TESIS

ANALISIS PENGGUNAAN METODE GITOPS PADA PENGEMBANGAN PERANGKAT LUNAK BERBASIS DEVOPS

Disusun oleh:

Nama : **Ramadoni**
NIM : **20.77.1287**
Konsentrasi : **Informatics Technopreneurship**

PROGRAM STUDI S2 TEKNIK INFORMATIKA
PROGRAM PASCASARJANA UNIVERSITAS AMIKOM YOGYAKARTA
YOGYAKARTA
2021

TESIS

ANALISIS PENGGUNAAN METODE GITOPS PADA PENGEMBANGAN PERANGKAT LUNAK BERBASIS DEVOPS

ANALYSIS OF GITOPS METHOD USAGE IN MODERN DEVOPS-BASED SOFTWARE DEVELOPMENT

Diajukan untuk memenuhi salah satu syarat memperoleh derajat Magister

Disusun oleh:

Nama : **Ramadoni**
NIM : **20.77.1287**
Konsentrasi : **Informatics Technopreneurship**

PROGRAM STUDI S2 TEKNIK INFORMATIKA
PROGRAM PASCASARJANA UNIVERSITAS AMIKOM YOGYAKARTA
YOGYAKARTA
2021

HALAMAN PENGESAHAN

ANALISIS PENGGUNAAN METODE GITOPS PADA PENGEMBANGAN PERANGKAT LUNAK BERBASIS DEVOPS

ANALYSIS OF GITOPS METHOD USAGE IN MODERN DEVOPS-BASED SOFTWARE DEVELOPMENT

Dipersiapkan dan Disusun oleh

Ramadoni

20.77.1287

Telah Diujikan dan Dipertahankan dalam Sidang Ujian Tesis
Program Studi S2 Teknik Informatika
Program Pascasarjana Universitas AMIKOM Yogyakarta
pada hari NamaHari, tanggal ujian tesis

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

Yogyakarta, 06 Desember 2021
Rektor

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

HALAMAN PERSETUJUAN

ANALISIS PENGGUNAAN METODE GITOPS PADA PENGEMBANGAN PERANGKAT LUNAK BERBASIS DEVOPS

ANALYSIS OF GITOPS METHOD USAGE IN MODERN DEVOPS-BASED SOFTWARE DEVELOPMENT

Dipersiapkan dan Disusun oleh

Ramadoni

20.77.1287

Telah Diujikan dan Dipertahankan dalam Sidang Ujian Tesis
Program Studi S2 Teknik Informatika
Program Pascasarjana Universitas AMIKOM Yogyakarta
pada hari NamaHari, tanggal ujian tesis

Pembimbing Utama	**Anggota Tim Penguji**
Prof. Dr. Ema Utami, S.Si., M.Kom.	**Alva Hendi Muhammad, S.T., M.Eng., Ph.D.**
NIK. 190302037	**NIK. 190302493**
	Dr. Dhani Ariatmanto, M.Kom.
Pembimbing Pendamping	**NIK. 190302197**
Hanif Al Fatta, M.Kom.	**Prof. Dr. Ema Utami, S.Si., M.Kom.**
NIK. 190302096	**NIK. 190302037**

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

Yogyakarta, 06 Desember 2021
Direktur Program Pascasarjana

Dr. Kusrini, M.Kom.
NIK. 190302106

HALAMAN PERNYATAAN KEASLIAN TESIS

Yang bertandatangan di bawah ini,

Nama mahasiswa : Ramadoni
NIM : 20.77.1287
Konsentrasi : Informatics Technopreneurship

Menyatakan bahwa Tesis dengan judul berikut:
Analisis Penggunaan Metode Gitops pada Pengembangan Perangkat Lunak Berbasis Devops

Dosen Pembimbing Utama : Prof. Dr. Ema Utami, S.Si., M.Kom.
Dosen Pembimbing Pendamping : Hanif Al Fatta, M.Kom.

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

Yogyakarta, 15 Desember 2021
Yang Menyatakan,

Ramadoni

HALAMAN PERSEMBAHAN

Dengan mengucap puji syukur kepada Allah S.W.T, sholawat serta salam kami haturkan kepada Nabi Muhammad S.A.W. Tesis ini kupersembahkan untuk:

1. Bapak Ashudi (Alm.) dan Ibu Sarmi tercinta yang telah merawat membesarkan aku dan selalu membimbing, mendukung, memotivasi, memberi apa yang terbaik bagiku serta selalu mendoakan aku untuk meraih kesuksesanku.
2. Bapak Suwanto & Ibu Agustina yang selalu memberikan motivasi dan dorongan agar aku sukses dalam sekolah dan karirku.
3. Istriku tercinta (WRPS Galuh Amanda) yang selalu mencintai, menyayangi dan mendukung aku dalam studi maupun karir.
4. Anakku Alifia Aqila Ramadani dan Qinara Adzkia Mumtaza yang selalu menjadi penyemangatku untuk selalu menjadi lebih baik, Ayah sangat mencintai dan menyayangi kalian.
5. Kakak, adik, saudara, teman yang telah ikut mendo'akan agar studiku berjalan lancar.
6. Teman-teman PJJ Universitas AMIKOM Yogyakarta angkatan 3 yang sudah menjadi seperti keluarga.

HALAMAN MOTTO

Barang siapa yang menghendaki kehidupan dunia, maka wajib baginya memiliki ilmu, dan barang siapa yang menghendaki kehidupan Akhirat, wajib baginya memiliki ilmu, dan barang siapa menghendaki keduanya maka wajib baginya memiliki ilmu.

(HR. At-Tirmidzi)

KATA PENGANTAR

Puji syukur alhamdullillah, penulis panjatkan kehadirat Allah, SWT, yang telah melimpahkan rahmat dan karunia-Nya. Shalawat dan salam penulis haturkan atas Nabi Muhammad SAW, keluarganya, para sahabatnya, dan pengikutnya yang setia sampai akhir zaman, sehingga penulis dapat menyelesaikan tesis dengan judul **"Analisis Penggunaan Metode Gitops pada Pengembangan Perangkat Lunak Berbasis Devops"** yang digunakan untuk memenuhi salah satu persyaratan untuk memperoleh gelar Magister Komputer.

Dengan segala kerendahan dan ketulusan hati, penulis menyampaikan penghargaan dan ucapan terima kasih yang sedalam-dalamnya kepada semua pihak yang telah membantu memberikan arahan, bimbingan, dan motivasi, baik secara langsung maupun tidak langsung, sehingga tesis ini dapat diselesaikandengan baik, yaitu kepada:

1. Bapak Prof. Dr. M. Suyanto, MM. selaku Rektor Universitas AMIKOM Yogyakarta.
2. Prof. Dr. Kusrini, M.Kom. selaku Direktur Program Pascasarjana Universitas AMIKOM Yogyakarta.
3. Prof. Dr. Ema Utami, S.Si., M.Kom selaku Dosen Pembimbing Utama Tesis yang tanpa mengenal lelah memberikan bimbingan, dukungan, semangat dan masukan kepada penulis.
4. Bapak Hanif Al Fatta, M.Kom. selaku Dosen Pembimbing Pendamping Tesis.

5. Staff /Karyawan /Dosen di lingkungan Universitas AMIKOM Yogyakarta.
6. Kedua orang tua, mertua, kakak dan adik serta seluruh keluarga tercinta yang telah memberikan doa dan dukungan selama ini tanpa kenal lelah.
7. Teman-teman semua di PJJ Universitas AMIKOM Yogyakarta.
8. Semua pihak yang telah memberikan bantuan baik tenaga maupun pikiran dalam penyelesaian tesis ini.

Penulis menyadari bahwa dalam penulisan karya tulis ini masih terdapat kekurangan, baik dalam analisis maupun cara penyajian materi. Oleh karena itu, kritik dan saran sangat penulis harapkan demi sempurnanya tesis ini. Semoga tesis ini dapat memberikan manfaat kepada pembaca.

Yogyakarta, 15 Desember 2021

Penulis

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

INTISARI

Saat ini permasalahan yang dihadapi oleh pengembang perangkat lunak dalam mengimplementasikan metode DevOps antara lain adalah masalah keamanan, yaitu kemampuan seseorang untuk secara langsung mengakses dan mengubah klaster, serta proses rollback yang tidak efektif dalam proses deployment aplikasi ke platform aplikasi.

Dalam penelitian ini, metode GitOps digunakan untuk menganalisa dan menguji bagaimana GitOps mampu memecahkan masalah ini. Penelitian ini menggunakan pendekatan *pull-based deployment* dan *declarative deployment* sebagai pengganti model *push-based deployment* yang biasa digunakan dalam pipeline CI/CD saat ini. Argo CD berfungsi sebagai operator, dan Kubernetes berfungsi sebagai platform untuk penerapan aplikasi berbasis kontainer.

Penelitian ini membuktikan bahwa model GitOps dapat memecahkan masalah keamanan terkait akses langsung ke klaster Kubernetes, karena GitOps menggunakan metode *single source of truth* dimana semua perubahan harus melalui Git dan semua perubahan juga akan tercatat secara jelas, tidak dimungkinkan lagi perubahan terjadi secara langsung tanpa melalui Git. Model GitOps ini juga mampu menjawab dan menjadi solusi terkait kurang efektifnya proses *rollback* yang ada saat ini dalam hal kemudahan melakukan *rollback* serta dalam hal kecepatan mengeksekusi *rollback* hingga 30 kali lipat lebih cepat dibandingkan menggunakan metode non GitOps, dari 30 detik menjadi 0.5 detik. Kajian ini juga terbukti dapat menjadi rekomendasi bagi perusahaan atau institusi serta individu yang ingin mulai mengadopsi DevOps atau yang ingin membawa implementasi DevOps ke level selanjutnya.

Kata kunci: GitOps, DevOps, Declarative Deployment, Pull-based Deployment, Argo CD

ABSTRACT

Currently, the problems faced by software developers in implementing the DevOps method include security issues of a person's ability to directly access and change clusters and the ineffective rollback process in the application deployment process to an application platform.

The GitOps method was applied in this study to investigate and test how GitOps was able to solve these problems. This research used pull-based deployment and declarative deployment approaches rather than the push-based deployment models commonly used in today's CI/CD pipelines. The Argo CD tool serves as an operator, and Kubernetes serves as a platform for deploying container-based applications.

This study proves that the GitOps model can solve security problems related to direct access to Kubernetes clusters, because GitOps uses a single source of truth method where all changes must go through Git and all changes will also be recorded clearly, it is no longer possible for changes to occur directly without going through Git. This GitOps model is also able to answer and can be a solution regarding the current ineffectiveness of the rollback process in terms of the ease of doing rollbacks and in terms of the speed of executing rollbacks up to 30 times faster than using non-GitOps methods, from 30 seconds to 0.5 second. This study also proves to be a recommendation for companies or institutions as well as individuals who want to start adopting DevOps or who want to take DevOps implementation to the next level.

Keyword: GitOps, DevOps, Declarative Deployment, Pull-based Deployment, Argo CD

BAB I

PENDAHULUAN

1.1. Latar Belakang Masalah

Perancang atau pembuat perangkat lunak telah memperkenalkan banyak model pengembangan dengan berbagai proses baik saat perancangan, pembuatan, penerapan, alur kerja dan konsep metode untuk menghasilkan perangkat lunak berkualitas. Pada umumnya, kualitas ini terdiri dari dua jenis: pertama, kualitas proyek yang mengacu pada masalah proses yang mempengaruhi proyek secara keseluruhan; seperti rentang waktu dan biaya pengelolaan. Kedua, kualitas produk, yang mengacu pada masalah yang terkait dengan produk perangkat lunak itu sendiri; seperti kinerja tinggi, keandalan, ketepatan, dan sebagainya. Kaitannya dengan menyeimbangkan antara dua jenis kualitas ini, maka para perancang atau pembuat perangkat setuju bahwa peningkatan pada proses adalah salah satu solusi masalah keterlambatan *delivery*, biaya, dan kualitas pada produk akhirnya (Abbass et al., 2019).

Berdasarkan hal ini dan juga dikarenakan semakin meningkatnya persaingan di pasar perangkat lunak, perancang atau pembuat perangkat lunak berlomba-lomba memberikan perhatian dan mengalokasikan sumber daya untuk mengembangkan dan merilis perangkat lunak berkualitas tinggi secara lebih cepat. Salah satu metode pengembangan perangkat lunak yang digunakan adalah metode DevOps. DevOps diartikan sebagai menghilangkan pembatas antara dua tim yang

biasanya terisolasi dalam struktur IT, yaitu tim pengembangan (Dev) dan tim pengoperasian (Ops) (Bolscher & Daneva, 2019).

Konsep DevOps ini memungkinkan peran kedua tim tersebut untuk saling berkoordinasi dan berkolaborasi untuk menghilangkan proses-proses manual diganti menjadi proses yang terintegrasi, dan terotomasi dengan bantuan perangkat-perangkat DevOps. Tujuan utamanya adalah menghasilkan produk dengan cepat, lebih baik dan lebih handal. Pada praktiknya, DevOps menggunakan beberapa sub metode diantaranya adalah proses integrasi secara berkelanjutan atau yang biasa disebut dengan *Continuous Integration (CI),* pengiriman atau umpan balik secara berkelanjutan atau Continuous Delivery (CDE) serta proses penerapan secara berkelanjutan atau *Continuous Deployment (CD)*. Hal ini dikemukakan oleh Kim et al. (2020) dalam bukunya berjudul The Phoenix Project yang mengemukakan tentang tiga cara yang merupakan prinsip yang mendasari proses dan praktik DevOps. Berkelanjutan diartikan sebagai terus terulangnya pola yang sama secara konsisten dan sistematis, mulai dari proses membangun atau membuat, menerapkan, mengoperasikan, hingga proses kontrol terhadap kualitas. Tiga submetode tersebut dapat menjadi solusi dan sangat membantu perancang atau pembuat perangkat lunak untuk mempercepat pengembangan produk atau perangkat lunak mereka dan merilis fitur secara cepat tanpa mengorbankan kualitas dari produknya (Abbass et al., 2019).

CI, CDE dan CD disebut sebagai pengembangan perangkat lunak berkelanjutan. Terkait erat dengan DevOps, praktik-praktik ini bekerja sama untuk memungkinkan alur kerja dengan memanfaatkan otomasi guna mengembangkan,

membangun, menguji, dan menerapkan kode yang berkualitas lebih tinggi secara lebih cepat. Metode ini memungkinkan organisasi untuk memahami dan merespon perubahan pasar dengan lebih cepat dan lebih mudah, dan telah terbukti bahwa pengembangan perangkat lunak berkelanjutan berkorelasi dengan meningkatnya performa dari sebuah organiasi dan kesuksesan bisnis (Bolscher & Daneva, 2019).

Meskipun penerapan metode DevOps ini memiliki manfaat yang nyata, proses implementasi CI, CDE dan CD ini tidaklah mudah. Proses pendefinisian tugas, proses menterjemahkan proses manual menjadi proses yang mampu dijalankan secara otomatis, proses pemilihan tools, proses adaptasi dalam menggunakan tools, miskonfigurasi dan lain-lain pasti akan menjadi tantangan tersendiri. Bahkan ketika sebuah tim proyek telah berhasil memperkenalkan budaya CI, CDE dan CD dalam proyek mereka, beberapa proses masih menjadi masalah tersendiri, diantara masalah yang dihadapi adalah terkait arsitektur (Bolscher & Daneva, 2019), masalah terkait tools (Proulx et al., 2018) dan metode baru (Abbass et al., 2019) (Leite et al., 2019), keamanan pada pipeline CI/CD (Shahin, Ali Babar, et al., 2017), serta masalah terkait tidak adanya mekanisme *rollback* yang efektif (Agarwal et al., 2019).

Berdasarkan masalah yang sudah dijelaskan diatas, penulis tertarik untuk melakukan analisis mengenai penggunaan metode GitOps, fokus pada bagaimana metode GitOps dapat membantu mengatasi masalah-masalah tersebut, GitOps adalah cara mengimplementasikan *Continuous Deployment* menggunakan Git sebagai *Single Source of Truth*. Di GitOps, seluruh status penerapan didefinisikan dalam repositori Git dan semua perubahan atau manajemen harus melaluinya, serta

memperkenalkan ArgoCD sebagai salah satu GitOps *tool* yang bisa membantu dalam proses penerapan perangkat lunak.

1.2. Rumusan Masalah

Berdasarkan latar belakang masalah yang telah dijelaskan di atas maka rumusan masalah dalam penelitian ini adalah sebagai berikut:

a. Seberapa besar effort yang dibutuhkan untuk mengadopsi GitOps, baik dari sisi perusahaan, pengembang maupun tim operasi?
b. Apakah penggunaan metode GitOps mampu mengatasi masalah keamanan dan efektifitas proses rollback pada perusahaan?
c. Faktor apa saja yang mempengaruhi keberhasilan penerapan GitOps dalam pengembangan perangkat lunak?

1.3. Batasan Masalah

Pembatasan suatu masalah digunakan untuk menghindari adanya penyimpangan maupun pelebaran pokok masalah agar penelitian tersebut lebih terarah dan memudahkan dalam pembahasan sehingga tujuan penelitian akan tercapai. Beberapa batasan masalah dalam penelitian ini adalah sebagai berikut:

a. Penggunaan pendekatan *pull-based deployment* dan *declarative deployment* sebagai pengganti model *push-based deployment* yang selama ini digunakan, dua model ini akan di-setup dan diuji, sebagai bagian dari proses pengujian mandiri.

b. Perangkat bantu yang digunakan adalah ArgoCD, sebuah proyek sumber terbuka *(open source)* yang dipersembahkan oleh komunitas Argo dan Intuit yang berfungsi sebagai perangkat bantu sistem *Continuous Delivery* (CD) deklaratif berlisensi Apache License 2.0. Penelitian ini didesain menggunakan pendekatan *pull-based deployment* dan *declarative deployment* sebagai bahan verifikasi terkait proses *Continuous Deployment* menggunakan model GitOps, yang akan di terapkan pada lingkungan Kubernetes sebagai platform akhir pemasangan aplikasi (*docker-based*) dan Github sebagai platform *Version Control System*.

c. Menggunakan pendekatan *docker image* sebagai basis manajemen paketnya. Aplikasi yang akan digunakan untuk proses pengujian, proses dari mulai *compile, test, package, run* dijalankan melalui metode docker dengan menggunakan *dockerfile*.

d. Masalah yang akan diuji dan diteliti terbatas pada masalah keamanan terkait permasalahan keamanan ketika seseorang bisa mengakses langsung ke klaster sehingga dimungkinkan untuk mengubah dan menghapus deployment yang ada serta masalah pada tidak efektifnya proses rollback dimana proses rollback tidak mudah dilakukan serta membutuhkan waktu yang lama.

e. Skenario pengujian keamanan yang akan dilakukan pada penelitian ini adalah dengan melakukan pengujian deployment aplikasi menggunakan metode non GitOps, kemudian melakukan perubahan secara langsung ke klaster, mencatat semua kejadian, kemudian membandingkan jika perubahan tersebut dilakukan pada metode GitOps. Sedangkan pada pengujian *rollback* dilakukan dengan

melakukan proses *rollback* pada metode non GitOps, kemudian mencatat *effort* dan berapa lama waktu yang dibutuhkan, kemudian membandingkan jika *rollback* dilakukan menggunakan metode GitOps dengan bantuan ArgoCD.

