BAB IV

HASIL DAN PEMBAHASAN

4.1 Persiapan

4.1.1 Informasi Kebutuhan Pengguna

Dalam merancang *load balance* s*erver*, terdapat beberapa informasi yang dibutuhkan dalam membangun sebuah *server* yang memiliki *failover* atau *load balance*. Informasi yang diperoleh merupakan hasil diskusi bersama dosen pembimbing, dan staf Biro Sistem Informasi (BSI) Universitas Muhammadiyah Yogyakarta (UMY). Berikut merupakan hasil dari analisa mengenai seberapa pentingya peran blog Universitas Muhammadiyah Yogyakarta dalam menunjang kegiatan belajar mengajar di UMY:

a. Mahasiswa membutuhkan blog sebagai sarana atau wadah dalam membuat dokumen, artikel, maupun arsip pribadi baik itu dari tulisan maupun dari video.
b. Mahasiswa membutuhkan blog sebagai sarana penunjang kebutuhan akademik seperti upload materi perkuliahan ataupun pengumpulan tugas dari dosen.
c. Pengguna blog membutuhkan akses kedalam blog yang cepat tanpa ada hambatan.

4.1.2 Spesifikasi Serta Konfigurasi Nginx *Server* Yang Dibutuhkan

Dalam membangun *server load balance*, pengguna dalam hal ini mahasiswa dan staf/dosen membutuhkan spesifikasi atau konfigurasi *server load balance* yang mumpuni untuk melayani *request user* yang berjumlah ribuan. Berikut merupakan spesifikasi atau konfigurasi *load balance* s*erver* yang dibutuhkan:

a. *server* yang memiliki *storage* yang cukup besar.
b. *server* dan *web server* Nginx dapat dengan mudah dikonfigurasi.

c. *Load balance server* beserta *backend server* mampu menangani jumlah permintaan atau *request* pengguna yang besar.
d. Konfigurasi *load balance*, baik *server load balance*, Nginx, maupun *backend server*, akan dimonitor kinerjanya secara berkala, konfigurasi harus dilakukan dengan baik sehingga performa dari *server* dan Nginx berjalan dengan optimal.

4.1.3 Evaluasi Dampak Terhadap *Server Load Balance* Yang Ada

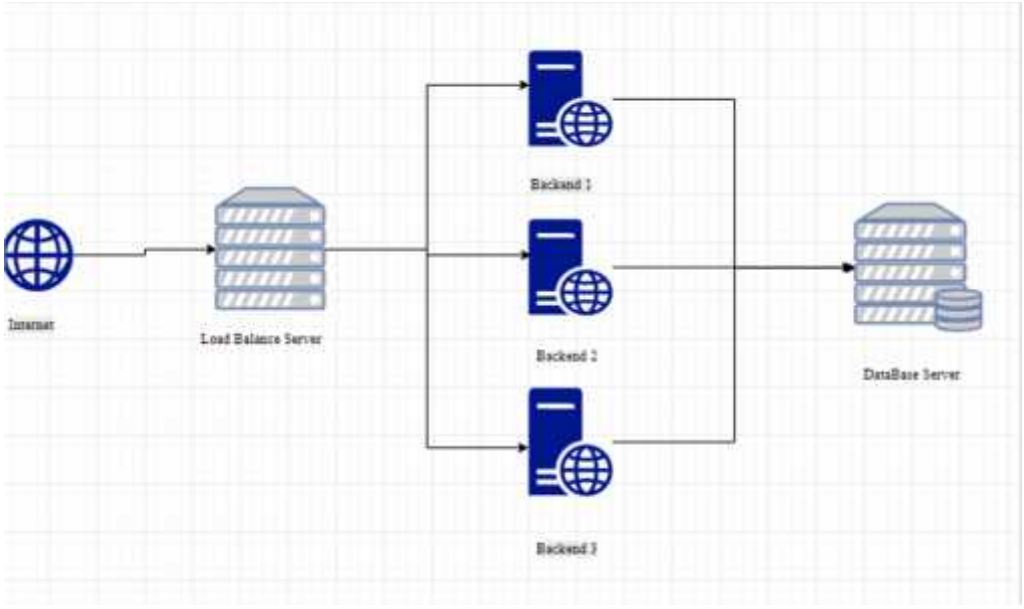
Dalam proses pengembangan *server load balance* pada blog UMY yang baru, mengamati bahwasannya penerapan mekanisme *load balance* sudah diterapkan pada pada blog UMY sekarang, namun untuk penerapan serta pengoperasiannya sendiri terbilang belum tergolong maksimal, dikarenakan tidak adanya petugas yang bertugas sebagai *sysadmin*, yang benar-benar bertugas untuk dukungan serta mengkonfigurasi web *services* Nginx. Sehingga masih sangat minim sekali dukungan dan pemelirahaan yang diberikan terhadap infrastruktur *server* khususnya di UMY.

Beberapa dampak yang terjadi terhadap kurangnya dukungan dan pemeliharaan terhadap insfrastruktur khususnya *server* di UMY:

a. Tidak *up-to-date* nya *software* atau aplikasi yang menjalankan beberapa *server* proses dan *operating system* yang digunakan.
b. Gagalnya beberapa proses layanan pada blog UMY yang berkaitan dengan *software* yang *out of date*.
c. Munculnya *security issue* yang meliputi *bug* atau *vulnerability* terhadap *system* yang terdapat pada *server* di UMY.

Dari beberapa hal permasalahan umum yang terdapat di lingkungan *server* tersebut, berdampak pada hal yang merugikan, seperti kasus penyebaran *botnet*, *malware* maupun sampai kepada kasus peretasan terhadap *website* maupun *server* UMY.

4.1.4 Merancang Desain Topologi *Server*



Gambar 4.1 Desain Topologi Fisik

4.2 Perencanaan

4.2.1 Observasi *Server*

Server yang digunakan dalam penelitian ini menggunakan dua buah *server* Lenovo type 3250 M5 dan 3650 M4. Dua buah *server* ini akan divirtualisasikan menggunakan proxmox dan vmware kedalam lima segmen *network*, yang masing-masing memiliki *ip address* 10.0.2.21 sebagai *database server*, 10.0.2.22 sebagai *load balancer server*,

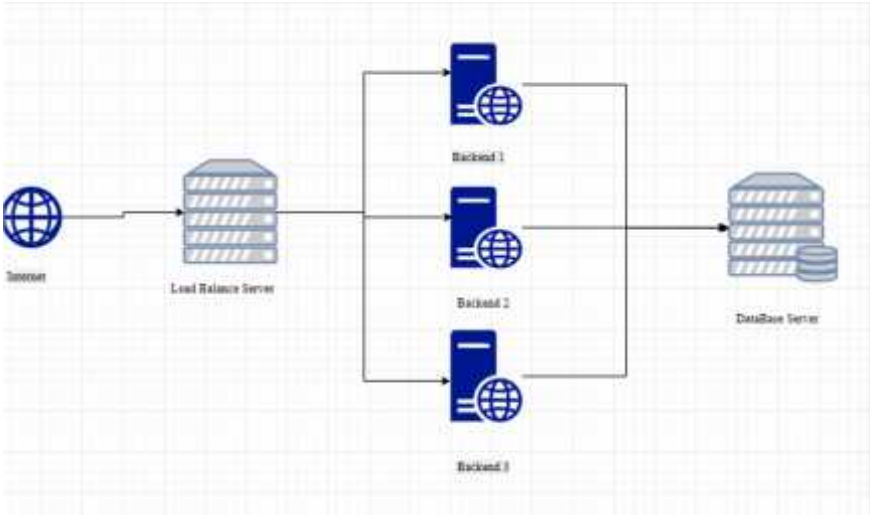
serta 10.0.2.23 sampai dengan 10.0.2.25 sebagai *backend server* dan konten desain antar muka pengguna.

