



**ITS**  
Institut  
Teknologi  
Sepuluh Nopember

TUGAS AKHIR - KI141502

## **RANCANG BANGUN PERANGKAT LUNAK INTERNET ACCESS MANAGEMENT BERBASIS KONTAINER**

FOURIR AKBAR  
NRP 05111440000115

Dosen Pembimbing I  
Royyana Muslim Ijtihadie, S.Kom, M.Kom, PhD

Dosen Pembimbing II  
Bagus Jati Santoso, S.Kom., Ph.D

JURUSAN INFORMATIKA  
Fakultas Teknologi Informasi dan Komunikasi  
Institut Teknologi Sepuluh Nopember  
Surabaya, 2018

*(Halaman ini sengaja dikosongkan)*



TUGAS AKHIR - KI141502

**RANCANG BANGUN PERANGKAT LUNAK INTERNET  
ACCESS MANAGEMENT BERBASIS KONTAINER**

FOURIR AKBAR  
NRP 05111440000115

Dosen Pembimbing I  
Royyana Muslim Ijtihadie, S.Kom, M.Kom, PhD

Dosen Pembimbing II  
Bagus Jati Santoso, S.Kom., Ph.D

JURUSAN INFORMATIKA  
Fakultas Teknologi Informasi dan Komunikasi  
Institut Teknologi Sepuluh Nopember  
Surabaya, 2018

*(Halaman ini sengaja dikosongkan)*

UNDERGRADUATE THESIS - KI141502

**DESIGN AND IMPLEMENTATION OF INTERNET ACCESS  
MANAGEMENT SOFTWARE USING CONTAINER**

FOURIR AKBAR  
NRP 05111440000115

Supervisor I  
Royyana Muslim Ijtihadie, S.Kom, M.Kom, PhD

Supervisor II  
Bagus Jati Santoso, S.Kom., Ph.D

Department of INFORMATICS  
Faculty of Information Technology and Communication  
Institut Teknologi Sepuluh Nopember  
Surabaya, 2018

*(Halaman ini sengaja dikosongkan)*

**LEMBAR PENGESAHAN**  
**RANCANG BANGUN PERANGKAT LUNAK INTERNET**  
***ACCESS MANAGEMENT* BERBASIS KONTAINER**

**TUGAS AKHIR**

Diajukan Guna Memenuhi Salah Satu Syarat  
Memperoleh Gelar Sarjana Komputer  
pada  
Bidang Studi Arsitektur dan Jaringan Komputer  
Program Studi S1 Jurusan Informatika  
Fakultas Teknologi Informasi dan Komunikasi  
Institut Teknologi Sepuluh Nopember

Oleh :

**FOURIR AKBAR**  
**NRP: 05111440000115**

Disetujui oleh Dosen Pembimbing Tugas Akhir :

Royyana Muslim Ijtihadie, S.Kom, M.Kom, PhD .....  
NIP: 197708242006041001 (Pembimbing 1)

Bagus Jati Santoso, S.Kom., Ph.D .....  
NIP: 051100116 (Pembimbing 2)

**SURABAYA**  
**Juni 2018**

*(Halaman ini sengaja dikosongkan)*



# **RANCANG BANGUN PERANGKAT LUNAK INTERNET ACCESS MANAGEMENT BERBASIS KONTAINER**

**Nama : FOURIR AKBAR**  
**NRP : 05111440000115**  
**Jurusan : Informatika FTIK**  
**Pembimbing I : Royyana Muslim Ijtihadie, S.Kom,  
M.Kom, PhD**  
**Pembimbing II : Bagus Jati Santoso, S.Kom., Ph.D**

## **Abstrak**

*Seiring dengan perkembangan zaman yang sangat pesat, negara-negara sudah mempunyai teknologi yang sangat maju. Teknologi mempunyai peranan yang sangat penting dalam kehidupan manusia, karena dengan adanya teknologi, manusia bisa saling berhubungan dengan mudah. Sekarang teknologi sudah semakin canggih. Teknologi yang paling populer sekarang ini adalah internet karena dengan adanya internet, banyak informasi-informasi yang dapat kita ambil dengan mudah. Internet merupakan suatu perpustakaan besar yang di dalamnya terdapat sangat banyak informasi yang berupa teks dalam bentuk media elektronik. Selain itu internet dikenal sebagai dunia maya, karena hampir seluruh aspek kehidupan di dunia nyata ada di internet, seperti olah raga, politik, hiburan, akademik, dan lain sebagainya. Internet juga mempunyai peranan yang sangat penting dalam dunia pendidikan, karena dengan adanya internet bisa menambah ilmu pengetahuan kita dan dapat menambah motivasi belajar siswa ataupun mahasiswa. Dengan dimanfaatkan internet dalam dunia pendidikan agar siswa atau mahasiswa dapat memiliki komitmen untuk belajar secara aktif dan memiliki teknis kemampuan khususnya di bidang pendidikan. Oleh karena itu, internet dapat*

*mempermudah proses belajar mengajar dengan baik.*

*Sudah hampir seperempat abad masyarakat dunia menggunakan internet. Hingga saat ini, internet sudah digunakan oleh masyarakat dari berbagai usia dan generasi. Hasil penelitian Yahoo dan Taylor Nelson Sofres (TNS) Indonesia pada tahun 2009 menunjukkan pengakses terbesar di Indonesia*

*Saat ini, dengan didukung oleh konsep SaaS (Software as a Service), aplikasi web berkembang dengan pesat. Para penyedia layanan aplikasi web berlomba-lomba memberikan pelayanan yang terbaik, seperti menjaga QoS (Quality of Service) sesuai dengan perjanjian yang tertuang dalam SLA (Service Level Agreement). Hal tersebut dikarenakan permintaan akses ke suatu aplikasi web biasanya meningkat dengan seiring berjalannya waktu. Keramaian akses sesaat menjadi hal yang umum dalam aplikasi web saat ini. Saat hal tersebut terjadi, aplikasi web akan di akses lebih banyak dari kebiasaan. Jika aplikasi web tersebut tidak menyediakan kemampuan untuk menangani hal tersebut, bisa menyebabkan aplikasi web tidak dapat berjalan dengan semestinya yang sangat merugikan pengguna.*

*Elastic cloud merupakan salah satu bagian dari komputasi awan yang sedang populer, dimana banyak riset dan penelitian yang berfokus di bidang ini. Elastic cloud bisa digunakan untuk menyelesaikan permasalahan di atas. Lalu sebuah perangkat lunak bernama Docker dapat diterapkan untuk mendukung elastic cloud.*

*Dalam tugas akhir ini akan dibuat sebuah rancangan sistem yang memungkinkan aplikasi web berjalan di atas Docker. Sistem ini bisa beradaptasi sesuai dengan kebutuhan dari aplikasi yang sedang berjalan. Jika aplikasi membutuhkan sumber daya tambahan, sistem akan menyediakan sumber daya berupa suatu container baru secara otomatis dan juga akan*

*mengurangi penggunaan sumber daya jika aplikasi sedang tidak membutuhkannya. Dari hasil uji coba, sistem dapat menangani sampai dengan 57.750 request dengan error request yang terjadi sebesar 7.83%.*

***Kata-Kunci:*** *aplikasi web, autoscale, docker, elastic cloud*

# DESIGN AND IMPLEMENTATION OF INTERNET ACCESS MANAGEMENT SOFTWARE USING CONTAINER

**Name** : FOURIR AKBAR  
**NRP** : 05111440000115  
**Major** : Informatics FTIK  
**Supervisor I** : Royyana Muslim Ijtihadie, S.Kom,  
M.Kom, PhD  
**Supervisor II** : Bagus Jati Santoso, S.Kom., Ph.D

## Abstract

*Nowdays, with the concept of SaaS (Software as a Service), web applications have developed a lot. Web service providers are competing to provide the best service, such as QoS (Quality of Service) requirements specified in the SLA (Service Level Agreement). The load of web applications usually very drastically along with time. Flash crowds are also very common in today's web applications world. When flash crowds happens, the web application will be accessed more than usual. If the web applications does not provide the ability to do so, it can make the web application not work properly which is very disadvantageous to the users.*

*Elastic cloud is one of the most popular part of cloud computing, with much researchs in this subject. Elastic clouds can be used to solve the above problems. Then a Docker can be applied to support the elastic cloud.*

*In this final task will be made an application system that allows web applications running on top of Docker. This system can adjust according to the needs of the running applications. If the application requires additional resources, the system will automatically supply the resources of a new container and will*

*also reduce resource usage if the application is not needing it. From the test results, the system can handle up to 57,750 requests and error ratio of 7.83%.*

***Keywords:*** *autoscale, docker, elastic cloud, web application*

*(Halaman ini sengaja dikosongkan)*

## KATA PENGANTAR

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ

Alhamdulillahirabbil'alamin, segala puji bagi Allah SWT, yang telah melimpahkan rahmat dan hidayah-Nya sehingga penulis dapat menyelesaikan Tugas Akhir yang berjudul **Rancang Bangun Perangkat Lunak Internet Access Management Berbasis Kontainer**. Pengerjaan Tugas Akhir ini merupakan suatu kesempatan yang sangat baik bagi penulis. Dengan pengerjaan Tugas Akhir ini, penulis bisa belajar lebih banyak untuk memperdalam dan meningkatkan apa yang telah didapatkan penulis selama menempuh perkuliahan di Teknik Informatika ITS. Dengan Tugas Akhir ini penulis juga dapat menghasilkan suatu implementasi dari apa yang telah penulis pelajari. Selesaiannya Tugas Akhir ini tidak lepas dari bantuan dan dukungan beberapa pihak. Sehingga pada kesempatan ini penulis mengucapkan syukur dan terima kasih kepada:

1. Allah SWT atas anugerahnya yang tidak terkira kepada penulis dan Nabi Muhammad SAW.
2. Bapak, Mama, dan keluarga Penulis yang selalu memberikan perhatian, dorongan dan kasih sayang yang menjadi semangat utama bagi diri Penulis sendiri baik selama penulis menempuh masa perkuliahan maupun pengerjaan Tugas Akhir ini.
3. Bapak Royyana Muslim Ijtihadie, S.Kom., M.Kom., PhD. selaku Dosen Pembimbing yang telah banyak meluangkan waktu untuk memberikan ilmu, nasihat, motivasi, pandangan dan bimbingan kepada Penulis baik selama Penulis menempuh masa kuliah maupun selama pengerjaan Tugas Akhir ini.
4. Bagus Jati Santoso, S.Kom., PhD. selaku dosen pembimbing yang telah memberikan ilmu, dan masukan kepada Penulis.

5. Seluruh tenaga pengajar dan karyawan Jurusan Teknik Informatika ITS yang telah memberikan ilmu dan waktunya demi berlangsungnya kegiatan belajar mengajar di Jurusan Teknik Informatika ITS.
6. Seluruh teman Penulis di Jurusan Teknik Informatika ITS yang telah memberikan dukungan dan semangat kepada Penulis selama Penulis menyelesaikan Tugas Akhir ini.
7. Teman-teman, Kakak-kakak dan Adik-adik *administrator* Laboratorium Arsitektur dan Jaringan Komputer yang selalu menjadi teman untuk berbagi ilmu.

Penulis menyadari bahwa Tugas Akhir ini masih memiliki banyak kekurangan. Sehingga dengan kerendahan hati, penulis mengharapkan kritik dan saran dari pembaca untuk perbaikan ke depannya.

Surabaya, Juni 2018

Fourir Akbar



# DAFTAR ISI

<b>ABSTRAK</b>	<b>vii</b>
<b>ABSTRACT</b>	<b>x</b>
<b>Kata Pengantar</b>	<b>xiii</b>
<b>DAFTAR ISI</b>	<b>xv</b>
<b>DAFTAR TABEL</b>	<b>xix</b>
<b>DAFTAR GAMBAR</b>	<b>xxi</b>
<b>DAFTAR KODE SUMBER</b>	<b>xxiii</b>
<b>BAB I PENDAHULUAN</b>	<b>1</b>
1.1 Latar Belakang . . . . .	1
1.2 Rumusan Masalah . . . . .	1
1.3 Batasan Masalah . . . . .	2
1.4 Tujuan . . . . .	2
1.5 Manfaat . . . . .	3
1.6 Metodologi . . . . .	3
1.6.1 Studi literatur . . . . .	3
1.6.2 Desain dan Perancangan Sistem . . . . .	4
1.6.3 Implementasi Sistem . . . . .	4
1.6.4 Uji Coba dan Evaluasi . . . . .	4
1.7 Sistematika Laporan . . . . .	4
<b>BAB II TINJAUAN PUSTAKA</b>	<b>7</b>
2.1 Python . . . . .	7
2.2 Flask . . . . .	7
2.3 Unicorn . . . . .	8
2.4 <i>Supervisor</i> . . . . .	8
2.4.1 <i>Supervisord</i> . . . . .	9
2.4.2 <i>Supervisorctl</i> . . . . .	9

2.5	<i>Nginx</i>	10
2.6	<i>Iptables</i>	10
2.7	MySQL	11
2.8	<i>Mitmproxy</i>	12
2.9	<i>VirtualBox</i>	13
2.10	<i>Crontab</i>	14
2.11	<i>Docker</i>	14
2.11.1	<i>Docker Container</i>	15
2.11.2	<i>Docker Images</i>	16
2.11.3	<i>Docker Registry</i>	16
<b>BAB III</b>	<b>DESAIN DAN PERANCANGAN</b>	<b>17</b>
3.1	Deskripsi Umum Sistem	17
3.2	Kasus Penggunaan	17
3.3	Arsitektur Sistem	20
3.3.1	Desain Umum Sistem	20
3.3.2	Pembuatan Halaman <i>Login</i> dari Sebuah Sistem.	23
3.3.3	Perancangan Pembuatan Aturan untuk Mengarahkan <i>Traffic Client</i> ke Halaman <i>Login</i> dari Sistem	26
3.3.4	Pembuatan <i>Middleware</i> untuk Menerima Permintaan dari <i>Client</i>	27
3.3.5	Perancangan Pemasangan Kontainer pada <i>Docker Host</i>	29
3.3.6	Pembuatan Aturan untuk Mengarahkan <i>Traffic Client</i> ke Kontainer <i>Docker</i> dari Tiap-Tiap <i>Client</i>	31
3.3.7	Pembuatan Halaman <i>Administrator</i> untuk Membaca <i>Log File</i> dari <i>Client</i>	32
<b>BAB IV</b>	<b>IMPLEMENTASI</b>	<b>35</b>
4.1	Lingkungan Implementasi	35
4.1.1	Perangkat Keras	35

4.1.2	Perangkat Lunak . . . . .	35
4.2	Implementasi Pembuatan Halaman <i>Login</i> dari Sebuah Sistem . . . . .	36
4.2.1	Implementasi <i>Web Service</i> pada Halaman <i>Login</i> . . . . .	37
4.2.2	Implementasi Basis Data pada Halaman <i>Login</i> . . . . .	40
4.3	Implementasi Pembuatan Aturan untuk Mengarahkan <i>Traffic Client</i> ke Halaman <i>Login</i> dari Sistem . . . . .	41
4.4	Implementasi Pembuatan <i>Middleware</i> . . . . .	42
4.4.1	Implementasi <i>Web Service</i> pada <i>Middleware</i> . . . . .	42
4.4.2	Implementasi Basis Data pada <i>Middleware</i> . . . . .	44
4.5	Implementasi Pemasangan Kontainer <i>Docker</i> pada <i>Docker Host</i> . . . . .	45
4.5.1	Menambahkan dan Memperbarui Kontainer <i>Docker</i> yang Berisikan Mitmproxy . . . . .	45
4.5.2	Menggunakan <i>Image</i> Kontainer <i>Docker</i> yang Sudah Dibuat . . . . .	48
4.6	Implementasi Pembuatan Aturan untuk Mengarahkan <i>Traffic Client</i> ke Kontainer <i>Docker</i> dari Tiap-Tiap <i>Client</i> . . . . .	49
4.7	Implementasi Pembuatan Halaman <i>Administrator</i> . . . . .	50
4.7.1	Rute <i>Web Service</i> pada Halaman <i>Administrator</i> . . . . .	50
4.7.2	Implementasi Pembacaan <i>Log File</i> dari <i>Client</i> . . . . .	52
4.7.3	Implementasi Antarmuka Halaman <i>Administrator</i> . . . . .	53
<b>BAB V</b>	<b>PENGUJIAN DAN EVALUASI</b>	<b>55</b>
5.1	Lingkungan Uji Coba . . . . .	55

5.2	Skenario Uji Coba . . . . .	56
5.2.1	Skenario Uji Coba Fungsionalitas . . . . .	57
5.2.2	Skenario Uji Coba Performa . . . . .	61
5.3	Hasil Uji Coba dan Evaluasi . . . . .	63
5.3.1	Uji Fungsionalitas . . . . .	63
5.3.2	Hasil Uji Performa . . . . .	67
<b>BAB VI PENUTUP</b>		<b>75</b>
6.1	Kesimpulan . . . . .	75
6.2	Saran . . . . .	75
<b>DAFTAR PUSTAKA</b>		<b>77</b>
<b>BAB A INSTALASI PERANGKAT LUNAK</b>		<b>79</b>
<b>BAB B KODE SUMBER</b>		<b>91</b>
<b>BIODATA PENULIS</b>		<b>93</b>

## DAFTAR TABEL

3.1	Daftar Kode Kasus Penggunaan . . . . .	19
3.1	Daftar Kode Kasus Penggunaan . . . . .	20
3.2	Atribut basis data nrp-mahasiswa . . . . .	24
3.3	Atribut basis data kontainer . . . . .	28
3.3	Atribut basis data kontainer . . . . .	29
4.1	Daftar Rute <i>Web Service</i> . . . . .	39
4.2	Daftar Rute <i>Web Service</i> . . . . .	43
4.3	Daftar Rute <i>Web Service</i> pada Halaman <i>Administrator</i> . . . . .	51
5.1	Spesifikasi Komponen . . . . .	55
5.2	IP dan Domain Server . . . . .	56
5.3	Skenario Uji Mengelola Aplikasi Berbasis Docker	58
5.4	Skenario Uji Fungsionalitas Aplikasi Dasbor . . .	60
5.5	Hasil Uji Coba Mengelola Aplikasi Berbasis Docker	64
5.6	Hasil Uji Fungsionalitas Aplikasi Dasbor . . . . .	65
5.7	Jumlah <i>Request</i> ke Aplikasi . . . . .	67
5.8	Jumlah <i>Container</i> . . . . .	68
5.9	Kecepatan Menangani <i>Request</i> . . . . .	69
5.10	Penggunaan CPU . . . . .	70
5.11	Penggunaan <i>Memory</i> . . . . .	71
5.12	<i>Error Ratio Request</i> . . . . .	72

*(Halaman ini sengaja dikosongkan)*

## DAFTAR GAMBAR

2.1	Perbandingan <i>docker</i> dan virtual machine . . . . .	15
3.1	Digram Kasus Penggunaan . . . . .	18
3.2	Arsitektur Komponen Sistem . . . . .	22
3.3	Desain Halaman <i>Login</i> . . . . .	25
3.4	Desain <i>Backend</i> dari Halaman <i>Login</i> . . . . .	26
3.5	Desain Mengarahkan <i>Traffic Client</i> ke Halaman <i>Login</i> . . . . .	27
3.6	Alur kerja dari <i>mitmproxy transparent</i> HTTP . . .	30
3.7	Alur kerja dari <i>mitmproxy transparent</i> HTTPS . .	31
3.8	Desain pembuatan aturan untuk mengarahkan <i>traffic client</i> ke kontainer <i>docker</i> . . . . .	32
3.9	Desain halaman dashboard <i>administrator traffic</i> <i>client</i> ke kontainer <i>docker</i> . . . . .	33
3.10	Desain pembuatan aturan untuk mengarahkan <i>traffic client</i> ke kontainer <i>docker</i> . . . . .	33
4.1	Halaman <i>Login</i> . . . . .	38
4.2	Halaman <i>Administrator Menu User List</i> . . . . .	53
4.3	Halaman <i>Administrator Menu History</i> . . . . .	54
5.1	Grafik Jumlah <i>Container</i> . . . . .	68
5.2	Grafik Kecepatan Menangani <i>Request</i> . . . . .	69
5.3	Grafik Penggunaan CPU . . . . .	70
5.4	Grafik Penggunaan Memory . . . . .	71
5.5	Grafik Error Ratio . . . . .	72

