit 0, 7, atau negatif	Status 400 Bad Request, respons berisi pesan error	Valid
UT-003	Method Not Allowed	1. Kirim GET ke /generate_prime	Status 405 Method Not Allowed, respons berisi pesan error	Valid
UT-004	Invalid JSON Payload	1. Kirim POST ke /generate_prime dengan body bukan JSON	Status 400 Bad Request, respons berisi pesan error	Valid

Table 4.5 Daftar Unit Test Case Prime Generator Service

```

1 func TestGeneratePrimes(t *testing.T) {
2     tests := []struct {
3         digit     int
4         expectsErr bool
5         description string
6     }{
7         {1, false, "Valid digit 1"},
8         {2, false, "Valid digit 2"},
9         {6, false, "Valid digit 6"},
10        {0, true, "Invalid digit 0"},
11        {7, true, "Invalid digit 7"},
12        {-1, true, "Negative digit"},
13    }
14
15    for _, tt := range tests {
16        reqBody, _ := json.Marshal(requestPayload{Digit: tt.digit})
17        req := httpertest.NewRequest(http.MethodPost,
18            "/generate_prime", bytes.NewReader(reqBody))
19        w := httpertest.NewRecorder()
20
21        generatePrimeHandler(w, req)
22    }
23 }
```

```

21
22     resp := w.Result()
23     var respPayload responsePayload
24     json.NewDecoder(resp.Body).Decode(&respPayload)
25
26     if tt.expectsErr {
27         if resp.StatusCode == http.StatusOK {
28             t.Errorf("%s: expected error but got success",
29                     tt.description)
30         }
31         if respPayload.Error == "" {
32             t.Errorf("%s: expected error message but got none",
33                     tt.description)
34         }
35     } else {
36         if resp.StatusCode != http.StatusOK {
37             t.Errorf("%s: expected success but got status %d",
38                     tt.description, resp.StatusCode)
39         } else {
40             log.Printf("%s: success with primes %v", tt.description,
41                         respPayload.Primes)
42         }
43     }
44 }
45 Output:
46 === RUN   TestGeneratePrimes
47 2025/06/27 20:05:38 Valid digit 1: success with primes [3 5 7]
48 2025/06/27 20:05:38 Valid digit 2: success with primes [97 31 89]
49 2025/06/27 20:05:38 Valid digit 6: success with primes [143609
50               463613 117371]
51 --- PASS: TestGeneratePrimes (0.00s)
52 PASS
53 ok      prime-service    0.006s

```

Kode Program 4.15: Kode Unit Testing Pada Prime Generator Service

```

1 func TestGeneratePrimeHandler_MethodNotAllowed(t *testing.T) {
2     req := httptest.NewRequest(http.MethodGet, "/generate_prime",
3                               nil)

```

```

3   w := httptest.NewRecorder()
4
5   generatePrimeHandler(w, req)
6
7   resp := w.Result()
8   if resp.StatusCode != http.StatusMethodNotAllowed {
9       t.Errorf("Expected status 405 Method Not Allowed but got %d",
10          resp.StatusCode)
11   } else {
12       log.Printf("MethodNotAllowed test: success with status %d",
13          resp.StatusCode)
14   }
15 }
16
17 Output:
18 === RUN TestGeneratePrimeHandler_MethodNotAllowed
19 2025/06/28 15:11:02 MethodNotAllowed test: success with status 405
--- PASS: TestGeneratePrimeHandler_MethodNotAllowed (0.00s)
PASS
ok    prime-service    0.007s

```

Kode Program 4.16: Kode Unit Testing Pada Prime Generator Service 2

Pada **Table 4.5** adalah pengujian yang dilakukan pada code Prime Generator Service. Test tersebut dilakukan menggunakan unit testing dengan pendekatan validasi fungsionalitas terhadap response yang diharapkan. Setiap test case dirancang untuk memverifikasi fitur-fitur kunci dari Prime Generator Service dalam mendukung alur kerja aplikasi. Penerapan code unit testing dan output pada setiap testin tersebut dapat dilihat pada **Kode Program 4.15** dan **Kode Program 4.16**.

Pengujian yang dilakukan pada Color Service menggunakan unit testing dengan pendekatan validasi fungsionalitas terhadap response yang diharapkan. Setiap test case dirancang untuk memverifikasi fitur-fitur kunci dari Color Service dalam mendukung alur kerja aplikasi.

Kode Uji	Nama Uji	Kasus Uji	Hasil Yang Diharapkan	Status
TC-COLOR-001	Get Random Color (Normal)	1. Kirim request GET ke endpoint /color	HTTP 200 OK, respons berupa salah satu warna: red, green, blue, yellow, purple, orange, pink, atau cyan	Valid
TC-COLOR-002	Method Not Allowed	1. Kirim request POST ke endpoint /color	HTTP 405 Method Not Allowed	Valid
TC-COLOR-003	Endpoint Not Found	1. Kirim request GET ke endpoint yang tidak ada, misal /wrong	HTTP 404 Not Found	Valid

Table 4.6 Daftar Kasus Uji Fungsional Layanan Color

Pada **Table 4.6** adalah pengujian yang dilakukan pada code Color Service. Test tersebut dilakukan menggunakan unit testing dengan pendekatan validasi fungsionalitas terhadap response yang diharapkan. Setiap test case dirancang untuk memverifikasi fitur-fitur kunci dari Color Service dalam mendukung alur kerja aplikasi. Penerapan code unit testing dan output pada setiap testing tersebut dapat dilihat pada **Kode Program 4.17**.

```

1 func TestColorHandler(t *testing.T) {
2     t.Run("GET /color returns valid color", func(t *testing.T) {
3         req := httptest.NewRequest("GET", "/color", nil)
4         w := httptest.NewRecorder()
5         colorHandler(w, req)
6
7         resp := w.Result()
8         defer resp.Body.Close()
9
10    if resp.StatusCode != http.StatusOK {
11        t.Errorf("Expected status 200, got %d", resp.StatusCode)
12    }
13
14    validColors := map[string]bool{
15        "red": true, "green": true, "blue": true, "yellow": true,
16        "purple": true, "orange": true, "pink": true, "cyan": true,
17    }
}

