Servidores Web de Altas Prestaciones (2021-2022)
Grado en Ingeniería Informática
Universidad de Granada

Memoria Práctica 3

Pedro Antonio Mayorgas Parejo

23 de abril de 2022

Índice

1 Balanceador de carga Nginx **4**
- 1.1 Intalación de Nginx - Servidores HTTP básicos M1 y M2 . . . 4
  - 1.1.1 Instalación de Nginx - configurando el sitio predeterminado . . . 5
- 1.2 Instalación del Balanceador de carga en M3 . . . 6
- 1.3 Weighted Server - Servidores Ponderados . . . 9
- 1.4 Persistencia de sesiones . . . 9
  - 1.4.1 Configurando el KeepAlive de ip_hash . . . 13
- 1.5 Spare server y apagando servidores . . . 14

2 Balanceador de carga HAProxy **15**
- 2.1 Configuración de HAProxy para balanceo de carga HTTP . . . 17
- 2.2 Weighted load-balancer . . . 20
- 2.3 Sección de stats . . . 22
  - 2.3.1 Activando las comprobaciones de estado health check de los servidores 24
  - 2.3.2 Ocultando versión de HAProxy . . . 24

3 GoBetween load-balancer **26**
- 3.1 Creación de demonio para SystemD y controlar el servicio de gobetween . 28
- 3.2 Configuración básica de gobetween . . . 32
- 3.3 Configuración de round robin de gobetween . . . 34
- 3.4 Configuración de weighted o ponderación . . . 35
- 3.5 Comprobación del estado de los servidores healthcheck . . . 37
- 3.6 Límites de ficheros abiertos de gobetween . . . 39

4 Instalación de Zevenet **40**
- 4.1 Modos de configuración del balanceador Zevenet . . . 42
- 4.2 Configurando LSLB . . . 44
- 4.3 Pruebas del backend . . . 47

5 AB Apache Benchmark **48**
- 5.1 Configuración round-robin . . . 50
  - 5.1.1 Prueba de estrés con nginx. . . . 52
  - 5.1.2 Prueba de estrés con HAProxy. . . . 54
  - 5.1.3 Prueba de estrés con gobetween . . . 55
  - 5.1.4 Prueba de estrés con Zevenet . . . 56
  - 5.1.5 GNUPlot comparativo de Round Robin . . . 57
- 5.2 Configuración ponderada (weighted) . . . 58
  - 5.2.1 Prueba de estrés con nginx . . . 60
  - 5.2.2 Prueba de estrés con HAProxy . . . 61
  - 5.2.3 Prueba de estrés con gobetween . . . 62
  - 5.2.4 Prueba de estrés con Zevenet . . . 63
  - 5.2.5 GNUPlot comparativo weighted . . . 63

6 Bibliografía 64

1. Balanceador de carga Nginx

Nota: No lo indico pero en cada cambio del fichero nginx.conf o otros, reinicio el servicio de nginx para que el proceso que es ejecutado de nuevo por el demonio, obtenga la información nueva. Recomiendo el siguiente comando siempre que se haga algún cambio

```
sudo systemctl restart nginx.service
```

Por supuesto en el caso de que algo falle como un guión o algún error de la directiva se recomienda utilizar

```
sudo systemctl status nginx.service
```

Para trabajar con el Balanceador de carga partimos del siguiente diseño:

- Zona back-end, Weighted == Ponderación ver subsección:
  - m1-pedroamp: Servidor Nginx - Principal Weighted=3
  - m2-pedroamp: Servidor Nginx - Respaldo Weighted=2
- Zona front-end:
  - m3-pedroamp: Load-Balancer Nginx - Algoritmo principal Round-Robin.
- Zona Clientes:
  - m4-pedroamp: cURL, herramienta benchmarking.

1.1. Intalación de Nginx - Servidores HTTP básicos M1 y M2

Para la instalación voy a instalar el nginx básico y el motor back-end PHP para poder visualizar en cURL luego a qué servidor le hacemos la petición final.

```
sudo apt-get install nginx php php-fpm
```

php-fpm == PHP fastCGI process manager.

```
debian@m2-pedroamp:~$ sudo apt-get install nginx php php-fpm
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  bzip2 fontconfig-config fonts-dejavu-core geoip-database libdeflate0 libfontconfig1 libgd3
  libgdbm-compat4 libgeoip1 libjbig0 libjpeg62-turbo libnginx-mod-http-geoip
  libnginx-mod-http-image-filter libnginx-mod-http-xslt-filter libnginx-mod-mail libnginx-mod-stream
  libnginx-mod-stream-geoip libperl5.32 libsodium23 libtiff5 libwebp6 libx11-6 libx11-data libxau6
  libxcb1 libxdmcp6 libxpm4 libxslt1.1 mailcap mime-support nginx-common nginx-core perl
  perl-modules-5.32 php-common php7.4 php7.4-cli php7.4-common php7.4-fpm php7.4-json php7.4-opcache
  php7.4-readline
Suggested packages:
  bzip2-doc libgd-tools geoip-bin fcgiwrap nginx-doc ssl-cert perl-doc libterm-readline-gnu-perl
  | libterm-readline-perl-perl make libtap-harness-archive-perl php-pear
The following NEW packages will be installed:
  bzip2 fontconfig-config fonts-dejavu-core geoip-database libdeflate0 libfontconfig1 libgd3
  libgdbm-compat4 libgeoip1 libjbig0 libjpeg62-turbo libnginx-mod-http-geoip
  libnginx-mod-http-image-filter libnginx-mod-http-xslt-filter libnginx-mod-mail libnginx-mod-stream
  libnginx-mod-stream-geoip libperl5.32 libsodium23 libtiff5 libwebp6 libx11-6 libx11-data libxau6
  libxcb1 libxdmcp6 libxpm4 libxslt1.1 mailcap mime-support nginx nginx-common nginx-core perl
  perl-modules-5.32 php php-common php-fpm php7.4 php7.4-cli php7.4-common php7.4-fpm php7.4-json
  php7.4-opcache php7.4-readline
0 upgraded, 45 newly installed, 0 to remove and 0 not upgraded.
Need to get 20.4 MB of archives.
After this operation, 90.2 MB of additional disk space will be used.
Do you want to continue? [Y/n]
```

Figura 1: Paquetes y dependencias a instalar de nginx y PHP

1.1.1. Instalación de Nginx - configurando el sitio predeterminado

Para configurar el sitio predeterminado debemos ir a la ruta /etc/nginx/sites-avaiable/default, donde debemos ver las siguientes opciones y cambiarlas por las de nuestro servidor.

- En la opción server_name configuramos el nombre del servidor m1-pedroamp o m2-pedroamp
- En la opción de index añadimos index.php para que lea nuestro fichero de PHP para que pueda hacer un trabajo automático.
- en el bloque de location php
  - include fast-cgi-php.conf ->Fichero de configuración de PHP
  - fastcgi_index index.php ->Indicamos el index.
  - fastcgi_pass unix:/run/php/php7.4-fpm.sock ->Indicamos el socket de UNIX de PHP

Siendo el resultado como el siguiente:

```
GNU nano 5.4                          /etc/nginx/sites-available/default
        #
        # include snippets/snakeoil.conf;

        root /var/www/html;

        # Add index.php to the list if you are using PHP
        index index.php;

        server_name m1-pedroamp;

        location / {
                # First attempt to serve request as file, then
                # as directory, then fall back to displaying a 404.
                try_files $uri $uri/ =404;
        }

        # pass PHP scripts to FastCGI server
        #
        location ~ \.php$ {
                fastcgi_index index.php;
                include fastcgi.conf;
                fastcgi_pass unix:/run/php/php7.4-fpm.sock;
        }
```

Figura 2: Fichero de configuración por defecto de Nginx

Luego en el index.php debemos incluir lo siguiente para que cuando solicitemos una página en el futuro a través del balanceador de carga podamos ver qué efectos produce.

```
  GNU nano 5.4                                /var/www/html/index.php
<hmtl>
        <head>
                <title>Website Request</title>
        </head>
        <body>
                <p>Prueba</p>
                <?php
                        echo "<br>";
                        echo $_SERVER['SERVER_NAME'];
                        echo "<br>";
                        echo $_SERVER['SERVER_ADDR'];
                        echo "<br>";
                ?>
        </body>
</html>
```

Figura 3: Index indicador de cuál servidor back-end es solicitado

1.2. Instalación del Balanceador de carga en M3

Ahora procedemos a instalar el balanceador de carga en m3-pedroamp. Dicho balanceador de carga es de NGINX, sin otras opciones necesarias.
Primero tenemos que desactivar de /etc/nginx/sites-enabled/ el enlace simbólico del sitio activado por defecto de nginx.

```
sudo rm /etc/nginx/sites-enabled/default
```

Nota: El dominio mswap-pedroamp, debe estar apuntando solo al servidor de balanceo de carga

A continuación en /etc/nginx/nginx.conf debemos configurar los siguientes parámetros:

- upstream <Nombre aplicación>Indica cuál es el ServerName de la agrupación de servidores, esto es como el VirtualHost, cuando se lea una solicitud a este nombre se pasa a la agrupación. En mi caso la llamaré mswap-pedroamp
- server <IP/NameServer>Indicamos cual es el servidor que va a ofrecer el mismo contenido en la agrupación de servidores.
- proxy_pass http://mswap-pedroamp. Indica que al escuchar una petición de HTTP o puerto 80 se lo pase a la agrupación de servidores.

```
 GNU nano 5.4                              /etc/nginx/nginx.conf
user www-data;
worker_processes auto;
pid /run/nginx.pid;
include /etc/nginx/modules-enabled/*.conf;

events {
        worker_connections 768;
        # multi_accept on;
}

http {
        upstream mswap-pedroamp {
                server m1-pedroamp;
                server m2-pedroamp;
        }

        server {
                listen 80;
                location / {
                        proxy_pass http://mswap-pedroamp;
                }

        }

        access_log /var/log/nginx/access.log;
        error_log /var/log/nginx/error.log;


}
```

Figura 4: Configuración final del balanceador de carga

Pruebas realizadas:

```
debian@m4-pedroamp:~$ curl mswap-pedroamp
<hmtl>
        <head>
                <title>Website Request</title>
        </head>
        <body>
                <p>Prueba</p>
                <br>m1-pedroamp<br>192.168.122.70<br>   </body>
</html>

debian@m4-pedroamp:~$ curl mswap-pedroamp
<html>
        <head>
                <title>Website Request</title>
        </head>
        <body>
                <p>Prueba</p>
                <br>m2-pedroamp<br>192.168.122.181<br>  </body>
</html>
debian@m4-pedroamp:~$ cat /etc/hosts | tail -n 5
192.168.122.70  m1-pedroamp
192.168.122.181 m2-pedroamp
192.168.122.179 m3-pedroamp
192.168.122.254 m4-pedroamp
192.168.122.179 mswap-pedroamp
debian@m4-pedroamp:~$
```

Figura 5: Pruebas de acceso al balanceador de carga

1.3. Weighted Server - Servidores Ponderados

Imaginemos que tenemos 5 peticiones pendientes a repartir con el algoritmo de Round-Robin a nuestros servidores m1 y m2.
Sabiendo que para m1-pedroamp weight=3 y m2-pedroamp weight=2.
El reparto de dichas peticiones realizadas por el cliente son 3 para m1-pedroamp y 2 para m2-pedroamp.
Entonces su configuración en /etc/nginx/nginx.conf del balanceador de carga es:

```
 GNU nano 5.4                          /etc/nginx/nginx.conf
events {
        worker_connections 768;
        # multi_accept on;
}

http {
        upstream mswap-pedroamp {
                server m1-pedroamp weight=3;
                server m2-pedroamp weight=2;
        }

        server {
                listen 80;
                location / {
                        proxy_pass http://mswap-pedroamp;
                }

        }
```

Figura 6: Configuración en el balanceador de carga solo con weighted - ponderación

Esta configuración sin embargo es demasiado básica y da muchos problemas en las webs modernas por lo que voy a explicar a continuación.

