TUGAS AKHIR - IF184802

IMPLEMENTASI STRATEGI SHADOW DEPLOYMENT UNTUK PENGEMBANGAN PERANGKAT LUNAK INFRASTRUKTUR CLOUD MENGGUNAKAN DOCKER DAN JENKINS

HILMI RADITYA PRAKOSO
NRP 05111640000164

Dosen Pembimbing I
Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D.

Dosen Pembimbing II
Rizky Januar Akbar, S.Kom., M.Eng.

DEPARTEMEN INFORMATIKA
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya 2020

[Halaman ini sengaja dikosongkan]

TUGAS AKHIR - IF184802

IMPLEMENTASI STRATEGI SHADOW DEPLOYMENT UNTUK PENGEMBANGAN PERANGKAT LUNAK INFRASTRUKTUR CLOUD MENGGUNAKAN DOCKER, DAN JENKINS

HILMI RADITYA PRAKOSO
NRP 05111640000164

Dosen Pembimbing I
Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D.

Dosen Pembimbing II
Rizky Januar Akbar, S.Kom., M.Eng.

DEPARTEMEN INFORMATIKA
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya 2020

UNDERGRADUATE THESIS - IF184802

IMPLEMENTATION OF SHADOW DEPLOYMENT STRATEGY FOR DEVELOPMENT OF CLOUD INFRASTRUCTURE SOFTWARE USING DOCKER AND JENKINS

HILMI RADITYA PRAKOSO
NRP 05111640000164

Dosen Pembimbing I
Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D.

Dosen Pembimbing II
Rizky Januar Akbar, S.Kom., M.Eng.

INFORMATICS DEPARTMENT
Faculty of Information Communication and Technology
Institut Teknologi Sepuluh Nopember
Surabaya 2020

[Halaman ini sengaja dikosongkan]

LEMBAR PENGESAHAN

Implementasi Strategi *Shadow Deployment* Untuk Pengembangan Perangkat Lunak Infrastruktur *Cloud* Menggunakan Docker dan Jenkins

TUGAS AKHIR

Diajukan Guna Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer pada
Bidang Studi Komputasi Berbasis Jaringan
Program Studi S-1 Departemen Teknik Informatika
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember

Oleh :
HILMI RADITYA PRAKOSO
NRP : 05111640000164

Disetujui oleh Dosen Pembimbing Tugas Akhir :

ROYYANA MUSLIM IJTIHADIE, S.Kom., M.Kom., Ph.D.
NIP: 197708242006041001
.....
(pembimbing I)

RIZKY JANUAR AKBAR, S.Kom., M.Eng.
NIP: 198701032014041001
.....
(pembimbing II)

SURABAYA
JULI 2020

IMPLEMENTASI STRATEGI SHADOW DEPLOYMENT UNTUK PENGEMBANGAN PERANGKAT LUNAK INFRASTRUKTUR CLOUD MENGGUNAKAN DOCKER DAN JENKINS

Nama Mahasiswa	: HILMI RADITYA PRAKOSO
NRP	: 05111640000164
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

Deployment adalah kegiatan yang bertujuan untuk menyebarkan aplikasi yang telah dikerjakan oleh para pengembang. Penyebarannya dapat menggunakan berbagai macam strategi, salah satunya adalah strategi Shadow Deployment.

Permasalahan yang sering dialami pada saat melakukan deployment adalah resiko akan terdapat error dan downtime yang menyebabkan downtime. Dari permasalahan ini ditemukan solusi, yaitu menggunakan strategy deployment.

Tujuan digunakan strategi ini untuk menghindari resiko error down time seminimal mungkin yang disebabkan kesalahan environment pada saat melakukan deployment aplikasi baru serta membuat integrasi antara GitHub dengan lingkungan production Kubernetes, sehingga deployment dapat dilakukan secara automatically.

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

IMPLEMENTATION OF SHADOW DEPLOYMENT STRATEGY FOR DEVELOPMENT OF CLOUD INFRASTRUCTURE SOFTWARE USING DOCKER, AND JENKINS.

Nama Mahasiswa	: HILMI RADITYA PRAKOSO
NRP	: 05111640000164
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

Deployment is an activity that aims to spread applications that have been done by developers. Spread can use a variety of strategies, one of which is the Shadow Deployment strategy.

The problem that is often experienced during deployment is the risk of errors and downtime that cause downtime. From this problem found a solution, using a deployment strategy.

The purpose of this strategy is to avoid the risk of minimal down time errors caused by environmental errors when deploying new applications and to make integration between GitHub and the Kubernetes production environment, so that deployment can be carried out automatically.

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

DAFTAR ISI

Abstrak **vi**

Abstract **vii**

DAFTAR ISI **viii**

DAFTAR GAMBAR **xi**

DAFTAR TABEL **xiii**

DAFTAR KODE SUMBER **xiv**

1 BAB I PENDAHULUAN **1**

1.1. Latar Belakang 1

1.2. Rumusan Masalah 1

1.3. Batasan Masalah 2

1.4. Tujuan 2

1.5. Manfaat 2

1.6. Metodologi 3

2 BAB II TINJAUAN PUSTAKA **6**

2.1. Git 6

2.2. Docker 6

2.3. Docker Hub 8

2.4. Jenkins 8

2.5. Kubernetes 9

2.6. Helm 11

2.7. Istio 11

2.8. Rancher 12

2.9. Chartmuseum 12

3 BAB III ANALISIS DAN PERANCANGAN 13
3.1. Deskripsi Umum 13
3.2. Arsitektur Sistem 14
3.2.1. Desain Umum 15
3.2.2. Perancangan Infrastruktur 16
3.2.3. Perancangan Alur Pipeline 16
4 BAB IV IMPLEMENTASI 21
4.1. Lingkungan Pengembangan Sistem 21
4.1.1. Perangkat Keras 21
4.1.2. Perangkat Lunak 22
4.1.3 Implementasi Infrastruktur 22
4.2. Implementasi Pipeline 23
5 BAB V UJI COBA DAN EVALUASI 34
5.1. Lingkugan Uji Coba 34
5.2. Skenario Uji Coba 35
5.2.1. Skenario Uji Fungsionalitas 36
5.2.2. Skenario Uji Performa 36
5.3. Hasil Uji Coba 36
5.3.1. Hasil Uji Fungsionalitas 36
5.3.2. Hasil Uji Performa 43
6 BAB VI KESIMPULAN DAN SARAN 48
6.1. Kesimpulan 48

6.2. Saran 48

7 DAFTAR PUSTAKA 50

DAFTAR GAMBAR

Gambar 2.1 Arsitektur Docker 7
Gambar 2.2 Alur Proses *Jenkins* 9
Gambar 3.1 Infrastruktur Pada Kubernetes *Cluster* 15
Gambar 3.2 Flowchart Pipeline Pada *Jenkins* 17
Gambar 3.3 Perancangan Alur Deployment pada Jenkins 18
Gambar 4.1 Proses Mirroring Traffic pada Laravel 25
Gambar 4.2 Proses Mirroring Traffic pada Kong Api Gateway 26
Gambar 4.3 File *jenkins_build.json* 27
Gambar 5.1 Aplikasi Berjalan Pada Kubernetes *Cluster* 37
Gambar 5.2 *Virtual Service* dan *Deployment* di Kubernetes *Cluster* Pada Kiali Graf 37
Gambar 5.3 Aplikasi yang Akan di *Deploy* dan Masih Berjalan Pada Localhost Pengembang 38
Gambar 5.4 Pipeline Berjalan Pada *Jenkins* Ketika Melakukan *Update* Aplikasi 38
Gambar 5.5 *Output* Pipeline yang berjalan Pada *Jenkins* Ketika Melakukan *Update* Aplikasi 38
Gambar 5.6 Aplikasi Versi Terbaru yang Berhasil di *Deploy* Pada Kubernetes *Cluster* 39
Gambar 5.7 *Virtual Service* dan *Deployment* Versi Terbaru yang Telah Berjalan Sukses di Kubernetes *Cluster* Pada Kiali Graf 39
Gambar 5.8 Proses *Mirroring Traffic* Pada Aplikasi versi Terbaru yang Menyebabkan *Scaling Up* Pod Sebelum Pod Menerima *Traffic* 40
Gambar 5.9 *Output* Pipeline yang Berjalan Pada *Jenkins* Ketika Melakukan *Update* Aplikasi 40
Gambar 5.10 Pipeline yang Berjalan Pada *Jenkins* Ketika Melakukan *Update* Aplikasi 40
Gambar 5.11 *Virtual Service* dan *Deployment* Versi Terbaru yang Telah Berjalan Sukses di Kubernetes

Cluster Pada Kiali Graf 41
Gambar 5.12 Aplikasi Versi Terbaru yang Berhasil di *Deploy* Pada Kubernetes *Cluster* 42
Gambar 5.13 Aplikasi yang Akan di *Deploy* Dalam Kondisi *Error* dan Masih Berjalan Pada Localhost Pengembang 42
Gambar 5.14 Pipeline yang Berjalan Pada *Jenkins* Ketika Melakukan *Update* Aplikasi 42
Gambar 5.15 *Output* Pipeline yang Berjalan Pada *Jenkins* Ketika Melakukan *Update* Aplikasi 43
Gambar 5.16 Versi Aplikasi Pada Kubernetes *Cluster* Masih Menggunakan Versi Sebelumnya Dikarenakan Terdapat *Error* 43
Gambar 5.17 *Summary Report* Apache JMeter 44
Gambar 5.18 *Deployment* Kong dan Postgres Dalam Kondisi *Idle* 45
Gambar 5.19 *Deployment* Kong Dalam Proses *Mirroring Traffic* 45
Gambar 5.20 Hasil *Load Test* 46
Gambar 5.21 *Summary Report* Apache JMeter 46

DAFTAR TABEL

Tabel 4.1 Daftar Node Pada Kluster 1 21
Tabel 4.2 Daftar Node Pada Kluster 2 22
Tabel 5.1 Spesifikasi Lingkungan Uji Coba 34

DAFTAR KODE SUMBER

Kode Sumber 4.1 Jenkinsfile 33

[Halaman ini sengaja dikosongkan]

BAB I
PENDAHULUAN

1.1 Latar Belakang

Pada saat tugas akhir ini dibuat, pada platform *cloud-native* kubernetes myITS ketika terdapat perubahan versi aplikasi, proses *deployment* untuk melakukan perubahan versi aplikasi masih dilakukan secara manual, sehingga berakibat masih terdapat *downtime* cukup besar.

