

***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
2021

LEMBAR PERSETUJUAN

“To Be Added Bismillah”

LEMBAR PENGESAHAN

“To Be Added Bismillah”

LEMBAR PERNYATAAN

“To Be Added Bismillah”

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

“To Be Added Bismillah”

KATA PENGANTAR

Puji syukur yang dapat penulis panjatkan atas kehadiran Allah Subhanahu Wa Ta'ala, karena berkat Rahmat dan Karunia-Nya, penulis dapat menyelesaikan laporan skripsi yang berjudul "**Automatic Deployment Aplikasi berbasis Microservice Pada Platform Kubernetes Dengan Metode Pull-Up**". Tidak lupa juga untuk menghaturkan Shalawat serta salam yang penulis panjatkan pada junjungan besar Nabi Muhammad Salallahu Alaihi Wa salam beserta keluarga,sahabat dan pengikutnya.

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
BAB I PENDAHULUAN	1
1.1 Latar Belakang	1
1.2 Rumusan Masalah	4
1.3 Tujuan Penelitian	4
1.4 Batasan Masalah	5
BAB II TINJAUAN PUSTAKA	6
2.1 Penelitian Terdahulu	6
BAB III METODOLOGI PENELITIAN	8
3.1 Rancangan Penelitian	8
3.2 Studi Literatur	8
3.3 Perancangan (Design)	9
3.4 Implementasi Code Automasi (Coding)	10
3.5 Pengujian dan Analisis (Testing dan Analysis)	10
3.6 Kesimpulan	10
BAB IV HASIL DAN PEMBAHASAN	11

4.1	Perancangan (Design)	11
4.1.1	Sistem Arsitektur Infrastruktur	11
4.1.2	Sistem Arsitektur Kubernetes Cluster	11
4.2	Pengkodean (Coding) / Implementasi	12
4.2.1	Implementasi Proxmox VE	12
4.2.2	Implementasi Talos OS	15
4.2.3	Pengkodean Instalasi Kubernetes Script	20
4.3	Implementasi ArgoCD pada cluster kubernetes	23
4.3.1	Instalasi ArgoCD	24
4.3.2	Konfigurasi High Availability	25
4.3.3	Integrasi dengan Monitoring	25
4.3.4	Manajemen Aplikasi	25
4.3.5	Keamanan	25
4.4	Implementasi Cloudflare Tunnel	26
4.4.1	Konfigurasi Git Repository	27
4.4.2	Integrasi dengan GitHub Actions	27
4.5	Testing	28
4.5.1	Pengujian (Testing)	28
4.5.2	Metodologi Pengujian	28
4.5.3	Hasil Pengujian	28
4.5.4	Analisis Hasil Pengujian	30
BAB V	PENUTUP	32
5.1	Kesimpulan	32
5.2	Saran	33
DAFTAR PUSTAKA	34	

DAFTAR TABEL

4.1 Konfigurasi script instalasi Kubernetes cluster pada Talos OS	21
4.2 Konfigurasi script instalasi Kubernetes cluster pada Talos OS	22
4.3 Daftar Test Case Black-Box Testing	29
4.4 Daftar Test Case Black-Box Testing (Lanjutan)	30

DAFTAR GAMBAR

3.1	Rancangan Penelitian	8
4.1	Arsitektur Infrastruktur High Level	11
4.2	Arsitektur Kubernetes High Level	12
4.3	Proses Pembuatan Bootable Proxmox Menggunakan Rufus	13
4.4	Tampilan Awal Instalasi Proxmox VE	14
4.5	Tampilan Terminal Setelah Instalasi Proxmox Selesai	14
4.6	Tampilan Web Interface Proxmox VE	15
4.7	Proses Download ISO Talos OS	15
4.8	Instalasi Talos OS 1	16
4.9	Instalasi Talos OS 2	16
4.10	Instalasi Talos OS 3	17
4.11	Instalasi Talos OS 4	17
4.12	Instalasi Talos OS 5	18
4.13	Instalasi Talos OS 6	18
4.14	Instalasi Talos OS 7	19
4.15	Instalasi Talos OS 8	19
4.16	Instalasi Talos OS 9	20
4.17	Tampilan Talos OS setelah instalasi kubernetes cluster	21
4.18	Tampilan echo.zeinfahrozi.my.id pada browser	23

BAB I

PENDAHULUAN

1.1 Latar Belakang

Sebuah perangkat lunak sebelum digunakan oleh pengguna secara luas perlu dilakukan proses deployment pada perangkat lunak tersebut. Deployment sebuah perangkat lunak dapat didefinisikan sebagai akuisisi dan eksekusi sebuah perangkat lunak. Proses ini biasanya dilakukan oleh seorang software deployer atau dalam bahasa yang sering digunakan belakangan ini seorang SRE (site reliability engineer) [1]. Maka dari itu dapat dikatakan deployment adalah aktivitas post-production sebuah perangkat lunak untuk digunakan oleh konsumen. Proses deployment sebuah perangkat lunak terdiri dari beberapa proses yang saling berhubungan seperti proses release sebuah perangkat lunak, instalasi perangkat lunak kedalam environment execution, dan aktivasi sebuah software [2].

Untuk mendeploy sebuah sistem perangkat lunak juga ada beberapa yang harus diperhatikan antara lain sub-komponen yang dibutuhkan atau package external, resource (hardware). Untuk melakukan deploymen sub-komponen ini dibutuhkannya sebuah konfigurasi yang mendeskripsikan versi sub-komponen yang digunakan oleh perangkat lunak. Dengan bahasa modern sekarang konfigurasi sub-komponen yang digunakan oleh main aplikasi biasanya sudah otomatis terbuat contohnya antara lain adalah; go modules [3] dalam bahasa Golang, requirement file [4] dalam bahasa Python, atau file RubySpec [5] pada bahasa Ruby.

Terdapat beberapa karakteristik yang mendasar pada deployment sebuah perangkat lunak yang ditulis oleh Alan Dearle yaitu [6]; **Release** berupa jembatan antara proses deployment dengan proses development. yang meliputi semua operasi yang diperlukan untuk mempersiapkan sebuah sistem untuk di-transfer ke konsumen. Aktivitas release ini juga menentukan resource yang dibutuhkan oleh sebuah sistem perangkat lunak untuk dapat beroperasi pada environment nya. Setelah itu dilakukan packaging pada sistem perangkat lunak. Package tersebut harus mengandung komponen yang dibutuhkan oleh sistem, deskripsi sistem, dependencies pada komponen eksternal, prosedur deployment, dan semua informasi yang relevan dari sistem tersebut pada environment yang akan dijalankan. **Installation** diperlukan untuk persiapan melakukan activation. **Activation** adalah proses ekspsi sebuah perangkat lunak pada waktu tertentu, biasa menggunakan grafik antar muka ataupun

proses daemon. Updating adalah proses untuk mengganti bagian dari perangkat lunak yang terinstal dengan versi yang lebih baru. Selanjutnya yaitu ***Undeployment*** yaitu proses menghapus software yang terinstall dalam sebuah mesin ini dapat disebut dengan *deinstallation*.

Menurut Mockus dkk [7] kualitas deployment sebuah perangkat lunak masuk kedalam faktor utama dalam persepsi konsumen dalam hal kualitas sebuah perangkat lunak. Jansen dan Brinkkemper [8] juga mengatakan bahwa kelancaran sebuah deployment perangkat lunak adalah hal esensial untuk meningkatkan produk perangkat lunak sebuah perusahaan/organisasi. Tetapi terdapat beberapa tantangan yang dihadapi pada saat melakukan aktivitas deploymet. Menurut Antonio Carzaniga [2] terdapat beberapa tantangan yang sering dihadapi pada saat melakukan deployment yaitu; **Mengganti** atau melakukan update sebuah sistem terhadap komponen yang sudah berjalan, **Dependencies** komponen antar satu sama lain, dan **Koordinasi** ketika melakukan update apakah akan mengganggu proses bisnis yang sedang berjalan atau tidak, dan juga **Mengatur** platform yang heterogen misalnya terhadap spesifik sistem operasi yang digunakan.

