

LAPORAN TUGAS AKHIR

PERANCANGAN DOCKER PADA INFRASTRUKTUR DEVOPS UNTUK PROSES *DEPLOYMENT* APLIKASI BERBASIS WEB

Disusun untuk memenuhi persyaratan memperoleh gelar Sarjana Teknik
di Jurusan Teknik Elektro Universitas Jenderal Soedirman



Disusun oleh:

Ahmad Nadzir Romadhon
H1A015064

**KEMENTERIAN RISET, TEKNOLOGI DAN PENDIDIKAN TINGGI
UNIVERSITAS JENDERAL SOEDIRMAN
FAKULTAS TEKNIK
JURUSAN/PROGRAM STUDI TEKNIK ELEKTRO
PURBALINGGA
2019**



LAPORAN TUGAS AKHIR

PERANCANGAN DOCKER PADA INFRASTRUKTUR DEVOPS UNTUK PROSES *DEPLOYMENT* APLIKASI BERBASIS WEB

Disusun untuk memenuhi persyaratan memperoleh gelar Sarjana
Teknik
di Jurusan Teknik Elektro Universitas Jenderal Soedirman



Disusun oleh:

Ahmad Nadzir Romadhon
H1A015064

**KEMENTERIAN RISET, TEKNOLOGI DAN PENDIDIKAN
TINGGI
UNIVERSITAS JENDERAL SOEDIRMAN
FAKULTAS TEKNIK
JURUSAN/PROGRAM STUDI TEKNIK ELEKTRO
PURBALINGGA
2019**

HALAMAN PENGESAHAN

Tugas Akhir dengan Judul:

PERANCANGAN DOCKER PADA INFRASTRUKTUR DEVOPS UNTUK PROSES *DEPLOYMENT* APLIKASI BERBASIS WEB

Disusun oleh:

Ahmad Nadzir Romadhon
H1A015064

Diajukan untuk memenuhi salah satu persyaratan
memperoleh gelar Sarjana Teknik pada
Jurusan/Program Studi Teknik Elektro
Fakultas Teknik
Universitas Jenderal Soedirman

Diterima dan disetujui

Pada Tanggal : _____

Pembimbing I

Pembimbing II

Acep Taryana, S.Si., M.T.
(NIP 197112152000031001)

Hari Siswantoro S.T., M.T., Ph.D
(NIP 197812242005011002)

Mengetahui:

Dekan Fakultas Teknik

Dr. Eng. Suroso, S.T., M.Eng
NIP 197812242001121002

HALAMAN PERNYATAAN

Dengan ini saya menyatakan bahwa dalam Laporan Tugas Akhir dengan judul ***“Perancangan Docker pada Infrastruktur DevOps Untuk Proses Deployment Aplikasi Berbasis Web”*** ini tidak terdapat karya yang pernah diajukan untuk memperoleh gelar kesarjanaaan di suatu Perguruan Tinggi, dan sepanjang pengetahuan saya juga tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis diacu dalam naskah ini dan disebutkan dalam daftar pustaka.

Purbalingga, 16 Agustus 2019

Ahmad Nadzir Romadhon
NIM. H1A015064

HALAMAN MOTTO DAN PERSEMBAHAN

MOTTO

Never stop living your dream

PERSEMBAHAN

Laporan Tugas Akhir ini dipersembahkan untuk:

1. seluruh mahasiswa Teknik Elektro Unsoed,
2. dosen pembimbing Tugas Akhir, dan
3. siapapun yang mungkin mendapatkan manfaat dari laporan ini.

RINGKASAN

PERANCANGAN DOCKER PADA INFRASTRUKTUR DEVOPS UNTUK PROSES *DEPLOYMENT* APLIKASI BERBASIS WEB

Ahmad Nadzir Romadhon

Secara umum terdapat 2 metode *deployment* aplikasi web ke server, menginstal aplikasi web langsung ke server tunggal dan memanfaatkan teknologi virtualisasi tradisional (berbasis *hypervisor*). Masing-masing metode memiliki kelebihan dan kekurangan. Dependensi dan penggunaan *resource* adalah masalah utama dari 2 metode tersebut.

Virtualisasi berbasis *container* adalah salah satu jenis metode virtualisasi untuk menciptakan sistem yang terisolasi pada level sistem operasi/OS yang dijalankan pada satu kernel. Berbeda dengan *Virtual Machine*, *container* tidak memiliki *hypervisor* dan saling berbagi kernel *host OS*, sehingga lebih ringan untuk dijalankan. Docker adalah salah satu teknologi virtualisasi berbasis *container* yang banyak digunakan saat ini sebagai salah satu *tools* yang mendukung metode *DevOps*.

DevOps adalah sebuah metode baru pada pengembangan aplikasi yang menekankan pada praktik berkelanjutan (*continuous integration, delivery, deployment*). Konsep *DevOps* berkaitan dengan *application development, operations* dan *services*, di mana kolaborasi antara pengembang aplikasi dan operator sangat ditekankan.

Docker adalah salah satu *tools* yang dimanfaatkan untuk proses *deployment* aplikasi pada arsitektur *DevOps*. Tujuan penggunaan *Docker* pada arsitektur *DevOps* adalah menciptakan teknologi komputasi yang ringan, untuk membantu pengembang dalam menempatkan aplikasi dan *environment* ke dalam *resource*/kontainer yang berbeda, sehingga kontainer/*resource* yang dibuat dan dijalankan di bagian pengembang (*Dev*), terlihat sama dengan bagian operator (*Ops*), dan dapat dijalankan di atas beberapa sistem operasi yang berbeda.

Kata kunci : metode *deployment*, virtualisasi berbasis kontainer, *Docker*, *DevOps*

SUMMARY

Docker Implementation on DevOps Architecture For Deploying Web-Based Application

Ahmad Nadzir Romadhon

Generally, there are two methods to deploy a web-based application to the server, first install directly on the single server, second utilize the traditional virtualization technology (hypervisor-based). Each of these methods has an advantage and disadvantage. Dependencies and resource utilization it the main issue.

Container-based virtualization is one of the virtualization methods to build an isolated system on the operating system level which running on the single kernel. Differ with the virtual machine, the container has no hypervisor and sharing host OS kernel instead, hence the system is more lightweight for running. Docker is one of the most popular container-based virtualization technology as a tool that supports DevOps term.

DevOps defined as a new software development methodology that emphasizes the continuous practices (continuous integration, delivery, deployment). The DevOps concept relates to the application development, operation, and services, where the collaboration between apps developer and IT operation are really intended.

Docker is a tool that supports DevOps architecture for deploying the application. The main goal of docker utilization on the DevOps architecture is to build a lightweight computing technology so the developers will be able to put the application and its environment into a different resource/container, so that the container which runs on the Dev stage will be equal on the Ops stage, and be able to run across multi-operating system simultaneously.

Keywords : deployment method, container-based virtualization, Docker, DevOps

PRAKATA

Puji syukur ke hadirat Allah S.W.T. yang telah melimpahkan berkah dan rahmat-Nya laporan Tugas Akhir ini dapat disusun. Terima kasih kami sampaikan kepada seluruh pihak yang telah membantu terwujudnya dokumen ini, di antaranya: Dekan FT Unsoed, Wakil Dekan Akademik FT Unsoed, Kajur Teknik Elektro Unsoed, Sekretaris Jurusan Teknik Elektro Unsoed, bapak-ibu dosen Teknik Elektro Unsoed, teman-teman Teknik Elektro Unsoed 2015 dan pihak-pihak lain yang tidak dapat kami sebutkan satu persatu.

Penulis berharap ilmu serta pengalaman yang didapatkan dapat bermanfaat khususnya bagi keberlanjutan studi penulisan pribadi. Laporan ini dapat penulis selesaikan setelah melangsungkan kegiatan tugas akhir selama 4 bulan di Kampus Teknik UNSOED. Laporan ini pada intinya berisi mengenai perancangan perangkat Docker pada infrastruktur *DevOps* untuk proses *deployment* aplikasi berbasis web.

Penulis menyadari bahwa masih banyak kekurangan-kekurangan dalam penulisan laporan ini, namun demikian semoga Laporan Tugas Akhir ini dapat bermanfaat bagi penulis pada khususnya dan pembaca pada umumnya.

Purbalingga, 16 Agustus 2019

Ahmad Nadzir Romadhon

DAFTAR ISI

HALAMAN PENGESAHAN.....	ii
HALAMAN PERNYATAAN.....	iii
HALAMAN MOTTO DAN PERSEMBAHAN.....	iv
RINGKASAN.....	v
SUMMARY.....	vi
PRAKATA.....	vii
DAFTAR ISI.....	viii
DAFTAR TABEL.....	x
DAFTAR GAMBAR.....	xi
DAFTAR LAMPIRAN.....	xii
DAFTAR PERINTAH.....	xiii
DAFTAR PROGRAM.....	xiv
DAFTAR ISTILAH DAN SINGKATAN.....	xv
BAB 1 PENDAHULUAN.....	1
1.1 Latar Belakang.....	1
1.2 Rumusan Masalah.....	3
1.3 Batasan Masalah.....	3
1.4 Hipotesis.....	4
1.5 Tujuan dan Manfaat.....	4
1.5.1 Tujuan.....	4
1.5.2 Manfaat.....	4
BAB 2 TINJAUAN PUSTAKA.....	6
2.1 Penelitian Terkait dan Produk Sejenis.....	6
2.2 Teknologi Virtualisasi.....	7
2.3 Mesin Virtual.....	8
2.4 Teknologi Virtualisasi Berbasis Kontainer.....	10
2.5 Docker.....	11
2.5.1 Objek-objek Docker.....	13
2.5.2 Arsitektur <i>Docker</i>	14
2.6 DevOps.....	15
BAB 3 METODE PENELITIAN.....	18

3.1	Tempat Penelitian.....	18
3.2	Alat dan Bahan Penelitian.....	18
3.3	Alur dan Tahapan Penelitian.....	18
3.3.1	Tahap Persiapan.....	19
3.3.2	Tahap Perancangan.....	19
3.3.3	Tahap Pengujian.....	20
3.4	Pembuatan Laporan.....	23
3.5	Waktu dan Jadwal Penelitian.....	24
BAB 4 HASIL DAN PEMBAHASAN.....		25
4.1	Instalasi <i>Docker</i>	25
4.1.1	Docker Compose.....	27
4.1.2	DockerFile.....	29
4.2	Integrasi Docker Pada Infrastruktur <i>DevOps</i>	29
4.3	Proses <i>Continuous Deployment</i> Pada Skenario CI/CD.....	33
4.4	Pengujian Kompatibilitas dan Efektivitas <i>Docker</i>	39
4.5	Pengujian Kinerja <i>Docker</i>	42
4.6	Manfaat Penggunaan <i>Docker</i> Pada Infrastruktur DevOps.....	44
4.7	Redeploy WordPress Container.....	47
BAB 5 KESIMPULAN DAN SARAN.....		48
5.1	Kesimpulan.....	48
5.2	Saran.....	49
DAFTAR PUSTAKA.....		50
LAMPIRAN.....		53
BIODATA PENULIS.....		62

DAFTAR TABEL

Tabel 2.1 Tabel perbandingan sistem.....	7
Tabel 3.1 Alat dan bahan penelitian.....	18
Tabel 3.2 Spesifikasi Laptop/Komputer.....	18
Tabel 3.3 Analisis pengujian kompatibilitas dan efektivitas Docker.....	22
Tabel 3.4 Spesifikasi beban kerja pengujian.....	23
Tabel 3.5 Jadwal Penelitian.....	24
Tabel 4.1 Hasil sebelum dan sesudah melakukan update resource list.....	41
Tabel 4.2 Error Request Server HTTP CMS dalam persentase.....	43
Tabel 4.3 Utilitas CPU server (Host) layanan HTTP request dalam persentase.....	44

DAFTAR GAMBAR

Gambar 2.1 Layer teknologi virtualisasi.....	8
Gambar 2.2 Layer teknologi virtualisasi OS.....	9
Gambar 2.3 Perbedaan arsitektur antara teknologi virtualisasi berbasis kontainer dengan virtualisasi tradisional [20].....	10
Gambar 2.4 Ilustrasi pengaturan proses deployment tidak menggunakan docker....	11
Gambar 2.5 Ilustrasi proses deployment menggunakan docker.....	12
Gambar 2.6 Ilustrasi infrastruktur Docker.....	15
Gambar 2.7 Ilustrasi hubungan antara continuous integration, delivery dan deployment[30].....	17
Gambar 3.1 Arsitektur CI/CD untuk proses deployment.....	20
Gambar 3.2 Arsitektur pengujian proses deployment aplikasi.....	21
Gambar 3.3 Skenario pengujian kompatibilitas Docker.....	22
Gambar 4.1 Membuat project-langkah 1.....	30
Gambar 4.2 Membuat project-langkah 2.....	30
Gambar 4.3 Konfigurasi Jenkins-langkah 1.....	31
Gambar 4.4 Konfigurasi Jenkins-repository URL.....	32
Gambar 4.5 Konfigurasi Jenkins-langkah 3.....	32
Gambar 4.6 Konfigurasi Jenkins-build history.....	33
Gambar 4.7 Konfigurasi Jenkins-proses build berhasil.....	33
Gambar 4.8 Push (meminahkan) file.....	36
Gambar 4.9 deployment kontainer ke Docker Hub.....	37
Gambar 4.10 Nama aplikasi Heroku yang berhasil dibuat.....	38
Gambar 4.11 Build dan push docker image ke container registry.....	39
Gambar 4.12 Rilis aplikasi ke server produksi.....	39
Gambar 4.13 Jumlah utilitas penyimpanan pada blok sda4 sebelum menjalankan kontainer.....	40
Gambar 4.14 Jumlah utilitas penyimpanan pada blok sda4 sesudah menjalankan kontainer.....	40
Gambar 4.15 Grafik perbandingan penggunaan penyimpanan.....	41
Gambar 4.16 PHP info.....	42
Gambar 4.17 Grafik error request server HTTP.....	43
Gambar 4.18 Grafik perbandingan utilitas CPU server HTTP.....	44
Gambar 4.19 Pull request dari branch coll-first-patch-1.....	46
Gambar 4.20 Review perubahan pada file dockerfile.....	46
Gambar 4.21 Pull request approval.....	46
Gambar 4.22 Recreate docker container.....	47

DAFTAR LAMPIRAN

Lampiran 1. Dockerfile.....	53
Lampiran 2. Console outout proses buil dan deploy kontainer pada Jenkins.....	54
Lampiran 3. Console output proses build docker image dan container.....	60

DAFTAR PERINTAH

Perintah 4.1 update status.....	25
Perintah 4.2 Instalasi paket apt-transport-https.....	26
Perintah 4.3 Penambahan GPG key.....	26
Perintah 4.4 Penambahan 8 karakter GPG key.....	26
Perintah 4.5 Penambahan repository docker.....	26
Perintah 4.6 Instalasi Docker CE.....	26
Perintah 4.7 Pengecekan list versi docker ce.....	27
Perintah 4.8 Instalasi versi Docker CE.....	27
Perintah 4.9 Test instalasi Docker.....	27
Perintah 4.10 Clone repository.....	35
Perintah 4.11 Menambahkan file ke repository.....	35
Perintah 4.12 Commit perubahan pada file/folder.....	35
Perintah 4.13 Push file ke repository.....	35
Perintah 4.14 Perintah pada execute shell.....	36
Perintah 4.15 Instal Heroku CLI.....	37
Perintah 4.16 Login ke Heroku.....	38
Perintah 4.17 Cloning repository.....	38
Perintah 4.18 Membuat aplikasi Heroku.....	38
Perintah 4.19 Build dan push docker image.....	38
Perintah 4.20 Rilis aplikasi.....	39
Perintah 4.21 docker-compose build.....	40
Perintah 4.22 docker-compose up.....	40

DAFTAR PROGRAM

Program 4.1 File docker-compose.yml.....	28
--	----

DAFTAR ISTILAH DAN SINGKATAN

- DevOps* : Pendekatan baru dalam pengembangan *software* yang terdiri dari sekumpulan praktik berkelanjutan (*continuous practices*), yaitu *continuous integration*, *continuous deployment* dan *continuous delivery*
- CI/CD : Sebuah praktik pada pengembangan perangkat lunak/*software* yang mengacu kepada *DevOps* yang bertujuan untuk menggabungkan lebih dari satu pengembang dalam satu *repository* sehingga proses *delivery/release* fitur pada *software* tersebut dapat dilakukan lebih cepat dan lebih sering.
- Idle* : Suatu kondisi pada prosesor/CPU yang menunjukkan kondisi ketika tidak ada proses yang dijalankan, direpresentasikan dengan nilai output berupa persentase
- CLI : Antar muka pengguna berbasis teks yang digunakan untuk mengolah file komputer
- Hypervisor : Bagian pada sebuah perangkat komputer yang berfungsi untuk menjalankan sebuah mesin virtual

BAB 1

PENDAHULUAN

Bab ini akan membahas mengenai latar belakang, perumusan masalah, batasan masalah, hipotesis, tujuan penelitian, manfaat penelitian, metodologi, dan sistematika penulisan.