1.4. Tujuan Penelitian

Adapun tujuan penulis dalam melakukan penelitian ini adalah sebagai berikut:

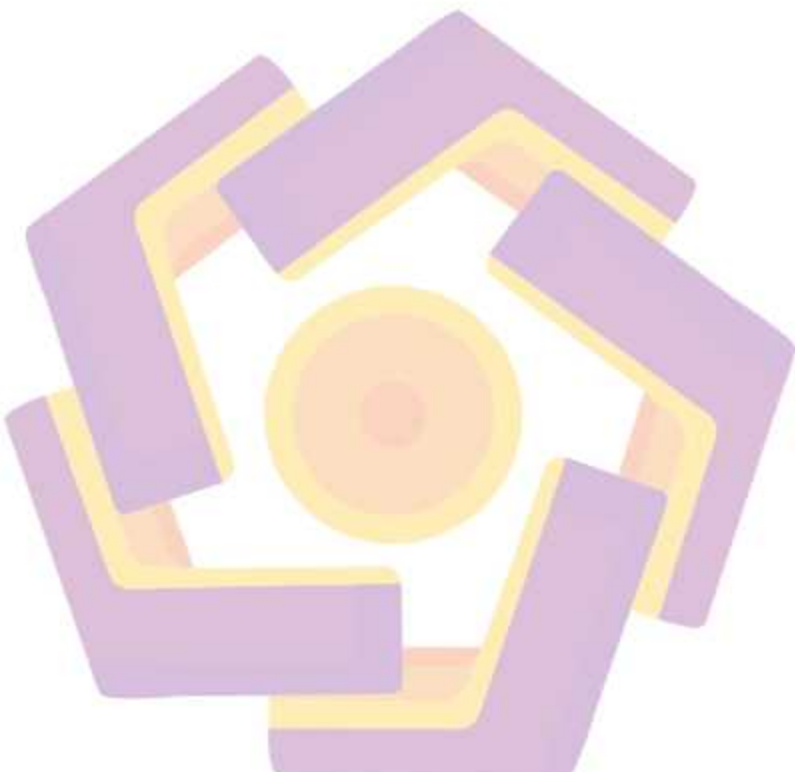
a. Mengetahui seberapa besar effort yang dilakukan saat mengadopsi GitOps baik dari sisi perusahaan, pengembang perangkat lunak dan tim operasional.
b. Memberikan gambaran penerapan GitOps untuk menyelesaikan permasalahan keamanan tentang kemampuan seseorang mengakses langsung klaster serta masalah kurang efektifnya proses rollback yang ada pada metode DevOps saat ini.
c. Membuat model penerapan praktik berkelanjutan dengan menggunakan metode GitOps.

1.5. Manfaat Penelitian

Manfaat yang didapatkan dari penelitian ini adalah sebagai berikut:

a. Dapat menjadi rekomendasi bagi perusahaan yang ingin mulai mengadopsi DevOps atau yang ingin meningkatkan penerapan DevOps menggunakan metode GitOps.
b. Dengan mengadopsi GitOps, proses pengembangan hingga proses approval penerapan perangkat lunak akan menjadi lebih cepat, menciptakan lingkungan

pengembangan yang aman menggunakan *single source of truth* serta semua perubahan yang terjadi pada proses pengembangan akan terdokumentasi dengan baik.



BAB II

TINJAUAN PUSTAKA

2.1. Tinjauan Pustaka

Pada penelitian (Bolscher & Daneva, 2019), peneliti menyebutkan bahwa salah satu penghambat dalam transforamasi ke Devops adalah masalah arsitektur, beberapa tipe karakteristik arsitektur yang bisa membantu dalam proses transformasi ke DevOps adalah *Production Versioning* dan *Rollback*. *Production Versioning* diartikan sebagai kemampuan untuk mempunyai beberapa versi pada servis yang sama di lingkungan produksi secara simultan, sedangkan karakter *Rollback* adalah kemampuan untuk mengurangi dampak kesalahan saat sebuah proses yang dilakukan berkali-kali dalam sehari mengalami kegagalan, sehingga dengan mudah mengembalikan ke *stage* normal sebelumnya. Kesamaan penelitian tersebut dengan penelitian yang sedang peneliti lakukan sat ini terletak pada proses mengkaji metode DevOps dan menjelaskan bagaimana cara mengadopsinya, sedangkan perbedaannya adalah pada penelitian yang dilakukan oleh peneliti saat ini menambahkan metode Gitops dalam rangka menyelesaikan permasalahan yang ditemukan di penelitian sebelumnya.

Selanjutnya pada penelitian (Abbass et al., 2019) peneliti memaparkan bahwa implementasi DevOps dalam hal ini *Continuous Delivery* menuntut sebuah perusahaan berubah dari metode lama ke ke metode baru, sehingga tantangan dan masalah sangat mungkin terjadi pada proses adopsinya. Masalah ini biasanya terkait dengan penggunaan alat dan metode yang baru. Masalah-masalah ini dapat

mengurangi manfaat *Continuous Delivery* atau bahkan membuat penerapan *Continuous Delivery* bertentangan dengan perusahaan. Penelitian tersebut memiliki kesamaan dengan penelitian yang dilakukan penulis saat ini, yaitu terdapat pembahasan tentang manfaat *Continuous Delivery* pada metode DevOps serta permasalahan terkait pemilihan tools dan metode saat hendak mengadopsinya, pada penelitian ini juga dibahas mengenai tools dan metode yang cocok saat ingin mengadopsi GitOps.

Merilis perangkat lunak kepada pelanggan berpotensi berisiko bagi penyedia perangkat lunak karena pelanggan mereka mungkin saja mendapatkan perangkat lunak yang buggy. Masalah ini dapat meningkat ketika proses adopsi DevOps telah dilakukan dengan mempraktikkan *Continuous Deployment*, karena proses penerapan ke lingkungan produksi dilakukan terus menerus dan dilakukan secara otomatis. Sangat penting bagi organisasi pembuat perangkat lunak untuk mengadopsi praktik untuk mengurangi potensi risiko dan masalah dalam waktu rilis, diantara praktik yang diidentifikasi penulis untuk tujuan ini adalah pembatasan testing hanya pada lingkungan terbatas dan menyiapkan proses *rollback* yang mudah untuk kembali ke *stable state*. Dalam penelitian tersebut juga dibahas kemungkinan masalah pada sisi keamanan, dimana *deployment pipeline* dapat terancam oleh kode berbahaya yang disebarkan melalui *pipeline* dan komunikasi langsung antara komponen dalam lingkungan pengujian dan produksi (Shahin, Ali Babar, et al., 2017). Biasanya, pipeline melakukan pembaruan langsung ke kluster komputer. Jadi, mereka mempunyai otorisasi untuk mengubah struktur kluster dan mengakses semua sumber dayanya yang memungkinkan penyerang untuk

menghapus atau mengekstrak semua data yang disimpan dan mengubah konfigurasi kluster dengan berbagai cara. Hal ini sejalan dengan penelitian yang penulis lakukan saat ini bahwa proses rollback yang tidak efektif serta penggunaan CI server yang memungkinkan terjadinya masalah pada keamanan karena CI server mempunyai akses langsung ke kluster akan dibahas dan disolusikan.

Pada penelitian (Proulx et al., 2018), penulis mengkategorikan tantangan yang dihadapi ketika mengadopsi *Continuous Deployment* menjadi enam kategori diantaranya adalah *Process* dan *Tools*. Penulis juga menuliskan salah satu solusi pada kategori *tools* adalah dengan memiliki konfigurasi pipeline dalam bentuk kode, sehingga pengembang dapat meletakkan konfigurasi *pipeline* dalam sistem kontrol versi (seperti Git atau Subversion), ini berarti bahwa semua riwayat konfigurasi selalu disimpan dan mudah untuk mengembalikan ke konfigurasi sebelumnya (*rollback*). Penelitian tersebut menegaskan tentang manfaat penggunaan sistem kontrol versi pada pipeline yang merupakan dasar dari metode GitOps yang menggunakan sistem versi kontrol sebagai satu-satunya sumber yang terpercaya untuk menyimpan, melakukan perubahan serta mengelola konfigurasi sistem.

(Agarwal et al., 2019) menuliskan dalam penelitiannya bahwa salah satu tantangan terbesar pada penerapan CI/CD adalah tidak adanya metode *rollback* yang efektif. Berdasarkan *2017 State of Devops Report*, sekitar 24.5% organisasi yang disurvei menolak penerapan otomatis karena mekanisme *rollback* yang digunakan masih secara manual. Problem yang terjadi pada penelitian sebelumnya

terkait pada proses rollback yang dilakukan secara manual akan diobservasi dan disolusikan pada penelitian ini.

(Leite et al., 2019) menyimpulkan bahwa implikasi dari penerapan DevOps memengaruhi cara engineer merancang sistem, berinteraksi dengan rekan mereka, dan bahkan cara mengadopsi proses, seperti penanganan insiden. DevOps harus dapat melakukan rollback aplikasi jika terjadi masalah setelah penerapan. Namun, dalam skenario dengan integrasi yang kompleks atau terkait dengan evolusi database proses rollback bisa menjadi tugas yang sangat rumit. Penelitian yang saat ini dilakukan membahas tentang bagaimana menangani insiden dengan cepat terotomasi melalui pendekatan yang dilakukan oleh metode GitOps yang merupakan masalah yang ditemukan dalam penelitian sebelumnya.

2.2. Keaslian Penelitian

Tabel 2.1. Matriks literatur review dan posisi penelitian

Analisis Penggunaan Metode Gitops pada Pengembangan Perangkat Lunak Berbasis Devops

No	Judul	Peneliti, Media Publikasi, dan Tahun	Tujuan Penelitian	Kesimpulan	Saran atau Kelemahan	Perbandingan
1	Designing Software Architecture to Support Continuous Delivery and DevOps: A Systematic Literature Review	Robin Bolscher dan Maya Daneva (ICSOFT, 2019)	Memberikan pemahaman tentang state-of-the-art tentang topik pendekatan arsitektur perangkat lunak yang mendukung implementasi Continuous Delivery (CD) dan DevOps, yang informatif bagi peneliti dan praktisi.	Arsitektur micro-services adalah model arsitektur yang paling dominan mendukung implementasi Continuous Delivery (CD) dan DevOps	Penelitian tentang bagaimana cara migrasi dari dari model monolitic ke microservice	Penerapan metode Gitops pada model aplikasi monolitic, menggunakan basis pengujian mandiri menggunakan ArgoCD, docker dan Kubernetes.
2	Adopting Continuous Integeration and Continuous Delivery for Small Teams	Abbass, dkk. (ICCCEEE, 2019)	Jurnal ini berisi tentang detail tantangan dan strategi menghadapi tantangan ketika hendak mengadopsi CD pada tim kecil.	Menerapkan CI / CD dalam tim kecil bisa jadi sulit terkait dengan proses testing, tetapi dengan konsep micro-pipeline dapat mengurangi konsumsi waktu secara radikal.	Berfokus hanya pada proses testing.	Penerapan metode Gitops untuk untuk memudahkan proses adopsi DevOps dengan memberikan contoh penerapan pada lingkungan sebenarnya.

Tabel 2.1. Lanjutan

No	Judul	Peneliti, Media Publikasi, dan Tahun	Tujuan Penelitian	Kesimpulan	Saran atau Kelemahan	Perbandingan
3	Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices	Shahin, dkk. (IEEE Access, 2017)	Meninjau secara sistematis *state of the art* praktik berkelanjutan untuk mengklasifikasikan pendekatan dan alat, mengidentifikasi tantangan dan praktik serta mengidentifikasi celah untuk penelitian di masa depan.	Tiga puluh pendekatan dan *tools* terkait telah diidentifikasi oleh penelitian ini, yang mampu memfasilitasi penerapan praktik berkelanjutan	Riset dalam mendapatkan pemahaman yang mendalam tentang bagaimana sistem perangkat lunak harus (kembali) dirancang untuk mendukung praktik berkelanjutan; dan mengatasi kurangnya pengetahuan dan alat untuk proses rekayasa dalam merancang dan menjalankan pipeline deployment yang aman.	Menjawab permasalahan tentang bagaimana penerapan pipeline deployment yang aman dengan menggunakan metode Gitops.
4	Problems And Solutions of Continuous Deployment: A Systematic Review	Proulx, dkk. (arXiv, 2018)	Mengidentifikasi tantangan dan solusi yang terkait dengan Penerapan Berkelanjutan (CD)	Artikel ini berfungsi sebagai referensi bagi praktisi yang ingin menemukan cara mengatasi tantangan tertentu saat menerapkan praktik penerapan berkelanjutan (CD)	Kategori dengan tantangan terbanyak yang tidak dapat dikaitkan dengan solusi adalah Infrastruktur, Arsitektur Aplikasi, dan Pengujian.	Penelitian ini menggunakan metode CD menggunakan pendekatan penggunaan CI server, belum menggunakan GitOps yang tidak lagi menggunakan CI server untuk menjalankan pipeline.

Tabel 2.1. Lanjutan

No	Judul	Peneliti, Media Publikasi, dan Tahun	Tujuan Penelitian	Kesimpulan	Saran atau Kelemahan	Perbandingan
5	Continuous and Integrated Software Development using DevOps	Agarwal, dkk. (IEEE, 2019)	Memberikan penyelidikan empiris pada beberapa *tools* dan tantangan yang dihadapi selama penerapan praktik berkelanjutan dalam Pengembangan perangkat lunak menggunakan DevOps.	Beberapa faktor yang mencegah organisasi untuk mengadopsi pendekatan DevOps adalah: kurangnya tes yang sepenuhnya otomatis, metodologi *rollback* yang buruk, pemeriksaan kualitas yang dijalankan secara manual.	Solusi yang ditawarkan hanya menyangkut masalah testing, tidak membahas tentang masalah proses rollback yang buruk	Metode GitOps menawarkan mekanisme *rollback* yang sangat baik, metode ini menjawab permasalahan yang ada yang dibahas pada paper ini terkait proses *rollback* yang buruk
6	A Survey of DevOps Concepts and Challenges	Leite, dkk. (ACM Computing Surveys, 2019)	Menyelidiki dan membahas tantangan DevOps dari perspektif *engineer*, manajer, dan peneliti.	DevOps telah mengaburkan batas antara pengembang dan operator sesuai dengan manifestonya.	Bagaimana organisasi menangani "peran DevOps" adalah topik yang menjanjikan untuk penelitian di masa mendatang	Akses security tidak dibahas di penelitian ini, sedangkan metode GitOps menawarkan keamanan yang lebih karena menggunakan single-source-of-truth (Git)

2.3. Landasan Teori

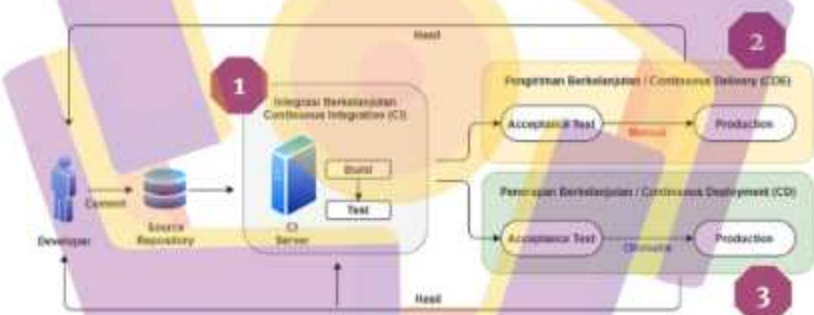
Terdapat beberapa landasan teori yang dibutuhkan dalam penelitian ini, mulai dari landasan teori mengenai proses pengembangan perangkat lunak modern, DevOps, siklus DevOps, *deployment model* serta model GitOps.

2.3.1. DevOps

DevOps adalah tentang menghilangkan pembatas antara dua tim yang biasanya bertentangan dalam struktur IT, yaitu tim pengembangan (Dev) dan tim pengoperasian (Ops) (Bolscher & Daneva, 2019). Salah satu faktor yang berkontribusi terhadap pertentangan ini adalah persaingan antara tim Dev dan Ops. Sering kali, Dev bertanggung jawab untuk merespon perubahan dalam pasar, menerapkan fitur dan mengubah produksi secepat mungkin. Ops bertanggung jawab untuk menyediakan TI dalam bentuk layanan yang stabil, andal, dan aman. Sangat menyulitkan atau bahkan tidak mungkin bagi siapa pun untuk melakukan perubahan pada lingkungan produksi secara cepat yang dapat membahayakan sistem produksi. Dari sini terlihat bahwa Dev dan Ops memiliki tujuan dan insentif yang sangat bertentangan (Kim et al., 2016).

Inti dari devops dimulai dengan orang-orang yang bekerja tidak hanya sebagai grup tetapi juga sebagai tim dengan keinginan untuk membangun rasa saling pengertian (Davis & Daniels, 2016). Konsep DevOps ini memungkinkan peran kedua tim tersebut untuk saling berkoordinasi dan berkolaborasi untuk menghilangkan proses-proses manual yang syarat dengan kesalahan diganti menjadi proses yang terintegrasi, proses otomasi dengan bantuan perangkat-

perangkat DevOps (Verona, 2016). Tujuan utamanya adalah menghasilkan produk dengan cepat, lebih baik dan lebih handal. Pada praktiknya, DevOps menggunakan beberapa metode diantaranya adalah proses integrasi secara berkelanjutan atau yang biasa disebut dengan Continuous Integration (CI), proses pengiriman atau umpan balik secara berkelanjutan atau Continuous Delivery (CDE), dan proses penerapan secara berkelanjutan atau Continuous Deployment (CD). Berkelanjutan diartikan sebagai terus terulangnya pola yang sama secara konsisten dan sistematis, mulai dari proses membangun atau membuat, menerapkan, mengoperasikan, hingga proses kontrol kualitas. Gambaran tentang konsep dasar DevOps bisa dilihat pada gambar 2.1.



Gambar 2.1. Konsep dasar DevOps

Tiga submetode tersebut dapat menjadi solusi dan sangat membantu perancang atau pembuat perangkat lunak untuk mempercepat pengembangan produk atau perangkat lunak mereka dan merilis fitur secara cepat tanpa mengorbankan kualitas dari produknya (Abbass et al., 2019). Dari ketiga metode berkelanjutan ini diharapkan dapat memberikan beberapa manfaat diantaranya

adalah mendapatkan umpan balik yang cepat dari proses pengembangan perangkat lunak dan juga dari pengguna, membantu dalam proses rilis agar lebih sering yang mengarah pada peningkatan kepuasan pelanggan atau pengguna dan kualitas produk perangkat lunak tersebut, melalui proses Continuous Deployment (CD), hubungan antara tim pengembangan atau development dan tim operasi atau operational atau support menjadi kuat dan tugas yang sifatnya manual bisa dihilangkan (Shahin, Ali Babar, et al., 2017).