Seiring berjalannya waktu, sebuah aplikasi cenderung menjadi semakin kompleks [9, 10]. Dengan team pengembang dan aplikasi yang selalu bertumbuh dari sisi kompleksitas dan *maintainability* biasanya menyebabkan model pengembangan aplikasi menjadi susah untuk dikembangkan atau dapat dikatakan terdapat *bottleneck* sehingga aplikasi menjadi tidak efisien [11]. Dengan berkembangnya kompleksitas sebuah aplikasi diperlukan sebuah arsitektur yang bisa menyelesaikan hal itu, salah satu caranya yaitu menggunakan arsitektur *microservice* [9]. Dengan *microservice* aplikasi dibagi menjadi bagian-bagian kecil (unit) yang terpisah satu dengan lainnya [9]. Masing-masing bagian aplikasi ini dapat dijalankan dan dikembangkan secara independent (dari sisi developer) [12].

Saat ini aplikasi berbasis *microservice* menjadi pilihan sebagai arsitektur utama ketika membangun aplikasi yang scalable [13]. Aplikasi berbasis *microservice* ini dulunya dikembangkan dengan menjalankan beberapa VM (virtual machine) yang saling berkomunikasi satu sama lain melalui REST/HTTP (Hypertext Transfer Protocol) ataupun RPC (*Remote Procedure Call*) [14]. Karena pada saat itu deployment *microservice* itu sendiri masih berbasis VM yang membutuhkan operasi manual dan juga biaya yang mahal, aplikasi berbasis *microservice* pun menjadi susah dan kompleks dari sisi biaya dan waktu yang dibutuhkan untuk dilakukan pengembangan dan juga *scaling* [14].

Teknologi VM sendiri sudah perlahan ditinggalkan ketika merancang *microservice*. Dengan adanya teknologi *containerization* aplikasi dapat dibungkus agar dapat dijalankan dengan mudah dan efisien [14]. Menjalankan aplikasi dengan abstraksi yang diberikan oleh sebuah *container* juga membawa fleksibilitas dalam pengelolaannya. Salah satu manfaatnya adalah scaling aplikasi yang jauh lebih mudah, yaitu hanya dengan melakukan penyesuaian jumlah *container* yang dijalankan [15].

Containerization [16] ini sangat berdampak pada aspek infrastruktur dan *runtime* sebuah aplikasi. Tapi dengan adanya *container* diperlukan juga sebuah sistem yang melakukan orkestrasi secara otomatis pada *container* tersebut. Dengan adanya kubernetes kita dapat memanfaatkan fitur fitur yang diberikan oleh kubernetes antara lain fitur automatic scaling [17, 18]. Saat ini untuk melakukan deployment sebuah *container* (*service*) dilakukan secara manual dengan merubah file deployment. Dengan demikian, diperlukan cara otomatis untuk melakukan deployment *service* yang baru/diubah.

Dengan banyaknya persaingan dalam dunia perangkat lunak yang terjadi dalam waktu ini sebuah perusahaan atau developer 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 developer 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].

Walaupun DevOps sendiri memiliki beberapa keunggulan, tetapi implementasi CI,CDE, dan CD bukanlah hal yang mudah. Pemilihan tools dan adaptasinya, adaptasi karyawan, dan miskonfigurasi yang biasa terjadi pada saat migrasi menggunakan metode DevOps.

Terdapat beberapa masalah yang sering terjadi pada saat menggunakan metode DevOps yaitu; Arsitektur [20], tools [22], metode baru [23, 24], Keamanan dari flow CI/CD [25], dan mekanisme rollback [26]. Pada penelitian yang dilakukan oleh Ramadoni dkk. [27] dilakukan analisis menggunakan metode GitOps yang dapat menyelesaikan permasalahan mekanisme rollback, dan keamanan flow CI/CD

Pada penelitian yang dilakukan oleh Ramadoni dkk. [27] digunakan metode pull-based untuk menyelesaikan permasalahan diatas. Pada penelitian tersebut tidak dijelaskan perbedaan kenapa harus menggunakan metode pull-based atau push-based dan juga tidak dijelaskan perbandingan tools yang digunakan sebagai operator GitOps yaitu ArgoCD pada sebuah cluster Kubernetes. Maka tujuan dari penelitian ini adalah untuk melakukan analisis pada tools yang digunakan sebagai operator GitOps dan melakukan perbandingan metode pull-based dan push-based pada metode DevOps.

1.2 Rumusan Masalah

Berdasarkan latar belakang masalah diatas, adapun rumusan masalah pada penelitian ini antara lain:

- a. Bagaimana cara sebuah service yang source nya diubah dilakukan deployment secara otomatis pada sistem kubernetes dengan metode GitOps ?
- b. Bagaimana cara provisioning infrastruktur yang dibutuhkan oleh service secara otomatis pada sistem microservice yang berjalan pada kubernetes?
- c. Bagaimana perbedaan mendasar dari tools yang digunakan sebagai operator GitOps pada sistem kubernetes?
- d. Bagaimana hasil analisis tipe deployment Pull-based dan Push-based pada continous deployment?

1.3 Tujuan Penelitian

Berdasarkan rumusan masalah yang telah diformulasikan, maka terdapat beberapa tujuan pada studi ini:

- a. Melakukan implementasi continous deployment pada sebuah sistem microservice yang berjalan pada kubernetes.

- b. Melakukan analisis terhadap metode deployment secara automatis menggunakan Pull-based dan Push-based pada continuous deployment.
- c. Melanjutkan saran dari penelitian sebelumnya [27] dengan merancang workflow continuous integration yang diintegrasikan dengan ArgoCD.

1.4 Batasan Masalah

Dalam penelitian ini, peneliti akan membatasi masalah yang akan diteliti antara lain:

- a. Menggunakan kubernetes sebagai alat orkestrasi *container*.
- b. Implementasi CI/CD dilakukan menggunakan GitLabCI.
- c. Container runtime yang digunakan adalah Docker.
- d. Aplikasi microservice yang digunakan berupa aplikasi berbasis web.

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.

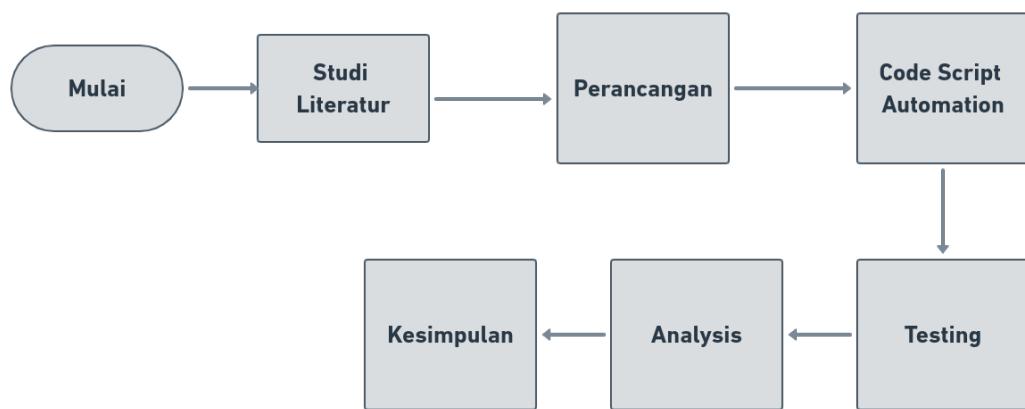
Berdasarkan tinjauan terhadap penelitian-penelitian terdahulu, dapat disimpulkan bahwa penggunaan Argo CD dalam workflow GitOps membawa dampak positif yang signifikan terhadap efisiensi, keamanan, dan auditability proses deployment aplikasi berbasis microservices di Kubernetes. Namun demikian, masih 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 mengembangkan fitur continuous integration dengan ArgoCD



Gambar 3.1 Rancangan Penelitian

3.2 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 [31]. Selain itu, peneliti juga mengkaji dokumentasi resmi Argo CD yang menjelaskan fitur, arsitektur, serta best practice dalam penerapannya pada workflow CI/CD [32]. 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.

