IMPLEMENTASI SSL/TLS PADA WEBSITE BERBASIS ACTIVE-ACTIVE HIGH AVAILABILITY CLUSTER SERVER

LAPORAN PROYEK AKHIR
MATA KULIAH COMP6372– COMPUTER NETWORK

KELAS LA20

OLEH :

2301938724 - TEGUH ARI PARDAMEAN NAINGGOLAN

2301930551 - FEBRIAN NUGROHO

2301877590 - FELIX FILIPI

Semester Ganjil 2020

MALANG

LEMBAR PERSETUJUAN PROYEK AKHIR

IMPLEMENTASI SSL/TLS PADA WEBSITE BERBASIS ACTIVE-ACTIVE HIGH AVAILABILITY CLUSTER SERVER

MATA KULIAH COMP6372– COMPUTER NETWORK

KELAS LA20

Semester Ganjil 2020

Laporan akhir proyek di atas adalah benar karya dari :

(Teguh Ari Pardamean Nainggolan)	(Febrian Nugroho)	(Felix Filipi)
<2301938724>	<2301930551>	<2301877590 >

Malang, …… - ……. ……..

(M. Aldiki)

D6394

DAFTAR ISI

LATAR BELAKANG

Berdasarkan hasil survei dari kominfo, penetrasi penggunaan internet di Indonesia tahun 2019 – 2020 telah mencapai angka 73,7%. Dengan kata lain, pengguna internet di Indonesia diperkirakan telah mencapai angka 196,7 juta pengguna dari total keseluruhan penduduk di Indonesia yaitu 266.911.900 juta jiwa [1]. Dari data tersebut, terlihat jelas bahwa Indonesia telah berada di era baru yang mana internet serta teknologi menjadi hal yang lumrah bagi masyarakat awam. Melihat peluang ini, perusahaan-perusahaan di Indonesia mulai melakukan investasi besar besaran di bidang ini. Terbukti dengan jumlah *startup* pada tahun 2018 yang telah mencapai angka 992 perusahaan rintisan [2].

Dengan semakin meningkatnya penggunaan internet di Indonesia, maka kebutuhan akan konten pada internet pun semakin meningkat. Oleh karena itu, hadirlah *startup* untuk memenuhi kebutuhan itu, sayangnya kehadiran *startup* ini seringkali memiliki keterbatasan biaya dan performa, sehingga aplikasi buatan mereka seringkali mendapat label yang kurang baik. Keterbatasan ini ada berbagai macam, contoh yang kami ambil disini adalah permasalahan di sisi *back-end*, yaitu permasalahan di sisi server yang kurang memadai.

Permasalahan server yang kurang memadai ini sangat krusial bagi sebuah *startup*, yang mana apabila *server* tersebut tidak dapat menampung *request* dari *user*, maka aplikasi dari *startup* tersebut akan mengalami peningkatan frekuensi kegagalan saat *client* melakukan *request.* Bukan hanya itu saja contoh lain adalah apabila sebuah *server* mengalami *crash* dan *server* tersebut *down*. Maka tidak ada *server back-up* untuk menangani *request* dari *client*.

Untuk mengatasi permasalahan tersebut, maka sebuah *startup* perlu menggunakan konsep sistem *Active - Active High Availability Cluster Server*, yang mana konsep ini selain dapat membuat *server* mereka memiliki kinerja yang lebih kuat, *startup* tersebut dapat memiliki *back-up server* sekaligus. Sehingga saat salah satu *server* mereka *down*, aplikasi tersebut dapat tetap berjalan seperti biasa, dan *end-user* tidak perlu mengetahui mengenai hal tersebut.

Konsep ini sangat menguntungkan bagi sebuah *startup* dan tentunya sangat penting untuk diimplementasikan mengingat sebuah *startup* membutuhkan banyak sumber daya untuk berbagai keperluan di bidang lainnya. Sehingga daripada kita membeli komputer dengan spesifikasi yang sangat tinggi untuk menjadi sebuah *server*, lebih baik kita menggunakan anggaran tersebut untuk membeli beberapa komputer dengan spesifikasi menengah untuk digabungkan menjadi sebuah *server* yang kuat dan fleksibel. Untuk penjelasan lebih lanjut mengenai teknis dari konsep ini, akan kami jelaskan pada bab selanjutnya.

PROBLEM

Sebuah *server* memerlukan kinerja yang tinggi dan seringkali untuk mendapatkan kinerja yang tinggi tersebut dibutuhkan biaya yang tidak sedikit. Selain *server* harus kuat dalam mengatasi banyak *request* dari *client*, sebuah *server* juga harus fleksibel, maksudnya adalah *server* tersebut harus kuat untuk berjalan dalam setiap kondisi baik siang maupun malam. Karena sebuah *server* harus berjalan siang dan malam, maka *server* itu dapat menjadi *down* pada kondisi tertentu, sehingga diperlukan *backup server* yang dapat mengambil tugas *server* utama, saat terjadi hal yang tidak diinginkan.

TUJUAN

Tujuan dari penelitian ini adalah untuk membangun sebuah *cluster server* dengan fitur *High Availability (HA)* serta *Load Balancing (LB)* yang mana *server* tersebut nantinya akan diuji dengan mengimplementasikan sebuah *website* yang sudah diintegrasikan dengan *SSL/TLS* untuk mengukur kemampuan menampung *request* dari *server* tersebut.

METODE

Konfigurasi dan Pemasangan Infrastruktur *Server*

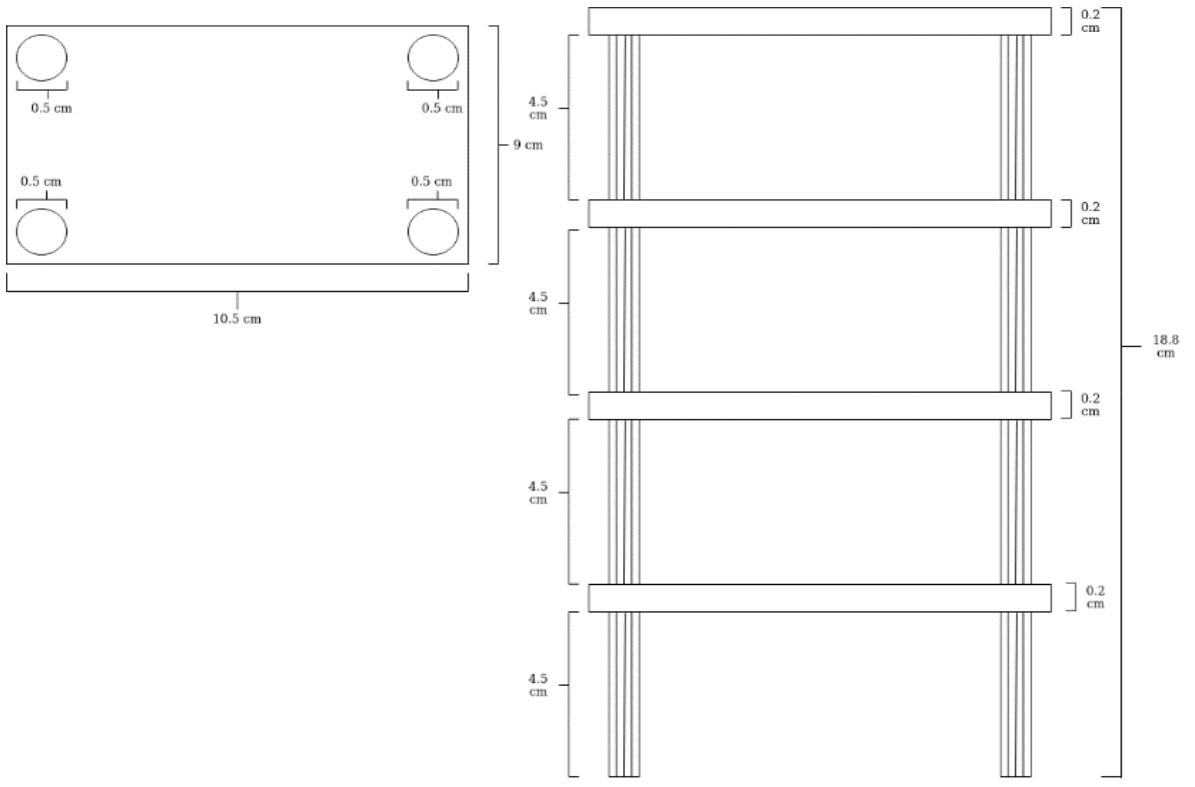
Tahap ini menjelaskan tentang langkah - langkah yang diperlukan dalam perancangan, konfigurasi, dan instalasi server kami. Secara rinci, akan dijelaskan tentang bagaimana cara kerja *server*, konfigurasi yang diperlukan, dan *software/hardware* yang diperlukan sehingga melancarkan jalannya penelitian ini. Adapun langkah - langkah yang kami lakukan adalah sebagai berikut.

