



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

DEPARTEMEN 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

DEPARTEMEN 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 Departemen 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

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. Kontainer berfungsi untuk mengisolasi aplikasi atau servis dan dependensinya. Untuk setiap servis atau aplikasi yang terisolasi dibutuhkan satu kontainer pada server host yang ada dan setiap kontainer akan menggunakan sumber daya yang ada pada server host selama kontainer tersebut menyala.

Dalam kasus ini, setiap user 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 user, contohnya manajemen bandwith, hak akses, waktu, dan lain sebagainya. Jika user telah selesai mengakses atau menggunakan jaringan ITS, maka kontainer pada user tersebut akan di-destroy, sehingga hal ini dapat meringankan beban server.

Kata-Kunci: Docker, Internet Access Management, Kontainer

(Halaman ini sengaja dikosongkan)

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

Nowadays, docker containers have been widely used in the word of technology. The docker containers is an operating system level virtualization to run some isolated linux systems (containers) on a host. Containers are used to isolate applications or services and its dependencies. For every service or app that isolated it takes one container on the existing host server and each container will use the existing resources on the host server as long as the container is on.

In this case, any user accessing or using ITS network is one service that a container will be created on the host server. This can simplify management of each user, for example bandwidth management, access rights, time, and many more. If the user has finished accessing or using the the network ITS, then the container on the user will be destroyed, so this can reduce the server load.

Keywords: Docker, Internet Access Management, Container.

(Halaman ini sengaja dikosongkan)

KATA PENGANTAR

Puji syukur Penulis panjatkan kepada Allah SWT. atas pimpinan, penyertaan, dan karunia-Nya sehingga Penulis dapat menyelesaikan Tugas Akhir yang berjudul **Rancang Bangun Perangkat Lunak Internet Acces Management Berbasis Kontainer**. Penggerjaan Tugas Akhir ini merupakan suatu kesempatan yang sangat baik bagi penulis. Dengan penggerjaan 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. Selesainya Tugas Akhir ini tidak lepas dari bantuan dan dukungan beberapa pihak. Sehingga pada kesempatan ini penulis mengucapkan syukur dan terima kasih kepada:

1. 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 penggerjaan Tugas Akhir ini.
2. 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 penggerjaan Tugas Akhir ini.
3. Bagus Jati Santoso, S.Kom., PhD. selaku dosen pembimbing yang telah memberikan ilmu, dan masukan kepada Penulis.
4. 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.
5. Seluruh teman Penulis di Jurusan Teknik Informatika ITS yang telah memberikan dukungan dan semangat kepada

Penulis selama Penulis menyelesaikan Tugas Akhir ini.

6. 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

ABSTRAK	vii
ABSTRACT	ix
KATA PENGANTAR	xi
DAFTAR ISI	xiii
DAFTAR TABEL	xvii
DAFTAR GAMBAR	xix
DAFTAR KODE SUMBER	xxi
BAB I PENDAHULUAN	1
1.1 Latar Belakang	1
1.2 Rumusan Masalah	1
1.3 Batasan Masalah	2
1.4 Tujuan	2
1.5 Manfaat	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
BAB II LANDASAN TEORI	7
2.1 Python	7
2.2 Flask	7
2.3 <i>Gunicorn</i>	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	<i>MySQL</i>	11
2.8	<i>Mitmproxy</i>	13
2.9	<i>VirtualBox</i>	13
2.10	<i>Proxmox VE</i>	14
2.11	<i>Docker</i>	15
2.11.1	<i>Docker Container</i>	16
2.11.2	<i>Docker Images</i>	16
2.11.3	<i>Docker Registry</i>	17
BAB III DESAIN DAN PERANCANGAN		19
3.1	Deskripsi Umum Sistem	19
3.2	Kasus Penggunaan	19
3.3	Arsitektur Sistem	22
3.3.1	Desain Umum Sistem	22
3.3.2	Pembuatan Halaman <i>Login</i> dari Sebuah Sistem	24
3.3.3	Perancangan Pembuatan Aturan untuk Mengarahkan <i>Traffic Client</i> ke Halaman <i>Login</i> dari Sistem	28
3.3.4	Pembuatan <i>Middleware</i> untuk Menerima Permintaan dari <i>Client</i>	29
3.3.5	Perancangan Pemasangan Kontainer pada <i>Docker Host</i>	30
3.3.6	Pembacaan <i>Log File</i> dari <i>Client</i>	32
3.3.7	Pembuatan Aturan untuk Mengarahkan <i>Traffic Client</i> ke Kontainer <i>Docker</i> dari Tiap-Tiap <i>Client</i>	32
BAB IV IMPLEMENTASI		35
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>	41
4.3	Implementasi Pembuatan Aturan untuk Mengarahkan <i>Traffic Client</i> ke Halaman <i>Login</i> dari Sistem	42
4.4	Implementasi Pembuatan <i>Middleware</i>	43
4.4.1	Implementasi <i>Web Service</i> pada <i>Middleware</i>	43
4.4.2	Implementasi Basis Data pada <i>Middleware</i>	46
4.5	Implementasi Pemasangan Kontainer <i>Docker</i> pada <i>Docker Host</i>	47
4.5.1	Menambahkan dan Memperbarui Kontainer <i>Docker</i> yang Berisikan Mitmproxy	47
4.5.2	Menggunakan <i>Image</i> Kontainer <i>Docker</i> yang Sudah Dibuat	50
4.6	Implementasi Pembuatan Aturan untuk Mengarahkan <i>Traffic Client</i> ke Kontainer <i>Docker</i> dari Tiap-Tiap <i>Client</i>	50
4.7	Implementasi Pembacaan <i>Log File</i> dari <i>Client</i>	51
BAB V	PENGUJIAN DAN EVALUASI	53
5.1	Lingkungan Uji Coba	53
5.2	Skenario Uji Coba	55
5.2.1	Skenario Uji Fungsionalitas	55
5.2.2	Skenario Uji Performa	58
5.3	Hasil Uji Coba dan Evaluasi	59
5.3.1	Uji Fungsionalitas	60
5.3.2	Uji Performa	67

BAB VI PENUTUP	75
6.1 Kesimpulan	75
6.2 Saran	76
DAFTAR PUSTAKA	77

DAFTAR TABEL

3.1	Daftar Kode Kasus Penggunaan	21
3.1	Daftar Kode Kasus Penggunaan	22
3.2	Atribut basis data nrp-mahasiswa	26
3.3	Atribut basis data kontainer	30
4.1	Daftar Rute <i>Web Service</i>	40
4.2	Daftar Rute <i>Web Service</i>	45
5.1	Komputer sebagai <i>Docker Host</i>	53
5.2	Komputer sebagai <i>Server</i> dari Halaman <i>Login</i> . .	54
5.3	<i>Client 1</i>	54
5.4	<i>Client 2</i>	55
5.5	Skenario Uji Coba User dapat <i>Login</i> ke Dalam Sistem	56
5.6	Skenario Uji Coba User dapat <i>Login</i> ke Dalam Sistem	57
5.7	Skenario Uji Coba <i>Docker Host</i> dapat Menerima Perintah Penyediaan Kontainer <i>Docker</i>	58
5.8	Hasil Skenario Uji Coba mengirim permintaan penyediaan kontainer	60
5.9	Kondisi Awal Penggunaan Sumberdaya <i>Docker Host</i> sebelum uji coba dijalankan	61
5.10	Hasil Skenario Uji Coba Sistem dapat melakukan Penghitungan AHP	61
5.11	Hasil Skenario Uji Coba <i>Docker Host</i> dapat menerima perintah penyediaan kontainer	62
5.12	Hasil Skenario Uji Coba <i>Docker Host</i> dapat Mengirim Data Resourceny Masing-Masing . . .	65
5.13	Kondisi Awal Ketersediaan Sumberdaya <i>Docker Host</i> sebelum uji coba dijalankan	68
5.14	Hasil Uji Coba Performa sistem dengan <i>Image Docker</i> httpd dan nginx menggunakan AHP	68

5.15 Rata-rata Kondisi Akhir Ketersediaan Sumberdaya <i>Docker Host</i> Setelah Uji Coba Dijalankan	69
5.16 Kondisi Awal Ketersediaan Sumberdaya <i>Docker Host</i> Sebelum Uji Coba Dijalankan	69
5.17 Hasil Uji Coba Performa Sistem dengan <i>Image Docker</i> httpd dan nginx Menggunakan <i>Round Robin</i>	69
5.18 Kondisi Akhir Ketersediaan Sumberdaya <i>Docker Host</i> Setelah Uji Coba Dijalankan	70
5.19 Kondisi Awal Ketersediaan Sumberdaya <i>Docker Host</i> Sebelum Uji Coba Dijalankan	70
5.20 Hasil Uji Coba Performa Sistem dengan <i>Docker Image</i> httpd dan nginx Menggunakan AHP	71

DAFTAR GAMBAR

2.1	Perbandingan <i>docker</i> dan virtual machine	15
3.1	Digram Kasus Penggunaan	20
3.2	Arsitektur Komponen Sistem	24
3.3	Desain Halaman <i>Login</i>	27
3.4	Desain <i>Backend</i> dari Halaman <i>Login</i>	27
3.5	Desain Mengarahkan <i>Traffic Client</i> ke Halaman <i>Login</i>	28
3.6	Alur kerja dari <i>mitmproxy transparent</i> HTTP . . .	31
3.7	Alur kerja dari <i>mitmproxy transparent</i> HTTPS . .	32
3.8	Desain pembuatan aturan untuk mengarahkan <i>traffic client</i> ke kontainer <i>docker</i>	33
5.1	Arsitektur Pengujian Performa	59
5.2	Gambar Hasil Uji Sistem dapat melakukan Penghitungan AHP	62
5.3	Web Service Mengirimkan perintah penyediaan kontainer	64
5.4	<i>Docker Host</i> berhasil membuat kontainer <i>docker</i> .	65
5.5	Data CPU Setiap Docker Host pada InfluxDB . .	67
5.6	Data RAM Setiap Docker Host pada InfluxDB . .	67
5.7	Data Penyimpanan File Setiap Docker Host pada InfluxDB	67
5.8	Grafik Kondisi Akhir Ketersediaan Rata-Rata CPU	71
5.9	Grafik Kondisi Akhir Ketersediaan Memori . . .	72
5.10	Grafik Kondisi Akhir Ketersedian Penyimpanaan File	72
5.11	Grafik Waktu Pendistribusian Docker oleh Sistem Berdasarkan Jumlah Kontainer	73

(Halaman ini sengaja dikosongkan)

DAFTAR KODE SUMBER

4.1	Command untuk installasi Python	37
4.2	Command untuk installasi Flask	37
4.3	Command untuk installasi Gunicorn	37
4.4	Command untuk installasi Supervisor	37
4.5	Konfigurasi tambahan Supervisor	38
4.6	Command untuk Reload Supervisor	38
4.7	Command untuk installasi Supervisor	38
4.8	Konfigurasi tambahan untuk Nginx	39
4.9	Command untuk mengaktifkan konfigurasi Nginx	39
4.10	Command untuk merestart Nginx	39
4.11	Pseudocode Web Service	41
4.12	<i>Query</i> untuk membuat tabel testing	42
4.13	Command untuk mengarahkan <i>client</i> ke halaman <i>login</i>	42
4.14	Command untuk installasi Python	43
4.15	Command untuk installasi Flask	44
4.16	Command untuk installasi Flask	44
4.17	Command untuk installasi Flask	44
4.18	Pseudocode Web Service	46
4.19	<i>Query</i> untuk membuat tabel testing	46
4.20	Perintah untuk installasi Docker	47
4.21	Perintah untuk installasi Ansible	47
4.22	Perintah untuk <i>Pull</i> Ubuntu	48
4.23	Perintah untuk Menjalankan <i>Image</i> Ubuntu	48
4.24	Perintah untuk Pemasangan <i>Mitmproxy</i>	48
4.25	Perintah untuk Mengaktifkan <i>ipv4.forwarding</i>	49
4.26	Perintah untuk Menghentikan Kontainer Docker	49
4.27	Perintah untuk <i>Commit</i> Kontainer Docker	49
4.28	Perintah untuk <i>Push Image</i> ke Docker Hub	50
4.29	Perintah untuk <i>Pull Image mitmproxy</i>	50
4.30	Perintah untuk <i>Pull Image mitmproxy</i>	50
4.31	Command untuk mengarahkan <i>client</i> ke halaman <i>login</i>	51
4.32	Perintah untuk Membaca <i>File Log</i> dari <i>Mitmproxy</i>	51

(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 mengarahkan *traffic* dari *client* ke halaman *login*?