Deployment adalah kegiatan yang bertujuan untuk menyebarkan aplikasi yang telah dikerjakan oleh para pengembang. Penyebarannya dapat menggunakan berbagai macam strategi, salah satunya adalah strategi *Shadow Deployment*.

Dengan menggunakan metode strategi Shadow Deployment, diharapkan proses deployment dapat berjalan pada platform *cloud-native* kubernetes myITS dengan *error* dan *downtime* seminimal mungkin, Serta meningkatkan efektivitas dalam proses *deployment* dalam pada tahap production.

Untuk dapat melakukan pengaturan traffic yang merupakan salah satu tahap untuk mengimplementasikan strategi *shadow deployment* pada platform *kubernetes*, diperlukannya *Istio*. *Istio* merupakan sebuah *proxy* berjalan pada basis lingkungan *cloud-native* kubernetes yang memiliki fungsi mengatur *traffic* antar container.

1.2 Rumusan Masalah

Perumusan masalah yang terdapat pada Tugas Akhir ini adalah sebagai berikut :

1. Bagaimana mengimplementasikan shadow deployment menggunakan Istio ?

2. Bagaimana mengimplementasikan shadow deployment pada pipeline CI/CD untuk platform myITS ?
3. Apakah tidak akan terjadi downtime ketika melakukan perubahan versi aplikasi dengan mengimplementasi strategi shadow deployment pada platform myITS?

1.3 Batasan Permasalahan

Berdasarkan masalah yang diuraikan oleh penulis, maka batasan masalah pada tugas akhir ini adalah:

1. Studi kasus yang digunakan adalah aplikasi berbasis web url shortener menggunakan framework Laravel dan Kong Api Gateway yang telah berjalan pada myITS.
2. Downtime yang disebabkan pada auto scale up dan terjadi pada studi kasus Kong Api Gateway tidak tercakup dalam tugas akhir ini (non 200 request).
3. Tahapan yang terjadi pada proses integrasi antara lingkungan pengembang dan produksi pada studi kasus web url shorten tidak tercakup dalam tugas akhir ini.

1.4 Tujuan

Tujuan dari pembuatan tugas akhir ini adalah:

1. Mendapatkan downtime serendah mungkin pada saat perubahan versi aplikasi (updating) si platform cloud-native kubernetes myITS
2. Mengimplementasikan strategi shadow deployment pada platform cloud-native kubernetes myITS menggunakan Istio sebagai proxy yang dapat mengatur traffic antar container.

1.5 Manfaat

Manfaat dari pembuatan tugas akhir ini adalah untuk mengimplementasikan *Strategi Shadow Deployment* sehingga

resiko downtime pada saat perubahan versi aplikasi cukup rendah pada platform *cloud-native* kubernetes myITS.

1.6 Metodologi

Pembuatan tugas akhir ini dilakukan dengan menggunakan metodologi sebagai berikut:

1.6.1 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.

1.6.2 Studi literatur

Pada tahap ini, akan dicari studi literatur 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.

1.6.3 Analisis dan desain arsitektur

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

1.6.4 Implementasi infrastruktur

Pada tahap ini akan dilakukan implementasi infrastruktur yang telah didesain sebelumnya. Selanjutnya, akan dilakukan pengujian terhadap performa layanan untuk kemudian ditarik kesimpulan dari implementasi tersebut.

1.6.5 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.

1.6.6 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**
   - Latar Belakang
   - Rumusan Masalah
   - Batasan Tugas Akhir
   - Tujuan
   - Metodologi
   - Sistematika Penulisan

2. **Tinjauan Pustaka**

3. **Desain dan Implementasi**

4. **Pengujian dan Evaluasi**

5. **Kesimpulan dan Saran**

6. Daftar Pustaka

1.6.7 Sistematika penulisan laporan

Sistematika penulisan laporan tugas akhir adalah 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 ini 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 Git

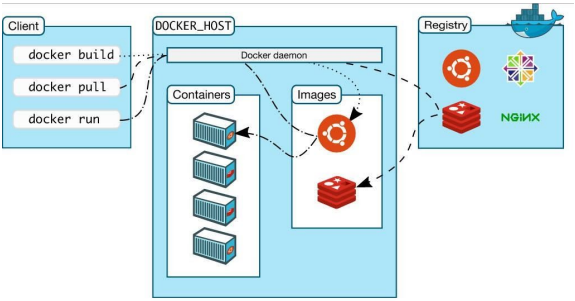
Git adalah salah satu sistem pengontrol versi *(Version Control System)* pada proyek perangkat lunak yang diciptakan oleh Linus Torvalds. Pengontrol versi bertugas mencatat setiap perubahan pada file proyek yang dikerjakan oleh banyak orang maupun sendiri. Git dikenal juga dengan distributed revision control (VCS terdistribusi), artinya penyimpanan database Git tidak hanya berada dalam satu tempat saja

2.2 *Docker*

*Docke*r awal mulanya dikembangkan oleh Solomon Hykes sebagai *project* 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 dimanapun sebagai sebuah wadah (*container*) yang ringan.

Arsitektur docker menggunakan *client* dan *server*. Docker *client* mengirim request ke docker daemon untuk untuk membangun, mendistribusikan, dan menjalankan *container* 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 container 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* container secara portabel meskipun lintas platform, menyediakan fitur pemanfaatan ulang komponen, berbagi, pengarsipan, dan *versioning* dari *image container.*

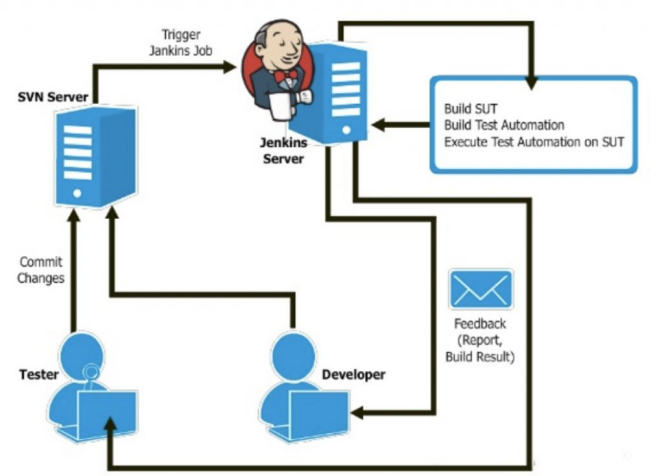
2.3 *Docker Hub*

Docker Hub adalah repositori berbasis cloud tempat pengguna membuat, menguji, menyimpan, dan mendistribusikan image. Melalui Docker Hub, pengguna dapat mengakses repository open source image, dan juga digunakan untuk membuat repository 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 container. Atau, mereka dapat memilih RDBMS yang disesuaikan dari repositori pribadi perusahaan mereka.

2.4 *Jenkins*

Jenkins adalah sebuah automasi server yang berbasis open source yang ditulis menggunakan bahasa pemrograman Java. Dengan dukungan komunitas yang cukup besar dan jumlah plugin yang cukup banyak, *Jenkins* menjadi salah satu tools yang cukup populer untuk mengimplementasikan *Continuous Integration* dan *Continous Delivery* atau biasa yang disebut *CI/CD* proses. *Jenkins* dulunya dikenal sebagai Hudson, namanya kemudian berubah setelah Oracle membeli Hudson dan memutuskan untuk mengembangkan Jenkins. Jenkins tetap berada di bawah lisensi MIT dan memiliki nilai yang tinggi akan simplicity, flexibility dan versatility.

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



Gambar 2.2. Alur Proses *Jenkins*

2.5 *Kubernetes*

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

Kubernetes menyediakan beberapa fitur, seperti *deployment*, *scaling*, *load balancing*, *logging*, dan *monitoring*. Kubernetes merupakan suatu sistem yang bersifat sebagai building block dan pluggable yang dapat digunakan untuk membangun sebuah platform yang dibutuhkan oleh developer dengan tetap mengutamakan konsep fleksibilitas. Kubernetes menyediakan manajemen *environment* yang berpusat pada

kontainer. Kubernetes melakukan orkestrasi terhadap *computing, networking*, dan infrastruktur penyimpanan.

2.5.1 *Fitur Kubernetes*

2.5.1.1 *Automatic Bin Packing*

Dapat secara otomatis menempatkan kontainer berdasarkan kebutuhan sumber daya yang dibutuhkan dan tidak mengobarkan ketersedian resource.

2.5.1.2 *Horizontal Scaling*

Melakukan scale up dan down aplikasi menggunakan perintah sederhana, dengan bantuan *User Interface*, atau dengan otomatis berdasarkan penggunaan *CPU*.

