BAB III
METODE PENELITIAN

Metode kuantitatif objektif digunakan dalam penelitian ini dengan menggunakan tipe data numerik dan statistik. Metode didasarkan pada model analisis data deduktif. Jadi analisis data dilakukan pada langkah terakhir sesudah percobaan selesai. Penelitian ini menggunakan metode eksperimental. Tujuan dari metode eksperimental untuk menyelidiki kemungkinan hubungan sebab akibat yang diuji oleh peneliti.

3.1 PERANGKAT YANG DIGUNAKAN

3.1.1 Perangkat Keras (*Hardware*)

Perangkat keras yang diperlukan pada penelitian ini berupa laptop dan komputer lab PSD. Laptop digunakan sebagai alat perantara untuk *remote* PC Laboratorium PSD dengan spesifikasi yang terdapat pada Tabel 3.1. PC Laboratorium digunakan untuk menjalankan virtualisasi dengan rincihan yang tercantum dalam Tabel 3.2.

Tabel 3.1 Spesifikasi Laptop

Sistem Operasi	*Windows 10*
Processor	Intel (R) Core (TM) i3-6006U CPU@ 2.00GHz
RAM	4.00 GB

Tabel 3.2 Spesifikasi PC Laboratorium PSD

Sistem Operasi	*Windows 10*
Processor	Intel (R) Core (TM) i7-9700KF CPU @ 3.60GHz
RAM	64,0 GB

3.1.2 Perangkat Lunak *(Software)*

3.1.2.1 Perangkat Virtual

Beberapa mesin virtual yang dijalankan pada penelitian ini yaitu satu *client*, satu *Openstack server*, satu *server* untuk *haproxy*, dan dua *web server*. Spesifikasi perangkat virtual yang digunakan terdapat pada Tabel 3.3.

Tabel 3.3 Perangkat Virtual

CLIENT	Sistem Operasi	*Ubuntu 1*8.04
	RAM	2 GB
	Hardisk	20 GB
	Alamat IP	172.24.4.195
OPENSTACK SERVER	Sistem Operasi	*Ubuntu* 20.04
	RAM	20 GB
	Hardisk	200 GB
	Alamat IP	192.168.1.242
WEB SERVER UTAMA	Sistem Operasi	*Ubuntu 1*8.04
	RAM	2 GB
	Hardisk	20 GB
	Alamat IP	172.24.4.166
WEB SERVER CADANGAN	Sistem Operasi	*Ubuntu 18*.04
	RAM	2 GB
	Hardisk	20 GB
	Alamat IP	172.24.4.208
SERVER HAPROXY	Sistem Operasi	*Ubuntu 18*.04
	RAM	2 GB
	Hardisk	20 GB
	Alamat IP	172.24.4.176

3.1.2.2 *Software Tool* dan Aplikasi

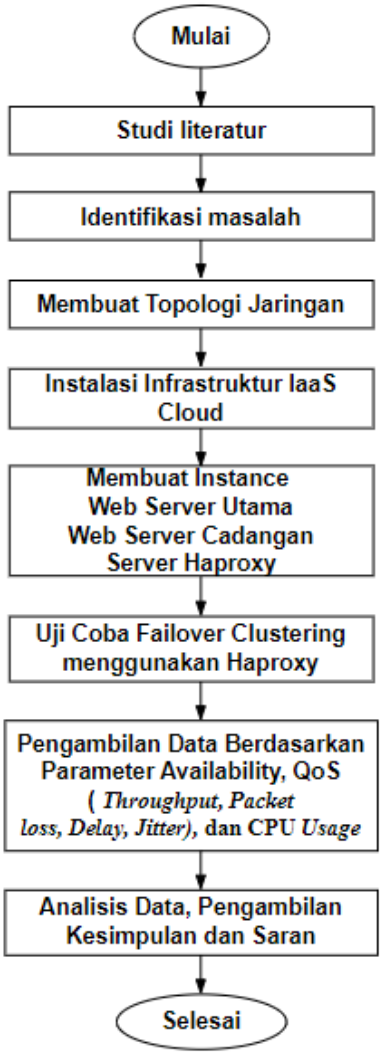
Penelitian ini menggunakan *software tool* dan aplikasi seperti pada Tabel 3.4. Tool dan aplikasi pada tabel 3.4 digunakan untuk melakukan simulasi *failover clustering* pada *web server*.

