



INSTITUT TEKNOLOGI DEL

Implementation Design Environment Docker, Git, and Paas.

Study Case: LAN Institut Teknologi Del

TUGAS AKHIR

Oleh:

13317005 Andronikus Silitonga

13317016 Lambok Parsaulian Silitonga

13317024 Hardiman Utama Hotlas Tambun

FAKULTAS INFORMATIKA DAN TEKNIK ELEKTRO

PROGRAM STUDI DIII TEKNOLOGI KOMPUTER

LAGUBOTI

JULI 2020



INSTITUT TEKNOLOGI DEL

Implementation Design Environment Docker, Git, and Paas.

Study Case: LAN Institut Teknologi Del

TUGAS AKHIR

Disampaikan Sebagai Bagian Dari Persyaratan Kelulusan

Diploma 3 Program Studi Teknologi Komputer

Oleh:

13317005 Andronikus Silitonga

13317016 Lambok Parsaulian Silitonga

13317024 Hardiman Utama Hotlas Tambun

FAKULTAS INFORMATIKA DAN TEKNIK ELEKTRO

PROGRAM STUDI DIII TEKNOLOGI KOMPUTER

LAGUBOTI

JULI 2020

HALAMAN PERNYATAAN ORISINALITAS

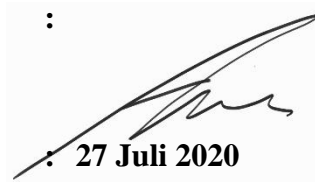
Tugas Akhir ini adalah hasil karya saya sendiri, dan semua sumber baik yang dikutip maupun dirujuk telah saya nyatakan dengan benar.

Nama : Andronikus Silitonga

NIM : 13317005

Tanda Tangan :

Tanggal : 27 Juli 2020



Nama : Lambok Parsaulian Silitonga

NIM : 13317016

Tanda Tangan :

Tanggal : 27 Juli 2020



Nama : Hardiman Utama Hotlas Tambun

NIM : 13317024

Tanda Tangan :

Tanggal : 27 Juli 2020



HALAMAN PENGESAHAN

Tugas Akhir ini diajukan oleh :

- 1 Nama : Andronikus Silitonga
NIM : 13317005
Program Studi : D3 Teknologi Komputer
- 2 Nama : Lambok Parsaulian Silitonga
NIM : 13317016
Program Studi : D3 Teknologi Komputer
- 3 Nama : Hardiman Utama Hotlas Tambun
NIM : 13317024
Program Studi : D3 Teknologi Komputer

Judul Tugas Akhir : **Implementation Design Environment Docker, Git, and Paas. Study Case: LAN Institut Teknologi Del.**

Telah berhasil dipertahankan dihadapannya dewan penguji dan diterima sebagai bagian persyaratan yang diperlukan untuk memperoleh gelar Diploma III, pada program studi Diploma III Teknologi Komputer Fakultas Informatika dan Teknik Elektro Institut Teknologi Del.

DEWAN PENGUJI

- | | | |
|---------------|--|-----|
| Pembimbing I | : Istas Manalu, S.Si., M.Sc | () |
| Pembimbing II | : Marojahan MT. Sigiroy, ST., M.Sc | () |
| Penguji I | : Eka Stephani Sinambela, SST., M.Sc | () |
| Penguji II | : Ahmad Zatnika Purwalaksana, S.Si.,M.Si | () |

Ditetapkan di : Laguboti

Tanggal : 27 Juli 2020

KATA PENGANTAR

Puji dan syukur kepada Tuhan Yang Maha Esa atas rahmat yang diberikan kepada penulis sehingga Tugas Akhir ini dapat diselesaikan dengan baik dan tepat waktu. Laporan Tugas Akhir ini bertujuan untuk merancang dan implementasikan teknologi DevOps dan menerapkannya dalam lingkungan civitas Institut Teknologi DEL melalui jaringan LAN Institut Teknologi Del. Laporan Tugas Akhir ini merupakan sebagai syarat kelulusan Diploma III program studi Teknologi Komputer di Institut Teknologi Del.

Selama mengikuti pendidikan Diploma III program studi Teknologi Komputer sampai dengan proses penyelesaian Tugas Akhir, berbagai pihak telah membantu, membimbing, memberi dukungan dan memberikan fasilitas kepada penulis. Untuk itu pada kesempatan ini, penulis menyampaikan rasa terimakasih khususnya kepada:

1. Istas Manalu, S.Si., M.Sc sebagai dosen pembimbing I dan Marojahan MT. Sigirow, ST., M.Sc sebagai dosen pembimbing II yang telah memberikan masukan, saran, bimbingan dan arahan kepada penulis selama pengerjaan Tugas Akhir ini.
2. Orang tua dari para penulis yang telah memberikan dukungan penuh kepada penulis dalam menyelesaikan Tugas Akhir ini.
3. Teman-teman kuliah, teman-teman kemah siaga, dan semua pihak yang membantu dan memberikan dukungan selama pengerjaan Tugas Akhir ini.

Penulis menyadari bahwa dalam laporan Tugas Akhir ini masih banyak terdapat kekurangan. Sehingga, penulis mengharapkan kritik dan saran yang bersifat membangun untuk perbaikan laporan Tugas Akhir di masa mendatang. Akhir kata kami mengucapkan terima kasih.

Laguboti, Agustus 2020

Penulis

HALAMAN PERNYATAAN PERSETUJUAN PUBLIKASI TUGAS AKHIR UNTUK KEPENTINGAN AKADEMIS

Sebagai civitas akademik Institut Teknologi Del, saya yang bertanda tangan di bawah ini :

- 1 Nama : Andronikus Silitonga
NIM : 13317005
Program Studi : D3 Teknologi Komputer
 - 2 Nama : Lambok Parsaulian Silitonga
NIM : 13317016
Program Studi : D3 Teknologi Komputer
 - 3 Nama : Hardiman Utama Hotlas Tambun
NIM : 13317024
Program Studi : D3 Teknologi Komputer
- Jenis Karya : Tugas Akhir

Demi pengembangan ilmu pengetahuan, menyetujui untuk memberikan kepada Institut Teknologi Del Hak Bebas Royalti Noneksklusif (*Non-exclusive Royalty-Free Right*) atas karya ilmiah saya yang berjudul: **Implementation Design Environment Docker, Git, and Paas. Study Case: LAN Institut Teknologi Del.**


Dengan Hak Bebas Royalti Noneksklusif ini Institut Teknologi Del berhak menyimpan, mengalih/media-format dalam bentuk pangkalan data (*database*), merawat, dan memublikasikan tugas akhir saya selama tetap mencantumkan nama saya sebagai penulis/pencipta dan sebagai pemilik hak cipta.

Dibuat di : Laguboti

Pada tanggal : 27 Juli 2020



(Andronikus Silitonga)



(Lambok Silitonga)



(Hardiman Utama Hotlas Tambun)

ABSTRAK

DevOps merupakan serangkaian praktik yang mengotomasisasi proses antara *software development* dan *development team* agar dapat melakukan proses *build*, *test*, dan *release software* lebih cepat dan lebih handal. DevOps yang sudah berjalan dengan baik akan menghasilkan produk yang stabil dan meningkatkan nilai dari produk itu sendiri. DevOps memiliki praktik yang disebut *continuous integration* (CI) dan *continuous delivery* (CD) yang berkonsep *life cycle*. Tugas Akhir dengan judul **“Implementation Design Environment Docker, Git, and Paas. Study Case: LAN Institut Teknologi Del”** memanfaatkan praktik DevOps untuk membuat sebuah sistem yang dapat digunakan *developer* dalam rangkaian pengembangan aplikasi atau *software development* secara otomatis berkonsep *life cycle*.

Tools praktik DevOps yang digunakan dalam penyelesaian Tugas Akhir ini antara lain *Docker*, *Git*, dan sebuah PaaS yaitu *Dokku*. *Docker* digunakan untuk mengepak sebuah *software* secara lengkap sehingga dapat berfungsi dengan baik. PaaS yaitu *Dokku* digunakan untuk penyebaran aplikasi yang dirancang dan digunakan melalui jaringan LAN, sehingga tidak terpengaruh oleh *bandwidth* internet. Selain sebagai manajemen *source code*, *Git* juga memiliki konsep CI/CD yang melakukan otomatisasi seluruh kegiatan *software development*, sehingga mempermudah *developer* dalam pengembangan aplikasi.

Hasil dari pengerjaan Tugas Akhir ini membuktikan bahwa praktik DevOps dalam pengembangan aplikasi atau *software development* menggunakan *tools Docker*, *Git*, dan Paas (*Dokku*) dapat berjalan dengan baik, cepat, aman, dan otomatis. Setiap *tools* yang digunakan akan berintegrasi satu sama lain melalui CI/CD *Gitlab*, memiliki konsep *life cycle*, dan dapat digunakan menggunakan jaringan LAN.

Kata Kunci: *DevOps, Docker, Git, PaaS, LAN, life cycle, CI/CD*

ABSTRACT

DevOps is a series of practices that automate the process between software development and development team so that the process of building, testing and releasing software is faster and more reliable. DevOps that are already running well will produce a stable product and increase the value of the product itself. DevOps has a practice called continuous integration (CI) and continuous development (CD) with a life cycle concept. Final Project with the title "Implementation Design Environment Docker, Git, and Paas. Study Case: LAN Institut Teknologi Del" utilizes the practice of DevOps to create a system that developers can use in a series of application development or software development automatically with a life cycle concept.

DevOps practice tools used in completing this Final Project include Docker, Git, and a PaaS namely Dokku. Docker is used to pack a complete software so that it can function properly. PaaS namely Dokku is used for application deployment that is designed and used over a LAN network, so it is not affected by internet bandwidth. Aside from being a source code management, Git also has a CI/CD concept that automates all software development activities, making it easier for developers to develop applications.

The results of this Final Project proves that the practice of DevOps in application development or software development using Docker, Git, and Paas (Dokku) tools can work well, quickly, safely, and automatically. Each tool used will integrate with each other through CI / CD Gitlab, has a life cycle concept, and can be used using a LAN network.

Key Words: *DevOps, Docker, Git, Paas, LAN, life cycle, CI/CD*

DAFTAR ISI

KATA PENGANTAR.....	v
DAFTAR TABEL	xii
DAFTAR GAMBAR.....	xiii
BAB 1 PENDAHULUAN	1
1.1 Latar Belakang	1
1.2 Rumusan Masalah.....	2
1.3 Tujuan	2
1.4 Ruang Lingkup.....	3
1.5 Pendekatan	3
1.6 Sistematika Penyajian	4
1.7 Istilah, Definisi, dan Singkatan	4
BAB 2 TINJAUAN PUSTAKA.....	7
2.1 Landasan Teori.....	7
2.1.1 DevOps	7
2.1.2 <i>Lifecycle</i>	8
2.1.3 <i>Docker</i>	8
2.1.3.1 <i>Dockerfile</i>	11
2.1.3.2 <i>Private Docker Registry</i>	11
2.1.4 <i>Git</i>	12
2.1.4.1 <i>Gitlab</i>	14
2.1.5 Paas	15
2.1.5.1 <i>Dokku</i>	16
2.1.6 <i>Continuous</i> Integration & Continuous Delivery (CI/CD).....	16
2.1.7 LAN	17
2.2 Kajian Penelitian yang Relevan	17
2.3 Jawaban dari permasalahan & kontribusi dalam Tugas Akhir.....	18
BAB 3 ANALISIS DAN PERANCANGAN SISTEM	19
3.1 Analisis	19
3.1.1 Analisis Masalah.....	19
3.1.2 Analisis Pemecahan Masalah.....	20
3.1.3 Analisis Kebutuhan Sistem	21
3.2 Perancangan Pembangunan Sistem.....	22

3.3	<i>Flowchart</i> Sistem	22
3.4	Perancangan Sistem	23
3.4.1	Gambaran Umum Sistem	23
3.5	Skenario Pengujian	26
3.5.1	Pengujian Mengunduh <i>Docker Image</i> LAN dan Internet IT Del	26
3.5.2	Pengujian <i>Update</i> dengan <i>Website Static</i> dan <i>Dinamic</i>	27
BAB 4 IMPLEMENTASI DAN PENGUJIAN.....		28
4.1	Topologi Sistem	28
4.2	Implementasi <i>Environment (Tools)</i> dan Alur Pengerjaan Proyek Menggunakan <i>Gitlab</i> , <i>Docker</i> , dan Paas <i>Dokku</i> dengan Konsep DevOps	28
4.2.1	Konfigurasi <i>Static IP</i> Adress pada Ubuntu 18.04	28
4.2.2	<i>Docker</i>	29
4.2.2.1	Instalasi <i>Docker</i> pada <i>Ubuntu</i> 18.04	29
4.2.2.2	<i>Dockerfile</i>	30
4.2.2.3	Install dan konfigurasi <i>Docker Registry</i>	32
4.2.3	<i>Gitlab</i>	34
4.2.3.1	Konfigurasi dan Instalasi <i>Gitlab</i> pada Server <i>Ubuntu</i> 18.04 Menggunakan <i>Docker</i>	35
4.2.3.2	Instalasi <i>Git</i> pada <i>Client</i>	36
4.2.4	Konfigurasi dan Instalasi Paas <i>Dokku</i>	37
4.2.5	Panduan Integrasi <i>Docker</i> , <i>Gitlab</i> , dan <i>Dokku</i> dengan Konsep <i>Lifecycle</i> DevOps dengan Menggunakan CI/CD <i>Docker Deployment</i> dengan <i>Dokku</i>	39
4.2.6	Metode Alternatif <i>Deploy</i> Aplikasi	42
4.2.7	Pengujian Jaringan	43
4.2.8	Perbandingan <i>Memory Usage</i> Ketika Menjalankan <i>Dokku</i> dan <i>Gitlab</i> secara bersamaan dan bergantian	46
4.2.9	Pengujian <i>Lifecycle</i> Menggunakan <i>Update Source Code Website</i>	46
4.2.9.1	Pengujian pada <i>Website HTML</i>	47
4.2.9.2	Uji Coba <i>Update Website Static</i>	50
4.2.9.3	Skenario pada <i>Website PHP</i>	52
4.2.9.4	Skenario pada <i>Website PHP</i> dan <i>Database MYSQL</i>	58
BAB 5 KESIMPULAN DAN SARAN.....		68
5.1	Kesimpulan	68
5.2	Saran	68
DAFTAR PUSTAKA DAN RUJUKAN.....		69

LAMPIRAN.....	72
----------------------	-----------

DAFTAR TABEL

Tabel 1 Daftar Istilah	5
Tabel 2 Daftar Singkatan	5
Tabel 3 Kebutuhan Hardware	21
Tabel 4 Kebutuhan Software	21

DAFTAR GAMBAR

Gambar 1. Docker.....	9
Gambar 2. Arsitektur Docker.....	9
Gambar 3. Jalan Kerja Docker.....	10
Gambar 4. Git.....	12
Gambar 5. Cara Kerja Git	13
Gambar 6. Gitlab.....	15
Gambar 7. Tampilan Dashboard Gitlab	15
Gambar 8. Alur Pengerjaan proyek dengan konsep DevOps.....	22
Gambar 9. Gambaran Umum Hubungan Antar Sistem	24
Gambar 10. Docker container dari docker registry yang berjalan	32
Gambar 11. List daftar docker image yang berhasil disimpan di private docker registry	33
Gambar 12. Tampilan login Gitlab	34
Gambar 13. Tampilan penyimpanan proyek.....	34
Gambar 14. Docker container Gitlab yang berjalan.....	36
Gambar 15. Instalasi Dokku	38
Gambar 16. Perbedaan Arsitektur VM dan docker.....	40
Gambar 17. Tampilan CI/CD Gitlab.....	41
Gambar 18. Proses CI/CD Gitlab.....	42
Gambar 19. Waktu dalam mengunduh docker image dari dockerhub menggunakan jaringan internet IT Del.....	43
Gambar 20. Hasil unduhan docker images	43
Gambar 21. Waktu mengunduh docker dari private docker registry menggunakan LAN IT Del.....	44
Gambar 22. Hasil skenario pengujian	45
Gambar 23. Docker container Gitlab dan Dokku berjalan bersamaan pada satu komputer	46
Gambar 24. Docker container Gitlab setelah dihentikan dan menyisakan Dokku yang berjalan pada sebuah komputer.....	46
Gambar 25. Code dockerfile	48
Gambar 26. Docker dari website yang telah berhasil dideploy ke dokku.....	50
Gambar 27. Website sebelum diupdate.....	50
Gambar 28. File website static yang diuji.....	50
Gambar 29. Website setelah diupdate.....	52
Gambar 30. Website PHP setelah dideploy	55
Gambar 31. Website PHP setelah diupdate	58
Gambar 32. Tampilan website PHP menggunakan database MySQL.....	66
Gambar 33. Mengupdate website dengan mengupdate database.....	66
Gambar 34. Tempilan website setelah database update, Jakarta menghilang dari website	67
Gambar 35. Docker image dapat dipush ke private docker registry	67

BAB 1 PENDAHULUAN

Pada bab ini dijelaskan mengenai latar belakang pemilihan topik, tujuan pelaksanaan Tugas Akhir, ruang lingkup kajian yang mendasari Tugas Akhir, pendekatan yang dilakukan selama melaksanakan kajian dan sistematika penyajian Tugas Akhir yang disediakan dalam laporan.

1.1 Latar Belakang

Kebutuhan setiap individu atau perusahaan akan teknologi yang semakin meningkat setiap harinya menjadikan keberadaan *software development* semakin penting dalam suatu industri khususnya industri teknologi. Salah satu permasalahan mendasar proses pengembangan *software* adalah kebutuhan yang tidak lengkap saat awal pengembangan atau abstraksi kebutuhan pengguna yang kurang terpetakan secara sistematis, rumit oleh pengembang, ketidaksesuaian dapat dilihat oleh *developer* setelah *developer* menyelesaikan perrilisan suatu produk [1].

Salah satu metode yang digunakan dalam proses pengembangan *software* adalah metode DevOps yang memiliki kemampuan untuk melakukan keselarasan kebutuhan pengguna dengan pengembangan aplikasi yang berkelanjutan, dan cepat selama pengembangan dan pengoperasian berlangsung [1]. Dalam dunia informasi dan teknologi, kecepatan *release* suatu *software* hingga memperoleh *feedback* sebanyak-banyaknya adalah salah satu keberhasilan untuk memenangkan pasar informasi dan teknologi. DevOps menerapkan konsep *release* produk yang disebut dengan konsep *life cycle* yang memiliki pengertian berulang dan tidak ada akhirnya[2]. Karena alasan itu, banyak perusahaan IT menerapkan konsep DevOps untuk mendukung. Dalam mendukung konsep DevOps, dibutuhkan berbagai *tools*, tidak hanya cepat, namun juga harus stabil, yaitu tidak mengganggu infrastruktur *software development*.

Institut Teknologi Del merupakan perguruan tinggi swasta yang terletak di Sitoluama, Laguboti, Toba Samosir, Sumatera Utara. Institut Teknologi Del memiliki tugas Mata Kuliah, Proyek akhir Tahun, dan Tugas Akhir yang diberikan kepada mahasiswa/i sebagai simulasi dari dunia kerja yang sesungguhnya. Dalam mengerjakan proyek tersebut mahasiswa/i Institut Teknologi Del masih

menggunakan metode manual dan belum mengikuti metode *software development* yang sedang dipakai di dunia industri pada saat ini, sehingga pengerjaan proyek tidak efisien karena tidak dapat berkolaborasi dengan baik antara sesama anggota kelompok dan aplikasi tidak berjalan sebagaimana mestinya karena kesalahan konfigurasi maupun OS yang berbeda.

Salah satu hal yang penting dalam pengoperasian *software development* adalah jaringan dan konektivitas yang baik dan lancar. Suatu proyek dari *source code* yang sedang dikerjakan oleh *developer* memiliki ukuran data yang tergolong besar sehingga diperlukan suatu jaringan yang cepat dan dapat diandalkan oleh mahasiswa/i Institut Teknologi Del. Salah satu solusi yang didapat adalah dengan menggunakan jaringan LAN Institut Teknologi Del yang memiliki konektivitas yang cukup lancar untuk *software development*.

Berdasarkan masalah yang didapat dari masalah pengerjaan tugas dan proyek di Institut Teknologi Del, maka penulis membuat sebuah sistem *environment* yang berjudul “Implementation Design Environment Docker, Git, and Paas. Study Case: LAN Institut Teknologi Del”. Dengan adanya *environment* tersebut, diharapkan mahasiswa/i Institut Teknologi Del dapat menggunakan dan menerapkan teknologi *software development* yang berkembang di industri informasi dan teknologi saat ini.

1.2 Rumusan Masalah

Rumusan masalah dari pembuatan Tugas Akhir ini adalah bagaimana menerapkan *tools environment* DevOps berkonsep *life cycle* dan memanfaatkannya dalam pengerjaan tugas dan proyek mahasiswa/i Institut Teknologi Del.

1.3 Tujuan

Adapun tujuan yang ingin dicapai dalam pelaksanaan Tugas Akhir ini adalah sebagai berikut:

1. Mengintegrasikan dan mengimplementasikan *tools* yang digunakan untuk membangun *system* yaitu *Git*, *Docker*, dan *Paas* sehingga menghasilkan sebuah *environment software development* yang berkembang dan digunakan oleh

industri informasi dan teknologi modern untuk diterapkan dan digunakan mahasiswa ke dalam lingkungan Institut Teknologi Del.

2. Merancang *guideline system environment development software* dengan konsep *life cycle*.

1.4 Ruang Lingkup

Adapun ruang lingkup dari Tugas Akhir ini adalah:

1. Pengimplementasian *environment software development Docker, Git, dan Paas* yang dapat dimanfaatkan mahasiswa/i Institut Teknologi Del.
2. Mengintegrasikan *Docker, Git, dan Paas* sehingga membentuk sebuah *environment* dengan konsep *Life cycle*.
3. Pembangunan source code proyek menggunakan *Dockerfile*.

1.5 Pendekatan

Pendekatan yang dilakukan oleh penulis dalam pengerjaan Tugas Akhir ini adalah:

1. Research Problem

Membuat beberapa pertanyaan-pertanyaan terkait topik yang telah ditetapkan.

2. Study literature

Melakukan *study literature* dengan membaca dan memahami beberapa jurnal yang berkaitan dengan Tugas Akhir.

3. Experiment Planning

Merancang sebuah *environment* sistem untuk mengimplementasikan seluruh komponen *tools* yang digunakan teknologi DevOps yang akan digunakan untuk pengerjaan tugas dan proyek mahasiswa/i Institut Teknologi Del.

4. Testing and Analysis

Melakukan pengujian terhadap implementasi yang telah dilakukan dan juga melakukan analisis terhadap hasil implementasi.

5. Documentation

Menuliskan laporan dan hasil dokumentasi selama masa pengerjaan Tugas Akhir.

1.6 Sistematika Penyajian

Secara garis besar dokumen ini disajikan dalam 5 bab yang disusun dengan sistematika berikut:

1. Bab 1 Pendahuluan

Pada bab ini berisi penjelasan mengenai latar belakang, tujuan pelaksanaan, ruang lingkup, pendekatan yang dilakukan serta sistematika penyajian Tugas Akhir.

2. Bab 2 Tinjauan Pustaka

Pada bab ini menguraikan setiap dasar teori serta perangkat yang akan digunakan pada pengerjaan Tugas Akhir.

3. Bab 3 Penelitian dan Analisis

Pada bab ini berisi tentang penjelasan yang didapat setelah mempelajari *literature* dan *experiment planning*.

4. Bab 4 Implementasi dan Pengujian

Pada bab ini berisi uraian implementasi yang dilakukan pada Tugas Akhir. Proses implementasi yang dilakukan adalah penerapan teknologi DevOps pada pembuatan tugas atau proyek website mahasiswa/i Institut Teknologi Del

5. Bab 5 Kesimpulan dan Saran

Pada bab ini berisi penjelasan mengenai kesimpulan dari pengerjaan Tugas Akhir serta saran-saran yang dibutuhkan untuk pengembangan penelitian.

1.7 Istilah, Definisi, dan Singkatan

Daftar istilah, definisi, dan singkatan diperlukan pada dokumen ini untuk mempermudah pembaca dalam memahami segala informasi yang terdapat pada

dokumen. Daftar istilah, definisi, dan singkatan yang terdapat dalam dokumen ini dapat dilihat pada tabel berikut:

Tabel 1 Daftar Istilah

Istilah	Defenisi
DevOps	Pengembangan <i>software</i> antar pengembang aplikasi (<i>Dev</i>) & bagian operasi aplikasi(<i>Ops</i>).
<i>Developer</i>	Tim pengembang suatu software
<i>Operations</i>	Bagian <i>Hardware</i> suatu sistem komputer
<i>Device</i>	Alat atau perangkat
<i>Event</i>	Suatu kejadian dari rangkaian peristiwa dalam suatu sistem
<i>Script</i>	Bahasa pemograman tingkat rendah
<i>Life cycle</i>	Siklus berputar
<i>Open source</i>	Terbuka untuk <i>user</i> atau <i>user</i>
<i>Tools</i>	Alat
<i>Host</i>	Perangkat yang terhubung ke jaringan komputer.
<i>Administrator</i>	Seseorang yang memegang kendali penuh terhadap segala kegiatan yang ada pada sebuah sistem
<i>User</i>	Pengguna
<i>Event</i>	Peristiwa yang terjadi
<i>Install</i>	Memasang program (<i>software</i>) ke dalam sistem
<i>Platform</i>	Sebuah <i>software</i> dieksekusi.
<i>Deploy/ Publish</i>	Kegiatan yang bertujuan untuk menyebarkan aplikasi yang telah dikerjakan.

