SEKOLAH TINGGI TEKNOLOGI TERPADU NURUL FIKRI

PENERAPAN *CONTINUOUS INTEGRATION/CONTINUOUS DEPLOYMENT* (CI/CD) DALAM PENGEMBANGAN WEB APLIKASI DI LINGKUNGAN KUBERNETES BERBASIS RANCHER KUBERNETES ENGINE

TUGAS AKHIR

MUHAMMAD RIDHO HAFIDZ
0110219051

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TINGGI TEKNOLOGI TERPADU NURUL FIKRI
FEBRUARI 2024

SEKOLAH TINGGI TEKNOLOGI TERPADU NURUL FIKRI

PENERAPAN *CONTINUOUS INTEGRATION/CONTINUOUS DEPLOYMENT* (CI/CD) DALAM PENGEMBANGAN WEB APLIKASI DI LINGKUNGAN KUBERNETES BERBASIS RANCHER KUBERNETES ENGINE

TUGAS AKHIR

Diajukan sebagai salah satu syarat untuk memperoleh gelar Sarjana Komputer (S.Kom)

MUHAMMAD RIDHO HAFIDZ
0110219051

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TINGGI TEKNOLOGI TERPADU NURUL FIKRI
FEBRUARI 2024

HALAMAN PERNYATAAN ORISINALITAS

Skripsi/Tugas Akhir ini adalah hasil karya penulis, dan semua sumber baik yang dikutip maupun dirujuk telah saya nyatakan dengan benar.

NAMA : Muhammad Ridho Hafidz
NIM : 0110219051

Jakarta, 26 Februari 2024

Muhammad Ridho Hafidz

HALAMAN PENGESAHAN

Skripsi/Tugas Akhir ini diajukan oleh:

Nama : Muhammad Ridho Hafidz

NIM : 0110219005

Program Studi : Teknik Informatika

Judul Skripsi : PENERAPAN *CONTINUOUS INTEGRATION/CONTINUOUS DEPLOYMENT* (CI/CD) DALAM PENGEMBANGAN WEB APLIKASI DI LINGKUNGAN KUBERNETES BERBASIS RANCHER KUBERNETES ENGINE

Telah berhasil dipertahankan di hadapan Dewan Penguji dan diterima sebagai bagian persyaratan yang diperlukan untuk memperoleh gelar Sarjana Komputer pada Program Studi Teknik Informatika, Sekolah Tinggi Teknologi Terpadu Nurul Fikri

DEWAN PENGUJI

Pembimbing I (Henry Saptono, S.Si., M.Kom)
Penguji I (Imam Haromain, S.Si, M.Kom)

Ditetapkan di : Jakarta

Tanggal : 26 Februari 2024

KATA PENGANTAR

Dengan segala rahmat dan karunia-Nya, puji syukur penulis panjatkan kepada Tuhan Yang Maha Esa, Allah Subhanahu Wa Ta'ala, yang telah memberikan kekuatan, petunjuk, dan berkah-Nya sehingga penulis dapat menyelesaikan tugas akhir ini. Tugas akhir ini merupakan bagian dari perjalanan penulis dalam meraih gelar Sarjana Komputer di Program Studi Teknik Informatika di Sekolah Tinggi Teknologi Terpadu Nurul Fikri. Penulis menyadari sepenuhnya bahwa tanpa petunjuk dan pertolongan-Nya, penulis tidak akan mampu menyelesaikan perjalanan ini. Oleh karena itu, dalam kesempatan ini, penulis ingin mengungkapkan rasa terima kasih yang mendalam dan penghormatan kepada Tuhan Yang Maha Esa atas segala limpahan rahmat-Nya. Oleh karena itu, penulis mengucapkan terima kasih kepada:

1. Allah Subhanahu Wa Ta'ala, Tuhan yang maha esa, atas segala rahmat dan hidayah-Nya sehingga penulis dapat menyelesaikan tugas akhir ini.
2. Nabi Muhammad Sholallahu 'Alaihi Wa Sallam, sebagai suri tauladan bagi seluruh umat manusia.
3. Bapak, Mama dan adik penulis yang selalu mendoakan kesuksesan penulis dalam menyelesaikan tugas akhir ini.
4. Bapak Dr. Lukman Rosyidi, ST., MM., MT. selaku Ketua Sekolah Tinggi Teknologi Terpadu Nurul Fikri.
5. Ibu Tifanny Nabarian, S.Kom., M.T.I, selaku Kepala Program Studi Teknik Informatika Sekolah Tinggi Teknologi Terpadu Nurul Fikri.
6. Bapak Henry Saptono, S.Si., M.Kom, selaku pembimbing penulis yang dengan penuh dedikasi membimbing dalam penulisan tugas akhir ini.
7. Para Dosen di lingkungan Sekolah Tinggi Teknologi Terpadu Nurul Fikri yang telah membimbing penulis dalam mengejar ilmu yang diberikan.
8. Mentor penulis, Om Arisyi, Om Aziz, Om Safaat dan Rikky yang selalu mampu memberikan arahan serta pengalaman yang luar biasa dalam *Research & Development (RnD)* bidang *cloud computing* serta memberikan dukungan moril kepada penulis untuk menyelesaikan tugas akhir ini.

Semoga tugas akhir ini dapat menjadi wujud kecil dari upaya penulis dalam mengabdikan diri kepada-Nya. Terima kasih kepada semua yang membantu, semoga karyanya bermanfaat dalam pengembangan ilmu pengetahuan, khususnya di bidang Teknologi Informasi.

Jakarta, 26 Februari 2024

Penulis

HALAMAN PERNYATAAN PERSETUJUAN PUBLIKASI TUGAS AKHIR UNTUK KEPENTINGAN AKADEMIS

Sebagai sivitas akademik Sekolah Tinggi Teknologi Terpadu Nurul Fikri, saya yang bertanda tangan di bawah ini:

Nama : Muhammad Ridho Hafidz
NIM : 0110219051
Program Studi : Teknik Informatika
Jenis karya : Tugas Akhir

demi pengembangan ilmu pengetahuan, menyetujui untuk memberikan kepada STT-NF **Hak Bebas Royalti Non Eksklusif** (***Non-exclusive Royalty - Free Right***) atas karya ilmiah saya yang berjudul:

"Penerapan *Continuous Integration/Continuous Deployment* (CI/CD) Dalam Pengembangan Web Aplikasi Di Lingkungan Kubernetes Berbasis Rancher Kubernetes Engine", beserta perangkat yang ada (jika diperlukan). Dengan Hak Bebas Royalti Non Eksklusif ini STT-NF berhak menyimpan, mengalih media/formatkan, mengelola dalam bentuk pangkalan data (*database*), merawat, dan mempublikasikan tugas akhir saya selama tetap mencantumkan nama saya sebagai penulis/pencipta dan sebagai pemilik Hak Cipta.

Demikian pernyataan ini saya buat dengan sebenarnya.

Dibuat di : Jakarta
Pada tanggal : 26 Februari 2024

Yang menyatakan,

(Muhammad Ridho Hafidz)

ABSTRAK

Nama : Muhammad Ridho Hafidz
NIM : 0110219051
Program Studi : Teknik Informatika
Judul : Penerapan *Continuous Integration/Continuous Deployment* (CI/CD) dalam pengembangan web aplikasi di lingkungan Kubernetes berbasis Rancher Kubernetes Engine

Dalam era pengembangan perangkat lunak yang dinamis, *Continuous Integration/Continuous Deployment* (CI/CD) menjadi kunci penting dalam mempercepat siklus hidup pengembangan aplikasi web. Penelitian ini menginvestigasi implementasi *Continuous Integration/Continuous Deployment* (CI/CD) dalam mempercepat proses pengembangan aplikasi web, di lingkungan Kubernetes yang diatur oleh Rancher Kubernetes Engine. Fokus diberikan pada pengembangan *pipeline* CI/CD yang efisien dan pengukuran efektivitasnya dalam memfasilitasi *deployment* yang berkelanjutan. Metode penelitian melibatkan pembuatan *pipeline* CI/CD menggunakan alat otomatisasi Jenkins, yang diintegrasikan dalam infrastruktur Kubernetes Rancher. Kinerja *pipeline* diukur berdasarkan beberapa kriteria, termasuk waktu *build*, kesuksesan *deployment*, dan kemampuan dalam menangani perubahan pada *source code* dan integrasi fitur baru. Hasil penelitian ini menunjukkan bahwa implementasi *pipeline* CI/CD yang dirancang secara signifikan meningkatkan alur kerja pengembangan dengan mengoptimalkan waktu pengembangan, meminimalkan kesalahan manusia, serta memberikan kemampuan responsif terhadap modifikasi kode dan penambahan fitur secara efektif. Temuan ini memberikan wawasan yang dapat dijadikan acuan bagi organisasi dan *developer* dalam mengimplementasikan CI/CD di lingkungan Kubernetes, dengan memanfaatkan Rancher Kubernetes Engine sebagai platform orkestrasi.

Kata Kunci: CI/CD, Kubernetes, Rancher Kubernetes Engine, Jenkins, Aplikasi Web.

ABSTRACT

Name : Muhammad Ridho Hafidz

NIM : 0110219051

Study Program : Informatics Engineering

Title : Implementation of Continuous Integration/Continuous Deployment (CI/CD) in Web Application Development in Kubernetes Environment based on Rancher Kubernetes Engine

In the dynamic era of software development, Continuous Integration/Continuous Deployment (CI/CD) has become a crucial key to accelerating the life cycle of web application development. This research investigates the implementation of Continuous Integration/Continuous Deployment (CI/CD) to expedite the web application development process within a Kubernetes environment managed by Rancher Kubernetes Engine. The focus is on developing an efficient CI/CD pipeline and measuring its effectiveness in facilitating continuous deployment. The research method involves creating a CI/CD pipeline using the Jenkins automation tool, integrated within the Rancher Kubernetes infrastructure. The pipeline's performance is measured based on several criteria, including build time, deployment success, and the ability to handle changes to the source code and the integration of new features. The results of this research indicate that the implemented CI/CD pipeline significantly enhances the development workflow by optimizing development time, minimizing human errors, and effectively providing a responsive capability to code modifications and feature additions. These findings offer insights that can serve as a reference for organizations and developers in implementing CI/CD within a Kubernetes environment, utilizing Rancher Kubernetes Engine as an orchestration platform.

Keywords: *CI/CD, Kubernetes, Rancher Kubernetes Engine, Jenkins, Web Application.*

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB I
PENDAHULUAN

Bab ini akan mengeksplorasi latar belakang masalah, merumuskan permasalahan, menjelaskan pendekatan yang diambil dalam penelitian, merinci tujuan yang diharapkan dari penelitian, mengidentifikasi manfaat yang diperoleh dari hasil penelitian, menyusun batasan masalah sebagai upaya menyederhanakan pembahasan penelitian, dan menyajikan sistematika penulisan sebagai kerangka penelitian yang digunakan dalam laporan penelitian.

1.1 Latar Belakang

Di era digital saat ini, kecepatan dan efisiensi dalam pengembangan perangkat lunak menjadi kunci utama keberhasilan sebuah proyek. *Continuous Integration* (CI) dan *Continuous Deployment* (CD) adalah praktik pengembangan perangkat lunak yang bertujuan untuk meningkatkan kecepatan dan efisiensi tersebut [1]. CI/CD memungkinkan pengembang untuk secara terus-menerus menggabungkan perubahan kode ke dalam repositori pusat dan secara otomatis menerapkan kode tersebut ke lingkungan produksi. Praktik ini membantu dalam mendeteksi dan mengatasi masalah lebih cepat, mengurangi waktu penerapan, dan meningkatkan kualitas perangkat lunak.

Kubernetes sebagai sistem orkestrasi kontainer yang terkemuka, memainkan peran penting dalam mendukung praktik CI/CD, terutama dalam pengembangan dan penerapan aplikasi berbasis web. Kubernetes menawarkan sebuah platform yang terbukti dapat diandalkan, bersifat skalabel, dan efisien dalam menjalankan berbagai aplikasi. Platform ini memfasilitasi manajemen sumber daya, peningkatan kinerja, dan keandalan operasional, sehingga menjadi pilihan ideal untuk pengembangan dan pengelolaan aplikasi modern [2]. Rancher Kubernetes Engine (RKE) adalah distribusi Kubernetes yang serbaguna dan mudah digunakan, yang lebih lanjut mempermudah penerapan dan manajemen Kubernetes, menjadikannya pilihan yang populer bagi banyak pengembang dan organisasi [3].

Penelitian ini berfokus pada penerapan CI/CD dalam pengembangan aplikasi web di lingkungan Kubernetes, dengan penekanan khusus pada penggunaan RKE. Hal ini melibatkan mengatasi tantangan seperti konfigurasi yang rumit dan manajemen sumber daya yang efisien. Tujuannya adalah untuk menyediakan wawasan komprehensif tentang bagaimana CI/CD dapat diterapkan secara efektif dalam konteks ini, mengeksplorasi praktik terbaik, dan menganalisis keuntungan serta tantangan yang ada.

Maka dari itu penulis mengambil judul penelitian tugas akhir sebagai berikut "**PENERAPAN *CONTINUOUS INTEGRATION/CONTINUOUS DEPLOYMENT* (CI/CD) DALAM PENGEMBANGAN WEB APLIKASI DI LINGKUNGAN KUBERNETES BERBASIS RANCHER KUBERNETES ENGINE**".

1.2 Rumusan Masalah

Berdasarkan latar belakang, perumusan masalah pada tugas akhir ini adalah sebagai berikut :

1. Bagaimana rancangan langkah atau *pipeline* untuk penerapan *Continuous Integration/Continuous Deployment* (CI/CD) dalam pengembangan web aplikasi di lingkungan Kubernetes berbasis Rancher Kubernetes Engine?
2. Bagaimana efektivitas dari *pipeline* yang digunakan untuk penerapan *Continuous Integration/Continuous Deployment* (CI/CD) dalam pengembangan web aplikasi di lingkungan Kubernetes berbasis Rancher Kubernetes Engine?

1.3 Tujuan dan Manfaat Penelitian

Tujuan dari penelitian ini adalah sebagai berikut:

1. Merancang langkah atau *pipeline Continuous Integration/Continuous Deployment* (CI/CD) yang efisien dan efektif dalam pengembangan web aplikasi di lingkungan Kubernetes berbasis Rancher Kubernetes Engine.
2. Mengetahui efektivitas dari rancangan *pipeline* untuk penerapan *Continuous Integration/Continuous Deployment* (CI/CD) dalam pengembangan web aplikasi di lingkungan Kubernetes berbasis Rancher Kubernetes Engine.

Manfaat dari penelitian ini adalah sebagai berikut:

1. Implementasi *Continuous Integration/Continuous Deployment* (CI/CD) akan mempercepat siklus pengembangan dengan otomatisasi proses pengujian, pembangunan, dan penerapan, sehingga menghasilkan rilis yang lebih cepat.
2. Penggunaan *Continuous Integration/Continuous Deployment* (CI/CD) dalam lingkungan Kubernetes berbasis Rancher Kubernetes Engine akan meningkatkan efisiensi pengembangan, mengurangi waktu dan usaha yang diperlukan untuk pengaturan dan penerapan aplikasi.

3. Pada bidang keilmuan teknologi informasi penelitian ini dapat memberikan wawasan tentang penggunaan CI/CD dalam pengembangan web aplikasi, khususnya di lingkungan Kubernetes berbasis Rancher Kubernetes Engine (RKE) dan juga tentunya menyediakan pemahaman mengenai bagaimana CI/CD dapat diintegrasikan dalama manajemen container menggunakan Kubernetes.

1.4 Batasan Masalah

Pembatasan masalah dalam penelitian ini adalah sebagai berikut:

1. Penelitian ini akan berfokus pada penerapan *Continuous Integration/Continues Deployment* (CI/CD) untuk aplikasi web sederhana, pengembangan aplikasi yang berbeda seperti aplikasi berbasis mobile atau perangkat lunak desktop tidak akan dibahas.
2. Dalam rancangan pengujian *pipeline* pada penelitian ini, pengujian unit testing dan *security* testing tidak disertakan.
3. Infrastruktur penelitian ini sepenuhnya terletak di dalam lingkungan virtualisasi *server*.

1.5 Sistematika Penulisan

Sistematika Penulisan penelitian mencakup 5 BAB untuk mempermudah pemahaman keseluruhan penelitian. Oleh karena itu, urutan sistematika penulisan tugas akhir pada penelitian ini diuraikan sebagai berikut:

1. BAB I PENDAHULUAN.

   Pada bab ini mencakup penjelasan latar belakang penelitian, rumusan masalah, tujuan dan manfaat penelitian, batasan masalah, serta sistematika penulisan.

2. BAB II KAJIAN LITERATUR.

   Pada bab ini, diuraikan eori-teori pendukung yang digunakan untuk menerapkan *Continuous Integration/Continues Deployment* (*CI/CD)* dalam pengembangan web aplikasi di lingkungan Kubernetes berbasis Rancher Kubernetes Engine.

3. BAB III METODOLOGI PENELITIAN.

   Pada bab ini dijelaskan langkah-langkah yang diperlukan dalam menyusun penelitian, meliputi pendahuluan, landasan teori, jenis metodologi penelitian, prosedur penelitian, tahapan penelitian, lingkungan penelitian, dan *timeline* penelitian.

4. BAB IV IMPLEMENTASI DAN EVALUASI.

   Pada bab ini menguraikan secara terperinci mengenai penerapan konsep, metode, atau sistem yang telah dirancang, serta melibatkan evaluasi terhadap hasil implementasi tersebut.

5. BAB V KESIMPULAN DAN SARAN.

   Pada bab ini disajikan kesimpulan dari penelitian tugas akhir ini beserta saran yang dapat digunakan untuk pengembangan kedepannya.

BAB II
KAJIAN LITERATUR

Bab ini bertujuan untuk menggambarkan alur penelitian dan bagaimana penelitian ini berkaitan dengan penelitian-penelitian yang sudah ada sebelumnya. Selain itu, bab ini juga menjelaskan berbagai teori yang berkaitan tentang penelitian yang penulis lakukan.

2.1 Tinjauan Pustaka

Dalam kerangka penelitian ini, disajikan landasan teori dan konsep-konsep yang akan menjadi dasar pelaksanaan penelitian yang berjudul "Penerapan *Continuous Integration/Continuous Deployment* (CI/CD) Dalam Pengembangan Web Aplikasi Di Lingkungan Kubernetes Berbasis Rancher Kubernetes Engine." Peneliti akan menjelaskan dan menguraikan beberapa aspek yang terkait dengan teori dan konsep penelitian tersebut.

2.1.1 Web Aplikasi

Web Aplikasi adalah program perangkat lunak yang diakses melalui browser web, beroperasi di *server*, dan memungkinkan pengguna mengaksesnya tanpa perlu menginstal perangkat lunak tambahan. Karakteristik umumnya mencakup akses melalui browser, platform agnostik, pengembangan dengan teknologi web, interaksi *real time*, dan tidak memerlukan instalasi lokal. Pengembangan web aplikasi telah cepat mengadopsi teknik rekayasa perangkat lunak yang berorientasi pada komponen dan komponen standar. Sebagai contoh, pencarian, sindikasi, dan penandaan telah menjadi komponen standar dari generasi baru aplikasi dan proses kolaboratif [4]. Perkembangan masa depan dalam aplikasi web akan didorong oleh kemajuan teknologi peramban, infrastruktur internet web, standar protokol, metode rekayasa perangkat lunak, dan tren aplikasi.