3.3 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 [31]
3. Perancangan mekanisme continuous integration yang terintegrasi dengan Argo CD untuk membentuk pipeline CI/CD yang lengkap
4. Perancangan strategi deployment dan rollback yang efektif untuk aplikasi berbasis microservice

3.4 Implementasi Code Automasi (Coding)

Pada tahap implementasi ini dilakukan penulisan kode terhadap beberapa microservice yang akan menjadi usecase yang akan digunakan menggunakan bahasa Go. Pada tahap ini juga akan dilakukan automasi script untuk membuat infrastruktur dimana ArgoCD akan berjalan. Tahap implementasi akan menghasilkan sebuah sistem automasi, microservices, beserta infrastructur tempat berjalannya ArgoCD yaitu infrastructure kubernetes.

3.5 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 akan dilakukan menggunakan black-box testing dengan metode validasi User Acceptance Testing dengan kriteria usability testing.

3.6 Kesimpulan

Kesimpulan adalah tahapan terakhir setelah dilakukan implementasi dan pengujian. Pada tahap ini peneliti akan membuat kesimpulan untuk bertujuan mengetahui letak kelebihan dan kekurangan sistem yang telah dikembangkan.

BAB IV

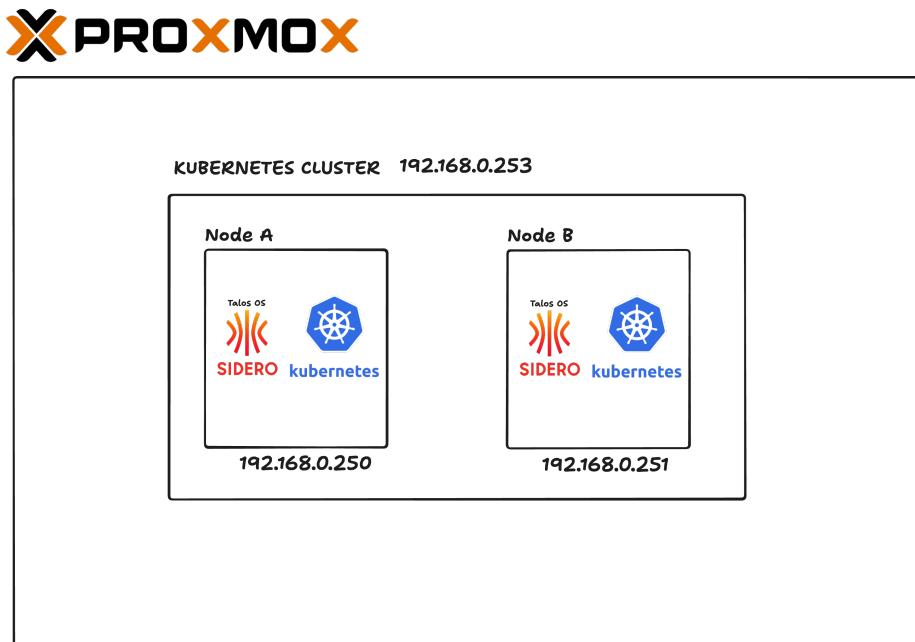
HASIL DAN PEMBAHASAN

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 Sistem Arsitektur Infrastruktur

Peneliti menggunakan Proxmox VE (Virtual Environment) yang didalamnya terdapat Virtual Machine berupa Talos OS Linux yang siap digunakan untuk menjalankan Kubernetes cluster

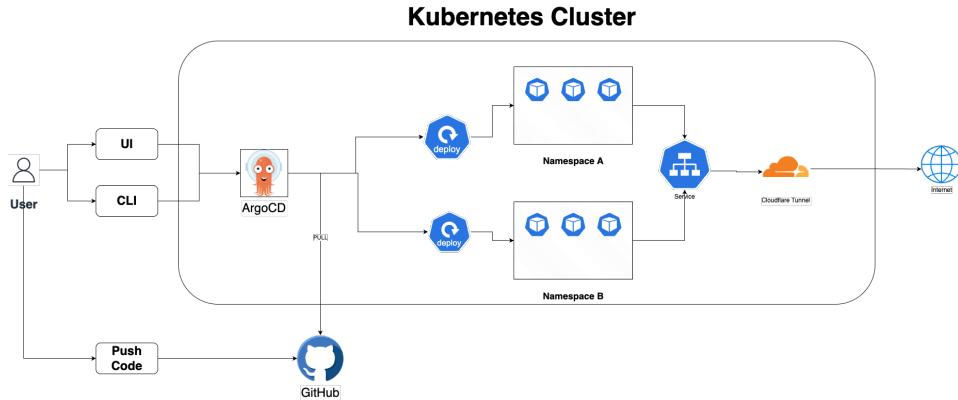


Gambar 4.1 Arsitektur Infrastruktur High Level

4.1.2 Sistem Arsitektur Kubernetes Cluster

Pada bagian sebelumnya peneliti sudah memaparkan arsitektur infrastruktur dimana Kubernetes cluster akan berjalan. Pada tahap ini peneliti akan memaparkan

arsitektur pada Kubernetes cluster nya itu sendiri dimana ArgoCD akan berjalan.



Gambar 4.2 Arsitektur Kubernetes High Level

Didalam sistem Kubernetes cluster yang dibuat terdapat komponen ArgoCD dan Cloudflare Tunnel (optional) yang bertujuan agar server microservice bisa diakses pada world wide web (internet).

4.2 Pengkodean (Coding) / Implementasi

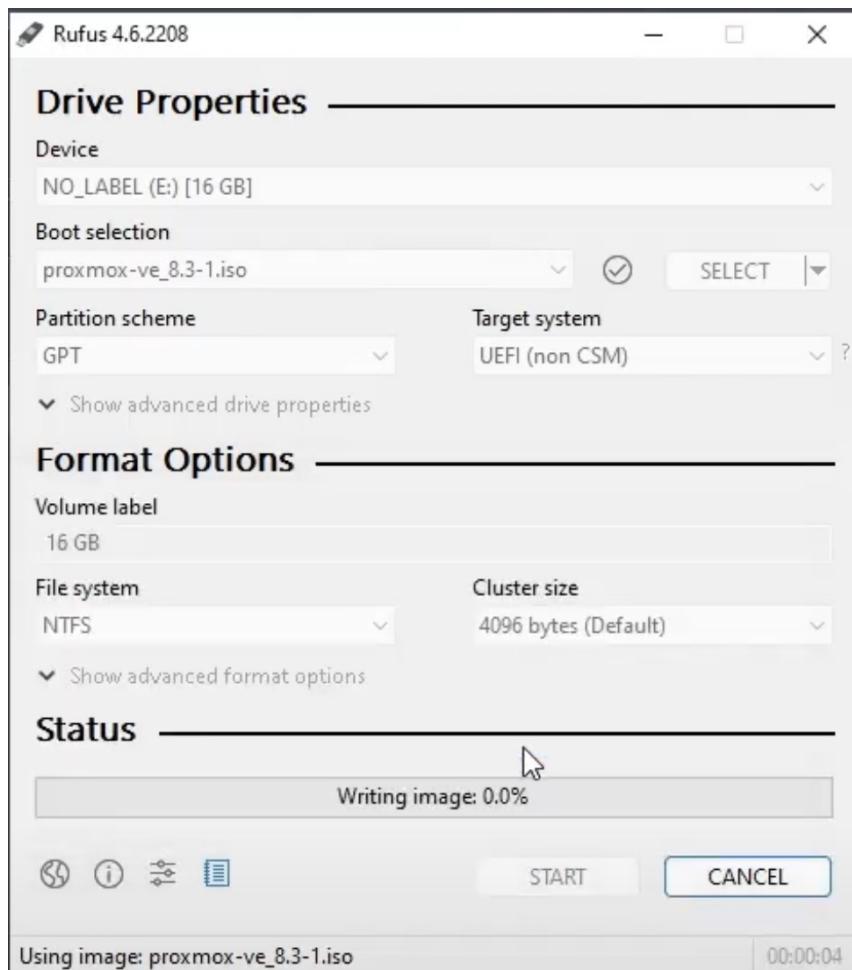
Tahap ini peneliti akan menjabarkan secara rinci pengkodean/implementasi rancangan sistem yang sudah dirancang. Peneliti akan melakukan implementasi rancangan sistem menggunakan beberapa komponen yaitu

