

# **IMPLEMENTASI *FAIL OVER* BERDASARKAN PERBANDINGAN WAKTU DAN *LOAD BALANCING* SERVER WEB BERBASIS DOCKER SWARM BERDASARKAN PENGGUNAAN SUMBER DAYA *MEMORY HOST***

SKRIPSI

Untuk memenuhi sebagian persyaratan  
memperoleh gelar Sarjana Komputer

Disusun oleh:

Mohamad Rexa Mei Bella

NIM: 145150200111190



PROGRAM STUDI TEKNIK INFORMATIKA  
JURUSAN TEKNIK INFORMATIKA  
FAKULTAS ILMU KOMPUTER  
UNIVERSITAS BRAWIJAYA  
MALANG  
2018

## DAFTAR ISI

DAFTAR ISI.....	ii
DAFTAR TABEL.....	v
DAFTAR GAMBAR.....	vi
BAB 1 Pendahuluan.....	1
1.1 Latar belakang .....	1
1.2 Rumusan masalah .....	2
1.3 Tujuan.....	2
1.4 Manfaat .....	2
1.5 Batasan masalah .....	3
1.6 Sistematika Pembahasan .....	3
BAB 2 LANDASAN KEPUSTAKAAN .....	5
2.1 Kajian Pustaka .....	5
2.2 Docker .....	6
2.3 Docker Swarm .....	6
2.4 Web Server NGIX.....	7
2.5 JSON .....	7
2.6 Load Balancing .....	8
2.6.1 <i>Fail Over</i> Berdasarkan Perbandingan Waktu .....	8
2.6.2 <i>Load Balancing</i> Berbasis Sumber Daya <i>Memory</i> .....	9
BAB 3 METODOLOGI PENELITIAN .....	10
3.1 Studi Literatur.....	11
3.2 Analisis Kebutuhan .....	11
3.2.1 Kebutuhan Fungsional .....	11
3.2.2 Kebutuhan Non Fungsional .....	11
3.2.2.1 Kebutuhan Perangkat Keras .....	11
3.2.2.2 Kebutuhan Perangkat Lunak .....	12
3.3 Perancangan Sistem .....	12
3.4 Implementasi Sistem.....	13
3.5 Pengujian dan Analisis Sistem.....	13
3.6 Kesimpulan dan Saran.....	13
BAB 4 PERANCANGAN.....	14
4.1 Perancangan.....	14

4.1.1 Topologi .....	14
4.1.2 <i>Fail Over</i> Berdasarkan Perbandingan Waktu .....	15
4.1.3 <i>Load Balancing</i> Berdasarkan Sumber Daya Memory .....	15
4.2 Perancangan Pengujian .....	16
4.2.1 Pengujian Kinerja .....	16
4.2.2 Pengujian <i>Fail Over</i> .....	16
4.2.3 Pengujian <i>Load balancing</i> .....	16
4.2.4 Pengujian Fungsional .....	17
4.3 Alur Kerja Sistem .....	17
BAB 5 IMPLEMENTASI .....	19
5.1 Instalasi Sistem .....	19
5.1.1 Instalasi Docker-Machine .....	19
5.1.2 Instalasi Docker .....	19
5.1.3 Konfigurasi Docker Swarm .....	19
5.2 Konfigurasi Layanan Docker Swarm .....	20
5.2.1 Membuat Service Nginx .....	20
5.2.2 Membuat Service Apache .....	21
5.3 Implementasi <i>Fail Over</i> dan <i>Load balancing</i> Berbasis Sumber Daya Memory .....	22
5.3.1 Implementasi <i>Pseudocode</i> Sender Dan Reciever Socket Programing.....	22
5.3.2 Implementasi <i>Pseudocode</i> Algoritma Loadbalancing Berbasis Sumber Daya Memory .....	22
5.3.3 Implementasi <i>Pseudocode</i> Algoritma <i>Fail Over</i> Berbasis Waktu .....	23
BAB 6 PENGUJIAN DAN ANALISIS .....	24
6.1 Pengujian .....	24
6.1.1 Skenario Pengujian Kinerja .....	24
6.1.2 Skenario Pengujian <i>Fail Over</i> .....	26
6.1.3 Skenario Pengujian <i>Load balancing</i> .....	26
6.1.4 Skenario Pengujian Fungsional .....	26
6.2 Analisa Pengujian .....	26
6.2.1 Analisa Pengujian Kinerja .....	26
6.2.2 Analisa Pengujian <i>Fail Over</i> .....	28
6.2.3 Analisa Pengujian <i>Load balancing</i> .....	31
6.2.4 Analisa Pengujian Fungsional .....	32
BAB 7 Kesimpulan dan saran .....	33
7.1 Kesimpulan .....	33

7.2 Saran.....	33
Daftar pustaka.....	34
LAMPIRAN .....	35

## DAFTAR TABEL

Tabel 2.1 Kajian Pustaka .....	5
Tabel 2.2 Perbandingan Antara Container Dan VM .....	6
Tabel 6.1 Jumlah <i>Client Load/Stress</i> pada <i>Web Server</i> .....	24
Tabel 6.2 Hasil Pengujian Menggunakan <i>CPU</i> dan <i>Memory</i> .....	27
Tabel 6.3 Hasil Pengujian Troughtput pada <i>host server</i> .....	27

## DAFTAR GAMBAR

Gambar 2.1 Arsitektur Docker Swarm .....	7
Gambar 2.2 <i>Pseudocode Fail Over</i> .....	8
Gambar 2.3 <i>Pseudocode load balancing</i> berbasis sumber daya <i>memory</i> .....	9
Gambar 3.1 Diagram Alur Penelitian .....	10
Gambar 3.2 Gambar Rancangan Sistem Docker Swarm .....	12
Gambar 4.1 Rancangan Arsitektur Web Service Dengan Docker Swarm .....	14
Gambar 4.2 <i>Pseudocode</i> perancangan <i>fail over</i> berdasarkan perbandingan waktu .....	15
Gambar 4.3 <i>Pseudocode</i> perancangan <i>load balancing</i> berdasarkan sumber daya <i>memory</i> ..	15
Gambar 4.5 <i>flowchart</i> implementasi <i>load balancing</i> menggunakan Docker Swarm .....	17
Gambar 5.1 Halaman Utama NGINX .....	20
Gambar 5.2 Halamam Utama Apache Port 8080 .....	21
Gambar 5.3 Implementasi <i>pseudocode</i> Sender <i>socket programing</i> .....	22
Gambar 5.4 Implementasi <i>pseudocode</i> Receiver <i>socket programing</i> .....	22
Gambar 5.5 Implemetasi <i>pseudocode</i> algortima <i>load balancing</i> berbasis sumber daya <i>Memory</i> .....	22
Gambar 5.6 Implementasi <i>pseudocode</i> algoritma <i>fail over berbasis waktu</i> .....	23
Gambar 6.1 Contoh data pengujian menggunakan top .....	25
Gambar 6.2 Contoh data pengujian Menggunakan free .....	25
Gambar 6.3 Contoh data pengujian throughput menggunakan Grafik JMeter™ .....	26
Gambar 6.4 <i>Output</i> program <i>fail over</i> ketika <i>Node Worker1 Down</i> .....	28
Gambar 6.5 <i>Output Mapping</i> Menggunakan Web Browser dengan <i>Node Worker1 Down</i> ....	28
Gambar 6.6 <i>Output</i> Main Program <i>fail over Node Worker2 Down</i> .....	29
Gambar 6.7 <i>Output Mapping</i> Menggunakan Web Browser dengan <i>Worker2 Down</i> .....	29
Gambar 6.8 <i>Output</i> Main Program <i>fail over Semua Node Worker Down</i> .....	30
Gambar 6.9 <i>Output Mapping</i> Menggunakan Web Browser .....	30
Gambar 6.10 <i>Output Load balancing</i> dan pengecekan <i>Mapping</i> menggunakan web browser .....	31
Gambar 6.11 <i>Output mapping load balancing</i> berdasarkan sumber daya <i>memory</i> .....	31
Gambar 6.12 <i>Output monitoring</i> pengiriman data oleh <i>Node Worker</i> .....	32
Gambar 6.13 Potongan <i>output monitoring</i> penerimaan data oleh <i>Node Manager</i> .....	32

# BAB 1

## PENDAHULUAN

### 1.1 Latar belakang

Pada Era *modern* seperti yang kita lihat pada kehidupan sehari-hari untuk mendapatkan informasi secara cepat dan fleksibel kita selalu mengakses website di internet. Selain itu hal yang lebih penting pula adalah website juga digunakan dalam bidang bisnis yang sering di akses oleh client dan juga harus selalu tersedia. Hal ini mengakibatkan banyak terjadi *traffic* yang besar didalam jaringan.

Untuk memperoleh akses yang cepat dan handal, perlunya adanya web server yang handal pula. Serta pengguna web server semakin meningkat pula seiring berjalannya waktu sehingga dibutuhkan server yang dapat berkerja optimal. Jika web server yang dimiliki hanya ada satu yang maka memungkinkan terjadinya "*a single point of failure*" (SPOF), yaitu kondisi server yang jika gagal merespons maka sistem akan tidak berfungsi. Hal itu bisa terjadi karena terlalu banyak request yang harus di handle oleh satu buah web server tadi (Julianto, 2017). Maka dibuatlah sebuah arsitektur jaringan dalam web server yang disebut arsitektur multiple server yaitu kumpulan dari beberapa web server yang saling terhubung dan bekerja sama, yang memiliki fungsi untuk mencapai keandalan(reliability) dan ketersediaan (availability) yang tinggi (Singh, 2015). Dalam arsitektur ini dibutuhkan mekanisme dalam pembagian beban *traffic* agar performa server tetap dalam keadaan optimal.

Salah satu mekanisme dalam pembagian beban *traffic* yaitu menggunakan load balancing, yaitu suatu mekanisme pembagian beban *traffic* yang di distribusikan pada dua atau lebih jalur koneksi. Load balancing memiliki banyak metode ataupun algoritma dalam implementasinya. Tujuan mekanisme tersebut agar memaksimalkan *throughput*, *traffic* dapat berjalan optimal, memperkecil waktu tanggap dan menghindari *overload* pada salah satu jalur koneksi (Dewobroto, 2009). Seperti pada penelitian sebelumnya menggunakan menggunakan metode berdasarkan *resource* atau sumber daya yang dimiliki oleh sebuah server sebagai acuan dalam mengambil keputusan atas server yang dipilih pada saat client melakukan (Julianto, 2017). Namun, peneliti ingin menggunakan metode berdasarkan penggunaan *memory* sebagai suatu acuan dalam pengambilan suatu keputusan atas suatu server yang akan dipilih saat *client* melakukan *request*. Karena dengan mengambil sumber daya yaitu *memory* yang paling ringan, memungkinkan meningkatkan dan meringankan kerja server.

Perkembangan jaringan sangatlah pesat karena jaringan semakin kompleks. Selain itu untuk *management* dan membangun server yang banyak juga membutuhkan biaya yang besar. Oleh karena itu penggunaan virtualisasi pada web server sangatlah diperlukan pada masa kini. Salah satu virtualisasi web server yang digunakan adalah Docker. Docker adalah salah satu software yang mengadopsi teknik containerisasi dan semakin banyak diterapkan di dalam lingkungan web *hosting* (Adiputra, 2015). Dan keunggulan Docker container adalah performanya lebih baik dibandingkan penggunaan *Virtual Machine* karena tidak menerapkan *Hypervisor* sehingga container Docker tidak perlu mengisolasi semua komponen hanya

mengisolasi library dan aplikasi yang di jalankan (Adiputra, 2015). Penerapan load balancing dalam web server sangat penting dan dapat menjadi solusi dalam menangani beban server yang sibuk sehingga dapat meningkatkan skalabilitas pada sistem terdistribusi (Nugroho, 2016).

Dari refrensi diatas dapat disimpulkan bahwa penggunaan *load balancing* dapat mengoptimalkan suatu web server dengan membagi beban *traffic* dengan berbagai algoritma dan metode yang berbeda. Oleh karena itu penulis akan melakukan penelitian dengan judul "**IMPLEMENTASI FAIL OVER BERDASARKAN PERBANDINGAN WAKTU DAN LOAD BALANCING SERVER WEB BERBASIS DOCKER SWARM BERDASARKAN PENGGUNAAN SUMBER DAYA MEMORY HOST**". Metode ini dipilih karena dengan mengetahui penggunaan *memory* web server kita bisa mengetahui *Node worker* mana yang bebannya sedikit memproses suatu *request* sehingga beban dapat di distribusikan dengan baik dan dapat mendekteksi *host* yang *down* ketika bekerja dengan menggunakan *fail over* berbasis waktu.

## 1.2 Rumusan masalah

Berdasarkan dari pemaparan latar belakang di atas, penulis dengan ini merumuskan rumusan masalah yang penulis akan kaji.

1. Bagaimana merancang arsitektur *Fail Over* dan *load balancing* server web berbasis Docker Swarm berdasarkan penggunaan sumber daya *memory host*?
2. Bagaimana mengimplementasikan arsitektur *Fail Over* dan *load balancing* server web berbasis Docker Swarm berdasarkan penggunaan sumber daya *memory host*?
3. Bagaimana kinerja dari arsitektur *Fail Over* dan *load balancing* server web berbasis Docker Swarm berdasarkan penggunaan sumber daya *memory host*?

## 1.3 Tujuan

Adapun tujuan dari pengimplementasian sistem ini secara umum, adalah :

1. Untuk merancang arsitektur Web Server menggunakan *Fail Over* dan *load balancing* dengan metode berbasis sumber daya *memory* pada Docker Swarm.
2. Untuk mengimplementasikan arsitektur *Fail Over* dan *load balancing* Dalam Web Server Menggunakan Metode berbasis sumber daya *memory* pada Docker Swarm.
3. Untuk mendapatkan hasil kinerja dari arsitektur *Fail Over* dan *load balancing* server web berbasis Docker Swarm berdasarkan penggunaan sumber daya *memory host* .

## 1.4 Manfaat

Keutamaan atau manfaat yang diperoleh dari pembuatan sistem ini antara lain :

1. Dapat meningkatkan kinerja layanan server web berbasis Docker Swarm.
2. Dapat meminimalisir terjadinya kegagalan (*failure*) layanan server web berbasis Docker Swarm.