2.5.1.3 *Self Healing*

Dapat secara otomatis menghidupkan kembali kontainer yang gagal, menggantikan dan menjadwalkan ulang kontainer saat *node worker* mati. mematikan container yang tidak merespon berdasarkan *health check* yang dibuat oleh pengguna, dan tidak mengadvertise container ke klien hingga siap untuk digunakan.

2.5.1.4 *Service discovery and load balancing*

Tidak diperlukan memodifikasi aplikasi untuk menggunakan mekanisme *service discovery* yang tidak dikenal. kubernetes memberikan alamat ip pada kontainer dan memberikan single DNS untuk sekumpulan kontainer dan dapat load balance diantara container.

2.5.1.5 *Secret and configuration management*

Deploy dan update secret dan konfigurasi aplikasi tanpa rebuild image anda tanpa memberitahukan secret dalam stack konfigurasi.

2.5.1.6 *Storage orchestration*

Secara otomatis akan mounting ke sistem penyimpanan yang sudah dipilih, baik penyimpanan local, public cloud storage seperti GCP atau AWS dan network storage seperti NFS, iSCSI, Gluster, Ceph, Cinder atau Flocker.

2.6 *Helm*

Helm merupakan sebuah *package manager* yang berguna untuk mengemas *konfigurasi kubernetes* dalam sebuah folder yang biasa disebut dengan *Helm Chart*, *Helm Chart* umumnya berisikan konfigurasi aplikasi saja, namun terkadang *Helm Chart* bisa berisikan keseluruhan infrastruktur dari aplikasi tersebut, seperti *service, persistent volume*, *persistent volume claim*, dan *ingress*.

2.7 *Istio*

Istio adalah aplikasi yang membantu untuk menghubungkan, mengamankan, mengontrol dan memantau services *(microservices)*. *Istio* mengatasi permasalahan kerumitan transisi dari aplikasi monolitik ke aplikasi terdistribusi dengan arsitektur *microservices*.

Ketika aplikasi berkembang kerumitan *microservices* semakin bertambah dan menyulitkan untuk mengelola dan memahami keterkaitan antara services satu dengan services

lainnya. Dari sinilah istilah *services mesh* muncul. Istilah ini menggambarkan jaringan microservices yang membentuk suatu aplikasi dan interaksi antara services tsb.

2.7.1 *Fitur Istio*

- *Load balancing* otomatis untuk *HTTP*, *TCP*, *WebSocket* dan *gRPC*.
- *Traffic management* dengan *routing rules*, *retries*, *failovers*, *circuit breaker*, *fault injection*.
- *Metriks* dan *monitoring*.
- Mengatur keamanan komunikasi antar *services*.
- *Deployment* dengan berbagai macam cara: rollout, A/B, canary, dan lain - lain.

2.8 *Rancher*

Rancher adalah platform pengelolaan *container open source*. Ini memungkinkan kamu untuk menjalankan dan mengelola Kubernetes dengan mudah. Rancher menyediakan layanan infrastruktur seperti jaringan *multi-host*, *load balancing*, dan *snapshot volume*, Rancher dapat diakses melalui *graphical user interface* berbasis website dan *command line interface*.

2.9 *Chartmuseum*

Chartmuseum adalah repositori berbasis cloud tempat penyimpanan Helm Chart. Pengguna Helm Chart 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.

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 pipeline untuk implementasi salah satu strategi deployment, yaitu *shadow deployment.* Infrastruktur ini dibentuk memanfaatkan *Docker* sebagai platform virtualisasi, Jenkins sebagai komponen integrasi, *Helm Chart* sebagai penyimpan konfigurasi Kubernetes, *Kubernetes* sebagai orkestrasi, *GitHub*, *Docker Hub*, *Chart Museum* sebagai pengontrol versi dan repositori. serta *Istio* untuk menghubungkan, mengamankan, mengontrol dan memantau services dan *traffic*. Studi kasus yang digunakan adalah aplikasi berbasis kerangka kerja Laravel serta Kong API Gateway.

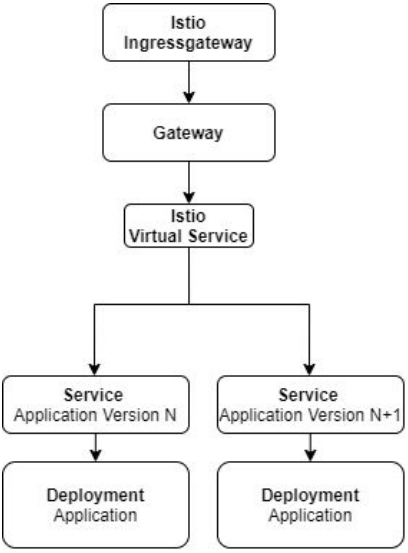
Dalam proses *deployment*, apabila Jenkins mendeteksi *deployment* pertama kali lewat helm chart, maka *chartmuseum* akan melakukan deployment tanpa melakukan *mirroring traffic*. Namun apabila Jenkins mendeteksi *deployment* aplikasi versi sebelumnya terdapat di kubernetes cluster, maka chartmuseum akan melakukan deployment dengan menggunakan *mirroring traffic*. Semua request yang masuk ke aplikasi versi baru akan dilakukan *mirroring traffic* ke aplikasi versi lama. Tujuan *mirroring traffic* pada Strategi *Deployment* ini adalah untuk melakukan pengecekan apakah aplikasi versi baru siap digunakan atau tidak.

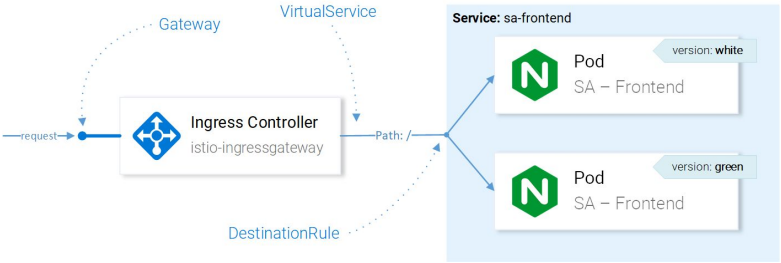
Apabila hasil dari *mirroring traffic* berhasil, maka *mirroring traffic* akan dihilangkan, semua *traffic request* akan diarahkan ke aplikasi versi baru dan aplikasi versi lama akan dihapus.

Apabila hasil dari *mirroring traffic* gagal, maka *mirroring traffic* akan dihapus, semua *traffic request* akan diarahkan ke aplikasi versi lama dan aplikasi versi baru akan dihapus.

3.2 Arsitektur Sistem

Berikut ini adalah gambar arsitektur yang ada telah dibuat :





Gambar 3.1 Infrastruktur Pada Kubernetes *Cluster* Ketika Kubernetes diintegrasikan dengan Istio

3.2.1 Desain Umum

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

1. Konfigurasi repository *GitHub* untuk menyimpan *source code* yang siap untuk dilakukan *deployment*
2. Konfigurasi *Docker* untuk membangun *image* yang akan digunakan
3. Konfigurasi *Kubernetes* untuk melakukan deployment aplikasi
4. Konfigurasi *Istio* untuk melakukan pengalihan traffic pada *Kubernetes* cluster
5. Konfigurasi *Jenkins* untuk menjalankan semua *trigger* yang dibuat untuk proses deployment dan pengalihan traffic.

Untuk memenuhi kebutuhan sistem tersebut, penulis membagi sistem menjadi beberapa komponen. Komponen yang akan dibuat antara lain :

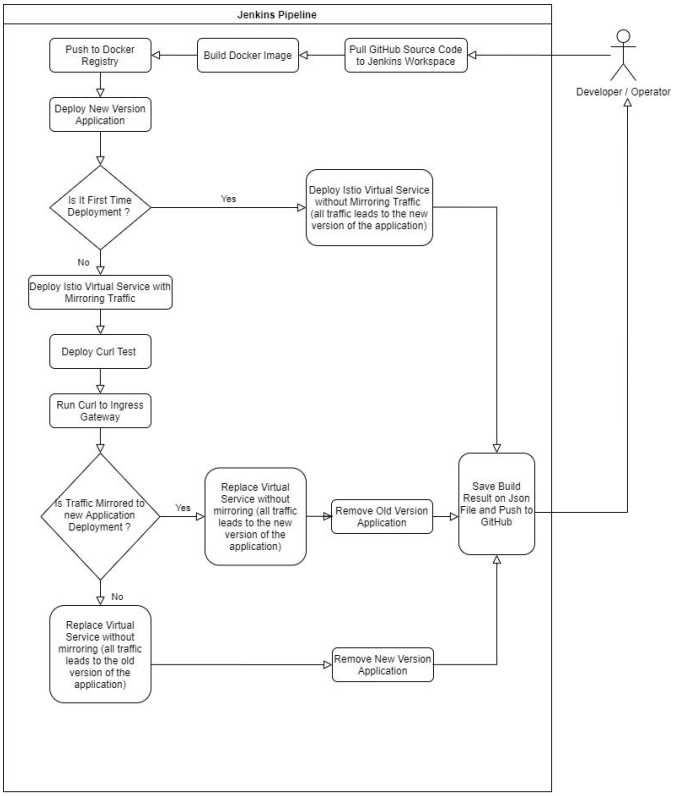
1. Komponen *infrastruktur* yang akan dibuat menggunakan *docker* untuk mengembangan aplikasi (Dockerfile).
2. Komponen konfigurasi *pipeline* yang akan digunakan *Jenkins* yang akan menggunakan *GitHub*, *DockerHub*, serta *Chartmuseum* sebagai tempat penyimpanan berbagai kebutuhan konfigurasi.