1. Proxmox VE (versi 8.4)
2. Talos OS (versi 1.9.5)
3. Kubernetes
4. ArgoCD
5. Cloudflare Tunnel
6. Git repository (GitHub)

4.2.1 Implementasi Proxmox VE

Untuk implementasi Proxmox VE sendiri peneliti melakukan instalasi proxmox pada 2 mesin dengan arsitektur CPU X86. Pertama kita 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 kita memerlukan sebuah Flashdisk atau media untuk bootable ISO tersebut disini peneliti menggunakan tools yang bernama Rufus



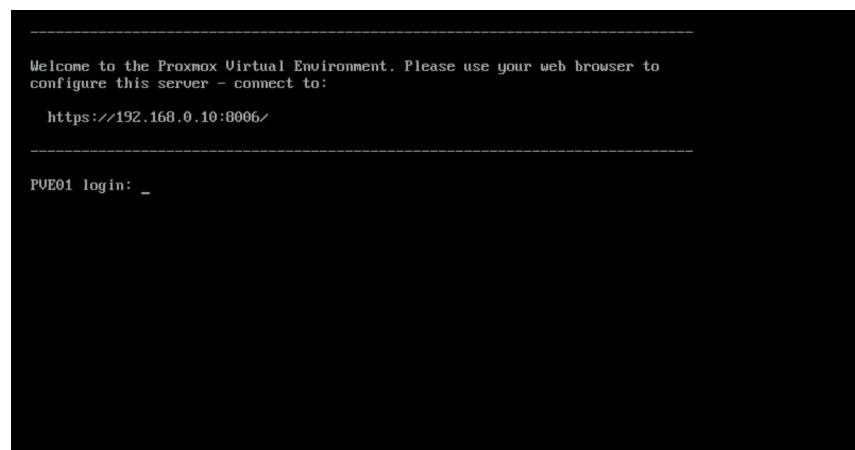
Gambar 4.3 Proses Pembuatan Bootable Proxmox Menggunakan Rufus

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. Akan terdapat layar instalasi Rufus seperti ini yang akan muncul pada mesin yang kita gunakan.



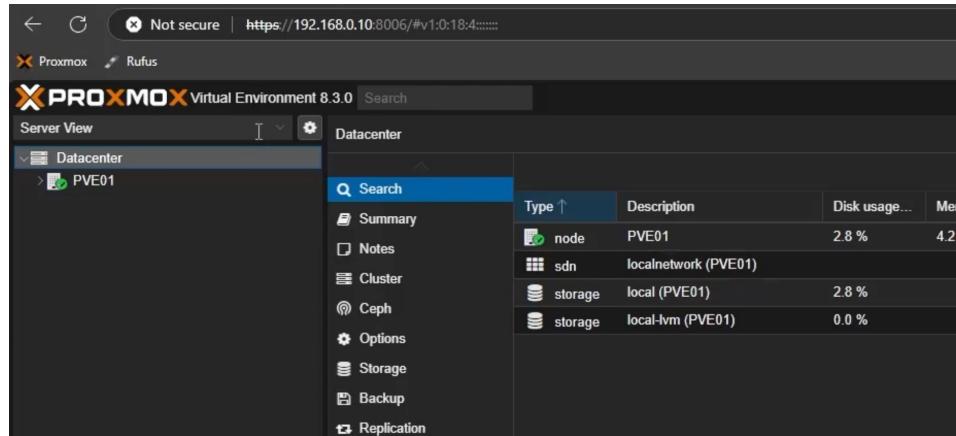
Gambar 4.4 Tampilan Awal Instalasi Proxmox VE

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



Gambar 4.5 Tampilan Terminal Setelah Instalasi Proxmox Selesai

Akses UI pada komputer yang ada pada network yang sama dengan Proxmox melalui web user interface. Peneliti akan mengulang implementasi ini untuk mesin kedua.

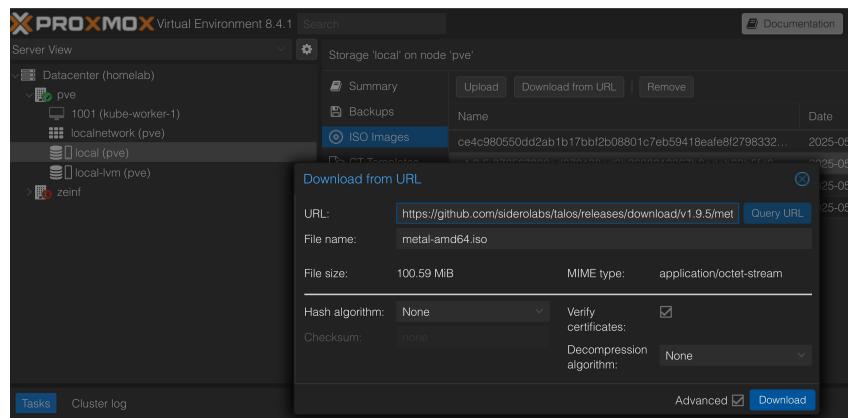


Gambar 4.6 Tampilan Web Interface Proxmox VE

4.2.2 Implementasi Talos OS

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 4.7 Proses Download ISO Talos OS

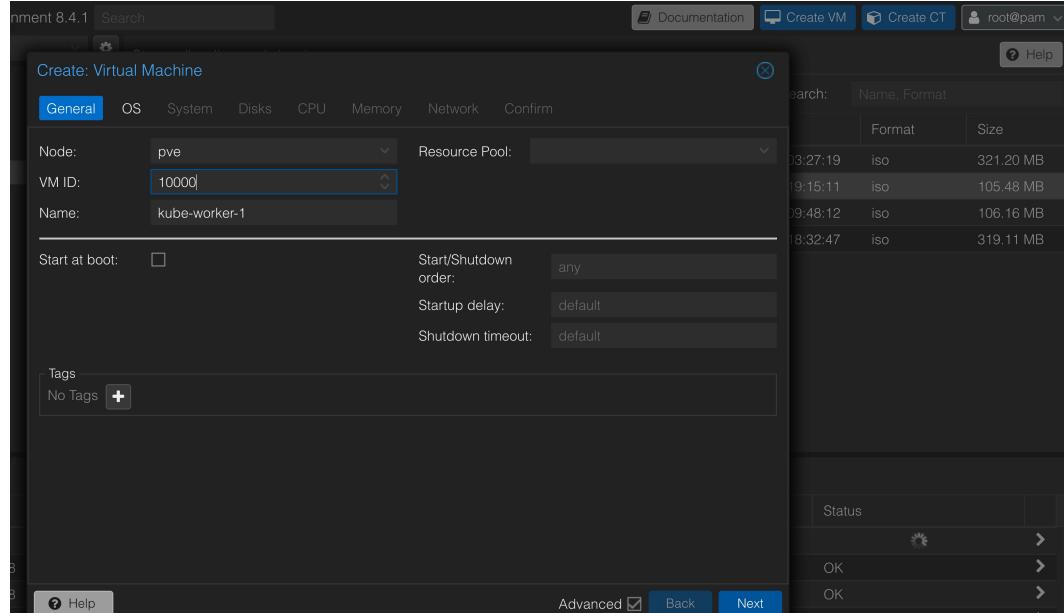
Url tersebut lalu didownload melalui proxmox agar tersimpan didalam proxmox. Lalu tahap selanjutnya adalah melakukan instalasi VM Talos OS pada proxmox.

