TUGAS AKHIR - IF184802

IMPLEMENTASI SCAFFOLDING UNTUK ALUR PROSES PIPELINE CI/CD PENGEMBANGAN APLIKASI PADA PLATFORM MYITS BERBASIS CLOUD-NATIVE MENGGUNAKAN JENKINS DAN KUBERNETES

ISMAIL SYARIEF
NRP 05111640000168

Dosen Pembimbing
Royyana Muslim Ijtihadie S.Kom, M.Kom., Ph.D
Rizky Januar Akbar, S.Kom., M.Eng.

DEPARTEMEN TEKNIK INFORMATIKA
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya 2020

TUGAS AKHIR – IF184802

IMPLEMENTASI SCAFFOLDING UNTUK ALUR PROSES PIPELINE CI/CD PENGEMBANGAN APLIKASI PADA PLATFORM MYITS BERBASIS CLOUD-NATIVE MENGGUNAKAN JENKINS DAN KUBERNETES

ISMAIL SYARIEF
NRP 05111640000168

Dosen Pembimbing
Royyana Muslim Ijtihadie S.Kom, M.Kom., Ph.D
Rizky Januar Akbar, S.Kom., M.Eng.

DEPARTEMEN TEKNIK INFORMATIKA
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya 2020

[Halaman ini sengaja dikosongkan]

UNDERGRADUATE THESIS – IF184802

IMPLEMENTATION OF SCAFFOLDING FOR PIPELINE CI / CD APPLICATION DEVELOPMENT PROCESS ON CLOUD-NATIVE BASED MYITS PLATFORM USING JENKINS AND KUBERNETES

ISMAIL SYARIEF
NRP 05111640000168

Advisor
Royyana Muslim Ijtihadie S.Kom, M.Kom., Ph.D
Rizky Januar Akbar, S.Kom., M.Eng.

INFORMATICS ENGINEERING DEPARTMENT
Faculty of Intelligent Electrical and Informatics Technology
Institut Teknologi Sepuluh Nopember
Surabaya 2020

[Halaman ini sengaja dikosongkan]

LEMBAR PENGESAHAN

IMPLEMENTASI SCAFFOLDING UNTUK ALUR PROSES PIPELINE CI/CD PENGEMBANGAN APLIKASI PADA PLATFORM MYITS BERBASIS CLOUD-NATIVE MENGGUNAKAN JENKINS DAN KUBERNETES

TUGAS AKHIR

Diajukan Guna Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer pada
Rumpun Mata Kuliah Rekayasa Perangkat Lunak
Program Studi S-1 Departemen Teknik Informatika
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember

Oleh :
ISMAIL SYARIEF
NRP : 05111640000168

Disetujui oleh Dosen Pembimbing Tugas Akhir :

Royyana Muslim Ijtihadie S.Kom, M.Kom., Ph.D
NIP: 197708242006041001
..... (pembimbing 1)

Rizky Januar Akbar, S.Kom., M.Eng.
NIP: 198701032014041001
..... (pembimbing 2)

SURABAYA
JULI 2020

[Halaman ini sengaja dikosongkan]

IMPLEMENTASI SCAFFOLDING UNTUK ALUR PROSES PIPELINE CI/CD PENGEMBANGAN APLIKASI PADA PLATFORM MYITS BERBASIS CLOUD-NATIVE MENGGUNAKAN JENKINS DAN KUBERNETES

Nama Mahasiswa	: ISMAIL SYARIEF
NRP	: 05111540000168
Jurusan	: Teknik Informatika ITS
Dosen Pembimbing I	: Royyana Muslim Ijtihadie S.Kom, M.Kom., Ph.D
Dosen Pembimbing II	: Rizky Januar Akbar, S.Kom., M.Eng.

Abstrak

Dengan berkembangnya teknologi di bidang cloud-native, platform myITS mengimplementasikan Kubernetes sebagai platform orkestrasinya, dengan kebutuhan aplikasi yang kian bertambah, mendatang akan ada aplikasi baru yang terus ditambahkan pada platform myITS.

Demi meningkatkan efektivitas dalam integrasi aplikasi baru dengan lingkungan Kubernetes pada platform myITS, diperlukan sebuah pipeline

Untuk memaksimalkan pipeline CI/CD pada platform myITS, suatu aplikasi perlu dikembangkan pada lingkungan yang serupa dengan lingkungan produksinya nanti, dengan ini perlu adanya sebuah standarisasi struktur atau scaffold dalam melakukan pengembangan dari suatu aplikasi. Kubernetes pada platform myITS melakukan orkestrasi terhadap kontainer docker, yang berarti seluruh aplikasi dijalankan di dalam satu atau lebih kontainer docker, dengan itu kontainer docker akan digunakan sebagai scaffold pengembangan bagi programmer.

Kata kunci: Kubernetes, Jenkins, Pipeline, Helm, Rancher, Docker

IMPLEMENTATION OF SCAFFOLDING FOR PIPELINE CI / CD APPLICATION DEVELOPMENT PROCESS ON CLOUD-NATIVE BASED MYITS PLATFORM USING JENKINS AND KUBERNETES

Nama Mahasiswa	: ISMAIL SYARIEF
NRP	: 05111540000168
Jurusan	: Teknik Informatika ITS
Dosen Pembimbing I	: Royyana Muslim Ijtihadie S.Kom, M.Kom., Ph.D
Dosen Pembimbing II	: Rizky Januar Akbar, S.Kom., M.Eng.

Abstract

With the development of technology in the cloud-native field, myITS platform implements Kubernetes as its orchestration platform, with growing application needs, new applications will continue to be added to the myITS platform.

To increase the effectiveness in integrating new applications with the Kubernetes environment on the myITS platform, a CI / CD pipeline is needed to support those needs.

To maximize the CI / CD pipeline on the myITS platform, an application needs to be developed in an environment similar to the production environment later, with this it is necessary to have a standardized structure or scaffold in developing an application. Kubernetes on the myITS platform orchestrates docker containers, which means that all applications are run in one or more docker containers, with which the docker container will be used as a development scaffold for the programmer.

Keywords: Kubernetes, Jenkins, Pipeline, Helm, Rancher, Docker

KATA PENGANTAR

بِسۡمِ ٱللَّهِ ٱلرَّحۡمَٰنِ ٱلرَّحِيمِ

Puji syukur kepada Allah Yang Maha Esa atas segala karunia dan rahmat-Nya sehingga penulis dapat menyelesaikan tugas akhir yang berjudul **"Implementasi Scaffolding Untuk Alur Proses Pipeline Ci/CD Pengembangan Aplikasi Pada Platform Myits Berbasis Cloud-native Menggunakan Jenkins Dan Kubernetes".**

Dalam pelaksanaan dan pembuatan tugas akhir ini tentunya sangat banyak bantuan yang penulis terima dari berbagai pihak, tanpa mengurangi rasa hormat penulis ingin mengucapkan terima kasih sebesar-besarnya kepada:

1. Allah SWT. dan Nabi Muhammad SAW. yang telah membimbing penulis selama hidup.
2. Keluarga penulis yang selalu memberikan dukungan baik berupa doa, moral, dan material yang tak terhingga kepada penulis, sehingga penulis dapat menyelesaikan tugas akhir ini.
3. Bapak Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D. dan Bapak Rizky Januar Akbar, S.Kom., Ph.D. selaku Dosen Pembimbing penulis yang telah membimbing, memberikan nasihat, dan memotivasi penulis sehingga penulis dapat menyelesaikan tugas akhir ini.
4. Bapak dan Ibu Dosen yang telah memberikan ilmunya selama penulis berkuliah di Informatika ITS.
5. Teman-teman sesama bimbingan TA (Hilmi, Satria, Azki) yang telah membantu penulis dalam *brainstorming*.
6. Teman-teman dari keluarga besar Laboratorium RPL yang telah menemani, memberi semangat, hiburan penulis selama berkuliah.
7. Teman-teman discord radio A56 yang telah menemani penulis selama *work from home*.

8. Teman-teman informatika ITS angkatan 2015, 2016, 2017.
9. Untuk orang-orang yang tidak dapat disebutkan satu persatu oleh penulis dan pembaca buku tugas akhir ini.

Penulis telah berusaha sebaik-baiknya dalam menyusun tugas akhir ini. Namun, penulis menyadari bahwa tugas akhir ini masih memiliki banyak kekurangan. Kritik dan saran yang membangun dapat disampaikan sebagai bahan perbaikan selanjutnya.

Harapan dari penulis, semoga apa yang tertulis di dalam buku tugas akhir ini dapat bermanfaat bagi pengembangan ilmu pengetahuan saat ini dan ke depannya, serta dapat memberikan kontribusi yang nyata.

Tetap semangat dan yakin karena setiap perjuangan kita insya Allah akan dicatat sebagai amal ibadah. Semoga rahmat Allah selalu menyertai kita semua. Aamiin.

Surabaya, Juli 2020

Ismail Syarief

[Halaman ini sengaja dikosongkan]

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR KODE SUMBER

DAFTAR TABEL

DAFTAR TERMINAL

[Halaman ini sengaja dikosongkan

BAB I
PENDAHULUAN

Bab pendahuluan membahas garis besar penyusunan Tugas Akhir yang meliputi latar belakang, tujuan pembuatan, rumusan dan batasan permasalahan, metodologi penyusunan Tugas Akhir, dan sistematika penulisan.

1.1. Latar Belakang

Dengan berkembangnya teknologi di bidang *cloud-native*, platform myITS mengimplementasikan Kubernetes sebagai platform orkestrasinya, dengan kebutuhan aplikasi yang kian bertambah, mendatang akan ada aplikasi baru yang terus ditambahkan pada platform myITS.

Demi meningkatkan efektivitas dalam integrasi aplikasi baru dengan lingkungan Kubernetes pada platform myITS, diperlukan sebuah *pipeline CI/CD* untuk menunjang kebutuhan tersebut. *CI/CD* itu sendiri adalah sebuah akronim dari *continous integration* dan *continous delivery* yang berarti integrasi dan pengiriman berkelanjutan, suatu proses *CI/CD* diharuskan berjalan secara otomatis, proses *CI/CD* terdiri dari kumpulan tahapan eksekusi yang berjalan secara berurutan[1], terminologi *pipeline* digunakan karena proses *CI/CD* menyerupai sebuah pipa yang mengalirkan air dari satu tempat ke tempat lain sama dengannya aplikasi yang ada pada lingkungan pengembangan menuju lingkungan produksi.

Untuk memaksimalkan *pipeline CI/CD* pada platform myITS, suatu aplikasi perlu dikembangkan pada lingkungan yang serupa dengan lingkungan produksi, dengan ini perlu adanya sebuah standarisasi struktur atau *scaffold* dalam melakukan pengembangan dari suatu aplikasi. Kubernetes pada platform myITS melakukan orkestrasi terhadap kontainer docker, yang berarti seluruh aplikasi dijalankan di dalam satu atau lebih kontainer docker, suatu lingkungan pengembangan juga perlu bersifat *reusable* dan *reproducible*, maksud *reusable* itu sendiri

adalah suatu lingkungan kerja dapat digunakan untuk mengembangkan aplikasi lain dan *reproducible* adalah suatu lingkungan kerja dapat dibuat kembali menjadi lingkungan pengembangan ataupun dibuat kembali pada lingkungan produksi, dengan itu kontainer docker akan digunakan sebagai *scaffold* pengembangan bagi *programmer*.

Pipline CI/CD akan dijalankan pada sebuah aplikasi bernama Jenkins, Jenkins akan mengeksekusi setiap tahapan dari *pipeline CI/CD*, Jenkins akan mengemas kode sumber pada *scaffold* pengembang menjadi *image* yang dapat dijalankan oleh kontainer docker pada klaster kubernetes myITS, lingkungan pengembang yang berjalan pada docker dengan lingkungan produksi yang berjalan pada Kubernetes dapat terintegrasi dengan baik dikarenakan kesamaan media yang dijalankan yaitu image, dengan kesamaan tersebut aplikasi yang dikemas dalam image dapat dijalankan pada docker dan Kubernetes meskipun docker dan Kubernetes berasal dari *vendor* yang berbeda.

Maka tugas akhir ini bertujuan untuk merancang dan mengimplementasikan *pipeline CI/CD* serta merancang *scaffold* lingkungan pengembangan bagi *programmer*.

1.2. Rumusan Masalah

Perumusan masalah yang terdapat pada Tugas Akhir ini, antara lain adalah:

1. Bagaimana menggunakan kontainer docker sebagai standar *scaffolding* pada lingkungan pengembangan.
2. Bagaimana menerapkan Jenkins dan Kubernetes dalam proses *pipeline CI/CD*
3. Bagaimana mengintegrasikan lingkungan pengembangan dengan lingkungan produksi?
4. Bagaimana membuat lingkungan pengembangan yang *reusable* dan *reproducible*?

1.3. Batasan Masalah

Batasan masalah yang terdapat pada Tugas Akhir ini, sebagai berikut:

1. Proses *Testing*, *Merge*, *Pull Request* dan pengelolaan *Backlog* tidak tercakup dalam tugas akhir ini, dikarenakan masih perlunya interaksi manusia dalam proses tersebut.

1.4. Tujuan

Tujuan dari pembuatan Tugas Akhir ini adalah sebagai berikut:

1. Mengetahui mekanisme integrasi GitHub, Docker, Helm, Jenkins, dan Kubernetes untuk pengimplementasian *pipeline*.
2. Mengimplementasikan kontainer docker sebagai *scaffold* lingkungan pengembangan.

Manfaat yang diharapkan dari pembuatan Tugas Akhir ini antara lain:

1. Diharapkan dengan dirancangnya infrastruktur ini pengembang dapat dipermudah dalam melakukan pengembangan perangkat lunak secara teknis dari tahap inisiasi hingga produksi.
2. Dengan mengimplementasikan scaffold, Dapat menjaga konsistensi lingkungan pengembangan.
3. Memudahkan dalam proses pengembangan aplikasi pada perangkat yang berbeda beda.
4. Memudahkan integrasi dengan lingkungan produksi.
5. Mempersingkat proses pembuatan lingkungan pengembangan.
6. Dengan Mengimplementasikan *pipeline CI/CD* dapat memudahkan proses deployment atau pembaruan aplikasi dikarenakan prosesnya yang berjalan secara otomatis

1.5. Metodologi

Tahap yang dilakukan untuk menyelesaikan Tugas Akhir ini adalah sebagai berikut:

1. **Penyusunan Proposal Tugas Akhir**

   Tahap pertama dalam proses pengerjaan tugas akhir ini adalah menyusun proposal tugas akhir yang berisi tentang latar belakang tugas akhir dibuat, rumusan masalah, tujuan dari pembuatan tugas akhir, dan manfaat hasil pembuatan tugas akhir. Selain itu, terdapat tinjauan pustaka digunakan sebagai referensi pendukung pembuatan tugas akhir. Lalu juga terdapat metodologi yang berisi penjelasan mengenai tahapan penyusunan tugas akhir mulai dari penyusunan proposal hingga penyusunan buku tugas akhir. Terakhir, pada proposal tugas akhir terdapat jadwal kegiatan pengerjaan tugas akhir.