3.2.2 Perancangan Infrastruktur

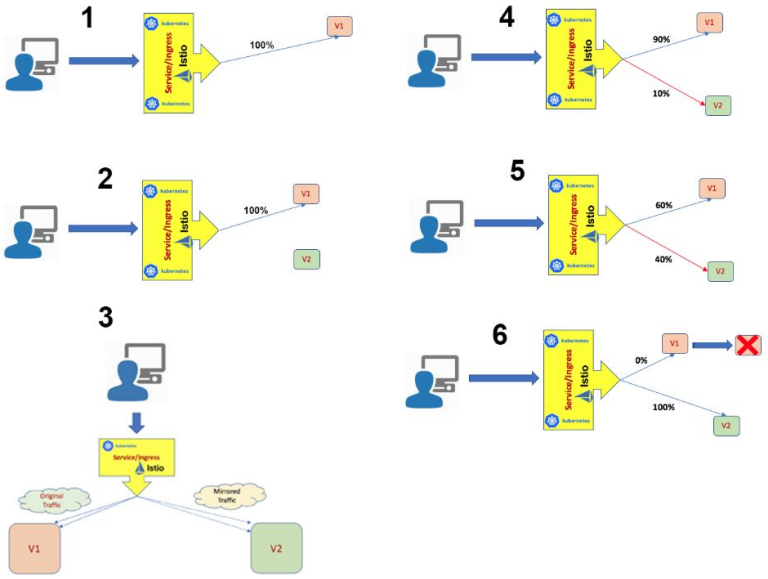
Pada *istio service mesh*, setiap request dari luar yang masuk ke dalam *kubernetes cluster* akan melalui *Ingress*, yang berguna untuk membuka akses ke dalam *internal network (proxy)* yang di expose dalam bentuk *virtual service*. Maka diperlukannya *Gateway* sebagai gerbang untuk menghubungkan antara *ingress controller* dengan *service* yang telah di *expose* sebelumnya di *virtual service*. Gateway di expose dengan bentuk *port 80 (http)* atau *443 (https)*.

Untuk dapat melakukan pengalihan traffic, maka dibutuhkan sebuah *virtual service*. *Virtual service* merupakan salah satu komponen dari *istio* yang berfungsi dapat melakukan konfigurasi *uri*, *traffic weight & mirroring* pada sebuah *deployment kubernetes*, serta *virtual service* dapat menghubungkan *service* ke *istio ingress-gateway.* Konfigurasi *Istio* dikemas dalam bentuk *helm chart* dan disimpan pada repositori *chart museum*.

3.2.3 Perancangan Alur Pipeline



Gambar 3.2 Flowchart Pipeline pada Jenkins



Gambar 3.3 Perancangan Alur Deployment yang akan berjalan pada Jenkins

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

Pada tahap perancangan pipeline ini, Jenkins merupakan komponen utama. Jenkins akan berjalan di dalam *kubernetes cluster* yang nantinya akan terhubung dengan GitHub, DockerHub, Chartmuseum. Jenkins akan menjalankan sebuah perintah yang ditulis dalam file *Jenkinsfile*, yang nantinya akan mengeksekusi satu persatu secara perintah perintah yang telah

ditulis dalam file tersebut. alur *pipeline* pada *Jenkinsfile* akan dibagi menjadi 4 tahap yang akan dijelaskan pada berikut ini :

3.2.3.1 *Pull Source Code*

Tahap ini merupakan proses *pulling source code* dari *repository GitHub* serta *branch* yang telah ditentukan, lalu *source code* disimpan pada *workspace* Jenkins.

3.2.3.2 *Build Image & Push to Docker Registry*

Di tahap ini, Jenkins akan melakukan *build* menggunakan Docker, dan menggunakan konfigurasi *Dockerfile* yang terdapat dari *source code* yang telah di clone pada tahap sebelumnya. Pada saat proses *build image* berhasil maka tahap selanjutnya adalah *push* dari hasil image yang berhasil di build ke *DockerHub*,

3.2.3.3 *Deploy Application & Configuration to Kubernetes Cluster*

Di tahap ini Jenkins akan melakukan trigger untuk melakukan deployment Aplikasi yang telah di *build* menjadi sebuah *image* pada tahap sebelumnya, Aplikasi akan di deploy dengan konfigurasi yang sudah disimpan pada repository *Chartmuseum*.

3.2.3.4 *Waiting for Deployed Application to be Ready*

Di tahap ini jenkins melakukan delay selama beberapa saat untuk menunggu *Pod* aplikasi agar siap untuk digunakan.

3.2.3.5 *Deploy Istio Configuration*

Di tahap ini jenkins akan melakukan deployment dengan konfigurasi yang sudah disimpan pada repository *Chartmuseum*. Konfigurasi yang di *deploy* pada tahap ini berisi Konfigurasi *Virtual Service*, Pada awal tahap ini akan dilakukan pengecekan terhadap *Deployment* Aplikasi yang telah terinstall.

Hasil dari pengecekan *deployment* hanya ada satu deployment, maka Jenkins akan melakukan deployment konfigurasi *virtual service* tanpa menggunakan fitur *mirroring traffic* dan Jenkins akan langsung menuju tahap berikutnya tanpa melakukan *traffic testing*.

Apabila hasil dari pengecekan Deployment terdapat lebih dari satu deployment, maka Jenkins akan melakukan deployment konfigurasi *virtual service* menggunakan fitur *mirroring traffic*, sehingga dapat melakukan *traffic testing*

3.2.3.6 *Remove Cache*

Jenkins akan membersihkan hasil *build image* yang telah tidak digunakan.

3.2.3.6 *Show Result*

Jenkins akan menampilkan *build* status dari deployment yang sedang dijalankan sekarang.

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 2 buah klaster Kubernetes. Dengan spesifikasi yang berbeda.

4.1.1. Perangkat Keras

Perangkat keras yang digunakan dalam pengembangan sistem adalah sebagai berikut:

1. Kubernetes cluster dengan 2 Node dengan 32 Core, 24GB RAM, 60GB penyimpanan.

2. Kubernetes cluster dengan 1 Node dengan 8 Core, 16GB RAM, 60GB penyimpanan.

Berikut adalah alamat dan peran dari pada klaster Kubernetes yang dapat dilihat pada Tabel 4.1 dan Table 4.2 Daftar Node:

Tabel 4.1 Daftar Node Pada Kluster 1

Nama Node	Peran	Alamat

Master	Master	k8s.its.ac.id
Node	Node	10.199.2.175
Node	Node	10.199.2.176

Tabel 4.2 Daftar Node Pada Kluster 2

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. Jenkins 2.236 untuk *pipeline*
2. Rancher 3 untuk mengelola klaster Kubernetes
3. Helm 3 untuk menyimpan konfigurasi Kubernetes
4. Istio versi 1.4.7 sebagai *service mesh*
5. GitHub sebagai repository penyimpanan *kode sumber*
6. DockerHub untuk menyimpan *image* docker

4.1.3 Implementasi Infrastruktur

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

Suatu deployment pada Kubernetes dapat disimpan dalam satu atau lebih file .yaml, ketika file tersebut dijalankan, secara otomatis konfigurasi akan berjalan pada cluster Kubernetes, namun hal tersebut memiliki limitasi dalam fleksibilitas dan kemudahan dalam pendistribusian, dengan itu pada tugas akhir ini

helm digunakan untuk menyimpan konfigurasi *traffic management Istio* dikarenakan kemampuannya untuk merubah .yaml yang statis menjadi dinamis dan kemudahan dalam pendistribusian karena dapat disimpan pada sebuah repository.

Helm Charts berguna untuk membantu untuk mendefinisikan, instalasi maupun pembaruan aplikasi yang paling kompleks sekalipun pada kubernetes cluster. Pengguna Helm Chart dapat membuat repositori pribadi menggunakan *chartmuseum*. *Chartmuseum* merupakan suatu *tools* yang digunakan untuk memudahkan pengembang untuk melakukan distribusi dan mengelola satu atau lebih *Helm Chart*.

4.2 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.

Jenkins memerlukan plugin Kubernetes CLI untuk terhubung dengan klaster Kubernetes, 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 x.x dapat dilihat Jenkinsfile yang terbagi menjadi beberapa bagian.

Pada bagian checkout Jenkins akan mengambil kode sumber terbaru dari aplikasi, pada step *build* Jenkins akan membuild Dockerfile menjadi image kemudian image terbaru akan diupload menuju DockerHub

Jenkins melakukan deployment Aplikasi yang sebelumnya telah di upload DockerHub, Aplikasi akan di deploy dengan konfigurasi yang sudah disimpan pada repository *Chartmuseum*. Setelah itu, Jenkins melakukan delay selama 1 menit untuk menunggu *Pod* aplikasi agar siap untuk digunakan.