## 1.5 Batasan masalah

Untuk menghindari pembahasan yang terlalu luas , penulis dalam menyelesaikan tugas akhir ini memiliki batasan-batasan masalah yang dibuat tanpa bermaksud menghilangkan maksud dan tujuan awal. Pembahasan masalah tersebut di antaranya adalah :

1. Teknologi virtualisasi berbasis kontainer yang digunakan adalah Docker;
2. Sistem Operasi yang digunakan *Host* adalah Linux 14.04 LTS 64-bit;
3. Sistem Operasi yang digunakan *Virtual Host* adalah Linux Server 17.10 64-bit;
4. Aplikasi *load balancing* yang digunakan adalah NGINX 18.05.0-CE;
5. Aplikasi web server yaang digunakan adalah APACHE 24.4.33;
6. *Resource* yang digunakan adalah *Memory*.

## 1.6 Sistematika Pembahasan

Penyusunan skripsi ini dibagi dalam tujuh bab, dengan kerangka pembahasan sebagai berikut:

### **BAB 1     PENDAHULUAN**

Bab ini membahas latar belakang penelitian, rumusan masalah, batasan masalah, tujuan dan manfaat penelitian, serta sistematika penulisan setiap bab dari skripsi ini

### **BAB 2     LANDASAN KEPUSTAKAAN**

Bab ini berisi teori dasar dan teori pendukung yang berhubungan dalam menyelesaikan penelitian yang akan diteliti. Kajian teori yang diambil berasal dari jurnal, buku, dan sumber referensi lainnya yang berhubungan dengan topik yang akan diteliti.

### **BAB 3     METODOLOGI PENELITIAN**

Bab ini berisi metode yang digunakan dalam penelitian yang terdiri dari studi literatur , perancangan, implementasi , pengujian dan analisis, serta pengambilan kesimpulan dan saran.

### **BAB 4     PERANCANGAN**

Membahas tentang analisa kebutuhan dan perancangan dari sistem yang dibuat dengan teori yang ada.

### **BAB 5     IMPLEMENTASI**

Memuat pembahasan tentang implementasi sistem dengan teori yang ada.

## **BAB 6      PENGUJIAN DAN ANALISIS**

Membahas tentang proses dan hasil pengujian, serta analisis terhadap sistem yang telah direalisasikan.

## **BAB 7      TINJUAN PUSTAKA**

Memuat kesimpulan yang diperoleh dari pembuatan dan pengujian sistem serta saran untuk mengembangkan lebih lanjut

## BAB 2

### LANDASAN KEPUSTAKAAN

Bab ini membahas tentang kajian pustaka dan dasar teori yang digunakan untuk menunjang penulisan skripsi *Implementasi Load Balancing Dalam Web Server Menggunakan Metode Berbasis Sumber Daya Memory pada Docker Swarm*. Beberapa dasar teori yang dibutuhkan untuk menyusun skripsi ini adalah Docker Swarm, Web Server NGIX, JSON dan Load Balancing.

#### 2.1 Kajian Pustaka

Pada subbab ini dilakukan kajian terhadap penelitian-penelitian sebelumnya yang terkait dengan penelitian ini dan dijadikan sebagai pedoman dalam pelaksanaan penelitian. Berikut tabel perbandingan penelitian terdahulu dan sekarang:

**Tabel 2.1 Kajian Pustaka**

No	Nama Penulis, Tahun dan Judul	Persamaan	Perbedaan	
			Penelitian Terdahulu	Rencana Penelitian
1	Adiputra Firmansyah [2015]. CONTAINER DAN DOCKER: TEKNIK VIRTUALISASI DALAM PENGELOLAAN BANYAK APLIKASI WEB	Menggunakan Web Server pada Docker Container	Menggunakan Containerisasi pada <i>host</i> yang sama	Menggunakan Docker Swarm sehingga container dapat berjalan pada <i>Multiple host</i>
2	M. Agung Nugroho, M.Kom ; Rikie Katardi . [2016]. ANALISIS KINERJA PENERAPAN CONTAINER UNTUK LOAD BALANCING WEB SERVER PADA RASPBERRY PI	Mengimplementasi <i>Load balancing</i> pada Docker	Mengimplementasi <i>Load balancing</i> pada Docker container dalam <i>host</i> yang sama	Mengimplementasi <i>Load balancing</i> pada Docker container dalam <i>host</i> yang berbeda
3	Tanjung P Kusuma <sup>1</sup> ; Dr. Ir. Rendy Munadi, M.T.; Danu Dwi Sanjoyo., S.T.; M.T. [2017]. IMPLEMENTASI DAN ANALISIS COMPUTER CLUSTERING SYSTEM DENGAN	Mengimplementasi <i>Load balancing</i> pada Docker Swarm	Mengimplementasi round-robin Load Balance menggunakan HPROXY	Mengimplementasi <i>Load balancing</i> Berbasis Sumber Daya Memory menggunakan NGINX

	MENGGUNAKAN VIRTUALISASI DOCKER			
--	------------------------------------	--	--	--

## 2.2 Docker

Docker adalah sebuah aplikasi yang bersifat open source yang berfungsi sebagai wadah/container untuk mengepak/memasukkan sebuah software secara lengkap beserta semua hal lainnya yang dibutuhkan oleh software tersebut dapat berfungsi. Pengaturan software beserta file/hal pendukung lainnya akan menjadi sebuah Image (istilah yang diberikan oleh Docker). Kemudian sebuah instan dari Image tersebut kemudian disebut Container. Perbandingan antara Docker Container dibandingkan VM dapat dilihat di Tabel 2.1.

Parameter	Virtual Machines	Containers
Guest OS	Setiap VM berjalan pada hardware dan kernel virtual yang dimuatkan ke dalam wilayah memorynya sendiri.	Semua guest berbagi pakai SO dan kernel yang sama. Image kernel dimuatkan ke dalam memory fisik.
Komunikasi	Melalui perangkat ethernet	Mekanisme IPC standard seperti Signal, Pipe dan Socket
Keamanan	Tergantung pada implementasi dari hypervisor	Kontrol akses <i>mandatory</i> dapat dimanfaatkan
Kinerja	VM mengalami overhead kecil karena instruksi mesin diterjemahkan dari <i>guest</i> OS ke Host.	Container menyediakan kinerja mendekati natif dibandingkan SO host yang mendasari.
Isolasi	Berbagi pakai pustaka, file-file antar guest dan antara guest dengan host tidaklah mungkin	Subdirektori dapat secara transparan dimount dan dibagi-pakaikan.
Waktu startup	Perlu beberapa menit untuk memulai (boot)	Dapat diboot dalam beberapa detik
Storage	Perlu lebih besar storage karena kernel OS lengkap dan program-program yang berasosiasi harus diinstal dan dijalankan.	Storage lebih kecil karena OS basis dibagi-pakaikan.

**Tabel 2.2 Perbandingan Antara Container Dan VM**

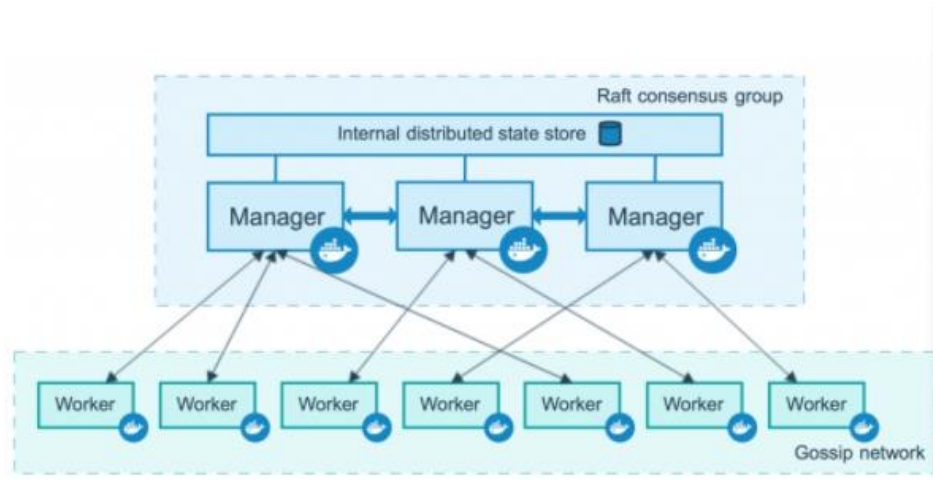
Sumber : (Adiputra, 2015)

## 2.3 Docker Swarm

Docker memperkenalkan Swarm mode pada versi 1.12. Mode ini memungkinkan pengguna untuk *me-deploy container* pada *multiple hosts* atau *Node*, menggunakan overlay network. Swarm mode merupakan bagian dari command line interface Docker yang memudahkan pengguna untuk mengelola komponen container (Docker, 2013).

Dalam Docker Swarm terdapat manager dan worker dimana kita bisa mengontrol lebih dari satu worker dengan manager. dengan Docker Swarm kita dapat disebut mengelompokkan suatu machine dalam grup tertentu dengan 1 manager atau lebih dan worker manager

berfungsi sebagai pengontrol bagi worker. Kelebihan dari Docker Swarm adalah jika salah satu *host down* maka service Docker akan digantikan dengan *host* yang sedang aktif.



**Gambar 2.1 Arsitektur Docker Swarm**

Sumber : (Docker, 2013)

## 2.4 Web Server NGIX

Web Server NGIX adalah server HTTP dan Proxy dengan kode sumber terbuka yang bisa juga berfungsi sebagai proxy IMAP/POP3. Kode sumber nginx ditulis oleh seorang warga negara Rusia yang bernama Igor Sysoev pada tahun 2002 dan dirilis ke publik pada tahun 2004. Nginx terkenal karena stabil, memiliki tingkat performansi tinggi dan minim mengonsumsi sumber daya (NGIX Software.INC, 2014).

Selain itu NGIX juga dapat digunakan sebagai *load balancing* karena memiliki fitur Proxy pass yang berfungsi untuk memindahkan jalur ke *Node* yang ditunjuk dan juga memiliki modul load balancing yang menggunakan metode :

1. Round-Robin;
2. Leastconn;
3. Weight.

## 2.5 JSON

JSON (*JavaScript Object Notation*) adalah format pertukaran data yang ringan, mudah dibaca dan ditulis oleh manusia, serta mudah diterjemahkan dan dibuat (*generate*) oleh komputer. Format ini dibuat berdasarkan bagian dari Bahasa Pemrograman JavaScript, Standar ECMA-262 Edisi ke-3 - Desember 1999. JSON merupakan format teks yang tidak bergantung pada bahasa pemrograman apapun karena menggunakan gaya bahasa yang umum digunakan oleh programmer keluarga C termasuk C, C++, C#, Java, JavaScript, Perl, Python dll. Oleh karena sifat-sifat tersebut, menjadikan JSON ideal sebagai bahasa pertukaran-data.

JSON terbuat dari dua struktur:

- Kumpulan pasangan nama/nilai. Pada beberapa bahasa, hal ini dinyatakan sebagai objek (*object*), rekaman (*record*), struktur (*struct*), kamus (*dictionary*), tabel hash (*hash table*), daftar berkunci (*keyed list*), atau *associative array*.
- Daftar nilai terurutkan (*an ordered list of values*). Pada kebanyakan bahasa, hal ini dinyatakan sebagai larik (*array*), vektor (*vector*), daftar (*list*), atau urutan (*sequence*).

Struktur-struktur data ini disebut sebagai struktur data universal. Pada dasarnya, semua bahasa pemrograman moderen mendukung struktur data ini dalam bentuk yang sama maupun berlainan. Hal ini pantas disebut demikian karena format data mudah dipertukarkan dengan bahasa-bahasa pemrograman yang juga berdasarkan pada struktur data ini (JSON ORG, 2017).

## 2.6 Load Balancing

Load balancing adalah salah satu teknik atau metode yang digunakan dalam pembagian beban *web server* dalam jaringan. Teknik ini mendistribusikan beban trafik pada dua atau lebih jalur koneksi secara seimbang, agar trafik dapat berjalan optimal. Load balancing juga mendistribusikan beban kerja secara merata di dua atau lebih komputer, *link* jaringan, *CPU*, *hard drive*, atau sumber daya lainnya, untuk mendapatkan pemanfaatan *resource* yang optimal (Dewobroto, 2009).

### 2.6.1 Fail Over Berdasarkan Perbandingan Waktu

Penulis menggunakan *fail over* berdasarkan perbandingan waktu antar *Node Manager* dan *Node Worker*. *Node Worker* dianggap *down* jika pengecekan waktu *Node Worker* ditambah waktu toleransi kurang dari waktu *real Node Manager*, dimana waktu toleransi adalah waktu yang dapat ditolerir oleh penulis menganggap *Node Worker* sedang *down*. Sehingga ketika salah satu *Node Worker down* maka dapat di monitor oleh *Node Manager* dan paket akan di-*forward* ke *Node Worker* yang Aktif.

**Gambar 2.2 Pseudocode Fail Over**

```
Begin
TM= Time_Manager
TW1= Time_Worker1;NW1=Node_Worker1;
TW2= Time_Worker2;NW2=Node_Worker2;
TR= Time_Tolerance
If ((TM< TW1+TR) && (TM<TW2+TR))
    Do Loadbalancing();
If ((TM< TW1+TR) && (TM>TW2+TR))
    Do Web Service route to NW1;
If ((TM> TW1+TR) && (TM<TW2+TR))
    Do Web Service route to NW2;
If ((TM> TW1+TR) && (TM>TW2+TR))
    Print ('All Web Server Down')
```

Pada Gambar 2.2 diatas menjelaskan cara kerja *Fail Over* dimana jika semua *Node Worker* Aktif atau waktu melebihi waktu *Node Manager* maka akan menjalankan method loadbalancer. Jika salah 1 *Node worker down* atau waktu *Mode Worker* kurang dari waktu *Node Manager* ketika pengecekan maka paket akan di forward ke *Node Worker* yang aktif. Dan Jika semua *Node Worker Down* atau semua waktu *Node Worker* kurang dari waktu pengecekan maka terdapat notifikasi bahwa semua *Node Worker Down*.

### 2.6.2 Load Balancing Berbasis Sumber Daya Memory

Penulis menggunakan *load balancing* metode berbasis sumber daya atau *resource*, karena dengan mengetahui *resource* dari *Node server (Node Worker)*, dapat memperkecil kemungkinan server gagal merespons atau *error* dengan mengetahui apakah beban server terlalu berat dengan melihat *resource*-nya yaitu nilai *Memory* yang terpakai. Ketika user melakukan request, sebelumnya server (*Node Worker*) telah mengirimkan data resource ke Manager *Node* dan membandingkan nilai *resource* tersebut yaitu berupa nilai *Memory Usage* server. Kemudian diambil nilai terkecil lalu *Node Manager* mengirimkan perintah untuk forward paket ke *Node Worker* dengan resource terkecil tersebut agar beban server seimbang.