2. **Studi Literatur**

   Pada tahap ini, akan dicari studi literature yang relevan untuk dijadikan referensi dalam pengerjaan tugas akhir. Studi literatur ini didapatkan dokumentasi resmi Docker, Jenkins, Kubernetes, Helm, dan materi pendukung yang berasal dari materi-materi kuliah yang berhubungan dengan metode yang akan digunakan.

3. **Analisis dan Desain**

   Pada tahap ini dilakukan perencanaan dan desain arsitektur. Hal ini bertujuan untuk mendapatkan desain arsitektur yang baik dan dapat diimplementasikan.

4. **Implementasi Infrastruktur**
   Pada tahap ini akan dilakukan implementasi infrastruktur yang telah didesain sebelumnya. Selanjutnya, akan dilakukan pengujian terhadap infrastruktur yang digunakan kemudian ditarik kesimpulan dari implementasi tersebut.

5. **Pengujian dan Evaluasi**
   Pengujian dilakukan untuk melihat efektivitas dari infrastruktur yang sudah dirancang bagi pengembang dan pengelola aplikasi. Lalu dari hasil pengujian itu akan dilakukan evaluasi terhadap infrastruktur yang digunakan.

6. **Penyusunan Buku Tugas Akhir**
   Pada tahap ini dilakukan penyusunan buku sebagai dokumentasi dari pelaksanaan tugas akhir yang mencakup seluruh konsep, teori, implementasi dan evaluasi. Sistematika penulisan buku tugas akhir ini secara garis besar antara lain:
   1. Pendahuluan
      a. Latar Belakang
      b. Rumusan Masalah
      c. Batasan Tugas Akhir
      d. Tujuan
      e. Metodologi
      f. Sistematika Penulisan
   2. Tinjauan Pustaka
   3. Desain dan Implementasi
   4. Pengujian dan Evaluasi
   5. Kesimpulan dan Saran
   6. Daftar Pustaka

1.6. Sistematika Penulisan

Buku Tugas Akhir ini terdiri atas beberapa bab yang tersusun secara sistematis, yaitu sebagai berikut.

1. **Bab I. Pendahuluan**
   Bab pendahuluan berisi penjelasan mengenai latar belakang masalah, rumusan masalah, batasan masalah, tujuan, manfaat dan sistematika penulisan Tugas Akhir.

2. **Bab II. Tinjauan Pustaka**
   Bab tinjauan pustaka berisi penjelasan mengenai dasar teori yang mendukung pengerjaan Tugas Akhir.

3. **Bab III. Analisis dan Perancangan**
   Bab ini berisi tentang desain sistem, rancangan basis data, diagram kasus penggunaan, diagram aktivitas dan rancangan antarmuka pengguna.

4. **Bab IV. Implementasi**
   Bab ini membahas implementasi dari desain yang telah dibuat pada bab sebelumnya. Penjelasan berupa tampilan antarmuka yang telah dibuat dan dapat berfungsi untuk mengakomodir kebutuhan fungsional yang ada.

5. **Bab V. Uji Coba dan Evaluasi**
   Bab ini menjelaskan kemampuan perangkat lunak dengan melakukan pengujian kebenaran dan pengujian kinerja dari sistem yang telah dibuat.

6. **Bab VI. Kesimpulan dan Saran**
   Bab ini merupakan bab terakhir yang menyampaikan kesimpulan dari hasil uji coba yang dilakukan dan saran untuk pengembangan perangkat lunak ke depannya.

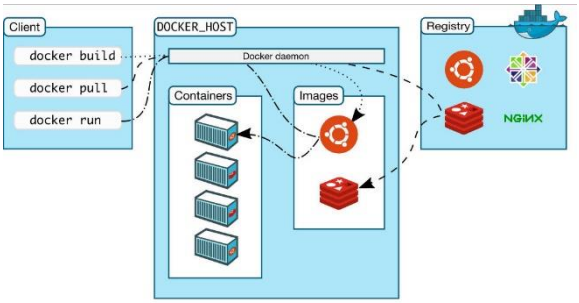
BAB II
TINJAUAN PUSTAKA

Bab tinjauan pustaka berisi pembahasan mengenai teori-teori dasar atau penjelasan dari metode dan alat yang digunakan dalam tugas akhir. Penjelasan ini bertujuan untuk memberikan gambaran secara umum terhadap program yang dibuat dan berguna sebagai penunjang dalam pengembangan riset yang berkaitan.

2.1. Docker

Docker awal mulanya dikembangkan oleh Solomon Hykes sebagai proyek internal di dotCloud, sebuah perusahaan *PaaS (platform as a service)*. Docker adalah sebuah proyek open-source yang menyediakan platform terbuka bagi developer maupun sysadmin untuk dapat membangun, mengemas, dan menjalankan aplikasi di mana pun sebagai sebuah wadah (kontainer) yang ringan.

Arsitektur docker menggunakan *client* dan **server**. Docker *client* mengirim *request* ke docker *daemon* untuk membangun, mendistribusikan, dan menjalankan kontainer docker. Docker *client* dan *daemon* dapat berjalan pada sistem yang sama. Antara docker *client* dan docker *daemon* berkomunikasi via *socket* menggunakan RESTful API.



Gambar 2.1 Arsitektur Docker

Docker memberikan beberapa keuntungan bagi pengembang perangkat lunak, termasuk dapat menggantikan peran dari VM, memudahkan pembuatan prototipe *software* yang banyak dengan setiap *software* dan *file* terkait ada di kontainer terisolasi, menyederhanakan pemaketan *software* sesuai dengan kemampuan pengembang bukan mengikuti kemampuan administrator web *hosting*, mengaplikasikan arsitektur *microservice*, memodelkan jaringan (terutama data *center*), memungkinkan produktivitas *full-stack* ketika *offline*, mengurangi biaya *debugging*, memudahkan dokumentasi ketergantungan dan *touchpoints* dari *software* memungkin *delivery* berkelanjutan, mampu melakukan *virtualisasi* pada level sistem operasi, men-*deploy* kontainer secara portabel meskipun lintas platform, menyediakan fitur pemanfaatan ulang komponen, berbagi, pengarsipan, dan *versioning* dari *image* [2].

2.2. Jenkins

Jenkins adalah sebuah *open source automation server* untuk mengotomatiskan tugas-tugas di dalam proses *continuous integration* and *delivery* sebuah perangkat lunak. Jenkins merupakan aplikasi berbasis Java yang dapat dipasang dari repositori Ubuntu atau dengan mengunduh dan menjalankan *file Web application ARchive* (WAR), sebuah koleksi *file* yang sudah lengkap dan tinggal dijalankan di sebuah *server* [3].

Kelebihan dari penggunaan Jenkins adalah mudah dipasang, memiliki lebih dari 1000 *plugin* untuk memudahkan pekerjaan, gratis, dan *portable* di semua platform dan sistem operasi [4].

2.3. Kubernetes

Kubernetes merupakan platform *open source* yang digunakan untuk melakukan manajemen *workloads* aplikasi yang dikontainerisasi, serta menyediakan konfigurasi dan otomatisasi secara deklaratif. Kubernetes berada di dalam ekosistem yang

besar dan berkembang cepat. *Service*, *support*, dan perkakas Kubernetes tersedia secara meluas.

Google membuka Kubernetes sebagai proyek *open source* pada tahun 2014. Kubernetes dibangun berdasarkan pengalaman Google selama satu setengah dekade dalam menjalankan *workloads* bersamaan dengan kontribusi berupa ide-ide terbaik yang diberikan oleh komunitas[5].

Kubernetes menyediakan manajemen *environment* yang berpusat pada kontainer. Kubernetes melakukan orkestrasi terhadap computing, networking, dan inftrastruktur penyimpanan, Kubernetes dijalankan pada tingkatan kontainer dan bukan pada tingkatan perangkat keras, Kubernetes bukanlah sistem monolitik, melainkan suatu sistem yang bersifat sebagai *bulding block* dan *pluggable* yang dapat digunakan untuk membangun sebuah platform yang dibutuhkan oleh *developer* dengan tetap mengutamakan konsep fleksibilitas. Kubernetes memiliki beberapa obyek umum, diantaranya adalah[6]:

1. **Pod**
   Pod merupakan unit terkecil yang ada di Kubernetes. *Pod* berisikan kontainer yang menjalankan suatu aplikasi dari sebuah docker *image*. *Pod* bisa menjalankan lebih dari satu kontainer. Pod memiliki tempat penyimpanan, jaringan, dan cara menjalankan yang berbeda-beda. Namun jika suatu *pod* terdiri dari banyak kontainer, maka biasanya kontainer yang satu dan yang lain akan saling berbagi sumber daya

2. **Service**
   Service digunakan untuk mengekpose *pod* supaya *pod* bisa diakses dari luar. *Service* seperti *Load Balancer* dan *pod* merupakan targetnya. Jadi satu *service* bisa mengarahkan *request* ke beberapa *pod*.

3. **Volume**
   Volume adalah sebuah direktori yang memungkinkan data di simpan di dalamnya dan terletak di luar kontainer.

4. **Namespace**
   Namespace digunakan untuk memisahkan *resource* atau *environment*. Anda dapat membuat tiap *project* di berbeda *namespace* untuk mengisolasi *project* supaya tidak saling terganggu satu sama lain. Jika anda tidak mendeklarasikan *namespace*, maka *namespace* yang akan digunakan adalah *namespace* default .

5. **Deployment**
   Deployment adalah sebuah kumpulan dari pod yang memiliki fungsi yang sama. *Deployment* bersifat *stateless* yang artinya pod yang satu dengan yang lainnya dalam satu *deployment* tidak mempengaruhi satu sama lain.

2.4. Helm

Helm berisi semua definisi sumber daya yang diperlukan untuk menjalankan aplikasi, atau layanan di dalam klaster Kubernetes[7], gabungan dari definisi sumber daya disebut dengan Helmchart, Helm dapat melakukan instalasi Helmchart kepada klaster Kubernetes, dalam artian Helm menjalankan seluruh konfigurasi Kubernetes yang sudah dibuat pada Helmchart ke sebuah klaster Kubernetes.

2.5. Scaffold

Istilah scaffold yang digunakan pada tugas akhir ini adalah suatu perancangan infrastruktur sebuah lingkungan yang digunakan oleh pengembang dalam melakukan suatu pengembangan aplikasi.

2.6. Pipeline CI/CD

Pipeline CI/CD adalah proses automasi yang dilakukan dalam pengiriman suatu versi dari aplikasi dari lingkungan pengembangan ke lingkungan produksi yang terdiri dari tahapan yang dieksekusi secara berurutan, CI/CD sendiri adalah akronim dari *continuous integration* dan *continuous delivery*.

2.7. GitHub

GitHub adalah salah satu sistem pengontrol versi (Version Control System) pada proyek perangkat lunak yang diciptakan oleh Linus Torvalds. Pengontrol versi bertugas mencatat setiap perubahan pada *file* proyek yang dikerjakan oleh banyak orang maupun sendiri. GitHub dikenal juga dengan *distributed revision control* (VCS terdistribusi), artinya penyimpanan *database* GitHub tidak hanya berada dalam satu tempat saja, github terdiri dari beberapa komponen seperti repositori dan *branch*, konsep dari repositori mirip folder utama yang menyimpan data pengguna, sedangkan cabang adalah *subfolder* atau folder dalam repositori yang umumnya digunakan untuk menyimpan versi yang berbeda pada repositori, github memiliki beberapa perintah dasar sebagai berikut:

1. **Git init**
   Menginisialisasi repositori Git baru dan mulai melacak direktori yang ada. Itu menambahkan subfolder tersembunyi di dalam direktori yang ada yang menampung struktur data internal yang diperlukan.
2. **Git clone**
   Membuat salinan lokal dari proyek yang sudah ada dari jarak jauh. Klon termasuk semua file proyek, sejarah, dan cabang.
3. **Git add**
   Mendeteksi adanya perubahan file pada suatu direktori dan menambahkan file pada daftar untuk kemudian di commit.
4. **Git commit**

Menyimpan perubahan pada file yang sebelumnya sudah di daftarkan melalui Git add.

5. **Git checkout**
   Melakukan perubahan cabang pada pada direktori ke cabang lainnya.
6. **Git pull**
   Memperbarui direktori lokal dengan versi terbaru dari sebuah cabang yang ada pada server.
7. **Git push**
   Untuk menyimpan perubahan yang telah di lakukan ke dalam sebuah repositori yang berada di git server atau *upload* ke sebuah repositori.

2.8. Laravel

Laravel merupakan framework PHP *open source* yang menekankan pada kesederhanaan dan fleksibilitas pada desainnya. Sama seperti *framework* PHP lainnya, Laravel dibangun dengan basis MVC (*Model-View-Controller*). Laravel dilengkapi *command line tool* yang bernama “Artisan” yang bisa digunakan untuk *packaging bundle* dan instalasi *bundle*. Pada laravel terdapat *routing* yang menjembatani antara *request* dari *user* dan *controller*, dengan kata lain *controller* tidak langsung menerima *request* tersebut.

2.9. MariaDB

MariaDB adalah sistem manajemen *database* relasional yang dikembangkan dari MySQL. MariaDB dikembangkan oleh komunitas pengembang yang sebelumnya berkontribusi untuk *database* MySQL.MariaDB didistribusikan secara gratis dibawah lisensi GPL (*General Public License*). Setiap pengguna dapat secara bebas menggunakan MariaDB, namun dengan batasan perangkat lunak tersebut tidak boleh dijadikan produk turunan yang bersifat komersial.

2.10. Nginx

NGINX adalah software web server yang open source. Ketika pertama kali dirilis, NGINX hanya berfungsi sebagai HTTP web serving saja. Namun sekarang, software tersebut juga berperan sebagai reverse proxy, HTTP load balancer, dan email proxy untuk IMAP, POP3, dan SMTP.

NGINX secara resmi diperkenalkan pada bulan Oktober 2004. Creator atau pembuat software ini, Igor Sysoev, memulai proyeknya di tahun 2002 untuk menjawab permasalahan C10k problem. C10k sendiri digambarkan sebagai sebuah tantangan yang dihadapi server ketika harus mengelola sepuluh ribu koneksi di waktu bersamaan. Hingga saat ini, jumlah koneksi yang dikelola web server terus bertambah. Karena itulah, NGINX menawarkan arsitektur yang event-driven dan asinkron. Adanya arsitektur ini menjadikan NGINX sebagai salah satu server yang kecepatan dan skalabilitasnya dapat diandalkan. Karena kecepatan dan kemampuannya dalam menangani jumlah koneksi yang banyak, layanan NGINX kerap digunakan oleh website dengan trafik yang tinggi.