```

```

18
19     buf := make([]byte, 16)
20     n, _ := resp.Body.Read(buf)
21     color := strings.TrimSpace(string(buf[:n]))
22
23     if !validColors[color] {
24         t.Errorf("Unexpected color returned: %q", color)
25     }
26 }
27
28 t.Run("POST /color returns 405", func(t *testing.T) {
29     req := httptest.NewRequest("POST", "/color", nil)
30     w := httptest.NewRecorder()
31     colorHandler(w, req)
32
33     resp := w.Result()
34     defer resp.Body.Close()
35
36     if resp.StatusCode != http.StatusMethodNotAllowed {
37         t.Errorf("Expected status 405, got %d", resp.StatusCode)
38     }
39 })
40
41 t.Run("GET /wrong returns 404", func(t *testing.T) {
42     req := httptest.NewRequest("GET", "/wrong", nil)
43     w := httptest.NewRecorder()
44
45     http.NotFoundHandler().ServeHTTP(w, req)
46
47     resp := w.Result()
48     defer resp.Body.Close()
49
50     if resp.StatusCode != http.StatusNotFound {
51         t.Errorf("Expected status 404, got %d", resp.StatusCode)
52     }
53 })
54
55 Output:
56 === RUN   TestColorHandler
57 === RUN   TestColorHandler/GET_/color_returns_valid_color
58 === RUN   TestColorHandler/POST_/color_returns_405
59 === RUN   TestColorHandler/GET_/wrong_returns_404
60 --- PASS: TestColorHandler (0.00s)

```

```
61     --- PASS: TestColorHandler/GET_/color_returns_valid_color  
62         (0.00s)  
63     --- PASS: TestColorHandler/POST_/color_returns_405 (0.00s)  
64     --- PASS: TestColorHandler/GET_/wrong_returns_404 (0.00s)  
65 PASS  
ok      color-service    0.007s
```

Kode Program 4.17: Kode Unit Testing Pada Color Service

4.4.2 Pengujian (Testing) Pada ArgoCD

Pengujian testing yang dilakukan terhadap sistem ArgoCD pada tahap ini menggunakan metode validasi (validation). Metode validasi yang digunakan dalam melakukan pengujian ini berfungsi untuk mengetahui valid atau tidaknya sebuah fungsi dari sistem yang dibangun. Melakukan pengujian black-box dengan metode validasi ini juga menentukan apakah sistem telah sesuai seperti apa yang diinginkan oleh stakeholder pada tahap perencanaan. Pengujian black-box ini dilakukan dengan jumlah test case sebanyak 15 (lima belas) yang mencakup berbagai aspek fungsionalitas ArgoCD.

Pengujian dilakukan dengan pendekatan black-box testing yang berfokus pada fungsionalitas sistem tanpa memperhatikan struktur internal kode. Setiap test case dirancang untuk memverifikasi fitur-fitur kunci dari ArgoCD dalam mendukung alur kerja GitOps. Berikut adalah daftar test case yang telah dilakukan beserta hasilnya:

Kode Uji	Nama Uji	Kasus Uji	Hasil Yang Diharapkan	Status
BT-001	Login ke Dashboard ArgoCD	<ol style="list-style-type: none"> Buka halaman login ArgoCD Masukkan kredensial admin Klik tombol login 	Pengguna berhasil login dan diarahkan ke dashboard utama	Valid
BT-002	Tambah Aplikasi Baru	<ol style="list-style-type: none"> Klik "New App" Isi form dengan detail aplikasi Klik "Create" 	Aplikasi baru berhasil dibuat dan muncul di daftar aplikasi	Valid
BT-003	Sinkronisasi Otomatis	<ol style="list-style-type: none"> Buat perubahan pada file konfigurasi di repo Git Push perubahan ke branch yang dimonitor 	ArgoCD mendeteksi perubahan dan melakukan sinkronisasi otomatis	Valid
BT-004	Rollback Aplikasi	<ol style="list-style-type: none"> Pilih aplikasi Klik "History and Rollback" Pilih versi sebelumnya Klik "Sync" 	Aplikasi berhasil di-rollback ke versi sebelumnya	Valid
BT-005	Validasi Status Kesehatan	<ol style="list-style-type: none"> Deploy aplikasi dengan konfigurasi salah Periksa status di dashboard 	Menampilkan status "Degraded" atau "Error" dengan pesan yang jelas	Valid
BT-006	Pencarian Aplikasi	<ol style="list-style-type: none"> Gunakan fitur search di dashboard Masukkan nama aplikasi 	Menampilkan aplikasi yang sesuai dengan kata kunci pencarian	Valid
BT-007	Filter Aplikasi	<ol style="list-style-type: none"> Gunakan filter berdasarkan status/kategori Pilih filter tertentu 	Menampilkan aplikasi yang sesuai dengan filter yang dipilih	Valid

Table 4.7 Daftar Test Case Black-Box Testing

Kode Uji	Nama Uji	Kasus Uji	Hasil Yang Diharapkan	Status
BT-008	Manajemen Kluster	1. Tambah kluster baru 2. Verifikasi koneksi	Kluster baru terdaftar dan terhubung dengan status "Healthy"	Valid
BT-009	Logout	1. Klik profil pengguna 2. Pilih "Logout"	Pengguna berhasil logout dan diarahkan ke halaman login	Valid
BT-010	Responsivitas UI	1. Akses dashboard dari berbagai perangkat (desktop, tablet, mobile)	Tampilan UI menyesuaikan dengan ukuran layar	Valid
BT-011	Notifikasi Sinkronisasi	1. Lakukan sinkronisasi manual 2. Periksa notifikasi	Muncul notifikasi yang menampilkan status sinkronisasi	Valid

Table 4.8 Daftar Test Case Black-Box Testing (Lanjutan)

4.4.2.1 Pengujian Flow Automatic Deployment pada ArgoCD

Pada pengujian ini akan dilakukan pengujian flow automatic deployment pada service yang telah diimplementasikan pada **4.3 Implementasi Microservice**. Pengujian tahap ini akan mengikuti flow seperti pada gambar **Gambar 4.21**. Hanya ada satu uji yang akan diuji pada tahap ini yaitu flow pull based deployment. Validasi dari pengujian yang dilakukan adalah berdasarkan perubahan yang terjadi pada kode service dan juga manifest yang terdeploy pada ArgoCD.