Perancangan Sistem

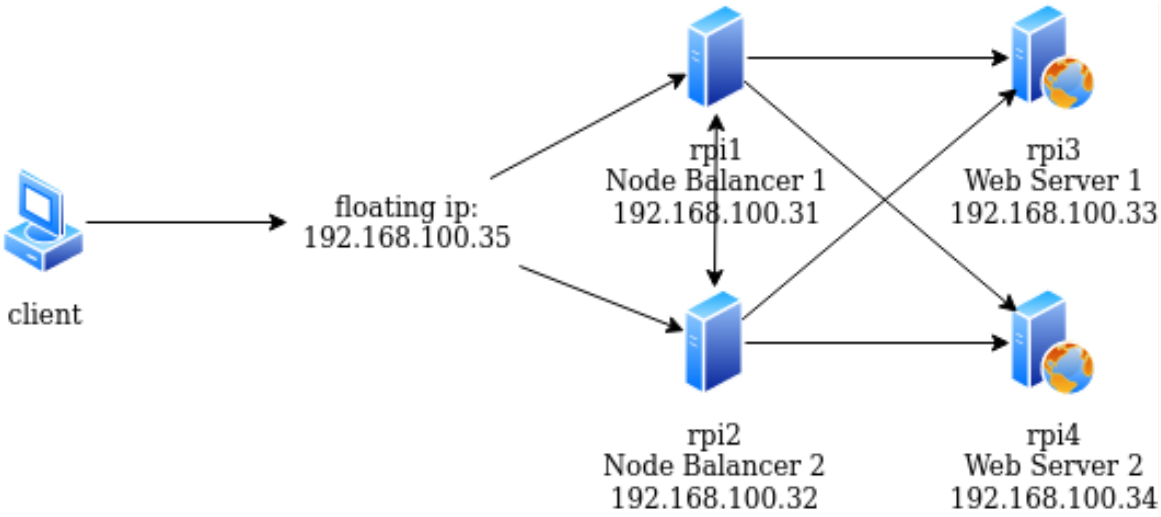
Pada tahap ini dilakukan perancangan infrastruktur, sistem, dan topologi jaringan yang dibutuhkan untuk penelitian kami.

Adapun komponen - komponen dan infrastruktur yang kami buat terdiri dari empat buah *mini-PC Raspberry Pi* yang terhubung satu sama lain dengan perantara *switch*. Kami juga melakukan *desain* serta membangun *case* sistem untuk tempat peletakan *mini-PC* yang akan dipasang.

Desain *case* sistem dari *server* kami adalah sebagai berikut.



Serta topologi dari *server* yang telah kami buat adalah sebagai berikut.



Infrastruktur server kami terdiri dari 4 buah *node*, yaitu: rpi1, rpi2, rpi3, rpi4. Rpi1 dan rpi2 bertugas sebagai *node* yang mengatur *load balancer* dan *node* ini dinamakan *master node*, sedangkan rpi 3 dan rpi 4 bertugas sebagai penyedia server *HTTP* dan *node* ini dinamakan *slave node*. Untuk konfigurasi *node balancing* pada *server*, kami menggunakan aplikasi berbasis *open source* bernama *HAProxy*. *Software HAProxy* ini sangat terkenal dalam kemampuannya melakukan *load balancing* karena konfigurasi-nya yang cepat, mudah dan efisien. *Software* ini juga kami gunakan sebagai *SSL/TLS Termination* untuk mempermudah manajemen pemasangan *SSL/TLS*. Sementara itu, konfigurasi *high availability* kami menggunakan aplikasi berbasis *open-source* yang bernama *keepalived*. Dan yang terakhir, kami menggunakan *software openSSL* yang akan menghasilkan teks *hash* yang digunakan untuk sertifikasi *SSL/TLS*.

Pemasangan *Operating System* (*OS*) pada masing-masing *PC*.

Untuk konfigurasi *Operating System*, kami memakai *Ubuntu Server OS* berbasis *Linux*. Pemasangan *OS Raspberry Pi* dilakukan dengan melakukan *booting* dengan *image OS* yang diinginkan. Pemasangan ini dilakukan dengan menggunakan *software BalenaEtcher* untuk melakukan *write* pada media *SD Card* sebagai memori internal dari *server* kami. Setelah melakukan *writing OS* pada *SD Card*, *SD Card* akan dipasang pada mini-*PC* sehingga *OS* berjalan.

Konfigurasi *static IP*, dan *hostname* pada masing-masing *node.*

Setelah itu kami melakukan konfigurasi *ip* statis pada setiap *mini-PC*. Tabel konfigurasi *host* dan *ip* server kami sebagai berikut.

Hostname	***Ip Address***	***Gateway***
rpi1	192.168.100.31	192.168.100.1
rpi2	192.168.100.32	192.168.100.1
rpi3	192.168.100.33	192.168.100.1
rpi4	192.168.100.34	192.168.100.1

Untuk mengganti *hostname*, kita dapat melakukannya dengan 2 cara:

1. *Edit* file pada konfigurasi nama *host*

```
$ sudo vim /etc/hostname
$ rpi1
```

2. *Edit* file pada konfigurasi *localhost ip*

```
$ sudo vim /etc/hosts
$ 127.0.0.1 localhost
$ 127.0.1.1 rpi1
```

Setelah mengganti *hostname* masing - masing *node*, kami mengganti konfigurasi *ip* menjadi *ip* statis dengan acuan pada tabel diatas. Adapun langkah - langkah yang diperlukan adalah:

1. Edit konfigurasi *netplan* pada *OS*

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

2. Menambahkan *syntax* YAML.

```
Network:
  Renderer: NetworkManager
    Ethernets:
      Eth0:
        Dhcp4: no
        Addresses:
         - 192.168.100.31/24
        Gateway4: 192.168.100.1
        Nameservers:
          Addresses: [8.8.8.8, 1.1.1.1]
        Optional: true
    Version: 2
```

Instalasi *Secure Shell (SSH)* pada *server*

Setelah konfigurasi *OS*, kita mengimplementasikan protokol *Secure Shell (SSH)* pada *node Raspberry Pi* kami agar dapat melakukan pengaksesan *terminal* secara *remote*. *Software* yang digunakan untuk mengakses *server* menggunakan *SSH* kami gunakan adalah *openssh* yang berbasis *open source*.

Pada *OS Ubuntu,* sintaks yang digunakan adalah:

```
$ sudo apt install openssh-server
$ sudo systemctl enable ssh
$ sudo systemctl start ssh
```

Instalasi *web server* pada *slave node*

Untuk pengaplikasian *server HTTP*, kami menggunakan *software Apache* yang berbasis *open source.* Karena kami menggunakan *OS ubuntu*, maka kami melakukan instalasi dengan menggunakan aplikasi *Advanced Package Tool (APT). APT* adalah salah satu manajemen paket dan instalasi yang digunakan oleh *ubuntu*.

HTTP server ini kami implementasikan pada *slave node*, yaitu *host* rpi3 dan rpi4 sebagai *server* penyedia *website*. Adapun sintaks yang dipakai dalam instalasi adalah sebagai berikut:

```
$ sudo apt install apache2
```

Konfigurasi *Transparent Proxy* pada *Apache HTTP server*

Setelah melakukan instalasi *apache* pada *slave node*, kami mengkonfigurasi *apache* sehingga menerima *HAProxy* sebagai *Transparent Proxy*. *Transparent Proxy* bekerja sebagai intermediasi antara *client* dan *web server* dengan menangkap setiap *request* yang ada.