Tabel 3.4 Tool dan Aplikasi

No	*Software* yang digunakan	Kegunaan *software*
1	*VirtualBox*	Virtualisasi *server*
2	*Apache*	*Web Server*
3	*Openstack*	Sebagai *Cloud Computing*
4	*Apache benchmark*	Mengukur QoS *web server*
5	*HTTPerf*	Mengukur parameter CPU

3.2 ALUR PENELITIAN

Diagram alur penelitian pada Gambar 3.1 menunjukkan tentang bagaimana penelitian ini dilakukan dengan beberapa tahapan atau langkah-langkah secara berurutan. Penelitian dimulai dengan melakukan studi literatur, mengidentifikasi masalah yang akan diangkat untuk penelitian, membuat topologi jaringan, membangun infrastruktur Iaas *cloud*, membuat *instance web server* utama, *instance web server* cadangan, dan membuat *instance haproxy server*, serta menguji sistem *failover clustering* menggunakan *haproxy*. Selanjutnya, pengambilan data berdasarkan parameter *availability*, parameter QoS, dan CPU *usage*. Kemudian menganalisis data-data tersebut dan yang terakhir membuat kesimpulan dan saran.



Gambar 3.1 Diagram Alur Penelitian

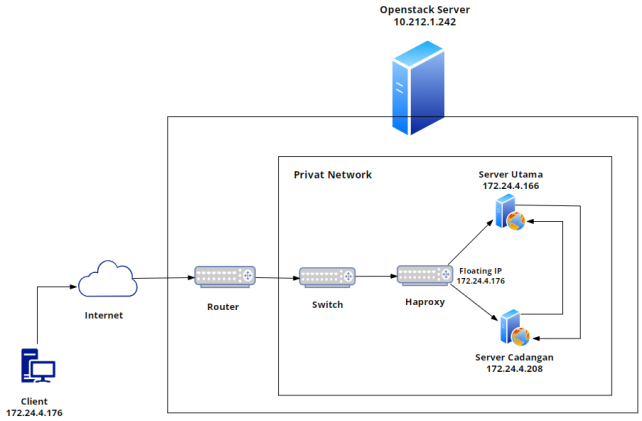
Tahapan penelitian yang diperlukan untuk mencapai hasil penelitian mengenai perancangan sistem ditunjukkan pada Gambar 3.1. Diawali dengan melakukan studi literatur sehingga dapat mengidentifikasi masalah yang terjadi, menentukan topologi jaringan sebagai dasar dari pengoperasian *failover clustering* dengan menggunakan layanan IaaS. Selanjutnya, infrastruktur layanan IaaS diinstal melalui *Openstack devstack*, yang berjalan di *node Openstack server* dengan sistem operasi *Ubuntu Server* 20.04. Setelah instalasi infrastruktur layanan IaaS, langkah selanjutnya adalah membuat *instance web server* utama, *instance web server* cadangan, dan *instance*

haproxy server. Tahap selanjutnya melakukan uji coba terhadap *failover* yang telah dikonfigurasi. Pengujian terhadap sistem dan pengambilan data dapat dilakukan berdasarkan parameter *availability*, parameter QoS *Throughput*, *Packet loss*, *Delay*, *Jitter*, dan CPU *Usage*.

Tahapan terakhir pada penelitian adalah menganalisis data-data yang diperoleh untuk mengetahui kinerja *failover clustering* pada *web server* menggunakan *haproxy* berdasarkan standar QoS dari TIPHON. Tujuan penelitian perlu diperhatikan dalam pembuatan kesimpulan untuk membantu memahami penelitian yang dilakukan dan untuk memperoleh hasil yang sesuai. Saran diberikan dengan tujuan supaya pembaca dapat menganalisis kekurangan yang ada pada penelitian.

3.3 TOPOLOGI JARINGAN

Terdapat satu *client*, satu *server*, *router*, dan *switch* seperti yang ditunjukkan pada Gambar 3.2. Gambar tersebut merupakan topologi jaringan yang dibuat sebelum penelitian dilakukan.



Gambar 3.2 Topologi Jaringan

Topologi tersebut menyambungkan *client* dengan *server openstack* yang terhubung ke internet kemudian menyambungkan ke *router*, selanjutnya *router* menghubungkan *switch* dan *switch* terhubung ke *haproxy*. Internet digunakan untuk koneksi jaringan pada *virtual machine* supaya dapat saling terhubung.