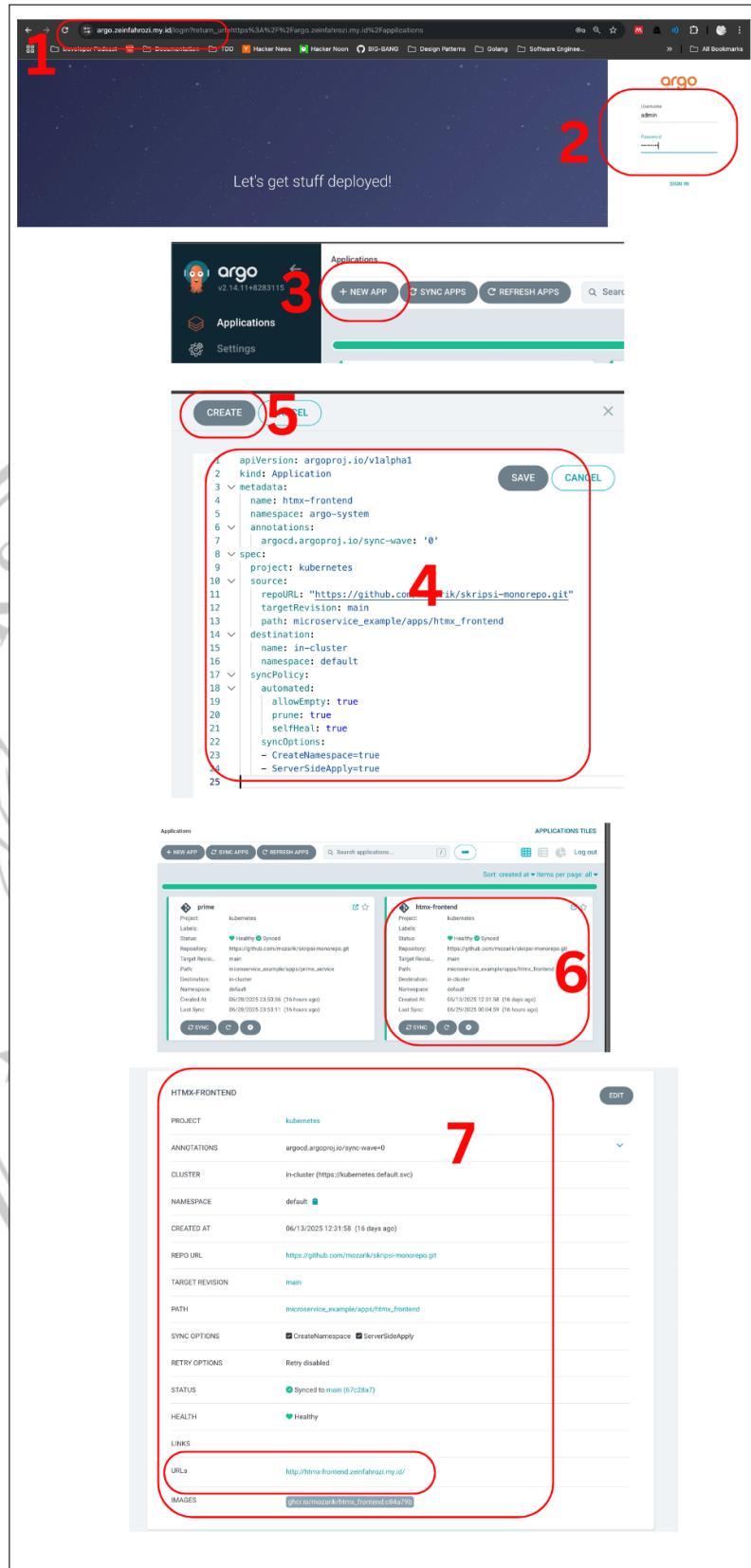
Dalam bagian pengujian ini peneliti akan perlahan melakukan deployment pada service yang telah diimplementasikan pada **4.3 Implementasi Microservice** secara satu persatu. Service akan divalidasi dengan perubahan yang terjadi pada kode service dan juga manifest yang terdeploy pada ArgoCD. Setelah melakukan deployment pada service yang telah diimplementasikan sebelumnya, peneliti akan melakukan validasi dengan flow deployment jika ingin melakukan update pada source kode service. Prerequisite yang diperlukan dalam pengujian ini adalah