*(Halaman ini sengaja dikosongkan)*



## DAFTAR KODE SUMBER

4.1	Command untuk Reload Supervisor . . . . .	37
4.2	Command untuk mengaktifkan konfigurasi Nginx . . . . .	37
4.3	Command untuk merestart Nginx . . . . .	38
4.4	Pseudocode Web Service . . . . .	40
4.5	<i>Query</i> untuk membuat tabel testing . . . . .	40
4.6	Command untuk mengarahkan <i>client</i> ke halaman <i>login</i> . . . . .	41
4.7	Command untuk instalasi Flask . . . . .	42
4.8	Pseudocode Web Service . . . . .	44
4.9	<i>Query</i> untuk membuat tabel testing . . . . .	45
4.10	Perintah untuk instalasi Ansible . . . . .	45
4.11	Perintah untuk <i>Pull</i> Ubuntu . . . . .	46
4.12	Perintah untuk Menjalankan <i>Image</i> Ubuntu . . . . .	46
4.13	Perintah untuk Pemasangan <i>Mitmproxy</i> . . . . .	47
4.14	Perintah untuk Mengaktifkan <i>ipv4.forwarding</i> . . . . .	47
4.15	Perintah untuk Menghentikan Kontainer <i>Docker</i> . . . . .	47
4.16	Perintah untuk <i>Commit</i> Kontainer <i>Docker</i> . . . . .	48
4.17	Perintah untuk <i>Push Image</i> ke Docker Hub . . . . .	48
4.18	Perintah untuk <i>Pull Image mitmproxy</i> . . . . .	49
4.19	Perintah untuk <i>Pull Image mitmproxy</i> . . . . .	49
4.20	Command untuk mengarahkan <i>client</i> ke halaman <i>login</i> . . . . .	49
4.21	Perintah untuk Membaca <i>File Log</i> dari <i>Mitmproxy</i> . . . . .	52
4.22	Perintah untuk Membaca <i>File Log</i> dari <i>Client</i> . . . . .	52
1.1	Isi Berkas <i>docker-compose.yml</i> . . . . .	81
1.2	Isi Berkas <i>registry.conf</i> . . . . .	81
1.3	Isi Berkas <i>confd.toml</i> . . . . .	85
1.4	Isi Berkas <i>haproxy.cfg.tmpl</i> . . . . .	85
1.5	Isi Berkas <i>haproxy.toml</i> . . . . .	87
2.1	Let's Encrypt X3 Cross Signed.pem . . . . .	91

*(Halaman ini sengaja dikosongkan)*

# BAB I

## PENDAHULUAN

Pada bab ini akan dipaparkan mengenai garis besar Tugas Akhir yang meliputi latar belakang, tujuan, rumusan dan batasan permasalahan, metodologi pembuatan Tugas Akhir, dan sistematika penulisan.

### 1.1 Latar Belakang

Saat ini penggunaan kontainer *docker* dalam dunia teknologi sangat banyak dilakukan. Kontainer *docker* merupakan *operating-system-level virtualization* untuk menjalankan beberapa sistem linux yang terisolasi (kontainer) pada sebuah *host* atau *server*. Kontainer berfungsi untuk mengisolasi aplikasi atau servis dan dependensinya.

Untuk setiap servis atau aplikasi yang terisolasi, dibutuhkan satu kontainer pada *server* host yang ada. Dalam kasus ini, setiap satu *client* yang mengakses atau menggunakan jaringan ITS merupakan satu servis yang nantinya akan dibuatkan satu kontainer pada *server* host. Hal ini dapat mempermudah manajemen dari masing-masing *client*, contohnya manajemen hak akses, waktu, maupun melihat *access log* dan lain sebagainya.

Setiap *client* yang akan menggunakan jaringan ITS, akan diarahkan ke sebuah halaman *login*. Setelah *client* tersebut berhasil *login*, barulah *client* tersebut dapat mengakses internet.

### 1.2 Rumusan Masalah

Berikut beberapa hal yang menjadi rumusan masalah dalam tugas akhir ini:

1. Bagaimana cara *client* untuk melakukan *autentifikasi*?
2. Bagaimana cara membuat sebuah kontainer *docker* secara otomatis ketika terdapat *client* yang akan mengakses

internet?

3. Bagaimana cara mengarahkan *traffic* dari *client* ke kontainer *docker* yang sesuai?
4. Bagaimana cara mencatat aktivitas dari *client*?
5. Bagaimana perbandingan performa antara IAM konvensional dengan IAM berbasis kontainer?
6. Bagaimana mengevaluasi penggunaan sumber daya dan skalabilitas pada *docker host*?

### 1.3 Batasan Masalah

Dari permasalahan yang telah diuraikan di atas, terdapat beberapa batasan masalah pada tugas akhir ini, yaitu:

1. Satu *client* yang berhasil *login* akan disediakan satu kontainer *docker*.
2. Kontainer yang digunakan adalah *docker*.
3. Parameter untuk mengetahui apa saja yang diakses oleh *client* adalah *access log* dari *client* tersebut.
4. Setiap *client* mendapatkan IP *private*.
5. Performa yang diukur adalah *response time*.
6. Bahasa pemrograman yang digunakan adalah *Python*.

### 1.4 Tujuan

Tugas akhir dibuat dengan beberapa tujuan. Berikut beberapa tujuan dari pembuatan tugas akhir:

1. Mengetahui cara bagaimana *client* dapat melakukan *autentifikasi*.
2. Mengimplementasikan metode untuk membuat sebuah kontainer terhadap *client* yang telah berhasil *login* ke jaringan ITS.
3. Mengetahui cara untuk mengarahkan *traffic* dari *client* ke kontainer *docker* yang sesuai.

4. Mengetahui bagaimana cara mencatat aktivitas *client*.
5. Mengetahui penggunaan sumber daya dan skalabilitas pada *docker host*.

## 1.5 Manfaat

Tugas akhir dibuat dengan beberapa manfaat. Berikut beberapa manfaat dari pembuatan tugas akhir:

1. Mengetahui cara bagaimana *client* dapat melakukan *autentifikasi*.
2. Mengetahui cara untuk mengarahkan *traffic* dari *client* ke kontainer *docker* yang sesuai.
3. Mempermudah pencatatan aktivitas dari masing-masing *client* yang mengakses internet.
4. Meringankan beban dari penggunaan *server* di ITS karena penggunaan kontainer *docker* lebih ringan.

## 1.6 Metodologi

Metodologi yang digunakan pada pengerjaan Tugas Akhir ini adalah sebagai berikut:

### 1.6.1 Studi literatur

Studi literatur merupakan langkah yang dilakukan untuk mendukung dan memastikan setiap tahap pengerjaan tugas akhir sesuai dengan standar dan konsep yang berlaku. Pada tahap studi literatur ini, akan dilakukan studi mendalam mengenai kontainer *docker*, *flask*, *mitmproxy*, dan pembuatan aturan dengan menggunakan *iptables*. Adapun literatur yang dijadikan sumber berasal dari paper, buku, materi perkuliahan, forum serta artikel dari internet.

### **1.6.2 Desain dan Perancangan Sistem**

Tahap ini meliputi perancangan sistem berdasarkan studi literatur dan pembelajaran konsep. Tahap ini merupakan tahap yang paling penting dimana bentuk awal aplikasi yang akan diimplementasikan didefinisikan. Pada tahapan ini dibuat kasus penggunaan yang ada pada sistem, arsitektur sistem, serta perencanaan implementasi pada sistem.

### **1.6.3 Implementasi Sistem**

Implementasi merupakan tahap membangun implementasi rancangan sistem yang telah dibuat. Pada tahapan ini merealisasikan apa yang telah didesain dan dirancang pada tahapan sebelumnya, sehingga menjadi sebuah sistem yang sesuai dengan apa yang telah direncanakan.

### **1.6.4 Uji Coba dan Evaluasi**

Pada tahapan ini dilakukan uji coba terhadap sistem yang telah dibuat. Pengujian dan evaluasi akan dilakukan dengan melihat kesesuaian dengan perencanaan. Selain itu, tahap ini juga akan melakukan uji performa sistem dan melakukan perbandingan dengan metode lain untuk mengetahui efisiensi penggunaan sumber daya serta evaluasi berdasarkan hasil uji performa tersebut.

## **1.7 Sistematika Laporan**

Buku tugas akhir ini bertujuan untuk mendapatkan gambaran dari pengerjaan tugas akhir ini. Selain itu, diharapkan dapat berguna bagi pembaca yang berminat melakukan pengembangan lebih lanjut. Secara garis besar, buku tugas akhir ini terdiri atas beberapa bagian seperti berikut:

1. **Bab I Pendahuluan**

Bab yang berisi latar belakang, tujuan, manfaat, permasalahan, batasan masalah, metodologi yang digunakan dan sistematika laporan.

2. **Bab II Dasar Teori**

Bab ini berisi penjelasan secara detail mengenai dasar-dasar penunjang dan teori-teori yang yang digunakan dalam pembuatan tugas akhir ini.

3. **Bab III Desain dan Perancangan**

Bab ini berisi tentang analisis dan perancangan sistem yang dibuat, termasuk di dalamnya mengenai analisis kasus penggunaan, desain arsitektur sistem, dan perancangan implementasi sistem.

4. **Bab IV Implementasi**

Bab ini membahas implementasi dari desain yang telah dibuat pada bab sebelumnya. Penjelasan berupa pemasangan alat dan kode program yang digunakan untuk mengimplementasikan sistem.

5. **Bab V Uji Coba dan Evaluasi**

Bab ini membahas tahap-tahap uji coba serta melakukan evaluasi terhadap sistem yang dibuat.

6. **Bab VI Kesimpulan dan Saran**

Bab ini merupakan bab terakhir yang memberikan kesimpulan dari hasil percobaan dan evaluasi yang telah dilakukan. Pada bab ini juga terdapat saran bagi pembaca yang berminat untuk melakukan pengembangan lebih lanjut.

*(Halaman ini sengaja dikosongkan)*



## **BAB II**

### **TINJAUAN PUSTAKA**

#### **2.1 Python**

Python adalah bahasa pemrograman interpretatif multiguna dengan prinsip agar sumber kode yang dihasilkan memiliki tingkat keterbacaan yang baik. Python diklaim sebagai bahasa yang menggabungkan kapabilitas, kemampuan, dengan sintaksis kode yang sangat jelas, dan dilengkapi dengan fungsionalitas pustaka standar yang besar serta komprehensif. Python mendukung beragam paradigma pemrograman, seperti pemrograman berorientasi objek, pemrograman imperatif, dan pemrograman fungsional. Python dapat digunakan untuk berbagai keperluan pengembangan perangkat lunak dan dapat berjalan di berbagai platform sistem operasi. [1]

#### **2.2 Flask**

Flask adalah sebuah kerangka kerja web. Artinya, Flask menyediakan perangkat, pustaka, dan teknologi yang memungkinkan seorang pengembang untuk membangun aplikasi berbasis web. Aplikasi web yang bisa dibangun bisa berupa sebuah halaman web, blog, wiki, bahkan untuk web komersial. Flask dibangun berbasiskan pada Werkzeug, Jinja 2, dan MarkupSafe yang mana menggunakan bahasa pemrograman Python sebagai basisnya. Flask sendiri pertama kali dikembangkan pada tahun 2010 dan didistribusikan dengan lisensi BSD. [2]

Flask termasuk sebagai perangkat kerja mikro karena tidak membutuhkan banyak perangkat atau pustaka tertentu agar bisa bekerja. Flask tidak menyediakan fungsi untuk melakukan interaksi dengan basis data, tidak mempunyai validasi *form* atau fungsi lain yang umumnya bisa digunakan dan disediakan pada sebuah kerangka kerja web. Meskipun memiliki kemampuan

yang minim, tapi Flask mendukung dan memberikan kemudahan bagi pengembang untuk menambahkan pustaka sendiri untuk mendukung aplikasinya. Berbagai pustaka seperti validasi *form*, mengunggah file, berbagai macam teknologi autentifikasi bisa digunakan dan tersedia untuk Flask. Bahkan pustaka-pustaka pendukung tersebut lebih sering diperbarui dibandingkan dengan Flasknya sendiri.

### 2.3 Unicorn

Unicorn atau '*Green Unicorn*' adalah Python WSGI HTTP *Server* untuk UNIX. Fungsi dari *Unicorn* ini adalah sebagai pelayan sebuah aplikasi atau sebagai server dari sebuah perangkat lunak yang dikembangkan oleh pengembang.

*Unicorn* sendiri merupakan salah satu dari sekian banyak *WSGI Server*. Keunggulan dari *Unicorn* sendiri adalah, *Unicorn* mampu menangani atau kompatibel dengan berbagai macam kerangka kerja web, sangat mudah untuk diimplementasikan, hanya membutuhkan sedikit sumber daya dari *server* yang terpasang *Unicorn*, dan juga kerja dari *Unicorn* yang sangat cepat.

*Unicorn* mengimplementasikan spesifikasi standar *server WSGI PEP3333* sehingga dapat menjalankan perangkat lunak berbasis *web* yang dikembangkan dengan bahasa pemrograman *python*. Sebagai contoh, perangkat lunak berbasis web yang digunakan oleh penulis menggunakan kerangka kerja *flask*, maka *Unicorn* dapat menanganinya.

### 2.4 Supervisor

*Supervisor* adalah sistem yang berbasis *client* atau server, yang memungkinkan pengguna untuk memantau dan juga mengontrol sejumlah proses pada sistem operasi untuk UNIX.

Beberapa faktor terbentuknya *supervisor* antara lain adalah, kenyamanan, ketepatan, delegasi, dan proses grup dalam menggunakan perangkat lunak *supervisor*. Beberapa keunggulan dari perangkat lunak *supervisor* antara lain, konfigurasi yang sederhana, proses yang terpusat, efisien, dapat diperluas penggunaannya, dan juga kompatibel dengan berbagai macam sistem operasi. Komponen dari *supervisor* terbagi menjadi dua, antara lain sebagai berikut.

#### **2.4.1 *Supervisord***

*Supervisord* merupakan bagian dari *supervisor* yang bertanggung jawab untuk memulai *child programs* atas permintaannya sendiri, menanggapi perintah dari *client*, melakukan *restart* secara otomatis ketika terjadi kerusakan pada proses, mencatat bagian dari proses *stdout* dan *stderr output*, juga menghasilkan dan menangani *events* yang berhubungan dengan bagian-bagian yang digunakan selama *subprocess* tersebut berjalan.

#### **2.4.2 *Supervisorctl***

*Supervisorctl* merupakan bagian dari *command-line* yang digunakan oleh *client*. *Supervisorctl* menyediakan antarmuka yang mirip dengan fitur *shell* yang disediakan oleh *supervisord*. Dari *supervisorctl*, pengguna dapat terhubung dengan proses *supervisord* yang berbeda satu per satu, mendapatkan status dari *subprocess* yang telah dikontrol, menghentikan atau memulai *subprocess* yang telah dikontrol, dan juga mendapatkan semua daftar proses yang berjalan pada *supervisord*.

*Command-line* dari *client* berhubungan ke *server* melalui *socket domain* UNIX atau melalui *socket internet* (TCP). *Server* dapat menyatakan bahwa *client* harus memberikan *autentifikasi* sebelum mengizinkannya untuk melakukan sebuah perintah.

Proses *client* biasanya menggunakan *file* konfigurasi yang sama dengan *server*.

## 2.5 *Nginx*

Nginx adalah sebuah perangkat lunak yang bisa digunakan untuk *web server*, *load balancer*, dan *reverse proxy*. Nginx terkenal karena stabil, memiliki tingkat performa tinggi dan konsumsi sumber daya yang minim. Pada kasus saat terjadi koneksi dalam jumlah yang banyak secara bersamaan, penggunaan *memory*, CPU, dan sumber daya sistem yang lain sangat kecil dan stabil. [xx]

Nginx bisa digunakan untuk menyajikan konten HTTP yang dinamis menggunakan FastCGI, SCGI untuk menangani scripts, aplikasi WSGI , dan bisa juga digunakan sebagai sebuah *load balancer*. Nginx menggunakan *asynchronous event-driven* untuk menangani permintaan. Dengan menggunakan model ini bisa, pengembang bisa melakukan prediksi kinerja Nginx saat terjadi jumlah permintaan yang banyak.

## 2.6 *Iptables*

*Firewall* merupakan sebuah mekanisme wajib *access* kontrol antar jaringan ataupun antar sistem. *Firewall* ini sangat penting karena bertujuan untuk memastikan keamanan dari sebuah jaringan. *Firewall* dapat menjadi *filter* yang sangat sederhana dan mudah digunakan, tetapi *firewall* juga dapat menjadi *filter* yang sangat penting bagi sebuah jalan keluar suatu jaringan. Prinsip dari penggunaan *firewall* tetaplah sama, dimana penggunaannya untuk *monitoring* dan *filtering* semua pertukaran informasi di jaringan *internal* dan juga di jaringan *external*.

*Netfilter* / *iptables* merupakan sebuah sistem *firewall* berbasis linux yang mempunyai fungsi yang sangat berguna.

*Netfilter / iptables kernel* menggunakan sebuah mekanisme baru, bernama *iptables*. *Iptables* sendiri merupakan sebuah perangkat lunak atau alat yang dapat melakukan manajemen *filter* dari sebuah paket yang ada pada suatu *kernel*. *Iptables* mempunyai *table* dan juga *chain* dari masing-masing *table*. *Table* pada *iptables* terdiri dari tiga, atau juga bisa disebut *iptables* memiliki tiga fungsi utama, antara lain menjadi penyaring paket, mentranslasikan suatu alamat, dan melakukan penghalusan paket seperti TTL, TOS, dan MARK.

*Filter table* merupakan sebuah konfigurasi *default* dari *iptables*, dimana pada *filter table* terdapat tiga *chain*, antara lain *chain* INPUT, FORWARD, dan OUTPUT. *NAT table* berfungsi untuk merubah tujuan dari sumber dari sebuah paket. Pada *NAT table* terdapat dua *chain*, antara lain *chain* PREROUTING dan POSTROUTING. *Mangle table* berfungsi untuk menghaluskan paket atau juga dapat mengubah isi dari sebuah data kecuali IP *address* dan *port address*. Pada *mangle table* terdapat dua *chain*, antara lain POSTROUTING dan OUTPUT.

## 2.7 MySQL

MySQL adalah sebuah perangkat lunak terbuka untuk melakukan manajemen basis data SQL atau DBMS. MySQL ditulis dalam bahasa pemrograman C dan C++. MySQL merupakan salah satu perangkat lunak terbuka yang banyak disukai oleh pengembang dan digunakan dalam banyak aplikasi web. Parser SQL yang digunakan ditulis dalam bahasa pemrograman yacc. MySQL bekerja pada banyak *platform*, seperti FreeBSD, HP-UX, Linux, macOS, Microsoft Windows, NetBSD, OpenBSD, OpenSolaris, Oracle Solaris, dan SunOS. MySQL tersedia sebagai perangkat lunak gratis di bawah lesensi *GNU General Public License* (GPL), tetapi juga tersedia lisensi komersial untuk kasus-kasus dimana penggunaanya tidak cocok

dengan penggunaan GPL.