4.2.2.1 Instalasi Talos OS

Tahap ini adalah bagian instalasi VM Talos OS. Peneliti melakukan instalasi pada proxmox VE melalui interface web. Tahap ini akan dilakukan pada 2 mesin

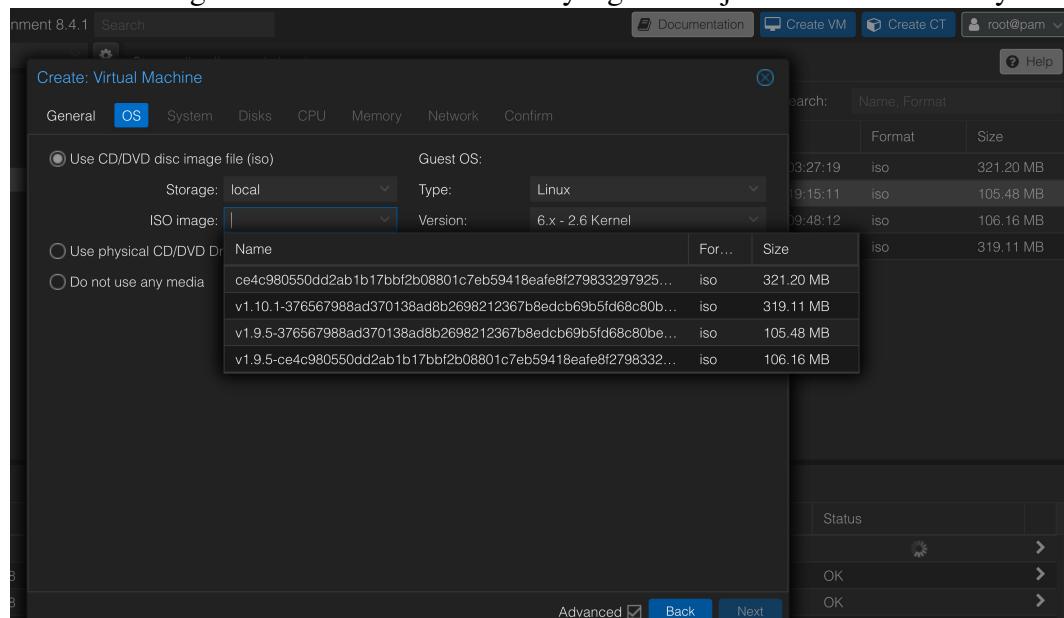
berbeda pada proxmox.