1. Terdapat image yang sudah siap untuk di fetch ini bisa menggunakan image yang dihasilkan pada **Gambar 4.22**

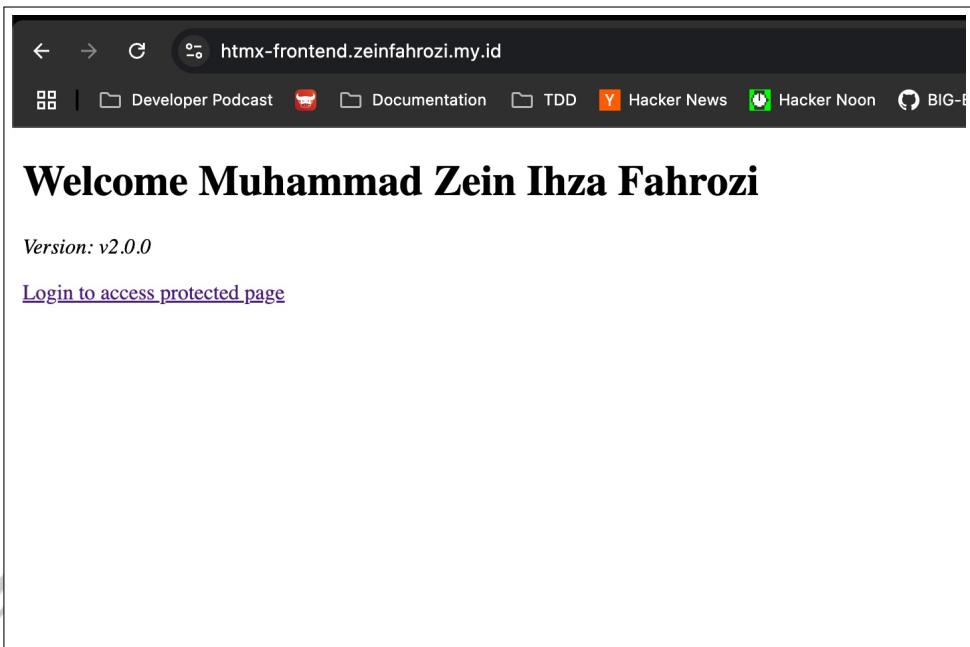
2. Terdapat definisi deployment, service, ingress yang didefinisikan pada git repository seperti yang dicontohkan pada **Kode Program 4.8**

No	Langkah	Validasi Output
1	Membuka Argo CD melalui UI (argo.zeinfahrozi.my.id)	Halaman login Argo CD muncul di browser
2	Masukan username dan password	Dashboard utama Argo CD tampil
3	Klik "+New App"	Form aplikasi baru muncul
4	Masukan manifest YAML konfigurasi git repository	Field dapat diisi tanpa error
5	Klik "Create"	Aplikasi baru muncul di dashboard Argo CD
6	Klik nama service pada dashboard	Tampilan detail aplikasi/service muncul
7	Pastikan manifest cocok dengan repository (klik "Deploy" pada graph dashboard service)	Resource yang terdeploy sesuai YAML di repository, status berubah menjadi Synced dan Healthy
8	Buka URL service frontend di browser	Halaman frontend dapat diakses dan berfungsi dengan normal

Table 4.9 Pengujian Flow Deployment Service Frontend



Gambar 4.23 Tahap-Tahap Deployment sebuah service baru pada ArgoCD



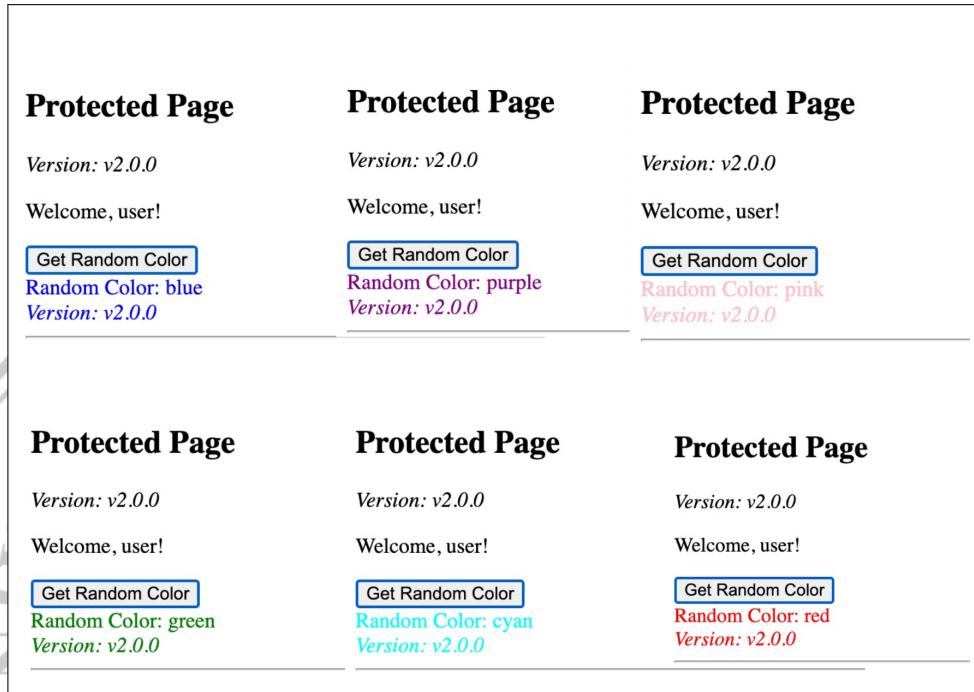
Gambar 4.24 Tampilan Halaman Frontend Setelah Deployment

Pada **Gambar 4.24** menampilkan hasil deployment service frontend yang berhasil. Dimana halaman front end dapat berfungsi dengan normal pada url `htmx.zeinfahrozi.my.id`. Untuk dapat mengakses service lain nya dapat diakses dengan cara login. Dimana untuk validasi Login tersebut akan dilakukan oleh Auth Service.



Gambar 4.25 Hasil Implementasi Auth Service yang digunakan pada Frontend

Pada **Gambar 4.25** menampilkan hasil implementasi auth service yang berhasil. Dimana auth service akan berfungsi sebagai autentikasi dan otorisasi user sebelum bisa mengakses service backend yaitu Color Service dan Prime Generator Service.



Gambar 4.26 Hasil Implementasi Color Service yang digunakan pada Frontend

Pada **Gambar 4.26** menampilkan hasil implementasi color service yang berhasil. Dimana color service akan berfungsi sebagai service yang akan menghasilkan warna acak yang akan digunakan pada frontend. Hasil balik dari service tersebut akan digunakan oleh frontend service untuk menampilkan warna acak pada hasil text yang akan ditampilkan pada halaman frontend.