Setiap pengguna dapat secara bebas menggunakan MySQL, namun dengan batasan perangkat lunak tersebut tidak boleh dijadikan produk turunan yang bersifat komersial. MySQL sebenarnya merupakan turunan salah satu konsep utama dalam basis data yang telah ada sebelumnya, yaitu SQL (*Structured Query Language*). SQL adalah sebuah konsep pengoperasian basis data, terutama untuk proses pemilihan atau seleksi dan pemasukan data, yang memungkinkan pengoperasian data dikerjakan dengan mudah.

Kehandalan suatu sistem basis data dapat diketahui dari cara kerja pengoptimasiannya dalam melakukan proses perintah-perintah SQL yang dibuat oleh pengguna maupun program-program aplikasi yang memanfaatkannya. Sebagai *server* basis data, MySQL mendukung operasi basis data transaksional maupun operasi basis data non-transaksional. Pada modus operasi non-transaksional, MySQL dapat dikatakan handal dalam hal unjuk kerja dibandingkan *server* basis data kompetitor lainnya. Namun pada modus non-transaksional tidak ada jaminan atas reliabilitas terhadap data yang tersimpan, karenanya modus non-transaksional hanya cocok untuk jenis aplikasi yang tidak membutuhkan reliabilitas data seperti aplikasi blogging berbasis web (wordpress), CMS, dan sejenisnya. Untuk kebutuhan sistem yang ditujukan untuk bisnis sangat disarankan untuk menggunakan modus basis data transaksional, hanya saja sebagai konsekuensinya unjuk kerja MySQL pada modus transaksional tidak secepat unjuk kerja pada modus non-transaksional.

## 2.8 *Mitmproxy*

*Mitmproxy* adalah sebuah *interception proxy* untuk HTTP dengan antarmuka pengguna *console* yang ditulis dengan

bahasa *Python*. *Mitmproxy* merupakan sebuah perangkat lunak yang interaktif dimana *Mitmproxy* memungkinkan dapat memotong dan memodifikasi HTTP *requests* atau *response* dengan sangat cepat.

*Mitmproxy* ada sebuah *proxy* berkemampuan SSL yang berfungsi sebagai *man-in-the-middle* untuk komunikasi HTTP dan HTTPS. Untuk dapat mengetahui atau memodifikasi komunikasi HTTPS, *mitmproxy* berupra-pura menjadi *server* ke *client* dan *client* ke server, sementara itu *mitmproxy* diposisikan di tengah-tengah berfungsi untuk menerjemahkan lalu lintas dari keduanya. *Mitmproxy* menghasilkan sertifikat *on-the-fly* untuk mengetahui *client* agar percaya bahwa mereka berkomunikasi dengan *server*.

Pertama kali *mitmproxy* dimulai, maka akan menghasilkan sertifikat SSL yang berada pada `/.mitmproxy/cert.pem`. Sertifikat ini akan digunakan untuk *browser-side*. Karena tidak akan cocok dengan *domain* yang *client* kunjungi, dan tidak akan memverifikasi terhadap otoritas sertifikasi, *client* harus menambahkan pengecualian untuk setiap situs yang *client* kunjungi. Permintaan SSL dicegat dengan hanya mengamsumsikan bahwa semua permintaan `CONNECT` adalah HTTPS. Sambungan dari *browser* dibungkus SSL, dan kita membaca permintaan dengan berpura-pura menjadi *server* yang menghubungkan.

## 2.9 *VirtualBox*

*VirtualBox* merupakan salah satu produk perangkat lunak yang sekarang dikembangkan oleh Oracle. Aplikasi ini pertama kali dikembangkan oleh perusahaan Jerman, Innotek GmbH. Februari 2008, Innotek GmbH diakuisi oleh Sun Microsystems. Sun Microsystems kemudian juga diakuisi oleh Oracle. *VirtualBox* berfungsi untuk melakukan virtualisasi sistem

operasi. *VirtualBox* juga dapat digunakan untuk membuat virtualisasi jaringan komputer sederhana. Penggunaan *VirtualBox* ditargetkan untuk *server*, desktop, dan penggunaan *embedded*.

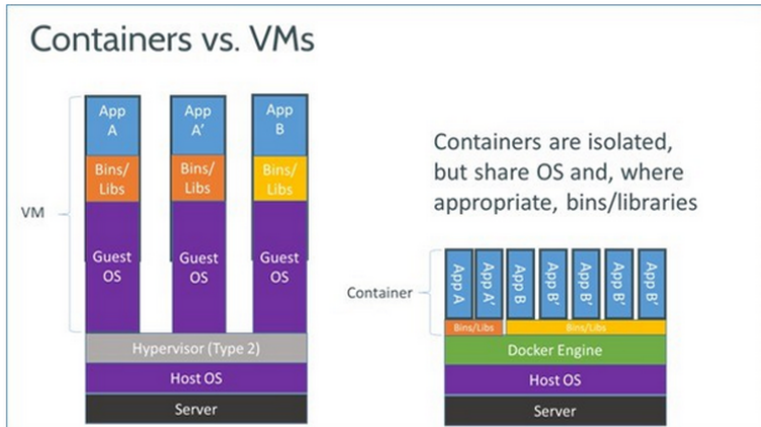
Berdasarkan jenis VMM yang ada, *VirtualBox* merupakan jenis *hypervisor type 2*. *VirtualBox* sendiri memiliki berbagai macam kegunaan, diantaranya *VirtualBox* dapat memainkan semua sistem operasi baik itu menggunakan windows, linux, atau turunan linux lainnya. *VirtualBox* juga dapat dipergunakan untuk mengujicoba OS baru. *VirtualBox* juga dapat digunakan sebagai media untuk membuat simulasi jaringan.

## 2.10 *Crontab*

## 2.11 *Docker*

Docker adalah sebuah aplikasi yang bersifat *open source* yang berfungsi sebagai wadah untuk memasukkan sebuah perangkat lunak secara lengkap beserta semua hal yang dibutuhkan oleh perangkat lunak tersebut agar dapat berfungsi sebagaimana mestinya. *Docker* dapat dijalankan di berbagai sistem operasi, pengembang dapat dengan mudah menggunakan layanan *docker* melalui <https://hub.docker.com> untuk mengunduh *images* ataupun membuat *images* yang diinginkan. [?] Perbedaan antara *docker* dan *virtual machine* ditunjukkan pada gambar 2.1





**Gambar 2.1:** Perbandingan *docker* dan virtual machine

### 2.11.1 *Docker Container*

*Docker container* atau kontainer *docker* bisa dikatakan sebagai sebuah wadah atau tempat, dimana kontainer *docker* ini dibuat dengan menggunakan *docker image*. Saat kontainer *docker* dijalankan, maka akan terbentuk sebuah *layer* di atas *docker image*. Contohnya saat menggunakan *image* Ubuntu, kemudian membuat sebuah kontainer *docker* dari *image* Ubuntu tersebut dengan nama *mitmproxy-ubuntu*. Setelah itu dilakukan pemasangan sebuah perangkat lunak, misalnya *mitmproxy*, maka secara otomatis kontainer *docker* *mitmproxy-ubuntu* akan berada di atas *layer image* Ubuntu, dan di atasnya lagi merupakan *layer mitmproxy* berada. *Docker Kontainer* atau Kontainer *docker* ke depannya dapat digunakan untuk menghasilkan sebuah *docker images*. *Docker images* yang dihasilkan dari kontainer *docker* itu sendiri nantinya dapat digunakan kembali untuk membuat kontainer *docker* yang lainnya.

### 2.11.2 *Docker Images*

*Docker images* adalah sebuah *blueprint* atau rancangan dasar dari sebuah perangkat lunak berbasis *docker* yang bersifat *read-only*. *Blueprint* ini sendiri merupakan sebuah sistem operasi atau sistem operasi yang telah dipasang berbagai perangkat lunak dan pustaka pendukung. *Docker images* berfungsi untuk membuat kontainer *docker*, dimana dengan menggunakan satu *docker image* dapat dibuat lebih dari satu kontainer *docker*. *Docker image* sendiri dapat menyelesaikan permasalahan yang dikenal dengan "*dependency hell*", dimana sulitnya untuk melengkapi dependensi sebuah perangkat lunak. Permasalahan tersebut dapat diselesaikan karena semua kebutuhan perangkat lunak sudah berada di dalamnya.

### 2.11.3 *Docker Registry*

*Docker Registry* adalah kumpulan dari berbagai macam *docker image* yang bersifat tertutup maupun terbuka yang dapat diakses di <https://hub.docker.com/> atau dapat diakses pada *server* sendiri. Dengan menggunakan *docker registry*, seseorang dapat menggunakan *docker image* yang telah dibuat oleh orang lainnya. Hal seperti ini dapat mempermudah seseorang untuk melakukan pengembangan dan juga transfer aplikasi.

## BAB III

### DESAIN DAN PERANCANGAN

Pada bab ini dibahas mengenai analisis dan perancangan dari sistem.

#### 3.1 Deskripsi Umum Sistem

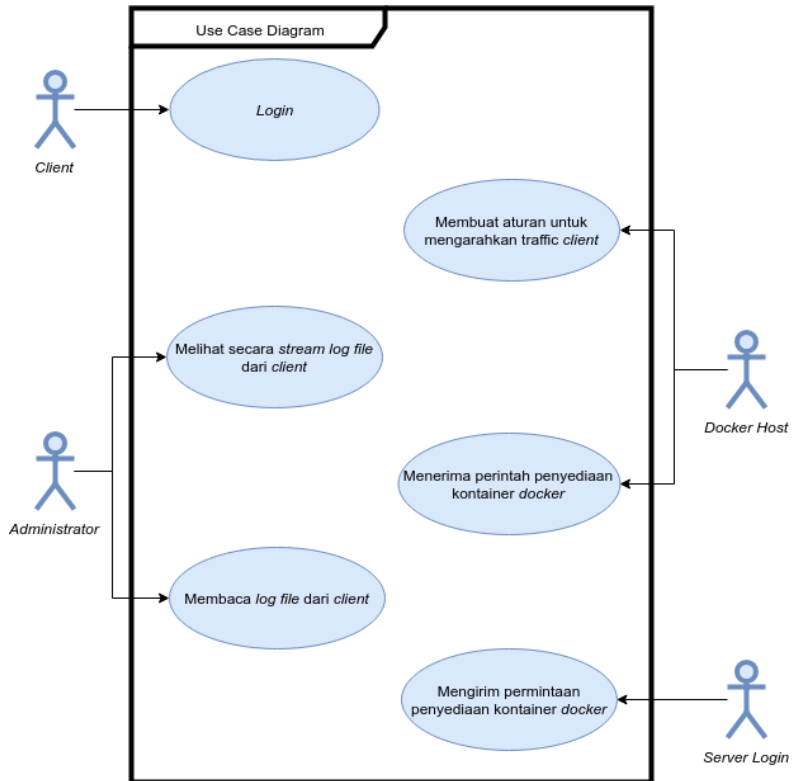
Sistem yang akan dibuat adalah sebuah sistem yang dapat membuat sebuah kontainer *docker* secara otomatis untuk setiap satu *client* yang telah *login* ke dalam sistem. Saat *client* belum *login* ke dalam sistem, maka *client* tersebut akan diarahkan ke halaman *login* dari sistem. Saat *client* mencoba untuk *login* ke dalam sistem, maka sistem akan melakukan pengecekan di dalam basis data apakah *username* dan *password* yang diinputkan sudah benar atau salah.

Setelah *client* berhasil *login* ke dalam sistem, sistem akan mengirimkan perintah untuk membuat kontainer *docker* yang berisikan *mitmproxy* ke *docker host*. Setelah berhasil membuat kontainer *docker* untuk *client* tersebut, maka *traffic* internet dari *client* tersebut akan diarahkan ke kontainer *docker* berisikan *mitmproxy* yang baru saja dibuat. Setelah itu *client* dapat mengakses internet.

#### 3.2 Kasus Penggunaan

Terdapat empat aktor dalam sistem yang akan dibuat yaitu *Client*, *Server Login*, *Administrator*, dan *Docker Host*. *Client* adalah aktor yang melakukan proses *login* ke dalam sistem, *server login* adalah aktor yang melakukan proses permintaan penyediaan kontainer *docker*, *administrator* adalah aktor yang melakukan monitoring kontainer *docker* yang sedang berjalan, sedangkan *docker host* adalah aktor yang akan menjadi tempat penyedia kontainer dan menerima perintah penyediaan kontainer. Diagram kasus penggunaan menggambarkan kebutuhan -

kebutuhan yang harus dipenuhi sistem. Diagram kasus penggunaan digambarkan pada Gambar 3.1.



**Gambar 3.1:** Digram Kasus Penggunaan

Digram kasus penggunaan pada Gambar 3.1 dideskripsikan masing-masing pada Tabel 3.1.

**Tabel 3.1:** Daftar Kode Kasus Penggunaan

<b>Kode Kasus Penggunaan</b>	<b>Nama Kasus Penggunaan</b>	<b>Keterangan</b>
UC-0001	<i>Login</i>	<i>Client</i> dapat <i>login</i> ke dalam sistem.
UC-0002	Mengirim Permintaan Penyediaan Kontainer <i>Docker</i>	<i>Server login</i> dapat mengirimkan permintaan penyediaan kontainer <i>docker</i> pada <i>docker host</i> .
UC-0003	Menerima Perintah Penyediaan Kontainer <i>Docker</i>	Proses dimana <i>docker host</i> akan menerima perintah dari sistem, untuk menyediakan kontainer secara otomatis.
UC-0004	Membuat Aturan untuk Mengarahkan <i>Traffic Client</i>	Proses dimana <i>docker host</i> akan membuat aturan untuk mengarahkan <i>traffic client</i> ke halaman <i>login</i> dari sistem atau untuk membuat aturan untuk mengarahkan <i>traffic client</i> ke kontainer <i>docker</i> dari tiap-tiap <i>client</i> .

**Tabel 3.1:** Daftar Kode Kasus Penggunaan

<b>Kode Kasus Penggunaan</b>	<b>Nama Kasus Penggunaan</b>	<b>Keterangan</b>
UC-0005	Membaca <i>Log File</i> dari <i>Client</i>	Proses dimana <i>administrator</i> dari sebuah jaringan dapat membaca <i>log file</i> dari <i>client</i> sampai pada <i>client</i> terakhir mengakses internet.
UC-0006	Melihat Secara <i>Stream Log File</i> dari <i>Client</i>	Proses dimana <i>administrator</i> dari sebuah jaringan dapat melihat <i>log file</i> dari <i>client</i> secara langsung atau <i>live</i> .

### 3.3 Arsitektur Sistem

Pada Sub-bab ini, dibahas mengenai tahap analisis arsitektur, analisis teknologi dan desain sistem yang akan dibangun.

#### 3.3.1 Desain Umum Sistem

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

1. Pembuatan halaman *login* dari sebuah sistem.
2. Pembuatan aturan untuk mengarahkan *traffic client* ke halaman *login* dari sistem.
3. Pembuatan *middleware* untuk menerima permintaan dari *client*.
4. Pembuatan aturan untuk mengarahkan *traffic client* ke kontainer *docker* dari tiap-tiap *client*.

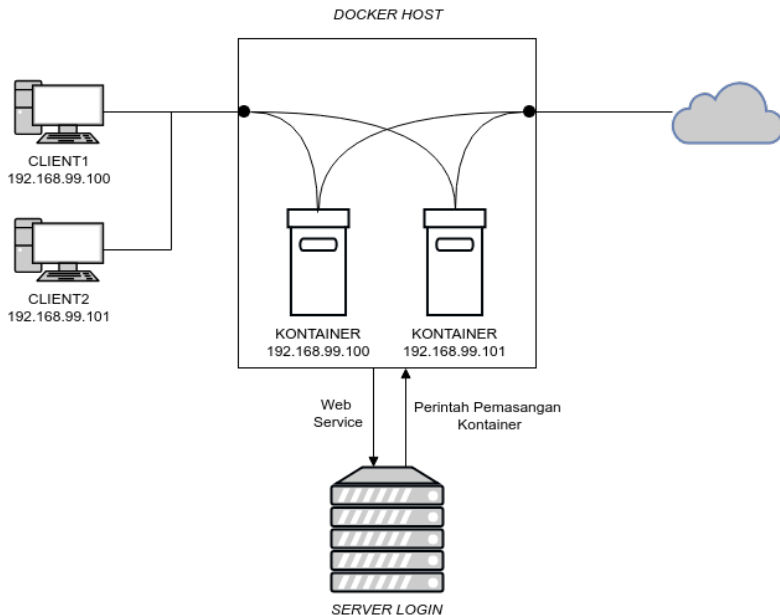
5. Pemasangan kontainer pada *docker host*.
6. Pembuatan halaman *administrator* untuk membaca *log file* dari *client*.

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

1. Pembuatan halaman *login* dari sebuah sistem.  
 Berfungsi sebagai tampilan antarmuka dari halaman *login* sebuah sistem untuk *client*. Selain itu juga berfungsi untuk mengirimkan permintaan penyediaan kontainer *docker* ke *docker host*.
2. Pembuatan aturan untuk mengarahkan *traffic client* ke halaman *login* dari sistem.  
 Berfungsi untuk mengarahkan tiap *client* yang belum *login* ke dalam sistem ke halaman *login* dari sistem. Hal ini dilakukan dengan menjalankan sebuah *script* dengan menggunakan *iptables* pada *docker host*.
3. Pembuatan *middleware* untuk menerima permintaan dari *client*. Berfungsi untuk menerima permintaan pembuatan kontainer *docker* dari *client*. Selain itu juga berfungsi untuk membuat kontainer *docker* secara otomatis.
4. Pembuatan aturan untuk mengarahkan *traffic client* ke kontainer *docker* dari tiap-tiap *client*.  
 Berfungsi untuk mengarahkan tiap *client* yang telah berhasil *login* ke kontainer *docker* dari tiap-tiap *client*. Hal ini dilakukan dengan menjalankan sebuah *script* dengan menggunakan *iptables* pada *docker host*.
5. Pemasangan kontainer pada *docker host*.  
 Berfungsi untuk memasang kontainer *docker* pada *docker host* secara otomatis. Hal ini dilakukan dengan menjalankan sebuah perintah penyediaan kontainer pada *docker host*.
6. Pembuatan halaman *administrator* untuk membaca *log file*

dari *client*.

Berfungsi untuk melihat apa saja yang telah diakses oleh *client*. Log yang tersimpan terdapat log HTTP maupun log HTTPS. Hal ini dilakukan dengan menjalankan sebuah perintah untuk melihat *log file* dari suatu *client*.



**Gambar 3.2:** Arsitektur Komponen Sistem

Pada Gambar 3.2 ditunjukkan arsitektur sistem secara umum dengan detail-detail dari komponen yang terdapat didalamnya. Setiap komponen tersebut akan diimplementasikan dengan teknologi pendukung yang dibutuhkan.

Nantinya tiap *client* akan mempunyai satu kontainer *docker* dan satu *port* secara pribadi. *Traffic* dari *client* tersebut akan diarahkan menuju ke kontainer *dockernya* dari tiap-tiap *client*, setelah itu *client* baru dapat mengakses internet.



### 3.3.2 Pembuatan Halaman *Login* dari Sebuah Sistem.

Pembuatan halaman *login* dari sebuah sistem adalah komponen yang bertugas untuk menyediakan tampilan antarmuka dari halaman *login* untuk *client*. Awalnya semua *traffic* diarahkan menuju ke halaman *login* dari sebuah sistem, karena diasumsikan bahwa semua *client* diasumsikan belum *login* ke dalam sistem. Supaya *client* dapat mengakses internet, maka *client* harus *login* ke dalam sistem terlebih dahulu dengan memasukkan *username* dan *password* dari *client* tersebut.