2.1.2 *Container*

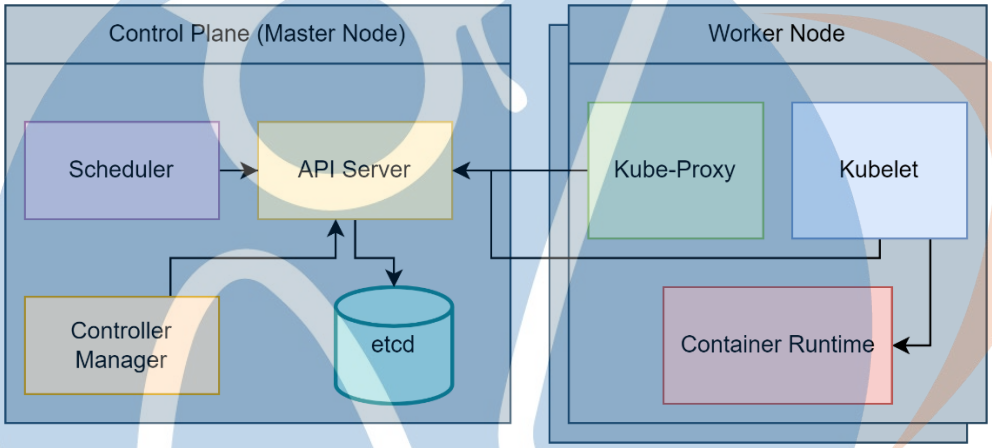
Teknologi *container* telah menjadi pilihan yang sangat popular dan sering digunakan dalam berbagai aplikasi berbasis *cloud.* Keberhasilan *Container* terletak pada kemampuannya untuk beroperasi pada sistem operasi yang sama, meningkatkan efisiensi *server*, dan mempercepat proses penyebaran aplikasi. Dengan menggunakan *container*, aplikasi dapat berjalan secara efisien dalam lingkungan yang konsisten. Manfaat lain dari teknologi *container* adalah peningkatan keandalan perangkat lunak, terutama saat berpindah dari satu IT *environment* ke *environment* lainnya. *Container* dapat membantu mengurangi kompleksitas platform aplikasi, *dependency*, dan strukturnya. Dengan kata lain, dalam pembangunan *container*, semua komponen yang diperlukan untuk menjalankan suatu aplikasi, termasuk aplikasi itu sendiri, *library*, dan *file* konfigurasi, yang dikemas secara menyeluruh ke dalam suatu *environment* yang dapat dijalankan [5].

2.1.3 Kubernetes

Kubernetes, sering kali disebut sebagai K8s, adalah platform sumber terbuka yang dirancang untuk mengotomatisasi, menyebarluaskan, mengelola, dan meningkatkan skalabilitas aplikasi yang dikemas dalam *container* (wadah). Kubernetes banyak digunakan untuk mengelola *container orchestration* dalam aplikasi yang berbasis *cluster*, seperti manajemen Docker *container*, otomatisasi penyebaran aplikasi dan penskalaan sebuah *container*. Selain itu, Kubernetes menyediakan otomatisasi memulai ulang *container* yang mengalami kegagalan serta melakukan penjadwalan ulang *container* jika *host* mengalami kegagalan. Fungsi-fungsi ini secara signifikan meningkatkan meningkatkan *availability* sebuah aplikasi dan memberikan kemudahan *developer* dalam penskalaan, otomatisasi penyebaran aplikasi, dan manajemen aplikasi berbasis *container* [6].

Arsitektur Kubernetes terdiri dari beberapa komponen utama. *API Server* adalah komponen yang menyediakan fungsionalitas utama Kubernetes, sementara *Controller Manager* berfungsi sebagai pengawas utama yang memonitor dan

mengubah status *cluster* melalui *API Server*. Komponen *scheduler*, sebagai bagian dari *control*, bertanggung jawab untuk menjadwalkan *pod* di berbagai *node*. Komponen basis data utama adalah etcd, sebuah *database* berbasis *key-value* yang menyimpan semua informasi konfigurasi terkait *cluster* Kubernetes [6]. Untuk berinteraksi dengan *API server* pada *node master*, *user* dapat menggunakan perintah kubectl. Gambar 2.1 menunjukkan ilustrasi arsitektur Kubernetes.



Gambar 2.1 Ilustrasi Arsitektur Kubernetes

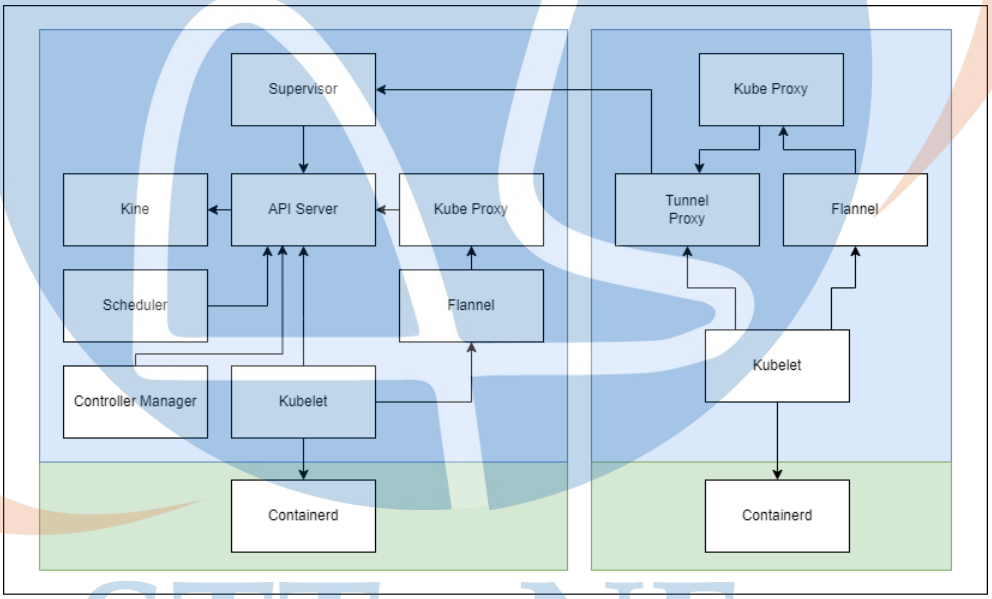
2.1.4 K3s

K3s adalah distribusi ringan dari Kubernetes yang dikembangkan oleh Rancher Labs, dirancang khusus untuk digunakan pada lingkungan dengan sumber daya terbatas seperti *edge computing* atau *cluster* kecil [7]. Meskipun tetap mempertahankan inti fungsionalitas Kubernetes, K3s memiliki ukuran yang lebih kecil dan konsumsi sumber daya yang lebih rendah, menjadikannya ideal untuk skenario *IoT* atau *edge computing*.

Dalam implementasinya, K3s menggantikan komponen penyimpanan data utama Kubernetes, etcd, dengan Kine. Kine merupakan pengganti yang lebih ringan dan sederhana, memberikan keuntungan dalam hal pengurangan ukuran dan kompleksitas. Hal ini penting terutama pada *cluster* yang lebih kecil atau perangkat *edge* dengan sumber daya terbatas, di mana *etcd* mungkin terlalu berat. Meskipun K3s dan Kine menyederhanakan beberapa aspek Kubernetes, keputusan untuk mengadopsi keduanya sebaiknya dipertimbangkan dengan cermat, sesuai dengan

kebutuhan spesifik *cluster* dan lingkungan pengguna [7]. Dengan demikian, pengguna dapat memanfaatkan fleksibilitas dan efisiensi yang ditawarkan oleh solusi ini sesuai dengan skenario penggunaan yang diinginkan. Beberapa komponen penyimpanan pada K3s dihapus dari distribusi Kubernetes K8s untuk menjaga ukurannya tetap kecil. Dalam konteks K3S, *Master node* disebut sebagai *server* dan *worker node* disebut *agent*, yang menggabungkan semua komponen dalam satu proses tunggal. Proses instalasi K3s dapat dilakukan melalui beberapa konfigurasi, salah satunya dengan menggunakan Ansible. Untuk mencapai *high availability* pada K3s, dapat dengan mudah menambahkan *worker node* pada *cluster* dengan beberapa perintah.

Berikut adalaha Gambar 2.2 yang menampilan arsitektur K3s.



Gambar 2.2 Arsitektur Kubernetes (K3s)

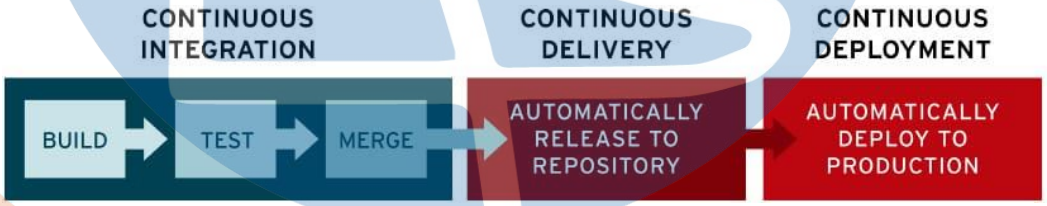
2.1.5 Rancher Kubernetes Engine (RKE)

Rancher Kubernetes Engine (RKE) adalah sebuah alat *open source* yang dikembangkan oleh Rancher Labs, yang dirancang untuk memudahkan pemasangan, konfigurasi, dan manajemen *cluster* Kubernetes. RKE memberikan cara yang lebih sederhana dan terkendali untuk mengelola *cluster* Kubernetes, dan memungkinkan pengguna untuk membuat dan mengelola *cluster* Kubernetes mereka sendiri dengan mudah [8].

Rancher Kubernetes Engine memungkinkan pengguna untuk dengan mudah mengelola dan melakukan *deployment cluster* Kubernetes sesuai kebutuhan mereka, dengan penekanan pada kontrol yang tinggi, pengulangan konfigurasi yang konsisten, dan integrasi yang lebih baik dengan alat dan proses yang ada. Hal ini menjadikan RKE sangat berguna dalam pengaturan, pengujian, dan pengembangan *cluster* Kubernetes untuk aplikasi web atau layanan yang kompleks.

2.1.6 *Continuous Integration/Continuous Deployment* (CI/CD)

Continuous Integration/Continuous Deployment (CI/CD) menjadi bagian integrasi dari pendekatan *DevOps*, menghubungkan tim *developer* dan tim *operations* dalam proses *automation* untuk mengintegrasikan *code*, menguji, dan merilis aplikasi. Dengan menerapkan CI/CD, tim pengembang dapat mengurangi waktu dan usaha yang diperlukan untuk menyatukan dan menguji perubahan yang dilakukan terhadap kode serta integrasi perubahan itu sendiri. CI/CD berfokus pada pengembangan aplikasi yang lebih cepat, otomatis, dan berkelenjutan [9]. Konsep CI/CD dapat dilihat pada Gambar 2.3.



Gambar 2.3 Konsep CI/CD [9]

Berikut ini merupakan konsep utama dari CI/CD.

1. *Continuous Integration* (CI)

   Continuous Integration merupakan suatu pendekatan di mana pengembang secara rutin mengintegrasikan perubahan *code* mereka ke dalam repositori bersama. Dalam CI, setiap kali ada perubahan kode, seperti penambahan fitur, perbaikan *bug*, atau perubahan struktur, secara otomatis akan diintegrasikan dan diuji coba dengan kode yang telah ada. Tujuannya adalah untuk memastikan bahwa perubahan-perubahan tersebut dapat diintegrasikan tanpa

konflik dan dapat diuji dengan cepat. CI membantu meningkatkan kolaborasi tim, meminimalkan resiko konflik, dan meningkatkan kualitas aplikasi.

2. *Continuous Delivery* (CD)

   Continuous Delivery (CD) adalah konsep di mana aplikasi yang telah melalui proses CI dapat secara otomatis dan siap untuk dikirimkan ke lingkungan produksi. Dalam CD, aplikasi yang telah diuji coba secara otomatis dianggap layak produksi sehingga CD akan mengotomatiskan kode yang telah divalidasi tersebut ke repositori. CD bertujuan untuk memastikan basis kode tersebut selalu siap diterapkan ke lingkungan produksi. Setiap tahap produksi mulai dari penggabungan perubahan kode hingga pengiriman kode siap produksi akan melibatkan otomatisasi pengujian dan otomatisasi rilis kode. Di akhir proses tersebut, tim operasi dapat menerapkan aplikasi ke produksi dengan cepat dan mudah.

3. *Continuous Deployment* (CD)

   Continuous Deployment adalah konsep yang lebih maju dari *Continuous Delivery*, di mana perubahan perangkat lunak yang telah melalui proses CI/CD secara otomatis dan langsung dikirimkan ke lingkungan produksi. Dalam *Continuous Deployment*, perubahan kode yang telah melewati tahap CI akan diintegrasikan ke dalam produksi secara langsung tanpa campur tangan manusia. Hal ini memungkinkan pengiriman perubahan aplikasi secara lebih cepat dan efisien. CD akan membantu mempercepat waktu rilis, meminimalkan waktu *downtime*, dan mendorong tim pengembang untuk berfokus pada inovasi dan fitur baru. CD akan mengotomatiskan rilis *build* siap produksi ke repositori kode lalu akan mengotomatiskan rilis aplikasi ke produksi. Setiap perubahan aplikasi dapat ditayangkan dalam beberapa menit setelah uji coba berhasil dilakukan. Hal ini juga akan proses produksi dan rilis lebih mudah dan cepat.

Dengan kata lain, CI/CD memastikan bahwa setiap kali ada perubahan kecil atau besar dalam kode, aplikasi tetap dalam kondisi yang siap untuk dikirimkan ke

lingkungan produksi. Ini tentu saja akan mempercepat proses pengembangan aplikasi.

2.1.8 *Pipeline* CI/CD

Pipeline Continuous Integration/Continuous Deployment (CI/CD) adalah serangkaian langkah otomatis yang diatur untuk mengotomatiskan proses pengembangan perangkat lunak, pengujian, dan penyebaran [10]. *Pipeline* ini memungkinkan tim pengembangan untuk secara cepat dan konsisten menggabungkan perubahan kode, menguji fungsionalitas aplikasi, dan merilis perangkat lunak ke lingkungan produksi [10]. Pemanfaatan CI/CD *pipeline* membantu mempercepat siklus pengembangan, meningkatkan kualitas perangkat lunak, dan meminimalkan risiko kesalahan saat merilis perubahan ke produksi.

2.1.8 Harbor *Registry*

Harbor merupakan proyek *open source* yang menyediakan *registry* Docker dan distribusi *container image* [11]. Sebagai *registry*, Harbor memungkinkan tim *developer* untuk menyimpan dan berbagi *container image* yang telah dibangun, memfasilitasi proses integrasi *Continuous Integration/Continuous Deployment* dalam siklus pengembangan perangkat lunak. Harbor akan menyimpan *container image* yang telah di *build* pada tahap *continuous integrations* [11]. Dengan kata lain, Harbor *Registry* adalah sebuah *registry* Docker yang dikembangkan untuk menyimpan, mengelola, dan mendistribusikan *container image*.

2.1.9 Jenkins

Jenkins merupakan alat otomatisasi *open source* yang digunakan untuk memfasilitasi proses pengembangan perangkat lunak dan otomatisasi *Continuous Integration/Continuous Deployment* (CI/CD). Sebagai *server* otomatisasi, jenkins membantu *developer* dalam mengintegrasikan perubahan kode secara teratur, menguji perangkat lunak, dan secara otomatis merilis aplikasi ke *environment production* atau *environment* lainnya [12].

Dengan bantuan jenkins, pengguna dapat menghindari upaya dan waktu yang besar yang biasanya diperlukan untuk mengonfigurasi ekosistem yang rumit. Jenkins mengelola sebagian besar tugas yang terkait. Jenkins otomatisasi proses *Continuous Integration/Continuous Delivery* (CI/CD) di lingkungan *cloud*, dengan mencari cara untuk menyatukan konfigurasi, plugin, dan kode yang sesuai. Jenkins menyediakan dukungan penuh untuk *Continuous Delivery* serta manajemen produk di berbagai lingkungan, termasuk pratinjau, produksi, dan pementasan.

2.1.10 Github

GitHub merupakan platform pengembangan perangkat lunak berbasis Git yang memfasilitasi kontrol versi distribusi. Menyediakan repositori git berbasis *cloud*, github mempermudah kolaborasi tim *developer* yang tersebar secara geografis. Fitur utamanya mencakup repositori untuk menyimpan proyek, proses *forking* untuk berkontribusi tanpa mempengaruhi repositori utama, *branching* untuk pengembangan fitur tanpa memengaruhi kode utama, *commit* untuk menyimpan perubahan dengan pesan penjelasan, *pull request* untuk menggabungkan perubahan, dan fitur *issues* untuk melacak, diskusikan, dan atribut pekerjaan tertentu [13]. Sebagai platform kolaborasi terbesar, github digunakan oleh komunitas *developer*, organisasi, dan proyek *open source*, memungkinkan manajemen efisien kode sumber dan dokumentasi proyek.

Integrasi github dengan jenkins adalah proses menghubungkan sistem manajemen kode sumber seperti github dengan alat otomatisasi seperti jenkins. Ini memungkinkan untuk membangun, menguji, dan menerapkan perubahan kode secara otomatis setiap kali ada pembaruan dalam repositori github. Integrasi ini berguna dalam pengembangan perangkat lunak berkelanjutan, terutama dalam konteks *Continuous Integration* (CI) dan *Continuous Deployment* (CD). Juga membantu menciptakan aliran kerja pengembangan perangkat lunak yang efisien dan otomatis. Setiap kali ada perubahan kode, jenkins akan secara otomatis memastikan bahwa perubahan tersebut lulus uji dan diterapkan, membantu dalam mencapai CI/CD yang mulus.

2.1.11 Ansible

Ansible merupakan sebuah platform *open source* yang digunakan untuk otomatisasi tugas-tugas IT, manajemen konfigurasi, dan penyebaran aplikasi. Ansible membantu tim *developer* dan tim *operations* untuk mengelola infrastruktur secara efisien dengan menyediakan cara untuk mendefinisikan, men-*deploy*, dan mengelola konfigurasi perangkat lunak dan sistem [14]. Ansible diimplementasikan dengan menggunakan arsitektur *push* yakni dengan *control node* mengirimkan perintah ke *host* target dan memiliki kemampuan untuk mengotomatisasi tugas-tugas dalam lingkungan yang kompleks dan terdistribusi.

Ansible sering dianggap sebagai solusi utama untuk memulai dan mengelola *computer cluster* atau infrastruktur terdistribusi. Platform ini menawarkan pendekatan efisien dalam perancangan, konfigurasi, dan otomatisasi pengelolaan *computer cluster*. Ansible menduduki posisi terdepan dalam manajemen *cluster* dengan menyediakan cara yang konsisten, terdokumentasi, dan dapat diulang untuk mengelola setiap node dalam suatu *cluster*.