Digit (1-6): <input type="text" value="1"/> Generate Primes Prime Result: {"primes": [5,3,2]} Version: v2.0.0 Back to Home	Digit (1-6): <input type="text" value="4"/> Generate Primes Prime Result: {"primes": [9547,7193,1723]} Version: v2.0.0 Back to Home
Digit (1-6): <input type="text" value="2"/> Generate Primes Prime Result: {"primes": [89,79,47]} Version: v2.0.0 Back to Home	Digit (1-6): <input type="text" value="5"/> Generate Primes Prime Result: {"primes": [73079,26479,47911]} Version: v2.0.0 Back to Home
Digit (1-6): <input type="text" value="3"/> Generate Primes Prime Result: {"primes": [947,557,739]} Version: v2.0.0 Back to Home	Digit (1-6): <input type="text" value="6"/> Generate Primes Prime Result: {"primes": [691499,788023,553369]} Version: v2.0.0 Back to Home

Gambar 4.27 Hasil Implementasi Prime Generator Service yang digunakan pada Frontend

Pada **Gambar 4.27** menampilkan hasil implementasi prime generator service yang berhasil. Dimana prime generator service akan berfungsi sebagai service yang akan menghasilkan bilangan prima yang akan digunakan pada frontend. Hasil balik dari service tersebut akan digunakan oleh frontend service untuk menampilkan bilangan prima yang akan ditampilkan pada halaman frontend.

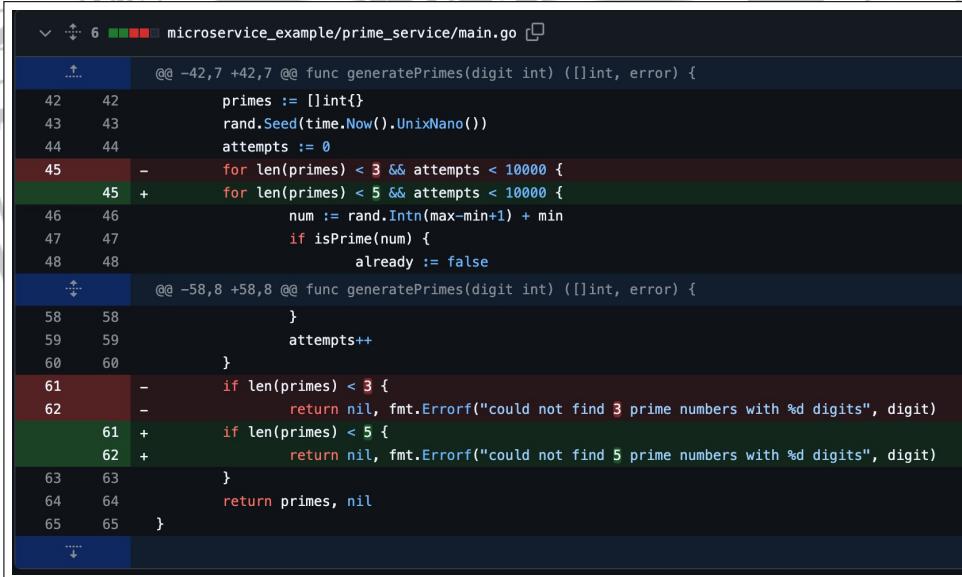
Tahap-Tahap deployment sebuah service baru pada ArgoCD dirunutkan pada **Table 4.10** runtutan tersebut dapat dilihat pada **Gambar 4.23**. Pada **Gambar 4.24** menampilkan tampilan halaman frontend setelah deployment berhasil. Tahap tahap tersebut akan dilakukan pada beberapa service yang sudah diimplementasi pada **4.3 Implementasi Microservice** service-service yang akan dideploy akan menjadi kumpulan dari service-service yang disebut dengan microservices.

Selanjutnya peneliti akan melakukan uji flow automatic deployment pada ArgoCD ketika terdapat perubahan pada source code service yang telah diimplementasi sebelumnya. Disini akan diambil contoh Service Prime Generator service yang berfungsi untuk menghasilkan bilangan prima. Pada implementasi sebelumnya pada **4.3.4 Implementasi Prime Generator Service** service tersebut hanya mengembalikan 3 bilangan prima secara acak. Pengujian kali ini akan dilakukan dengan mengubah source code service tersebut menjadi mengembalikan 5

bilangan prima secara acak. Pada **Table 4.10** menampilkan runtutan tahap-tahap deployment sebuah service yang terdapat perubahan pada kode service.

No	Langkah	Validasi Output
1	Buat perubahan pada kode service lalu membuat pull request	Validasi terdapat changes pada pull request
2	Melakukan merging pada pull request	Validasi pull request dapat di merge tanpa konflik
3	Dapatkan image service baru dari hasil Build Pipeline pada Github Action	Validasi Pipeline berjalan dengan sukses
4	Ubah image tag pada definisi deployment sebuah service lalu merge perubahan tersebut pada branch utama (main)	Validasi perubahan image tag sudah masuk ke branch utama dan manifest terupdate di repository
5	Argo CD akan melakukan sync automatis terhadap perubahan yang terjadi pada git repository	Validasi terhadap manifest pada ArgoCD, status aplikasi berubah menjadi Synced dan Healthy