Sedangkan Wordpress yang digunakan adalah Wordpress *Latest Stable Release* yang merupakan CMS *default* pada blog UMY sebagai desain antar muka *user*, sedangkan pada *database* menggunakan MariaDB.

4.3 Desain Topologi

4.3.1 Desain Topologi Fisik

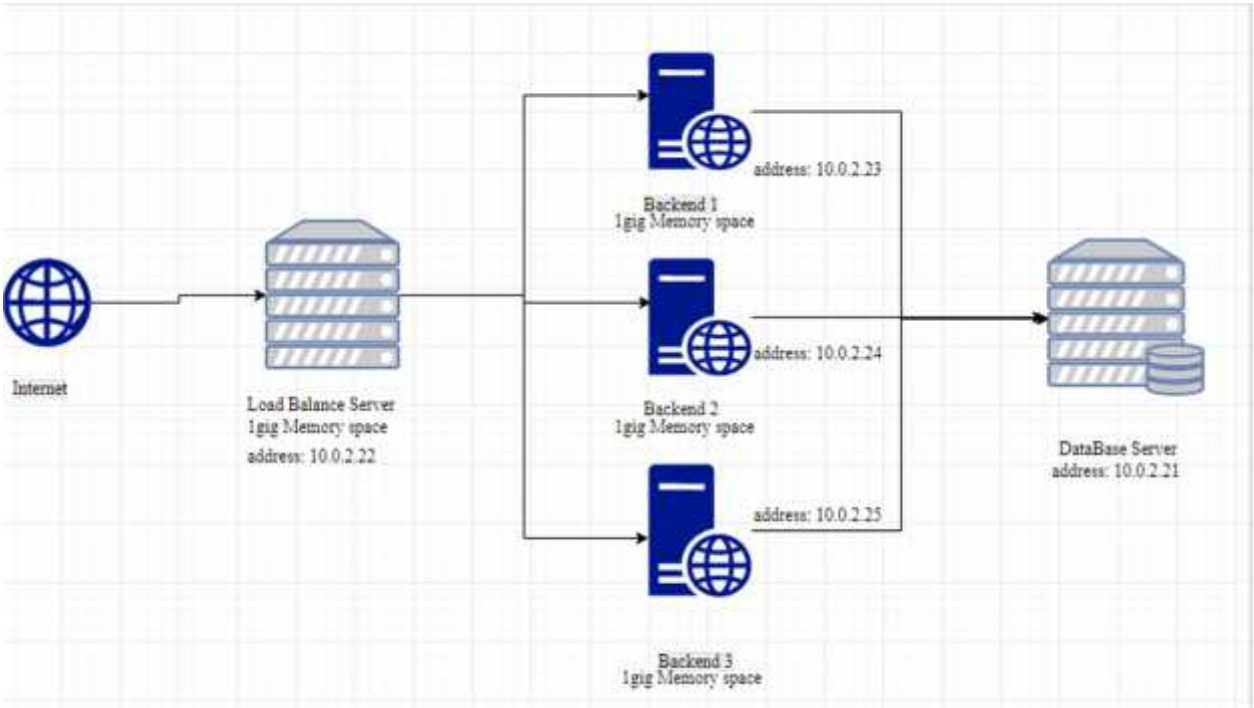
Desain topologi fisik membantu *sysadmin* dalam merancang dan menghitung kebutuhan *system* yang akan dibangun, tujuannya adalah, ketika dalam pemasangan perangkat keras maupun *software* nantinya, *sysadmin* dapat memperkirakan *system* yang akan dibangun. Desain topologi fisik ini membahas tentang bagaimana seorang *sysadmin* dalam melakukan pemasangan perangkat-perangkat keras yang menunjang jaringan atau *system* yang akan dibangun. Topologi fisik untuk *load balance* blog UMY dapat dilihat pada gambar 4.2 berikut.



Gambar 4.2 Desain Topologi Fisik

4.3.2 Desain Topologi Logikal

Desain topologi logikal memiliki fungsi sebagai panduan seorang *administrator* dalam mengimplementasikan *system* dan jaringan secara keseluruhan, dimulai dari perencanaan *ip address*, sampai dengan perencanaan alokasi *space server* dan *system* yang ada pada masing-masing *server*. Pada penelitian ini aturan terkait masalah teknis alokasi *memory* per *server* beserta *ip address* dari masing-masing *server* sudah diatur dan ditentukan oleh *sysadmin* dari UMY. Gambar 4.3 berikut merupakan topologi logikal dari *server load balance*.



Gambar 4.3 Desain Topologi Logikal

Berdasarkan gambar 4.3, Terlihat bahwa terdapat 5 *server* yang mendukung dalam pembangunan *load balance* blog UMY, *server* pertama yang beralamat pada *ip address* 10.0.2.21 merupakan *server* yang khusus dipersiapkan untuk *database server* yang terpisah dari *server* lain baik secara virtual maupun secara fisik. Pemisahan ini

bermaksud untuk apabila terjadi hal yang tidak diinginkan misalnya, peretasan yang dilakukan pada *website,* dapat segera mengisolir *web server* dan *server* yang terkena serangan, sehingga bisa meminimalisir kemungkinan terjadinya peretasan terhadap *server-server* lainnya seperti *database server*. *server* yang digunakan sebagai *load balance* memiliki alamat pada *ip address* 10.0.2.22, menggunakan Nginx sebagai *service* yang menjalankan *load balance.* Untuk metode yang digunakan dalam *load balance* yaitu menggunakan metode *Least Connected load balance,* yang akan membagi beban s*erver* secara merata ke *backend server*. Konfigurasi *Least Connected load balance* pada *server* dapat terlihat seperti pada gambar 4.4.