2. Bagaimana cara membuat sebuah kontainer *docker* secara otomatis ketika terdapat *client* yang terhubung dengan jaringan ITS?
3. Bagaimana cara mengarahkan *traffic* dari *client* ke kontainer *docker* yang sesuai?
4. Bagaimana cara mengetahui apa saja yang diakses oleh *client*?
5. Bagaimana perbandingan performa antara IAM konvensional dengan IAM berbasis kontainer?

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 untuk mengarahkan *traffic* dari *client* ke halaman *login*.
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 apa saja yang diakses oleh *client*.
5. Mengetahui perbandingan performa antara IAM konvensional dengan IAM berbasis kontainer.

1.5 Manfaat

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

1. Mengetahui cara untuk mengarahkan *traffic* dari *client* ke halaman *login*.
2. Mengetahui cara untuk mengarahkan *traffic* dari *client* ke kontainer *docker* yang sesuai.
3. Mempermudah pengecekan *access log* dari setiap *client*.
4. Mengetahui apa saja yang diakses oleh *client* yang menggunakan jaringan ITS.
5. Meringankan beban dari penggunaan *server* di ITS karena penggunaan kontainer *docker* lebih ringan.

1.6 Metodologi

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

1.6.1 Studi literatur

Studi literatur merupakan langkah yang dilakukan untuk mendukung dan memastikan setiap tahap penggerjaan 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 penggerjaan 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.

BAB II

LANDASAN TEORI

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 [?]

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

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 mempunya 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 *Gunicorn*

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

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

Gunicorn 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 *Gunicorn* dapat menanganiinya.

2.4 *Supervisor*

Supervisor adalah sistem yang berbasis *client* atau *server*, yang memungkinkan penggunanya 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. [?]

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 melakuakn 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 lisensi *GNU General Public License (GPL)*, tetapi juga tersedia lisensi

komersial untuk kasus-kasus dimana penggunanya 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 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 mengamumsikan 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*

VirtuaBox 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 diakuisisi oleh Sun Microsystems. Sun Microsystems kemudian juga diakuisisi 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.

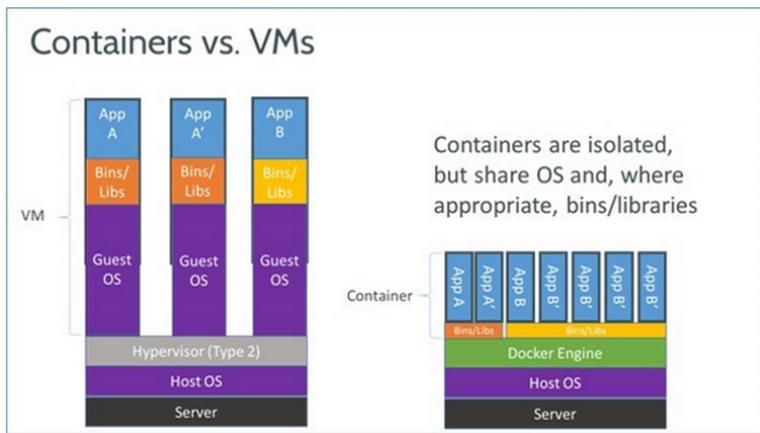
2.10 *Proxmox VE*

Proxmox VE adalah proyek *open source*, dikembangkan dan dikelola oleh Proxmox Server Solutions GmbH. *Proxmox VE* dilisensikan di bawah GPLv3. Fungionalitas dari *Proxmox VE* dapat dijalankan pada *browser*, salah satunya adalah mengintegrasikan tampilan console dari *virtual machine*, integrasi dan manajemen *cluster Proxmox VE*, teknologi AJAX untuk pembaruan sumber daya secara dinamis, dan mengamankan akses untuk semua *virtual machine* dengan menggunakan enkripsi SSL.

Proxmox VE adalah *tweak* dari distribusi Debain yang telah terinstall fitur manajemen berbasis web, dan mempunyai *kernel* yang telah dioptimalkan. *Proxmox VE* menawarkan dua solusi untuk melakuakn virtualisasi, antara lain *full-or paravirtualization* dengan menggunakan KVM dan OpenVZ virtualisasi berbasis kontainer.

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 terbentu 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 diatasnya 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 iamges* berfungsi untuk membuat kontainer *docker*, dimana dengan menggunakan satu *docker iamge* 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.

(Halaman ini sengaja dikosongkan)

BAB III

DESAIN DAN PERANCANGAN

Pada bab ini dibahas mengenai analisis dan perancangan dari sistem.

3.1 Deskripsi Umum Sistem

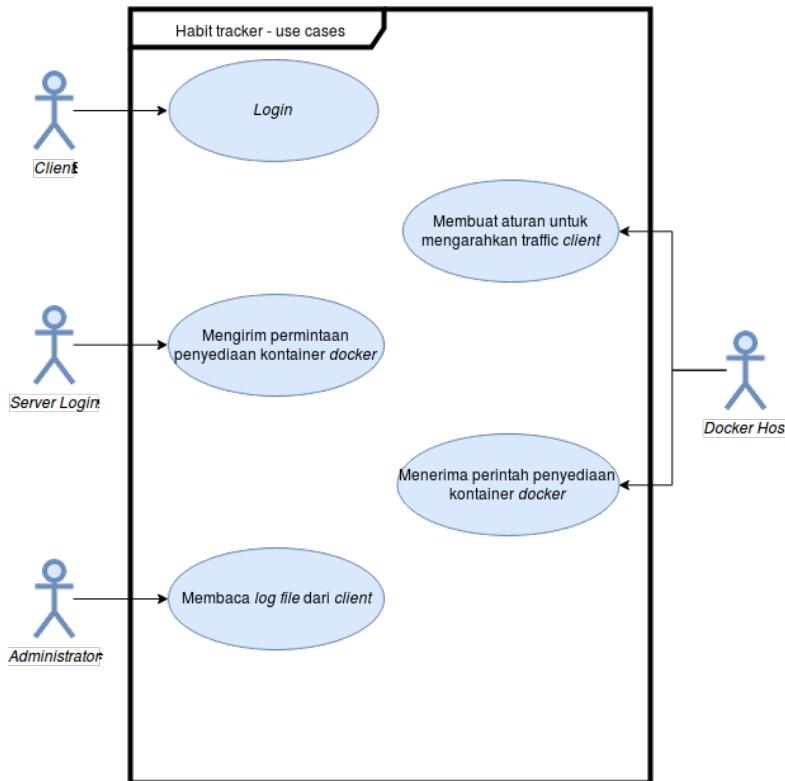
Sistem yang akan dibuat adalah sebuah sebuah sistem yang dapat membuat sebuah kontainer *docker* secara otomatis untuk setiap satu *client* yang telah *login* ke dalam sistem. Saat *client* belum *login* ke dalam sistem, maka *client* tersebut akan diarahkan ke halaman *login* dari sistem. Saat *client* 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

Kode Kasus Penggunaan	Nama Kasus Penggunaan	Keterangan
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

Kode Kasus Penggunaan	Nama Kasus Penggunaan	Keterangan
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 client secara <i>live</i> ataupun juga ketika <i>client</i> telah selesai menggunakannya.

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. Pembacaan *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.

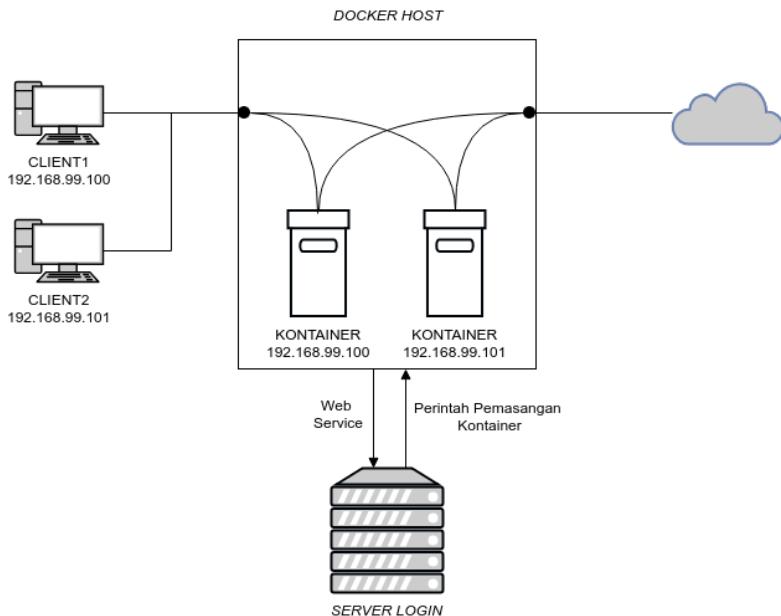
4. Pembuatan aturan untuk mengarahkan *traffic client* ke kontainer *docker* dari tiap-tiap *client*.

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 memasangkan kontainer *docker* pada *docker host* secara otomatis. Hal ini dilakukan dengan menjalankan sebuah perintah penyediaan kontainer pada *docker host*.

6. Pembacaan *log file* dari *client*. Berfungsi untuk melihat apa saja yang telak 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 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 itnernet.