1.1 Latar Belakang

Aplikasi web adalah sebuah program aplikasi yang beroperasi di atas sebuah web server yang diakses melalui internet [1]. Seiring dengan perkembangan teknologi informasi, penggunaan web semakin terasa manfaatnya [2, 3]. Sehingga, kemudahan proses *deployment* (penyebaran) aplikasi web ke server sangat dibutuhkan untuk dapat merespon kebutuhan pengguna [4]. Secara umum terdapat 2 metode *deployment* aplikasi web ke server. Pertama menginstal aplikasi web langsung ke server tunggal. Kelebihan dari metode ini adalah *setup server* mudah, sederhana dan cepat. Tetapi metode tersebut memiliki kekurangan, yaitu setiap aplikasi yang masing-masing memiliki ketergantungan dengan paket versi tertentu yang dapat menimbulkan konflik dependensi (*conflicting dependencies*). Metode yang kedua yaitu dengan memanfaatkan teknologi virtualisasi tradisional (berbasis *hypervisor*). Kelebihan dari metode ini adalah memungkinkan untuk menjalankan lebih dari satu sistem operasi/kernel pada satu perangkat (*hardware*)/komputer. Dengan adanya VM tersebut dapat meningkatkan skalabilitas, karena setiap aplikasi dan *environment* berjalan pada *resource* (CPU, *memory*, I/O) yang berbeda sehingga dapat dengan mudah ditambahkan dan dikurangkan sesuai dengan kebutuhan pengembangan [5,6]. Metode virtualisasi tersebut memiliki kekurangan, yaitu *overhead* pada sistem

karena setiap VM berjalan pada satu *hardware* dan kernel virtual sehingga *resource* dibutuhkan lebih besar.

Virtualisasi berbasis *container* adalah salah satu jenis metode virtualisasi untuk menciptakan sistem yang terisolasi pada level sistem operasi (OS) yang dijalankan pada satu kernel [7]. *Container* adalah sebuah *environment* yang terisolasi untuk menjalankan sebuah paket aplikasi. Berbeda dengan *Virtual Machine*, *container* tidak memiliki *hypervisor* dan saling berbagi kernel *host OS*, sehingga lebih ringan untuk dijalankan. *Docker* adalah salah satu teknologi virtualisasi berbasis *container* yang banyak digunakan saat ini [8,9]. Konsep *Docker* adalah memisahkan antara aplikasi dengan infrastrukturnya, sehingga lebih ringan untuk digunakan. Konsep *Docker* digunakan sebagai salah satu *tools* yang mendukung metode *DevOps*. *DevOps* adalah sebuah metode baru pada pengembangan aplikasi yang menekankan pada praktik berkelanjutan (*continuous integration, delivery, deployment*), sehingga lebih cepat dalam menyediakan aplikasi dan layanan sesering mungkin [9,10,11]. Konsep *DevOps* berkaitan dengan *application development, operations* dan *sevices*, di mana kolaborasi antara pengembang aplikasi dan operator sangat ditekankan. Metode *DevOps* memanfaatkan banyak perangkat/*tools* yang saling terintegrasi. *Docker* adalah salah satu *tools* yang dimanfaatkan untuk proses *deployment* aplikasi pada arsitektur *DevOps*. Tujuan penggunaan *Docker* pada arsitektur *DevOps* adalah menciptakan teknologi komputasi yang ringan, untuk membantu pengembang dalam menempatkan aplikasi dan *environment* ke dalam *resource*/kontainer yang berbeda, sehingga kontainer/*resource* yang dibuat dan dijalankan di bagian

pengembang (*Dev*), terlihat sama dengan bagian operator (*Ops*), dan dapat dijalankan di atas beberapa sistem operasi yang berbeda tetapi tetap terintegrasi [10,11]. Oleh karena itu, akan dibangun sebuah infrastruktur *Docker* pada arsitektur *DevOps* dengan tujuan untuk dapat melakukan proses *deployment* secara berkelanjutan/*continuous deployment*.

1.2 Rumusan Masalah

Berdasarkan latar belakang di atas, maka rumusan masalah pada penelitian ini, sebagai berikut.

1. Bagaimana infrastruktur *Docker* yang dirancang untuk *deployment* aplikasi?
2. Bagaimana konfigurasi *Docker* pada infrastruktur *DevOps*?
3. Bagaimana proses *deployment* aplikasi menggunakan *Docker* pada infrastruktur *DevOps*?

1.3 Batasan Masalah

Agar bahasan penelitian tidak terlalu luas, penelitian ini memiliki batasan-batasan masalah sebagai berikut.

1. Tidak membahas mengenai cara instalasi server *Jenkins* pada arsitektur yang dirancang.
2. Aplikasi web yang digunakan adalah *WordPress* berjenis *blog*.
3. Tidak membahas proses pengujian (*testing*) pada sumber kode/aplikasi.
4. Tidak membahas keamanan pada infrastruktur yang dibangun.
5. Instalasi *Docker* dilakukan pada Sistem Operasi Linux Ubuntu.

6. Server produksi yang digunakan adalah *Docker hub* dan Heroku

1.4 Hipotesis

Penelitian ini memiliki hipotesis yaitu teknologi *Docker* dapat dimanfaatkan untuk *continuous deployment* pada arsitektur *DevOps*.

1.5 Tujuan dan Manfaat

Tujuan dan manfaat penelitian sebagai berikut.

1.5.1 Tujuan

Bertumpu pada rumusan masalah sebagaimana diuraikan di atas, penelitian ini bertujuan, sebagai berikut.

1. Membuat dokumentasi langkah-langkah konfigurasi *Docker* pada infrastruktur *DevOps*.
2. Membuat dokumentasi langkah-langkah proses *deployment* aplikasi menggunakan *Docker* pada infrastruktur *DevOps*.
3. Membangun dan menguji infrastruktur *Docker* untuk *deployment* web aplikasi.

1.5.2 Manfaat

1. Manfaat teoretis

Hasil penelitian ini diharapkan menjadi referensi bahan rujukan untuk pengembangan wawasan IPTEK, khususnya teknologi informasi dalam metodologi pengembangan perangkat lunak/*software* yang dapat menambah efisiensi pengembangan *software*, dan sebagai informasi ilmiah tentang perkembangan arsitektur *microservices*.

2. Manfaat praktis

Apabila infrastruktur ini diterapkan, manfaat yang didapatkan antara lain : mengurangi penggunaan biaya untuk proses *deployment* bagi pengembang aplikasi, memberikan kemudahan integrasi apabila terdapat lebih dari satu pengembang/*developer* dalam proses pengembangan sebuah aplikasi/*software* dan memberikan kemudahan konfigurasi pada *environment* produksi apabila terdapat lebih dari satu jenis sistem operasi.

BAB 2

TINJAUAN PUSTAKA

Bab ini akan membahas mengenai penelitian terkait beserta teknologi yang digunakan dalam penelitian tugas akhir.

2.1 Penelitian Terkait dan Produk Sejenis

S. Apridayanti et al [12] merancang sebuah konfigurasi untuk *deployment* aplikasi web menggunakan *Docker*. Aplikasi yang digunakan merupakan aplikasi jual beli dengan mengimplementasikan teknologi virtualisasi *server* menggunakan *Docker* untuk dipindahkan (*deploy*) ke server *Docker*. Skenario yang diterapkan pada penelitian ini dengan menjalankan (*running*) aplikasi tersebut di dua *Client* yang berbeda yaitu Windows 7 dan Windows 8.

R. Bik dan M. Fadlulloh [13] merancang sebuah sistem pengelolaan banyak aplikasi web (*web hosting*) dengan memanfaatkan teknologi *Docker* sebagai pengganti teknologi virtualisasi untuk proses *deployment*. Skenario yang digunakan pada penelitian ini dengan membangun beberapa kontainer berisi aplikasi beserta *environment* yang dibutuhkan. Kemudian masing-masing kontainer diberikan domain. Pada penelitian ini juga menganalisis total memori yang digunakan.

Y. T. Sumbogo, M. Data, dan R. Andria Sirega [14] merancang sebuah sistem *web server* menggunakan teknologi virtualisasi. *Web server* yang digunakan adalah NGINX. Skenario yang digunakan pada penelitian ini dengan menggunakan Kubernetes, sebuah *platform open source* yang digunakan untuk menjalankan mekanisme *failover* dan *autoscaling* pada kontainer berbasis *Docker*

untuk mengetahui unjuk kerja *web server* sehingga dapat menerima *request* dari pengguna.

Tabel 2.1 Tabel perbandingan sistem

Pembanding	Pembahasan unjuk kerja <i>Docker</i>	Integrasi <i>Docker</i> dengan perangkat lain	Implementai <i>Docker</i> pada <i>DevOps</i>
S. Apridayanti et al [12]	Tidak	Tidak	Tidak
R. Bik dan M. Fadlulloh [13]	Ya	Tidak	Tidak
Y. T. Sumbogo, M. Data, dan R. Andria Siregar [14]	Ya	Tidak	Tidak

2.2 Teknologi Virtualisasi

Virtualisasi adalah sebuah proses dari pembentukan sebuah bentuk atau versi virtual yang merepresentasikan sesuatu, seperti aplikasi virtual, *server*, penyimpanan (*storage*) dan jaringan (*network*) dengan tujuan untuk mengefisiensikan biaya dan meningkatkan efisiensi serta keandalan (*agility*) [15,16]. Virtualisasi memungkinkan kita berbagi *hardware* untuk digunakan beberapa sistem operasi [17]. Virtualisasi berbasis *hypervisor* (disebut juga *virtual machine monitor*) adalah teknologi virtualisasi yang biasa digunakan oleh pengembang. Seperti yang terlihat pada Gambar 2.1, sebuah *hypervisor* (aplikasi untuk menjalankan teknik virtualisasi) akan mengisolasi sistem operasi dan aplikasi dari perangkat *hardware* komputer, sehingga *host machine* dapat menjalankan beberapa mesin virtual sebagai *guest host* dan berbagi *resource*, seperti prosesor, memori dan *bandwidth* jaringan [18].

Layer aplikasi/ <i>software</i>	Layer aplikasi/ <i>software</i>	Layer aplikasi/ <i>software</i>
Sistem Operasi	Sistem Operasi	Sistem Operasi
Layer Virtualisasi		
Layer Hardware		

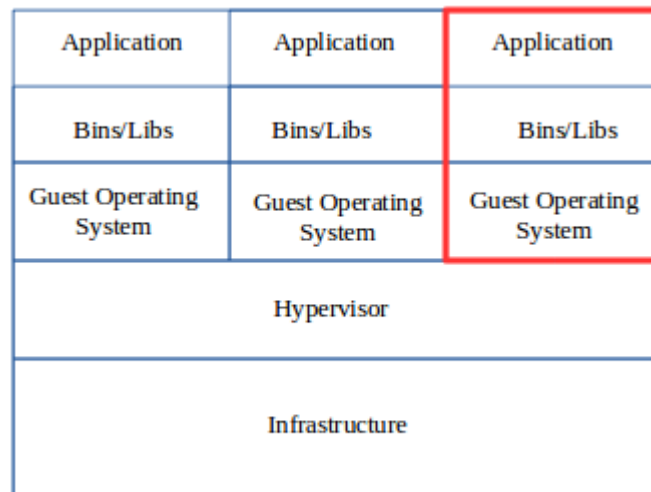
Gambar 2.1 Layer teknologi virtualisasi

Sebagian besar teknologi virtualisasi yang berkembang, fokus pada kernel linux dan terbagi menjadi 3 kategori: *full-virtualization*, *para-virtualization* dan *container-based virtualization*. *Full-virtualization* tidak membutuhkan penyesuaian kernel, sedangkan pada *para-virtualization*, kernel pada mesin virtual dimodifikasi untuk mengoptimalkan unjuk kerja di lingkungan virtual (*virtual environment*) [7].

2.3 Mesin Virtual

Mesin virtual adalah implementasi perangkat lunak dari sebuah mesin komputer yang dapat menjalankan program sama seperti layaknya sebuah komputer asli [19]. Prinsip kerja mesin virtual adalah mengisolasi setiap sistem operasi dan aplikasi dari mesin virtual yang lain. Setiap mesin virtual berisi *guest operating system*, sistem binari, sistem file dan aplikasi. Seperti yang terlihat pada Gambar 2.2, setiap mesin virtual memiliki *hypervisor* yang memisahkan antara sistem operasi komputer dan aplikasi dari perangkat keras fisik yang mendasarinya,

secara umum mesin virtual menyediakan *environment* dengan *resource* yang lebih dari yang dibutuhkan sebuah aplikasi [20,21].

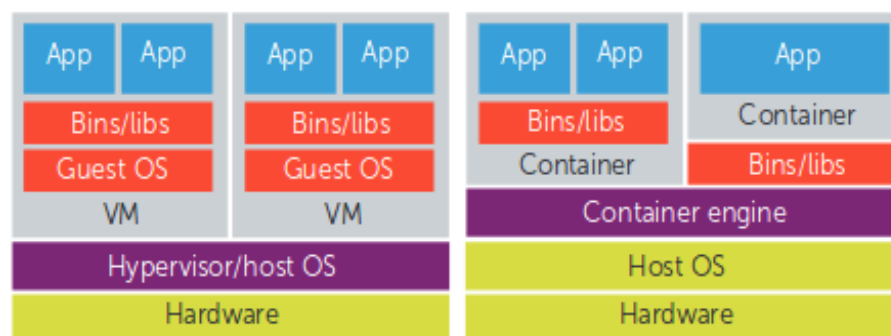


Gambar 2.2 Layer teknologi virtualisasi OS

Guest operating system (guest OS) adalah sebuah perangkat lunak yang terinstal pada sebuah mesin virtual atau *disk* partisi yang mewakili sebuah sistem operasi yang berbeda dengan *host operating system (host OS)*. Teknologi virtualisasi mengizinkan sebuah komputer untuk menjalankan beberapa sistem operasi secara bersamaan [22]. Sebagai contoh, jika sebuah *host OS* menjalankan Linux, maka setiap *guest OS* pada *disk* partisi harus menjalankan Linux juga. Sedangkan pada lingkungan yang tervisualisasi, sebuah *guest OS* dapat berbeda dengan *host OS*. Mesin virtual meningkatkan portabilitas dan fleksibilitas proyek, karena tidak bergantung pada *hardware* spesifik mana pun serta dapat bermigrasi ke *cloud*, dan *local environment* mana saja.

2.4 Teknologi Virtualisasi Berbasis Kontainer

Kontainer adalah sebuah perangkat yang digunakan untuk mengirimkan aplikasi dengan menggunakan prinsip virtualisasi sistem operasi [23]. Berbeda dengan mesin virtual, kontainer berbagi OS dan kernel yang sama, sehingga menghemat lebih banyak penggunaan *resource* [8]. Kontainer membutuhkan sebuah sistem operasi yang mendasarinya dari perangkat fisik/*hardware* sebagai penyedia layanan untuk semua aplikasi yang terkontainerisasi menggunakan memori virtual. Sedangkan sebuah *hypervisor*, menjalankan beberapa mesin virtual dengan sistem operasinya masing-masing menggunakan perangkat keras mesin virtual, seperti yang terlihat pada Gambar 2.3.



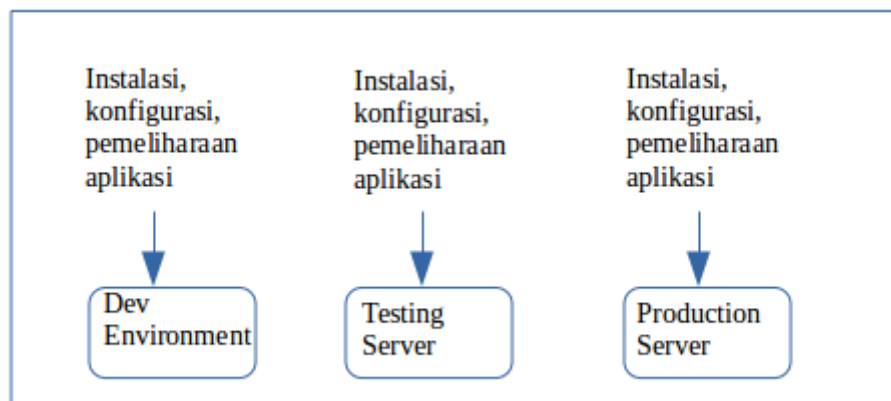
Gambar 2.3 Perbedaan arsitektur antara teknologi virtualisasi berbasis kontainer dengan virtualisasi tradisional [20]

Sistem kontainer memiliki *overhead* yang lebih rendah dibandingkan dengan mesin virtual dan sistem kontainer juga mengisolasi *environment* antara satu kontainer dengan kontainer lainnya, sehingga layanan kontainer seperti sistem file atau jaringan memiliki akses *resource* yang terbatas [24]. LXC (*Linux Container*) adalah salah satu antarmuka ruang pengguna untuk fitur kontainerisasi kernel linux dengan API (*Application Programming Interface*) dan perangkat/*tools* yang mudah digunakan untuk membuat dan mengatur sistem atau aplikasi dari

kontainer. Tujuan dari LXC adalah untuk membentuk sebuah *environment* sedekat mungkin dengan standar instalasi linux namun tanpa harus pada kernel yang terpisah [25].