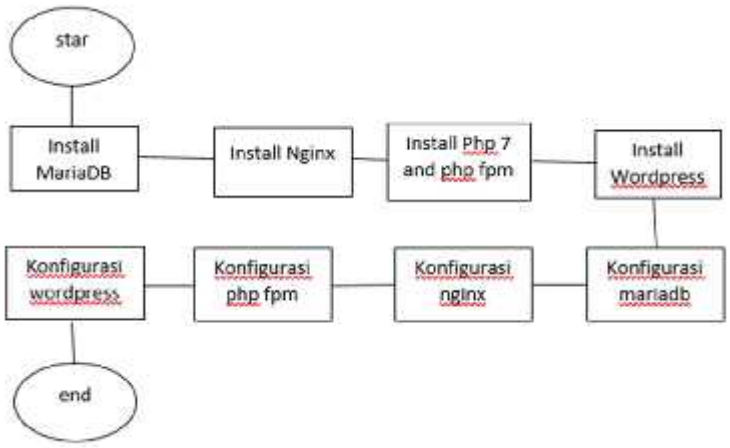
Gambar 4.4 Pengaturan *Load Balance Least Connected*

Berdasarkan gambar 4.4 terlihat bahwa dalam penelitian ini menggunakan 3 *backend server* yang ditandai dengan *ip* 10.0.2.23, 10.0.2.24, dan 10.0.2.25, metode yang digunakan dapat terlihat dari adanya keterangan *least_conn* yang menandakan bahwa *load balance* ini menggunakan metode *least_conn* yang akan membagi *request* dari *client* ke *backend server* yang dianggap memiliki kerja yang tidak terlampau sibuk

berdasarkan *session per ip connection.* Metode umum yang paling banyak digunakan pada saat mengimplementasikan *load balance* pada Nginx terbagi atas tiga metode. Metode yang pertama adalah *round-robin* yang merupakan *default* dari *load balance* yang terdapat pada Nginx, dimana dalam metode ini *request* dari *client* akan dibagi sama rata kepada *backend server*. Metode kedua yaitu *Least Connected*, dimana dalam membagi permintaan dari *client*, Nginx yang difungsikan sebagai *load balance proxy* akan membagi beban ke *backend server* yang paling sedikit bebannya dan akan langsung mengalihkan permintaan yang dianggap memakan waktu lama berdasarkan parameter yang telah ditentukan yaitu *session* pada *client ip address*. Sehingga permintaan *client* pada waktu selanjutnya akan diproses pada *backend server* yang berbeda. Metode terakhir adalah metode *ip_hash,* dimana dalam metode ini *request* yang berasal dari *client* akan disimpan berdasarkan *cache* dari *client session,* dimana *cache client session* inilah nantinya yang akan menjadi penentu akan diarahkan kepada *backend server* mana nantinya.

4.3.3 Implementasi Dan Konfigurasi Nginx *Load Balance Server*

Terdapat beberapa tahapan dalam mengimplementasikan *load balance* pada penelitian ini yang ditunjukan pada gambar 4.5 berikut, dan akan dijelaskan pada subbab-subbab berikutnya.



Gambar 4.5 Langkah Kerja Implementasi *Load Balance*

4.3.4 Instalasi MariaDB

Dalam mengkonfigurasi atau membuat *server load balance* pertama kali melakukan instalasi dan konfigurasi pada *database* yang yang akan digunakan. *Database* menggunakan MariaDB yang terinstal pada fisik *server* yang berbeda dengan aplikasi atau *web server* dengan alasan keamanan. MariaDB di *maintenance* oleh organisasi yang bernama MariaDB *foundation* yang mana merupakan *forked* atau diambil dari MySQL *database* program. MariaDB memiliki pengaturan konfigurasi yang terletak pada */etc/mysql/my.cnf*. Dalam menginstal MariaDB sebagai *database*, mengambil dari *repository* MariaDB dan langsung ter*update* dari *trusted vendor*. Prosedur instalasinya sangat mudah dan semua petunjuknya sudah tersedia melalui *link* berikut https://downloads.MariaDB.org/MariaDB/repositories/.

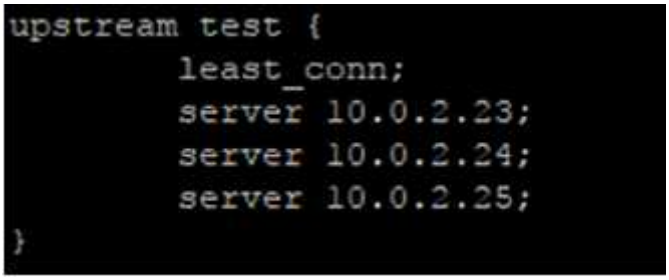
MariaDB memiliki beberapa *mirror* atau *repository* yang terletak di Indonesia dan khusus GNU/Linux yaitu Ubuntu distro yang digunakan pada penelitian ini, dapat di *download* dan di instal melalui *website* berikut. https://www.downloads.MariaDB.org.

Berikut merupakan contoh dari instalasi MariaDB sebagai *database* pada Ubuntu *server* 16.04 LTS (*Long Term Support/Service*) sesuai dengan *operating system* yang digunakan pada penelitian kali ini.

Pertama kali instal kebutuhan MariaDB yaitu *software-properties-common* dengan cara, masuk ke terminal *server* lalu ketikan "*sudo apt* instal *software-properties-common*", lalu tambahkan *keyServer* yang berguna sebagai otentikasi pada saat *update* dan instal melalui *repository*, masukan juga pada *terminal server* "*sudo apt-key adv --recv-keys --keyserver hkp://keyserver.ubuntu.com:80 0xF1656F24C74CD1D8*". Selanjutnya langkah terakhir yaitu dengan cara meng*update* dan menginstal MariaDB *database* dengan cara ketikan perintah berikut pada terminal atau *cli server sudo apt update; sudo apt install mariadb-server*.

4.3.5 Implementasi Dan Konfigurasi Nginx *Load Balance* Sebagai *Backend Server*

Server backend memiliki peranan penting sebagai *server* yang akan melayani *client*. Pada penelitian ini menggunakan *backend server* sebanyak tiga virtual *server* yang berguna sebagai *fail-over*, dan masing-masing *server backend* beralamatkan pada *ip* 10.0.2.23, 10.0.2.24, dan 10.0.2.25. Masing-masing *server backend* terinstal dan terkonfigurasi Nginx versi 1.12.2, serta Wordpress sebagai blog dan desain antar muka pengguna, yang terhubung kedalam *database MariaDB*. Konfigurasi pada *server backend* dapat dilihat pada gambar 4.6.



Gambar 4.6 Pengaturan *Load Balance*

Dari gambar 4.6, terlihat dalam implementasi *load balance* menggunakan metode *least connection load balance*, yang akan memilih berdasarkan prioritas *backend server* secara acak, dan meminimalisir terjadinya *request overload*, dikarenakan *request* yang terlalu lama akan langsung dialihkan kepada *backend server* yang lain, sesuai dengan parameter *http limit request module*. Konfigurasi pada *server load balance* terletak pada direktori dan *file /etc/nginx/conf.d/lb.conf*.

Proses instalasi Nginx sebagai *backend* maupun *load balance server* memiliki dua cara. Cara yang pertama adalah melalui *repository default Operating System* atau *repository local Ubuntu*, dengan cara menjalankan perintah *sudo apt update; sudo apt install nginx* pada terminal *server*. Akan tetapi cara ini dinilai kurang efektif jika terdapat *update* dari Nginx *developer* terkait dengan paket serta dependensi dari Nginx, oleh karena itu permasalahan ini dapat diatasi dengan cara menambahkan *Nginx repository*. Masuk terlebih dahulu melalui terminal editor baik *vim,vi,* ataupun *nano* pada file */etc/apt/sources.list*, lalu tambahkan **deb http://nginx.org/packages/ubuntu/ $release nginx** dan **deb-src http://nginx.org/packages/ubuntu/ $release nginx** pada akhir baris atau letakan pada *line* terakhir paling bawah. Patameter **$release** diganti sesuai dengan *version name* dari *operating system server* yang digunakan**,** contoh pada penelitian menggunakan Ubuntu 14.04.5 LTS (Trusty Tahr)**,** sehingga **$release** ganti dengan

trusty, terakhir masukan perintah *sudo apt update && sudo apt* instal *nginx* pada terminal atau *command prompt server*.