Dikarenakan ada beberapa kebutuhan yang harus dipenuhi, komponen pada pembuatan halaman *login* dari sebuah sistem dibagi lagi menjadi dua sub komponen, yaitu:

1. Basis Data

Basis data pada komponen pembuatan halaman *login* dari sebuah sistem berfungsi sebagai tempat penyimpanan data *username* dan *password* yang digunakan untuk *login* ke dalam sistem. Basis data juga berfungsi sebagai tempat penyimpanan data kontainer *docker* yang sudah dibuat.

2. *Web Service*

*Web service* berfungsi sebagai antarmuka untuk *client* ketika *client* akan *login* ke dalam sistem. Selain itu *web service* juga berfungsi untuk mengirimkan permintaan penyediaan kontainer *docker* ke *docker host*. Selain itu *web service* juga berfungsi sebagai penerima permintaan dari *client*, yang nantinya akan membuat sebuah kontainer *docker* secara otomatis pada *docker host*.

Pada tugas akhir ini, bahasa Python dipilih sebagai bahasa pemrograman yang digunakan untuk mengimplementasikannya. Lalu, pada bagian penyimpanan data atau basis data, MySQL dipilih sebagai RDBMS untuk tugas akhir ini.

### 3.3.2.1 Desain Basis Data

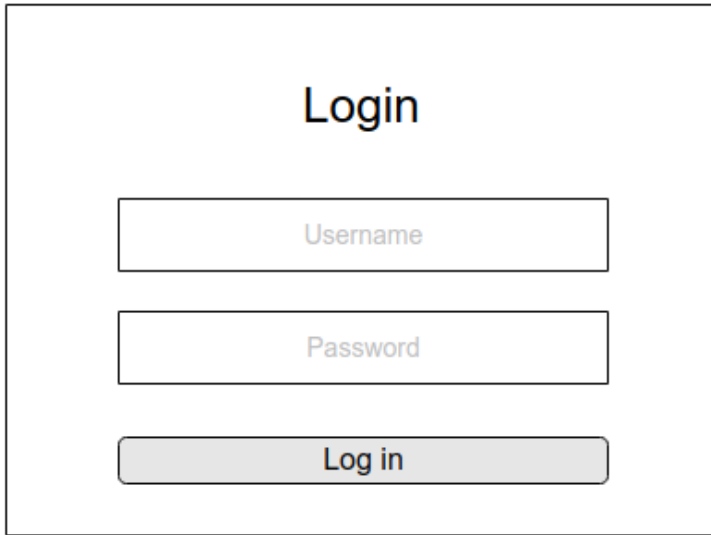
Komponen basis data berfungsi sebagai tempat penyimpanan data *username* dan *password* yang digunakan untuk *login* ke dalam sistem. Dalam basis data ini terdapat satu entitas dan empat atribut, ditunjukkan pada Tabel 3.2

**Tabel 3.2:** Atribut basis data nrp-mahasiswa

No	Kolom	Tipe	Keterangan
1	id	int(11)	Sebagai primary key pada tabel, nilai awal adalah AUTO_INCREMENT.
2	username	varchar(50)	Menunjukkan NRP dari mahasiswa yang telah terdaftar.
3	password	varchar(50)	Menunjukkan password dari NRP mahasiswa yang telah terdaftar.
4	isLogin	int(11)	Status apakah nrp tersebut sedang digunakan (1), atau sedang tidak digunakan (0).

### 3.3.2.2 Desain Web Service

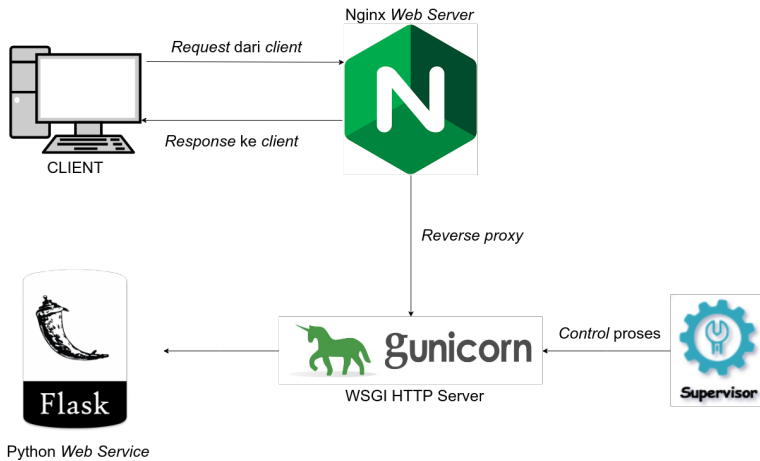
Komponen *web service* berfungsi untuk menyediakan antar muka halaman *login* untuk *client* dan untuk mengirimkan permintaan pembuatan kontainer *docker* secara otomatis pada *docker host* setelah terdapat *client* yang berhasil *login* ke dalam sistem. Halaman *login* akan menggunakan Material UI untuk mendapatkan tampilan yang sederhana dan nyaman untuk digunakan. Desain *web service* untuk halaman *login* dari sistem dapat dilihat pada Gambar 3.3.



The image shows a login form design within a rectangular border. At the top center is the word "Login" in a large, bold, black font. Below it are three input fields stacked vertically. The first field is labeled "Username" in a light gray font. The second field is labeled "Password" in a light gray font. The third field is a button labeled "Log in" in a bold black font, with a light gray background and a darker gray border.

**Gambar 3.3:** Desain Halaman *Login*

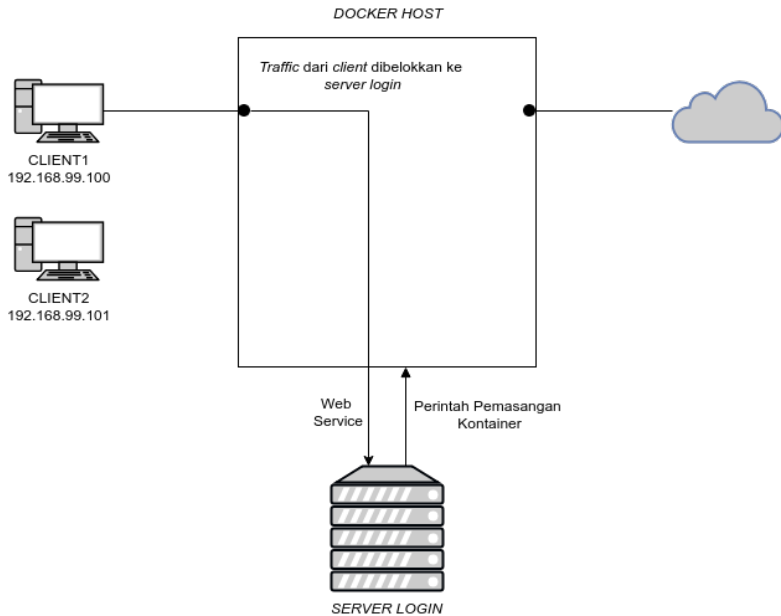
Lalu untuk desain *backend* dari *web serve* untuk halaman *login* akan menggunakan bahasa pemrograman Python dengan kerangka kerja *flask* yang akan dijalankan dengan *unicorn*. Kemudian *unicorn* akan dijalankan dengan *supervisor* sebagai *service*. Lalu akan digunakan *nginx* sebagai *web server* dari halaman *login* dari sebuah sistem. Desain *backend* dari *web service* untuk halaman *login* dapat dilihat pada Gambar 3.4.



**Gambar 3.4:** Desain *Backend* dari Halaman *Login*

### 3.3.3 Perancangan Pembuatan Aturan untuk Mengarahkan *Traffic Client* ke Halaman *Login* dari Sistem

Pembuatan aturan untuk mengarahkan *traffic client* ke halaman *login* dari sistem adalah komponen yang bertugas untuk membelokkan *traffic* dari *client* yang akan menuju ke internet. Awalnya semua *traffic* dari satu *subnet client* tersebut akan diarahkan ke halaman *login* dari sistem dengan membuat sebuah aturan menggunakan *iptables*, karena asumsinya adalah belum ada *client* yang berhasil *login* ke dalam sistem. Desain perancangan pembuatan aturan untuk mengarahkan *traffic client* ke halaman *login* dapat dilihat pada Gambar 3.5 .



**Gambar 3.5:** Desain Mengarahkan *Traffic Client* ke Halaman *Login*

### 3.3.4 Pembuatan *Middleware* untuk Menerima Permintaan dari *Client*

Pembuatan *middleware* untuk menerima permintaan dari *client* adalah komponen yang bertugas untuk menerima permintaan dari *client* yang telah berhasil *login* ke dalam sistem. Permintaan yang dikirimkan oleh *client* adalah permintaan untuk membuat kontainer *docker* secara otomatis. Nantinya setiap satu *client* yang berhasil *login* ke dalam sistem akan dibuatkan satu kontainer *docker*.

Dikarenakan ada beberapa kebutuhan yang harus dipenuhi, komponen pada pembuatan *middleware* untuk menerima permintaan dari *client* dibagi lagi menjadi dua buah komponen, yaitu:

### 1. Basis Data

Basis data berfungsi sebagai tempat penyimpanan data kontainer *docker* yang sudah dibuat.

### 2. *Web Service*

*Web service* berfungsi sebagai penerima permintaan dari *client*, yang nantinya akan membuat sebuah kontainer *docker* secara otomatis pada *docker host*.

Sama seperti komponen pembuatan halaman *login* dari sebuah sistem, pada tugas akhir ini, bahasa Python dipilih sebagai bahasa pemrograman yang digunakan untuk mengimplementasikannya. Lalu, pada bagian penyimpanan data atau basis data, MySQL dipilih sebagai RDBMS untuk tugas akhir ini.

#### 3.3.4.1 Desain Basis Data

Komponen basis data berfungsi sebagai tempat penyimpanan data kontainer *docker* yang sudah dibuat. Dalam basis data ini terdapat satu entitas dan empat atribut, ditunjukkan pada Tabel 3.3.

**Tabel 3.3:** Atribut basis data kontainer

No	Kolom	Tipe	Keterangan
1	id	int(11)	Sebagai primary key pada tabel, nilai awal adalah <code>AUTO_INCREMENT</code> .
2	username	varchar(50)	Menunjukkan NRP dari mahasiswa yang telah berhasil dibuatkan satu kontainer <i>docker</i> .

**Tabel 3.3:** Atribut basis data kontainer

No	Kolom	Tipe	Keterangan
3	ip	varchar(50)	Menunjukkan IP dari <i>client</i> yang telah berhasil dibuatkan satu kontainer <i>docker</i> .
4	port	varchar(50)	Menunjukkan port dari <i>client</i> yang telah berhasil dibuatkan satu kontainer <i>docker</i> .
5	createdAt	datetime	Menunjukkan waktu pertama kali kontainer <i>docker</i> tersebut dibuat.

### 3.3.4.2 Desain Web Service

Komponen *web service* berfungsi untuk menerima permintaan dari *client* untuk membuat satu kontainer *docker* pada *docker host*. Kontainer *docker* yang akan dibuat pada *docker host* akan dibuat secara otomatis oleh sistem, dan kontainer *docker* yang dibuat akan memiliki nama sesuai dengan IP dari *client* yang telah berhasil *login* ke dalam sistem. Setelah menerima permintaan dari *client*, maka sistem akan mengirimkan perintah untuk membuat kontainer *docker* khusus untuk satu *client*.

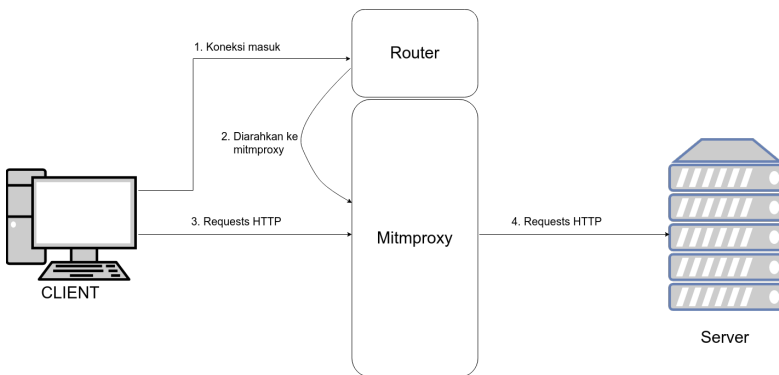
### 3.3.5 Perancangan Pemasangan Kontainer pada Docker Host

Pemasangan kontainer adalah komponen yang berfungsi untuk memasang kontainer *docker* yang berisi *mitmproxy* pada *docker host* setelah ada permintaan dari *client* yang telah berhasil *login* ke dalam sistem. Proses ini dilakukan secara otomatis, dan nama

dari kontainer *docker* tersebut akan sesuai dengan IP dari *client* yang telah berhasil *login* ke dalam sistem.

Saat kontainer *docker* telah berhasil dibuat, maka kontainer *docker* tersebut akan mempunyai satu port khusus yang sama dengan *client* yang baru saja *login*. Port khusus nantinya akan digunakan untuk mengarahkan *traffic* dari *client* yang akan mengakses internet.

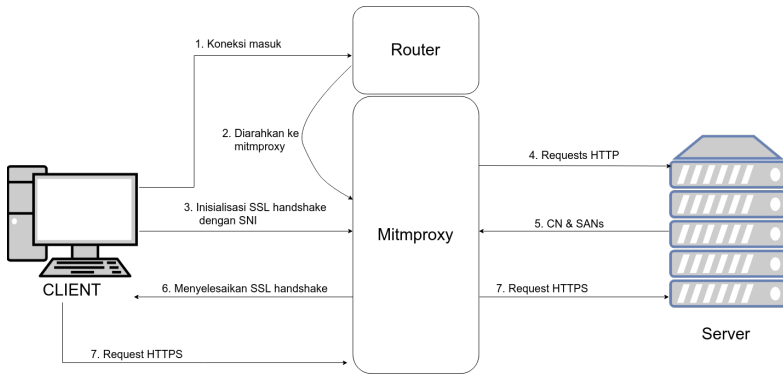
*Image mitmproxy* dipilih sebagai *image* pada kontainer *docker* karena *mitmproxy* merupakan sebuah perangkat lunak yang interaktif dimana *mitmproxy* memungkinkan dapat memotong dan memodifikasi HTTP *requests* atau *response* dengan sangat cepat. *Mitmproxy* sendiri juga dapat berjalan dengan *transparent mode* sehingga *client* tidak mengetahui jika *traffic* dari *client* tersebut ternyata melalui *mitmproxy*. Gambar alur kerja dari *mitmproxy* dengan *transparent mode* untuk HTTP dapat dilihat pada Gambar 3.6.



**Gambar 3.6:** Alur kerja dari *mitmproxy* *transparent* HTTP

Sedangkan gambar alur kerja dari *mitmproxy* dengan *transparent mode* untuk HTTPS dapat dilihat pada Gambar 3.7.

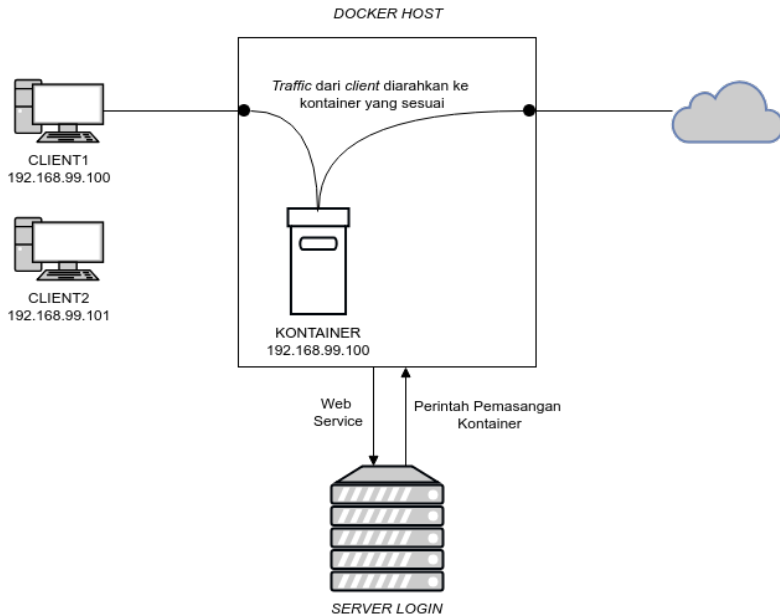




**Gambar 3.7:** Alur kerja dari *mitmproxy* transparent HTTPS

### 3.3.6 Pembuatan Aturan untuk Mengarahkan *Traffic Client* ke Kontainer *Docker* dari Tiap-Tiap *Client*

Pembuatan aturan untuk mengarahkan *traffic client* ke kontainer *docker* dari tiap-tiap *client* adalah komponen yang bertugas untuk mengarahkan satu client ke satu kontainer *docker* yang sesuai. Setelah *client* berhasil *login* ke dalam sistem, maka aturan ini akan dibuat menggunakan *iptables*. Lalu *client* juga akan diberikan sebuah aturan dengan menggunakan *iptables* yang memperbolehkan *client* tersebut mengakses internet. Desain dari pembuatan aturan untuk mengarahkan *traffic client* ke kontainer *docker* dari tiap-tiap *client* dapat dilihat pada Gambar 3.8.

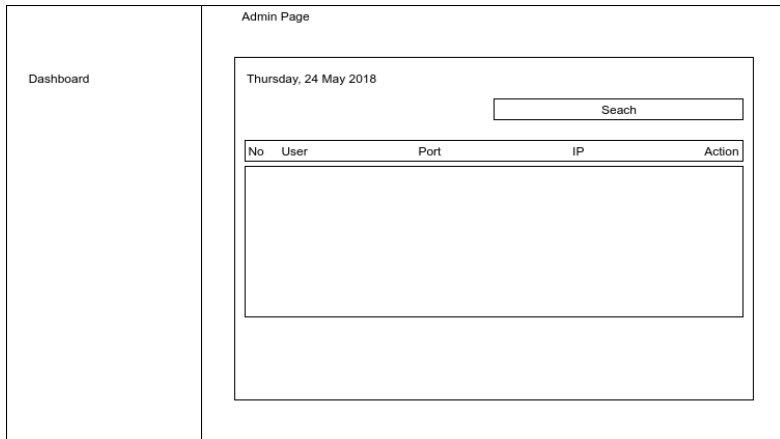


**Gambar 3.8:** Desain pembuatan aturan untuk mengarahkan *traffic client* ke kontainer *docker*

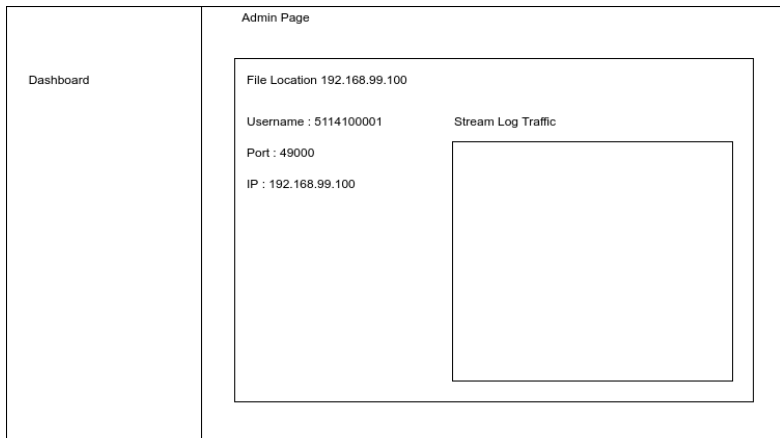
### 3.3.7 Pembuatan Halaman *Administrator* untuk Membaca *Log File* dari *Client*

Pembuatan halaman *administrator* untuk membaca *log file* dari *client* adalah komponen yang bertugas untuk mengunduh *log file* dari *client* dan juga untuk membaca *log file* dari *client* secara langsung atau *live* maupun hanya pada saat *client* terakhir mengakses internet.