1.4. Persistencia de sesiones

Muchas veces al tener balanceador de carga necesitamos una persistencia de las sesiones debido a que los datos que están almacenados en variables locales del lado de servidor, no las tiene todo el upstream, por lo tanto se pueden perder muchos datos como la información de sesión del carrito...
Entonces necesitamos decirle al balanceador en el caso de que llegue una cierta petición, la mantenga en el servidor un tiempo definido por un keepalive por defecto.
¿Cómo funciona el mecanismo?
Se genera un hash a partir de la dirección IPv4 del cliente que realiza una petición HTTP REQUEST. Entonces con el hash es una clave que determina cual servidor del upstream o grupo de servidores es seleccionado para los subsecuentes RESPONSE HTTP o respuesta HTTP.
La directiva es dentro del upstream en /etc/nginx/nginx.conf ponemos *ip_ hash*

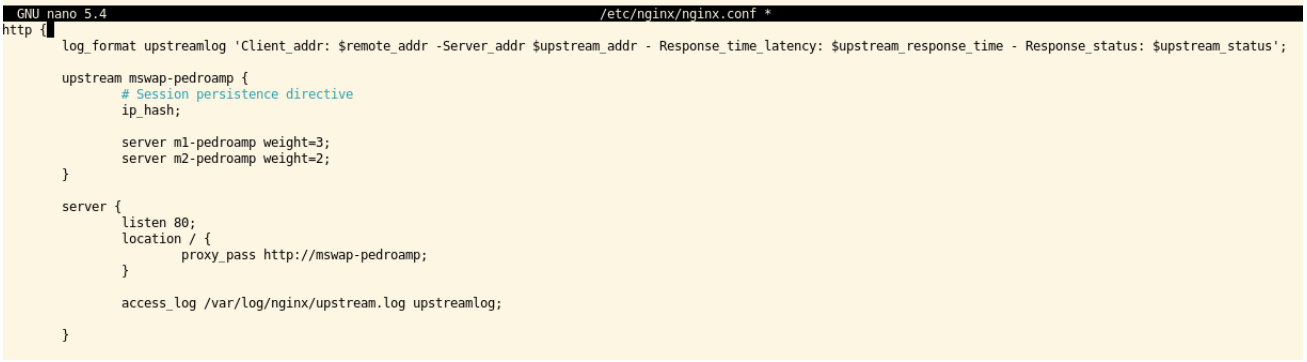

Figura 7: Configuración del balanceador con el ip_hash y el log

He creado una directiva de log especial que me permita detectar las direcciones de los clientes y a cual servidor del upstream le reenvia la solicitud.
Consiste en la sección de /etc/nginx/nginx.conf del balanceador configuremos lo siguiente:

- En http: Tengamos que poner la directiva log_format name string;
  - La utilidad de esta directiva es permitir un log format concreto el elaborado por mí está compuesto por el siguiente formato:
  - log_format Ćlient_addr: $remote_addr - Server_addr: $upstream_addr - Response_time_latency: $upstream_response_time - Response_status: $upstream_status ´
  - Todas las variables se pueden encontrar en la documentación de nginx que añado en la bibliografía, en concreto *todas afectan al módulo upstream de nginx*. Que nos da en resumen quién es el que solicita la petición, a qué servidor le reenviamos la petición, qué tiempo de respuesta ha tenido el servidor desde que le reenviamos la petición y con qué código de HTTP responde.
- En server: indicamos el acces_log que indica la ruta donde se va a almacenar el log de upstream

```
 GNU nano 5.4                                                    /etc/nginx/nginx.conf *
http {
        log_format upstreamlog 'Client_addr: $remote_addr -Server_addr $upstream_addr - Response_time_latency: $upstream_response_time - Response_status: $upstream_status';

        upstream mswap-pedroamp {
                # Session persistence directive
                ip_hash;

                server m1-pedroamp weight=3;
                server m2-pedroamp weight=2;
        }

        server {
                listen 80;
                location / {
                        proxy_pass http://mswap-pedroamp;
                }

                access_log /var/log/nginx/upstream.log upstreamlog;

        }
```

Figura 8: Configuración del log para registrar los movimientos del balanceador con el ip_hash

Realizo unas peticiones de HTTP desde un cliente con cURL, donde gracias a PHP podemos ver a qué servidor llega gracias a su variable global $_SERVER[SERVER_ADDR]:

```
debian@m4-pedroamp:~$ curl mswap-pedroamp
<hmtl>
        <head>
                <title>Website Request</title>
        </head>
        <body>
                <p>Prueba</p>
                <br>m1-pedroamp<br>192.168.122.70<br>   </body>
</html>

debian@m4-pedroamp:~$ curl mswap-pedroamp
<hmtl>
        <head>
                <title>Website Request</title>
        </head>
        <body>
                <p>Prueba</p>
                <br>m1-pedroamp<br>192.168.122.70<br>   </body>
</html>

debian@m4-pedroamp:~$
```

Figura 9: Peticiones de cURL - cliente

En el log podemos ver la consistencia que tiene con respecto a la respuesta obtenida con el cliente:

```
debian@m3-pedroamp:~$ cat /var/log/nginx/upstream.log
Client_addr: 192.168.122.254 -Server_addr 192.168.122.70:80 - Response_time_latency: 0.000 - Response_status: 200
Client_addr: 192.168.122.254 -Server_addr 192.168.122.70:80 - Response_time_latency: 0.000 - Response_status: 200
Client_addr: 192.168.122.254 -Server_addr 192.168.122.70:80 - Response_time_latency: 0.000 - Response_status: 200
```

Figura 10: Log del balanceador, se puede ver la redirección

1.4.1. Configurando el KeepAlive de ip_hash

Para no sobrecargar al servidor con sesiones abiertas eternamente, o un servidor siempre se encargue del mismo cliente, tenemos que definir un keepalive que haga que el balanceador termine su sesión pasado cierto tiempo.
He configurado 3 directivas:

- Directiva de conexiones Keepalive que se mantienen en cada thread/proceso worker cuando se supere el n de conexiones se cierran las mas antiguas. Un n muy alto supone consumir CPU, esto es que si un servidor nginx tiene, 4 hilos de proceso son 4 x 40 conexiones simultaneas - keepalive 40;
- Directiva de limite de peticiones que los servidores deben responder, un n alto supone consumir muchos recursos al mantenerse en espera con el estado de la conexion, ademas de ser susceptible el servidor a un ataque DoS al mantener la conexion con demasiadas peticiones. En mi caso las fijo con 100 - keepalive_request 100;
- Directiva de timeout o finalizacion de tiempo, es necesaria para en los casos en los que una conexion se mantiene en espera sin contestacion. Se recomienda el tiempo mas bajo - keepalive_timeout 30s;

```
upstream mswap-pedroamp {
        # Session persistence directive
        ip_hash;

        server m1-pedroamp weight=3;
        server m2-pedroamp weight=2;

        # Keepalive directives
        # Directiva de conexiones Keepalive que se mantienen en cada thread/proceso worker
        # cuando se supere el n de conexiones se cierran las mas antiguas.
        # Un n muy alto supone consumir CPU, esto es que si un servidor nginx tiene,
        # 4 hilos de proceso son 4 x 40 conexiones simultaneas
        keepalive 40;
        # Directiva de limite de peticiones que los servidores deben responder,
        # un n alto supone consumir muchos recursos al mantenerse en espera con el estado
        # de la conexion, ademas de ser susceptible el servidor a un ataque DoS al mantener
        # la conexion con demasiadas peticiones.
        # En mi caso las fijo con 100
        keepalive_requests 100;
        # Directiva de timeout o finalizacion de tiempo, es necesaria para en los casos en
        # los que una conexion se mantiene en espera sin contestacion. Se recomienda el
        # tiempo mas bajo
        keepalive_timeout 30s;
}
server {
```

Figura 11: Directivas aplicadas

Ahora con esta directiva se mantienen los clientes casi permanentemente conectados al mismo servidor. Recomiendo quitar la ponderación para ver resultados mas variados ya que suele preguntar al mismo servidor que tenga mayor peso porque no hay un número grande de clientes.

1.5. Spare server y apagando servidores

Finalmente la última parte de nginx avanzado consiste en que podamos poner servidores spare de respaldo y podamos parar servidores.
He quitado las directivas de keepalive y de ip_hash para poder agilizar el proceso de pruebas y ver como afecta la parada e introdución de un nuevo servidor. Además el parámetro de backup según tengo entendido con la documentación no es compatible con ip_hash.

```
http {
        log_format upstreamlog 'Client_addr: $remote_addr -Server_addr $upstream_addr - Response_time_la>

        upstream mswap-pedroamp {
                server m1-pedroamp down;
                server m2-pedroamp down;
                server m5-pedroamp backup;
        }
        server {
                listen 80;
                location / {
                        proxy_pass http://mswap-pedroamp;
                }

                access_log /var/log/nginx/upstream.log upstreamlog;

        }

        access_log /var/log/nginx/access.log;
        error_log /var/log/nginx/error.log;
```

Figura 12: Configuración de servidor de backup.