4.3.6 Implementasi Dan Konfigurasi PHP 7.0 dan PHP 7.0 Fpm

PHP merupakan bahasa pemrograman yang digunakan pada CMS Wordpress, PHP didalam penelitian kali ini berperan sebagai bahasa pemrograman utama yang digunakan, sedangkan php-fpm berfungsi sebagai *server side scripting* dan merupakan satu kesatuan dari PHP yang digunakan oleh Nginx sebagai perantara penterjemah bahasa PHP kepada *client* atau *end user*.

PHP yang digunakan pada penelitian adalah PHP versi 7.0, proses instalasi dilakukan dengan cara sebagai berikut:

a. *sudo apt-get install python-software-properties software-properties-common*
b. *sudo LC_ALL=C.UTF-8 add-apt-repository ppa:ondrej/php*
c. *sudo apt-get update; sudo apt-get install php7.0 php7.0-fpm php7.0-mysql*

4.3.7 Konfigurasi MySQL *Remote Database*

Setelah selesai melakukan instalasi *database server* dan PHP, langkah selanjutnya adalah membuat *database* untuk Wordpress pada *backend server* dan memastikan *database* tersebut dapat terkoneksi dengan *backend server*. Langkah pertama yang akan di lakukan yaitu membuat *database* untuk Wordpress, tahap pertama yaitu, masuk kedalam *database* yang akan dibuat dengan cara *mysql -u <username> -p* jika *service*

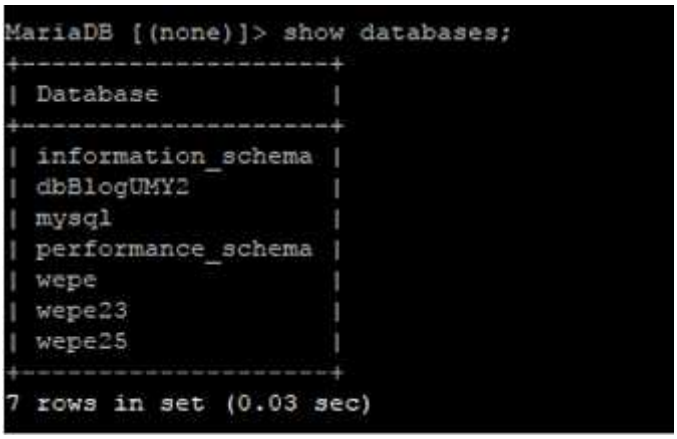
mysql belum aktif, maka aktifkan terlebih dahulu dengan menjalankan perintah ini pada terminal *server* "*sudo /etc/init.d/mysql start"*.

Tahap selanjutnya yaitu, membuat *database* untuk masing-masing *backend web server* yang menggunakan Wordpress, *login* terlebih dahulu pada *database* lalu masukan perintah seperti pada gambar 4.7 berikut.

```
CREATE DATABASE wordpress;
```

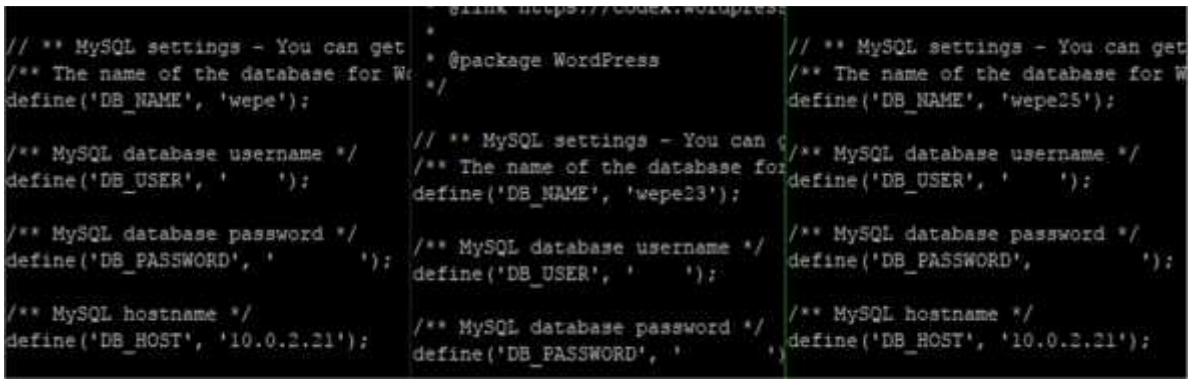
Gambar 4.7 Membuat Database pada MySQL

Sehingga jika ditampilkan keseluruhan database yang telah dibuat akan menampilkan seperti gambar 4.8 berikut.



Gambar 4.8 Menampilkan Keseluruhan Database

Dapat terlihat bahwa *database* yang digunakan yaitu database *wepe, wepe23, wepe25*, yang melakukan *remote* koneksi terhadap s*erver database* dari ketiga *backend web server* tersebut, serta didefinisikan pada DB_HOST yang terdapat pada Wordpress. Remot *database* konfigurasi terlihat pada gambar 4.9.



Gambar 4.9 Wordpress Remot Database Konfigurasi

Setelah melakukan beberapa konfigurasi remot *database* untuk Wordpress langkah selanjutnya yaitu, membuat *database* tersebut dapat diakses secara remot dari *backend web server*. Pertama-tama aktifkan fitur *bind-address*, fitur yang terdapat pada MySQL *database* ini yang nantinya akan berfungsi dalam melakukan remot koneksi dari *backend* terhadap *database server*. Konfigurasi lengkap dapat terlihat pada gambar 4.10, sedangkan parameter *bind-address* yang memiliki *network range* 0.0.0.0 merupakan konfigurasi agar dapat menerima request dari *ip network range* berapapun guna memudahkan konfigurasi pada penelitian ini.

```
#skip-external-locking
#
# Instead of skip-networking the default is now to listen only on
# localhost which is more compatible and is not less secure.
bind-address            = 0.0.0.0
```

Gambar 4.10 Remote Database Konfigurasi

4.3.8 Implementasi Dan Konfigurasi Wordpress

Langkah terakhir dalam implementasi *load balance* adalah menginstal serta mengkonfigurasi Wordpress. Wordpress merupakan CMS (*Content Management*

System) yang digunakan sebagai blog UMY, Wordpress dapat di *download* pada *link* berikut *https://wordpress.org/download/*.