2.1.12 Load Balancer Nginx

Load balancer Nginx merupakan proses pendistribusian *traffic* atau lalu lintas jaringan secara efisien ke dalam sekelompok *server*. Load balancing ini berguna agar salah satu *server* dari website yang mendapatkan yang mendapatkan banyak lalu lintas kunjungan tidak mengalami kelebihan beban [15]. Dalam konteks ini Nginx dapat berfungsi sebagai load balancer dengan menggunakan modul khusus yang dirancang untuk menangani distribusi lalu lintas. Load balancer Nginx dapat dikonfigurasi untuk bekerja beberapa mode, seperti *round-robin* (distribusi secara bergantian), *least connection* (mengarahkan lalu lintas ke *server* dengan jumlah koneksi paling sedikit), dan sejumlah metode distribusi lalu listas lainnya [16]. Dengan menggunakan load balancer Nginx, suatu organisasi atau perusahaan dapat meningkatkan dan keandalan layanan yang digunakan, mengelola lalu lintas lebih efisien, serta menyebarkan beban kerja di antara *server-server* yang tersedia sehingga tidak ada satu *server* pun menjadi *bottleneck*.

2.2 Tinjauan Penelitian

Dalam penelitian ini, penulis mendapatkan beberapa inspirasi dari penelitian-penelitian yang terlihat pada Tabel 2.*1* yang berkaitan dengan latar belakang masalah pada penelitian ini. Diantara penelitian terdahulu yang berkaitan dengan latar belakang penelitian penulis adalah:

Penelitian yang dilakukan oleh Naveen Badisa, Grandhi Jayanth Krishna, Lasya Kallam, Reddy Eda Mokshita, Srinivasu Nulaka, Bulla Sunitha (2023) dengan judul "*Efficient Automation of Web Application Development and Deployment Using Jenkins: A Comprehensive CI/CD Pipeline for Enhanced Productivity and Quality*" secara sistematis mengeksplorasi penggunaan Jenkins dalam implementasi *pipeline Continuous Integration* dan *Continuous Deployment* (CI/CD). Studi ini fokus pada bagaimana Jenkins dapat meningkatkan otomatisasi, produktivitas, dan kualitas dalam pengembangan dan penyebaran aplikasi web.

Penelitian yang dilakukan oleh Alde Alanda, H.A. Mooduto, dan Rizka Hadelina (2022), dengan judul "*Continuous Integration and Continuous Deployment (CI/CD) for Web Applications on Cloud Infrastructures"* mendalami penerapan CI/CD dalam pengembangan aplikasi web di infrastruktur awan. CI/CD memungkinkan otomatisasi dan percepatan siklus pengembangan, pengujian, dan penyebaran aplikasi. Infrastruktur awan memberikan fleksibilitas dan skalabilitas tinggi, cocok untuk aplikasi web. Penelitian ini membahas keterkaitan antara CI/CD dan infrastruktur awan, menyoroti pentingnya praktik ini dalam pengembangan aplikasi web yang efisien.

Penelitian yang dilakukan oleh Hardikar Sanjay, Ahirwar Pradeep, Rajan Sameer (2021), dengan judul *"Containerization: Cloud Computing based Inspiration Technology for Adoption through Docker and Kubernetes"* mengenalkan teknologi *containerization* sebagai solusi untuk meningkatkan efisiensi dan fleksibilitas dalam pengembangan serta implementasi aplikasi di lingkungan *cloud computing*.

Tabel 2.1 Penelitian Terkait

No	Nama dan Tahun	Judul	Topik	Subjek	Hasil
1	Naveen B, Grandhi J, Lasya K, Reddy E, Srinivasu N, Bulla S 2023	*Efficient Automation of Web Application Development and Deployment Using Jenkins: A Comprehensive CI/CD Pipeline for Enhanced Productivity and Quality*	Otomatisasi efisien dalam proses pengembangan aplikasi web untuk meningkatkan produktivitas dan kualitas	Penggunaan Jenkins untuk CI/CD, dengan tujuan meningkatkan produktivitas dan kualitas dalam pengembangan perangkat lunak	Pengembangan sebuah *pipeline* CI/CD yang menyeluruh dengan tujuan meningkatkan produktivitas dan kualitas dalam pengembangan aplikasi web
2	Alde Alanda, H.A. Mooduto, dan Rizka Hadelina 2022	*Continuous Integration and Continuous Deployment (CI/CD) for Web Applications on Cloud Infrastructures*	Eksplorasi dan implementasi praktik-praktik CI/CD dalam konteks pengembangan dan pengelolaan aplikasi web yang di-host di *cloud infrastructure* (infrastruktur awan)	Implementasi, manajemen, dan evaluasi praktik CI/CD dalam konteks pengembangan dan pengelolaan aplikasi web yang di-host di *cloud infrastructure*	Impelemtasi Praktik CI/CD untuk *Deployment* web pada *cloud infrastructure*

3	Hardikar Sanjay, Ahirwar Pradeep, Rajan Sameer 2022	*Containerization: Cloud Computing based Inspiration Technology for Adoption through Docker and Kubernetes*	Implementasi teknologi *containerization* dengan Docker dan Kubernetes dalam lingkungan *cloud computing*	Implementasi teknologi *containerization* dengan Docker dan Kubernetes dalam konteks *cloud computing*	Pengenalan teknologi *containerization* untuk meningkatkan efisiensi dan fleksibilitas dalam pengembangan dan implementasi aplikasi di lingkungan *cloud computing*

BAB III
METODOLOGI PENELITIAN

Bab ini berisi tahapan yang dilakukan penulis dalam penelitian diantaranya terdapat jenis metodologi penelitian, tahapan penelitian, lingkungan penelitian, serta timeline penelitian. Jenis metodologi penelitian yakni menjelaskan terkait metode penelitian yang digunakan pada penulisan ini. Tahapan penelitian menjelaskan bagaimana langkah-langkah yang penulis lakukan pada penelitian ini mulai dari awal penelitian hingga akhir penelitian. Lingkungan penelitian menjelaskan terkait penggunaan *environment* penelitian. Sedangkan *timeline* penelitian menunjukkan grafik waktu dari setiap tahapan yang penulis rincikan.

3.1. Jenis Metodologi Penelitian

Pada penelitian ini menggunakan jenis metode penelitian deskriptif kualitatif. Dimana metode penelitian deskriptif kualitatif adalah sebuah metode yang digunakan peneliti untuk menemukan pengetahuan atau teori terhadap suatu penelitian untuk memahami bagaimana interaksi dengan penggunaan teknologi informasi [17]. Dengan penggunaan metode deskriptif kualitatif ini, hasil dari skripsi atau tugas akhir penulis berupa sesuatu yang dapat diukur berdasarkan kualitas yang dihasilkan dari penelitian. Pada penelitian ini, penulis mencari beberapa referensi yang berkaitan dengan penelitian dilakukan, yaitu berkaitan dengan Penerapan *Continuous Integration/Continuous Deployment* (CI/CD) dalam pengembangan web aplikasi di lingkungan Kubernetes berbasis Rancher Kubernetes Engine (RKE).

3.2. Jenis Metode Analisis Data

Penelitian ini menggunakan metode analisis data kualitatif untuk memahami penerapan CI/CD dalam lingkungan Kubernetes RKE. Metode ini melibatkan pengumpulan data melalui observasi, dan analisis dokumen. Data kemudian dianalisis dengan interpretasi data untuk menghasilkan temuan dan

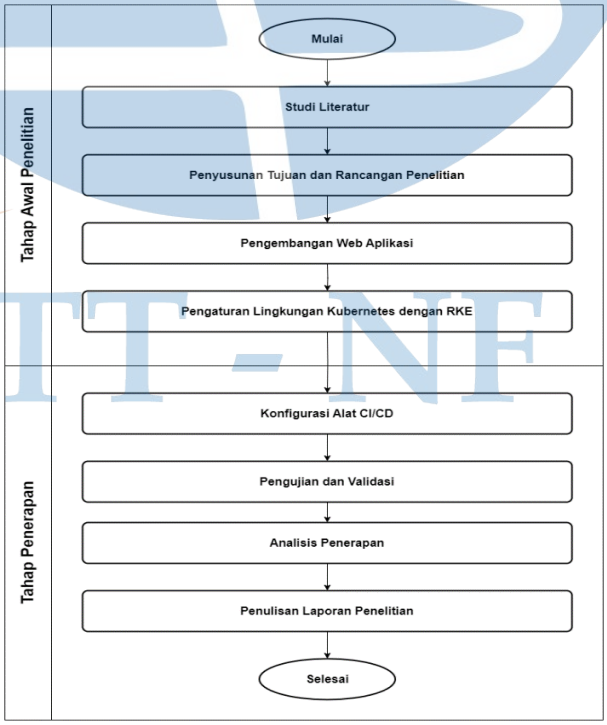
kesimpulan yang mendalam mengenai penerapan CI/CD dalam lingkungan Kubernetes RKE.

3.3. Jenis Metode Pengumpulan Data

Dalam penelitian ini, digunakan metode pengumpulan data melalui studi literatur tujuan untuk mengungkapkan berbagai teori-teori yang relevan dengan permasalahan yang sedang dihadapi/diteliti sebagai bahan rujukan dalam pembahasan hasil penelitian. Sehingga penulis dalam penelitian memahami konsep dasar *Continuous Integration/Continuous Deployment* (CI/CD), Kubernetes, Rancher Kubernetes Engine (RKE), serta praktik dalam pengembangan aplikasi web.

3.4. Tahapan Penelitian

Tahapan penelitian merupakan serangkaian langkah atau proses dalam melakukan sebuah penelitian. Pada bagian ini penulis akan menjabarkan mengenai sebuah tahapan penelitian yang dapat dilihat pada Gambar 3.1.



Gambar 3.1 Tahapan Penelitian

3.4.1. Studi Literatur

Pada tahap ini, penulis melakukan studi literatur untuk memahami konsep dasar CI/CD, Kubernetes, Rancher RKE, serta praktik dalam pengembangan aplikasi web. Hasil dari studi literatur ini menjadi dasar yang kuat bagi penulis untuk mendalami penelitian dan memahami landasan teoritis dan praktik terbaik dalam topik penelitian tersebut.

3.4.2. Penyusunan Tujuan dan Rancangan Penelitian

Pada tahap ini, penulis melakukan penyusunan tujuan dan rancangan penelitian untuk mengarahkan dan merencanakan penelitian lebih lanjut. Dimana menentukan tujuan penelitian yang secara jelas pada penerapan CI/CD dalam lingkungan Kubernetes berbasis RKE serta pembuatan rancangan penelitian yang mencakup metode, alat, teknologi yang akan digunakan, serta kerangka waktu penelitian.

3.4.3. Pengembangan Web Aplikasi

Pada tahap ini, penulis mengembangkan web aplikasi sederhana yang akan disesuaikan atau dioptimalkan untuk penggunaan dalam CI/CD dengan menerapkan praktik pengembangan seperti *containerization* (Docker), manajemen kode sumber (Git), dan dokumentasi aplikasi.

3.4.4. Pengaturan Lingkungan Kubernetes dengan RKE

Pada tahap ini, penulis melakukan konfigurasi *cluster* Kubernetes dengan penerapan teknologi Rancher Kubernetes Engine (RKE). Penerapan teknologi tersebut untuk mengelola *container* yang nantinya akan digunakan untuk menjalankan aplikasi web.

3.4.5. Konfigurasi Alat CI/CD

Pada tahap ini, penulis melakukan konfigurasi untuk penerapan CI/CD dengan menggunakan alat CI/CD seperti Jenkins, dengan tujuan memastikan bahwa

proses pengembangan, pengujian, dan *deployment* aplikasi web di lingkungan Kubernetes berbasis RKE berjalan secara otomatis dan efisien.

3.4.6. Pengujian dan Validasi

Pada tahap ini, penulis melakukan pengujian dan validasi dari hasil penerapan CI/CD dengan tujuan memastikan keandalan dan efisiensi waktu dalam otomasi pada pengembangan web aplikasi menggunakan CI/CD.

3.4.7. Analisis Penerapan

Setelah tahap pengujian dan validasi selesai, penulis melakukan analisis data hasil penerapan teknologi CI/CD dengan tujuan memastikan bahwa penerapan teknologi tersebut dapat memberikan efisiensi, keandalan, serta peningkatan dalam proses pengembangan dan pengelolaan aplikasi web di lingkungan Kubernetes berbasis Rancher Kubernetes Engine (RKE).

3.4.8. Penulisan Laporan Penelitian

Laporan ditulis sebagai bentuk pertanggungjawaban penulis terhadap penelitian yang sudah penulis kerjakan.

3.5. Lingkungan Penelitian

Pada penelitian kali ini, penulis menjalankan penelitian di lokasi infrastruktur *server* sebuah perusahaan sebagai bahan *Research and Development* (RnD).

3.6. Metode Pengujian

Metode pengujian dalam konteks penelitian penerapan *Continuous Integration/Continuous Deployment* (CI/CD) dalam pengembangan web aplikasi di lingkugan Kubernetes berbasis Rancher Kubernetes Engine dapat diuraikan, sebagai berikut ini :

1. Simulasi Perubahan *Code*: Tahap ini melibatkan inisiasi perubahan pada kode sumber atau penambahan data baru dalam repositori lokal. Hal ini meniru perubahan sebenarnya yang dilakukan selama pengembangan aplikasi.
2. Penerapan Perubahan Menggunakan Git Push: Setelah simulasi perubahan kode selesai, perubahan tersebut diimplementasikan ke repositori terpusat melalui perintah git *push*. Langkah ini memicu proses *pipeline* CI/CD, yang secara otomatis akan memulai rangkaian pengujian dan penerapan.
3. Pemantauan Progres *Pipeline* CI/CD: Tahap ini melibatkan observasi dan dokumentasi dari proses dan kemajuan pipeline CI/CD. Tujuannya adalah untuk menilai efektivitas *pipeline* dalam menangani perubahan yang diterapkan, termasuk proses pengujian otomatis, integrasi kode, dan penerapan ke lingkungan produksi atau pengujian.
4. Akses Aplikasi Web: Setelah proses *pipeline* selesai dan aplikasi telah diterapkan, dilakukan verifikasi akhir dengan mengakses aplikasi web tersebut. Ini memastikan bahwa perubahan yang diterapkan berfungsi sebagaimana mestinya di lingkungan produksi atau pengujian.

Melalui metode pengujian ini, peneliti dapat mengevaluasi seberapa efektif *pipeline* CI/CD dalam mengelola dan menerapkan perubahan *code* dalam pengembangan aplikasi web di lingkungan Kubernetes berbasis Rancher Kubernetes Engine.

3.7. *Timeline* Penelitian

Berikut ini merupakan tabel kegiatan timeline penelitian yang direncanakan oleh peneliti dalam melakukan penelitian ini dengan kurun waktu kurang lebih 7 bulan, yang dimulai dari bulan September 2023 sampai bulan Maret 2024. Sebagai berikut :

Tabel 3.1 *Timeline* Penelitian

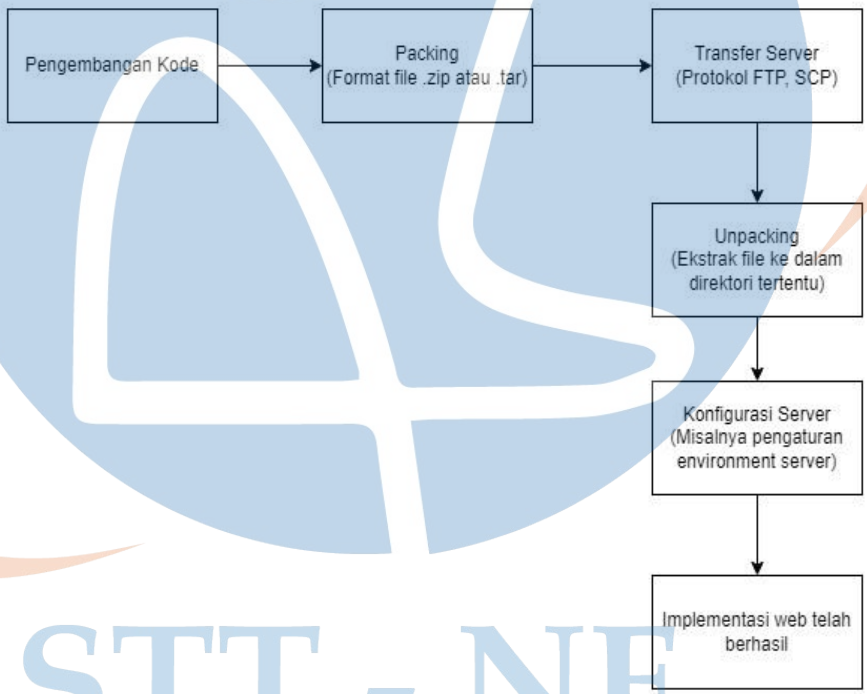
Kegiatan Penelitian		Waktu Kegiatan						
		2023				2024		
No.	Nama Kegiatan	Sept	Okt	Nov	Des	Jan	Feb	Mar
1	Pengajuan Topik dan Judul Tugas Akhir							
2	Menentukan Rumusan Masalah, Tujuan dan Manfaat							
3	Menentukan Batasan Masalah dan Kata Kunci							
4	Perumusan Studi Literatur dan Tinjauan Pustaka							
5	Mencari Penelitian Terkait							
6	Menentukan Metode & Tahapan Penelitian							
7	Melakukan Analisa kebutuhan pada sistem							
8	Persiapan Environment CI/CD pada Kubernetes berbasis RKE							
9	Implementasi CI/CD							
10	Integrasi CI/CD dengan Kubernetes berbasis RKE							
11	Penulisan Laporan Sampai Selesai							
12	Penyusunan dan Penyampaian Presentasi Akhir (Sidang Akhir)							

BAB IV
IMPLEMENTASI DAN EVALUASI

Bab ini membahas secara rinci mengenai penerapan dari suatu konsep, metode, atau sistem yang dirancang. Selain itu, bab ini juga mencakup evaluasi terhadap hasil implementasi tersebut.

4.1. Analisis dan Perancangan Sistem

4.1.1. Analisis Sistem Berjalan



Gambar 4.1 Proses manual *deployment* web ke *Server*

Gambar 4.1 merupakan proses *deployment* aplikasi web yang umumnya masih dilaksanakan secara manual melibatkan beberapa langkah kritikal. Ini dimulai dari tahap pengembangan, di mana kode direncanakan dan diuji untuk memastikan kelayakan sebelum dilepas ke produksi. Kode yang telah selesai dan terverifikasi kemudian dikemas dalam format arsip yang efisien untuk

memfasilitasi transfer ke *server*. Transfer ke *server* dilakukan menggunakan protokol yang aman, seperti FTP atau SCP, untuk menjaga keutuhan data. Setibanya di *server*, proses '*Unpacking*' dilakukan dengan mengekstrak isi paket ke lokasi tertentu, menyiapkan aplikasi untuk dijalankan. Langkah selanjutnya adalah mengonfigurasi *server* sesuai dengan spesifikasi teknis yang dibutuhkan oleh aplikasi, termasuk pengaturan database dan variabel lingkungan. Setelah konfigurasi diverifikasi, aplikasi dianggap siap dioperasikan dan tersedia bagi pengguna.

Untuk meningkatkan efisiensi proses ini, disarankan penerapan praktik *DevOps*, dengan otomatisasi menjadi komponen inti. *Continuous Integration* diadopsi untuk memastikan bahwa setiap integrasi kode memenuhi standar yang telah ditetapkan dan teruji dengan baik. Adopsi pendekatan ini berujung pada peningkatan produktivitas dan kualitas dari aplikasi yang di *deploy*.