Ahora con la directiva *down* paramos el servidor m1-pedroamp para probar la configuración de backup y que el cliente consiga obtener la información a través de m5-pedroamp. En mi caso he tenido que parar los dos servidores para poder hacer que el de backup pueda responder peticiones de HTTP.

```
debian@m4-pedroamp:~$ curl mswap-pedroamp
<html>
        <head>
                <title>Website Request</title>
        </head>
        <body>
                <p>Prueba</p>
                <br>m2-pedroamp<br>192.168.122.181<br>  </body>
</html>
debian@m4-pedroamp:~$ curl mswap-pedroamp
<hmtl>
        <head>
                <title>Website Request</title>
        </head>
        <body>
                <p>Prueba</p>
                <br>m5-pedroamp<br>192.168.122.219<br>        </body>
</html>
debian@m4-pedroamp:~$
```

Figura 13: Comprobación de nginx

2. Balanceador de carga HAProxy

Debemos parar el servicio de nginx como balanceador de carga en m3-pedroamp debido a que permanece en escucha en el mismo puerto que el balanceador de HAProxy.
He ejecutado los siguientes comandos para parar el servicio de nginx:
El primer comando es para desactivar el autoarranque del servicio de nginx..

```
sudo systemctl disable nginx.service
```

El segundo comando es para parar el servicio de nginx.

```
sudo systemctl stop nginx.service
```

```
debian@m3-pedroamp:~$ sudo systemctl disable nginx.service
Synchronizing state of nginx.service with SysV service script with /lib/systemd/systemd-sysv-install.
Executing: /lib/systemd/systemd-sysv-install disable nginx
Removed /etc/systemd/system/multi-user.target.wants/nginx.service.
debian@m3-pedroamp:~$ sudo systemctl stop nginx.service
debian@m3-pedroamp:~$ sudo systemctl status nginx.service
● nginx.service - A high performance web server and a reverse proxy server
     Loaded: loaded (/lib/systemd/system/nginx.service; disabled; vendor preset: enabled)
     Active: inactive (dead)
       Docs: man:nginx(8)

Apr 01 19:53:27 m3-pedroamp systemd[1]: Starting A high performance web server and a reverse proxy serve>
Apr 01 19:53:27 m3-pedroamp systemd[1]: Started A high performance web server and a reverse proxy server.
Apr 01 19:55:32 m3-pedroamp systemd[1]: Stopping A high performance web server and a reverse proxy serve>
Apr 01 19:55:32 m3-pedroamp systemd[1]: nginx.service: Succeeded.
Apr 01 19:55:32 m3-pedroamp systemd[1]: Stopped A high performance web server and a reverse proxy server.
Apr 01 19:55:32 m3-pedroamp systemd[1]: Starting A high performance web server and a reverse proxy serve>
Apr 01 19:55:32 m3-pedroamp systemd[1]: Started A high performance web server and a reverse proxy server.
Apr 01 19:57:28 m3-pedroamp systemd[1]: Stopping A high performance web server and a reverse proxy serve>
Apr 01 19:57:28 m3-pedroamp systemd[1]: nginx.service: Succeeded.
Apr 01 19:57:28 m3-pedroamp systemd[1]: Stopped A high performance web server and a reverse proxy server.
debian@m3-pedroamp:~$
```

Figura 14: Parando nginx

Para instalar HAProxy ponemos el siguiente comando:

```
sudo apt-get install haproxy
```

```
debian@m3-pedroamp:~$ sudo apt-get install haproxy
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  liblua5.3-0
Suggested packages:
  vim-haproxy haproxy-doc
The following NEW packages will be installed:
  haproxy liblua5.3-0
0 upgraded, 2 newly installed, 0 to remove and 0 not upgraded.
Need to get 2019 kB of archives.
After this operation, 4316 kB of additional disk space will be used.
Do you want to continue? [Y/n]
```

Figura 15: Instalación de HAProxy

Ahora comprobamos que el servicio de HAProxy se encuentre activado:

```
sudo systemctl status haproxy.service
```

```
debian@m3-pedroamp:~$ sudo systemctl status haproxy.service
● haproxy.service - HAProxy Load Balancer
     Loaded: loaded (/lib/systemd/system/haproxy.service; enabled; vendor preset: enabled)
     Active: active (running) since Tue 2022-04-05 13:00:48 UTC; 29min ago
       Docs: man:haproxy(1)
             file:/usr/share/doc/haproxy/configuration.txt.gz
    Process: 551 ExecStartPre=/usr/sbin/haproxy -f $CONFIG -c -q $EXTRAOPTS (code=exited, status=0/SUCCE>
   Main PID: 556 (haproxy)
      Tasks: 3 (limit: 4678)
     Memory: 48.9M
        CPU: 329ms
     CGroup: /system.slice/haproxy.service
             ├─556 /usr/sbin/haproxy -Ws -f /etc/haproxy/haproxy.cfg -p /run/haproxy.pid -S /run/haproxy>
             └─560 /usr/sbin/haproxy -Ws -f /etc/haproxy/haproxy.cfg -p /run/haproxy.pid -S /run/haproxy>

Apr 05 13:00:48 m3-pedroamp systemd[1]: Starting HAProxy Load Balancer...
Apr 05 13:00:48 m3-pedroamp haproxy[556]: [NOTICE] 094/130048 (556) : New worker #1 (560) forked
Apr 05 13:00:48 m3-pedroamp systemd[1]: Started HAProxy Load Balancer.
debian@m3-pedroamp:~$
```

Figura 16: Comprobación del estado del servicio de HAProxy

2.1. Configuración de HAProxy para balanceo de carga HTTP

Debemos ir al fichero haproxy.cfg en la ruta /etc/haproxy/haproxy.cfg
Dentro del documento tenemos que poner al igual que en nginx como balanceador de carga aka Proxy Inverso. Una sección *frontend* que indica el bind (vínculo) con el servidor HTTP al que le tenemos que reenviar la petición de HTTP entrante al balanceador.
Por defecto el balanceador HAproxy viene preparado en su sección *defaults* para las peticiones HTTP. Que son las siguientes:

- Mode indica en qué protocolo de la Capa 4 (Transporte TCP-UDP full-duplex) o de la capa 7 (Aplicación HTTP). Debe trabajar.
- timeout connect - timeout de 5000 ms == 5 secs de la conexión.
- timeout client - timeout de 50000 ms == 50 secs del cliente, esto es para cuando el cliente tarda demasiado en enviar el dato/petición después del Three-Way handshake de TCP inicial antes de iniciar la conexión HTTP Hello o cuando se espera un ACK Acknowledge por parte del cliente ante una respuestas (response) del servidor.
- timeout server - timeout de 50000 ms == 50 secs del servidor, esto se determina para cuando tengamos un tiempo de respuesta demasiado alto del servidor en el cual tarda mucho en enviar la respuesta (response) HTTP.
- se recomienda que el timeout del server y cliente sean iguales ante la dificultad que se presentaría a la hora de buscar errores (debug) por la situación compleja que plantearía la diferencia de timeout.

Ahora tenemos que crear una sección llamada *frontend nginxfrontend* para indicar cuál es el balanceador de carga y a qué puerto se reciben peticiones HTTP.

- bind *:80 se vincula el balanceador de carga a la escucha de todas las peticiones de todas las direcciones IP en el puerto 80.
- default_backend nginxservers Se indica a qué servidores se deben reenviar las peticiones escuchadas.

En la sección *backend nginxservers*, se ponen las direcciones IP o nombres de los servidores.
Quedando finalmente como sigue:

```
  GNU nano 5.4                                /etc/haproxy/haproxy.cfg
global
        log /dev/log    local0
        log /dev/log    local1 notice
        chroot /var/lib/haproxy
        stats socket /run/haproxy/admin.sock mode 660 level admin expo
        stats timeout 30s
        user haproxy
        group haproxy
        daemon

        # Default SSL material locations
        ca-base /etc/ssl/certs
        crt-base /etc/ssl/private

        # See: https://ssl-config.mozilla.org/#server=haproxy&server-v
        ssl-default-bind-ciphers ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-R
        ssl-default-bind-ciphersuites TLS_AES_128_GCM_SHA256:TLS_AES_2
        ssl-default-bind-options ssl-min-ver TLSv1.2 no-tls-tickets

defaults
        log     global
        mode    http
        option  httplog
        option  dontlognull
        timeout connect 5000
        timeout client  50000
        timeout server  50000
        errorfile 400 /etc/haproxy/errors/400.http
        errorfile 403 /etc/haproxy/errors/403.http
        errorfile 408 /etc/haproxy/errors/408.http
        errorfile 500 /etc/haproxy/errors/500.http
        errorfile 502 /etc/haproxy/errors/502.http
        errorfile 503 /etc/haproxy/errors/503.http
        errorfile 504 /etc/haproxy/errors/504.http

frontend nginxfrontend
        bind *:80
        default_backend nginxservers

backend nginxservers
        server m1-pedroamp m1-pedroamp:80
        server m2-pedroamp m2-pedroamp:80
```

Figura 17: Fichero de configuración de HAProxy

Luego a continuación reiniciamos el servicio y probamos su funcionamiento.

```
sudo systemctl restart haproxy.service
```

Figura 18: Pruebas con round-robin

2.2. Weighted load-balancer

En HAProxy podemos también hacer una configuración de servidores ponderados. Como en Nginx, debemos añadir un parámetro a la directiva server que hemos visto anteriormente para indicar la ponderación.
Sigo el mismo procedimiento que antes, pongo al primer servidor un peso de 3 y al segundo un peso de 2.

```
  GNU nano 5.4                              /etc/haproxy/haproxy.cfg
global
        log /dev/log    local0
        log /dev/log    local1 notice
        chroot /var/lib/haproxy
        stats socket /run/haproxy/admin.sock mode 660 level admin expose-fd
        stats timeout 30s
        user haproxy
        group haproxy
        daemon

        # Default SSL material locations
        ca-base /etc/ssl/certs
        crt-base /etc/ssl/private

        # See: https://ssl-config.mozilla.org/#server=haproxy&server-versio
        ssl-default-bind-ciphers ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AE
        ssl-default-bind-ciphersuites TLS_AES_128_GCM_SHA256:TLS_AES_256_GC
        ssl-default-bind-options ssl-min-ver TLSv1.2 no-tls-tickets

defaults
        log     global
        mode    http
        option  httplog
        option  dontlognull
        timeout connect 5000
        timeout client  50000
        timeout server  50000
        errorfile 400 /etc/haproxy/errors/400.http
        errorfile 403 /etc/haproxy/errors/403.http
        errorfile 408 /etc/haproxy/errors/408.http
        errorfile 500 /etc/haproxy/errors/500.http
        errorfile 502 /etc/haproxy/errors/502.http
        errorfile 503 /etc/haproxy/errors/503.http
        errorfile 504 /etc/haproxy/errors/504.http

frontend nginxfrontend
        bind *:80
        default_backend nginxservers

backend nginxservers
        server m1-pedroamp m1-pedroamp:80 weight 3
        server m2-pedroamp m2-pedroamp:80 weight 2
```