Proses selanjutnya yaitu, masuk pada tahap instalasi dan konfigurasi Wordpress pada Nginx *web server*, langkah pertama yaitu dengan cara membuat direktori baru yang berguna untuk meletakan *file-file* pendukung serta konfigurasi Wordpress, dalam penelitian kali menggunakan direktori yang diberi nama *wp* pada path direktori */usr/share/nginx/*, dengan cara masuk terlebih dahulu ke directory */usr/share/nginx/*, kemudian ketikan pada *console* atau *terminal server* "*cd /usr/share/nginx/*", *cd* merupaka singkatan dari *change directory* yang artinya pindah ke lain *directory*. Kemudian buatlah sebuah direktori baru bernama *wp* dengan cara *mkdir wp*. Lalu *download* Wordpress dan ekstrak *file compress* hasil *download* Wordpress lalu pindahkan kedalam direktori yang telah dibuat sebelumnya, dengan cara *mv /usr/share/nginx/wp* atau *cp /usr/share/nginx/wp*, *mv* merupakan singkatan dari *move* dan *cp* merupakan singkatan dari *copy*. Tahapan selanjutnya yaitu konfigurasi Wordpress *config*, dengan cara masuk ke dalam direktori tempat dimana letak file dan direktori Wordpress tadi berada. Pada penelitian ini diambil salah satu contoh, meletakan pada direktori */usr/share/nginx/wp/wordpress*, kemudian *copy wp-config-sample.php* kedalam file baru dengan cara *cp wp-config-sample.php wp-config.php*.

Selanjutnya yaitu melakukan konfigurasi terhadap *file wp-config.php* sesuai dengan nama dari *database* yang telah dibuat. *Username* dan *password database*, serta *ip* dari *server database*.

Langkah selanjutnya yaitu, mengkonfigurasi Nginx dan mengarahkan *file* dan direktori Wordpress yang telah dibuat sebelumnya. Cara melakukan konfigurasi Wordpress agar berjalan pada Nginx *web server* adalah sebagai berikut:

1. *cd /etc/nginx/conf.d* (masuk kedalam Nginx direktori), direktori inilah yang menjadi pusat pengaturan pada Nginx *web server*.
2. *mv default.conf default.conf.backup*, berfungsi untuk membuat *default* konfigurasi Nginx menjadi *backup* Nginx konfigurasi.
3. *cp default.conf.backup wp.conf*, berfungsi menjadikan *wp.conf* sebagai *default* konfigurasi pada Nginx.
4. *nano wp.conf*, *nano* merupakan salah satu *terminal text editor* yang terdapat pada Ubuntu baik *server* maupun *desktop*, jika tidak terlalu familiar menggunakan *nano* alternatif lain dapat mencoba *terminal text editor* seperti *pico,vim*, atau *vi*.
5. Edit pada bagian *server_name*, ganti yang semula *localhost* menjadi *domain name* ataupun *ip address* dari Ubuntu *server*.
6. Selanjutnya pindahkan *root directory default* menjadi *path directory* dimana terdapat *file* dan *directory* dari Wordpress. Contoh dalam penelitian ini direktori Wordpress terdapat pada *root* /usr/share/nginx/wp/wordpress;.

Konfigurasi lengkap dapat terlihat seperti pada gambar 4.11 dan 4.12.

```
listen       80;
server_name  10.0.2.25;
root /usr/share/nginx/wp/wordpress;
index index.php;
limit_conn backend3_limit_conn 1;
limit_req zone=mybackend3 burst=10 nodelay;

port_in_redirect off;
server_tokens off;
autoindex off;

client_max_body_size 15m;
client_body_buffer_size 128k;

#charset koi8-r;
access_log  /var/log/nginx/testaccess.log  main;
error_log /var/log/nginx/testerror.log;

location / {
    limit_conn backend3_limit_conn 1;
    limit_req zone=mybackend3 burst=10 nodelay;
    try_files $uri $uri/ /index.php?$args;
    autoindex off;
}
```

Gambar 4.11 Nginx konfigurasi 1

```
# proxy the PHP scripts to Apache listening on 127.0.0.1:80
#
#location ~ \.php$ {
#    proxy_pass   http://127.0.0.1;
#}

# pass the PHP scripts to FastCGI server listening on 127.0.0.1:9000
#
location ~ \.php$ {
#    root           html;
    fastcgi_pass   127.0.0.1:9000;
    fastcgi_index  index.php;
    fastcgi_param  SCRIPT_FILENAME  $document_root$fastcgi_script_name;
    include        fastcgi_params;
    fastcgi_split_path_info ^(.+\.php)(/.+)$;
    fastcgi_intercept_errors on;
    fastcgi_ignore_client_abort off;
    fastcgi_connect_timeout 60;
    fastcgi_send_timeout 180;
    fastcgi_read_timeout 180;
    fastcgi_buffer_size 128k;
    fastcgi_buffers 4 256k;
    fastcgi_busy_buffers_size 256k;
    fastcgi_temp_file_write_size 256k;
}

# deny access to .htaccess files, if Apache's document root
# concurs with nginx's one
#
location ~ /\.ht {
    deny  all;
}
```

Gambar 4.12 Nginx konfigurasi 2

4.4 *Passing a Request to a Proxied Server* Menggunakan *Module Proxy_Pass*

Fungsi *proxy_pass* dalam metode *load balance* yang diterapkan pada penelitian ini adalah untuk menterjemahkan atau meprosess permintaan *client* terhadap *server*. Pada penelitian ini, *proxy_pass* dipergunakan untuk pendistribusian *client request* terhadap *backend server*, artinya *proxy_pass* digunakan untuk *menghandle request client* terhadap *backend server*. Selain untuk mendistribusikan beban *server load balance* kepada *backend server*, *proxy_pass* juga dapat digunakan untuk mengatur letak suatu *file* berdasarkan *url* seperti pada gambar 4.13.

```
location /some/path/ {
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_pass http://localhost:8000;
}
```

Gambar 4.13 Pengaturan *proxy_pass* berdasarkan *file path*

4.4.1 Penjelasan Cara Kerja Konfigurasi *Load Balance*

Upstream, merupakan *module* yang digunakan oleh Nginx dalam mendefinisikan kelompok-kelompok dari *backend server*. Pada penelitian difungsikan sebagai *module* yang mendefinisikan *server-server backend*. Selanjutnya adalah mendefinisikan metode *load balancing* yang akan digunakan. Pada penelitian ini menggunakan metode *least_conn* yang akan mendistribusikan *request client* kepada *backend server* secara merata sehingga tidak terjadi *overload* atau beban yang berlebihan pada *server-server backend*.

Selanjutnya yaitu mendefinisikan *module server* yang berguna untuk mendefinisikan beberapa fungsi seperti *server_name, proxy_pass, listening port*, serta *location* dari *server block*. Nginx melayani permintaan dari *client* berdasarkan *server block*, didalam Nginx konfigurasi, bisa terdapat lebih dari satu *server block*, seperti pada gambar 4.14.

```
server {
    listen       80;
    server_name  example.org  www.example.org;
    ...
}

server {
    listen       80;
    server_name  *.example.org;
    ...
}
```

Gambar 4.14 *Server Block*

Didalam *server block* terdapat beberapa fungsi seperti *server_name* yang berfungsi untuk mencari *host domain* pada s*erver block*, sedangkan *listen* mendefinisikan *server block* tersebut *running* dan *listen* pada *port* berapa. *Location* digunakan dalam mendefinisikan *root path* dari direktori *server block* dan didalam *root path* tersebut kembali memanggil *module proxy_pass* yang digunakan untuk mendistribusikan *request client* kepada *server-server backend*.