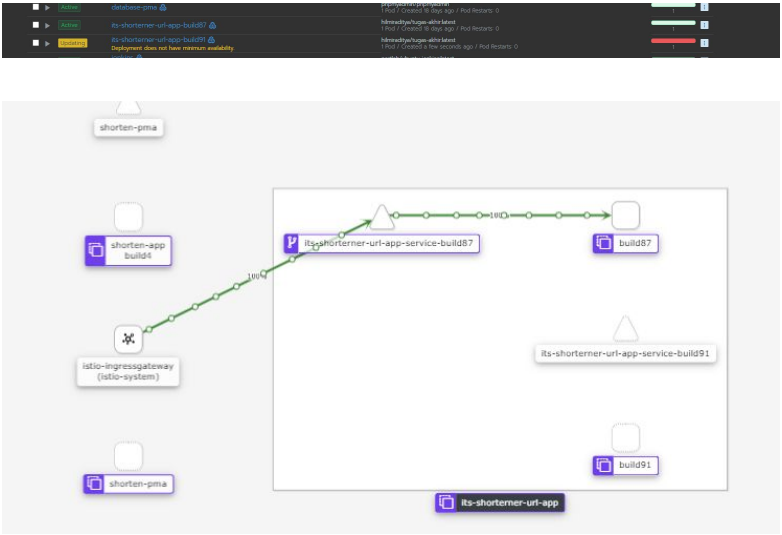
Di tahap selanjutnya, jenkins akan melakukan deployment dengan konfigurasi yang sudah disimpan pada repository *Chartmuseum*. Konfigurasi yang di *deploy* pada tahap ini berisi Konfigurasi *Virtual Service*, Pada awal tahap ini akan dilakukan pengecekan terhadap *Deployment* Aplikasi yang telah terinstall.

Hasil dari pengecekan *deployment* hanya ada satu deployment, maka Jenkins akan melakukan deployment konfigurasi *virtual service* tanpa menggunakan fitur *mirroring traffic* dan Jenkins akan langsung menuju tahap berikutnya tanpa melakukan *traffic testing*.

Apabila hasil dari pengecekan Deployment terdapat lebih dari satu deployment, maka Jenkins akan melakukan pengecekan *build status* pada file *jenkins_build.json* untuk mendapatkan nama build terbaru pada deployment di cluster lalu melakukan deployment konfigurasi *virtual service* menggunakan fitur *mirroring traffic*, sehingga dapat melakukan *traffic testing*

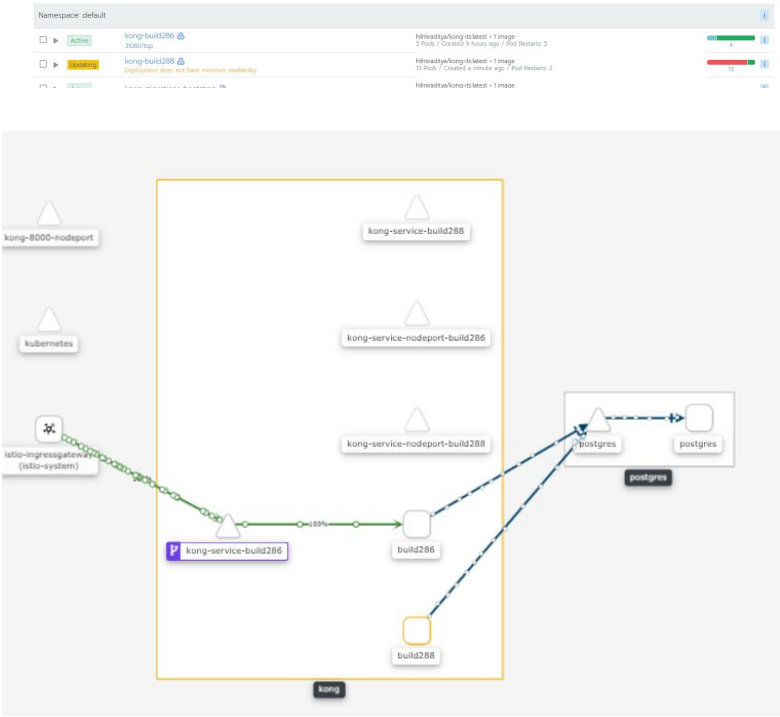
Untuk menjalankan *traffic testing* pada uji kasus *Framework Laravel*, Jenkins akan melakukan *Deployment* yang berisi image *Curl*. Pod yang berisi image *Curl* ini akan mengakses *url ingress gateway* dari *istio*. *Traffic testing* dapat dikatakan berhasil apabila *logs* dari *pod* yang berisi aplikasi versi terbaru terdapat *response status* 200. Apabila *logs* dari *pod* yang berisi aplikasi versi terbaru terdapat *response status* 200, semua

traffic request akan diarahkan ke aplikasi versi baru dan aplikasi versi lama akan dihapus.



Gambar 4.1 Proses Mirroring Traffic pada Studi Kasus Laravel

Sedangkan *traffic testing* pada uji kasus *Kong Api Gateway*, Jenkins akan melakukan *Deployment* yang berisi image *Curl*. Pod yang berisi image *Curl* ini akan mengakses *url ingress gateway* dari *istio*. *Traffic testing* pada uji kasus *Kong Api Gateway* bertujuan untuk melakukan *triggering HPA* (*Horizontal Pod Autoscaler),* agar pada saat pengalihan *traffic* ke *Deployment* versi terbaru menghindari downtime berlebih dikarenakan Pod versi terbaru belum siap untuk melakukan penerimaan traffic dalam jumlah besar.



Gambar 4.2 Proses Mirroring Traffic pada Studi Kasus Kong Api Gateway yang bertujuan untuk *triggering* HPA pada *pod* versi terbaru

Di tahap terakhir, Jenkins akan membersihkan hasil *build image* yang telah tidak digunakan serta melakukan *push* ke GitHub untuk melakukan update *build status* pada file *jenkins_build.json* dan di akhir akan ada output yang menampilkan *build* status dari deployment yang sedang dijalankan sekarang.

Gambar 4.3 File *jenkins_build.json* yang berisi *build version* terbaru dan nama *deployment* pada kubernetes.

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,credentialsId
){
            appImg = docker.build(dockerImage,'.')
            appImg.push()
        }
    }
}

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

def installKubectl(){
    sh "curl -LO
https://storage.googleapis.com/kubernetes-release/r
elease/`curl -s
https://storage.googleapis.com/kubernetes-release/r
elease/stable.txt`/bin/linux/amd64/kubectl"
    sh "chmod +x ./kubectl"
}

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$latestBuildNumber
--set
application.initialDelaySeconds=$initialDelaySecond
s $applicationName-build$latestBuildNumber
chartmuseum/app"
        try{
            sh "helm install --set
application.buildNumber=build$latestBuildNumber
--set
```

```
application.initialDelaySeconds=$initialDelaySecond
s $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
        }
    }
}

def uninstallCurlTest(String applicationName,
String credentialsId, String serverUrl){
    withKubeConfig(
        credentialsId: credentialsId,
        serverUrl: serverUrl){
            sh "helm uninstall
$applicationName-curl-test"
    }
}

def trafficTest(String applicationName, String
latestBuildNumber, String buildNumberBefore, String
initialDelaySeconds){
    sh '''
        ./kubectl exec -it $(./kubectl get pod -l
app=curl-test -o jsonpath={.items..metadata.name})
-c sleep -- sh -c 'curl http://10.199.16.58:31380/'
    '''
    appPodLogs=null
    try{
        getPod=sh(returnStdout:true,
script:"./kubectl get pod -l
app=$applicationName-app,version=build$latestBuildN
umber -o jsonpath={.items..metadata.name}").trim()
        timeout(time: 5, unit: 'SECONDS') {
```

```
            appPodLogs=sh(returnStdout:true,
script:"./kubectl logs -f $getPod -c c-shorten |
grep 200").trim()
        }
    }catch(Exception e){
        currentBuild.result = "SUCCESS"
    }
    if(appPodLogs){
        echo 'Aplikasi berpindah ke versi baru'
        sh "helm upgrade --set
application.buildNumber=build$latestBuildNumber
--set
application.initialDelaySeconds=$initialDelaySecond
s $applicationName-virtual-service
chartmuseum/app-virtual-service"
        sh "helm uninstall
$applicationName-build$buildNumberBefore"
        return "The latest version of the
application has been successfully deployed"
    }else{
        echo 'Aplikasi berpindah ke versi lama
dikarenakan belum siap'
        sh "helm upgrade --set
application.buildNumber=build$buildNumberBefore
--set
application.initialDelaySeconds=$initialDelaySecond
s $applicationName-virtual-service
chartmuseum/app-virtual-service"
        sh "helm uninstall
$applicationName-build$latestBuildNumber"
        return "The latest version of the
application failed to deploy, rollback to the old
version"
    }
}

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/clusters/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'
    def trafficTestResult

    stage('Pull Source Code'){
        checkoutApp(
            githubRepository as String,
            githubCredentialsId as String,
            githubBranch as String
        )
    }
    stage('Build Image & Push to Docker Registry'){
        buildImagePush(
            dockerRegistryUrl as String,
            dockerRegistryCredentialsId as String,
            dockerImage as String
        )
    }
    stage('Deploy Application & Configuration to
Kubernetes Cluster') {
        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'){
        sh "sleep $initialDelaySeconds"
    }
    stage('Deploy Istio Configuration'){
        withKubeConfig(
            credentialsId:
'1f5d0fb9-0918-4cfb-8246-fe08bf683df3',
            serverUrl:
'https://k8s.its.ac.id:20443/k8s/clusters/c-dwlkj')
{
            helmAddRepo(chartMuseumUrl)
            installKubectl()
            try{
                sh "helm upgrade --set
application.buildNumber=build$latestBuildNumber
--set
application.buildNumberBefore=build$buildNumberBefo
re --set
application.initialDelaySeconds=$initialDelaySecond
s $applicationName-virtual-service
chartmuseum/app-virtual-service-mirror"

                trafficTest = trafficTest(
                    applicationName as String,
                    latestBuildNumber as String,
                    buildNumberBefore as String,
                    initialDelaySeconds as String
                )

                uninstallCurlTest(
                    applicationName as String,
                    kubernetesCredentialsId as
String,
                    kubernetesServerUrl as String
                )
            }catch(Exception e){
```