Mekanisme pemilihan server dengan algoritma *resource* based dapat dilihat pada *pseudocode* berikut.

```
Begin
NW= Node_Worker
M= Free_Memory_Host
NW_M = { (NW,M) }
If No Resource In List
Find maximum M in NW_M
Return NW
Do Web Service route to NW
End
```

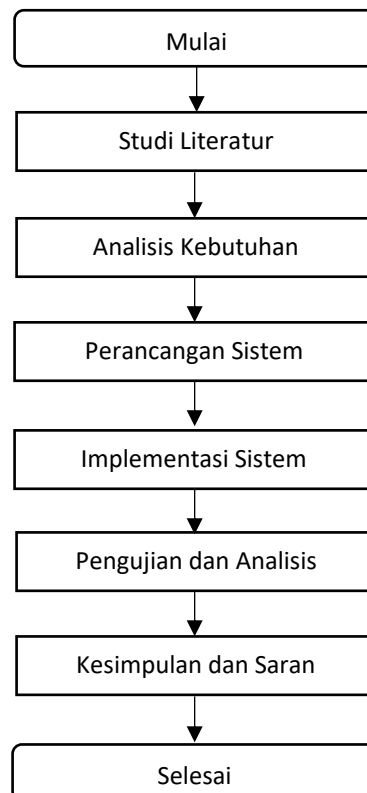
**Gambar 2.3 Pseudocode load balancing berbasis sumber daya memory**

Pada Gambar 2.3 diatas menjelaskan cara kerja load balancing dengan metode *memory* resource yaitu memilih IP dan *Memory* Resource dari *Node Worker* 1 sebagai default jika tidak ada resource yang dikirim ke *Node Manager*. Jika terdapat *Node Worker* yang memiliki penggunaan free space *memory* terbanyak maka *Node Manager* akan memerintahkan agar *Node Worker* yang menggunakan free space *memory* terbanyak untuk mengambil alih *route* web service. Pembagian beban pada web server dapat dilakukan dengan seimbang maka kinerja web server dapat memberikan respons yang baik terhadap *request* dari user (Julianto, 2017).

## BAB 3

### METODOLOGI PENELITIAN

Dalam bab ini akan menjelaskan sistematika dalam penulisan skripsi ini, dari tahap pertama sampai keseluruhan penelitian. Proses awal dari penelitian ini adalah mencari literatur yang sesuai dengan topik penelitian. Analisis Kebutuhan adalah mendata segala kebutuhan yang nantinya akan digunakan dalam penelitian. Perancangan adalah proses untuk memberikan gambaran umum tentang alur dari penelitian. Implementasi adalah menerapkan dan mengatur segala sesuatu yang akan membuat sistem dapat berjalan sesuai dengan yang dikehendaki. Pengujian adalah menjalankan semua komponen apakah sudah berjalan sesuai keinginan atau belum. Analisis adalah proses untuk mencari data dari hasil dari pengujian untuk nantinya diolah untuk menjadi kesimpulan dari penelitian. Kesimpulan adalah hasil dari keseluruhan penelitian dan menyertakan saran untuk ke arah pengembangan selanjutnya.



**Gambar 3.1 Diagram Alur Penelitian**



### 3.1 Studi Literatur

Studi literatur mempelajari mengenai penjelasan dasar teori yang digunakan untuk menunjang penulisan skripsi. Teori-teori pendukung tersebut diperoleh dari buku, artikel, jurnal, *E-book*, dan dokumentasi *project*. Teori-teori pendukung tersebut meliputi :

1. Docker Swarm;
2. Web Server NGIX;
3. JSON;
4. *Load Balancing*.

### 3.2 Analisis Kebutuhan

Analisa kebutuhan dilakukan untuk mendapatkan komponen komponen yang dibutuhkan untuk penerapan *Fail Over* dan *load balancing* menggunakan sumber daya *memory* pada Docker Swarm.

#### 3.2.1 Kebutuhan Fungsional

Kebutuhan fungsional adalah kebutuhan yang harus disediakan dalam sistem. Kebutuhan fungsional untuk penerapan penerapan *Fail Over* dan *load balancing* menggunakan sumber daya *memory* pada Docker Swarm yaitu :

1. Sistem Worker dapat membuat dan menulis data dari *resource host* berupa data json.
2. Sistem Worker dapat mengirim data antar *Node*;
3. Sistem Manager dapat mengecek data resource *Node Worker* dalam data json;
4. Sistem Manager dapat me-forward paket ke *Node Worker* yang lain;
5. Sistem Manager dapat memonitor *Node Worker* yang aktif dan tidak aktif.

#### 3.2.2 Kebutuhan Non Fungsional

Kebutuhan non fungsional adalah kebutuhan batasan fungsi agar sistem dapat bekerja dengan maksimal. Kebutuhan fungsional untuk penerapan penerapan *Fail Over* dan *load balancing* menggunakan sumber daya *memory* pada Docker Swarm yang akan di jelaskan pada subab di bawah ini.

##### 3.2.2.1 Kebutuhan Perangkat Keras

Menjelaskan spesifikasi dari perangkat keras yang digunakan dalam pengujian penelitian skripsi ini. Berikut kebutuhan perangkat keras laptop yang digunakan:

- Merk : ASUS A455L
- *Memory* : 4 GB

- Processor : Intel Core i3

Selain itu terdapat kebutuhan *Virtual Machine* yang digunakan:

- Jumlah *Node* : 3
- *Memory* : 700 Mb / *Node*

### 3.2.2.2 Kebutuhan Perangkat Lunak

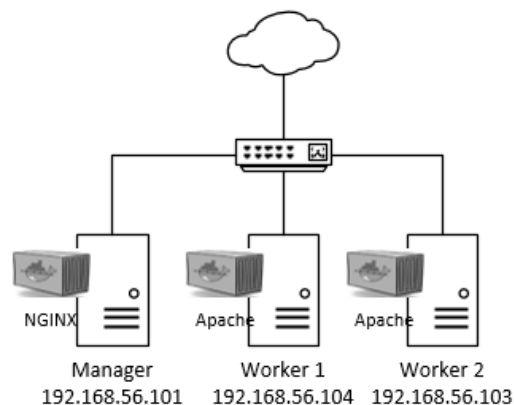
Menjelaskan spesifikasi dari perangkat lunak yang digunakan dalam pengujian penelitian skripsi ini. Berikut kebutuhan perangkat lunak yang digunakan:

- Sistem Operasi *Host* : Linux Ubuntu Desktop 14.04 LTS 64-bit
- Sistem Operasi *Virtual Host* : Linux Ubuntu Server 17.10 64-bit
- Docker-machine
- Docker-ce
- Nginx
- Virtual Box
- Apache

### 3.3 Perancangan Sistem

Perancangan sistem dilakukan setelah studi literatur dan semua kebutuhan sistem yang akan digunakan telah diperoleh melalui tahapan analisis kebutuhan. Perancangan sistem dapat di lihat pada Gambar 3.2.

Tahap perancangan berbasis container ini menggunakan metode routing mesh, dimana antara *Node* yang menggunakan virtualisasi container ini di jadikan satu sebagai Swarm mode. Routing mesh memungkinkan setiap *Node* di Swarm untuk menerima koneksi pada port yang dipublish untuk layanan yang berjalan di Swarm.



**Gambar 3.2 Gambar Rancangan Sistem Docker Swarm**

### 3.4 Implementasi Sistem

Implementasi sistem dibuat berdasarkan perancangan kebutuhan dan perancangan sistem yang telah dibuat sebelumnya. Adapun implementasi sistem sebagai berikut.

1. Melakukan konfigurasi Docker Swarm
2. Melakukan konfigurasi Web Server
3. Memasang Load Balancing
4. Mengakses halaman Web Server oleh *client*
5. Menganalisa hasil *output*

### 3.5 Pengujian dan Analisis Sistem

Dalam tahap pengujian, peneliti akan melakukan pengujian terhadap jumlah prosesor apakah mempengaruhi sistem atau tidak. Selanjutnya, melakukan analisis sekaligus menilai *throughput, response time dan Memory Usage* yang terjadi pada web server (*Node Worker*) atau sistem *load balancing*.

### 3.6 Kesimpulan dan Saran

Kesimpulan pada penelitian ini adalah hasil analisa terhadap rancangan sistem yang telah dibangun dan diperoleh pada saat pengujian. Pada kesimpulan ini harus dapat memberikan hasil dari rumusan masalah yang diantaranya adalah untuk merancang, mengimplementasikan dan mengukur beban pada web server.

Tahap terakhir penulisan adalah saran yang dimaksudkan untuk memperbaiki kesalahan-kesalahan yang terjadi dan menyempurnakan penulisan serta untuk memberikan saran atas pengembangan selanjutnya.

## BAB 4

### PERANCANGAN

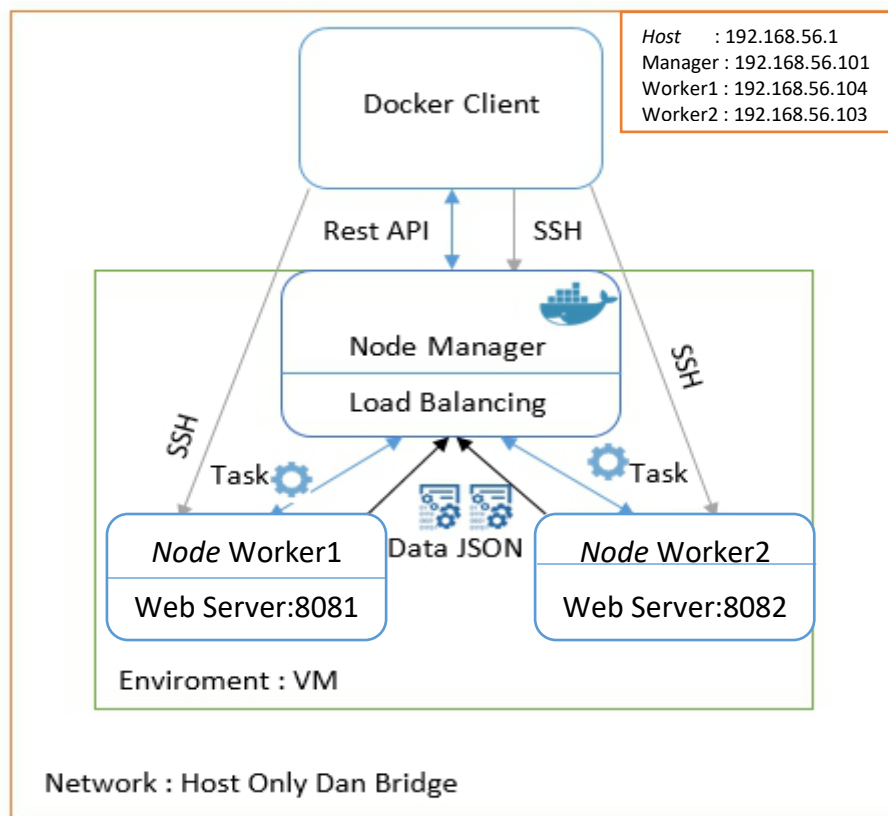
Pada bagian ini dijelaskan langkah-langkah untuk merancang dan mengimplementasikan sebuah sistem Web Server dengan menggunakan Docker Swarm dengan *fail over* berdasarkan waktu dan *load balancing* berdasarkan sumber daya komputer dengan berpedoman pada metodologi penelitian yang telah dibahas sebelumnya.

#### 4.1 Perancangan

Perancangan dilakukan untuk melakukan perencanaan terhadap sistem yang dibangun berdasarkan kebutuhan yang telah dispesifikasikan.

##### 4.1.1 Topologi

Untuk dapat memanfaatkan sistem Web Service pada Docker Swarm dengan menggunakan load-balancing berbasis sumber daya komputer pada suatu jaringan, diperlukan topologi dimana setiap *Node host* dapat saling terhubung dan dapat diakses oleh client. Gambar topologi yang dapat memenuhi kriteria tersebut dapat dilihat pada gambar 4.1. Dan *Host Virtual* yang digunakan adalah 3 buah Virtual Machine.



Gambar 4.1 Rancangan Arsitektur Web Service Dengan Docker Swarm

#### 4.1.2 Fail Over Berdasarkan Perbandingan Waktu

```
Begin
TM= Time_Manager;NM=Node_Manager;
TW1= Time_Worker1;NW1=Node_Worker1;
TW2= Time_Worker2;NW2=Node_Worker2;
TR= Time_Tolerance
If ((TM< TW1+TR) && (TM<TW2+TR))
    Do Loadbalancing();
If ((TM< TW1+TR) && (TM>TW2+TR))
    Print ('Node Worker1 Down')
    Do Web Service route to NW1;
If ((TM> TW1+TR) && (TM<TW2+TR))
    Print ('Node Worker2 Down')
    Do Web Service route to NW2;
If ((TM> TW1+TR) && (TM>TW2+TR))
    Do Web Service route to NM;
    Print ('All Web Server Down')
```

**Gambar 4.2 Pseudocode perancangan *fail over* berdasarkan perbandingan waktu**

*Fail over* berdasarkan perbandingan waktu pada Docker Swarm berjalan di *Node* master (192.168.56.101) yang dapat memproses *request user* yang masuk dan meneruskannya ke *Node* yang ada di dalam Swarm. Dalam penyebaran Swarm, *fail over* ini bekerja ketika data waktu dari *Node Worker* diproses. Ketika semua data waktu dari *Node Worker* lebih dari waktu nyata *Node Manager* maka akan di teruskan ke method *loadbalancing*. Ketika salah satu data waktu dari *Node Worker* kurang dari waktu nyata *Node Manager* maka beban traffic akan di bebaskan pada salah satu *Node Worker* yang memiliki data waktu lebih besar dari waktu nyata *Node Manager* dan menghasilkan output bahwa salah satu *Node Worker down*. Dan jika semua data waktu dari *Node Worker* kurang dari waktu nyata *Node Manager* maka beban traffic akan diambil oleh *Node Manager* dan menghasilkan output bahwa semua *Node Worker down*.