4.1.2. Analisis Kebutuhan Sistem

Dalam hal konteks Penerapan *Continuous Integration/Continuous Deployment* (CI/CD) dalam pengembangan web aplikasi di lingkungan Kubernetes berbasis Rancher Kubernetes Engine (RKE), analisis kebutuhan sistem mencakup beberapa aspek yang perlu diperhatikan. Berikut adalah beberapa poin yang dapat menjadi dasar analisis kebutuhan sistem:

1. **Analisis Kebutuhan Perangkat Keras (*Hardware*)**

Tabel 4.1 memberikan detail perangkat keras untuk infrastruktur CI/CD. Jenkins *Server* mengotomatisasi integrasi dan *deployment*, Kubernetes *Cluster* menyediakan *container* untuk aplikasi dengan skalabilitas, Registry *Server* menyimpan *image container*, dan Load Balancer Nginx *Server* mengatur trafik masuk. Semua ini bekerja bersama untuk pengelolaan aplikasi web yang efisien dan aman.

Tabel 4. 1 Perangkat Keras *(Hardware)*

No	Nama Perangkat Keras	Jumlah	Deskripsi
1.	Jenkins *Server*	1	*Server* utama untuk otomatisasi CI/CD, menjalankan tugas integrasi dan *deployment*.
2.	Kubernetes *Cluster*	4	*Node-node* yang membentuk *cluster*, menjalankan aplikasi dalam *container*.
3.	Registry *Server*	1	*Server* untuk penyimpanan dan manajemen *image container*.
4.	Load Balancer Nginx *Server*	1	*Server* yang mengatur beban trafik dan meningkatkan ketersediaan dan responsivitas aplikasi.

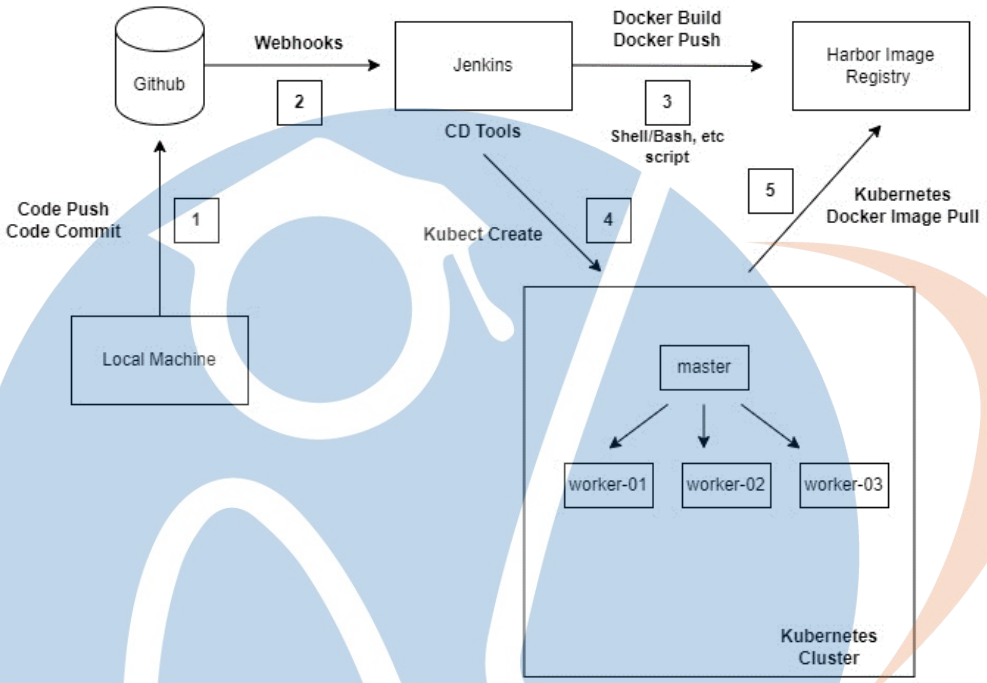
2. Analisis Kebutuhan Perangkat Lunak *(Software)*

Dalam menganalisis kebutuhan perangkat lunak, Tabel 4. *2* memuat daftar perangkat lunak yang diperlukan. Dipilih CentOS *Server* 7 (Core) sebagai sistem operasi untuk *virtual machine*, Rancher Kubernetes Engine 1.4.10 untuk memfasilitasi pemasangan, konfigurasi, dan manajemen *cluster* Kubernetes, serta Jenkins 2.414.3 sebagai alat otomatisasi dalam siklus pengembangan perangkat lunak. Docker 24.0.7 berfungsi sebagai *runtime container* di Kubernetes, Ansible 2.9.27 untuk *deployment* aplikasi ke dalam *environment* RKE, Harbor 2.9.0 untuk penyimpanan *image container*, dan Load Balancer Nginx 1.24.0 untuk mengelola lalu lintas TCP ke beberapa *backend*. Pemilihan perangkat lunak ini dilakukan secara cermat untuk mendukung implementasi infrastruktur CI/CD.

Tabel 4. 2 Perangkat Lunak *(Software)*

No	Nama Perangkat Lunak	Versi	Kegunaan
1.	CentOS *Server*	7 (Core)	Sistem operasi yang digunakan untuk *virtual machine*.
2.	Rancher Kubernetes Engine	1.4.10	Untuk memudahkan pemasangan, konfigurasi, dan manajemen *cluster* Kubernetes.
3.	Jenkins	2.414.3	Alat otomatisasi untuk menyelanggarakan berbagai tahap dalam siklus pengembangan perangkat lunak.
4.	Docker	24.0.7	Sebagai *runtime container* di Kubernetes, ini berfungsi untuk menjalankan aplikasi dalam *container*.
5.	Ansible	2.9.27	Untuk melakukan proses *deployment* aplikasi ke dalam *environment* RKE.
6.	Harbor	2.9.0	Untuk menyimpan *image container* bersama dengan fitur-fitur keamanan dan manajemen akses yang diperlukan.
7.	Load Balancer Nginx	1.24.0	Digunakan untuk untuk mengelola dan mendistribusikan lalu lintas TCP ke beberapa *backend*, membantu meningkatkan ketersediaan, ketahanan, dan skalabilitas sistem.

4.1.3. Perancangan Sistem



Gambar 4.2 Rancangan Arsitektur CI/CD

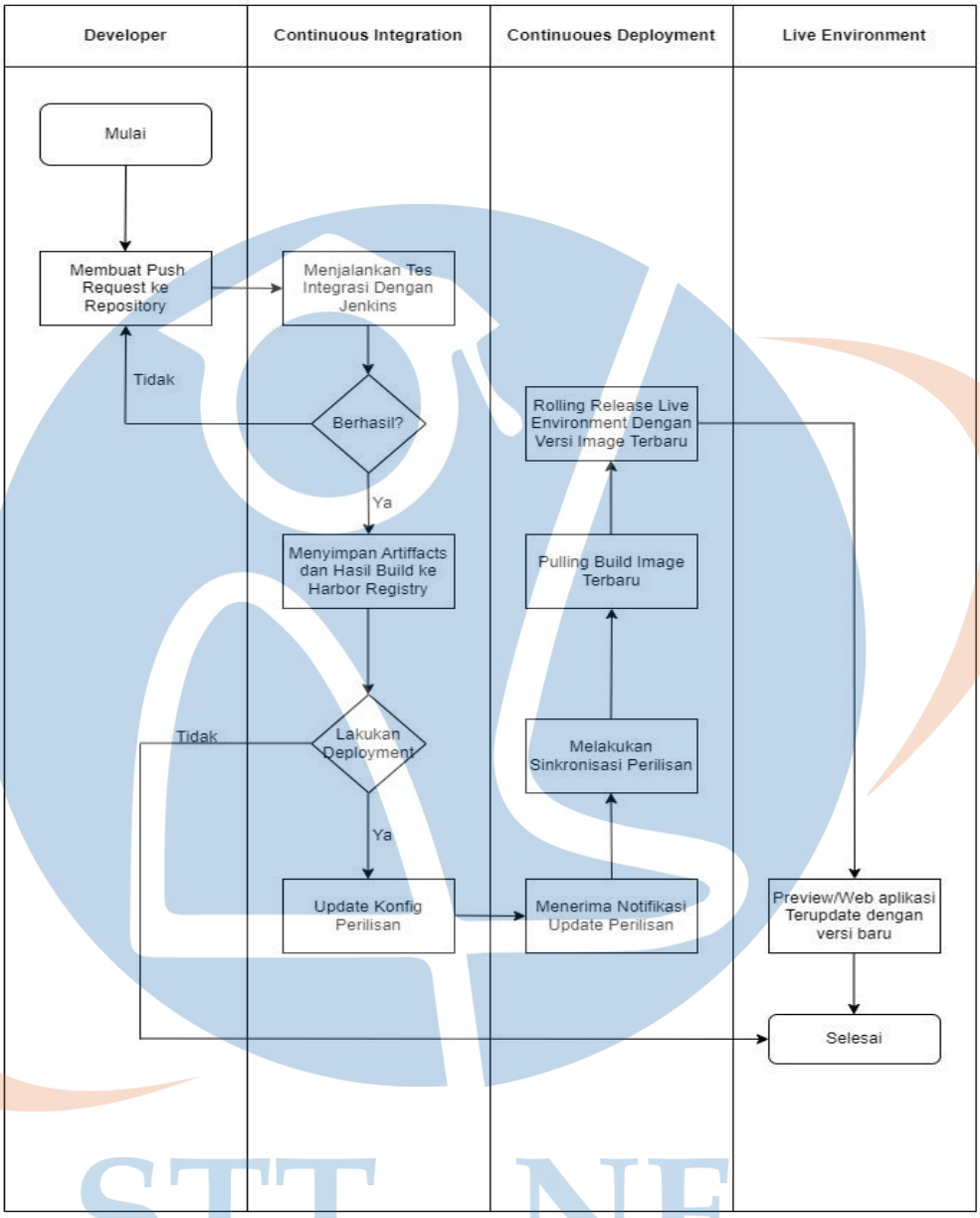
Gambar 4.2 merupakan ilustrasi sebuah rancangan sistem arsitektur *Continuous Integration* (CI) dan *Continuous Deployment* (CD), yang merupakan komponen penting dalam *DevOps* untuk memastikan pengembangan yang efisien dan penyebaran kode yang stabil. Berikut adalah penjelasan dari rancangan tersebut:

1. *Local Machine - Code Push/Commit*: Proses dimulai dengan pengembang yang mengerjakan kode pada mesin lokal. Setelah pengembangan, kode di-*push* (atau di-*commit*) ke repositori kontrol versi seperti GitHub.
2. GitHub - *Webhooks*: Setelah kode di-*push* ke GitHub, sebuah *webhooks* dikirimkan ke server CI (dalam hal ini Jenkins), yang merupakan sinyal otomatis bahwa perubahan telah terjadi. *Webhooks* ini mengaktifkan *pipeline* CI.
3. Jenkins - Docker *Build/Push*: Jenkins, sebagai alat CI, menerima pemicu dari *webhooks* dan menjalankan tugas yang telah ditentukan. Biasanya, ini

meliputi membangun image Docker dari kode yang baru di-*commit* dan kemudian mem-*push image* tersebut ke sebuah *registry image*, yang dalam kasus ini adalah Harbor.

4. CD *Tools* - Kubectl *Create*: Alat CD yang terintegrasi dengan Jenkins (atau mungkin sebagai bagian dari *pipeline* Jenkins itu sendiri) kemudian menggunakan kubectl, sebuah alat baris perintah untuk Kubernetes, untuk membuat atau meng-*update resources* di dalam *cluster* Kubernetes, berdasarkan Docker *image* yang baru dibangun.
5. Harbor *Image Registry* - Kubernetes Docker *Image Pull*: *Cluster* Kubernetes kemudian menarik *image* Docker yang baru dibangun dari Harbor *Image Registry* untuk dikerahkan dalam *cluster*.
6. Kubernetes *Cluster*: Di bagian bawah gambar, *cluster* Kubernetes digambarkan dengan satu master node yang mengelola *cluster* dan beberapa *worker nodes* (dalam diagram ini, tiga: *worker*-01, *worker*-02, dan *worker*-03) yang menjalankan aplikasi yang dikontainerisasi. *Master node* bertanggung jawab untuk mengoordinasikan penyebaran dan pengelolaan *container* pada *worker nodes*.

Setiap komponen dalam rancangan ini berperan penting untuk memastikan bahwa perubahan kode bisa diintegrasikan, diuji, dan diterapkan dengan cepat dan efisien, meminimalisir kesalahan manusia dan mempercepat waktu ke pasar untuk fitur baru.



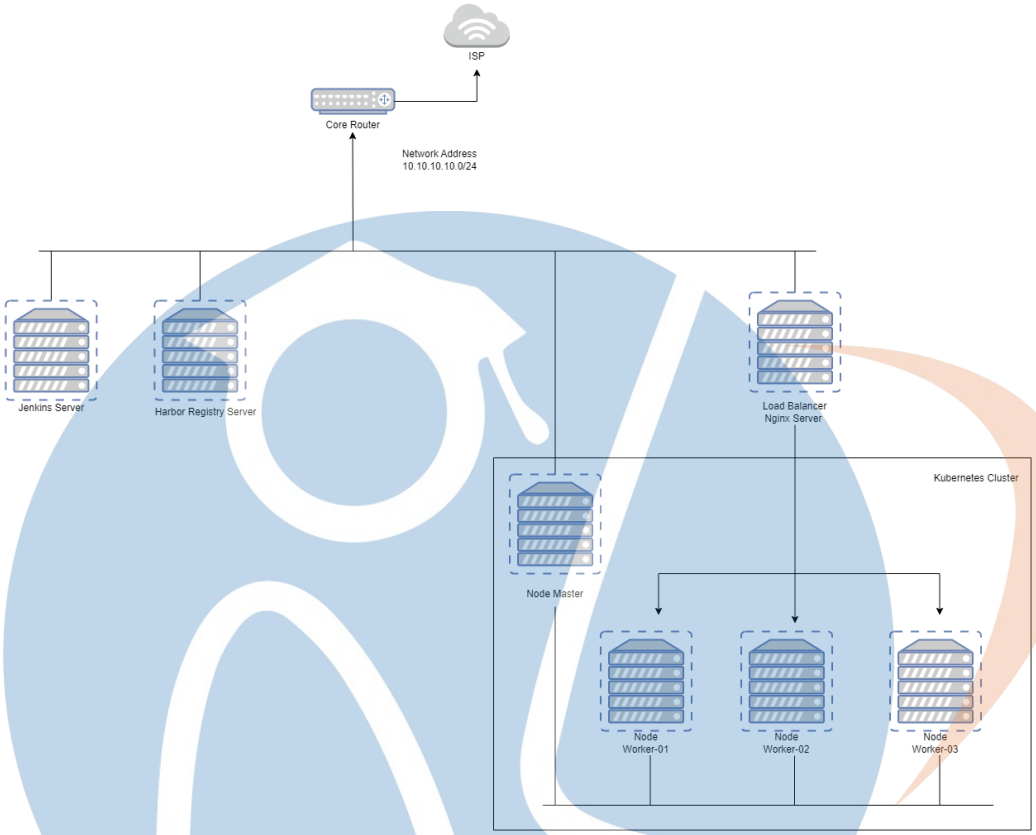
Gambar 4.3 Diagram alur proses perilisan CI/CD Pipeline

Gambar 4.3 memperlihatkan rancangan alur proses *pipeline Continuous Integration/Continuous Deployment* (CI/CD) yang diadopsi dalam penelitian ini. Proses ini diinisiasi oleh pengembang yang mengirimkan perubahan kode ke repositori dengan melakukan *push request*. Setelah itu, sistem *Continuous Integration* memulai serangkaian tes otomatis menggunakan Jenkins, sebuah *server* otomatisasi *open source*, untuk memverifikasi integrasi kode yang baru.

Ketika tes integrasi selesai, hasilnya dievaluasi. Jika tes berhasil (berhasil tanpa kesalahan), maka *artifacts* yang merupakan versi kode yang dapat dieksekusi serta hasil *build* akan disimpan ke dalam Harbor *Registry*, yaitu sebuah *registry* artefak *cloud* yang berfungsi untuk menyimpan dan mendistribusikan versi *build*. Jika tes gagal, *pipeline* menginstruksikan pengembang untuk memperbaiki kesalahan dengan mengirimkan perubahan yang diperlukan kembali ke repositori. Jika berhasil, proses bergerak ke fase *Continuous Deployment*, di mana sistem otomatis melakukan pulling terhadap *build image* terbaru yang disimpan dalam Harbor *Registry* untuk dipersiapkan rilis ke lingkungan live.

Selanjutnya, konfigurasi perlisian disinkronisasikan untuk memastikan bahwa semua pengaturan sistem adalah terkini dan konsisten. Setelah sinkronisasi, sistem mengirimkan notifikasi tentang update perlisian, yang merupakan langkah penting untuk memastikan bahwa tim terkait diberi tahu tentang perubahan yang terjadi. Akhirnya, sebuah *preview* atau versi *live* dari aplikasi dengan *build* terbaru dijalankan untuk validasi akhir. Validasi ini memastikan bahwa aplikasi berfungsi sebagaimana mestinya setelah diperbarui. Proses ini diakhiri dengan penandaan 'Selesai', yang menandakan suksesnya proses rilis dan *deployment* aplikasi.

Dalam konteks penelitian ini, proses CI/CD ini memegang peran vital dalam mempercepat dan meminimalisir kesalahan dalam proses pengembangan dan perlisian aplikasi, yang merupakan tujuan utama dari penerapan metodologi *DevOps*. Dengan menggunakan pendekatan ini, penelitian ini menguji efektivitas otomatisasi *pipeline* dalam pengembangan perangkat lunak.



Gambar 4.4 Rancangan Fisik infrasktruktur CI/CD

Gambar 4.4 menampilkan rancangan fisik dari infrastruktur CI/CD yang telah diimplementasikan. Infrastruktur ini dibagi menjadi beberapa komponen utama, yaitu Jenkins *Server*, Harbor *Registry Server*, Load Balancer yang menggunakan Nginx, dan Kubernetes *Cluster* yang terdiri dari satu *master node* dan tiga *worker nodes*. Jenkins *Server* berfungsi sebagai otomasi *server* untuk *Continuous Integration*, memungkinkan pengembangan dan integrasi kode secara terus-menerus. Harbor *Registry Server* digunakan untuk menyimpan dan mengelola images Docker yang dibuat selama proses CI. Load Balancer Nginx bertugas mendistribusikan trafik masuk ke *cluster* Kubernetes, meningkatkan ketersediaan dan redundansi. Ini memastikan bahwa permintaan dari pengguna disebar ke seluruh *nodes* yang ada untuk menangani beban kerja secara efisien. Kubernetes *Cluster* adalah inti dari infrastruktur *deployment*, dengan satu *master node* yang mengelola *cluster* dan *worker nodes* yang menjalankan aplikasi yang dikerahkan.

Node master berkomunikasi dengan *nodes worker*-01, *worker*-02, dan *worker*-03 untuk mengatur penempatan dan manajemen *container*.

Rancangan ini juga menunjukkan konektivitas jaringan antara komponen, dimulai dari *Internet Service Provider* (ISP) yang terhubung ke *Core Router* dengan *network address* yakni 10.10.10.0/24. Dari *Core Router*, trafik akan terdistribusi ke Jenkins *Server*, Harbor Registry *Server*, dan Load Balancer sebelum akhirnya mencapai Kubernetes *Cluster*. Rancangan fisik ini dirancang untuk menjamin proses CI/CD yang efisien dan otomatis, memungkinkan *deployment* yang cepat dan handal serta memudahkan pengelolaan dan skalabilitas aplikasi.