Figura 19: Wheighted servers

Después de establecer el peso reiniciamos el servicio.
Pruebas con el cliente, en teoría de cada 5 peticiones 3 son desviadas hacia el m1-pedroamp, 2 de ellas desviadas al m2-pedroamp.

```
debian@m4-pedroamp:~$ curl mswap-pedroamp
<hmtl>
        <head>
                <title>Website Request</title>
        </head>
        <body>
                <p>Prueba</p>
                <br>m1-pedroamp<br>192.168.122.70<br>   </body>
</html>

debian@m4-pedroamp:~$ curl mswap-pedroamp
<hmtl>
        <head>
                <title>Website Request</title>
        </head>
        <body>
                <p>Prueba</p>
                <br>m1-pedroamp<br>192.168.122.70<br>   </body>
</html>

debian@m4-pedroamp:~$ curl mswap-pedroamp
<html>
        <head>
                <title>Website Request</title>
        </head>
        <body>
                <p>Prueba</p>
                <br>m2-pedroamp<br>192.168.122.181<br>  </body>
</html>
debian@m4-pedroamp:~$ curl mswap-pedroamp
<hmtl>
        <head>
                <title>Website Request</title>
        </head>
        <body>
                <p>Prueba</p>
                <br>m1-pedroamp<br>192.168.122.70<br>   </body>
</html>

debian@m4-pedroamp:~$ curl mswap-pedroamp
<html>
        <head>
                <title>Website Request</title>
        </head>
        <body>
                <p>Prueba</p>
                <br>m2-pedroamp<br>192.168.122.181<br>  </body>
</html>
debian@m4-pedroamp:~$ curl mswap-pedroamp
<hmtl>
        <head>
                <title>Website Request</title>
        </head>
        <body>
                <p>Prueba</p>
                <br>m1-pedroamp<br>192.168.122.70<br>   </body>
</html>
```

Figura 20: Pruebas con wheight

2.3. Sección de stats

Para poder configurar las estadísticas de HAProxy necesitamos añadir una nueva sección de *frontend stats* donde indicamos que activamos las estadísticas bajo los siguientes parámetros:

- bind *:8404 - El enlace permite las escuchas de todas las direcciones IPv4 hacia el puerto TCP 8404.
- stats enable - Indicamos que queremso activar las estadísticas con la configuración indicada en frontend stats
- stats uri /stats - Es una ruta de Uniform Resource Indentifier donde indicamos que queremos las estadísticas en esa ruta.
- stats refresh 10s - Indica la tasa de refresco automática de las estadísticas.
- auth admin:admin - Indicamos directamente en la configuración que nos pregunte por un usuario y una contraseña al acceder a la dirección y al recurso indicado. Se puede configurar varios usuarios para acceder al informe estadístico. Lo mejor es poner una contraseña fuerte.

```
frontend stats
        bind *:8404
        stats enable
        stats uri /stats
        stats refresh 10s
        stats auth  admin:admin
```

Figura 21: Configuración básica de estadística de HAProxy

Esta es la configuración estándar donde podemos acceder a la página previa pregunta de autenticación.

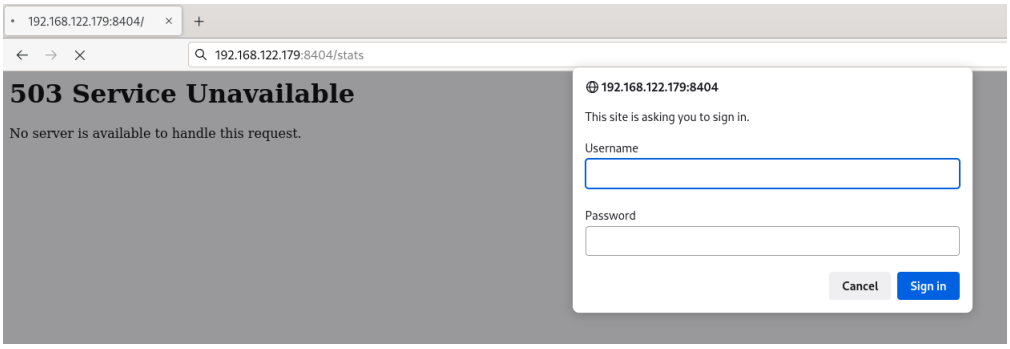

Figura 22: Login de HAProxy

Ahora dentro podemos ver que tenemos 3 secciones.

- La sección de nginxfrontend, existe ya que los balanceadores de carga se puede hacer de manera redundante entonces, en esta sección existe una configuración más avanzada que permite que un balanceador de carga secundario pueda tomar el control en el caso de caída del principal.
- La sección de nginxserver, se puede visualizar que los servidores son comprobados y tienen métricas de reenvío, sesiones, cola de peticiones, Errores, denegación.
- La sección activada previamente de status, ya que no queremos que la autenticación se pida además en las peticiones de nginx.

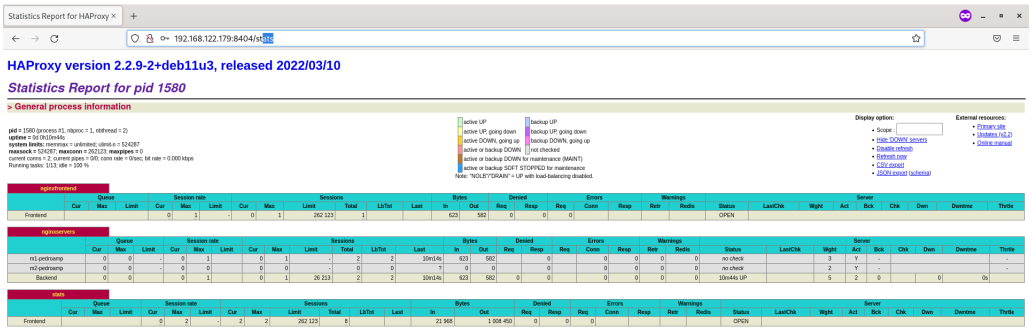

Figura 23: Panel de estadísticas de HAProxy

2.3.1. Activando las comprobaciones de estado health check de los servidores

Para poder comprobar el estado en tiempo real de los servidores que trabajan con el balanceador de carga debemos poner un argumento más en la entrada de cada server. Dicho argumento es *server IP:PORT check*

```
backend nginxservers
        server m1-pedroamp m1-pedroamp:80 check weight 3
        server m2-pedroamp m2-pedroamp:80 check weight 2
```

Figura 24: Activando comprobación de estado de servidores en HAProxy

Reiniciamos el servicio de HAProxy y a continuación he parado el servicio de nginx en el m2-pedroamp.

```
debian@m2-pedroamp:~$ sudo systemctl stop nginx.service
debian@m2-pedroamp:~$ sudo systemctl status nginx.service
● nginx.service - A high performance web server and a reverse proxy server
     Loaded: loaded (/lib/systemd/system/nginx.service; enabled; vendor preset: enabled)
     Active: inactive (dead) since Tue 2022-04-05 18:33:40 UTC; 5min ago
       Docs: man:nginx(8)
    Process: 545 ExecStartPre=/usr/sbin/nginx -t -q -g daemon on; master_process on; (code=exited, status=0/SUCCESS)
    Process: 569 ExecStart=/usr/sbin/nginx -g daemon on; master_process on; (code=exited, status=0/SUCCESS)
    Process: 1563 ExecStop=/sbin/start-stop-daemon --quiet --stop --retry QUIT/5 --pidfile /run/nginx.pid (code=exited, status=0/SUCCESS)
   Main PID: 592 (code=exited, status=0/SUCCESS)
        CPU: 67ms

Apr 05 13:00:44 m2-pedroamp systemd[1]: Starting A high performance web server and a reverse proxy server...
Apr 05 13:00:45 m2-pedroamp systemd[1]: Started A high performance web server and a reverse proxy server.
Apr 05 18:33:40 m2-pedroamp systemd[1]: Stopping A high performance web server and a reverse proxy server...
Apr 05 18:33:40 m2-pedroamp systemd[1]: nginx.service: Succeeded.
Apr 05 18:33:40 m2-pedroamp systemd[1]: Stopped A high performance web server and a reverse proxy server.
```

Figura 25: Parada de servicio de nginx.

Finalmente vemos como cambia el estado m2-pedroamp en el panel de estadísticas de HAProxy.

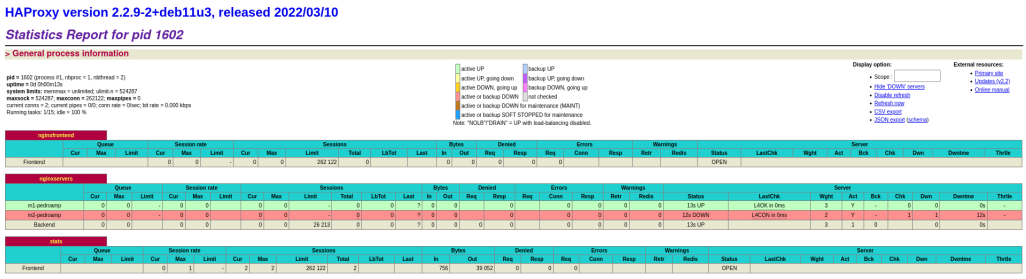

Figura 26: Verificación del servidor parado en HAProxy.

2.3.2. Ocultando versión de HAProxy

Si exponemos el panel de estadísticas de HAProxy debido a que suele ser un blanco de ataques, se recomienda para dificultar los ataques ocultar la versión que está actualmente en ejecución.
Debemos poner la directiva *stats hide-version*

```
frontend stats
        bind *:8404
        stats enable
        stats uri /stats
        stats refresh 10s
        stats auth  admin:admin
        stats hide-version
```

Figura 27: Ocultando la versión de HAProxy

Comprobación en la Web de estadísticas.