Pada konfigurasi *apache*, cara ini dapat dilakukan dengan menambah sintaks *"%{X-Forwarded-For}"* dalam fail *apache.conf*.

Adapun langkah - langkah yang perlu dilakukan, yaitu.

```
$ sudo vim /etc/apache2/apache2.conf
```

Mengganti teks *"LogFormat"* dengan:

```
#LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\"" combined

LogFormat "%{X-Forwarded-For}i %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\"" combined
```

Konfigurasi *HAProxy health checking* pada *Apache HTTP Server*

Kami telah mengkonfigurasi *HTTP server* sehingga menerima informasi kesehatan (*health*) dari setiap *server* yang diberikan oleh *HAProxy*. Informasi pengecekan ini berada pada fail direktori *"/var/www/check.txt"*.

Adapun langkah - langkah yang dibutuhkan:

```
$ sudo vim /etc/apache2/sites-available/000-default.conf
```

Menambahkan *syntax*:

```
SetEnvIf Request_URI "^/check\.txt$" dontlog
CustomLog /var/log/apache2/access.log combined env=!dontlog
```

Melakukan *restart service apache*:

```
$ sudo systemctl restart apache2
```

Langkah terakhir, membuat fail "*check.txt"* berdasarkan informasi pengecekan *health* "*000-default.conf*"

```
$ touch /var/www/check.txt
```

Instalasi *HAProxy* pada *master node*

Setelah menyelesaikan konfigurasi *node slave*, kami telah mengkonfigurasikan *node master* sebagai *load balancer* dengan fitur *high availability.*

Cara ini dilakukan dengan menggunakan bahasa *bash* dengan *syntax APT* pada *OS Ubuntu*:

```
$ sudo apt install haproxy
```

Konfigurasi *HAProxy* pada *master node*

Setelah melakukan instalasi, kami mengubah konfigurasi fail *HAProxy* pada direktori "*/etc/haproxy/haproxy.cfg*" sehingga dapat melakukan *load balancing* pada dua *web server*. Algoritma yang kami gunakan dalam *node balancing* ini adalah algoritma *least connection.* Algoritma ini bekerja dengan mencari kuantitas koneksi *client* dengan *web server* yang terkecil. Sebagai catatan, *parameter* konfigurasi pada *load balancer* akan berbeda pada *server* lain tergantung pada *ip server, hostname* dan algoritma *load balancing* yang dipakai.

Hal ini dapat dilakukan dengan langkah:

```
$sudo vim /etc/haproxy/haproxy.cfg
```

Lalu menambahkan *syntax*:

```
frontend http_front
  bind *:80 #
  stats uri /haproxy?stats
  default_backend http_back

backend http_back
  balance leastconn
  server rpi3 192.168.100.33:80 check
  server rpi4 192.168.100.34:80 check
```

Mengaktifkan inisialisasi *HAProxy*

Setelah konfigurasi, kita akan mengaktifkan inisialisasi *HAProxy* dengan menambahkan parameter "*ENABLED*" pada fail direktori *"/etc/default/haproxy"*.

```
$ sudo nano /etc/default/haproxy
ENABLED=1
```

Terakhir, mengaktivasi kembali *service HAProxy*. Jika tidak ada *error*, maka proses akan berjalan seperti biasa. Namun, jika terdapat *error*, maka dapat melakukan pengecekan dengan sintaks dibawah.

```
$ sudo systemctl restart haproxy #mengaktifkan kembali
$ sudo systemctl status haproxy #pengecekan error
```

Instalasi *software Keepalived* pada *master node*

Setelah pemasangan node balancer pada *HAProxy*, kami mengimplementasikan fitur *high availability* pada server dengan menggunakan aplikasi bernama *keepalived*. *Keepalived* dapat diinstal pada *OS Ubuntu* dengan menggunakan *APT*.

```
$ sudo apt install keepalived
```

Konfigurasi *software Keepalived* pada *master node*

Setelah melakukan instalasi, kami mengkonfigurasi *high availability* dan *floating ip* pada *keepalived*. Parameter konfigurasi berbeda - beda tergantung pada infrastruktur *server* yang dipakai. Adapun langkah langkah yang diperlukan:

```
$ sudo vim /etc/keepalived/keepalived.conf
```

Sintaks konfigurasi yang perlu ditambahkan:

```
vrrp_script chk_haproxy {
script “killall -0 haproxy” #mengecek modul haproxy
interval 2 #pengecekan setiap 2 detik
weight 2 #jika haproxy berjalan dengan baik, maka mendapat poin
2
}

vrrp_instance VI_1{
interface eth0 #interface koneksi yang dipakai
state MASTER #peran pada load balancer server
virtual_router_id 51
priority 101 #prioritas server yang digunakan
  virtual_ipaddress{
    192.168.100.35 #konfigurasi floating ip
  }
track_script{
  chk_haproxy
}
}
```

Pada konfigurasi diatas telah tertulis bahwa sintaks “state” menjelaskan peran pada *load balancer*. Terdapat dua jenis peran yang dijalankan, yaitu *master* sebagai *load balancer* utama, dan *backup* sebagai *load balancer* cadangan. *Load balancer* cadangan berfungsi sebagai cadangan saat *load balancer* utama *down* atau putus koneksi. Sedangkan *load balancer* utama berfungsi sebagai *load balancer* yang dipakai pertama kali saat *booting*. Ada juga konfigurasi *virtual ip* atau *floating ip* yang berfungsi sebagai *ip* yang akan mengarahkan secara dinamis kepada server. Setelah konfigurasi, kami memulai kembali *service keepalived* dengan sintaks berikut.

```
$ sudo systemctl restart keepalived
```

Jika tidak terdapat *error*, maka proses akan berjalan seperti biasa. Adapun pengecekan status berjalannya *keepalived* dan *floating ip* terdapat pada sintaks berikut.

```
$ sudo systemctl status keepalived
$ ip a
```

Terakhir, untuk pengecekan *floating ip*, jika pada *terminal* terdapat *ip* yang diinginkan, maka konfigurasi berjalan dengan sukses.

Pengaksesan *web* yang telah dibuat.

Kami membuat tampilan *website* kami dengan halaman *default* yang disediakan oleh *apache* dengan ditambahkan indikator bahwa *web server 1* dan *web server 2* diakses. Server ini dapat diakses dengan *ip* ‘192.168.100.35’. Tahap selanjutnya adalah pemasangan protokol kriptografis bernama *TLS/SSL*.

Instalasi *OpenSSL*.

Setelah instalasi dan konfigurasi *web server*, *load balancer*, dan *high availability*, kami akan melakukan implementasi *Transport Layer Security (TLS)* pada *server* kami. Implementasi ini dilakukan dengan instalasi *Secure Socket Layer (SSL)* pada *web server*. Kami memakai aplikasi *OpenSSL* yang berfungsi untuk menghasilkan enkripsi *private key* dan *certificate* yang diperlukan untuk pemasangan *TLS/SSL*. Adapun instalasi *OpenSSL* pada sistem operasi *Ubuntu* menggunakan *APT* dan berbasis *open source*.

```
$ sudo apt install openssl
```

Melakukan *generate private key* dan *certificate key* dengan *OpenSSL*.

Dengan *OpenSSL*, kita akan melakukan proses *generate private key* dan *certificate key*. Untuk proses autentikasi *certificate key* kami melakukannya dengan *self-signed*, dikarenakan proses sertifikasi yang membutuhkan biaya. Adapun sintaks yang diperlukan untuk proses ini sebagai berikut.

```
$ openssl req -newkey rsa:2048 -nodes -keyout key.pem -x509 -
days 365 -out certificate.pem
```

Setelah proses *generate* berjalan, terdapat dua file berformat "*.pem*", yaitu "*key.pem*" dan "*certificate.pem*". Kedua file akan digabungkan menjadi satu file dengan nama "*certificate.pem*" dengan sintaks sebagai berikut.

```
$ cat ./certificate.pem | tee -a ./key.pem
```

Mengubah konfigurasi pada *HAProxy*.