Daftar singkatan yang terdapat dalam dokumen Tugas Akhir ini dapat dilihat pada Tabel berikut:

Tabel 2 Daftar Singkatan

No	Singkatan	Deskripsi
1.	LAN	<i>Local Area Network</i>
2.	TA	Tugas Akhir

3.	IT	Informasi Teknologi
4.	Paas	<i>Platform as a Service</i>
5.	PC	<i>Personal Computer</i>
6.	CLI	<i>Command Line Interface</i>
7.	DVCS	<i>Distributed Version Control Sistem</i>
8.	API	<i>Aplication Programming Interface</i>
9.	OS	<i>Operating Sistem</i>
10.	Guest OS	<i>Guest Operating Sistem</i>
11.	IT Del	Institut Teknologi Del

BAB 2 TINJAUAN PUSTAKA

Pada bab ini akan dijelaskan mengenai landasan teori yang digunakan sebagai dasar teori dalam pengerjaan Tugas Akhir dan penelitian-penelitian terdahulu yang berhubungan dengan penelitian yang dilakukan pada Tugas Akhir.

2.1 Landasan Teori

Pada landasan teori akan dijelaskan mengenai dasar teori yang berhubungan dengan Tugas Akhir seperti pengertian dan konsep dasar mengenai sistem.

2.1.1 DevOps

Teknologi DevOps merupakan model yang menggabungkan praktik dan alat yang membantu meningkatkan nilai dari suatu produk[3]. Teknologi DevOps adalah serangkaian praktik yang digunakan dalam mengotomasi proses pengembangan tim *development* dan tim *operations*. Organisasi tradisional melakukan pengembangan aplikasi oleh tim *development* dan tim *operations* yang masing-masing memiliki tujuan tersendiri[4]. Organisasi tradisional dalam tim *development* bertugas menciptakan fungsi baru dan memperbaiki *bug source code*, sementara tim *operations* membuat infrastruktur yang agar *software* berjalan dengan stabil. Menurut penelitian “*Deliver Software Faster*”, kerjasama antara tim informasi dan teknologi untuk pengembangan dan pemeliharaan *software* sangat dibutuhkan, contohnya dengan mengadopsi DevOps.

DevOps mendukung kecepatan produksi *software* dan otomasi mengurangi usaha yang dibutuhkan ketika menyiapkan perilisan, sehingga memungkinkan organisasi untuk rilis lebih sering sesuai keperluan[5]. Dengan menerapkan pola kerja DevOps, perusahaan dapat menambah fitur-fitur baru dengan pengujian terhadap sebagian *user*, sehingga kualitas layanan dapat selalu meningkat. Dengan meningkatnya kualitas layanan, maka kepuasan pelanggan dapat lebih mudah terpenuhi.

2.1.2 *Lifecycle*

Pengembangan aplikasi memerlukan waktu yang cukup banyak karena spesifikasi dan kompleksitasnya masing masing. Dalam pengiriman aplikasi dalam rentang waktu singkat, pengembangan *software* mengikuti serangkaian praktik universal yang disebut siklus hidup atau *life cycle* DevOps.

Transformasi efisien diadopsi dalam organisasi IT untuk prinsip-prinsip integrasi berkelanjutan dalam SDLC yang dapat meningkatkan efisiensi pengembangan dalam proyek. Dengan waktu telah disadari bahwa optimasi tidak membantu hanya dalam integrasi yang berkelanjutan untuk membuat proses pengiriman *software* menjadi efisien. Kecuali jika semua modul dalam siklus hidup pengiriman *software* dirancang dengan baik, diimplementasikan dan dioptimalkan. Ini adalah masalah dengan teknologi sebelumnya dan DevOps mengatasinya[6].

Konsep *Lifecycle* memiliki makna antara *Life* & *cycle*, merupakan pengembangan *software* tersebut tidak memiliki akhir dan akan di-*update* kembali ketika pengembangan *software* diperlukan, sehingga untuk meningkatkan kecepatan *release* hasil proyek.

2.1.3 *Docker*

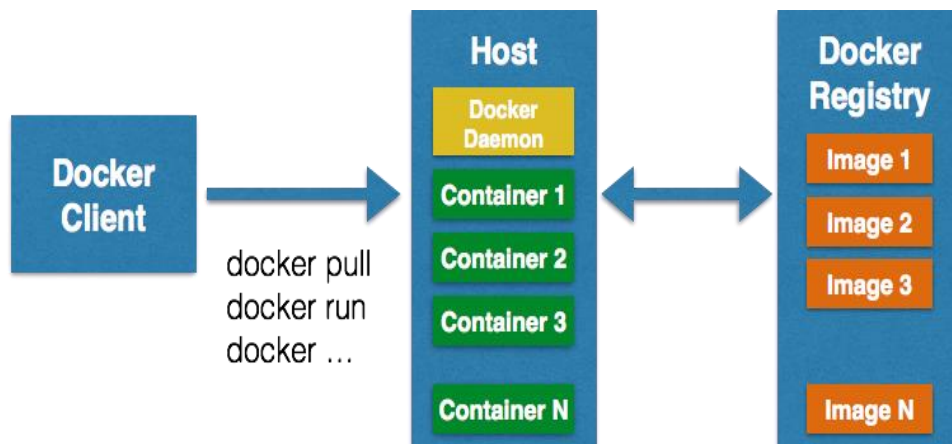
Virtualisasi berbasis *container software development* ringan karena kernel dapat membagi *useran resource* antar *container*, bertujuan agar kinerja *container* tidak saling terganggu antar lainnya[7]. Dengan *container*, sebuah program ‘diikat’ beserta *library*-nya, *file* konfigurasi, dan seluruh hal yang dibutuhkannya. Perbedaan yang sangat terlihat dibandingkan dengan virtualisasi adalah *container* memiliki ukuran *file* yang jauh lebih kecil karena tidak perlu menyiapkan OS secara penuh. Salah satu teknologi dengan konsep *container* itu adalah *Docker*. *Docker* adalah sebuah teknologi *container* yang banyak digunakan oleh pengembang aplikasi karena dianggap sebagai solusi tren saat ini. *Docker* dapat memudahkan proses penyebaran aplikasi, meningkatkan produktivitas pengembang, mengurangi beban server, dan mengurangi jumlah penyimpanan[7].



Gambar 1. *Docker*

(Sumber: <https://www.Docker.com/>)

Docker akan mengumpulkan pengaturan *software* beserta *file* atau hal pendukung lainnya dan menjadikannya sebuah *image* (istilah yang diberikan oleh *Docker*). Kemudian sebuah instansiasi dari *image* yang berjalan tersebut akan disebut *container*. Konsep arsitektur dari *Docker* dapat dilihat pada gambar berikut:



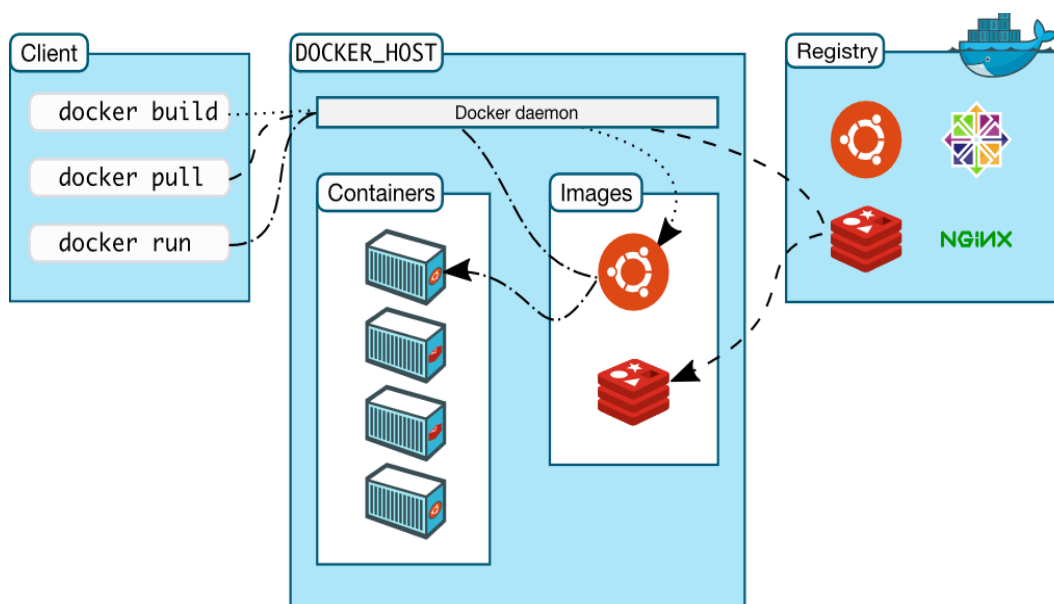
Gambar 2. Arsitektur *Docker*

(Sumber: <https://corenux.com/2019/06/23/belajar-Docker-part-1-pengenalan-Docker/>)

Pada Gambar 2. Arsitektur *Docker*, *Docker* memiliki arsitektur *client-server*. *Docker daemon* atau *server* bertanggung jawab untuk semua tindakan yang terkait dengan *container*. *Daemon* menerima perintah dari *client Docker* melalui

CLI atau *REST API*. *Client Docker* berada di *host* yang sama dengan *daemon* atau bisa juga ada di *host* lain.

Image adalah blok bangunan dasar *Docker*, *container* dibuat dari *image*. *Image* dapat dikonfigurasi dengan aplikasi dan digunakan sebagai *template* untuk membuat *container*, diatur secara berlapis yang artinya setiap perubahan pada *image* ditambahkan sebagai *layer* di atasnya, ketika *image* yang lama diperbaharui, maka secara otomatis *image* yang baru akan ditambahkan keatas urutan *image* yang lama.



Gambar 3. Jalan Kerja Docker

(Sumber: <https://medium.com/@muhammadardivan/Docker-orchestration-743bc2f2587b>)

Gambar 3 merupakan jalan kerja *Docker*, *Docker* menggunakan arsitektur *client-server*, *client Docker* berkomunikasi dengan *Docker daemon* yang mempunyai *task* untuk membangun, menjalankan, serta mendistribusikan wadah *Docker* yang berasal dari *client*. *Client* dapat menggunakan perintah melalui *command line* untuk memberi perintah atau berhubungan dengan *Docker host* atau *Docker server*, baik untuk menjalankan *container* atau mengunduh *image* yang berasal dari *Docker registry*.

Fitur terbaik *Docker* adalah kolaborasi. *Image Docker* dapat didorong ke repositori dan dapat ditarik ke *host* lain untuk menjalankan *container* dari *image* tersebut. *Dockerhub* memiliki ribuan *image* yang dibuat oleh *user* dan dapat menarik *image* tersebut ke *host* berdasarkan persyaratan aplikasi, *Dockerhub* merupakan *Docker registry official* yang bersifat *online*.

2.1.3.1 Dockerfile

Dockerfile adalah *recipe*, *source code*, untuk membuat sebuah *Docker image*, *user* dapat dengan mudah menentukan *software* yang diinstal pada *Docker image* proyek yang akan dibangun dengan menuliskan *script* pada *Dockerfile* terkait, *Dockerfile* menyediakan *script* singkat instalasi yang dapat dibaca manusia [8]

Dockerfile tidak terdapat ekstensi program, Ketika membangun sebuah *image*, *Docker* akan mencari *file* dengan nama *Dockerfile* secara default, pada umumnya, ketika ingin membangun sebuah *Docker image*, *command* akan dilakukan melalui *directory* lokasi *Dockerfile* berada, dengan tujuan pembangunan *Docker image* akan lebih cepat, karena *Docker* akan mencari *Dockerfile* pertama kali melalui lokasi dimana *command* pembangunan *Docker image* dilakukan, jika pembangunan *Dockerfile* menjadi *image* dilakukan melalui *directory* yang berbeda, maka *command* pembangunan *Docker image* tersebut mengarah pada lokasi *Dockerfile*.

2.1.3.2 Private Docker Registry

Docker registry adalah kumpulan dari *Docker image*, dengan menggunakan *Docker registry*, *user* dapat menggunakan *Docker image* yang telah dibuat oleh *developer* yang lain, sehingga mempermudah *user* dalam pengembangan aplikasi. *Docker* menyediakannya *docker registry* yang bersifat *online* melalui *dockerhub*, tetapi terkadang ada beberapa *developer* yang ingin agar *image docker* mereka dapat bersifat *private* agar *docker image* tersebut tidak dapat digunakan oleh orang lain dengan alasan kepentingan pribadi, sehingga muncul istilah yang mengacu pada *private docker registry*, yang merupakan persamaan dengan *dockerhub* tetapi *docker registry* hanya dapat diakses oleh *user* yang terhubung

dalam satu jaringan yang sama, sehingga *user* yang berada pada jaringan yang berbeda tidak dapat mengaksesnya. Keuntungan lainnya dari *private docker registry* adalah, karena diterapkan dalam jaringan LAN, sehingga cakupan kecepatan mengunduh *docker image* lebih cepat dibanding jaringan internet. *user* dapat mengunduh *docker image* yang berada pada *dockerhub* melalui internet dan menyimpannya kembali pada *private docker registry* agar dapat diakses oleh *user* lain, sehingga menghemat waktu, *user* dapat menyimpannya ke *private docker registry* yang terhubung pada jaringan LAN apabila *user* lain membutuhkannya.

2.1.4 *Git*

Git adalah sebuah *software open source*, memiliki performa tinggi, fleksibel, dan merupakan contoh dari DVCS yang digunakan dalam kernel proyek *linux*[8]. DVCS (*Distributed Version Control Sistem*) adalah suatu sistem yang tidak hanya memiliki satu tempat tunggal untuk menyimpan sejarah lengkap dari sebuah *software*, sistem tersebut mencatat setiap perubahan terhadap sebuah berkas atau kumpulan berkas sehingga pada suatu saat dapat kembali kepada salah satu versi dari berkas tersebut[9].



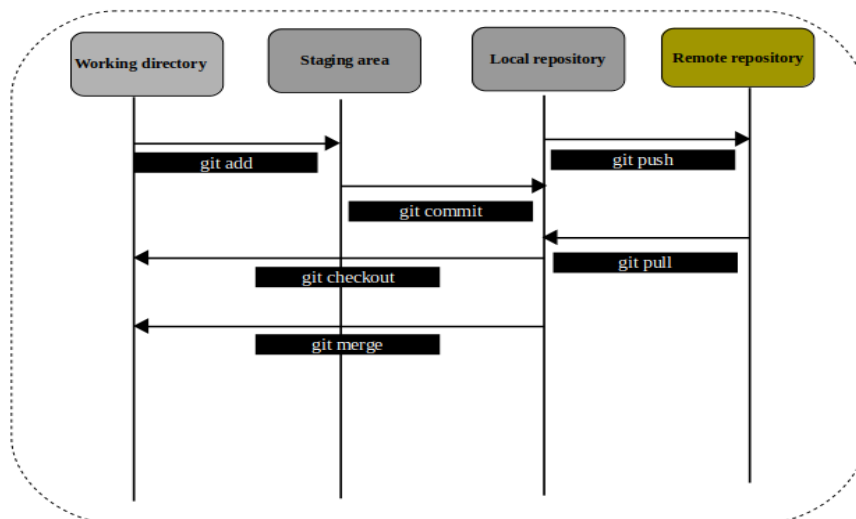
Gambar 4. *Git*

(Sumber: <https://iconscout.com/icon/git-16>)

Faktor yang membuat *Git* menjadi populer, antara lain:

1. Menjamin penghindaran penyelesaian sebagian saat menangani konten dengan *Git*, yang memastikan tidak ada kehilangan data atau perbedaan versi yang terjadi.
2. Berfokus pada hubungan *file* dan mengambil *snapshot* dari keseluruhan *file* mengatur *file* setiap kali versi dibuat.
3. Tidak dapat memodifikasi konten apa pun atau *file* tanpa pemberitahuan[9].

Git memungkinkan *developer* menyimpan, berbagi, menyebarkan, *testing*, dan berkolaborasi antar proyek berbasis *website*. *Git* dapat diimplementasikan pada OS *windows*, *Linux*, dan *MacOS* dan dapat digunakan untuk mengembangkan *software* secara bersama-sama.



Gambar 5. Cara Kerja Git

Pada Gambar 5. Cara kerja *Git*, *Git* menggunakan *remote repository* merupakan tempat dimana *Git* menyimpan *metadata* dan *object database* untuk proyek yang sedang dikerjakan. Ketika *user* ingin mengambil data dari *remote repository*, *user* dapat mengambil dengan *Git pull*. Kemudian, semua data yang berada pada proyek yang berada pada *remote repository* akan berpindah ke *local repository* atau ke dalam komputer *user*.

Selanjutnya ketika sudah pada lokal repository, pada saat *user* lain ingin menambah *branch* atau mengganti *branch* dapat menggunakan perintah *Git*

checkout. Perubahan yang dilakukan di lokal komputer akan tersimpan pada *working directory*. Semua perubahan terhadap kondisi awal akan selalu tersimpan dan dipresentasikan dalam bentuk kondisi *file* dan nama *file* tersebut.

Ketika telah selesai dimodifikasi, proyek yang tadinya sudah dilakukan perubahan atau penambahan *file*, masuk ke tahap *staging area* yang artinya tahap atau kondisi dimana *user* sudah menandai data yang diubah pada versi saat ini untuk menjadi *commit snapshot*. Selanjutnya suatu *file* sudah di dalam tahap *finish*.

Ketika suatu *file* atau proyek sudah dalam tahap *finish* untuk disimpan ke *Git* atau *remote repository*, *file* atau proyek tersebut siap untuk di *commit* untuk menandai apa saja *file* yang sudah ditambahi atau diubah.

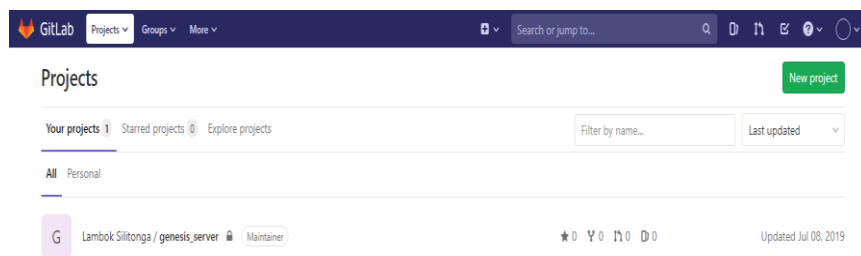
2.1.4.1 Gitlab

Gitlab merupakan sebuah vendor yang menyediakan teknologi *Git*. Manajemen proyek *software* sering terjadi permasalahan yang mengakibatkan pembangunan dan pengembangan *software* terganggu. Salah satu hal terpenting dalam sebuah repositori proyek *software* adalah sistem yang dibutuhkan untuk mengomentari, menambahkan, dan menggabungkan *source code*. *Gitlab* adalah sebuah manajer repositori *Git* berbasis *website* dengan banyak fitur dan pelacakan masalah menggunakan lisensi *open source* yang dikembangkan oleh *Gitlab Inc*[9]. *Gitlab* menyediakan *platform online* yang digunakan pengembang untuk menulis dan berbagi kode[16]. Teknologi *Gitlab* dapat dimanfaatkan melalui jaringan LAN, sehingga pengguna tetap dapat menggunakan *Gitlab* tanpa koneksi internet.



Gambar 6. Gitlab

Keunggulan unik *Gitlab* adalah dapat menginstal *software* di mana pun, tanpa membayar biaya lisensi dan dipersilakan untuk memperluas *software* secara langsung, alih-alih dibatasi membuat *add ons* melalui API[10].



Gambar 7. Tampilan *Dashboard Gitlab*

2.1.5 Paas

Paas atau *Platform as a Service* merupakan jenis *cloud computing* yang menyediakan *software* dan *tools*, seperti aplikasi web seperti *nginx* yang bisa digunakan oleh *user* untuk melakukan *develop* sebuah aplikasi dengan lebih cepat lewat internet. *User* hanya perlu melakukan instalasi aplikasi pada *resource* yang sudah disediakan sementara *user* hanya perlu mengelola aplikasi dan datanya. PaaS digunakan untuk membangun, menguji dan menyebarkan aplikasi yang sedang dalam tahap pengembangan[13]. Paas merupakan layanan berbasis *cloud*, sehingga *user* tidak ada pengaturan, pemeliharaan, *patching*, *upgrades*, dan otentikasi

sehingga *user (users)* dapat fokus untuk menciptakan *user experience* sebaik mungkin.

2.1.5.1 Dokku

Dokku (Implementasi Paas terkecil) adalah bagian dari *software open source* yang membantu *developer* membangun dan mengelola siklus hidup atau *life cycle*. *Dokku* dapat digunakan menggunakan *Docker* untuk membuat dan menjalankan aplikasi di dalam wadah yang mendukung platform *Ubuntu 14.04* atau *16.04*, *Debian 8.2* dan *CentOS 7*.

Dokku merupakan *Platform as a Service (Paas)* sehingga *dokku* dapat melakukan *deploy* aplikasi ke *Dokku* dengan melakukan konfigurasi aplikasi yang dan menyediakan platform yang digunakan untuk mengembangkan, menjalankan, dan mengelola aplikasi tanpa kompleksitas membangun dan memelihara infrastruktur yang terkait dengan pengembangan dan peluncuran aplikasi.

Dokku juga memungkinkan *developer* untuk mendorong aplikasi menggunakan *Git*, *Dokku* akan membangun aplikasi dan mengemasnya ke dalam wadah *Docker*. *Dokku* juga akan secara otomatis mengkonfigurasi *Nginx* sehingga aplikasi dapat dijangkau dengan dukungan nama *host* virtualnya. Jika *buildpack* tidak memenuhi kebutuhan, *Dokku* didukung oleh *Docker*, *developer* dapat menyesuaikan proses *build* dengan menyediakan *Dockerfile* sendiri dan *Dokku* yang akan menggunakannya.

2.1.6 Continuous Integration & Continuous Delivery (CI/CD)

CI adalah pendekatan pengembangan di mana setiap pengembang tim melakukan kode untuk integrasi setiap hari. Selama integrasi, *software* akan melalui pengujian dan pembuatan otomatis. Setelah itu, sistem memberikan umpan balik langsung tentang keadaan *software*. Berkat proses ini, tim mampu mendeteksi bug yang mengatur dan menanganinya dengan cepat.

CD adalah praktik pengembangan *software* yang memungkinkan *software* siap untuk ditempatkan ke dalam produksi kapan saja dan di mana saja. Tujuan

utama CD adalah untuk mengimplementasikan proses membangun, menguji, menyebarkan dan merilis aplikasi secara otomatis[14].

Praktik dengan Continuous Integration dan Continuous Delivery (CI/CD) telah meningkatkan efisiensi proyek[8], sehingga banyak penggiat IT mulai menerapkan konsep tersebut dalam pengerjaan proyek IT. Para pendukung teknologi mendukung kemampuan DevOps dengan mengotomatisasi tugas-tugas. Otomasi memfasilitasi pengiriman dan penyebaran berkelanjutan dengan menyediakan jalur tunggal ke produksi untuk semua perubahan pada sistem tertentu, baik untuk kode, infrastruktur dan lingkungan manajemen konfigurasi [15].

2.1.7 LAN

LAN merupakan singkatan dari *Local Area Network* yang artinya suatu jaringan komputer dimana cakupan wilayah yang jaringannya terkesan kecil atau terbatas. Dengan menggunakan jaringan LAN, *user* juga dapat berkomunikasi dengan *user* yang lain menggunakan aplikasi yang sesuai.

Fungsi utama LAN adalah untuk menghubungkan beberapa komputer di dalam jaringan sehingga proses kerja menjadi lebih mudah dan cepat.

2.2 Kajian Penelitian yang Relevan

Pada sub bab ini akan dibahas mengenai jurnal yang terkait dengan Tugas Akhir ini. Berikut jurnal yang terkait dengan Tugas Akhir ini, yaitu:

“Implementation of DevOps Architecture in the project development and deployment with help of tools” [23]

Pada jurnal ini dibahas mengenai bagaimana DevOps telah memenangkan pasar IT dengan kecepatan *release* suatu produk dengan cara mendapatkan *feedback* berulang-ulang dari pelanggan dan kembali melakukan *release* kekurangan dari suatu produk. Tim *Development* selalu ingin memberikan perubahan dalam produk sesegera mungkin sedangkan tim *operations*

menginginkan keandalan dalam stabilitas suatu produk sehingga memberikan beban di antara kedua tim bukan hanya dalam segi mental tetapi juga alat yang mereka praktikkan. Dalam penelitian ini penulis merancang bagaimana mengimplementasikan arsitektur DevOps dengan menggunakan *software* seperti *Maven*, *Git*, *Jenkins*, dan *ansible*, serta berfokus pada pengiriman produk *software* secara cepat dan mengurangi tingkat kegagalan rilis dalam membuat produk yang efisien.