#### 4.1.3 Load Balancing Berdasarkan Sumber Daya Memory

```
Begin
NW= Node_Worker
M= Free_Memory_Host
NW_M = { (NW,M) }
If No Resource In List
Find maximum M in NW_M
Return NW
Do Web Service route to NW
End
```

**Gambar 4.3 Pseudocode perancangan *load balancing* berdasarkan sumber daya *memory***

*Load balancing* pada Docker Swarm berjalan di *Node* master (192.168.56.101) yang dapat memproses *request user* yang masuk dan meneruskannya ke *Node* yang ada di dalam Swarm. Dalam penyebaran Swarm, *loadbalancer* menangani permintaan klien masuk serta permintaan layanan ke layanan internal. Sistem ini membutuhkan koneksi internet agar masing-masing komponen atau *Node* saling terhubung. Proses diawali dengan membuat suatu konten atau layanan pada server master yang telah menerapkan Docker Swarm. Lalu secara otomatis konten atau layanan yang telah di buat akan tersebar ke semua worker yang tergabung ke dalam Swarm. Setelah itu pengembang memberikan konfigurasi *load balancing* pada server master untuk membagi beban request. Kemudian user dapat mengakses layanan atau konten yang sudah di buat pada sistem tersebut. *Load balancing* akan meneruskan request user ke *Node* worker yang memiliki free space *memory* terbesar.

## 4.2 Perancangan Pengujian

Pada perancangan pengujian dilakukan untuk memastikan bahwa sistem sesuai dengan tujuan atau *goal* yang telah di tentukan oleh peneliti. Pengujian yang akan dilakukan yaitu:

1. Pengujian kinerja;
2. Pengujian *load balancing*;
3. Pengujian *fail over*;
4. Pengujian fungsional.

### 4.2.1 Pengujian Kinerja

Pada pengujian kinerja dilakukan dengan cara membanjiri sistem menggunakan aplikasi JMeter™ dengan user sebanyak 50 user, 250 user dan 500 user. Dalam aplikasi JMeter™ terdapat fitur untuk mendeteksi throughput pada jaringan.

Selain itu dalam pengujian kinerja juga mengambil parameter *CPU* utilization dan *Memory* utilization dengan menggunakan aplikasi Free dan TOP pada sistem Linux.

### 4.2.2 Pengujian *Fail Over*

Pada pengujian *fail over* dilakukan dengan cara pengecekan perbandingan waktu antar *Node Worker* dan *Node Manager* dan juga menggunakan metode *Mapping*. Dimana dalam pengecekan *fail over* dapat di cek dalam *output* file dan juga dapat dimonitor oleh sistem.

Dalam metode *Mapping* di lakukan dengan cara mengakses halaman website dengan menggunakan IP *Node Manager*.

### 4.2.3 Pengujian *Load balancing*

Pada pengujian *load balancing* dilakukan dengan cara pengecekan perbandingan *memory* antar *Node Worker* dan juga menggunakan metode *Mapping*. Dimana dalam perbandingan *memory* antar *Node Worker* dapat di cek dalam *output* file dan juga dapat dimonitor oleh sistem.

Dalam metode *Mapping* di lakukan dengan cara mengakses halaman website dengan menggunakan IP *Node Manager*.

#### 4.2.4 Pengujian Fungsional

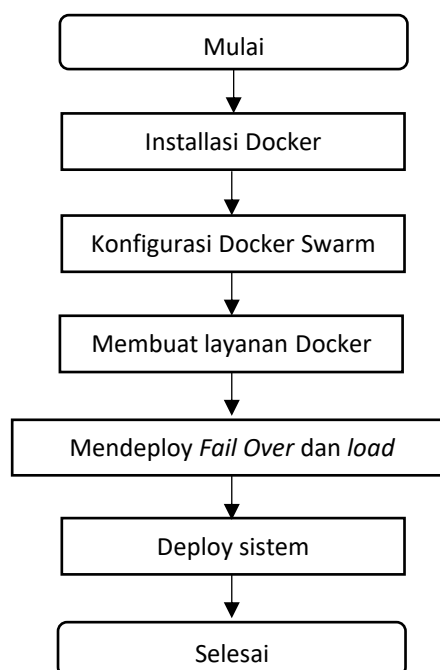
Pada pengujian fungsional dilakukan dengan pengujian white-box dan black-box. Dimana pengujian white-box dilakukan untuk pengecekan logika sistem dan dalam pengujian black-box dilakukan untuk memeriksa fungsional dari sistem.

Pengujian dianggap berhasil jika semua fungsi yang di uji tidak menunjukkan kegagalan dan sesuai dengan apa yang diharapkan. Parameter fungsional yang di ujikan adalah:

1. Sistem harus dapat mengirim data resource berupa data json dan juga dapat di tampilkan di monitor;
2. Sistem harus dapat menerima data resource dan juga dapat di tampilkan di monitor,
3. Sistem harus dapat mengolah data resource dan dapat di tampilkan di monitor,
4. Sistem harus dapat menyimpan log data dari semua proses.

#### 4.3 Alur Kerja Sistem

Pada bab ini dijelaskan perancangan sistem dari tahap konfigurasi Docker Swarm dan *loadbalancer*, pembuatan *container* hasil pengujian dan kesimpulan yang dilakukan pada sistem. Dibawah ini adalah *flowchart* implementasi *Fail Over* dan *load balancing* menggunakan Docker Swarm:



**Gambar 4.4 *flowchart* implementasi *load balancing* menggunakan Docker Swarm**

Pada gambar 4.2 dapat dilihat bahwa di tahap perancangan ini diawali dengan menginstal komponen yang dibutuhkan dan konfigurasi DockerSwarm. Untuk penjelasan lebih detail mengenai gambar 4.2 akan dibahas sebagai berikut:

1. Instalasi Docker, pada tahap ini melakukan instalasi dan dependency Docker agar dapat bekerja dalam sistem yang telah di sediakan
2. Melakukan konfigurasi DockerSwarm. Konfigurasi dilakukan untuk menggabungkan beberapa *Node* yang akan di jadikan Swarm.

3. Proses membuat konten dan layanan dilakukan setelah seluruh *Node* sudah dimasukkan dalam *Swarm mode*. Pembuatan konten dan layanan dilakukan dengan cara *pull* atau *download image* yang telah disediakan oleh Docker yang dibutuhkan untuk membuat layanan seperti Nginx dan Apache.
4. Mendeploy *Fail Over* dan *load balancing* yang telah di implementasikan sehingga dapat digunakan dalam sistem.
5. *Deploy* sistem dilakukan setelah sistem sudah berjalan dengan benar dan sesuai yang diharapkan, lalu di uji untuk mengetahui jika terjadi error pada sistem



## BAB 5

### IMPLEMENTASI

#### 5.1 Instalasi Sistem

Pada Subbab ini menjelaskan proses instalasi program yang digunakan untuk membangun sistem yang akan dibuat.

##### 5.1.1 Instalasi Docker-Machine

Docker *Machine* adalah alat yang memungkinkan Anda menginstal Docker *Engine* pada *hosts*, dan mengelola *virtual host* dengan perintah Docker-*machine*. Menggunakan perintah Docker-*machine*, Anda dapat memulai, memeriksa, menghentikan, dan *re-start* *host* yang dikelola, *upgrade* klien dan *daemon* Docker, dan mengkonfigurasi klien Docker untuk dapat saling berkomunikasi dengan *host* Anda. Docker-Machine digunakan untuk manajemen *Node* Docker.

1. Mendownload *tool* Docker *machine* pada *github* resmi Docker.
2. Mengecek versi dari Docker-Machine.

##### 5.1.2 Instalasi Docker

Docker merupakan *open platform* untuk pengembangan, pengiriman, dan menjalankan aplikasi. Dengan Docker, Anda dapat mengelola infrastruktur Anda dengan cara yang sama seperti mengelola aplikasi Anda. Docker diinstal pada *virtual host*. Dengan memanfaatkan metodologi Docker untuk mengirim, menguji, dan menerapkan kode dengan cepat.

Sebelum Anda menginstal Docker-*CE* untuk pertama kalinya pada mesin *virtual host* baru, Anda perlu mengatur *repository* Docker. Setelah itu, Anda dapat menginstal dan memperbarui Docker dari *repository*.

Mengatur repository:

1. Memperbarui indeks *apt package* pada masing masing *host* virtual
2. Instal *package* untuk memungkinkan *apt* untuk menggunakan repository melalui *HTTPS*.
3. Menambahkan kunci *GPG* resmi Docker:

Instal Docker *CE*:

1. Memperbarui indeks *apt-package*
2. Instal versi terbaru Docker-*CE*
3. Lalu cek versi dari Docker-*CE* yang sudah diinstal

##### 5.1.3 Konfigurasi Docker Swarm

Setelah Docker sudah terinstal pada masing masing *virtual host*, kemudian lakukan konfigurasi untuk menggabungkan *virtual host* yang telah dibuat ke dalam Swarm. Pertama kita tentukan dahulu *Node* atau *virtual host* yang akan di jadikan *leader* pada Swarm.

1. Hal pertama yang harus dilakukan adalah menginisialisasi Swarm. Saya akan *SSH* ke mesin *master* dan menginisialisasi Swarm di sana.
2. Lakukan langkah-langkah berikut untuk menjadikan *Node* atau virtual *host* yang sudah dibuat untuk dijadikan *leader* Swarm.
3. Kemudian melihat status *Node* Swarm

Setelah Swarm *leader* ditentukan, kemudian masukan perintah berikut ke dalam virtual *host* atau *Node* yang sudah dibuat untuk dijadikan *worker*. Anda harus menggunakan perintah *join-token <role>* :

1. Untuk menambah sebuah *worker*, kita dapat memasukan perintah yang telah disediakan pada *Node master*.
2. Kemudian kita salin token yang ada pada *Node master*.
3. Kemudian melihat status Docker Swarm apakah sudah berhasil.

## 5.2 Konfigurasi Layanan Docker Swarm

Sekarang setelah kita mengkonfigurasi *Node*, saatnya untuk membuat suatu layanan, disini kami menggunakan *Nginx* sebagai layanan atau aplikasi. Kami akan fokus pada aplikasi dan tidak khawatir tentang di mana aplikasi akan berjalan. Yang harus kita lakukan adalah memberi tahu manajer untuk menjalankan layanan atau aplikasi dan akan mengurus penjadwalan kontainer, mengirim perintah ke *Node* dan mendistribusikannya.

### 5.2.1 Membuat Service Nginx

1. Membuat Service *Load balancing* Dengan menggunakan *Nginx*.
2. Mengecek Layananan bahwa Service berjalan.
3. Setelah layanan dibuat, maka akses halaman NGINX dengan menuliskan alamat IP pada URL Web Browser.



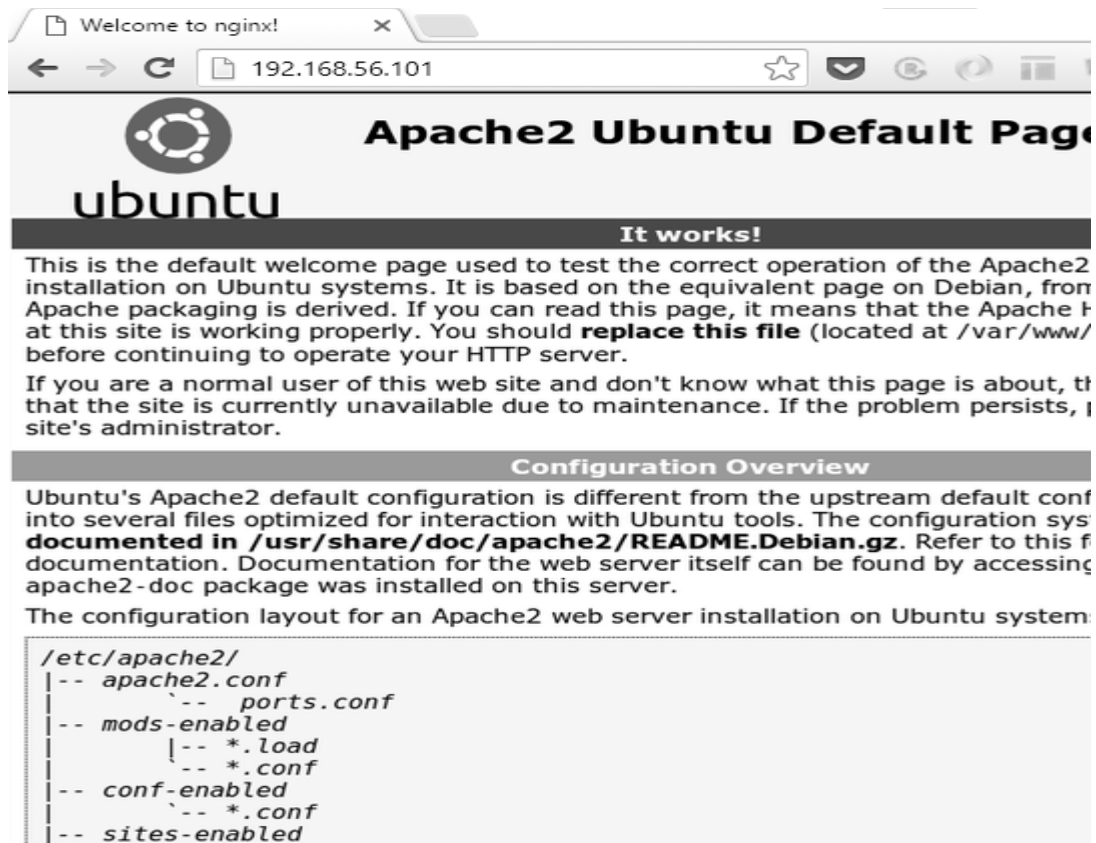
**Gambar 5.1 Halaman Utama NGINX**

4. Setelah NGINX telah dapat di akses hapus default.conf pada Container.
5. Kemudian reload service NGINX pada container .

### 5.2.2 Membuat Service Apache

Setelah NGIX telah dikonfigurasi maka, membuat service Apache sebagai Web Server

1. Membuat Service Apache dengan Port yang berbeda.
2. Mengecek Layanan bahwa Service berjalan.
3. Setelah layanan dibuat, maka akses halaman Apache dengan menuliskan alamat IP pada URL Web Browser.



Gambar 5.2 Halamam Utama Apache Port 8080

4. Setelah layanan dibuat, maka akses halaman Apache dengan menuliskan alamat IP Manager pada URL Web Browser dengan tambahan port yang berbeda sesuai service yang di daftarkan.

### 5.3 Implementasi *Fail Over* dan *Load balancing* Berbasis Sumber Daya *Memory*

Pada Subab ini menjelaskan proses algoritma yang digunakan dalam implementasi *Fail Over* dan *load balancing* berbasis sumber daya *memory*. Selain itu juga terdapat algoritma yang digunakan untuk membuat kebutuhan fungsional dapat berjalan sesuai dengan yang diharapkan penulis.