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

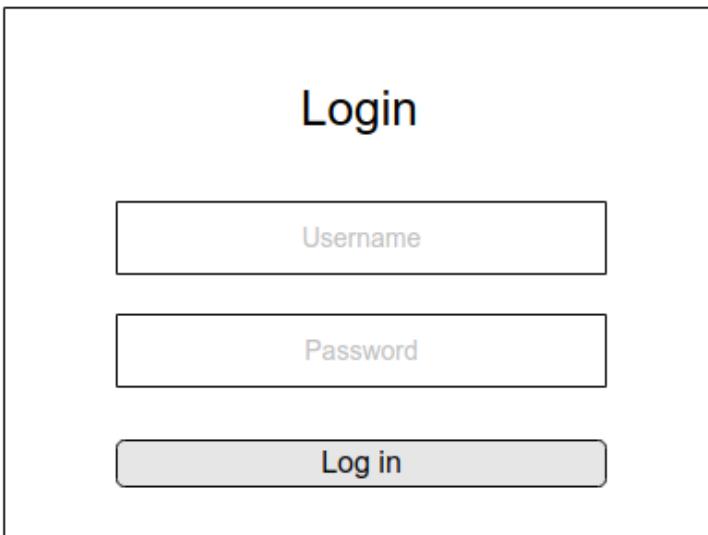
Tabel 3.2: Atribut basis data npn-mahasiswa

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 npn tersebut sedang digunakan (1), atau sedang tidak digunakan (0).

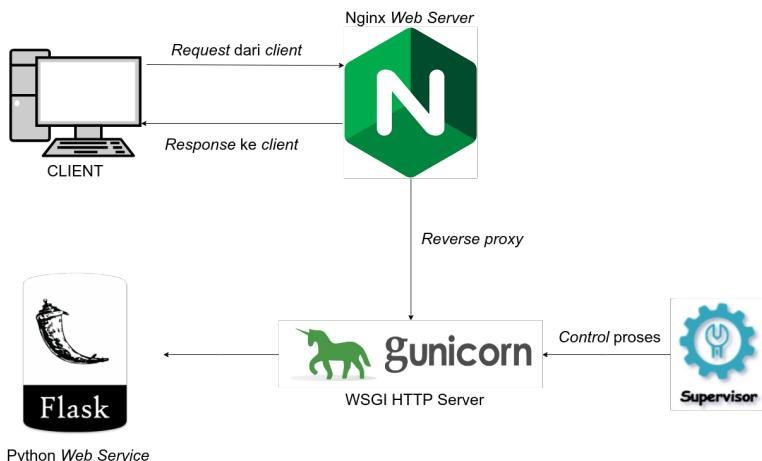
3.3.2.2 Desain Web Service

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.

Lalu untuk desain *backend* dari *web serive* 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.



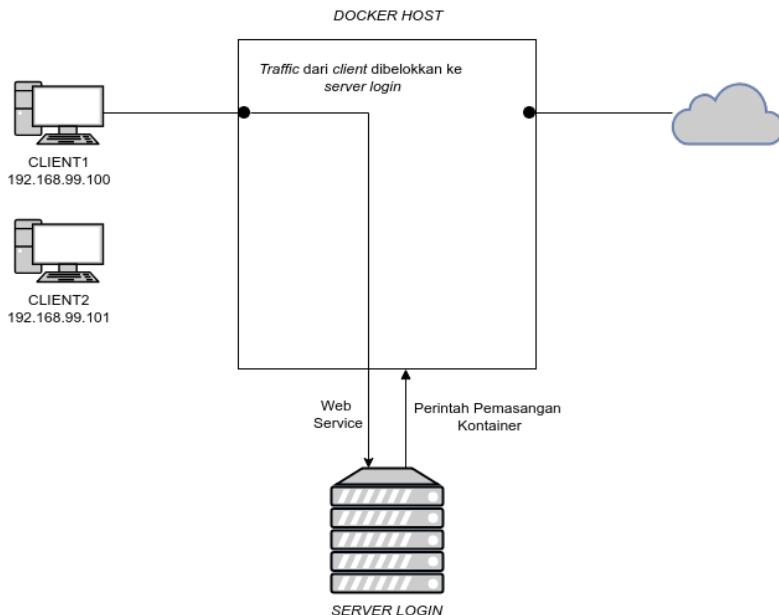
Gambar 3.3: Desain Halaman *Login*



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 AUTO_INCREMENT.
2	username	varchar(50)	Menunjukkan NRP dari mahasiswa yang telah berhasil dibuatkan satu kontainer <i>docker</i> .
3	ip	varchar(50)	Menunjukkan IP dari <i>client</i> yang telah berhasil satu kontainer <i>docker</i> .
4	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*.

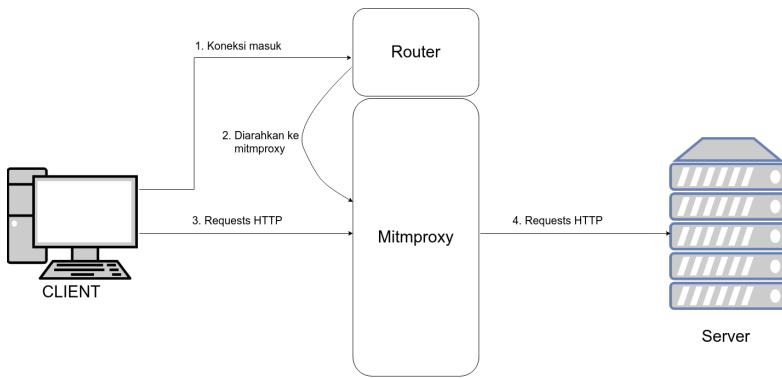
3.3.5 Perancangan Pemasangan Kontainer pada Docker Host

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.

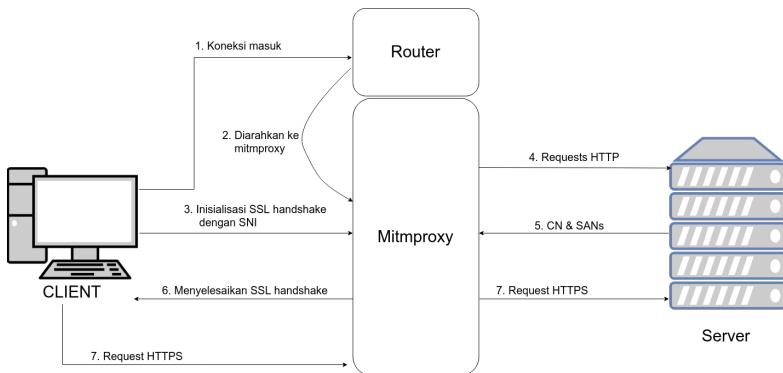
Saat kontainer *docker* telah berhasil dibuat, maka kontainer *docker* tersebut akan mempunyai satu port khusus yang sama dengan *client* yang baru saja *login*. Port khusus nantinya akan digunakan untuk mengarahkan *traffic* dari *client* yang akan mengakses internet.

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.



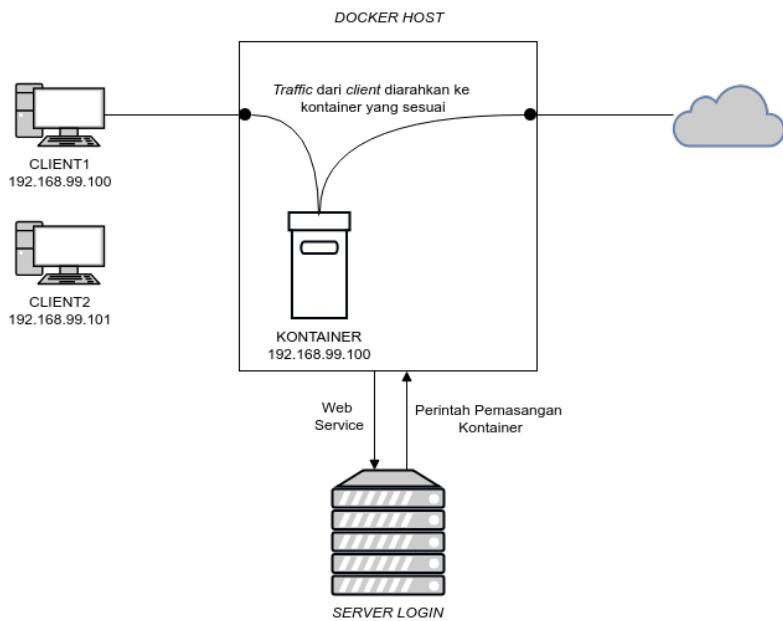
Gambar 3.7: Alur kerja dari *mitmproxy transparent HTTPS*

3.3.6 Pembacaan *Log File* dari *Client*

Pembacaan *log file* dari *client* adalah komponen yang bertugas untuk mengetahui apa saja yang telah diakses oleh *client*. *Log* yang dapat dibaca adalah *log HTTP* maupun *log HTTPS*. *Log file* nantinya akan dibuat semudah mungkin untuk dibaca oleh manusia.

3.3.7 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*

(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 dipasangkan pada *server*.
12. *Iptables* versi 1.6.0 untuk membuat aturan terhadap *client*.
13. *VIM* versi 7.4.1 sebagai *text editor*.

4.2 Implementasi Pembuatan Halaman *Login* dari Sebuah Sistem

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 IP SERVER 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 IP SERVER 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. Untuk melakukan pemasangan *Python* versi 3.5.2 pada sistem operasi Ubuntu *Server*, jalankan *command* pada terminal seperti pada Kode Sumber 4.1.

```
sudo apt-get update
sudo apt-get install python3
```

Kode Sumber 4.1: Command untuk installasi Python

Lalu untuk melakukan pemasangan *Flask* versi 1.0.2 pada sistem operasi Ubuntu *Server*, jalankan *command* pada teminal seperti pada Kode Sumber 4.2.

```
sudo apt-get install python3-pip
sudo pip install flask
```

Kode Sumber 4.2: Command untuk installasi Flask

Kemudian untuk melakukan pemasangan *Gunicorn* versi 19.8.1 pada sistem operasi Ubuntu *Server*, jalankan *command* pada terminal seperti pada Kode Sumber 4.3.

```
sudo pip install gunicorn
```

Kode Sumber 4.3: Command untuk installasi Gunicorn

Setelah itu untuk melakukan pemasangan *Supervisor* versi 3.2.0 pada sistem operasi Ubuntu *Server*, jalankan *command* pada terminal seperti pada Kode Sumber 4.4.

```
sudo apt-get install python-setuptools
sudo apt-get install supervisor
```

Kode Sumber 4.4: Command untuk installasi Supervisor

Lalu agar *Supervisor* dapat selalu menjalankan perangkat lunak *Gunicorn*, perlu dilakukan konfigurasi tambahan dengan

menambahkan Kode Sumber 4.5 pada file `/etc/supervisor/conf.d/app.conf`.

```
[program:flask-loginpage]
command =
/home/fourirakbar/flask-loginpage/flask-loginpageenv/bin/python
/home/fourirakbar/flask-loginpage/flask-loginpageenv/bin/gunicorn
-b 0.0.0.0:400 app:app
directory = /home/fourirakbar/flask-loginpage
user = fourirakbar
stdout_logfile =
/home/fourirakbar/flask-loginpage/logs/app_stdout.log
stderr_logfile =
/home/fourirakbar/flask-loginpage/logs/app_stderr.log
redirect_stderr = True
autostart = True
environment = PATH =
"/home/fourirakbar/flask-loginpage/flask-loginpageenv/bin",
PRODUCTION=1
```

Kode Sumber 4.5: Konfigurasi tambahan Supervisor

Setelah selesai menambah konfigurasi seperti pada Kode Sumber 4.5, *reload Supervisor* dengan menjalankan *command* pada terminal seperti pada Kode Sumber 4.6.