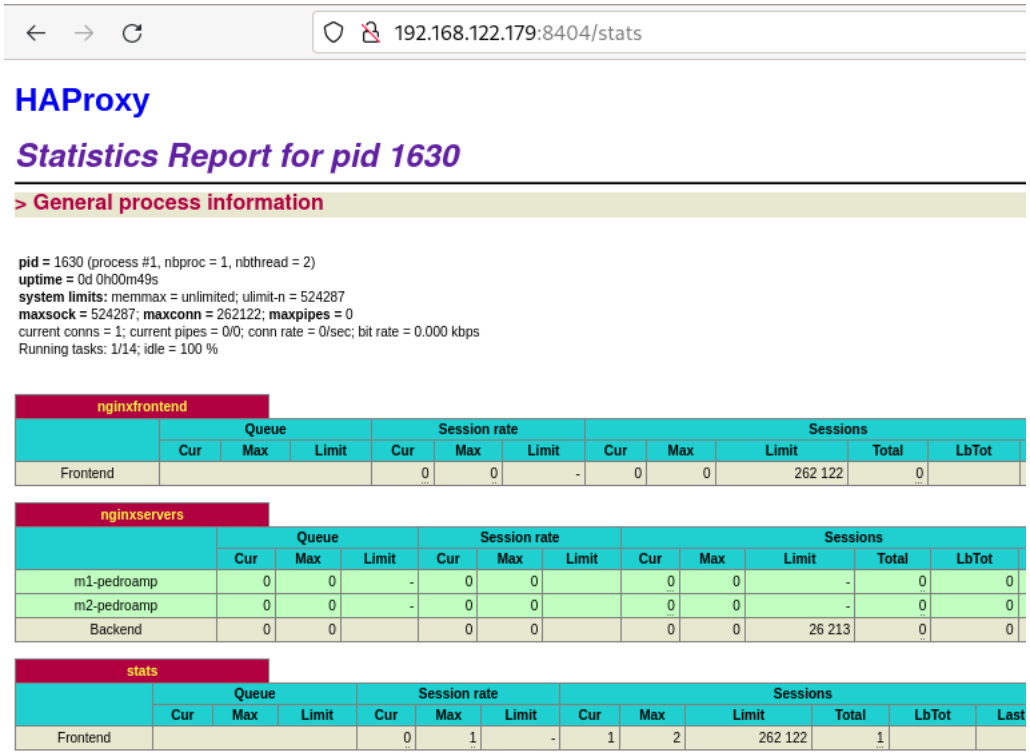

Figura 28: Ocultando la versión de HAProxy visualización en la web

3. GoBetween load-balancer

Para su instalación desde su página web nos lleva a un enlace de descarga de github. Que debemos descargar con wget.

```
Try "whatis --help" or "whatis --usage" for more information.
debian@m3-pedroamp:/tmp/gobetween$ sudo gobetween
2022/04/07 11:14:03 gobetween v0.8.0
Modern & minimalistic load balancer for the Cloud era

Usage:
  gobetween [flags]
  gobetween [command]

Available Commands:
  from-consul Start using config from Consul
  from-file   Start using config from file
  from-url    Start using config from URL
  help        Help about any command

Flags:
  -c, --config string         Path to configuration file
  -f, --format string         Configuration file format: "toml" or "json" (default "toml")
  -h, --help                  help for gobetween
  -p, --pidfile string        Write pid to specified file
  -e, --use-config-env-vars   Enable env variables interpretation in config file
  -v, --version               Print version information and quit

Use "gobetween [command] --help" for more information about a command.
debian@m3-pedroamp:/tmp/gobetween$ ls /etc/gobetween/
gobetween.toml
debian@m3-pedroamp:/tmp/gobetween$ sudo whereis gobetween
gobetween: /etc/gobetween /usr/local/sbin/gobetween
debian@m3-pedroamp:/tmp/gobetween$
```

Figura 29: Descarga del fichero de instalación.

Necesitamos crear una carpeta temporal en /tmp para poder manejar los residuos que se suele dejar con esta clase de descargas.

```
mkdir /tmp/gobetween
cd /tmp/gobetween
wget https://github.com/yyyar/gobetween/releases/download/0.8.0/
 gobetween_0.8.0_linux_amd64.tar.gz
tar xf gobetween_0.8.0_linux_amd64.tar.gz
sudo mkdir /etc/gobetween
sudo mv config/gobetween.toml /etc/gobetween/
sudo chmod +x gobetween
sudo mv gobetween /usr/local/sbin/
```

Para saber si has realizado los pasos correctamente debemos comprobar con los siguientes comandos que su instalación ha sido correcta.

```
// ejecutamos el comando para ver si responde
sudo gobetween
// vemos si el fichero de config se ha movido en su ruta
ls /etc/gobetween
// nos indica que ha sido configurado correctamente
sudo whereis gobetween
```

```
Try "whatis --help" or "whatis --usage" for more information.
debian@m3-pedroamp:/tmp/gobetween$ sudo gobetween
2022/04/07 11:14:03 gobetween v0.8.0
Modern & minimalistic load balancer for the Cloud era

Usage:
  gobetween [flags]
  gobetween [command]

Available Commands:
  from-consul Start using config from Consul
  from-file   Start using config from file
  from-url    Start using config from URL
  help        Help about any command

Flags:
  -c, --config string         Path to configuration file
  -f, --format string         Configuration file format: "toml" or "json" (default "toml")
  -h, --help                  help for gobetween
  -p, --pidfile string        Write pid to specified file
  -e, --use-config-env-vars   Enable env variables interpretation in config file
  -v, --version               Print version information and quit

Use "gobetween [command] --help" for more information about a command.
debian@m3-pedroamp:/tmp/gobetween$ ls /etc/gobetween/
gobetween.toml
debian@m3-pedroamp:/tmp/gobetween$ sudo whereis gobetween
gobetween: /etc/gobetween /usr/local/sbin/gobetween
debian@m3-pedroamp:/tmp/gobetween$
```

Figura 30: Instalación correcta de gobetween.

3.1. Creación de demonio para SystemD y controlar el servicio de gobetween

Gobetween no tiene como tal un control del servicio como nginx o HAProxy. Que permita verificar el estado, controlar los errores, reiniciar el servicio, etc...
Simplemente se nos dá un binario que lee con su opción -c un fichero de configuración localizado en la carpeta donde esté localizado el fichero .toml. Por lo que debemos crear un demonio personalizado que pueda controlar el estado del binario y sus estados de error o ok.
Para ello tenemos que ir a la ruta siguiente y crear un fichero nuevo llamado gobetween.service.

```
cd /etc/systemd/system
sudo nano gobetween.service
```

El fichero quedaría como sigue.

```
 GNU nano 5.4                  /etc/systemd/system/gobetween.service
[Unit]
Description=gobetween load balancer service
After=network.target auditd.service


[Service]
Type=simple
User=root
ExecStart=/usr/local/sbin/gobetween -c /etc/gobetween/gobetween.toml
ExecReload=/usr/local/sbin/gobetween -c /etc/gobetween/gobetween.toml
Restart=always

[Install]
WantedBy=multi-user.target
```

Figura 31: gobetween.service creado.

Ahora ya tenemos que recargar SystemD para que detecte el nuevo servicio. Si tenemos los servicios de HAProxy o de Nginx, también debemos desactivarlos.

```
sudo systemctl daemon-reload
sudo systemctl disable nginx.service haproxy.service
sudo systemctl stop nginx.service haproxy.service
sudo systemctl status nginx.service haproxy.service
```

```
debian@m3-pedroamp:/tmp/gobetween$ sudo systemctl daemon-reload
debian@m3-pedroamp:/tmp/gobetween$ sudo systemctl disable haproxy.service
Synchronizing state of haproxy.service with SysV service script with /lib/systemd/systemd-sysv-
install.
Executing: /lib/systemd/systemd-sysv-install disable haproxy
Removed /etc/systemd/system/multi-user.target.wants/haproxy.service.
debian@m3-pedroamp:/tmp/gobetween$ sudo systemctl stop haproxy.service
debian@m3-pedroamp:/tmp/gobetween$ sudo systemctl status nginx.service haproxy.service
● nginx.service - A high performance web server and a reverse proxy server
     Loaded: loaded (/lib/systemd/system/nginx.service; disabled; vendor preset: enabled)
     Active: inactive (dead)
       Docs: man:nginx(8)

● haproxy.service - HAProxy Load Balancer
     Loaded: loaded (/lib/systemd/system/haproxy.service; disabled; vendor preset: enabled)
     Active: inactive (dead)
       Docs: man:haproxy(1)
             file:/usr/share/doc/haproxy/configuration.txt.gz

Apr 07 10:40:53 m3-pedroamp systemd[1]: Started HAProxy Load Balancer.
Apr 07 11:49:01 m3-pedroamp haproxy[569]: [WARNING] 096/114901 (569) : Exiting Master process.
Apr 07 11:49:01 m3-pedroamp haproxy[569]: [NOTICE] 096/114901 (569) : haproxy version is 2.2.9
Apr 07 11:49:01 m3-pedroamp haproxy[569]: [NOTICE] 096/114901 (569) : path to executable is /u
Apr 07 11:49:01 m3-pedroamp haproxy[569]: [ALERT] 096/114901 (569) : Current worker #1 (572) e
Apr 07 11:49:01 m3-pedroamp haproxy[569]: [WARNING] 096/114901 (569) : All workers exited. Exi
Apr 07 11:49:01 m3-pedroamp systemd[1]: Stopping HAProxy Load Balancer...
Apr 07 11:49:01 m3-pedroamp systemd[1]: haproxy.service: Succeeded.
Apr 07 11:49:01 m3-pedroamp systemd[1]: Stopped HAProxy Load Balancer.
Apr 07 11:49:01 m3-pedroamp systemd[1]: haproxy.service: Consumed 1.754s CPU time.
debian@m3-pedroamp:/tmp/gobetween$
```

Figura 32: Cerrando los viejos servicios.

Finalmente tenemos que activar el servicio y arrancarlo.

```
debian@m3-pedroamp:/tmp/gobetween$ sudo systemctl enable gobetween.service
Created symlink /etc/systemd/system/multi-user.target.wants/gobetween.service → /etc/systemd/sy
stem/gobetween.service.
debian@m3-pedroamp:/tmp/gobetween$ sudo systemctl start gobetween.service
debian@m3-pedroamp:/tmp/gobetween$ sudo systemctl status gobetween.service
● gobetween.service - gobetween load balancer service
     Loaded: loaded (/etc/systemd/system/gobetween.service; enabled; vendor preset: enabled)
     Active: active (running) since Thu 2022-04-07 11:53:16 UTC; 2s ago
   Main PID: 1536 (gobetween)
      Tasks: 8 (limit: 4678)
     Memory: 13.8M
        CPU: 51ms
     CGroup: /system.slice/gobetween.service
             └─1536 /usr/local/sbin/gobetween -c /etc/gobetween/gobetween.toml

Apr 07 11:53:16 m3-pedroamp gobetween[1536]: 2022/04/07 11:53:16 gobetween v0.8.0
Apr 07 11:53:16 m3-pedroamp gobetween[1536]: 2022-04-07 11:53:16 [INFO ] (manager): Initializi>
Apr 07 11:53:16 m3-pedroamp gobetween[1536]: 2022-04-07 11:53:16 [INFO ] (server): Creating 's>
Apr 07 11:53:16 m3-pedroamp gobetween[1536]: 2022-04-07 11:53:16 [INFO ] (scheduler): Starting>
Apr 07 11:53:16 m3-pedroamp gobetween[1536]: 2022-04-07 11:53:16 [INFO ] (udp/server): Creatin>
Apr 07 11:53:16 m3-pedroamp gobetween[1536]: 2022-04-07 11:53:16 [INFO ] (scheduler): Starting>
Apr 07 11:53:16 m3-pedroamp gobetween[1536]: 2022-04-07 11:53:16 [INFO ] (manager): Initialized
Apr 07 11:53:16 m3-pedroamp gobetween[1536]: 2022-04-07 11:53:16 [INFO ] (metrics): Metrics di>
Apr 07 11:53:16 m3-pedroamp gobetween[1536]: 2022-04-07 11:53:16 [INFO ] (api): Starting up API
Apr 07 11:53:16 m3-pedroamp gobetween[1536]: 2022-04-07 11:53:16 [INFO ] (api): Starting HTTP >
debian@m3-pedroamp:/tmp/gobetween$
```

Figura 33: Arrancando el servicio gobetween.