#### 5.3.1 Implementasi *Pseudocode* Sender Dan Reciever Socket Programing

No	Sender (Worker Node)
1	Use Socket UDP and Ipv4
2	Initialize IP Host , Port,And Buffer
3	Initialize Time Sender
4	Initialize Data Worker=Data_worker['Name'], Data_worker['Time'],
5	Data_worker['FreeMemory'], Data_worker['IP']
6	Send Data Worker To Reciever
7	Close connection socket

**Gambar 5.3 Implementasi *pseudocode* Sender socket programing**

No	Receiver (Manager Node)
1	Use Socket UDP and Ipv4
2	Initialize IP Host,Port,Buffer And Reuseable Address
3	Initialize Time Receive
4	Set Timeout
5	recieve Data from Buffer and save Data to file
6	If Time of receive data = Timeout:
7	Close connection socket

**Gambar 5.4 Implementasi *pseudocode* Receiver socket programing**

Pada gambar 5.3 Dan Gambar 5.4 adalah *pseudocode* dari pengiriman dan penerimaan data menggunakan *socket programming*. Dimana Sender bertujuan untuk mengirimkan data Json yang berisikan data *Node Worker* yang akan dikirimkan kepada *Node Manager* dan dijalankan oleh masing-masing *Node Worker*. Receiver bertujuan untuk menerima data dari sender dan mengolahnya menjadi file JSON dan dijalankan oleh *Node Manager*. Program sender dan receiver menggunakan pengiriman paket UDP dan menggunakan Ipv4.

#### 5.3.2 Implementasi *Pseudocode* Algoritma Loadbalancing Berbasis Sumber Daya *Memory*

No	Load Balancing Berbabasis Sumber Daya <i>Memory</i>
1	Load Data_Worker1
2	Load Data_Worker2
3	If (Data_Worker1["FreeMemory"]> Data_Worker2["FreeMemory"]):
4	Return Data =Data_Worker1
5	Else if (Data_Worker1["FreeMemory"]< Data_Worker2["FreeMemory"]):
6	Return Data= Data_Worker2
7	Else:
8	Pass
9	Print Data
10	Create NGINX configuration and Do Web Server Route to Data["IP"]
11	Reload NGINX service

**Gambar 5.5 Implemetasi *pseudocode* algoritma load balancing berbasis sumber daya *Memory***

Dalam Penggunaan Loadbalancer berbasis sumber daya komputer pada Web server data utama yaitu free space *memory*. Data tersebut dikirimkan oleh setiap *Node* worker dan disimpan dalam file JSON dan juga didalam filenya waktu pengambilan data, nama *Node* , IP *Node*. File tersebut dikirim menggunakan UDP ke *Node* Manager. Data yang telah diterima oleh *Node* Manager akan di proses dan hasil dari proses tersebut menentukan *host* mana yang akan menerima request dari user. Souce Code dari Algoritma Load Balancing Berbasis Sumber Daya Komputer Dapat di lihat pada Gambar 5.5.

### 5.3.3 Implementasi *Pseudocode* Algoritma *Fail Over* Berbasis Waktu

No	Algoritma <i>Fail Over</i>
1	Load Data_Worker1
2	Load Data_Worker2
3	Set Time_Host_Now
4	Set Time_To_Assume_Node_Worker_Down
5	Worker1TimeMoreXsecond=Data_Worker1['Time']+Time_To_Assume_Node_Worker_Down
6	Worker2TimeMoreXsecond=Data_Worker2['Time']+Time_To_Assume_Node_Worker_Down
7	Worker2TimeMoreXsecond=Data_Worker2['Time']+Time_To_Assume_Node_Worker_Down
8	Worker_Down
9	If (Worker1TimeMoreXsecond > Time_Host_Now and Worker2TimeMoreXsecond > Time_Host_Now):
10	Go To Loadbalancer (Data_Woker1,Data_Worker2)
11	If (Worker1TimeMoreXsecond > Time_Host_Now and Worker2TimeMoreXsecond < Time_Host_Now):
12	Print 'Host Worker 2 Down'
13	Return Data=Data_Woker1
14	If (Worker1TimeMoreXsecond < Time_Host_Now and Worker2TimeMoreXsecond > Time_Host_Now):
15	Print 'Host Worker 1 Down'
16	Return Data=Data_Woker2
17	If (Worker1TimeMoreXsecond < Time_Host_Now and Worker2TimeMoreXsecond < Time_Host_Now):
18	Print 'Semua Host Down'
19	Return Data='Fail'
20	If (Data!=Fail):
21	Create NGINX configuration and Do Web Server Route to Data["IP"]
22	Reload NGINX service
23	Else:
24	Create NGINX configuration and Do Web Server Route to Manager Node
25	Reload NGINX service
26	
27	
28	
29	

**Gambar 5.6 Implementasi *pseudocode* algoritma *fail over* berbasis waktu**

Dalam Gambar 5.6 dijelaskan bahwa Algoritma *Fail Over* ditentukan oleh penambahan asumsi waktu (waktu toleransi) yang menganggap server atau *Node* worker tersebut *down*. Jika kedua *Node* masih hidup dan sesuai dengan kondisi maka akan dilanjutkan ke method Loadbalancer. Jika tidak memenuhi kondisi tersebut atau salah satu mati maka data akan langsung diolah sesuai dengan *Node* yang masih hidup dan terdapat *output* warning. Jika semua *Node* worker atau server *down* maka akan terdapat *output* warning dan web service akan di alihkan ke *Node* Manager dan mengunggu *Node* worker kembali hidup.

## BAB 6

### PENGUJIAN DAN ANALISIS

#### 6.1 Pengujian

Pengujian dilakukan berdasarkan beberapa parameter uji yang telah ditentukan sebelumnya, yaitu sumber daya komputasi dan jaringan. Kedua parameter tersebut dibagi menjadi beberapa bagian yaitu, penggunaan *memory* dan *CPU*, *throughput* dan *mapping*.

1. Penggunaan *memory* dan *CPU* (*Resource Utilization*), mengetahui kemampuan router dalam menggunakan sumber daya saat melakukan sebuah proses tertentu. Pengujian sumber daya *host* tersebut dicek menggunakan aplikasi TOP berfungsi menampilkan penggunaan *memory* dan *CPU* secara real time dan Free yang berfungsi menampilkan sisa *free memory*.
2. *Throughput*, *Throughput* merupakan kecepatan pengiriman data pada waktu tertentu berdasarkan penggunaan bandwidth yang ada. Pengujian *throughput* dilakukan pada pengiriman data antara *client* dan *server*.

##### 6.1.1 Skenario Pengujian Kinerja

Pada subbab ini menjelaskan skenario dan parameter yang digunakan dalam pengujian kinerja

##### 6.1.1.1 Pengujian *Memory* dan *CPU* (*Resource Utilization*)

Pengujian *Memory* dan *CPU* dilakukan dengan cara melakukan *load/stress* testing menggunakan aplikasi JMeter™ dimana menggunakan *ThreadGroup* dengan *Client* yang telah ditentukan. Dalam *ThreadGroup*, *Client* secara terus menerus melakukan *load/stress* pada web server. Jumlah *Client* dapat dilihat pada gambar 6.1

**Tabel 6.1 Jumlah *Client Load/Stress* pada *Web Server***

No	Jumlah Client	Akses
1	50	Forever
2	250	Forever
3	500	Forever

Proses pengambilan data pada pengujian ini dilakukan menggunakan aplikasi Top dan Free yang ada pada masing-masing *Node* worker. Top merupakan aplikasi bawaan pada sistem operasi linux yang berfungsi menampilkan penggunaan *memory* dan *CPU* secara real time. Dan Free berfungsi untuk pengecekan sisa *memory* pada *host* yang tersedia. Proses pengujian pada setiap *host* tidak berbeda.

```
top - 15:40:23 up 1:23, 3 users, load average: 0.79, 0.26, 0.20
Threads: 303 total, 4 running, 299 sleeping, 0 stopped, 0 zombie
%Cpu(s): 11.9 us, 2.5 sy, 0.0 ni, 83.4 id, 2.2 wa, 0.0 hi, 0.0 si, 0.0 st
KiB Mem: 1653760 total, 1115088 used, 538672 free, 77208 buffers
KiB Swap: 7238652 total, 0 used, 7238652 free. 525264 cached Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1069	root	20	0	141948	46632	24940	S	6.6	2.8	4:27.97	Xorg
7	root	20	0	0	0	0	R	0.0	0.0	0:11.14	rcu_sched
29	root	20	0	0	0	0	S	0.0	0.0	0:06.46	kworker/0:1
2211	root	20	0	0	0	0	S	0.3	0.0	0:05.38	kworker/1:1
14	root	20	0	0	0	0	S	0.0	0.0	0:04.37	kworker/1:0
1	root	20	0	4444	2444	1424	S	0.0	0.1	0:03.29	init
1177	root	20	0	38076	4380	3384	S	0.0	0.3	0:02.00	upowerd
953	root	20	0	2192	648	520	S	0.0	0.0	0:01.91	acpid
13	root	20	0	0	0	0	R	0.0	0.0	0:01.27	ksoftirqd/1
763	root	20	0	53432	9928	4792	S	0.0	0.6	0:01.16	NetworkManager
949	root	20	0	3976	704	516	S	0.0	0.0	0:01.02	irqbalance
275	root	20	0	3008	620	472	S	0.0	0.0	0:00.56	upstart-udev-br
3	root	20	0	0	0	0	S	0.0	0.0	0:00.49	ksoftirqd/0
1072	root	20	0	36536	6008	3160	S	0.0	0.4	0:00.36	accounts-daemon

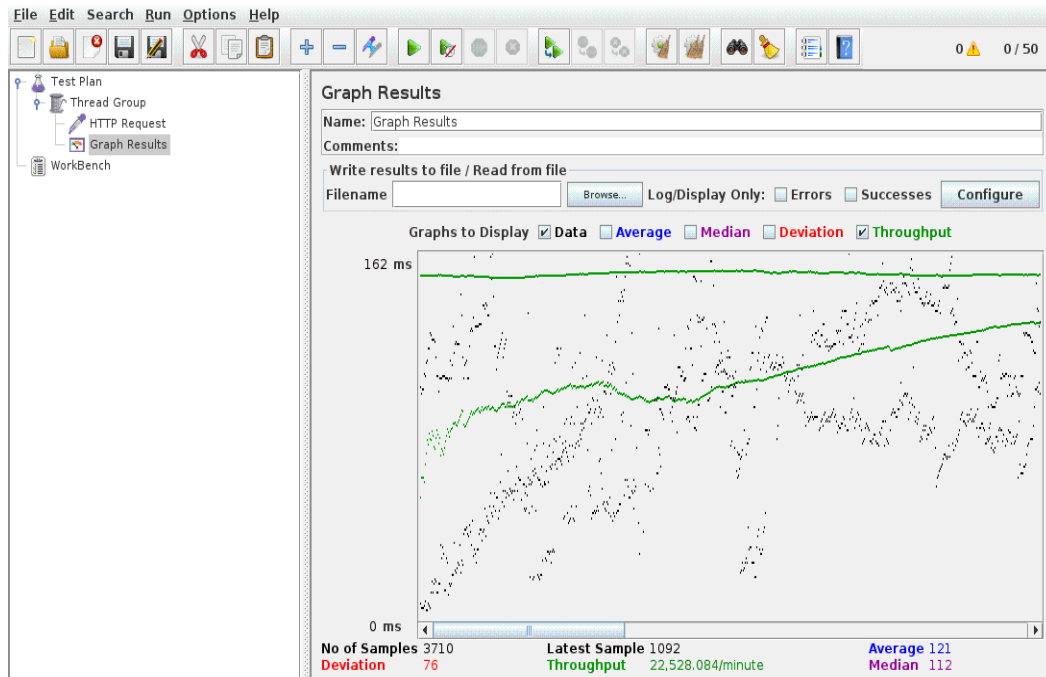
Gambar 6.1 Contoh data pengujian menggunakan top

root@worker1: /home/worker1						
root@worker1:/home/worker1# free -m						
	total	used	free	shared	buff/cache	available
Mem:	669	170	273	3	224	393
Swap:	727	0	727			
root@worker1:/home/worker1#						
root@worker2: /home/worker2						
root@worker2:/home/worker2# free -m						
	total	used	free	shared	buff/cache	available
Mem:	669	176	195	3	297	388
Swap:	1023	0	1023			
root@worker2:/home/worker2#						

Gambar 6.2 Contoh data pengujian Menggunakan free

### 6.1.1.2 Pengujian Throughput

*Throughput* antara client dengan server diuji menggunakan aplikasi JMeter™ dimana menggunakan *ThreadGroup* dengan *Client* yang telah ditentukan. Dalam *ThreadGroup*, *Client* secara terus menerus melakukan *load/stress* pada web server. Jumlah *Client* sama dengan pengujian penggunaan *memory* dan *CPU* pada Tabel 6.1. Dalam JMeter™ juga terdapat modul untuk pengecekan *Throughput* beserta grafiknya.



**Gambar 6.3 Contoh data pengujian throughput menggunakan Grafik JMeter™**

### **6.1.2 Skenario Pengujian *Fail Over***

Dalam pengujian *Fail Over* dilakukan dengan cara pengujian *Mapping*. Pengujian *Mapping* adalah pengecekan dengan cara mengakses website dengan menggunakan web browser apakah sesuai dengan hasil *output* proses dari program.

### **6.1.3 Skenario Pengujian *Load balancing***

Dalam pengujian *load balancing* dilakukan dengan cara pengujian *Mapping*. Pengujian *Mapping* adalah pengecekan dengan cara mengakses website dengan menggunakan web browser apakah sesuai dengan hasil *output* proses dari program.

### **6.1.4 Skenario Pengujian Fungsional**

Dalam pengujian *fungsional* dilakukan dengan cara memonitor pengiriman dan penerimaan data, *output* program dan Log program yang telah menyimpan hasil *output* program.

## **6.2 Analisa Pengujian**

Analisa hasil dilakukan setelah melakukan skenario pengujian yang telah dirancang sebelumnya. Sehingga dapat mendapatkan hasil yang sesuai.

### **6.2.1 Analisa Pengujian Kinerja**

Pada subab ini menjelaskan analisa dari pengujian kinerja sistem yang telah di uji dengan skenario pengujian yang telah di tetapkan.



#### 6.1.4.1

##### 6.1.4.1 Analisa Pengujian *Memory* dan *CPU* (Resource Utilization)

*Memory* dan *CPU* merupakan komponen penting dalam *host* dan dapat dijadikan sebagai acuan dalam menentukan kinerja *host* tersebut. Semakin efektif penggunaan *memory* dan *CPU* pada *host*, semakin bagus pula kinerjanya. Sebuah *host* dapat dikatakan efektif, jika dapat menggunakan *CPU* dan *memory* secukupnya untuk mendapat hasil yang maksimal. Untuk mengetahui pengaruh loadbalancer pada *host* terhadap *CPU* dan *memory*, dilakukan pengujian dengan menggunakan load/stress client dengan beberapa skenario. Berikut hasil pengujian penggunaan *memory* dan *CPU* pada *host* web server Docker Swarm.

Number of clients	CPU Utilization (%)		Memory Utilization (Mb)	
	Worker Node 1	Worker Node 2	Worker Node 1	Worker Node 2
50	3.8	2.4	214	185
250	4.6	3.0	214	187
500	8.6	4.0	214	187

**Tabel 6.2 Hasil Pengujian Menggunakan *CPU* dan *Memory***

##### 6.1.4.2 Analisa Pengujian Throughput

*Throughput* merupakan parameter yang umum dijadikan acuan dalam menentukan kualitas koneksi jaringan. Semakin tinggi nilai *throughput* semakin bagus juga kualitas jaringannya. *Throughput* dapat dijadikan acuan dalam melihat kinerja *Host* Web Server dalam menghubungkan antara jumlah client yang dapat di handle . Berikut hasil pengujian *throughput* dengan menggunakan load/stress client.

Jumlah Client	Troughtput
50	48,059904 Mb/s
250	49,091328 Mb/s
500	43,529728 Mb/s

**Tabel 6.3 Hasil Pengujian Troughtput pada *host* server**

Dalam hasil pengujian Troughtput pada Tabel 6.3 di jelaskan bahwa penurunan Troughtput berbanding lurus dengan penambahan jumlah client. Tetapi ketika jumlah Client 250 ada kenaikan troughput yang disebabkan karena faktor penurunan penggunaan data oleh *host* dan pengguna jaringan pada jaringan yang sama pada saat pengujian.

## 6.2.2 Analisa Pengujian *Fail Over*

Pada subab ini menjelaskan hasil *output fail over* dari program ketika salah satu *Node Worker down* dan Semua *Node Worker Down*.

### 6.2.3.1 Pengujian *Fail Over Node Worker1 Down*

Pada Subab ini menjelaskan hasil *output fail over* dari program ketika *Node Worker1* Aktif dan *Node Worker2* aktif . Hasil *output* dari program dapat dilihat pada Gambar 6.4 dan hasil pengecekan dalam web browser dapat dilihat pada Gambar 6.5 .