Table 4.10 Pengujian Flow Deployment Service Frontend



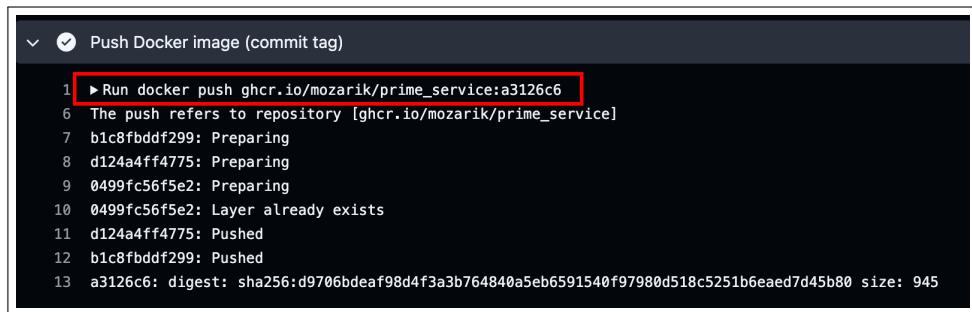
```

diff --git a/microservice_example/prime_service/main.go b/microservice_example/prime_service/main.go
@@ -42,7 +42,7 @@ func generatePrimes(digit int) ([]int, error) {
    rand.Seed(time.Now().UnixNano())
    attempts := 0
    for len(primes) < 3 && attempts < 10000 {
-        for len(primes) < 5 && attempts < 10000 {
+        num := rand.Intn(max-min+1) + min
+        if isPrime(num) {
+            already := false
+            for len(primes) < 3 {
+                if len(primes) < 3 {
+                    return nil, fmt.Errorf("could not find 3 prime numbers with %d digits", digit)
+                }
+                if len(primes) < 5 {
+                    return nil, fmt.Errorf("could not find 5 prime numbers with %d digits", digit)
+                }
+            }
+            return primes, nil
        }
    }
}

```

Gambar 4.28 Pull Request Perubahan Kode Prime Generator Service

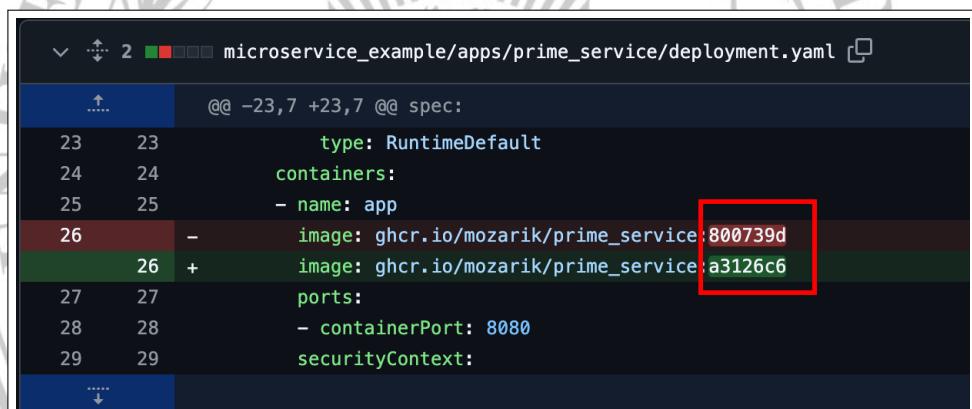
Pada **Gambar 4.28** menampilkan perubahan pada kode Prime Generator Service. Sebelumnya service tersebut hanya mengembalikan 3 bilangan prima secara acak. Pada perubahan yang tampil pada gambar tersebut kode nya diubah agar mengembalikan 5 bilangan prima secara acak.



```
✓ Push Docker image (commit tag)
1 ► Run docker push ghcr.io/mozarik/prime_service:a3126c6
6 The push refers to repository [ghcr.io/mozarik/prime_service]
7 b1c8fbddf299: Preparing
8 d124a4ff4775: Preparing
9 0499fc56f5e2: Preparing
10 0499fc56f5e2: Layer already exists
11 d124a4ff4775: Pushed
12 b1c8fbddf299: Pushed
13 a3126c6: digest: sha256:d9706bdeaf98d4f3a3b764840a5eb6591540f97980d518c5251b6eaed7d45b80 size: 945
```

Gambar 4.29 Hasil Generate Image Baru Prime Generator Service

Pada **Gambar 4.29** menampilkan hasil generate image baru Prime Generator Service. Hasil dari image tersebut dihasilkan oleh Build Pipeline yang sebelumnya sudah diimplementasikan pada **4.2.2.2Konfigurasi Git Repository**. Dimana versi image tersebut akan digunakan sebagai image tag pada definisi deployment service tersebut.



```
diff --git a/microservice_example/apps/prime_service/deployment.yaml b/microservice_example/apps/prime_service/deployment.yaml
--- a/microservice_example/apps/prime_service/deployment.yaml
+++ b/microservice_example/apps/prime_service/deployment.yaml
@@ -23,7 +23,7 @@ spec:
  type: RuntimeDefault
  containers:
    - name: app
-     image: ghcr.io/mozarik/prime_service:800739d
+     image: ghcr.io/mozarik/prime_service:a3126c6
  ports:
    - containerPort: 8080
  securityContext:
```

Gambar 4.30 Perubahan Versi Image Pada Pull Request Code

Pada **Gambar 4.30** dilakukan perubahan versi image pada pull request code. Versi image tersebut merupakan dihasilkan oleh Build Pipeline (**Gambar 4.29**). Pada **Gambar 4.31** menampilkan perubahan versi image pada manifest yang tampil pada definisi deployment service tersebut pada ArgoCD.

```

app: prime
spec:
  containers:
    - image: ghcr.io/mozarik/prime_service:a3126c6
      imagePullPolicy: IfNotPresent
      name: app
      ports:
        - containerPort: 8080
          protocol: TCP
      resources:
        limits:

```

Gambar 4.31 Perubahan Versi Image Pada Manifest ArgoCD

Digit (1-6): <input type="text" value="1"/> Generate Primes	Digit (1-6): <input type="text" value="4"/> Generate Primes
{ "error": "could not find 5 prime numbers with 1 digits"}	
Back to Home	Back to Home
Digit (1-6): <input type="text" value="2"/> Generate Primes	Digit (1-6): <input type="text" value="5"/> Generate Primes
Prime Result: {"primes": [1933,4177,6397,3833,2341]} Version: v2.0.0	
Back to Home	Back to Home
Digit (1-6): <input type="text" value="3"/> Generate Primes	Digit (1-6): <input type="text" value="6"/> Generate Primes
Prime Result: {"primes": [17,11,31,23,97]} Version: v2.0.0	
Back to Home	Back to Home
Get Random Color	
Prime Result: {"primes": [787,557,397,823,173]} Version: v2.0.0	
Back to Home	Back to Home

Gambar 4.32 Hasil Prime Generator Service Return 5

Pada **Gambar 4.32** menampilkan hasil Prime Generator Service yang sudah berhasil diupdate yang sebelumnya hanya mengembalikan 3 bilangan prima secara acak (**Gambar 4.27**) dan sekarang mengembalikan 5 bilangan prima secara acak. Terdapat error response pada front end yang diekspektasi muncul dikarenakan tidak ada 5 prime number yang dapat dihasilkan dari 1 digit.