Halaman *administrator* akan menggunakan Bootstrap 4 untuk mendapatkan tampilan yang sederhana dan nyaman untuk digunakan. Desain dari halaman *administrator* dapat dilihat pada Gambar 3.9 dan Gambar 3.10 .



**Gambar 3.9:** Desain halaman dashboard *administrator traffic client* ke kontainer *docker*



**Gambar 3.10:** Desain pembuatan aturan untuk mengarahkan *traffic client* ke kontainer *docker*

*(Halaman ini sengaja dikosongkan)*

## BAB IV

### IMPLEMENTASI

Setelah melewati proses perancangan mengenai sistem yang akan dibuat, maka akan dilakukan implementasi dari sistem tersebut. Bab ini akan membahas mengenai implementasi dari sistem yang meliputi proses pembuatan setiap komponen sehingga sistem dapat berjalan dengan baik. Masing-masing proses pembuat komponen akan dilengkapi dengan *pseudocode* atau konfigurasi dari sistem.

#### 4.1 Lingkungan Implementasi

Dalam mengimplementasikan sistem, digunakan beberapa perangkat pendukung sebagai berikut.

##### 4.1.1 Perangkat Keras

Perangkat keras yang digunakan dalam pengembangan sistem adalah sebagai berikut:

1. Komputer dengan *processor* Intel(R) Core(TM) i5-2120 CPU @ 3.30GHz dan RAM 8GB
2. Dua Komputer dengan *processor* Intel(R) Core(TM)2 Duo CPU E7200 @ 2.53GHz dan RAM 1GB

##### 4.1.2 Perangkat Lunak

Perangkat lunak yang digunakan dalam pengembangan sistem adalah sebagai berikut:

1. Sistem Operasi Linux Mint 18.03 64 Bit sebagai *docker host*.
2. Sistem Operasi Ubuntu 14.04 LTS 64 Bit sebagai *client*.
3. Sistem Operasi Ubuntu Server 16.04 LTS 64 Bit sebagai *server login*.
4. *Python* versi 3.5.2 untuk pengembangan web service.
5. *Flask* versi 1.0.2 sebagai kerangka kerja *Python*.

6. *Gunicorn* versi 19.8.1
7. *Supervisor* versi 3.2.0
8. *Nginx* versi 1.10.3
9. *Mitmproxy* versi 3.0.4 untuk mencatat semua *traffic* dari *client*.
10. MySQL versi 5.7.18 untuk Sistem Manajemen Basis Data.
11. *Docker* versi 1.13.1 sebagai kontainer yang akan di pasangkan pada *server*.
12. *Iptables* versi 1.6.0 untuk membuat aturan terhadap *client*.
13. *Laravel* versi 5.4 sebagai kerangka kerja untuk halaman *administrator*.
14. *VIM* versi 7.4.1 sebagai *text editor*.

#### 4.2 Implementasi Pembuatan Halaman *Login* dari Sebuah Sistem

Halaman *login* dibangun pada sebuah *server* dengan IP 10.151.36.173 dengan menggunakan sistem operasi Ubuntu Server 16.04 LTS 64 Bit. Pada implementasi pembuatan halaman *login* dari sebuah sistem menggunakan perangkat lunak antara lain:

1. *Python* versi 3.5.2.
2. *Flask* versi 1.0.2.
3. *Gunicorn* versi 19.8.1.
4. *Supervisor* versi 3.2.0.
5. *Nginx* versi 1.10.3.

Lalu sistem operasi yang digunakan adalah sistem operasi Ubuntu Server 16.04 LTS 64 Bit, yang akan dipasang pada *virtual machine* di *Proxmox*. *Python* akan berfungsi sebagai komponen dasar pembangunan sistem yang akan dibangun dengan menggunakan kerangka kerja *Flask* dan dijalankan dengan *Gunicorn* pada *server* dengan IP 10.151.36.173 dengan *port* 4000. Lalu *Supervisor* akan berfungsi sebagai

sebuah *service* yang akan selalu menjalankan *Gunicorn*. Sedangkan *Nginx* akan berfungsi sebagai *web server* untuk perangkat lunak halaman *login* yang dijalankan oleh *Gunicorn* pada *server* dengan IP 10.151.36.173 dengan *port* 4000 supaya bisa diakses oleh *client*. Implementasi pembuatan halaman *login* dari sebuah sistem akan terbagi menjadi implementasi *web service* dan implementasi basis data.

#### 4.2.1 Implementasi *Web Service* pada Halaman *Login*

Diperlukan beberapa tahap, antara lain pemasangan perangkat lunak dan tahap konfigurasi. Tahap pemasangan perangkat lunak dan tahap konfigurasi pada *server* untuk halaman *login* dijelaskan pada Lampiran A.

Perlu diperhatikan ketika menambahkan atau mengubah konfigurasi *Supervisor* pada `/etc/supervisor/conf.d/` di *server* untuk halaman *login*, perlu dilakukan *reload Supervisor* dengan menjalankan *command* pada terminal seperti pada Kode Sumber 4.1.

```
sudo supervisorctl reread
sudo supervisorctl reload
sudo supervisorctl status
```

**Kode Sumber 4.1:** Command untuk Reload Supervisor

Perlu diperhatikan pula ketika menambahkan atau mengubah konfigurasi *Nginx* pada `/etc/nginx/sites-available/` di *server* untuk halaman *login*, perlu dilakukan aktivasi konfigurasi *Nginx* dengan menjalankan *command* pada terminal seperti pada Kode Sumber 4.2.

```
sudo ln -s /etc/nginx/sites-available/app
    /etc/nginx/sites-enabled/app
```

**Kode Sumber 4.2:** Command untuk mengaktifkan konfigurasi Nginx

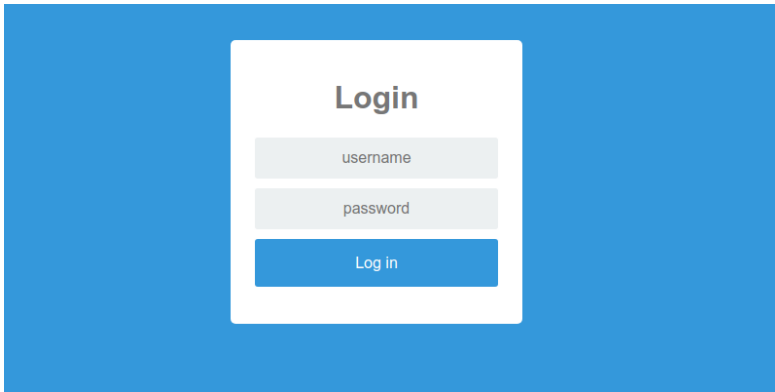
Setelah itu, jalankan Kode Sumber 4.3 supaya konfigurasi yang baru saja diaktifkan dapat digunakan.

```
sudo service nginx restart
```

**Kode Sumber 4.3:** Command untuk merestart Nginx

#### 4.2.1.1 Implementasi Tampilan Antarmuka Halaman *Login*

Halaman *login* merupakan halaman utama yang menampilkan sebuah *form input* untuk *client*. Pada halaman ini terdapat dua *form input*, yaitu *form input* untuk Username atau NRP dari *client* dan juga *form input* untuk Password dari *client*. Implementasi antarmuka halaman *login* dapat dilihat pada Gambar 4.1.



**Gambar 4.1:** Halaman *Login*

#### 4.2.1.2 Rute *Web Service* pada Halaman *Login*

Pada halaman *login* diperlukan adanya rute-rute yang bisa diakses untuk melayani *client*, supaya *client* dapat membuka tampilan antar muka dari halaman *login* dan juga supaya *client*



dapat mengirimkan permintaan untuk membuat kontainer *docker* pada *docker host*. Daftar rute yang disediakan oleh halaman *loign* tertera pada Tabel 4.1.

**Tabel 4.1:** Daftar Rute *Web Service*

HTTP Method	Rute	Deskripsi
GET	/	Berfungsi untuk mengarahkan <i>redirect</i> ke rute <i>login</i> dengan <i>method</i> GET.
GET	/login	Berfungsi untuk menampilkan tampilan grafis antar muka halaman <i>login</i> ketika <i>client</i> belum <i>login</i> ke dalam sistem dan untuk menampilkan tampilan grafis antar muka halaman sukses <i>login</i> ketika <i>client</i> telah berhasil <i>login</i> ke dalam sistem.
POST	/login	Berfungsi untuk menyimpan data hasil <i>input</i> dari <i>client</i> dan mengirimkan perintah untuk membuat kontainer <i>docker</i> yang berisikan <i>mitmproxy</i> secara otomatis pada <i>docker host</i> .

#### 4.2.1.3 *Pseudocode Web Service pada Halaman Login*

Ketika *client* belum *login* ke dalam sistem, maka akan diarahkan ke tampilan grafis antar muka dari halaman *login*. Lalu setelah *client* berhasil *login* ke dalam sistem, maka akan diarahkan ke tampilan grafis antar muka halaman sukses *login*. Pada Kode Sumber 4.4 diperlihatkan bagaimana implementasinya dalam bentuk *pseudocode*.

**Kode Sumber 4.4:** Pseudocode Web Service

```

1   Check whether the client is already login
    or not yet
2
3   if session.get login
4   open welcome page
5   else
6   open login page
7   if login success
8   session.get login = True
9   return
  
```

#### 4.2.2 Implementasi Basis Data pada Halaman *Login*

Berdasarkan hasil desain dan perancangan basis data pada bab 3 terdapat satu entitas yang diimplementasikan menjadi suatu tabel pada basis data MySQL, yaitu entitas *nrp-mahasiswa*. Detail implementasi *query* untuk membuat basis data dengan entitas *nrp-mahasiswa* seperti pada Kode Sumber 4.5.

```

CREATE TABLE nrp-mahasiswa (
  id int(11) PRIMARY KEY AUTO_INCREMENT,
  nrp VARCHAR(50)
  password VARCHAR(50)
  )
  
```

```
isLogin  int(11)
);
```

**Kode Sumber 4.5:** *Query* untuk membuat tabel testing

### 4.3 Implementasi Pembuatan Aturan untuk Mengarahkan *Traffic Client* ke Halaman *Login* dari Sistem

Pada implementasi pembuatan aturan untuk mengarahkan *traffic client* ke halaman *login* dari sistem diasumsikan bahwa belum ada *client* yang telah *login* ke dalam sistem. Karena diasumsikan bahwa belum ada *client* yang telah berhasil *login* ke dalam sistem, maka semua *client* tidak diperbolehkan untuk mengakses internet. Kemudian untuk mengarahkan *traffic* dari *client* dibuatkan beberapa *rules* dengan menggunakan *iptables* pada *Docker Host* dengan IP 10.151.36.134. seperti Kode Sumber 4.6.

```
iptables -I FORWARD 1 -s 192.168.99.0/24 -j
REJECT
iptables -I FORWARD 1 -s 192.168.99.0/24 -p
tcp d 10.151.36.173 --dport 4000 -j
ACCEPT
iptables -t nat -I PREROUTING 1 -p tcp -s
192.168.99.0/24 --dport 80 -j DNAT --to
10.151.36.173:4000
```

**Kode Sumber 4.6:** Command untuk mengarahkan *client* ke halaman *login*

*Rules* pertama berfungsi untuk melarang semua *client* untuk melewati *router*. *Rules* kedua berfungsi untuk mengizinkan semua *client* membuka halaman *login*. Sedangkan *rules* ketiga berfungsi untuk mengarahkan semua *traffic client* ke halaman *login*.

#### 4.4 Implementasi Pembuatan *Middleware*

*Middleware* dibangun pada *Docker Host* dengan IP 10.151.36.134 dengan menggunakan sistem operasi Linux Mint 18.03 64 Bit. *Middleware* merupakan komponen yang akan menerima permintaan dari *client*, mengirimkan perintah untuk membuat kontainer *docker* secara otomatis pada *docker host*, dan menentukan rute *traffic* dari *client* menuju ke internet sesuai kontainer *docker* masing-masing user. Implementasi *middleware* akan terbagi menjadi implementasi basis data dan implementasi *web service*.

Pada implementasi pembuatan *middleware* menggunakan perangkat lunak antara lain:

1. *Python* versi 3.5.2.
2. *Flask* versi 1.0.2.
3. *Docker* versi 1.13.1.

*Python* akan berfungsi sebagai komponen dasar pembangunan sistem, salah satunya adalah sebagai komponen dasar pembuatan *middleware*, sedangkan *Flask* akan berfungsi sebagai kerangka kerja untuk pembuatan *middleware*. Implementasi pembuatan *middleware* akan terbagi menjadi implementasi *web service* dan implementasi basis data dan.

##### 4.4.1 Implementasi *Web Service* pada *Middleware*

Diperlukan beberapa tahap, antara lain pemasangan perangkat lunak dan tahap konfigurasi. Tahap pemasangan perangkat lunak dan tahap konfigurasi pada *middleware* di *Docker Host* dijelaskan pada Lampiran B.

Perlu diperhatikan supaya *docker* dapat dijalankan ketika *docker host* menyala, jalankan Kode Sumber 4.7.

```
sudo systemctl enable docker
```

**Kode Sumber 4.7:** Command untuk instalasi Flask

#### 4.4.1.1 Rute *Web Service* pada *Middleware*

*Middleware* tidak memiliki antar muka grafis. Namun tetap diperlukan adanya rute-rute yang bisa diakses untuk melayani permintaan penyediaan kontainer *docker* dari *client*. Daftar rute yang disediakan oleh *middleware* tertera pada Tabel 4.2.

**Tabel 4.2:** Daftar Rute *Web Service*

HTTP Method	Rute	Deskripsi
POST	/test/endpoint/	Berfungsi untuk menyimpan data hasil <i>input</i> dari <i>client</i> dan mengirimkan perintah untuk membuat kontainer <i>docker</i> yang berisikan <i>mitmproxy</i> secara otomatis pada <i>docker host</i> .

#### 4.4.1.2 *Pseudocode Web Service* pada *Middleware*

Saat *client* telah memasukkan *input* ke sistem, sistem akan mencocokkan terlebih dahulu dengan basis data *kontainer*. Jika benar, maka sistem akan mengirimkan data *input* dari *client* ke *middleware*. Lalu *middleware* akan menyimpan data *input* dari *client* ke dalam sebuah *file*. Setelah itu *middleware* akan mengirimkan perintah untuk membuat sebuah kontainer *docker* yang berisikan *mitmproxy* pada *docker host*.

Saat *middleware* menyimpan data *input* dari *client* ke dalam sebuah *file*, yang disimpan adalah *username* atau NRP, *IP Address*, dan *port*. Nantinya *port* tersebut akan menjadi *port*

khusus untuk kontainer *docker* yang berisikan *mitmproxy* untuk *client* tersebut.

Saat kontainer *docker* yang berisikan *mitmproxy* akan dibuat pada *docker host*, sistem akan membuat kontainer *docker* dengan *mode network=host*, nama sesuai *IP Address* dari *client* tersebut, dan *port* kontainer *docker* sesuai dengan *port* yang sudah disimpan pada *file*.

Setelah kontainer *docker* yang berisikan *mitmproxy* berhasil dibuat, maka sistem akan membuat *rules* yang berfungsi untuk mengarahkan *traffic* dari *client* menuju ke kontainer *docker* milik *client* tersebut, dan memperbolehkan *client* untuk mengakses internet. Pada Kode Sumber 4.8 diperlihatkan bagaimana implementasinya dalam bentuk *pseudocode*.

**Kode Sumber 4.8:** Pseudocode Web Service

```

1  Check whether the client is already login
   or not yet
2
3  if session.get login
4  create container
5  add new rules to container
6  return
7  else
8  add new rules
9  client open page login
10 client login
11 return

```

#### 4.4.2 Implementasi Basis Data pada *Middleware*

Berdasarkan hasil perancangan basis data pada bab 3 terdapat 2 entitas yang diimplementasikan menjadi suatu tabel pada basis data MySQL, yaitu entitas *kontainer*. Detail implementasi

entitas `kontainer` tertera pada Kode Sumber 4.9.

```
CREATE TABLE kontainer (  
  id int(11) PRIMARY KEY AUTO_INCREMENT,  
  username VARCHAR(50)  
  ip VARCHAR(50)  
  port VARCHAR(50)  
  createdAt DATETIME  
);
```

**Kode Sumber 4.9:** *Query* untuk membuat tabel testing

## 4.5 Implementasi Pemasangan Kontainer *Docker* pada *Docker Host*

Setelah berhasil melakukan pemasangan *docker* versi 1.13.1 pada *docker host* dengan IP 10.151.36.134, sekarang lakukan konfigurasi supaya *docker* tidak hanya dapat digunakan oleh *root user* dari sebuah sistem. Hal ini dapat dilakukan dengan menjalankan perintah pada Kode Sumber 4.10.

```
sudo groupadd docker  
sudo usermod -aG docker $USER
```

**Kode Sumber 4.10:** Perintah untuk instalasi Ansible

### 4.5.1 Menambahkan dan Memperbarui Kontainer *Docker* yang Berisikan Mitmproxy

Setelah berhasil melakukan pemasangan *docker* pada *docker host* dan melakukan konfigurasi supaya *docker* tidak hanya dapat digunakan oleh *root user* dari sebuah sistem, selanjutnya dapat mencoba membuat sebuah kontainer *docker* yang berisi aplikasi

*mitmproxy*. Untuk membuat sebuah kontainer *docker* yang berisi *mitmproxy*, penulis melakukannya dengan sistem operasi Ubuntu dalam format *docker* yang disediakan oleh Docker Hub. Untuk melakukan unduh, jalankan perintah berikut pada Kode Sumber 4.11.

```
docker pull ubuntu
```

**Kode Sumber 4.11:** Perintah untuk *Pull* Ubuntu

Setelah berhasil diunduh, selanjutnya jalankan sistem operasi Ubuntu dengan menggunakan perintah yang tertera pada Kode Sumber 4.12.

```
docker run --name testmitmproxy --  
    privileged=True  
--network=host ubuntu
```

**Kode Sumber 4.12:** Perintah untuk Menjalankan *Image* Ubuntu

Parameter `--name` berguna untuk memberikan nama pada kontainer *docker* agar mudah dikenali dimana lokasi aplikasi saat dijalankan. Pada kasus ini kontainer *docker* diberi nama dengan *testmitmproxy*. Parameter `--privileged=True` berguna untuk memberikan kendali hak akses penuh kepada kontainer *docker* tersebut, sama seperti dengan *root user*. Parameter `--network=host` berguna untuk mendefinisikan jaringan yang akan digunakan oleh kontainer *docker* tersebut. Setelah menjalankannya, kontainer *docker* yang terbentuk dapat digunakan lebih lanjut, misalnya dengan mengubah data yang ada didalamnya, menambahkan fitur baru, atau hanya sekedar mengganti nama dari aplikasi.