Client digunakan sebagai tempat untuk melakukan pengujian, satu *server* sebagai layanan IaaS yang dibangun dengan *server Openstack* sebagai *cloud*

computing yang digunakan untuk mengatur semua perangkat virtual. *Openstack* dibangun untuk mengelola sumber daya di pusat data, khususnya alat komputasi, penyimpanan, dan jaringan. *Openstack* terhubung ke dua jaringan yaitu jaringan privat dan jaringan publik. Di dalam *server openstack* terdapat *router, switch, haproxy* dan dua *web server*.

Router digunakan untuk menghubungkan jaringan privat supaya dapat terhubung ke internet. *Router* terdapat pada jaringan internet karena menggunakan *router* virtual yang terdapat di dalam *openstack. Router* masuk kedalam jaringan lokal sehingga hanya perangkat internal saja yang dapat mengakses. Alamat IP yang digunakan *router* untuk menghubungkan jaringan adalah 172.24.4.0/24.

Switch dalam topologi digunakan untuk menghubungkan *haproxy* yang sudah terhubung dengan *web server* utama dan *web server* cadangan di dalam satu jaringan lokal/privat. Hal ini memungkinkan kedua *web server* tersebut berada dalam sebuah jaringan yang sama (*clustering*) sehingga dapat saling berkomunikasi satu sama lain. *Haproxy* membutuhkan *switch* dalam proses peralihan trafik *web server* utama menjadi *web server* cadangan. Selain itu, *switch* digunakan untuk membuat *server* utama dan *server* cadangan berada pada satu jaringan.

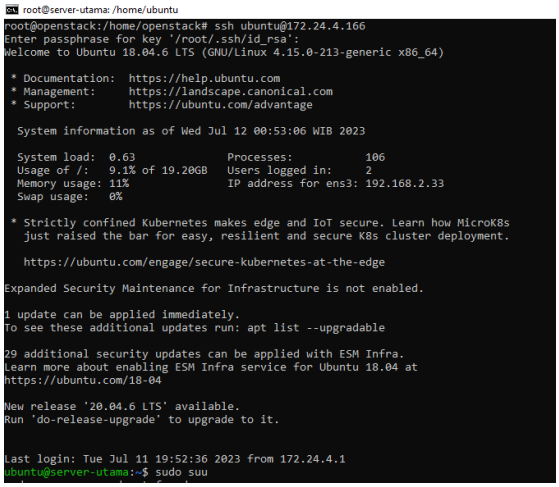
Server di dalam *openstack* merupakan jaringan lokal. Supaya *client* dapat masuk ke lingkup *openstack* maka harus melalui jaringan publik. *Floating* IP berfungsi agar *instance* yang mempunyai IP *address* internal tetap dapat mengakses perangkat lunak yang bertugas untuk mengatur *network*.

3.4 SKENARIO PENGUJIAN

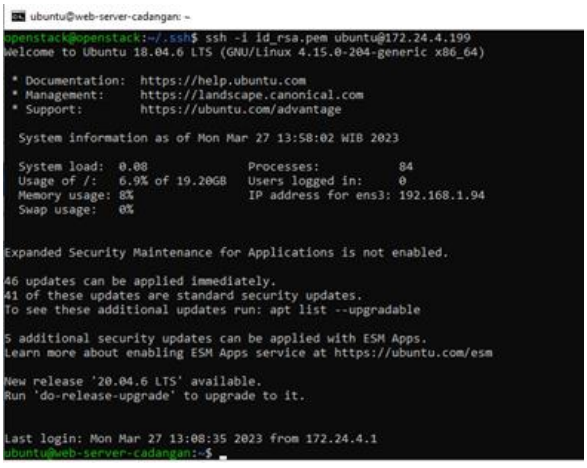
3.4.1 Membuat Skenario Jaringan

Pengujian yang dilakukan pada penelitian ini adalah *availability,* QoS (*throughput, packet loss, delay, jitter*) dan CPU *usage*. Gambar 3.3 dan Gambar 3.4 menunjukan hasil perancangan yang digunakan dalam penelitian ini. Gambar 3.3 menunjukan tampilan berbasis CLI dari *server* utama dan *server* cadangan yang di-*remote* dari terminal PC *Host*. Gambar 3.4 menunjukan topologi jaringan *logic instance* yang digunakan oleh layanan *neutron Openstack.*