2.3.2. CI/CDE/CD

Continuous Integration (CI) atau Integrasi Berkelanjutan adalah metode pengembangan perangkat lunak yang umum digunakan pada praktik DevOps. Praktik ini mengacu pada praktik dari DevOps di mana kita sebagai developer bisa mengintegrasikan kode ke dalam repositori kode seperti GitHub dan menjalankan pengujian secara cepat dan otomatis. CI juga merupakan sebuah metode untuk menemukan masalah perangkat lunak sedini mungkin dalam siklus pengembangan dan memastikan semua bagian platform keseluruhan dapat terintegrasi satu sama lain dengan benar (Verona et al., 2016). Tersedia banyak sekali tools DevOps yang populer yang bisa digunakan untuk mempraktikkan CI, diantaranya Travis CI, CircleCI, Jenkins dan AppVeyor (Gallaba, 2019).

Tujuan Continuous Integration (CI) adalah mengumpulkan kode dari anggota tim pengembangan kedalam satu repositori kode sumber atau *revision control system* terpusat dimana kode dapat digabungkan dan perubahan dapat dikontrol versinya. Dalam repositori kode sumber, versi produksi dan versi

development perangkat lunak dapat disimpan secara terpisah, sehingga proses pengembangan perangkat lunak tidak mengganggu kode sumber produksi (Swartout, 2012). *Revision control system* ini juga sering dianggap sebagai jantung dari sebuah lingkungan pengembangan (Verona et al., 2016).

Langkah kedua dalam integrasi berkelanjutan adalah membangun kode sumber menjadi perangkat lunak yang berfungsi dan kemudian menjalankan tes otomatis pada kode tersebut. Dikarenakan proses integrasi dilakukan seketika dan otomatis, konflik dalam kode biasanya akan dengan cepat ditemukan dan diselesaikan atau dikembalikan (Shahin, Ali Babar, et al., 2017).

Meskipun mengotomatiskan pengujian termasuk bagian penting dari CI (Marijan et al., 2018), itu saja tidak cukup. Perlu mengubah budaya tim atau alur kerja pengembang untuk memastikan bahwa pengembang tidak bekerja berhari-hari pada suatu fitur tanpa menggabungkan (merge) perubahan kembali ke cabang utama serta harus menerapkan budaya sistematis dan cepat (Zampetti et al., 2019) (Abbass et al., 2019). Baik menggunakan pengembangan berbasis trunk atau cabang (branch) (Katal et al., 2019), penting bagi pengembang untuk mengintegrasikan perubahan secepat mungkin ke repositori utama. Jika membiarkan kode berada di cabang atau di komputer pengembang terlalu lama, maka akan berisiko mengalami terlalu banyak konflik saat memutuskan untuk menggabungkan kembali ke repositori utama. Dengan mengintegrasikan lebih awal, dapat mengurangi cakupan perubahan yang membuatnya menjadi lebih mudah diperbaiki jika terjadi konflik (Mishra & Otaiwi, 2020). Keuntungan lainnya

adalah mempermudah berbagi pengetahuan di antara pengembang karena mereka akan mendapatkan perubahan yang lebih mudah dipahami.

CDE memfokuskan organisasi untuk membangun proses rilis perangkat lunak otomatis yang efisien. Inti dari proses rilis adalah proses umpan balik berulang. Otomasi adalah proses kunci dari CDE. CDE juga bisa diartikan sebagai sebuah metode pengiriman sebuah perangkat lunak yang dapat bekerja dengan baik dan yang telah diuji dalam ukuran potoongan-potongan kecil sehingga menjadi platform produksi yang utuh (Verona et al., 2016). Diantara kelemahan proses manual adalah tidak benar-benar dapat diulang dan tidak dapat diandalkan kecuali jika sudah dalam bentuk kode dan dapat dijalankan secara otomatis sesuai permintaan (Proulx et al., 2018). Tugas otomatis dapat disusun bersama untuk membuat tingkat otomatisasi lebih lanjut. Proses umpan balik dari CDE adalah pemeriksaan ulang yang konsisten terhadap kualitas yang dikirimkan (Zahedi et al., 2018).

Continuous DElivery (CDE) adalah kemampuan untuk mendapatkan semua jenis perubahan, termasuk fitur baru, perubahan konfigurasi, perbaikan bug, dan eksperimen dalam lingkungan produksi, atau ke dalam tangan pengguna, dengan aman dan cepat dengan cara yang berkelanjutan (Leszko, 2020). CI dianggap sebagai bagian wajib dari CDE, dengan demikian CDE terdiri dari CI dan langkah-langkah tambahan dari CDE. Dalam CDE pengiriman versi perangkat lunak yang diinginkan dilakukan secara terus menerus dan secara otomatis diinstal ke platform yang diinginkan, misalnya ke server web (Shahin, Babar, et al., 2017). Biasanya platform ini adalah lingkungan yang mirip dengan lingkungan produksi sehingga

dapat diverifikasi bahwa perangkat lunak tersebut dapat diterapkan ke lingkungan produksi kapan saja. Biasanya CDE dilakukan ke sistem pengujian setiap kali terjadi perubahan kode sumber.

Dalam CDE, penerapan ke produksi juga dapat diotomatiskan, tetapi memerlukan interaksi manual di beberapa status, itulah yang membedakan CDE dan CD. Manfaat memiliki pipa CDE ke beberapa lingkungan adalah bahwa pemasangannya telah dilakukan dijalankan dengan cara yang sama beberapa kali dan sudah teruji dan berfungsi dengan benar.

Integrasi berkelanjutan (CI) dan Pengiriman Berkelanjutan (CDE) adalah basis untuk Penerapan Berkelanjutan. Penerapan berkelanjutan adalah kombinasi dari fase-fase tersebut dan yang paling canggih versi pipeline pengiriman otomatis. Dalam CD, produk diterapkan sepenuhnya secara otomatis sampai ke lingkungan produksi tanpa interaksi manusia dalam perjalanan [Shahin et al., 2017a]. Semua perubahan yang dilakukan pada kode sumber secara otomatis digunakan. Di banyak domain, pengiriman otomatis ke lingkungan produksi tidak dimungkinkan karena kebijaksanaan perusahaan. Salah satu cara untuk melihat perbedaannya adalah dengan mempertimbangkan penggunakan CDE, penerapan berbasis tarik (*pull-based deployment*) yang dilakukan secara manual untuk pemasangan dilingkungan produksi, CD adalah berbasis push-based karena perubahan secara otomatis didorong ke dalam lingkungan produksi.

2.3.3. Model Penerapan

Software Development Life Cycle (SDLC) mendefinisikan langkah-langkah tertentu yang berbeda satu sama lain yang memiliki tujuan spesifik dalam proses pembuatan perangkat lunak (Kneuper, 2018). Setiap metodologi pengembangan memiliki fase yang mencakup pengiriman atau penerapan perangkat lunak. Dalam *Agile Development*, fase ini akan terjadi relatif sering dibandingkan dengan metode *Waterfall* (Kneuper, 2018). Pada model *waterfall* semua fase termasuk fase pengiriman dan penerapan dijalankan secara sequence, dimana setelah fase tertentu tercapai, tidak ada cara untuk kembali ke fase sebelumnya (Kneuper, 2018). Pada tahun 2010 dan sebelumnya, tim pengembangan dan tim operasi berdiri sendiri-sendiri dan seringkali menimbulkan gesekan saat proses perpindahan dari fase pengembangan ke operasi, untuk menutup celah tersebut metode DevOps diperkenalkan dengan tujuan untuk menyatukan pengembangan dan operasi atau setidaknya lebih dekat (Rossberg, 2019).

Seperti yang disebutkan sebelumnya, hal ini telah menjadi suatu hal yang rumit di masa lalu dan DevOps diperkenalkan sebagai praktik untuk menjembatani kesenjangan tersebut. Secara historis, penerapan perangkat lunak merupakan pekerjaan yang cukup rumit. Menerapkan paket sering membutuhkan pengetahuan ahli tentang perangkat lunak, maintenance window, mencari tahu dependensi dari perangkat lunak terkait, memperbarui dokumentasi dan bahkan dibutuhkan pengetahuan tentang mesin server yang akan digunakan untuk instalasi perangkat lunak baru. Setelah ini perangkat lunak dapat digunakan secara manual, misalkan

dengan masuk ke server melalui ssh, menyalin paket ke folder yang benar, memperbarui konfigurasi terkait dan merestart layanan. Pekerjaan semacam ini sangat imperatif di mana penerapan didefinisikan selangkah demi selangkah, da setiap langkah mempunyai potensi kesalahan. Skrip telah digunakan selama beberapa dekade untuk mengotomatiskan bagian dari penerapan tetapi seringkali ini dibuat khusus untuk aplikasi dan tidak tersedia dan dapat diterapkan secara umum.

Perangkat lunak dapat ditulis menggunakan dua paradigma yang berbeda, perangkat lunak juga dapat digunakan dengan dua pendekatan berbeda: imperatif dan deklaratif (Attardi et al., 2018). Pendekatan ini berbeda satu sama lain tentang bagaimana penerapannya dilakukan dari sudut pandang pengguna. Dalam *imperative deployment*, setiap langkah penerapan dilakukan dengan menjalankan perintah satu persatu yang sudah ditentukan, sedangkan dalam *declarative deployment* hanya perlu mendefinisikan keadaan yang diinginkan (Attardi et al., 2018). Pada model deklaratif, satu pernyataan bisa menggambarkan apa yang seharusnya menjadi 10 atau lebih baris kode pada model imperative (Kim et al., 2016). Untuk alasan sebelumnya, *imperative deployment* memerlukan seseorang atau sesuatu misalkan Programmer atau pipeline (CI/CD) untuk dieksekusi perintahnya satu per satu ke setiap server secara terpisah untuk mencapai keadaan yang diinginkan. *declarative deployment* hanya perlu menjelaskan dalam apa yang diinginkan dalam sebuah dokumen, dokumen itu kemudian diteruskan pada sebuah *tool* yang akan mengeksekusi dan mengelola hingga keadaan yang diinginkan tercapai berdasarkan model yang didefiniskan pada dokumen. Penerapan semacam

ini setidaknya harus memiliki manfaat berikut: reproduktifitas ketika setiap penerapan terpisah menghasilkan keadaan yang identik, *testability* karena deklarasi dapat disisipkan dan skenario tes (Wurster et al., 2018) dan *verifiability* karena berkas deklarasi berlaku juga sebagai dokumentasi.

Salah satu implikasi mendasar dari penerapan imperatif adalah bahwa ia tidak memiliki model. Ketiadaan model juga berarti tidak bisa me-*maintain state* karena tidak ada *state* untuk memulai. Hal ini membuat penerapan imperatif atau alur kerja pada dasarnya merupakan proses langkah-langkah yang dieksekusi dalam urutan tertentu. Ini tidak selalu menjadi hal yang buruk karena model ini memberikan banyak kendali proses tetapi pada saat yang sama dapat menyebabkan situasi yang rumit untuk dipertahankan ketika jumlah komponen bertambah besar (Breitenbücher et al., n.d.).

Sebuah analogi untuk menggambarkan tentang *imperative deployment* adalah sebuah *checklist*. *Checklist* mendefiniskan secara eksplisit apa yang harus dilakukan dan dalam urutan seperti apa dan semua terserah penulis untuk membuat item apa saja yang akan dimasukkan ke dalam *checklist* agar sesuai dengan keinginan dan kebutuhan.

Untuk mengotomatiskan proses penerapan aplikasi, banyak teknologi yang mendukung model penerapan deklaratif (*Declarative deployment)*, yang mendeskripsikan struktur aplikasi yang akan digunakan termasuk semua komponen dan hubungan mereka (Harzenetter et al., 2019). Model deklaratif menyatakan keadaan yang diinginkan di mana aplikasi atau bagiannya ditransfer. Sebaliknya,

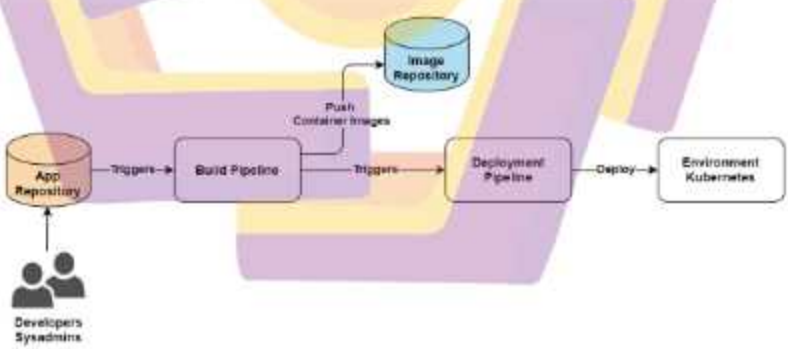
model imperatif menggambarkan langkah-langkah penerapan secara prosedural (Wurster et al., 2020).

Declarative deployment adalah praktik yang bergantung pada *tools* karena menggunakan template untuk mendeskripsikan state penerapan yang diinginkan. Template biasanya ditulis dalam format yaml, atau json. File teks ini biasanya disebut "*as code*" karena mereka dapat disimpan dalam repositori kendali sumber, dibagikan, diedit, atau diimpor (Asthana et al., 2018). File template dibaca oleh alat yang ditulis dalam bahasa lain. Proses ini bisa disebut dengan *read, diff, plan* and *execution* yang dalam praktiknya mengikuti prinsip CRUD (Tao & Etchevers, 2018). Fase *read* membaca file template dan mengubahnya menjadi model, model ditentukan oleh *state* yang diinginkan dari sistem, fase *diff* memeriksa keadaan saat ini dari lingkungan target dan menghasilkan model perbedaan dari kedua sistem, fase *plan* menerjemahkan model *diff* menjadi operasi atau *state* tunggal dan akhirnya fase *execution* menjalankan masing-masing operasi ini (Tao & Etchevers, 2018).

Declarative deployment menyederhanakan proses penulisan dan menerbitkan layanan ke beberapa mesin orkestrasi (Asthana et al., 2018). Dibandingkan dengan *imperative deployment*, *declarative deployment* akan terlihat sangat berbeda. Perintah penerapan kita terdiri dari *deployment file* dan perintah yang mengeksekusi *tools* untuk menerapkan. *Deployment file* akan berisi informasi paket apa yang akan digunakan dan di mana akan diterapkan. Pendekatan deklaratif biasanya memiliki dukungan yang lebih lemah untuk pola konfigurasi lanjutan sedangkan pendekatan imperatif mudah menjadi tidak elastis, misalnya,

penggunaan nama dengan *hard code* dalam pemrograman imperatif (Hakimzadeh & Dowling, 2019).

Pada *Declarative deployment* terdapat dua pendekatan cara melakukan deployment yaitu *Push-Based Deployment* dan *Pull-Based Deployments*, Push-Based Deployment atau strategi penerapan berbasis *push* diimplementasikan oleh *tools* CI/CD populer seperti Jenkins, CircleCI, atau Travis CI. Kode sumber aplikasi berada di dalam suatu repositori aplikasi bersama dengan YAML Kubernetes yang diperlukan untuk menerapkan aplikasi. Setiap kali kode aplikasi diperbarui, *pipeline build* akan dipicu, kemudian akan menjalankan proses pembuatan *container image*, kemudian *container image* akan diunggah ke repositori *container image* dan terakhir repositori konfigurasi sistem akan diperbarui dengan deskriptor penerapan baru, gambaran proses dari *Push-based Deployment* terlihat pada gambar 2.2.



Gambar 2.2. Push-based Deployment

Perubahan pada repositori konfigurasi memicu pipeline penerapan. Pipeline ini bertanggung jawab untuk menerapkan semua manifes yang ada di repositori konfigurasi ke lingkungan infrastruktur. Dengan pendekatan ini, diperlukan

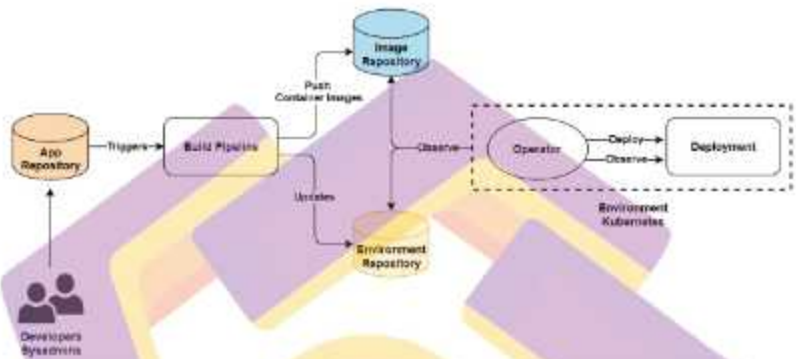
kredensial untuk mengakses dan melakukan perubahan pada lingkungan penerapan. Untuk kebutuhan ini pipeline perlu diberikan mode akses administrator.

Dalam beberapa kasus penggunaan, penerapan berbasis Push tidak dapat dihindari saat menjalankan penyediaan infrastruktur cloud secara otomatis. Dalam kasus seperti itu, sangat disarankan untuk menggunakan sistem otorisasi yang sangat terperinci yang dapat dikonfigurasi dari penyedia cloud untuk izin penerapan yang lebih ketat.

Hal penting lainnya yang perlu diingat saat menggunakan pendekatan ini adalah *pipeline* penerapan hanya dipicu saat repositori lingkungan berubah. Biasanya dikarenakan pengembang melakukan *commit* baru pada kode sumber yang ada di Git. Itu tidak dapat secara otomatis melihat adanya penyimpangan dan keadaan yang diinginkannya. Artinya, perlu adanya beberapa cara pemantauan, sehingga seseorang dapat melakukan intervensi/mengeksekusi sesuatu jika kondisi aktual sistem tidak cocok dengan apa yang dijelaskan dalam repositori.

Pull-Based Deployments atau strategi penerapan berbasis *Pull* menggunakan konsep yang sama dengan varian berbasis push tetapi berbeda dalam cara kerja pipeline penerapan. Pipeline CI/CD tradisional dipicu oleh kejadian eksternal, misalnya saat kode baru didorong ke repositori aplikasi. Dengan pendekatan penerapan berbasis Pull, operator diperkenalkan. Operator mengambil alih peran pipeline dengan terus membandingkan status yang diinginkan di repositori dengan status sebenarnya di infrastruktur yang sudah diterapkan. Setiap kali ada perbedaan yang terlihat, operator memperbarui infrastruktur agar sesuai dengan repositori. Selain itu, *image registry* dapat dipantau untuk menemukan

image versi terbaru untuk digunakan. Rangkaian proses Pull-based Deployment dengan menggunakan operator sebagai pengganti fungsi CI/CD pipeline tergambarkan pada gambar 2.3.