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

Kode Sumber 4.6: Command untuk Reload Supervisor

Lalu untuk melakukan pemasangan *Nginx* versi 1.10.3 pada sistem operasi Ubuntu *Server*, jalankan *command* pada terminal seperti pada Kode Sumber 4.7.

```
sudo apt-get install nginx
```

Kode Sumber 4.7: Command untuk installasi Supervisor

Lalu agar *Nginx* dapat dijalankan sebagai *web server* dari perangkat lunak halaman *login*, perlu dilakukan konfigurasi tambahan dengan menambahkan Kode Sumber 4.8 pada file `/etc/nginx/sites-available/app`.

```

server {
    listen 80;
    server_name 10.151.36.120;

    add_header X-Frame-Options SAMEORIGIN;
    add_header X-Content-Type-Options nosniff;
    add_header X-XSS-Protection "1; mode=block";

    access_log /home/fourirakbar/flask-loginpage/logs/app_access.log;

    location / {
        proxy_pass          http://127.0.0.1:4000;
        proxy_set_header Host            $host;
        proxy_set_header X-Real-IP   $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }

    location ^~ /static/ {
        include      /etc/nginx/mime.types;
        alias       /home/fourirakbar/flask-loginpage/static/;
    }
}

```

Kode Sumber 4.8: Konfigurasi tambahan untuk Nginx

Setelah itu, aktifkan konfigurasi diatas dengan menjalankan *command* pada terminal seperti pada Kode Sumber 4.9.

```

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

```

Kode Sumber 4.9: Command untuk mengaktifkan konfigurasi Nginx

Setelah itu, jalankan Kode Sumber 4.10 supaya konfigurasi yang baru saja diaktifkan dapat digunakan.

```

sudo service nginx restart

```

Kode Sumber 4.10: Command untuk merestart Nginx

4.2.1.1 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 *login* tertera pada Tabel 4.1.

Tabel 4.1: Daftar Rute *Web Service*

HTTP Method	Rute	Deskripsi
GET	/	Berfungsi untuk mengarahkan <i>redirect</i> ke ruta <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> .

4.2.1.2 *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 4.11 diperlihatkan bagaimana implementasinya dalam bentuk *pseudocode*.

Kode Sumber 4.11: Pseudocode Web Service

```
1  Check whether the client is already login  
2      or not yet  
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 4.12.

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

Kode Sumber 4.12: 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* seperti Kode Sumber 4.13.

```
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 10.151.36.130
--dport 4000 -j ACCEPT
iptables -t nat -I PREROUTING 1 -p tcp -s 192.168.99.0/24
--dport 80 -j DNAT --to 10.151.36.130:4000
```

Kode Sumber 4.13: 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 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 unak antara lain:

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

Lalu sistem operasi yang digunakan adalah sistem operasi Linux Mint 18.03 64 Bit yang akan digunakan sebagai *docker host*. *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. Untuk melakukan pemasangan *Python* versi 3.5.2 pada *docker host*, jalankan *command* pada terminal seperti pada Kode Sumber 4.14.

```
sudo apt-get update  
sudo apt-get install python3
```

Kode Sumber 4.14: Command untuk instalasi Python

Lalu untuk melakukan pemasangan *Flask* versi 1.0.2 pada

docker host, jalankan *command* pada teminal seperti pada Kode Sumber 4.15.

```
sudo apt-get install python3-pip  
sudo pip install flask
```

Kode Sumber 4.15: Command untuk installasi Flask

Lalu untuk melakukan pemasangan *Docker* versi 1.13.1 pada *docker host*, jalankan *command* pada terminal seperti pada Kode Sumber 4.20.

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

Kode Sumber 4.16: Command untuk installasi Flask

Setelah itu jalankan Kode Sumber 4.17 supaya *docker* dapat dijalankan ketika *docker host* menyala.

```
sudo systemctl enable docker
```

Kode Sumber 4.17: Command untuk installasi 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 Pseduocode *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 4.18 diperlihatkan bagaimana implementasinya dalam bentuk *pseudocode*.

Kode Sumber 4.18: Pseudocode Web Service

```

1   Check whether the client is already
2       login or not yet
3
4   if session.get login
5       create container
6       add new rules to container
7       return
8   else
9       add new rules
10      client open page login
11      client login
12      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 4.19.

```

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

```

Kode Sumber 4.19: Query untuk membuat tabel testing

4.5 Implementasi Pemasangan Kontainer Docker pada Docker Host

Pada implementasi pemasangan kontainer *docker* pada *docker host* menggunakan perangkat lunak *docker*. Diperlukan beberapa tahap untuk dapat menggunakan perangkat lunak *docker*, yaitu tahap pemasangan dan konfigurasi. Untuk melakukan pemasangan *docker* versi 1.13.1 pada sistem operasi Linux Mint jalankan Kode Sumber 4.20 pada terminal.

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

Kode Sumber 4.20: Perintah untuk installasi Docker

Setelah berhasil melakukan pemasangan *docker* versi 1.13.1, 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 4.21.

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

Kode Sumber 4.21: Perintah untuk installasi Ansible

4.5.1 Menambahkan dan Memperbarui Kontainer Docker yang Berisi 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 4.22.

```
docker pull ubuntu
```

Kode Sumber 4.22: Perintah untuk *Pull* Ubuntu

Setelah berhasil diunduh, selanjutnya jalankan sistem operasi Ubuntu dengan menggunakan perintah yang tertera pada Kode Sumber 4.23.

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

Kode Sumber 4.23: 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 4.24.

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

Kode Sumber 4.24: 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 4.25

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

Kode Sumber 4.25: 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 4.26.

```
docker stop [nama_container]
```

Kode Sumber 4.26: 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 4.27.

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

Kode Sumber 4.27: 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 *mitiproxy-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 4.28.

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

Kode Sumber 4.28: 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 *mitiproxy*, penulis tidak perlu melakukannya lagi. Penulis hanya perlu memanggil kontainer *docker* dengan menjalankan perintah pada Kode Sumber 4.29.

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

Kode Sumber 4.29: Perintah untuk *Pull Image mitiproxy*

Lalu untuk menjalankan kontainer *docker* yang sudah di *pull*, jalankan perintah pada Kode Sumber 4.30.