4.1.4. Perancangan Pengujian

Perancangan pengujian ini bertujuan untuk menguraikan secara sistematis prosedur dan alat yang digunakan, serta hasil yang diperoleh dari proses pengujian tersebut. Tabel 4. *3* menyajikan rancangan pengujian yang telah disusun.

Tabel 4. 3 Rancangan Pengujian CI/CD

No.	Deskripsi Pengujian	Status *Pipeline* CI/CD	Hasil Tayangan pada Server Web
1.	Merubah *source code*	*Build*, dan *deployment* berjalan	Tayangan perubahan tampilan pada web
2.	Menambahkan fitur terbaru pada *source code*	*Build*, dan *deployment* berjalan	Tayangan penambahan tampilan fitur pada web
3.	Integrasi asset berupa *icon image* sebagai fitur tambahan pada *source code*	*Build*, dan *deployment* berjalan	Tayangan penambahan asset pada web

4.2. Impelementasi Sistem

Tahap implementasi adalah salah satu tahap paling penting dalam penelitian ini, karena didalamnya mencakup banyak hal yang merupakan inti dari sebuah penelitian, yakni Penerapan *Continuous Integration/Continuous Deployment* dalam pengembangan web aplikasi di lingkungan Kubernetes berbasis Rancher Kubernetes Engine (RKE). Adapun langkah-langkah pada tahap implementasi adalah sebagai berikut:

4.2.1. Instalasi dan Konfigurasi Kubernetes Berbasis RKE

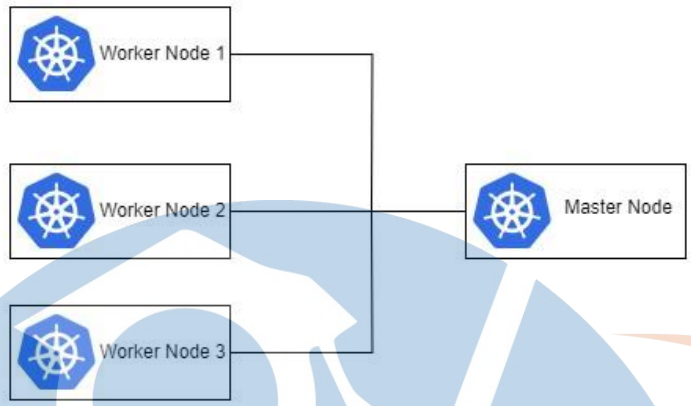
Pada tahapan ini akan dijelaskan langkah-langkah instalasi dan konfigurasi Kubernetes berbasis RKE, sebelum instalasi perangkat lunak kubernetes *cluster* berbasis RKE dibutuhkan persyaratan dengan spesifikasi *virtual machine* yang digunakan:

Tabel 4. 4 Spesifikasi *Virtual Machine*

Operating System	CentOS linux 7
vCpu	4 Core
RAM	8 GB
Disk	60 GB

Pada Tabel 4. *4* menguraikan spesifikasi dari *Virtual Machine* (VM) yang diimplementasikan dalam penelitian ini. VM ini beroperasi pada CentOS Linux 7, dipilih karena keandalannya. Dari segi *hardware*, VM dilengkapi dengan 4 *core vCPU* dan 8 GB *RAM*, kombinasi ini menyediakan kinerja yang solid untuk pengembangan dan pengujian aplikasi. Untuk penyimpanan, VM memiliki *disk* 60 GB, yang cukup untuk menampung sistem operasi, aplikasi, dan data yang diperlukan. Spesifikasi ini memastikan lingkungan yang stabil dan responsif untuk kebutuhan penelitian.

Dan untuk memberikan gambaran yang lebih jelas, maka peneliti akan menjelaskan topologi Kubernetes *cluster* berbasis RKE yang mencakup tiga *node worker* dan satu *node master*.



Gambar 4.5 Topologi Kubernetes Cluster RKE

Peneliti memulai proses instalasi dan konfigurasi *cluster* Kubernetes berbasis RKE dengan langkah-langkah berikut:

- Sebagai langkah pertama, peneliti melakukan pembaruan paket sistem operasi menggunakan perintah sudo yum -y update. Tujuan dari langkah ini adalah untuk memastikan bahwa semua perangkat lunak dan paket yang terinstal pada *node* tetap aman dan terkini.
- Selanjutnya, peneliti membuat pengguna khusus untuk RKE dengan menjalankan serangkaian perintah sudo useradd -m rke, sudo passwd rke, dan sudo usermod -s /bin/bash rke. Langkah ini bertujuan untuk menyediakan akun terpisah yang akan digunakan untuk pengelolaan *cluster*.
- Untuk mendukung otomatisasi dan eksekusi perintah dengan hak istimewa, peneliti mengkonfigurasi akses sudo tanpa kata sandi untuk pengguna RKE. Ini dilakukan dengan mengedit file *sudoers* melalui perintah sudo vim /etc/sudoers.d/rke dan menambahkan rke ALL=(ALL:ALL) NOPASSWD: ALL.
- Peneliti melanjutkan dengan menginstal Docker pada setiap *node* dalam *cluster* sebagai *runtime container*. Proses ini dilakukan dengan menjalankan skrip sebagai berikut:

  ```
  $ curl https://releases.rancher.com/install-
  docker/20.10.sh | sudo bash –.
  ```

- Setelah instalasi, peneliti mengaktifkan dan memulai layanan Docker menggunakan sudo systemctl enable --now docker.
- Selanjutnya dalam tahapan ini adalah menambahkan pengguna RKE ke grup Docker. Peneliti menjalankan sudo usermod -aG docker rke untuk memungkinkan pengguna RKE mengelola *container* Docker.
- Langkah berikutnya yakni peneliti menginstal kubectl untuk memberikan akses dan kontrol terhadap *cluster* Kubernetes dengan serangkaian perintah `curl` dan `chmod` berikut:

```
$ curl -LO https://storage.googleapis.com/kubernetes-release/release/`curl -s https://storage.googleapis.com/kubernetes-release/release/stable.txt`/bin/linux/amd64/kubectl
$ chmod +x ./kubectl
$ sudo mv ./kubectl /usr/local/bin/kubectl
$ kubectl version –client
```

- Selanjutnya, peneliti menginstal RKE sebagai alat utama untuk pembuatan dan pengelolaan *cluster*. Proses ini melibatkan pengunduhan dan konfigurasi RKE melalui perintah terminal berikut:

```
$ curl -s https://api.github.com/repos/rancher/rke/releases/latest | grep download_url | grep amd64 | cut -d '"' -f 4 | wget -qi -
$ chmod +x rke_linux-amd64
$ sudo mv rke_linux-amd64 /usr/local/bin/rke
$ rke –version
```

- Sebagai langkah persiapan, peneliti melakukan instalasi Helm, yang merupakan manajer paket aplikasi untuk Kubernetes. Proses instalasi Helm dilakukan melalui pengunduhan dan eksekusi skrip yang disediakan oleh Helm. Berikut adalah perintah yang digunakan dalam proses instalasi tersebut:

```
$ curl -fsSL -o get_helm.sh
https://raw.githubusercontent.com/helm/helm/master/sc
ripts/get-helm-3
$ chmod 700 get_helm.sh
$ ./get_helm.sh
```

- Lalu langkah berikutnya peneliti menjalankan ssh-keygen -t ed25519 untuk menghasilkan kunci SSH publik dan pribadi. Selanjutnya, kunci publik (/home/rke/.ssh/id_rsa.pub) disalin ke setiap *node master* dan *worker* dalam *cluster*. Dan kemudian peneliti memverifikasi bahwa pengaturan kunci SSH telah berhasil dan pengguna RKE dapat mengakses setiap *node* tanpa kata sandi.
- Selanjutnya peneliti membuat file konfigurasi yang akan digunakan oleh RKE untuk *deployment cluster*. Peneliti menjalankan rke config --name cluster.yml dan mengisi detail seperti jumlah host, alamat IP, pengguna SSH, dan lainnya sesuai kebutuhan *cluster* seperti berikut:

```
$ rke config --name cluster.yml
  [+] Cluster Level SSH Private Key Path [~/.ssh/id_rsa]:
  ~/.ssh/id_ed25519
  [+] Number of Hosts [1]: 4
  [+] SSH Address of host (1) [none]: 10.10.10.158
  [+] SSH Port of host (1) [22]: 22
  [+] SSH Private Key Path of host (10.10.10.158) [none]:
  ~/.ssh/id_ed25519
  [+] SSH User of host (10.10.10.158) [ubuntu]: rke
  [+] Is host (10.10.10.158) a Control Plane host (y/n)? [y]: y
  [+] Is host (10.10.10.158) a Worker host (y/n)? [n]: n
  [+] Is host (10.10.10.158) an etcd host (y/n)? [n]: y
  [+] Override Hostname of host (10.10.10.158) [none]: lab-
  master01.rke.ridhohafidz.com
  [+] Internal IP of host (10.10.10.158) [none]: 10.10.10.158
  [+] Docker socket path on host (10.10.10.158)
  [/var/run/docker.sock]: /var/run/docker.sock
  [+] Network Plugin Type (flannel, calico, weave, canal, aci)
  [canal]: calico
  [+] Authentication Strategy [x509]:
```

```
[+] Authorization Mode (rbac, none) [rbac]:
[+] Kubernetes Docker image [rancher/hyperkube:v1.22.6-
rancher1]: rancher/hyperkube:v1.26.8-rancher1
[+] Cluster domain [cluster.local]: rke.ridhohafidz.com
[+] Service Cluster IP Range [10.43.0.0/16]:
[+] Enable PodSecurityPolicy [n]:
[+] Cluster Network CIDR [10.42.0.0/16]:
[+] Cluster DNS Service IP [10.43.0.10]:
[+] Add addon manifest URLs or YAML files [no]:
```

- Kemudian peneliti menjalankan rke up untuk memulai proses deployment *cluster*. Selama proses ini, peneliti memantau output untuk memastikan bahwa tidak ada kesalahan yang terjadi.
- Selanjutnya peneliti menambahkan path kubeconfig ke file ~/.bashrc dengan baris export KUBECONFIG=./kube_config_cluster.yml bertujuan untuk memudahkan akses ke *cluster* kubernetes melalui antarmuka baris perintah.
- Setelah proses deployment *cluster* kubernetes selesai, langkah selanjutnya adalah memverifikasi dan memastikan bahwa kluster beroperasi dengan baik. Peneliti melakukan verifikasi ini dengan menggunakan perintah berikut untuk memperoleh informasi status dari semua node.

```
$ kubectl get nodes
NAME                                STATUS   ROLES                AGE   VERSION
lab-master01.rke.ridhohafidz.com    Ready    controlplane,etcd    56d   v1.26.8
lab-worker01.rke.ridhohafidz.com    Ready    etcd,worker          56d   v1.26.8
lab-worker02.rke.ridhohafidz.com    Ready    etcd,worker          56d   v1.26.8
lab-worker03.rke.ridhohafidz.com    Ready    etcd,worker          56d   v1.26.8
```

output perintah tersebut menunjukkan bahwa ada satu *node master* (lab-master01.rke.ridhohafidz.com) dan tiga *node worker* (lab-worker01.rke.ridhohafidz.com, lab-worker02.rke.ridhohafidz.com, dan lab-worker03.rke.ridhohafidz.com), semua dengan status `Ready` dan menjalankan versi `v1.26.8`.

4.2.2. Instalasi dan Konfigurasi Jenkins

Sebagai bagian dari penelitian, peneliti mengimplementasikan Jenkins, sebuah alat otomatisasi *open source*, untuk mengelola dan mengotomatisasi proses pengembangan perangkat lunak. Berikut adalah langkah instalasi dan konfigurasi Jenkins yang dilakukan oleh peneliti:

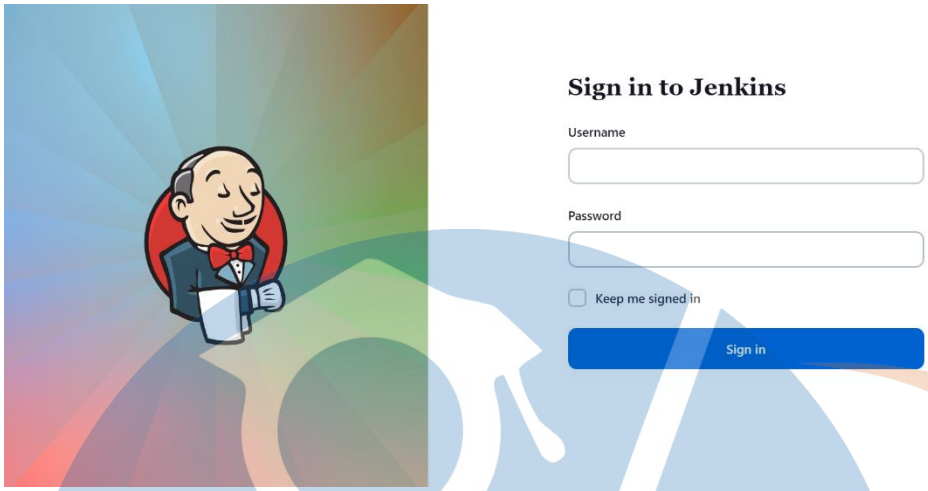
- Langkah awal memperbarui semua paket pada sistem dengan menjalankan sudo yum update. Hal ini memastikan bahwa sistem memiliki semua pembaruan keamanan dan pemeliharaan terkini.
- Selanjutnya peneliti menambahkan repositori Jenkins ke sistem dengan menggunakan perintah wget untuk mengunduh dan menyimpan file repositori ke direktori /etc/yum.repos.d/. Kunci GPG dari repositori Jenkins diimpor ke sistem untuk memastikan integritas dan keaslian paket yang diunduh. Kemudian peneliti menjalankan sudo yum upgrade untuk memperbarui paket-paket sistem sesuai dengan repositori yang baru ditambahkan.
- Langkah berikutnya peneliti menginstal Java OpenJDK versi 11 dan Ansible, yang merupakan prasyarat untuk instalasi dan operasi Jenkins. Berikut perintah yang digunakan untuk instalasi dependesi.

  ```
  $ sudo yum install java-11-openjdk ansible
  ```

- Setelah menyelesaikan langkah sebelumnya, peneliti melanjutkan dengan instalasi Jenkins dengan menjalankan perintah berikut:

  ```
  $ sudo yum install Jenkins
  ```

- Selanjutnya peneliti mengaktifkan Jenkins untuk memastikan layanan tersebut akan berjalan pada saat boot dengan sudo systemctl enable jenkins. Layanan Jenkins dijalankan dengan sudo systemctl start jenkins untuk memulai operasi Jenkins.
- Setelah menjalankan serangkaian perintah instalasi, peneliti berhasil menginstal antarmuka Jenkins pada sistem.



Gambar 4.6 Antarmuka Pengguna Grafis (GUI) Jenkins

4.2.3. Instalasi dan Konfigurasi Harbor *Registry*

Dalam rangkaian penelitian ini, peneliti memfokuskan pada penerapan Harbor *registry*, yang merupakan platform *registry container* bersifat *open source*. Harbor dipilih karena kemampuannya yang luas dalam manajemen dan otomatisasi *container*, yang menjadi kunci dalam proses pengembangan perangkat lunak yang efisien. Implementasi Harbor ini bertujuan untuk memperkuat infrastruktur pengembangan dengan menyediakan solusi penyimpanan *container* yang aman, dapat diandalkan, dan mudah diintegrasikan dengan berbagai alat CI/CD. Berikut adalah detil proses instalasi dan konfigurasi Harbor yang dijalankan oleh peneliti:

- Langkah awal Sebelum instalasi Docker, adalah penting untuk memastikan sistem operasi berada dalam keadaan terbaru dan aman. Ini mencakup pembaruan keamanan dan perbaikan *bug* yang esensial. Peneliti menjalankan perintah sudo yum -y update pada terminal. Perintah ini memperbarui paket-paket yang ada ke versi terbaru yang tersedia
- Setelah itu peneliti menggunakan skrip instalasi yang disediakan oleh Rancher memudahkan proses instalasi Docker, memastikan bahwa semua komponen yang dibutuhkan terpasang secara otomatis. Peneliti menjalankan curl https://releases.rancher.com/install-docker/20.10.sh | sudo bash - Skrip ini diunduh dan dieksekusi, yang memulai proses instalasi Docker.

- Langkah berikutnya Penting untuk mengaktifkan Docker agar secara otomatis berjalan saat sistem dihidupkan. Ini memastikan bahwa layanan Docker selalu tersedia. Peneliti menggunakan sudo systemctl enable --now docker, yang tidak hanya mengaktifkan Docker tetapi juga memulainya secara langsung.
- Setelah itu untuk menghindari kebutuhan penggunaan *superuser privilege* secara terus-menerus, yang dapat menimbulkan risiko keamanan, peneliti menambahkan pengguna ke grup Docker dengan menjalankan sudo usermod -aG docker $USER dan kemudian memverifikasi keanggotaan grup dengan id $USER. Keanggotaan grup ini memungkinkan pengguna menjalankan perintah Docker tanpa perlu hak akses *superuser*.
- Langkah selanjutnya peneliti mengunduh installer Harbor terbaru menggunakan metode pengunduhan otomatis melalui perintah curl sebagai berikut:

```
$ curl -s
https://api.github.com/repos/goharbor/harbor/releases
/latest | grep browser_download_url | cut -d '"' -f 4
| grep '\.tgz$' | wget -i -
```

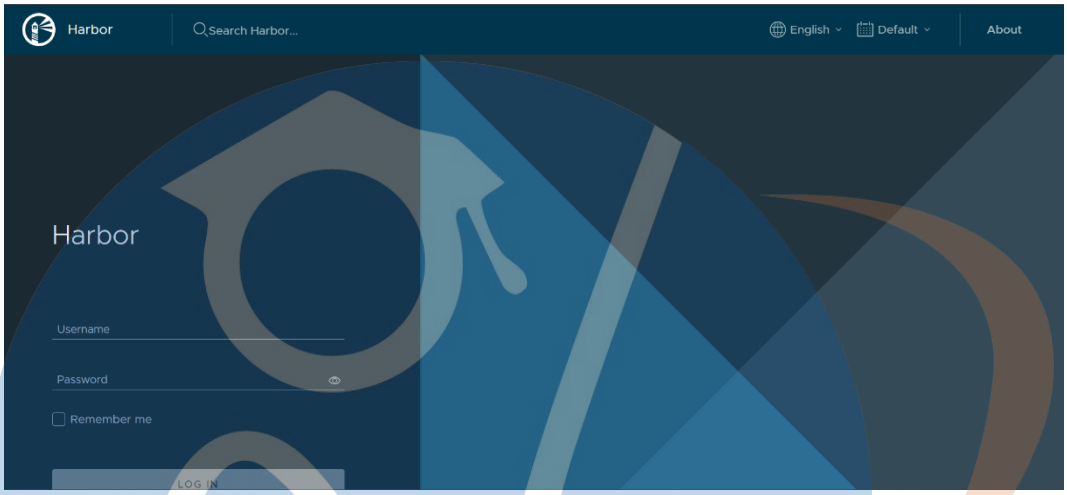
- Untuk menyiapkan installer, file .tgz yang diunduh diekstrak, memberikan peneliti akses ke skrip instalasi dan file konfigurasi sampel dengan perintah berikut:

```
$ tar xvzf harbor-offline-installer*.tgz
```

- Peneliti kemudian melakukan kustomisasi file harbor.yml. Modifikasi ini mencakup penentuan sertifikat SSL untuk mengamankan komunikasi dengan registry dan pengaturan kata sandi administrator. Setiap parameter dalam file konfigurasi disesuaikan dengan kebutuhan spesifik penelitian, termasuk konfigurasi jaringan dan pengaturan keamanan.
- Setelah konfigurasi selesai, peneliti menjalankan skrip instalasi Harbor. Proses ini meliputi inisialisasi berbagai komponen dan layanan yang diperlukan oleh Harbor. Untuk perintah yang digunakan sebagai berikut:

```
$ ./install.sh
```

- Setelah menjalankan proses instalasi, peneliti memverifikasi keberhasilan instalasi dengan mengakses antarmuka pengguna grafis (GUI) Harbor melalui web browser.