2.3 Jawaban dari permasalahan & kontribusi dalam Tugas Akhir

Berdasarkan kajian penelitian yang berkaitan tentang konsep DevOps terhadap *tools* yang sering dipakai di DevOps, maka dapat disimpulkan bahwa *tools* terhadap konsep DevOps dapat memudahkan dan memungkinkan kolaborasi antar para *developer*. Berdasarkan kajian penelitian tersebut, penelitian sebelumnya menggunakan *tools Maven, Git, Jenkins*, serta *Ansible* dalam mempraktikkan konsep DevOps, perbandingan antara kajian penelitian dengan Tugas Akhir yang dibuat oleh penulis adalah penulis menggunakan teknologi *Docker container* yang untuk dapat membuat pengerjaan tugas dan proyek mahasiswa/i Institut Teknologi Del, dibantu dengan *tools Git*, dan *Dokku*, sehingga penulis akan mempraktikkan konsep *lifecycle* DevOps dalam pengerjaan tugas dan proyek yang saling terintegrasi dengan memanfaatkan jaringan LAN Institut Teknologi Del dengan mengkombinasikan beberapa *tools* baik *Docker, Git*, dan PaaS dengan konsep DevOps sehingga mahasiswa/i Institut Teknologi Del dapat mengenal, mempraktikkan, dan menerapkan teknologi & konsep DevOps yang berkembang dalam industri informasi dan teknologi.

BAB 3 ANALISIS DAN PERANCANGAN SISTEM

Pada bab ini akan diuraikan bagaimana analisis yang terjadi pada sistem. Berdasarkan hasil analisis, dapat dirancang sistem uji yang sesuai dengan pandangan dari studi literatur. Analisis dan perancangan sistem dalam Tugas Akhir ini yaitu sebagai berikut:

3.1 Analisis

Pada sub-bab analisis dibahas mengenai analisis terhadap masalah, analisis pemecahan masalah, analisis kebutuhan sistem serta skenario uji yang digunakan untuk menentukan solusi pemecahan terhadap masalah yang terjadi.

3.1.1 Analisis Masalah

Teknologi DevOps memungkinkan pengembangan *software* dan metode pengiriman aplikasi yang mengambil pendekatan kolaboratif dan terpadu antara bagian pengembangan aplikasi (Dev) dan bagian operasional (Ops). Sehingga terjadinya otomatisasi proses anatar *software development* dan *development team* agar dapat melakukan proses *build*, *test*, dan *release software* menjadi cepat dan handal. Ketika praktik DevOps berjalan dengan baik maka akan menghasilkan produk yang stabil dan meningkatkan nilai dari produk itu sendiri.

Salah satu contoh kasus adalah masalah pengerjaan proyek mahasiswa/i di Institut Teknologi Del. Institut Teknologi Del merupakan perguruan tinggi swasta yang terletak di Laguboti, Toba. Mahasiswa di Institut Teknologi Del memiliki tugas perkuliahan maupun proyek seperti aplikasi web dengan tujuan agar mahasiswa/i dapat menerapkannya di dalam dunia profesional. Namun selama ini, mahasiswa/i belum mempraktikkan konsep *software development* yang dipakai dalam industri informasi dan teknologi saat ini, seperti ketika mahasiswa/i masih melakukan pengerjaan *source code* tugas dan proyek web di laptop masing masing, lalu ketika hendak menyatukan codingan tugas dan proyek terjadi kesalahan akibat OS yang berbeda maupun ketidak sesuaian *source code*, dan ketika melakukan *update source code* aplikasi harus melakukan tindakan manual sehingga tidak

terjadinya kolaborasi dan kerjasama tim yang maksimal yang mengakibatkan kerugian dalam segi efisiensi dan efektifitas. Selain itu ketika hendak melakukan *deploy* biasanya mahasiswa/i menggunakan jaringan internet yang memiliki bandwidth yang terbatas, sehingga mengakibatkan proses *deploy* dan tergolong lama karena banyaknya mahasiswa/i yang menggunakan koneksi internet yang sama didalam lingkungan kampus.

3.1.2 Analisis Pemecahan Masalah

Pemecahan masalah *software development* yang dilakukan oleh mahasiswa/i Institut Teknologi Del saat ini yaitu penulis memanfaatkan Teknologi praktik DevOps menggunakan *Docker*, *Git*, dan PaaS (Dokku) yang memiliki keunggulan CI/CD dan bersifat *life cycle*, agar pengerjaan tugas dan proyek mahasiswa/i dapat dilakukan secara otomatis, efisien, dan efektif. Mahasiswa/i yang mengerjakan tugas dan proyek web lebih mudah dalam melakukan pengerjaannya, mahasiswa/i tidak memerlukan cara lama dan manual.

Git akan berperan dalam manajemen *source code* tugas dan proyek mahasiswa/i, sehingga dapat disimpan pada *Gitlab*, yaitu *platform* yang digunakan pengembang untuk menulis dan berbagi kode[16] maka tidak terjadi ketidaksesuaian *source code* akibat pengerjaan tugas dan proyek *source code website* secara bersama di laptop masing masing.

Docker yang dapat memudahkan proses penyebaran aplikasi, meningkatkan produktivitas pengembang, mengurangi beban server, dan mengurangi jumlah penyimpanan[17]. Lalu menggunakan PaaS yaitu *Dokku* yang digunakan untuk proses *deploy* aplikasi yang dapat digunakan dalam jaringan LAN, sehingga mahasiswa/i tidak memerlukan koneksi internet yang dibatasi *bandwidth*. Seluruh tools tersebut akan saling berintegrasi dengan konsep CI/CD yang bersifat *life cycle*. Selain itu juga dapat membangun konsep kerjasama dalam tim dan secara bersamaan mempercepat pengerjaan suatu tugas dan proyek mahasiswa/i serta

dapat menggunakan dan menerapkan praktik yang diterapkan sebagai budaya kerja dalam dunia profesional industri teknologi dan informasi saat ini.

3.1.3 Analisis Kebutuhan Sistem

Pada sub-bab ini dijelaskan kebutuhan yang diperlukan selama proses pengerjaan Tugas Akhir. Kebutuhan sistem tersebut mencakup kebutuhan *hardware* dan *software*.

1. Kebutuhan *Hardware* (*Hardware*)

Hardware yang dibutuhkan dalam merancang dan membangun sistem dapat dilihat pada Tabel 3.

Tabel 3 Kebutuhan *Hardware*

Nama Perangkat	Spesifikasi	Keterangan
Laptop	Model Processor: Intel Core i5 Processor 2.5 GHz Memori Standar: 4GB DDR4	Wadah dalam menjalankan tools Docker, Gitlab, dan Paas.

2. Kebutuhan *Software*

Software yang dibutuhkan dalam merancang dan membangun sistem dapat dilihat pada Tabel 4.

Tabel 4 Kebutuhan *Software*

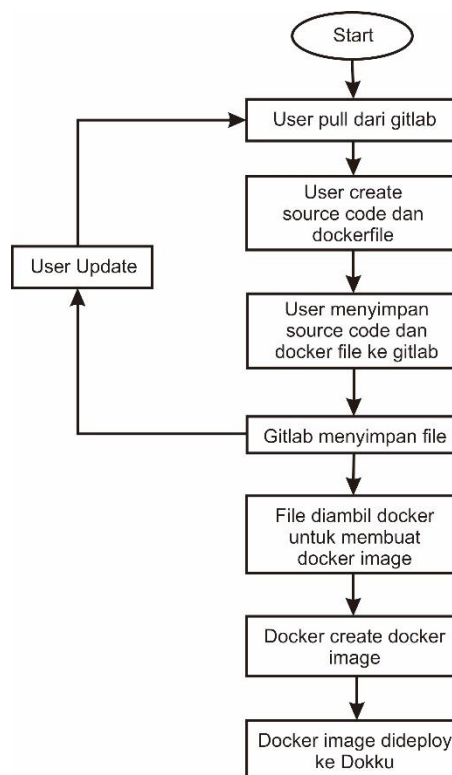
Jenis	Versi	Keterangan
Docker	2.1.0.5	Tools DevOps yang digunakan
Gitlab	Community Edition	Tools DevOps yang digunakan
Paas	Dokku	Platform Deploy aplikasi
ISO Ubuntu	18.04	OS Docker server
Browser	Google Chrome	Untuk menampilkan antarmuka atau dashboard Gitlab.

3.2 Perancangan Pembangunan Sistem

Perancangan dalam pembangunan sistem merupakan tahap awal dalam pengerjaan Tugas Akhir yang melibatkan beberapa *Hardware* dan *Software*. Perancangan digunakan sebagai acuan dalam mengimplementasikan sistem pada Tugas Akhir.

3.3 Flowchart Sistem

Perancangan *flowchart* sistem bertujuan untuk dapat melihat gambaran dari alur kerja pengerjaan proyek sesuai pola DevOps yang berulang sehingga membentuk *life cycle*. Dalam *flowchart* ini digambarkan bagaimana integrasi cara kerja dari *tools Docker*, *Gitlab*, dan *Dokku* dalam menjalankan alur kerja DevOps untuk mengerjakan proyek dengan menggunakan jaringan LAN Institut Teknologi Del.



Gambar 8. Alur Pengerjaan proyek dengan konsep *DevOps*

Pada Gambar 8, Dari *flowchart* diatas dapat dilihat bahwa alur kerja dari DevOps berjalan secara *life cycle*, dimulai dari *user* melakukan *pull* dari *Gitlab*,

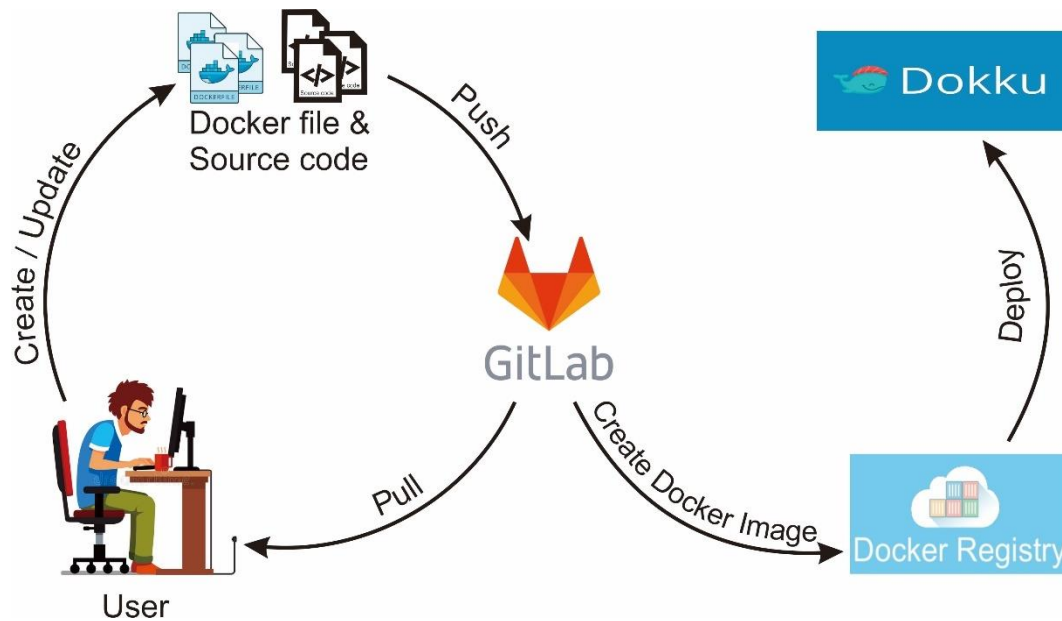
selanjutnya menyimpan *source code* tersebut bersamaan dengan *Dockerfile* ke *Gitlab repository* yang diimplementasikan di jaringan LAN Institut Teknologi Del. Jika proyek tersebut telah selesai, *Docker* akan menggabungkan *code* proyek dengan *Dockerfile* menjadi *Docker image*. Kemudian *Docker image* dari proyek tersebut, akan dijalankan di atas *production* Paas *Dokku*, ketika proyek tersebut ingin diperbaharui, maka posisi kembali ke tahap awal yaitu memperbaharui *source code* yang tersedia, mengirim kembali ke repository *Gitlab* dan kembali menjalankan proyek tersebut diatas Paas. Siklus yang berputar tersebut dinamakan *life cycle*.

3.4 Perancangan Sistem

Pada Sub-bab berikut akan dijelaskan mengenai perancangan sistem secara keseluruhan yang dilakukan pada pengerjaan Tugas Akhir baik secara umum maupun lebih spesifik. Melalui perancangan ini dapat membantu penulis maupun pembaca untuk lebih memahami dan mengerti mengenai sistem yang dibangun.

3.4.1 Gambaran Umum Sistem

Gambaran umum sistem bertujuan untuk melihat bagaimana sistem dibangun secara umum sehingga memudahkan pembaca dalam memahami sistem. Dengan perancangan ini maka penulis dapat mempersiapkan *environment* yang dibutuhkan sehingga dapat mempermudah dalam pengimplementasian Tugas Akhir. Melalui perancangan ini juga diharapkan dapat mempermudah penulis dan pembaca dalam memahami alur kerja sistem yang dibangun. Secara umum, sistem ini bekerja dengan alur kerja sesuai dengan gambar berikut:



Gambar 9. Gambaran Umum Hubungan Antar Sistem

Pada Gambar 9, terdapat 3 komponen utama yang bekerja pada sistem yaitu *Docker* sebagai inti dari komponen yang diasumsikan sebagai penampung dari hasil jadi suatu proyek, *Gitlab* sebagai tempat penyimpanan *source code* dari sebuah proyek, dan Paas sebagai *platform* dari berjalannya hasil proyek, setiap komponen akan dihubungkan menggunakan jaringan LAN Institut Teknologi Del. *User* dapat menggunakan berkolaborasi antar tim untuk menjalankan sistem ataupun mempraktikkan *life cycle* DevOps yang akan dirancang dalam Tugas Akhir.

Langkah-langkahnya antara lain:

1. Saat ingin memulai pengerjaan proyek, *developer* pertama sekali melakukan *pull source code* dari *Gitlab*.
2. Saat selesai melakukan *update source code* aplikasi, baik menambah ataupun mengubah struktur *source code* aplikasi proyek, *developer* melakukan *push source code* ke dalam *Gitlab*.
3. Saat melakukan *update*, *Gitlab* akan melakukan *build* lalu pengujian pada cabang *Gitlab*.

4. Jika berhasil maka CI/CD akan melakukan *build docker image*, agar dapat melakukan *deploy* ke dalam *Dokku*.
5. Kemudian *Gitlab* akan melakukan *deployment* ke *Dokku*. *Gitlab* akan melakukan *pull image* dahulu dari *private docker registry*, sehingga dapat melakukan *push* ke *Dokku*.
6. Ketika keseluruhan proses sudah bekerja dengan baik, maka *Dokku* akan menjalankan *image* tersebut, sehingga akan menampilkan hasil pada *end-user*.

Sebelum dapat menggunakan *Docker* sebagai *platform* untuk menjalankan proyek, *Docker image* harus di konfigurasi dahulu dengan *Dockerfile*. *Dockerfile* yang akan menjadi *Docker image*, dan ketika *Docker image* dijalankan akan menjadi *Docker container* dengan bahasa pemrograman yang digunakan untuk membuat *Dockerfile* adalah dengan *yaml language*.

Dalam mengkonfigurasi atau membuat sebuah *Dockerfile*, terlebih dahulu dibutuhkan sebuah *Docker image* dasar yang dimana *Docker image* ini menjadi dasar tempat berjalannya *code* dari aplikasi proyek. *Image* tersebut merupakan sebuah *OS* yang dapat diperoleh melalui *Dockerhub*. *Dockerhub* merupakan tempat penyimpanan dari banyak *Docker image* yang bersifat *online*. Masalahnya, *Docker image* memiliki ukuran yang tergolong besar, sehingga jika memanfaatkan jaringan *internet* yang disediakan IT Del, terkesan tidak efektif dan boros waktu. Sehingga, ide yang ada adalah bagaimana menciptakan *Dockerhub* untuk *private*, yang disediakan khusus untuk IT Del, yaitu menggunakan *Docker registry*.

Docker registry merupakan bentuk *mirroring* dari *Dockerhub*, dengan memanfaatkan jaringan LAN IT Del, maka mengunduh *Docker image* akan menjadi lebih cepat, dibanding mengunduh dari *Dockerhub* yang berada pada jaringan *internet*.

Dari gambar 9, setelah mahasiswa/i selesai membuat *code* proyek dan *Dockerfile*, maka *code* tersebut disimpan ke *Gitlab*, kemudian, *Docker* akan menggabungkan *code* proyek dengan *Dockerfile*, dengan *Docker image* dasar yang

berasal dari *private Docker registry*, setelah menjadi *Docker image*, *Dokku* akan *deploy* hasil *Docker image* agar dapat dilihat oleh *user*, *event push*(menyimpan *code*) ke *Gitlab* dapat dilakukan bersamaan dengan *deploy Docker image* dengan konsep CI/CD, sehingga *user* tidak perlu melakukan *publish* hasil proyek dengan cara manual, *Dokku* akan mengatur segala hal yang diperlukan seperti jaringan, port, bagaimana menjalankan hasil *Docker image* proyek dan lain-lain, dengan kata lain, *user* hanya perlu menyimpan *code* tersebut ke *Gitlab*, dan dengan CI/CD, *Dokku* akan menyelesaikan permasalahan bagaimana proyek akan berjalan.

3.5 Skenario Pengujian

Pada pengerjaan tugas akhir ini akan dilakukan beberapa simulasi pengujian. Pada skenario uji ini terdapat beberapa komponen penting seperti *Docker registry*, LAN IT Del dan jaringan internet IT Del. Nantinya akan dilakukan pengukuran perbandingan waktu, berapa lama waktu yang digunakan dalam mengunduh sebuah *Docker image* untuk menunjukkan perbandingan efektifitas dalam hal kecepatan jaringan serta bagaimana proses *lifecycle* pada pengerjaan proyek *website* dengan cara *update code* ataupun *database website*, sehingga terciptanya siklus *lifecycle*.

3.5.1 Pengujian Mengunduh *Docker Image* LAN dan Internet IT Del

Salah satu tujuan dari pengerjaan TA ini adalah untuk meningkatkan kinerja dari segi waktu mahasiswa/i dengan meningkatkan kecepatan jaringan untuk mengerjakan tugas dan proyek. Kami menggunakan jaringan LAN IT Del untuk meningkatkan kecepatan jaringan, dan untuk membuktikan bahwa jaringan LAN IT Del lebih cepat dibanding jaringan *internet* IT Del, penulis membandingkan kecepatan mengunduh *Docker image* dengan menggunakan jaringan internet IT Del dengan jaringan LAN IT Del. Penulis mengunduh salah satu *Docker image* yang berasal dari *Dockerhub*, tempat penyimpanan *online Docker registry*, menggunakan jaringan internet IT Del, kemudian mengunduh *Docker image*, yang berasal dari *private Docker registry* yang terpasang pada jaringan LAN IT Del. Perbandingan data yang digunakan dalam satuan waktu (jam, detik, menit). *Docker* menyediakan perintah *time* untuk mengukur berapa lama waktu yang digunakan

untuk mengunduh *Docker image*. Hasil dari perbandingan jaringan akan dibandingkan, untuk melihat jaringan mana yang lebih efektif dalam mendukung konsep DevOps yang terkesan cepat dalam pengerjaan proyek mahasiswa/i IT Del.

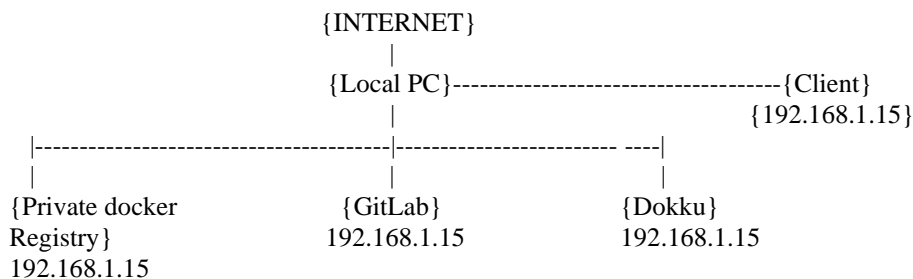
3.5.2 Pengujian *Update* dengan *Website Static* dan *Dinamic*

Pada pengujian, penulis membuat *website static* dengan HTML, *website semi static* dengan *arraylist*, serta *website dynamic* menggunakan PHP dan *database MySQL*, penulis melakukan *update source code website* untuk memperoleh *life cycle* menggunakan konsep DevOps. Sehingga membuktikan bahwa Tugas Akhir dapat membentuk siklus DevOps dengan memanfaatkan tools Docker, Git, & PaaS.

BAB 4 IMPLEMENTASI DAN PENGUJIAN

Pada bab ini dijelaskan mengenai implementasi terhadap sistem yang telah dilakukan. Proses implementasi yang dilakukan berupa proses instalasi, konfigurasi serta pengujian yang dilakukan pada sistem berdasarkan skenario uji yang dijelaskan pada bab sebelumnya.

4.1 Topologi Sistem



Pada sistem yang terpasang, penulis menerapkan *Private Docker Registry*, *GitLab*, dan *Dokku* pada sebuah laptop, sehingga IP yang digunakan oleh *private docker registry*, *GitLab*, dan *Dokku* adalah sama, yaitu 192.168.1.15, apabila *client* ingin mengakses salah satu sistem, seperti *GitLab*, maka *user* dapat mengetikkan alamat IP 192.168.1.15, selama *client* berada pada satu jaringan yang sama. Sebagai *client*, yang menggunakan sistem, karena hanya memiliki sebuah laptop, lokal PC penulis gunakan sebagai *client* itu sendiri, sehingga *client* juga memiliki IP yaitu 192.168.1.15. Untuk spesifikasi, laptop memiliki memory 4GB, CPU *intel core i5*, dan *processor Core i5*.

4.2 Implementasi *Environment (Tools)* dan Alur Pengerjaan Proyek Menggunakan *Gitlab*, *Docker*, dan Paas *Dokku* dengan Konsep DevOps

Pada pengerjaan Tugas Akhir ini dilakukan proses instalasi dan konfigurasi pada *tools Gitlab*, *Docker*, dan Paas. Untuk dapat melakukan instalasi pada *tools*, dapat dilakukan dengan mengikuti tahap demi tahap sesuai ketentuan yang ada agar *software* dapat berfungsi dengan baik nantinya.

4.2.1 Konfigurasi *Static IP Adress* pada Ubuntu 18.04

Buka *file* konfigurasi dengan *text editor nano*.

```
$ sudo nano /etc/netplan/50-cloud-init.yaml
```

Contoh *default file* konfigurasi.

```
network:
  ethernets:
    enp0s3:
      addresses: []
      dhcp4: true
      optional: true
  version: 2
```

Kemudian, *set* IP, akan aman jika mengikuti IP yang sudah ada.

```
network:
  ethernets:
    enp0s3:
      addresses: [192.168.1.15/24]
      gateway4: 192.168.1.1
      nameservers:
        addresses: [8.8.8.8,8.8.4.4]
      dhcp4: no
  version: 2
```

Keterangan:

- 1 `enp0s3`: *network interface*.
- 2 `addresses`: IP *address* yang diberikan dengan subnet /24 (255.255.255.0).
- 3 `nameservers – addresses`: IP *address* untuk dns *resolver*.

4.2.2 Docker

Docker adalah aplikasi yang menyederhanakan proses mengelola proses aplikasi dalam di dalam *container*. Container memungkinkan menjalankan aplikasi dalam proses yang terisolasi sumber daya.

4.2.2.1 Instalasi *Docker* pada *Ubuntu 18.04*

1. Perbarui paket yang ada pada sistem

```
# sudo apt update
```

2. Selanjutnya, instal beberapa paket prasyarat yang memungkinkan apt menggunakan paket melalui HTTPS:

```
# sudo apt install apt-transport-https ca-certificates curl software-properties-common
```

3. Kemudian tambahkan kunci GPG untuk repositori *Docker* resmi ke sistem :

```
# curl -fsSL https://download.Docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

4. Tambahkan repositori *Docker* ke sumber APT:

```
#sudo add-apt-repository "deb [arch=amd64] https://download.Docker.com/linux/ubuntu bionic stable"
```

5. Selanjutnya, perbarui database paket dengan paket *Docker* dari repo yang baru ditambahkan:

```
# sudo apt update
```

6. Dan terakhir install *Docker* dengan perintah :

```
# sudo apt install Docker-ce
```

4.2.2.2 *Dockerfile*

Dockerfile menggabungkan semua kebutuhan *software* yang dibutuhkan dalam membangun proyek dengan *source code* proyek itu sendiri.

Berikut merupakan salah satu isi dari *Dockerfile* dan contoh *source code* dari sebuah *website*.