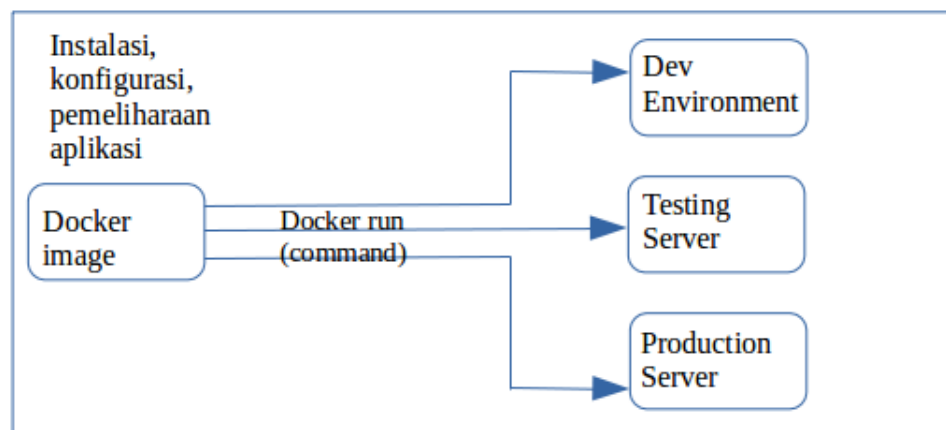
2.5 Docker

Docker adalah sebuah perangkat/tools yang digunakan untuk mengemas aplikasi dan semua independensinya ke dalam sebuah kontainer [7]. Pada dasarnya, *Docker* adalah sebuah sistem pengelolaan yang digunakan untuk membuat, mengelola dan mengawasi kontainer Linux. *Docker* menggunakan skema yang sama untuk membuat kontainer di mesin virtual berbasis linux. Kontainer memungkinkan untuk mengemas satu aplikasi/*software* beserta komponen-komponen yang dibutuhkan untuk menjalankan aplikasi tersebut (*source code*, *dependency*, dan *runtime*). Sebelum dengan teknologi *Docker*, proses *deployment* ke *environment* yang berbeda membutuhkan usaha yang signifikan, seperti yang terlihat pada Gambar 2.4, setiap *environment* membutuhkan konfigurasi dan *resource management*, sehingga penggunaan sumber daya akan lebih banyak dan mempengaruhi kerja dari perangkat.



Gambar 2.4 Ilustrasi pengaturan proses *deployment* tidak menggunakan *docker*

Teknologi Docker memisahkan proses konfigurasi dengan *resource management*, memberikan sebuah *environment* (kontainer), seperti yang terlihat pada Gambar 2.5, di mana kode/aplikasi dapat diuji dan di-*deploy* secara efektif dan cepat. Ketika pengembang menjalankan perintah *docker run*, sebuah *environment* dari *docker image* akan tertarik (*pull*). *Environment* tersebut menggunakan *resource* yang lebih kecil.



Gambar 2.5 Ilustrasi proses deployment menggunakan docker

Manfaat dari penerapan metodologi Docker ini antara lain.

1. Menggantikan mesin virtual

Docker digunakan karena penggunaan *resource* yang lebih kecil dan lebih mudah untuk dipindah dari satu sistem operasi ke sistem operasi lain jika dibandingkan dengan mesin virtual.

2. Pengemasan perangkat lunak

Penggunaan *Docker* untuk pengemasan perangkat lunak dirasa sangat efektif karena dengan konsep *Docker image*, aplikasi yang sudah dikemas dalam

bentuk *image* dapat dijalankan di sistem Linux modern manapun, seperti contoh Java, kita tidak membutuhkan sebuah JVM (*Java Virtual Machine*).

3. *Continuous Delivery/Deployment* (CD)

Metode *Docker* dapat mempermudah implementasi CD, karena *environment* yang dibangun menggunakan *Docker* dapat lebih mudah untuk direplikasi.

2.5.1 Objek-objek *Docker*

1. *Docker image*

Docker image adalah sebuah templat *read-only* dengan instruksi untuk membentuk sebuah *Docker container*. Sebuah *image* terdiri dari file dan metadata. Metadata memuat informasi variabel *environment*, pemetaan *port*, volume dan detail lainnya. Terdapat dua cara untuk membangun sebuah *Docker image*. Pertama dengan menggunakan templat *read-only*. Dasar dari semua *image* adalah *base image*. Sebagai contoh, sebuah *image* sistem operasi (Ubuntu 16.04 LTS) adalah sebuah *base image*. Kedua dengan menggunakan *docker file*. *Dockerfile* berisi sintaksis sederhana untuk mendefinisikan langkah-langkah yang dibutuhkan untuk membuat dan menjalankan sebuah *image*. Setiap instruksi pada sebuah *Dockerfile*, akan membentuk sebuah lapisan pada *image*. Pada dasarnya, ketika ada perubahan atau modifikasi pada *Dockerfile* atau *image*, yang berubah adalah lapisan-lapisan tersebut, sehingga *image* sangat ringan, kecil dan cepat apabila dibandingkan dengan mesin virtual [26].

2. *Docker Container*

Pendefinisian kontainer dapat dibuat analogi dengan sebuah program dan proses. Sebuah proses dapat diartikan sebagai sebuah aplikasi yang sedang dijalankan,

kontainer pada *Docker* dapat diartikan sebagai sebuah *Docker image* yang sedang dieksekusi/dijalankan, sehingga dapat dikatakan kontainer dibuat dari sebuah *image*, inherit dengan sistem file dan metadata sebuah *image* untuk menentukan konfigurasi startup [27]. Kontainer dapat dibuat, dijalankan, dihentikan, dipindahkan atau dihapus menggunakan API *Docker* atau CLI (*Command Line Interface*).

3. *Docker Registries*

Docker image disimpan pada *Docker registries*. Cara kerja dari *Docker registries* sama dengan cara kerja *repository* sumber kode, seperti *Github*. Pengembang/developer dapat melakukan *pull*, *push* pada *Docker image*.

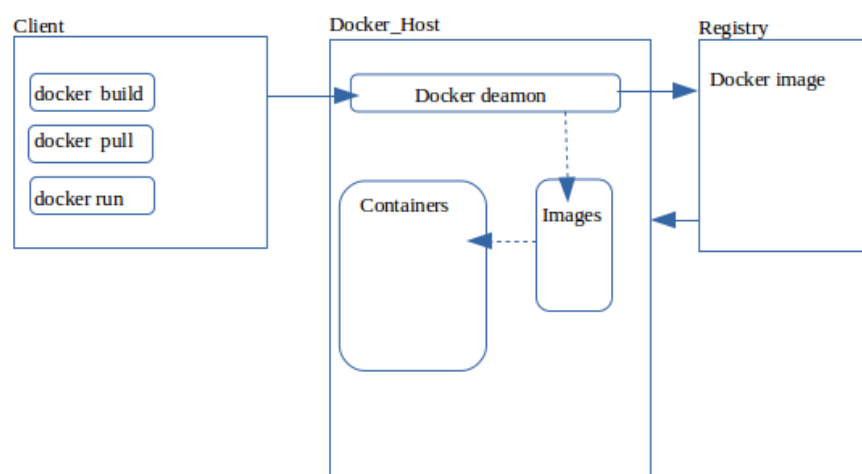
4. *Docker Client Sever*

Docker server/daemon mendapatkan *request* dari *docker client*. Keduanya dapat dijalankan di atas mesin yang sama atau *docker client* dari *localhost* dapat dihubungkan dengan *daemon* di mesin yang berbeda [28].

2.5.2 **Arsitektur *Docker***

Docker menggunakan arsitektur *client-server*. Pada sisi *client* terdapat *Docker client* yang digunakan untuk berinteraksi dengan *Docker*. Ketika sebuah perintah pada *client* dijalankan, *client* akan mengirimkan perintah tersebut ke *Docker daemon* (*Dockerd*) menggunakan *Docker RESTful API* (*Application Programming Interface*). Seperti yang terlihat pada Gambar 2.6, sisi server terdapat *Docker daemon* yaitu sebuah *service* yang berjalan di atas dasar *Docker host* yang bertugas untuk mengatur, membangun dan menjalankan objek-objek *Docker*, seperti *image*, dan *container*. Berbeda dengan *Docker client* dan *Docker*

daemon yang berada pada jaringan lokal, *Docker registry* berada pada jaringan publik, sehingga dapat diakses semua *user*. *Docker registry* berfungsi untuk menyimpan *Docker image*. *Docker hub* adalah sebuah *registry* pada jaringan internet yang dikembangkan oleh *Docker*. Sebuah *image* pada *Docker hub* dapat diakses oleh *Docker daemon* melalui RESTful API.

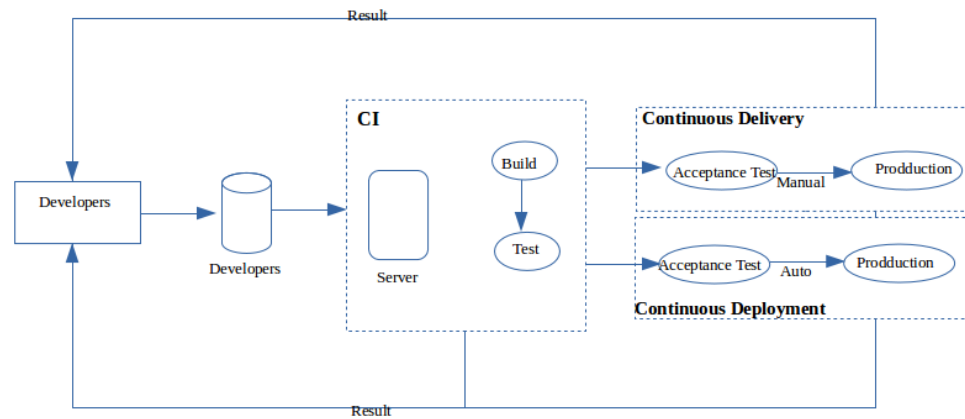


Gambar 2.6 Ilustrasi infrastruktur Docker

2.6 DevOps

DevOps adalah sebuah pendekatan baru dalam pengembangan *software* yang terdiri dari sekumpulan praktik berkelanjutan (*continuous pactices*) yang dimaksudkan untuk mengurangi waktu yang dibutuhkan antara melakukan perubahan pada sistem dan perubahan yang terjadi pada ranah produksi dengan tetap memaksimalkan kualitas yang tinggi [29,30], sehingga dengan kata lain *DevOps* adalah sebuah konsep yang menekankan pada kolaborasi antara pengembangan aplikasi/*software* dan operator IT. Salah satu manfaat dari *DevOps* adalah mengurangi jarak/*gap* antara tim pengembang dengan tim IT, dengan tidak

mengganggu sebuah fitur/produk yang sudah pada area produksi [31]. Tujuan dari konsep *DevOps* adalah bukan hanya meningkatkan intensitas laju perubahan tetapi juga untuk perpindahan (*deployment*) fitur-fitur ke ranah produksi tanpa menyebabkan eror atau masalah pada *service* lainnya. Pendekatan *DevOps* meliputi *Continuous Integration* (CI), dan *Continuous Delivery* (CDE). *Continuous Integration* bertujuan untuk menghubungkan setiap *developer*, menggabungkan pekerjaan (kode) pada satu *repository*, sehingga hasil dari pekerjaan dapat lebih termonitor dan berkualitas. *Continuous Delivery* bertujuan untuk memastikan sebuah aplikasi selalu dalam keadaan siap produksi setelah melewati tahap *testing* untuk dipindahkan (*deploy*) ke *production environment*. Beberapa manfaat dari penerapan praktik ini adalah mengurangi resiko proses *deployment*, biaya yang lebih rendah dan proses umpan balik pengguna yang lebih cepat. Seperti yang terlihat pada Gambar 2.7, proses *continuous integration* diperlukan untuk melakukan proses *continuous delivery*. Sedangkan *Continuous Deployment* (CD) bertujuan untuk memindahkan (*deploy*) aplikasi ke *production/customer environment*. Perbedaan CD dari CDE adalah pada *production environment*, yaitu lingkungan pengguna sesungguhnya, sehingga prosesnya tidak dapat dilakukan secara manual.



Gambar 2.7 Ilustrasi hubungan antara continuous integration, delivery dan deployment[30]

Setiap fase (CI,CD, dan CDE) terdiri dari beberapa perangkat/*tools* yang terintegrasi, di mana setiap *tool* memiliki perannya masing-masing pada setiap fase di tahapan pengembangan perangkat lunak/*software*. Salah satu tantangan pada *DevOps* adalah mempertahankan konsistensi dengan mengemas dan mereplikasi *dependencies* dan paket yang sama pada lingkup produksi/*production environment* [32]. Kontainerisasi adalah teknologi yang dapat digunakan untuk menyelesaikan masalah inkonsistensi tersebut. *Docker* adalah salah satu perangkat yang dapat menunjang teknologi kontainerisasi. Selain untuk kontainerisasi, *Docker* juga digunakan sebagai salah satu perangkat untuk proses *deployment* pada tahapan pengembangan *software*.

BAB 3 METODE PENELITIAN

Bab ini akan membahas mengenai tempat penelitian, alat dan bahan penelitian, alur dan atau tahapan penelitian serta waktu dan jadwal penelitian.

3.1 Tempat Penelitian

Penelitian dilaksanakan dalam waktu 5 bulan dimulai dari bulan Maret 2019 sampai dengan bulan Agustus 2019 bertempat di Laboratorium KSI Kampus Teknik Elektro Universitas Jenderal Soedirman yang terletak di Jalan Mayjen Sungkono Km 5 Blater, Purbalingga.

3.2 Alat dan Bahan Penelitian

Alat dan bahan yang digunakan dalam penelitian ini sebagai berikut.

Tabel 3.1 Alat dan bahan penelitian

Alat	Bahan
Laptop Koneksi internet; Server <i>DevOps</i> Elektro; Docker; <i>Git</i> ; Apache J-meter	<i>WordPress</i> blog

Tabel 3.2 Spesifikasi Laptop/Komputer

Spesifikasi Komputer/Laptop Client dan Server	
Merk	HP
OS	Ubuntu 18.04 LTS
CPU	AMD A8-6410 (1 Ghz)
RAM	4 GiB
Disk	78GiB

3.3 Alur dan Tahapan Penelitian

Tahap-tahap dalam penelitian yaitu dimulai dari tahap persiapan, kemudian dilanjutkan dengan tahap perancangan, tahap pengujian, dan yang

terakhir yaitu tahap pembuatan laporan. Rincian dari setiap tahap adalah sebagai berikut.

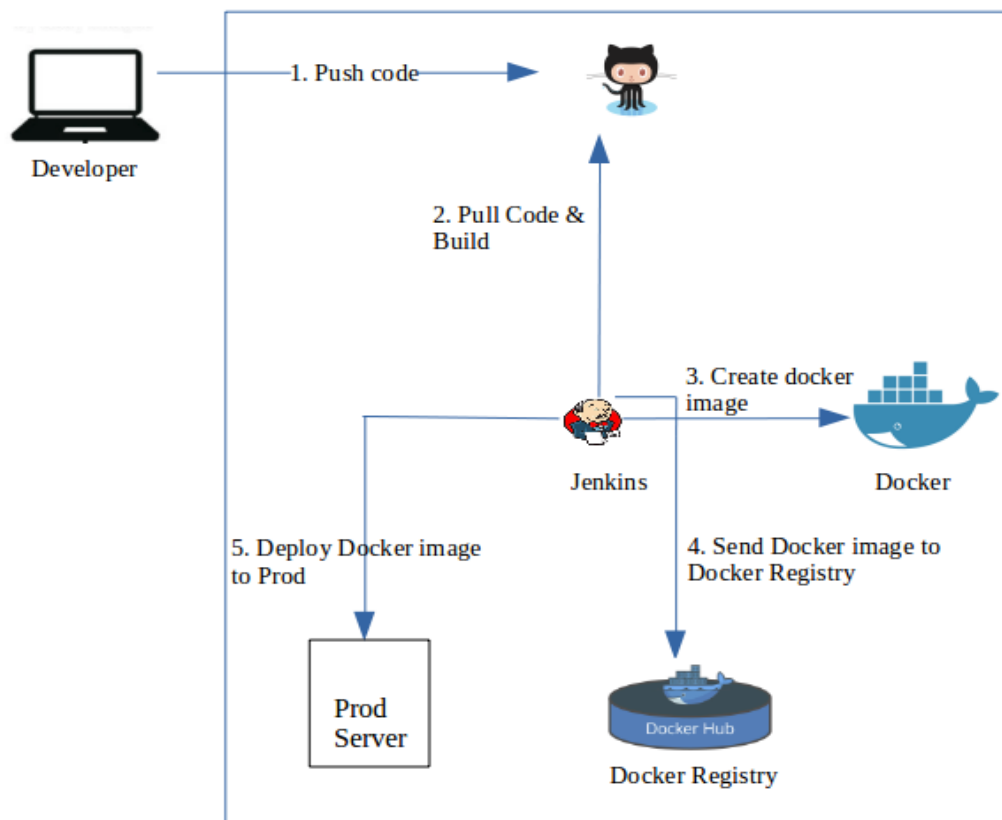
3.3.1 Tahap Persiapan

Tahap ini merupakan tahap untuk mempersiapkan segala bahan yang mendukung dalam melakukan penelitian. Pemasangan/*installation Docker* pada *localhost* (Linux Ubuntu 18.04 LTS). Kemudian pemasangan/*installation WordPress blog* dengan *docker-compose.yml* sebagai aplikasi untuk skenario pengujian, maupun pemasangan *WordPress blog* dengan instalasi manual (*apache, phpmyadmin, mysql*).

3.3.2 Tahap Perancangan

Tahap perancangan penelitian akan dilakukan dengan membuat diagram alur kerja/*workflow* pada arsitektur *DevOps*, seperti yang dapat dilihat pada Gambar 3.1 sehingga dapat diimplementasikan untuk proses *continuous deployment*. Perangkat-perangkat/*tools* yang digunakan adalah *Docker, Jenkins* dan *Github*. *Continuous Integration (CI)* merupakan fase pertama dari tahapan proses *continuous deployment*. *Tools* yang digunakan adalah *Jenkins, Git* dan *Github*. *Github* sebagai *repository* kode, digunakan untuk mengatur folder atau file dari aplikasi. Aplikasi akan didorong (*push*) pada *master repository*. *Jenkins* digunakan untuk membangun dan menyusun paket aplikasi dan memindahkan (*deploy*) paket kode aplikasi ke lingkup produksi, mesin virtual ataupun server. setiap perintah *commit* pada *Github* akan memicu *Jenkins* secara otomatis untuk membangun/*build* perubahan yang dilakukan dengan konfigurasi *automation build* atau perintah/*command* pada kolom *execute shell*. *Jenkins* akan menarik

(pull) sumber kode dari *repository Github* ke server *Jenkins*. Kemudian kode tersebut dibangun (*build*). Keseluruhan proses dilakukan secara otomatis di dalam *Jenkins*. Setelah melewati fase pertama, kemudian aplikasi tersebut akan disebar ke tahap produksi (*production server*) menggunakan perangkat *Docker*.