4.4.2 Mengoptimalkan Performa Nginx

Dalam mewujudkan keadaan *network*, serta kinerja dari *server* yang optimal, *server administration* perlu mengoptimalkan kinerja dari *server-server* yang dikelola, diantaranya akan dijelaskan seperti berikut:

a. *log_format* atau *ngx_http_log_module*, merupakan *module* yang berisikan fungsi yang digunakan untuk melakukan *log* atau *record* terhadap *remote address server block*, *remote user*, *request user*, *request http user*, dan tergantung dengan fungsi apa yang akan dipakai. Pada contoh gambar 4.15 terdapat *path directory* dari *access_log*.

```
log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                  '$status $body_bytes_sent "$http_referer" '
                  '"$http_user_agent" "$http_x_forwarded_for"'
                  '"$host" sn="$server_name" '
                  'rt=$request_time '
                  'ua="$upstream_addr" us="$upstream_status" '
                  'ut="$upstream_response_time" ul="$upstream_response_length" '
                  'cs=$upstream_cache_status' ;

access_log  /var/log/nginx/access.log  main;
```

Gambar 4.15 Nginx *http log module*

b. Langkah selanjutnya mengaktifkan fitur *module gzip compression*, fitur ini seringkali digunakan oleh para *sysadmin* untuk mengurangi atau memfilter data yang masuk dari *client* berdasarkan *gzip compress*. Konfigurasinya dapat terlihat pada gambar 4.16.

```
gzip  on;
gzip_comp_level  3;
gzip_static on;
gzip_disable "msie6";
gzip_vary on;
gzip_proxied any;
gzip_min_length 1000;
gzip_buffers 16 8k;
gzip_http_version 1.1;
gzip_types text/css text/javascript text/xml text/plain text/x-component
application/javascript application/x-javascript application/json
application/xml  application/rss+xml font/truetype application/x-font-ttf
font/opentype application/vnd.ms-fontobject image/svg+xml;
```

Gambar 4.16 Nginx *Gzip Compression Module*

c. Mengoptimalkan *client body buffer size* yang bertujuan untuk membatasi *request* berlebihan yang dapat menghabiskan *bandwith* pada *server*, dengan cara membaca dan menetapkan batas maksimal dari *client request*. Konfigurasi ini

juga bergantung pada ukuran dari *memory* yang akan digunakan, konfigurasinya dapat terlihat pada gambar 4.17.

```
client_max_body_size 100k;
client_body_timeout 10;
client_header_timeout 10;
client_body_buffer_size  128k;
client_header_buffer_size 1k;
large_client_header_buffers 2 4k;
```

Gambar 4.17 Nginx client body buffer size

4.5 TCP *Load Balancing*

TCP merupakan singkatan dari *Transmission Control Protocol*, merupakan *protocol* atau aturan yang terdapat pada layer ke 4 yaitu *transport layer*, didalam *OSI layer*. *Layer* 4 merupakan wadah bagi TCP dan UDP. TCP merupakan *protocol* yang melayani proses seperti *web, ssh, ftp, telnet, icmp process,* ataupun *smpt*. Didalam penelitian ini menggunakan protocol TCP untuk keperluan *web service* yang menggunakan mekanisme *load balance*. Proses TCP berkerja pada IP (*Internet Protocol*) untuk melakukan pertukaran data. TCP juga digunakan dalam mendukung metode *load balance,* dengan cara menambahkan fungsi *upstream module*. Untuk konfigurasi dalam penelitian, dapat terlihat pada gambar 4.18 berikut.

```
upstream test {
        least_conn;
        server 10.0.2.23;
        server 10.0.2.24;
        server 10.0.2.25;
}
```

Gambar 4.18 Nginx TCP *Load Balance*

4.5.1 Pencegahan DDOS *Attack*

Mengimplementasikan pencegahan DDOS *attack* pada penelitian *load balance* merupakan upaya untuk mendukung proses simulasi nantinya. Dikarenakan sesuai dengan *issue* yang terdapat pada *server* blog UMY, dimana *server* blog sering kehabisan *resources bandwith*, konfigurasi pencegahan dapat dilakukan dengan cara membatasi *bandwith* per *ip client* yang masuk, sehingga jika *client* memakai *bandwith* melampaui dengan ketentuan yang telah ditetapkan maka koneksi *client* akan di *drop*. Gambar 4.19 berikut ini merupakan contoh dari konfigurasi pencegahan *DDOS attack* pada Nginx.

```
limit_conn_zone $binary_remote_addr zone=backend1_limit_conn:5m;
limit_req_zone $binary_remote_addr zone=mybackend1:10m rate=5r/s;
```

```
location / {
    limit_conn backend1_limit_conn 1;
    limit_req zone=mybackend1 burst=10 nodelay;
    try_files $uri $uri/ /index.php?$args;
}
```

Gambar 4.19 Nginx Ddos Proteksi

Berikut merupakan beberapa penjelasan dari fungsi yang tertera pada gambar 4.19.

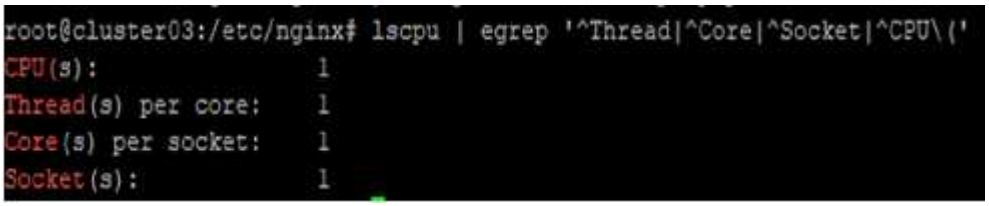
a. Dapat membatasi jumlah *request* per *ip client* yang akan masuk dengan cara memberikan fungsi *limit_conn* ataupun *limit_req_zone*. *Limit_req_zone module* ini nantinya yang akan membagi *request* per *ip client* yang di tentukan berdasarkan letak *path directory* yang akan dibatasi koneksi *client* dalam penelitian ini. Pada penelitian, *root path* direktori diberi nama *mybackend1*. *Rate* menunjukan berapa banyak *request* yang akan dilayani dalam satu detik, dalam penelitian ini membatasi hanya akan melayani lima *request client* dalam satu detik.

b. Selanjutnya adalah *module* atau fungsi *burst* dan *nodelay*, *burst* merupakan elemen juga yang sering digunakan dalam pencegahan *DDOS attack* atau dalam memaksimalkan performa dari Nginx. Terdapat pula *Burst* yang memiliki fungsi yaitu sebagai pembatas atau *limitation* terhadap suatu prosess yang sedang berlangsung berdasarkan *rate* dari *limit_req_zone*, jadi suatu proses tidak akan diteruskan pada *backend server* jika melebihi kapasitas akses per detik.

4.5.2 *Worker Proccess And Worker Connection*

Gambar 4.20 Nginx Worker Process and Worker Connection

Nginx berjalan berdasarkan proses pada CPU, dimana jumlah CPU yang berjalan merupakan salah satu faktor yang menentukan optimal tidaknya sebuah proses pada Nginx. Selain dengan kapasitas *hard disk* dan *memory* RAM, *Worker_processes* pada Nginx disesuaikan dengan jumlah CPU *core* pada *server*, dalam penelitian ini *worker_processes* bernilai satu, berdasarkan jumlah CPU yang digunakan pada *server*, terlihat pada gambar 4.21.