Setelah proses *generate* dan penggabungan fail, dilakukan pengubahan fail konfigurasi *HAProxy* dengan sintaks sebagai berikut.

```
/etc/haproxy/haproxy.cfg
#bind *:80
Bind *:443 ssl crt /home/ubuntu/openssl-key/cert.pem
```

Sintaks diatas terdiri dari *port ip address* untuk protokol *HTTPS*, *parameter ssl* dan *crt*, dan direktori tempat fail ".pem" yang telah dihasilkan *OpenSSL* tadi. Setelah pengubahan fail dilakukan, mulai kembali *service haproxy*. Jika tidak terdapat *error*, maka proses pengubahan berhasil.

```
$ sudo systemctl restart haproxy
```

Verifikasi *SSL* dan *website*.

Untuk tahap pengecekan, kami akan mengakses *web server* dengan protokol *HTTPS*. Sebagai catatan, karena kami tidak melakukan sertifikasi kepada otoritas yang *valid*, dengan kata lain *self-signed*, maka *browser* akan menampilkan peringatan dengan pesan "*self-signed certificate*" pada *floating IP website* kami.

Ini adalah tahap terakhir dari instalasi dan konfigurasi *server* kami. Tahap selanjutnya adalah dilakukannya konfigurasi *load testing*.

Metode dan konfigurasi *load testing*

Setelah melakukan pemasangan, instalasi, dan konfigurasi *server*, kami akan melakukan *load testing* pada *server website* yang kami buat. *Load testing* bertujuan untuk menganalisa performa, ketahanan, dan resistansi *server* saat diberi beban yang cukup banyak.

Kami melakukan *testing* performa *server* dengan menggunakan aplikasi *Apache JMeter*. Apache *JMeter* adalah aplikasi penyedia alat-alat yang diperlukan untuk melakukan *testing* sebuah *server*. Sementara itu, kami menggunakan *software* dari *APT* bernama *s-tui* untuk menganalisa dan *memonitoring* kondisi komponen tiap *node* pada *server*. Adapun hasil yang ingin kami dapatkan antara lain.

- Rata-rata dari *response latency server* per detiknya.
- Rata-rata penggunaan *processor* pada *server* per detiknya.

Untuk *testing load balancer*, kami telah melakukan simulasi menggunakan *apache JMeter* dengan membuat *request* sebanyak 3000 *thread* dengan *ramp-up* 30 detik. Setelah itu kami membiarkan 3000 *request* tadi untuk tetap terkoneksi dengan *server* selama 120 detik untuk menganalisa kondisi *server*.

Adapun kondisi kasus pada simulasi *testing* ini sebagai berikut.

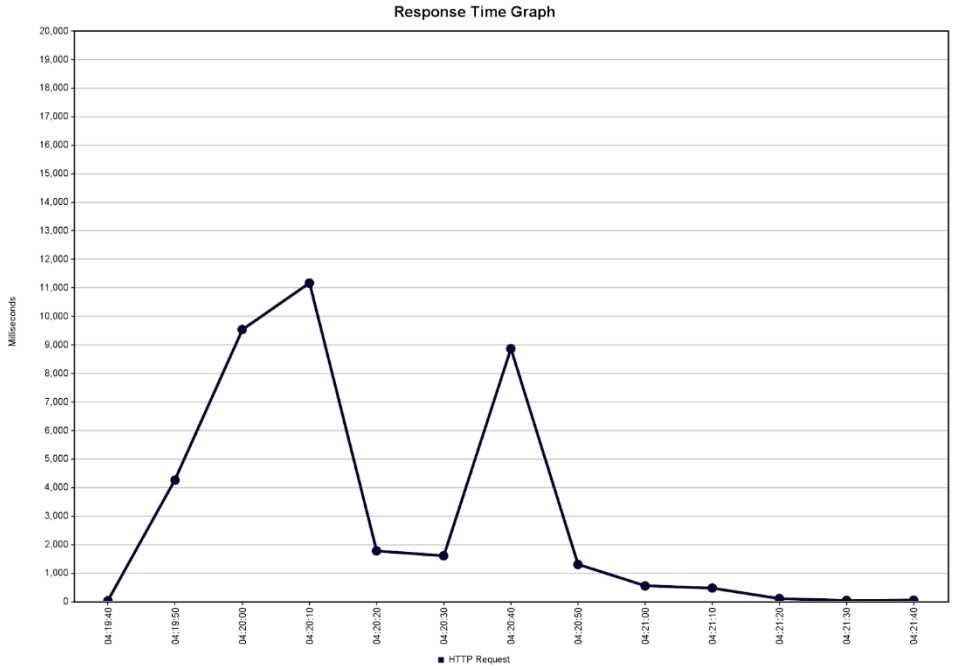
1. Semua *Load Balancer* dan *web server up.*
2. *Load Balancer 1 down, Load Balancer 2 up.*
3. *Load Balancer 1 up, Load Balancer 2 down.*
4. *Web Server 1 down, Web Server 2 up.*
5. *Web Server 1 up, Web Server 2 down.*

ANALISA

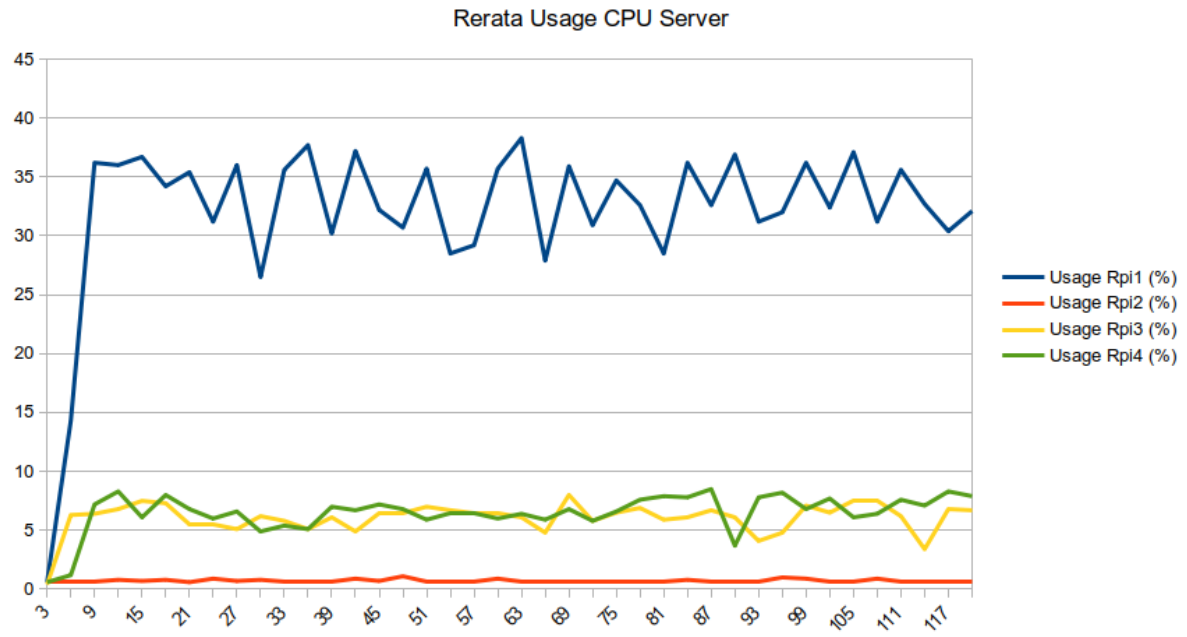
Setelah melakukan proses *load testing* pada *server* menggunakan JMeter, kami mendapatkan data *load testing* berdasarkan kondisi yang sudah dibahas sebelumnya, sebagai berikut.

Kasus pertama: semua *load balancer* dan *web server up*

Rata - rata respon *latency* pada kasus ini adalah sebagai berikut.