```
                sh "helm install --set
application.buildNumber=build$latestBuildNumber
--set
application.initialDelaySeconds=$initialDelaySecond
s --set
application.initialDelaySeconds=$initialDelaySecond
s $applicationName-virtual-service
chartmuseum/app-virtual-service"
                 trafficTest = "The latest version
of the application has been successfully deployed"
            }
        }
    }
    stage('Remove Cache') {
        dir(".") {
            sh 'docker image prune --all'
            sh ''
        }
    }
    stage('Show Result') {
         echo trafficTest
    }
}
```

Kode Sumber 4.1 Jenkinsfile

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 2 klaster Kubernetes, dimana 1 klaster Kubernetes dengan framework Laravel dan 1 klaster Kubernetes dengan Kong Api Gateway . Spesifikasi setiap komputer dapat dan klaster Kubernetes dilihat pada tabel dibawah ini.

Tabel 5.1 Spesifikasi Lingkungan Uji Coba

Komponen	**Spesifikasi**
Klaster Kubernetes 1 (Framework Laravel)	
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

Klaster Kubernetes 2 (Kong API Gateway)	
CPU	Intel(R) Xeon Gold 5120 CPU @ 2.20GHz 32 Core

Sistem Operasi	Ubuntu 18.10 64 bit
Memori	RAM 32 GB
Penyimpanan	217 GB
Konfigurasi Jaringan	IP Address: 10.199.2.175, 10.199.2.176 Netmask: 255.255.255.0 Gateway: 10.151.32.1 Hostname: dptsi

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 Coba Fungsionalitas Pipeline

   Bertujuan untuk memastikan fitur-fitur yang ada pada pipeline telah berjalan secara sempurna dengan Mengakses *ingress gateway* pada url *http://10.199.2.175:31380/* untuk studi kasus *Kong api gateway* dan *http://10.199.16.58:31380/* untuk studi kasus *Laravel* selama proses perubahan versi aplikasi menggunakan berbagai tingkat *concurrency*.

2. Uji Coba Performa

   Bertujuan untuk membuktikan apakah infrastruktur yang dibuat dapat mencapai proses *deployment* aplikasi dengan *down time* seminimal mungkin dengan Melihat hasil total *requests*, dan persentase *non-200 reques*t pada saat pengujian.

Pada uji coba ini *developer* melakukan *push* pada repository GitHub untuk melakukan update pada aplikasi telah ter *deploy* sebelumnya, kemudian akan dilihat apakah update pada aplikasi tersebut berhasil berjalan pada lingkungan produksi

5.2.1 Skenario Uji Coba Fungsionalitas Pipeline

Uji fungsionalitas Pipeline dilakukan dengan cara menjalankan sistem yang telah dibuat. Lalu memastikan fitur-fitur yang ada pada sistem telah berjalan sebagaimana mestinya. Uji fungsionalitas berfungsi memenuhi kebutuhan sistem, yaitu:

1. Tidak terdapat *error* pada aplikasi yang akan di deploy, sehingga proses deployment berjalan secara normal.
2. Terdapat *error* pada aplikasi yang akan di deploy, sehingga terjadi pembatalan pada *deployment* dan *pipeline pada jenkins* melakukan *rollback* ke aplikasi versi sebelumnya.

5.2.2 Skenario Uji Coba Performa

Pada uji performa, akan dilakukan update aplikasi dengan melakukan push ke GitHub repository. Penilaian performa sistem didasarkan pada beberapa poin, yaitu:

1. Persentase *traffic request* yang sukses pada saat proses update berjalan dengan kasus tidak terdapat error pada aplikasi yang akan di deploy
2. Persentase *traffic request* yang sukses pada saat proses update berjalan dengan kasus terdapat error pada aplikasi yang akan di deploy

5.3 Hasil Uji Coba

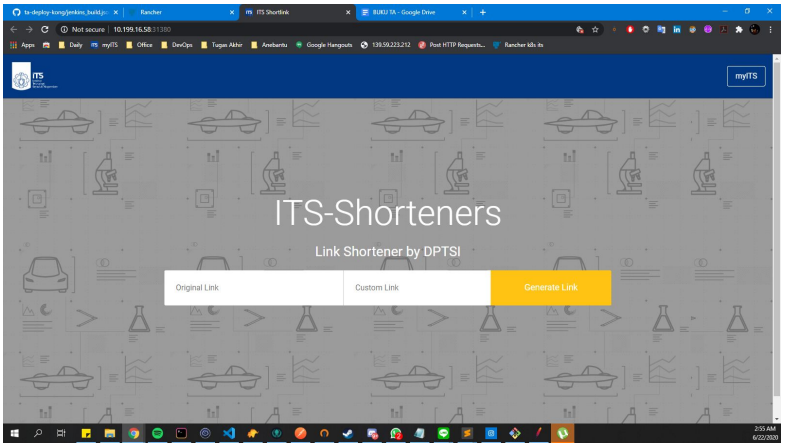
Pada bagian ini, akan dibahas mengenai hasil uji coba sistem sesuai skenario yang telah didefinisikan pada 5.2 Skenario Uji Coba Hasil uji coba dibagi menjadi dua, hasil uji fungsionalitas dan hasil uji performa.

5.3.1 Hasil Uji Coba Fungsionalitas Pipeline

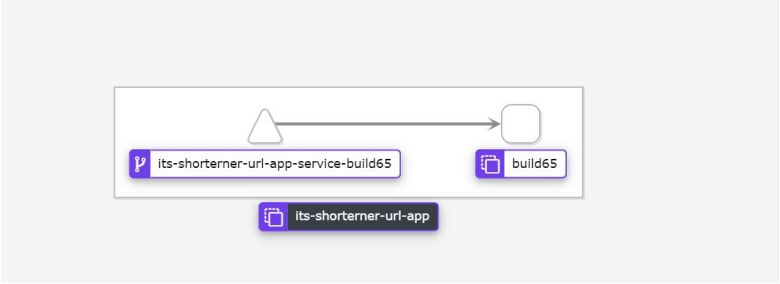
Berikut hasil uji coba fungsionalitas pada sistem akan dijelaskan pada bagian berikut ini.

5.3.1.1 Hasil Uji Coba Tidak Terdapat Error Pada Aplikasi Yang Akan Di Deploy

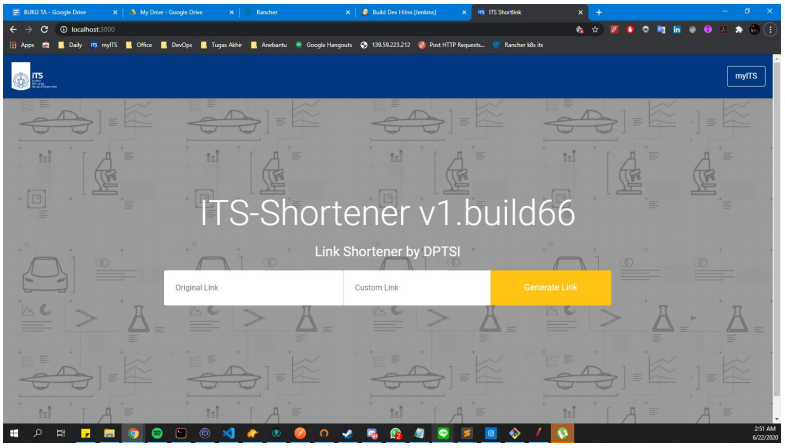
1. **Studi Kasus Framework Laravel**



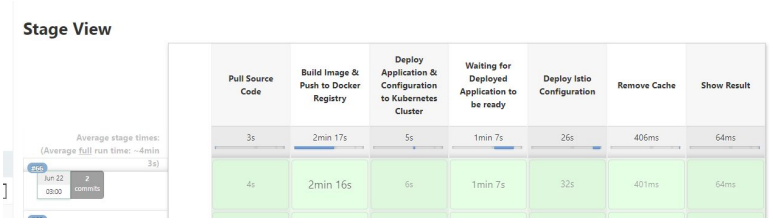
Gambar 5.1 Aplikasi Berjalan Pada Kubernetes *Cluster*



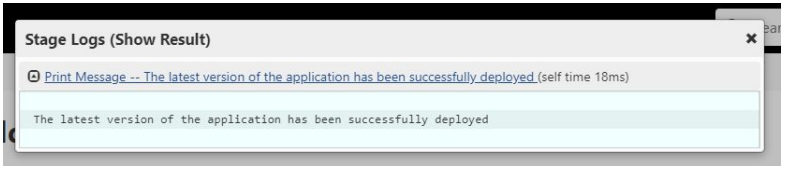
Gambar 5.2 *Virtual Service* dan *Deployment* di Kubernetes *Cluster* Pada Kiali Graf



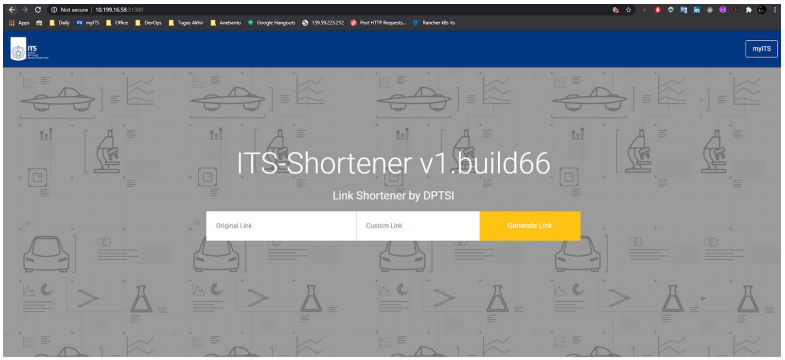
Gambar 5.3 Aplikasi yang Akan di *Deploy* dan Masih Berjalan Pada *Localhost* Pengembang



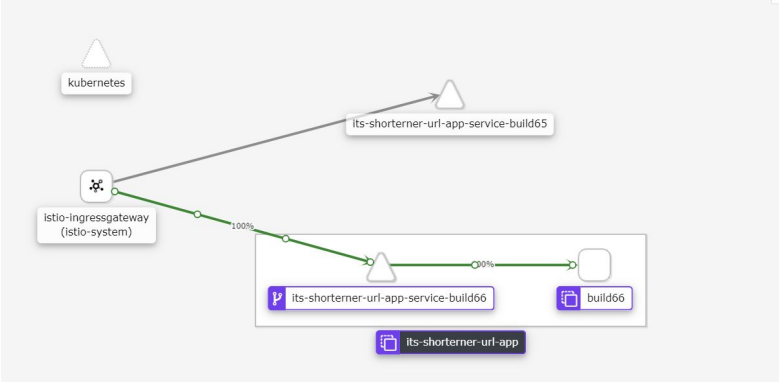
Gambar 5.4 Pipeline Berjalan Pada *Jenkins* Ketika Melakukan *Update* Aplikasi



Gambar 5.5 *Output* Pipeline yang Berjalan Pada *Jenkins* Ketika Melakukan *Update* Aplikasi

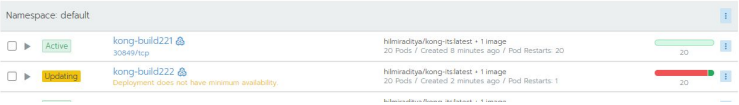