```
#Dockerfile
FROM nginx:alpine

COPY default.conf /etc/nginx/conf.d/default.conf
COPY index.html /usr/share/nginx/html/index.html
```

EXPOSE 80

RUN mkdir /myapp

```
#default.conf

server {

    listen    80;

    server_name localhost;

    location / {

        root  /usr/share/nginx/html;

        index index.html;

    }

}
```

```
#index.html

<!DOCTYPE html>

<html lang="en">

    <head>

        <meta charset="UTF-8" />

        <meta name="viewport" content="width=device-width, initial-scale=1.0" />

        <meta http-equiv="X-UA-Compatible" content="ie=edge" />

        <title>Contoh proyek</title>

    </head>

    <body>

        <h1>Hey, it works!</h1>

        <h1>Kelompokk 5 TA2 here.</h1>

    </body>

</html>
```

```
</body>
</html>
```

4.2.2.3 Install dan konfigurasi *Docker Registry*

Setelah melakukan instalasi *service Docker* pada *server*, selanjutnya dilakukan konfigurasi *Docker registry* yang bersifat *private* dengan memanfaatkan jaringan LAN. Karena *Docker Private Registry* berada dalam jaringan LAN, maka akan menghemat *bandwidth* secara *signifikan*. Alamat IP yang digunakan adalah 192.168.1.15.

1. Untuk dapat menkonfigurasi *Private Docker Registry*, sebelumnya harus menggunakan jaringan Internet untuk mengunduh *image Docker registry* yang tersedia di *Docker hub*

```
# sudo docker pull registry:2
```

2. Kemudian jalankan *image* tersebut dengan CLI.

```
# sudo docker run -d -p 5000:5000 --restart=always --name registry registry:2
```

Untuk selanjutnya, maka *Docker container* tersebut akan berjalan, *developer* dapat dengan cepat memanfaatkan *bandwith* jaringan LAN dalam penyimpanan *Docker image* yang bersifat besar ke *private Docker registry*. Tetapi dengan syarat, *image* yang dibangun memiliki alamat *private Docker registry* di depan nama *image*.

A screenshot of a terminal window showing the status of a Docker container named 'registry'. The container ID is 'ed90648ee8d7'. It is running the 'registry:2' image. The entrypoint is '/entrypoint.sh /etc...'. It was created 2 days ago and has been up for 9 seconds. The ports are mapped as 0.0.0.0:5000->5000/tcp. The container is running on the 'registry' host.

```
ed90648ee8d7 registry:2 "/entrypoint.sh /etc..." 2 days ago Up 9 seconds 0.0.0.0:5000->5000/tcp registry
```

Gambar 10. *Docker container* dari *docker registry* yang berjalan

3. *Docker tag* adalah salah satu contoh dalam fungsi *Docker* untuk mengganti nama *image Docker*. Beri *tag* baru pada *image* tersebut sesuai dengan lokasi *Docker Registry*, dalam hal ini alamat *Docker registry* 192.168.1.15 dengan perintah:

```
# sudo docker tag $docker image 192.168.1.15:5000/$Docker image
```

4. Ketika ingin melakukan penyimpanan *Docker image*, maka *developer* perlu melakukan *push* sesuai nama *image* yang telah di *tag*.

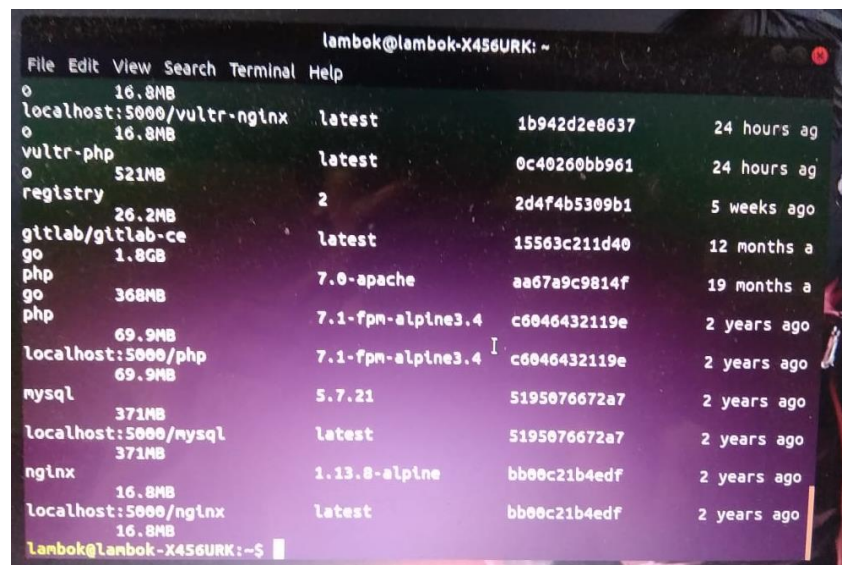
```
# sudo docker push 192.168.1.15:5000/$Docker image
```

5. Bila berhasil, maka *Docker image* tersebut telah disimpan di dalam *Docker Registry* LAN. Agar *user* dapat mengunduh *Docker image* yang telah di *push* ke jaringan LAN *private Docker registry* maka tambahkan sebuah *file atauetcatauDockerataudaemon.json* pada komputer client.

```
# {  
  
  "insecure-registries" : ["192.168.1.15:5000"]  
  
}
```

6. Kemudian *restart Docker*

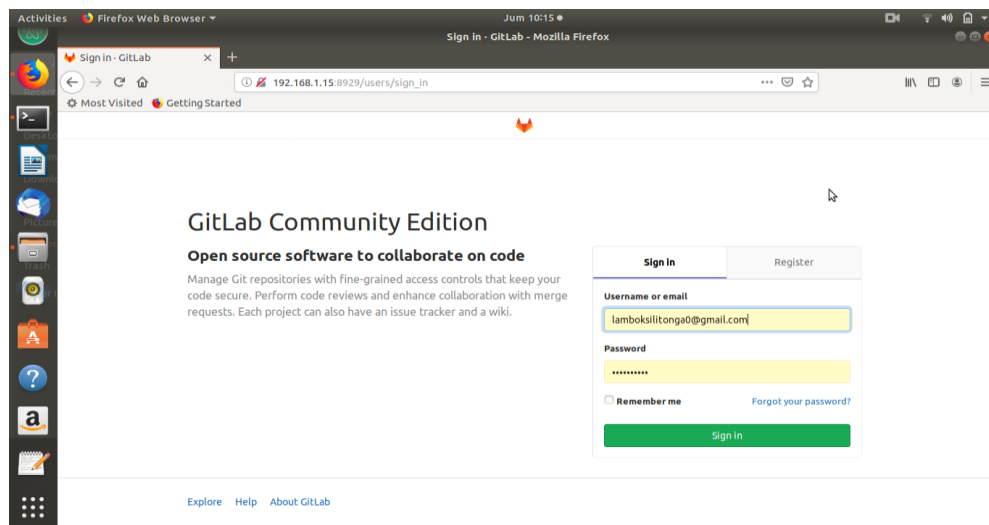
```
# systemctl restart docker.service
```



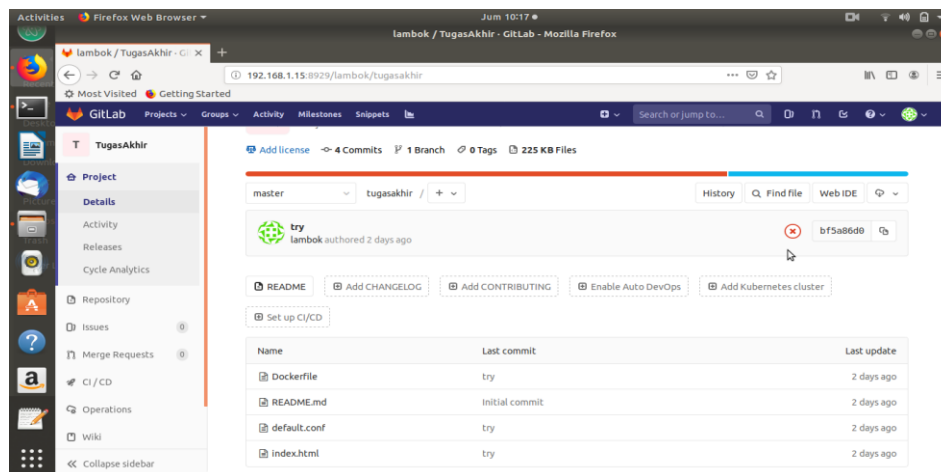
Gambar 11. List daftar *docker image* yang berhasil disimpan di *private docker registry*

4.2.3 Gitlab

Gitlab merupakan manajemen *repository* yang membantu mengelola siklus pengembangan *software*. Dalam *userannya* Gitlab yang disediakan, dikarenakan tidak memiliki *domain*, Gitlab yang dibangun sama saja *userannya*, tetapi yang perlu diperhatikan adalah karena tidak memiliki *domain*, Gitlab yang dibangun hanya dapat diakses melalui ip, pada gambar dapat dilihat bahwa ip server 192.168.1.15:8929.



Gambar 12. Tampilan login Gitlab



Gambar 13. Tampilan penyimpanan proyek

4.2.3.1 Konfigurasi dan Instalasi *Gitlab* pada Server *Ubuntu 18.04* Menggunakan *Docker*

Pada implementasi *Git* yang digunakan adalah *Gitlab* yang menggunakan alamat IP *static* 192.168.1.15.

1. Sebelum dapat menggunakan *Gitlab*, terlebih dahulu melakukan instalasi *Docker* pada server, pada kali ini, server yang digunakan adalah *Ubuntu 18.04*. Install beberapa paket yang digunakan untuk memungkinkan untuk menggunakan paket melalui *HTTPS*.

```
# apt install apt-transport-https ca-certificates curl software-properties-common
```

2. Setelah berhasil, kemudian menambahkan *GPG key* untuk repositori resmi

```
# curl -fsSL https://download.Docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

3. Tambahkan repositori *Docker* ke server *ubuntu*.

```
# sudo add-apt-repository "deb [arch=amd64]
https://download.Docker.com/linux/ubuntu bionic stable"
```

4. Update repositori server, dan lanjutkan dengan menginstal *Docker*.

```
# sudo apt install Docker-ce
```

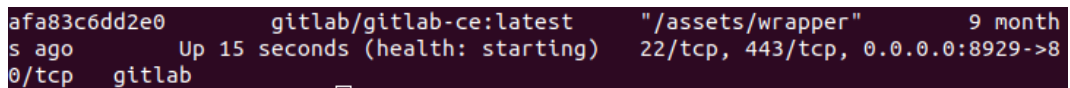
5. Setelah melakukan instalasi *Docker*, selanjutnya dapat dilakukan konfigurasi *Gitlab* dengan mengunduh *image Gitlab* yang berada pada *official* repositori *Docker*, *Gitlab* akan dibangun dan dikonfigurasi dengan menggunakan *Docker*. Pada tahap ini akan memerlukan waktu tergantung kecepatan internet dan besar ukuran *Docker image Gitlab*, pada ukuran *image Docker Gitlab* sendiri berukuran sekitar 2 GB.

```
# Docker pull store/Gitlab/Gitlab-ce:11.10.4-ce.0
```

6. Setelah *Docker image* terunduh, dapat dijalankan dengan menggunakan CLI pada server, sebagai media penyimpanan, sebaiknya *Docker image* dijalankan dengan *persistent*, atau selalu tersedia. Agar ketika suatu saat

terjadi suatu masalah dengan *Docker* seperti matinya *server* atau hal lain, maka data yang tersimpan tidak hilang.

```
# sudo Docker run --detach \  
  
--publish 8929:8929 --publish 2289:22 \  
  
--name Gitlab \  
  
--restart always \  
  
--volume $GITLAB_HOME/Gitlab/config:/etc/Gitlab \  
  
--volume $GITLAB_HOME/Gitlab/logs:/var/log/Gitlab \  
  
--volume $GITLAB_HOME/Gitlab/data:/var/opt/Gitlab \  
  
Gitlab/Gitlab-ce:latest
```



```
afa83c6dd2e0    gitlab/gitlab-ce:latest    "/assets/wrapper"    9 month  
s ago         Up 15 seconds (health: starting)    22/tcp, 443/tcp, 0.0.0.0:8929->8  
0/tcp        gitlab
```

Gambar 14. *Docker container Gitlab* yang berjalan

Pada *command* tersebut, *option detach* artinya *container Docker* atau *Docker image* berjalan pada *background*, *option publish* agar melakukan *expose port* selain 80 & 443. Pada *container* yang dijalankan menggunakan *port 8929* sebagai *website interface*. *Option name* digunakan sebagai nama dari *container* atau *Docker image* yang sedang berjalan. *Option volume* menjaga agar data pada *Gitlab* selalu tersedia.

4.2.3.2 Instalasi *Git* pada *Client*

Pada *Git* di *Windows*, Pergi ke alamat <http://msysGit.Github.io/> dan unduh versi *installer* terbaru untuk windows dan jalankan *installer* agar dapat digunakan. Pada *Git* di Linux dengan OS Ubuntu, lakukan perintah

```
# sudo apt-get install git
```

4.2.4 Konfigurasi dan Instalasi Paas *Dokku*

Untuk melakukan *deploy* hasil proyek yang sudah dibangun didasarkan oleh *Docker*, maka membutuhkan Paas. Pada Tugas Akhir, Paas yang digunakan adalah *Dokku*. Biasanya Paas hanya dapat melakukan *publish* proyek berbasis *code*, tetapi agar dapat menghasilkan siklus *life cycle* dari DevOps pada pengerjaan proyek, maka diperlukan Paas yang dapat bukan hanya dalam bentuk *source code*, tapi juga dalam bentuk *Docker container*. Dengan *Dokku*, *developer* dapat memilih untuk mengembangkan aplikasi dengan *Docker*.

1. Mengunduh repositori *Dokku*

```
# wget -nv -O - https://packagecloud.io/Dokku/Dokku/gpgkey | apt-key add -  
# export SOURCE="https://packagecloud.io/Dokku/Dokku/ubuntu/"  
# export OS_ID="$(lsb_release -cs 2>/dev/null || echo "bionic")"  
# echo "xenial bionic focal" | grep -q "$OS_ID" || OS_ID="bionic"  
# echo "deb $SOURCE $OS_ID main" | tee /etc/apt/sources.list.d/Dokku.list  
# apt-get update  
# apt-get install Dokku  
# Dokku plugin:install-dependencies --core
```

Kemudian pergi ke alamat *server*, dan tampilan *installer* akan terlihat seperti gambar berikut:

Dokku Setup v0.8.0

ADMIN ACCESS

Public Key

```
ssh-rsa
AAAAB3NzaC1yc2EAAAABJQAAAQEAst1FRkYP5M87P1uqsdBHPNwx7usrLt5RDMK0Xe9
EUBrD2iaFXm0HcXj+aw1kKAQ3ANdk24QVFe8KyCOFlunI2ZjhPJS0LEr4fHypx0Uo/ETfHMK
VYyntZmpLuopBT/2smc5BLHiG8ZN8if1uxBLjEeXcXZy+OfPOzn6No/cOpiceCbsxg+EvaZ31
6ORxbJYCUXA5BfVlyuNDnjhozO8w/o6xeZWz++cNWXNJCuQ3nJWJAxAQhidT1A1ye6ikN
miltqgnhlopO42qsvFkii5skczl370Ar0AnAbyVvW74/ZBhziW5yQ9aZo1LEp4RJZkekjdIrV8jDnT
6nlsfV2zQ== dokku-rsa-key
```

HOSTNAME CONFIGURATION

Hostname

dokku.example.com

☒ Use virtualhost naming for apps

Your app URLs will look like:

http://<app-name>.dokku.example.com

Finish Setup

Gambar 15. Instalasi Dokku

2. Pada *Public Key*, dapat disesuaikan dengan *rsa key*, *rsa key* dapat dibuat dengan membuat pada server yang menginstall *Dokku*.

```
# ssh-keygen -t rsa
```

3. Langkah selanjutnya, simpan *Key* dan *Passphrase*, akan terlihat tampilan berikut

Enter file in which to save the key (/home/lambok/.ssh/id_rsa):

Pertanyaan tersebut untuk menentukan lokasi menyimpan *key*, dan lokasi *key* akan tersimpan di *directory* *atau home* *atau lambok* *atau .ssh* *atau id_rsa*. Proses pembuatan *rsa key* akan terlihat seperti berikut

```
ssh-keygen -t rsa
```

Generating public/private rsa key pair.

Enter file in which to save the key (/home/lambok/.ssh/id_rsa):

Enter passphrase (empty for no passphrase):

Enter same passphrase again:

Your identification has been saved in /home/lambok/.ssh/sshatauid_rsa.

Your public key has been saved in /home/lambok/.ssh/id_rsa.pub.

The key fingerprint is:

4a:dd:0a:c6:35:4e:3f:ed:27:38:8c:74:44:4d:93:67 lambok@a

The key's randomart image is:

+--[RSA 2048]-----+

| .oo. |

| . o.Z |

| + . o o |

| . = = . = |

| = S = . S |

| o + = + |

| . o + o . |

| . o |

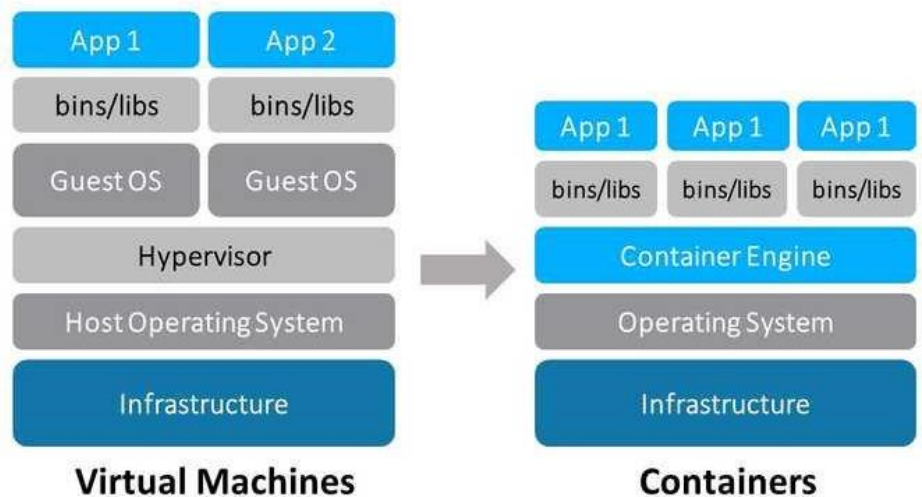
| |

+-----+

Selanjutnya *user* dapat melakukan *copy source code* dari *file* ssh-rsa ke tabel *Dokku setup installer*. Pada *tab hostname* konfigurasi, jika dicentang maka proyek memerlukan *domain*, dan apabila tidak dicentang maka sebagai pengganti *domain* yang dapat diakses adalah ip dan *domain* tidak tersedia. Jika telah selesai dapat menekan tombol *finish set up*, pada konfigurasi Tugas Akhir ini penulis tidak mengcentangnya dikarenakan tidak tersedianya *hostname*.

4.2.5 Panduan Integrasi Docker, Gitlab, dan Dokku dengan Konsep Lifecycle DevOps dengan Menggunakan CI/CD Docker Deployment dengan Dokku

Docker adalah bagian dari *software* untuk menyederhanakan pembuatan, penyebaran, dan pelaksanaan aplikasi dengan menggunakan *container*. *Container* memungkinkan untuk mengemas aplikasi di luar lingkungan sistem operasi.



Gambar 16. Perbedaan Arsitektur VM dan *docker*

Untuk dapat melakukan otomatisasi keseluruhan service tools yang digunakan dalam praktik DevOps, maka digunakan bantuan yang dikenal dengan konsep CI/CD yang dimiliki oleh *Gitlab* sehingga menghasilkan konsep *life cycle* DevOps. Untuk melakukan *deploy* hasil proyek yang telah menjadi *Docker image*, maka *developer* perlu mengakses *server* secara langsung, tetapi hal itu terkesan menyusahkan. Dengan CI/CD, *developer* dapat melakukan *push git*, maka secara otomatis disaat yang bersamaan, hasil proyek tersebut ter-*deploy* ke *Dokku*. Maka untuk melaksanakannya, mahasiswa/i perlu membuat konfigurasi:

```
lambok$ ssh-keygen -P " " -C 'GitLab/Dokku Integration' -f gitlab
Generating public/private rsa key pair.
Your identification has been saved in gitlab.
Your public key has been saved in gitlab.pub.
```

Salin *public* key ke *host Dokku*, dan *Dokku* menerimanya: -

```
dokku-host$ sudo dokku ssh-keys:add "GitLab/Dokku Integration" /tmp/gitlab.pub
SHA256:<...SHA256 hash...>
```

Selanjutnya, buka pengaturan CI / CD di *GitLab* repositori (misalnya, http://192.168.1.15/ta/settings/ci_cd), dan buka bagian Environment variables. Buat entri baru bernama *SSH_PRIVATE_KEY*, dan *paste* nilai pada konten *file gitlab*: -

Environment variables ?

Collapse

Environment variables are applied to environments via the runner. They can be protected by only exposing them to protected branches or tags. You can use environment variables for passwords, secret keys, or whatever you want. You may also add variables that are made available to the running application by prepending the variable key with `K8S_SECRET_`. [More information](#)

SSH_PRIVATE_KEY	*****	Protected <input checked="" type="checkbox"/>
Input variable key	Input variable value	Protected <input checked="" type="checkbox"/>

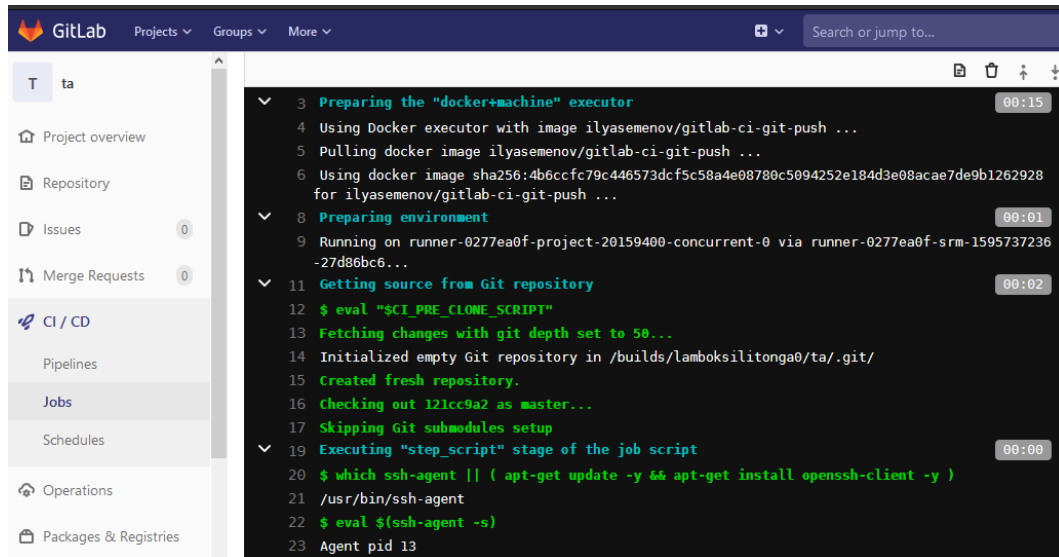
[Save variables](#) [Reveal value](#)

Gambar 17. Tampilan CI/CD *Gitlab*

File `.gitlab-ci.yml` digunakan oleh *GitLab* untuk menentukan tugas yang harus dijalankan, ketika repo diperbarui. Ini memungkinkan melakukan beberapa hal yang cukup rumit, tetapi untuk contoh ini kami akan membuatnya tetap sederhana. Dalam salinan lokal dari repo, buat *file* `.gitlab-ci.yml` dengan yang berikut: -

```
before_script:
  mkdir -p ~/.ssh
  echo "$SSH_PRIVATE_KEY" | tr -d '\r' > ~/.ssh/id_rsa
  chmod 600 ~/.ssh/id_rsa
  ssh-keyscan -H '192.168.1.15' >> ~/.ssh/known_hosts
stages:
  deploy

deploy_to_dokku:
  stage: deploy
  script:
    git checkout master
    git pull
    git push dokku@192.168.1.15:ta2 master
```



Gambar 18. Proses CI/CD Gitlab

Kemudian *push* ke *GitLab*, lalu *pipeline* akan melakukan *deploy* proyek ke *platform dokku* secara otomatis.