Hal pertama yang dilakukan adalah melakukan konfigurasi untuk *openstack server*. Pada *Openstack server* terdapat *instance web server* utama dan *web server* cadangan sebagai tempat untuk implementasi *failover clustering* menggunaan *haproxy*. *Client* akan terus dapat mengakses *web server* meskipun *server* utama mengalami masalah. Hal ini terjadi karena *client* megakses *web server* dengan menggunakan *floating* IP. Pada saat *server* utama mati atau mengalami kegagalan, tugas *server* utama akan langsung diambil alih oleh *server* cadangan. *Client* tidak akan mengetahui jika terdapat kegagalan *server* karena terdapat *Server* cadangan yang berfungsi untuk mem-*backup*. Konfigurasi seluruh *instance* dilakukan melalui *remote* pada PC *Host.*



(a)

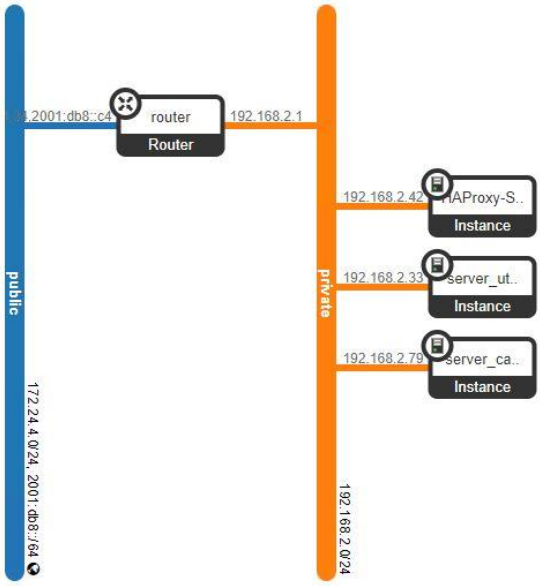


(b)

Gambar 3.3 Tampilan Berbasis CLI

(a) CLI *Server* Utama

(b) CLI *Server* Cadangan

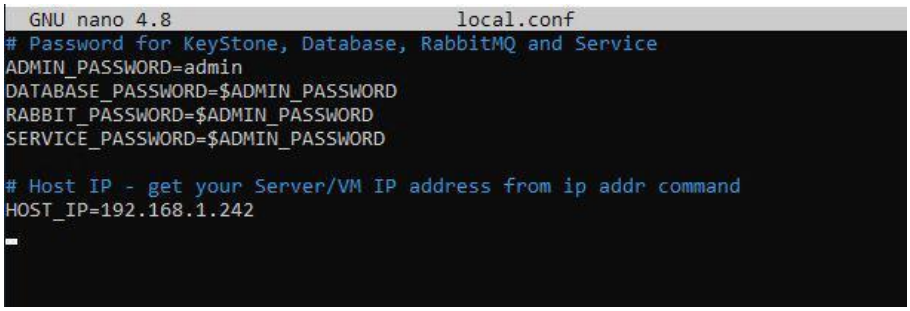


Gambar 3.4 Topologi Jaringan *Openstack*

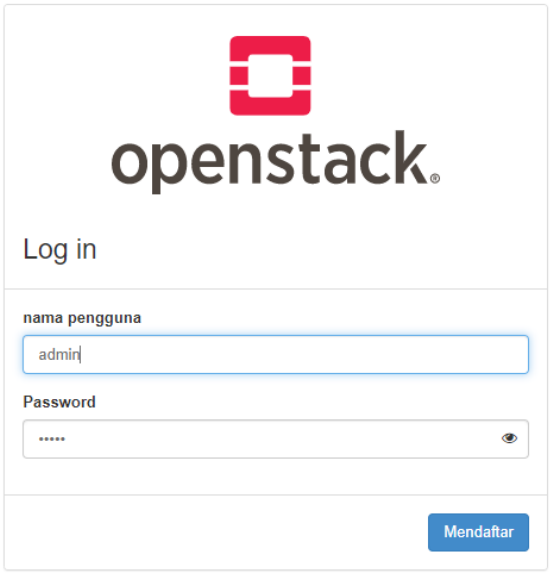
3.4.2 Konfigurasi *Openstack* Sebagai *Cloud Computing*