2.11. PhpMyAdmin

PhpMyAdmin merupakan aplikasi web yang bersifat *open source* (sumber terbuka) sejak pertama dibuat dan dikembangkan. Dengan dukungan dari banyak *developer* dan translator, aplikasi web PhpMyAdmin mengalami perkembangan yang cukup pesat dengan ketersediaan banyak pilihan bahasa. Sampai saat ini, ada kurang lebih 65 bahasa yang sudah didukung oleh aplikasi web PhpMyAdmin. PhpMyAdmin menawarkan fitur yang mencangkup pengelolaan keseluruhan server MySQL (memerlukan *super-user*) dan basis data tunggal. PhpMyAdmin juga mempunyai sistem internal untuk mengelola metadata dan mendukung fitur-fitur untuk operasi tingkat lanjut. Melalui sistem administrator, PhpMyAdmin juga dapat mengelola users dan sekaligus hak aksesnya (*privilage*).

2.12. Docker Hub

Docker Hub adalah repositori berbasis *cloud* tempat pengguna membuat, menguji, menyimpan, dan mendistribusikan image. Melalui Docker Hub, pengguna dapat mengakses repositori open source image, dan juga digunakan untuk membuat repositori pribadi mereka sendiri. Sebagai contoh, seorang profesional *DevOps* dapat mengunduh *image* resmi sistem manajemen database objek relasional PostgreSQL dari Docker Hub untuk digunakan dalam aplikasi yang digunakan dalam kontainer. Atau, mereka dapat memilih RDBMS yang disesuaikan dari repositori pribadi perusahaan mereka.

2.13. Chartmuseum

Chartmuseum adalah repositori berbasis *cloud* tempat penyimpanan Helmchart. Pengguna Helmchart dapat membuat repositori pribadi yang dapat dibagikan saat ingin melalukan *deployment*. Chartmuseum dapat berjalan pada kontainer sehingga Chartmuseum dapat dijalankan di manapun, Chartmuseum dapat diakses melalui API dan CLI.

2.14. Rancher

Rancher adalah platform pengelolaan kontainer opensource. Ini memungkinkan kamu untuk menjalankan dan mengelola Docker dan Kubernetes dengan mudah. Rancher menyediakan layanan infrastruktur seperti jaringan *multi-host*, *load balancing*, dan *snapshot volume*.

2.15. Docker Compose

Docker Compose adalah sebuah *tools* yang dapat menjalankan banyak kontainer secara bersamaan, Docker Compose didefinisikan dalam *file* konfigurasi berformat docker-compose.yaml, pengguna dapat mendefinisikan *image* apa saja yang ingin dijalankan sebagai kontainer beserta konfigurasinya.

Docker Compose dapat membuat lingkungan yang terisolasi dengan lingkungan lainnya. Pada praktiknya, pengguna dapat

membuat banyak salinan dari suatu lingkungan aplikasi yang sama dengan versi yang berbeda secara bersamaan, docker compose akan mempertahankan volume yang dibuat dimulai dari dijalankannya docker-compose up, volume yang ada tidak akan hilang jika pengguna menghentikan dan menjalankan kembali container, volume akan hilang ketika perintah docker-compose down dijalankan[8].

[Halaman ini sengaja dikosongkan]

BAB III
ANALISIS DAN PERANCANGAN

Bab ini membahas mengenai perancangan implementasi sistem yang dibangun pada tugas akhir. Bagian yang akan dijelaskan pada bab ini berawal dari deskripsi umum sistem, alur, perancangan pengujian, hingga gambaran implementasi sistem.

3.1. Deskripsi Umum

Pada tugas akhir ini akan dilakukan perancangan *scaffolding* lingkungan pengembangan menggunakan *docker container* serta melakukan integrasi melalui *pipeline CI/CD* dari lingkungan pengembang kepada lingkungan produksi platform myITS menggunakan Jenkins.

Pada penerapan *scaffolding* untuk lingkungan pengembangan, seluruh aplikasi yang dibutuhkan dalam proses pengembangan akan dijalankan di dalam kontainer, konfigurasi kontainer docker akan dikemas dalam *docker compose*, *docker compose* dapat menjalankan lebih dari satu *docker container* dalam waktu bersamaan[8]. hal tersebut bertujuan untuk menjaga standar suatu lingkungan pengembang supaya tetap konsisten sesuai dengan lingkungan produksi,.

Dalam proses integrasi, Jenkins akan berperan sebagai *integrator* utama dari kedua lingkungan, proses utama dari *pipeline* dieksekusi pada Jenkins. Proses berjalannya *pipeline* diawali dengan *programmer* yang melakukan *commit* kode sumber terbaru pada repositori GitHub, *webhook* Jenkins yang terhubung pada GitHub akan mendeteksi perubahan yang terjadi, setelah perubahan terdeteksi Jenkins akan menjalankan serangkaian perintah untuk melakukan *pull dan build image* terbaru dari aplikasi, kemudian menyimpannya pada *registry* DockerHub, selanjutnya Jenkins akan mengeksekusi perintah Helm untuk melakukan pemasangan Helmchart yang merupakan konfigurasi dari *deployment* Kubernetes. Helm akan melakukan pemasangan ataupun perbaruan dari sebuah aplikasi mengikuti konfigurasi yang ada pada Helmchart.

Pada tugas akhir ini studi kasus yang digunakan adalah aplikasi *Url Shortener* berbasis *web* menggunakan *framework* PHP Laravel dijalankan oleh *web server* Nginx. *database* yang akan digunakan adalah MariaDB, serta PhpMyAdmin untuk memudahkan pengelolaan *database*.

3.2. Arsitektur Sistem

Pada sub-bab ini, dibahas mengenai tahap analisis arsitektur, analisis teknologi, dan desain sistem yang akan dibangun. Berikut adalah gambar arsitektur komponen-komponen yang ada dalam sistem.

3.2.1. Desain Umum

Berdasarkan deskripsi umum sistem yang telah ditulis diatas, dapat diperoleh kebutuhan sistem ini, diantaranya:

1. Konfigurasi *docker compose* pada *scaffolding* lingkungan pengembang.
2. Konfigurasi Helmchart untuk melakukan *deployment* aplikasi pada Kubernetes.
3. Konfigurasi Jenkinsfile untuk *Pipeline CI/CD*.

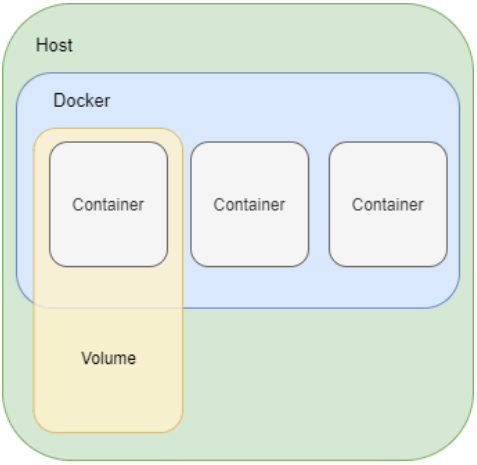
3.2.2. Perancangan Scaffolding

Pada komponen ini akan dibuat *scaffold* menggunakan konfigurasi *docker compose* untuk melakukan kontainerisasi lingkungan produksi pada lingkungan kerja *programmer*. Dengan itu lingkungan di mana aplikasi dikembangkan akan serupa dengan lingkungan produksi, hal tersebut dapat memudahkan dalam proses integrasi, mengurangi potensi terjadinya *error* pada lingkungan produksi, dan memberikan kemudahan pada *programmer* dalam menginisiasi lingkungan pengembangan.

Aplikasi *Url Shortener* membutuhkan beberapa teknologi untuk berjalan, seperti *web server*, *package manager*, *database*, dan bahasa pemrograman, dengan itu *programmer* perlu membuat tiga kontainer untuk menjalankan aplikasi *Url*

Shortener, kontainer untuk menjalankan *database*, kontainer untuk PhpMyAdmin dan kontainer untuk aplikasi itu sendiri.

Kontainer yang pertama akan berisikan web server berupa Nginx, *package manager* berupa *composer*, dan bahasa pemrograman Php versi 7, kemudian kontainer akan dihubungkan dengan *filesystem* menggunakan *volume* pada docker sehingga kode pada komputer pengembang dapat terhubung dengan kontainer yang berjalan[9], kontainer yang kedua akan berisikan MariaDB sebagai *database* aplikasi, dan PhpMyAdmin pada kontainer ke tiga. Diagram dari arsitektur scaffold dapat dilihat pada Gambar 3.1 Arsitektur Scaffold



Gambar 3.1 Arsitektur Scaffold

3.2.3. Perancangan Deployment Kubernetes

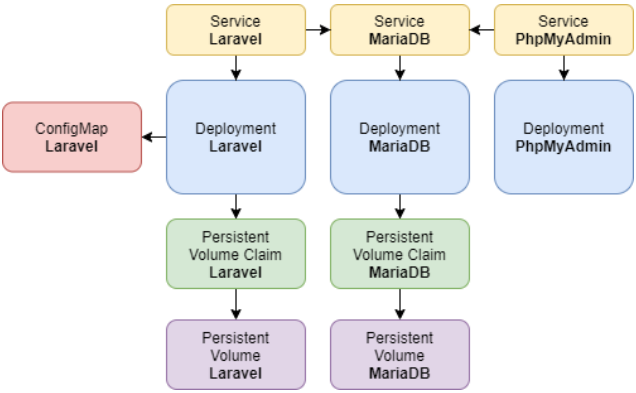
Pada komponen ini akan dirancang konfigurasi Kubernetes *deployment* untuk menjalankan tiga aplikasi pada lingkungan produksi, di antaranya adalah aplikasi *Url Shortener*, *database* MariaDB dan aplikasi pengelola *database* PhpMyAdmin.

Kubernetes *deployment* terdiri dari kumpulan *pod*, *pod* adalah kumpulan kontainer yang menjalankan *image*, disini *image* akan dijalankan sebagai kontainer yang dikemas dalam sebuah *pod* yang berada pada sebuah Kubernetes *deployment*[10].

Lingkungan produksi memiliki konfigurasi aplikasi yang berbeda dengan linkungan pengembangan, aplikasi pada lingkungan produksi akan menggunakan koneksi *database* produksi, konfigurasi tersebut bisa dipisahkan dalam bentuk *configmap* pada Kubernetes, *configmap* berguna untuk mendefinisikan konfigurasi global pada sebuah aplikasi, *configmap* bersifat *reusable* sehingga bisa digunakan oleh *deployment* lain jika memiliki konfigurasi yang sama[11].

Untuk menyimpan *file* statis dibutuhkan direktori terpusat yang nantinya akan digunakan oleh *deployment*, hal tersebut dapat dicapai menggunakan *persistent* volume dan *persistent* volume *claim* pada Kubernetes, *persistent* volume berguna untuk mendefinisikan direktori mana yang akan digunakan pada *node*, sedangkan *persistent* volume *claim* berguna untuk mendefinisikan direktori mana yang akan dihubungkan dengan *persistent* volume pada *deployment*[12].

Selanjutnya, untuk menghubungkan antar *deployment* memerlukan *service*, *service* berguna untuk mengekspose suatu *port* pada sebuah *pod*, hal ini dapat digunakan untuk menghubungkan ketiga *deployment*, *service* aplikasi url shortener dan aplikasi PhpMyAdmin akan mengakses *service database* MariaDB. Selanjutnya konfigurasi akan dikemas dalam sebuah paket yang bernama Helmchart, kemudian Helmchart akan disimpan pada repositori bernama Chartmuseum yang selanjutnya akan di *deploy* secara berkala melalui Jenkins. Arsitektur Kubernetes dari aplikasi url shortener dapat dilihat pada



Gambar 3.2 Aristektur Deployment Url Shortene

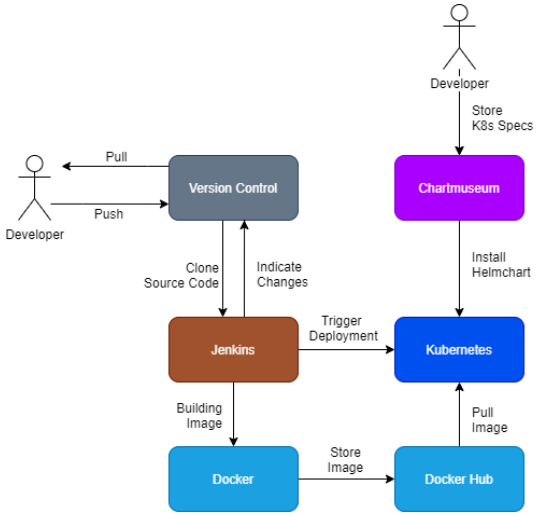
3.2.4. Perancangan Pipeline

Setelah lingkungan pengembang dan produksi telah terancang, *pipeline* akan dibangun untuk mengintegrasikan kedua lingkungan tersebut. *Pipeline* terdiri dari beberapa komponen seperti Jenkins, GitHub, DockerHub, Helm, Chartmuseum dan klaster Kubernetes.

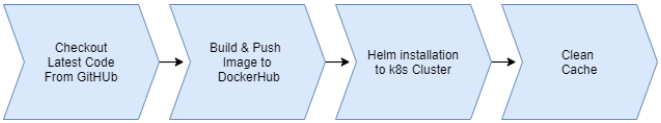
Jenkins menjadi komponen utama pada rancangan *pipeline* ini, Jenkins akan berjalan pada klaster Kubernetes yang nantinya akan terhubung dengan GitHub, DockerHub, Helm, dan Chartmuseum. Jenkins akan menjalankan perintah yang ditulis pada file bernama Jenkinsfile. Jenkinsfile yang digunakan pada *pipeline* ini secara garis besar berisikan tahapan perintah yang dibagi menjadi empat tahapan pada Gambar 3.4 Alur Step Pada Jenkinsfile:

1. Melakukan *checkout* dan *clone* kode terbaru pada branch yang sudah ditentukan.
2. Melakukan build Dockerfile menjadi docker *image* dan melakukan push ke repositori DockerHub.
3. Melakukan deployment aplikasi ke Kubernetes cluster.
4. Membersihkan *cache* dari proses *build*

Dapat dilihat pada Gambar 3.3 Arsitektur Alur Pipeline, *pipeline* dimulai ketika *programmer* melakukan push pada repositori github, *webhook* mendeteksi adanya perubahan dan jenkins menjalankan *step pipeline* yang sudah didefinisikan pada Jenkinfile, *image* di *build* pada *worker* docker kemudian disimpan pada registry dockerhub, setelah proses tersebut selesai jenkins akan menjalankan perintah Helm install yang membuat klaster kubernetes mengimplementasikan spesifikasi yang sudah didefinisikan pada Helmchart.



Gambar 3.3 Arsitektur Alur Pipeline



Gambar 3.4 Alur Step Pada Jenkinsfile

BAB IV
IMPLEMENTASI

Bab ini membahas implementasi yang dilakukan berdasarkan rancangan yang telah dijabarkan pada bab sebelumnya. Sebelum penjelasan implementasi akan ditunjukkan terlebih dahulu lingkungan untuk melakukan implementasi.