Gambar 2.3. Pull-based Deployment

Sama seperti penerapan berbasis push, varian ini memperbarui lingkungan setiap kali repositori lingkungan berubah. Namun, dengan operator, perubahan dilakukan berdasarkan hasil pengecekan operator. Setiap kali infrastruktur yang diterapkan berubah dengan cara apa pun yang tidak dijelaskan dalam *environment repository*, perubahan ini akan dikembalikan. Ini memastikan bahwa semua perubahan dapat dilacak di log Git, dengan tidak mungkinnya membuat perubahan langsung ke cluster.

Perubahan arah ini memecahkan masalah penerapan berbasis push, di mana lingkungan hanya diperbarui saat repositori lingkungan diperbarui. Namun, ini tidak berarti kita tidak butuh mekanisme monitoring. Sebagian besar operator mendukung pengiriman email atau pemberitahuan Slack jika proses penerapan gagal karena alasan apa pun, misalnya kegagalan dalam menarik *container image*.

Selain itu, kita mungkin harus menyiapkan pemantauan untuk operator itu sendiri, karena tidak ada lagi proses penerapan otomatis tanpanya.

Untuk tujuan keamanan, operator harus selalu berada di lingkungan atau kluster yang sama dengan aplikasi yang akan diterapkan. Seperti yang terjadi pada pendekatan berbasis push, di mana kredensial untuk melakukan penerapan diketahui oleh pipeline CI/CD. Dalam *Pull-Based Deployments* tidak ada kredensial yang perlu diketahui oleh layanan eksternal. Mekanisme otorisasi dari platform penerapan yang digunakan dapat digunakan untuk membatasi izin dalam melakukan penerapan. Ini berdampak besar dalam hal keamanan. Saat menggunakan Kubernetes, konfigurasi RBAC dan *service account* dapat digunakan untuk tujuan keamanan.

2.3.4. GitOps

Istilah GitOps diperkenalkan oleh sebuah perusahaan bernama Weaveworks pada Agustus 2017 dan istilahnya berasal dari praktik DevOps yang sudah ada (Limoncelli, 2018). GitOps adalah cara mengimplementasikan *Continuous Deployment* yang menggunakan Git sebagai sumber tunggal kebenaran untuk infrastruktur dan aplikasi (Yuan et al., 2021). GitOps merupakan pendekatan deklaratif untuk pengelolaan infrastruktur dan deployment (Boronin, 2020). Berfokus pada pengalaman para pengembang saat mengoperasikan infrastruktur, dengan menggunakan alat yang sudah dikenal oleh pengembang, dalam hal ini adalah Git dan *Continuous Deployment tools*. Ide inti dari GitOps adalah memiliki repositori Git yang selalu berisi deskripsi deklaratif dari infrastruktur yang

diinginkan dan proses otomatis untuk membuat lingkungan produksi sesuai dengan status yang didefinisikan dalam repositori. Jika kita ingin menerapkan aplikasi baru atau memperbarui yang sudah ada, kita hanya perlu memperbarui repositori, proses otomatis akan menangani sisanya. Ini seperti memiliki *cruise control* untuk mengelola aplikasi kita dalam lingkungan produksi.

DevOps digambarkan sebagai budaya dan filosofi baru dalam proses rekayasa perangkat lunak yang memanfaatkan tim lintas fungsi (development, operations, security, dan QA) untuk membuat, menguji, dan merilis perangkat lunak dengan lebih cepat dan lebih andal melalui otomatisasi (Macarthy & Bass, 2020) sedangkan GitOps berfungsi mengelola proyek baik untuk pengembangan dan operasi. Mereka melakukan ini dengan mengalihkan fokus dari CI/CD (yang merupakan jantung DevOps) di *pipeline* dan menyatakan bahwa semuanya harus ada dalam kontrol versi seperti git. GitOps berpendapat; bahwa Git atau *Source Control* adalah adalah satu-satunya sumber kebenaran tentang sebuah sistem. Di GitOps, seluruh status penerapan didefinisikan dalam repositori Git dan semua perubahan atau manajemen harus melaluinya (Paez, 2018). Berfikir tentang kontrol sumber sebagai satu-satunya sumber kebenaran adalah penting bagi GitOps. Hal ini memungkinkan GitOps untuk mengelola dan mengamati segala sesuatu dari satu lokasi terpusat. Melalui proses review *commit,* Git bertindak sebagai broker yang dapat memberitahu semua orang tentang isi dari proyek mereka. Begitu pula jika ingin melakukan perubahan pada sistem, perubahan hanya perlu dilakukan di satu lokasi yang kemudian akan disebarkan dan akan tersedia untuk semua orang.

Menempatkan semuanya dalam source control memiliki beberapa implikasi yang membuat GitOps sangat berguna dan menarik. Mendorong penggunaan *Pull Request (PR)* sebagai bagian dari proses review kode. Setelah pengembang menyelesaikan fitur, sang pengembang mengajukan PR dari branch miliknya untuk memulai proses review agar dapat menggabungkan perubahan ke branch master. Pada tahap ini terbuka kemungkinan untuk diskusi tentang perubahan terkait, menerima atau menolak. PR juga bisa diperbarui jika diperlukan berdasarkan diskusi. Pengembang lain atau pemilik proyek adalah orang yang membuat keputusan akhir jika permintaan tersebut disetujui untuk digabungkan. Ini membuat *Version Control System (VCS)* menjadi sumber kebenaran tunggal, satu-satunya, dan semua operasi yang terjadi pada kode dan lingkungan harus melewatinya.

Setiap teknologi *Continuous Deployment* berjanji untuk membuat penerapan lebih cepat dan memungkinkan kita melakukan penerapan sesering mungkin dengan tingkat akurasi yang meningkat dikarenakan dengan bantuan tools dan otomasi. Yang menarik dari GitOps adalah tidak perlunya melakukan penggantian *tools* yang ada untuk menerapkan aplikasi Anda. Semuanya perubahan hanya dilakukan di sistem kontrol versi (Git) yang besar kemungkinan sudah digunakan saat ini untuk mengembangkan aplikasi.

Dengan GitOps kita dimungkinkan untuk memiliki riwayat lengkap tentang bagaimana sistem berubah dari waktu ke waktu. Ini membuat pemulihan kesalahan (*rollback*) semudah menjalankan *git revert* dan tinggal memonitor sistem kita dipulihkan, sesuai dengan standar operasi Git (Arndt & Martin, 2019).

GitOps memungkinkan kita untuk mengelola penerapan sepenuhnya dari dalam lingkungan kita. Untuk itu, lingkungan sistem hanya membutuhkan akses ke repositori dan *image registry*. Dengan demikian kita tidak perlu memberi akses langsung kepada pengembang ke lingkungan produksi. Dengan GitOps, setiap perubahan pada lingkungan apa pun harus dilakukan melalui repositori, yang memungkinkan kita selalu dapat memeriksa cabang master dan mendapatkan deskripsi lengkap tentang apa yang diterapkan, serta riwayat lengkap dari setiap perubahan yang pernah dilakukan pada sistem. Kita bisa mendapatkan jejak audit dari setiap perubahan dalam sistem secara gratis.

Menggunakan Git untuk menyimpan deskripsi lengkap dari infrastruktur yang kita terapkan memungkinkan semua orang di tim kita untuk memeriksa perubahannya dari waktu ke waktu. Dengan pesan komit yang dilakukan secara jelas dan benar, setiap orang dapat mereproduksi proses perubahan infrastruktur dan juga dengan mudah menemukan contoh proses bagaimana membuat sistem baru jika dibutuhkan.

BAB III

METODE PENELITIAN

3.1. Jenis, Sifat, dan Pendekatan Penelitian

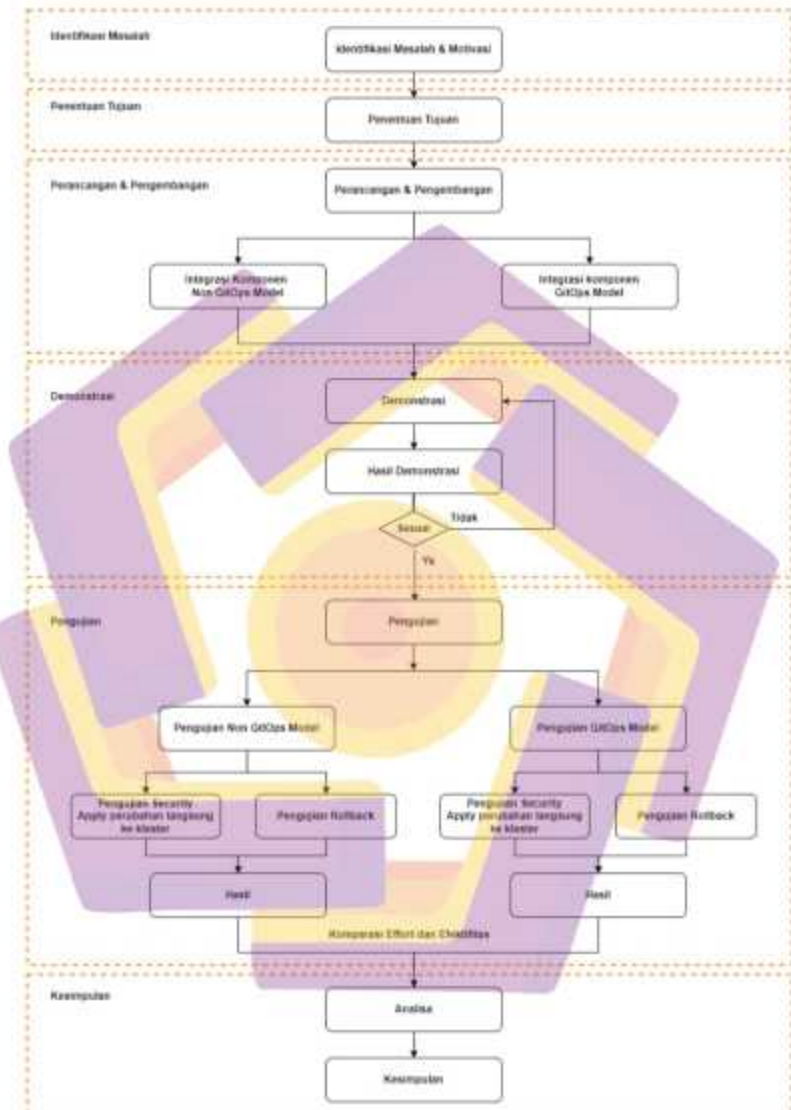
Metodologi penelitian yang digunakan mengacu pada tahapan *Design Sciense Research Method* (DSRM) yang dikemukan oleh Ken Peffers, Tuure Tuunanen, Marcus A. Rothenberger, dan Samir Chatterjee pada jurnal berjudul *"A Design Science Research Methodology for Information Systems Research"*. DSRM merupakan sebuah metodologi yang berorentasikan pada desain sistem informasi, DSRM juga merupakan kerangka prosedur yang digunakan untuk mempermudah penelitian di bidang teknologi informasi yang digunakan sebagai proses pemahaman serta mengulas untuk mengenali dan mengevaluasi hasil penelitian (Peffers et al., 2007).

Design Science Research Methodology (DSRM) terdiri enam proses yaitu *Problem Identification and Motivation* (Identifikasi Masalah dan Motivasi), *Objective of the Solution* (Mendefinisikan Objek dari Solusi Permasalahan), *Design and Development* (Perencanaan dan Pengembangan), *Demonstration* (Demonstrasi), *Evaluation* (Evaluasi) dan *Communication* (Komunikasi). Konsep Metodologi Penelitian untuk DSR (Design Science Research) ini dapat dilihat pada gambar 3.1.

Gambar 3.1. Bagan Alir metodologi DSR (Design Science Research).

3.2. Alur Penelitian

Dalam melakukan penelitan, diperlukan adanya tahapan-tahapan yang diurutkan secara sistematis agar pelaksanaan penelitian dapat berjalan degan baik. Adapun alur penelitian pada penelitian ini dapat dilihat pada gambar 3.2.



Gambar 3.2. Alur penelitian

3.2.1. Identifikasi Masalah dan Motivasi

Pada tahapan ini dilakukan identifikasi terhadap masalah yang dihadapi oleh penerapan model DevOps saat ini, penelitian ini fokus pada masalah keamanan terkait kemampuan seseorang untuk mengakses klaster dan melakukan perubahan pada klaster secara langsung, juga pada masalah tidak efektifnya proses rollback yang ada.

3.2.2. Penentuan Tujuan dari Penelitian

Studi literatur bertujuan guna mendapatkan data-data pendukung dan memperkuat teori-teori, metode kualitatif dengan melakukan pengujian terhadap proses pengolahan data yang dikumpulkan melalui instrumen berdasar pada variabel yang sebelumnya telah ditentukan, metode observasi dengan melakukan pengujian mandiri untuk kebutuhan validasi. Studi literatur juga dilakukan untuk mencari rumusan masalah yang akan dibahas pada penelitian. Tahapan ini digunakan untuk menentukan secara objektif sebuah solusi untuk menentukan tujuan dari penelitian atau bagaimana sistem baru ini dapat mendukung penyelesaian masalah yang sekarang ditangani. Sumber daya yang diperlukan adalah pengetahuan tentang keadaan masalah saat ini. Tahapan ini adalah mencari semua kebutuhan yang di perlukan dalam pembangunan sistem GitOps yang digunakan untuk menggantikan atau melengkapi fungsi dari sistem DevOps yang ada saat ini. Komponen yang digunakan dapat dilihat pada tabel 3.1.

Tabel 3.1. Komponen GitOps

Komponen	GitOps	Website
Source Code Repository	Github	github.com
CI Server	Jenkins	jenkins.io
Operator	ArgoCD	argoproj.github.io
Deployment Platform	Kubernetes	kubernetes.io

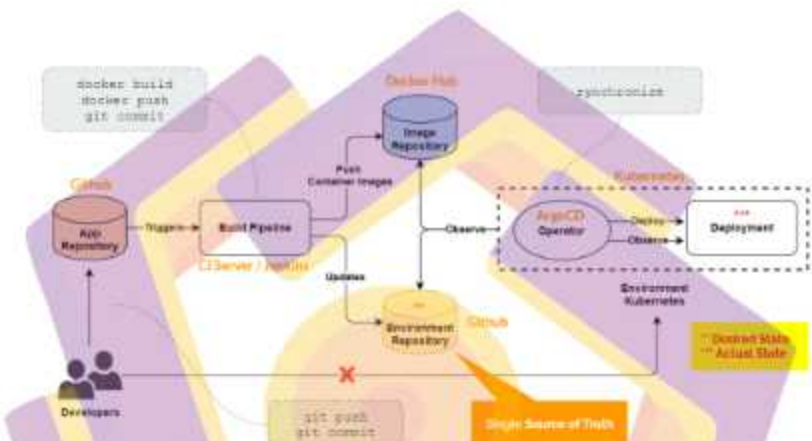
Github, Jenkins, ArgoCD dan Kubernetes merupakan perangkat lunak terbuka yang bisa diunduh dan digunakan secara bebas tanpa perlu membayar.

3.2.3. Perancangan dan pengembangan

Perancangan ditujukan untuk membuat pemodelan sementara terhadap cara dan model yang ada yang dapat membantu sistem yang sedang berjalan saat ini. Perancangan yang dimaksudkan adalah:

a. Perancangan terhadap cara sebuah aplikasi di-deploy ke dalam sebuah platform penerapan menggunakan metode GitOps.
b. Pengembangan terhadap metode yang saat ini digunakan, menggunakan pendekatan metode GitOps dengan mengubah alur kerja dari yang tidak menggunakan Git sebagai single source of truth menjadi menggunakan Git sebagai satu-satunya cara untuk melakukan perubahan pada klater.
c. Penambahan perangkat lunak bantu untuk memudahkan proses implementasi metode GitOps.

Hal pertama yang dilakukan dalam pengembangan design model implementasi GitOps ini adalah membuat rancangan komponen yang terintegrasi sehingga dapat digunakan untuk fase selanjutnya. Gambaran dari rancangan integrasi model penerapan GitOps dapat dilihat pada gambar 3.3.



Gambar 3.3. Rancangan integrasi model penerapan GitOps

Pada gambar diatas terlihat bahwa sistem penerapan GitOps terdiri dari beberapa komponen:

1. *Developers* adalah seseorang yang bertugas untuk mewujudkan desain sebuah produk atau layanan, biasanya berupa software. Developer membuat produk dengan cara menulis baris-baris kode yang rumit dengan menggunakan berbagai bahasa pemrograman dan kemudian mengunggah kodenya ke App repository. Proses mengunggah kode tersebut akan men-*trigger* sebuah *pipeline*.
2. *App Repository* yaitu sebuah repository yang berguna untuk menyimpan kode sumber sebuah aplikasi, dalam penelitian ini peneliti menggunakan github.com.

3. *Pipeline* diartikan sebagai seperangkat proses dan alat otomatis yang memungkinkan tim developer dan tim operation untuk berkolaborasi dalam membangun dan menerapkan kode ke lingkungan produksi. Pipeline berisikan *sets of command* yang akan dijalankan sesuai dengan yang sudah dideklarasikan. Pertama, pipeline pada penelitian ini berfungsi untuk menjalankan *"docker build"* yang bertujuan untuk melakukan pembuatan sebuah docker image berdasarkan berkas dockerfile yang ada. Kedua, pipeline berfungsi untuk menjalankan perintah *"docker push"* yang berfungsi untuk mengunggah *docker image* yang sudah dibuat ke sebuah *image repository*, dalam penelitian ini, peneliti menggunakan hub.docker.com.
4. *Environtmen Repository* adalah sebuah repository yang berguna untuk menyimpan kode dari sebuah environment klaster atau biasa disebut dengan *"desired state"* sebuah klaster. Repositori inilah yang menjadi *single source of truth* atas semua perubahan yang terjadi pada klaster.
5. *Operator* melakukan observasi terhadap *environment repository* dan *image registry*, dan memastikan kondisi yang ada pada *desired state* sama dengan yang ada pada klaster. Pada penelitian ini menggunakan ArgoCD sebagai perangkat bantu penerapan GitOps yang berfungsi sebagai operator.
6. Klaster atau Platform penerapan, adalah lokasi dimana docker image yang berisikan kode sumber aplikasi tersebut akan diterapkan atau di-deploy. Pada penelitian ini peneliti menggunakan Kubernetes.

3.2.4. Demonstrasi

Tahapan demonstrasi dilakukan dengan cara melakukan integrasi antar stack dengan tujuan untuk mendapatkan komposisi yang tepat pada tahap penerapan dan menunjukkan penggunaan artefak untuk memecahkan satu atau lebih dari masalah yang ada. Pada tahap ini akan dilakukan tahap uji coba terhadap sistem yang sudah dirancang. Proses ujicoba diawali dengan menyediakan seperangkat sistem yang terdiri dari Version Control System menggunakan Github sebagai lokasi dimana kode diletakkan. ArgoCD berfungsi sebagai GitOps operator yang di-install di dalam Kubernetes, yang akan melakukan check setiap saat apakah ada perbedaan antara *state* yang ada di klaster dengan state yang ada di *environment repository*. Kubernetes sendiri digunakan sebagai platform pemasangan akhir, *namespace* digunakan untuk membedakan antara lingkungan produksi, *staging* dan *testing*.