Jenis model layanan *cloud* pada penelitian ini menggunakan *Infrastructure as a service* (IAAS) berupa *openstack*. *Openstack* digunakan sebagai *server cloud* yang didalamnya terdapat *web server*, *server haproxy*, dan *client* sebagai perangkat virtual. Konfigurasi *Openstack* menggunakan *devstack* untuk menerapkan *failover clustering* yang dijalankan pada *instance Openstack*. Konfigurasi dilakukan pada *file local.conf* untuk mengidentifikasi pengguna, kata sandi, dan *host* IP yang digunakan untuk mendapatkan akses ke *dashboard Openstack*. *File local.conf* harus dikonfigurasi di direktori node *Openstack server /home/devstack* seperti yang terdapat pada Gambar 3.5. Setelah menyelesaikan konfigurasi *file local.conf*, selanjutnya melakukan proses instalasi *Openstack*. Proses instalasi *openStack* membutuhkan waktu yang relatif lama untuk dapat mengakses *dashboard*. *Username*, *password*, dan IP *openstack* yang muncul pada saat instalasi menunjukan bahwa proses instalasi telah selesai dilakukan. *Username* dan *password* yang digunakan untuk masuk ke *dashboard openstack* adalah

"admin". Tampilan *Openstack* setelah proses instalasi selesai ditunjukan pada Gambar 3.6.



Gambar 3.5 Konfigurasi *Openstack*



Gambar 3.6 Visualisasi *Dashboard Openstack*

3.4.3 Konfigurasi *Web Server*

Konfigurasi *web server* dilakukan pada dua *instance Openstack* yang sebelumnya telah dipersiapkan sebagai *server*. Setiap *instance* yang telah dibuat dapat dilakukan konfigurasi. Konfigurasi awal yang dilakukan yaitu melakukan instalasi *Apache* sebagai *web server*. *Apache* digunakan sebagai *web server* karena menyediakan *web server* yang aman, efisien, dan dapat dikembangkan serta dikonfigurasi dengan mudah. Konfigurasi yang dilakukan pada saat menginstal *web server* ditampilkan pada Gambar 3.7. Cara mengubah isi *file index.html* terletak pada

direktori */var/www/html/index.html* yang berguna untuk megubah tampilan *web server*. Gambar 3.8 menunjukan contoh tampilan yang terdapat pada *web server*.

```
$ sudo su
$ apt-get update
$ apt-get install apache2
```

Gambar 3.7 Konfigurasi *Web Server*

Gambar 3.8 Contoh Tampilan *Web Server*

3.4.4 Konfigurasi Dan Uji Coba *Failover Clustering* Pada *Web Server* Menggunakan *Haproxy*

Implementasi *failover clustering* pada penelitian ini menggunakan *haproxy*. Yaitu dengan menggantikan *server* utama saat mengalami kegagalan untuk meningkatkan ketersediaan layanan yang stabil di *web server*. Penelitian ini dilakukan untuk memastikan *web server* cadangan akan terus melayani perintah dari *client* ketika *server* utama mengalami kegagalan. Sebelum melakukan uji coba *failover clustering*, beberapa konfigurasi seperti konfigurasi *Openstack* dan *web server* harus dilakukan. Pada saat melakukan konfigurasi *haproxy* pada *server*, dilakukan konfigurasi IP

Address pada kedua *web server* dan kemudian melakukan tes koneksi menggunakan terminal pada PC Host. Selanjunya membuat *web server* menggunakan *Apache* dan menjalankan perintah untuk mengelola *service Apache*, terakhir melakukan akses pada kedua *web server* melalui *browser*. *Browser* yang digunakan pada penelitian ini menggunakan *Google Chrome*.