```
root@manager: /home/manager/manager_script

[ ok ] Reloading nginx: nginx.
[Threshold] 12
[Load Balancer] worker2 2018-06-04 14:06:55 192.168.56.103 389 [Waktu Cek] 2018-06-04 14:06:59
[Host] Worker1 :Down
[ ok ] Reloading nginx: nginx.
[Received File] worker2_memory.json 2018-06-04 14:06:57
File worker2_memory.json Downloaded

[Threshold] 12
[Load Balancer] worker2 2018-06-04 14:07:00 192.168.56.103 389 [Waktu Cek] 2018-06-04 14:07:02
[Host] Worker1 :Down
[ ok ] Reloading nginx: nginx.
[Threshold] 12
[Load Balancer] worker2 2018-06-04 14:07:00 192.168.56.103 389 [Waktu Cek] 2018-06-04 14:07:05
[Host] Worker1 :Down
[Received File] worker2_memory.json 2018-06-04 14:07:02
[ ok ] Reloading nginx: nginx.
File worker2_memory.json Downloaded

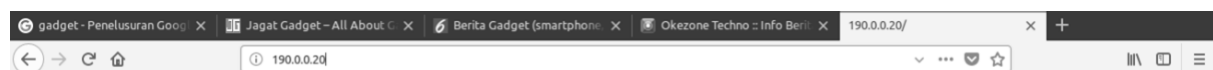
[Threshold] 12
[Load Balancer] worker2 2018-06-04 14:07:05 192.168.56.103 389 [Waktu Cek] 2018-06-04 14:07:08
[Host] Worker1 :Down
[ ok ] Reloading nginx: nginx.
[Received File] worker2_memory.json 2018-06-04 14:07:07
**Data 350H Gagal Di Load**
[Threshold] 12
[Load Balancer] worker2 2018-06-04 14:07:05 192.168.56.103 389 [Waktu Cek] 2018-06-04 14:07:11
[Host] Worker1 :Down
File worker2_memory.json Downloaded

[ ok ] Reloading nginx: nginx.
[Threshold] 12
[Load Balancer] worker2 2018-06-04 14:07:10 192.168.56.103 389 [Waktu Cek] 2018-06-04 14:07:14
[Host] Worker1 :Down
[ ok ] Reloading nginx: nginx.
]
```

Gambar 6.4 *Output program fail over ketika Node Worker1 Down*

Dari Gambar 6.4 menerangkan bahwa *fail over* dapat di jalankan dan dapat di monitor oleh layar. Ketika *Node Worker1 Down*, maka beban *traffic* akan di bebaskan pada *Node Worker2*. Dan terdapat output program berupa warning yang menyatakan bahwa *Node Worker1 Down*. Dan juga ketika dilakukan pengecekan *Mapping* pada web browser bahwa beban *traffic* di arahkan pada *Node Worker2* sesuai dengan *output* program pada gambar 6.5.

Gambar 6.5 *Output Mapping Menggunakan Web Browser dengan Node Worker1 Down*



Name HOST:Worker2

### 6.2.3.2 Pengujian Fail Over Node Worker1 Aktif dan Node Worker2 Down

Pada Subbab ini menjelaskan hasil *output fail over* dari program ketika *Node Worker1* Aktif dan *Node Worker2 down*. Hasil *output* dari program dapat dilihat pada Gambar 6.6 dan hasil pengecekan dalam web browser dapat dilihat pada Gambar 6.7.

**Gambar 6.6 Output Main Program fail over Node Worker2 Down**

```
root@manager: /home/manager/manager_script
File worker1_memory.json Downloaded

[Threshold] 12
[Load Balancer] worker1 2018-06-04 14:08:17 192.168.56.104 394 [Waktu Cek] 2018-06-04 14:08:21
[Host] Worker2 : Down
[ ok ] Reloading nginx: nginx.
[Received File] worker1_memory.json 2018-06-04 14:08:19
**Data JSON Gagal Di Load**
[Threshold] 12
[Load Balancer] worker1 2018-06-04 14:08:17 192.168.56.104 394 [Waktu Cek] 2018-06-04 14:08:24
[Host] Worker2 : Down
File worker1_memory.json Downloaded

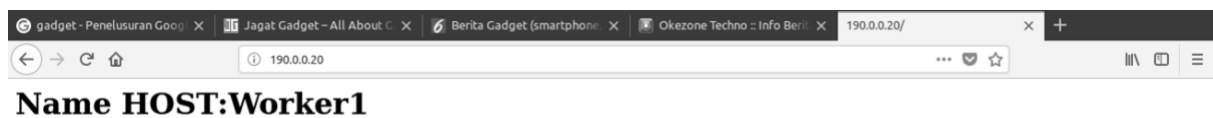
[ ok ] Reloading nginx: nginx.
[Threshold] 12
[Load Balancer] worker1 2018-06-04 14:08:22 192.168.56.104 394 [Waktu Cek] 2018-06-04 14:08:27
[Host] Worker2 : Down
[ ok ] Reloading nginx: nginx.
[Received File] worker1_memory.json 2018-06-04 14:08:24
File worker1_memory.json Downloaded

[Threshold] 12
[Load Balancer] worker1 2018-06-04 14:08:27 192.168.56.104 394 [Waktu Cek] 2018-06-04 14:08:30
[Host] Worker2 : Down
[ ok ] Reloading nginx: nginx.
[Threshold] 12
[Load Balancer] worker1 2018-06-04 14:08:27 192.168.56.104 394 [Waktu Cek] 2018-06-04 14:08:33
[Host] Worker2 : Down
[Received File] worker1_memory.json 2018-06-04 14:08:29
[ ok ] Reloading nginx: nginx.
File worker1_memory.json Downloaded

[Threshold] 12
[Load Balancer] worker1 2018-06-04 14:08:32 192.168.56.104 394 [Waktu Cek] 2018-06-04 14:08:36
[Host] Worker2 : Down
[ ok ] Reloading nginx: nginx.
```

Dari Gambar 6.6 menerangkan bahwa *fail over* dapat di jalankan dan dapat di monitor oleh layar. Ketika *Node Worker2 Down*, maka beban *traffic* akan di bebaskan pada *Node Worker1*. Dan terdapat output program berupa warning yang menyatakan bahwa *Node Worker2 Down*. Dan juga ketika dilakukan pengecekan *Mapping* pada web browser bahwa beban *traffic* di arahkan pada *Node Worker1* sesuai dengan *output* program pada Gambar 6.7.

**Gambar 6.7 Output Mapping Menggunakan Web Browser dengan Worker2 Down**



Name HOST:Worker1

### 6.2.3.3 Pengujian Fail Over Semua Node Down

Pada Subab ini menjelaskan hasil *output fail over* dari program ketika semua *Node Worker down* sehingga diambil alih oleh *Node Manager*. Hasil *output* dari program dapat dilihat pada Gambar 6.8 dan hasil pengecekan dalam web browser dapat dilihat pada Gambar 6.9.

Gambar 6.8 Output Main Program *fail over* Semua Node Worker Down

```
root@manager: /home/manager/manager_script
-----INTERFACES-----
lo : 127.0.0.1
enp0s3 : 192.168.56.101
enp0s8 : 190.0.0.20
docker_gwbridge : 172.18.0.1
docker0 : 172.17.0.1
-----
Pilih IP Host Manager yang terdaftar dalam Intefaces
Enter untuk Default Interface 'lo' (127.0.0.1) :192.168.56.101

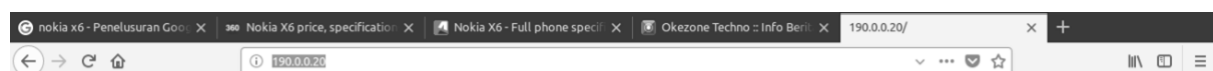
Masukan Waktu untuk mengitung Host Down dalam setiap X second
Enter untuk Default (30s) : 5

Masukan Waktu untuk pengecekan Loadbalancer dalam setiap X second
Enter untuk Default (3s) :

IP Server : 192.168.56.101
Waktu Host Down : 5
Waktu Pengecekan Loadbalancer : 3
-----PROGRAM START-----
**SEMUA HOST DOWN **
[ ok ] Reloading nginx: nginx.
**SEMUA HOST DOWN **
[ ok ] Reloading nginx: nginx.
**SEMUA HOST DOWN **
[ ok ] Reloading nginx: nginx.
**SEMUA HOST DOWN **
[ ok ] Reloading nginx: nginx.
**SEMUA HOST DOWN **
[ ok ] Reloading nginx: nginx.
**SEMUA HOST DOWN **
[ ok ] Reloading nginx: nginx.
**SEMUA HOST DOWN **
[ ok ] Reloading nginx: nginx.
**SEMUA HOST DOWN **
[ ok ] Reloading nginx: nginx.
```

Dari Gambar 6.8 menerangkan bahwa *fail over* dapat di jalankan dan dapat di monitor oleh layar. Ketika semua *Node Worker down*, maka beban *traffic* akan di bebaskan pada *Node manager*. Dan terdapat output program berupa warning yang menyatakan bahwa semua *Node Down*. Dan juga ketika dilakukan pengecekan *Mapping* pada web browser bahwa beban *traffic* di arahkan pada *Node Manager* sesuai dengan *output* program pada Gambar 6.9.

Gambar 6.9 Output Mapping Menggunakan Web Browser



Semua Host Down

### 6.2.3 Analisa Pengujian *Load balancing*

Pada Subab ini menjelaskan hasil *output Load balancing* dari program ketika *memory Node Worker1* lebih besar atau sama dengan dari *Node Worker* lainnya. Dan Mengakses Web Browser dengan alamat manager untuk pengecekan *Mapping* adalah benar.

```
root@manager:/home/manager/manager_script
[Load Balancer] worker1 2018-06-04 14:04:08 192.168.56.104 396 [Waktu Cek] 2018-06-04 14:05:50
[Received File] worker2_memory.json 2018-06-04 14:05:50
[ ok ] Reloading nginx: nginx.
File worker2_memory.json Downloaded

[Threshold] 12
[Load Balancer] worker1 2018-06-04 14:04:08 192.168.56.104 396 [Waktu Cek] 2018-06-04 14:05:53
[Received File] worker1_memory.json 2018-06-04 14:05:52
[ ok ] Reloading nginx: nginx.
File worker1_memory.json Downloaded

[Received File] worker2_memory.json 2018-06-04 14:05:55
**Data JSON Gagal Di Load**
[Threshold] 12
[Load Balancer] worker1 2018-06-04 14:04:08 192.168.56.104 396 [Waktu Cek] 2018-06-04 14:05:56
File worker2_memory.json Downloaded

[ ok ] Reloading nginx: nginx.
[Received File] worker1_memory.json 2018-06-04 14:05:57
**Data JSON Gagal Di Load**
[Threshold] 12
[Load Balancer] worker1 2018-06-04 14:04:08 192.168.56.104 396 [Waktu Cek] 2018-06-04 14:05:59
File worker1_memory.json Downloaded

[ ok ] Reloading nginx: nginx.
[Received File] worker2_memory.json 2018-06-04 14:06:00
File worker2_memory.json Downloaded