Dalam kasus ini penulis menambahkan fitur baru, yaitu menambah *mitmproxy*. Untuk menambah atau memasang



*mitmproxy* pada kontainer *docker* yang baru saja dibuat, jalankan perintah berikut pada Kode Sumber 4.13.

```
sudo apt-get update
sudo apt-get install python3 python3-dev
python3-pip
sudo pip3 install cryptography
sudo pip3 install mitmproxy
```

**Kode Sumber 4.13:** Perintah untuk Pemasangan *Mitmproxy*

*Mitmproxy* versi 3.0.4 membutuhkan *Python* minimal versi 3.5, maka dari itu penulis memasang *Python* versi 3.5.2. *Mitmproxy* juga membutuhkan modul *cryptography* yang berguna untuk melakukan enkripsi maupun dekripsi ketika *mitmproxy* sedang berjalan. Lalu aktifkan *ipv4.forwarding* dengan menjalankan perintah pada Kode Sumber 4.14.

```
sudo sysctl -w net.ipv4.ip_forward=1
```

**Kode Sumber 4.14:** Perintah untuk Mengaktifkan *ipv4.forwarding*

Setelah berhasil melakukan pemasangan *mitmproxy* pada kontainer *docker*, jika ingin membuat *images* baru dari kontainer *docker* tersebut, maka hal pertama yang harus dilakukan adalah menghentikan kontainer *docker* yang sedang berjalan dengan menggunakan perintah seperti pada Kode Sumber 4.15.

```
docker stop [nama_container]
```

**Kode Sumber 4.15:** Perintah untuk Menghentikan Kontainer *Docker*

Nama *container* ini tergantung dari nama kontainer *docker* yang sudah dibuat. Untuk kasus yang digunakan oleh penulis, penulis menggunakan perintah `docker stop testmitmproxy`. Setelah

itu lakukan *commit* dengan menjalankan perintah seperti pada Kode Sumber 4.16.

```
docker commit [nama_container] [
    nama_repository]
```

**Kode Sumber 4.16:** Perintah untuk *Commit Kontainer Docker*

Nama *container* ini tergantung dari nama kontainer *docker* yang sudah dibuat. Sedangkan nama *repository* ini tergantung dari nama *repository* yang telah dibuat di Docker Hub. Untuk kasus yang digunakan oleh penulis, penulis menggunakan perintah `docker commit testmitmproxy fourirakbar/mitmproxy-oing:version1`. Pada bagian nama *repository* ini memiliki tiga bagian dengan pola seperti `[URL]/[nama]:[versi]`. Artinya membuat *image* dengan URL *repository* pada Docker Hub dengan nama `fourirakbar`. Kemudian nama dari *image*-nya sendiri adalah `mitmproxy-oing` dan versinya adalah `version1`. Setelah melakukan *commit*, maka *image* baru akan terbentuk. Langkah terakhir adalah melakukan *push image* ke Docker Hub dengan menggunakan perintah seperti Kode Sumber 4.17.

```
docker push [nama_container] [
    nama_repository]
```

**Kode Sumber 4.17:** Perintah untuk *Push Image* ke Docker Hub

#### 4.5.2 Menggunakan *Image Kontainer Docker* yang Sudah Dibuat

Setelah berhasil menambahkan dan memperbarui kontainer *docker* yang berisikan *mitmproxy*, penulis tidak perlu melakukannya lagi. Penulis hanya perlu memanggil kontainer

*docker* dengan menjalankan perintah pada Kode Sumber 4.18.

```
docker pull fourirakbar/mitmproxy-oing:
version1
```

**Kode Sumber 4.18:** Perintah untuk *Pull Image mitmproxy*

Lalu untuk menjalankan kontainer *docker* yang sudah di *pull*, jalankan perintah pada Kode Sumber 4.19.

```
docker run --name [IP_CLEINT] --privileged=
True --network=host fourirakbar /
mitmproxy-oing:version1
```

**Kode Sumber 4.19:** Perintah untuk *Pull Image mitmproxy*

#### 4.6 Implementasi Pembuatan Aturan untuk Mengarahkan *Traffic Client* ke Kontainer *Docker* dari Tiap-Tiap *Client*

Pada implementasi pembuatan aturan untuk mengarahkan *traffic client* ke kontainer *docker* dari tiap-tiap client dibuat ketika terdapat *client* yang telah berhasil *login* ke dalam sistem. Setelah *client* berhasil *login* ke dalam sistem, maka akan dibuatkan beberapa *rules* dengan menggunakan *iptables* seperti Kode Sumber 4.20.

```
iptables -I FORWARD 1 -s [IP_CLIENT] -j
ACCEPT
iptables -t nat -I PREROUTING 1 -s [
IP_CLIENT] -p tcp --dport 80 -j REDIRECT
--to-ports [PORTS_CLIENT]
iptables -t nat -I PREROUTING 1 -s [
IP_CLIENT] -p tcp --dport 443 -j
REDIRECT --to-ports [PORTS_CLIENT]
```

```
iptables -t nat -I POSTROUTING 1 -o wlp3s0
-j MASQUERADE -s [IP_CLIENT]
```

**Kode Sumber 4.20:** Command untuk mengarahkan *client* ke halaman *login*

Pada *rules* pertama berfungsi untuk mengizinkan atau memperbolehkan *traffic* dari *client* melewati *router*. Lalu *rules* kedua dan ketiga berfungsi untuk mengarahkan *traffic client* ke kontainer *docker* yang sudah dibuat dengan satu port khusus untuk *client* tersebut. Lalu *rules* keempat berfungsi untuk mengizinkan atau memperbolehkan *client* untuk mengakses internet.

## 4.7 Implementasi Pembuatan Halaman *Administrator*

Halaman *administrator* dibangun pada *Docker Host* dengan IP 10.151.36.134 dengan port 5001. Fungsi dari halaman *administrator* adalah untuk melihat siapa saja *client* yang berhasil *login* ke dalam sistem dan megakses internet, juga untuk melihat rekap *client* yang telah berhasil *login* dan mengakses internet. Pada sub-bab ini akan dibagi lagi menjadi beberapa bagian, antara lain rute *web service* pada halaman *administrator*, implementasi pembacaan *log file* dari *client* dan implementasi antarmuka.

### 4.7.1 Rute *Web Service* pada Halaman *Administrator*

Pada halaman *administrator* diperlukan adanya rute-rute yang bisa diakses untuk melayani permintaan dari *user* yang sedang membuka halaman *administrator*, yaitu untuk melihat *log* dari *client*. Daftar rute yang disediakan tertera pada Tabel 4.3.

**Tabel 4.3:** Daftar Rute *Web Service* pada Halaman *Administrator*

<b>HTTP Method</b>	<b>Rute</b>	<b>Parameter</b>	<b>Deskripsi</b>
GET	/table	-	Berfungsi untuk menampilkan halaman <i>dashboard</i> yang menunjukkan siapa saja <i>client</i> yang telah berhasil <i>login</i> ke dalam sistem dan sedang aktif mengakses internet.
GET	/stream/	id, ip, user, port	Berfungsi untuk melihat <i>log</i> dari <i>client</i> secara langsung atau <i>live</i> dengan memasukkan parameter berupa id, ip, user, dan port.
GET	/lihat/	id, ip, user, port	Berfungsi untuk melihat <i>log</i> dari <i>client</i> sampai dengan terakhir <i>client</i> mengakses internet dengan memasukkan parameter berupa id, ip, user, dan port.

**Tabel 4.3:** Daftar Rute *Web Service*

HTTP Method	Rute	Parameter	Deskripsi
GET	/history	-	Berfungsi untuk menampilkan rekap atau daftar <i>client</i> yang telah berhasil <i>login</i> ke dalam sistem.

#### 4.7.2 Implementasi Pembacaan *Log File* dari *Client*

Pada implementasi pembacaan *log file* dari *client* dilakukan dengan membaca *file* hasil *output* dari *mitmproxy*. *File* hasil *output* dari *mitmproxy* berbentuk *binary* dimana yang bisa membacanya hanya komputer saja. Maka dari itu perlu dilakukan pembacaan lagi *file* yang berisi *binary* tersebut dengan menjalankan *command* seperti pada Kode Sumber 4.21.

```
mitmdump -nr [NAMA-FILE] --set flow_detail
=2 --showhost > [NAMA-FILE]
```

**Kode Sumber 4.21:** Perintah untuk Membaca *File Log* dari *Mitmproxy*

Sedangkan untuk membaca *log file* dari *client* secara langsung atau *live* dapat dilakukan dengan menjalankan *command* seperti pada Kode Sumber 4.22.

```
tail -f -c +0 [NAMA_FILE] | mitmdump -n -r
- --set flow_detail=1 --showhost
```

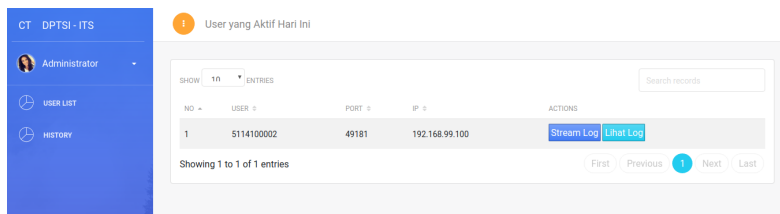
**Kode Sumber 4.22:** Perintah untuk Membaca *File Log* dari *Client*

Parameter *-nr* berfungsi untuk tidak menjalankan *proxy server* dari *mitmproxy* sendiri, dan juga berfungsi untuk melakukan analisa dari *file output mitmproxy* yang berbentuk

*binary*. Sedangkan parameter `--set flow detail` berfungsi untuk menampilkan detail dari analisa yang dilakukan oleh *mitmproxy*. Terdapat tingkat satu sampai dengan tiga, semakin tinggi tingkat yang diberikan maka semakin jelas detail dari *log* yang dianalisa oleh *mitmproxy*. Lalu parameter `--showhost` berfungsi untuk menampilkan *header* dari URL yang telah diakses oleh *client*.

### 4.7.3 Implementasi Antarmuka Halaman *Administrator*

Halaman *administrator* diperuntukkan bagi *User* yang mempunyai akses ke *Docker Host*. Halaman ini berguna sebagai *dashboard* dari *administrator*. Pada halaman ini menampilkan siapa saja *Client* yang telah berhasil *login* ke dalam sistem. Terdapat dua *button*, yaitu *Stream Log* yang berfungsi untuk melihat secara langsung atau *live log* dari *client*. Lalu terdapat *button* Lihat Log yang berfungsi untuk melihat *log* dari *client* sampai terakhir *client* tersebut mengakses internet.



**Gambar 4.2:** Halaman *Administrator* Menu *User List*

Pada bagian *sidebar* terdapat dua menu yaitu *User List* dan *History*. Menu *User List* berguna untuk melihat *client* yang telah berhasil *login* ke dalam sistem. Sedangkan menu *history* berguna untuk melihat rekap *client* yang telah berhasil *login* dan mengakses internet pada hari-hari sebelumnya. Implementasi antarmuka halaman *administrator* pada menu *User List* dapat

dilihat pada Gambar 4.2. Dan implementasi antarmuka halaman *administrator* oada nenu *History* dapat dilihat pada Gambar 4.3

CT DPTSI - ITS

Administrator

USER LIST

HISTORY

Rekap User

SHOW 10 ENTRIES

Search records

NO	USER	PORT	IP	TANGGAL	ACTIONS
1	5114100001	49180	192.168.99.100	27-05-2018	<a href="#">Lihat Log</a>
2	5114100002	49181	192.168.99.100	28-05-2018	<a href="#">Lihat Log</a>

Showing 1 to 2 of 2 entries

First Previous 1 Next Last

**Gambar 4.3:** Halaman *Administrator* Menu *History*



## BAB V

### PENGUJIAN DAN EVALUASI

#### 5.1 Lingkungan Uji Coba

Lingkungan pengujian menggunakan komponen-komponen yang terdiri dari: satu *server load balancer*, satu *server master host*, satu *server controller*, satu *server docker registry*, dan enam komputer penguji. Semua *server* menggunakan Virtual Private Server dari DigitalOcean. Lalu, untuk komputer penguji menggunakan lima buah desktop dan satu buah VPS sebagai *docker* klien yang digunakan untuk membuat *docker image*. Pengujian dilakukan di Laboratorium Pemrograman Jurusan Teknik Informatika ITS.

Spesifikasi untuk setiap komponen yang digunakan ditunjukkan pada Tabel 5.1.

**Tabel 5.1:** Spesifikasi Komponen

No	Komponen	Perangkat Keras	Perangkat Lunak
1	Load balancer	2 core processor, 4GB RAM, 20GB SSD	Ubuntu 14.04.5 LTS, HAProxy, Python 2.7
2	Master host	8 core processor, 16GB RAM, 20GB SSD	Ubuntu 14.04.5 LTS, Docker 17.03.0-ce, Python 2.7
3	Controller	2 core processor, 4GB RAM, 20GB SSD	Ubuntu 14.04.5 LTS, Redis, MySQL, Python 2.7
4	Docker registry	1 core processor, 512MB RAM, 20GB SSD	Ubuntu 14.04.5 LTS, Docker 17.03.0-ce, Python 2.7
5	Komputer penguji	Processor Core2Duo E7300, 2GB RAM	Windows 8, JMeter 3.2

**Tabel 5.1:** Spesifikasi Komponen

No	Komponen	Perangkat Keras	Perangkat Lunak
6	Docker klien	1 core processor, 1GB RAM, 20GB SSD	Ubuntu 16.04 LTS, Docker 17.03.0-ce

Untuk akses ke masing-masing komponen, digunakan IP publik yang disediakan untuk masing-masing komponen tersebut. Selain menggunakan IP, ada sebagian *server* yang bisa diakses melalui domain. Detailnya ditunjukkan pada Tabel 5.2.

**Tabel 5.2:** IP dan Domain Server

No	Server	IP dan Domain
1	Load balancer	128.199.160.188
2	Master host	128.199.182.29
3	Controller	128.199.250.137 <a href="http://controller.nota-no.life">http://controller.nota-no.life</a>
4	Docker registry	139.59.97.244 <a href="https://registry.nota-no.life">https://registry.nota-no.life</a>

## 5.2 Skenario Uji Coba

Uji coba akan dilakukan untuk mengetahui keberhasilan sistem yang telah dibangun. Skenario pengujian dibedakan menjadi 2 bagian, yaitu:

- **Uji Fungsionalitas**

Pengujian ini didasarkan pada fungsionalitas yang disajikan sistem.

- **Uji Performa**

Pengujian ini untuk menguji ketahanan sistem terhadap sejumlah permintaan ke aplikasi secara bersamaan. Pengujian dilakukan dengan melakukan *benchmark* pada

sistem.

## 5.2.1 Skenario Uji Coba Fungsionalitas

Uji fungsionalitas dibagi menjadi 2, yaitu uji mengelola aplikasi berbasis *docker* dan uji fungsionalitas menu aplikasi Dasbor.

### 5.2.1.1 Uji Mengelola Aplikasi Berbasis Docker

Pengujian ini dilakukan untuk mengetahui apakah sistem sudah bisa menyimpan dan mengelola data *docker image* dari aplikasi yang dimasukkan oleh pengembang. Pengujian menggunakan VPS yang berperan sebagai *docker* klien. Pengujian dilakukan dengan memasukkan data *docker image* ke *server docker registry* yang sudah disediakan. Dengan menggunakan sebuah komputer lain yang digunakan untuk membuat aplikasi web berbasis *docker*, dari sana aplikasi tersebut akan ditaruh ke *server docker registry*.

Alamat dari *Docker registry* yang digunakan adalah <https://registry.nota-no.life>. Setelah berhasil melakukan login pada *server docker registry*, selanjutnya adalah memasukkan *image* baru tersebut. *Image* yang dibuat adalah sebuah aplikasi web berbasis PHP 7 dengan *web server* Apache. Aplikasi web tersebut menyediakan sebuah halaman yang berisi sebuah teks yang dibuat melalui pemanggilan fungsi PHP.

Pertama kali *image* tersebut dimasukkan ke *server docker registry*, kemudian data dari *image* tersebut akan disimpan di *server controller*. Setelah data tersimpan, selanjutnya adalah menjalankan aplikasi melalui dasbor yang disediakan. Sebelum menjalankan aplikasi, terlebih dahulu mengatur *port* dari aplikasi yang akan berjalan. Setelah mengatur *port* dengan benar, selanjutnya adalah menjalankan aplikasinya. Jika aplikasi berhasil dijalankan, maka aplikasi dapat diakses melalui domain

yang disediakan.

Setelah aplikasi berjalan, selanjutnya adalah memperbarui aplikasi dengan mengganti teks yang ditampilkan. Untuk itu, pada komputer yang digunakan untuk membuat *image* sebelumnya, maka dibuatkan *image* baru dari aplikasi dengan versi terbaru. *Image* versi terbaru ini kemudian di *push* ke *docker registry*. Setelah selesai melakukan *push*, harapannya aplikasi yang sebelumnya sudah berjalan, akan diperbarui secara otomatis oleh sistem.

Terakhir, pengujian yang dilakukan adalah menghentikan aplikasi yang sudah berjalan. Fungsi untuk menghentikan aplikasi yang sedang berjalan ini terdapat pada dasbor. Jika proses ini berhasil, maka domain yang sebelumnya digunakan untuk mengakses aplikasi akan hilang dan pengguna tidak bisa lagi melakukan akses terhadap aplikasi.

Daftar uji fungsionalitas menambahkan dan memperbarui aplikasi dijelaskan pada Tabel 5.3.

**Tabel 5.3:** Skenario Uji Mengelola Aplikasi Berbasis Docker

No	Uji Coba	Hasil Harapan
1	Pengguna melakukan <i>login</i> ke <i>server docker registry</i>	Pengguna berhasil melakukan <i>login</i> dengan menggunakan <i>username</i> dan <i>password</i> yang sudah ditentukan.
2	Pengguna menambahkan <i>image</i> baru dari sebuah aplikasi ke <i>server docker registry</i> .	Pengguna berhasil menambahkan <i>image</i> baru dan data tersimpan pada <i>server controller</i> .

**Tabel 5.3:** Skenario Uji Mengelola Aplikasi Berbasis Docker

No	Uji Coba	Hasil Harapan
3	Pengguna bisa mengatur <i>port</i> dari aplikasi menggunakan dasbor yang disediakan	Data <i>port</i> dari aplikasi yang tersimpan bisa diganti sesuai dengan kebutuhan pengguna.
4	Pengguna bisa menjalankan aplikasi melalui fitur yang ada pada dasbor	Aplikasi berhasil berjalan dan pengguna mendapatkan domain yang digunakan untuk mengakses aplikasi.
5	Pengguna memperbarui aplikasi yang sedang berjalan dengan melakukan <i>push</i> ke <i>server docker registry</i> .	Aplikasi yang sedang berjalan akan diperbarui secara otomatis tanpa perlu perintah dari pengguna.
6	Pengguna menghentikan aplikasi yang sedang berjalan.	Aplikasi berhasil dihentikan dan pengguna tidak bisa lagi melakukan akses aplikasi.

#### 5.2.1.2 Uji Fungsionalitas Menu Aplikasi Dasbor

Aplikasi Dasbor digunakan untuk mengelola dan memantau aplikasi. Aplikasi Dasbor terdiri dari 4 bagian utama, yaitu halaman beranda, informasi aplikasi, informasi *container*, dan metrik dari aplikasi. Rancangan pengujian dan hasil yang diharapkan ditunjukkan dengan Tabel 5.4.

**Tabel 5.4:** Skenario Uji Fungsionalitas Aplikasi Dasbor

<b>No</b>	<b>Menu</b>	<b>Uji Coba</b>	<b>Hasil Harapan</b>
1	Kelola aplikasi	Menambahkan aplikasi baru atau memperbarui aplikasi	Dasbor dapat menampilkan daftar aplikasi terbaru yang dimasukkan atau diperbarui oleh pengembang.
		Menjalankan aplikasi yang sudah masuk ke dalam sistem	Aplikasi dapat berjalan dan pengguna mendapatkan domain untuk mengakses aplikasi.
		Menghentikan aplikasi yang sedang berjalan	Aplikasi yang sedang berjalan dapat dihentikan dan pengguna tidak bisa lagi melakukan akses terhadap aplikasi.
		Mengganti <i>port</i> aplikasi agar dapat berjalan dengan baik	Pengguna dapat mengganti <i>port</i> aplikasi agar aplikasi dapat berjalan dengan benar.

**Tabel 5.4:** Skenario Uji Fungsionalitas Aplikasi Dasbor

No	Menu	Uji Coba	Hasil Harapan
2	Lihat informasi aplikasi	Memilih salah satu aplikasi yang ada	Pengguna dapat melihat informasi secara lengkap tentang aplikasi.
3	Lihat informasi <i>container</i>	Memilih salah satu aplikasi yang ada	Pengguna dapat melihat informasi secara lengkap tentang <i>container</i> yang sedang berjalan untuk aplikasi tersebut.
4	Lihat metrik aplikasi	Memilih salah satu aplikasi yang ada	Pengguna dapat melihat grafik penggunaan CPU dan <i>memory</i> dari aplikasi .