Sementara itu rata-rata *usage processor* pada tiap *node* di kasus ini:



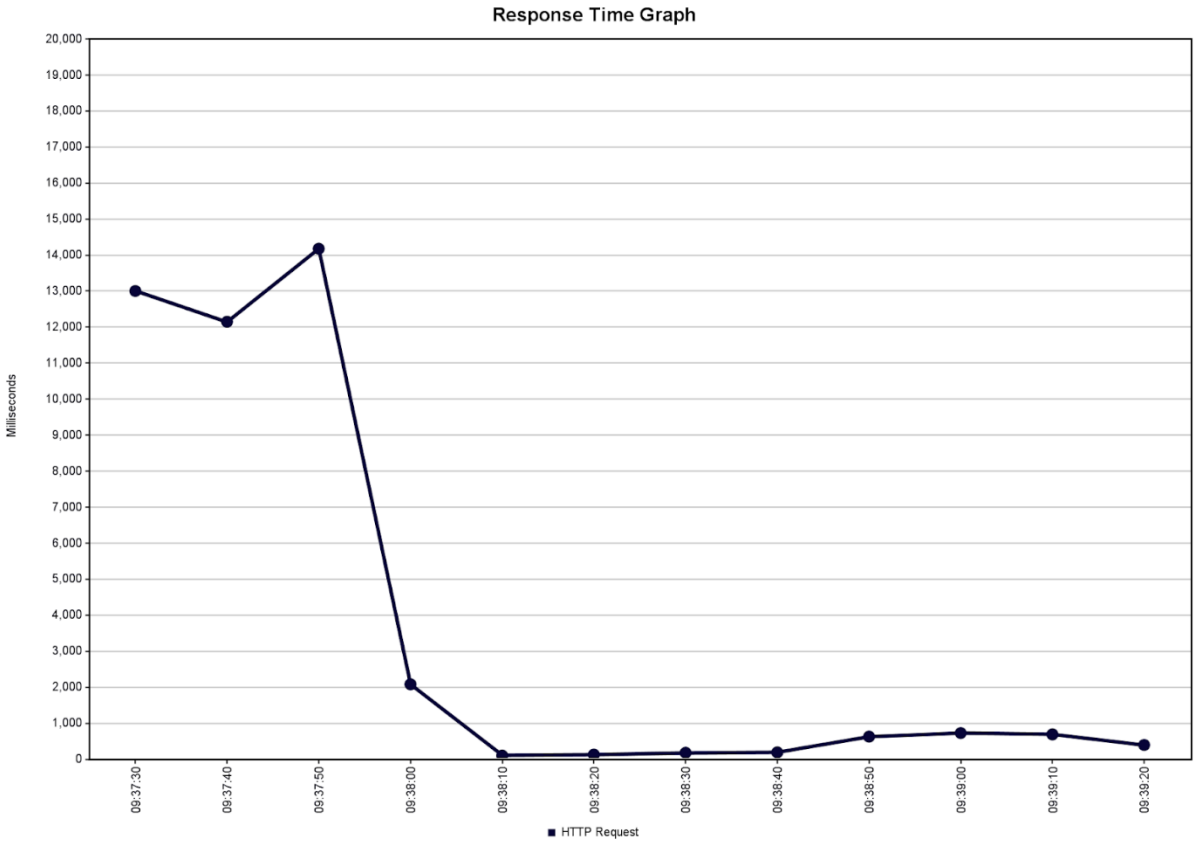
Pada kasus ini dapat dilihat bahwa fitur *load balancing* bekerja dengan baik. Rpi1 sebagai *load balancer* utama berhasil melakukan kontrol *traffic* kepada rpi3 dan rpi4. Hasilnya adalah rata - rata *usage CPU* pada *web server* hampir sama di rpi3 dan rpi4. Sedangkan pada rpi1 memiliki *usage* yang cukup besar, karena rpi1 melakukan fungsi *load balancing*, sehingga dibutuhkan kerja yang cukup besar. Meskipun memiliki kerja yang cukup besar.

Kasus kedua: *Load Balancer 1 down, Load Balancer 2 up.*

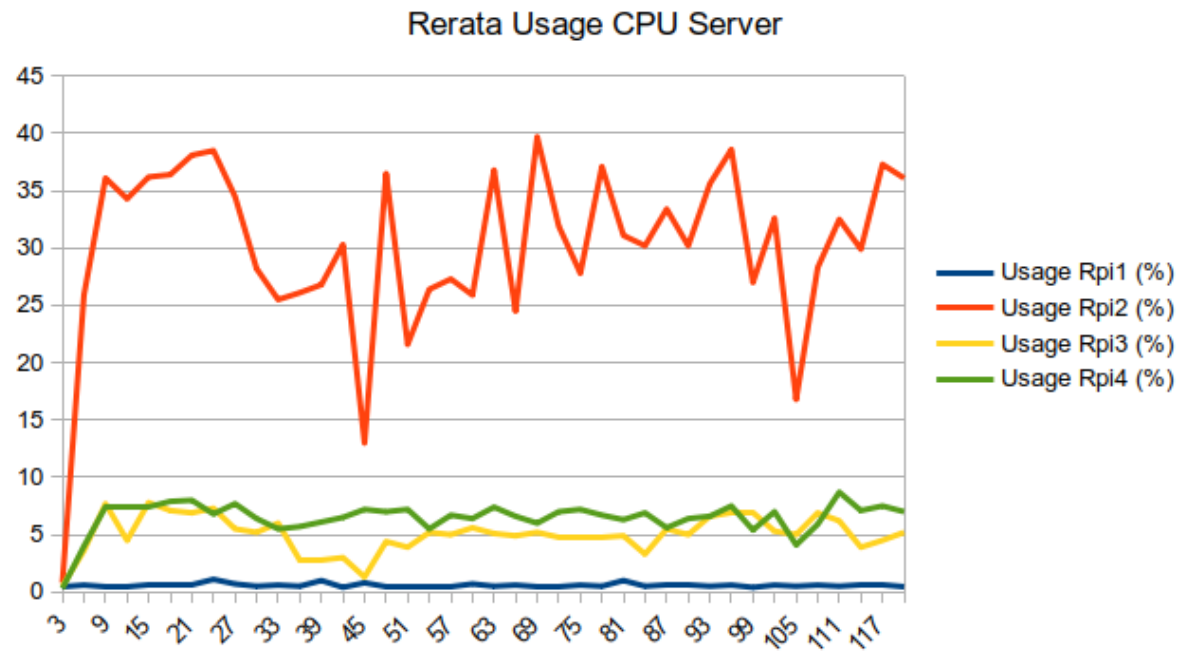
Kami melakukan pengujian pada kasus kedua dengan mematikan *service load balancing*, yaitu *HAProxy* pada *node load balancer* utama. Adapun sintaks yang diperlukan untuk melakukan testing ini yaitu:

```
$ sudo systemctl stop haproxy
```

Rata - rata respon *latency* yang didapatkan pada kasus ini adalah sebagai berikut.