Pada bagian implementasi ini juga akan dijelaskan mengenai fungsi-fungsi yang digunakan dalam program Tugas Akhir ini dan disertai dengan kode semu masing-masing fungsi utama.

4.1. Lingkungan Pengembangan Sistem

Pada tugas akhir ini, digunakan 1 buah komputer, yang digunakan oleh pengembang dan 1 buah klaster Kubernetes.

4.1.1. Perangkat Keras

Perangkat keras yang digunakan dalam pengembangan sistem adalah sebagai berikut:

1. Komputer dengan *processor* Intel(R) Core(TM) i5-9400F CPU @ 4.1GHz, 16GB RAM, 900GB penyimpanan.
2. Node dengan 8 *Core*, 16GB RAM, 60GB penyimpanan.

Berikut adalah alamat dan peran dari pada klaster Kubernetes yang dapat dilihat pada Tabel 4.1 Daftar *Node*:

Tabel 4.1 Daftar *Node*

Nama Node	Peran	Alamat
Master	Master	k8s.its.ac.id
Node	Node	10.199.16.58

4.1.2. Perangkat Lunak

Perangkat lunak yang digunakan pada pengembangan sistem adalah sebagai berikut:

1. Sistem Operasi Windows 10 Pro 1909
2. Docker versi 19.03.8
3. Jenkins 2.236 untuk *pipeline*
4. Rancher 3 untuk mengelola klaster Kubernetes
5. Laravel 5.8 untuk kerangka kerja *web service*
6. PHP 7.2 untuk pengembang *web service*
7. Nginx untuk *web server*
8. PhpMyAdmin untuk pengelolaan database
9. MariaDB untuk database web *service*
10. Helm untuk menyimpan konfigurasi Kubernetes
11. DockerHub untuk menyimpan *image* docker

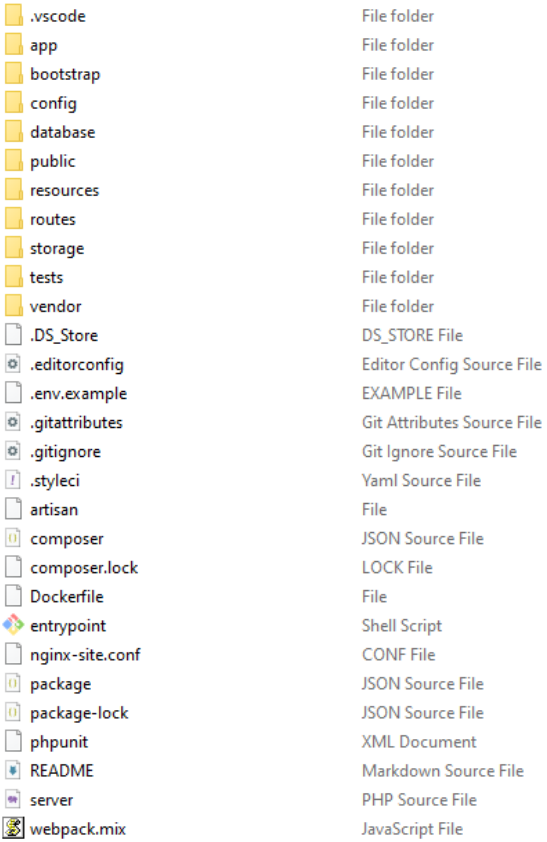
4.2. Implementasi Scaffold

Infrastruktur pengembang akan berjalan pada kontainer docker, hal ini ditujukan supaya lingkungan di mana aplikasi dibuat memiliki kesamaan dengan lingkungan produksi, penggunaan kontainer docker juga dapat mempermudah *programmer* dalam melakukan membangun lingkungan kerja.

4.2.1. Implementasi Dockerfile

Sesuai dengan penjelasan di bagian sebelumnya, aplikasi ini terbagi menjadi tiga kontainer, MariaDB, PhpMyAdmin, dan Laravel, pada tahap ini Dockerfile akan dibuat untuk menjalankan aplikasi Laravel pada kontainer docker.

Daftar direktori dari aplikasi *Url Shortener* dapat dilihat pada Gambar 4.1. Aplikasi *Url Shortener* dapat berjalan pada lingkungan yang sudah memiliki *web server* Nginx, Bahasa pemrograman PHP versi 7.4 dan Composer, oleh karena itu *base image* yang digunakan disini adalah php:7.4-fpm, selanjutnya web server Nginx dan Composer akan ditambahkan pada *base image* melalui Dockerfile.



Gambar 4.1 Direktori Laravel

file: Dockerfile

```
FROM php:7.4-fpm

RUN apt-get update -y \
    && apt-get install -y nginx \
    && apt-get install -y \
    git \
    zip \
    nano \
    curl
```

```
# PHP_CPPFLAGS are used by the docker-php-ext-*
scripts
ENV PHP_CPPFLAGS="$PHP_CPPFLAGS -std=c++11"

RUN docker-php-ext-install pdo_mysql \
    && docker-php-ext-install opcache \
    && apt-get install libicu-dev -y \
    && docker-php-ext-configure intl \
    && docker-php-ext-install intl \
    && apt-get remove libicu-dev icu-devtools -y

RUN curl -sS https://getcomposer.org/installer |
php
RUN mv composer.phar /usr/local/bin/composer
RUN chmod +x /usr/local/bin/composer

RUN { \
        echo 'opcache.memory_consumption=128'; \
        echo 'opcache.interned_strings_buffer=8'; \
        echo 'opcache.max_accelerated_files=4000';
\
        echo 'opcache.revalidate_freq=2'; \
        echo 'opcache.fast_shutdown=1'; \
        echo 'opcache.enable_cli=1'; \
    } > /usr/local/etc/php/conf.d/php-opocache-
cfg.ini

COPY nginx-site.conf /etc/nginx/sites-
enabled/default
COPY entrypoint.sh /etc/entrypoint.sh

COPY --chown=www-data:www-data . /var/www/mysite

WORKDIR /var/www/mysite

RUN composer install

EXPOSE 80 443
RUN chmod +x /etc/entrypoint.sh
ENTRYPOINT ["/etc/entrypoint.sh"]
```

Kode Sumber 4.1 Dockerfile

file: nginx-site.conf

```
server {
    root    /var/www/mysite/public;

    include /etc/nginx/default.d/*.conf;

    index index.php index.html index.htm;

    client_max_body_size 30m;

    location / {
        try_files $uri $uri/
/index.php$is_args$args;
    }

    location ~ [^/]\.php(/|$) {
        fastcgi_split_path_info ^(.+?\.php)(/.*)$;
        # Mitigate https://httpoxy.org/
vulnerabilities
        fastcgi_param HTTP_PROXY "";
        fastcgi_pass 127.0.0.1:9000;
        fastcgi_index index.php;
        include fastcgi.conf;
    }
}
```

Kode Sumber 4.2 nginx-site.conf

file: entrypoint.sh

```
#!/usr/bin/env bash
service nginx start
php-fpm
```

Kode Sumber 4.3 entrypoint.sh

Terdapat tiga *file* yang akan digunakan dalam proses pembuatan *image*, yaitu Kode Sumber 4.3 entrypoint.sh sebagai *file* yang dieksekusi Ketika *image* dijalankan, Kode Sumber 4.2 nginx-site.conf adalah *file* konfigurasi *web server* nginx dan Kode Sumber 4.1 Dockerfile yang akan digunakan pada konfigurasi docker compose untuk di *build* dan kemudian dijalankan pada lingkungan pengembangan.

4.2.2. Implementasi Docker Compose

Pada tahap ini seluruh kontainer yang dibutuhkan akan didefinisikan pada file docker-compose.yaml. direktori pada Gambar 4.1 Direktori Laravel akan diletakan pada sub direktori /code supaya direktori tetap rapih dan memudahkan dalam penulisan docker-compose.yaml, susunan direktori dapat dilihat pada Gambar 4.2 Direktori Docker-Compose

Gambar 4.2 Direktori Docker-Compose

File Kode Sumber 4.4 Docker-Compose.yaml berisikan definisi ketiga kontainer beserta konfigurasinya, masing masing kontainer bernama c-app berisikan Laravel, Php.7.2, dan Nginx, c-db berisikan MariaDB, c-pma berisikan aplikasi PhpMyAdmin, volume didefinisikan pada c-app untuk menghubungkan kode sumber pada direktori ./code menuju direktori /var/www/html pada kontainer sehingga perubahan kode sumber pada komputer pengembang dapat secara langsung dibaca dan dijalankan oleh *web server* Nginx.

file: docker-compose.yaml

```
version: "3.1"
services:

    mariadb:
      container_name: c-db
      restart: always
      image: mariadb:latest
      ports:
        - "3306:3306"
      environment:
        - MYSQL_DATABASE=laravel
```

```
        - MYSQL_ROOT_PASSWORD=root

    pma:
      container_name: c-pma
      restart: always
      image: phpmyadmin/phpmyadmin:latest
      ports:
        - "8080:80"
      environment:
        - PMA_HOST=mariadb

    app:
      container_name: c-app
      build: code
      restart: always
      ports:
        - "80:80"
      volumes:
        - ./code:/var/www/html
```

Kode Sumber 4.4 Docker-Compose.yaml

Pengembang dapat langsung menjalankan perintah docker-compose up -d untuk menginisiasi lingkungan kerja pada komputer pengembang, pengembang dapat menjalankan perintah *docker-compose stop* untuk mematikan lingkungan kerja, pengembang dapat menjalankan perintah *docker-compose start* untuk menghidupkan kembali lingkungan kerja, dan terakhir *docker-compose down* untuk menghancurkan lingkugan kerja.

terminal: Pengembang

```
$ docker-compose up -d
Creating network "its-url-shortener_default" with
the default driver
Creating c-db  ... done
Creating c-pma ... done
Creating c-app ... done
```

Terminal 4.1 Menjalankan Docker-Compose Up

terminal: Pengembang

`$ docker-compose stop`
`Stopping c-app ... done`
`Stopping c-pma ... done`
`Stopping c-db  ... done`

Terminal 4.2 Menjalankan Docker-Compose Stop

terminal: Pengembang

`$ docker-compose start`
`Starting mariadb ... done`
`Starting pma     ... done`
`Starting app     ... done`

Terminal 4.3 Menjalankan Docker-Compose Start

terminal: Pengembang

`$ docker-compose down`
`Stopping c-app ... done`
`Stopping c-pma ... done`
`Stopping c-db  ... done`
`Removing c-app ... done`
`Removing c-pma ... done`
`Removing c-db  ... done`
`Removing network its-url-shortener_default`

Terminal 4.4 Menjalankan Docker-Compose Down

Sesuai dengan konfigurasi pada Kode Sumber 4.4 Docker-Compose.yaml aplikasi akan berjalan pada tiga port, 80 untuk Laravel, 8080 untuk phpyadmin, dan 3306 untuk MariaDB. Ketika mengakses port 80 akan tampil error seperti pada Gambar 4.3 *Error Browser*, hal itu menandakan package dan konfigurasi yang dibutuhkan laravel untuk berjalan belum diinisiasikan. Jalankan perintah pada kontainer c-app untuk menginisiasi package dan konfigurasi yang dibutuhkan, selanjutnya ubah file .env pada direktori /code/.env menyesuaikan dengan database yang berjalan pada kontainer c-db, kofigurasi .env dapat dilihat pada Kode Sumber 4.5 *File* .env. Terakhir jalankan perintah untuk melakukan migrasi database Terminal 4.6 Menjalankan Perintah Migrasi *Database*. Perintah inisiasi dapat dilihat pada Terminal 4.5 Menjalankan Perintah Inisiasi *Project*.

terminal: Pengembang

`$ docker exec c-app composer install`
`$ docker exec c-app cp .env.example .env`
`$ docker exec c-app php artisan key:generate`

Terminal 4.5 Menjalankan Perintah Inisiasi *Project*

Perintah pada Terminal 4.5 Menjalankan Perintah Inisiasi *Project* diawali dengan docker exec c-app dikarenakan perintah tersebut akan di eksekusi pada kontainer tersebut, untuk menjalankan suatu perintah pada kontainer yang berjalan adalah dengan mengikuti format berikut “docker exec <nama-container> <perintah>”.

file: ./code/.env

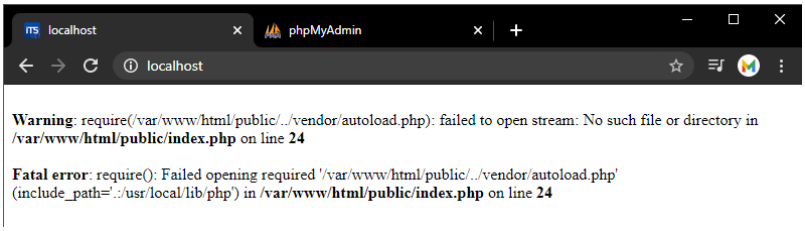
```
APP_NAME=Laravel
APP_ENV=local
APP_KEY=base64:2xtitdrlF/66ry+saKbOGpwhc3Oxj6Wb2Wa9U
4tRj8w=
APP_DEBUG=true
APP_URL=http://localhost

LOG_CHANNEL=stack

DB_CONNECTION=mysql
DB_HOST=mariadb
DB_PORT=3306
DB_DATABASE=laravel
DB_USERNAME=root
DB_PASSWORD=root

BROADCAST_DRIVER=log
CACHE_DRIVER=file
QUEUE_CONNECTION=sync
SESSION_DRIVER=file
SESSION_LIFETIME=120

REDIS_HOST=127.0.0.1
REDIS_PASSWORD=null
REDIS_PORT=6379
```

```
MAIL_DRIVER=smtp
MAIL_HOST=smtp.mailtrap.io
MAIL_PORT=2525
MAIL_USERNAME=null
MAIL_PASSWORD=null
MAIL_ENCRYPTION=null

AWS_ACCESS_KEY_ID=
AWS_SECRET_ACCESS_KEY=
AWS_DEFAULT_REGION=us-east-1
AWS_BUCKET=

PUSHER_APP_ID=
PUSHER_APP_KEY=
PUSHER_APP_SECRET=
PUSHER_APP_CLUSTER=mt1

MIX_PUSHER_APP_KEY="${PUSHER_APP_KEY}"
MIX_PUSHER_APP_CLUSTER="${PUSHER_APP_CLUSTER}"
```

Kode Sumber 4.5 *File* .env

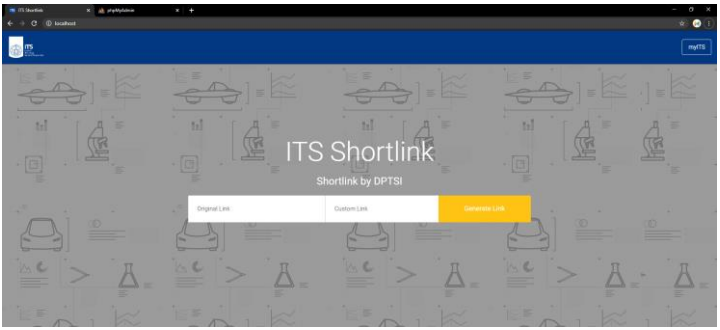
terminal: Pengembang
`$ docker exec c-app php artisan migrate`

Terminal 4.6 Menjalankan Perintah Migrasi *Database*



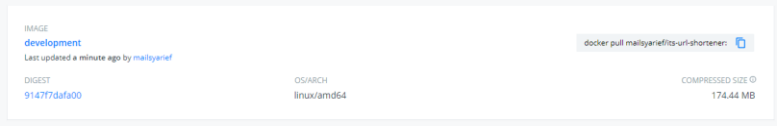
Gambar 4.3 *Error Browser*

Dapat dilihat pada Gambar 4.4 Aplikasi Berjalan aplikasi sudah berjalan pada *browser* pengembang, perubahan yang dilakukan oleh pengembang dapat langsung terlihat hasilnya.