Pertama-tama pengembang akan menulis kode yang akan dieksekusi melalui *Version Control System* (Github), setelah itu, perangkat lunak tersebut akan memasuki tahap build. Pada tahap ini, pengembang akan melanjutkan pengembangan kodenya, kode tersebut akan dikembalikan ke *Version Control* untuk pembaruan. kode baru dan kode yang pertama ditulis akan digabungkan (merge), dan akan di-build dalam bentuk *docker image*. Setelah tahap *build* selesai, masuk ke tahap pengujian. Pada tahap ini akan dilakukan berbagai jenis pengujian untuk menguji kelayakan dari perangkat lunak yang sudah dibuat. Setelah tahap pengujian selesai, akan dimulai tahap *deploy*. Pada tahap *deploy*, paket yang sudah di-*build* dalam bentuk *docker image* akan di-*deploy* ke *namespace* produksi

(Deploy to Production). Dalam setiap tahap, jika terdapat *error/bug* dan semacamnya tim pengembang dapat melakukan perbaikan ataupun *rollback* terhadap perangkat lunak tersebut, ini disebut sebagai *Production Feedback*. Pengembang akan memperbarui versi melalui *version control*, dan setiap tahap di atas akan berjalan secara berulang kembali. Siklus tersebut akan berulang sebanyak mungkin hingga diperoleh kode yang dapat dijalankan ke server produksi.

3.2.5. Pengujian & Evaluasi

Pengujian dan evaluasi dilakukan dengan tujuan untuk mengetahui sejauh mana metode baru yang dikembangkan dapat berjalan dengan baik sesuai dengan fungsionalitasnya. Pengujian juga bermanfaat untuk mengukur sejauh mana metode ini dapat menyelesaikan masalah dan sesuai dengan keinginan pengguna. Evaluasi juga berarti mengamati dan mengukur seberapa baik sistem yang baru dalam menyelesaikan masalah ini. Kegiatan ini melibatkan, membandingkan tujuan baik secara actual dari hasil yang diamati dalam penggunaan artefak ketika demonstrasi. Tahap ini membutuhkan pengetahuan tentang ukuran yang relevan dan teknik analisis, tergantung pada sifat dari masalah dan artefak.

Pada saat dilakukan pengujian ditemukan beberapa masalah, baik yang sudah direncanakan untuk diteliti maupun yang tidak direncanakan. Pada hal keamanan ditemukan bahwa selain masalah keamanan terkait seseorang yang dengan mudah dapat melakukan perubahan ke dalam klaster juga ditemukan masalah keamanan yang lain.

Masalah-masalah keamanan yang ditemukan adalah sebagai berikut:

a. Masalah terkait dengan kemungkinan seseorang untuk dapat melakukan perubahan langsung pada klaster. Dikarenakan tidak adanya kontrol pada apa atau siapa yang bisa melakukan perubahan, maka sangat dimungkinkan seseorang mengubah dan menghapus klaster yang artinya resiko sistem menjadi tidak stabil bahkan tidak bisa diakses menjadi sangat besar, dan akan beresiko pada ketersediaan layanan dari aplikasi tersebut. Resiko selanjutnya adalah resiko reputasi terhadap perusahaan dikarenakan layanannya menjadi tidak bisa digunakan oleh pelanggan.
b. Masalah tekait dengan lokasi user dan password, dimana jika tidak menggunakan GitOps, lokasi user dan password berada di luar klaster, sehingga seseorang yang bisa mendapatkan akses terhadap user dan password tersebut, baik legal maupun ilegal, mampu melakukan apapun terhadap klaster. Yang dimaksud dengan user dan password dalam penelitian ini adalah berkas kubeconfig yang digunakan untuk melakukan koneksi ke Kubernetes API menggunakan tool kubectl. Ketika seseorang mempunyai akses ke kubeconfig dan tool kubectl maka seseorang tersebut dapat melakukan hal apapun terkait klaster kubernetes dan deployment yang ada di atasnya.
c. Masalah terkait kemungkinan terjadinya Configuration Drift. Dikarenakan perubahan dapat dilakukan secara langsung, maka kemungkinan apa yang ada di klaster yang disebut sebagai actual state bisa jadi berbeda dengan apa yang tercatat dalam manifest yang ada di repository. Hal ini mengakibatkan configuration drift yang sangat berisiko pada ketersediaan sistem. Sebagai

contoh, jika nama klaster atau deployment yang tertulis dalam repository berbeda dengan yang ada di dalam klaster, kemungkinan seseorang menghapus klaster tersebut sangatlah besar, karena seseorang tersebut mengira bahwa klaster dan deployment tersebut tidak ada.

d. Masalah terkait dengan kemampuan sebuah sistem diaudit, dalam hal ini perubahan yang terjadi di klaster maupun deployment apakah bisa dilakukan audit atau tidak, misalkan dibutuhkan dalam mencari rootcause saat terjadinya problem, atau ketika pihak auditor baik internal maupun eksternal mengharapkan control dan event terkait perubahan yang terjadi. Dalam penelitian ini ditemukan bahwa dengan adanya konsep single source of truth, maka semua perubahan tercatat secara sistematis dalam log Git, sehingga mudah untuk dilakukan audit, mencari rootcauce, dan juga menjadi mudah untuk melakukan pelacakan terhadap perubahan yang terjadi pada klaster dan deployment. Catatan perubahan yang dimaksud digambarkan pada gambar 3.4.

Gambar 3.4. Catatan Perubahan pada Git.

Pada pengujian terkait rollback, masalah yang direncanakan untuk diteliti terkait dengan tidak efektifnya proses rollback, dalam hal waktu yang dibutuhkan dan kemudahan dalam melakukan rollback, dapat diselesaikan menggunakan metode GitOps dengan bantuan ArgoCD sebagai operator. Dimana proses rollback menjadi lebih mudah dikarenakan hanya dibutuhkan beberapa klik pada GUI dashboard ArgoCD, serta proses rollback menjadi lebih cepat hingga lebih dari 30 kali lipat, dari yang semula membutuhkan waktu 30 detik menjadi kurang dari datu detik. Hal ini dimungkinkan karena proses rollback tanpa GitOps dilakukan dengan cara melakukan langkah-langkah deployment dari awal, yang berbeda hanya proses triggernya saja, yang tadinya dari Git push/commit menjadi Git revert. Sedangkan pada GitOps tidak perlu melakukan langkah-langkah deployment dari awal, hanya perlu melakukan pengecekan ke environment repository, dan kemudian rollback dieksekusi.

Pada penelitian ini juga ditemukan masalah terkait proses monitoring dan juga pengechekan bagian mana yang berubah pada repository yang bisa disolusikan dengan penggunaan ArgoCD. Pada ArgoCD kita dapat melakukan monitoring terhadap kondisi klaster, deployment bahkan hingga ke bagian detailnya secara visual. Perubahan yang terjadi juga bisa dideteksi dengan mudah oleh ArgoCD dimana ArgoCD mempunya fitur diff yang bisa menampilkan bagian kode yang mana perubahan terjadi, bagian kode mana yang berbeda antara repository dan kondisi aktual pada klaster. Fitur diff pada ArgoCD terlihat pada gambar 3.5.

Gambar 3.5. Fitur diff pada ArgoCD.

3.2.6. Komunikasi

Setelah penelitian selesai, proses yang dilakukan adalah mendokumentasikan pengetahuan yang dikumpulkan kemudian dibahas dalam komunitas akademik dengan cara mempublikasikannya dalam bentuk artikel atau jurnal.

Hasil penelitian ini sudah dipresentasikan pada International Conference on Information and Communications Technology 2021 (4th ICOIACT 2021) yang dilakukan pada 30-31 August 2021, dengan judul *Analysis on the Use of Declarative and Pull-based Deployment Models on GitOps Using Argo CD*. Publikasinya bisa diakses di https://ieeexplore.ieee.org/document/9563984.

BAB IV

HASIL PENELITIAN DAN PEMBAHASAN

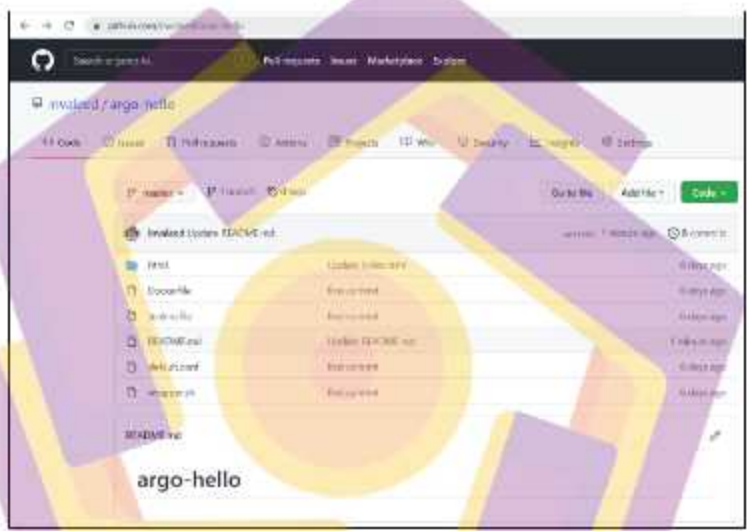
4.1. Hasil Penelitian

Pada bab ini, peneliti akan menguraikan hasil dan data penelitian yang bertujuan untuk mengetahui apakah metode GitOps dapat menyelesaikan masalah yang dihadapi oleh pengguna metode DevOps saat ini. Sebagaimana yang telah diuraikan pada bab sebelumnya, peneliti menggunakan metode kualitatif untuk menjawab pertanyaan tersebut di atas. Selanjutnya, peneliti juga menggunakan metode observasi dan dokumentasi untuk mengetahui lebih dalam dan jelas serta mentriangulasi mengenai data yang telah ada untuk kemudian dianalisis. Analisis itu sendiri akan terfokus pada hasil dari pengujian mandiri yang peneliti lakukan.

4.1.1. Perancangan & Pengembangan

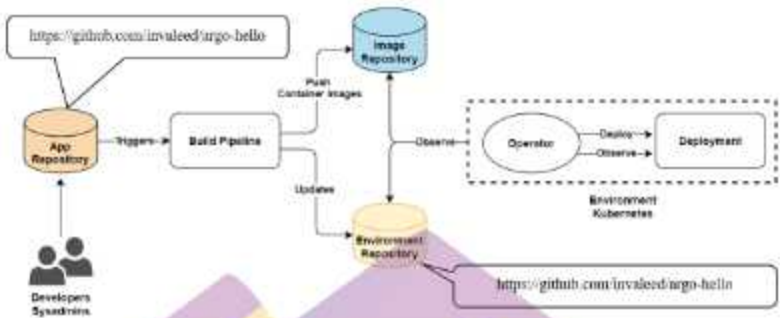
Tahap ini dimulai dengan pengumpulan dan pengolahan data yang akan digunakan oleh peneliti dalam pengujian mandiri, peneliti menggunakan dua kode sumber aplikasi yang berada di Github peneliti yang beralamat di https://github.com/invaleed/argo-hello yang berfungsi sebagai repositori yang digunakan oleh developer sehari-hari untuk meletakkan kode dan memperbaiki kode, repositori ini biasa disebut dengan repositori aplikasi atau *application repository*. Pada penelitian ini, bahasa pemrograman yang digunakan adalah HTML. Dalam kode sumber tersebut terdapat berkas Jenkinsfile yang merupakan *script pipeline* yang digunakan untuk melakukan langkah-langkah yang sudah

didefinisikan, berkas Dockerfile berfungsi untuk membuat Docker image yang nantinya akan diletakkan di *image registry* dan kemudian akan dipasang atau di-deploy pada *Kubernetes environment*. Detail isi dari repositorinya bisa kita lihat dengan mengakses halaman repositori melalui peramban web, seperti yang dapat dilihat pada gambar 4.1.



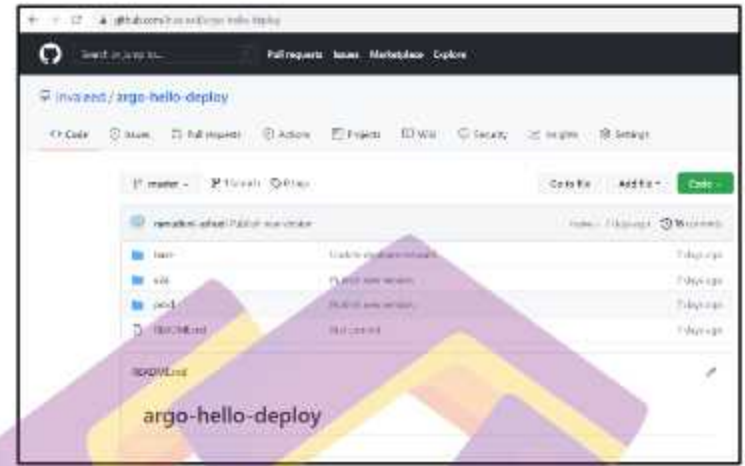
Gambar 4.1. Detail isi repositori aplikasi

Repositori yang ke-dua beralamat di https://github.com/invaleed/argo-hello-deploy yang berfungsi sebagai *environment repository* atau *configuration repository*, yang akan digunakan oleh operator sebagai satu-satunya sumber *deployment* yang dapat dipercaya. Gambaran penggunaan kedua repositori tersebut terlihat pada gambar 4.2.



Gambar 4.2. Gambaran alur penggunaan repositori

Detail isi dari *environment repository* atau *configuration repository* bisa kita lihat dengan mengakses halaman repositori melalui peramban web. Folder "base" berisi konfigurasi dasar atau inisial konfigurasi dari deployment aplikasi "argo-helo", sedangkan folder "e2e" berisi konfigurasi yang akan digunakan pada proses deployment ke enviroment *end to end testing* atau biasa disebut dengan *environment testing*, sedangkan folder "prod" berisi konfigurasi yang akan digunakan pada proses deployment ke lingkungan produksi. Isi dari *environment repository* dapat dilihat pada gambar 4.3.



Gambar 4.3. Detail Isi Environment Repository

Setelah semua kode sumber sudah tersedia, selanjutnya adalah mempersiapkan platform kubernetes yang akan digunakan sebagai target akhir proses deployment. Pada proses penerapannya, pertama-tama peneliti harus mempersiapkan klaster Kubernetesnya, hal ini peneliti lakukan dengan bantuan tools otomasi agar proses provisioningnya menjadi lebih cepat. Dalam hal ini peneliti menggunakan tools Vagrant untuk melakukan provisioning dan instalasi klaster Kubernetes.

Langkah pertama yang dilakukan untuk melakukan instalasi Kubernetes menggunakan bantuan tool Vagrant adalah membuat berkas Vagrantfile dan beberapa berkas bootstrap seperti yang terlihat pada lampiran 1-4, kemudian jalankan baris perintah "*vagrant up*" pada *Windows command line*, maka proses

pembuatan virtual mesin, instalasi semua paket yang dibutuhkan, instalasi Kubernetes, dan proses join klaster akan dilakukan secara otomatis.

Untuk spesifikasi server yang digunakan dalam penelitian ini adalah menggunakan virtual mesin yang berjalan di atas tool Virtualbox dengan spesifikasi masing-masing mesin adalah seperti yang terlihat pada tabel 4.1.

Tabel 4.1. Spesifikasi Server Kubernetes

Hostname	IP Address	OS	CPU	RAM	Disk	K8s Version
k8s-master	192.168.1.150	CentOS 7	2	4GB	50 GB	v1.22.1
k8s-worker1	192.168.1.151	CentOS 7	1	4GB	50 GB	v1.22.1
k8s-worker2	192.168.1.152	CentOS 7	1	4GB	50 GB	v1.22.1

4.1.2. Demonstrasi

Setelah proses penyiapan klaster Kubernetes selesai, proses selanjutnya adalah proses pemasangan Argo CD. Argo CD dipasang di klaster Kubernetes pada namespace "argocd". Proses instalasi ini cukup dengan menjalankan dua baris perintah seperti yang terlihat pada gambar 4.4.

```
# kubectl create namespace argocd
# kubectl apply -n argocd -f install.yaml
```

Gambar 4.4. Instalasi Argo CD pada klaster Kubernetes

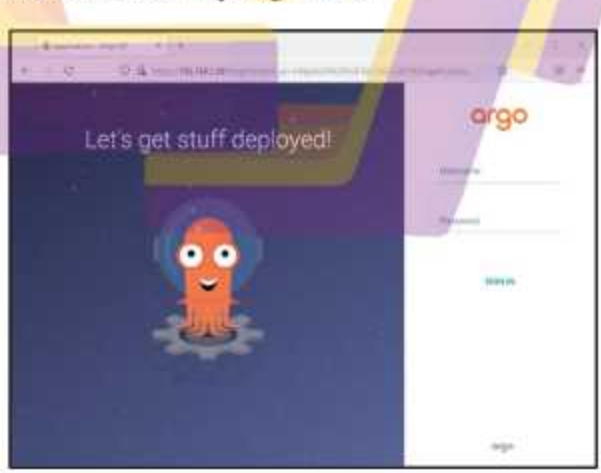
Proses instalasi ArgoCD sendiri memakan waktu 2-5 menit, tergantung pada kecepatan koneksi internet yang digunakan. Setelah proses instalasi selesai, langkah

verifikasi perlu dilakukan apakah proses instalasi Argo CD berjalan dengan baik atau tidak. Verifikasi dapat dilakukan dengan menjalankan baris perintah sesuai dengan gambar 4.5. Jika semua status *pod* adalah *Running*, itu artinya bahwa semua servis yang berhubungan dengan aplikasi Argo CD sudah berjalan dengan baik.

```
ramadoni@orange:~$ kubectl get pods -n argocd
NAME                                  READY   STATUS    RESTARTS   AGE
argocd-application-controller-0       1/1     Running   0          37m
argocd-dex-server-6dcf645b6b-g6tf1    1/1     Running   0          37m
argocd-redis-5b6967fdfc-c66wk         1/1     Running   0          37m
argocd-repo-server-7598bf5999-sh6js   1/1     Running   0          37m
argocd-server-79f9bc9b44-sjb9c        1/1     Running   0          37m
```

Gambar 4.5. Status deployment Argo CD pada klaster Kubernetes

Setelah semua servis dipastikan berjalan dengan baik, langkah selanjutnya adalah mengakses aplikasi Argo CD melalui peramban web, Argo CD ini akan berfungsi sebagai operator yang akan selalu melakukan pemeriksaan terhadap kondisi dan perubahan yang terjadi di *environment repository*. Tampilan halaman depan aplikasi Ardo CD terlihat pada gambar 4.6.