Haproxy terdiri dari beberapa konfigurasi, konfigurasi yang pertama yaitu konfigurasi untuk *backup file* yang terdapat pada Gambar 3.9. Konfigurasi *backup file* merupakan konfigurasi untuk mencadangkan *file* sebelum membuat perubahan supaya ketika terjadi kesalahan saat konfigurasi dapat dikembalikan ke *file* asli. Konfigurasi kedua yaitu konfigurasi *file haproxy* yang terdapat pada Gambar 3.10. Konfigurasi ini merupakan konfigurasi untuk mengedit isi *file* yang ada pada *haproxy* dengan tujuan untuk menentukan dan mengatur *server* utama dan *server* yang menjadi cadangan. */etc/haproxy/haproxy.cfg* merupakan lokasi untuk menyimpan konfigurasi *haproxy*. Selanjutnya yaitu konfigurasi *service haproxy* yang terdapat pada Gambar 3.11. Konfigurasi *service haproxy* merupakan konfigurasi untuk melakukan pengoperasian pada *haproxy* contohnya untuk melihat status *haproxy*, untuk memulai dan menghentikan *haproxy*.

```
cp /etc/haproxy/haproxy.cfg /etc/haproxy/haproxy.cfg.back
```

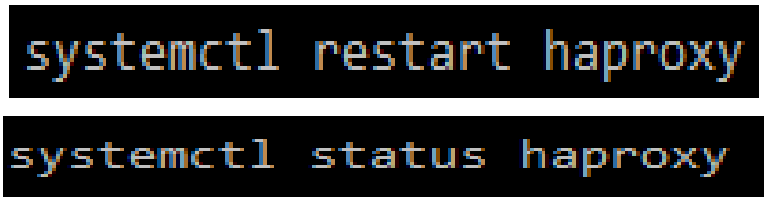
Gambar 3.9 Konfigurasi *Backup File*

```
nano /etc/haproxy/haproxy.cfg
```

```
frontend http_front
    bind *:80
    default_backend http_back

backend http_back
    mode http
    balance roundrobin
    server server-utama 172.24.4.166:80 check
    server server-cadangan 172.24.4.208:80 check backup
```

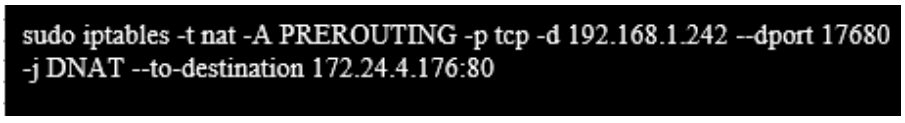
Gambar 3.10 Konfigurasi *File*



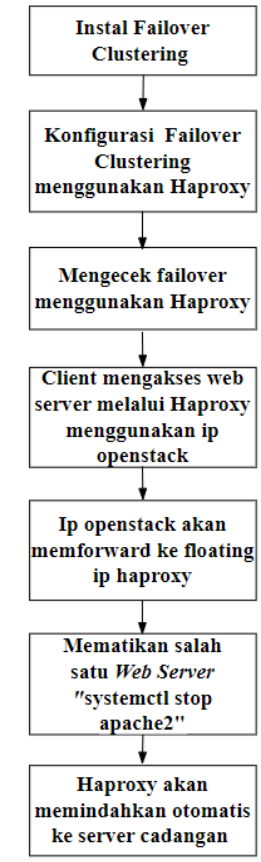
Gambar 3.11 Konfigurasi *Service Haproxy*

Setelah konfigurasi *failover* selesai, maka dapat memulai penggunakan *failover clustering* menggunakan *haproxy***.** Pengetesan *failover* pada *haproxy* telah berjalan dapat diketahui dengan melakukan tes pada *web server*. *Client* tidak langsung mengakses *web server* karena *client* mengakses *web server* melalui *haproxy*. *Client* mengakses *web server* yang sedang aktif melalui *haproxy* menggunakan *floating* IP *haproxy* yang diakses melalui IP *Openstack*. IP *Openstack* tersebut akan di *forward* ke *floating* IP *haproxy*. Konfigurasi IP *Openstack* untuk melakukan *forward* ke *floating* IP *haproxy* terdapat pada Gambar 3.12. Konfigurasi untuk *forward* IP tersebut dilakukan pada saat awal masuk ke *openstack*.

Haproxy digunakan untuk *failover* sehingga pada saat salah satu *server* mengalami kegagalan, *haproxy* secara otomatis akan memindakan *server* tersebut ke *server* cadangan. *Server* cadangan akan berpindah ke *server* utama secara otomatis pada saat *server* utama sudah dapat beroperasi kembali. Ketika *client* sudah mengakses *web server* menggunakan *floating* IP *haproxy* pada *browser*, *server* utama yang sedang sedang aktif dibuat tidak aktif dengan cara mematikan *server* utama tersebut. *Haproxy* akan membuat *server* yang mati tersebut berpindah ke *server* cadangan yang sedang *standby* untuk menggantikan tugas dari *server* yang aktif sebelumnya. Jika *server* berpindah ke *server* cadangan yang sedang *standby* maka pengujian *failover* pada *haproxy* berhasil. Gambar 3.13 menunjukan langkah-langkah untuk menguji *failover clustering* menggunakan *haproxy*.