4.2.6 Metode Alternatif Deploy Aplikasi

Pada metode alternatif, penulis menggunakan fitur *Git deployment* yang disediakan oleh *Dokku*. Untuk menggunakan fitur ini, pastikan telah melakukan inisiasi *git repository* ke *directory* proyek.

```
# git init
```

Kemudian, akses server *dokku*, IP *dokku* berada pada 192.168.1.15

```
# git remote add dokku dokku@<your-server-ip>:<your-app-name>
```

```
# git push dokku master
```

Dokku akan mencari *Dockerfile* di direktori *root* repositori dan membangun *docker image* yang berasal dari *dockerfile* di server *Dokku*. Dengan demikian, *user* tidak memerlukan akses *Dokku* secara manual untuk melakukan *deploy* hasil proyek, karena cara manual terkesan membutuhkan waktu yang lama, dimulai tahap mengakses server, kemudian mengambil *source code* proyek dari lokal komputer,

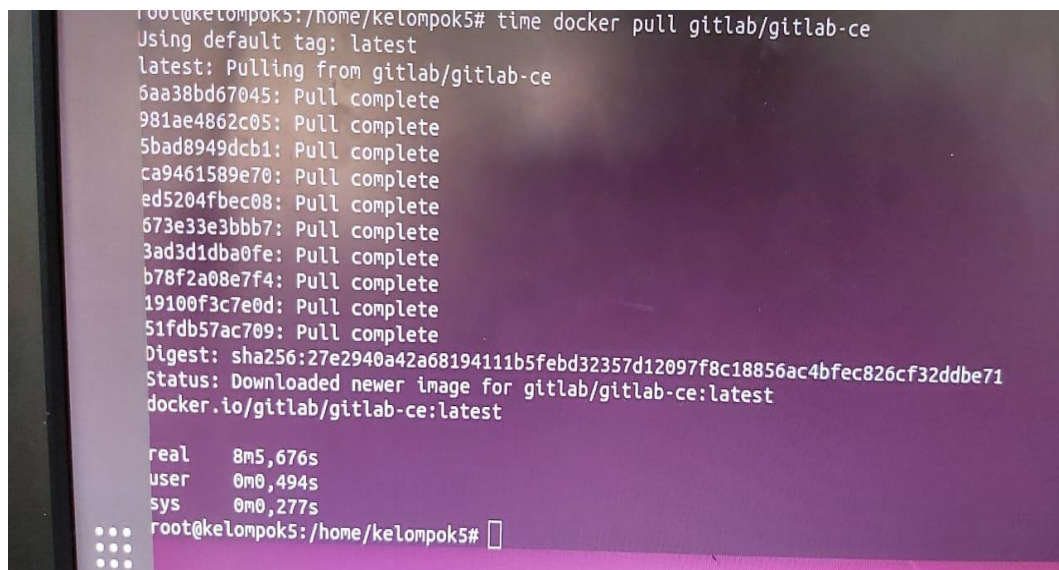
membangun *docker image*, lalu *deploy docker image*, cara sebelumnya dapat dilakukan dengan CI/CD, sehingga lebih mudah dan meningkatkan efisien waktu.

4.2.7 Pengujian Jaringan

Penulis akan melakukan pengujian apakah terjadi peningkatan performa dari jaringan menggunakan LAN IT Del, dibandingkan dengan jaringan internet IT Del. Sehingga nantinya diketahui apakah lebih efisien menggunakan jaringan LAN IT Del atau tidak.

Pengujian jaringan dengan mengunduh *Docker image* dari *Dockerhub* dengan menggunakan jaringan internet IT Del.

Skenario uji pertama adalah dengan cara mengunduh *Docker image* yang berada pada *Dockerhub*(*Docker registry online official*). *Docker* menyediakan *CLI*(*Command Line Interface*) ‘*time*’ untuk mengetahui berapa lama waktu yang digunakan untuk mengunduh sebuah *Docker image*. Dari gambar diketahui, untuk mengunduh *Docker image* berukuran 1.87GB dengan menggunakan jaringan internet IT Del membutuhkan waktu 8 menit dan 5.676 detik.



```
root@kelompok5:/home/kelompok5# time docker pull gitlab/gitlab-ce
Using default tag: latest
latest: Pulling from gitlab/gitlab-ce
6aa38bd67045: Pull complete
981ae4862c05: Pull complete
5bad8949dcb1: Pull complete
ca9461589e70: Pull complete
ed5204fbec08: Pull complete
673e33e3bbb7: Pull complete
3ad3d1dba0fe: Pull complete
b78f2a08e7f4: Pull complete
19100f3c7e0d: Pull complete
51fdb57ac709: Pull complete
Digest: sha256:27e2940a42a6819411b5febd32357d12097f8c18856ac4bfec826cf32ddbe71
Status: Downloaded newer image for gitlab/gitlab-ce:latest
docker.io/gitlab/gitlab-ce:latest

real    8m5,676s
user    0m0,494s
sys     0m0,277s
root@kelompok5:/home/kelompok5#
```

Gambar 19. Waktu dalam mengunduh *docker image* dari *dockerhub* menggunakan jaringan internet IT Del

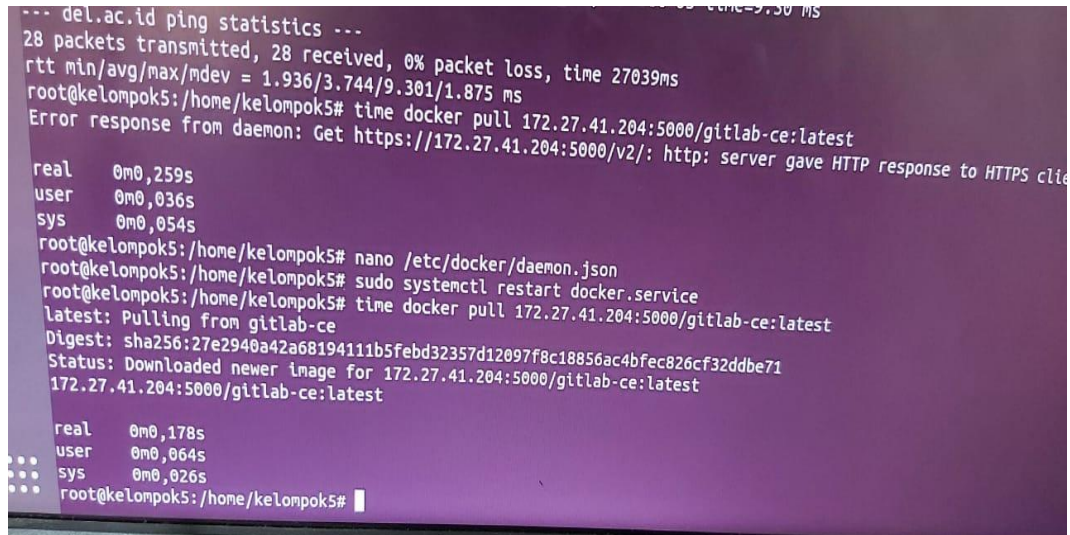
Docker image yang diunduh berukuran 1.87GB



REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
localhost:5000/gitlab-ce	latest	5e6e899aa256	6 days ago	1.87GB
gitlab/gitlab-ce	latest	5e6e899aa256	6 days ago	1.87GB

Gambar 20. Hasil unduhan *docker images*

Pengujian jaringan dengan mengunduh *Docker image* dari *Docker private registry* yang telah diimplementasikan pada server yang terhubung dengan jaringan LAN IT Del. Skenario uji kedua adalah dengan cmengunduh *Docker image* yang berada pada *Docker private registry* yang telah diimplementasikan pada server yang terhubung dengan jaringan LAN IT Del.

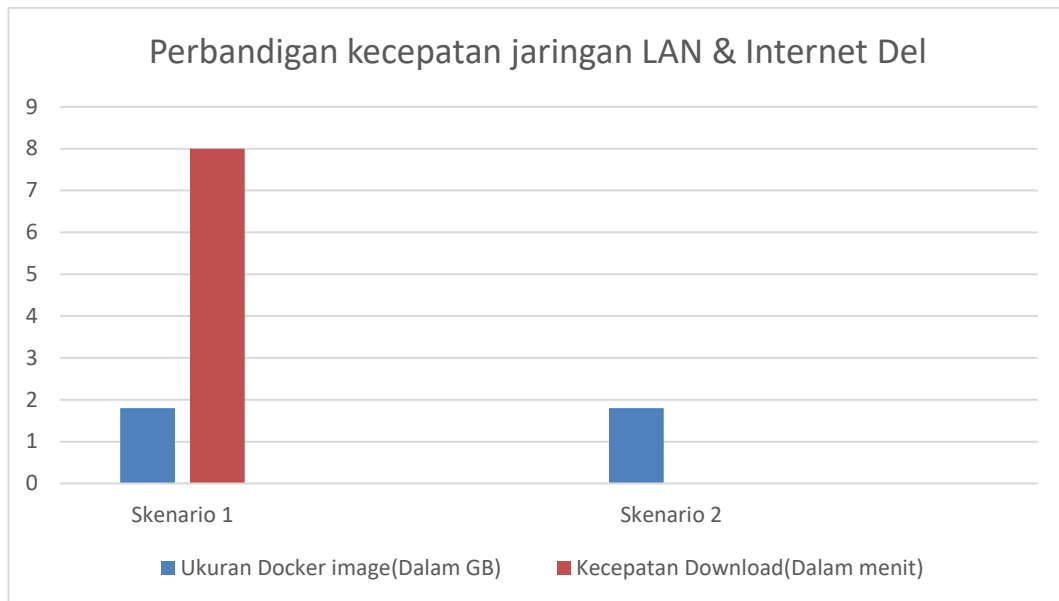


```
--- del.ac.id ping statistics ---
28 packets transmitted, 28 received, 0% packet loss, time 27039ms
rtt min/avg/max/mdev = 1.936/3.744/9.301/1.875 ms
root@kelompok5:/home/kelompok5# time docker pull 172.27.41.204:5000/gitlab-ce:latest
Error response from daemon: Get https://172.27.41.204:5000/v2/: http: server gave HTTP response to HTTPS cli
real    0m0,259s
user    0m0,036s
sys     0m0,054s
root@kelompok5:/home/kelompok5# nano /etc/docker/daemon.json
root@kelompok5:/home/kelompok5# sudo systemctl restart docker.service
root@kelompok5:/home/kelompok5# time docker pull 172.27.41.204:5000/gitlab-ce:latest
latest: Pulling from gitlab-ce
Digest: sha256:27e2940a42a6819411b5febd32357d12097f8c18856ac4bfec826cf32ddb71
Status: Downloaded newer image for 172.27.41.204:5000/gitlab-ce:latest
172.27.41.204:5000/gitlab-ce:latest
real    0m0,178s
user    0m0,064s
sys     0m0,026s
root@kelompok5:/home/kelompok5#
```

Gambar 21. Waktu mengunduh *docker* dari *private docker registry* menggunakan LAN IT Del

Pada gambar 21, dapat dilihat bahwa *Docker* yang berukuran 1.87 GB, dapat diunduh dengan waktu 0,178 detik.

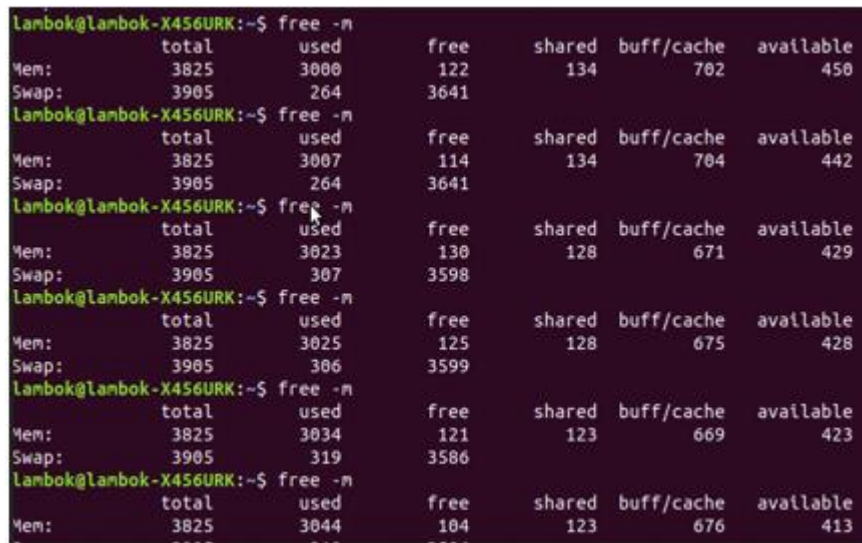
Hasil skenario pengujian



Gambar 22. Hasil skenario pengujian

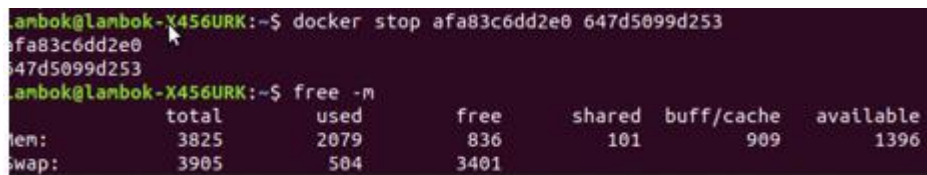
Dari pengujian diatas, dapat ketahui bahwa terjadi peningkatan transfer data saat menggunakan jaringan LAN IT Del, *Docker image* yang diunduh keduanya berukuran 1.87 GB agar tidak terdapat perbedaan. Hasilnya dengan ukuran 1.8 GB, jaringan internet IT Del mengunduh dengan waktu yang dibutuhkan adalah sekitar 8 menit, sedangkan LAN IT Del hanya membutuhkan 1 detik, dari skenario uji ini dapat dibuktikan bahwa kecepatan transfer data dengan jaringan LAN IT Del lebih efisien dan singkat, dibandingkan dengan jaringan internet IT Del.

4.2.8 Perbandingan *Memory Usage* Ketika Menjalankan *Dokku* dan *Gitlab* secara bersamaan dan bergantian



```
lambok@lambok-X456URK:~$ free -m
              total        used        free      shared  buff/cache   available
Mem:           3825         3000          122         134         702         450
Swap:          3905          264        3641
lambok@lambok-X456URK:~$ free -m
              total        used        free      shared  buff/cache   available
Mem:           3825         3007          114         134         704         442
Swap:          3905          264        3641
lambok@lambok-X456URK:~$ free -m
              total        used        free      shared  buff/cache   available
Mem:           3825         3023          130         128         671         429
Swap:          3905          307        3598
lambok@lambok-X456URK:~$ free -m
              total        used        free      shared  buff/cache   available
Mem:           3825         3025          125         128         675         428
Swap:          3905          306        3599
lambok@lambok-X456URK:~$ free -m
              total        used        free      shared  buff/cache   available
Mem:           3825         3034          121         123         669         423
Swap:          3905          319        3586
lambok@lambok-X456URK:~$ free -m
              total        used        free      shared  buff/cache   available
Mem:           3825         3044          104         123         676         413
Swap:          3905          306        3599
```

Gambar 23. *Docker container Gitlab dan Dokku* berjalan bersamaan pada satu komputer



```
lambok@lambok-X456URK:~$ docker stop afa83c6dd2e0 647d5099d253
lambok@lambok-X456URK:~$ free -m
              total        used        free      shared  buff/cache   available
Mem:           3825         2079          836         101         909        1396
Swap:          3905          504        3401
```

Gambar 24. *Docker container Gitlab* setelah dihentikan dan menyisakan *Dokku* yang berjalan pada sebuah komputer

Dapat dilihat antara gambar 23 dan gambar 24, penggunaan *memory* ketika *Gitlab* dan *Dokku* berjalan memerlukan 3GB *memory*, ketika *Gitlab container* dihentikan, penggunaan *memory* turun dari 3GB ke 836Mb. *Gitlab* tergolong membebani *traffic memory* tetapi untuk *service Dokku* sendiri tidak terlalu membebani, sehingga didapat penjelasan bahwa semakin besar ukuran dari *container* maka semakin berat untuk dijalankan, sehingga membutuhkan CPU dengan kinerja yang lebih tinggi.

4.2.9 Pengujian *Lifecycle* Menggunakan *Update Source Code Website*

Pada subbab ini akan dilakukan pengujian life cycle menggunakan website HTML, website static, dan website PHP

4.2.9.1 Pengujian pada Website HTML

Agar *source code website* dapat disimpan ke gitlab, maka lakukan perintah berikut:

1. `git config --global user.name 'namauser'`
2. `git config --global user.email 'email.user'`

Perintah diatas bertujuan agar tidak mengetikkan *username* apabila ingin berkali-kali menyimpan code ke git, dikarenakan kemungkinan *update source code*.

3. `git clone 192.168.1.100:8929:directorylink.git`

Perintah tersebut akan mengambil *directory* yang berasal dari *gitlab*, dan akan mengunduh *directory* kosong yang terdapat pada *gitlab*.

4. Selanjutnya *user* dapat menambahkan *source code* yang ingin disimpan ke dalam *gitlab*.

Karena keterbatasan perangkat, *website* yang penulis ciptakan adalah *website* sederhana dengan menggunakan HTML. Langkah pertama yang diperlukan dari menggunakan *website* ini adalah code *website website* itu sendiri. Berikut adalah contoh *source code* yang akan disimpan ke dalam *gitlab*, *website* berikut hanya akan menampilkan kata “Kelompokk 5 TA2 here”.

```
#index.html

<!DOCTYPE html>

<html lang="en">

  <head>

    <meta charset="UTF-8" />

    <meta name="viewport" content="width=device-width, initial-scale=1.0" />

    <meta http-equiv="X-UA-Compatible" content="ie=edge" />

    <title>Contoh project</title>

  </head>
```

```
<body>

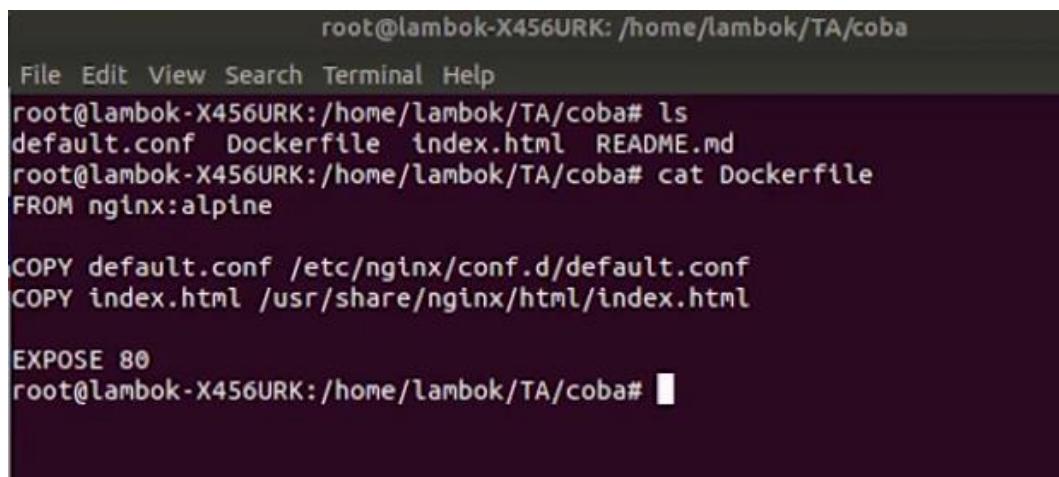
  <h1>Hey, it works!</h1>

  <h1>Kelompokk 5 TA2 here.</h1>

</body>

</html>
```

Kemudian, untuk berlangsung ke tahap selanjutnya, *developer* perlu membuat *dockefile*, *Dockerfile* ini berfungsi menyediakan *infrastructure* seperti jaringan dan lain lain, agar sebuah *website* dapat di-*publish*, dapat dilihat *website* agar dapat diakses, maka dibuka port 80(EXPOSE), sedangkan perintah lainnya adalah COPY, yang bertujuan untuk menyatukan semua *code file* yang dibutuhkan.



```
root@lambok-X456URK: /home/lambok/TA/coba
File Edit View Search Terminal Help
root@lambok-X456URK:/home/lambok/TA/coba# ls
default.conf Dockerfile index.html README.md
root@lambok-X456URK:/home/lambok/TA/coba# cat Dockerfile
FROM nginx:alpine

COPY default.conf /etc/nginx/conf.d/default.conf
COPY index.html /usr/share/nginx/html/index.html

EXPOSE 80
root@lambok-X456URK:/home/lambok/TA/coba#
```

Gambar 25. Code *dockerfile*

```
#Dockerfile

FROM nginx:alpine


COPY default.conf /etc/nginx/conf.d/default.conf

COPY index.html /usr/share/nginx/html/index.html
```

```
EXPOSE 80
```

```
RUN mkdir /myapp
```

Selanjutnya terdapat *file* `default.conf`, *file* berikut berisi *setting* yang digunakan untuk mengkonfigurasi *nginx*, sehingga terdapat didalam *Docker image*, yang bertujuan sebagai web server.

```
#default.conf
server {
    listen    80;

    server_name localhost;

    location / {
        root /usr/share/nginx/html;

        index index.html;
    }
}
```

5. Selanjutnya, jalankan perintah CI/CD, CI/CD ini bertujuan agar mahasiswa/i tidak memerlukan *deploy website* secara manual, ketika *user* menyimpan *source code* ke dalam *gitlab*, secara bersamaan, *Docker image* dari *website* akan dibangun, dan dijalankan diatas *Dokku* (Paas).

```
# git remote add origin 192.168.1.15:8969/ta/ta2.git
```

```
# git push origin master
```

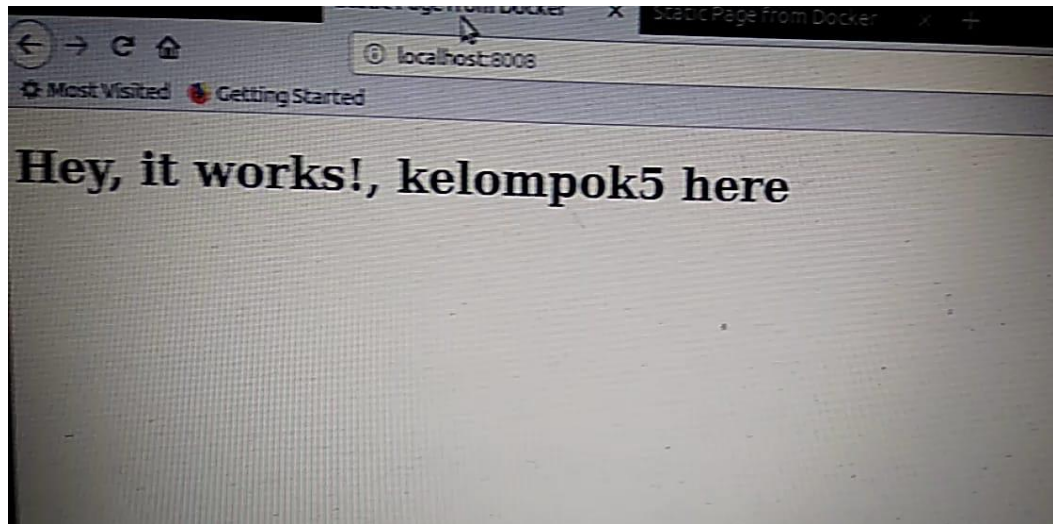
Kemudian *command* diatas akan menyimpan *source code* ke *gitlab*, dan sekaligus melakukan *deploy* ke *Dokku*.

```

lambok@lambok-X456URK:~$ docker ps
CONTAINER ID   IMAGE          COMMAND                  NAMES
9294354f54c5   dokku/coba5:latest  "nginx -g 'daemon of..."  Abou
Up About a minute   80/tcp          coba5.web.1

```

Gambar 26. Docker dari website yang telah berhasil dideploy ke dokku



Gambar 27. Website sebelum diupdate

4.2.9.2 Uji Coba Update Website Static

1. Apabila mahasiswa/i lain yang ingin meng-update website, maka user perlu mengunduh *source code* pada *gitlab* dengan cara.

```
# git pull origin master
```

```

root@lambok-X456URK:/home/lambok/TA/coba# ls
default.conf Dockerfile index.html README.md
root@lambok-X456URK:/home/lambok/TA/coba#