### 5.2.2 Skenario Uji Coba Performa

Uji performa dilakukan dengan menggunakan lima buah desktop untuk melakukan akses secara bersamaan ke aplikasi menggunakan aplikasi JMeter. Desktop akan mencoba mengakses halaman dari aplikasi web yang sudah berjalan, dengan domain aplikasi.nota-no.life. Halaman yang akan diakses berisi sebuah teks yang dihasilkan dari pemanggilan fungsi PHP.

Percobaan dilakukan dengan lima skenario jumlah *concurrent user* yang berbeda, yaitu sebanyak 800, 1600, 2400, 3200, dan 4000 pengguna dalam rentang waktu inisialisasi  $\pm 15$  detik. Waktu tersebut menunjukkan masing-masing pengguna akan mengirimkan request selama  $\pm 15$  detik, namun tidak termasuk waktu menunggu balasan dari *server*, yang artinya

keseluruhan permintaan tersebut akan lebih dari waktu tersebut dan bergantung pada kemampuan *server* untuk memberikan respon. Pengujian *request* ini bertujuan untuk mengukur kemampuan dari *proactive model*. Untuk masing-masingnya, dicoba sebanyak empat perhitungan *proactive model* yang berbeda menggunakan ARIMA yang berbeda, yaitu ARIMA(1,1,0), ARIMA(2,1,0), ARIMA(3,1,0), ARIMA(4,1,0). *Proactive model* sendiri berguna untuk mengetahui jumlah *request* kedepannya agar sistem bisa menyediakan sumber daya berdasarkan prediksi tersebut.

Selain itu, untuk memperkirakan sumber daya yang dibutuhkan sistem kedepannya, digunakan *reactive model*. *Model* tersebut akan menghitung jumlah *container* yang sumber daya CPU dan *memory*-nya sudah melebihi batas yang ditentukan. Sistem akan membentuk *container* baru berdasarkan perhitungan *reactive model* tersebut jika ada *container* yang penggunaannya sudah melebihi batas atas dan mengurangi *container* jika ada *container* yang tidak digunakan. Percobaan akan dilakukan sebanyak enam kali dan berikutnya akan dijelaskan data apa yang diuji untuk masing-masingnya.

#### **5.2.2.1 Uji Performa Kecepatan Menangani Request**

Pengujian dilakukan dengan mengukur jumlah waktu yang diperlukan untuk menyelesaikan *request* yang dilakukan oleh komputer penguji. Waktu yang diukur adalah perbedaan jarak antara *request* pertama dan yang terakhir dilakukan oleh klien yang mendapatkan balasan dari *server*.

#### **5.2.2.2 Uji Performa Penggunaan CPU**

Pengujian dilakukan dengan menghitung penggunaan CPU yang terjadi pada *server master host*. Penggunaan CPU di sini



adalah penggunaan dari *container* aplikasi yang sedang berjalan. Perhitungan dilakukan dengan mengambil nilai rata-rata penggunaan CPU dari masing-masing *container* selama proses pengujian dilakukan. Nilai yang didapatkan berupa total persen penggunaan CPU oleh *container* dibandingkan dengan keseluruhan kemampuan CPU.

#### **5.2.2.3 Uji Performa Penggunaan *Memory***

Pengujian dilakukan dengan menghitung penggunaan *memory* yang terjadi pada *server master host*. Penggunaan *memory* di sini adalah penggunaan dari *container* aplikasi yang sedang berjalan. Perhitungan dilakukan dengan mengambil nilai rata-rata penggunaan *memory* dari masing-masing aplikasi selama proses pengujian dilakukan.

#### **5.2.2.4 Uji Performa Keberhasilan *Request***

Pengujian dilakukan dengan menghitung jumlah *request* yang gagal dilakukan selama skenario dijalankan. Dari semua jumlah *request* yang dikirimkan selama pengujian, akan didapatkan persen *request* yang gagal dilakukan.

### **5.3 Hasil Uji Coba dan Evaluasi**

Berikut dijelaskan hasil uji coba dan evaluasi berdasarkan skenario yang telah dijelaskan pada subbab 5.2.

#### **5.3.1 Uji Fungsionalitas**

Berikut dijelaskan hasil pengujian fungsionalitas pada sistem yang dibangun.

### 5.3.1.1 Uji Mengelola Aplikasi Berbasis Docker

Pengujian dilakukan sesuai dengan skenario yang dijelaskan pada subbab 5.2.1.1 dan pada Tabel 5.3. Hasil pengujian seperti tertera pada Tabel 5.5.

**Tabel 5.5:** Hasil Uji Coba Mengelola Aplikasi Berbasis Docker

No	Uji Coba	Hasil
1	Pengguna melakukan <i>login</i> ke <i>server docker registry</i>	OK.
2	Pengguna menambahkan <i>image</i> baru dari sebuah aplikasi ke <i>server docker registry</i> .	OK.
3	Pengguna bisa mengatur <i>port</i> dari aplikasi menggunakan dasbor yang disediakan	OK.
4	Pengguna bisa menjalankan aplikasi melalui fitur yang ada pada dasbor.	OK.
5	Pengguna memperbarui aplikasi yang sedang berjalan dengan melakukan <i>push</i> ke <i>server docker registry</i> .	OK.
6	Pengguna menghentikan aplikasi yang sedang berjalan.	OK.

Sesuai dengan skenario uji coba yang diberikan pada Tabel 5.3, hasil uji coba menunjukkan semua skenario berhasil ditangani.

### 5.3.1.2 Uji Fungsionalitas Menu Aplikasi Dasbor

Sesuai dengan skenario pengujian yang dilakukan pada aplikasi dasbor. Pengujian dilakukan dengan menguji setiap

menu pada aplikasi dasbor. Hasil uji coba dapat dilihat pada Table 5.6. Semua skenario yang direncanakan berhasil ditangani.

**Tabel 5.6:** Hasil Uji Fungsionalitas Aplikasi Dasbor

No	Menu	Uji Coba	Hasil
1	Kelola aplikasi	Menambahkan aplikasi baru atau memperbarui aplikasi	Dasbor berhasil menampilkan daftar aplikasi terbaru yang dimasukkan atau diperbarui oleh pengembang.
		Menjalankan aplikasi yang sudah masuk ke dalam sistem	Aplikasi berhasil berjalan dan pengguna mendapatkan domain untuk mengakses aplikasi.
		Menghentikan aplikasi yang sedang berjalan	Aplikasi yang sedang berjalan berhasil dihentikan dan pengguna tidak bisa lagi melakukan akses terhadap aplikasi.

**Tabel 5.6:** Hasil Uji Fungsionalitas Aplikasi Dasbor

No	Menu	Uji Coba	Hasil
		Mengganti <i>port</i> aplikasi agar dapat berjalan dengan baik	Pengguna berhasil mengganti <i>port</i> aplikasi agar aplikasi dapat berjalan dengan benar.
2	Lihat informasi aplikasi	Memilih salah satu aplikasi yang ada	Pengguna berhasil melihat informasi secara lengkap tentang aplikasi.
3	Lihat informasi <i>container</i>	Memilih salah satu aplikasi yang ada	Pengguna berhasil melihat informasi secara lengkap tentang <i>container</i> yang sedang berjalan untuk aplikasi tersebut.
4	Lihat metrik aplikasi	Memilih salah satu aplikasi yang ada	Pengguna berhasil melihat grafik penggunaan CPU dan <i>memory</i> dari aplikasi .

### 5.3.2 Hasil Uji Performa

Seperti yang sudah dijelaskan pada subbab 5.2 pengujian performa dilakukan dengan melakukan akses ke aplikasi dengan sejumlah pengguna secara bersama-sama. Pengujian dilakukan dengan memberikan *request* secara berkelanjutan dengan jumlah pengguna terdiri dari lima bagian, yaitu 800, 1600, 2400, 3200, dan 4000 pengguna. Untuk jumlah *request* yang dihasilkan dari masing-masing pengguna selama rentang waktu request  $\pm 15$  detik dapat dilihat pada Tabel 5.7. Jumlah tersebut akan diolah oleh *reactive model*. Lalu jumlah penggunaan CPU dan memory selama menangani *request* tersebut akan digunakan oleh *proactive model* untuk menambahkan atau mengurangi *container* yang ada.

**Tabel 5.7:** Jumlah *Request* ke Aplikasi

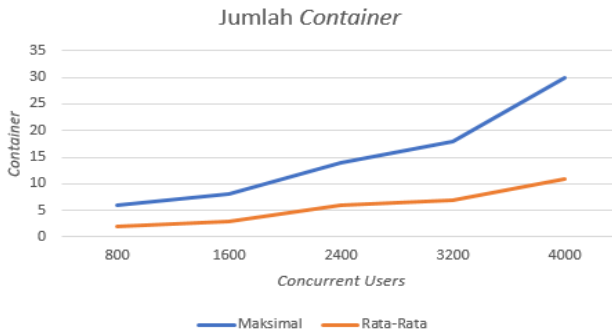
<b><i>Concurrent Users</i></b>	<b><i>Jumlah Request</i></b>
800	$\pm 16.925$
1.600	$\pm 26.650$
2.400	$\pm 34.943$
3.200	$\pm 50.092$
4.000	$\pm 57.750$

Pada Tabel 5.8 dapat dilihat jumlah *container* yang terbentuk selama proses *request* dari *user* yang dilakukan selama enam kali. Nilai yang ditampilkan berupa nilai rata-rata selama percobaan dibulatkan ke atas. Sistem dapat menyediakan *container* sesuai dengan jumlah *request* yang diberikan, semakin banyak *request* yang dilakukan, maka *container* yang disediakan akan semakin banyak. Nilai *container* tersebut didapatkan dari perhitungan *proactive model*. Selain melihat jumlah *request*, penentuan *container* yang dibentuk juga dari jumlah sumber daya yang digunakan *container* berdasarkan perhitungan

menggunakan *reactive model*. Pada Gambar 5.1 dapat dilihat grafik dari jumlah *container* yang terbentuk berdasarkan jumlah *request* yang dilakukan.

**Tabel 5.8:** Jumlah *Container*

<i>Concurrent Users</i>	Maksimal <i>Container</i>	Rata-rata <i>Container</i>
800	6	2
1.600	8	3
2.400	14	6
3.200	18	7
4.000	30	11



**Gambar 5.1:** Grafik Jumlah *Container*

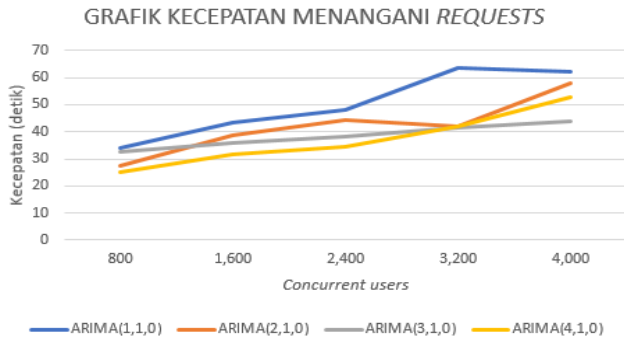
### 5.3.2.1 Kecepatan Menangani *Request*

Dari hasil uji coba kecepatan menangani *request*, dapat dilihat pada Table 5.9 dalam satuan detik bahwa semakin banyak *concurrent users*, semakin lama pula waktu yang diperlukan untuk menyelesaikannya. Request paling cepat ditangani dengan menggunakan prediksi ARIMA(4,1,0) dan paling lambat

menggunakan ARIMA(1,1,0). Hal tersebut terjadi karena kurang bagusnya hasil prediksi yang dihasilkan oleh ARIMA(1,1,0) yang mana kadang hasil prediksinya terlalu rendah atau terlalu tinggi. Dari hasil percobaan tersebut, dapat dilihat bahwa hampir semua *request* dapat ditangani di bawah satu menit. Lalu grafik hasil uji coba perhitungan kecepatan menangani *request* ditunjukkan pada Gambar 5.2.

**Tabel 5.9:** Kecepatan Menangani *Request*

	800	1600	2400	3200	4000
ARIMA(1,1,0)	34.167	43.286	48.143	63.857	62.286
ARIMA(2,1,0)	27.429	38.571	44.143	42.143	57.857
ARIMA(3,1,0)	32.429	36.000	38.429	41.571	43.857
ARIMA(4,1,0)	24.857	31.571	34.429	42.143	52.714



**Gambar 5.2:** Grafik Kecepatan Menangani *Request*

### 5.3.2.2 Penggunaan CPU

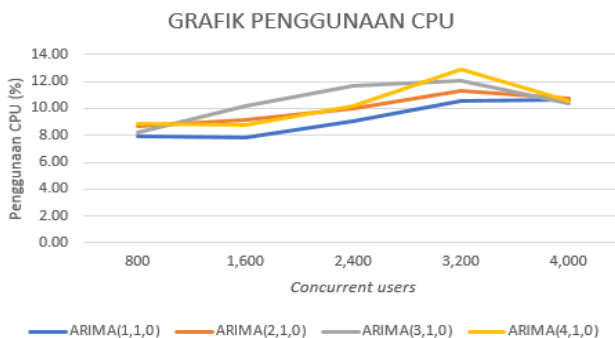
Dari hasil uji coba penggunaan CPU pada *server master host*, penggunaan CPU berada di bawah 15%. Penggunaan CPU yang diukur adalah penggunaan CPU yang dilakukan oleh *container*

dari aplikasi, tidak termasuk sistem. Jumlah *core* yang dimiliki oleh *processor* di *server master host* adalah 8 buah, yang artinya kurang lebih hanya satu *core* yang digunakan untuk menangani semua *request*. Hasil pengukuran penggunaan CPU dapat dilihat pada Tabel 5.10

**Tabel 5.10:** Penggunaan CPU

	800	1600	2400	3200	4000
ARIMA(1,1,0)	7.1%	7.8%	9.1%	10.5%	10.7%
ARIMA(2,1,0)	8.5%	9.2%	10.1%	11.3%	10.7%
ARIMA(3,1,0)	8.8%	10.2%	11.6%	12.1%	10.3%
ARIMA(4,1,0)	8.0%	8.3%	10.1%	12.9%	10.5%

Dari hasil uji coba, penggunaan prediksi yang berbeda tidak terlalu berpengaruh terhadap penggunaan CPU. Lalu, penggunaan CPU tergolong rendah, yaitu hanya sebesar  $\pm 10\%$  untuk menangani semua *request* yang diberikan. Hasil uji coba performa penggunaan CPU ditunjukkan oleh dalam grafik pada Gambar 5.3.



**Gambar 5.3:** Grafik Penggunaan CPU



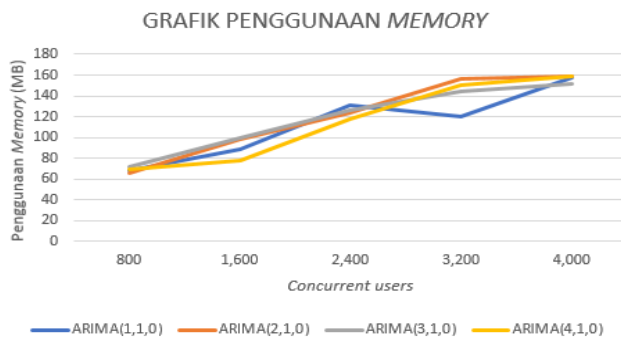
### 5.3.2.3 Penggunaan *Memory*

Dari hasil uji coba penggunaan *memory*, semakin banyak *request* yang diterima, semakin banyak *memory* yang diperlukan. Perhitungan penggunaan *memory* adalah rata-rata penggunaan dari masing-masing *container* sebuah aplikasi. Untuk masing-masing *container*, dibatasi penggunaan maksimal *memory* adalah 512 MB. Dari hasil uji coba ini, dapat dilihat pada Tabel 5.11 bahwa penggunaan terbesar hanya sebesar 158.71 MB. Artinya jumlah tersebut hanya menggunakan sepertiga dari keseluruhan *memory* yang bisa digunakan.

**Tabel 5.11:** Penggunaan *Memory*

	800	1600	2400	3200	4000
ARIMA(1,1,0)	67.91	88.97	130.79	120.14	157.73
ARIMA(2,1,0)	65.89	97.98	123.47	156.64	158.33
ARIMA(3,1,0)	72.20	99.72	125.56	144.42	152.14
ARIMA(4,1,0)	69.60	77.34	117.39	149.76	158.71

Hasil uji coba performa penggunaan *memory* dalam grafik ditunjukkan pada Gambar 5.4.



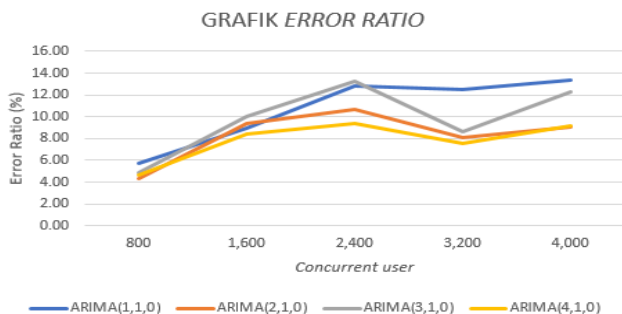
**Gambar 5.4:** Grafik Penggunaan Memory

### 5.3.2.4 Keberhasilan *Request*

Pada uji coba ini, dilakukan perhitungan seberapa besar jumlah *request* yang gagal dilakukan. Untuk jumlah *concurrent user* pada tingkat 800 dan 1600, dapat dilihat pada Table 5.12 *error* yang terjadi hampir sama. Prediksi menggunakan ARIMA(4,1,0) berhasil unggul karena menggunakan parameter yang lebih banyak. Namun hal tersebut tidak berlaku untuk ARIMA(3,1,0) karena walaupun parameternya lebih banyak dari ARIMA(2,1,0), tapi hasil prediksinya bisa meleset saat terjadi kondisi dimana koefisien negatif atau koefisien ke dua dikalikan dengan sebuah parameter bukan nol, dan koefisien lain dikalikan dengan parameter nol, maka hasil prediksinya akan negatif, yang mana seharusnya tidak mungkin ada *request* negatif.

**Tabel 5.12:** *Error Ratio Request*

	800	1600	2400	3200	4000
ARIMA(1,1,0)	5.72%	8.96%	12.85%	12.54%	13.38%
ARIMA(2,1,0)	4.31%	9.35%	10.68%	8.11%	9.04%
ARIMA(3,1,0)	4.84%	10.02%	13.22%	8.63%	12.24%
ARIMA(4,1,0)	4.62%	8.41%	9.39%	7.52%	9.21%



**Gambar 5.5:** Grafik Error Ratio

Dari uji coba itu, 90% lebih *request* berhasil ditangani. Hasil uji coba jumlah *request* yang gagal ditunjukkan dengan grafik pada Gambar 5.5.

*(Halaman ini sengaja dikosongkan)*

## BAB VI

### PENUTUP

Bab ini membahas kesimpulan yang dapat diambil dari tujuan pembuatan sistem dan hubungannya dengan hasil uji coba dan evaluasi yang telah dilakukan. Selain itu, terdapat beberapa saran yang bisa dijadikan acuan untuk melakukan pengembangan dan penelitian lebih lanjut.

#### 6.1 Kesimpulan

Dari proses perancangan, implementasi dan pengujian terhadap sistem, dapat diambil beberapa kesimpulan berikut:

1. Sistem dapat mengarahkan *client* ke halaman *login* dari sistem.
2. Sistem dapat membuat kontainer *docker* yang berisi *mitmproxy* secara otomatis ketika terdapat *client* yang berhasil *login* ke dalam sistem.
3. Sistem dapat mengarahkan *traffic* dari *client* ke kontainer *docker* yang sudah dibuat dan digunakan sebagai internet *access management* untuk *client* untuk memperbolehkan atau mengijinkan *client* tersebut untuk mengakses internet.