[Threshold] 12
[Load Balancer] worker1 2018-06-04 14:04:08 192.168.56.104 396 [Waktu Cek] 2018-06-04 14:06:02
[ ok ] Reloading nginx: nginx.
```

**Gambar 6.10 *Output Load balancing* dan pengecekan *Mapping* menggunakan web browser**

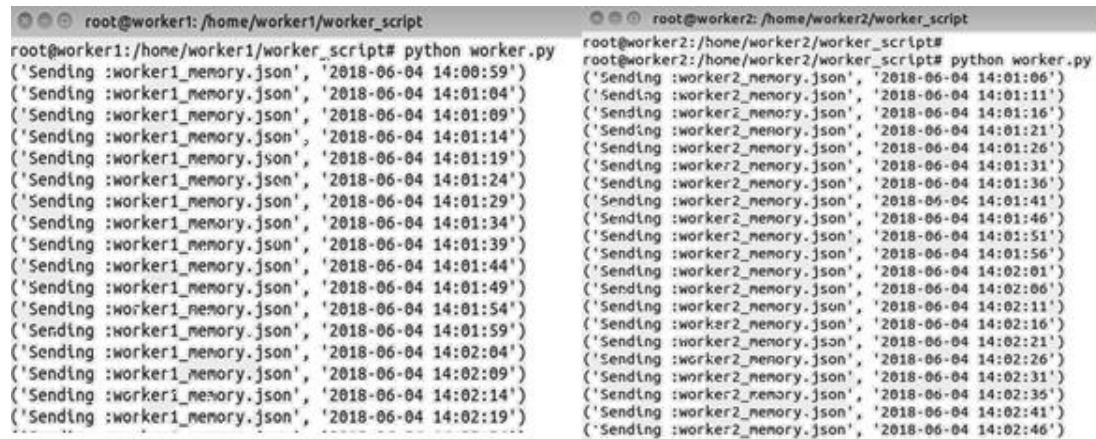
Dari Gambar 6.10 menerangkan bahwa *load balancing* berdasarkan sumber daya *memory* dapat di jalankan dan dapat di monitor oleh layar. Dan juga ketika dilakukan pengecekan *Mapping* pada web browser bahwa beban *traffic* di arahkan pada *Node Worker1* sesuai dengan *output* program pada gambar 6.11 .



**Gambar 6.11 *Output mapping load balancing* berdasarkan sumber daya *memory***

## 6.2.4 Analisa Pengujian Fungsional

Pada Subab ini menjelaskan output monitoring pengiriman data oleh *Node Worker* ke *Node Manager*. Dalam output monitoring pengiriman data dari *Node Worker* terdapat waktu pengiriman data dan data yang telah dikirim dapat dilihat pada Gambar 6.13.

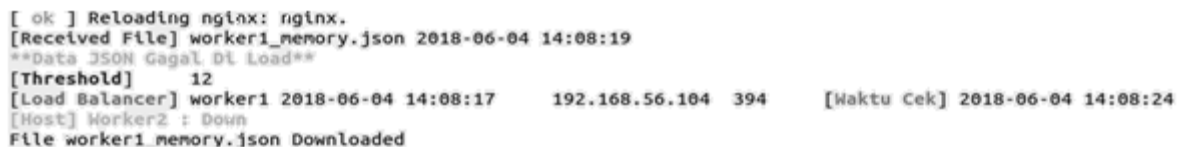


```
root@worker1:/home/worker1/worker_script# python worker.py
('Sending :worker1_memory.json', '2018-06-04 14:00:59')
('Sending :worker1_memory.json', '2018-06-04 14:01:04')
('Sending :worker1_memory.json', '2018-06-04 14:01:09')
('Sending :worker1_memory.json', '2018-06-04 14:01:14')
('Sending :worker1_memory.json', '2018-06-04 14:01:19')
('Sending :worker1_memory.json', '2018-06-04 14:01:24')
('Sending :worker1_memory.json', '2018-06-04 14:01:29')
('Sending :worker1_memory.json', '2018-06-04 14:01:34')
('Sending :worker1_memory.json', '2018-06-04 14:01:39')
('Sending :worker1_memory.json', '2018-06-04 14:01:44')
('Sending :worker1_memory.json', '2018-06-04 14:01:49')
('Sending :worker1_memory.json', '2018-06-04 14:01:54')
('Sending :worker1_memory.json', '2018-06-04 14:01:59')
('Sending :worker1_memory.json', '2018-06-04 14:02:04')
('Sending :worker1_memory.json', '2018-06-04 14:02:09')
('Sending :worker1_memory.json', '2018-06-04 14:02:14')
('Sending :worker1_memory.json', '2018-06-04 14:02:19')

root@worker2:/home/worker2/worker_script# python worker.py
('Sending :worker2_memory.json', '2018-06-04 14:01:06')
('Sending :worker2_memory.json', '2018-06-04 14:01:11')
('Sending :worker2_memory.json', '2018-06-04 14:01:16')
('Sending :worker2_memory.json', '2018-06-04 14:01:21')
('Sending :worker2_memory.json', '2018-06-04 14:01:26')
('Sending :worker2_memory.json', '2018-06-04 14:01:31')
('Sending :worker2_memory.json', '2018-06-04 14:01:36')
('Sending :worker2_memory.json', '2018-06-04 14:01:41')
('Sending :worker2_memory.json', '2018-06-04 14:01:46')
('Sending :worker2_memory.json', '2018-06-04 14:01:51')
('Sending :worker2_memory.json', '2018-06-04 14:01:56')
('Sending :worker2_memory.json', '2018-06-04 14:02:01')
('Sending :worker2_memory.json', '2018-06-04 14:02:06')
('Sending :worker2_memory.json', '2018-06-04 14:02:11')
('Sending :worker2_memory.json', '2018-06-04 14:02:16')
('Sending :worker2_memory.json', '2018-06-04 14:02:21')
('Sending :worker2_memory.json', '2018-06-04 14:02:26')
('Sending :worker2_memory.json', '2018-06-04 14:02:31')
('Sending :worker2_memory.json', '2018-06-04 14:02:36')
('Sending :worker2_memory.json', '2018-06-04 14:02:41')
('Sending :worker2_memory.json', '2018-06-04 14:02:46')
```

Gambar 6.12 *Output monitoring* pengiriman data oleh *Node Worker*

Dan pada *Output monitoring* penerimaan data oleh *Node Manager* dapat dilihat pada Gambar 6.14 beserta waktu menerima data dari *Node Worker*.



```
[ ok ] Reloading nginx: nginx.
[Received File] worker1_memory.json 2018-06-04 14:08:19
**Data JSON Gagal Di Load**
[Threshold] 12
[Load Balancer] worker1 2018-06-04 14:08:17 192.168.56.104 394 [Waktu Cek] 2018-06-04 14:08:24
[Host] Worker2 : Down
File worker1_memory.json Downloaded
```

Gambar 6.13 Potongan *output monitoring* penerimaan data oleh *Node Manager*

## BAB 7

### KESIMPULAN DAN SARAN

#### 7.1 Kesimpulan

Dari hasil analisis percobaan yang telah dilakukan terhadap data uji yang telah diujikan, penulis menyimpulkan bahwa :

1. *Fail over* berdasarkan perbandingan waktu dan *load balancing* berdasarkan sumber daya *memory* dapat di gunakan atau di implementasikan dalam Docker Swarm;
2. Teknik *load balancing* dapat mendistribusikan beban web server dapat ditunjukan dengan hasil pengujian yang telah dilakukan;
3. Teknik *fail over* dapat mem-*forward* beban *traffic* ke server lainnya ketika server yang dituju *down* sehingga sistem memiliki ketersediaan yang tinggi;
4. Teknik *load balancing* dan *fail over* bertujuan agar mengurangi gegalan kinerja dari web server.

#### 7.2 Saran

Saran yang diberikan untuk pengembangan kedepan *fail over* dan *load balancing* pada Docker Swarm antara lain:

1. Dalam Metode *load balancing* pada Docker Swarm , seharusnya dapat membagi pembuatan container secara rata pada setiap host ketika salah 1 container rusak.
2. Menggunakan lebih dari 1 parameter agar kinerja *load balancer* lebih baik dalam membagi beban dalam setiap *node*.

## DAFTAR PUSTAKA

- Adiputra, F. (2015). CONTAINER DAN DOCKER: TEKNIK VIRTUALISASI DALAM PENGELOLAAN BANYAK APLIKASI WEB. 4(3).
- Dewobroto. (2009). Jurnal Jarkom. 1.
- Docker. (2013, January 1). Docker. (Docker) Dipetik February 15, 2018, dari Docker Web Site: <https://www.Docker.com/what-container>
- JSON ORG. (2017, Juny 2). JSON. (JSON Org) Dipetik February 15, 2018, dari JSON Web Site: <https://www.json.org/json-id.html>
- Julianto, R. (2017). Implementasi Load Balancing Di Web Server Menggunakan Metode. 1.
- Kusuma1, T. P., Dr. Ir. Rendy Munadi, M., Danu Dwi Sanjoyo., S., & M.T. (2017). IMPLEMENTASI DAN ANALISIS COMPUTER CLUSTERING SYSTEM DENGAN MENGGUNAKAN VIRTUALISASI DOCKER. *Universitas Telelkomonukasi Indonesia*.
- M. Agung Nugroho, R. K. (2016). ANALISIS KINERJA PENERAPAN CONTAINER UNTUK LOAD BALANCING WEB SERVER PADA RASPBERRY PI. *Stimik Akakom*.
- NGIX Software.INC. (2014, January 1). NGIX. (NGIX Software.INC) Dipetik February 15, 2018, dari NGIX: <https://www.nginx.com/blog/>
- Nugroho, M. (2016). ANALISIS KINERJA PENERAPAN CONTAINER UNTUK LOAD BALANCING WEB SERVER PADA RASPBERRY PI. 01(02).
- Singh, H. (2015). WSQ: Web Server Queueing Algorithm for Dynamic Load Balancing.



## LAMPIRAN

### A. Source Code

#### Worker.py

No	Worker.py
1	from socket import *
2	import threading
3	import time
4	import datetime
5	from subprocess import *
6	import json
7	
8	#inisialisasi waktu kirim dalam second
9	waktu_kirim=5.0
10	waktu_penulis_data_memory=1.0
11	
12	def kirim():
13	threading.Timer(waktu_kirim, kirim).start()
14	try:
15	s = socket(AF_INET, SOCK_DGRAM)
16	host = "192.168.56.101"
17	port = 9999
18	buf = 1024
19	addr = (host, port)
20	ts = time.time()
21	st = datetime.datetime.fromtimestamp(ts).strftime('%Y-%m-%d
22	%H:%M:%S')
23	hostname=check_output("hostname")
24	file_name =(hostname.replace("\n", "")+'_memory.json')
25	s.sendto((file_name.encode()), addr)
26	
27	f=open(file_name, "rb")
28	data = f.read(buf)
29	while (data):
30	if(s.sendto(data, addr)):
31	data = f.read(buf)
32	print ("Sending :"+file_name,st)
33	with open('output_dataupload.txt', 'a') as
34	dataoutput:
35	dataoutput.write("\n"+file_name+"\t"+st)
36	dataoutput.close()
37	s.close()
38	except IOError :
39	print("Membuat File Json")
40	
41	
42	def menulis_data_memory():
43	threading.Timer(waktu_penulis_data_memory,
44	menulis_data_memory).start()
45	
46	hostname=check_output("hostname")
47	IP=check_output("ifconfig enp0s3   grep 'inet'   cut -d: -f2
48	awk '{ print \$2}'", shell=True)
49	Waktu=check_output("(date -d '+0 hour' '+%F %T')", shell=True)
50	Memory=check_output("(free -m   awk 'FNR == 2 {print
51	\$7}')", shell=True)
52	
53	data_json_host=[
54	{
55	>NamaWorker": ""+hostname.replace("\n", "")+"",

56	"IP":""+IP.replace("\n","")+""
57	"Waktu":""+Waktu.replace("\n","")+""
58	"freeMemory":""+Memory.replace("\n","")+""
59	}
60	]
61	with open (hostname.replace("\n","")+'_memory.json','w') as
62	data_json:
63	json.dump(data_json_host, data_json)
64	
65	def handle_format():
66	with open('output_dataupload.txt', 'w') as dataupload:
67	dataupload.write("\n" + "filename" + "\t\t" + "Waktu Send")
68	dataupload.close()
69	
70	if __name__ == '__main__':
71	handle_format()
72	t1 = threading.Thread(target=menulis_data_memory)
73	t2 = threading.Thread(target=kirim)
74	t3 = threading.Thread(target=menulis_data_memory)
75	t3.start()
76	t1.start()
77	t2.start()

### Manager.py

No	Manager.py
1	#!/ python3
2	
3	import json
4	from socket import *
5	import datetime
6	import time
7	import threading
8	from subprocess import *
9	import ipaddress
10	import array
11	import struct
12	import fcntl
13	
14	#Inisialisasi waktu dalam second dan jumlah host
15	set_timeout=1
16	time_delta_Second=''
17	IP_Server=""
18	waktu_cek_loadbalancer=3
19	#Inisialisasi viarable global
20	backend=''
21	Backend_nama_worker1=''
22	Backend_nama_worker2=''
23	ip_dict={}
24	boolean_input=False
25	boolean_threshold=False
26	boolean_input_IP=False
27	NamaWorker = ''
28	Waktu = ''
29	IP = ''
30	Memory = ''
31	threshold=''
32	String_caution=''
33	
34	class bcolors: #Class Color Ansi

```

35
36 OKBLUE = '\033[1;34m'#Blue
37 RECIEVED_CYAN = '\033[1;36m'#Cyan
38 threshold_color='\033[1;33m'#Orange
39 str_waktucek_color='\033[1;35m'#Purple
40 start_program='\033[1;32m'#Green
41 interface_color='\033[0;32m'#Green
42 WARNING = '\033[1;31m'#RED
43 ENDC = '\033[0m'#Reset ANSI COLOR
44
45 def receiver(): # Method Reciever
46     global boolean_input, IP_Server
47     while (True):
48         if (boolean_input == True):
49             ts = time.time()
50             st = datetime.datetime.fromtimestamp(ts).strftime('%Y-%m-
51 %d %H:%M:%S')
52             st_file =
53 datetime.datetime.fromtimestamp(ts).strftime('%Y-%m-%d')
54             host = str(IP_Server)
55             port = 9999
56             s = socket(AF_INET, SOCK_DGRAM)
57             s.setsockopt(SOL_SOCKET, SO_REUSEADDR, 1)
58             s.bind((host, port))
59             addr = (host, port)
60             buf = 1024
61
62             data, addr = s.recvfrom(buf)
63             filename = data.decode("ascii")
64             print "[" + bcolors.RECIEVED_CYAN + "Received File" +
65 bcolors.ENDC + "]" + filename + " " + st)
66             f = open(filename, 'wb')
67
68             data, addr = s.recvfrom(buf)
69             try:
70                 while (data):
71                     f.write(data)
72                     s.settimeout(set_timeout)
73                     data, addr = s.recvfrom(buf)
74             except timeout:
75                 with open('output_datasave_' + st_file + '.txt', 'a')
76 as dataoutput:
77                     dataoutput.write("\n" + filename + "\t" + st)
78                     dataoutput.close()
79                     f.close()
80                     s.close()
81                     print("File " + filename + " Downloaded \n")
82
83 def Cek_Value():#Method Check Value
84     global
85 NamaWorker, Waktu, IP, Memory, backend, waktu_cekvalue, String_caution
86     ts = time.time()
87     st = datetime.datetime.fromtimestamp(ts).strftime('%Y-%m-%d
88 %H:%M:%S')
89     st_file = datetime.datetime.fromtimestamp(ts).strftime('%Y-%m-
90 %d')
91     HostTime = datetime.datetime.strptime(st, '%Y-%m-%d %H:%M:%S')
92     threading.Timer(waktu_cek_loadbalancer, Cek_Value).start()#Waktu
93 Threading
94     if(boolean_input==True):
95         try:

```