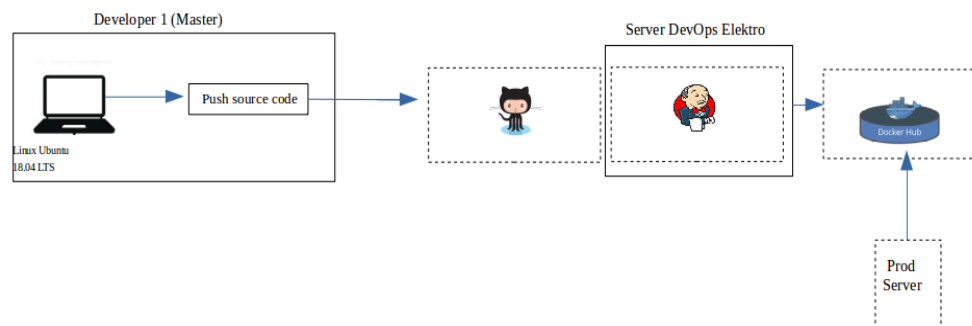


Gambar 3.1 Arsitektur CI/CD untuk proses deployment

3.3.3 Tahap Pengujian

Setelah tahap perancangan selesai, tahap berikutnya adalah tahap pengujian. Pengujian dilakukan di Lab KSI Kampus Teknik Elektro Unsoed menggunakan Server *DevOps Elektro*. Skenario pada pengujian ini adalah terdapat pengembang/*developers* pada satu *repository Github*. Pengembang akan

melakukan *push* Aplikasi (*docker compose*) yang sudah dibangun melalui *Git* ke *Github*. Aplikasi yang sudah dipindahkan ke *Github* kemudian dibangun (*build*) menggunakan *Jenkins*. *Docker container* sebagai keluaran dari *Jenkins*, kemudian dipindahkan (*deploy*) ke *Docker Hub*, sebagai *repository* akhir pada bagian pengembang. Kemudian pada produksi/*production environment*, aplikasi (kontainer) pada *docker hub* akan ditarik (*pulled*) secara manual.

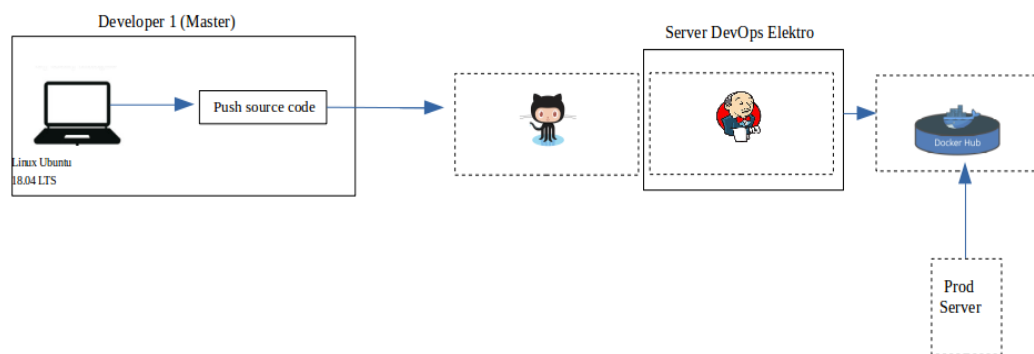


Gambar 3.2 Arsitektur pengujian proses deployment aplikasi

Skenario 1

Pengujian terhadap kompatibilitas pada Docker untuk menguji keunggulan Docker dalam hal kompatibilitas. Pengujian akan dilakukan dengan menggunakan laptop dengan spesifikasi OS Linux Ubuntu 18.04 LTS sebagai *local host*, *Jenkins* sebagai *automation server*, dan Heroku sebagai *production server*. Studi kasus yang digunakan adalah sebuah CMS *WordPress* berjenis *blog* yang dikemas dengan dan tanpa kontainer (*docker compose*). Setelah berhasil dibentuk (*build*) pada *Jenkins server*, aplikasi tersebut akan dipindahkan (*deploy*) *production server*, Heroku. Untuk mengetahui kompatibilitas *docker*, aplikasi (*WordPress* yang dipasang dengan *docker compose*) akan dijalankan pada *local host* dengan spesifikasi Sistem Operasi Windows 10 tanpa melakukan perubahan apapun pada

local host dan kontainer seperti yang terlihat pada gambar 3.3. Hasil pengujian akan diuraikan pada Tabel 3.2.



Gambar 3.3 Skenario pengujian kompatibilitas Docker

Tabel 3.3 Analisis pengujian kompatibilitas dan efektivitas Docker

CMS	Analisis			
	Kompatibilit as	Perbandingan Penggunaan Ruang Penyimpanan		
		Sebelum instalasi (KB)	Sesudah instalasi (KB)	Peningkatan Penyimpanan (MB) Hasil = Sesudah - Sebelum
Container WordPress (Docker)				
WordPress				

Skenario 2

Pengujian terhadap beban kerja pada *Docker* untuk mengetahui kinerja CPU untuk merespon simulasi *request* dengan rentang waktu tertentu dan menghasilkan sejumlah sampel. *Jmeter* akan digunakan sebagai simulator beban kerja dengan mengirimkan jumlah *request* yang bertambah secara bersamaan. Skenario ini dilakukan untuk mengetahui seberapa banyak *request* yang dapat dijalankan *server* sampai pada batas kemampuan memori dan menimbulkan

masalah pada kinerja *server* [33]. Utilitas CPU saat pengujian juga dihitung dengan menggunakan persamaan mengurangi 100% (mewakili seluruh sumber daya CPU yang tersedia) dengan rata-rata persentase CPU dalam keadaan *idle*, dengan skenario utilitas CPU tidak boleh melebihi 20% [34, 35]. Pengujian akan dilakukan terhadap studi kasus yang sama dengan skenario 1, dan hanya dilakukan kepada layanan HTTP. Perbedaan parameter pada pengujian variasi beban kerja layanan HTTP terletak pada jumlah *user* yang akan mengakses web aplikasi dalam satu detik, dengan sebuah skenario, server harus dapat menerima minimal 10 *request* dalam satu detik pada interval waktu 10 detik [35]. Spesifikasi beban kerja yang digunakan terlihat pada Tabel 3.4 sebagai berikut.

Tabel 3.4 Spesifikasi beban kerja pengujian

Pengujian pada layanan HTTP	Beban kerja		
	ringan	sedang	berat
-Kinerja server			
1. Jumlah <i>user</i>	50	100	500
2. Rentang waktu (<i>delay</i> 1 detik)	0.02 detik	0.01 detik	0.002 detik
-Utilitas CPU			
1. Jumlah <i>user</i>	100	100	100
2. Jumlah <i>looping</i>	1	3	5

3.4 Pembuatan Laporan

Tahap terakhir setelah perancangan infrastruktur *deployment* aplikasi telah selesai dikonfigurasi dengan apa yang diinginkan, maka dilanjutkan dengan pembuatan laporan penelitian ini yang berjudul “Perancangan *Docker* pada Arsitektur *DevOps* untuk Proses *Deployment* Aplikasi Berbasis Web”. Penulisan laporan dilakukan dengan cara mendokumentasikan cara menginstal perangkat-

Penelitian ini akan dilaksanakan dari bulan Maret sampai dengan Agustus tahun 2019, dengan rincian jadwal kegiatan dilihat pada Tabel 3.5.

[illegible]

BAB 4 HASIL DAN PEMBAHASAN

Pada bab ini akan dibahas langkah-langkah instalasi *Docker*, pembuatan *docker image/docker compose*, integrasi *Docker* dengan *Github* dan *Jenkins* untuk *deployment*, serta analisis dari penggunaan *Docker*.

4.1 Instalasi *Docker*

Pada proses instalasi *Docker Community Edition* (CE), dibutuhkan minimal tipe Sistem Operasi 64-bit. Pada penelitian ini digunakan Sistem Operasi Ubuntu $\times 86_64$ versi Xenial 18.04 (LTS). Instalasi *Docker* pada Sistem Operasi Linux relatif lebih mudah dibandingkan dengan Windows atau Mac, karena *Docker* tersedia pada *repository* standar pada sebagian besar distribusi Sistem Operasi Linux. Instalasi *Docker* dapat dilakukan dengan beberapa langkah berbeda yang disesuaikan dengan kebutuhan. Instalasi *Docker* pada penelitian ini menggunakan *repository Docker*. Langkah-langkah instalasi *Docker* adalah sebagai berikut.

Pengaturan *Repository*

1. Memperbarui indeks paket *apt* dengan Perintah 4.1.

Perintah 4.1 update status

```
$ sudo apt-get update
```

2. Memasang paket untuk mengizinkan *apt* menggunakan *repository* melalui HTTPS dengan Perintah 4.2.

Perintah 4.2 Instalasi paket apt-transport-https

```
$ sudo apt-get install apt-transport-https \
    ca-certificates curl gnupg-agent \
    software-properties-common
```

3. Menambahkan GPG key dengan Perintah 4.3.

Perintah 4.3 Penambahan GPG key

```
$ curl -fsSL \ https://download.docker.com/linux/ubuntu/gpg |
sudo apt-key add -
```

Menambahkan 8 karakter terakhir GPG key 0EBF CD88 dengan Perintah 4.4.

Perintah 4.4 Penambahan 8 karakter GPG key

```
$ sudo apt-key fingerprint 0EBFCD88
```

4. Menambahkan *repository Docker* dengan Perintah 4.5.

Perintah 4.5 Penambahan repository docker

```
$ sudo add-apt-repository \
    "deb [arch=amd64]
https://download.docker.com/linux/ubuntu \
    $(lsb_release -cs) \
    stable"
```

Instalasi Docker CE

1. Memperbarui indeks paket *apt* dengan menjalankan Perintah 4.1.
2. Memasang versi terbaru *Docker CE* dengan menjalankan Perintah 4.6.

Perintah 4.6 Instalasi Docker CE

```
$ sudo apt-get install docker-ce
```

3. Memasang versi spesifik dari *Docker CE*.
 - a) Mengetahui daftar list versi yang tersedia di *repository* dengan Perintah 4.7.

Perintah 4.7 Pengecekan list versi docker ce

```
$ apt-cache madison docker-ce
```

- b) Memasang versi spesifik dari *Docker CE* dengan Perintah 4.8.

Perintah 4.8 Instalasi versi Docker CE

```
$ sudo apt-get install docker-ce=<VERSION_STRING>
```

Setelah mengetahui versi yang tersedia pada *repository*, kemudian memasang *Docker* dengan versi spesifik, sebagai contoh pada penelitian ini digunakan versi *17.09.0-ce-0~ubuntu*.

```
$ sudo apt-get install docker-ce=17.09.0~ce-0~ubuntu
```

4. Memeriksa apakah pemasangan/*installation Docker CE* berhasil dengan menjalankan Perintah 4.9.

Perintah 4.9 Test instalasi Docker

```
$ sudo docker run hello-world
```

4.1.1 Docker Compose

Docker compose adalah sebuah perangkat/*tools* yang digunakan untuk menjalankan beberapa kontainer aplikasi *Docker*. *Docker compose* berbentuk sebuah file dengan format *YAML* yang mendefinisikan bagaimana sebuah kontainer *Docker* bekerja pada ruang produksi. Penggunaan *Docker compose* pada dasarnya terdiri dari tiga langkah, pertama mendefinisikan lingkungan aplikasi dengan *dockerfile*, kedua mendefinisikan layanan-layanan (*services*) pada aplikasi yang akan dibangun dengan *docker-compose.yml* dan ketiga, menjalankan

Docker compose. Versi *Docker compose* yang digunakan pada penelitian ini adalah versi 1.8. Pada penelitian ini digunakan sebuah CMS *WordPress* jenis *blog*. *WordPress* tersebut memiliki 3 *services* yang masing-masing menjalankan satu *image*, yaitu *database server service*, *web server service* dan *web service* (*WordPress*). *Docker compose* pada penelitian ini digunakan untuk membandingkan penggunaan *resource* antara aplikasi *WordPress* tanpa dan dengan *Docker*. File *Docker compose* dapat dilihat pada Program 4.1.

Program 4.1 File docker-compose.yml

```
version: '2'
services:
  db:
    image: mysql:5.7
    restart: always
    volumes:
      - db_data:/var/lib/mysql
    environment:
      MYSQL_ROOT_PASSWORD: root
      MYSQL_DATABASE: WordPress
      MYSQL_USER: root
      MYSQL_PASSWORD: root
    networks:
      - back
  phpmyadmin:
    depends_on:
      - db
    image: phpmyadmin/phpmyadmin
    restart: always
    ports:
      - 8080:80
    environment:
      PMA_HOST: db
      MYSQL_ROOT_PASSWORD: root
    networks:
      - back
  web:
    build: .
    depends_on:
      - db
    restart: always
    ports:
      - 8001:80
    volumes:
      - ./wp-app:/var/www/html
    networks:
      - back
networks:
  back:
volumes:
  db_data:
```

4.1.2 DockerFile

Penelitian ini menggunakan satu file *dockerfile* untuk mendefinisikan lingkungan/*environment* dari *web service* (WordPress). *Environment* pada *web service* akan didefinisikan di dalam *Dockerfile* karena aplikasi WordPress yang digunakan dibuat sesuai dengan kebutuhan yang digunakan pada penelitian ini. Naskah di dalam *dockerfile* diperoleh dari web resmi *docker hub*¹. Terdapat dua file yang digunakan, yaitu *dockerfile* dan *docker-entrypoint.sh*. Konfigurasi *environment* juga dilakukan di dalam *dockerfile*, yaitu penambahan *database environment* yang akan digunakan. Konsep *dockerfile* dan *docker compose* dapat dikatakan sama, yaitu berfungsi untuk menjalankan kontainer, perbedaannya adalah *docker compose* mengeksekusi *default image* yang tersedia pada *Docker Hub*, sedangkan, *Dockerfile* mengeksekusi *default image* maupun sebuah *Docker image* yang dikembangkan sendiri atau secara *custom*. Program dari *Dockerfile* dapat dilihat pada Lampiran 1.

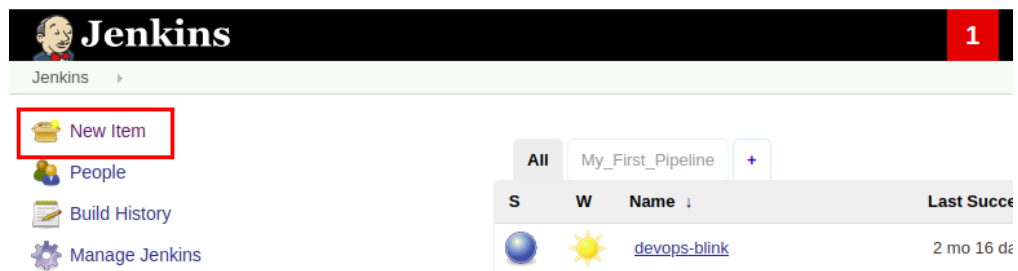
4.2 Integrasi Docker Pada Infrastruktur DevOps

Pada penelitian ini dirancang sebuah infrastruktur *DevOps* untuk melakukan proses *deployment* aplikasi, sehingga dapat mengimplementasikan praktik CI/CD. *Github*, *Docker* dan *Jenkins* diintegrasikan untuk membangun infrastruktur *Continuous Integation/Continuous Deployment*. Langkah pertama untuk mengintegrasikan *Jenkins* dengan *repository Github* adalah memasang *Git* dan *Docker Plugin* pada *Jenkins server*. Kemudian melakukan konfigurasi pada project *Jenkins* yang akan dibangun, berikut adalah dokumentasi langkah-langkah

1 <https://hub.docker.com/>

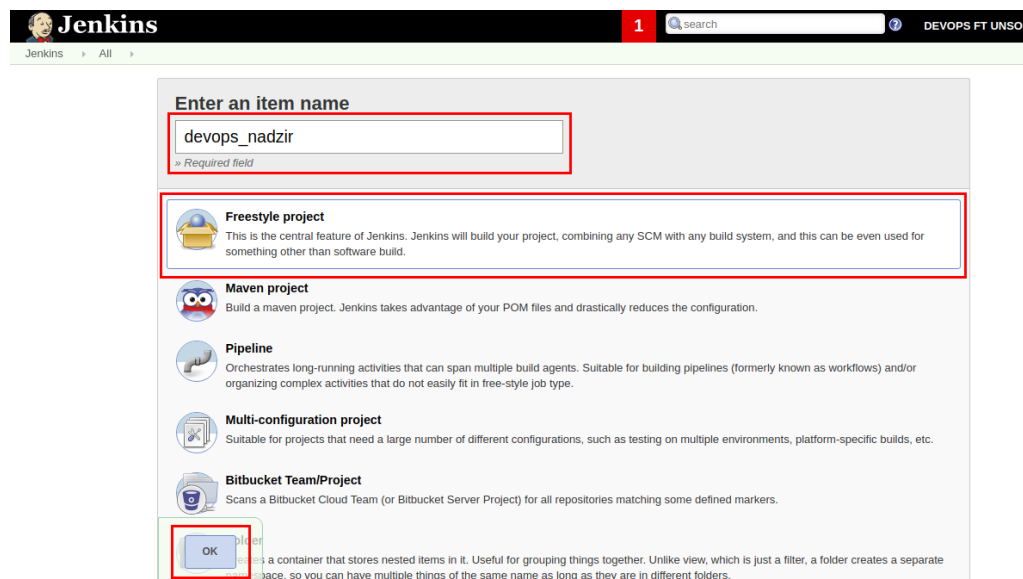
mengintegrasikan project *Jenkins* dengan *Github*, sehingga *Jenkins* dapat melakukan *pull* dan *build* sumber kode dari *Github repository*.