Gambar 4.4 Aplikasi Berjalan

Langkah selanjutnya adalah melakukan *build* Dockerfile menjadi *image* kemudian disimpan pada DockerHub, hal ini bertujuan untuk menjadikan *image* tersebut sebagai *base image* pada Dockerfile yang akan digunakan pada Jenkins *pipeline*. Untuk melakukan *build* dan *push* ke DockerHub dapat dilihat pada kode Terminal 4.7 Melakukan *Build* & Push *Image image* yang sudah di *push* ke DockerHub dapat dilihat pada Gambar 4.5 Isi Registry.



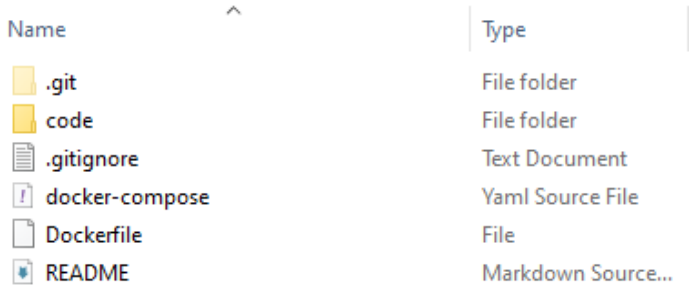
Gambar 4.5 Isi Registry

terminal: Pengembang
`$ docker build -t mailsyarief/its-url-shortener:development ./code`

```
$ docker push mailsyarief/its-url-
shortener:development
```

Terminal 4.7 Melakukan *Build* & Push *Image*

Terdapat dua perintah berbeda pada Terminal 4.7 Melakukan *Build* & Push *Image*, perintah pertama untuk melakukan build dan perintah kedua untuk melakukan push ke registry Dockerhub. Setelah berhasil membuat *base image* dan menyimpannya pada DockerHub, Dockerfile perlu kembali dibuat, Kode Sumber 4.6 Dockerfile akan melakukan modifikasi pada *base image* mailsyarief/its-url-shortener:development. Susunan direktori dapat dilihat pada Gambar 4.6 Direktori Project



Gambar 4.6 Direktori Project

file: Dockerfile

```
FROM mailsyarief/its-url-shortener:development

WORKDIR /var/www/html
RUN ls -a -l
COPY /code /var/www/html
RUN ls -a -l
RUN chown -R www-data:www-data
/var/www/html/storage
RUN a2enmod rewrite

RUN composer update
```

Kode Sumber 4.6 Dockerfile

Seluruh *file* yang sudah dibuat disimpan pada GitHub repositori, nantinya dapat digunakan oleh pengembang lain untuk dikembangkan, dengan begitu proyek aplikasi ini dapat dengan mudah dijalankan oleh pengembang lain hanya dengan menjalankan beberapa perintah.

4.3. Implementasi Deployment

Sesuai dengan tahap perancangan, lingkungan yang digunakan adalah klaster Kubernetes, aplikasi *Url Shortener* beserta aplikasi pendukungnya akan dijalankan pada klaster Kubernetes. Rancher digunakan untuk manajemen klaster Kubernetes, dengan menggunakan rancher pengguna dapat dengan mudah mengelola klaster Kubernetes melalui GUI.

Pada lingkungan produksi Helmchart digunakan untuk menyimpan konfigurasi dari aplikasi yang akan berjalan pada klaster Kubernetes, konfigurasi yang telah dikemas dalam Helmchart akan disimpan pada Chartmuseum[13], Chartmuseum digunakan untuk memudahkan pengembang untuk melakukan distribusi dan mengelola satu atau lebih Helmchart.

4.3.1. Implementasi Helm

Suatu *deployment* pada Kubernetes dapat disimpan dalam satu atau lebih *file* .yaml, ketika *file* tersebut dijalankan, secara otomatis konfigurasi akan berjalan pada klaster Kubernetes, namun hal tersebut memiliki limitasi dalam fleksibilitas dan kemudahan dalam pendistribusian, dengan itu pada tugas akhir ini helm digunakan untuk menyimpan konfigurasi Kubernetes dikarenakan kemampuannya untuk merubah .yaml yang statis menjadi dinamis dan kemudahan dalam pendistribusian karena dapat disimpan pada sebuah repositori.

Sesuai dengan tahap perancangan, Helmchart akan dibuat untuk menjalankan *image* pada klaster Kubernetes, yaitu *image* Laravel, MariaDB, dan PhpMyAdmin. Pastikan helm sudah terpasang dan dapat terhubung dengan klaster Kubernetes,

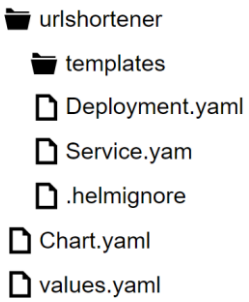
jalankan perintah seperti pada Terminal 4.8 Inisiasi Helmchart untuk menginisiasi Helmchart.

terminal: Pengembang

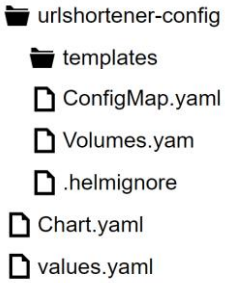
`$ helm init urlshortener`
`$ helm init urlshortener-config`
`$ helm init phpmyadmin`
`$ helm init mariadb`

Terminal 4.8 Inisiasi Helmchart

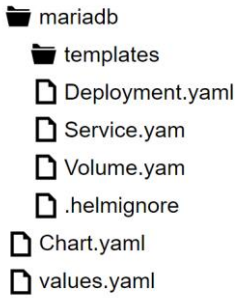
Setelah perintah dijalankan, direktori Helmchart akan tergenerate, direktori tersebut berisikan direktori default yang dapat diubah menyesuaikan kebutuhan pengguna, pada tahap ini Helmchart akan dimodifikasi menyesuaikan *image* yang akan dijalankan, Helmchart urlshortener terdiri dari *Deployment* dan *Service*, Helmchart urlshortener-config terdiri dari *ConfigMap* dan *Volume*, Helmchart PhpMyAdmin terdiri dari *Deployment*, *Service* dan *Volume*, dan Helmchart MariaDB hanya terdiri dari *Deployment* dan *Service*. Berikut adalah isi dari direktori dari seluruh Helmchart.



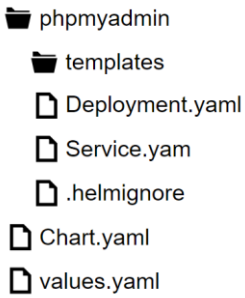
Gambar 4.7 Direktori Helmchart Urlshortener



Gambar 4.8 Direktori Helmchart Config Urlshortener



Gambar 4.9 Direktori Helmchart Mariadb



Gambar 4.10 Direktori Helmchart Phpmyadmin

File Chart.yaml berisikan deskripsi dan deklarasi versi dari sebuah Helmchart, Values.yaml berisikan variabel global yang dapat digunakan oleh file yaml pada direktori /templates, berikut adalah isi file dari file pada Helmchart untuk tiap *image*.

file: /urlshortener/Chart.yaml

```
apiVersion: v2
name: shorten-app
description: DPTSI Url Shortener Dev
type: application
version: 0.1.0
appVersion: "dev"
```

Kode Sumber 4.7 Chart.yaml Urlshortener

file: /urlshortener/values.yaml

```
image:
  name: mailsyarief/its-url-shortener:development
  version: v1
timestamp: "000000"

application:
  name: shorten-app
  buildNumber: v2
  buildNumberBefore: v1
  initialDelaySeconds: 45

uri:
  appprefix: /

host: "*"
```

Kode Sumber 4.8 values.yaml Urlshortener

file: /urlshortener/templates/Deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
    name: {{.Values.application.name}}-
         {{.Values.application.buildNumber}}
    labels:
```

```
        app: {{.Values.application.name}}
spec:
    selector:
        matchLabels:
            app: {{.Values.application.name}}
            version:
         {{.Values.application.buildNumber}}
    template:
        metadata:
            labels:
                app: {{.Values.application.name}}
                version:
         {{.Values.application.buildNumber}}
        spec:
            containers:
            - name: c-shorten
              image: {{.Values.image.name}}
              ports:
              - containerPort: 80
              volumeMounts:
              - name: config
                mountPath: /var/www/mysite/.env
                subPath: .env
              - name: public
                mountPath:
         /var/www/mysite/public/storage
                subPath: storage
              readinessProbe:
                httpGet:
                  path: /
                  port: 80
                initialDelaySeconds:
         {{.Values.application.initialDelaySeconds}}
                periodSeconds: 10
                timeoutSeconds: 3
                successThreshold: 1
                failureThreshold: 3
            volumes:
            - name: public
              persistentVolumeClaim:
                claimName:
         {{.Values.application.name}}-pvc
            - name: config
              configMap:
```

```
                name: {{.Values.application.name}}-
        configmap
                items:
                - key: config
                  path: .env
```

Kode Sumber 4.9 Deployment.yaml Urlshortener

file: /urlshortener/templates/Service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: {{.Values.application.name}}-nodeport-
        service-{{.Values.application.buildNumber}}
  labels:
    app: {{.Values.application.name}}
spec:
    type: NodePort
    ports:
    - nodePort: 30010
      port: 80
      protocol: TCP
      targetPort: 80
  selector:
    app: {{.Values.application.name}}
    version: {{.Values.application.buildNumber}}
```

Kode Sumber 4.10 Service.yaml Urlshortener

file: /urlshortener-config/Chart.yaml

```
apiVersion: v2
name: shorten-app-config
description: DPTSI Url Shortener Configuration
type: application
version: 0.1.0
appVersion: "dev"
```

Gambar 4.11 Chart.yaml Urlshortener-config

file: /urlshortener-config/values.yaml

```
application:
  name: shorten-app
  buildNumber: v2
  buildNumberBefore: v1
  initialDelaySeconds: 45
```

Gambar 4.12 Values.yaml Urlshortener-config

file: /urlshortener-config/templates/Volume.yaml

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: {{.Values.application.name}}-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
---
apiVersion: v1
kind: PersistentVolume
metadata:
  name: {{.Values.application.name}}-pv
spec:
  accessModes:
    - ReadWriteOnce
  capacity:
    storage: 1Gi
  hostPath:
    path: "/mnt"
```

Kode Sumber 4.11 Volume.yaml Urlshortener

file: /urlshortener-config /templates/ConfigMap.yaml

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{.Values.application.name}}-configmap
data:
```

```
  config : |
    APP_NAME=Laravel
    APP_ENV=local

         APP_KEY=base64:JROLVxxVCG3j+e5vRzbXPqE/GC58LA
         65gsFHbKkIGco=
    APP_DEBUG=true
    APP_URL=http://localhost
    LOG_CHANNEL=stack
    DB_CONNECTION=mysql
    DB_HOST=database-db-service
    DB_PORT=3306
    DB_DATABASE=laravel
    DB_USERNAME=root
    DB_PASSWORD=root
    BROADCAST_DRIVER=log
    CACHE_DRIVER=file
    QUEUE_CONNECTION=sync
    SESSION_DRIVER=file
    SESSION_LIFETIME=120
    REDIS_HOST=127.0.0.1
    REDIS_PASSWORD=null
    REDIS_PORT=6379
    MAIL_DRIVER=smtp
    MAIL_HOST=smtp.mailtrap.io
    MAIL_PORT=2525
    MAIL_USERNAME=null
    MAIL_PASSWORD=null
    MAIL_ENCRYPTION=null
---
```

Kode Sumber 4.12 ConfigMap.yaml Urlshortener

file: /mariadb/Chart.yaml

```
apiVersion: v2
name: shorten-mariadb
description: DPTSI Url Shortener Dev
type: application
version: 0.1.0
appVersion: "dev"
```

Kode Sumber 4.13 Chart.yaml MariaDB

file: /mariadb/Values.yaml

```
env:
  database: laravel
  password: root
```

Kode Sumber 4.14 Values.yaml MariaDB

file: /mariadb/templates/Deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
    name: shorten-db
spec:
    selector:
        matchLabels:
            app: shorten-db
    template:
        metadata:
            labels:
                app: shorten-db
        spec:
            volumes:
            - name: storage
              persistentVolumeClaim:
                claimName: pvc-db
            containers:
            - name: c-mariadb
              image: mariadb
              volumeMounts:
              - name: storage
                mountPath: /var/lib/mysql
              ports:
              - containerPort: 3306
              env:
              - name: MYSQL_DATABASE
                value: {{.Values.env.database}}
              - name: MYSQL_ROOT_PASSWORD
                value: {{.Values.env.password}}
```

Kode Sumber 4.15 Deployment.yaml MariaDB

file: /mariadb/templates/Service.yaml

```
apiVersion: v1
kind: Service
metadata:
    name: shorten-db
spec:
    ports:
    - port: 3306
    selector:
        app: shorten-db
```

Kode Sumber 4.16 Service.yaml MariaDB

file: /mariadb/templates/Volumes.yaml

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-db
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi

---

apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-db
spec:
  accessModes:
    - ReadWriteOnce
  capacity:
    storage: 1Gi
  hostPath:
    path: "/mnt/mariadb"
```

Kode Sumber 4.17 Volumes.yaml MariaDB

file: /phpmyadmin /Chart.yaml

```
apiVersion: v2
name: shorten-pma
description: DPTSI Url Shortener Dev
type: application
version: 0.1.0
appVersion: "dev"
```

Kode Sumber 4.18 Chart.yaml PhpMyAdmi

file: /phpmyadmin /Values.yaml

```
env:
    host: shorten-db
```

Kode Sumber 4.19 Values.yaml PhpMyAdmin

file: /phpmyadmin /templates/Deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
    name: shorten-pma
spec:
    selector:
        matchLabels:
            app: shorten-pma
    template:
        metadata:
            labels:
                app: shorten-pma
        spec:
            containers:
            - name: c-pma
              image: phpmyadmin/phpmyadmin
              ports:
              - containerPort: 80
              env:
              - name: PMA_HOST
                value: {{.Values.env.host}}
```

Kode Sumber 4.20 Deployment.yaml PhpMyAdmin

file: /phpmyadmin /templates/Service.yaml

```
apiVersion: v1
kind: Service
metadata:
    name: shorten-pma
spec:
    type: NodePort
    ports:
    - nodePort: 30002
      port: 80
      protocol: TCP
      targetPort: 80
    selector:
        app: shorten-pma
```

Kode Sumber 4.21 Service.yaml PhpMyAdmin

Selanjutnya, keempat Helmchart yang sudah dirancang akan disimpan pada Chartmuseum, untuk melakukan *upload* Helmchart menuju Chartmuseum memerlukan *plugin* helm yang bernama helm *push*[14], instalasi dapat dilakukan dengan menjalankan perintah pada Terminal 4.9 Install Helm Push.

terminal: Pengembang

```
$ helm plugin install
         https://github.com/chartmuseum/helm-push.git
Downloading and installing helm-push v0.8.1 ...
https://github.com/chartmuseum/helm-
         push/releases/download/v0.8.1/helm-
         push_0.8.1_darwin_amd64.tar.gz
Installed plugin: push
```

Terminal 4.9 Install Helm Push

Setelah plugin helm *push* ditambahkan, jalankan perintah seperti pada Terminal 4.10 Menambahkan Repositori untuk menambahkan Chartmuseum pada daftar repositori helm.