```

Gambar 28. File website static yang diuji

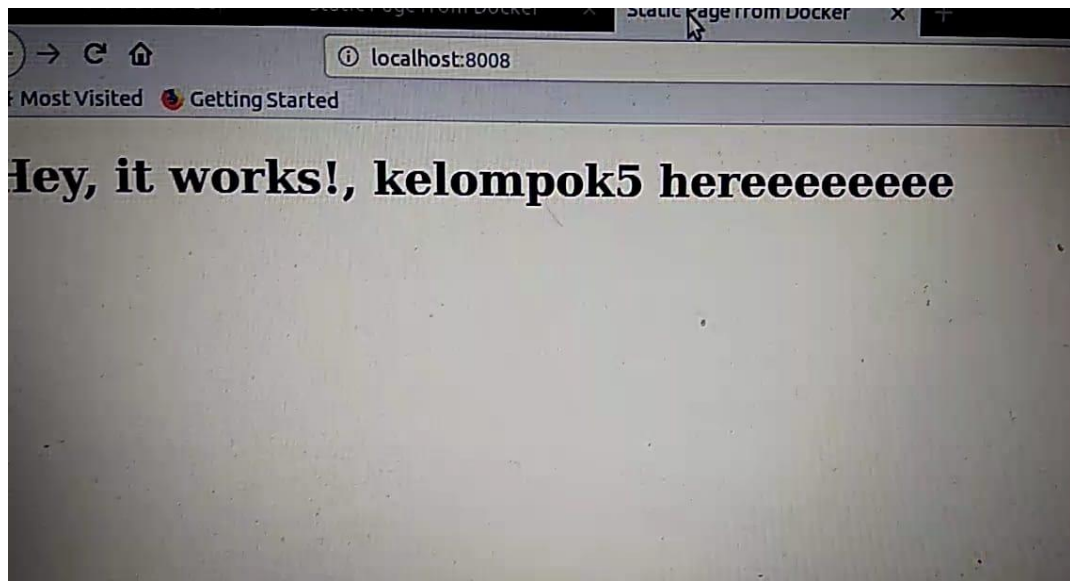
2. Kemudian, perintah tersebut akan mengunduh 3 file sebelumnya, yaitu *index.html*, *Dockerfile*, dan *default.conf*. Pada case ini, penulis meng-update kata pada file dari “Kelompokkk 5 TA2 here” menjadi “Kelompokkk 5 TA2 hereeeee”, untuk membuktikan adanya *update*.

```
#index.html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <meta http-equiv="X-UA-Compatible" content="ie=edge" />
    <title>Contoh project</title>
  </head>
  <body>
    <h1>Hey, it works!</h1>
    <h1>Kelompokk 5 TA2 hereeeeeee.</h1>
  </body>
</html>
```

3. Selanjutnya, *user* dapat meng-*push* kembali *source code* sekaligus *deploy* ke dalam *Dokku* menggunakan CI/CD, sehingga tercipta siklus *life cycle*, pada proses CI/CD tidak dapat menunjukkan di *GitLab*, sehingga untuk melihat proyek yg telah *update*, terlebih dahulu penulis mematikan *service GitLab*, kemudian mengakses kembali proyek untuk *check website*.

```
# git remote add origin 192.168.1.15:8969/ta/ta2.git
```

```
# git push origin master
```

Gambar 29. Website setelah diupdate

4.2.9.3 Skenario pada Website PHP

Masuk ke *directory* proyek, kemudian buatlah sebuah *Dockerfile* dengan perintah:

```
# nano Dockerfile
```

Kemudian masukkan isi file berikut pada *dockerfile*.

```
#Dockerfile
FROM php:7.0-apache
RUN apt-get update && \
    apt-get install -y php5-mysql && \
    apt-get clean
COPY myapp /var/www/html/
EXPOSE 8919:80
```

Pada *directory* yang sama, ciptakan sebuah *directory* baru dengan nama '*myapp*'.

```
# mkdir myapp  
  
#cd myapp
```

Lalu masukkan *file code* proyek pada *folder myapp*. berikut salah satu contoh code PHP yang digunakan dalam skenario uji.

```
# /myapp/index.php  
  
<!DOCTYPE html>  
  
<html>  
  
  <head>  
  
    <title>Telerik® UI for PHP &hearts; Twitter Bootstrap: Responsive  
demo</title>  
  
    <meta name="viewport" content="width=device-width, initial-scale=1.0,  
maximum-scale=1.0, user-scalable=no">  
  
    <?php require_once 'includes/includes.php'; ?>  
  
    <?php require_once 'lib/Kendo/Autoload.php'; ?>  
  
    <script src="theme-chooser.js"></script>  
  
  
    <link rel="stylesheet" href="styles.css" />  
  
  </head>  
  
  <body>  
  
    <?php include 'includes/header.php';?>  
  
  
    <div id="example" class="container">  
  
      <?php include 'includes/whatIsThisPage.php';?>
```

```
<?php include 'includes/menu.php';?>
```

```
<div class="row clearfix">
```

```
    <div class="col-lg-4">
```

```
        <?php include 'includes/profile.php';?>
```

```
    </div>
```

```
    <div class="col-lg-8">
```

```
        <?php include 'includes/tabstrip.php';?>
```

```
    </div>
```

```
</div>
```

```
<?php include 'includes/profileSetup/profileSetup.php';?>
```

```
<?php include 'includes/orders.php';?>
```

```
<?php include 'includes/schedule.php';?>
```

```
<?php include 'includes/latestPhotoUploads.php';?>
```

```
<?php include 'includes/faq.php';?>
```

```

<footer>Copyright &copy; <?php echo date("Y"); ?>, Progress Software
Corporation and/or its subsidiaries or affiliates. All Rights Reserved.</footer>

</div>

</body>

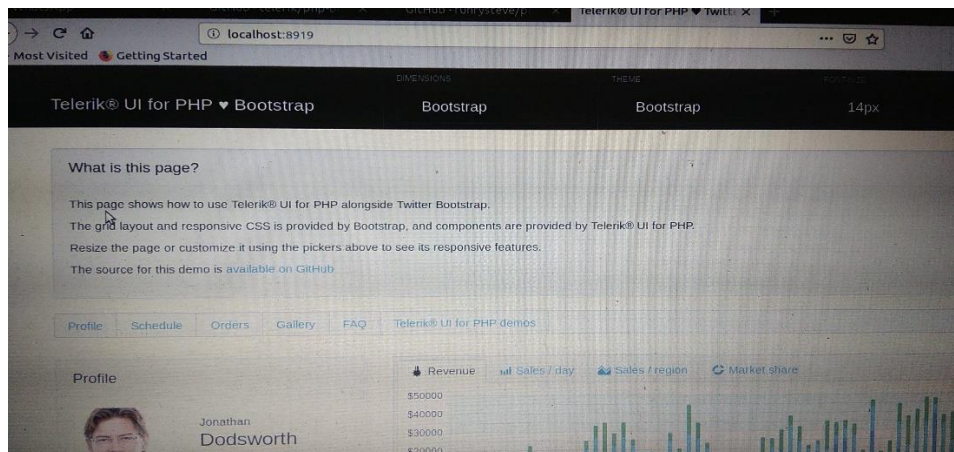
</html>

```

Kemudian lakukan *deployment website* pada *dokku* dengan menggunakan *CI/CD* dengan cara:

```
# git remote set-url origin 192.168.1.15:8929/nama-lokasi-repo.git
```

```
# git push origin master
```



Gambar 30. Website PHP setelah *dideploy*

Kemudian, *website* kembali di-*update*, *user* menambahkan kalimat lain pada *source code index.php* pada yang bertujuan untuk meng-*update header website*, lalu kembali lakukan *CI/CD*, sehingga tercipta *lifecycle* dengan *update website*. Pada *website* PHP, laptop yang digunakan setelah *deploy* proyek *overheat*, karena tidak mampu menahan *GitLab* dan PHP yang telah *deploy* memakan cukup banyak *resource memory*.

```
<!DOCTYPE html>
```

```

<html>

  <head>

    <title>Telerik® UI for PHP &hearts kelompok5; Twitter Bootstrap:
    Responsive demo</title>

    <meta name="viewport" content="width=device-width, initial-scale=1.0,
    maximum-scale=1.0, user-scalable=no">

    <?php require_once 'includes/includes.php'; ?>

    <?php require_once 'lib/Kendo/Autoload.php'; ?>

    <script src="theme-chooser.js"></script>

    <link rel="stylesheet" href="styles.css" />

  </head>

  <body>

    <?php include 'includes/header.php';?>

    <div id="example" class="container">

      <?php include 'includes/whatIsThisPage.php';?>

      <?php include 'includes/menu.php';?>

      <div class="row clearfix">

        <div class="col-lg-4">

          <?php include 'includes/profile.php';?>

        </div>

```

```

        <div class="col-lg-8">

            <?php include 'includes/tabstrip.php';?>

        </div>

    </div>

    <?php include 'includes/profileSetup/profileSetup.php';?>

    <?php include 'includes/orders.php';?>

    <?php include 'includes/schedule.php';?>

    <?php include 'includes/latestPhotoUploads.php';?>

    <?php include 'includes/faq.php';?>

    <footer>Copyright &copy; <?php echo date("Y"); ?>, Progress Software
    Corporation and/or its subsidiaries or affiliates. All Rights Reserved.</footer>

    </div>

</body>

</html>

```

```

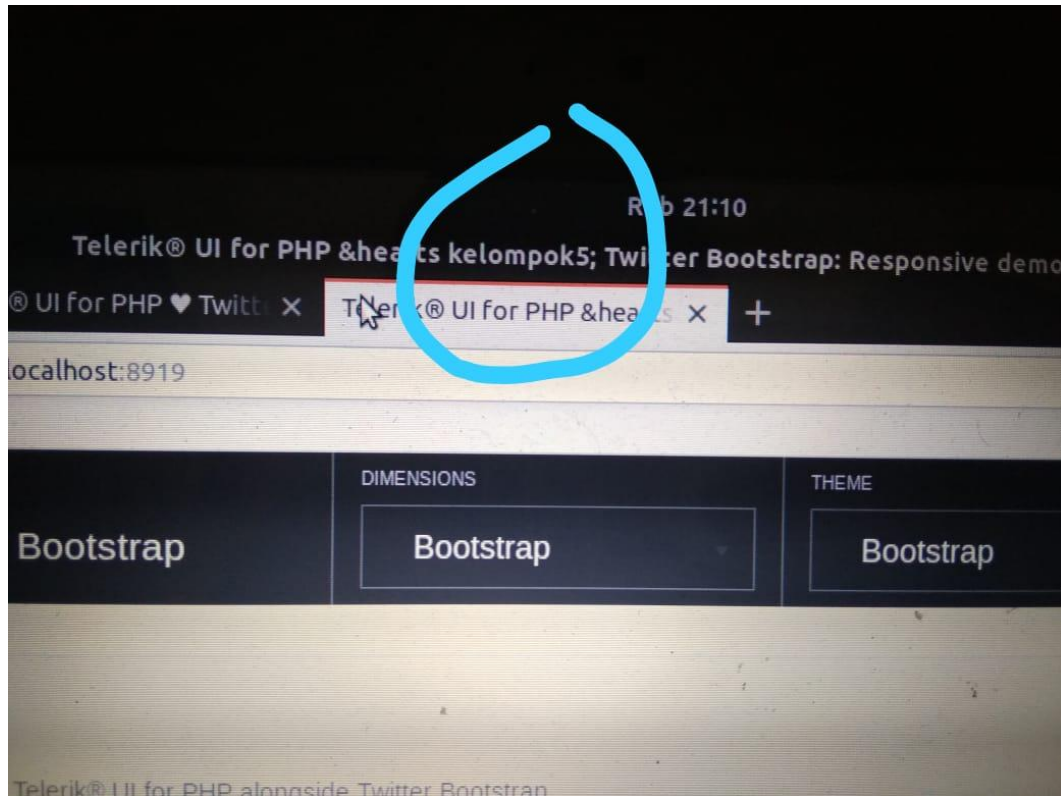
# git remote add origin 192.168.1.15:8929/nama-lokasi-repo.git

```

```

# git push master

```



Gambar 31. Website PHP setelah diupdate

4.2.9.4 Skenario pada *Website PHP* dan *Database MYSQL*

Pada skenario *website PHP* yang menggunakan *database MYSQL*, penulis tidak menggunakan *CI/CD*, karena *gitlab* menggunakan banyak *resource memory*, yang mengakibatkan laptop *overheat*. Sehingga untuk menggantikan *CI/CD*, penulis menyimpan hasil *docker* proyek ke *private docker registry*. Pada skenario, aplikasi sederhana yang membaca daftar kota dari *database* dan menampilkannya di halaman web.

```
mkdir ~/docker  
cd ~/docker  
mkdir php nginx app
```

Perintah diatas akan menciptakan *directory source code website*. Penulis menggunakan *php-fpm* untuk terhubung ke server web *Nginx*. Penulis

menggunakan *image docker* dasar PHP *official*. Di dalam *folder php* buat *file* bernama *Dockerfile* dan masukkan konten berikut ke dalamnya.

Pada *folder php*, ciptakan sebuah *Dockerfile*.

```
# cd php/  
# nano Dockerfile
```

Kemudian, masukkan *dockerfile* berikut.

```
FROM php:7.1-fpm-alpine3.4  
RUN apk update --no-cache \  
    && apk add --no-cache $PHPIZE_DEPS \  
    && apk add --no-cache mysql-dev \  
    && docker-php-ext-install pdo pdo_mysql
```

Kemudian bangun *dockerfile* menjadi *docker image* dengan perintah:

```
# docker build -t vultr-php .
```

Kemudian, menciptakan sebuah *docker-compose*, *docker-compose* berfungsi untuk mengelola sejumlah *container* melalui *file* konfigurasi sederhana..

```
# nano app/docker-compose.yml  
version: '2'  
services:  
  php:  
    image: vultr-php  
    volumes:  
      - ./:/app  
    working_dir: /app
```



```
# docker-compose up -d
```

Kemudian, langkah selanjutnya adalah membuat konfigurasi untuk *webserver*, Tetapi dalam hal ini, kami hanya menyediakan konfigurasi untuk *virtual host*.

```
# cd ~/docker
# nano nginx/Dockerfile
FROM nginx:1.13.8-alpine
COPY ./default.conf /etc/nginx/conf.d/default.conf
```

Penulis menggunakan *docker image Nginx* berdasarkan *Alpine*. *Dockerfile* menyalin *file* konfigurasi ke pengaturan *docker*.

```
# touch nginx/default.conf
server {
    listen 80 default_server;
    listen [::]:80 default_server ipv6only=on;

    root /app;
    index index.php;

    #server_name server_domain_or_IP;

    location / {
        try_files $uri $uri/ /index.php?$query_string;
    }

    location ~ \.php$ {
        try_files $uri /index.php =404;
        fastcgi_split_path_info ^(.+\.php)(/.+)$;
```

```
    fastcgi_pass php:9000;
    fastcgi_index index.php;
    fastcgi_param                                SCRIPT_FILENAME
$document_root$fastcgi_script_name;
    include fastcgi_params;
}
}
```

Pada baris `fastcgi_pass php:9000`, PHP *container* dalam *service block* konfigurasi *file* `docker-compose.yml`. Secara internal, *docker-compose* membuat jaringan dan menetapkan nama layanan sebagai nama *host* untuk setiap layanan yang ditentukan. Sekarang *docker-image Nginx* dapat dibangun.

Kemudian, lakukan perintah dari *directory docker/* :

```
# docker build -t vultr-nginx nginx/
```

Update kembali *file* `app/docker-compose.yml`

```
version: '2'
services:
  php:
    image: vultr-php
    volumes:
      - ./:/app
    working_dir: /app
  web:
    image: vultr-nginx
    volumes:
      - ./:/app
    depends_on:
      - php
    ports:
```

```
- 80:80
```

Setelah *container docker Nginx* yang membutuhkan layanan PHP sepenuhnya diinisialisasi, memetakan konfigurasi dalam opsi `depend_on`. Konfigurasi *port* memetakan *port host* ke *port container*, memetakan *port 80* di *host* ke *port 80* dalam *container*.

Kemudian ciptakan *file index.php* pad *folder app/*, dan masukkan *file* berikut.

```
<?php phpinfo();  
# cd ~/docker/app  
# docker-compose up -d
```

Untuk selanjutnya, kita perlu mengkonfigurasi *container database MySQL*. *Update* kembali *file ~/docker/app/docker-compose.yml*

```
version: '2'  
services:  
  php:  
    image: vultr-php  
    volumes:  
      - ./app  
    working_dir: /app  
  web:  
    image: vultr-nginx  
    volumes:  
      - ./app  
    depends_on:  
      - php  
    ports:  
      - 80:80  
  mysql:
```

```
image: mysql:5.7.21
volumes:
  - ./app
  - dbdata:/var/lib/mysql
environment:
  - MYSQL_DATABASE=world
  - MYSQL_ROOT_PASSWORD=root
working_dir: /app
volumes:
  dbdata:
```

Service untuk *database* telah dikonfigurasi. Pada Baris `dbdata:/var/lib/mysql`, Ini *me-mount path* pada *container* `/var/lib/mysql` ke *volume* persisten yang dikelola oleh *docker*, dengan cara ini data *database* tetap ada setelah *container* dihapus.

Kemudian unduh *database*

```
# curl -L http://downloads.mysql.com/docs/world.sql.gz -o world.sql.gz
# gunzip world.sql.gz
```

Kemudian lakukan perintah:

```
# docker-compose up -d
```

Kemudian, konfigurasi *database* yang telah diunduh sebelumnya:

```
# docker-compose exec -T mysql mysql -u root -p root world < world.sql
```

Kemudian, untuk memastikan bahwa *database* telah terpasang pada *docker*, lakukan perintah:

```
# docker-compose exec mysql mysql -uroot -proot world
```

Pada *prompt MySQL*, jalankan perintah:

```
# select * from city limit 10;
```

Akan menampilkan isi *database* dengan list kota/*city*.

Untuk keluar dari database, lakukan perintah :

```
# mysql> exit
```

Selanjutnya, untuk menampilkan *website*, *update file app/index.php*.

```
<?php

$pdo = new PDO('mysql:host=mysql;dbname=world;charset=utf8', 'root', 'root');

$stmt = $pdo->prepare("
    select city.Name, city.District, country.Name as Country, city.Population
    from city
    left join country on city.CountryCode = country.Code
    order by Population desc
    limit 10
");
$stmt->execute();
$cities = $stmt->fetchAll(PDO::FETCH_ASSOC);

?>

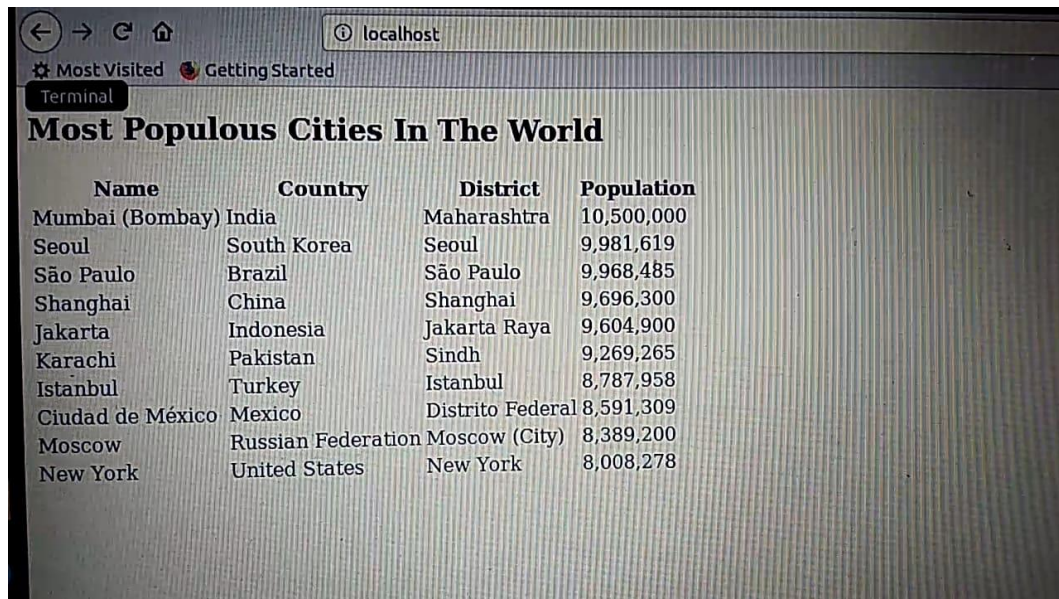
<!doctype html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Vultr Rocks!</title>
</head>
```

```

<body>
  <h2>Most Populous Cities In The World</h2>
  <table>
    <thead>
      <tr>
        <th>Name</th>
        <th>Country</th>
        <th>District</th>
        <th>Population</th>
      </tr>
    </thead>
    <tbody>
      <?php foreach($cities as $city): ?>
        <tr>
          <td><?=$city['Name']?></td>
          <td><?=$city['Country']?></td>
          <td><?=$city['District']?></td>
          <td><?=number_format($city['Population'], 0)?></td>
        </tr>
      <?php endforeach ?>
    </tbody>
  </table>
</body>
</html>

```

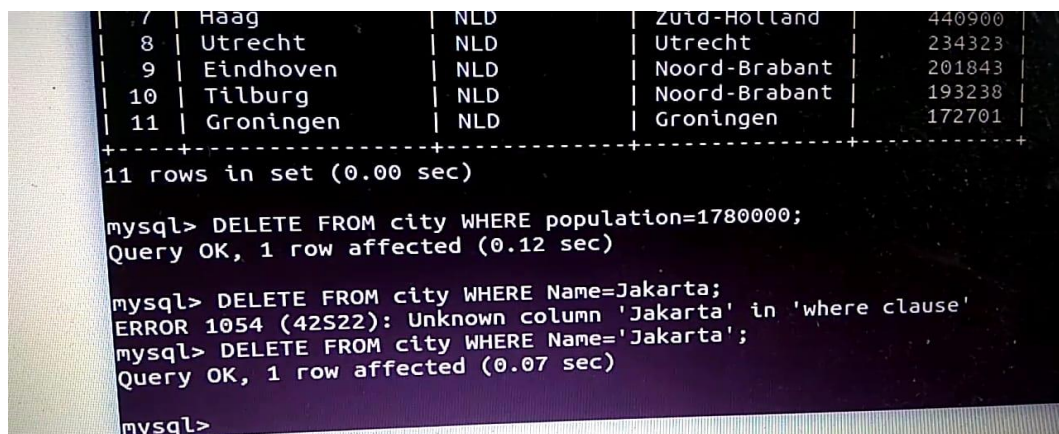
Lalu mengakses *docker container* [vultr-instance-ip] di peramban *website*, akan melihat daftar kota terpadat di dunia.



Name	Country	District	Population
Mumbai (Bombay)	India	Maharashtra	10,500,000
Seoul	South Korea	Seoul	9,981,619
São Paulo	Brazil	São Paulo	9,968,485
Shanghai	China	Shanghai	9,696,300
Jakarta	Indonesia	Jakarta Raya	9,604,900
Karachi	Pakistan	Sindh	9,269,265
Istanbul	Turkey	Istanbul	8,787,958
Ciudad de México	Mexico	Distrito Federal	8,591,309
Moscow	Russian Federation	Moscow (City)	8,389,200
New York	United States	New York	8,008,278

Gambar 32. Tampilan website PHP menggunakan *database MySQL*

Selanjutnya, untuk mempraktikkan *lifecycle*, kami kembali mengupdate database.



```

+---+---+---+---+---+
| 7 | Haag | NLD | Zuid-Holland | 440900 |
| 8 | Utrecht | NLD | Utrecht | 234323 |
| 9 | Eindhoven | NLD | Noord-Brabant | 201843 |
| 10 | Tilburg | NLD | Noord-Brabant | 193238 |
| 11 | Groningen | NLD | Groningen | 172701 |
+---+---+---+---+---+
11 rows in set (0.00 sec)

mysql> DELETE FROM city WHERE population=1780000;
Query OK, 1 row affected (0.12 sec)

mysql> DELETE FROM city WHERE Name=Jakarta;
ERROR 1054 (42S22): Unknown column 'Jakarta' in 'where clause'
mysql> DELETE FROM city WHERE Name='Jakarta';
Query OK, 1 row affected (0.07 sec)

mysql>

```

Gambar 33. Mengupdate website dengan mengupdate *database*

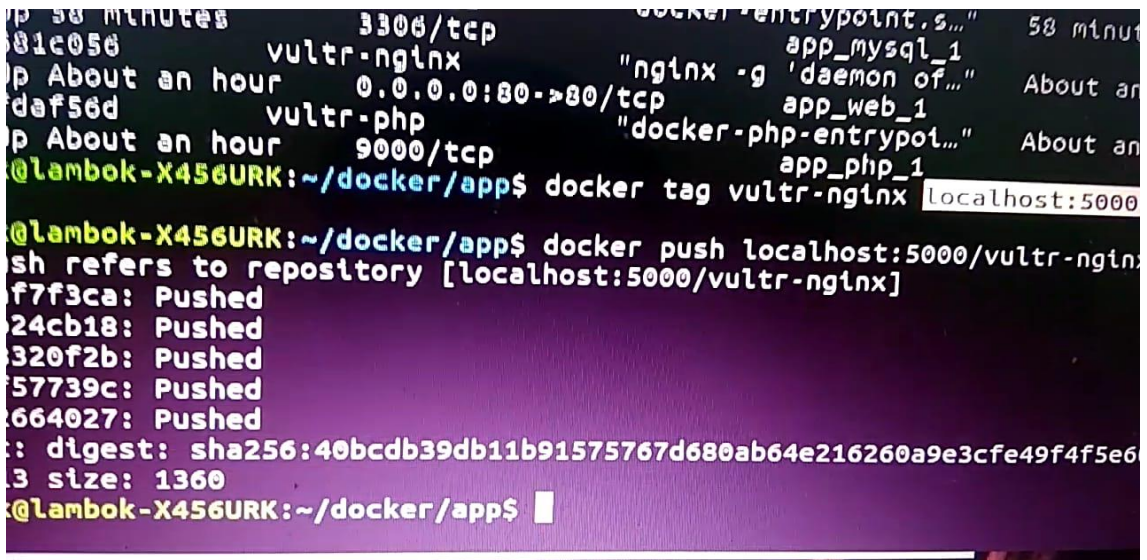
Tujuan perintah di atas adalah, kami menghapus Jakarta dari *database* kota terpadat didunia, dan ketika di-reload, secara otomatis, kota Jakarta terhapus dari *website*.