#### 6.2 Saran

Berikut beberapa saran yang diberikan untuk pengembangan lebih lanjut:

1. Sistem dapat dikembangkan dengan menggunakan *server* lebih dari satu untuk meringankan beban kerja dari *server* itu sendiri.
2. Sistem dapat dikembangkan dengan menentukan beban dari setiap *server*, dengan menambahkan kriteria-kriteria yang sesuai dengan lingkungan sistem yang ada, seperti jarak antara *docker host* dengan *middleware* atau

kecepatan bandwidth dari setiap *docker host* merupakan kriteria yang baik.

## DAFTAR PUSTAKA

- [1] “Welcome to Python.org,” 29 Mei 2018. [Daring]. Tersedia pada: <https://www.python.org/>. [Diakses: 29 Mei 2018].
- [2] “Welcome | Flask (A Python Microframework),” 29 Mei 2018. [Daring]. Tersedia pada: <http://flask.pocoo.org/>. [Diakses: 29 Mei 2018].

*(Halaman ini sengaja dikosongkan)*



## LAMPIRAN A

### INSTALASI PERANGKAT LUNAK

#### Instalasi Lingkungan Docker

Proses pemasangan Docker dapat dilakukan sesuai tahap berikut:

- Menambahkan repository Docker

Langkah ini dilakukan untuk menambahkan *repository* Docker ke dalam paket *apt* agar dapat di unduh oleh Ubuntu. Untuk melakukannya, jalankan perintah berikut:

```
sudo apt-get -y install \
    apt-transport-https \
    ca-certificates \
    curl

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

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

sudo apt-get update
```

- Mengunduh Docker

Docker dikembangkan dalam dua versi, yaitu CE (*Community Edition*) dan EE (*Enterprise Edition*). Dalam pengembangan sistem ini, digunakan Docker CE karena merupakan versi Docker yang gratis. Untuk mengunduh Docker CE, jalankan perintah `sudo apt-get -y install docker-ce`.

- Mencoba menjalankan Docker  
Untuk melakukan tes apakah Docker sudah terpasang dengan benar, gunakan perintah `sudo docker run hello-world`.

## Instalasi Docker Registry

Docker Registry dikembangkan menggunakan Docker Compose. Dengan menggunakan Docker Compose, proses pemasangan Docker Registry menjadi lebih mudah dan fleksibel untuk dikembangkan ditempat lain. Docker Registry akan dijalankan pada satu *container* dan Nginx juga akan dijalankan di satu *container* lain yang berfungsi sebagai perantara komunikasi antara Docker Registry dengan dunia luar. Berikut adalah proses pengembangan Docker Registry yang penulis lakukan:

- Pemasangan Docker Compose  

```
$ sudo apt-get -y install python-pip
```

```
$ sudo pip install docker-compose
```
- Pemasangan paket `apache2-utils`  
 Pada paket `apache2-utils` terdapat fungsi `htpasswd` yang digunakan untuk membuat *hash password* untuk Nginx. Proses pemasangan paket dapat dilakukan dengan menjalankan perintah `sudo apt-get -y install apache2-utils`.
- Pemasangan dan pengaturan Docker Registry  
 Buat folder `docker-registry` dan data dengan menjalankan perintah berikut:  

```
$ mkdir /docker-registry && cd $_
```

```
$ mkdir data
```

 Folder `data` digunakan untuk menyimpan data yang dihasilkan dan digunakan oleh *container* Docker Registry. Kemudian di dalam folder `docker-registry` buat sebuah berkas dengan nama `docker-compose.yml` yang akan

digunakan oleh Docker Compose untuk membangun aplikasi. Tambahkan isi berkasnya sesuai dengan Kode Sumber 1.1.

```
nginx:
image: "nginx:1.9"
ports:
  - 443:443
  - 80:80
links:
  - registry:registry
volumes:
  - ./nginx/:/etc/nginx/conf.d
registry:
image: registry:2
ports:
  - 127.0.0.1:5000:5000
environment:
  REGISTRY_STORAGE_FILESYSTEM
  _ROOTDIRECTORY: /data
volumes:
  - ./data:/data
  - ./registry/config.yml:/etc/docker
    /registry/config.yml
```

**Kode Sumber 1.1:** Isi Berkas docker-compose.yml

- Pemasangan *container* Nginx Buat folder nginx di dalam folder docker-registry. Di dalam folder nginx buat berkas dengan nama `registry.conf` yang berfungsi sebagai berkas konfigurasi yang akan digunakan oleh Nginx. Isi berkas sesuai dengan Kode Sumber 1.2.

```
upstream docker-registry {
  server registry:5000;
}
```

```

server{
    listen 80;
    server_name registry.nota-no.life;
    return 301 https://
        $server_name$request_uri;
}
server{
    listen 443;
    server_name registry.nota-no.life;
    ssl on;
    ssl_certificate /etc/nginx/conf.d/
        cert.pem;
    ssl_certificate_key /etc/nginx/conf.d
        /privkey.pem;
    client_max_body_size 0;
    chunked_transfer_encoding on;
    location /v2/{
        if ($http_user_agent ~ "(docker
            \/\1\.(3|4|5(?:!\.[0-9]-dev))|Go )
            .*$" ){
            return 404;
        }
        auth_basic "registry.localhost";
        auth_basic_user_file /etc/nginx/
            conf.d/registry.password;
        add_header 'Docker-Distribution-API
            -Version' 'registry/2.0' always;
        proxy_pass http://docker-registry;
        proxy_set_header Host $http_host;
        proxy_set_header X-Real-IP
            $remote_addr;
        proxy_set_header X-Forwarded-For
            $proxy_add_x_forwarded_for;
    }
}

```

```

        proxy_set_header X-Forwarded-Proto
            $scheme;
        proxy_read_timeout 900;
    }
}

```

**Kode Sumber 1.2:** Isi Berkas registry.conf

## Instalasi Pustaka Python

Dalam pengembangan sistem ini, digunakan berbagai pustaka pendukung. Pustaka pendukung yang digunakan merupakan pustaka untuk bahasa pemrograman Python. Berikut adalah daftar pustaka yang digunakan dan cara pemasangannya:

- Python Dev  
\$ sudo apt-get install python-dev
- Flask  
\$ sudo pip install Flask
- docker-py  
\$ sudo pip install docker
- MySQLd  
\$ sudo apt-get install python-mysqldb
- Redis  
\$ sudo pip install redis
- RQ  
\$ sudo pip install rq

## Instalasi HAProxy

HAProxy dapat dipasang dengan mudah menggunakan apt-get karena perangkat lunak tersebut sudah tersedia pada *repository* Ubuntu. Untuk melakukan pemasangan HAProxy, gunakan perintah apt-get install haproxy.

Setelah HAProxy diunduh, perangkat lunak tersebut belum berjalan karena belum diaktifkan. Untuk mengaktifkan *service haproxy*, buka berkas di `/etc/default/haproxy` kemudian ganti nilai `ENABLED` yang awalnya bernilai `0` menjadi `ENABLED=1`. Setelah itu *service haproxy* dapat dijalankan dengan menggunakan perintah `service haproxy start`. Untuk konfigurasi dari HAProxy nantinya akan diurus oleh *confd*. *confd* akan menyesuaikan konfigurasi dari HAProxy sesuai dengan kebutuhan aplikasi yang tersedia.

## Instalasi etcd dan confd

*etcd* dapat di unggah dengan menjalankan perintah berikut, `curl https://github.com/coreos/etcd/releases/download/v3.2.0-rc.0/etcd-v3.2.0-rc.0-linux-amd64.tar.gz`. Setelah proses unduh berhasil dilakukan, selanjutnya yang dilakukan adalah melakukan ekstrak berkasnya menggunakan perintah `tar -xvzf etcd-v3.2.0-rc.0-linux-amd64.tar.gz`. Berkas binary dari *etcd* bisa ditemukan pada folder `./bin/etcd`. Berkas inilah yang digunakan untuk menjalankan perangkat lunak *etcd*. Untuk menjalankannya, dapat dilakukan dengan menggunakan perintah `etcd --listen-client-urls http://0.0.0.0:5050 --advertise-client-urls http://128.199.250.137:5050`. Perintah tersebut memungkinkan *etcd* diakses oleh *host* lain dengan IP `128.199.250.137`, yang merupakan *host* dari *load balancer* dan *confd*. Setelah proses tersebut, *etcd* sudah siap untuk digunakan.

Setelah *etcd* siap digunakan, selanjutnya adalah memasang *confd*. Untuk menginstall *confd* gunakan rangkaian perintah berikut:

```
$ mkdir -p $GOPATH/src/github.com/kelseyhightower
$ git clone https://github.com/kelseyhightower/
```

```

confd.git $GOPATH/src/github.com/kelseyhightower/
confd
$ cd $GOPATH/src/github.com/kelseyhightower/confd
$ ./build

```

Setelah berhasil memasang confd, selanjutnya buka berkas `/etc/confd/confd.toml` dan isi berkas sesuai dengan Kode Sumber 1.3. Pengaturan tersebut bertujuan agar confd melakukan *listen* terhadap server etcd dan melakukan tindakan jika terjadi perubahan pada etcd.

```

confdir = "/etc/confd"
interval = 20
backend = "etcd"
nodes = [
    "http://128.199.250.137:5050"
]
prefix = "/"
scheme = "http"
verbose = true

```

**Kode Sumber 1.3:** Isi Berkas `confd.toml`

Setelah melakukan konfigurasi confd, selanjutnya adalah membuat *template* konfigurasi untuk HAProxy. Buka berkas di `/etc/confd/templates/haproxy.cfg.tmpl`. Jika berkas tidak ada maka buat berkasnya dan isi berkas sesuai dengan Kode Sumber 1.4.

```

global
    log /dev/log      local0
    log /dev/log      local1 notice
    chroot /var/lib/haproxy
    stats socket /run/haproxy/admin.
        sock mode 660 level admin
    stats timeout 30s

```

```

daemon
defaults
    log      global
    mode     http
    option   httplog
    option   dontlognull
    timeout  connect 5000
    timeout  client  50000
    timeout  server  50000
    errorfile 400 /etc/haproxy/errors
                /400.http
    errorfile 403 /etc/haproxy/errors
                /403.http
    errorfile 408 /etc/haproxy/errors
                /408.http
    errorfile 500 /etc/haproxy/errors
                /500.http
    errorfile 502 /etc/haproxy/errors
                /502.http
    errorfile 503 /etc/haproxy/errors
                /503.http
    errorfile 504 /etc/haproxy/errors
                /504.http
frontend http-in
    bind *:80

    # Define hosts
    {{range gets "/"images/*"}}
    {{$data := json .Value}}
        acl host_{{$data.image_name}}
            hdr(host) -i {{$data.
                domain}}.nota-no.life
    {{end}}

```



```

## Figure out which one to use
{{range gets "/images/*"}}
  {{$data := json . Value}}
    use_backend {{$data .
      image_name}}_cluster if
      host_{{$data . image_name
        }}
    {{end}}
{{range gets "/images/*"}}
  {{$data := json . Value}}
backend {{$data . image_name}}_cluster
  mode http
  balance roundrobin
  option forwardfor
  cookie JSESSIONID prefix
  {{range $data . containers}}
  server {{. name}} {{. ip}}:{{. port}}
    check
  {{end}}
{{end}}

```

**Kode Sumber 1.4:** Isi Berkas haproxy.cfg.tpl

Langkah terakhir adalah membuat berkas konfigurasi untuk HAProxy di `/etc/confd/conf.d/haproxy.toml`. Jika berkas tidak ada, maka buat berkasnya dan isi berkas sesuai dengan Kode Sumber 1.5.

```

[ template ]
src = "haproxy.cfg.tpl"
dest = "/etc/haproxy/haproxy.cfg"
keys = [
  "/images"
]

```

```
reload_cmd = "iptables -I INPUT -p tcp --
              dport 80 --syn -j DROP && sleep 1 &&
              service haproxy restart && iptables -D
              INPUT -p tcp --dport 80 --syn -j DROP"
```

**Kode Sumber 1.5:** Isi Berkas haproxy.toml

Setelah melakukan konfigurasi, selanjutnya adalah menjalankan confd dengan menggunakan perintah `confd &`.

## Pemasangan Redis

Redis dapat dipasang dengan mempersiapkan kebutuhan pustaka pendukungnya. Pustaka yang digunakan adalah `build-essential` dan `tc18.5`. Untuk melakukan pemasangannya, jalankan perintah berikut:

```
$sudo apt-get install build-essential
```

```
$sudo apt-get install tc18.5
```

Setelah itu unduh aplikasi Redis dengan menjalankan perintah

```
wget
http://download.redis.io/releases/redis-
stable.tar.gz. Setelah selesai diunduh, buka file dengan
perintah berikut:
```

```
$tar xzf redis-stable.tar.gz && cd redis-stable
```

Di dalam folder `redis-stable`, bangun Redis dari kode sumber dengan menjalankan perintah `make`. Setelah itu lakukan tes kode sumber dengan menjalankan `make test`. Setelah selesai, pasang Redis dengan menggunakan perintah `sudo make install`. Setelah selesai melakukan pemasangan, Redis dapat diaktifkan dengan menjalankan berkas `bash` dengan nama `install_server.sh`.

Untuk menambah pengaman pada Redis, diatur agar Redis hanya bisa dari `localhost`. Untuk melakukannya, buka file `/etc/redis/6379.conf`, kemudian cari baris `bind`

127.0.0.1. Hapus komen jika sebelumnya baris tersebut dalam keadaan tidak aktif. Jika tidak ditemukan baris dengan isi tersebut, tambahkan pada akhir berkas baris tersebut.

### **Pemasangan kerangka kerja React**

Pada pengembangan sistem ini, penggunaan pustaka React dibangun di atas konfigurasi Create React App. Untuk memasang Create React App, gunakan perintah `npm install -g create-react-app`. Setelah terpasang, untuk membangun aplikasinya jalankan perintah `create-react-app fe-controller`. Setelah proses tersebut, dasar dari aplikasi sudah terbangun dan siap untuk dikembangkan lebih lanjut.

*(Halaman ini sengaja dikosongkan)*

## LAMPIRAN B

### KODE SUMBER

#### Let's Encrypt Cross Signed

```
-----BEGIN CERTIFICATE-----
MIIEkjCCA3qgAwIBAgIQCgFBQgAAAVOF
    c2oLheynCDANBgkqhkiG9w0BAQsFADA/
MSQwIgYDVQQKEExtEaWdpdGFsIFNpZ25h
    dHVyZSBUcnVzdCBDby4xFzAVBgNVBAMT
DkRRTVCBSb290IENBIFgzMB4XDTE2MDMx
    NzE2NDA0NloXDTIxMDMxNzE2NDA0Nlow
SjELMAkGA1UEBhMCVVMxHjAuBgNVBAoT
    DUxldCdzIEVuY3J5cHQxIzAhBgNVBAMT
GkxldCdzIEVuY3J5cHQxXV0aG9yaXR5
    IFgzMIIBIjANBgkqhkiG9w0BAQEFAAO
AQ8AMIIBCgKCAQEAAnNMM8FrILke3cl03
    g7NoYzDq1zUmGSXhvb418XCSL7e4S0EF
q6meNQhY7LEqxGiHC6PjdeTm86dicbp5
    gWAf15Gan/
PQeGdxyGkOlZHP/uaZ6WA8
SMx+yk13EiSdRxta67nsHjcAHJyse6cF
    6
s5K671B5TaYucv9bTyWaN8jKkKQDIZ0
Z8h/pZq4UmEUEz9l6YKH9v6Dlb2honz
    hT+Xhq+
w3Brvaw2VFf3EK6BlspkENnWA
a6xK8xuQSXgvopZPKiAlKQTGdMDQMc2P
    MTiVFrqoM7hD8bEfwbZ/onkxEz0tNvjj
/Pizark5McWvxI0NHWWQM6r6hCm21AvA
    2
H3DkwIDAQABo4IBfTCCAXkwEgYDVR0T
AQH/BAGwBgEB/wIBADAQBgNVHQ8BAf8E
    BAMCAYYwfwYIKwYBBQUHAQEecBxMDIG
CCsGAQUFBzABhiZodHRwOi8vaXNyZy50
    cnVzdGlkLm9jc3AuaWRLbnRydXN0LmNv
bTA7BggrBgEFBQcwAoYvaHR0cDovL2Fw
    cHMuaWRLbnRydXN0LmNvbS9yb290cy9k
-----
```

```

c3Ryb290Y2F4My5wN2MwHwYDVR0jBBgw
  FoAUxKexpHsscfrb4UuQdf/EFWCfiRAw
VAYDVR0gBE0wSzAIBgZngQwBAGewPwYL
  KwYBBAGC3xMBAQEwMDAuBggrBgEFBQcC
ARYiaHR0cDovL2Nwcy5yb290LXgxLmxl
  dHNIbmNyeXB0Lm9yZzA8BgNVHR8ENTAz
MDGgL6AthitodHRwOi8vY3JsLmlkZW50
  cnVzdC5jb20vRFNUUk9PVENBWDNDUkwu
Y3JsMB0GA1UdDgQWBBSoSmpjBH3duubR
  ObemRWXv86jsoTANBgkqhkiG9w0BAQsF
AAOCAQEA3TPXEfNjWDjdGBX7CVW+d1a5
  cEilaUcne8IkCJLxWh9KEik3JHRRHGJo
uM2VcGf196S8TihRzZvoroe6ti6WqEB
  mtzw3Wodatg+VyOeph4EYpr/1wXKtx8/
wApIvJSwtmVi4MFU5aMqrSDE6ea73Mj2
  tcMyo5jMd6jmeWUHK8so/joWUoHOUgwu
X4Po1QYz+3dszkDqMp4fklxBwXRsw10K XzPMTZ+
  sOPAveyxindmjkW8lGy+QsRlG
PfZ+G6Z6h7mjem0Y+iWlkYcV4PIWL1iw
  Bi8saCbGS5jN2p8M+X+Q7UNKEkROb3N6
KOqkqm57TH2H3eDJAKsnh6/DNFu0Qg==
-----END CERTIFICATE-----

```

**Kode Sumber 2.1:** Let's Encrypt X3 Cross Signed.pem

## BIODATA PENULIS



**Fourir Akbar**, akrab dipanggil Oing, lahir pada tanggal 25 April 1996 di Surabaya. Penulis merupakan seorang mahasiswa yang sedang menempuh studi di Departemen Informatika Institut Teknologi Sepuluh Nopember. Memiliki beberapa hobi antara lain futsal dan DOTA. Pernah menjadi asisten dosen pada mata kuliah sistem operasi dan mata kuliah jaringan komputer pada semester 2016/2017 dan 2017/2018. Lalu juga pernah menjadi asisten dosen pada mata kuliah sistem terdistribusi pada tahun ajaran 2017/2018. Penulis juga pernah menjadi asisten dosen pendidikan informatika dan komputer terapan (PIKTI) ITS pada tahun ajaran 2016/2017 dan 2017/2018. Selama menempuh pendidikan di kampus, penulis juga aktif dalam organisasi kemahasiswaan, antara lain sebagai Staff Departemen Hubungan Luar Himpunan Mahasiswa Teknik Computer-Informatika pada tahun ajaran 2015/2016. Penulis juga aktif dalam kepanitiaan Schematics, antara lain sebagai Staff Biro Revolutionary Entertainment and Expo with Various Arts pada tahun ajaran 2015/2016 dan menjadi Badan Pengurus Harian (BPH) Biro Perlengkapan dan Transportasi pada tahun 2016/2017. Penulis juga merupakan salah satu administrator aktif pada Laboratorium Arsitektur dan jaringan Komputer di Departemen Informatika ITS.