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

Kode Sumber 4.30: Perintah untuk *Pull Image mitiproxy*

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 4.31.

```
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 4.31: 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 cleint* 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 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* berbentu *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 4.32.

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

Kode Sumber 4.32: Perintah untuk Membaca *File Log* dari *Mitmproxy*

(Halaman ini sengaja dikosongkan)

BAB V

PENGUJIAN DAN EVALUASI

Pada bab ini akan dibahas uji coba dan evaluasi dari sistem yang telah dibuat. Sistem akan diuji coba fungsionalitas dan performanya dengan menjalankan skenario uji coba yang sudah ditentukan. Uji coba dilakukan untuk mengetahui hasil dari sistem ini sehingga dapat menjawab rumusan masalah pada tugas akhir ini.

5.1 Lingkungan Uji Coba

Uji coba sistem ini dilakukan dengan menggunakan 1 buah komputer sebagai *docker host*, 1 buah komputer sebagai *server* dari halaman *login* sistem, dan 2 komputer sebagai *client*. Semua *client* merupakan virtual komputer menggunakan *VirtualBox*.

1. Komputer sebagai *Docker Host*

Tabel 5.1: Komputer sebagai *Docker Host*

Perangkat Keras	Processor Intel(R) Core(TM) i5-2120 CPU @ 3.30GHz RAM 8GB Hard disk 500GB
Perangkat Lunak	Linux Mint 18.03 64 bit. Docker versi 1.13.1. MySQL versi 5.7.18. Python versi 3.5.2. Flask versi 1.0.2. VIM versi 7.4.1. Iptables versi 1.6.0.
Konfigurasi Jaringan	IP address : 10.151.36.38 Netmask : 255.255.255.0 Gateway : 10.151.36.1 Hostname : X450LD

2. Komputer sebagai *Server* dari Halaman *Login*

Tabel 5.2: Komputer sebagai *Server* dari Halaman *Login*

Perangkat Keras	Processor Intel(R) Core(TM) i5-2120 CPU @ 3.30GHz RAM 1GB Hard disk 20GB
Perangkat Lunak	Ubuntu 16.04 64 bit. Python versi 3.5.2. Flask versi 1.0.2. VIM versi 7.4.1.
Konfigurasi Jaringan	IP address : 10.151.36.130 Netmask : 255.255.255.0 Gateway : 10.151.36.38 Hostname : SERVERLOGIN

3. Komputer sebagai *Client*

(a) *Client 1*

Tabel 5.3: *Client 1*

Perangkat Keras	Processor Intel(R) Core(TM) i5-2120 CPU @ 3.30GHz RAM 1GB Hard disk 20GB
Perangkat Lunak	Linux Ubuntu 16.04 64 bit Firefox Quantum versi 58.0.2.
Konfigurasi Jaringan	IP address : 192.168.99.100 Netmask : 255.255.255.0 Gateway : 192.168.99.1 Hostname : CLIENT1

(b) *Client 2*

Tabel 5.4: *Client 2*

Perangkat Keras	Processor Intel(R) Core(TM) i5-2120 CPU @ 3.30GHz
	RAM 1GB
	Hard disk 20GB
Perangkat Lunak	Linux Ubuntu 16.04 64 bit
	Firefox Quantum versi 58.0.2.
Konfigurasi Jaringan	IP address : 192.168.99.101
	Netmask : 255.255.255.0
	Gateway : 192.168.99.1
	Hostname : CLIENT1

5.2 Skenario Uji Coba

Uji Coba ini dilakukan untuk menguji apakah fungsionalitas yang diidentifikasi pada tahap kebutuhan benar-benar telah diimplementasikan dan bekerja seperti yang seharusnya. Uji coba akan didasarkan pada beberapa skenario untuk menguji kesesuaian respon sistem. Skenario pengujian dibedakan menjadi 2 bagian yaitu:

- **Uji Fungsionalitas** Pengujian ini didasarkan pada kebutuhan sistem yang telah didasarkan pada kebutuhan sistem yang diidentifikasi pada bab 3.
- **Uji Performa** Pengujian ini digunakan untuk mengukur efisiensi sistem dan performa sistem dalam menjalankan fungsionalitas sistem.

5.2.1 Skenario Uji Fungsionalitas

Uji coba fungsionalitas dilakukan dengan cara menjalankan sistem yang telah dibuat, dan melakukan pengujian terhadap fitur yang telah dibuat. Uji coba fungsionalitas akan berfungsi untuk

memastikan sistem sudah memenuhi kebutuhan yang tertera pada Bab 3, yaitu meliputi:

1. Pengujian *client* dapat *login* ke dalam sistem.
2. Pengujian *client* dapat mengirimkan permintaan penyediaan kontainer *docker* ke *docker host*.
3. Pengujian *docker host* dapat menerima perintah penyediaan kontainer *docker*.

5.2.1.1 Uji Coba *Client* dapat *Login* ke Dalam Sistem

Uji coba ini dilakukan oleh *client* dalam keadaan belum *login* ke dalam sistem. Saat *client* mencoba membuka sebuah web http, maka akan langsung diarahkan ke halaman *login* dari sebuah sistem. Saat *client* sudah diarahkan ke halaman *login* sistem, maka *client* harus memasukkan *username* atau NRP mahasiswa dan *password*. Lalu *client* akan mengirim *http requests* kepada *docker host*. Daftar uji fungsionalitas *client* dapat *login* ke dalam sistem dijelaskan pada Tabel 5.5.

Tabel 5.5: Skenario Uji Coba User dapat *Login* ke Dalam Sistem

No	Routes	Uji Coba	Hasil Harapan
1	/login	Mencocokkan hasil <i>input username</i> dan <i>password</i> dari <i>client</i> dengan basis data yang tersedia.	<i>client</i> berhasil <i>login</i> ke dalam sistem.

5.2.1.2 Uji Coba Client dapat Mengirimkan Permintaan Penyediaan Kontainer Docker ke Docker Host

Uji coba ini dilakukan setelah *client* memasukkan *input username* dan *password*, lalu sistem mencocokkannya dan bernilai benar. Setelah itu, *client* akan mengirim *http requests* kepada *docker host*. Daftar uji fungsionalis *client* dapat mengirimkan permintaan penyediaan kontainer *docker* ke *docker host* dijelaskan pada Tabel 5.6.

Tabel 5.6: Skenario Uji Coba User dapat *Login* ke Dalam Sistem

No	Routes	Uji Coba	Hasil Harapan
1	/login	Mengirim request menuju <i>docker host</i> melalui browser.	<i>Request</i> berhasil diterima oleh <i>docker host</i> .

5.2.1.3 Uji Coba Docker Host dapat Menerima Perintah Penyediaan Kontainer Docker

Uji coba ini dilakukan dengan mengakses sistem melalui ruta /tests/endpoint. *Client* akan mengirim *http requests* kepada *docker host*. Lalu sistem akan mengirimkan perintah penyediaan kontainer *docker*. Pada pengujian ini, *image* yang digunakan untuk dipasangkan pada setiap kontainer *docker* adalah *image mitmproxy*, yaitu *image* untuk melakukan *blocking* web maupun menulis *log* untuk semua aktifitas dari *client*. Daftar uji fungsionalitas *docker host* dapat memerlukan perintah penyediaan kontainer *docker* dijelaskan pada Tabel 5.7.

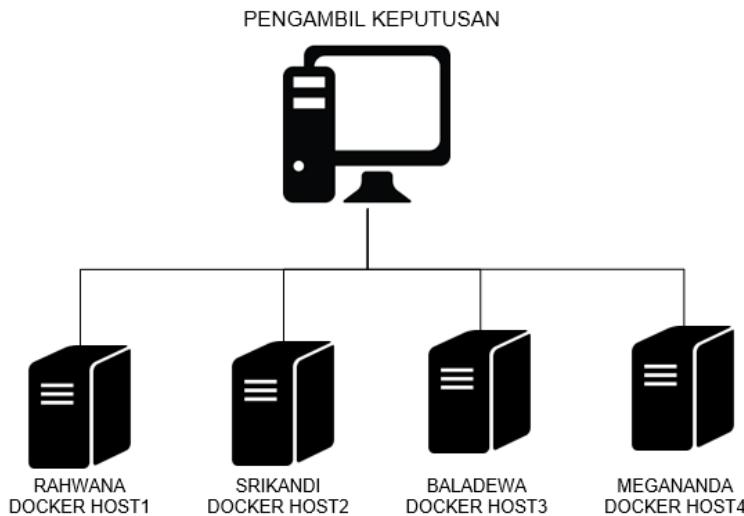
Tabel 5.7: Skenario Uji Coba *Docker Host* dapat Menerima Perintah Penyediaan Kontainer *Docker*

No	Routes	Uji Coba	Hasil Harapan
1	/tests/endpoint	Menerima <i>requests</i> dari <i>client</i> yang menuju ke <i>docker host</i> melalui browser.	<i>Docker Host</i> dapat menerima perintah penyediaan kontainer <i>docker</i> dan menyediakan kontainer <i>docker</i> yang diinginkan.

5.2.2 Skenario Uji Performa

Sistem yang dibuat pada TA ini menggunakan algoritma *analytic hierarchy process* sebagai algoritma pengambilan keputusan untuk menentukan *docker host* terbaik untuk menyediakan kontainer *docker* yang diinginkan. AHP dipakai agar penempatan kontainer pada *docker host* dapat dilakukan secara efisien. Hal ini dilakukan dengan membandingkan ketersediaan sumberdaya pada masing-masing *docker host* dengan bobot priritas masing-masing kriteria sumber daya. Pengujian dilakukan dengan membandingkan hasil kerja sistem yang menggunakan AHP dengan sistem yang menggunakan algortima *round robin* untuk menentukan *docker host* yang akan dipilih untuk menyediakan kontainer. Setelah itu akan dilihat bagaimana performa sistem dengan algoritma AHP yang menggunakan multi kriteria dibandingkan dengan performa sistem yang menggunakan algortima *round robin* yang tidak menggunakan multi kriteria untuk menentukan *docker host* yang akan dipilih. Selain itu pengujian juga akan menggunakan *image docker* variasi yaitu image httpd, nginx, moodle, dan mysql. Image yang bervariasi digunakan agar kebutuhan sumber daya masing-masing *docker host* yang dipakai berbeda-beda untuk

setiap image *docker* yang akan dipasangkan pada setiap *docker host*. Selain itu, Uji Performa juga akan menilai kemampuan sistem dengan algoritma AHP bersarkan jumlah kontainer yang akan dipasangkan. Performa sistem akan di evaluasi secara bertahap dengan melihat pertumbuhan penggunaan sumber daya pada masing-masing *docker host*. Uji coba performa akan menguji efisiensi sistem dalam menempatkan kontainer *docker* yang diinginkan oleh user pada *docker host* yang ada dan ketepatan sistem dalam menempatkan kontainer *docker* yang diinginkan. Arsitektur pengujian tertera pada gambar 5.1



Gambar 5.1: Arsitektur Pengujian Performa

5.3 Hasil Uji Coba dan Evaluasi

Berikut dijelaskan hasil uji coba dan evaluasi berdasarkan skenario yang sudah dijelaskan pada bab 5.2.

5.3.1 Uji Fungsionalitas

Berikut dijelaskan hasil pengujian fungsionalitas pada sistem.

5.3.1.1 Uji Coba User Mengirim Permintaan Penyediaan Kontainer

Uji coba ini dilakukan dengan mengakses sistem melalui rute dan parameter yang telah ditentukan pada Tabel ???. Pengguna akan mengirim *http request* kepada web service yang telah disediakan pada Komputer Penerima Permintaan Penyediaan Kontainer. Hasil uji coba seperti tertera pada tabel 5.8.

Tabel 5.8: Hasil Skenario Uji Coba mengirim permintaan penyediaan kontainer

No	Routes	Uji Coba	Hasil
1	/data/httpd	Mengirim request menuju rute web service melalui browser.	OK.

5.3.1.2 Uji Coba Sistem dapat Melakukan Penghitungan AHP

Uji coba ini dilakukan dengan mengakses sistem melalui rute yang telah ditentukan pada Tabel ???. Pengguna akan mengirim *http request* kepada web service yang telah disediakan pada Komputer Penerima Permintaan Penyediaan Kontainer dan sistem akan melakukan penghitungan AHP terhadap data sumber daya setiap *docker host* yang ada sebagai acuan. Hasil uji coba seperti tertera pada Tabel 5.9 dan 5.10.

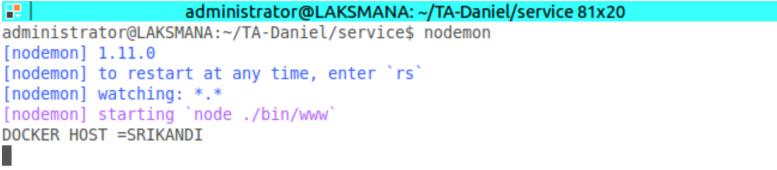
Tabel 5.9: Kondisi Awal Penggunaan Sumberdaya *Docker Host* sebelum uji coba dijalankan

No	Docker Host	CPU	RAM	Storage
1	RAHWANA	0.2%	797M/1.9G	11G/16G
2	SRIKANDI	0.3%	532M/2.9G	8.8G/50G
3	BALADEWA	0.1%	354M/1.9G/	37G/57G
4	MEGANANDA	0.1%	555M/1.9G	10G/16G

Tabel 5.10: Hasil Skenario Uji Coba Sistem dapat melakukan Penghitungan AHP

No	Docker Host	Uji Coba	Hasil
1	/data/httpd	Mengirim request menuju rute web service melalui browser.	web service berhasil melakukan AHP dan menghasilkan hostname dari <i>docker host</i> terpilih, yaitu dalam uji coba ini menghasilkan hostname SRIKANDI.

Pada Uji Coba berhasil didapatkan hostname dari *Docker Host* terbaik, yaitu SRIKANDI seperti yang ditunjukkan pada Gambar 5.2, dikarenakan SRIKANDI memiliki ketersediaan sumber daya yang lebih baik dibanding *Docker Host* lain berdasarkan penghitungan dengan menggunakan algoritma AHP.



```
administrator@LAKSMANA:~/TA-Daniel/service 81x20
administrator@LAKSMANA:~/TA-Daniel/service$ nodemon
[nodemon] 1.11.0
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: `.*`
[nodemon] starting `node ./bin/www`
DOCKER HOST =SRIKANDI
```

Gambar 5.2: Gambar Hasil Uji Sistem dapat melakukan Penghitungan AHP

5.3.1.3 Uji Coba *Docker Host* dapat Menerima Perintah Penyediaan Kontainer

Uji coba ini dilakukan dengan mengakses sistem melalui rute, parameter yang telah ditentukan pada Tabel ???. Pengujian dilakukan image httpd yang akan diujikan pada setiap *docker host*. Hasil uji coba ditunjukkan pada Tabel 5.11

Tabel 5.11: Hasil Skenario Uji Coba *Docker Host* dapat menerima perintah penyediaan kontainer

No	Route	Docker Host	Uji Coba	Hasil
1	/data/httpd	<i>Docker Host</i> 1	Mengirim request penyediaan <i>docker</i> dengan image httpd menuju rute web service melalui browser.	<i>Docker Host</i> berhasil menerima perintah penyediaan kontainer dan menyediakan kontainer <i>docker</i> dengan image httpd.

Tabel 5.11: Hasil Skenario Uji Coba *Docker Host* dapat menerima perintah penyediaan kontainer

No	Route	Docker Host	Uji Coba	Hasil
2	/data/httpd	<i>Docker Host 2</i>	Mengirim request penyediaan <i>docker</i> dengan image httpd menuju rute web service melalui browser.	<i>Docker Host</i> berhasil menerima perintah penyediaan kontainer dan menyediakan kontainer <i>docker</i> dengan image httpd.
3	/data/httpd	<i>Docker Host 3</i>	Mengirim request penyediaan <i>docker</i> dengan image httpd menuju rute web service melalui browser.	<i>Docker Host</i> berhasil menerima perintah penyediaan kontainer dan menyediakan kontainer <i>docker</i> dengan image httpd.

Tabel 5.11: Hasil Skenario Uji Coba *Docker Host* dapat menerima perintah penyediaan kontainer

No	Route	Docker Host	Uji Coba	Hasil
4	/data/httpd	<i>Docker Host 4</i>	Mengirim request penyediaan <i>docker</i> dengan image httpd menuju rute web service melalui browser.	<i>Docker Host</i> berhasil menerima perintah penyediaan kontainer dan menyediakan kontainer <i>docker</i> dengan image httpd.

Pada Gambar 5.3 dan Gambar 5.4 ditunjukkan bagaimana kondisi saat sistem menghasilkan perintah untuk menyediakan *docker* dan saat *docker* sudah terpasang pada *docker host*