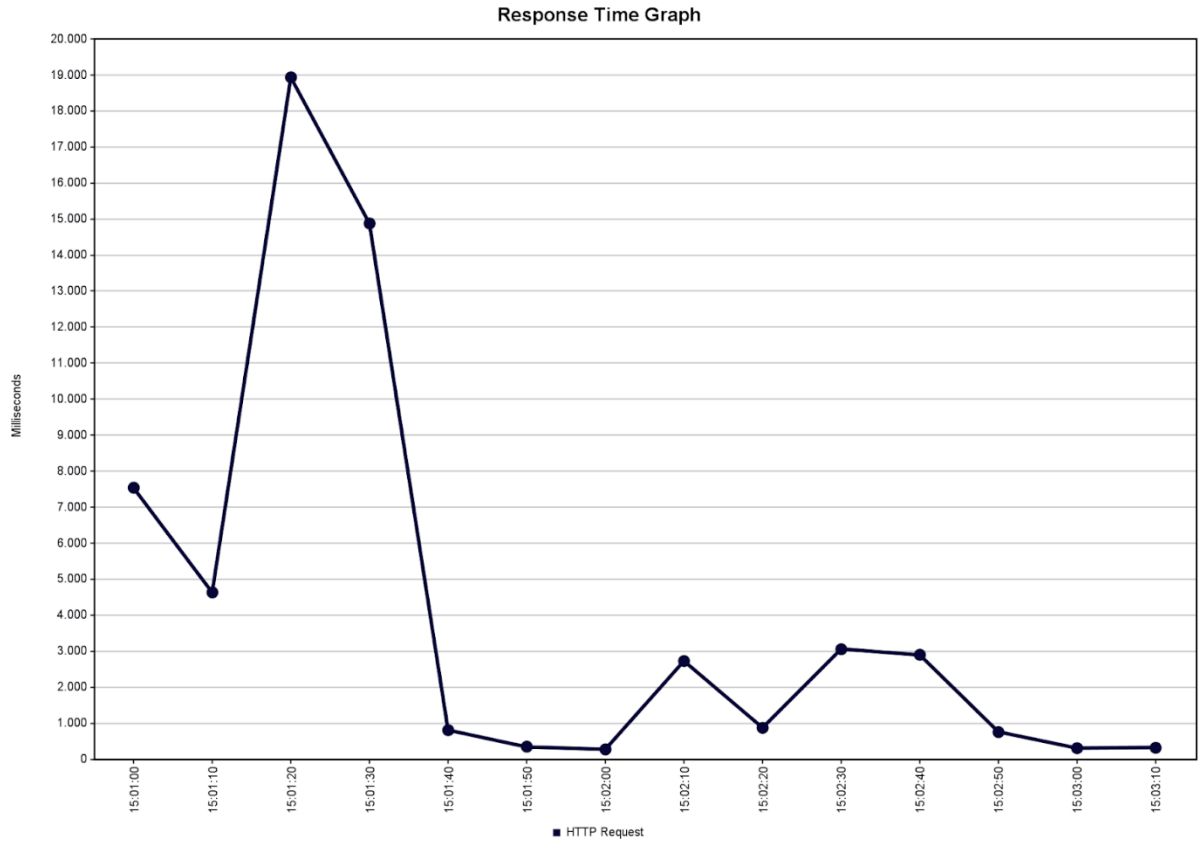
Sementara itu rata - rata *CPU Usage* pada kasus ini sebagai berikut.



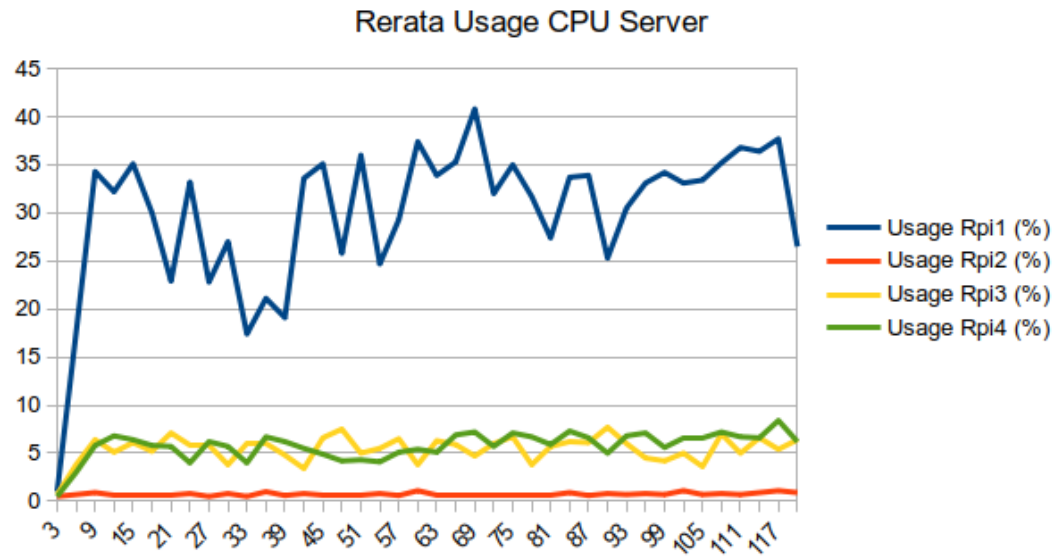
Pada grafik di atas terlihat bahwa fitur *high availability* pada *load balancer* berjalan dengan normal. Saat *load balancer* utama mati, maka *server* akan langsung memindahkan fungsi *load balancer* pada *load balancer* kedua atau rpi2 sehingga dapat dilihat bahwa *usage* pada rpi2 mengalami kenaikan yang signifikan.

Kasus ketiga: *Load Balancer 1 up, Load Balancer 2 down.*

Adapun grafik *latency* yang didapatkan pada kasus ini sebagai berikut.



Sementara itu rata - rata *usage CPU* pada kasus ini adalah sebagai berikut.

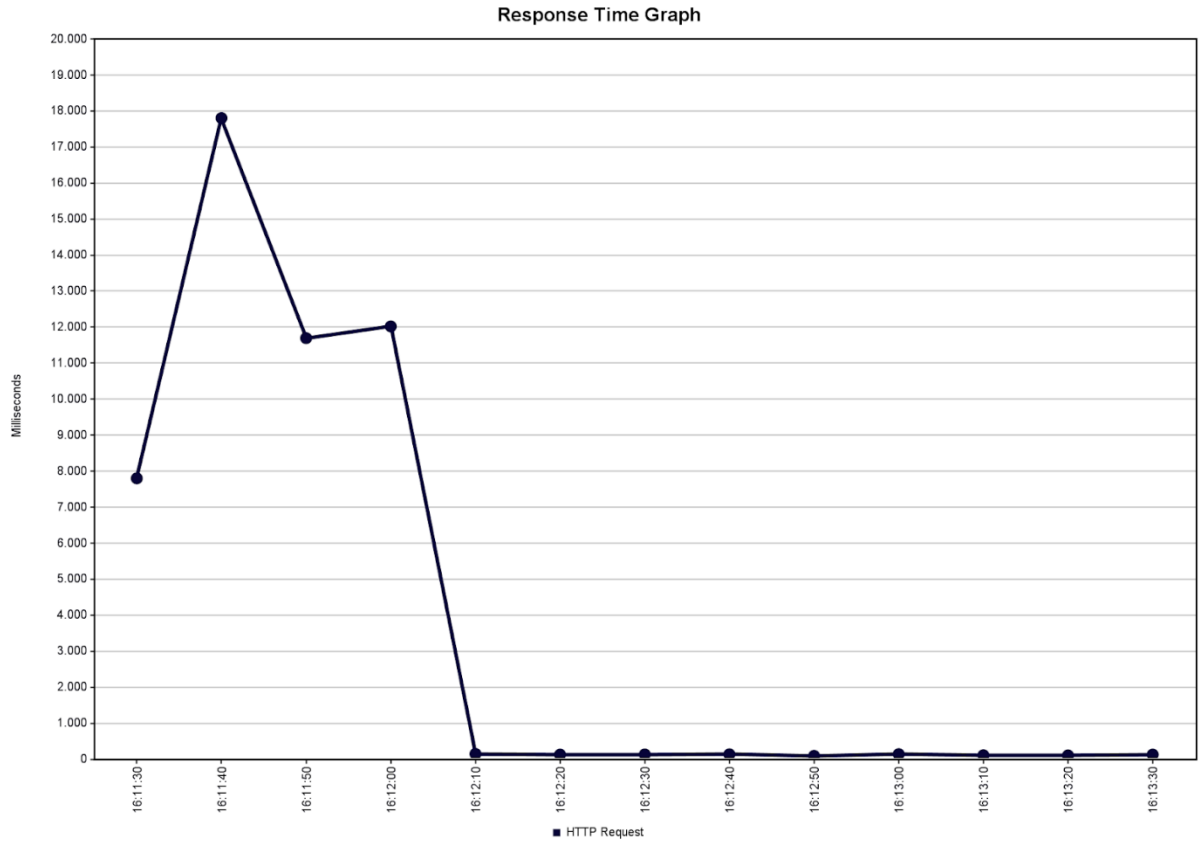


Dapat dilihat bahwa tidak ada efek pada kinerja *load balancer* 1 maupun pada *web server*, hal ini dikarenakan, *load balancer* 2 tidak melakukan kerja apa pun , dan hanya berperan sebagai *server back-up* saja, sehingga *load balancer 2* hanya akan bekerja apabila *load balancer* utama *down.*