1. Membuat *project* baru dengan mengklik *New Item* pada dashboard *Jenkins*, seperti yang terlihat pada Gambar 4.1.



Gambar 4.1 Membuat project-langkah 1

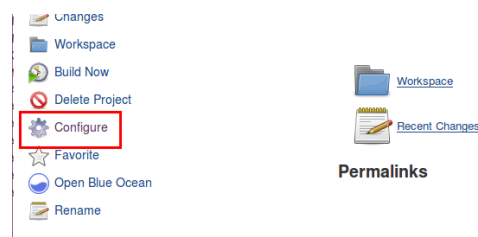
2. Mengisikan kolom item dengan nama project yang akan dibuat, pada penelitian ini digunakan nama *devops_nadzir*, kemudian memilih *Freestyle project*, seperti yang terlihat pada Gambar 4.2, dan mengklik *OK*.



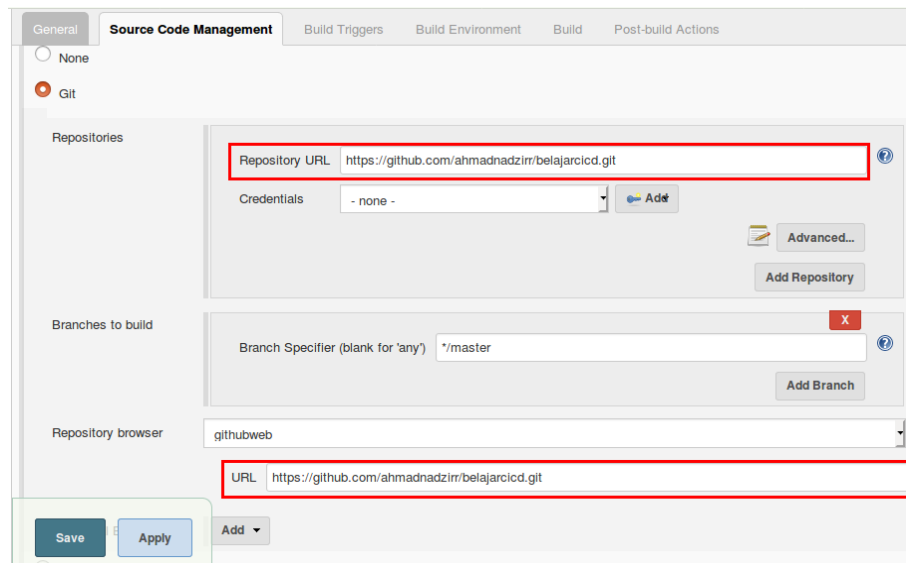
Gambar 4.2 Membuat project-langkah 2

3. Setelah berhasil membuat sebuah *project*, langkah selanjutnya adalah mengintegrasikan project *Github* dengan *Jenkins*, dengan melakukan konfigurasi seperti langkah- langkah berikut.

- a) Memastikan untuk masuk ke *dashboard project* yang ingin diintegrasikan, kemudian memilih *Configure*, seperti yang terlihat pada Gambar 4.3. Setelah itu memilih *Source Code Management*. Mengisi kolom *Repository URL*, seperti yang terlihat pada Gambar 4.4 dengan alamat URL pada project *Github*. Kemudian memeriksa apakah terjadi eror setelah memasukkan alamat URL project *Github*. Memilih *Githubweb* pada *repository browser*, kemudian memasukkan alamat URL project *Github* pada kolom URL. Kemudian memilih *Apply* dan *Save*.

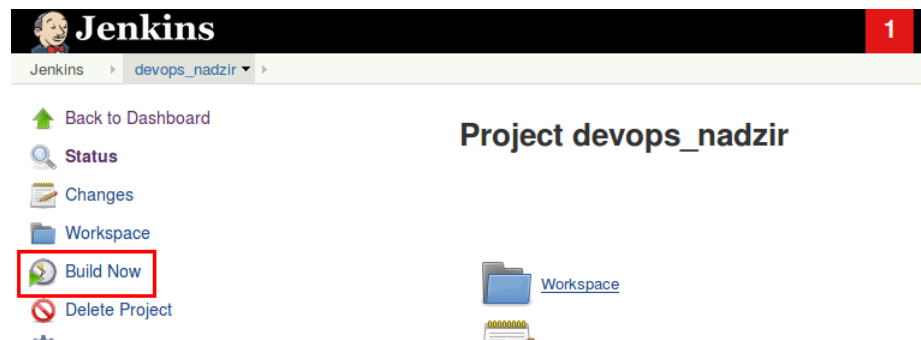


Gambar 4.3 Konfigurasi Jenkins-langkah 1



Gambar 4.4 Konfigurasi Jenkins-repository URL

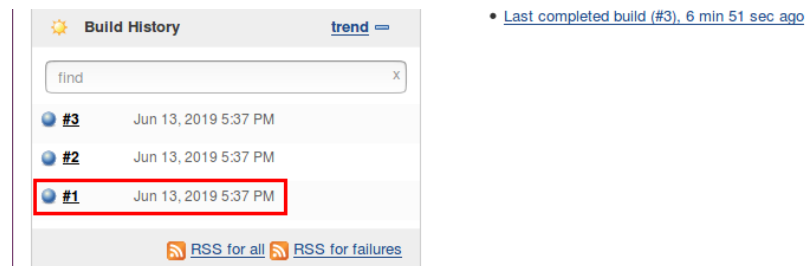
- b) Setelah selesai melakukan konfigurasi *Jenkins* dengan *Github*. Kemudian mengklik *Build Now* pada *dashboard* project *Jenkins*, seperti pada Gambar 4.5 untuk memulai membangun konfigurasi tersebut.



Gambar 4.5 Konfigurasi Jenkins-langkah 3

- c) Setelah mengklik *Build Now*, maka akan muncul *Build History*. Memastikan proses integrasi berhasil dibangun dengan cara melihat *Console Output* dari setiap proses *build* yang dilakukan, seperti yang

terlihat pada gambar Gambar 4.6. Apabila berhasil terdapat tulisan *Finished: SUCCESS*, seperti yang terlihat pada Gambar 4.7.



Gambar 4.6 Konfigurasi Jenkins-build history



Gambar 4.7 Konfigurasi Jenkins-proses build berhasil

4.3 Proses Continuous Deployment Pada Skenario CI/CD

Aplikasi WordPress yang telah berhasil dipasang/*installed* pada *local host*, kemudian disebar (*deploy*) ke *production server*. Pada penelitian ini, *production server* yang digunakan adalah Heroku dan *Docker Hub*. Penerapan *continuous deployment* pada penelitian ini melibatkan beberapa *tools*, *Git*, *Github*, *Jenkins* dan *Docker*. Aplikasi WordPress yang tersimpan pada *localhost* akan disebar (*deploy*) menggunakan *Jenkins*, sehingga prosesnya bisa berjalan secara otomatis dengan memanfaatkan *execute shell* pada *server Jenkins*. *Dockerfile* yang

disimpan pada *repository* akan dieksekusi oleh *Jenkins*, apabila kontainer berhasil dibentuk, kemudian akan tersebar (*deployed*) ke *Docker Hub*. Berikut adalah langkah-langkah untuk melakukan *deployment* aplikasi dengan skenario CI/CD menggunakan *Docker*, *Jenkins* dan *Github*.

1. Penempatan file yang akan dideploy ke *Jenkins* ditempatkan pada *repository Github*, konfigurasi *repository* pada server *Jenkins* dapat dilihat pada Gambar 4.4, dengan menambahkan alamat URL *repository*.
2. Mempersiapkan file aplikasi yang akan dipindahkan ke *repository*. Kemudian membuat sebuah folder kosong pada *localhost*. Penelitian ini menggunakan aplikasi *WordPress* yang dikemas dalam sebuah file *Dockerfile*. File *Dockerfile* dibutuhkan untuk menuliskan perintah-perintah yang akan dieksekusi oleh *Jenkins*. Aplikasi *WordPress* pada penelitian ini terdiri dari 2 file pada *Dockerfile* dan sebuah dependensi *docker-entry.sh*. Selain *Dockerfile*, file/dependensi lain yang dibutuhkan disesuaikan dengan aplikasi yang dibangun.
3. Melakukan *push* file dari *localhost* ke *repository* dengan *Git Control*. Langkah-langkah untuk melakukan *push* file adalah sebagai berikut.
 - a) Menavigaikan terminal pada folder kosong, memastikan file tersalin di dalam folder tersebut, kemudian mengklik kanan dan memilih *Open in Terminal*.
 - b) Meng-*clone repository Github* ke folder *localhost* dengan menuliskan Perintah 4.10.

Perintah 4.10 Clone repository

```
git clone https://github.com/ahmadnadzirr/belajarcicd.git
```

- c) Setelah berhasil diklon, folder *repository* akan tersalin pada *localhost*, kemudian masuk ke folder klon tersebut, kemudian menyalin file ke folder tersebut. Mengklik kanan *Open in Terminal*, lalu menuliskan Perintah 4.11.

Perintah 4.11 Menambahkan file ke repository

```
git add .
```

- d) Kemudian meng-*commit* file tersebut dengan Perintah 4.12, komentar di dalam tanda petik dapat ditulis dengan kata atau kalimat bebas, sesuai dengan kebutuhan.

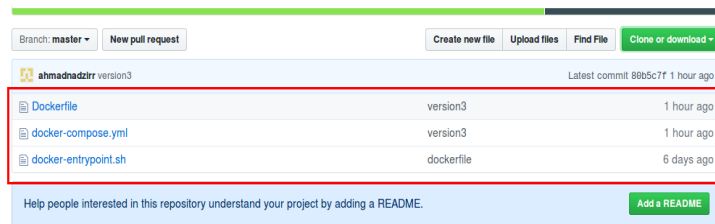
Perintah 4.12 Commit perubahan pada file/folder

```
git commit -m "version1"
```

- e) Setelah berhasil, kemudian mem-*push* file tersebut dengan Perintah 4.13. File berhasil disebar ke *repository* seperti yang terlihat pada Gambar 4.8.

Perintah 4.13 Push file ke repository

```
git push -u origin master
```



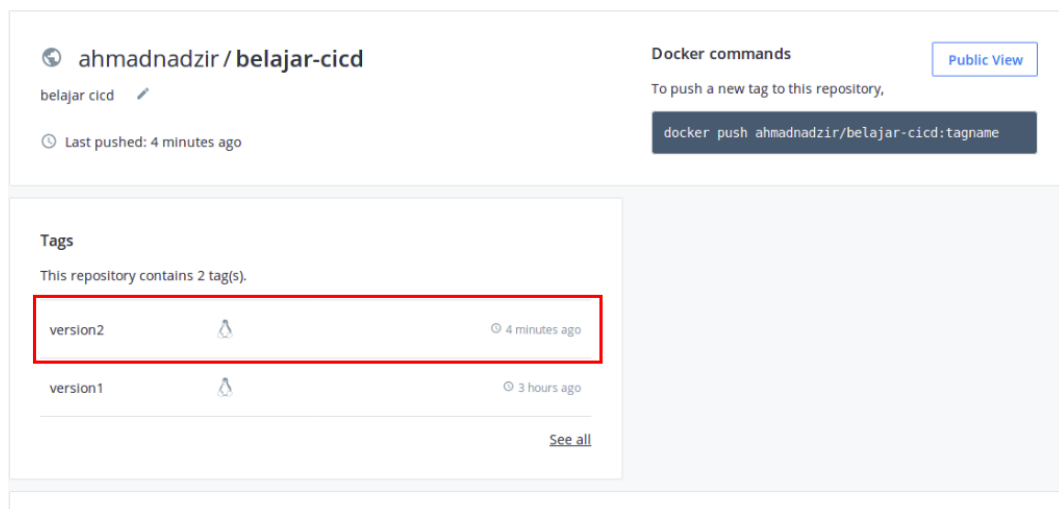
Gambar 4.8 Push (meminahkan) file

4. Melakukan konfigurasi *Github* pada proyek *Jenkins*. Memilih *Configure* kemudian pada bagian *General*, mencentang *Github project*. Pada bagian *Source Code Management*, mencentang *Git*, kemudian mengisikan alamat URL repository *Github*, setelah itu pada bagian *Build Trigger* memilih *GitHub Hook trigger for GITScm polling*.
5. Membuat akun pada *Docker Hub* dan sebuah repository di dalamnya.
6. Langkah terakhir adalah menambahkan langkah *Execute shell* pada bagian *Build*. Kemudian menuliskan *command* yang disesuaikan dengan kebutuhan. Berikut adalah barisan perintah pada *execute shell*, seperti yang terlihat pada Perintah 4.14. Urutan perintah yang dituliskan pada *execute shell* merupakan perintah-perintah yang dilakukan secara manual untuk *deployment* secara manual melalui CLI.

Perintah 4.14 Perintah pada *execute shell*

```
#!/bin/bash
git init
git pull https://github.com/ahmadnadzirr/belajarcicd.git
sudo docker build --tag=belajarcicd .
sudo docker login --username "ahmadnadzir" --password
"romadhon19"
sudo docker tag belajarcicd ahmadnadzir/belajar-cicd
sudo docker push ahmadnadzir/belajar-cicd
```

Ketika proses *build* berhasil, seperti yang terlihat pada Gambar 4.9, kontainer dengan sebuah *tag:version2* berhasil disebar (*deployed*) ke *Docker Hub* pada *repository belajar-cicd*. Proses *build* kontainer pada *Jenkins* dapat dilihat pada Lampiran 2.



Gambar 4.9 deployment kontainer ke Docker Hub

Proses *deployment* tersebut dilakukan secara otomatis menggunakan *Jenkins*, berbeda pada tahap *delivery* aplikasi ke ranah *production server*, yaitu Heroku. Proses *delivery* dilakukan secara manual, dengan menggunakan Heroku CLI. Langkah-langkah untuk memindahkan/*deliver* aplikasi ke ranah *production server* adalah sebagai berikut.

1. Memasang Heroku CLI pada komputer server (*host*), seperti yang terlihat pada Perintah 4.15.

Perintah 4.15 Instal Heroku CLI

```
$ curl https://cli-assets.heroku.com/install-ubuntu.sh | sh
```

2. Membuka akun Heroku dengan menggunakan Perintah 4.16.

Perintah 4.16 Login ke Heroku

```
$ sudo heroku container:login
```

3. Setelah berhasil *login*, meng-*clone repository* yang digunakan dengan Perintah 4.17. Langkah ini dapat dilewatkan, sehingga hanya perlu menavigasikan terminal pada direktori aplikasi yang digunakan untuk melakukan *push* ke *Github*.

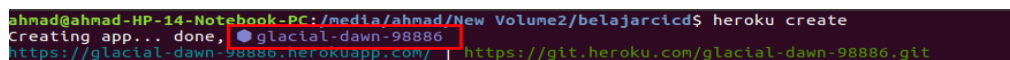
Perintah 4.17 Cloning repository

```
$ git clone https://github.com/ahmadnadzirr/belajarcicd.git
```

4. Kemudian membuat aplikasi Heroku dengan Perintah 4.18, apabila berhasil akan muncul nama aplikasi secara acak, seperti yang terlihat pada Gambar 4.10.

Perintah 4.18 Membuat aplikasi Heroku

```
$ sudo heroku create
```



Gambar 4.10 Nama aplikasi Heroku yang berhasil dibuat

5. Setelah berhasil dibuat, *mem-build* dan *mem-push Docker image* dengan menggunakan Perintah 4.19. Apabila berhasil, maka keluaran pada terminal akan terlihat seperti pada Gambar 4.11.

Perintah 4.19 Build dan push docker image

```
$ heroku container:push web
```

```
d8a33133e477: Pushed
latest: digest: sha256:4a8ea842ca188b3e6808e49c2b4cd4f3ec10bb1a0b10ca106c44ec1071ac707f size: 4704
Your image has been successfully pushed. You can now release it with the 'container:release' command.
ahmad@ahmad-HP-14-Notebook-PC:~/belajarcicd/belajarcicd$ sudo heroku container:release web
```

Gambar 4.11 Build dan push docker image ke container registry

6. Langkah terakhir adalah merilis aplikasi dengan Perintah 4.20. Seperti yang terlihat pada Gambar 4.12, aplikasi berhasil dirilis ke server produksi (Heroku).

Perintah 4.20 Rilis aplikasi

```
$ sudo heroku container:release web
```

```
ahmad@ahmad-HP-14-Notebook-PC:~/media/ahmad/New Volume2/belajarcicd$ sudo heroku container:release web
Releasing images web to glacial-dawn-98886... done
```

Gambar 4.12 Rilis aplikasi ke server produksi

4.4 Pengujian Kompatibilitas dan Efektivitas *Docker*

Untuk melakukan pengujian penyimpanan terhadap penggunaan sumber daya dilakukan dengan menghitung jumlah *resource* yang terpakai pada partisi yang digunakan. Partisi yang digunakan untuk menjalankan *docker container* adalah sda4, sehingga perintah yang digunakan adalah `df /dev/sda4`. Penggunaan penyimpanan sebelum menjalankan *Docker container* adalah 10590356 KB (10590,356 MB), seperti yang terlihat pada Gambar 4.13. Kemudian, *docker image* dibuat dengan menjalankan Perintah 4.21 pada file *docker-compose.yml*. Daftar/list kontainer yang berjalan dan *docker images* yang berhasil dibuat pada penelitian ini dapat dilihat pada Gambar 4.14. Penggunaan penyimpanan setelah menjalankan kontainer dengan Perintah 4.22 pada blok sd4 mengalami kenaikan sebesar 10745372 KB (10745,372 MB). Sehingga empat buah *Docker image* (*WordPress*, *apache2*, *mysql:5.7* dan *phpmyadmin*) yang digunakan untuk

menjalankan aplikasi *WordPress* ini menggunakan penyimpanan sebesar 155,016 MB. Apabila dibandingkan dengan penggunaan *resource* jika memasang *WordPress* secara manual (tanpa menggunakan *Docker image*), penggunaan penyimpanan mengalami perubahan yang signifikan, yaitu sebesar 636,232 MB, seperti yang dapat dilihat Tabel 3.3.