Gambar 4.6. Halaman depan aplikasi Argo CD

Langkah selanjutnya adalah mendaftarkan repositori kode sumber yang akan digunakan pada proses pengujian mandiri. Repositori yang didaftarkan pada Argo CD adalah *environment repository* bukan *application repository*. Jika proses penambahan berhasil, maka status yang terlihat adalah *Successful*. Tampilan konfigurasi Argo CD yang telah berhasil ditambahkan *environment repository* terlihat pada gambar 4.7.

Gambar 4.7. Konfigurasi repositori di Argo CD

Hingga tahapan ini, proses selanjutnya adalah melakukan pengukuran seberapa besar resource yang dibutuhkan untuk menjalankan perangkat lunak ArgoCD ini. Resources yang digunakan oleh ArgoCD terlihat pada gambar 4.8.

```
ramadoni@orange:~$ kubectl top pods -n argocd
NAME                                  CPU(cores)   MEMORY(bytes)
argocd-application-controller-0       1m           16Mi
argocd-dex-server-6dcf645b6b-g6tfl    1m           15Mi
argocd-redis-5b6967fdfc-666wk         3m           6Mi
argocd-repo-server-7598bf5999-sh6js   3m           20Mi
argocd-server-79f9bc9b44-sjb9c        2m           22Mi
ramadoni@orange:~$
```

Gambar 4.8. Resources yang digunakan oleh ArgoCD

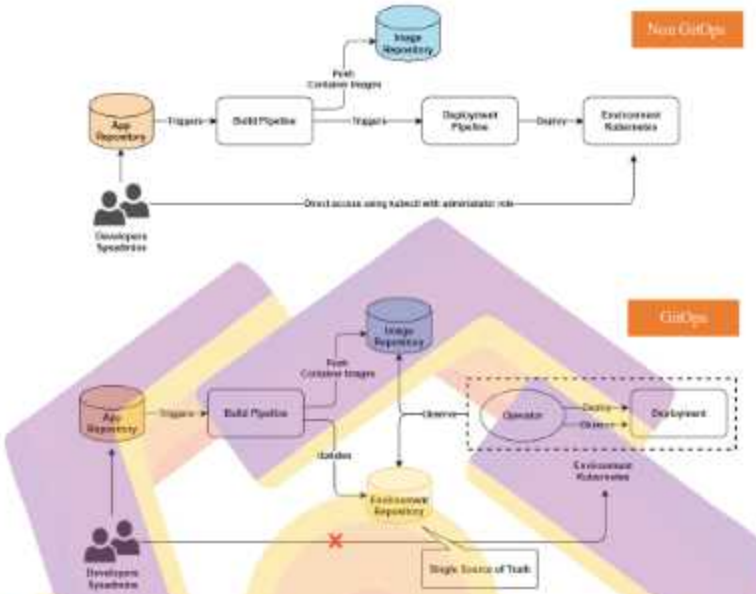
Dari gambar di atas terlihat bahwa ArgoCD menjalankan 5 pod yang secara total menggunakan CPU sebesar 10 *milicores* dan menggunakan kapasitas memori sebesar 79 *megabytes*.

4.1.3. Pengujian

Proses pengujian dilakukan setelah selesai menempatkan *environment repository* yang berada di dalam Github ke dalam konfigurasi di Argo CD. Ada beberapa beberapa pengujian yang akan dilakukan dalam penelitian ini, pengujian pertama terkait Security, pengujian kedua terkait proses Rollback.

4.1.3.1. Pengujian Security

Pada pengujian security, peneliti menganalisa tentang faktor keamanan, dimana pada CI/CD yang umum dipakai sekarang, seseorang dimungkinkan mempunyai kemampuan untuk mengakses langsung ke dalam klaster, yang artinya ini sangat berbahaya, karena ketika seseorang misalkan developer atau sysadmin mempunyai akses langsung ke klaster, maka mereka mampu melakukan perubahan terhadap klaster, perubahan yang dimaksud adalah perubahan tentang kondisi klaster seperti menghapus deployment, mengurangi jumlah pod, bahkan mampu menghapus klaster. Di sini banyak faktor kesalahan manusia yang sangat bisa terjadi. Secara sederhana perbandingan antara metode Non GitOps dan GitOps dalam hal akses ke klaster bisa dilihat pada gambar 4.9.



Gambar 4.9. Perbandingan metode GitOps dan Non GitOps

Setelah menempatkan semuanya di Git, langkah selanjutnya adalah penerapan aplikasi pada ArgoCD dengan menambahkan aplikasi melalui dashboard Argo CD. Pada proses ini Argo CD sebagai operator akan melakukan pengecekan status yang ada di manifest Git, dan melakukan pemeriksaan ke klaster kubernetes, kemudian membandingkan status antara keduanya. Sampai tahap ini status aplikasi adalah *OutOfSync* yang artinya aplikasi belum diterapkan ke platform Kubernetes, sehingga kondisi di *actual state* berbeda dengen kondisi di *desired state*. status *OutofSync* ini digambarkan pada gambar 4.10.

Gambar 4.10. Aplikasi sebelum diterapkan di klaster Kubernetes

Untuk melakukan instalasi ke klaster Kubernetes, kita perlu memastikan bahwa antara desired state dan actual state harus sama, untuk itu kita bisa melakukannya dengan melakukan *sync*, dengan menekan tombol sync pada dashboard Argo CD. Hal ini sebetulnya bisa terjadi secara otomatis jika kita mengaktifkan "auto sync" pada konfigurasi aplikasi Argo CD. Setelah proses sync dijalankan, maka tampilan dashboard akan berubah, karena Argo CD secara otomatis akan membandingkan kembali kondisi *actual state* dan *desired state* seperti terlihat pada gambar 4.11.

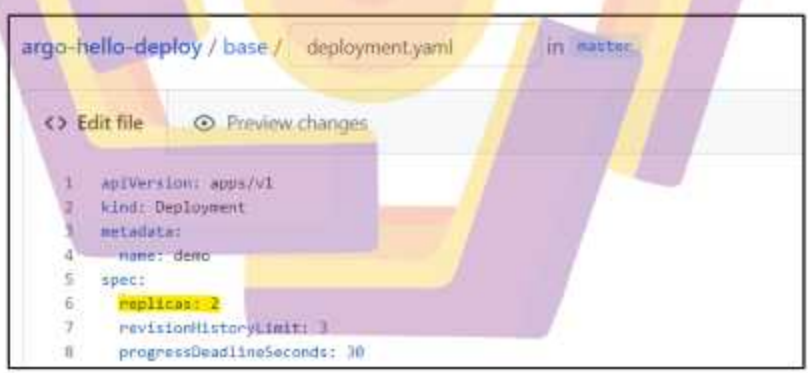
Gambar 4.11. Status aplikasi setelah diterapkan di klaster Kubernetes

Jika kita periksa kedalam klaster Kubernetes, maka aplikasi argo-hello sudah terpasang sesuai dengan manifes aplikasi yang kita tentukan, sebagai contoh di manifes kita sebutkan replicas=2 maka jumlah pod aplikasi yang terpasang juga akan berjumlah 2 replika. Kondisi ini tergambar pada gambar 4.12.

```
ramadoni@orange:~$ kubectl get pods -n e2e
NAME                              READY   STATUS    RESTARTS   AGE
demo-hello-e2e-d9fcdd664-9bfgb    1/1     Running   0          7m47s
demo-hello-e2e-d9fcdd664-gsgwk    1/1     Running   0          7m47s
```

Gambar 4.12. Jumlah Pod pada namespace e2e

Jika kita bandingkan dengan manifest yang ada di Git di folder base, maka kondisi di atas akan sesuai, sehingga Argo CD sebagai operator akan menampilkan status synced. Berikut ini adalah tampilan manifest yang ada di repository pada folder base, terlihat pada gambar 4.13.



Gambar 4.13. Tampilan manifes repository

Dalam proses pengujian pertama ini peneliti mencoba melakukan perubahan konfigurasi tanpa melalui perubahan pada manifes di Git, tetapi

langsung melakukan perubahan ke dalam klaster menggunakan tool kubectl, ini dimungkinkan karena peneliti mempunyai akses langsung ke klaster. Peneliti mengubah jumlah replika menjadi 1 seperti yang tergambar pada gambar 4.14.

```
ramadoni@orange:~$ kubectl get deployment -n e2e
NAME             READY   UP-TO-DATE   AVAILABLE   AGE
demo-hello-e2e   2/2     2            2           26m
ramadoni@orange:~$ kubectl scale deployment demo-hello-e2e --replicas=1 -n e2e
deployment.apps/demo-hello-e2e scaled
ramadoni@orange:~$ kubectl get deployment -n e2e
NAME             READY   UP-TO-DATE   AVAILABLE   AGE
demo-hello-e2e   1/1     1            1           26m
ramadoni@orange:~$ kubectl get pods -n e2e
NAME                              READY   STATUS    RESTARTS   AGE
demo-hello-e2e-5d949c9747-7q7tf   1/1     Running   0          6m53s
```

Gambar 4.14. Perubahan langsung ke Klaster

Jika tanpa GitOps, kondisi ini akan terlihat normal karena tidak adanya operator yang melakukan pengecekan akan hal ini, tidak ada operator yang berfungsi membandingkan *actual state* dengan *desired state*, padahal, kondisi di *actual state* sudah berubah, bahkan bisa saja menjadi sebuah masalah besar dan mengganggu servis yang sedang berjalan. Sebagai contoh adalah pada kondisi jumlah pod adalah 2, permintaan yang bisa dilayani berjumlah 100, jika di actual statenya sudah berkurang menjadi 1 pod, maka jumlah permintaan yang bisa dilayani hanya 50, jika ternyata permintaan pada saat itu berjumlah 80, maka 30 permintaan akan tidak bisa dilayani, artinya akan ada resiko disana, dan jika servis tersebut adalah layanan publik, maka tidak hanya berdampak pada sistem yang mati, tapi bisa berdampak pada reputasi.

Jika kita menggunakan model GitOps, Argo CD akan memberitahu bahwa status penerapan di klaster dan yang ada di git manifes terdapat perbedaan, maka

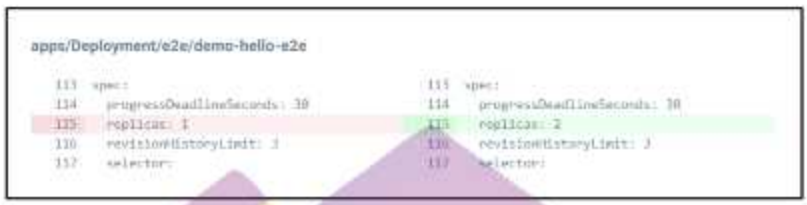
Argo CD akan mengubah status penerapan aplikasi menjadi *OutOfSync* seperti telihat pada gambar 4.15.

Gambar 4.15. Status OutOfSync

Saat terjadi kondisi seperti ini kita memiliki beberapa pilihan tindakan. Dalam kebanyakan kasus, kita cukup menyinkronkan lagi penerapan. Ini akan membuang perubahan yang dilakukan secara manual tadi dan mengembalikan klaster ke status yang sama dengan git manifes. Jika kita yakin bahwa perubahan yang dilakukan secara manual tadi memang penting dan diperlukan, cara yang benar adalah dengan mengubah manifes yang ada di git. Dengan cara ini Argo CD akan membandingkan lagi kondisi di klaster dan kondisi manifes yang ada di Git. Skenario membuang perubahan manual juga dapat dilakukan secara otomatis dengan mengaktifkan opsi auto sync sehingga Argo CD selalu menerapkan status klaster berdasarkan kondisi manifest di Git.

Jadi saat seseorang melakukan perubahan manual ke dalam klaster, Argo CD mendeteksinya dan menandai penerapan sebagai OutOfSync. Argo CD tidak hanya mengetahui bahwa aplikasi tidak lagi mencerminkan status di git manifes,

tetapi juga menawarkan tampilan diff yang berguna untuk menjelaskan dan memberi tahu kita apa yang diubah, seperti yang terlihat pada gambar 4.16.



Gambar 4.16. Tampilan diff perubahan pada manifes klaster

Terlihat bahwa Argo CD dapet menampilkan bahwa perubahan yang tadi dilakukan adalah mengubah jumlah replika dari 2 menjadi 1. Ini sangat membantu sysadmin atau developer untuk mengetahui perubahan yang ada, tanpa perlu membandingkan satu per satu baris kode yang ada.

Perubahan yang tepat adalah dengan mengubah manifes yang ada pada kode sumber di Git, jika perubahan dilakukan melalui kode sumber di Git, setiap perubahan akan dicatat dalam log Git, dan skema *dual control* yang bisa diterapkan adalah dengan menggunakan proses *code review* dan *approval* pada setiap proses *merge* pada kode sumber. Perubahan yang dilakukan dengan mengubah manifes pada kode sumber juga akan menghasilkan status OutOfSync, dan untuk mengatasi ini perlu dilakukan sinkronisasi secara manual maupun otomatis agar apa yang diterapkan di klaster akan selalu sama dengan yang ada di manifes. Dari proses pengujian didapatkan hasil seperti terlihat pada tabel 4.2.

Tabel 4.2. Hasil Pengujian Security

Faktor	Non GitOps	GitOps
Akses Klaster	Menjadi tidak aman karena seseorang bisa mengubah konfigurasi klaster bahkan menghapus klaster secara langsung.	Menjadi aman dikarenakan menggunakan metode *single source of truth*, maka akses langsung ke dalam cluster tidak dimungkinkan.
Pull vs Push	Dengan menggunakan metode push, user dan password akan disimpan di luar lingkungan klaster, sehingga kemungkinan diretas sangat besar.	Menggunakan model pull, user dan password tidak keluar dari lingkungan klaster, jadi kemungkinan diretas menjadi sangat kecil.
Configuration Drift	Tanpa adanya operator, proses pipeline dengan cara mengubah langsung ke klaster bisa menyebabkan configuration drift.	Menggunakan operator yang selalu membandingkan *desired state* dan *actual state*, sehingga configuration drift tidak dimungkinkan.
Catatan perubahan	Perubahan yang dilakukan secara langsung mengakibatkan tidak adanya catatan perubahan yang sudah dilakukan.	Dengan menggunakan git sebagai repository, semua perubahan tercatat dalam log git.

4.1.3.2. Pengujian Rollback

Pengujian selanjutnya adalah tentang proses *rollback*. Dalam aplikasi yang menggunakan CI tradisional, kita perlu mencari dan memicu *pipeline* penerapan

masing-masing dengan parameter berbeda. Dengan Argo CD kita cukup memilih rilis git lain pada *dashboard* yang akan digunakan untuk proses sinkronisasi dan memastikan bahwa setelah beberapa saat klaster akan kembali ke *commit hash* tersebut. Pengujian ini dimulai dengan mendeploy aplikasi ke dalam klaster kubernetes, aplikasi yang digunakan adalah aplikasi yang sama dengan yang digunakan pada pengujian pertama. Tampilan dari halaman web yang telah di-deploy terlihat pada gambar 4.17.

Gambar 4.17. Tampilan halaman depan web

Langkah selanjutnya kita lakukan perubahan pada kode yang berada dalam Github untuk mengubah kata "Hello World" menjadi "Hello AMIKOM". Semua perubahan yang terjadi di Github akan tercatat dan mempunyai *identifier* unik yang biasa disebut dengan *Commit Hash* atau *Commit id*. Commit hash pada perubahan kali ini adalah **ae24bfa** dengan *commit message* nya adalah "Change to Hello AMIKOM" seperti terlihat pada gambar 4.18.

Gambar 4.18. Tampilan perubahan pada Git

Saat perubahan ini di-commit, maka pipeline di server CI akan berjalan sesuai dengan apa yang sudah ditentukan pada berkas Jenkinsfile yang ada di *application repository*. Pada berkas Jenkinsfile terlihat bahwa *pipeline* tersebut akan dijalankan menjadi beberapa *stage* atau langkah. Langkah pertama adalah menarik kode dari repositori yang sudah ditentukan, kemudian "Build" yang artinya dari kode tersebut akan dibuat menjadi sebuah *Docker image* berdasarkan berkas Dockerfile yang ada, kemudian akan di-upload ke *container registry* di dockerhub, kemudian akan melakukan *deployment* Docker image tersebut ke lingkungan "e2e". Apabila disetujui maka langkah selanjutnya adalah mendeploy ke lingkungan produksi. Berikut ini tampilan dari CI server saat menjalankan pipeline, dalam penelitian ini peneliti menggunakan Jenkins. Dalam gambar 4.19, terlihat langkah-langkah yang sudah didefinisikan di berkas Jenkinsfile diaplikasikan dengan sesuai oleh Jenkins.

Gambar 4.19. Tampilan pipeline pada Jenkins.

Terlihat juga di dashboard Jenkins bahwa proses dari awal hingga akhir pipeline ini memerlukan waktu 26 detik seperti terlihat pada gambar 4.20.

Gambar 4.20. Tampilan waktu untuk menjalankan pipeline.

Sampai langkah ini, seharusnya Argo CD akan mendeteksi perubahan yang terjadi pada environment repository dikarenakan perubahan pada Id Docker image dikarenakan proses *build* pada *pipeline*. Perubahan yang terjadi dapat dilihat pada berkas kustomization.yaml dengan commit id nya adalah **cc63450** seperti terlihat pada gambar 4.21.

Gambar 4.21. Tampilan perubahan pada berkas kustomization.yml.

Dikarenakan ada perubahan tersebut, maka Argo CD akan mendeteksi bahwa ada perubahan pada *desired state* dan akan mengubah status *deployment* menjadi OutOfSync seperti terlihat pada gambar 4.22. Dalam gambar tersebut juga terlihat bahwa commit id nya **(cc03450)** sama dengan yang ada di repository.

Gambar 4.22. Perubahan status OutOfSync.

Hingga tahap ini, tampilan halaman web masih belum berubah, karena perubahan terakhir belum diterapkan oleh Argo CD, hal ini sebetulnya bisa

dilakukan secara otomatis tetapi untuk keperluan penelitian menu *auto sync* tidak di-enable. Untuk mendapatkan tampilan web yang baru, perlu dilakukan *sync* antara *desired state* dan *actual state.* Ketika status di Argo CD sudah berubah ke status synced, maka tampilan web akan berubah secara otomatis seperti yang terilhat pada gambar 4.23.

Gambar 4.23. Tampilan halaman web setelah perubahan.

Jika perubahan yang telah dilakukan ini ternyata salah atau mengalami kesalahan saat eksekusinya, maka perlu dilakukan proses rollback dengan sesegera mungkin agar tidak terjadi downtime terlalu lama atau aplikasi tidak bisa digunakan dan akan berimbas pada layanan kepada konsumen.

Pada metode non GitOps kita perlu melakukan step dari mulai commit ke application repositori menggunakan *git revert* yang artinya kita akan memerlukan waktu 30 detik lagi untuk bisa mendapatkan tampilan web sebelumnya. Sedangkan dengan bantuan Argo CD kita hanya perlu memilih rilis git lain pada dashboard yang akan digunakan untuk proses sinkronisasi dan memastikan bahwa setelah

beberapa saat klaster akan kembali ke commit hash tersebut. Menu rollback pada Argo CD digambarkan pada gambar 4.24.