Finalmente tenemos una configuración de un servicio que nos permite controlar lo que se hace además de que en el caso de retocar los ficheros de configuración podemos reiniciarlo para que recargue su configuración nueva.
En la información consultada SystemD tiene la particularidad que al hacer systemctl stop gobetween.service es equivalente a hacer un kill SIGNTERM PID, pero sin tener que trazarlo el administrador de sistemas al mandar el binario a background con el & y registrando un PID que se puede perder. Hay que fijarse en el Main PID que nos ofrece el comando systemctl status gobetween.service.

```
debian@m3-pedroamp:/tmp/gobetween$ sudo systemctl status gobetween.service
● gobetween.service - gobetween load balancer service
     Loaded: loaded (/etc/systemd/system/gobetween.service; enabled; vendor preset: enabled)
     Active: active (running) since Thu 2022-04-07 11:53:16 UTC; 2s ago
   Main PID: 1536 (gobetween)
      Tasks: 8 (limit: 4678)
     Memory: 13.8M
        CPU: 51ms
     CGroup: /system.slice/gobetween.service
             └─1536 /usr/local/sbin/gobetween -c /etc/gobetween/gobetween.toml

Apr 07 11:53:16 m3-pedroamp gobetween[1536]: 2022/04/07 11:53:16 gobetween v0.8.0
Apr 07 11:53:16 m3-pedroamp gobetween[1536]: 2022-04-07 11:53:16 [INFO ] (manager): Initializi>
Apr 07 11:53:16 m3-pedroamp gobetween[1536]: 2022-04-07 11:53:16 [INFO ] (server): Creating 's>
Apr 07 11:53:16 m3-pedroamp gobetween[1536]: 2022-04-07 11:53:16 [INFO ] (scheduler): Starting>
Apr 07 11:53:16 m3-pedroamp gobetween[1536]: 2022-04-07 11:53:16 [INFO ] (udp/server): Creatin>
Apr 07 11:53:16 m3-pedroamp gobetween[1536]: 2022-04-07 11:53:16 [INFO ] (scheduler): Starting>
Apr 07 11:53:16 m3-pedroamp gobetween[1536]: 2022-04-07 11:53:16 [INFO ] (manager): Initialized
Apr 07 11:53:16 m3-pedroamp gobetween[1536]: 2022-04-07 11:53:16 [INFO ] (metrics): Metrics di>
Apr 07 11:53:16 m3-pedroamp gobetween[1536]: 2022-04-07 11:53:16 [INFO ] (api): Starting up API
Apr 07 11:53:16 m3-pedroamp gobetween[1536]: 2022-04-07 11:53:16 [INFO ] (api): Starting HTTP >
debian@m3-pedroamp:/tmp/gobetween$ sudo systemctl stop gobetween.service
debian@m3-pedroamp:/tmp/gobetween$ sudo systemctl status gobetween.service
● gobetween.service - gobetween load balancer service
     Loaded: loaded (/etc/systemd/system/gobetween.service; enabled; vendor preset: enabled)
     Active: inactive (dead) since Thu 2022-04-07 12:00:43 UTC; 1s ago
    Process: 1536 ExecStart=/usr/local/sbin/gobetween -c /etc/gobetween/gobetween.toml (code=k>
   Main PID: 1536 (code=killed, signal=TERM)
        CPU: 236ms

Apr 07 11:53:16 m3-pedroamp gobetween[1536]: 2022-04-07 11:53:16 [INFO ] (scheduler): Starting>
Apr 07 11:53:16 m3-pedroamp gobetween[1536]: 2022-04-07 11:53:16 [INFO ] (udp/server): Creatin>
Apr 07 11:53:16 m3-pedroamp gobetween[1536]: 2022-04-07 11:53:16 [INFO ] (scheduler): Starting>
Apr 07 11:53:16 m3-pedroamp gobetween[1536]: 2022-04-07 11:53:16 [INFO ] (manager): Initialized
Apr 07 11:53:16 m3-pedroamp gobetween[1536]: 2022-04-07 11:53:16 [INFO ] (metrics): Metrics di>
Apr 07 11:53:16 m3-pedroamp gobetween[1536]: 2022-04-07 11:53:16 [INFO ] (api): Starting up API
Apr 07 11:53:16 m3-pedroamp gobetween[1536]: 2022-04-07 11:53:16 [INFO ] (api): Starting HTTP >
Apr 07 12:00:43 m3-pedroamp systemd[1]: Stopping gobetween load balancer service...
Apr 07 12:00:43 m3-pedroamp systemd[1]: gobetween.service: Succeeded.
Apr 07 12:00:43 m3-pedroamp systemd[1]: Stopped gobetween load balancer service.
debian@m3-pedroamp:/tmp/gobetween$
```

Figura 34: Parada del servicio gobetween.

3.2. Configuración básica de gobetween

No se indica en el manual oficial, pero deberíamos cambiar el /path/to/logging a /var/-log/gobetween.log para que se cree un fichero correcto de log. En la parte de [logging]. Además debemos poner una contraseña a un panel de información REST que nos da acceso a gobetween. Activando *[api.basic_ auth]*. Debemos indicar la ip del servidor.

```
#
# Logging configuration
#
[logging]
level = "info"    # "debug" | "info" | "warn" | "error"
output = "stdout" # "stdout" | "stderr" | "/var/log/gobetween.log"
format = "text"   # (optional) "text" | "json"

#
# Pprof profiler configuration
#
[profiler]
enabled = false # false | true
bind = ":6060"  # "host:port"

#
# REST API server configuration
#
[api]
enabled = true  # true | false
bind = "192.168.122.179:8888"  # "host:port"
cors = false    # cross-origin resource sharing

  [api.basic_auth]   # (optional) Enable HTTP Basic Auth
  login = "admin"    # HTTP Auth Login
  password = "admin"  # HTTP Auth Password

#  [api.tls]                        # (optional) Enable HTTPS
#  cert_path = "/path/to/cert.pem"  # Path to certificate
#  key_path = "/path/to/key.pem"    # Path to key
```

Figura 35: Configuración del log y autenticación básica.

Luego pasamos a configurar la sección de *[defaults]* que son los limitadores globales de conexión de nº de conexiones, timeouts de cliente y servidores. Yo los he configurado como sigue, pero adicionalmente el 0 indica que no hay limitación.

```
#
# Default values for server configuration, may be overriden in [servers] sections.
# All "duration" fields (for examole, postfixed with '_timeout') have the following format:
# <int><duration> where duration can be one of 'ms', 's', 'm', 'h'.
# Examples: "5s", "1m", "500ms", etc. "0" value means no limit
#
[defaults]
max_connections = 0              # Maximum simultaneous connections to the server
client_idle_timeout = "1m"        # Client inactivity duration before forced connection drop
backend_idle_timeout = "1m"       # Backend inactivity duration before forced connection drop
backend_connection_timeout = "1m" # Backend connection timeout (ignored in udp)
```

Figura 36: Sección [defaults].

Finalmente pasamos a la sección *[servers]*, en la cual debemos comentar la parte de *[servers.udpsample]* porque el protocolo con el que trabajamos es TCP.
Además de dejar configurada la zona de servidores como *[server.mswap-pedroamp]*, indicar que servidores son en una lista estática y la IP o dominio enlazada (bind) al propio servidor que hace de balanceador de carga. Así como el puerto TCP en el que trabaja go-between.
Se que he puesto los hostname en el fichero, pero me he visto obligado a cambiarlos por sus direcciones IP privadas porque cuando lleva muchas peticiones HTTP seguidas el programa llama al resolver DNS para que resuelva dichos hostname que por supuesto no existen. Quedándose bloqueado.

```
#
[servers]

# ---------- tcp example ----------- #

[servers.mswap-pedroamp]
protocol = "tcp"
bind = "192.168.122.179:80"

  [servers.mswap-pedroamp.discovery]
  kind = "static"
  static_list = [
      "m1-pedroamp:80",
      "m2-pedroamp:80",
     # "localhost:8002 weight=100 priority=0"
  ]

# ---------- udp example ----------- #

#[servers.udpsample]
#bind = "localhost:4000"
#protocol = "udp"

#  [servers.udpsample.udp]
#  max_responses = 1

#  [servers.udpsample.discovery]
#  kind = "static"
#  static_list = [
#      "8.8.8.8:53",
#      "8.8.4.4:53",
#      "91.239.100.100:53"
#  ]

#
# -------------------- example ---------------
```

Figura 37: Sección [servers].

Pruebas de funcionamiento:

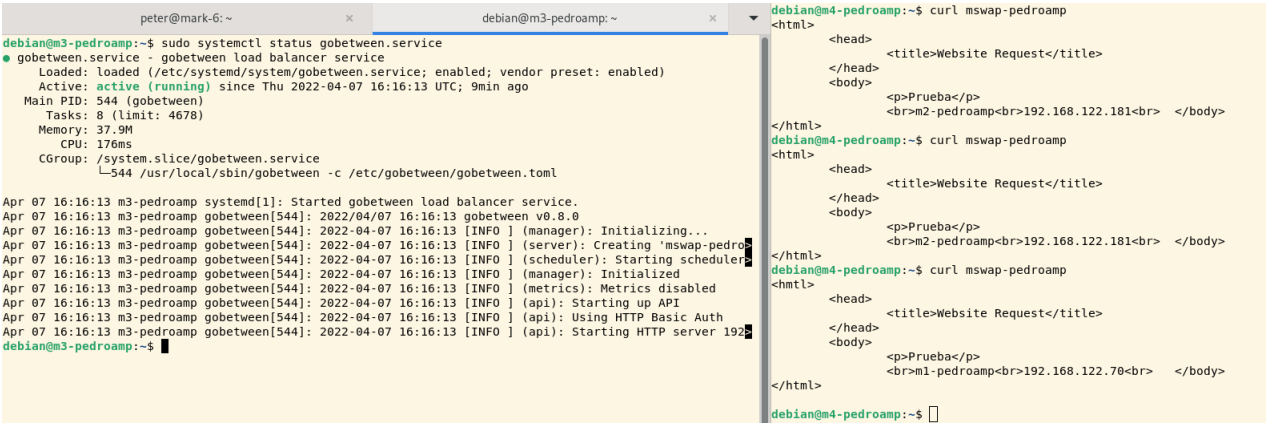

Figura 38: Pruebas del funcionamiento.