```

96         if (boolean_threshold==True):
97             data_worker1 = json.load(open('worker1_memory.json'))
98     # Open file
99             data_worker2 = json.load(open('worker2_memory.json'))
100    # Open file
101            Worker1TimeMoreXsecond =
102    datetime.datetime.strptime(data_worker1[0]["Waktu"], '%Y-%m-%d
103    %H:%M:%S') + datetime.timedelta(seconds=time_delta_Second)
104            Worker2TimeMoreXsecond =
105    datetime.datetime.strptime(data_worker2[0]["Waktu"], '%Y-%m-%d
106    %H:%M:%S') + datetime.timedelta(seconds=time_delta_Second)
107            # Cek Waktu (Jika Semua Server UP )
108            if (Worker1TimeMoreXsecond >= HostTime and
109    Worker2TimeMoreXsecond >= HostTime):
110                String_caution=""
111                # Cek Perpindahan Jika Server UP semua
112                if (int(data_worker1[0]["freeMemory"]) -
113    int(data_worker2[0]["freeMemory"]))==(threshold)):
114                    NamaWorker = data_worker1[0]["NamaWorker"]
115                    Waktu = data_worker1[0]["Waktu"]
116                    IP = data_worker1[0]["IP"]
117                    Memory = data_worker1[0]["freeMemory"]
118
119                elif (int(data_worker2[0]["freeMemory"]) -
120    int(data_worker1[0]["freeMemory"]))==(threshold)):
121                    NamaWorker = data_worker2[0]["NamaWorker"]
122                    Waktu = data_worker2[0]["Waktu"]
123                    IP = data_worker2[0]["IP"]
124                    Memory = data_worker2[0]["freeMemory"]
125                # Cek Waktu (Jika hanya host worker1 UP dan host
126    worker2 Down )
127                elif (Worker1TimeMoreXsecond > HostTime and
128    Worker2TimeMoreXsecond < HostTime):
129                    NamaWorker = data_worker1[0]["NamaWorker"]
130                    Waktu = data_worker1[0]["Waktu"]
131                    IP = data_worker1[0]["IP"]
132                    Memory = data_worker1[0]["freeMemory"]
133                    String_caution="[Host] Worker2 : Down"
134                # Cek Waktu (Jika hanya host worker1 down dan host
135    worker2 UP )
136                elif (Worker1TimeMoreXsecond < HostTime and
137    Worker2TimeMoreXsecond > HostTime):
138                    NamaWorker = data_worker2[0]["NamaWorker"]
139                    Waktu = data_worker2[0]["Waktu"]
140                    IP= data_worker2[0]["IP"]
141                    Memory = data_worker2[0]["freeMemory"]
142                    String_caution="[Host] Worker1 :Down"
143                # Cek Waktu (Jika hanya host worker1 Down dan host
144    worker2 Down )
145            else:
146                NamaWorker = 'Fail'
147                # print((waktu_sekarang_lebih))
148
149            elif(boolean_threshold==False):
150                print(bcolors.WARNING+"**Data Threshold Tidak
151    Ditemukan**"+bcolors.ENDC)
152
153
154            except ValueError :
155                print(bcolors.WARNING+"**Data JSON Gagal Di
156    Load**"+bcolors.ENDC)

```

```

157         except IOError :
158             print (bcolors.WARNING+"**Data                JSON                Tidak
159 Ditemukan**"+bcolors.ENDC)
160
161         #Penentuan Backend
162         if (NamaWorker==Backend_nama_worker1):
163             backend=1
164         elif (NamaWorker==Backend_nama_worker2):
165             backend=2
166         #OUTPUT
167         if (NamaWorker=='Fail'):
168             print (bcolors.WARNING+"**SEMUA                HOST                DOWN
169 **"+bcolors.ENDC)
170             with open('loadbalancer.conf', 'w') as loadbalancer:
171                 loadbalancer.write("upstream backend {ip_hash; server
172 "+IP_Server+":808"+str('0')+";}server {listen 80;location /
173 {proxy_pass http://backend;""}}")
174                 loadbalancer.close()
175                 call ("(Docker cp loadbalancer.conf $(Docker container ls
176 |grep 'nginx'|awk '{ print $1}' ):/etc/nginx/conf.d/loadbalancer.conf)",shell=True)
177                 call ("(Docker exec -it $(Docker container ls |grep
178 'nginx'|awk '{ print $1}') service nginx reload)",shell=True)
179                 #save output log
180
181
182         elif (NamaWorker!="" and NamaWorker!='Fail'):
183
184         print ("["+bcolors.threshold_color+"Threshold"+bcolors.ENDC+"]
185 \t"+str(threshold))
186             print ("["+bcolors.OKBLUE+"Load balancing"+bcolors.ENDC+"]
187 "+NamaWorker + "\t" + Waktu + "\t" + IP + "\t" + Memory
188 +"\t["+bcolors.str_waktucek_color+"Waktu Cek"+bcolors.ENDC+"] "+st
189 +"\n"+bcolors.WARNING+String_caution+bcolors.ENDC)
190             # make NGINX loadbalancer.conf
191             with open('loadbalancer.conf', 'w') as loadbalancer:
192                 loadbalancer.write("upstream
193 backend"+str(backend)+"{ip_hash; server
194 "+IP_Server+":808"+str(backend)+";}server {listen 80;location /
195 {proxy_pass http://backend"+str(backend)+";}}")
196                 loadbalancer.close()
197             # copy config dan reload NGINX agar berjalan dalam conf
198 yang baru
199             call ("(Docker cp loadbalancer.conf $(Docker container ls
200 |grep 'nginx'|awk '{ print $1}' ):/etc/nginx/conf.d/loadbalancer.conf)",shell=True)
201             call ("(Docker exec -it $(Docker container ls |grep
202 'nginx'|awk '{ print $1}') service nginx reload)",shell=True)
203             #save output log
204             with open('output_cekvalue'+st_file+'.txt', 'a') as f:
205                 f.write("\n"+NamaWorker + "\t\t" + Waktu + "\t" + IP
206 + "\t" + Memory + "\t" + st)
207                 f.close()
208             else:
209                 print (bcolors.WARNING+"**Program Waiting**"+bcolors.ENDC)
210
211
212 def threshold_method():# Method Set Threshold
213     global
214     threshold,boolean_threshold,NamaWorker,Waktu,IP,Memory,Backend_nama_
215     worker1,Backend_nama_worker2
216     while (True):
217         try:

```

```

218         if (boolean_threshold==False) :
219             data_worker1 = json.load(open('worker1_memory.json'))
220 # Open file
221             data_worker2 = json.load(open('worker2_memory.json'))
222 # Open file
223             Backend_nama_worker1=data_worker1[0]["NamaWorker"]
224             Backend_nama_worker2=data_worker2[0]["NamaWorker"]
225             threshold=(abs(int(data_worker1[0]["freeMemory"])-
226 int(data_worker2[0]["freeMemory"])))
227             #Cek Awal Server yang akan digunakan
228             if (data_worker1[0]["freeMemory"] >=
229 data_worker2[0]["freeMemory"]):
230                >NamaWorker = data_worker1[0]["NamaWorker"]
231                 Waktu = data_worker1[0]["Waktu"]
232                 IP = data_worker1[0]["IP"]
233                 Memory = data_worker1[0]["freeMemory"]
234
235             elif (data_worker1[0]["freeMemory"] <
236 data_worker2[0]["freeMemory"]):
237                >NamaWorker = data_worker2[0]["NamaWorker"]
238                 Waktu = data_worker2[0]["Waktu"]
239                 IP = data_worker2[0]["IP"]
240                 Memory = data_worker2[0]["freeMemory"]
241
242             boolean_threshold=True
243             break
244         else :
245             continue
246     except ValueError :
247         continue
248     except IOError :
249         continue
250     time.sleep(3)
251
252 def get_local_interfaces():#Method Check IP in Interface
253     global ip_dict
254     # Max possible bytes for interface result. Will truncate if more
255 than 4096 characters to describe interfaces.
256     MAX_BYTES = 4096
257     # We're going to make a blank byte array to operate on. This is
258 our fill char.
259     FILL_CHAR = b'\0'
260     # Command defined in ioctl.h for the system operation for get
261 iface list
262     # Defined at
263 https://code.woboq.org/qt5/include/bits/ioctls.h.html under
264 # /* Socket configuration controls. */ section.
265     SIOCGIFCONF = 0x8912
266     # Make a dgram socket to use as our file descriptor that we'll
267 operate on.
268     sock = socket(AF_INET, SOCK_DGRAM)
269     # Make a byte array with our fill character.
270     names = array.array('B', MAX_BYTES * FILL_CHAR)
271     # Get the address of our names byte array for use in our struct.
272     names_address, names_length = names.buffer_info()
273     # Create a mutable byte buffer to store the data in
274     mutable_byte_buffer = struct.pack('iL', MAX_BYTES, names_address)
275     # mutate our mutable_byte_buffer with the results of
276 get_iface_list.
277     # NOTE: mutated_byte_buffer is just a reference to
278 mutable_byte_buffer - for the sake of clarity we've defined them as

```

```

279     # separate variables, however they are the same address space -
280     that's how fcntl.ioctl() works since the mutate_flag=True
281     # by default.
282     mutated_byte_buffer = fcntl.ioctl(sock.fileno(), SIOCGIFCONF,
283     mutable_byte_buffer)
284     # Get our max_bytes of our mutated byte buffer that points to the
285     names variable address space.
286     max_bytes_out, names_address_out = struct.unpack('iL',
287     mutated_byte_buffer)
288     # Convert names to a bytes array - keep in mind we've mutated the
289     names array, so now our bytes out should represent
290     # the bytes results of the get iface list ioctl command.
291     namestr = names.tostring()
292     namestr[:max_bytes_out]
293     bytes_out = namestr[:max_bytes_out]
294     # Each entry is 40 bytes long. The first 16 bytes are the name
295     string.
296     # the 20-24th bytes are IP address octet strings in byte form -
297     one for each byte.
298     # Don't know what 17-19 are, or bytes 25:40.
299     ip_dict = {}
300     for i in range(0, max_bytes_out, 40):
301         name = namestr[ i: i+16 ].split(FILL_CHAR, 1)[0]
302         name = name.decode('ascii')
303         ip_bytes = namestr[i+20:i+24]
304         full_addr = []
305         for netaddr in ip_bytes:
306             if isinstance(netaddr, int):
307                 full_addr.append(str(netaddr))
308             elif isinstance(netaddr, str):
309                 full_addr.append(str(ord(netaddr)))
310         ip_dict[name] = '.'.join(full_addr)
311     return ip_dict
312
313 def save_log():#Method Create Log
314     ts = time.time()
315     st_file = datetime.datetime.fromtimestamp(ts).strftime('%Y-%m-
316     %d')
317
318     with open ('output_cekvalue_'+st_file+'.txt','w') as cekvalue:
319         cekvalue.write("NamaWorker" + "\t" + "Waktu Host" + "\t\t" +
320         "IP" + "\t\t" + "Memory" + "\t" + "Waktu Cek")
321         cekvalue.close()
322     with open('output_datasave_'+st_file+'.txt', 'w') as
323     datadownload:
324         datadownload.write("filename" + "\t\t" + "Waktu Recieve")
325         datadownload.close()
326
327 def input_user():#Method Input User
328     global
329     time_delta_Second,IP_Server,boolean_input,waktu_cek_loadbalancer,boo
330     lean_input_IP
331     print("Program By : Mohammad Rexa Mei Bella")
332     print("-----")
333     "+bcolors.start_program+"INTERFACES"+bcolors.ENDC+"-----"
334     "-----")
335     for x in ip_dict:
336         print(bcolors.interface_color+x+bcolors.ENDC
337         +'\t:\t'+ip_dict[x])
338         print("-----")
339     "-----")

```

```

340     while (True):
341         if(boolean_input==False):
342             try:
343                 while (boolean_input_IP==False):
344                     Input_user_IP=ipaddress.ip_address(input("Pilih
345 IP Host Manager yang terdaftar dalam Intefaces \nEnter untuk Default
346 Interface 'lo' (" + ip_dict['lo']+" ) :") or ip_dict['lo'] )
347                     for x in ip_dict:
348                         if (str(Input_user_IP) in ip_dict[x]):
349                             IP_Server = str(Input_user_IP)
350                             boolean_input_IP=True
351                             break
352                         elif (str(Input_user_IP) not in ip_dict[x]):
353                             continue
354                         if(str(Input_user_IP) not in ip_dict[x]):
355                             print(bcolors.WARNING+"**IP Tidak Tersedia
356 Dalam Interfaces**"+bcolors.ENDC)
357
358                             time_delta_Second = int(input("\nMasukan Waktu untuk
359 mengitung Host Down dalam setiap X second \nEnter untuk Default (30s)
360 : ")or 30)
361
362                             waktu_cek_loadbalancer=int(input("\nMasukan Waktu
363 untuk pengecekan Loadbalancer dalam setiap X second \nEnter untuk
364 Default (3s) : ")or 3)
365                             boolean_input=True
366                             except ValueError or NameError:
367                                 print(bcolors.WARNING+"**Invalid
368 Input**"+bcolors.ENDC)
369                                 continue
370                             else:
371                                 print("\nIP Server \t\t\t: "+IP_Server)
372                                 print("Waktu Host Down \t\t: "+str(time_delta_Second))
373                                 print("Waktu Pengecekan Loadbalancer \t:
374 "+str(waktu_cek_loadbalancer))
375                                 print("-----
376 "+bcolors.start_program+"PROGRAM START"+bcolors.ENDC+"-----
377 -----")
378                                 break
379
380 if __name__ == '__main__':#Main
381     t0=threading.Thread(target=input_user)
382     t1=threading.Thread(target=receiver)
383     t2=threading.Thread(target=threshold_method)
384     t3=threading.Thread(target=Cek_Value)
385     get_local_interfaces()
386     save_log()
387     t0.start()
388     t1.start()
389     t2.start()
390     t3.start()
391
392     t0.join()
393     t1.join()
394     t2.join()
395     t3.join()

```