Name	Country	District	Population
Mumbai (Bombay)	India	Maharashtra	10,500,000
Seoul	South Korea	Seoul	9,981,619
São Paulo	Brazil	São Paulo	9,968,485
Shanghai	China	Shanghai	9,696,300
Karachi	Pakistan	Sindh	9,269,265
Istanbul	Turkey	Istanbul	8,787,958
Ciudad de México	Mexico	Distrito Federal	8,591,309
Moscow	Russian Federation	Moscow (City)	8,389,200
New York	United States	New York	8,008,278
Tokyo	Japan	Tokyo-to	7,980,230

Gambar 34. Tampilan website setelah *database* update, Jakarta menghilang dari website

Pada gambar 34. Setelah project diupdate dengan cara menghapus kota jakarta dari *database*, dapat dibuktikan bahwa proyek telah berhasil *update* sehingga memenuhi proses *life cycle*, dapat dilihat pada gambar bahwa Jakarta telah berhasil dihapus dari tampilan *website*.



```

@lambok-X456URK:~/docker/app$ docker tag vultr-nginx localhost:5000/vultr-nginx
@lambok-X456URK:~/docker/app$ docker push localhost:5000/vultr-nginx
The push refers to repository [localhost:5000/vultr-nginx]
f7f3ca: Pushed
24cb18: Pushed
320f2b: Pushed
57739c: Pushed
664027: Pushed
Digest: sha256:40bcdb39db11b91575767d680ab64e216260a9e3cfe49f4f5e6
Size: 1360
@lambok-X456URK:~/docker/app$

```

Gambar 35. Docker image dapat dipush ke private docker registry

Pada Gambar 35, untuk menggantikan proses CI/CD, penulis menyimpan hasil proyek *docker image* ke *private docker registry*, sehingga secara teori, dapat dibuktikan bahwa CI/CD akan berhasil.

BAB 5 KESIMPULAN DAN SARAN

Pada bab ini berisi kesimpulan dan saran mengenai Tugas Akhir yang sudah dilakukan, dan saran-saran yang akan disampaikan untuk masa yang akan datang bagi pengembangan dan juga untuk pembaca.

5.1 Kesimpulan

Setelah melakukan implementasi pengerjaan proyek menggunakan konsep DevOps dengan mengintegrasikan *tools software Docker, Gitlab, dan Dokku* dengan menggunakan jaringan LAN Institut Teknologi Del, dapat disimpulkan bahwa ketiga *tools* tersebut mampu menghasilkan konsep pengerjaan proyek yang bersifat *life cycle* yang diterapkan dalam DevOps. Setelah membandingkan waktu dalam kecepatan mengunduh *Docker image*, menggunakan jaringan LAN lebih efektif dibandingkan menggunakan jaringan internet IT Del.

5.2 Saran

Setelah dilakukan implementasi *tools*, menurut penulis ketiga *tools* tersebut tidak dapat diterapkan dalam sebuah komputer (server) dengan spesifikasi menengah sampai kebawah dikarenakan beban *traffic* dari *tools* tersebut yang terkesan besar dan berat. Diketahui bahwa beban *traffic* CPU untuk menerapkan *tools Docker, Gitlab, dan Dokku*, tidak cukup hanya dengan komputer standar. Sehingga untuk implementasi diharapkan mampu mengimplementasikan *tools* tersebut pada komputer memadai, sehingga beban CPU pada komputer tidak terlalu berat dan tidak menyebabkan *system overheat*.

DAFTAR PUSTAKA DAN RUJUKAN

- [1] Fadli, A., Nurshiami, S., Taryana, A. (2020). *Merancang Software Sistem Penjaminan Mutu Internal (SPMI) Perguruan Tinggi yang Memiliki Daya Adaptasi Terhadap Perubahan Kebutuhan Pengguna secara Cepat dan Sering*. Jurnal Al-azhar Indonesia Seri Sains dan Teknologi Vol.5, No.3.
- [2] Arvind. (2019). *DevOps Life cycle: Everything You Need To Know About DevOps Life cycle Phases*. Di akses 20 April, 2020, dari [https://www.edureka.co/blog/DevOps-life cycle](https://www.edureka.co/blog/DevOps-life-cycle).
- [3] Jha, P. & Khan, R. (2018). *A Review Paper on DevOps: Beginning and More To Know*. International Journal of Computer Applications (0975 – 8887) Volume 180 – No.48.
- [4] Lestari W. & Sujarwo, A. (2018). *DevOps: Disrupsi pengelolaan ICT pendidikan tinggi*. Seminar Nasional Aplikasi Teknologi Informasi.
- [5] Feijter, R., dkk. (2017). *Towards the adoption of DevOps in software product organizations: A maturity model approach*. Department of Information and Computing Sciences Utrecht University, Utrecht, The Netherlands.
- [6] Boettiger, C. & Eddelbuettel, D. (2017). *An Introduction to Rocker: Docker Containers for R*. The R Journal Vol.9, No.2.
- [7] Khalida, R., Muhajirin, A., Setiawati, S. (2019). *Teknis Kerja Docker Container untuk Optimalisasi Penyebaran Aplikasi*. Penelitian Ilmu Komputer Sistem Embedded and Logic No.167 – 176.
- [8] Perera, I. & Suriya, S. (2018). *Continuous Integration and Continuous Delivery Pipeline Automation for Agile Software Project Management*. University of Moratuwa: Sri Lanka.
- [9] Saputra, W., Wahanani, H., Wahono, B. (2019). *Perancangan Infrastruktur Server VCS (Version Control System) dengan Gitlab berbasis Git*. Jurnal Program

Studi Teknik Informatika Fakultas Ilmu Komputer, Universitas Pembangunan Nasional Veteran Vol.14, No.2.

[10] Westby, Emma Jane Hogbin. (2015). *Git for Teams*. O'Reilly Media: Amerika Serikat.

[11] Masombuka, T. & Mnkandla, E. (2018). *A DevOps collaboration culture acceptance model*. South African Institute of Computer Scientists and Information Technologists.

[12] Grance, T. & Mell, P. (2012). *The NIST Definition of Cloud Computing*. National Institute of Standards and Technology: U.S. Department of Commerce.

[13] Alim, Z. & Cancer, Y. (2016). *Platform as a Service (Paas) sebagai Layanan Sistem Operasi Cloud Computing*. Jurnal Times Vol. 5, No. 1: 32-35.

[14] Trinh, P. & Doan, M. (2016). *Implementation of Continuous Integration and Continuous Delivery in Scrum. Case Study: Food 'N Stuff and WebRTC Applications*. Lahti University of Applied Sciences: Finlandia.

[15] Humble, J. & Molesky, J. (2011). *Why enterprises must adopt DevOps to enable continuous delivery*. Cutter IT Journal Vol.24, No.8.

[16] Bella, M., Data, M, Yahya, W. (2019). *Implementasi Load Balancing Server Web Berbasis Docker Swarm Berdasarkan Penggunaan Sumber Daya Memory Host*. Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer Vol. 3, No. 9

[17] Anonim. (2019). *Disinformation and 'fake news': Final Report*. House of Commons Digital, Culture, Media and Sport Committee.

[18] Abrahamsson, H., dkk. (2002). *A Multi-path Routing Algorithm for IP Networks Based on Flow Optimisation*. SICS – Swedish Institute of Computer Science pp. 135–144.

[19] Singh, R., Singh, Y., Yadav, A. (2016). *Loop Free Multipath Routing Algorithm*. Department Indian Institute of Technology, Kanpur Kanpur: India.

- [20] Colle, D., dkk. (2011). *Enabling fast failure recovery in OpenFlow networks*. International Workshop on the Design of Reliable Communication Networks (DRCN).
- [21] Clark, D., Pogran, K., Reed, D. (1978). *An Introduction to Local Area Networks*. Proceedings of The IEEE, Vol. 66, No. 11.
- [22] Simasundaram, Ravishankar. (2013). *Git: Version control for everyone*. Packt Publishing Ltd: Birmingham.
- [23] Kiran, R. & Mamatha, C. (2018). *Implementation of DevOps Architecture in the proyek development and deployment with help of tools*. International Journal of Scientific Research in Computer Science and Engineering Vol. 6. No.87-95.
- [24] Lwakatare, L., dkk. (2016). *DevOps Adoption Benefits and Challenges in Practice: A Case Study*. DevOps AdoLecture Notes in Computer Science book series No. 590-597.
- [25] Ashari, A. & Setiawan, H. (2011). *Cloud Computing : Solusi ICT ?*. Jurnal Sistem Infomasi Vol. 2, No.3.

LAMPIRAN

Lampiran 1. Guideline pengerjaan proyek

Guideline pengerjaan proyek menggunakan tools *Docker*, *Gitlab*, dan *Dokku* dengan konsep DevOps .

1. *Developer* melakukan *pull source code* menggunakan *Gitlab*.
2. *Developer* membuat *code* proyek, kemudian membuat *Dockerfile*, yang akan mengubah hasil *code* proyek, menjadi *Docker image*, *Dockerfile* adalah dokumen teks yang berisi semua perintah yang dapat dipanggil *user* pada baris perintah untuk merakit *image* yang baru dan berisikan *code* yang dibutuhkan agar sebuah proyek dapat berjalan, seperti perintah untuk menginstal *nginx*, *php*, dan software lainnya, yang dibutuhkan sebuah proyek untuk berjalan.
3. Ketika proyek dirasa sudah cukup dan berjalan dengan baik, *Dockerfile* dan *file* untuk proyek disimpan ke dalam *Gitlab* yang berjalan di jaringan LAN Institut Teknologi Del. Ketika *user* mengubah atau melakukan *update* sesuatu di dalam *repository*, maka *log* atau *history* nya secara otomatis akan tercatat.
4. Kemudian, jika ingin sekaligus *deploy* proyek tersebut ke *Dokku*, CI/CD dapat digunakan dengan perintah :

```
# git remote add origin 192.168.1.15:8969/lambok/ta2.git(sesuai link directory).  
# git push origin master
```

5. Kemudian *dokku* akan meng-*update* kembali proyek sesuai lokasi project *gitlab*.
6. Apabila proyek ingin diperbaharui, *user* dapat kembali mengambil hasil *code* proyek dari *Gitlab*, dan mengulangi dari step 1, sehingga akan menciptakan *life cycle* DevOps yang terkesan berulang dalam pengerjaan proyek.

Lampiran 2. Panduan Memulai Git

Untuk mengkonfigurasi *Git* agar dapat bekerja dan berjalan, jalankan perintah berikut:

```
# git config --global user.name
```

Langkah tersebut akan mencetak hasil kosong, itu karena tepat setelah *username* tidak dikonfigurasi. Untuk mengatur konfigurasi *username* gunakan perintah berikut :

```
$ git config --global user.name "Lambok Silitonga"
```

Ganti Lambok Silitonga dengan mengetik nama *User*. Selanjutnya, jalankan perintah :

```
$ git config --global user.email "lamboksilitonga@example.com"
```

Perintah ini akan mengatur *email user*. Jika ingin membuat *commit* dalam repositori *Git*, *user* harus mengkonfigurasi dua pengaturan : *user.name* dan *user.email* . Jika tidak, ketika menjalankan perintah `$ Git commit` , *Git* akan mencetak peringatan. *string* yang digunakan seperti nilai *user.name* dan *user.email* akan disimpan dalam setiap *commit* yang dibuat.

Lampiran 3. Panduan Membuat *Dockerfile*

Pada penggunaan pembangunan *Dockerfile*, terdapat hal-hal penting yang harus diperhatikan :

1. *Source code*

Code dari sebuah proyek merupakan hal mendasar yang diperlukan, agar nantinya *source code* tersebut diubah menjadi *Docker image*, dengan cara menggabungkannya dengan *Dockerfile*.

2. *Dockerfile*

Setelah proyek yang dijalankan cukup telah memenuhi, maka selanjutnya adalah membuat *Dockerfile*. Ada beberapa hal penting yang diperlukan dalam membuat *Dockerfile* :

a. *Dockerfile* diciptakan dengan menggunakan *YAML Language*, *YAML Language* adalah bahasa pemrograman yang tergolong mudah dibaca manusia secara langsung, dalam penggunaannya, hal sensitif yang perlu diperhatikan adalah, tidak diizinkan menggunakan *tab*, dan kesejajaran antara program dari atas kebawah menyatakan jika program tersebut memiliki kesetaraan. Sebagai contoh:

```
1 ---
2 #Blog about YAML
3
4 title: YAML Ain't Markup Language
5 author:
6   first_name: Lauren
7   last_name: Malhoit
8   twitter: "@Malhoit"
9 learn:
10  - Basic Data Structures
11  - Commenting
12  - When and How
```

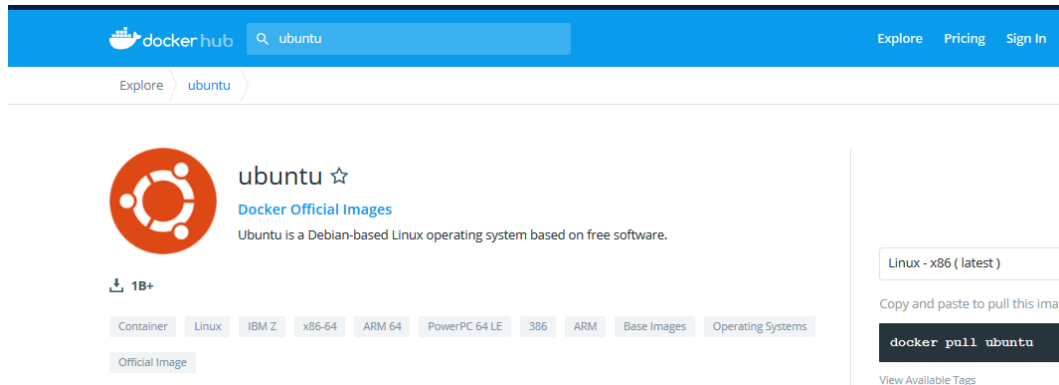
Dari gambar, 'title' dan 'author' memiliki kesetaraan, tetapi apabila 'author' dan 'first_name' merupakan hubungan antar cabang, dimana 'first_name' merupakan bagian dari 'author', tetapi 'title' bukan bagian dari 'first_name'.

Selanjutnya, yang perlu diperhatikan dalam pembuatan *Dockerfile* adalah, beberapa *command* dalam *YAML Language*, beberapa hal penting yang perlu diperhatikan dalam menciptakan *Dockerfile* disebutkan sebagai berikut :

b. 'FROM'

```
| FROM nginx:alpine
```

Script command 'FROM', adalah merupakan salah satu *command* yang wajib disematkan dalam penciptaan *Dockerfile*, karena dari *command* inilah yang akan menjadi landasan *Docker* untuk menjalankan sebuah proyek *Docker*, karena sebuah web membutuhkan wadah, maka 'nginx' merupakan salah satu dasar *image Docker* yang dapat digunakan.



Pada gambar diatas, terdapat *Docker image* dengan nama ‘ubuntu’, OS juga tersedia untuk digunakan dalam membangun sebuah proyek, *command* ‘FROM’ akan mencari *Docker image* sesuai script, apabila *Docker image* tidak tersedia pada local komputer, maka secara otomatis *Docker* akan mengunduh *Docker image* yang berasal dari *Docker official registry*.

c. COPY

```
COPY default.conf /etc/nginx/conf.d/default.conf
COPY index.html /usr/share/nginx/html/index.html
```

Perintah *COPY* merupakan salah satu perintah dalam *Dockerfile*, tujuannya adalah untuk mengcopy *file* lain kedalam *Dockerfile* yang akan diubah menjadi *Docker image*, apabila ingin memperbaharui *source code* dari *file* yang telah diubah menjadi *Docker container*, maka *user* perlu masuk ke dalam *Docker container* tersebut, dengan cara melakukan *Docker CLI* yaitu :

```
#Docker exec -it bash (nama_Dockercontainer)
```

Kemudian mencari *file* dan mengubah *file* yang ada.

d. EXPOSE

```
EXPOSE 80
```


EXPOSE merupakan salah satu perintah dalam *Dockerfile* untuk membuka port, dalam konfigurasi sebelumnya, *port* yang dibuka adalah 80, sehingga hasil dari proyek tersebut dapat diakses oleh *user* lain.

e. *RUN*

```
RUN apt-get update -qq && apt-get install -y build-essential  
libmariadb-dev nodejs
```

```
RUN mkdir /myapp
```

Seperti OS lainnya, *Docker* juga memungkinkan perintah seperti CLI pada umumnya, tergantung dari OS *image* yang digunakan.

f. *WORKDIR*

```
WORKDIR /myapp
```

WORKDIR merupakan *command* yang digunakan untuk sebagai *directory* tempat data dari *Docker image* disimpan, sehingga apabila terdapat kesalahan, seperti *Docker* akan di *restart*, maka data dari *Docker* tidak hilang, istilah ini dalam *Docker* dikenal dengan *persistent*.

Lampiran 4. Panduan Perintah *CLI* dalam penggunaan *Docker Engine*

Pada penggunaan nya, *Docker* memanfaatkan CLI (*Command Line Interface*) dan tidak tersedianya *Interface*.

a. Melihat *Docker image* yang tersedia pada lokal komputer

Docker image memiliki fungsi untuk menyediakan *Docker container*, dengan sebuah *Docker image user* dapat membuat banyak *Docker container*.

```
# docker image
```

b. Mengunduh *image*

Dalam penggunaannya, *Docker image* rata-rata berukuran besar, dengan memanfaatkan jaringan LAN, *Docker image* yang tersedia secara global di internet, dapat juga disimpan dengan *Docker registry*, yaitu *Docker repository* yang bersifat *private*, sehingga dengan memanfaatkan jaringan LAN.

```
# docker pull $namaimage
```

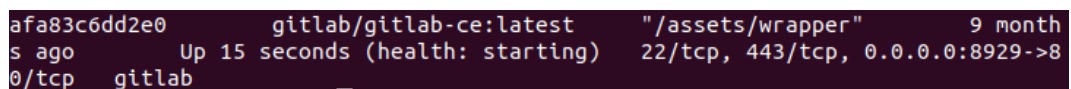
c. Menjalankan *Docker image*.

Dalam penggunaannya, *Docker image* yang dijalankan, disebut dengan *Docker container*, terdapat beberapa bantuan *option* ketika menjalankan *Docker image*, Semua *option* dapat dilihat dengan menggunakan `$Docker -h`, tetapi dalam penggunaannya, terdapat beberapa *option* yang sering atau wajib digunakan.

```
# docker run -d -n $namacontainer <parameter tambahan> $namaimage
```

Sesuai kebutuhan, parameter dapat disesuaikan, parameter dapat dilihat dengan *option* `-h` seperti yang telah disebutkan sebelumnya.

d. Melihat *Docker container* yang berjalan



```
afa83c6dd2e0      gitlab/gitlab-ce:latest      "/assets/wrapper"      9 month
ago             Up 15 seconds (health: starting)  22/tcp, 443/tcp, 0.0.0.0:8929->8
0/tcp          gitlab
```

Agar melihat *Docker* yang berjalan dapat menggunakan CLI :

```
#docker ps
```

Dari gambar yang terlihat, kumpulan huruf dan angka yang pertama adalah nama *container*, nama tersebut adalah jika tidak menggunakan parameter `-n` seperti yang telah dijelaskan sebelumnya, beberapa kata setelahnya dapat dilihat informasi dari *container* yang berjalan, seperti pada gambar, yang terlihat adalah nama dari *Docker image* yang dijalankan, atau berapa lama *Docker* tersebut telah dijalankan,

dan *port* apa saja yang dapat digunakan untuk mengakses *Docker image* tersebut, terdapat berbagai informasi tambahan.

e. Masuk kedalam *container* yang sedang berjalan.

Ketika dibutuhkan konfigurasi pada *Docker image*, tetapi ketika *Docker image* tersebut berjalan tidak mungkin untuk menghentikannya terlebih dahulu, dikarenakan akan mengakibatkan sistem yang berjalan akan mati, *Docker* menyediakan cara agar permasalahan tersebut dapat diselesaikan tanpa menghentikan *Docker container* yang sedang berjalan.

```
> docker exec -it php72 /bin/sh
/var/www/html # ls /
bin      etc      lib      mnt      root    sbin     sys      usr
dev      home    media    proc     run      srv      tmp      var
/var/www/html #
```

```
# docker exec -it <nama_container> /bin/sh
```

Exec bertujuan untuk menjalankan *command* pada *Docker container* yang sedang berjalan.

f. Menghentikan *Docker image* yang sedang berjalan

Ketika *Docker container* yang sedang berjalan dapat dihentikan dengan

```
# docker container stop <nama_container>
```

g. Menghapus *Docker container* yang telah di-*stop*

Docker container yang telah distop/diberhentikan, maka *Docker* tersebut hanya akan berhenti, tetapi *Docker* tersebut tetap berjalan di *background* dan memakan ukuran *disk*, apabila dibiarkan akan menumpuk dan memakan banyak partisi *disk*.

```
# docker ps -a
# docker container rm <nama_container>
```

h. Menghapus *Docker image*

Menghapus *Docker image* dapat dijalankan dengan perintah berikut.

```
# docker rmi <nama_image>
```

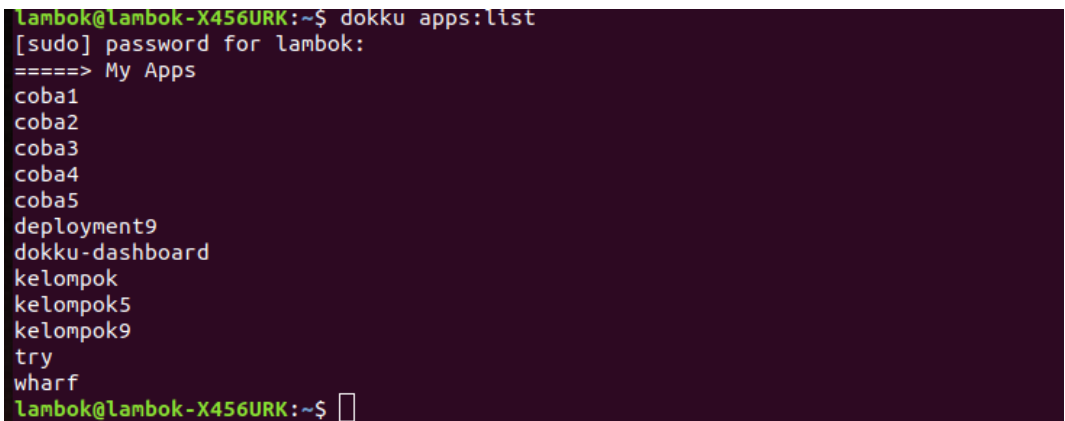
Lampiran 5. Paas (Dokku dan panduan manual dalam *deploy Docker*)

Dokku digunakan sebagai *software* untuk *deployment* ataupun penyebaran dengan proyek yang akan dirilis, tetapi berbeda dengan kebanyakan Paas yang menyediakan perangkat antarmuka, *Dokku* tidak memilikinya, sehingga untuk melakukan aksi *deployment*, maka *developer* memerlukan akses langsung ke *server* dengan cara *ssh*.

Langkah – langkah *deploy Docker* dengan menggunakan *Dokku*:

1. *Dokku* menggunakan yang namanya *apps* yang merupakan semacam cabang atau nama proyek yang nantinya cabang ini yang akan di *deploy* dan bukan *Docker* secara langsung, untuk menglist daftar *apps* dapat menggunakan *command line* dibawah.

```
# dokku apps:list
```



```
Lambok@Lambok-X456URK:~$ dokku apps:list
[sudo] password for lambok:
=====> My Apps
coba1
coba2
coba3
coba4
coba5
deployment9
dokku-dashboard
kelompok
kelompok5
kelompok9
try
wharf
Lambok@Lambok-X456URK:~$
```

2. Jika belum terdapat *apps*, maka perlu diciptakan terlebih dahulu.

```
# dokku apps:create 'nameapps'
```

Lalu untuk melihat kembali daftar *apps* yang telah diciptakan, dapat kembali menggunakan *command*:

```
# dokku apps:list
```

3. Untuk *check* apakah didalam *apps/branch* telah terdapat *Docker image* dapat perintah dengan

```
# dokku tags try
```

```
lambok@lambok-X456URK:~$ dokku tags try
====> Image tags for dokku/try
REPOSITORY      TAG              IMAGE ID          CREATED
SIZE
dokku/try        latest          dd99e98820c2      4 weeks ag
o
```

4. Jika belum terdapat *Docker image* didalam *apps/branch*, langkah selanjutnya adalah dengan memasukkan *Docker image* kedalam *apps/branch* tersebut, dengan cara merename nama *Docker image* tersebut.

```
# dokku tags:create 'namaapps'
```

```
lambok@lambok-X456URK:~$ dokku tags:create try v1
Error response from daemon: No such image: dokku/try:latest
lambok@lambok-X456URK:~$
```

Pada gambar, akan terdapat *output error*, *Dokku* mengharuskan kesesuaian nama antara *apps*, dengan *Docker image* yang akan dimasukkan kedalam *apps*, maka diperlukan perubahan nama pada *Docker image*.

```
lambok@lambok-X456URK:~$ docker tag kelompok5 dokku/try
```

Dan kemudian coba kembali dengan perintah yang sama.

```
lambok@lambok-X456URK:~$ dokku tags:create try v1
====> Added v1 tag to dokku/try
lambok@lambok-X456URK:~$ dokku tags try
====> Image tags for dokku/try
REPOSITORY      TAG              IMAGE ID          CREATED
SIZE
dokku/try        latest          dd99e98820c2      4 weeks ag
o
dokku/try        v1              dd99e98820c2      4 weeks ag
o
```

Maka *Docker image* yang namanya diubah, akan masuk kedalam *apps* tadi.