```
administrator@LAKSMANA:~/TA-Daniel/service 81x20
administrator@LAKSMANA:~/TA-Daniel/service$ nodemon
[nodemon] 1.11.0
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: ***!
[nodemon] starting 'node ./bin/www'
DOCKER HOST =SRIKANDI
ansible-playbook /home/administrator/TA-Daniel/ansible-test/playbooks/apache.yml
--extra-vars "image=httpd port=8080 host=SRIKANDI name=container1"
GET /data/httpd 200 13970.506 ms - 457
GET /favicon.ico 404 696.841 ms - 1834
GET /favicon.ico 404 43.313 ms - 1834
```

Gambar 5.3: Web Service Mengrimkan perintah penyediaan kontainer

```
administrator@SRIKANDI:~ 80x20
CONTAINER ID        IMAGE       COMMAND      CREATED          STATUS          PORTS
9e168894c96e        httpd      "httpd-foreground"   28 seconds ago
Up 23 seconds      0.0.0.0:8080->80/tcp    container1
administrator@SRIKANDI:~
```

Gambar 5.4: Docker Host berhasil membuat kontainer docker

5.3.1.4 Uji Coba Docker Host dapat Mengirim Data Resourcanya Masing-Masing

Dilakukan pengujian pada setiap *docker host*. Setiap *docker host* akan mengirimkan data ketersediaan sumber daya CPU, RAM dan File Storage nya masing-masing ke InfluxDB yang terdapat pada *middleware*. Hasil ditunjukkan pada Tabel 5.12

Tabel 5.12: Hasil Skenario Uji Coba Docker Host dapat Mengirim Data Resourcanya Masing-Masing

No	Docker Host	Uji Coba	Hasil
1	<i>Docker Host 1</i>	Mengirim data sumber daya (RAM, CPU dan File Storage) ke InfluxDB pada <i>middleware</i> .	Data sumber daya CPU, RAM dan File Storage berhasil dikirimkan dan tersimpan pada InfluxDB.

Tabel 5.12: Hasil Skenario uji Coba *Docker Host* dapat Mengirim Data Resource

No	Docker Host	Uji Coba	Hasil
2	<i>Docker Host 2</i>	Mengirim data sumber daya (RAM, CPU dan <i>File Storage</i>) ke InfluxDB pada <i>middleware</i> .	Data sumber daya CPU, RAM dan File Storage berhasil dikirimkan dan tersimpan pada InfluxDB.
3	<i>Docker Host 3</i>	Mengirim data sumber daya (RAM, CPU dan <i>File Storage</i>) ke InfluxDB pada <i>middleware</i> .	Data sumber daya CPU, RAM dan File Storage berhasil dikirimkan dan tersimpan pada InfluxDB.
4	<i>Docker Host 4</i>	Mengirim data sumber daya (RAM, CPU dan <i>File Storage</i>) ke InfluxDB pada <i>middleware</i> .	Data sumber daya CPU, RAM dan File Storage berhasil dikirimkan dan tersimpan pada InfluxDB.

Sesuai dengan Gambar 5.5,5.6 dan 5.7, ,semua *docker host* berhasil mengirimkan data sumber daya CPU, RAM dan File Storage nya masing-masing ke InfluxDB.

- Data CPU Setiap Docker Host pada InfluxDB

```

key
cpu_value,host=BALADEWA,instance=0,type=percent,type_instance=idle
cpu_value,host=BALADEWA,instance=1,type=percent,type_instance=idle
cpu_value,host=MEGANANDA,instance=0,type=percent,type_instance=idle
cpu_value,host=RAHWANA,instance=0,type=percent,type_instance=idle
cpu_value,host=RAHWANA,instance=1,type=percent,type_instance=idle
cpu_value,host=SRIKANDI,instance=0,type=percent,type_instance=idle
cpu_value,host=SRIKANDI,instance=1,type=percent,type_instance=idle

```

Gambar 5.5: Data CPU Setiap Docker Host pada InfluxDB

- Data RAM Setiap Docker Host pada InfluxDB

```

memory_value,host=BALADEWA,type=memory,type_instance=free
memory_value,host=BALADEWA,type=percent,type_instance=free
memory_value,host=MEGANANDA,type=memory,type_instance=free
memory_value,host=MEGANANDA,type=percent,type_instance=free
memory_value,host=RAHWANA,type=memory,type_instance=free
memory_value,host=RAHWANA,type=percent,type_instance=free
memory_value,host=SRIKANDI,type=memory,type_instance=free
memory_value,host=SRIKANDI,type=percent,type_instance=free

```

Gambar 5.6: Data RAM Setiap Docker Host pada InfluxDB

- Data Penyimpanan File Setiap Docker Host pada InfluxDB