Gambar 4.21 Jumlah CPU Core

Selanjutnya merupakan *worker_connection* yang dimana angka dari *worker_connection* tersebut menggambarkan jumlah koneksi *client* yang mampu ditangani oleh Nginx secara serentak.

4.6 Nginx *Gzip Compression Module*

4.6.1 Nginx *Http Gzip Module*

Nginx menggunakan *module gzip* sebagai alat untuk memfilter dan melakukan kompresi terhadap data yang akan dikirimkan kepada *client*, karena cepat atau lambatnya sebuah *website* dalam merespon permintaan *client* ada kalanya tergantung dari ukuran dari sebuah data yang akan dikirimkan kepada *client*. Seorang *sysadmin* sebuah *server* terkadang dituntut untuk mengoptimalkan kinerja sebuah *web server* yang dikelola, salah satunya bisa dilakukan dengan cara melakukan kompresi terhadap ukuran *load data* atau *file* yang akan dikirim kepada *client*, semakin sedikit atau semakin kecil ukuran data bukan hanya membuat *website* menjadi lebih cepat untuk diakses, akan tetapi juga dapat mengurangi konsumsi *bandwith* sebuah *website* tersebut. Gzip merupakan fitur pada Nginx yang memiliki fungsi *data compression program* atau bisa juga disebut dengan program atau fungsi yang akan di eksekusi oleh Nginx dan berguna untuk melakukan kompresi dari ukuran sebuah data tersebut.

Gambar 4.22 yang mana merupakan konfigurasi *default* jika ingin mengaktifkan *module* atau fungsi *gzip compression*.

```
gzip  on;
gzip_comp_level  2;
gzip_static on;
gzip_disable "msie6";
gzip_vary on;
gzip_proxied any;
gzip_min_length 1000;
gzip_buffers 16 8k;
gzip_http_version 1.1;
gzip_types text/css text/javascript text/xml text/plain text/x-component
application/javascript application/x-javascript application/json
application/xml  application/rss+xml font/truetype application/x-font-ttf
font/opentype application/vnd.ms-fontobject image/svg+xml;
```

Gambar 4.22 Nginx Gzip Compression

Defaultnya layanan *gzip compression* dimatikan pada saat pertama menginstal Nginx, untuk mengaktifkannya dengan cara menghilangkan tanda pagar (‘#’) dan memberikan opsi *on* sehingga menjadi *gzip on.* Selanjutnya *gzip_comp_level* yang mana merupakan tingkat sebuah data dikompresi, terdapat *level* 1 sampai dengan *level* 9 tingkatan kompresi yang mana level 9 merupakan level dengan kompresi paling terkompres. Selanjutnya merupakan *gzip_static* yang merupakan perujukan jenis *file* yang telah terkompresi, biasanya *file* tersebut akan berubah extension menjadi .*gz* seperti pada gambar 4.23 dibawah ini.

```
Sep 29 13:45 access.log.10.gz
Sep 26 12:54 access.log.11.gz
Sep 25 13:32 access.log.12.gz
Sep 23 15:17 access.log.13.gz
Sep 22 17:57 access.log.14.gz
Okt 24 13:08 access.log.2.gz
Okt 23 09:35 access.log.3.gz
Okt 19 15:24 access.log.4.gz
Okt 14 14:37 access.log.5.gz
Okt 11 09:29 access.log.6.gz
Okt  6 11:17 access.log.7.gz
Okt  4 15:25 access.log.8.gz
Sep 30 13:30 access.log.9.gz
```

Gambar 4.23 Nginx Gzip Static

Tahap selanjutnya yaitu *gzip_min_length* dan *gzip_buffer*, yang pertama adalah *gzip_min_legth* yang merupakan *module gzip* yang berguna untuk menentukan panjang atau *size* dari ukuran *file* yang telah dikompresi yang diambil dari konten dan *header field* suatu *data*. Sedangkan *gzip_buffer* adalah *module* yang disiapkan untuk memberikan respon terhadap suatu kompresi yang artinya akan menentukan ukuran suatu *file* hasil kompresi.

4.7 Optimasi Dan Mini *Security*

4.7.1 HTTP *Secure Header*

Penelitian ini juga mencoba untuk mengurangi permasalahan *security* pada situs blog UMY. Tahap pertama ketika *client* akan mengakses sebuah *website* dengan memasukan *domain name* dari *website* tersebut, *browser* akan melakukan *http get request* dan dilanjutkan oleh *http respone* dari *web server website* tersebut, akan tetapi data yang terkandung selama proses tersebut haruslah tetap terjaga keamanannya untuk menghindari *man in the middle attack* ataupun masuknya *malware* ataupun *backdoor* kedalam *web server* tersebut, untuk itu maka perlu dilakukan beberapa konfigurasi tambahan seperti gambar 4.24 berikut.

```
# X-Frame-Options is to prevent from clickJacking attack
add_header X-Frame-Options SAMEORIGIN;

#  disable content-type sniffing on some browsers.
add_header X-Content-Type-Options nosniff;

# This header enables the Cross-site scripting (XSS) filter
add_header X-XSS-Protection "1; mode=block";

# This will enforce HTTP browsing into HTTPS and avoid ssl stripping attack
add_header Strict-Transport-Security "max-age=31536000; includeSubdomains;";
```

Gambar 4.24 Nginx HTTP secure Header

Berikut ini merupakan fungsi dari masing-masing parameter yang tertera pada gambar 4.24:

a. *add_header X-Frame-Options SAMEORIGIN*

   add_header X-Frame-Options SAMEORIGIN berfungsi sebagai anti *click jacking* yang artinya akan meminimalisir fungsi dari *untrusted iframe* untuk dieksekusi oleh *client*.

b. *add_header X-Content-Type-Options nosniff*

   Parameter *add_header X-Content-Type-Options nosniff* yang berguna untuk mencegah *sniffing* terhadap *browser user* atau *client*.

c. *add_header X-XSS-Protection "1; mode=block"*

   Parameter *add_header X-XSS-Protection "1; mode=block"* yang menandakan bahwa parameter ini memiliki fungsi sebagai XSS *attack* filter, yang artinya akan melakukan filter atau *protection* terhadap *client* yang akan melakukan injeksi XSS *Attack Script* pada *web server*.

d. *Strict-Transport-Security "max-age=31536000; includeSubdomains;"*

   Parameter terakhir memiliki fungsi sebagai *Http Strict Transport Security* yang mana artinya jika terdapat *client* yang mengakses *website* yang menggunakan TLS(*Transport Layer Security*) atau biasa dikenal dengan Https, akan tetapi user atau pengguna pada browsernya masih menggunakan *http protocol* atau bahkan tanpa menggunakan tag *http/https* (contoh: facebook.com atau http://www.facebook.com), parameter *strict transport security* ini yang nantinya akan mengalihkan *url browser* menjadi *https*.

4.8 Pengujian

4.8.1 Simulasi *Load Balance*

Simulasi *load balance* ini dilakukan dengan cara sebuah *client device* berupa *laptop* yang akan bertindak sebagai *client* akan mengakses Nginx *web server* yang menjadi *load balancer server* dan akan diarahkan kepada tiga *backend server* secara acak sesuai dengan *module* yang telah ditetapkan pada *load balance server*. Dapat dilihat pada gambar 4.25, 4.26, 4.27.