Perintah 4.21 *docker-compose build*

```
sudo docker-compose build
```

Perintah 4.22 *docker-compose up*

```
Sudo docker-compose up -d
```

```
ahmad@ahmad-HP-14-Notebook-PC:~/belajar ckd$ df /dev/sda4
Filesystem      1K-blocks    Used Available Use% Mounted on
/dev/sda4        80501116 10590356  65798380  14% /
ahmad@ahmad-HP-14-Notebook-PC:~/belajar ckd$ sudo docker-compose up -d
[sudo] password for ahmad:
Pulling db (mysql:5.7)...
5.7: Pulling from library/mysql
```

Gambar 4.13 Jumlah utilitas penyimpanan pada blok sda4 sebelum menjalankan kontainer

```
ahmad@ahmad-HP-14-Notebook-PC:~/belajar ckd$ sudo docker image ls
REPOSITORY      TAG         IMAGE ID      CREATED       SIZE
belajarcld_web  latest      62b4d8e77c3f  5 minutes ago 492MB
php             7.1-apache  6e2b5270ea5c  7 days ago   405MB
mysql          5.7        f6509bac4980  7 days ago   373MB
phpmyadmin/phpmyadmin latest      026319eaebed  7 weeks ago  421MB
ahmad@ahmad-HP-14-Notebook-PC:~/belajar ckd$ sudo docker run -d -p 8001:80 belajarcld_web
abc48eed53f2d80b19ad7fbc68092b0e32f055f4a008f445197b0ebc66
ahmad@ahmad-HP-14-Notebook-PC:~/belajar ckd$ sudo docker container ls
CONTAINER ID   IMAGE          COMMAND                  CREATED       STATUS      PORTS                    NAMES
abc48eed535k   belajarcld_web "docker-entrypoint.s..." 17 seconds ago Up 8 seconds 0.0.0.0:8001->80/tcp      brave_gollic
ad3764aa7a97   phpmyadmin/phpmyadmin "/docker-entrypoint.s..." 3 hours ago   Up 8 minutes 0.0.0.0:8080->80/tcp      belajarcld
abc48eed535k   mysql:5.7      "docker-entrypoint.s..." 3 hours ago   Up 8 minutes 3306/tcp, 33060/tcp      belajarcld
ahmad@ahmad-HP-14-Notebook-PC:~/belajar ckd$ df /dev/sda4
Filesystem      1K-blocks    Used Available Use% Mounted on
/dev/sda4        80501116 10745372  65643364  15% /
ahmad@ahmad-HP-14-Notebook-PC:~/belajar ckd$
```

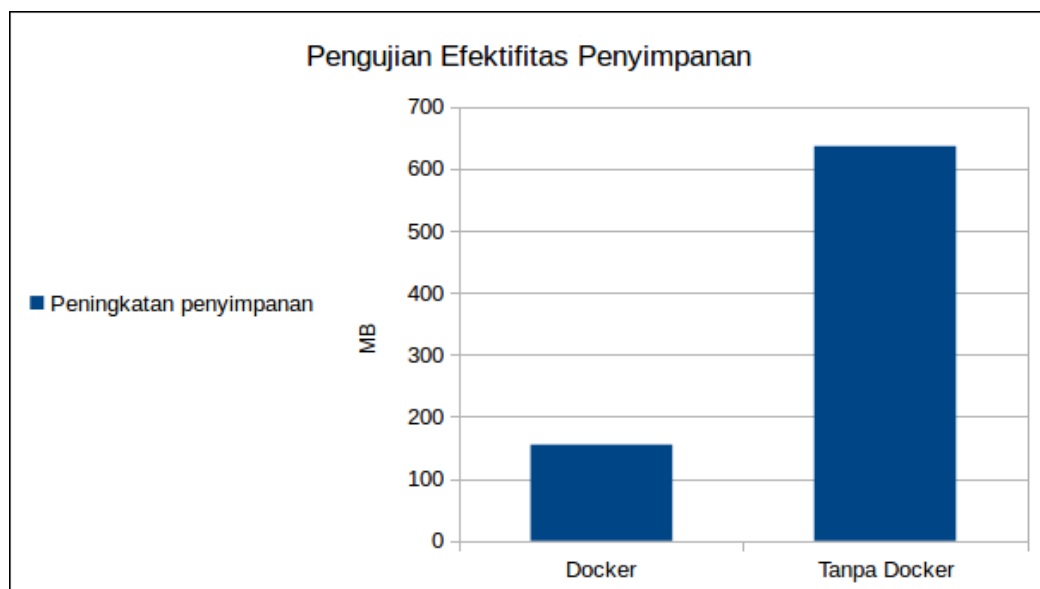
Gambar 4.14 Jumlah utilitas penyimpanan pada blok sda4 sesudah menjalankan kontainer

Penyimpanan pada *Docker* jauh lebih efektif jika dibandingkan dengan tidak menggunakan *Docker*. *Docker layering* menjadi solusi untuk menangani masalah penggunaan *resource* yang berlebihan dengan mekanisme *copy-on-write*, di mana ketika ada sebuah objek baru yang dibuat, tidak semua data

tersalin, hanya perubahan data pada objek tersebut saja yang tersalin di dalam ruang *disk* baru menjadi sebuah *layer*. *Layer-layer* tersebut dapat digunakan pada beberapa kontainer yang berbeda, sehingga dapat mengurangi penggunaan *resource*. Keluaran *console* pada proses *build Docker image* dan *container* dapat dilihat pada Lampiran 3.

Tabel 4.1 Hasil sebelum dan sesudah melakukan *update resource list*

CMS	Analisis			
	Kompatibilitas	Perbandingan Penggunaan Ruang Penyimpanan		
		Sebelum instalasi (KB)	Sesudah instalasi (KB)	Peningkatan Penyimpanan (MB) Hasil = Sesudah - Sebelum
Container WordPress (Docker)	Kompatibel	10590356	10745372	155,016
WordPress	-	8779984	9416464	636,232

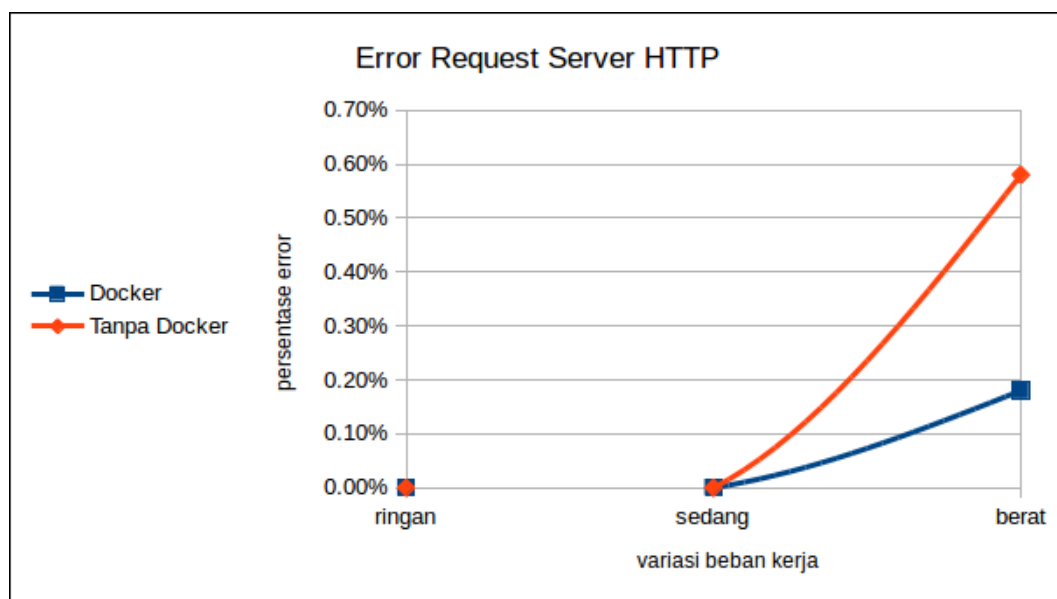


Gambar 4.15 Grafik perbandingan penggunaan penyimpanan

hasil uji kinerja menggunakan variabel *error request* dapat dilihat pada Tabel 4.2. Terdapat perbedaan tingkat *error request* pada skenario pengujian beban kerja berat, di mana aplikasi *WordPress* dengan kontainer lebih unggul dibandingkan dengan tanpa kontainer, meskipun tidak terlihat perbedaan yang sangat signifikan.

Tabel 4.2 Error Request Server HTTP CMS dalam persentase

CMS	Analisis		
	Load Test		
	ringan	sedang	berat
	% error		
Container WordPress (Docker)	0.00%	0.00%	0.18%
WordPress	0.00%	0.00%	0.58%



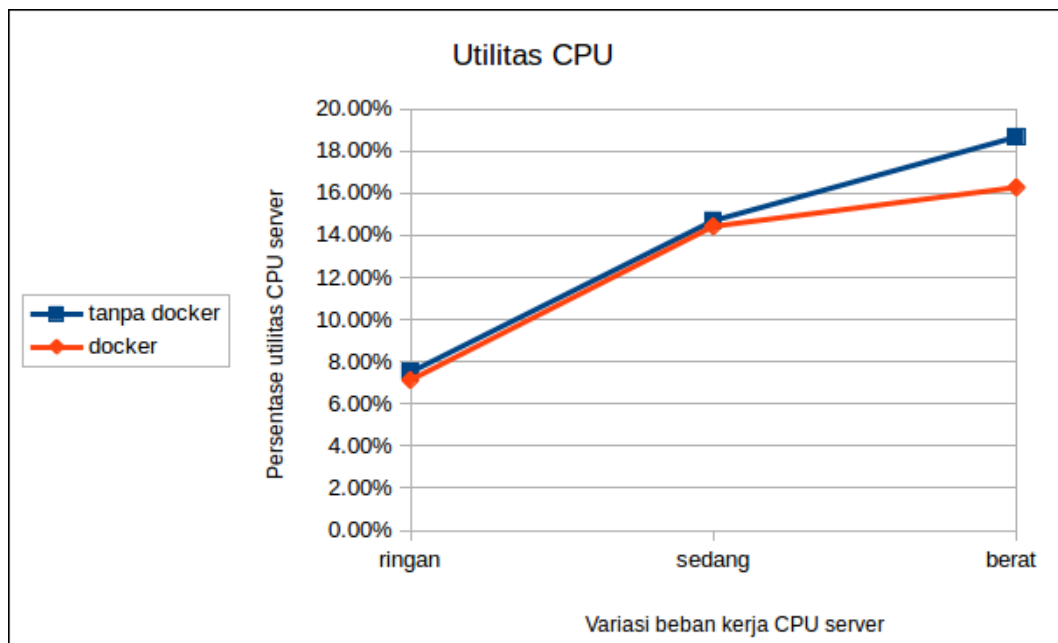
Gambar 4.17 Grafik error request server HTTP

Sedangkan hasil pengujian tingkat utilitas CPU server dapat dilihat pada Tabel 4.3. Data diambil dengan mengakses server web melalui SSH, kemudian menjalankan perintah *sar 1 10*. Data tersebut menunjukkan utilitas CPU yang diuji pada server *Docker* lebih unggul, baik pada beban kerja ringan, sedang maupun berat. Walaupun masing-masing server memberikan kinerja yang baik,

yaitu tidak melebihi 20% dari utilitas CPU total, seperti yang terlihat pada Gambar 4.18.

Tabel 4.3 Utilitas CPU server (Host) layanan HTTP request dalam persentase

CMS	Analisis		
	Penggunaan/utilitas CPU Server		
	ringan	sedang	berat
	100% -dengan rata-rata persentase CPU dalam keadaan <i>idle</i>		
Container WordPress (Docker)	7.13%	14.43%	16.28%
WordPress	7.51%	14.70%	18.67%



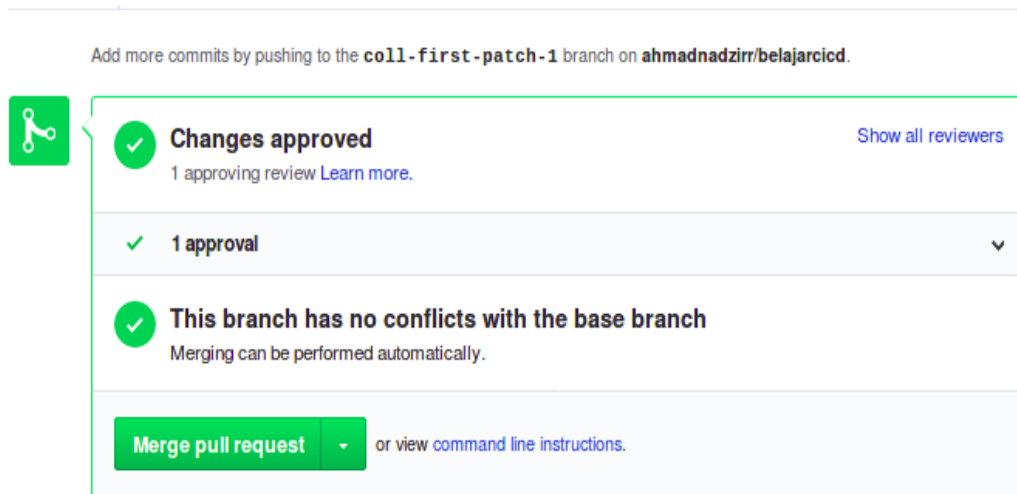
Gambar 4.18 Grafik perbandingan utilitas CPU server HTTP

4.6 Manfaat Penggunaan Docker Pada Infrastruktur DevOps

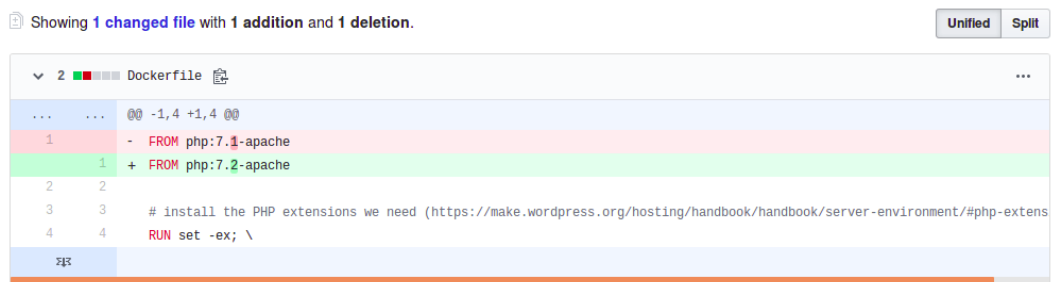
Penggunaan *Docker* pada ranah *development* memberikan banyak keuntungan apabila dibandingkan dengan tidak menggunakan *Docker*. Sebagai contoh penggunaan *Docker compose* pada penelitian ini. Proses Instalasi LAMP (*Linux, Apache, Mysql,PHP*) stack hanya perlu dilakukan dengan menginstal

Docker compose, kemudian membuat satu buah file *docker compose* dengan format file *.yaml*, seperti yang terlampir pada Program 4.1. Proses *running* masing-masing *service* dapat dilakukan dengan mudah, dengan menjalankan Perintah 4.22, sehingga dengan penggunaan *Docker compose*, masing-masing *service* beserta dependensinya dapat terisolasi dan terkonfigurasi dengan mudah dalam satu file.

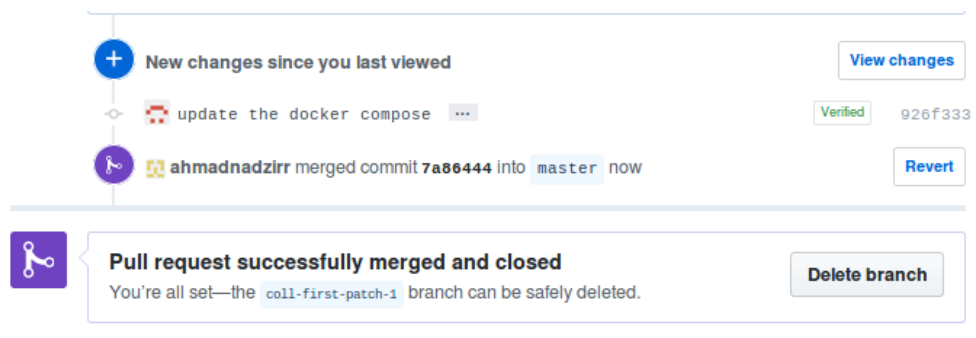
Penggunaan infrastruktur DevOps juga memberikan manfaat yang besar untuk pengembang pada tahap *development*, dengan penerapan praktik *continuous integration*, beberapa *developers* dapat berkolaborasi dalam satu *repository* sumber kode. Pada penelitian ini, disimulasikan terdapat 2 *developers* yang berkolaborasi dalam *repository belajarcicd*, terdapat dua *branch*, yaitu *master branch* dan satu *branch* kontributor, *coll-first-patch-1*. Kontributor tersebut melakukan *pull request* untuk melakukan perubahan pada sumber kode yang ada di *branch master*, seperti yang terlihat pada Gambar 4.19. Perubahan yang dilakukan dapat dilihat pada Gambar 4.20. Apabila *branch master* menyetujui *pull request* tersebut, seperti yang terlihat pada Gambar 4.21, maka file pada *branch master* akan terbaru.