Gambar 5.6 Aplikasi Versi Terbaru yang Berhasil di *Deploy* Pada Kubernetes *Cluster*

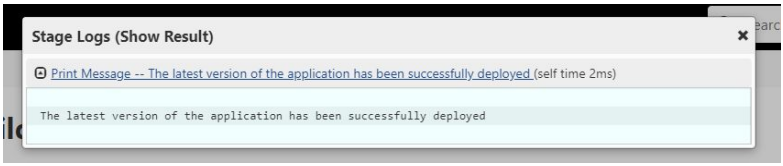


Gambar 5.7 *Virtual Service* dan *Deployment* Versi Terbaru yang Telah Berjalan Sukses di Kubernetes *Cluster* Pada Kiali Graf

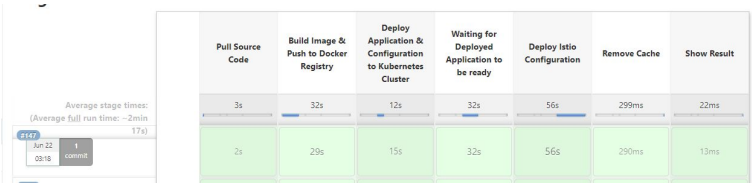
2. Studi Kasus Kong Api Gateway



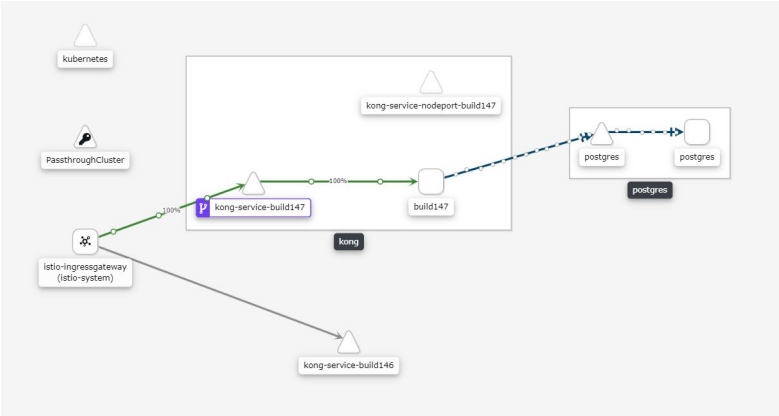
Gambar 5.8 Proses *Mirroring Traffic* Pada Aplikasi Versi Terbaru yang Menyebabkan *Scaling Up Pod* Sebelum Pod Menerima *Traffic*



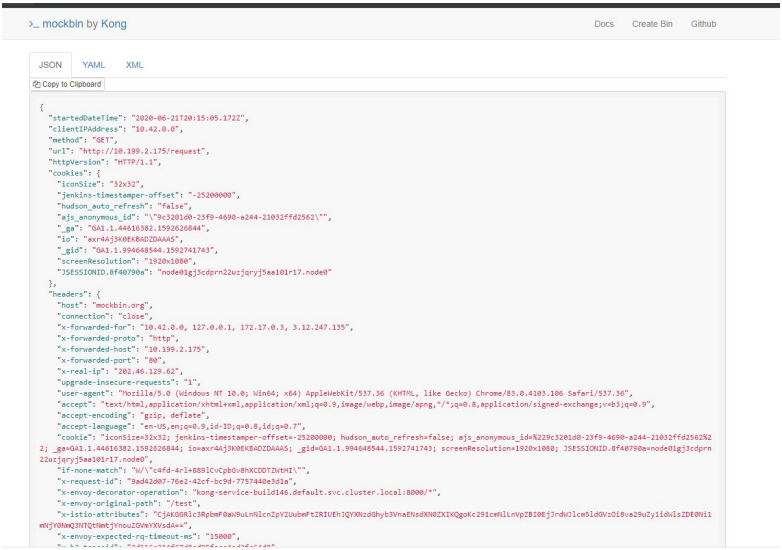
Gambar 5.9 *Output* Pipeline yang Berjalan Pada *Jenkins* Ketika Melakukan *Update* Aplikasi



Gambar 5.10 Pipeline yang Berjalan Pada *Jenkins* Ketika Melakukan *Update* Aplikasi



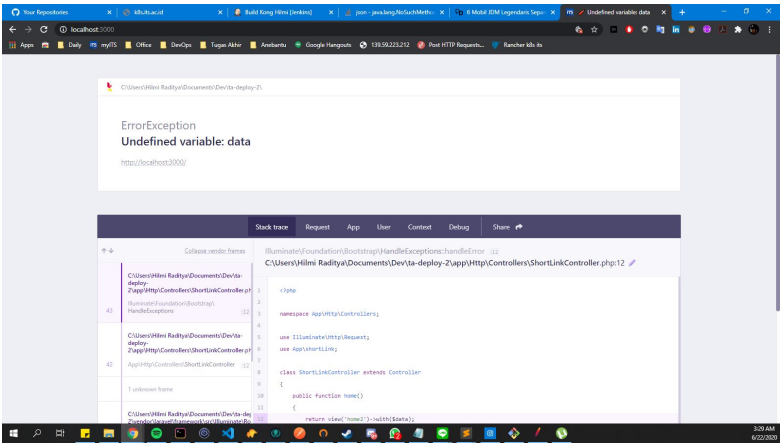
Gambar 5.11 *Virtual Service* dan *Deployment* Versi Terbaru yang Telah Berjalan Sukses di Kubernetes *Cluster* Pada Kiali Graf



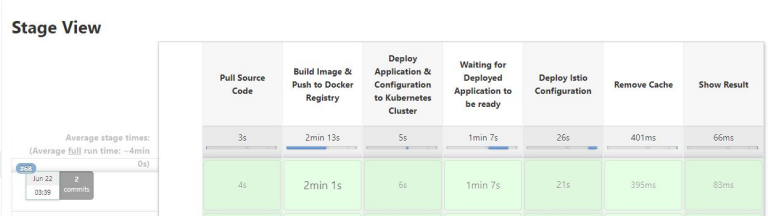
Gambar 5.12 Aplikasi Versi Terbaru yang Berhasil di *Deploy* Pada kubernetes *cluster*

5.3.1.2 Hasil Uji Coba Terdapat Error Pada Aplikasi Yang Akan Di Deploy

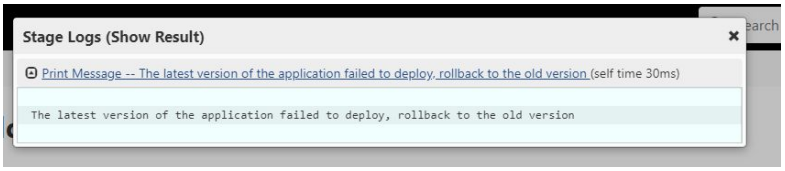
1. **Studi Kasus Framework Laravel**



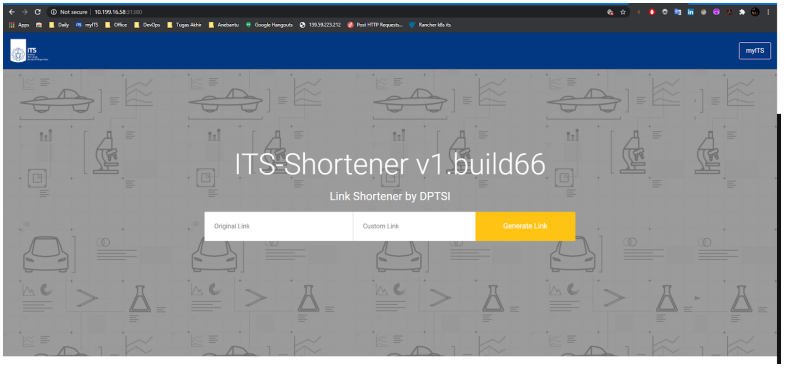
Gambar 5.13 Aplikasi yang Akan di *Deploy* Dalam Kondisi *Error* dan Masih Berjalan Pada *Localhost* Pengembang



Gambar 5.14 Pipeline yang Berjalan Pada *Jenkins* Ketika Melakukan *Update* Aplikasi



Gambar 5.15 *Output* Pipeline yang Berjalan Pada *Jenkins* Ketika Melakukan *Update* Aplikasi



Gambar 5.16 Versi Aplikasi Pada Kubernetes *Cluster* Masih Menggunakan Versi Sebelumnya Dikarenakan Terdapat *Error*

2. Studi Kasus Kong Api Gateway

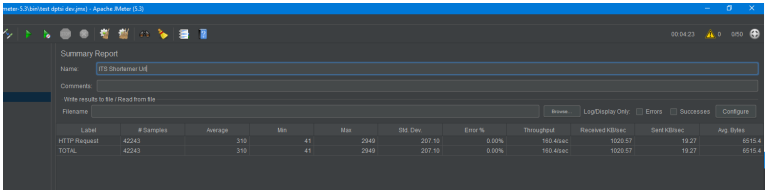
Pada uji coba studi kasus Kong Api Gateway, Kong tidak dapat berjalan apabila ada error, dikarenakan pada saat proses *build image* terdapat *validation*. Sehingga pengujian dengan *case* Terdapat *error* pada aplikasi yang akan di deploy pada studi kasus *Kong Api Gateway* tidak dilakukan.

5.3.2 Hasil Uji Coba Performa

Pada Uji coba ini digunakan aplikasi Apache Jmeter untuk melakukan uji coba performa, dan hasil dari pengujian ini berbentuk jumlah *samples* request yang dikirimkan serta jumlah persentase error request yang terjadi. Durasi uji coba performa adalah sama dengan durasi yang dilakukan Jenkins Pipeline dalam menjalankan satu build.

5.3.2.1 Hasil Uji Coba Performa dengan Kasus tidak terdapat error pada aplikasi yang akan di deploy.

1. **Studi Kasus Framework Laravel**



Gambar 5.17 *Summary Report* Apache JMeter

Pada studi kasus *Framework* digunakan *apache jmeter* sebagai *load tester*, Pengujian dilakukan dari *personal computer*.

Dari yang dilakukan Pipeline Jenkins dalam menjalankan satu build, didapatkan 42.243 HTTP request dengan tingkat persentase error *(non 200 request)* sebesar 0%. Sehingga dapat dikatakan proses pengalihan traffic ke aplikasi versi baru *zero downtime*.

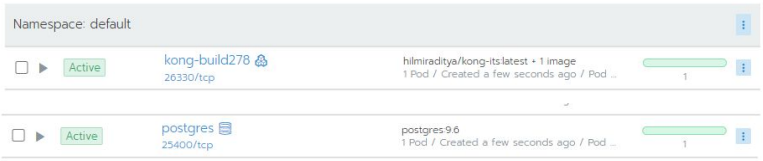
2. **Studi Kasus Kong Api Gateway**

Pada studi kasus *Kong Api Gateway* digunakan *siege* sebagai *load tester*, dikarenakan pengujian dilakukan dari sistem operasi *Ubuntu* dan tidak dari *personal computer*.

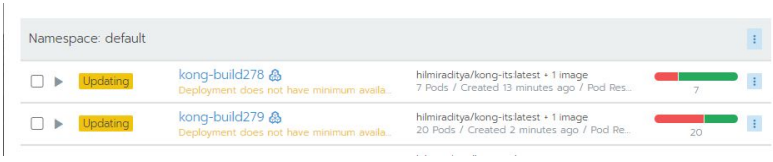
Pada pengujian ini, *Siege* melakukan akses pada endpoint */mock* dengan method POST yang terdapat *bearer token* pada

header, *bearer token* didapatkan dari *my.its.ac.id*, dan pada saat proses load test, dilakukan perubahan *image* pada *Kong* serta *Sidecar Kong* ke versi terbaru

Dalam studi kasus ini dilakukan dengan 1000 request tiap detik dengan nilai maksimal *Horizontal Pod Auto Scaler* pada *Deployment Kong* sebesar 20 *Pod* serta PostgreSQL sebesar 20 *Pod* dengan durasi load tester sebesar 7 menit. Load test dimulai dengan kondisi *idle*, yang dimana masing - masing Deployment *Kong* dan *PostgreSQL* berjumlah 1 Pod.



Gambar 5.18 *Deployment* Kong dan Postgres Dalam Kondisi *Idle*



Gambar 5.19 *Deployment* Kong Dedang Dalam Proses *Mirroring Traffic*

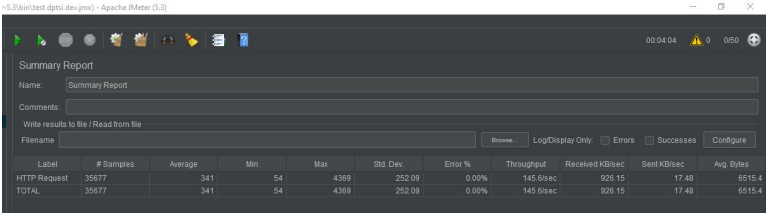
```
Lifting the server siege...
Transactions:                  96933 hits
Availability:                  95.77 %
Elapsed time:                 419.59 secs
Data transferred:             138.38 MB
Response time:                  1.10 secs
Transaction rate:             231.02 trans/sec
Throughput:                     0.33 MB/sec
Concurrency:                  254.51
Successful transactions:       96933
Failed transactions:            4278
Longest transaction:           18.10
Shortest transaction:           0.00