Gambar 4.24. Tampilan menu rollback Argo CD.

Setelah masuk ke menu History And Rollback kita hanya perlu memilih kita akan rollback ke commit hash yang mana dan kemudian klik Rollback, dan terlihat juga bahwa kita hanya perlu kurang dari 1 detik untuk melakukan rollback seperti terlihat pada gambar 4.25.

Gambar 4.25. Tampilan menu History and Rollback Argo CD.

Pada tahapan ini tampilan web sudah berubah menjadi "Hello World" kembali. Kemampuan rollback yang cepat dan efektif ini sangat membantu dalam mengurangi waktu downtime dan mengurangi berapa lama servis tidak bisa melayani konsumen yang artinya dapat mengurangi potensi resiko dan kerugian perusahaan. Berikut ini adalah hasil dari pengujian *rollback* yang tergambar dalam tabel 4.3.

Tabel 4.3. Hasil Pengujian Rollback

Faktor	Non GitOps	GitOps
Waktu Deploy	Rata-rata waktu deploy adalah 30 detik.	Rata-rata waktu deploy adalah 30 detik.
Jumlah Langkah	4 Langkah yang meliputi: Pull Code Repository → Build Docker Image → Upload Docker Image → Deploy ke Kubernetes	5 Langkah yang meliputi: Pull Code Repository → Build Docker Image → Upload Docker Image → Update Environment Repository → Deploy ke Kubernetes
Kemudahan	Tidak mudah untuk melakukan rollback karena proses rollback musti menjalankan semua stage build dari awal.	Menjadi mudah karena dengan adanya GUI/Dashboard Argo CD, proses rollback hanya dengan klik tombol rollback.
Waktu Rollback	Lama waktu yang digunakan untuk rollback sama dengan saat pertama kali aplikasi di-deploy. Pada pengujian ini membutuhkan waktu 30 detik.	Proses rollback tidak memerlukan waktu lama karena tidak mengulang proses deploy dari awal, pada pengujian ini hanya membutuhkan waktu rata-rata kurang dari 1 detik.
Monitoring	Tanpa adanya GUI/Dashboard, proses monitoring menjadi lebih susah.	Proses monitoring menjadi lebih mudah karena menggunakan GUI.

4.2. Pembahasan

Hasil dari pengujian pada penelitian ini menyempurnakan dan menyelesaikan masalah yang diungkapkan dalam penelitian sebelumnya yang menjadi rujukan dalam paper ini, penelitian (Bolscher & Daneva, 2019) mengungkapkan bahwa arsitektur micro-services adalah model arsitektur yang paling dominan mendukung implementasi Continuous Delivery (CD) dan DevOps, dengan menggunakan model push-based deployment. Pada prosesnya, penggunaan metode GitOps yang menggunakan pendekatan pull-based deployment akan menjadikan proses deployment menjadi lebih aman. Dalam hal keamanan, masalah-masalah yang dapat diselesaikan oleh metode GitOps meliputi:

1. Seseorang tidak dimungkinkan untuk bisa melakukan perubahan ke klaster secara langsung, karena dengan metode Git sebagai *single source of truth*, tidak dimungkinkan lagi seseorang bisa mengakses klaster secara langsung, karena proses perubahan yang terjadi harus melewati Git.
2. User dan password akan tersimpan tidak di luar klaster sehingga mengurangi kemungkinan digunakan oleh pihak yang tidak bertanggung jawab.
3. Configuration drift yang bisa mengakibatkan sistem tidak stabil bahkan hilang tidak dimungkinkan lagi dengan penggunaan GitOps, karena dengan bantuan operator ArgoCD yang akan selalu membandingkan kondisi di repositori (*desired state*) dan kondisi yang ada di klaster (*actual state*). Jika terdapat perbedaan, maka operator akan

memberitahukan dan kemudian akan mengembalikan statenya ke desired state.

4. Auditable, dengan adanya GitOps, sistem akan menjadi mudah untuk dilakukan audit, karena seluruh perubahan akan tercatat pada log Git.

Penelitian yang dilakukan oleh (Shahin, Babar, et al., 2017) fokus pada mengidentifikasi 30 pendekatan dan tools yang mampu memfasilitasi penerapan praktik berkelanjutan DevOps, seperti halnya DevOps, pemilihan tools yang tepat pada GitOps juga perlu dilakukan karena tools yang digunakan dalam metode DevOps tidak sepenuhnya cocok digunakan dalam model GitOps, tools yang ada tetap bisa diintegrasikan ke tools GitOps. Pada penelitian sebelumnya tidak dibahas mengenai tools yang digunakan pada metode GitOps, pada penelitian ini tools GitOps dideskripsikan dengan jelas beserta bagaimana cara instalasi, integrasi dan penggunaannya menggunakan ArgoCD. Dalam hal ini ArgoCD diintegrasikan dengan tools DevOps yang ada, karena proses CI sendiri masih harus tetap menggunakan pendekatan CI yang ada saat ini.

Pada penelitian (Agarwal et al., 2019) menyebutkan beberapa faktor yang mencegah organisasi untuk mengadopsi pendekatan DevOps adalah metodologi rollback yang buruk, masalah ini akan dengan mudah diselesaikan oleh metode GitOps. Pada penelitian ini dibahas bagaimana proses *rollback* bisa dilakukan hanya dengan menekan satu tombol, dan proses rollbacknya pun menjadi lebih cepat karena tidak perlu melakukan proses mengeksekusi *pipeline* dari awal lagi secara sekuensial, seperti proses *build docker image* dan *upload docker image*, dimana proses *build docker image* dan *upload docker image* ini biasanya adalah

proses yang paling lama membutuhkan waktu dibanding dengan tahapan pipeline yang lain. Secara Masalah-masalah terkait proses rollback yang diselesaikan oleh metode GitOps meliputi:

1. Dengan menggunakan GitOps proses *rollback* menjadi lebih mudah dikarenakan semua bisa dilakukan dengan melakukan beberapa klik pada dashboard ArgoCD.
2. Waktu yang diperlukan untuk melakukan rollback menjadi lebih sedikit, dari 30 detik menjadi 0.5 detik.
3. Proses mengetahui komponen kode apa yang berubah juga menjadi lebih mudah dengan adanya fitur diff pada ArgoCD. Sehingga proses pencarian rootcause ketika terjadi masalah pada saat deployment maupun post deployment menjadi mudah.

Pada penerapannya, GitOps hanya mengubah satu bagian dari sekian bagian yang ada pada keseluruhan *pipeline* penerapan. Dari hasil pengujian, didapati bahwa Effort yang dilakukan oleh perusahaan, tim pengembang, dan tim operasi dalam mengadopsi GitOps sangatlah kecil. Effort diartikan sebagai seberapa besar perubahan yang dilakukan untuk mengadopsi metode baru.

1. Perusahaan tidak perlu menyediakan perangkat baru untuk mengadopsi GitOps, karena dari hasil demonstrasi, penggunaan operator ArgoCD hanya membutuhkan resources sebesar 10 milicores CPU dan 79 *megabytes* memori.
2. Dari sisi pengembang, kode aplikasi yang tersedia saat ini tidak perlu adanya perubahan sama sekali, perubahan hanya terjadi pada pembuatan

repositori tambahan, yaitu *environment repository* dan perubahan pada *pipeline* yang tadinya tidak perlu mengupdate *environment repository* menjadi perlu mengupdate *environment repository* jika terjadi perubahan. Karena, seperti kita ketahui, proses perubahan pada kode aplikasi bisa berdampak sangat besar pada keseluruhan bisnis proses yang ada.

3. Dari sisi operation, IT Operation dalam hal ini seorang sysadmin atau DevOps hanya perlu melakukan instalasi operator dan mengkonfigurasinya, dimana instalasi operator ini bisa dilakukan dengan hanya melakukan dua baris perintah saja yang membutuhkan waktu sekitar 2-5 menit tergantung dari kecepatan koneksi internet yang digunakan.

Beberapa catatan penting yang diperoleh dari penelitian ini adalah, dengan GitOps, konten dari repositori git memberi tahu apa yang harus diterapkan di klaster. Riwayat commit Git pada dasarnya juga dapat berfungsi sebagai catatat atau riwayat penerapan klaster. Dalam penerapan tradisional, beberapa kali anggota tim mungkin melakukan perubahan pada klaster yang tidak direkam di mana pun. Ini adalah kejadian yang sangat umum misalkan untuk melakukan *quick fix*, atau pembaruan darurat lainnya. Meskipun menggunakan tool kubectl adalah cara yang sangat efektif untuk mengelola sebuah klaster, anggota tim juga dapat menyalahgunakan kubectl dan membuat masalah *configuration drift*. Argo CD memiliki cara yang sangat cerdas untuk mengatasi masalah *configuration drift* ini. Setelah penerapan awal dilakukan, Argo CD terus memantau klaster dan

membandingkannya dengan git manifes. Jika suatu saat seseorang melakukan perubahan manual dalam klaster, ArgoCD mendeteksinya dan menandai penerapan sebagai OutOfSync. Visibilitas instan dari penerapan yang tidak sinkron ini sangat membantu. ArgoCD tidak hanya mengetahui bahwa aplikasi tidak lagi mencerminkan status git manifes, tetapi juga menawarkan tampilan diff yang berguna untuk menjelaskan apa yang diubah. Penyimpangan konfigurasi adalah masalah yang sering terjadi, tetapi semakin lama hal itu berlangsung maka akan semakin fatal. Tim yang tidak tahu persis apa yang diterapkan pada sebuah klaster dapat mengalami banyak masalah karena hal tersebut.

Berdasarkan keseluruhan analisa yang sudah dilakukan, berikut ini adalah desain penerapan akhir dan tabel hasil analisa dan perbandingan proses yang digunakan untuk menjawab dan telah diuji serta berhasil mensolusikan semua masalah penerapan pada model DevOps, baik yang sudah didefinisikan di awal ataupun masalah yang ditemukan saat penelitian dilakukan. Detail Analisa dapat dilihat pada Tabel 4.4.

Tabel 4.4. Tabel Analisa Perbandingan Non GitOps vs GitOps

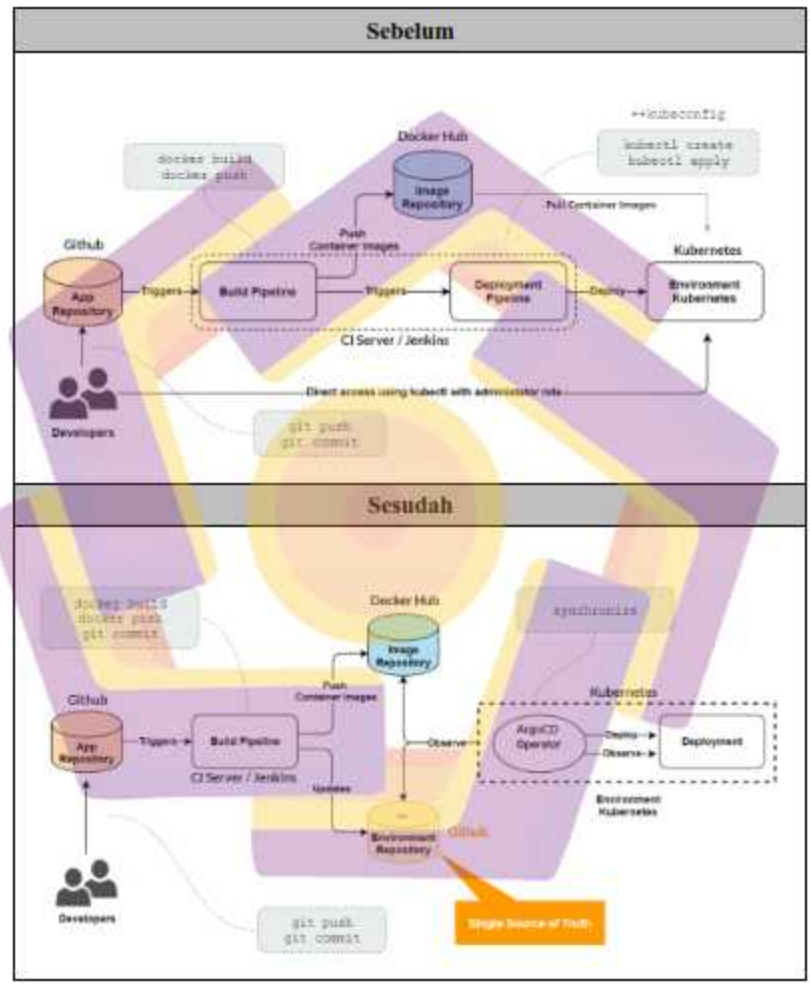
Analisis Penggunaan Metode Gitops pada Pengembangan Perangkat Lunak Berbasis Devops

Model			**Non GitOps**	**GitOps**
Komponen yang digunakan			1. Code Repository : Github.com 2. CI Server : Jenkins 3. Image Repository : Dockerhub 4. Platform Deployment : Kubernetes	1. Code Repository : Github.com 2. CI Server : Jenkins 3. Image Repository : Dockerhub 4. Platform Deployment : Kubernetes 5. Operator : ArgoCD
Metode			Declarative, Push Based	Declarative, Pull Based
Hasil Analisa	**Security**	**Akses Klaster**	Dimungkinkan dilakukan perubahan langsung ke klaster.	Menerapkan konsep Single Source of Truth sehingga perubahan ke klaster hanya bisa dilakukan melalui Git.
		Password	Lokasi user dan password (kubeconfig) ada di luar klaster.	Komunikasi ke klaster dilakukan oleh ArgoCD menggunakan service account dan berada dalam satu lingkungan klaster.
		Configuration Drift	Memungkinkan terjadinya configuration drift karena setiap orang yang mempunyai akses ke kubeconfig dapat melakukan perubahan menggunakan tool kubectl.	Menggunakan Operator yang berfungsi untuk memeriksa kondisi di Desired state vs Actaul state.
		Catatan perubahan	Karena perubahan bisa dilakukan secara langsung, catatan perubahan dimungkinkan tidak tersedia.	Perubahan hanya bisa dilakukan melalui Git, dan setiap perubahan akan tercatat pada log Git.

Tabel 4.4. Lanjutan

Model			Non GitOps	GitOps
Hasil Analisa	Rollback	**Waktu Deploy**	Rata-rata waktu deploy adalah 30 detik.	Rata-rata waktu deploy adalah 30 detik.
		Jumlah Langkah *Deploy*	Langkah Deployment: 1. Pull Code Repository 2. Build Docker Image 3. Upload Docker Image 4. Deploy ke Kubernetes	Langkah Deployment: 1. Pull Code Repository 2. Build Docker Image 3. Upload Docker Image 4. Update Environment Repository 5. Deploy ke Kubernetes
		Jumlah Langkah *Rollback*	Langkah Rollback: 1. Revert Code Repository 2. Build Docker Image 3. Upload Docker Image 4. Deploy ke Kubernetes	Langkah Rollback: 1. Check Environment Repository 2. Deploy ke Kubernetes
		Kemudahan	Tidak mudah untuk melakukan rollback karena proses rollback musti menjalankan semua stage build dari awal.	Menjadi mudah karena dengan adanya GUI/Dashboard Argo CD, proses rollback hanya dengan klik tombol rollback.
		Waktu yang dibutuhkan untuk *Rollback*	Lama waktu yang digunakan untuk rollback sama dengan saat pertama kali aplikasi di-deploy. Pada pengujian ini membutuhkan waktu rata-rata 30 detik.	Proses rollback tidak memerlukan waktu lama karena tidak mengulang proses deploy dari awal, pada pengujian ini waktu rollback kurang dari 1 detik.
		Monitoring	Tanpa adanya GUI/Dashboard, proses monitoring menjadi tidak mudah dan tidak user friendly.	Proses monitoring menjadi lebih mudah dan user friendly karena menggunakan GUI.

Dalam hal desain arsitektur yang digunakan untuk observasi dan ujicoba dalam penelitian ini dapat dilihat pada gambar 4.26.



Gambar 4.26. Desain arsitektur Non GitOps vs GitOps.

BAB V

PENUTUP

5.1. Kesimpulan

Dari hasil pembahasan dan pengujian, penulis menyimpukan bahwa metode GitOps merupakan metode yang bisa menyelesaikan masalah-masalah yang dihadapi oleh model DevOps saat ini. Berikut ini beberapa hal yang terkait dengan kesimpulan tersebut:

1. Pada penerapannya, GitOps hanya mengubah satu bagian dari sekian bagian yang ada pada keseluruhan *pipeline* penerapan. *Effort* yang dilakukan oleh perusahaan, pengembang, dan tim operasi sangatlah kecil. Perusahaan tidak perlu menambah spesifikasi server karena ArgoCD hanya membutuhkan resources sebesar 10 milicores CPU dan 79 *megabytes* memori. Tidak perlu adanya perubahan pada kode aplikasi yang ada saat ini, dan tim operasi hanya butuh waktu 5-10 menit untuk melakukan instalasi dan konfigurasi ArgoCD.
2. Semua masalah yang diangkat pada penelitian ini, yang terdapat pada proses DevOps yang ada saat ini, baik yang direncanakan untuk diteliti maupun yang diketahui saat penelitian berlangsung yang terdiri dari masalah keamanan dan kurang efektifnya proses rollback bisa diselesaikan dan disolusikan dengan penggunaan metode GitOps. Masalah akses langsung ke klaster, configuration drift, dan kemudahan untuk di audit dapat diselesaikan dengan metode single source of truth dari Git. Sedangkan masalah user dan password yang ada diluar klaster disolusikan dengan metode pull-deployment menggunakan operator.

Sedangkan untuk masalah kurang efektifnya rollback, hasil penelitian menunjukkan bahwa proses rollback menjadi lebih cepat dari yang semula membutuhkan waktu 30 detik menjadi hanya 0.5 detik.

3. Terdapat beberapa faktor yang mempengaruhi keberhasilan penerapan GitOps pada sebuah perusahaan. Pertama, mengadopsi GitOps jauh lebih mudah pada perusahaan yang sudah menggunakan Git sebagai repositori kode sumber mereka. Faktor selanjutnya adalah *culture*, dimana dibutuhkan budaya kerjasama yang baik antara *Developer* dan *Operation*, sehingga proses *development* dan *deployment* bisa berjalan dengan baik dan cepat. Dan faktor yang tidak kalah penting adalah kemauan dari masing-masing stakeholder untuk mempelajari dan mengadopsi teknologi baru, menerima perubahan sebagai bagian dari pembelajaran, karena perkembangan IT khususnya pada bidang *software development* terjadi sangat cepat.