Gambar 4.25 Backend *Server* 1

Gambar 4.26 Backend *Server* 2

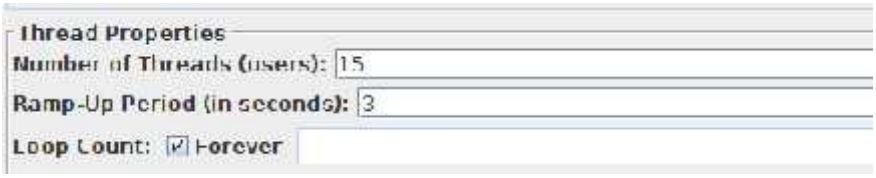
Gambar 4.27 Backend *Server* 3

Terlihat dari ketiga gambar yaitu, gambar 4.25,4.26, dan 4.27, *client* mengakses *load balancer server* pada waktu yang berbeda dan mendapatkan *backend server* yang berbeda pula, hal ini menunjukan kerja *load balance* telah berhasil.

4.8.2 DOS/DDOS *Attack* Simulasi

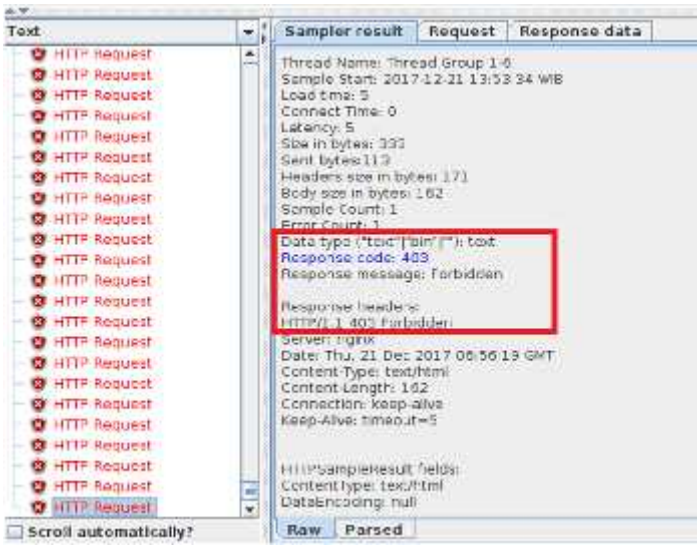
Pengujian dengan cara DOS *attack* atau dengan simulasi DOS ini bertujuan untuk mengetahui tingkat ketahanan *server* dalam meng*handle* atau menerima jumlah *request* dari pengguna layanan blog UMY, serta menguji beberapa *module* konfigurasi yang digunakan dalam mengatasi *request* atau permintaan terhadap layanan blog dari *client* yang tidak wajar atau dalam jumlah yang banyak dalam waktu yang singkat. Pada simulasi DOS ini jumlah dari *client* yang akan melakukan *request* terhadap *web server* telah ditentukan yaitu sebanyak 15 *user* (Terlihat pada gambar 4.28) dikarenakan jumlah

user ini dirasa telah cukup dalam melakukan simulasi DOS skala kecil berdasarkan perameter *module* pencegahan DOS pada gambar 4.19. Sedangkan waktu yang dibutuhkan untuk melakukan *reload request* sebanyak 3 detik (Terlihat pada gambar 4.28) sekali juga dirasa cukup dikarenakan telah melampaui batas yang ditentukan pada module *limit_conn* dan *limit_req* sebelumnya yaitu pada gambar 4.19.

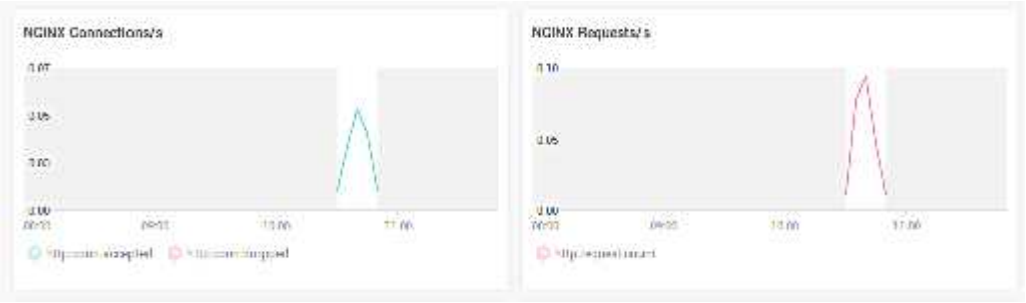


Gambar 4.28 Pengaturan jumlah user/request DDOS

Hasil dari pengujian ini akan terlihat pada gambar 4.29, dimana pada gambar tersebut terlihat Nginx melakukan penolakan atau *rejecting* terhadap proses *http request* yang tidak normal atau melebihi kapasitas yang sudah ditentukan. Pada jumlah permintaan yang meningkat, layanan blog masih dapat berjalan dikarenakan *request* telah ditolak dengan *http respon 403* (*forbidden*) dikarenakan telah membatasi jumlah *request* per *second* yang akan diterima oleh *web server* Nginx, dengan cara menambahkan module *limit_conn* seperti yang terlihat pada gambar 4.19.



Gambar 4.29 HTTP Request DOS



Gambar 4.30 Nginx Connection Request Result

Pada gambar 4.30 terlihat juga Nginx *connection* yang terhubung serta Nginx *request* yang terhubung, terlihat dari Nginx *request* yang begitu tinggi pada jam 10:15 sampai dengan 10.58 waktu percobaan ini dilakukan, yang menandakan telah terjadinya *request* atau permintaan terhadap layanan blog yang tinggi pada waktu tersebut.

4.8.3 Mencegah DOS/DDOS *Attack* dengan *Limit Connection*

Limit connection dan *limit request connection* pada Nginx dapat dipergunakan untuk membatasi jumlah akses *user* terhadap web *server*, hal ini diperuntukan untuk menjaga agar *web server* Nginx tidak terjadi *overload request* dan memastikan Nginx tetap

berkerja sebagai mana mestinya. Ada beberapa *rules* yang akan diterapkan pada *module limit_conn* dan *limit_req* yang akan membatasi akses terhadap *web server* seperti pada gambar 4.31.

```
#charset koi8-r;
access_log  /var/log/nginx/access.log  main;
error_log /var/log/nginx/error.log;
location / {
    limit_conn backend1_limit_conn 1;
    limit_req zone=mybackend1 burst=10 nodelay;
    limit_req_status 403;
    try_files $uri $uri/ /index.php?$args;
}
```

Gambar 4.31 Nginx *Limit Connection and Request*

Seperti yang terlihat pada gambar 4.31, *limit connection* (*limit_conn*) yang terapkan pada *directory root path* dari Nginx bernama *backend* 1 dengan *limit connection* 1, sedangkan pada *limit request* (*limit_req*) bernama *mybackend1* dan hanya akan mampu melayani jumlah dari *request* sebanyak 10 *request* per detik tanpa jeda, dan apabila melampaui ketetapan, maka akan ditampilkan status *forbidden* seperti gambar 4.29.