Gambar 3.12 Konfigurasi *Forward* IP



Gambar 3.13 Langkah Uji Coba *Failover Clustering* Menggunakan *Haproxy*

3.4.5 Pengujian *Availability* Dan QoS *Web Server* Berdasarkan Jumlah Permintaan

Pada skenario pengujian konfigurasi dimulai dengan membangun *Openstack Server* sebagai tempat implementasi *failover clustering* menggunakan *haproxy*. Terdapat satu *client* sebagai simulasi untuk melakukan pengujian. *Client* akan mengakses *web server* menggunakan *floating* IP. Ketika menggunakan *floating* IP *web server* akan terus berjalan karena *server* cadangan akan menggantikan tugas *server* utama pada saat *server* utama mengalami kegagalan. Pengujian *failover clustering* dilakukan untuk memastikan sebuah *web server* akan terus menjalankan perintah dari *client* walaupun terdapat kegagalan pada *server utama*.

Pengujian *availability* merupakan ketersediaan layanan dari *web server*. Pengujian dilakukan untuk mendapatkan hasil kinerja *failover clustering* pada *web*

server untuk melihat kualitas layanan yang ditawarkan oleh *web server* meskipun terjadi kegagalan berdasarkan skenario pengujian yang telah ditentukan. Berikut merupakan skenario yang akan digunakan untuk pengujian:

1. Skenario pertama: Kedua *web server* dalam keadaan hidup, *server* utama dimatikan
2. Skenario kedua: Kedua *web server* dalam keadaan mati, *server* cadangan dihidupkan.
3. Skenario ketiga: Kedua *web server* dalam keadaan hidup, *server* cadangan dimatikan
4. Skenario keempat: Kedua *web server* dalam keadaan mati, *server* utama dihidupkan
5. Skenario kelima: *Server* utama *restart*

Skenario pengujian tersebut akan didapatkan data berupa waktu *uptime* dan waktu *downtime server* utama dan cadangan. Waktu *uptime* dan *downtime* yang dimaksud adalah waktu pada saat *web server* dalam keadaan aktif dan waktu pada saat *web server* diberikan gangguan hingga pemulihan *web server* selesai atau sampai *web server* dapat diakses. Sebagai contoh diambil dari skenario 2 yaitu kedua *web server* dalam keadaan mati, kemudian *server* cadangan dihidupkan. Artinya posisi awal kedua *web server* tersebut dalam keadaan aktif sebelum kedua *web server* dimatikan. Waktu *downtime* diambil saat kedua *web server* dimatikan sampai *server* cadangan dapat diakses karena telah diaktifkan kembali. Waktu *uptime* diambil saat *web server* dalam keadaan hidup baik itu *server* utama maupun *server* cadangan karena telah terpasang *failover* sehingga ketika *server* utama gagal maka akan berpindah ke *server* cadangan yang aktif.

Pengujian selanjutnya yaitu untuk mencari nilai QoS dan CPU *usage* berdasarkan 5 variasi pengujian dengan berbagai jumlah permintaan yang beragam yaitu 500, 1000, 1500, 2000 dan 2500 seperti yang terdapat pada Tabel 3.5. Supaya mendapatkan data nilai rata-rata QoS dan CPU *usage*, setiap variasi pengujian dilakukan sebanyak 15 kali percobaan dari setiap parameter QoS yang ditentukan seperti *Throughput*, *Packet loss*, *Delay*, *Jitter*. Pengujian QoS dan CPU *usage*

dilakukan dengan menggunakan *software apache benchmarking* dan *HTTPerf* untuk memperoleh data yang akan diproses sesuai dengan parameter *Quality Of Service*.

Tabel 3.5 Jumlah Permintaan dan Permintaan per detik

No	Jumlah permintaan	Permintaan per detik
1	500	100
2	1000	100
3	1500	100
4	2000	100
5	2500	100