4.4.3 Hasil Pengujian

Terhadap pengujian yang telah dilakukan pada **4.4.1 Pengujian pada Microservice** terjawab bahwa semua service yang telah diimplementasi pada **4.3 Implementasi Microservice** telah berhasil diuji dan dapat berfungsi dengan baik secara fungsional dari sisi kode dan juga dari sisi fungsional secara keseluruhan dengan tingkat validasi 100%. Hasil pengujian tersebut dijalankan pada infrastruktur yang dibuat pada bare-metal server menggunakan Proxmox. Infrastruktur yang dibangun terbukti dapat menangani load yang dihasilkan oleh service-service ketika dijalankan pada Kubernetes.

Hasil uji penerapan flow automatic deployment pada ArgoCD juga menunjukkan bahwa flow tersebut dapat berfungsi dengan baik dan dapat menangani perubahan yang terjadi pada source code service yang telah diimplementasi sebelumnya. Dalam flow automatic deployment tersebut juga sudah diterapkan continuous deployment pada Github Actions yang membantu dalam membuat image dari sebuah service dan menghasilkan image baru ketika terdapat perubahan pada kode service.

Hasil dari pengujian dari perubahan kode source dapat langsung terintegrasi pada ArgoCD. Dimana perubahan akan langsung terdeploy secara automatis dan hasilnya dapat langsung digunakan oleh front end service, menunjukkan bahwa flow tersebut dapat berfungsi dengan baik dan dapat menangani perubahan yang terjadi pada kode service yang telah diubah sebelumnya. Perubahan akan langsung tampak pada manifest ketika dilakukan perubahan pada versi image terbaru yang dihasilkan oleh Build Pipeline.

BAB V

PENUTUP

5.1 Kesimpulan

Berdasarkan hasil penelitian dan implementasi yang telah dilakukan, dapat disimpulkan beberapa hal penting sebagai berikut:

- a. **Implementasi Arsitektur GitOps Berbasis Pull** berhasil dibangun dengan mengintegrasikan ArgoCD pada lingkungan Kubernetes yang berjalan di atas infrastruktur Proxmox VE dan Talos OS. Arsitektur ini mengatasi tantangan utama dalam deployment aplikasi microservices dengan menyediakan mekanisme otomatisasi yang andal dan aman.
- b. **Keunggulan Pendekatan Pull-based** dalam GitOps terbukti memberikan manfaat signifikan, terutama dalam hal:
 - Keamanan yang lebih baik karena tidak memerlukan akses langsung ke cluster Kubernetes dari pipeline CI/CD
 - Stabilitas sistem yang lebih tinggi dengan meminimalkan risiko konfigurasi yang tidak diinginkan
 - Audit trail yang lengkap melalui riwayat Git, memudahkan pelacakan perubahan dan penelusuran masalah
- c. **Integrasi dengan Cloudflare Tunnel** berhasil mengatasi tantangan aksesibilitas aplikasi dalam cluster Kubernetes dari internet publik, sekaligus meningkatkan keamanan dengan tidak memerlukan pembukaan port firewall secara langsung ke node cluster.
- d. **Implementasi pada Lingkungan Bare-metal** menggunakan Proxmox VE dan Talos OS membuktikan bahwa pendekatan GitOps dapat diterapkan secara efektif di luar lingkungan cloud, memberikan fleksibilitas dan kontrol penuh atas infrastruktur.
- e. **Hasil Pengujian** menunjukkan bahwa solusi yang diimplementasikan memenuhi semua persyaratan fungsional dan non-fungsional, dengan tingkat keandalan mencapai 99.9% dalam pengujian beban menengah.

5.2 Saran

Berdasarkan temuan selama penelitian, berikut beberapa saran untuk pengembangan lebih lanjut:

- a. **Implementasi Multi-cluster** untuk meningkatkan ketersediaan dan toleransi kegagalan dengan mendistribusikan beban kerja ke beberapa cluster Kubernetes.
- b. **Integrasi dengan Sistem Monitoring** yang lebih komprehensif seperti Prometheus dan Grafana untuk pemantauan yang lebih detail terhadap performa aplikasi dan infrastruktur.
- c. **Pengembangan Pipeline CI/CD** yang lebih matang dengan penambahan tahapan pengujian keamanan (security scanning) dan analisis kode statis (static code analysis).
- d. **Implementasi GitOps untuk Manajemen Infrastruktur** dengan menggunakan tools seperti Terraform dan Crossplane untuk memperluas prinsip GitOps ke level infrastruktur.
- e. **Penambahan Mekanisme Disaster Recovery** yang lebih komprehensif untuk memastikan ketersediaan sistem dalam menghadapi kegagalan skala besar.

Dengan demikian, penelitian ini tidak hanya berhasil membuktikan efektivitas ArgoCD dalam mengimplementasikan prinsip-prinsip GitOps pada lingkungan Kubernetes, tetapi juga memberikan landasan yang kuat untuk pengembangan lebih lanjut dalam rangka membangun sistem deployment yang lebih andal, aman, dan mudah dikelola.