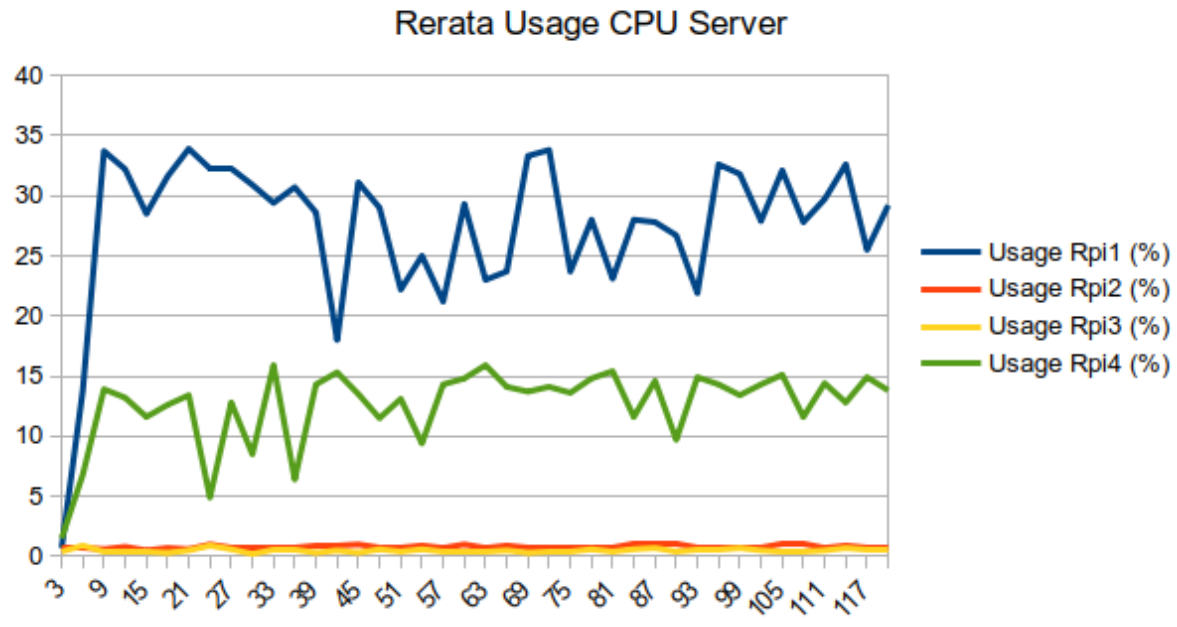
Kasus keempat: *Web Server 1 down, Web Server 2 up.*

Untuk kasus ini, kami akan mematikan *service apache web server* pada penyedia *HTTP server* pertama, yaitu rpi3. Langkah ini bertujuan untuk mengetahui dan menganalisa fitur *high availability* serta *load balancer* pada *web server*. Dengan kata lain, fitur *high availability* tidak hanya ada pada *node load balancer*, tetapi kami juga memberikan fitur tersebut pada *web server* kami.

Adapun data hasil *testing latency* pada kasus ini adalah sebagai berikut.



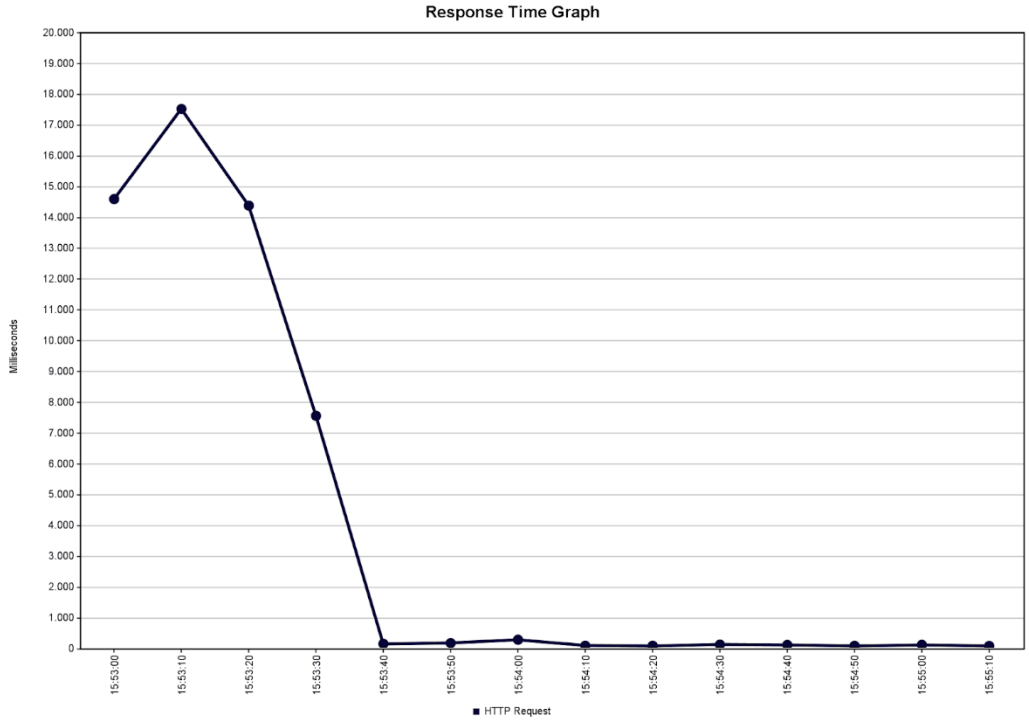
Juga, hasil *testing* pada rata-rata *usage CPU* di kasus ini adalah sebagai berikut.



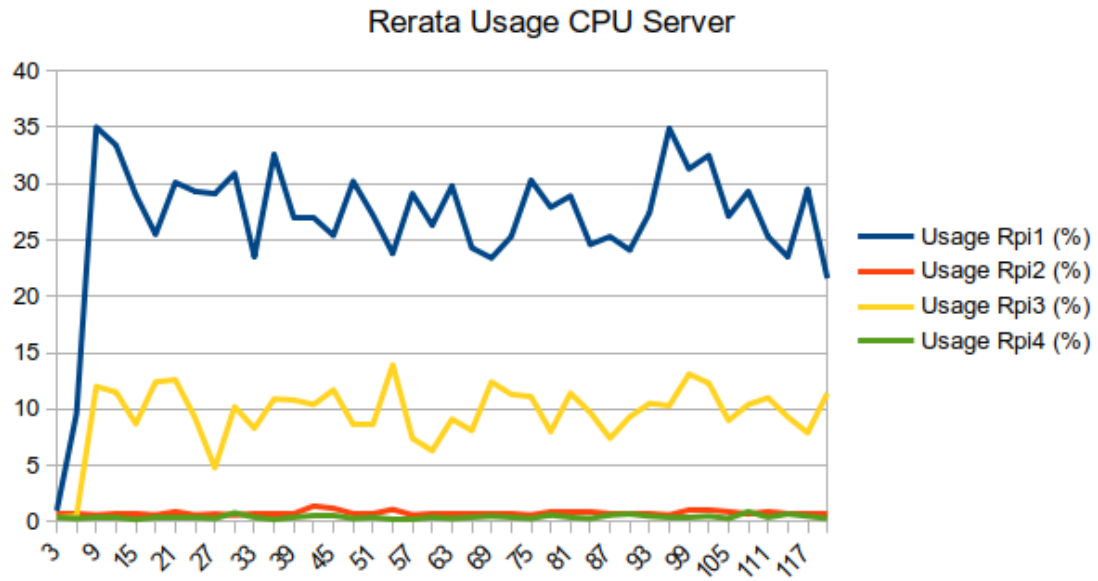
Dapat terlihat pada grafik *usage* ini, bahwa kinerja pada *web server* 2 atau rpi4 meningkat dua kali lipat dari sebelumnya. Ini menandakan bahwa fitur *load balancing* bekerja dengan optimal pada aplikasi ini, dan kinerja pada *load balancer* 1 turun dibanding data sebelumnya, karena *node web server* yang digunakan hanya satu saja.