Gambar 4.19 Pull request dari branch coll-first-patch-1



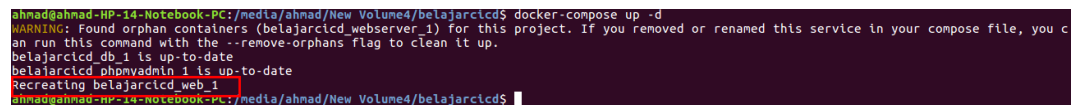
Gambar 4.20 Review perubahan pada file dockerfile



Gambar 4.21 Pull request approval

4.7 Redeploy WordPress Container

Aplikasi *WordPress* akan terbaru jika proses *redeploy* dilakukan, akan tetapi untuk menjalankan kontainer yang terbaru, kontainer lama harus dihentikan, kemudian menjalankan kontainer baru. Data aplikasi terdapat pada sebuah variabel *volume*, sebuah *repository* data yang digunakan oleh beberapa *service* kontainer yang didefinisikan pada penelitian ini, sehingga setiap terjadi perubahan, sebuah kontainer baru akan terbentuk dari satu *base image*. *Docker container* akan terbaru seperti yang terlihat pada Gambar 4.22.



```

ahmad@ahmad-HP-14-Notebook-PC:/media/ahmad/New_Volume4/belajarctcd$ docker-compose up -d
WARNING: Found orphan containers (belajarctcd_webserver_1) for this project. If you removed or renamed this service in your compose file, you can run this command with the --remove-orphans flag to clean it up.
belajarctcd_db_1 is up-to-date
belajarctcd_phpmyadmin_1 is up-to-date
Recreating belajarctcd_web_1
ahmad@ahmad-HP-14-Notebook-PC:/media/ahmad/New_Volume4/belajarctcd$

```

Gambar 4.22 Recreate docker container

BAB 5

KESIMPULAN DAN SARAN

Kesimpulan dan Saran pada penelitian ini memuat hasil penelitian secara singkat dan jelas sesuai dengan tujuan penelitian.

5.1 Kesimpulan

Berdasarkan hasil penelitian, diperoleh kesimpulan sebagai berikut.

1. Sebuah *custom* file, yaitu *Dockerfile* dibutuhkan pada infrastruktur *continuous deployment* menggunakan perangkat *Docker* yang terintegrasi dengan *Git* dan *Jenkins*. Karena *Docker compose* hanya berfungsi untuk menjalankan *service* pada *localhost* yang menggunakan sebuah *default docker image* dari *Docker Hub*.
2. Aplikasi yang dikemas dengan *Docker container* menggunakan lebih sedikit *resource* penyimpanan jika dibandingkan dengan tidak menggunakan *Docker container*, 24% lebih ringan (studi kasus instalasi aplikasi *WordPress*) , dan menggunakan lebih sedikit *resource* CPU ketika dilakukan uji *load test*, 2.8% lebih baik pada studi kasus yang sama.
3. *Docker* sangat efektif untuk mengatasi permasalahan *conflicting dependencies*, sebuah *Docker container* akan mengisolasi semua dependensi aplikasi di dalam sebuah lingkungan virtual yang sangat ringan, karena masing-masing dari kontainer saling berbagi kernel/OS.
4. Kontainer *database* tidak dapat disebar (*deploy*) secara independen, melainkan dikemas dalam satu kontainer dengan aplikasi. Pemisahan

kontainer *database* dan kontainer aplikasi dapat dilakukan pada ranah *local development* dengan menggunakan *Docker compose*, yang didefinisikan sebagai sebuah *service*.

5.2 Saran

Berdasarkan hasil penelitian dan pembahasan pada bab-bab sebelumnya, maka penulis mengajukan saran sebagai berikut.

1. Penggunaan *Jenkins* untuk proses *automation building* dapat digantikan dengan fitur CI/CD yang tersedia pada *Gitlab*. *Gitlab* juga menyediakan semua proses pada satu *pipeline (building, testing, deploying)*, sehingga tidak membutuhkan terlalu banyak integrasi dengan perangkat lain.
2. Pemisahan antara komputer *server* dengan komputer *client* sangat dianjurkan pada dua perangkat keras yang berbeda untuk meningkatkan kinerja CPU, sehingga *resource* yang terpakai, tidak berpengaruh pada proses pengujian dan analisis hasil yang didapatkan dapat lebih akurat.
3. Penggunaan aplikasi web dinamis pada studi kasus penelitian selanjutnya dapat menjelaskan skenario penerapan infrastruktur *DevOps* yang lebih detail di setiap tahapan pada siklus pengembangan sebuah perangkat lunak/*software*.

DAFTAR PUSTAKA

- [1] “What is a Web Application?” [Daring]. Tersedia pada: <https://www.maxcdn.com/one/visual-glossary/web-application/> [Diakses: 08-Mar-2019].
- [2] S. H. Prayoga dan D. I. Sensuse, “Analisis usability pada aplikasi berbasis web dengan mengadopsi model kepuasan pengguna (user satisfaction),” *J. Sist. Inf. MTI-UI Jkt.*, 2010.
- [3] admin, “Manfaat Aplikasi Web Based,” *Ruhi Desain*, 24-Agu-2017. .
- [4] L. Ellen dan Lwakatare, *DevOps adoption and implementation in software development practice : concept, practices, benefits and challenges*. Oulu : University of Oulu, 2017.
- [5] R. Bik dan M. Fadlulloh, “Implementasi Docker Untuk Pengelolaan Banyak Aplikasi Web (Studi Kasus: Jurusan Teknik Informatika Unesa),” *J. Manaj. Inform.*, vol. 7, no. 2, 2017.
- [6] J. Sahoo, S. Mohapatra, dan R. Lath, “Virtualization: A survey on concepts, taxonomy and associated security issues,” dalam *2010 Second International Conference on Computer and Network Technology*, 2010, hlm. 222–226.
- [7] M. J. Scheepers, “Virtualization and containerization of application infrastructure: A comparison,” dalam *21st twente student conference on IT*, 2014, vol. 1, hlm. 1–7.
- [8] T. Azzam, “Container VS VM (Virtual Machine),” *Core Network Laboratory Tech Page*, 22-Sep-2018. .
- [9] C. Anderson, “Docker [software engineering],” *IEEE Softw.*, vol. 32, no. 3, hlm. 102-c3, 2015.
- [10] “DEVOPS AND AGILE DEVELOPMENT,” *vmware*, Apr 2017.
- [11] ReactiveOps, “The Benefits of using Docker for Development and Operations,” *Medium*, 28-Sep-2017. .
- [12] S. Apridayanti, I. Isnawaty, dan R. A. Saputra, “DESAIN DAN IMPLEMENTASI VIRTUALISASI BERBASIS DOCKER UNTUK DEPLOYMENT APLKASI WEB,” *semanTIK*, vol. 4, no. 2, 2018.
- [13] R. Bik dan M. Fadlulloh, “Implementasi Docker Untuk Pengelolaan Banyak Aplikasi Web (Studi Kasus: Jurusan Teknik Informatika Unesa),” *J. Manaj. Inform.*, vol. 7, no. 2, 2017.
- [14] Y. T. Sumbogo, M. Data, dan R. Andria Siregar, “Implementasi Failover Dan Autoscaling Kontainer Web Server Nginx Pada Docker Menggunakan Kubernetes,” vol. 2, hlm. 6849–6854, Desember 2018.
- [15] “Virtualization Technology & Virtual Machine Software: What is Virtualization?,” *VMWare*. [Daring]. Tersedia pada: <https://www.vmware.com/solutions/virtualization.html>. [Diakses: 19-Mar-2019].
- [16] “Keuntungan Teknologi Virtualisasi & Cloud Computing – PT. Excellent Infotama Kreasindo.” [Daring]. Tersedia pada: <https://www.excellent.co.id/product-services/vmware/keuntungan-teknologi-virtualisasi-cloud-computing/>. [Diakses: 19-Mar-2019].

- [17] “Konsep Dasar Virtualisasi,” *School of Information Systems*. [Daring]. Tersedia pada: <https://sis.binus.ac.id/2014/10/11/konsep-dasar-virtualisasi/>. [Diakses: 19-Mar-2019].
- [18] “What is virtualization architecture? - Definition from WhatIs.com,” *WhatIs.com*. [Daring]. Tersedia pada: <https://whatis.techtarget.com/definition/virtualization-architecture>. [Diakses: 19-Mar-2019].
- [19] U. Kimfa, “Mesin Virtual.”
- [20] K. Shaw, “What is a hypervisor?,” *Network World*, 19-Des-2017. [Daring]. Tersedia pada: <https://www.networkworld.com/article/3243262/what-is-a-hypervisor.html>. [Diakses: 23-Mar-2019].
- [21] “Get Started, Part 1: Orientation and setup,” *Docker Documentation*, 22-Mar-2019. [Daring]. Tersedia pada: <https://docs.docker.com/get-started/>. [Diakses: 23-Mar-2019].
- [22] “What is guest OS (guest operating system)? - Definition from WhatIs.com,” *SearchServerVirtualization*. [Daring]. Tersedia pada: <https://searchservervirtualization.techtarget.com/definition/guest-OS>. [Diakses: 23-Mar-2019].
- [23] C. Pahl, “Containerization and the paas cloud,” *IEEE Cloud Comput.*, vol. 2, no. 3, hlm. 24–31, 2015.
- [24] “What’s the Difference Between Containers and Virtual Machines? (.PDF Download),” *Electronic Design*, 15-Jul-2016. [Daring]. Tersedia pada: <https://www.electronicdesign.com/datasheet/what-s-difference-between-containers-and-virtual-machines-pdf-download>. [Diakses: 24-Mar-2019].
- [25] “Linux Containers - LXC - Introduction.” [Daring]. Tersedia pada: <https://linuxcontainers.org/lxc/introduction/>. [Diakses: 25-Mar-2019].
- [26] “Docker overview,” *Docker Documentation*, 22-Mar-2019. [Daring]. Tersedia pada: <https://docs.docker.com/engine/docker-overview/>. [Diakses: 25-Mar-2019].
- [27] I. Miell dan A. H. Sayers, *Docker in practice*. Manning Publications Co., 2016.
- [28] B. B. Rad, H. J. Bhatti, dan M. Ahmadi, “An introduction to docker and analysis of its performance,” *Int. J. Comput. Sci. Netw. Secur. IJCSNS*, vol. 17, no. 3, hlm. 228, 2017.
- [29] L. Zhu, L. Bass, dan G. Champlin-Scharff, “DevOps and its practices,” *IEEE Softw.*, vol. 33, no. 3, hlm. 32–34, 2016.
- [30] M. Shahin, M. A. Babar, dan L. Zhu, “Continuous integration, delivery and deployment: a systematic review on approaches, tools, challenges and practices,” *IEEE Access*, vol. 5, hlm. 3909–3943, 2017.
- [31] A. Taryana, I. Setiawan, A. Fadli, dan E. Murdyantoro, “Pioneering the automation of Internal quality assurance system of higher education (IQAS-HE) using DevOps approach,” dipresentasikan pada 2017 International Conference on Sustainable Information Engineering and Technology (SIET), 2017, hlm. 259–264.
- [32] “DevOps Tools: How To Orchestrate Them To Solve Our Problems,” *Edureka*, 18-Okt-2017. .

- [33] M. J. Scheepers, "Virtualization and containerization of application infrastructure: A comparison," dipresentasikan pada 21st twente student conference on IT, 2014, vol. 1, hlm. 1–7.
- [34] R. Rasian dan P. Mursanto, "Perbandingan Kinerja Pendekatan Virtualisasi," *J. Sist. Inf.*, vol. 5, no. 2, hlm. 90–99, 2009.
- [35] C. David Graziano, "A performance analysis of Xen and KVMhypervisors for hosting the Xen Worlds Project," *Iowa State Univ.*, 2011.

LAMPIRAN

Lampiran 1. Dockerfile

```
FROM php:7.1-apache

# install the PHP extensions we need
(https://make.WordPress.org/hosting/handbook/handbook/server-environment/#php-extensions)
RUN set -ex; \
    \
    savedAptMark="$(apt-mark showmanual)"; \
    \
    apt-get update; \
    apt-get install -y --no-install-recommends \
        libjpeg-dev \
        libmagickwand-dev \
        libpng-dev \
    ; \
    \
    docker-php-ext-configure gd --with-png-dir=/usr --with-jpeg-dir=/usr; \
    docker-php-ext-install -j "$(nproc)" \
        bcmath \
        exif \
        gd \
        mysqli \
        opcache \
        zip \
    ; \
    pecl install imagick-3.4.4; \
    docker-php-ext-enable imagick; \
    \
    # reset apt-mark's "manual" list so that "purge --auto-remove"
    # will remove all build dependencies
    apt-mark auto '.*' > /dev/null; \
    apt-mark manual $savedAptMark; \
    ldd "$(php -r 'echo ini_get("extension_dir");')'/*.so \
        | awk '/=>/ { print $3 }' \
        | sort -u \
        | xargs -r dpkg-query -S \
        | cut -d: -f1 \
        | sort -u \
        | xargs -rt apt-mark manual; \
    \
    apt-get purge -y --auto-remove -o
APT::AutoRemove::RecommendsImportant=false; \
rm -rf /var/lib/apt/lists/*

# set recommended PHP.ini settings
# see https://secure.php.net/manual/en/opcache.installation.php
RUN { \
    \
    echo 'opcache.memory_consumption=128'; \
    echo 'opcache.interned_strings_buffer=8'; \
    echo 'opcache.max_accelerated_files=4000'; \
    echo 'opcache.revalidate_freq=2'; \
    echo 'opcache.fast_shutdown=1'; \
    } > /usr/local/etc/php/conf.d/opcache-recommended.ini
# https://codex.WordPress.org/Editing\_wp-config.php#Configure\_Error\_Logging
RUN { \
```

```

        echo 'error_reporting = 4339'; \
        echo 'display_errors = Off'; \
        echo 'display_startup_errors = Off'; \
        echo 'log_errors = On'; \
        echo 'error_log = /dev/stderr'; \
        echo 'log_errors_max_len = 1024'; \
        echo 'ignore_repeated_errors = On'; \
        echo 'ignore_repeated_source = Off'; \
        echo 'html_errors = Off'; \
    } > /usr/local/etc/php/conf.d/error-logging.ini

RUN a2enmod rewrite expires

VOLUME /var/www/html/belajarcicd/wp-content
ENV WordPress_DB_NAME: WordPress
ENV WordPress_DB_USERNAME: root
ENV WordPress_DB_HOST: db:3306
ENV WordPress_DB_PASSWORD: root
ENV WordPress_VERSION 5.2.2
ENV WordPress_SHA1 3605bcbe9ea48d714efa59b0eb2d251657e7d5b0

RUN set -ex; \
    curl -o WordPress.tar.gz -fSL
    "https://WordPress.org/WordPress-${WordPress_VERSION}.tar.gz"; \
    echo "${WordPress_SHA1} *WordPress.tar.gz" | sha1sum -c
-; \
# upstream tarballs include ./WordPress/ so this gives us
/usr/src/WordPress
    tar -xzf WordPress.tar.gz -C /usr/src/; \
    rm WordPress.tar.gz; \
    chown -R www-data:www-data /usr/src/WordPress

COPY docker-entrypoint.sh /usr/local/bin/
RUN ["chmod", "+x", "/usr/local/bin/docker-entrypoint.sh"]

ENTRYPOINT ["docker-entrypoint.sh"]
CMD ["apache2-foreground"]

```

Lampiran 2. Console outout proses buil dan deploy kontainer pada Jenkins

```

Started by user DEVOPS FT UNSOED
Running as SYSTEM
Building in workspace /var/lib/jenkins/workspace/belajarcicd
No credentials specified
> git rev-parse --is-inside-work-tree # timeout=10
Fetching changes from the remote Git repository
> git config remote.origin.url
https://github.com/ahmadnadzirr/belajarcicd.git # timeout=10
Fetching upstream changes from
https://github.com/ahmadnadzirr/belajarcicd.git
> git --version # timeout=10
> git fetch --tags --progress
https://github.com/ahmadnadzirr/belajarcicd.git