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



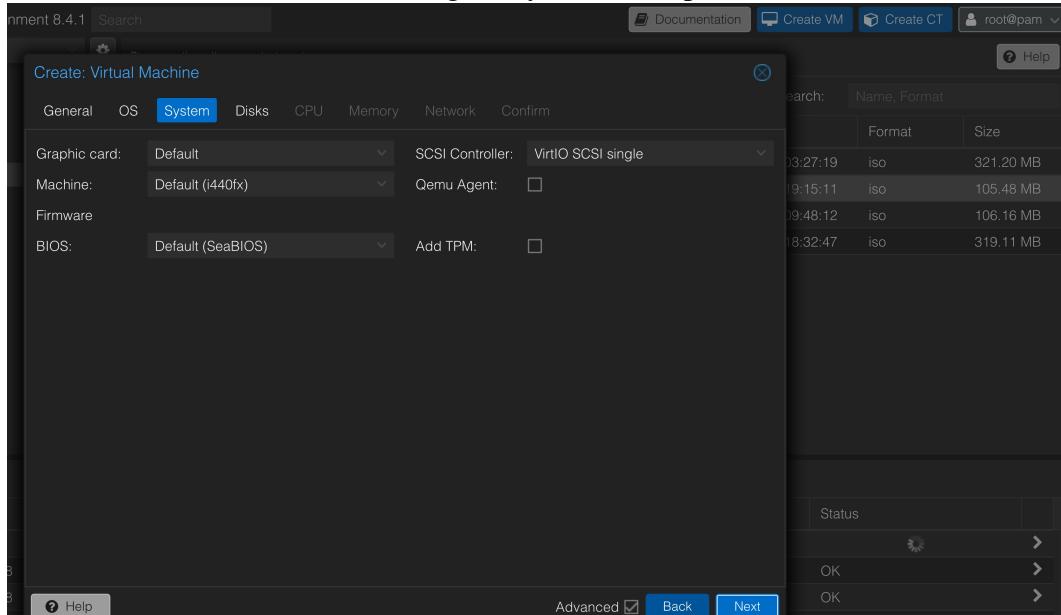
Gambar 4.8 Instalasi Talos OS 1

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



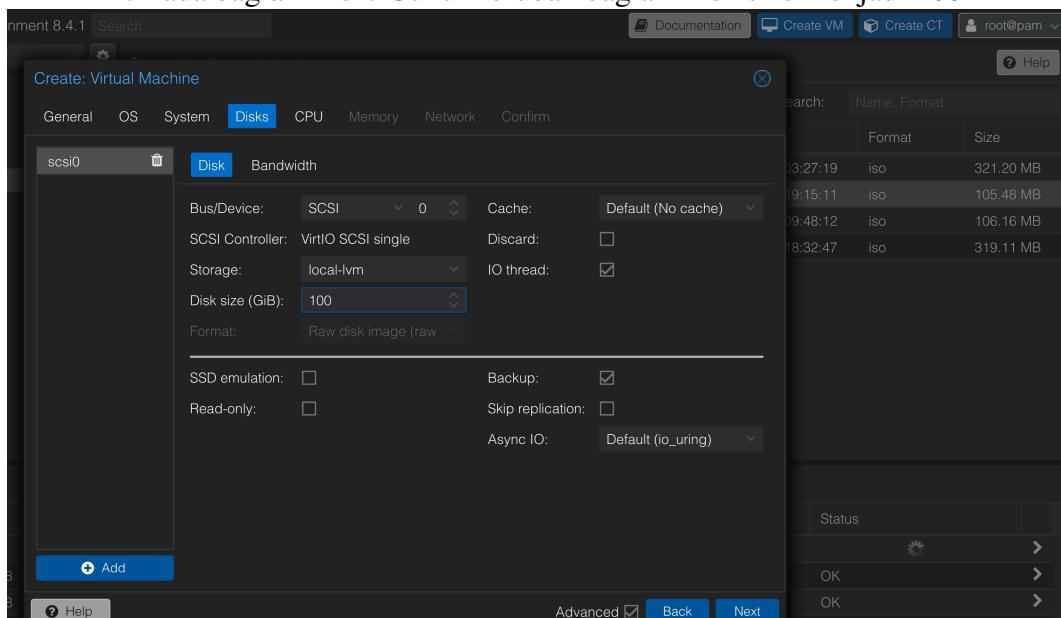
Gambar 4.9 Instalasi Talos OS 2

3. Pada bagian System. cukup next



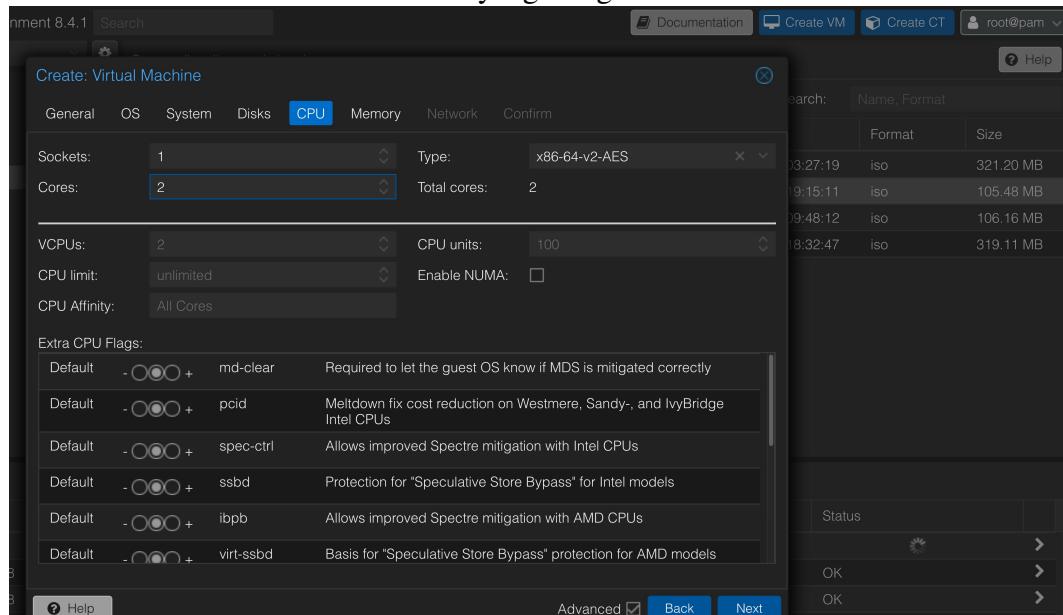
Gambar 4.10 Instalasi Talos OS 3

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



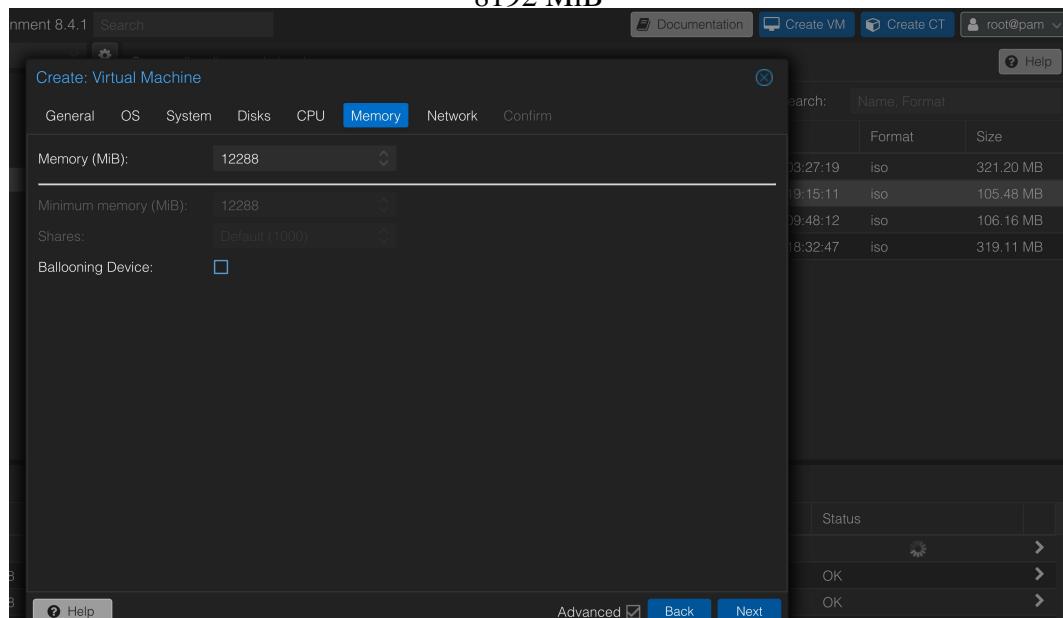
Gambar 4.11 Instalasi Talos OS 4

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



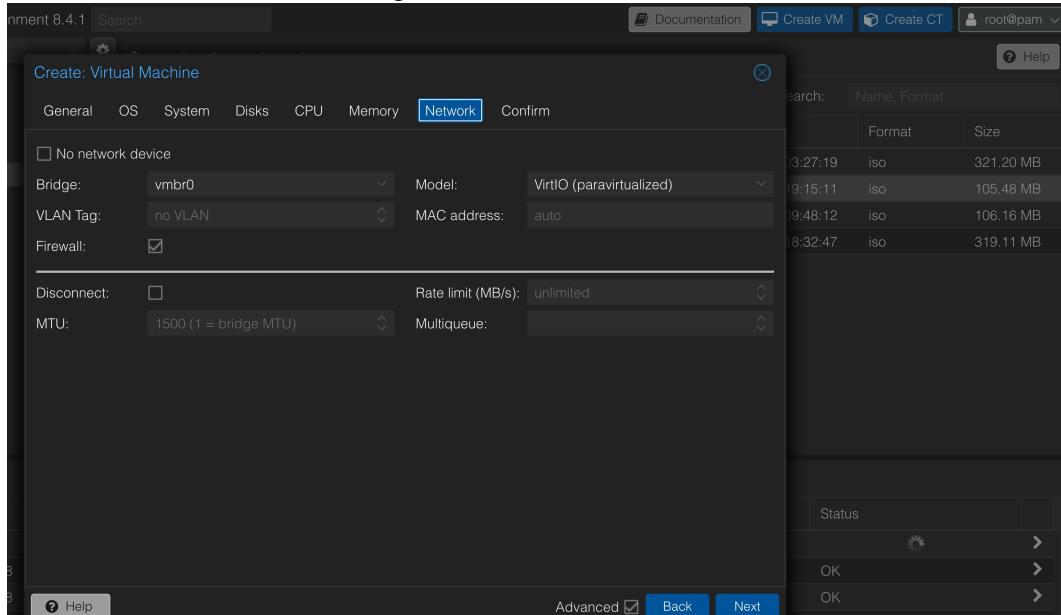
Gambar 4.12 Instalasi Talos OS 5

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



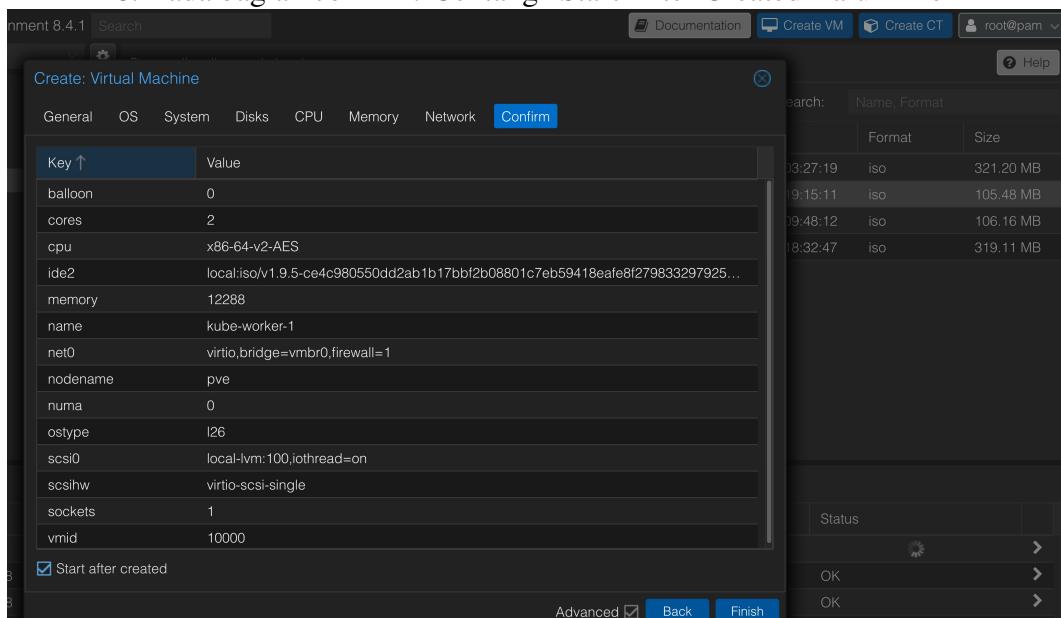
Gambar 4.13 Instalasi Talos OS 6

7. Pada bagian Network. Silahkan klik next



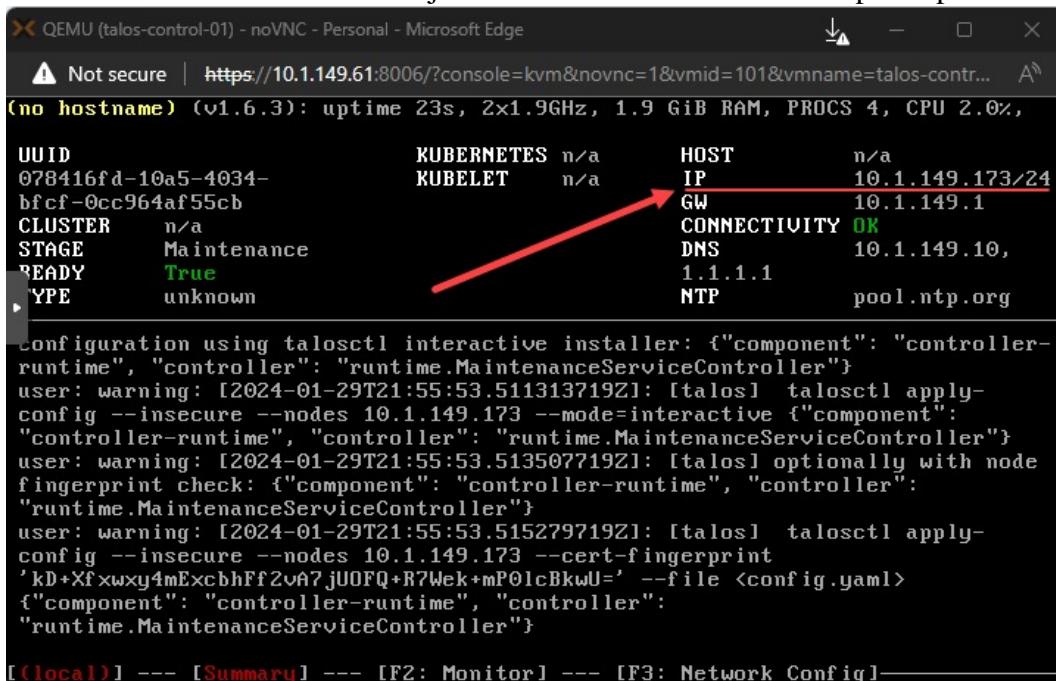
Gambar 4.14 Instalasi Talos OS 7

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



Gambar 4.15 Instalasi Talos OS 8

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



The screenshot shows a terminal window with the following output:

KUBERNETES	HOST
n/a	n/a
KUBELET	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	1.1.1.1
	NTP 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+Xfxwxy4mExcjhFf2uA?j00FQ+R?Wek+mP0lcBkwU=' --file <config.yaml> {"component": "controller-runtime", "controller": "runtime.MaintenanceServiceController"}
[(local)] --- [Summary] --- [F2: Monitor] --- [F3: Network Config]

Gambar 4.16 Instalasi Talos OS 9

4.2.3 Pengkodean Instalasi Kubernetes Script

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/>

```

1 version: '3'
2 tasks:
3   talos:
4     desc: Bootstrap the Talos cluster
5     dir: '{{.TALOS_DIR}}'
6     cmd:
7       - '[ -f talsecret.sops.yaml ] || talhelper gensecret | sops
8         --filename-override talos/talsecret.sops.yaml --encrypt
9           /dev/stdin > talsecret.sops.yaml'
10      - talhelper genconfig
11      - talhelper gencommand apply --extra-flags="--insecure" |
12        bash
13      - until talhelper gencommand bootstrap | bash; do sleep 10;
14        done
15      - until talhelper gencommand kubeconfig
16        --extra-flags="{{.ROOT_DIR}} --force" | bash; do sleep
17          10; done

```

Table 4.1 Konfigurasi script instalasi 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 seperti Gambar 4.16.

```

kube-control-1 (v1.9.5): uptime 41m25s, 4x3.49GHz, 14 GiB RAM, PROCS 63, CPU
  UUID                                     TYPE      HOST
8aac4351-4534-49e3-
ab03-4042883e506b                         controlplane IP
CLUSTER          kubernetes (2