```
terminal: Pengembang
$ helm repo add chartmuseum
        http://10.199.16.58:32357/
  "dptsi" has been added to your repositories

$ helm repo list
  NAME     URL
  chartmuseum http://10.199.16.58:32357/
```

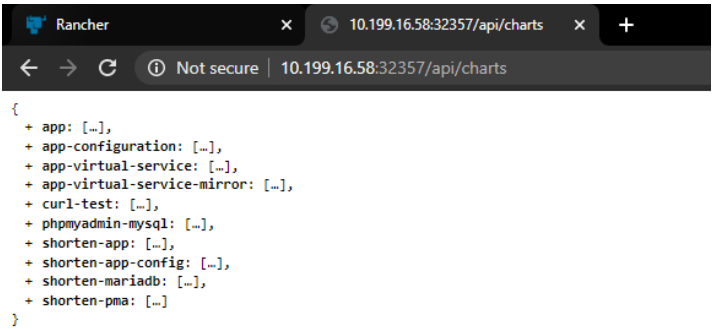
Terminal 4.10 Menambahkan Repositori

Selanjutnya jalankan perintah helm *push* seperti pada Terminal 4.11 Upload Helmchart dengan parameter direktori Helmchart dan repositori tujuan untuk melakukan *upload* Helmchart pada komputer lokal menuju Chartmuseum pada alamat http://10.199.16.58:32357/.

```
terminal: Pengembang
$ helm push urlshortener/ chartmuseum
  Pushing urlshortener.tgz to chartmuseum...
  Done.

$ helm push urlshortener-config/ chartmuseum
  Pushing urlshortener.tgz to chartmuseum...
  Done.

$ helm push mariadb/ chartmuseum
  Pushing mariadb.tgz to chartmuseum...
  Done.

$ helm push phpmyadmin/ chartmuseum
  Pushing phpmyadmin.tgz to chartmuseum...
  Done.
```

Terminal 4.11 Upload Helmchart

Setelah proses *push* selesai dapat dilihat pada http://10.199.16.58:32357/api/charts akan menampilkan seluruh Helmchart yang sudah tersimpan. Gambar 4.13 Isi Chartmuseum.



Gambar 4.13 Isi Chartmuseum

Helm chart yang tersimpan pada Chartmuseum dapat dikelola melalui *endpoint* yang dimiliki oleh Chartmuseum, berikut table *endpoint* dari Chartmuseum.

Tabel 4.2 Endpoint Helmchart

Perintah	Deskripsi
POST /api/charts	Mengupload Helmchart
DELETE /api/charts/<name>/<version>	Menghapus Helmchart
GET /api/charts	Melihat daftar Helmchart
GET /api/charts/<name>	Melihat dafter versi Helmchart
GET /api/charts/<name>/<version>	Melihat detail versi Helmchart
HEAD /api/charts/<name>	Mengecek ketersediaan
HEAD /api/charts/<name>/<version>	Mengecek ketersediaan versi
GET /	Menampilkan home chartmuseum
GET /health	Mengecek kesehatan chartmuseum

Selanjutnya Helmchart pada Chartmuseum akan di jalankan pada klaster Kubernetes, untuk menjalankan Helmchart pengembang dapat menjalankan perintah helm *install*. Dapat dilihat pada Terminal 4.12 Installasi Helmchart. Parameter dari perintah helm *install* adalah nama dari Helmchart dan repositorinya.

terminal: Pengembang

```
$ helm install shorten-pma chartmuseum/shorten-pma
  NAME: shorten-pma
  LAST DEPLOYED: Fri May 22 01:01:37 2020
  NAMESPACE: default
  STATUS: deployed
  REVISION: 1
```

Terminal 4.12 Installasi Helmchart

Untuk memperbarui suatu deployment dapat menggunakan perintah helm *upgrade*. Dapat dilihat pada Terminal 4.13 Upgrade Helmchart jumlah revision akan bertambah setiap dilakukan upgrade.

terminal: Pengembang

```
$ helm upgrade shorten-pma chartmuseum/shorten-pma
  NAME: shorten-pma
  LAST DEPLOYED: Fri May 22 01:01:37 2020
  NAMESPACE: default
  STATUS: deployed
  REVISION: 2
```

Terminal 4.13 Upgrade Helmchart

Untuk melihat Helmchart apa saja yang sedang berjalan dapat menggunakan perintah helm *list*. Terminal 4.14 Helm list.akan menampilkan daftar Helmchart yang sudah ter install serta jumlah revision dan status dari Helmchart tersebut.

terminal: Pengembang

```
$ helm list
  NAME            REVISION        STATUS
  shorten-pma     2               deployed
```

Terminal 4.14 Helm list

Untuk menghapus Helmchart yang sedang berjalan dapat menggunakan perintah helm *uninstall*. Terminal 4.15 Uninstall Helmchart.

terminal: Pengembang
`$ helm uninstall shorten-pma` `  release "shorten-pma" uninstalled`

Terminal 4.15 Uninstall Helmchart

4.4. Implementasi Integrasi Lingkungan

Pada tahap ini kedua lingkungan kerja akan di integrasikan menggunakan sebuah *pipeline*. Jenkins menjadi media integrasi antar kedua lingkugan. Integrasi ini menghasilkan transisi aplikasi dari lingkungan pengembangan ke lingkungan produksi yang lebih praktis.

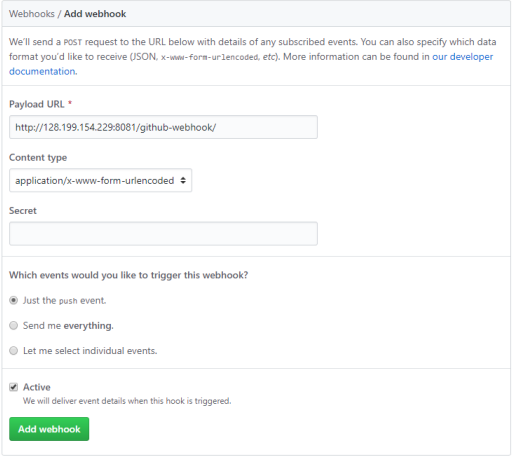
Sesuai pada tahap perancangan, Jenkins akan menghubungkan GitHub dengan Kubernetes klaster, Jenkins akan merespon perubahan kode pada GitHub, yang selanjutnya akan dilakukan build pada Jenkins, setelah *build* selesai Jenkins melakukan upload image menuju docker dockerHub, setelah itu Jenkins akan menjalankan perintah helm untuk menginisiasi atau memperbarui *deployment* pada klaster Kubernetes.

4.4.1. Implementasi Pipeline

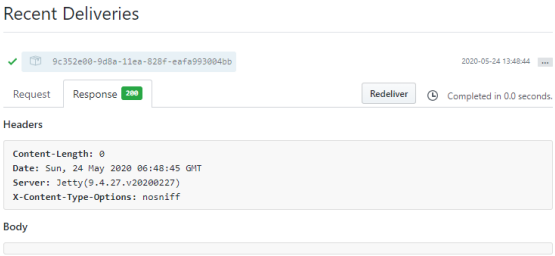
Pada tahap ini Jenkins akan diintegrasikan dengan GitHub repositori, DockerHub, dan klaster Kubernetes, untuk menghubungkan GitHub dengan Jenkins pengembang perlu menambahkan *webhook* Jenkins pada repositori GitHub.

Untuk menambahkan *webhook* pada repositori Gambar 4.14 Menambahkan Webhook ,pengembang dapat mengakses menu *webhook* dari menu *settings*. Tambahkan alamat Jenkins dengan rute tambahan pada *form Payload URL* /github-webhook, pengguna dapat memilih *events* apa saja pada yang akan melakukan *trigger* berjalannya *webhook*[15]. Setelah *webhook* ditambahkan, GitHub akan secara otomatis melakukan *request* ke

alamat Jenkins, GitHub akan menampilan kode respon200 jika *webhook* berhasil terhubung Gambar 4.15 Response Webhook.



Gambar 4.14 Menambahkan Webhook



Gambar 4.15 Response Webhook

Jenkins memerlukan *plugin* Kubernetes CLI untuk terhubung dengan klaster Kubernetes[16], Jenkins juga memerlukan beberapa *credential* untuk mengakses GitHub repositori dan dockerHub, *credentials* yang ditambahkan adalah *username password* untuk dockerHub, GitHub dan file .kubeconfig, setelah

credentials ditambahkan pengembang dapat menambahkan *credentials id* pada Jenkinsfile. Pada Kode Sumber 4.22 Jenkinsfile dapat dilihat Jenkinsfile[17] yang terbagi menjadi empat bagian, *checkout*, *build*, *deployment*, dan *cleanup*. Pada bagian *checkout* Jenkins akan mengambil kode sumber terbaru dari aplikasi, pada step *build* Jenkins akan melakukan *build* pada Dockerfile menjadi *image* kemudian *image* terbaru akan diunggah menuju DockerHub, pada tahap *deployment* Jenkins akan menajalankan perintah helm untuk menjalankan Helmchart yang sudah tersimpan pada Chartmuseum, dan terakhir pada tahap *cleanup* Jenkins akan membersihkan sisa dari proses *build image.*

file: Jenkinsfile

```
def getBuildNumber(Integer build_number,String
        applicationName){
    def searchBuildNumber = null
    try {
        sh "helm list | grep '$applicationName-
        build$build_number'"
        return build_number
    } catch (Exception e) {
        getBuildNumber(build_number-
        1,applicationName)
    }
}

def checkoutApp(String githubRepository, String
        credentialsId, String githubBranch){
    checkout([
        $class: 'GitSCM',
        branches: [[name: githubBranch]],
        doGenerateSubmoduleConfigurations:
        false,
        extensions: [],
```

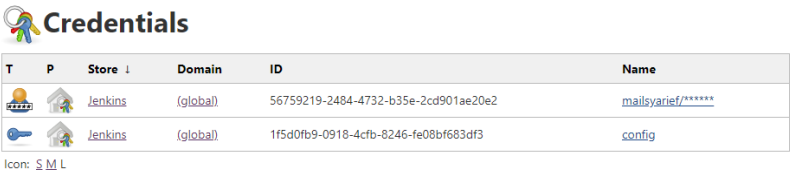
```
        submoduleCfg: [],
        userRemoteConfigs: [[
            credentialsId: credentialsId,
            url: githubRepository
        ]]
    ])
}

def buildImagePush(String dockerRegistryUrl,
        String credentialsId, String
        dockerImage){
    def appImg
    script {

        docker.withRegistry(dockerRegistryUrl,cr
        edentialsId){
            appImg =
        docker.build(dockerImage,'.')
            appImg.push()
        }
    }
}

def helmAddRepo(String chartMuseumUrl){
    sh "helm repo add chartmuseum
        $chartMuseumUrl"
    sh "helm repo update"
}

def deployApp(String credentialsId,String
        serverUrl,Integer
        latestBuildNumber,Integer
        initialDelaySeconds, String
        applicationName, String chartMuseumUrl){
    withKubeConfig(
        credentialsId: credentialsId,
        serverUrl: serverUrl){
        helmAddRepo(chartMuseumUrl)
        sh "helm install --set
```

```
        application.buildNumber=build$latestBuil
        dNumber --set
        application.initialDelaySeconds=$initial
        DelaySeconds $applicationName-
        build$latestBuildNumber chartmuseum/app"
         try{
             sh "helm install --set
        application.buildNumber=build$latestBuil
        dNumber --set
        application.initialDelaySeconds=$initial
        DelaySeconds $applicationName-
        configuration chartmuseum/app-
        configuration"
             sh "helm install $applicationName-
        phpmyadmin-mysql chartmuseum/phpmyadmin-
        mysql"
             return null
         }catch(Exception e){
             sh "helm install $applicationName-
        curl-test chartmuseum/curl-test"
             buildNumberBefore =
        getBuildNumber(latestBuildNumber-
        1,applicationName)
             return buildNumberBefore
         }
    }
}

node{
    def applicationUrl =
        'http://10.199.16.58:31380/'
    def initialDelaySeconds = '60'
    def applicationName = 'its-shorterner-url'
    def appPodLogs,buildNumberBefore
    def latestBuildNumber = "${BUILD_NUMBER}"

    def kubernetesCredentialsId = '1f5d0fb9-
        0918-4cfb-8246-fe08bf683df3'
    def kubernetesServerUrl =
```