Kasus kelima: *Web Server 1 up, Web Server 2 down.*

Adapun hasil uji *latency* yang kami peroleh sebagai berikut



Serta data rata - rata *usage* pada kasus ini adalah sebagai berikut.



Terlihat bahwa kinerja pada kasus kelima memiliki rata rata yang hampir sama dengan contoh kasus ke 4, dan hanya mengalami sedikit penurunan yang tidak signifikan. Sehingga dapat disimpulkan bahwa *load balancing* pada uji coba ini, dapat bekerja dengan baik.

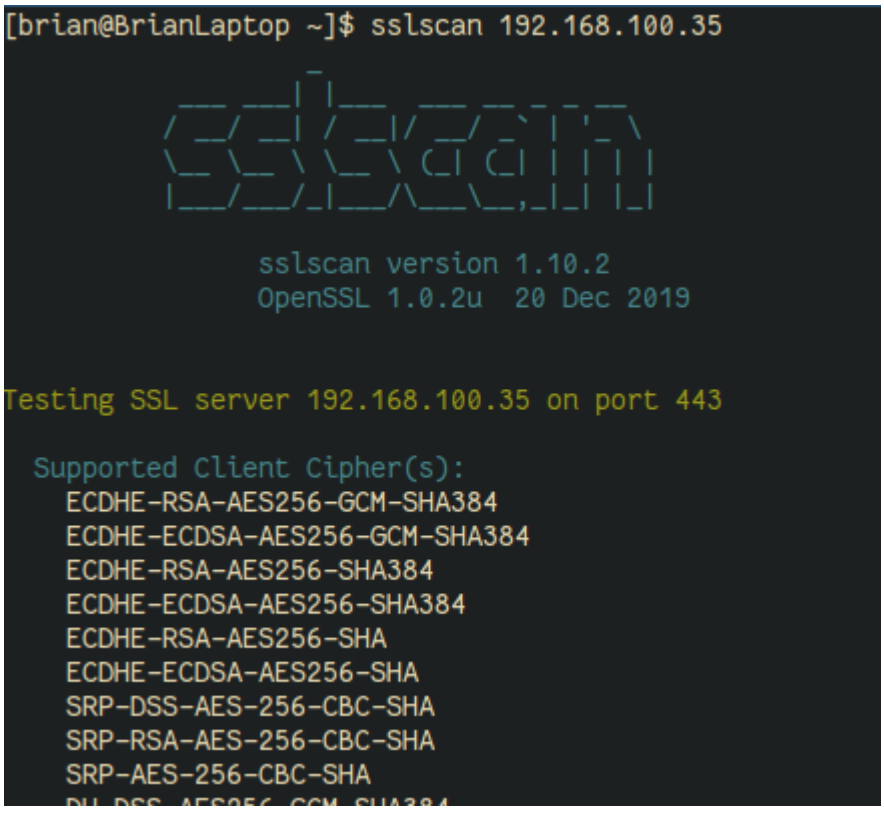
Setelah melakukan *testing*, kami juga melakukan analisa dan konfirmasi pemasangan *TLS/SSL* pada *website* dengan menggunakan aplikasi berbasis *open source* bernama *sslscan*. *sslscan* adalah aplikasi yang berfungsi untuk melakukan *scan* dan melakukan analisa protokol *TLS/SSL* yang ada pada *website*. Prosedur *scanning* kami lakukan pada *laptop* pengendali *server*. Karena *OS laptop* yang digunakan berbasis *Arch Linux*, maka instalasi dapat dilakukan dengan sintaks berikut.

```
$ sudo pacman -S sslscan
```

Berikutnya kita dapat melakukan *scan* protokol *TLS/SSL* dengan sintaks ini.

```
$ sslscan 192.168.100.35
```

Setelah melakukan scan, didapat algoritma - algoritma yang dipakai dalam penyandian hubungan *client* dengan *server*. Jika di total, terdapat 363 jumlah sandi yang dipakai pada *server* kami. Adapun dibawah ini adalah *screenshot* program *sslscan* yang dijalankan.



KESIMPULAN DAN SARAN

Kesimpulan

Setelah melakukan analisa, *testing*, dan percobaan, dapat menyimpulkan bahwa pembangunan pada *server* yang kami buat telah berhasil. Penggunaan *load balancing* pada sebuah *server* sangatlah penting karena bertujuan untuk menyeimbangkan *usage CPU* saat mendapatkan *traffic*. Selain itu konsep *high availability* juga sangat dibutuhkan untuk *server* yang bersifat jangka panjang, artinya sistem yang mengharuskan *server* berjalan terus-menerus tanpa henti baik siang maupun malam. Dengan *high availability*, sistem dapat mengganti dan mengubah peran *load balancer* dan *web server* saat terdapat *node* yang mati. Terakhir, pemasangan *TLS/SSL* pada *HTTP Server* merupakan hal yang sangat penting karena berfungsi sebagai penyedia keamanan saat komunikasi antara *server* dan *client*. Selain itu, protokol ini juga menyediakan privasi saat *client* akan berkomunikasi dengan server. Ini dapat mencegah informasi dan privasi penting yang bocor saat melakukan komunikasi baik pada sisi *client* dan sisi *server*.

Saran

1. Agar mendapatkan hasil pengujian yang lebih baik, kedepannya penulis akan menggunakan lebih banyak *node* dalam membangun *server*.

2. Untuk mendapatkan hasil yang lebih real, kedepannya penulis akan menggunakan sebuah komputer server yang lebih kuat, dan cepat dalam melakukan *clustering*.

DAFTAR RUJUKAN

[1] Irso. (2020). Dirjen PPI: Survei Penetrasi Pengguna Internet di Indonesia Bagian Penting dari Transformasi Digital. Kominfo. https://www.kominfo.go.id/content/detail/30653/dirjen-ppi-survei-penetrasi-pengguna-internet-di-indonesia-bagian-penting-dari-transformasi-digital/0/berita_satker

[2] Daon001. (2019). Jumlah Startup di Indonesia Ratusan atau Ribuan? Kominfo. https://kominfo.go.id/content/detail/17233/jumlah-startup-di-indonesia-ratusan-atau-ribuan/0/sorotan_media#:~:text=Kementerian Komunikasi dan Informatika (Kemenkominfo,itu sudah melahirkan 525 startup.

[3]. Vugt, V. S. (2014). Pro Linux High Availability Clustering (1st ed.). Apress.

[4]. Bhaskaran, S., & Matthews, A. R. (2003). Dynamic load balancer for multiple network servers. U.S. Patent No. 6,601,084. Washington, DC: U.S. Patent and Trademark Office.

[5]. Filipi, F. (2021, June 3). felixfilipi/Active-Active-High-Availability-Cluster-Server. GitHub. https://github.com/felixfilipi/Active-Active-High-Availability-Cluster-Server

[6]. Timme, F. (n.d.). Setting Up A High-Availability Load Balancer With HAProxy/Keepalived On Debian Lenny. HowtoForge. Retrieved 6 January 2021, from https://www.howtoforge.com/setting-up-a-high-availability-load-balancer-with-haproxy-keepalived-on-debian-lenny

[7]. Liu, M. N., & Zhang, J. T. (2014). The Study and Application of Based on the LVS+ KEEPALIVED High Avaliablility Load Balance [J]. Techniques of Automation and Applications, 11, 006.

[8]. Oppliger, R. (2016). SSL and TLS: Theory and Practice. Artech House.

[9]. Viega, J., Messier, M., & Chandra, P. (2002). Network security with openSSL: cryptography for secure communications. " O'Reilly Media, Inc.".

[10]. Rankin, K., & Hill, B. M. (2010). Official Ubuntu Server Book, The. Prentice Hall Press.