3.3. Configuración de round robin de gobetween

Para poder configurar el balanceador de gobetween en el algoritmo más básico que es el round robin, debemos configurar en la sección *[server.mswap-pedroamp]*. *balance=roundrobin"*

```
# ---------- tcp example ----------- #

[servers.mswap-pedroamp]
protocol = "tcp"
bind = "192.168.122.179:80"
balance="roundrobin"

  [servers.mswap-pedroamp.discovery]
  kind = "static"
  static_list = [
      "192.168.122.70:80",
      "192.168.122.181:80",
     # "localhost:8002 weight=100 priority=0"
  ]

# ---------- udp example ----------- #
```

Figura 39: Gobetween en roundrobin.

Reiniciamos el servicio para que lea el fichero de configuración.

```
sudo systemctl restart gobetween.service
```

Al realizar las pruebas podemos ver que efectivamente existe un comportamiento diferente al que viene por defecto ya demostrado anteriormente que en Round Robin cuando el primer servidor recibe la petición, pasa al siguiente esté como esté. En vez de recibir 2 peticiones seguidas el mismo.

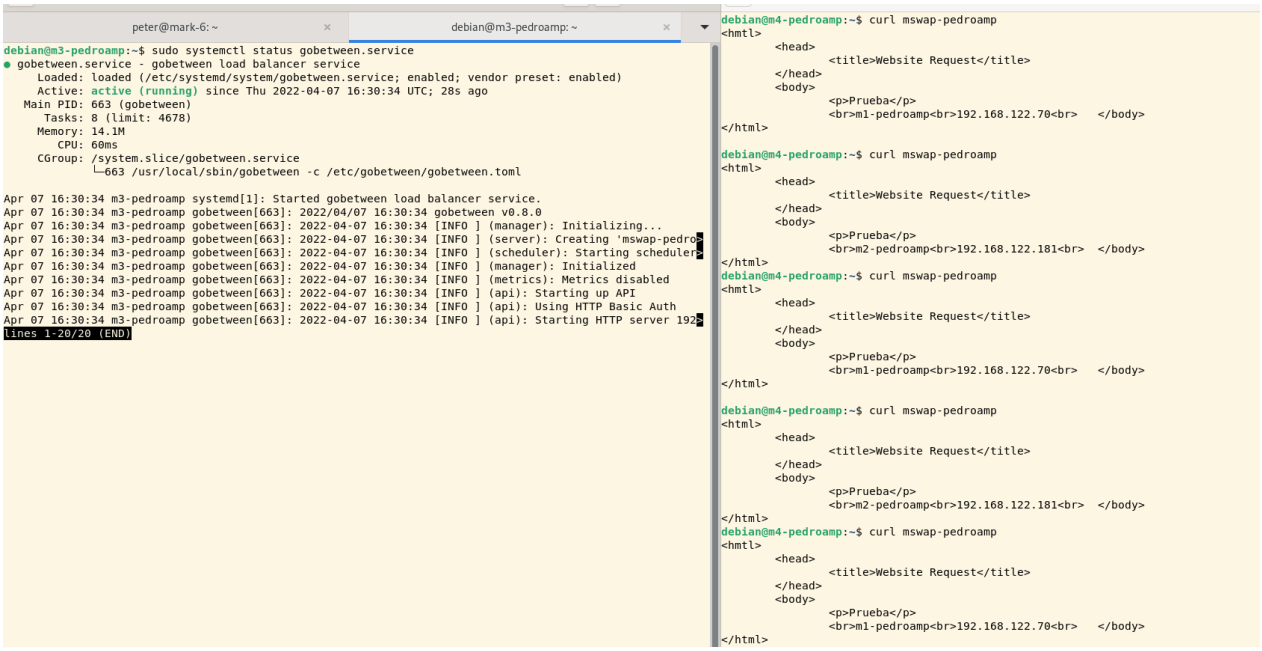

Figura 40: Gobetween en roundrobin, pruebas.

3.4. Configuración de weighted o ponderación

Para realizar la configuración de servidores ponderados debemos de poner lo siguiente, *balance="weight"*. Luego en cada servidor la variable *weight=N*.

```
# ---------- tcp example ----------- #

[servers.mswap-pedroamp]
protocol = "tcp"
bind = "192.168.122.179:80"
#balance="roundrobin"
balance="weight"

  [servers.mswap-pedroamp.discovery]
  kind = "static"
  static_list = [
        "192.168.122.70:80 weight=3",
        "192.168.122.181:80 weight=2",
     # "localhost:8002 weight=100 priority=0"
  ]

  [servers.mswap-pedroamp.healthcheck]
  kind="ping"
  interval = "2s"
  ping_timeout_duration = "500ms"
  fails = 1
  passes = 1

# ---------- udp example ----------- #
```

Figura 41: Gobetween en weighted, configuración.

Después de configurar el servicio reiniciamos el servicio.

Realizamos las pruebas con cURL para ver si de cada 5 peticiones llegan como hemos indicado anteriormente.

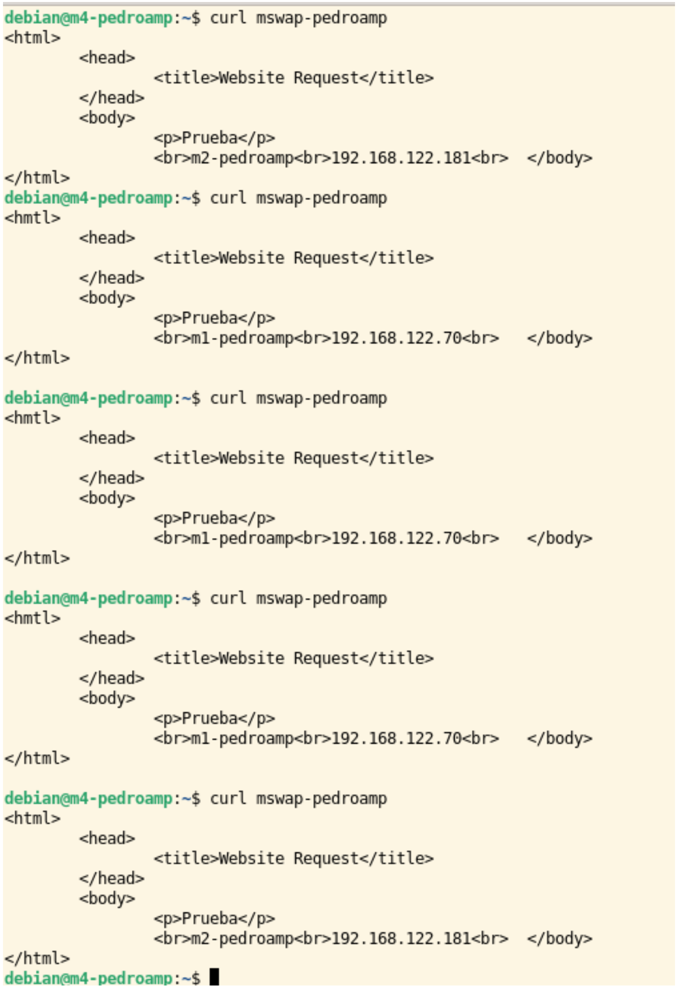

Figura 42: Gobetween en weighted, pruebas.

3.5. Comprobación del estado de los servidores healthcheck

Para comprobar el estado de los servidores tenemos que configurarlo de la siguiente manera:

```
# ---------- tcp example ----------- #

[servers.mswap-pedroamp]
protocol = "tcp"
bind = "192.168.122.179:80"
balance="roundrobin"

  [servers.mswap-pedroamp.discovery]
  kind = "static"
  static_list = [
      "192.168.122.70:80",
      "192.168.122.181:80",
     # "localhost:8002 weight=100 priority=0"
  ]

  [servers.mswap-pedroamp.healthcheck]
  kind="ping"
  interval = "2s"
  ping_timeout_duration = "500ms"
  fails = 1
  passes = 1
```

Figura 43: Gobetween healthcheck.

Donde el tipo de prueba es con ping. Se realiza en un intérvalo cada 2 segundos y se define un timeout para indicar que no funciona correctamente.
Además de una variable de fails y passes de valor 1 para indicar el cambio de estado.
Recordemos que debemos reiniciar el servicio para poder aplicar los cambios.
Luego para ver el estado tenemos la API Rest que hemos indicado anteriormente que debemos acceder a través de *IPorHostnameLoadbalancer:PORTRestAPI*. Que en mi caso es:
http://192.168.122.179:8888. Con la URI /servers/mswap-pedroamp/stats, donde el balanceador de carga genera el recurso que indica el estado de los servidores. En mi caso por la configuración indicada anteriormente me pide una autenticación.
La información mostrada es como la que sigue:

Figura 44: Gobetween healthcheck viendo el estado de los servidores con la API Rest.

La API Rest es útil en el caso de querer utilizarla con sistemas de monitorización donde podemos leer los datos y registrar en qué momento se puede caer un servidor del backend. Así como si tienes otro cluster donde el balanceador también tiene reenvio de peticiones lo podemos filtrar por la URI.