```
        'https://k8s.its.ac.id:20443/k8s/cluster
        s/c-dwlkj'

    def chartMuseumUrl =
        'http://10.199.16.58:30661'

    def dockerImage = 'hilmiraditya/tugas-
        akhir:latest'
    def dockerRegistryUrl =
        'https://registry.hub.docker.com'
    def dockerRegistryCredentialsId = '197410bc-
        3f13-42c0-86bf-ec91dbb833f0'

    def githubRepository =
        'https://github.com/hilmiraditya/ta-
        deploy-2.git'
    def githubCredentialsId = '197410bc-3f13-
        42c0-86bf-ec91dbb833f0'
    def githubBranch = '*/deploy'

    stage('Checkout'){
        checkoutApp(
            githubRepository as String,
            githubCredentialsId as String,
            githubBranch as String
        )
    }
    stage('Build'){
        buildImagePush(
            dockerRegistryUrl as String,
            dockerRegistryCredentialsId as
        String,
            dockerImage as String
        )
    }
    stage('Deployment') {
        buildNumberBefore = deployApp(
            kubernetesCredentialsId as String,
            kubernetesServerUrl as String,
```

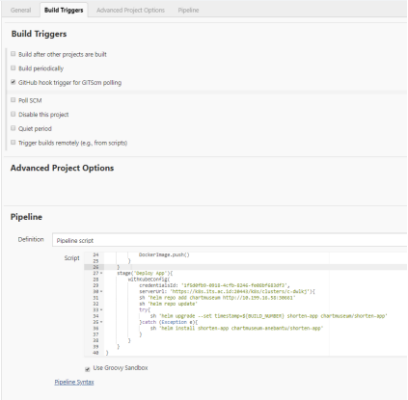
```
                latestBuildNumber as Integer,
                initialDelaySeconds as Integer,
                applicationName as String,
                chartMuseumUrl as String
            )
        }
        stage('Waiting for Deployed Application to
            be ready'){
            //konfigurasi Istio
        }
        stage('Deploy Virtual Service (Istio
            Configuration)'){
            //konfigurasi Istio
        }
        stage('cleanup') {
            dir(".") {
                sh 'docker image prune --all'
            }
        }
}
```

Kode Sumber 4.22 Jenkinsfile



Gambar 4.16 Jenkins Credentials

Selanjutnya pengembang dapat membuat *pipeline job*, dapat dilihat pada Gambar 4.17 Pipeline Job Jenkinsfile yang sudah dibuat sebelumnya diletakan pada bagian *pipeline*, pada bagian *build trigger* opsi *GitHub hook trigger for GITScm polling* dipilih supaya *job* akan berjalan Ketika terdapat perubahan pada repositori GitHub.



Gambar 4.17 Pipeline Job

Selanjutnya jalankan perintah helm pada Terminal 4.16 Instalasi Aplikasi Pendukung Shortenuntuk melakukan instalasi Helmchart berisikan MariaDB, Konfigurasi dari aplikasi Url Shortener dan PhpMyAdmin, dapat dilihat pada Gambar 4.18 Stage View, Ketika *job pipeline* dijalankan progress dari setiap step pada Jenkinsfile akan ditampilan pada aplikasi Jenkins.

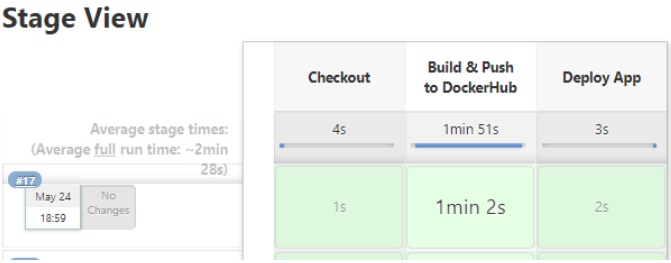
terminal: Pengembang

```
$ helm install shorten-pma dptsi/shorten-mariadb
  NAME: shorten-mariadb
  LAST DEPLOYED: Fri May 22 01:01:37 2020
  NAMESPACE: default
  STATUS: deployed
  REVISION: 1

$ helm install shorten-pma dptsi/shorten-pma
  NAME: shorten-pma
  LAST DEPLOYED: Fri May 22 01:01:37 2020
  NAMESPACE: default
  STATUS: deployed
  REVISION: 1

$ helm install shorten-app-config dptsi/ shorten-app-
```

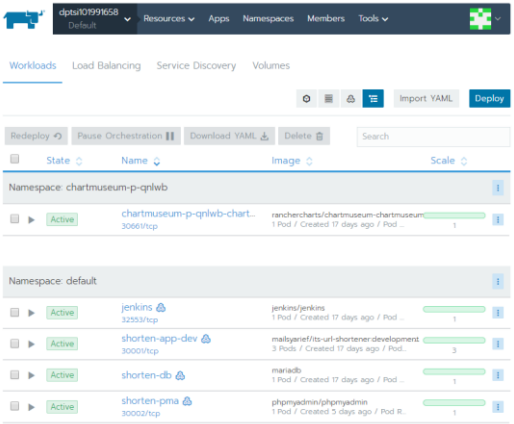
```
        config
NAME:  shorten-app-config
LAST DEPLOYED: Fri May 22 01:01:37 2020
NAMESPACE: default
STATUS: deployed
REVISION: 1
```

Terminal 4.16 Instalasi Aplikasi Pendukung Shorten

Setelah Helmchart shorten-db, shorten-app-config, shorten-pma, dijalankan dan *pipeline job* berhasil di eksekusi, akan terlihat pada *dashboard* aplikasi Rancher ketiga aplikasi sudah berjalan pada klaster Kubernetes. Sesuai dengan konfigurasi pada Kode Sumber 4.9 Deployment.yaml Urlshortener dan Kode Sumber 4.20 Deployment.yaml PhpMyAdmin aplikasi Urlshortener akan berjalan pada port 30001 dan aplikasi PhpMyAdmin berjalan pada port 30002.



Gambar 4.18 Stage View



Gambar 4.19 Dashboard Rancher

Dari seluruh proses yang telah dijelaskan diatas dapat dilakukan kembali untuk melakukan deployment aplikasi *Url Shortener* untuk versi lainnya, sebagai contoh yaitu versi *production*, hal tersebut dapat dicapai dengan cara membuat Pipeline Job, Jenkinsfile dan Helmchart baru untuk versi *production* kemudian melakan perubahan *branch* pada Jenkinsfile, merubah nama Helmchart, .env, port, dan versi *docker image* pada Helmchart sesuai dengan versi *production*.

BAB V
UJI COBA DAN EVALUASI

Bab ini membahas mengenai uji coba yang dilakukan dan evaluasi sesuai dengan rancangan dan implementasi. Dari hasil yang didapatkan setelah melakukan uji coba, akan dilakukan evaluasi sehingga dapat diambil kesimpulan untuk bab selanjutnya.

5.1. Lingkugan Uji Coba

Uji coba dilakukan dengan menggunakan total 3 komputer dan 1 klaster Kubernetes, dimana 3 komputer bertindak sebagai komputer pengembang, dan 1 klaster Kubernetes bertindak sebagai lingkungan produksi. Spesifikasi setiap komputer dapat dan klaster Kubernetes dilihat pada tabel dibawah ini.

Tabel 5.1 Spesifikasi Lingkungan Uji Coba

Komponen	**Spesifikasi**
Klaster Kubernetes	
CPU	Intel(R) Xeon Gold 5120 CPU @ 2.20GHz 8 Core
Sistem Operasi	Ubuntu 18.10 64 bit
Memori	RAM 8 GB
Penyimpanan	217 GB
Konfigurasi Jaringan	IP Address: 10.199.16.58 Netmask: 255.255.255.0 Gateway: 10.151.32.1 Hostname: dptsi
Komputer 1	
CPU	Intel(R) Core(TM) i5-9400F CPU @4.10 GHz 6 Core
Sistem Operasi	Windows 10 Pro
Memori	RAM 16 GB
Penyimpanan	900 GB
Komputer 2	
CPU	Intel(R) Core(TM) i5-7267U CPU

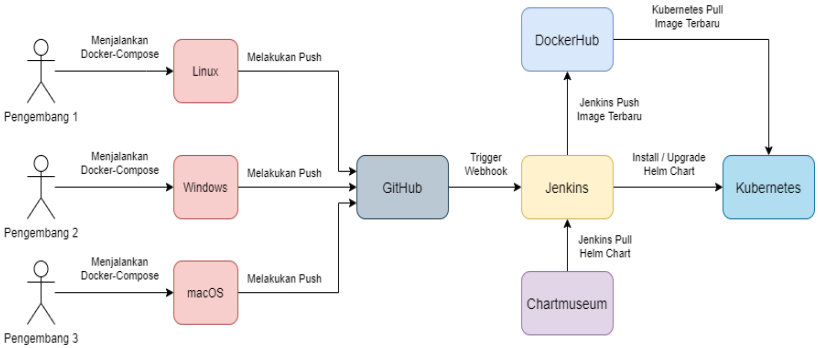
	@ 3.50GHz 2 Core
Sistem Operasi	MacOS Catalina
Memori	RAM 8 GB
Penyimpanan	256 GB
Komputer 3	
CPU	Intel(R) Core(TM) i5-9400F CPU @4.10 GHz 2 Core
Sistem Operasi	Ubuntu 20.04 64 bit
Memori	RAM 2048 GB
Penyimpanan	20 GB

5.2. Skenario Uji Coba

Uji coba ini dilakukan untuk menguji apakah fungsionalitas sistem sudah sesuai dan bekerja seperti yang seharusnya. Uji coba akan didasarkan pada beberapa skenario untuk menguji kesesuaian respon sistem. Skenario pengujian dibedakan menjadi 2 yaitu:

1. Uji Fungsionalitas
   Bertujuan untuk memastikan fitur-fitur yang ada pada sistem telah berjalan sebagaimana mestinya
2. Uji Performa
   Bertujuan untuk membuktikan apakah infrastruktur yang dibuat dapat mempercepat proses *deployment* aplikasi.

Pada uji coba ini, pengembangan akan dilakukan pada 3 komputer dengan sistem operasi yang berbeda, selanjutnya tiap pengembang akan melakukan push repositori GitHub, pengembang akan melakukan perubahan pada versi *development* dan versi *production*, kemudian akan dilihat apakah perubahan pada kedua versi tersebut berhasil berjalan pada lingkungan produksi, berikut adalah diagram komponen dari alur uji coba Gambar 5.1 Diagram Komponen Uji Coba.



Gambar 5.1 Diagram Komponen Uji Coba

5.2.1. Skenario Uji Fungsionalitas

Uji fungsionalitas dilakukan dengan cara menjalankan sistem yang telah dibuat. Lalu memastikan fitur-fitur yang ada pada sistem telah berjalan sebagaimana mestinya. dapat dilihat pada Gambar 5.1 Diagram Komponen Uji Coba, uji coba akan dilakukan mulai dari awal scaffold dijalankan hingga melakukan deployment versi terbaru aplikasi pada klister Kubernetes. Uji fungsionalitas berfungsi memenuhi kebutuhan sistem, yaitu:

1. Scaffold pengembangan dapat berjalan tanpa melakukan banyak konfigurasi.
2. *push* terbaru pada GitHub repositori akan memperbarui aplikasi pada lingkungan produksi.

5.2.1.1. Skenario Uji Fungsionalitas Scaffold

Uji ini bertujuan untuk memastikan fungsional *scaffolding* pengembang berjalan dengan baik. Uji coba ini dilakukan dengan menjalankan *script* docker-compose.yaml yang sudah dibuat sebelumnya, *script* akan dijalankan pada tiga komputer dengan spesifikasi dan sistem operasi yang berbeda.

Selanjutnya, setelah skrip dijalankan ada 3 fungsionalitas yang akan di uji, yaitu:

1. Scaffold pengembangkan dapat berjalan pada 3 sistem operasi yang berbeda.
2. Pengembang dapat melakukan perubahan kode.
3. Scaffold pengembang memiliki konfigurasi yang sama dengan produksi.

5.2.1.2. Skenario Uji Fungsionalitas Pipeline

Uji coba ini bertujuan untuk melihat apakah integrasi antara lingkungan pengembang dengan lingkungan produksi berjalan dengan baik. Pada uji coba ini akan dilakukan perubahan pada branch development, kemudian di lihat apakah perubahan kode tersebut teraplikasikan pada aplikasi yang sedang berjalan pada lingkungan produksi.

5.2.2. Skenario Uji Performa

Pada uji performa, akan dilakukan perubahan aplikasi Url Shortener pada lingkungan kerja pengembang yang selanjutnya dilakukan *push* ke GitHub repositori, perubahan pada GitHub repositori akan ter aplikasikan pada aplikasi yang sedang berjalan di lingkungan produksi. Penilaian performa sistem didasarkan pada beberapa poin, yaitu:

1. Waktu yang dibutuhkan untuk menyiapkan lingkungan kerja pada komputer pengembang.
2. Resource yang dibutuhkan untuk menjalankan scaffold pengembangan.
3. Waktu yang dibutuhkan untuk melakukan perbaruan aplikasi pada lingkungan produksi.

5.2.2.1. Skenario Uji Waktu Inisiasi Scaffold Pengembangan

Dalam pengujian performa, skrip bash mengeksekusi perintah yang dijalankan sebanyak 5 kali.

file: uji_coba.sh

```
echo "=== BEGIN ==="
START=$(date +%s)
git clone https://github.com/mailsyarief/its-url-
        shortener.git
cd its-url-shortener
git checkout development
docker-compose up -d
docker exec c-app composer install
docker exec c-app cp .env.example .env
docker exec c-app php artisan key:generate
echo "=== END ==="
END=$(date +%s)
TOTAL=$(( $END - $START ))
echo "total $TOTAL detik"
```

Kode Sumber 5.1 Skrip Bash Uji Coba

5.2.2.2. Skenario Uji Memory yang Dibutuhkan Untuk Menjalankan Scaffold Pengembangan.

Dalam pengujian performa sistem, akan dilihat berapa resource memory dan storage yang digunakan oleh docker saat lingkungan kerja berjalan menggunakan docker stats.

5.2.2.3. Skenario Uji Waktu Lingkungan Produksi Melakukan Perbaruan

Dalam pengujian performa sistem, akan dilihat berapa durasi yang dibutuhkan pipeline untuk melakukan update versi aplikasi, hal tersebut dapat dilihat melalui durasi berjalannya *job* Jenkins, pengujian akan dijalankan sebanyak 5 kali.

5.3. Hasil Uji Coba

Pada bagian ini, akan dibahas mengenai hasil uji coba sistem sesuai skenario yang telah didefinisikan pada 5.2 Skenario Uji Coba, hasil uji coba dibagi menjadi dua, hasil uji fungsionalitas dan hasil uji performa.

5.3.1. Hasil Uji Fungsionalitas

Berikut hasil uji coba fungsionalitas pada sistem

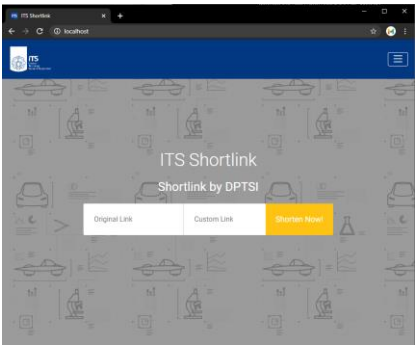
5.3.1.1. Hasil Uji Scaffold Pengembang Dapat Dijalankan

Berikut adalah hasil dari uji coba Ketika scaffold dijalankan pada 3 sistem operasi yang berbeda.

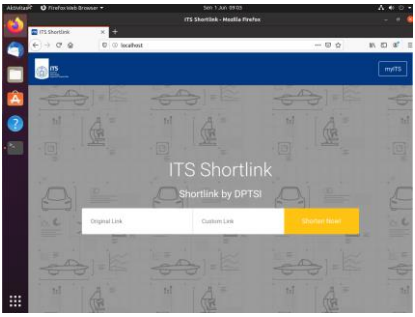
terminal: komputer

```
$ docker-compose up -d
Creating network "its-url-shortener_default" with the
         default driver
Creating c-pma ... done
Creating c-db  ... done
Creating c-app ... done