Gambar 4.7 Tampilan Antarmuka Registry Harbor

4.2.4. Instalasi dan Konfigurasi Load Balancer Nginx

Selanjutnya peneliti menjelaskan secara rinci proses instalasi dan konfigurasi nginx, yang digunakan sebagai TCP load balancer dalam lingkungan *cluster*. Nginx dipilih karena kemampuannya yang efektif dalam mengelola beban lalu lintas jaringan dan menyediakan distribusi yang merata ke berbagai *server worker*. Berikut adalah langkah-langkah yang diambil:

- Langkah awal peneliti menginstal yum-utils menggunakan perintah sudo yum install yum-utils. Utilitas ini penting untuk mengelola paket dan repositori pada sistem CentOS.
- Selanjutnya peneliti membuat file konfigurasi repositori di /etc/yum.repos.d/nginx.repo. Dalam file ini, peneliti menentukan dua repositori: nginx-stable dan nginx-mainline. Masing-masing repositori dikonfigurasi dengan URL, kunci GPG, dan pengaturan lainnya. Repositori *stable* diaktifkan sementara *mainline* dinonaktifkan, memberikan akses ke versi nginx yang stabil dan teruji.

- Langkah berikutnya Peneliti menjalankan sudo yum install nginx -y untuk menginstal nginx. Proses ini mengunduh dan menginstal versi terbaru Nginx dari repositori yang telah dikonfigurasi.
- Pada tahap selanjutnya setelah berhasil melakukan instalasi nginx, penelitian ini melangkah ke proses konfigurasi file nginx, yang terletak pada direktori /etc/nginx/nginx.conf. Tujuan utama dari konfigurasi ini adalah untuk mencapai distribusi beban lalu lintas jaringan yang efisien dan merata ke berbagai *server worker*. Proses konfigurasi ini dilakukan dengan mengatur parameter-parameter tertentu di dalam file konfigurasi nginx. Berikut adalah detil dari isi file konfigurasi yang digunakan untuk load balancer pada nginx:

```
stream {
    upstream stream_backend_80 {
       server 10.10.10.100:31319;
       server 10.10.10.91:31319;
       server 10.10.10.39:31319;
    }
    upstream stream_backend_443 {
       server 10.10.10.100:32712;
       server 10.10.10.91:32712;
       server 10.10.10.39:32712;
    }
    server {
        listen     80;
        proxy_pass stream_backend_80;
    }
    server {
        listen     443;
        proxy_pass stream_backend_443;
    }
```

- Setelah berhasil mengonfigurasi nginx sebagai load balancer, langkah selanjutnya yang dilakukan peneliti adalah mengaktifkan layanan nginx untuk memastikan bahwa layanan tersebut otomatis berjalan pada saat sistem dihidupkan, serta memulai layanan tersebut secara langsung. Ini dilakukan dengan menjalankan perintah sudo systemctl enable --now nginx, yang tidak

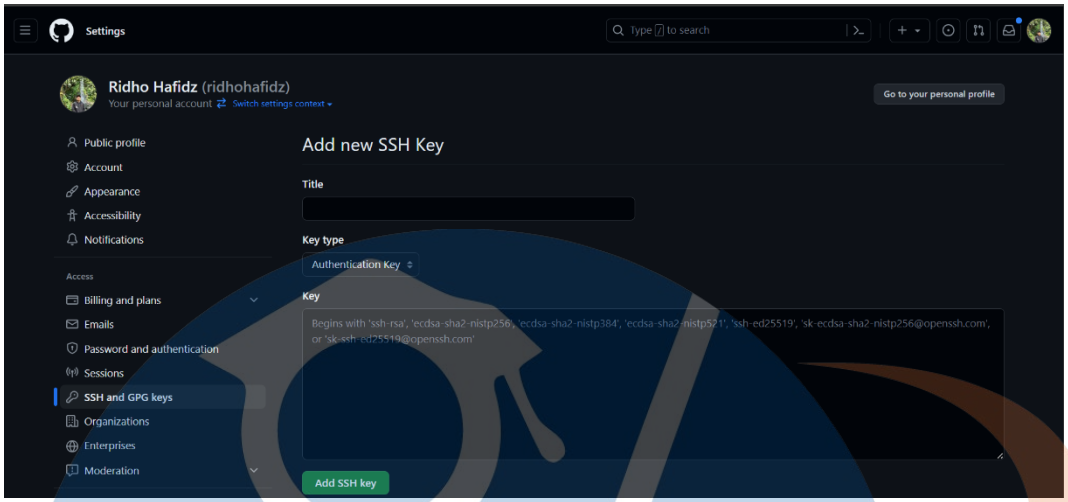
hanya mengatur nginx untuk aktif saat *boot*, tapi juga segera memulai layanan nginx, memastikan bahwa load balancer siap beroperasi.

4.2.3. Implementasi *Pipeline Continuous Integration/Continuous Deployment* (CI/CD)

Sebagai bagian dari proses implementasi *pipeline* CI/CD, langkah awal yang dilakukan adalah inisialisasi dan konfigurasi *server* Jenkins. Inisialisasi ini melibatkan pembuatan kunci SSH yang akan digunakan untuk otentikasi yang aman antara *server* Jenkins dan *repository* GitHub. Pembuatan SSH *key* ini dilakukan melalui eksekusi perintah berikut pada terminal *server* Jenkins:

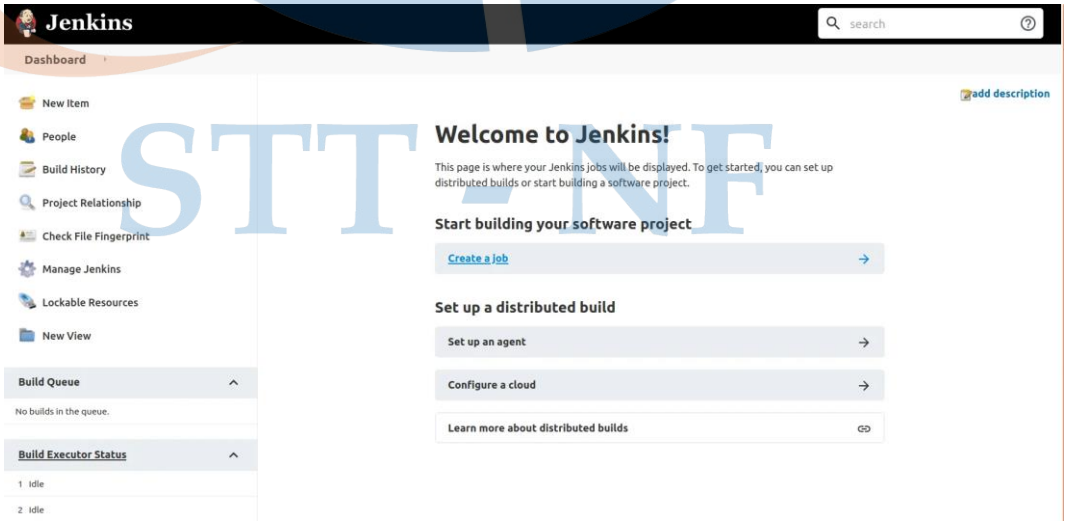
```
$ ssh-keygen -t ed25519
```

Perintah ini menghasilkan sepasang *key*, yaitu *private key* dan *public key*. *Public key* (id_ed25519.pub) yang dihasilkan perlu ditambahkan ke dalam akun GitHub untuk memungkinkan komunikasi yang terautentikasi. Proses penambahan kunci publik ke GitHub dilakukan dengan mengakses pengaturan SSH and GPG *keys* melalui antarmuka pengguna GitHub, kemudian memilih *'New SSH key'* dan menempelkan *public key* yang telah digenerasi. Langkah ini merupakan prasyarat untuk memungkinkan *server* Jenkins melakukan operasi git, seperti *clone* dan *pull*, terhadap *repository* tanpa perlu otentikasi manual. Dengan demikian, konfigurasi ini mendukung otomasi proses CI/CD yang membutuhkan interaksi yang aman dan berulang dengan *repository* kode sumber pada GitHub. Prosedur ini diilustrasikan pada Gambar 4.8, yang menunjukkan *interface* pengguna GitHub saat penambahan *public key*.



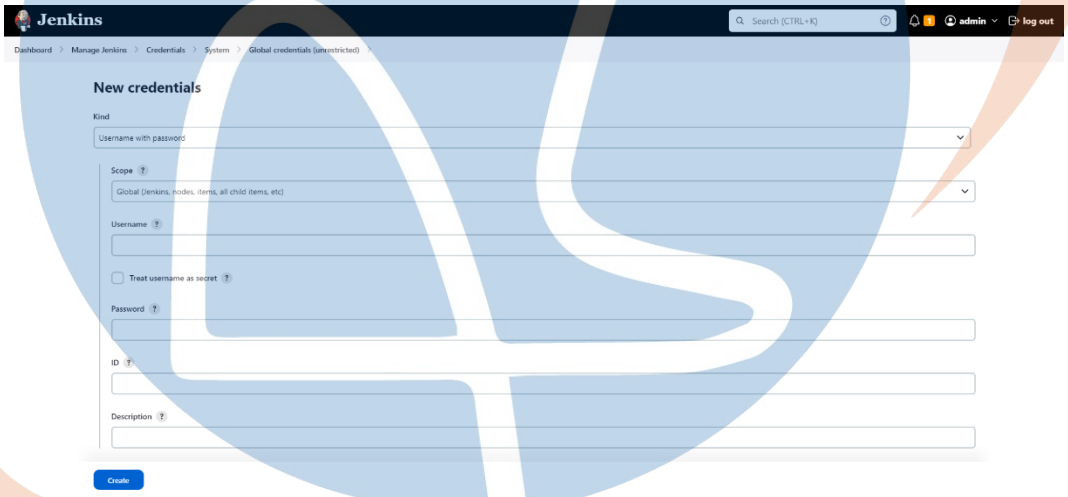
Gambar 4.8 Penambahan Kunci Publik SSH pada Pengaturan GitHub

Penelitian ini melibatkan konfigurasi *pipeline* CI/CD menggunakan Jenkins, dimana dua tipe proyek disediakan yang terdapat *Freestyle*, yang menonjol dengan kemudahan konfigurasinya, dan *Pipeline*, yang ditandai dengan alur kerja terdefinisi yang mendukung integrasi berbagai agen dan tahapan untuk proses *build*, *test*, dan *deployment*. Gambar 4.9 merupakan *interface* jenkins, yang dapat diakses pasca-autentikasi, menyajikan *dasboard* yang memberikan ikhtisar lengkap dan performa dari *pipeline* yang dikonfigurasi, memfasilitasi pemantauan dan navigasi proyek yang efektif.



Gambar 4.9 Halaman utama atau *Dashboard* Jenkins

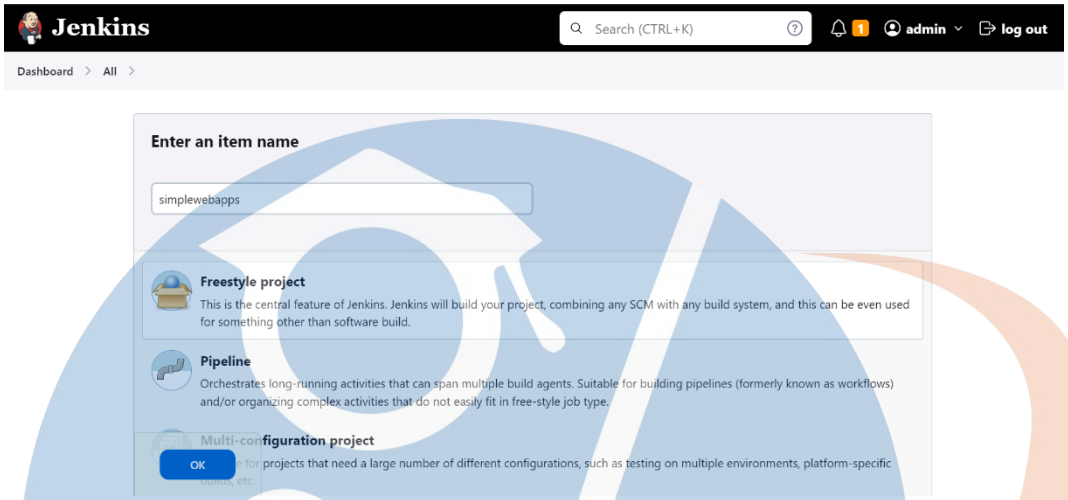
Setelah integrasi kunci publik SSH dengan akun GitHub, langkah selanjutnya adalah mengonfigurasi kunci *privat* pada sistem Jenkins. Proses ini melibatkan akses ke menu administratif Jenkins dengan menavigasi melalui *'Managed Jenkins'*. Dari sini, pengguna harus memilih *'Credentials'* yang diikuti dengan *'System'* dan kemudian *'Global credentials'* untuk mencapai area di mana kunci SSH *privat* dapat ditambahkan ke konfigurasi Jenkins. Langkah-langkah ini diilustrasikan pada Gambar 4.10, yang menunjukkan hierarki menu dan prosedur penambahan kunci. Penambahan kunci *privat* ini memungkinkan Jenkins untuk melakukan akses ke repositori GitHub secara aman dan melakukan operasi SCM seperti *clone*, *fetch*, dan *pull*, yang merupakan komponen penting dari proses *Continuous Integration* dan *Continuous Deployment*.



Gambar 4.10 Penambahkan ssh *private-key* di Jenkins

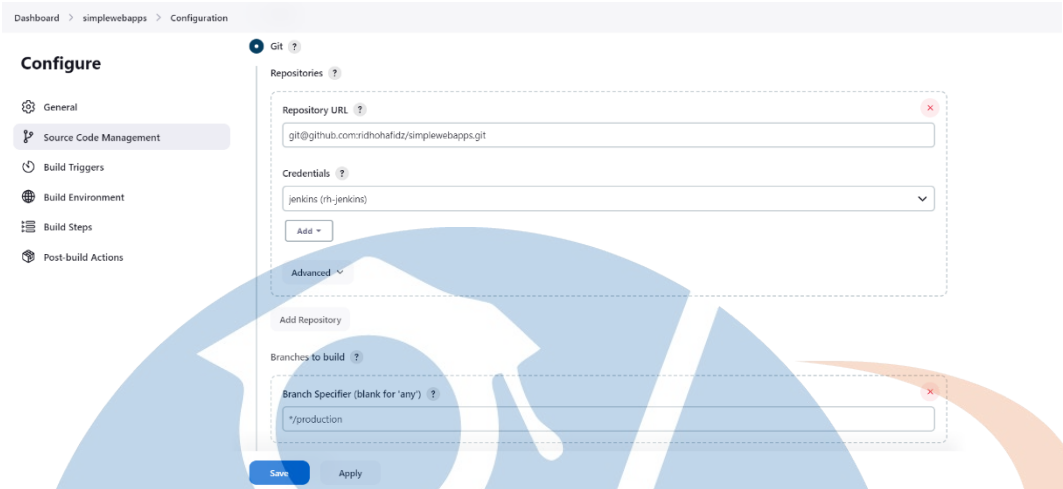
Pada tahapan selanjutnya, proses inisialisasi proyek baru dalam Jenkins diawali dengan memilih opsi *'New Item'* yang terletak pada bagian atas kiri antarmuka pengguna. Pengguna diinstruksikan untuk menginputkan nama yang akan diberikan pada proyek yang dibuat. Setelah penamaan proyek selesai, pengguna kemudian akan memilih jenis proyek *'Freestyle'* dari daftar opsi yang tersedia. Penyelesaian proses ini dilakukan dengan mengeklik tombol *'OK'*. Gambar 4.11 merupakan proyek *'Freestyle'* dalam Jenkins memungkinkan pengguna untuk

memiliki konfigurasi yang sederhana namun fleksibel, yang dapat mendukung berbagai jenis tugas otomasi tanpa memerlukan *script pipeline* yang kompleks.



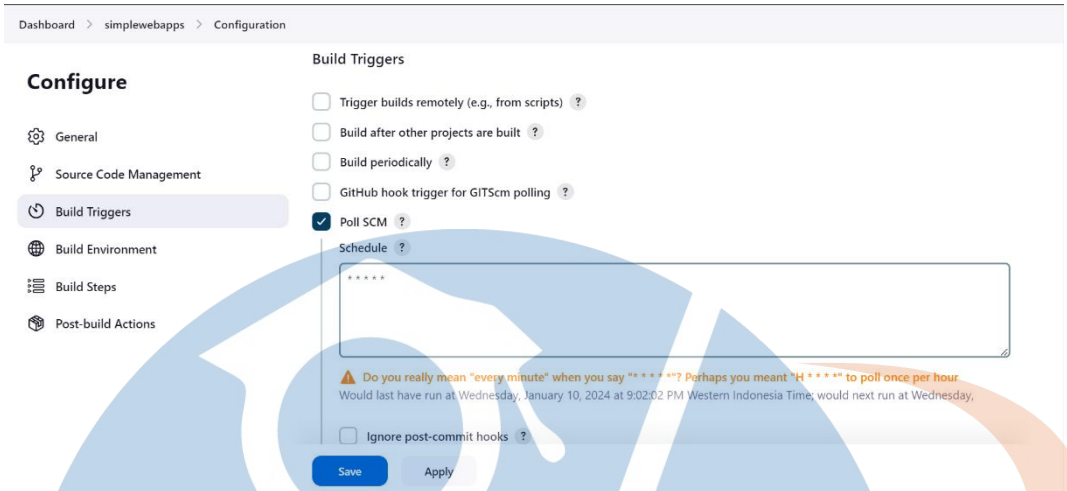
Gambar 4.11 Antarmuka Inisialisasi Proyek Jenkins

Langkah selanjutnya yang diilustrasikan dalam Gambar 4.12 adalah integrasi informasi *version control system* dalam konfigurasi proyek Jenkins. GitHub dipilih sebagai *version control system* untuk keperluan penelitian ini. Pada antarmuka konfigurasi, 'GitHub *project*' dipilih dan URL repositori dimasukkan. Bagian *'Source Code Management'* meminta kredensial yang diperlukan untuk akses *version control system* serta penentuan *branch* yang akan digunakan. Untuk instalasi Jenkins yang baru, kredensial diinput melalui tombol *'Add'*, dengan memasukkan nama pengguna dan kata sandi sebelum akhirnya disimpan. Proses ini memastikan bahwa Jenkins dapat mengakses dan berinteraksi dengan repositori GitHub, yang merupakan langkah penting dalam konfigurasi otomasi *Continuous Integration*. Penambahan kredensial ini memungkinkan Jenkins untuk melakukan operasi seperti *clone*, *pull*, dan *push*, yang diperlukan untuk pengelolaan dan integrasi kode secara otomatis.