```

df_value,host=BALADEWA,instance=root,type=df_complex,type_instance=free
df_value,host=BALADEWA,instance=root,type=percent_bytes,type_instance=free
df_value,host=MEGANANDA,instance=root,type=df_complex,type_instance=free
df_value,host=MEGANANDA,instance=root,type=percent_bytes,type_instance=free
df_value,host=RAHWANA,instance=root,type=df_complex,type_instance=free
df_value,host=RAHWANA,instance=root,type=percent_bytes,type_instance=free
df_value,host=SRIKANDI,instance=root,type=df_complex,type_instance=free
df_value,host=SRIKANDI,instance=root,type=percent_bytes,type_instance=free

```

Gambar 5.7: Data Penyimpanan File Setiap Docker Host pada InfluxDB

5.3.2 Uji Performa

Seperti yang dijelaskan pada bab 5.2 pengujian performa akan dilakukan pada 4 *docker host* yang tersedia dengan menggunakan sistem yang ada, namun dengan 2 algoritma yang berbeda yaitu menggunakan algoritma pengambilan keputusan AHP dan menggunakan algoritma pembagian kerja dasar, *round robin*[?]. Selain itu pengujian juga akan menggunakan beberapa

image *docker* yang akan di pasangkan pada setiap *docker host* yang tersedia. Pengujian akan dijalankan dengan mengirimkan permintaan penyediaan kontainer pada sistem dengan jumlah permintaan penyediaan kontainer yang beragam.

5.3.2.1 Pengujian Dengan Algoritma AHP

Kondisi awal ketersediaan sumberdaya *docker host* sebelum uji coba sistem dengan image *docker* httpd dan nginx menggunakan AHP dijalankan ditunjukkan pada Tabel 5.13.

Tabel 5.13: Kondisi Awal Ketersediaan Sumberdaya *Docker Host* sebelum uji coba dijalankan

No	Docker Host	CPU	RAM	Storage
1	RAHWANA	98%	1.3G/1.9G	5G/16G
2	SRIKANDI	93%	2.3G/2.9G	41.2G/50G
3	BALADEWA	99%	1G/1.9G	20G/57G
4	MEGANANDA	91%	1.3G/1.9G	6G/16G

Hasil pengujian sistem dengan algoritma AHP menggunakan image *docker* httpd dan nginx ditunjukkan pada Tabel 5.14.

Tabel 5.14: Hasil Uji Coba Performa sistem dengan *Image Docker* httpd dan nginx menggunakan AHP

No	Jumlah Kontainer	Kontainer Terpasang pada <i>Docker Host</i>			
		Rahwana	Srikandi	Baladewa	Megananda
1	254	39	135	27	53
2	254	40	131	32	51
3	254	37	147	20	50
4	254	41	143	23	47
5	254	49	132	24	49

Kondisi akhir ketersediaan sumber daya *docker host* pada

pengujian sistem dengan algoritma AHP menggunakan *image docker* httpd dan nginx ditunjukkan pada Tabel 5.15.

Tabel 5.15: Rata-rata Kondisi Akhir Ketersediaan Sumberdaya Docker Host Setelah Uji Coba Dijalankan

No	Docker Host	CPU	RAM	Storage
1	RAHWANA	90%	628M/1.9G	5G/16G
2	SRIKANDI	85%	708M/2.9G	41.2G/50G
3	BALADEWA	92%	832M/1.9G	20G/57G
4	MEGANANDA	90%	654M/1.9G	6G/16G

5.3.2.2 Pengujian Dengan Algoritma *Round Robin*

Kondisi awal ketersediaan sumberdaya *docker host* sebelum uji coba sistem dengan *image docker* httpd dan nginx menggunakan *round robin* dijalankan ditunjukkan pada Tabel 5.16.

Tabel 5.16: Kondisi Awal Ketersediaan Sumberdaya Docker Host Sebelum Uji Coba Dijalankan

No	Docker Host	CPU	RAM	Storage
1	RAHWANA	98%	1.2G/1.9G	5G/16G
2	SRIKANDI	93%	2.3G/2.9G	41.2G/50G
3	BALADEWA	99%	1G/1.9G	20G/57G
4	MEGANANDA	91%	1.4G/1.9G	6G/16G

Hasil dari pengujian sistem dengan algoritma *round robin* ditunjukkan pada Tabel 5.17.

Tabel 5.17: Hasil Uji Coba Performa Sistem dengan *Image Docker* httpd dan nginx Menggunakan *Round Robin*

No	Jumlah Kontainer	Kontainer Terpasang pada Docker Host			
		Rahwana	Srikandi	Baladewa	Megananda

1	254	63	63	63	63
2	254	63	63	63	63
3	254	63	63	63	63
4	254	63	63	63	63
5	254	63	63	63	63

Kondisi akhir ketersediaan sumberdaya *docker host* pada pengujian sistem dengan algoritma *round robin* menggunakan *image docker* httpd dan nginx ditunjukkan pada Tabel 5.18.

Tabel 5.18: Kondisi Akhir Ketersediaan Sumberdaya *Docker Host* Setelah Uji Coba Dijalankan

No	Docker Host	CPU	RAM	Storage
1	RAHWANA	92%	161M/1.9G	5G/16G
2	SRIKANDI	93%	1.3G/2.9G	41.1G/50G
3	BALADEWA	92%	95M/1.9G	20G/57G
4	MEGANANDA	91%	335M/1.9G	6G/16G

5.3.2.3 Pengujian Performa Sistem dengan Algoritma AHP Berdasarkan Jumlah Kontainer yang Akan Dipasangkan

Kondisi awal ketersediaan sumberdaya *docker host* sebelum uji coba ditunjukkan pada Tabel 5.19.

Tabel 5.19: Kondisi Awal Ketersediaan Sumberdaya *Docker Host* Sebelum Uji Coba Dijalankan

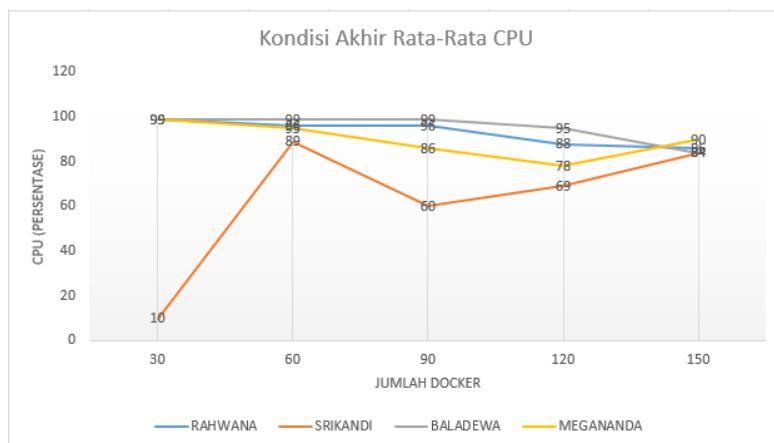
No	Docker Host	CPU	RAM	Storage
1	RAHWANA	99%	1.3G/1.9G	3.8G/16G
2	SRIKANDI	99%	2.3G/2.9G	37G/50G
3	BALADEWA	99%	1G/1.9G	16G/57G
4	MEGANANDA	99%	1.4G/1.9G	3.8G/16G

Hasil pengujian sistem dengan algoritma AHP menggunakan *docker image* httpd, nginx, moodle, dan mysql ditunjukkan pada Table 5.20.

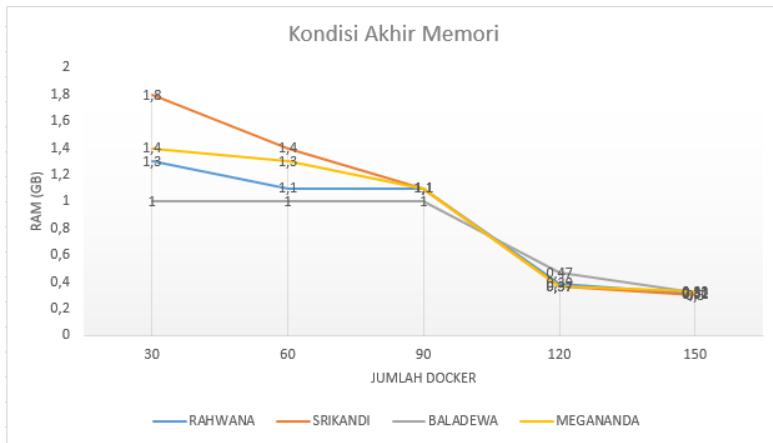
Tabel 5.20: Hasil Uji Coba Performa Sistem dengan *Docker Image* httpd dan nginx Menggunakan AHP

No	Jumlah Kontainer	Kontainer Terpasang pada Docker Host			
		Rahwana	Srikandi	Baladewa	Megananda
1	30	0	30	0	0
2	60	3	55	0	2
3	100	5	73	0	12
4	120	11	88	5	16
5	150	34	46	33	37

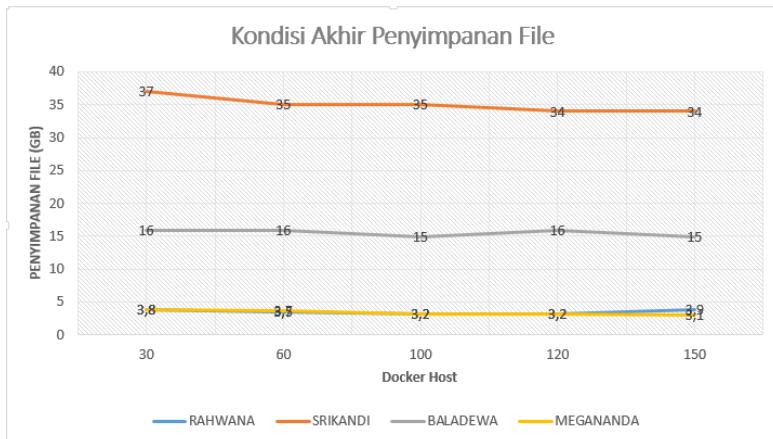
Kondisi akhir ketersediaan sumberdaya *docker host* setelah uji coba dijalankan ditunjukkan pada Gambar 5.8 untuk ketersediaan CPU, Gambar 5.9 untuk ketersediaan Memori dan Gambar 5.10 untuk ketersediaan penyimpanan berkas.



Gambar 5.8: Grafik Kondisi Akhir Ketersediaan Rata-Rata CPU



Gambar 5.9: Grafik Kondisi Akhir Ketersediaan Memori



Gambar 5.10: Grafik Kondisi Akhir Ketersedian Penyimpanan File

Waktu Pendistribusian Docker oleh Sistem Berdasarkan Jumlah Kontainer ditunjukkan pada Gambar 5.11



Gambar 5.11: Grafik Waktu Pendistribusian Docker oleh Sistem Berdasarkan Jumlah Kontainer

Dari data hasil uji coba yang didapat, dapat dilihat bahwa algoritma AHP dapat membagikan kontainer ke *docker host* yang ada dengan lebih efisien. Dibandingkan dengan algoritma *round robin* yang menyebarluaskan kontainer merata ke setiap *docker host*, sistem dengan AHP menghasilkan kondisi ketersediaan sumber daya yang lebih merata pada akhir uji coba. Selain itu pendistribusian pada setiap *docker host* juga memiliki ketepatan yang baik, hal ini terlihat pada saat akan mendistribusikan 30 kontainer, sistem dengan AHP mendistribusikan seluruh kontainer tersebut pada *docker host* SRIKANDI, hal ini disebabkan SRIKANDI memiliki ketersediaan yang jauh lebih dominan dibanding *docker host* yang lain. Namun, kondisi dimana *docker image* yang bervariasi menimbulkan hasil akhir dari uji coba tidak linear, hal ini disebabkan variasi dari *docker image* akan membuat kebutuhan sumber daya yang dibutuhkan dari setiap *docker host* juga bervariasi.

(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 mendistribusikan 100% penyediaan kontainer ke *docker host* yang ada dengan multi kriteria dengan cara mendistribusikan perintah penyediaan kontainer *docker* melalui protokol ssh.
2. Sistem dapat menentukan penyedia kontainer dengan menggunakan algoritma AHP dengan kriteria-kriteria seperti penggunaan CPU, RAM , dan Penyimpanan File pada masing-masing *docker host* yang ada.
3. Dengan menggunakan AHP, dibandingkan dengan metode *round robin*, sistem dapat menentukan *docker host* yang akan menyediakan kontainer docker dengan efisien berdasarkan penggunaan CPU, RAM , dan File Storage pada masing-masing *docker host* yang ada, dimana dengan menggunakan AHP ketersediaan akhir memori dari setiap *docker host* lebih merata dengan rentang 628MB sampai 832MB, sedangkan dengan *round robin* ketersediaan akhir memori sangat tidak merata, dengan rentang yang cukup jauh dimulai 95MB sampai 1.3GB.

6.2 Saran

Berikut beberapa saran yang diberikan untuk pengembangan lebih lanjut:

- Sistem dapat dikembangkan dengan menambahkan kriteria-kriteria yang sesuai dengan lingkungan sistem yang ada, seperti jarak antara *docker host* dan *server* middleware atau kecepatan bandwith dari setiap *docker host* merupakan kriteria yang lebih baik untuk sistem yang memasangkan kontainer yang tidak memiliki perkembangan penggunaan pada aplikasi yang terdapat pada kontainer. Sedangkan sumber daya seperti file storage dapat digunakan untuk sistem yang mengalami perkembangan penggunaan pada aplikasi yang terdapat pada kontainernya.
- AHP merupakan algoritma MCDM yang paling umum digunakan dikarenakan proses yang tidak rumit dan memiliki unsur objektif dan subjektif dalam pengambilan keputusannya. Untuk pengembangan kedepannya sistem dapat diimplementasikan dengan menggunakan algortima MCDM lain untuk meningkatkan performa sistem, seperti algoritma Fuzzy AHP.

DAFTAR PUSTAKA

(Halaman ini sengaja dikosongkan)

LAMPIRAN A

FILE KONFIGURASI

1.1 File Konfigurasi Collectd

Berikut File Konfigurasi Keseluruhan untuk perangkat lunak Collectd.

```
FQDNLookup true
LoadPlugin syslog

<Plugin syslog>
  LogLevel info
</Plugin>

LoadPlugin cpu
LoadPlugin df
LoadPlugin memory
LoadPlugin network

<Plugin cpu>
  ReportByCpu true
  ReportByState true
  ValuesPercentage true
</Plugin>

<Plugin memory>
  ValuesAbsolute true
  ValuesPercentage true
</Plugin>

<Plugin df>
  Device "/dev/sda6"
  MountPoint "/"
  FSType "ext4"

  IgnoreSelected false

  ReportInodes false

  ValuesAbsolute true
  ValuesPercentage true
</Plugin>

<Plugin network>
  Server "10.151.36.37" "25826"
</Plugin>
```

```
<Include "/etc/collectd/collectd.conf.d">
  Filter "*.*.conf"
</Include>
```

Kode Sumber 1.1: Kode sumber Model Auth

1.2 File Konfigurasi InfluxDB

Berikut File Konfigurasi Keseluruhan untuk perangkat lunak InfluxDB.

```
reporting-disabled = false

[meta]
  # Where the metadata/raft database is stored
  dir = "/var/lib/influxdb/meta"

  retention-autocreate = true

  # If log messages are printed for the meta service
  logging-enabled = true
  pprof-enabled = false

  # The default duration for leases.
  lease-duration = "1m0s"

[data]
  # Controls if this node holds time series data shards in the
  # cluster
  enabled = true

  dir = "/var/lib/influxdb/data"

  # These are the WAL settings for the storage engine >= 0.9.3
  wal-dir = "/var/lib/influxdb/wal"
  wal-logging-enabled = true
  data-logging-enabled = true

### [cluster]
### Controls non-Raft cluster behavior, which generally includes
### how data is
```

```
### shared across shards.  
###  
  
[cluster]  
shard-writer-timeout = "5s" # The time within which a remote  
    shard must respond to a write request.  
write-timeout = "10s" # The time within which a write request  
    must complete on the cluster.  
max-concurrent-queries = 0 # The maximum number of concurrent  
    queries that can run. 0 to disable.  
query-timeout = "0s" # The time within a query must complete  
    before being killed automatically. 0s to disable.  
max-select-point = 0 # The maximum number of points to scan in a  
    query. 0 to disable.  
max-select-series = 0 # The maximum number of series to select in  
    a query. 0 to disable.  
max-select-buckets = 0 # The maximum number of buckets to select  
    in an aggregate query. 0 to disable.  
  
###  
### [retention]  
###  
### Controls the enforcement of retention policies for evicting old  
    data.  
###  
  
[retention]  
enabled = true  
check-interval = "30m"  
  
###  
### [shard-precreation]  
###  
### Controls the precreation of shards, so they are available  
    before data arrives.  
### Only shards that, after creation, will have both a start- and  
    end-time in the  
### future, will ever be created. Shards are never precreated that  
    would be wholly  
### or partially in the past.  
  