5.2. Saran

Argo CD hanyalah salah satu komponen dari fase penerapan yang lebih besar. Menjadi penting untuk mengintegrasikan Argo CD dengan platform otomasi yang lain yang dapat memberikan fungsi penuh dari siklus hidup perangkat lunak. Argo CD tidak dapat digunakan sendiri untuk seluruh siklus hidup perangkat lunak. Argo CD sangat bagus untuk men-deploy aplikasi, yang merupakan langkah terakhir dari proses *deployment*, tetapi harus digabungkan dengan platform lain yang perlu melakukan semua langkah yang ada sebelumnya dan langkah *post deployment*, semisal proses testing dan monitoring.

Sebagai bahan untuk penelitian selanjutnya, penelitian pada metode GitOps yang tidak terbatas hanya pada deployment tetapi juga integrasi dengan CI/CD dan *automatic testing* serta *observability*. Memiliki solusi yang dapat mencakup siklus hidup perangkat lunak secara menyeluruh sangatlah penting terutama bagi perusahaan yang ingin mempraktikkan praktik berkelanjutan secara *end to end*.

DAFTAR PUSTAKA

PUSTAKA BUKU

Davis, J., & Daniels, R. (2016). Effective DevOps: Building a Culture of Collaboration, Affinity, and Tooling at Scale (1st ed.). O'Reilly Media.

Kim, G., Behr, K., & Spafford, G. (2014). The Phoenix Project: A Novel about IT, DevOps, and Helping Your Business Win (Reprint ed.). IT Revolution Press.

Kim, G., Debois, P., Willis, J., Humble, J., & Allspaw, J. (2016). The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations (Illustrated ed.). IT Revolution Press.

Kmetiuk, A. (2018). Mastering Functional Programming: Functional techniques for sequential and parallel programming with Scala. Packt Publishing.

Kneuper, R. (2018). Software Processes and Life Cycle Models: An Introduction to Modelling, Using and Managing Agile, Plan-Driven and Hybrid Processes (1st ed. 2018 ed.). Springer.

Leszko, R. (2017). Continuous Delivery with Docker and Jenkins: Delivering software at scale. Packt Publishing.

Swartout, P. (2012). Continuous Delivery and DevOps: A Quickstart guide (40912th ed.). Packt Publishing.

Verona, J. (2016). Practical DevOps: Harness the power of DevOps to boost your skill set and make your IT organization perform better. Packt Publishing.

Verona, J., Duffy, M., & Swartout, P. (2016). Learning DevOps: Continuously Deliver Better Software. Packt Publishing.

Yuen, B., Matyushentsev, A., Ekenstam, T., & Suen, J. (2021). GitOps and Kubernetes: Continuous Deployment with Argo CD, Jenkins X, and Flux. Manning Publications.

PUSTAKA MAJALAH, JURNAL ILMIAH ATAU PROSIDING

Abbass, M. K. A., Osman, R. I. E., Mohammed, A. M. H., & Alshaikh, M. W. A. (2019). Adopting continuous integeration and continuous delivery for small teams. *Proceedings of the International Conference on Computer, Control, Electrical, and Electronics Engineering 2019, ICCCEEE 2019*. https://doi.org/10.1109/ICCCEEE46830.2019.9070849

Agarwal, A., Gupta, S., & Choudhury, T. (2019). Proceedings of the 2019 9th International Conference on Advances in Computing and Communication, ICACC 2019. *Proceedings of the 2019 9th International Conference on Advances in Computing and Communication, ICACC 2019*, *June*, 290–293.

Arndt, N., & Martin, M. (2019). Decentralized collaborative knowledge management using git. *The Web Conference 2019 - Companion of the World Wide Web Conference, WWW 2019*, 952–953. https://doi.org/10.1145/3308560.3316523

Asthana, N., Chefalas, T., Karve, A., Segal, A., Dubey, M., & Zeng, S. (2018). A

declarative approach for service enablement on hybrid cloud orchestration engines. *IEEE/IFIP Network Operations and Management Symposium: Cognitive Management in a Cyber World, NOMS 2018*, 1–7. https://doi.org/10.1109/NOMS.2018.8406175

Attardi, G., Barchiesi, A., Colla, A., DI Lallo, R., & Galeazzi, F. (2018). Declarative modeling for deploying a container platform. *Proceedings - 32nd IEEE International Conference on Advanced Information Networking and Applications Workshops, WAINA 2018, 2018-Janua*, 386–389. https://doi.org/10.1109/WAINA.2018.00116

Bolscher, R., & Daneva, M. (2019). Designing software architecture to support continuous delivery and DevOps: A systematic literature review. *ICSOFT 2019 - Proceedings of the 14th International Conference on Software Technologies*, 27–39. https://doi.org/10.5220/0007837000270039

Boronin, M. (2020). *Hybrid Cloud Migration Challenges A case study at King Master Thesis Mikhail Boronin Supervisor : David Johnson*.

Breitenbücher, U., Képes, K., Leymann, F., & Wurster, M. (n.d.). *Institute of Architecture of Application Systems Declarative vs . Imperative : How to Model the Automated Deployment of IoT Applications ? Declarative vs . Imperative : How to Model the Automated Deployment of IoT Applications ? SummerSOC 2017*.

Gallaba, K. (2019). Improving the Robustness and Efficiency of Continuous Integration and Deployment. *Proceedings - 2019 IEEE International Conference on Software Maintenance and Evolution, ICSME 2019*, 619–623. https://doi.org/10.1109/ICSME.2019.00099

Hakimzadeh, K., & Dowling, J. (2019). Ops-Scale: Scalable and Elastic Cloud Operations by a Functional Abstraction and Feedback Loops. *International Conference on Self-Adaptive and Self-Organizing Systems, SASO, 2019-June*, 62–71. https://doi.org/10.1109/SASO.2019.00017

Harzenetter, L., Breitenbucher, U., Leymann, F., Saatkamp, K., Weder, B., & Wurster, M. (2019). Automated generation of management workflows for applications based on deployment models. *Proceedings - 2019 IEEE 23rd International Enterprise Distributed Object Computing Conference, EDOC 2019*, 216–225. https://doi.org/10.1109/EDOC.2019.00034

Katal, A., Bajoria, V., & Dahiya, S. (2019). DevOps: Bridging the gap between development and operations. *Proceedings of the 3rd International Conference on Computing Methodologies and Communication, ICCMC 2019*. https://doi.org/10.1109/ICCMC.2019.8819631

Leite, L., Rocha, C., Kon, F., Milojicic, D., & Meirelles, P. (2019). A survey of DevOps concepts and challenges. In *ACM Computing Surveys*. https://doi.org/10.1145/3359981

Limoncelli, T. A. (2018). GitOps: A path to more Self-service IT. *Queue*, *16*(3), 1–14. https://doi.org/10.1145/3236386.3237207

Macarthy, R. W., & Bass, J. M. (2020). An Empirical Taxonomy of DevOps in Practice. *Proceedings - 46th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2020*, 221–228. https://doi.org/10.1109/SEAA51224.2020.00046

Marijan, D., Liaaen, M., & Sen, S. (2018). DevOps Improvements for Reduced Cycle Times with Integrated Test Optimizations for Continuous Integration. *Proceedings - International Computer Software and Applications Conference*. https://doi.org/10.1109/COMPSAC.2018.00012

Mishra, A., & Otaiwi, Z. (2020). DevOps and software quality: A systematic mapping. *Computer Science Review*, *38*, 100308. https://doi.org/10.1016/j.cosrev.2020.100308

Paez, N. (2018). Versioning Strategy for DevOps Implementations. *Congreso Argentino de Ciencias de La Informática y Desarrollos de Investigación, CACIDI 2018*. https://doi.org/10.1109/CACIDI.2018.8584362

Peffers, K., Tuunanen, T., Rothenberger, M. A., & Chatterjee, S. (2007). A design science research methodology for information systems research. *Journal of Management Information Systems*, *24*(3), 45–77. https://doi.org/10.2753/MIS0742-1222240302

Proulx, A., Raymond, F., Roy, B., & Petrillo, F. (2018). *Problems and Solutions of Continuous Deployment: A Systematic Review*. *December 2018*. http://arxiv.org/abs/1812.08939

Rossberg, J. (2019). Agile Project Management with Azure DevOps. In *Agile Project Management with Azure DevOps*. https://doi.org/10.1007/978-1-4842-4483-8

Shahin, M., Ali Babar, M., & Zhu, L. (2017). Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices. *IEEE Access*, *5*(Ci), 3909–3943. https://doi.org/10.1109/ACCESS.2017.2685629

Shahin, M., Babar, M. A., Zahedi, M., & Zhu, L. (2017). Beyond Continuous Delivery: An Empirical Investigation of Continuous Deployment Challenges. *International Symposium on Empirical Software Engineering and Measurement*, *2017-Novem*(November), 111–120. https://doi.org/10.1109/ESEM.2017.18

Tao, X., & Etchevers, X. (2018). Poster : A Declarative Approach for Updating Distributed. *2018 IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion)*, *May*, 392–393.

Wurster, M., Breitenbücher, U., Falkenthal, M., Krieger, C., Leymann, F., Saatkamp, K., & Soldani, J. (2020). The essential deployment metamodel: a systematic review of deployment automation technologies. *Software-Intensive Cyber-Physical Systems*, *35*(1–2), 63–75. https://doi.org/10.1007/s00450-019-00412-x

Wurster, M., Breitenbucher, U., Kopp, O., & Leymann, F. (2018). Modeling and automated execution of application deployment tests. *Proceedings - 2018 IEEE 22nd International Enterprise Distributed Object Computing Conference, EDOC 2018*, 171–180. https://doi.org/10.1109/EDOC.2018.00030

Zahedi, S., Babar, M., & Zhu, L. (2018). *An Empirical Study of Architecting and Organizing for DevOps*. *August*, 1–31. https://digital.library.adelaide.edu.au/dspace/bitstream/2440/116767/1/02whole.pdf

Zampetti, F., Bavota, G., Canfora, G., & Penta, M. Di. (2019). A Study on the Interplay between Pull Request Review and Continuous Integration Builds. *SANER 2019 - Proceedings of the 2019 IEEE 26th International Conference on Software Analysis, Evolution, and Reengineering*, 38–48. https://doi.org/10.1109/SANER.2019.8667996

PUSTAKA LAPORAN PENELITIAN

Boronin, M. (2020). *Hybrid Cloud Migration Challenges A case study at King Master Thesis Mikhail Boronin Supervisor : David Johnson.*

LAMPIRAN

LAMPIRAN 1. Berkas Vagrantfile

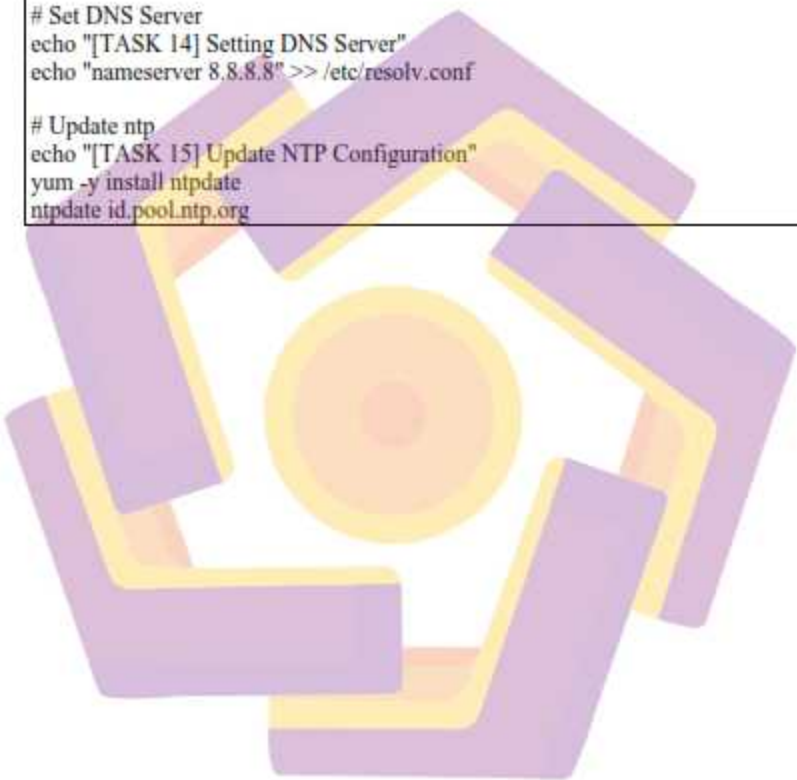
```
ENV['VAGRANT_NO_PARALLEL'] = 'yes'

Vagrant.configure(2) do |config|
 config.vm.provision "shell", path: "bootstrap.sh"

 # Kubernetes Master Server
 config.vm.define "k8s-master" do |kmaster|
  kmaster.vm.box = "centos/7"
  kmaster.vm.hostname = "k8s-master.example.com"
  kmaster.vm.network "public_network", ip: "192.168.1.150"
  kmaster.vm.provider "virtualbox" do |v|
   v.name = "k8s-master"
   v.memory = 4096
   v.cpus = 2
   # Prevent VirtualBox from interfering with host audio stack
   v.customize ["modifyvm", :id, "--audio", "none"]
  end
  kmaster.vm.provision "shell", path: "bootstrap_kmaster_calico.sh"
 end

 NodeCount = 2

 # Kubernetes Worker Nodes
 (1..NodeCount).each do |i|
  config.vm.define "k8s-worker#{i}" do |workernode|
   workernode.vm.box = "centos/7"
   workernode.vm.hostname = "k8s-worker#{i}.example.com"
   workernode.vm.network "public_network", ip: "192.168.1.15#{i}"
   workernode.vm.provider "virtualbox" do |v|
    v.name = "k8s-worker#{i}"
    v.memory = 4096
    v.cpus = 1
    # Prevent VirtualBox from interfering with host audio stack
    v.customize ["modifyvm", :id, "--audio", "none"]
   end
   workernode.vm.provision "shell", path: "bootstrap_kworker.sh"
  end
 end
end
```

LAMPIRAN 2. Berkas bootstrap.sh

```
#!/bin/bash

# Update hosts file
echo "[TASK 1] Update /etc/hosts file"
cat >>/etc/hosts<<EOF
127.0.0.1 `hostname` `hostname`
127.0.0.1 localhost localhost.localdomain localhost4 localhost4.localdomain4
::1    localhost localhost.localdomain localhost6 localhost6.localdomain6

192.168.1.150 k8s-master.example.com k8s-master
192.168.1.151 k8s-worker1.example.com k8s-worker1
192.168.1.152 k8s-worker2.example.com k8s-worker2
EOF

# Install docker from Docker-ce repository
echo "[TASK 2] Install docker container engine"
yum install -y -q yum-utils device-mapper-persistent-data lvm2 > /dev/null 2>&1
yum-config-manager --add-repo
https://download.docker.com/linux/centos/docker-ce.repo > /dev/null 2>&1
yum install -y -q docker-ce >/dev/null 2>&1

# Enable docker service
echo "[TASK 3] Enable and start docker service"
mkdir /etc/docker
cat << EOF > /etc/docker/daemon.json
{
 "exec-opts": ["native.cgroupdriver=systemd"],
 "log-driver": "json-file",
 "log-opts": {
  "max-size": "100m"
 },
 "storage-driver": "overlay2"
}
EOF
mkdir -p /etc/systemd/system/docker.service.d
systemctl enable docker >/dev/null 2>&1
systemctl start docker

# Disable SELinux
echo "[TASK 4] Disable SELinux"
setenforce 0
sed -i --follow-symlinks 's/^SELINUX=enforcing/SELINUX=disabled/'
/etc/sysconfig/selinux
```

```
# Stop and disable firewalld
echo "[TASK 5] Stop and Disable firewalld"
systemctl disable firewalld >/dev/null 2>&1
systemctl stop firewalld

# Add sysctl settings
echo "[TASK 6] Add sysctl settings"
cat >>/etc/sysctl.d/kubernetes.conf<<EOF
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
EOF
sysctl --system >/dev/null 2>&1

# Disable swap
echo "[TASK 7] Disable and turn off SWAP"
sed -i '/swap/d' /etc/fstab
swapoff -a

# Add yum repo file for Kubernetes
echo "[TASK 8] Add yum repo file for kubernetes"
cat >>/etc/yum.repos.d/kubernetes.repo<<EOF
[kubernetes]
name=Kubernetes
baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-x86_64
enabled=1
gpgcheck=1
repo_gpgcheck=1
gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg
   https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg
EOF

# Install Kubernetes
echo "[TASK 9] Install Kubernetes (kubeadm, kubelet and kubectl)"
yum install -y -q kubeadm kubelet kubectl >/dev/null 2>&1

# Start and Enable kubelet service
echo "[TASK 10] Enable and start kubelet service"
systemctl enable kubelet >/dev/null 2>&1
systemctl start kubelet >/dev/null 2>&1

# Enable ssh password authentication
echo "[TASK 11] Enable ssh password authentication"
sed -i 's/^PasswordAuthentication no/PasswordAuthentication yes/'
/etc/ssh/sshd_config
systemctl reload sshd
```

```
# Set Root password
echo "[TASK 12] Set root password"
echo "kubeadmin" | passwd --stdin root >/dev/null 2>&1

# Update vagrant user's bashrc file
echo "[TASK 13] Update vagrant user's bashrc file"
echo "export TERM=xterm" >> /etc/bashrc

# Set DNS Server
echo "[TASK 14] Setting DNS Server"
echo "nameserver 8.8.8.8" >> /etc/resolv.conf

# Update ntp
echo "[TASK 15] Update NTP Configuration"
yum -y install ntpdate
ntpdate id.pool.ntp.org
```



LAMPIRAN 3. bootstrap_kmaster_calico.sh

```
#!/bin/bash

# Initialize Kubernetes
echo "[TASK 1] Initialize Kubernetes Cluster"
kubeadm init --apiserver-advertise-address=192.168.1.150 --pod-network-
cidr=172.16.0.0/16 >> /root/kubeinit.log 2>/dev/null

# Copy Kube admin config
echo "[TASK 2] Copy kube admin config to Vagrant user .kube directory"
mkdir /home/vagrant/.kube
cp /etc/kubernetes/admin.conf /home/vagrant/.kube/config
chown -R vagrant:vagrant /home/vagrant/.kube

# Deploy Calico network
echo "[TASK 3] Deploy Calico network"
su - vagrant -c "kubectl create -f
https://docs.projectcalico.org/v3.16/manifests/calico.yaml"

# Generate Cluster join command
echo "[TASK 4] Generate and save cluster join command to /joincluster.sh"
kubeadm token create --print-join-command > /joincluster.sh
```

LAMPIRAN 4. bootstrap_kworker.sh

```
#!/bin/bash

# Join worker nodes to the Kubernetes cluster
echo "[TASK 1] Join node to Kubernetes Cluster"
yum install -q -y sshpass >/dev/null 2>&1
sshpass -p "kubeadmin" scp -o UserKnownHostsFile=/dev/null -o
StrictHostKeyChecking=no k8s-master.example.com:/joincluster.sh
/joincluster.sh 2>/dev/null
bash /joincluster.sh >/dev/null 2>&1
```