```

```

+refs/heads/*:refs/remotes/origin/*
> git rev-parse refs/remotes/origin/master^{commit} # timeout=10
> git rev-parse refs/remotes/origin/origin/master^{commit} #
timeout=10
Checking out Revision 80b5c7fd74a040bd3ea1cbe662bd22ace182f57a
(refs/remotes/origin/master)
> git config core.sparsecheckout # timeout=10
> git checkout -f 80b5c7fd74a040bd3ea1cbe662bd22ace182f57a
Commit message: "version3"
> git rev-list --no-walk
80b5c7fd74a040bd3ea1cbe662bd22ace182f57a # timeout=10
[belajarcid] $ /bin/bash /tmp/jenkins6135334901186211570.sh
Reinitialized existing Git repository in
/var/lib/jenkins/workspace/belajarcid/.git/
From https://github.com/ahmadnadzirr/belajarcid
* branch          HEAD          -> FETCH_HEAD
Already up to date.
Sending build context to Docker daemon 99.84kB

Step 1/17 : FROM php:7.1-apache
---> 710b2d810b66
Step 2/17 : RUN set -ex;                                savedAptMark="$(apt-mark
showmanual)";                                apt-get update;                                apt-get
install -y --no-install-recommends                                libjpeg-dev
libmagickwand-dev                                libpng-dev                                ;
docker-php-ext-configure gd --with-png-dir=/usr --with-jpeg-
dir=/usr;                                docker-php-ext-install -j "$(nproc)"
bcmath                                exif                                gd
mysqli                                opcache                                zip                                ;
pecl install imagick-3.4.4;                                docker-php-ext-enable
imagick;                                apt-mark auto '.*' > /dev/null;
apt-mark manual $savedAptMark;                                ldd "$(php -r 'echo
ini_get("extension_dir");')/*.*.so                                | awk '/=>/
{ print $3 }'                                | sort -u                                | xargs
-r dpkg-query -S                                | cut -d: -f1                                |
sort -u                                | xargs -rt apt-mark manual;
apt-get purge -y --auto-remove -o
APT::AutoRemove::RecommendsImportant=false;                                rm -rf
/var/lib/apt/lists/*
---> Using cache
---> 5ba9ad148e16
Step 3/17 : RUN {                                echo
'opcache.memory_consumption=128';                                echo
'opcache.interned_strings_buffer=8';                                echo
'opcache.max_accelerated_files=4000';                                echo
'opcache.revalidate_freq=2';                                echo
'opcache.fast_shutdown=1';                                } >
/usr/local/etc/php/conf.d/opcache-recommended.ini
---> Using cache
---> e130dcf63aeb
Step 4/17 : RUN {                                echo 'error_reporting = 4339';
echo 'display_errors = Off';                                echo
'display_startup_errors = Off';                                echo 'log_errors
= On';                                echo 'error_log = /dev/stderr';
echo 'log_errors_max_len = 1024';                                echo
'ignore_repeated_errors = On';                                echo
'ignore_repeated_source = Off';                                echo 'html_errors
= Off';                                } > /usr/local/etc/php/conf.d/error-logging.ini
---> Using cache
---> 623479333e3c
Step 5/17 : RUN a2enmod rewrite expires
---> Using cache
---> c1470b5bdfd8

```



```

Step 6/17 : VOLUME /var/www/html
----> Using cache
----> 9132a6de3b2d
Step 7/17 : ENV WordPress_DB_NAME: WordPress
----> Using cache
----> 356fb94c6302
Step 8/17 : ENV WordPress_DB_USERNAME: root
----> Using cache
----> 164343a76964
Step 9/17 : ENV WordPress_DB_HOST: db:3306
----> Using cache
----> 236471331fe8
Step 10/17 : ENV WordPress_DB_PASSWORD: root
----> Using cache
----> 1281f078bc1e
Step 11/17 : ENV WordPress_VERSION 5.2.2
----> Using cache
----> 156106ceccdf
Step 12/17 : ENV WordPress_SHA1
3605bcbe9ea48d714efa59b0eb2d251657e7d5b0
----> Using cache
----> a59d9855daf6
Step 13/17 : RUN set -ex; curl -oWordPress.tar.gz -fSL
"https://WordPress.org/WordPress-${WordPress_VERSION}.tar.gz";
echo "$WordPress_SHA1 *WordPress.tar.gz" | sha1sum -c -;
tar -xzf WordPress.tar.gz -C /usr/src/; rm
WordPress.tar.gz; chown -R www-data:www-data
/usr/src/WordPress
----> Using cache
----> 8b56d619ab5e
Step 14/17 : COPY docker-entrypoint.sh /usr/local/bin/
----> Using cache
----> 38dae049688e
Step 15/17 : RUN ["chmod", "+x", "/usr/local/bin/docker-
entrypoint.sh"]
----> Using cache
----> d2565359f920
Step 16/17 : ENTRYPOINT ["docker-entrypoint.sh"]
----> Using cache
----> 09578e78a260
Step 17/17 : CMD ["apache2-foreground"]
----> Using cache
----> f84e6d7721b1
Successfully built f84e6d7721b1
Successfully tagged belajar-cicd:latest
WARNING! Using --password via the CLI is insecure. Use
--password-stdin.
WARNING! Your password will be stored unencrypted in
/var/lib/jenkins/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
The push refers to repository [docker.io/ahmadnadzir/belajar-
cicd]
db3eb6d56e83: Preparing
2b4a9d2298ac: Preparing
747e9c78a3f5: Preparing
b7d1665b7c06: Preparing
d4b880eafd57: Preparing
dac968e5ba1b: Preparing
51bc9ad8f58a: Preparing

```

```

7c4ac87d038e: Preparing
0c347b36628d: Preparing
bb39723e34a7: Preparing
8a8734644be2: Preparing
847528dab20f: Preparing
d35c07543039: Preparing
48a37b97f339: Preparing
dcf2066a324d: Preparing
eb20543b3ab7: Preparing
d8fa1bbc8af1: Preparing
512d83aca3f9: Preparing
d8a33133e477: Preparing
dac968e5ba1b: Waiting
d35c07543039: Waiting
48a37b97f339: Waiting
dcf2066a324d: Waiting
eb20543b3ab7: Waiting
51bc9ad8f58a: Waiting
7c4ac87d038e: Waiting
8a8734644be2: Waiting
0c347b36628d: Waiting
d8fa1bbc8af1: Waiting
512d83aca3f9: Waiting
bb39723e34a7: Waiting
847528dab20f: Waiting
d8a33133e477: Waiting
d4b880eafd57: Layer already exists
747e9c78a3f5: Layer already exists
2b4a9d2298ac: Layer already exists
b7d1665b7c06: Layer already exists
db3eb6d56e83: Layer already exists
51bc9ad8f58a: Layer already exists
dac968e5ba1b: Layer already exists
7c4ac87d038e: Layer already exists
0c347b36628d: Layer already exists
bb39723e34a7: Layer already exists
8a8734644be2: Layer already exists
847528dab20f: Layer already exists
48a37b97f339: Layer already exists
d35c07543039: Layer already exists
dcf2066a324d: Layer already exists
d8fa1bbc8af1: Layer already exists
eb20543b3ab7: Layer already exists
512d83aca3f9: Layer already exists
d8a33133e477: Layer already exists
latest: digest:
sha256:ca8e6cdcd2573a8eb62910c7d34a50c603e1274885e717c56df0f5e60a
824082 size: 4289
40da4e860053: Preparing
b60cf95f6699: Preparing
9003404e64e0: Preparing
b7d1665b7c06: Preparing
d4b880eafd57: Preparing
dac968e5ba1b: Preparing
51bc9ad8f58a: Preparing
7c4ac87d038e: Preparing
0c347b36628d: Preparing
bb39723e34a7: Preparing
8a8734644be2: Preparing
847528dab20f: Preparing
d35c07543039: Preparing
48a37b97f339: Preparing
dcf2066a324d: Preparing

```

```

eb20543b3ab7: Preparing
d8fa1bbc8af1: Preparing
512d83aca3f9: Preparing
d8a33133e477: Preparing
b7d1665b7c06: Layer already exists
d4b880eafd57: Layer already exists
dac968e5ba1b: Waiting
51bc9ad8f58a: Waiting
7c4ac87d038e: Waiting
0c347b36628d: Waiting
bb39723e34a7: Waiting
8a8734644be2: Waiting
847528dab20f: Waiting
d35c07543039: Waiting
48a37b97f339: Waiting
dcf2066a324d: Waiting
eb20543b3ab7: Waiting
d8fa1bbc8af1: Waiting
512d83aca3f9: Waiting
d8a33133e477: Waiting
dac968e5ba1b: Layer already exists
51bc9ad8f58a: Layer already exists
7c4ac87d038e: Layer already exists
0c347b36628d: Layer already exists
bb39723e34a7: Layer already exists
8a8734644be2: Layer already exists
847528dab20f: Layer already exists
d35c07543039: Layer already exists
48a37b97f339: Layer already exists
dcf2066a324d: Layer already exists
eb20543b3ab7: Layer already exists
d8fa1bbc8af1: Layer already exists
512d83aca3f9: Layer already exists
d8a33133e477: Layer already exists
40da4e860053: Layer already exists
9003404e64e0: Layer already exists
b60cf95f6699: Layer already exists
version1: digest:
sha256:db1983ef84a69fe205609cb6a766df4c1441041514cf64c93c7083c1ea
b9f65f size: 4289
40da4e860053: Preparing
b60cf95f6699: Preparing
9003404e64e0: Preparing
b7d1665b7c06: Preparing
d4b880eafd57: Preparing
dac968e5ba1b: Preparing
51bc9ad8f58a: Preparing
7c4ac87d038e: Preparing
0c347b36628d: Preparing
bb39723e34a7: Preparing
8a8734644be2: Preparing
847528dab20f: Preparing
d35c07543039: Preparing
48a37b97f339: Preparing
dcf2066a324d: Preparing
eb20543b3ab7: Preparing
d8fa1bbc8af1: Preparing
512d83aca3f9: Preparing
b60cf95f6699: Layer already exists
40da4e860053: Layer already exists
d8a33133e477: Preparing
9003404e64e0: Layer already exists
d4b880eafd57: Layer already exists

```

```

dac968e5ba1b: Layer already exists
b7d1665b7c06: Layer already exists
51bc9ad8f58a: Layer already exists
48a37b97f339: Waiting
7c4ac87d038e: Layer already exists
8a8734644be2: Waiting
dcf2066a324d: Waiting
847528dab20f: Waiting
eb20543b3ab7: Waiting
d35c07543039: Waiting
d8fa1bbc8af1: Waiting
0c347b36628d: Layer already exists
512d83aca3f9: Waiting
8a8734644be2: Layer already exists
d8a33133e477: Waiting
847528dab20f: Layer already exists
d35c07543039: Layer already exists
48a37b97f339: Layer already exists
d8a33133e477: Layer already exists
bb39723e34a7: Layer already exists
dcf2066a324d: Layer already exists
eb20543b3ab7: Layer already exists
d8fa1bbc8af1: Layer already exists
512d83aca3f9: Layer already exists
version2: digest:
sha256:db1983ef84a69fe205609cb6a766df4c1441041514cf64c93c7083c1ea
b9f65f size: 4289
db3eb6d56e83: Preparing
2b4a9d2298ac: Preparing
747e9c78a3f5: Preparing
b7d1665b7c06: Preparing
d4b880eafd57: Preparing
dac968e5ba1b: Preparing
51bc9ad8f58a: Preparing
7c4ac87d038e: Preparing
0c347b36628d: Preparing
bb39723e34a7: Preparing
8a8734644be2: Preparing
2b4a9d2298ac: Layer already exists
847528dab20f: Preparing
d35c07543039: Preparing
48a37b97f339: Preparing
dcf2066a324d: Preparing
b7d1665b7c06: Layer already exists
eb20543b3ab7: Preparing
d8fa1bbc8af1: Preparing
747e9c78a3f5: Layer already exists
512d83aca3f9: Preparing
d8a33133e477: Preparing
d4b880eafd57: Layer already exists
51bc9ad8f58a: Layer already exists
d35c07543039: Waiting
bb39723e34a7: Layer already exists
7c4ac87d038e: Layer already exists
48a37b97f339: Waiting
8a8734644be2: Waiting
0c347b36628d: Layer already exists
dcf2066a324d: Waiting
847528dab20f: Waiting
db3eb6d56e83: Layer already exists
eb20543b3ab7: Waiting
8a8734644be2: Layer already exists
d8fa1bbc8af1: Waiting

```

```

dac968e5ba1b: Layer already exists
847528dab20f: Layer already exists
512d83aca3f9: Waiting
d8a33133e477: Waiting
512d83aca3f9: Layer already exists
d35c07543039: Layer already exists
dcf2066a324d: Layer already exists
48a37b97f339: Layer already exists
d8a33133e477: Layer already exists
d8fa1bbc8af1: Layer already exists
eb20543b3ab7: Layer already exists
version3: digest:
sha256:ca8e6cdcd2573a8eb62910c7d34a50c603e1274885e717c56df0f5e60a
824082 size: 4289
Finished: SUCCESS

```

Lampiran 3. Console output proses build docker image dan container

```

$ sudo docker build .
Sending build context to Docker daemon   233kB
Step 1/17 : FROM php:7.1-apache
--> 6e2b5270ea5c
Step 2/17 : RUN set -ex; savedAptMark="$(apt-mark
showmanual)"; apt-get update; apt-get
install -y --no-install-recommends libjpeg-dev
libmagickwand-dev libpng-dev ;
docker-php-ext-configure gd --with-png-dir=/usr --with-jpeg-
dir=/usr; docker-php-ext-install -j "$(nproc)"
bcmath exif gd
mysqli opcache zip ;
pecl install imagick-3.4.4; docker-php-ext-enable
imagick; apt-mark auto '.*' > /dev/null;
apt-mark manual $savedAptMark; ldd "$(php -r 'echo
ini_get("extension_dir");')/*.*.so | awk '/=>/
{ print $3 }' | sort -u | xargs
-r dpkg-query -S | cut -d: -f1 |
sort -u | xargs -rt apt-mark manual;
apt-get purge -y --auto-remove -o
APT::AutoRemove::RecommendsImportant=false; rm -rf
/var/lib/apt/lists/*
--> Using cache
--> 31705c8672bc
Step 3/17 : RUN { echo
'opcache.memory_consumption=128'; echo
'opcache.interned_strings_buffer=8'; echo
'opcache.max_accelerated_files=4000'; echo
'opcache.revalidate_freq=2'; echo
'opcache.fast_shutdown=1'; } >
/usr/local/etc/php/conf.d/opcache-recommended.ini
--> Using cache
--> b13cfd6d81f9
Step 4/17 : RUN { echo 'error_reporting = 4339';
echo 'display_errors = Off'; echo
'display_startup_errors = Off'; echo 'log_errors
= On'; echo 'error_log = /dev/stderr';
echo 'log_errors_max_len = 1024'; echo
'ignore_repeated_errors = On'; echo
'ignore_repeated_source = Off'; echo 'html_errors
= Off'; } > /usr/local/etc/php/conf.d/error-logging.ini

```

```

---> Using cache
---> 21e847ea9fca
Step 5/17 : RUN a2enmod rewrite expires
---> Using cache
---> 2ee11d3a0d67
Step 6/17 : VOLUME /var/www/html/belajarcicd/wp-content
---> Using cache
---> 15fb03ce221f
Step 7/17 : ENV WordPress_DB_NAME: WordPress
---> Using cache
---> 0bba7b59d307
Step 8/17 : ENV WordPress_DB_USERNAME: root
---> Using cache
---> 689e49834064
Step 9/17 : ENV WordPress_DB_HOST: db:3306
---> Using cache
---> 806101104bb8
Step 10/17 : ENV WordPress_DB_PASSWORD: root
---> Using cache
---> 7d7866162856
Step 11/17 : ENV WordPress_VERSION 5.2.2
---> Using cache
---> c17a44a04e11
Step 12/17 : ENV WordPress_SHA1
3605bcbe9ea48d714efa59b0eb2d251657e7d5b0
---> Using cache
---> 205239aa3c2c
Step 13/17 : RUN set -ex;          curl -o WordPress.tar.gz -fSL
"https://WordPress.org/WordPress-${WordPress_VERSION}.tar.gz";
echo "$WordPress_SHA1 *WordPress.tar.gz" | sha1sum -c -;
tar -xzf WordPress.tar.gz -C /usr/src/;          rm
WordPress.tar.gz;          chown -R www-data:www-data
/usr/src/WordPress
---> Using cache
---> 818aad7d7da
Step 14/17 : COPY docker-entrypoint.sh /usr/local/bin/
---> 1eef2e48e551
Step 15/17 : RUN ["chmod", "+x", "/usr/local/bin/docker-
entrypoint.sh"]
---> Running in b4ae3feb4fa9
Removing intermediate container b4ae3feb4fa9
---> af31036b668c
Step 16/17 : ENTRYPOINT ["docker-entrypoint.sh"]
---> Running in a1a8812352d4
Removing intermediate container a1a8812352d4
---> 897dde40424a
Step 17/17 : CMD ["apache2-foreground"]
---> Running in 1e1f5059e951
Removing intermediate container 1e1f5059e951
---> fe70b64b9dde
Successfully built fe70b64b9dde
$ sudo docker-compose up -d
belajarcicd_db_1 is up-to-date
belajarcicd_phpmyadmin_1 is up-to-date
Creating belajarcicd_web_1

```

BIODATA PENULIS



A. Identitas

Nama : Ahmad Nadzir Romadhon
NIM : H1A015064
Tempat, tanggal lahir : Cirebon, 19 Januari 1997
Alamat : Jalan Saleh Gang Mulya 1 Kelurahan Kesenden, Cirebon
No. Telp. : 089633420591
Alamat e-mail : ahmadnadzirromadhon@gmail.com

B. Riwayat Pendidikan Akademik

Periode	Jenjang	Institusi
2015 – sekarang	S1	Teknik Elektro Universitas Jenderal Soedirman
2012 – 2015	SMA	SMAN 2 Cirebon
2009 – 2012	SMP	SMPN 2 Cirebon

C. Riwayat Pendidikan Non Formal

Tahun	Keahlian	Penyelenggara	Kota
2018	Sistemasi Bisnis	NPA	Purwokerto
2015	Open Source dan arduino	Open Source Community UNSOED	Purbalingga

D. Prestasi

Tahun	Tingkat	Prestasi
2015	Kota	Juara 1 lomba Fotografi
2015	Nasional	Finalis Electronic City Photography Competition

E. Keahlian

Memiliki ketertarikan pada topik DevOps dan bisnis startup. Mampu mendesign sebuah *wireframe* untuk pengembangan *software/website* menggunakan aplikasi Figma dan merancang pemodelan use case perangkat lunak/*software*

menggunakan *software* Visual Paradigm. Terlibat dalam *group* daring jejaring sosial Docker Indonesia, IDDevOps dan Startup.