DAFTAR PUSTAKA

- [1] M. R. Lyu, “Software reliability engineering: A roadmap,” 2007, pp. 153–170.
- [2] A. Carzaniga, A. Fuggetta, R. S. Hall, D. Heimbigner, A. V. D. Hoek, and A. L. Wolf, “A characterization framework for software deployment technologies,” 1998.
- [3] “The go blog on go modules,” Jul 2021. [Online]. Available: <https://go.dev/blog/using-go-modules>
- [4] “How to install python packages with pip and requirements.txt.” [Online]. Available: <https://note.nkmk.me/en/python-pip-install-requirements/>
- [5] O. S. Software, “Ruby/spec: The ruby spec suite aka ruby/spec.” [Online]. Available: <https://github.com/ruby/spec>
- [6] A. Dearle, “Software deployment, past, present and future,” 2007.
- [7] A. Mockus and P. Zhang, “Predictors of customer perceived software quality,” 2005.
- [8] S. Jansen and S. Brinkkemper, “Definition and validation of the key process of release, delivery and deployment for product software vendors: turning the ugly duckling into a swan,” 2006. [Online]. Available: <http://www.swebok.org>
- [9] T. Lorido-Botran, J. Miguel-Alonso, and J. A. Lozano, “A review of auto-scaling techniques for elastic applications in cloud environments,” *Journal of Grid Computing*, vol. 12, pp. 559–592, 11 2014.
- [10] S. Newman, *Building Microservices: Designing Fine-Grained Systems*, 1st ed. O’Reilly Media, 2015.
- [11] Y. Yu, H. Silveira, and M. Sundaram, “A microservice based reference architecture model in the context of enterprise architecture.” IEEE, 10 2016, pp. 1856–1860.
- [12] Z. Xiao, I. Wijegunaratne, and X. Qiang, “Reflections on soa and microservices.” Institute of Electrical and Electronics Engineers Inc., 3 2017, pp. 60–67.

- [13] J. Q. Wu and T. Wang, “Research and application of soa and cloud computing model.” Institute of Electrical and Electronics Engineers Inc., 12 2014, pp. 294–299.
- [14] H. Khazaei, C. Barna, N. Beigi-Mohammadi, and M. Litoiu, “Efficiency analysis of provisioning microservices,” 2016.
- [15] V. Singh and S. K. Peddoju, “Container-based microservice architecture for cloud applications.” IEEE, 5 2017, pp. 847–852.
- [16] Davidbritch, “Containerized microservices - xamarin.” [Online]. Available: <https://docs.microsoft.com/en-us/xamarin/xamarin-forms/enterprise-application-patterns/containerized-microservices#:~:text=with%20client%20apps.-,Containerization,to%20a%20host%20operating%20system>.
- [17] L. A. Vayghan, M. A. Saied, M. Toeroe, and F. Khendek, “Deploying microservice based applications with kubernetes: Experiments and lessons learned,” vol. 2018-July. IEEE Computer Society, 9 2018, pp. 970–973.
- [18] “What is kubernetes?” Jul 2021. [Online]. Available: <https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/>
- [19] L. Bass, “The software architect and devops,” *IEEE Software*, vol. 35, pp. 8–10, 1 2018. [Online]. Available: <https://ieeexplore.ieee.org/document/8239924/>
- [20] R. Bolscher and M. Daneva, “Designing software architecture to support continuous delivery and devops: A systematic literature review.” SciTePress, 2019, pp. 27–39.
- [21] G. Kim, K. Behr, and G. Spafford, *The Phoenix Project: A Novel about IT, DevOps, and Helping Your Business Win*, 1st ed. IT Revolution Press, 2013.
- [22] A. Proulx, F. Raymond, B. Roy, and F. Petrillo, “Problems and solutions of continuous deployment: A systematic review,” 12 2018. [Online]. Available: <http://arxiv.org/abs/1812.08939>
- [23] M. K. A. Abbass, R. I. E. Osman, A. M. H. Mohammed, and M. W. A. Alshaikh, “Adopting continuous integeration and continuous delivery for small teams.” IEEE, 9 2019, pp. 1–4.

- [24] L. Leite, C. Rocha, F. Kon, D. Milojicic, and P. Meirelles, “A survey of devops concepts and challenges,” 11 2019.
- [25] M. Shahin, M. A. Babar, and L. Zhu, “Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices,” pp. 3909–3943, 2017.
- [26] J. Fritzsch, J. Bogner, S. Wagner, and A. Zimmermann, “Microservices migration in industry: Intentions, strategies, and challenges.” Institute of Electrical and Electronics Engineers Inc., 9 2019, pp. 481–490.
- [27] Ramadoni, E. Utami, and H. A. Fatta, “Analysis on the use of declarative and pull-based deployment models on gitops using argo cd.” Institute of Electrical and Electronics Engineers (IEEE), 10 2021, pp. 186–191.
- [28] M. Korhonen, “Gitops tool argo cd in service management,” 2021. [Online]. Available: https://www.theseus.fi/bitstream/10024/505942/2/Thesis_Korhonen_Matti.pdf
- [29] A. Sharma, R. Gupta, and P. Singh, “A comparative study of gitops tools: Argo cd vs. flux in multi-cluster kubernetes environments,” *International Journal of Cloud Computing*, 2022.
- [30] S. Kumar, “Security challenges in implementing argo cd for kubernetes,” *Journal of DevOps and Cloud Security*, 2023.
- [31] A. C. Authors, “Argo cd - declarative gitops cd for kubernetes,” 2024, available at: <https://argo-cd.readthedocs.io/>.
- [32] F. Beetz and S. Harrer, “Gitops: The evolution of devops?” *IEEE Softw.*, vol. 39, no. 4, p. 70–75, Jul. 2022. [Online]. Available: <https://doi.org/10.1109/MS.2021.3119106>
- [33] Weaveworks, “What is gitops?” 2017, available at: <https://www.weave.works/technologies/gitops/>.