LOG FILE: /var/log/log/siege.log
You can disable this log file notification by editing
/home/user/.siege/siege.conf and changing 'show-logfile' to false.
[error] unable to create log file: /var/log/log/siege.log: No such file or directory
user@worker01:~$
```

Gambar 5.20 Hasil *Load Test*

Dari yang dilakukan Pipeline Jenkins dalam menjalankan satu build, didapatkan 96.933 HTTP request dengan tingkat persentase error *(non 200 request)* sebesar 4.4 %.

5.3.2.2 Hasil Uji Coba Performa dengan Kasus terdapat error pada aplikasi yang akan di deploy.

1. Studi Kasus Framework Laravel



Gambar 5.21 *Summary Report* Apache JMeter

Dari yang dilakukan Pipeline Jenkins dalam menjalankan satu build, didapatkan 35.667 HTTP request dengan tingkat persentase error *(non 200 request)* sebesar 0%. Sehingga dapat

dikatakan proses pengalihan traffic ke aplikasi versi baru *zero downtime*.

2. Studi Kasus Kong Api Gateway

Pada uji coba studi kasus Kong Api Gateway, Kong tidak dapat berjalan apabila ada error, dikarenakan pada saat proses *build image* terdapat *validation*. Sehingga pengujian dengan *case* Terdapat *error* pada aplikasi yang akan di deploy pada studi kasus *Kong Api Gateway* tidak dilakukan.

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 berhasil mengimplementasikan shadow deployment menggunakan Istio. Konfigurasi *mirroring traffic* pada Istio dapat dilakukan pada Virtual Service, yang merupakan salah satu komponen file dari Istio.
2. Telah berhasil diimplementasikan shadow deployment pada pipeline CI/CD untuk platform myITS, konfigurasi Istio yang dibutuhkan pada pipeline dikemas dalam bentuk Helm Chart dan disimpan pada chartmuseum lalu diintegrasikan dengan Kubernetes CLI dan Jenkins.
3. Downtime tidak terjadi pada studi kasus web URL shorten, namun pada studi kasus Kong api gateway yang terjadi adalah error non-200 request dan terjadi pada saat proses *scale up*, yang dimana tidak tercakup dalam tugas akhir ini.

6.2 Saran

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

1. Pada tugas akhir ini, sistem hanya dapat mengakomodir

perubahan pada aplikasi, tidak pada database, untuk pengembangan kedepannya sistem dapat diimplementasikan untuk mengakomodir perubahan yang terjadi pada database.

2. Pada tugas akhir ini, proses testing pada *mirroring traffic* yang bertujuan untuk menghindari error pada lingkungan produksi hanya menggunakan *curl* dan mengakses suatu endpoint pada suatu *pod,* Untuk tahap pengembangan kedepannya *mirroring traffic* dapat diintegrasikan dengan *PHP Unit Test* untuk studi kasus Framework Laravel.

DAFTAR PUSTAKA

[1]	"Istio Traffic Mirroring" [Online]. Available: https://istio.io/latest/docs/tasks/traffic-management/mirroring/ [Accessed 07 06 2020].
[2]	"Six Strategies for Application Deployment," 21 10 2017. [Online]. Available: https://thenewstack.io/deployment-strategies/. [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]	"Performance Test Sederhana dengan jMeter", 08 05 2020. [Online]. Available:http://blog.hendri.web.id/performance-test-sederhana-dengan-jmeter//. [Accessed 07 06 2020].

[8]	“Kiali : Service mesh management for Istio” [Online]. Available: https://kiali.io/. [Accessed 07 06 2020].

\

BIODATA PENULIS

Hilmi Raditya Prakoso lahir di Jakarta pada tanggal 11 Maret 1998. Penulis menempuh pendidikan formal di TK Pembangunan Jaya (2002-2004), SD Al - Azhar (2004-2010), SMP An - Nisaa' (2010-2013), dan SMA di Al - Azhar (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 Manajemen Informasi. Penulis dapat dihubungi melalui nomor handphone 087855857881 atau di email raditya113@gmail.com.