5. Langkah terakhir adalah *deploy image* tersebut, dapat diketikkan dengan perintah

```
# dokku tags:deploy 'nameapps'
```

Note : pada contoh proyek yang di-*publish*, nama *apps* adalah 'try'

```
lambok@lambok-X456URK:~$ dokku tags:deploy try
----> Setting config vars
      DOKKU_DOCKERFILE_PORTS: 80/tcp
----> Releasing try (dokku/try:latest)...
----> Deploying try (dokku/try:latest)...
----> No Procfile found in app image
----> DOKKU_SCALE file not found in app image. Generating one based on
      Procfile...
      DOKKU_SCALE declares scale -> web=1
====> Processing deployment checks
      No CHECKS file found. Simple container checks will be performed

      For more efficient zero downtime deployments, create a CHECKS file. See http://dokku.viewdocs.io/dokku/deployment/zero-downtime-deployments/ for examples
----> Attempting pre-flight checks (web.1)
      Waiting for 10 seconds ...
      Default container check successful!
----> Running post-deploy
----> Creating new /home/dokku/try/VHOST...
----> Configuring try.dokku.del.com...(using built-in template)
----> Creating http nginx.conf
      Reloading nginx
----> Renaming containers
      Renaming container (93f741291703) eloquent_fermat to try.web.1
====> Application deployed:
```

Gambar diatas merupakan progress *deployment* yang dilakukan oleh *Dokku*.

```
lambok@lambok-X456URK:~$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
93f741291703	dokku/try:latest	"nginx -g 'daemon of..."	About a minute ago	Up About a minute	80/tcp	try.web.1

Dari gambar diatas, terlihat bahwa proyek sebelumnya telah dibangun berdasarkan *Docker* telah berjalan di *background* server, karena tidak memiliki *domain*, maka pada Tugas akhir ini dilakukan pengujian dengan mengakses IP *server*.

Panduan mengunduh *Docker image* dari *private Docker Registry*

1. Agar *pull* dari *Server*, maka tambahkan sebuah *file/etc/Docker/daemon.json* pada lokal komputer.

```
# {  
  "insecure-registries" : ["192.168.1.15:5000"]  
}
```

2. Sesuaikan IP dalam *file* tersebut sesuai dengan IP server dimana *Docker registry* berada. *Restart service Docker* pada lokal komputer dengan perintah:

```
# sudo systemctl restart Docker.service
```

3. Kemudian, lakukan unduh *Docker image* dengan perintah :

```
# sudo docker pull 192.168.1.15:5000/alpine:latest
```

Sesuaikan IP sesuai *IP Docker Registry* berada.

Lampiran *sourcecode website PHP*

```
# /myapp/styles.css  
  
@font-face {  
    font-family: "Kendo UI";  
    src:  
url("http://cdn.kendostatic.com/2013.3.1030/styles/images/kendoui.woff")  
format("woff"),  
    url("http://cdn.kendostatic.com/2013.3.1030/styles/images/kendoui.ttf")  
format("truetype");  
}
```

```
body:before  
  
{  
    font-family: "Kendo UI";  
    content: "\a0";  
    font-size: 0;  
    width: 0;  
    height: 0;  
    position: absolute;  
    z-index: -1;  
}
```

```
html, body {  
    margin: 0;  
    padding: 0;  
    min-width: 320px;  
}
```

```
#example {  
    padding-top: 2em;  
}
```

```
header {  
    margin: 0;  
    font-size: 20px;
```



```
color: #fff;

background-color: #272727;

}
```

```
header h1,

header button {

    display: inline-block;

    vertical-align: middle;

}
```

```
header .container h1 {

    margin: 0;

    font-size: 21px;

    line-height: 65px;

    padding-top: 20px;

    padding-left: 0;

}
```

```
#configure,

header label {

    border-style: solid;

    border-color: #636363;

    border-width: 0 0 0 1px;

    margin-bottom: 0;
```

```
font-weight: normal;
}

header .container {
    position: relative;
}

header .description {
    text-transform: uppercase;
    color: #ccc;
    font-size: 10px;
    line-height: 29px;
}

header .k-dropdown,
header #font-size-value {
    font-size: 18px;
    line-height: 45px;
    margin-bottom: 9px;
}

header .k-dropdown {
    width: 100%;
}
```

```
header .k-dropdown-wrap {  
    background-image: none !important;  
}  
  
header .k-dropdown,  
header .k-dropdown-wrap.k-state-default {  
    background-color: transparent;  
    border-color: #5c5c5c;  
    border-radius: 0;  
}  
  
header .k-dropdown-wrap.k-state-hover,  
header .k-dropdown-wrap.k-state-active,  
header .k-dropdown-wrap.k-state-focused {  
    background-color: #030303;  
}  
  
header .k-dropdown-wrap.k-state-focused {  
    box-shadow: none;  
}  
  
header .k-dropdown .k-input {  
    padding-top: 0;
```

```

padding-bottom: 0;

height: auto;

text-indent: 14px;
}

header .k-dropdown .k-input,
header .k-dropdown-wrap .k-select {

    line-height: 45px;
}

header .k-dropdown .k-i-arrow-s {

    background-image:
url("http://cdn.kendostatic.com/2013.2.918/styles/Bootstrap/sprite.png");

    background-position: -16px -32px;
}

header .k-dropdown-wrap.k-state-default .k-input {

    color: #fff;
}

.k-popup.ra-list {

    background-color: #010101;

    border-color: #5c5c5c;

    color: #fff;

```

```
padding: 0;

border-radius: 0;

}

.k-popup.ra-list .k-state-hover,
.k-popup.ra-list .k-state-selected,
.k-popup.ra-list .k-state-focused {

    background-image: none;

    background-color: #555;

    box-shadow: none;

    color: #fff;

    border-color: #555;

}

.k-popup.ra-list .k-item {

    border-radius: 0;

    text-indent: 7px;

}

#configure {

    z-index: 10;

    width: 50px;

    height: 50px;

    overflow: hidden;
```

```
border-width: 0 0 0 1px;

background-color: transparent;

position: absolute;

top: 0; right: 0;

color: #fff;

}

#demo {

padding-top: 46px;

}

#menu {

margin-bottom: 30px;

}

#profile {

position: relative;

}

.ra-well-title {

font-size: 1.2857em;

line-height: 1.2857em;

border-bottom: 1px solid #e7e7e7;

margin: -5px -19px 0.8333em;
```

```
padding: 0 19px 0.7222em;
}

.ra-avatar {
  border: 1px solid #e7e7e7;
  border-radius: 2px;
}

.ra-first-name {
  display: block;
  margin-top: 0.8571em;
}

.ra-last-name {
  display: block;
  font-size: 1.7143em;
  line-height: 1.3em;
}

.ra-position {
  font-size: 0.8571em;
  color: #999;
  padding-bottom: 2em;
}
```

```
.form-group .k-widget,  
.form-group .k-textbox {  
    width: 100%;  
}  
  
.buttons-wrap {  
    border-top: 1px solid #e7e7e7;  
    padding-top: .5em;  
    text-align: right;  
}  
  
.ra-section {  
    margin-bottom: 20px;  
}  
  
.ra-well-overlay {  
    margin: -16px -20px -19px;  
}  
  
#tabstrip .k-content {  
    min-height: 156px;  
}
```



```

#tabstrip .k-chart {
    height: 156px;
}

#tabstrip .k-content {
    padding: 1px;
}

#tabstrip-4 {
    text-align: center;
}

#tabstrip .km-icon:after {
    font: 1.3em/1em "Kendo UI" !important;
}

.revenue:after { content: "\E08C"; }
.spd:after { content: "\E04B"; }
.spr:after { content: "\E050"; }
.share:after { content: "\E04E"; }

#tabstrip .k-tabstrip-items span {
    float: left;
    line-height: 1.3em;
}

```

```
vertical-align: middle;

}

#tabstrip .k-tabstrip-items .hidden-xs {

    margin-left: 4px;

}

.market-donut {

    display: inline-block;

    width: 170px;

}

#panelbar .k-content {

    padding: 1em;

}

#panelbar ul {

    margin-bottom: 10px;

}

#listview {

    list-style-type: none;

    padding: 0 0 15px;

}
```

```
#listview figure {  
    border: 1px solid #e7e7e7;  
    border-radius: 3px;  
    padding: 5px;  
    margin-top: 15px;  
}  
  
figure h4 {  
    font-size: 1.15em;  
}  
  
figure p.hidden-sm {  
    min-height: 80px;  
}  
  
footer {  
    text-align: right;  
    font-size: 0.8571em;  
    padding: 2em 0;  
}  
  
header .container h1.visible-sm {  
    padding-top: 10px;
```

```
line-height: 33px;

padding-bottom: 10px;
}

@media (max-width: 767px) {

.k-menu.k-menu-horizontal .k-item {

    float: none;
}

header .container {

    padding-right: 0;
}

header .container h1 {

    padding-top: 0;

    line-height: 50px;
}

header label {

    border-width: 1px 0 0 0;

    display: block;
}

#configurator-wrap {
```

```
position: absolute;

overflow: hidden;

height: 260px;

right: 0;

top: 50px;

z-index: 1000;

padding: 0;

width: 320px;

}

#configurator {

background-color: #272727;

position: absolute;

width: 100%;

margin: 0;

}

}

.k-black body,

.k-metroblack body,

.k-black .well,

.k-metroblack .well,

.k-black .ra-well-title,

.k-metroblack .ra-well-title,
```

```
.k-black .buttons-wrap,  
.k-metroblack .buttons-wrap  
{  
    border-color: #444;  
    background-color: #1e1e1e;  
    color: #fff;  
}  
  
.k-highcontrast body,  
.k-highcontrast .well  
.k-highcontrast .ra-well-title,  
.k-highcontrast .buttons-wrap  
{  
    border-color: #664e62;  
    background-color: #2c232b;  
    color: #fff;  
}  
  
.k-moonlight body,  
.k-moonlight .well,  
.k-moonlight .ra-well-title,  
.k-moonlight .buttons-wrap  
{  
    border-color: #171e28;
```

```

background-color: #212a33;

color: #fff;

}

#menu > li:nth-child(2) > a:nth-child(1){

    display: none;

}

#configurator span.k-input{

    height: auto;

}

```

```

# /myapp/theme-choser.js

(function() {

    function cookie(key, value, end, path, domain, secure) {

        if (arguments.length === 1) {

            return decodeURIComponent(document.cookie.replace(new
RegExp("(?:(?:^|.*;)\\s*" + encodeURIComponent(key).replace(/[\\-\\.\\+\\*]/g,
"\\$&") + "\\s*\\|=\\s*([^;]*).*\\$|^.*$"), "$1")) || null;

        }

        if (!key || /^(?:expires|max\\-age|path|domain|secure)$/i.test(key)) { return
false; }

        var expires = "";

```

```

    if (end) {
        switch (end.constructor) {

            case Number:

                expires = end === Infinity ? "; expires=Fri, 31 Dec 9999 23:59:59
GMT" : "; max-age=" + end;

                break;

            case String:

                expires = "; expires=" + end;

                break;

            case Date:

                expires = "; expires=" + end.toUTCString();

                break;

        }
    }

    document.cookie    =    encodeURIComponent(key)    +    "="    +
encodeURIComponent(value) + expires + (domain ? "; domain=" + domain : "")
+ (path ? "; path=" + path : "") + (secure ? "; secure" : "");

    return true;
}

$(document).ready(function() {

    ThemeChooser.publishTheme(window.kendoTheme);

});

var doc = document,

```



```

    extend = $.extend,

    proxy = $.proxy,

    kendo = window.kendo,

    animation = {

        show: {

            effects: "fadeIn",

            duration: 300

        },

        hide: {

            effects: "fadeOut",

            duration: 300

        }

    },

    skinRegex = /kendo\.[\w\-\.]+\(\.min\)?\.(less|css)/i,

    dvSkinRegex = /kendo\.dataviz\.(?!min)\w+?\(\.css|\.min.css)/gi;

var Details = kendo.ui.Widget.extend({

    init: function(element, options) {

        kendo.ui.Widget.fn.init.call(this, element, options);

        this._summary = this.element.find(".tc-activator")

            .on("click", proxy(this.toggle, this));

        this._container = kendo.wrap(this._summary.next(), true);
    }
});

```

```

        this._container.css("display", "none");
    },
    options: {
        name: "Details"
    },
    toggle: function() {
        var options = this.options;

        var show = this._container.is(":visible");

        var animation = kendo.fx(this._container).expand("vertical");

        animation.stop()[show ? "reverse" : "play"]();

        this._summary.toggleClass("tc-active", !show);
    }
});

kendo.ui.plugin(Details);

var ThemeChooser = kendo.ui.ListView.extend({
    init: function(element, options) {
        kendo.ui.ListView.fn.init.call(this, element, options);

        this.bind("change", this._changeCss);
    }
});

```

```

    },
    options: {
        name: "ThemeChooser",
        template: "",
        selectable: true,
        value: ""
    },
    dataItem: function(element) {
        var uid = $(element).closest("[data-uid]").attr("data-uid");
        return this.dataSource.getByUid(uid);
    },
    _changeCss: function(e) {
        // make the item available to event listeners
        e.item = this.dataItem(this.select());
    },
    value: function(value) {
        if (!arguments.length) {
            var dataItem = this.dataItem(this.select());
            return dataItem.name;
        }

        var data = this.dataSource.data();

        for (var i = 0; i < data.length; i++) {

```

```

        if (data[i].value == value) {

            this.select(this.element.find("[data-uid='" + data[i].uid + '""));

            break;

        }

    }

}

});

```

```

var MOBILE_CLASSES = "km-ios km-ios4 km-ios5 km-ios6 km-ios7 km-
android km-android-dark km-android-light km-blackberry km-wp km-wp-dark
km-wp-light km-flat";

```

```

var ThemeChooserViewModel = kendo.observable({

    themes: [

        { value: "default", name: "Default", colors: [ "#ef6f1c", "#e24b17",
"#5a4b43" ] },

        { value: "blueopal", name: "Blue Opal", colors: [ "#076186", "#7ed3f6",
"#94c0d2" ] },

        { value: "bootstrap", name: "Bootstrap", colors: [ "#3276b1", "#67afe9",
"#fff" ] },

        { value: "silver", name: "Silver", colors: [ "#298bc8", "#515967",
"#eaeaec" ] },

        { value: "uniform", name: "Uniform", colors: [ "#666", "#ccc", "#fff" ] },

        { value: "metro", name: "Metro", colors: [ "#8ebc00", "#787878", "#fff"
] },

    ] },

```

```

        { value: "black", name: "Black", colors: [ "#0167cc", "#4698e9",
"#272727" ] },

        { value: "metroblack", name: "Metro Black", colors: [ "#00aba9",
"#0e0e0e", "#565656" ] },

        { value: "highcontrast", name: "High Contrast", colors: [ "#b11e9c",
"#880275", "#1b141a" ] },

        { value: "moonlight", name: "Moonlight", colors: [ "#ee9f05", "#40444f",
"#212a33" ] },

        { value: "flat", name: "Flat", colors: [ "#363940", "#2eb3a6", "#fff" ] }
    ],

    mobileThemes: [

        { name: "iOS7", value:"ios7", colors: [ "#007aff", "#f5f5f5", "#ffffff" ]
    },

        { name: "iOS6", value: "ios", colors: [ "#4a86ec", "#6982a3", "#c3ccd5"
    ] },

        { name: "Android Light", value: "android-light", colors: [ "#33b5e5",
"#cacaca", "#fcfcfc" ] },

        { name: "Android Dark", value: "android-dark", colors: [ "#33b5e5",
"#000000", "#4c4c4c" ] },

        { name: "BlackBerry", value: "blackberry", colors: [ "#357fad",
"#d9d9d9", "#ffffff" ] },

        { name: "WP8 Light", value: "wp-light", colors: [ "#01abaa", "#000000",
"#ffffff" ] },

        { name: "WP8 Dark", value: "wp-dark", colors: [ "#01abaa", "#ffffff",
"#000000" ] },

        { name: "Flat Skin", value: "flat", colors: [ "#10c4b2", "#dcdcdc",
"#f4f4f4" ] }
    ]

```

```

],

sizes: [

    { name: "Standard", value: "common" },

    { name: "Bootstrap", value: "common-bootstrap", relativity: "larger" }

],

selectedTheme: window.kendoTheme,

selectedMobileTheme: window.kendoMobileTheme,

selectedSize: window.kendoCommonFile,

updateMobileTheme: function(e) {

    var that = this;

    setTimeout(function () { that.setMobileTheme(e.item.value) }, 0);

},

updateTheme: function(e) {

    var themeName = e.item.value;

    ThemeChooser.changeTheme(themeName, true);

},

updateCommon: function(e) {

    ThemeChooser.changeCommon(e.item.value, true);

    cookie("commonFile", e.item.value, Infinity, "/");

```

```

    },

    setMobileTheme: function(themeName) {

        var mobileContainer = $("#mobile-application-container");

        mobileContainer.removeClass(MOBILE_CLASSES).addClass("km-" +
themeName + (" km-" + themeName.replace(/-./, "")));

        $("#device-wrapper").removeClass("ios7 ios wp-dark wp-light android-
light android-dark blackberry flat").addClass(themeName);

        cookie("mobileTheme", themeName, Infinity, "/");

        kendo.resize(mobileContainer);

    }

});

kendo.ui.plugin(ThemeChooser);

window.ThemeChooserViewModel = ThemeChooserViewModel;

$(document).ready(function() {

    kendo.bind($(".themechooser"), ThemeChooserViewModel);

});

extend(ThemeChooser, {

    preloadStylesheet: function (file, callback) {

        var element = $("<link rel='stylesheet' media='print' href='" + file + "'
/>").appendTo("head");

```

```

        setTimeout(function () {

            callback();

            element.remove();

        }, 100);

    },

    getCurrentCommonLink: function () {

        return $("head link").filter(function () {

            return (/kendo\.common/gi).test(this.href);

        });

    },

    getCurrentThemeLink: function () {

        return $("head link").filter(function () {

            return (/kendo\./gi).test(this.href)

            &&
            !(/common|rtl|dataviz|mobile/gi).test(this.href);

        });

    },

    getCurrentMobileThemeLink: function () {

        return $("head link").filter(function () {

            return (/kendo\[^\.\.\/\]+?.mobile/gi).test(this.href)

            &&
            !(/common|rtl|dataviz/gi).test(this.href);

        });

    },

```



```

    },

    getCurrentDVThemeLink: function () {
        return $("head link").filter(function () {
            return dvSkinRegex.test(this.href);
        });
    },

    getCommonUrl: function (common) {
        var currentCommonUrl =
ThemeChooser.getCurrentCommonLink().attr("href");

        return currentCommonUrl.replace(skinRegex, "kendo." + common +
"$1.$2");
    },

    getThemeUrl: function (themeName) {
        var currentThemeUrl =
ThemeChooser.getCurrentThemeLink().attr("href");

        return currentThemeUrl.replace(skinRegex, "kendo." + themeName +
"$1.$2");
    },

    getDVThemeUrl: function (themeName) {

```

```

        var currentThemeUrl =
ThemeChooser.getCurrentDVThemeLink().attr("href");

        if (currentThemeUrl) {

            return currentThemeUrl.replace(dvSkinRegex, "kendo.dataviz." +
themeName + "$1");

        }

    },

    replaceCommon: function(commonName) {

        var newCommonUrl = ThemeChooser.getCommonUrl(commonName),

            themeLink = ThemeChooser.getCurrentCommonLink();

        ThemeChooser.updateLink(themeLink, newCommonUrl);

    },

    replaceWebTheme: function (themeName) {

        var newThemeUrl = ThemeChooser.getThemeUrl(themeName),

            oldThemeName = $(doc).data("kendoSkin"),

            themeLink = ThemeChooser.getCurrentThemeLink();

        ThemeChooser.updateLink(themeLink, newThemeUrl);

        $(doc.documentElement).removeClass("k-" +
oldThemeName).addClass("k-" + themeName);

    },

```

```

replaceWebMobileTheme: function (themeName) {

    var newThemeUrl = ThemeChooser.getThemeUrl(themeName +
".mobile"),

    themeLink = ThemeChooser.getCurrentMobileThemeLink();

    ThemeChooser.updateLink(themeLink, newThemeUrl);

},

replaceDVTheme: function (themeName) {

    var newThemeUrl = ThemeChooser.getDVThemeUrl(themeName),

    themeLink = ThemeChooser.getCurrentDVThemeLink();

    if (newThemeUrl) {

        ThemeChooser.updateLink(themeLink, newThemeUrl);

    }

},

updateLink: function(link, url) {

    var newLink,

    exampleElement = $("#example"),

    less = window.less,

    isLess = /\.less$/ .test(link.attr("href"));

```

```

        if (kendo.support.browser.msie && kendo.support.browser.version < 11)
        {
            newLink = $(doc.createStyleSheet(url));
        } else {
            newLink = link.eq(0).clone().attr("href", url);
            link.eq(0).before(newLink);
        }

        link.remove();

        if (isLess) {
            $("head style[id^='less']").remove();

            less.sheets = $("head link[href$='.less']").map(function () {
                return this;
            });

            less.refresh(true);
        }

        if (exampleElement.length) {
            exampleElement[0].style.cssText = exampleElement[0].style.cssText;
        }
    },

```

```

replaceTheme: function(themeName) {

    ThemeChooser.replaceWebTheme(themeName);

    ThemeChooser.replaceWebMobileTheme(themeName);

    ThemeChooser.replaceDVTheme(themeName);

    ThemeChooser.publishTheme(themeName);

    cookie("theme", themeName, Infinity, "/");

},

publishTheme: function (themeName) {

    var themable = ["Chart", "TreeMap", "Diagram", "StockChart",
"Sparkline", "RadialGauge", "LinearGauge"];

    if (kendo.dataviz && themeName) {

        for (var i = 0; i < themable.length; i++) {

            var widget = kendo.dataviz.ui[themable[i]];

            if (widget) {

                widget.fn.options.theme = themeName;

            }

        }

    }

    if (themeName) {

```

```

$(doc).data("kendoSkin", themeName);

}

$("#example").trigger("kendo:skinChange");

},

animateCssChange: function(options) {

    options = $.extend({ complete: $.noop, replace: $.noop }, options);

    if (options.prefetch == options.link.attr("href")) {

        return;

    }

    ThemeChooser.preloadStylesheet(options.prefetch, function () {

        var example = $("#example");

        example.kendoStop().kendoAnimate(extend({}, animation.hide, {

            complete: function (element) {

                if (element[0] == example[0]) {

                    example.css("visibility", "hidden");

                    options.replace();

                    setTimeout(function () {

```

```

        example

        .css("visibility", "visible")

        .kendoStop()

        .kendoAnimate(animation.show);

        options.complete();

    }, 100);

    }

    }

    ));

    });

    },

    changeCommon: function(commonName, animate) {

        ThemeChooser.animateCssChange({

            prefetch: ThemeChooser.getCommonUrl(commonName),

            link: ThemeChooser.getCurrentCommonLink(),

            replace: function() {

                ThemeChooser.replaceCommon(commonName);

            }

        });

    },

    changeTheme: function(themeName, animate, complete) {

```

```

        // Set transparent background to the chart area.

        extend(kendo.dataviz.ui.themes[themeName].chart, { chartArea: {
background: "transparent"} });

        if (animate) {
            ThemeChooser.animateCssChange({
                prefetch: ThemeChooser.getThemeUrl(themeName),
                link: ThemeChooser.getCurrentThemeLink(),
                replace: function() {
                    ThemeChooser.replaceTheme(themeName);
                },
                complete: complete
            });
        } else {
            ThemeChooser.replaceTheme(themeName);
        }
    }
});

window.kendoThemeChooser = ThemeChooser;
})();

```