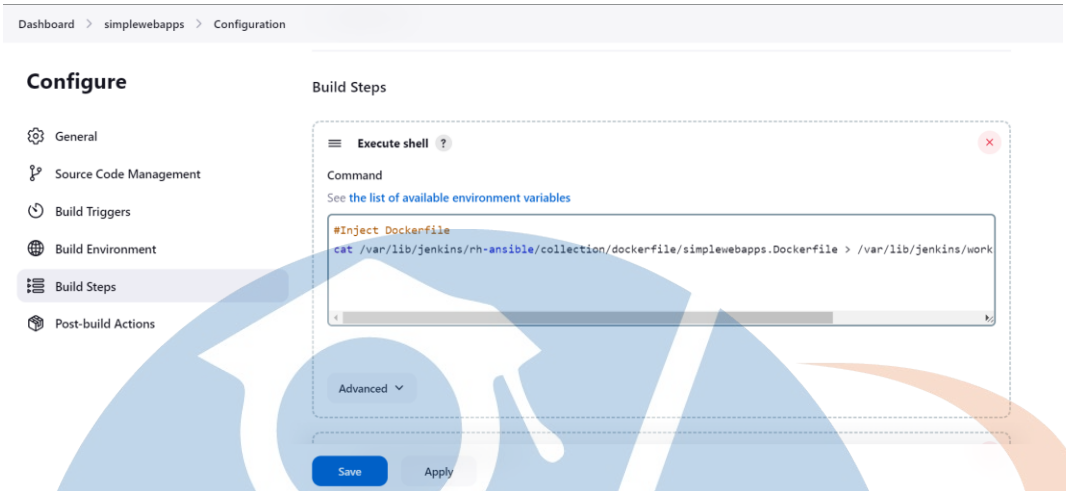
Gambar 4.12 Konfigurasi Sistem Manajemen Kode Sumber pada Jenkins

Pada Gambar 4.13 diilustrasikan prosedur penambahan *'Build Triggers'* dalam Jenkins, yang merupakan mekanisme untuk menetapkan pemicu yang akan menginisiasi pembangunan proyek secara otomatis. *Webhooks* biasanya diadopsi sebagai metode standar untuk aktivasi ini. Namun, dalam konteks penelitian ini, dipilih *'Poll SCM'*. Metode ini bertugas melakukan verifikasi berkelanjutan ke sistem kontrol versi (VCS) sesuai dengan jadwal yang telah diprogram. Jadwal yang diterapkan adalah pemantauan tiap menit, yang direpresentasikan dengan notasi '* * * * *', sesuai dengan konvensi sintaks *Crontab*. Pendekatan ini dirancang untuk memastikan bahwa setiap modifikasi pada repositori akan memicu Jenkins untuk memulai proses *build* secara otomatis.



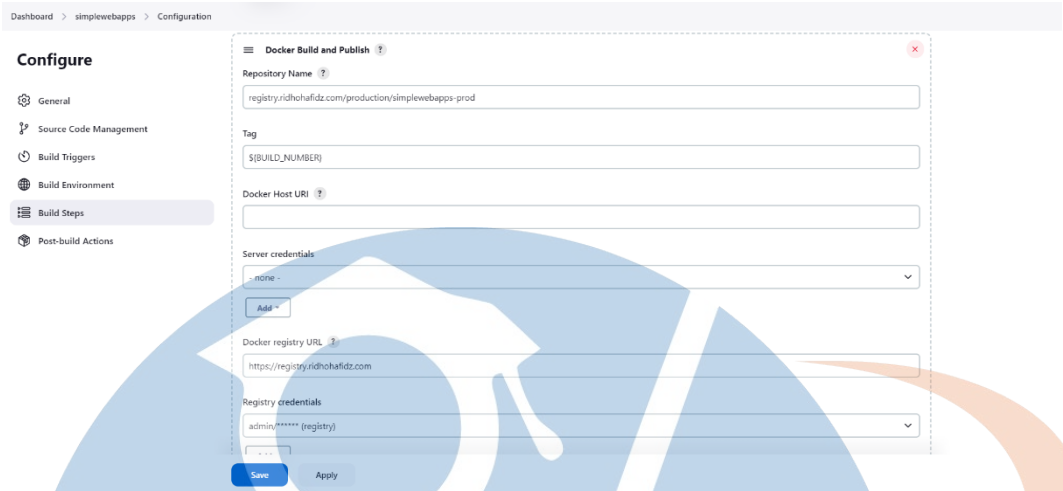
Gambar 4.13 Konfigurasi *Poll SCM* Sebagai *Build Trigger* di Jenkins

Sebagaimana ditampilkan dalam Gambar 4.14, langkah implementasi berikutnya dalam *pipeline* CI/CD melibatkan pengisian *'Build Steps'* di konfigurasi Jenkins. Peneliti menentukan *shell script* untuk membangun *image Docker*, yang pertama-tama melibatkan penyalinan *Dockerfile* ke *workspace* Jenkins. Ini ditujukan untuk mempersiapkan lingkungan tempat *image* akan dibangun. Perintah yang digunakan adalah cat /var/lib/jenkins/rh-ansible/collection/dockerfile/simplewebapps.Dockerfile>/var/lib/jenkins/workspace, yang merupakan operasi penyuntikan konten *Dockerfile* ke *workspace* Jenkins. Dalam *Dockerfile* tersebut, perintah FROM nginx:stable menetapkan basis image yang digunakan, sedangkan COPY . /usr/share/nginx/html/ menginstruksikan penyalinan konten direktori saat ini ke dalam direktori yang ditentukan di dalam kontainer. Perintah `EXPOSE` digunakan untuk menentukan *port* yang akan dibuka oleh kontainer Docker. Keseluruhan proses ini menegaskan otomatisasi dalam pembangunan dan *deployment* aplikasi web, sesuai dengan prinsip-prinsip CI/CD yang diadopsi dalam penelitian ini.



Gambar 4.14 Langkah Pembangunan Build Steps pada Jenkins

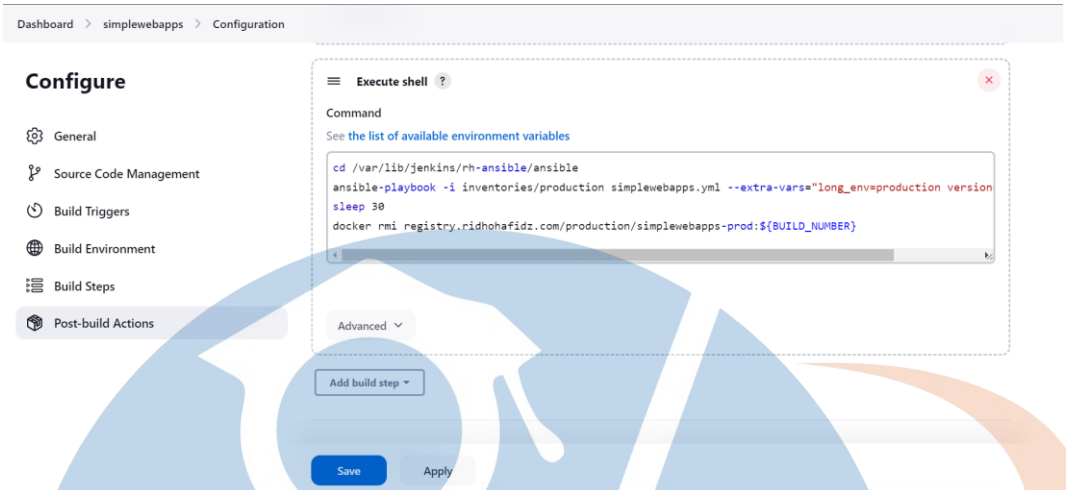
Seperti yang terillustrasi pada Gambar 4.15, tahap selanjutnya dalam penelitian ini melibatkan konfigurasi *'Post-build Actions'* di Jenkins, yang bertujuan untuk menetapkan langkah-langkah otomatis setelah proses *build* selesai. Dalam fase ini, nama repositori ditentukan beserta *tag* yang akan digunakan untuk versi *image* Docker yang dibangun, di mana *tag* tersebut umumnya mencakup nomor versi *build*. URI *host* Docker juga ditetapkan, yang menunjuk ke lokasi Docker *daemon* yang akan mengeksekusi pembangunan *image*. Untuk otentikasi dengan Docker *Host* dan Docker *registry*, peneliti menambahkan kredensial ke dalam konfigurasi Jenkins. Input URL *registry* Docker dan kredensial *registry* memungkinkan Jenkins untuk menerbitkan image yang telah dibangun ke *registry* yang spesifik. Langkah ini krusial untuk memastikan bahwa image yang telah dibangun dapat diakses dan digunakan untuk distribusi serta *deployment* ke lingkungan produksi atau pengujian lebih lanjut, memperlancar siklus pengembangan perangkat lunak sesuai dengan metodologi *Continuous Integration* dan *Continuous Deployment*.



Gambar 4.15 Konfigurasi *Docker Build and Publish* pada Jenkins

Langkah selanjutnya, seperti yang diilustrasikan pada Gambar 4.16, menampilkan bagian *'Build Steps'* dari konfigurasi proyek Jenkins. Di sini, terdapat perintah *shell* yang dieksekusi sebagai bagian dari proses build. Perintah yang diberikan pertama-tama melakukan navigasi ke direktori yang berisi playbook Ansible dengan menggunakan perintah cd. Setelah itu, playbook Ansible dijalankan menggunakan perintah ansible-playbook. Playbook ini dikonfigurasi untuk men-deploy aplikasi web sederhana.

Untuk menunda proses selanjutnya, digunakan perintah tambahan seperti *sleep* 30, memberikan waktu bagi langkah sebelumnya untuk menyelesaikan secara penuh. Terakhir, *image* Docker yang telah dibangun dihapus dari *registry* lokal menggunakan perintah docker rmi. Hal ini dilakukan untuk memastikan bahwa hanya versi image terbaru yang disimpan. *Tag* yang digunakan untuk *image* tersebut bersifat dinamis, mengacu pada nomor *build* dari Jenkins, yang ditunjukkan dengan variabel ${*BUILD_NUMBER*}.



Gambar 4.16 Konfigurasi *Build Steps* dalam Jenkins

Peneliti telah merancang sebuah file YAML yang krusial untuk proses *deployment* dalam lingkungan orkestrasi *container*. File ini mengandung konfigurasi terstruktur yang menyediakan definisi dari berbagai objek Kubernetes, termasuk *'namespace'*, *'service'*, dan *'deployment'*.

Namespace yang dibuat berfungsi sebagai partisi logis yang memisahkan kumpulan sumber daya, memungkinkan pengelolaan multi-tenant yang efisien dalam satu *cluster* Kubernetes. *Service* yang dikonfigurasi mendefinisikan bagaimana aplikasi dapat diakses oleh komponen lain atau dari luar *cluster*, menetapkan aturan eksposur melalui *port* yang spesifik. *Deployment*, pada gilirannya, mengatur instantiasi dan pembaruan aplikasi, menjamin ketersediaan dan pengelolaan versi aplikasi yang terkontrol.

Dalam proses otomatisasi ini, file YAML berperan sebagai dokumen deklaratif yang menentukan keadaan yang diinginkan untuk sumber daya dalam *cluster*. Berikut adalah rincian teknis dari konfigurasi yang telah didefinisikan dalam file YAML:

```
---
- name: Deploy Kubernetes Resources
  hosts: localhost
  tasks:
    - name: Create Namespace
      community.kubernetes.k8s:
        kubeconfig: "/root/.kube/config"
```

```
    state: present
    definition:
      apiVersion: v1
      kind: Namespace
      metadata:
        name: simplewebapps-prod

- name: Create Deployment
  community.kubernetes.k8s:
    kubeconfig: "/root/.kube/config"
    state: present
    definition:
      apiVersion: apps/v1
      kind: Deployment
      metadata:
        name: simplewebapps-prod
        namespace: simplewebapps-prod
      spec:
        selector:
          matchLabels:
            app: simplewebapps-prod
        replicas: 1
        template:
          metadata:
            labels:
              app: simplewebapps-prod
          spec:
            imagePullSecrets:
            - name: registry.tukanginfra.my.id
            containers:
            - name: simplewebapps-prod
              image:
              registry.ridhohafidz.com/production/simplewebapp
              s-prod:11
              resources:
                requests:
                  memory: "10M"
                  cpu: "10m"
                limits:
                  memory: "1000M"
                  cpu: "1000m"
              readinessProbe:
                tcpSocket:
                  port: 80
                initialDelaySeconds: 10
                periodSeconds: 10
            dnsConfig:
              nameservers:
              - "1.1.1.1"
              - "9.9.9.9"

- name: Create Service
  community.kubernetes.k8s:
    kubeconfig: "/root/.kube/config"
    state: present
```

```
definition:
  apiVersion: v1
  kind: Service
  metadata:
    name: simplewebapps-prod
    namespace: simplewebapps-prod
  spec:
    selector:
      app: simplewebapps-prod
    ports:
    - port: 80
      targetPort: 80
    type: ClusterIP
```

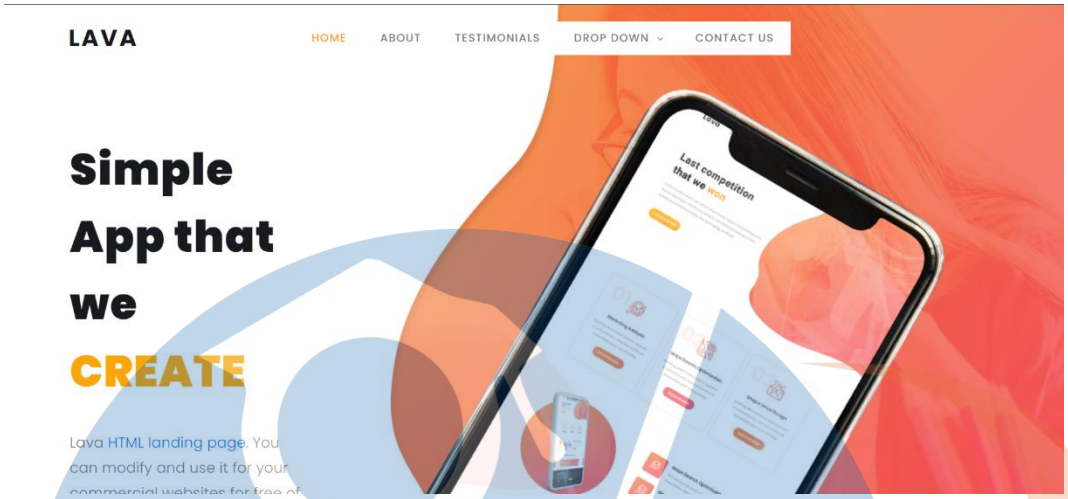
Setelah menyelesaikan konfigurasi proyek, peneliti melanjutkan dengan langkah finalisasi yang terdapat pada Gambar 4.17. Ini dilakukan dengan mengklik tombol *'Save'* yang terletak di bagian bawah layar antarmuka proyek *'simplewebapps'*. Langkah ini memastikan bahwa semua pengaturan yang telah dikonfigurasi disimpan secara permanen. Selanjutnya, untuk memulai proses *build* pertama, peneliti mengeklik pada opsi *'Build Now'* yang terpampang pada sisi kiri layar, di bawah menu *'Workspace'*. Aksi ini memicu Jenkins untuk menjalankan sequence *build* yang telah ditentukan, dimulai dengan menjalankan serangkaian task yang ada dalam *pipeline Continuous Integration* dan *Continuous Deployment* (CI/CD) untuk aplikasi yang diteliti. Proses ini merupakan langkah dalam verifikasi dan implementasi automasi pengujian serta penyebaran aplikasi yang dikembangkan.

Gambar 4.17 Tampilan Proyek simplewebapps pada Jenkins

Pada tahap akhir proses pembangunan menggunakan Jenkins, seperti yang digambarkan dalam Gambar 4.18, peneliti melakukan evaluasi terhadap *'Console Output'*. Fitur ini, yang terpadu dalam antarmuka Jenkins proyek *'simplewebapps'*, diakses melalui pemilihan opsi *'Console Output'* di bagian *'Permalinks'*. Langkah ini memungkinkan peneliti untuk melihat hasil build dan untuk memvalidasi proses *Continuous Integration* dan *Continuous Deployment* (CI/CD) yang telah diimplementasikan. Pemeriksaan *'Console Output'* adalah instrumen vital untuk mengidentifikasi keberhasilan atau kegagalan dalam proses *build*, serta untuk memverifikasi pemenuhan standar kualitas yang telah ditetapkan dalam pengembangan perangkat lunak.

Gambar 4.18 Antarmuka *Console Output* Jenkins Proyek *simplewebapps*

Sebagai tahapan konklusif pasca-proses *Continuous Integration* dan *Continuous Deployment* (CI/CD), peneliti mengadakan verifikasi fungsionalitas aplikasi web *'simplewebapps'*, seperti yang digambarkan dalam Gambar 4.19. Peneliti juga menguji interaktivitas dan fitur-fitur operasional untuk memastikan bahwa implementasi CI/CD telah berhasil dan aplikasi berjalan sesuai dengan spesifikasi yang ditetapkan. Proses ini penting untuk memvalidasi bahwa *pipeline deployment* telah efektif dalam menggulirkan versi terbaru ke lingkungan produksi, yang dapat diakses melalui alamat web aplikasi yang ditampilkan.



Gambar 4.19 Antarmuka Aplikasi Web Pasca-*Deployment*

4.3. Evaluasi Sistem

Setelah implementasi *Continuous Integration/Continuous Deployment* (CI/CD) menggunakan Jenkins dalam lingkungan Kubernetes yang dikelola oleh Rancher Kubernetes Engine, peneliti melakukan evaluasi sistem untuk dapat mengetahui tingkat efektivitas dari *pipeline* CI/CD yang telah dibangun dengan melakukan rangkaian pengujian yang telah peneliti rancang sebelumnya.

Tabel yang disajikan di bawah ini menampilkan hasil dari serangkaian pengujian sistematis yang dilaksanakan sebanyak tiga kali untuk mengevaluasi efektivitas *pipeline Continuous Integration/Continuous Deployment* (CI/CD). *Pipeline* ini dikembangkan menggunakan Jenkins dalam konteks lingkungan Kubernetes yang dikelola oleh Rancher Kubernetes Engine.