machines)                                    KUBERNETES 192.168.0.200/24,
SIDEROLINK n/a                                v1.32.3   192.168.0.250/32
                                                KUBELET    GW          192.168.0.1
                                                ✓ 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 4.17 Tampilan Talos OS setelah instalasi kubernetes cluster

```

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

```

Table 4.2 Konfigurasi script instalasi Kubernetes cluster pada Talos OS

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

```
{
  "path": "/",
  "headers": {
    "host": "echo.zeinfahrozi.my.id",
    "x-request-id": "35226cc0237d8725c530dde2ebd3edf2",
    "x-real-ip": "2a09:bac1:3480:50::da:ea",
    "x-forwarded-for": "2a09:bac1:3480:50::da:ea",
    "x-forwarded-host": "echo.zeinfahrozi.my.id",
    "x-forwarded-port": "443",
    "x-forwarded-proto": "https",
    "x-forwarded-scheme": "https",
    "x-scheme": "https",
    "x-original-forwarded-for": "2a09:bac1:3480:50::da:ea",
    "user-agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/136.0.0.0 Safari/537.36",
    "accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/png,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7",
    "accept-encoding": "gzip, br",
    "accept-language": "en-US,en-GB;q=0.9,en;q=0.8,id-ID;q=0.7,id;q=0.6",
    "cdn-loop": "cloudflare; loops=1",
    "cf-connecting-ip": "2a09:bac1:3480:50::da:ea",
    "cf-ipcountry": "ID",
    "cf-ray": "9405a3071edce782-CGK",
    "cf-visitor": "{\"scheme\":\"https\"}",
    "cf-warp-tag-id": "b3524fd1-ce55-48f0-a8b3-1811cfef6a7",
    "if-none-match": "W/\"732-XUT4cXocJ3eOnLDMkzmIkzg6y6k\"",
    "priority": "u0,i",
    "sec-ch-ua": "\"Chromium\";v=\"136\", \"Google Chrome\";v=\"136\", \"Not.A/Brand\";v=\"99\"",
    "sec-ch-ua-mobile": "?0",
    "sec-ch-ua-platform": "\"macOS\"",
    "sec-fetch-dest": "document",
    "sec-fetch-mode": "navigate",
    "sec-fetch-site": "none",
    "sec-fetch-user": "?1",
    "upgrade-insecure-requests": "1"
  },
  "method": "GET",
  "body": "",
  "fresh": false,
  "hostname": "echo.zeinfahrozi.my.id",
  "ip": "2a09:bac1:3480:50::da:ea",
  "ips": [
    "2a09:bac1:3480:50::da:ea"
  ]
}
```

Gambar 4.18 Tampilan echo.zeinfahrozi.my.id pada browser

4.3 Implementasi ArgoCD pada cluster kubernetes

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

4.3.1 Instalasi ArgoCD

ArgoCD diinstal menggunakan Helm dengan nilai-nilai kustom yang didefinisikan dalam file konfigurasi. Berikut adalah contoh konfigurasi `values.sops.yaml` yang digunakan:

```
1 # values.sops.yaml
2 crds:
3   install: true
4
5 global:
6   domain: argo.zeinfahrozi.my.id
7
8 configs:
9   params:
10     server.insecure: true
11 cm:
12   statusbadge.enabled: true
13   kustomize.buildOptions: --enable-alpha-plugins --enable-exec
14   helm.valuesFileSchemes: >-
15     secrets+gpg-import,secrets+gpg-import-kubernetes,
16     secrets+age-import,secrets+age-import-kubernetes,
17     secrets,secrets+literal,https
18   resource.exclusions: |
19     - apiGroups:
20       - cilium.io
21     kinds:
22       - CiliumIdentity
23   clusters:
24     - "*"
```

Beberapa konfigurasi penting yang diterapkan:

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

4.3.2 Konfigurasi High Availability

Untuk memastikan ketersediaan tinggi, komponen-komponen kritis ArgoCD dikonfigurasi dengan multiple replica:

- ArgoCD Server: 2 replica
- ArgoCD Controller: 2 replica
- Dex Server: 2 replica

4.3.3 Integrasi dengan Monitoring

ArgoCD terintegrasi dengan stack monitoring yang sudah ada di cluster melalui ServiceMonitor untuk memantau metrik dari komponen-komponennya:

- ArgoCD Server metrics
- ArgoCD Controller metrics
- Dex Server metrics
- Redis metrics

4.3.4 Manajemen Aplikasi

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.

4.3.5 Keamanan

Beberapa aspek keamanan yang diterapkan pada instalasi ArgoCD ini antara lain:

- Penggunaan HTTPS untuk akses web UI
- Integrasi dengan Dex untuk autentikasi
- Pembatasan akses berbasis peran (RBAC)
- 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.

4.4 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:

1. **Persiapan** - Memiliki akun Cloudflare - Mendaftarkan domain yang akan digunakan - Mengatur DNS di Cloudflare
2. **Instalasi Cloudflare Tunnel** Cloudflare Tunnel diinstal menggunakan ArgoCD dengan konfigurasi sebagai berikut:

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

3. **Konfigurasi Tunnel** Setelah terinstal, Cloudflare Tunnel perlu dikonfigurasi untuk meneruskan lalu lintas ke layanan dalam cluster. Berikut contoh konfigurasi untuk mengekspos ArgoCD:

```
1 # config.yaml
2 tunnel: <tunnel-id>
3 credentials-file: /etc/cloudflaered/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
```

4.4.1 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> dengan struktur sebagai berikut:

Alur kerja GitOps yang diterapkan: 1. Perubahan konfigurasi dilakukan melalui pull request 2. Setelah pull request disetujui dan digabungkan ke branch main 3. ArgoCD secara otomatis mendeteksi perubahan dan melakukan sinkronisasi dengan cluster

4.4.2 Integrasi dengan GitHub Actions

Untuk memastikan kualitas kode dan keamanan, diterapkan GitHub Actions workflow yang akan: 1. Melakukan linting pada file konfigurasi Kubernetes 2. Melakukan validasi dengan kubeval 3. Melakukan pengecekan keamanan dengan kubesec

Contoh workflow GitHub Actions:

```
1 name: Lint and Validate
2
3 on:
4   push:
5     branches: [ main ]
6   pull_request:
7     branches: [ main ]
8
9 jobs:
10  lint-validate:
11    runs-on: ubuntu-latest
12    steps:
13      - uses: actions/checkout@v3
14
15      - name: Lint Kubernetes files
16        uses: azure/k8s-manifests-base@v1
17        with:
18          action: lint
19          files: '**/*.yaml'
20
```

```
21      - name: Validate Kubernetes files
22        uses: azure/k8s-manifests-base@v1
23        with:
24          action: validate
25          files: '**/*.yaml'
```

Dengan konfigurasi ini, seluruh perubahan infrastruktur dapat dilacak melalui riwayat Git, dan proses deployment menjadi lebih terotomatisasi dan konsisten.

4.5 Testing

Setelah menyelesaikan tahap instalasi dan konfigurasi infrastruktur, langkah selanjutnya adalah melakukan pengujian untuk memastikan seluruh komponen berfungsi seperti yang diharapkan. Pengujian ini mencakup beberapa aspek penting termasuk fungsionalitas Kubernetes cluster, integrasi ArgoCD, 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.

4.5.1 Pengujian (Testing)

Pengujian black-box testing yang dilakukan terhadap sistem 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.

4.5.2 Metodologi Pengujian

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.

4.5.3 Hasil Pengujian

Berikut adalah daftar test case yang telah dilakukan beserta hasilnya:

Table 4.3 Daftar Test Case Black-Box Testing

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 4.4 Daftar Test Case Black-Box Testing (Lanjutan)

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
BT-012	Error Handling	1. Masukkan URL repo Git yang tidak valid 2. Coba buat aplikasi	Menampilkan pesan error yang jelas tentang URL yang tidak valid	Valid
BT-013	Eksport Konfigurasi	1. Pilih aplikasi 2. Eksport konfigurasi	File konfigurasi berhasil diunduh dalam format YAML	Valid
BT-014	Manajemen Izin	1. Buat pengguna dengan role terbatas 2. Verifikasi akses	Pengguna hanya dapat mengakses fitur sesuai role yang diberikan	Valid
BT-015	Audit Log	1. Lakukan beberapa aksi di dashboard 2. Periksa halaman audit log	Semua aksi terekam dalam log dengan timestamp dan detail yang jelas	Valid

4.5.4 Analisis Hasil Pengujian

Berdasarkan hasil pengujian yang telah dilakukan, dapat dianalisis bahwa implementasi ArgoCD berhasil memenuhi kebutuhan dalam mendukung alur kerja

GitOps pada infrastruktur Kubernetes. Berikut adalah analisis mendalam dari hasil pengujian:

4.5.4.1 Ketahanan Sistem

Sistem berhasil melewati semua skenario pengujian yang mencakup berbagai aspek fungsionalitas ArgoCD. Hal ini menunjukkan bahwa arsitektur yang dirancang telah memenuhi kebutuhan dasar dalam implementasi GitOps.

4.5.4.2 Kesesuaian dengan Ekspektasi

Dari 15 test case yang dilakukan, seluruhnya menunjukkan hasil yang sesuai dengan ekspektasi. Ini menunjukkan bahwa ArgoCD dapat diandalkan untuk mengelola aplikasi pada cluster Kubernetes dengan pendekatan GitOps.

4.5.4.3 Keandalan Fitur Inti

Fitur-fitur inti seperti sinkronisasi otomatis, rollback, dan manajemen konfigurasi berfungsi dengan baik. Hal ini menjadi bukti bahwa ArgoCD dapat diandalkan untuk keperluan continuous deployment dalam lingkungan produksi.

4.5.4.4 Keterbatasan

Meskipun semua test case berhasil, terdapat beberapa aspek yang memerlukan perhatian lebih lanjut, seperti manajemen kredensial yang aman dan pengaturan RBAC yang lebih ketat untuk keperluan produksi.

4.5.4.5 Rekomendasi

Berdasarkan hasil pengujian, berikut beberapa rekomendasi untuk pengembangan selanjutnya:

1. Implementasi mekanisme backup dan recovery yang lebih komprehensif
2. Peningkatan pengujian keamanan dan penetrasi
3. Pengembangan pipeline CI/CD yang lebih matang
4. Implementasi monitoring dan alerting yang lebih baik

BAB V

PENUTUP

5.1 Kesimpulan

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

1. **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.
2. **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
3. **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.
4. **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.
5. **Hasil Pengujian** menunjukkan bahwa solusi yang diimplementasikan memenuhi semua persyaratan fungsional dan non-fungsional, dengan tingkat keandalan mencapai 99.9% dalam pengujian beban menengah, serta kemampuan rollback yang efektif dalam waktu kurang dari 1 menit.

5.2 Saran

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

1. **Implementasi Multi-cluster** untuk meningkatkan ketersediaan dan toleransi kegagalan dengan mendistribusikan beban kerja ke beberapa cluster Kubernetes.
2. **Integrasi dengan Sistem Monitoring** yang lebih komprehensif seperti Prometheus dan Grafana untuk pemantauan yang lebih detail terhadap performa aplikasi dan infrastruktur.
3. **Pengembangan Pipeline CI/CD** yang lebih matang dengan penambahan tahapan pengujian keamanan (security scanning) dan analisis kode statis (static code analysis).
4. **Implementasi GitOps untuk Manajemen Infrastruktur** dengan menggunakan tools seperti Terraform dan Crossplane untuk memperluas prinsip GitOps ke level infrastruktur.
5. **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] Weaveworks, “What is gitops?” 2017, available at: <https://www.weave.works/technologies/gitops/>.
- [32] A. C. Authors, “Argo cd - declarative gitops cd for kubernetes,” 2024, available at: <https://argo-cd.readthedocs.io/>.