[shard-precreation]  
enabled = true  
check-interval = "10m"  
advance-period = "30m"  
  
###
```

```
### Controls the system self-monitoring, statistics and diagnostics
.

### The internal database for monitoring data is created
    automatically if
### if it does not already exist. The target retention within this
    database
### is called 'monitor' and is also created with a retention period
    of 7 days
### and a replication factor of 1, if it does not exist. In all
    cases the
### this retention policy is configured as the default for the
    database.

[monitor]
  store-enabled = true # Whether to record statistics internally.
  store-database = "_internal" # The destination database for
      recorded statistics
  store-interval = "10s" # The interval at which to record
      statistics

### [admin]
### Controls the availability of the built-in, web-based admin
    interface. If HTTPS is
### enabled for the admin interface, HTTPS must also be enabled on
    the [http] service.
###

[admin]
  enabled = true
  bind-address = ":8083"
  https-enabled = false
  https-certificate = "/etc/ssl/influxdb.pem"

### [http]
### Controls how the HTTP endpoints are configured. These are the
    primary
### mechanism for getting data into and out of InfluxDB.
###

[http]
  enabled = true
  bind-address = ":8086"
```

```
auth-enabled = false
log-enabled = true
write-tracing = false
pprof-enabled = false
https-enabled = false
https-certificate = "/etc/ssl/influxdb.pem"
max-row-limit = 10000

#####
###  [[graphite]]
#####
### Controls one or many listeners for Graphite data.
###

[[graphite]]
enabled = false

#####
###  [collectd]
#####
### Controls one or many listeners for collectd data.
###

[[collectd]]
enabled = true
bind-address = ":25826"
database = "collectd"
retention-policy = ""
# These next lines control how batching works. You should have
# this enabled
# otherwise you could get dropped metrics or poor performance.
# Batching
# will buffer points in memory if you have many coming in.

batch-size = 5000 # will flush if this many points get buffered
batch-pending = 10 # number of batches that may be pending in
# memory
batch-timeout = "10s" # will flush at least this often even if
# we haven't hit buffer limit
read-buffer = 0 # UDP Read buffer size, 0 means OS default. UDP
# listener will fail if set above OS max.
typesdb = "/usr/share/collectd/types.db"

#####
###  [opentsdb]
###
```

```

### Controls one or many listeners for OpenTSDB data.
###

[[opentsdb]]
enabled = false

###

### [[udp]]
###

### Controls the listeners for InfluxDB line protocol data via UDP.
###

[[udp]]
enabled = false

###

### [continuous_queries]
###

### Controls how continuous queries are run within InfluxDB.
###

[continuous_queries]
log-enabled = true
enabled = true
# run-interval = "1s" # interval for how often continuous queries
# will be checked if they need to run

```

Kode Sumber 1.2: Kode sumber Model Auth

1.3 File Inventory Ansible

Berikut File Inventory dari perangkat lunak Ansible.

```

[worker]
RAHWANA ansible_user=administrator
SRIKANDI ansible_user=administrator
BALADEWA ansible_user=administrator
MEGANANDA ansible_user=administrator

```

Kode Sumber 1.3: Kode sumber Model Auth

LAMPIRAN B

KODE SUMBER

2.1 Web Service

Berikut File Inventory dari perangkat lunak Ansible.

```
var express = require('express');
var mysql = require('mysql');
var Influx = require('influx');
var router = express.Router();
var conn = require('../db-mysql.js');
var Servers=require('../models/Servers');
var math = require('mathjs');
var cmd = require('node-cmd');
var Promise = require('bluebird');

var flag = 1;
//apache and nginx
var port = 8080;

var influx = new Influx.InfluxDB({
  host: '10.151.36.37',
  database: 'collectd'
})

function indexOfMax(arr) {
  if (arr.length === 0) {
    return -1;
  }

  var max = arr[0];
  var maxIndex = 0;

  for (var i = 1; i < arr.length; i++) {
    if (arr[i] > max) {
      maxIndex = i;
      max = arr[i];
    }
  }

  return maxIndex;
}

router.get('/', function(req,res){
  var query="SELECT * FROM servers";
  var resources=[];
  var counter=0
```

```

var df=0
var cpu=0
conn.query(query, function(err, rows, fields) {
  if(err) {
    return res.json({"Error" : true, err});
  } else {
    servers=rows.length;
    for (var i = 0; i < servers; i++) {
      output= {
        'hostname' : '',
        'memory' : '',
        'cpu' : '',
        'df' : ''
      }
      resources.push(output);
    Promise.all([
      influx.query(`select last("value") from memory_value
                    where type='percent' and type_instance='free' and
                    time > now() - 24h and host=${Influx.escape.
                    stringLit(rows[i].hostname)} group by host`),
      influx.query(`select mean("value") from cpu_value where
                    type='percent' and type_instance='idle' and time >
                    now() - 24h and host=${Influx.escape.stringLit(rows[
                    i].hostname)} group by host`),
      influx.query(`select last("value") from df_value where
                    type='percent_bytes' and type_instance='free' and
                    time > now() - 24h and host=${Influx.escape.
                    stringLit(rows[i].hostname)} group by host`)
    ]).spread(function(query1,query2,query3){
      // console.log(query3)
      resources[counter].hostname=query1[0].host
      resources[counter].memory=query1[0].mean
      resources[counter].cpu=query2[0].mean
      resources[counter].df=query3[0].last
      counter++
      if(counter==servers){
        //AHP
        //Comparison Matrix
        // [       CPU   MEM   DF ]
        // [CPU    1     0.25  0.5]
        // [MEM    4     1     5  ]
        // [DF     2     0.2   1  ]
        var CM = math.matrix([[1, 0.25, 0.5, 0, 0], [4, 1,
          5, 0, 0], [2, 0.2, 1, 0, 0]]);
        //get 3rd root of Comparison Matrix
        CM.subset(math.index(0, 3),math.pow((CM.subset(math
          .index(0, 0))*CM.subset(math.index(0, 1)))*CM.

```

```

        subset(math.index(0, 2))), 1/3));
CM.subset(math.index(1, 3),math.pow((CM.subset(math
    .index(1, 0))*CM.subset(math.index(1, 1))*CM.
    subset(math.index(1, 2))), 1/3));
CM.subset(math.index(2, 3),math.pow((CM.subset(math
    .index(2, 0))*CM.subset(math.index(2, 1))*CM.
    subset(math.index(2, 2))), 1/3));
//get priority vector of Comparison Matrix
var sum = CM.subset(math.index(0, 3))+CM.subset(
    math.index(1, 3))+CM.subset(math.index(2, 3));
CM.subset(math.index(0, 4),CM.subset(math.index(0,
    3)) / sum)
CM.subset(math.index(1, 4),CM.subset(math.index(1,
    3)) / sum)
CM.subset(math.index(2, 4),CM.subset(math.index(2,
    3)) / sum)

//getting rate of each server
var range = servers + 2;
var mem = math.ones(servers,range)
var cpu = math.ones(servers,range)
var df = math.ones(servers,range)
for(i=0;i < servers; i++) {
    for (j = 0; j < servers; j++) {
        mem.subset(math.index(i, j), resources[i].
            memory/resources[j].memory)
        cpu.subset(math.index(i, j), resources[i].cpu/
            resources[j].cpu)
        df.subset(math.index(i, j), resources[i].df/
            resources[j].df)
    }
}

var priority = []
for (var i = 0; i < servers; i++) {
    priority.push({
        memory:'',
        cpu:'',
        df:''
    });
}

//get 3rd root of product(rop) and priority of MEM
var rop_mem=[];
var ropsum=0;
for(i=0;i < servers; i++){
    var ropmval=1;

```

```

        for (j = 0; j < servers; j++) {
            ropmval = ropmval * mem.subset(math.index(i,
                j))
            if(j==servers-1){
                // console.log(ropmval)
                rop_mem[i]=math.pow(ropmval,1/3);
                ropmsum = ropmsum + rop_mem[i];
            }
        }
        if(i==servers-1)
        var pmval=1;
        for(k=0;k < rop_mem.length; k++) {
            pmval = rop_mem[k]/ropmsum;
            priority[k].memory=pmval;
        }
    }

    //get 3rd root of product(rop) and priority of DF
    var rop_df=[];
    var ropdsum=0;
    for(i=0;i < servers; i++){
        var ropdval=1;
        for (j = 0; j < servers; j++) {
            ropdval = ropdval * df.subset(math.index(i, j
                ))
            if(j==servers-1){
                // console.log(ropmval)
                rop_df[i]=math.pow(ropdval,1/3);
                ropdsum = ropdsum + rop_df[i];
            }
        }
        if(i==servers-1)
        var pdval=1;
        for(k=0;k < rop_df.length; k++) {
            pdval = rop_df[k]/ropdsum;
            priority[k].df=pdval;
        }
    }

    //get 3rd root of product(rop) and priority of CPU
    var rop_cpu=[];
    var ropcsum=0;
    for(i=0;i < servers; i++){
        var ropcval=1;
        for (j = 0; j < servers; j++) {
            ropcval = ropcval * cpu.subset(math.index(i,
                j))
        }
    }
}

```

```

        if(j==servers-1){
            // console.log(ropmval)
            rop_cpu[i]=math.pow(ropcval,1/3);
            ropcsum = ropcsum + rop_cpu[i];
        }
    }
    if(i==servers-1)
        var pcval=1;
    for(k=0;k < rop_cpu.length; k++){
        pcval = rop_cpu[k]/ropcsum;
        priority[k].cpu=pcval;
    }
}
//getting score => SIGMA(priority per criteria *
// weight of node for each criteria)
var criteria = 3
var score=[];
for (var i = 0; i < servers; i++) {
    score[i] = CM.subset(math.index(0, 4))*priority
                [i].cpu + CM.subset(math.index(1, 4))*priority[i].memory + CM.subset(math.index
                (2, 4))*priority[i].df
}
var decision = indexOfMax(score);

command = 'ansible-playbook /home/administrator/TA-
Daniel/ansible-test/playbooks/apache.yml --
extra-vars "port='+port+' host='+resources[
decision].hostname+' name=container'+flag+'";'
flag++;
port++;
cmd.get(
    command,
    function(err, data, stderr){
        res.json(data)
    }
);
})
);
}
});
module.exports=router;

```

Kode Sumber 2.1: Kode sumber Model Auth

(Halaman ini sengaja dikosongkan)

BIODATA PENULIS



Fourir Akbar, 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 sebanyak dua semester dan pernah menjadi asisten dosen pada mata kuliah sistem terdistribusi sebanyak satu semester. Penulis juga pernah menjadi asisten dosen pendidikan informatika dan komputer terapan (PIKTI) ITS sebanyak empat semester. 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 ke-2. Penulis juga aktif dalam kepanitiaan Schematics, antara lain sebagai Staff Biro Revolutionary Entertainment and Expo with Various Arts pada tahun ke-2 dan menjadi Badan Pengurus Harian (BPH) Biro Perlengkapan dan Transportasi pada tahun ke-3. Penulis juga merupakan salah satu administrator aktif pada Laboratorium Arsitektur dan jaringan Komputer di Departemen Informatika ITS.