Tabel 4. 5 Pengujian Sistem

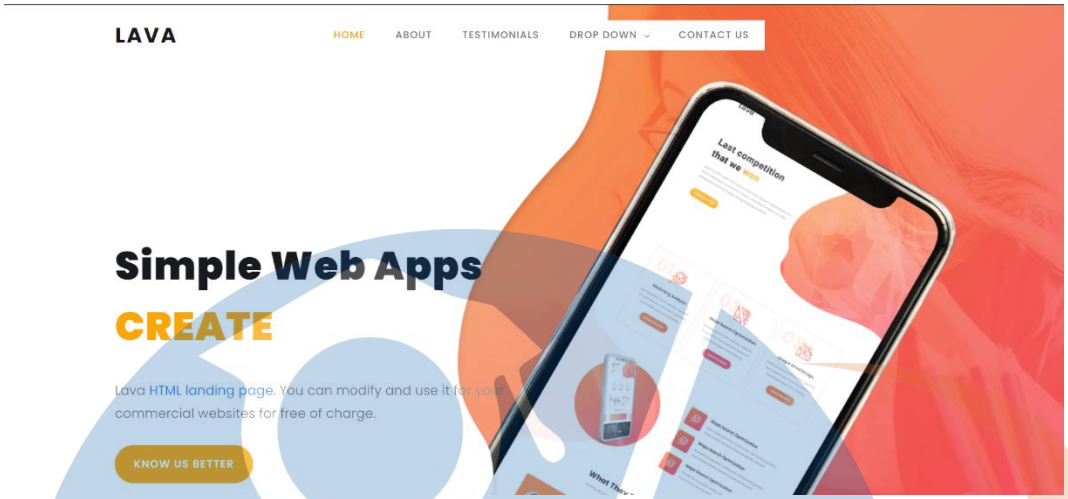
No.	Deskripsi Pengujian	Status *Pipeline* CI/CD	Keberhasilan (%)
1.	Perubahan *source code*	*Success*	100%
2.	Penambahan fitur terbaru pada *source code*	*Success*	100%

3.	Integrasi asset berupa *icon image* sebagai fitur tambahan pada *source code*	*Success*	100%

Pengujian pertama diarahkan untuk mengevaluasi kapasitas sistem dalam mengelola perubahan pada *source code*. Sebagaimana diilustrasikan pada Gambar 4.20, status *pipeline* CI/CD menunjukkan hasil *'Success'*, mengindikasikan bahwa proses integrasi dan *deployment* berjalan tanpa hambatan. Hal ini menegaskan bahwa *pipeline* yang telah dikonfigurasi dapat mengakomodasi modifikasi kode dengan efisien, memastikan kelancaran proses *development* berkelanjutan.

Gambar 4.20 Pengujian Pertama *Build Success* pada Jenkins CI/CD *Pipeline*

Selanjutnya, Gambar 4.21 memperlihatkan antarmuka web setelah penerapan perubahan *code*. Tampilan ini merupakan hasil langsung dari pengujian yang telah dilakukan, memberikan validasi visual atas fungsi dan estetika dari perubahan yang diimplementasikan. Hal ini penting untuk memverifikasi bahwa perubahan tidak hanya berhasil melewati *pipeline* CI/CD tetapi juga terimplementasi dengan tepat pada lingkungan target.

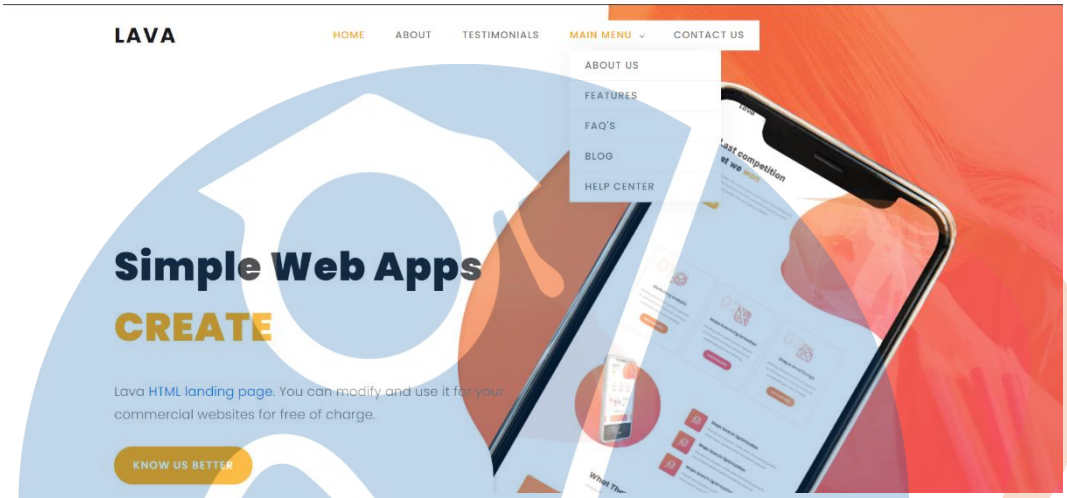


Gambar 4.21 Tayangan perubahan tampilan pada web

Skenario pengujian kedua melibatkan penambahan *source code* baru untuk evaluasi efektivitas *pipeline* CI/CD dalam mengintegrasikan fitur dan mengelola proses *build* serta *deployment*. Keberhasilan integrasi dan *deployment* ini dapat dilihat pada hasil tampilan web serta pada Gambar 4.22, yang menunjukkan status *'Success'* pada *pipeline* CI/CD, menegaskan fungsi *pipeline* yang efektif.

Gambar 4.22 Pengujian Kedua *Build Success* pada Jenkins CI/CD *Pipeline*

Gambar 4.23 menggambarkan hasil antarmuka usai pelaksanaan pengujian penambahan fitur baru, sejalan dengan metode yang dijelaskan dalam skenario pengujian kedua.

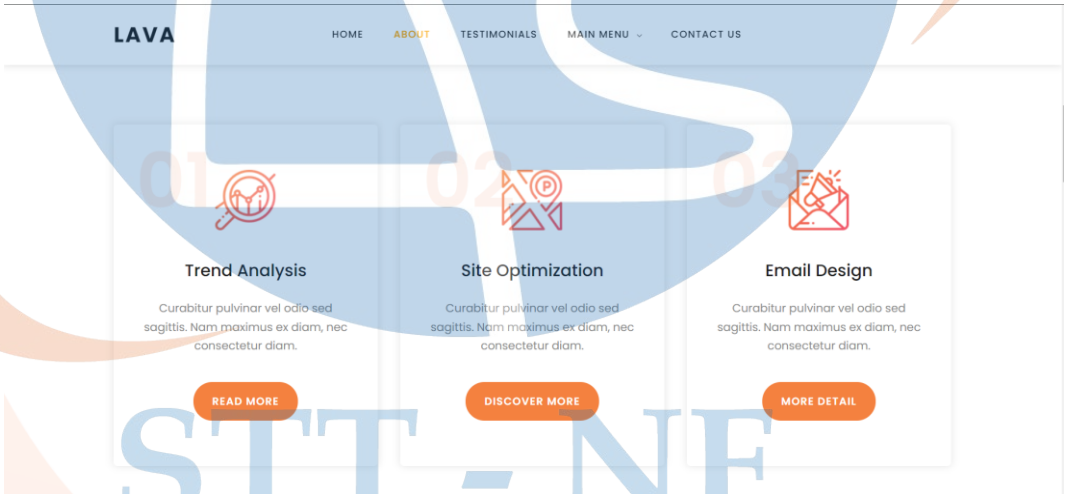


Gambar 4.23 Tayangan penambahan tampilan fitur pada web

Skenario pengujian terakhir difokuskan pada integrasi asset berupa ikon gambar sebagai fitur tambahan dalam *source code*. Pengujian ini secara khusus mengevaluasi proses integrasi asset baru, memverifikasi fungsionalitas dan representasi visualnya dalam aplikasi web. Selama proses ini, *build* dan *deployment* yang dilaksanakan melalui Jenkins diuji coba untuk menjamin bahwa asset yang diintegrasikan terwakili secara akurat pada situs web. Keberhasilan dari tahapan integrasi dan *deployment* ini termanifestasi dalam tampilan akhir pada web dan dikonfirmasi oleh Gambar 4.24, yang menampilkan status *'Success'* pada *pipeline* CI/CD, menunjukkan kinerja *pipeline* yang efisien dan efektif.

Gambar 4.24 Pengujian Ketiga *Build Success* pada Jenkins CI/CD *Pipeline*

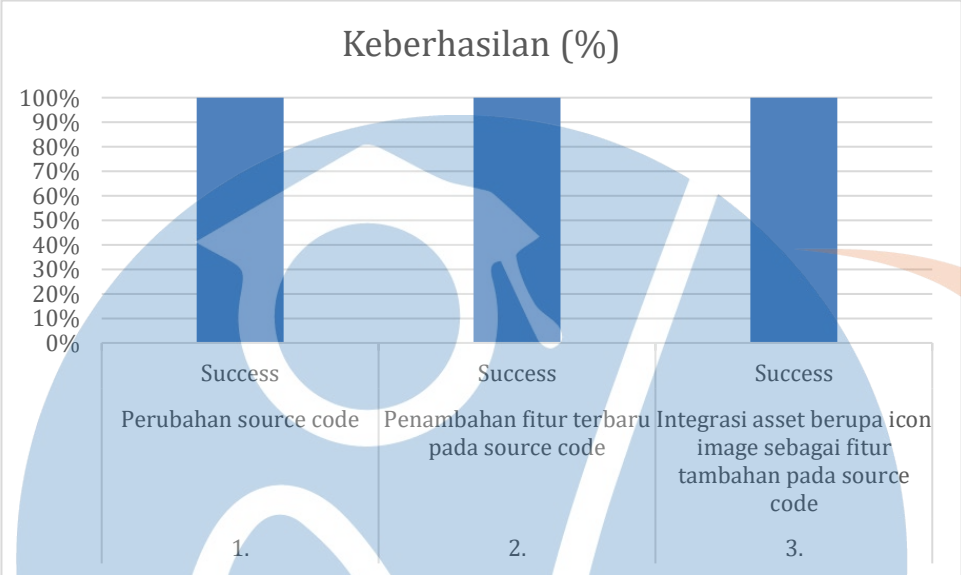
Dalam konteks penelitian ini, Gambar 4.25 memberikan representasi visual dari antarmuka web yang telah mengalami modifikasi pasca-integrasi asset berupa ikon gambar.



Gambar 4.25 Tayangan penambahan asset pada web

Diagram pada Gambar 4.26 menampilkan efektivitas *pipeline* CI/CD dalam melaksanakan serangkaian pengujian. Setiap kolom menunjukkan persentase keberhasilan sebesar 100% untuk ketiga skenario pengujian: perubahan pada *source code*, penambahan fitur baru pada *source code*, serta integrasi asset berupa ikon gambar ke dalam *source code*. Hasil yang konsisten ini menunjukkan bahwa

pipeline CI/CD yang diimplementasikan dapat mengeksekusi berbagai pengujian dengan efektif.



Gambar 4.26 Diagram Pengujian CI/CD *Pipeline*

BAB V
PENUTUP

5.1. Kesimpulan

Berdasarkan hasil penelitian yang dilakukan, dapat disimpulkan bahwa penelitian ini berhasil menerapkan *Continuous Integration/Continuous Deployment* (CI/CD) dalam pengembangan aplikasi web di lingkungan Kubernetes berbasis Rancher Kubernetes Engine. Penelitian ini juga berhasil memberikan jawaban terhadap rumusan masalah yang telah didefinisikan sebelumnya. Beberapa kesimpulan yang dapat diambil meliputi:

1. Rancangan Langkah CI/CD:
   - Penelitian ini merancang sebuah *pipeline* CI/CD efektif untuk pengembangan aplikasi web dalam lingkungan Kubernetes yang dioperasikan melalui Rancher Kubernetes Engine (RKE).
   - Tahapan pipeline meliputi inisialisasi infrastruktur dengan RKE, integrasi repositori kode, otomatisasi pengujian, pembangunan aplikasi, dan deployment yang berkesinambungan.
   - Penggunaan alat seperti Git untuk manajemen *code*, Jenkins untuk integrasi berkelanjutan, dan Docker untuk kontainerisasi aplikasi, menjadi inti dalam rancangan ini.
   - Konsep *DevOps* terintegrasi secara penuh dalam *pipeline*, memungkinkan kolaborasi dan otomatisasi yang efisien antara pengembangan dan operasi.
2. Efektivitas *Pipeline* CI/CD:
   - Efektivitas pipeline ditandai dengan peningkatan efektivitas dan keandalan dalam pengiriman aplikasi.
   - Otomatisasi yang dilakukan mengurangi kesalahan manusia dan meningkatkan efisiensi dari waktu siklus pengembangan hingga deployment.
   - Kubernetes di RKE menawarkan skalabilitas dan pengelolaan yang efisien, yang sangat mendukung praktik CI/CD.

- *Feedback* dan pembaruan dapat dilakukan dengan tingkat efektivitas dan efisiensi yang lebih tinggi, memungkinkan proses iterasi pengembangan menjadi lebih lancar.

5.2. Saran

Berdasarkan hasil penelitian yang telah dilakukan, diperoleh kesimpulan bahwa masih terdapat sejumlah kekurangan. Oleh karena itu, disarankan beberapa langkah perbaikan yang dapat diterapkan agar penelitian serupa di masa depan dapat dilaksanakan dengan lebih optimal. Beberapa saran yang dapat diberikan antara lain:

1. Optimasi *Pipeline*:
   - Penting untuk terus mengoptimalkan pipeline CI/CD, misalnya dengan memperbaiki skrip otomatisasi, meningkatkan keamanan, dan mengadopsi praktik terbaik *DevOps*.
   - Penyesuaian terhadap kebutuhan spesifik proyek dan lingkungan operasional sangat dianjurkan untuk efisiensi yang lebih besar.
2. Integrasi dan Adaptasi Teknologi Terbaru:
   - Menjaga *pipeline* agar tetap ter*update* dengan perkembangan teknologi terkini, seperti penggunaan *container* yang lebih efisien atau alat CI/CD baru, sangat penting.
   - Adaptasi dengan inovasi seperti *serverless computing* dapat meningkatkan efisiensi dan efektivitas *pipeline*.
3. Pengukuran dan Evaluasi Berkelanjutan:
   - Melakukan pengukuran kinerja dan evaluasi secara berkala terhadap *pipeline* CI/CD untuk memastikan efektivitasnya dan mengidentifikasi area yang memerlukan peningkatan.
   - Penelitian ini menunjukkan potensi besar dari penerapan CI/CD dalam pengembangan aplikasi web di lingkungan Kubernetes dengan Rancher, yang tidak hanya meningkatkan efisiensi tetapi juga memfasilitasi inovasi dan penyebaran aplikasi dengan lebih efektif.

DAFTAR REFERENSI

[1] B. Naveen, J. K. Grandhi, K. Lasya, E. M. Reddy, N. Srinivasu, and S. Bulla, "Efficient Automation of Web Application Development and Deployment Using Jenkins: A Comprehensive CI/CD Pipeline for Enhanced Productivity and Quality," in *2023 International Conference on Self Sustainable Artificial Intelligence Systems (ICSSAS)*, IEEE, Oct. 2023, pp. 751–756. doi: 10.1109/ICSSAS57918.2023.10331631.

[2] J. Mahboob and J. Coffman, "A Kubernetes CI/CD Pipeline with Asylo as a Trusted Execution Environment Abstraction Framework," in *2021 IEEE 11th Annual Computing and Communication Workshop and Conference (CCWC)*, IEEE, Jan. 2021, pp. 0529–0535. doi: 10.1109/CCWC51732.2021.9376148.

[3] H. Fathoni, C.-T. Yang, C.-Y. Huang, C.-Y. Chen, and T.-F. Hsieh, "Aquaculture Monitoring Systems Based on Lightweight Kubernetes and Rancher," 2022, pp. 38–45. doi: 10.1007/978-3-031-20398-5_4.

[4] L. Luo, "Web Application Software Engineering Technology and Process," in *2021 IEEE Asia-Pacific Conference on Image Processing, Electronics and Computers (IPEC)*, IEEE, Apr. 2021, pp. 935–937. doi: 10.1109/IPEC51340.2021.9421250.

[5] C. Pahl, A. Brogi, J. Soldani, and P. Jamshidi, "Cloud Container Technologies: A State-of-the-Art Review," *IEEE Transactions on Cloud Computing*, vol. 7, no. 3, pp. 677–692, Jul. 2019, doi: 10.1109/TCC.2017.2702586.

[6] S. G and P. D. R, "Deploying a Kubernetes Cluster with Kubernetes Operation kops on AWS Cloud Experiments and Lessons Learned," *Int J Eng Adv Technol*, vol. 9, no. 5, pp. 984–989, Jun. 2020, doi: 10.35940/ijeat.E1023.069520.

[7] S. Böhm and G. Wirtz, "Profiling Lightweight Container Platforms: MicroK8s and K3s in Comparison to Kubernetes," Oct. 2021.

[8] S. Hardikar, P. Ahirwar, and S. Rajan, “Containerization: Cloud Computing based Inspiration Technology for Adoption through Docker and Kubernetes,” in *2021 Second International Conference on Electronics and Sustainable Communication Systems (ICESC)*, IEEE, Aug. 2021, pp. 1996–2003. doi: 10.1109/ICESC51422.2021.9532917.

[9] A. Alanda, H. A. Mooduto, and R. Hadelina, “Continuous Integration and Continuous Deployment (CI/CD) for Web Applications on Cloud Infrastructures,” *JITCE (Journal of Information Technology and Computer Engineering)*, vol. 6, no. 02, pp. 50–55, Sep. 2022, doi: 10.25077/jitce.6.02.50-55.2022.

[10] S. Mysari and V. Bejgam, “Continuous Integration and Continuous Deployment Pipeline Automation Using Jenkins Ansible,” in *2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE)*, IEEE, Feb. 2020, pp. 1–4. doi: 10.1109/ic-ETITE47903.2020.239.

[11] J. Bjørgeengen, “A Multitenant Container Platform with OKD, Harbor Registry and ELK,” 2019, pp. 69–79. doi: 10.1007/978-3-030-34356-9_7.

[12] S. Mysari and V. Bejgam, “Continuous Integration and Continuous Deployment Pipeline Automation Using Jenkins Ansible,” in *2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE)*, IEEE, Feb. 2020, pp. 1–4. doi: 10.1109/ic-ETITE47903.2020.239.

[13] A. Schreiber and C. de Boer, “Modelling Knowledge about Software Processes using Provenance Graphs and its Application to Git-based Version Control Systems,” in *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*, New York, NY, USA: ACM, Jun. 2020, pp. 358–359. doi: 10.1145/3387940.3392220.

[14] M. Niazi, S. Abbas, A.-H. Soliman, T. Alyas, S. Asif, and T. Faiz, “Vertical Pod Autoscaling in Kubernetes for Elastic Container Collaborative燜

ramework," *Computers, Materials & Continua*, vol. 74, no. 1, pp. 591–606, 2023, doi: 10.32604/cmc.2023.032474.

[15] N. Nguyen and T. Kim, "Toward Highly Scalable Load Balancing in Kubernetes Clusters," *IEEE Communications Magazine*, vol. 58, no. 7, pp. 78–83, Jul. 2020, doi: 10.1109/MCOM.001.1900660.

[16] L. H. Pramono, R. C. Buwono, and Y. G. Waskito, "Round-robin Algorithm in HAProxy and Nginx Load Balancing Performance Evaluation: a Review," in *2018 International Seminar on Research of Information Technology and Intelligent Systems (ISRITI)*, IEEE, Nov. 2018, pp. 367–372. doi: 10.1109/ISRITI.2018.8864455.

[17] P. Ware and M. Warschauer, "Qualitative Research on Information and Communication Technology," in *The Encyclopedia of Applied Linguistics*, Wiley, 2019, pp. 1–6. doi: 10.1002/9781405198431.wbeal0984.pub2.