$ docker exec c-app composer install
$ docker exec c-app cp .env.example .env
$ docker exec c-app php artisan key:generate
```

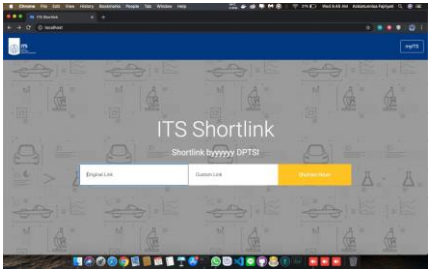
Terminal 5.1 Inisiasi Scaffold Pengembang



Gambar 5.2 Hasil Inisiasi Windows 10



Gambar 5.3 Hasil Inisiasi Ubuntu 20



Gambar 5.4 Hasil Inisiasi MacOS

Selanjutnya terdapat 3 fungsionalitas yang akan diuji, berikut adalah hasil uji dari setiap fungsionalitas.

5.3.1.2. Hasil Uji Pengembang Dapat Melakukan Perubahan Kode

Pada uji ini akan dilihat apakah pengembang dapat melakukan perubahan kode pada lingkungan pengembangan, perubahan akan dilakukan pada file home.blade.php.

file: home.blade.php

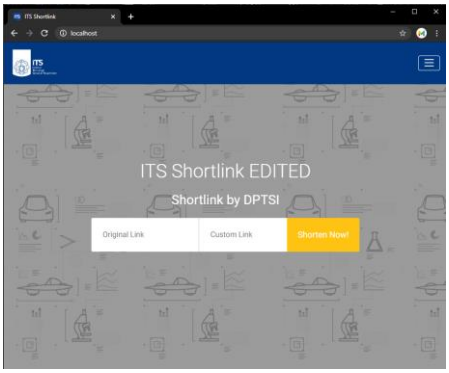
```
<div class="col-md-12">
 <div class="slider-content_wrap">
  <h1 style="color:white;">ITS Shortlink</h1>
  <h5 style="color:white;">Shortlink by DPTSI</h5>
 </div>
</div>
```

Kode Sumber 5.2 Sebelum Perubahan

file: home.blade.php

```
<div class="col-md-12">
 <div class="slider-content_wrap">
  <h1 style="color:white;">ITS Shortlink EDITED</h1>
  <h5 style="color:white;">Shortlink by DPTSI</h5>
 </div>
</div>
```

Kode Sumber 5.3 Setelah Perubahan



Gambar 5.5 Hasil Perubahan

Bisa dilihat perubahan yang dilakukan dapat langsung terlihat pada browser pengembang

5.3.1.3. Hasil Uji Lingkungan Pengembang Memiliki Konfigurasi Yang Sama Dengan Produksi.

Pada uji ini akan dilihat apakah lingkungan kerja pengembang sama dengan lingkungan produksi, hal tersebut dapat dilihat dari versi aplikasi yang berjalan pada kontainer.

```
root@its-shorterner-url-app-build93-644459858d-zbm97:/var/www/mysite# php -v
PHP 7.4.6 (cli) (built: May 15 2020 13:01:21) ( NTS )
Copyright (c) The PHP Group
Zend Engine v3.4.0, Copyright (c) Zend Technologies
    with Zend OPcache v7.4.6, Copyright (c), by Zend Technologies
root@its-shorterner-url-app-build93-644459858d-zbm97:/var/www/mysite# nginx -v
nginx version: nginx/1.14.2
root@its-shorterner-url-app-build93-644459858d-zbm97:/var/www/mysite#
```

Gambar 5.6 Cek Versi Pada Lingkungan Produksi

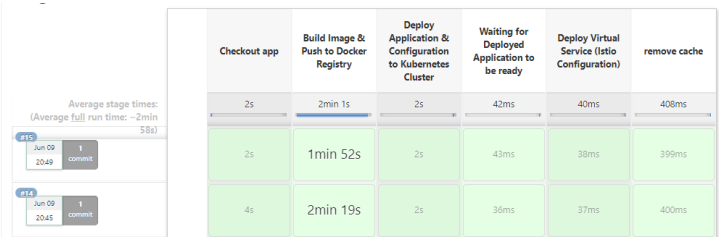
```
ismai@DESKTOP-RVUBUHK MINGW64 ~
$ docker exec -it c-app sh
# php -v
PHP 7.4.6 (cli) (built: May 15 2020 13:01:21) ( NTS )
Copyright (c) The PHP Group
Zend Engine v3.4.0, Copyright (c) Zend Technologies
    with Zend OPcache v7.4.6, Copyright (c), by Zend Technologies
# nginx -v
nginx version: nginx/1.14.2
# exit
```

Gambar 5.7 Cek Versi Pada Lingkungan Pengembang

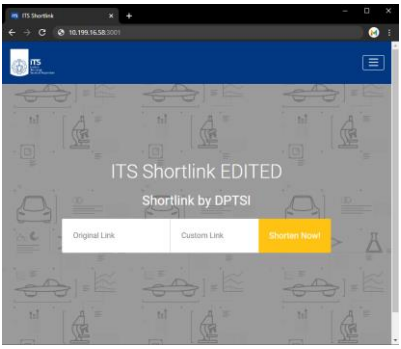
Pada gambar diatas dapat dilihat bahwa versi nginx dan php pada lingkungan kerja produksi dan pengembangan memiliki versi yang sama.

5.3.1.4. Hasil Uji Fungsionalitas Pipeline

Berikut adalah hasil uji fungsionalitas pipeline yang menunjukan push terbaru pada repositori GitHub akan mentrigger berjalannya pipeline dan melakukan pembaruan aplikasi pada lingkungan produksi.



Gambar 5.8 Proses Pipeline Pada Jenkins



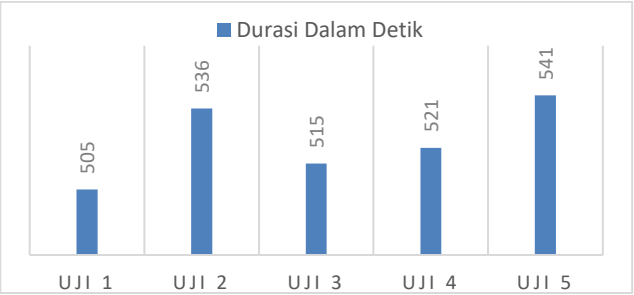
Gambar 5.9 Aplikasi Terbaru Pada Lingkungan Produksi

5.3.2. Hasil Uji Performa

Berikut hasil uji coba performa pada sistem

5.3.2.1. Hasil Uji Waktu Inisiasi Scaffold Pengembangan.

Berikut adalah hasil dari uji coba untuk melihat waktu yang dibutuhkan dalam menjalankan inisiasi scaffold pengembang.



Gambar 5.10 Waktu Inisiasi Scaffold

Dari 5 kali pengujian rata rata dalam melakukan inisiasi dibutuhkan waktu 523.6 detik atau sekitar 8.72 menit.

5.3.2.2. Hasil Uji Jumlah Memory Scaffold Pengembangan.

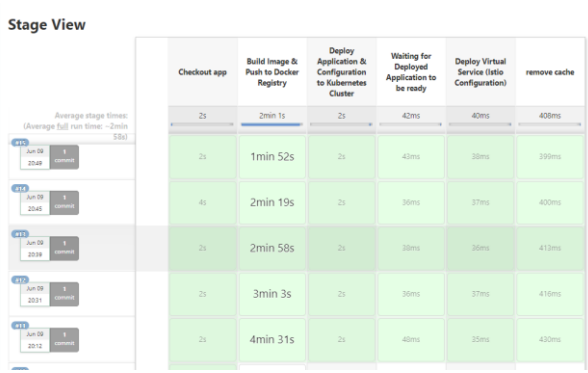
Berikut adalah hasil dari uji coba untuk melihat berapa memory yang dibutukan docker pada scaffold pengembang menggunakan docker stats.

```
CONTAINER ID   NAME    CPU %   MEM USAGE / LIMIT     MEM %
20db1c582b01   c-app   0.00%   10.64MiB / 1.944GiB   0.53%
ee5792344e62   c-db    0.03%   69.38MiB / 1.944GiB   3.48%
bf0da63daf87   c-pma   0.00%   10.11MiB / 1.944GiB   0.51%
```

Gambar 5.11 Hasil Docker Stats

5.3.2.3. Hasil Uji Waktu Melakukan Perbaruan Pada Lingkungan Produksi

Berikut adalah hasil dari uji coba untuk melihat berapa waktu yang dibutuhkan pada satu proses pipeline.



Stage View

	Checkout app	Build Image & Push to Docker Registry	Deploy Application & Configuration to Kubernetes Cluster	Waiting for Deployed Application to be ready	Deploy Virtual Service (Istio Configuration)	remove cache
Average stage times: (Average full run time: ~2min 58s)	2s	2min 1s	2s	42ms	40ms	408ms
#15 Jun 09 20:49 1 commit	2s	1min 52s	2s	43ms	38ms	399ms
#14 Jun 09 20:45 1 commit	4s	2min 19s	2s	36ms	37ms	400ms
#13 Jun 09 20:39 1 commit	2s	2min 58s	2s	38ms	36ms	413ms
#12 Jun 09 20:31 1 commit	2s	3min 3s	2s	36ms	37ms	416ms
#11 Jun 09 20:12 1 commit	2s	4min 31s	2s	48ms	35ms	430ms

Gambar 5.12 Durasi Pipeline

Dari 5 kali pengujian, rata rata berjalannya pipeline dibutuhkan waktu 181.4 detik atau sekitar 3.04 menit.

BAB VI
KESIMPULAN DAN SARAN

Bab ini berisi tentang kesimpulan yang diperoleh selama pengerjaan Tugas Akhir ini berdasarkan hasil pengujian dan hal lainnya yang telah dilakukan. Selain itu, juga terdapat beberapa saran terhadap Tugas Akhir yang penulis ajukan terhadap pengembangan Tugas Akhir ini untuk pengembangan kedepannya.

6.1. Kesimpulan

Kesimpulan yang diperoleh dari hasil uji coba dan evaluasi pada tugas akhir ini adalah sebagai berikut:

1. Telah diimplementasikan kontainer docker sebagai standar scaffold pada lingkungan pengembang menggunakan docker-compose untuk melakukan virutalisasi lingkungan pengembang yang menyerupai lingkungan produksi.
2. Telah diimplementasikan Jenkins untuk mengintegrasikan lingkungan kerja pengembang dengan lingkungan produksi yang berjalan pada klaster Kubernetes, Jenkins dan Kubernetes terhubung menggunakan plugin Kubernetes CLI pada Jenkins.
3. Integrasi dapat tercapai dengan mengimplementasikan scaffold pengembangan sehingga aplikasi pada lingkungan pengembang dapat dengan mudah dijalankan pada lingkungan produksi, dikarenakan kesamaan media yang digunakan yaitu *image*.
4. Lingkungan pengembangan yang *reusable* dan *reproducible* dapat dibuat dengan mengimplementasikan kontainer docker yang didefinisikan dalam docker-compose.

6.2. Saran

Saran yang diberikan dari hasil uji coba dan evaluasi pada tugas akhir ini adalah sebagai berikut:

1. Sistem dapat dikembangkan dengan melakukan optimasi saat melakukan build image untuk mempersingkat durasi build pada pipeline.

DAFTAR PUSTAKA

[1] "Understanding the CI/CD Pipeline: What It Is, Why It Matters" 29 07 2019 [Online]. Available: https://www.plutora.com/blog/understanding-ci-cd-pipeline [Accessed 07 06 2020].

[2] "My Digital Notes," 19 06 2015. [Online]. Available: https://andykamt.com/belajar-docker-mengenal-docker-dan-install-docker/. [Accessed 07 06 2020]

[3] "Cara Memasang Jenkins di Ubuntu 16.04," Codepolitan, 17 12 2017. [Online]. Available: https://www.codepolitan.com/cara-memasang-jenkins-di-ubuntu-1604-5a3541d206a02. [Accessed 07 06 2020].

[4] Saurabh, "What is Jenkins? | Jenkins For Continuous Integration | Edureka," edureka!, 22 05 2019. [Online]. Available: https://www.edureka.co/blog/what-is-jenkins/. [Accessed 07 06 2020].

[5] "Konsep Kubernetes", 10 04 202. [Online]. Available: https://kubernetes.io/id/docs/concepts/#ikhtisar. [Accessed 07 06 2020].

[6] “Understanding Kubernetes Objects”, 11 04 2020. [Online]. Available: https://kubernetes.io/docs/concepts/overview/working-with-objects/kubernetes-objects/. [Accessed 07 06 2020].

[7] “Helm Big Concepts”, 08 05 2020. [Online]. Available: https://helm.sh/docs/intro/using_helm/. [Accessed 07 06 2020].

[8] “Overview of Docker Compose”. [Online]. Available: https://docs.docker.com/compose/. [Accessed 12 06 2020].

[9] “Use Volumes”. [Online]. Available: https://docs.docker.com/storage/volumes/. [Accessed 12 06 2020].

[10] “Deployments”, 30 05 2020. [Online]. Available: https://kubernetes.io/id/docs/concepts/workloads/controllers/deployment/ [Accessed 07 06 2020].

[11] “Config Map”, 30 05 2020. [Online]. Available: https://kubernetes.io/docs/concepts/configuration/configmap/ [Accessed 07 06 2020].

[12] “Volumes”, 30 05 2020. [Online]. Available: https://kubernetes.io/id/docs/concepts/storage/volumes/ [Accessed 07 06 2020].

[13] “Chartmuseum Docs”, [Online]. https://chartmuseum.com/docs/ [Accessed 18 06 2020].

[14] “Helm Push Plugins”, 16 12 2019. [Online]. Available: https://github.com/chartmuseum/helm-push [Accessed 18 06 2020].

[15] “Jenkins with Github”, 23 05 2020. [Online]. Available: https://www.jenkins.io/solutions/github/ [Accessed 18 06 2020].

[16] “Using Jenkinsfile”, 23 05 2020. [Online]. Available: https://www.jenkins.io/doc/book/pipeline/jenkinsfile/ [Accessed 18 06 2020].

[17] “Kubernetes CLI”, 23 05 2020. [Online]. Available: https://plugins.jenkins.io/kubernetes-cli/ [Accessed 18 06 2020].

BIODATA PENULIS

Ismail Syarief lahir di Jakarta Selatan pada tanggal 7 April 1998. Penulis menempuh pendidikan formal di TK Auliya (2002-2004), SD Auliya (2004-2010), SMP Auliya (2010-2013), SMA Kharisma Bangsa (2013-2016), dan Informatika ITS Surabaya (2016-2020). Bidang studi yang diambil oleh penulis saat berkuliah di Departemen Informatika ITS adalah Arsitektur Jaringan dan Komputer (AJK). Penulis aktif dalam organisasi Himpunan Mahasiswa Teknik Computer-Informatika (2017-2019). Penulis juga aktif dalam kegiatan kepanitian seperti SCHEMATICS 2017 dan 2018. Penulis pernah menjalani kerja praktik di PT Sigma Caraka Kantor Pusat Bandung periode Januari 2018. Selama berkuliah, penulis juga menjadi *administrator* di Lab Rekayasa Perangkat Lunak. Penulis dapat dihubungi melalui nomor *handphone* 08176543784 atau di email ismail8syarief@gmail.com.