3.6. Límites de ficheros abiertos de gobetween

Al realizar el benchmark en una configuración muy bruta, de 10000 peticiones con 1000 clientes en paralelo, generando un total de 10.000.000 de ficheros donde he llegado al límite físico donde gobetween no puede abrir más ficheros de tipo socket, rompiendo el benchmark. Cosa a considerar si queremos evitar ataques DoS, porque se queda pillado con todos esos archivos .socket abiertos.
Para averiguar eso se puede ver en el estado del servicio donde indica en su log que ha tocado el límite de ficheros abiertos, parando de recibir peticiones.
Luego a continuación en el fichero del proceso limits localizado en: /proc/PID_Gobetween/, podemos ver que efectivamente no puede superar los 524288 de ficheros abiertos en Hard Limit.

```
debian@m3-pedroamp:~$ sudo systemctl status gobetween.service
● gobetween.service - gobetween load balancer service
     Loaded: loaded (/etc/systemd/system/gobetween.service; enabled; vendor preset: enabled)
     Active: active (running) since Thu 2022-04-07 16:45:31 UTC; 15min ago
   Main PID: 904 (gobetween)
      Tasks: 8 (limit: 4678)
     Memory: 34.3M
        CPU: 2.052s
     CGroup: /system.slice/gobetween.service
             └─904 /usr/local/sbin/gobetween -c /etc/gobetween/gobetween.toml

Apr 07 16:55:38 m3-pedroamp gobetween[904]: 2022-04-07 16:55:38 [ERROR] (server.handle [192.168.122.179:80]): dial tcp 192.168.122.70:80: socket: too many open files
Apr 07 16:55:38 m3-pedroamp gobetween[904]: 2022-04-07 16:55:38 [ERROR] (server.handle [192.168.122.179:80]): dial tcp 192.168.122.181:80: socket: too many open files
Apr 07 16:55:38 m3-pedroamp gobetween[904]: 2022-04-07 16:55:38 [ERROR] (server.handle [192.168.122.179:80]): dial tcp 192.168.122.70:80: socket: too many open files
Apr 07 16:55:38 m3-pedroamp gobetween[904]: 2022-04-07 16:55:38 [ERROR] (server.handle [192.168.122.179:80]): dial tcp 192.168.122.181:80: socket: too many open files
Apr 07 16:55:38 m3-pedroamp gobetween[904]: 2022-04-07 16:55:38 [ERROR] (server.handle [192.168.122.179:80]): dial tcp 192.168.122.70:80: socket: too many open files
Apr 07 16:55:38 m3-pedroamp gobetween[904]: 2022-04-07 16:55:38 [ERROR] (server.handle [192.168.122.179:80]): dial tcp 192.168.122.181:80: socket: too many open files
Apr 07 16:55:38 m3-pedroamp gobetween[904]: 2022-04-07 16:55:38 [ERROR] (server.handle [192.168.122.179:80]): dial tcp 192.168.122.70:80: socket: too many open files
Apr 07 16:55:38 m3-pedroamp gobetween[904]: 2022-04-07 16:55:38 [ERROR] (server.handle [192.168.122.179:80]): dial tcp 192.168.122.181:80: socket: too many open files
Apr 07 16:55:38 m3-pedroamp gobetween[904]: 2022-04-07 16:55:38 [ERROR] (server.handle [192.168.122.179:80]): dial tcp 192.168.122.70:80: socket: too many open files
Apr 07 16:55:38 m3-pedroamp gobetween[904]: 2022-04-07 16:55:38 [ERROR] (server.handle [192.168.122.179:80]): dial tcp 192.168.122.181:80: socket: too many open files
debian@m3-pedroamp:~$ sudo cat /proc/904/limits
Limit                     Soft Limit           Hard Limit           Units
Max cpu time              unlimited            unlimited            seconds
Max file size             unlimited            unlimited            bytes
Max data size             unlimited            unlimited            bytes
Max stack size            8388608              unlimited            bytes
Max core file size        0                    unlimited            bytes
Max resident set          unlimited            unlimited            bytes
Max processes             15595                15595                processes
Max open files            1024                 524288               files
Max locked memory         65536                65536                bytes
Max address space         unlimited            unlimited            bytes
Max file locks            unlimited            unlimited            locks
Max pending signals       15595                15595                signals
Max msgqueue size         819200               819200               bytes
Max nice priority         0                    0
Max realtime priority     0                    0
Max realtime timeout      unlimited            unlimited            us
debian@m3-pedroamp:~$
```

Figura 45: Gobetween no puede abrir más conexiones.

4. Instalación de Zevenet

En su repositorio oficial, incluido en la bibliografía. Se indica que para instalarlo en una versión de Debian GNU/Linux hay que incluir los repositorios que ellos tienen preparados. La instalación completa necesita casi 1.5 GiB de almacenamiento.
Dichos repositorios son los siguientes:

```
echo "deb http://repo.zevenet.com/ce/v5 buster main"
| sudo tee /etc/apt/sources.list.d/zevenet.list
wget -qO - http://repo.zevenet.com/zevenet.com.gpg.key
        | sudo apt-key add -
```

Necesita de los paquetes de gpg para poder añadir la clave

```
        sudo apt-get install gnupg
```

```
debian@m3-pedroamp:~$ echo "deb http://repo.zevenet.com/ce/v5 buster main" | sudo tee /etc/apt/sources.li
st.d/zevenet.list
deb http://repo.zevenet.com/ce/v5 buster main
debian@m3-pedroamp:~$ wget -qO - http://repo.zevenet.com/zevenet.com.gpg.key | sudo apt-key add -
Warning: apt-key is deprecated. Manage keyring files in trusted.gpg.d instead (see apt-key(8)).
OK
debian@m3-pedroamp:~$
```

Figura 46: Repositorios añadidos.

Ahora para proceder con la instalación ejecutamos el siguiente comando:

```
        sudo apt-get update && sudo apt-get install zevenet
        sudo apt-get install zevenet-gui-ce
```

```
Completing the Zevenet installation...
Reloading ssh configuration (via systemctl): ssh.service.
Stopping snmpd (via systemctl): snmpd.service.
vserver!1!ssl_certificate_file = /usr/local/zevenet/config/zencert.pem
Migrating certificate of http server
migrated vserver!1!ssl_certificate_file = /usr/local/zevenet/config/certificates/zencert.pem
migrated vserver!1!ssl_certificate_key_file = /usr/local/zevenet/config/certificates/zencert-c.key
Zevenet will be unavailable while the service is restarting.
sh: 1: /usr/local/sbin/nft: not found
sh: 1: /usr/local/sbin/nft: not found
Reloading configuration files for periodic command scheduler: cron.
Zevenet Load Balancer installation completed.
Setting up libauthen-simple-pam-perl (0.2-4.1) ...
Processing triggers for man-db (2.9.4-2) ...
Processing triggers for libc-bin (2.31-13+deb11u3) ...
debian@m3-pedroamp:~$ sudo apt-get update && sudo apt-get install zevenet
```

Figura 47: Zevenet instalado.

Finalmente tenemos que activar su servicio y configurarlo.

```
debian@m3-pedroamp:~$ sudo systemctl status zevenet.service
● zevenet.service - LSB: zevenet
     Loaded: loaded (/etc/init.d/zevenet; generated)
     Active: inactive (dead)
       Docs: man:systemd-sysv-generator(8)
debian@m3-pedroamp:~$ sudo systemctl enable zevenet.service
zevenet.service is not a native service, redirecting to systemd-sysv-install.
Executing: /lib/systemd/systemd-sysv-install enable zevenet
sydebian@m3-pedroamp:~$ sudo systemctl start zevenet.service
debian@m3-pedroamp:~$ sudo systemctl status zevenet.service
● zevenet.service - LSB: zevenet
     Loaded: loaded (/etc/init.d/zevenet; generated)
     Active: active (exited) since Mon 2022-04-18 10:14:55 UTC; 7s ago
       Docs: man:systemd-sysv-generator(8)
    Process: 13042 ExecStart=/etc/init.d/zevenet start (code=exited, status=0/SUCCESS)
        CPU: 432ms

Apr 18 10:14:54 m3-pedroamp zevenet[13044]: (ERROR) system :: last command failed!
Apr 18 10:14:54 m3-pedroamp zevenet[13044]: (INFO) zevenet: Modules loaded
Apr 18 10:14:54 m3-pedroamp zevenet[13044]: (INFO) zevenet: Setting up NIC interfaces...
Apr 18 10:14:55 m3-pedroamp zevenet[13044]: (INFO) zevenet: Setting up VLAN interfaces...
Apr 18 10:14:55 m3-pedroamp zevenet[13044]: (INFO) zevenet: Setting up virtual interfaces...
Apr 18 10:14:55 m3-pedroamp zevenet[13044]: (INFO) zevenet: Setting up farms...
Apr 18 10:14:55 m3-pedroamp zevenet[13044]: (INFO) Running /usr/local/zevenet/config/zlb-start ...
Apr 18 10:14:55 m3-pedroamp zevenet[13044]: (INFO) End of /usr/local/zevenet/config/zlb-start
Apr 18 10:14:55 m3-pedroamp zevenet[13044]: (INFO) Start process finished.
Apr 18 10:14:55 m3-pedroamp systemd[1]: Started LSB: zevenet.
```

Figura 48: Zevenet con el servicio activado.

4.1. Modos de configuración del balanceador Zevenet

El balanceador Zevenet funciona como LSLB o DSLB o GSLB. Es decir:

- LSLB - Significa Local Service Load Balancing.
- DSLB - Significa Datalink Service Load Balancing.
- GSLB - Significa Global Service Load Balancing. - Solo versión Enterprise de Pago

El modo local funciona como balanceador en una granja web local donde se agrupan tanto servidores webs como los balanceadores en el mismo lugar. Por lo tanto el DNS apunta a la dirección/es IP/s pública/s de la granja que sale por esos puntos. El balanceador está en la frontera (edge), donde la dirección IP pública recae sobre el, siendo el intermediario entre la granja y el cliente.
El modo Datalink, funciona de manera que el balanceador es un router que recibe el tráfico de la red interna (NAT) hacia Internet o viceversa. Balancea la carga entre todos los routers WAN que existe en el Datacenter o localización. Esto es importante para el caso en el que un router WAN se caiga, o se sobrecargue de tráfico el balanceador-router lo pase a otro en mejor estado o con menor peso que reciba el tráfico sobrante que el router con mayor peso no puede soportar.
El modo global funciona de manera que si tenemos los servidores en datacenters diferentes. El balanceador a*punta a ellos cuando el cliente haga una consulta DNS a la Zona del Dominio de la que es autoridad de la granja. El balanceador comprueba el estado de los servidores de la zona de autoridad DNS, obtiene uno que esté en el mejor de los estados (Activo, su turno en RR, etc..). Finalmente resuelve el nombre retornando la dirección IP del servidor al cliente. Para finalmente el cliente realiza la petición HTTP hacia el servidor directo, con su dirección pública. En el modo global no quiere decir que haya un servidor por IP pública, se puede combinar el modo local que reparte las peticiones entre los servidores del propio datacenter y el global que lo reparte entre los datacenters globales.*
Nos centramos en el LSLB. El LSLB, tiene el funcionamiento siguiente:

- HTTP profile: Este perfil funciona como un proxy inverso, igual que en NGINX. Redireccionando peticiones HTTP y HTTP. Se añaden cabeceras especiales para redireccionar en el backend (esto es los servidores de la granja).
- L4XNAT profile: Este perfil funciona como un router balanceador de carga HTTP, con capacidad de manejar todo el tráfico indicado en el manual. Funciona con Iptables.

Nos centramos a su vez en el HTTP profile que es el objetivo de la práctica. Para poder configurarlo tenemos que acceder a su sitio web en la dirección IP o nombre. el usuario es el usuario root, junto con su contraseña establecida en PAM.

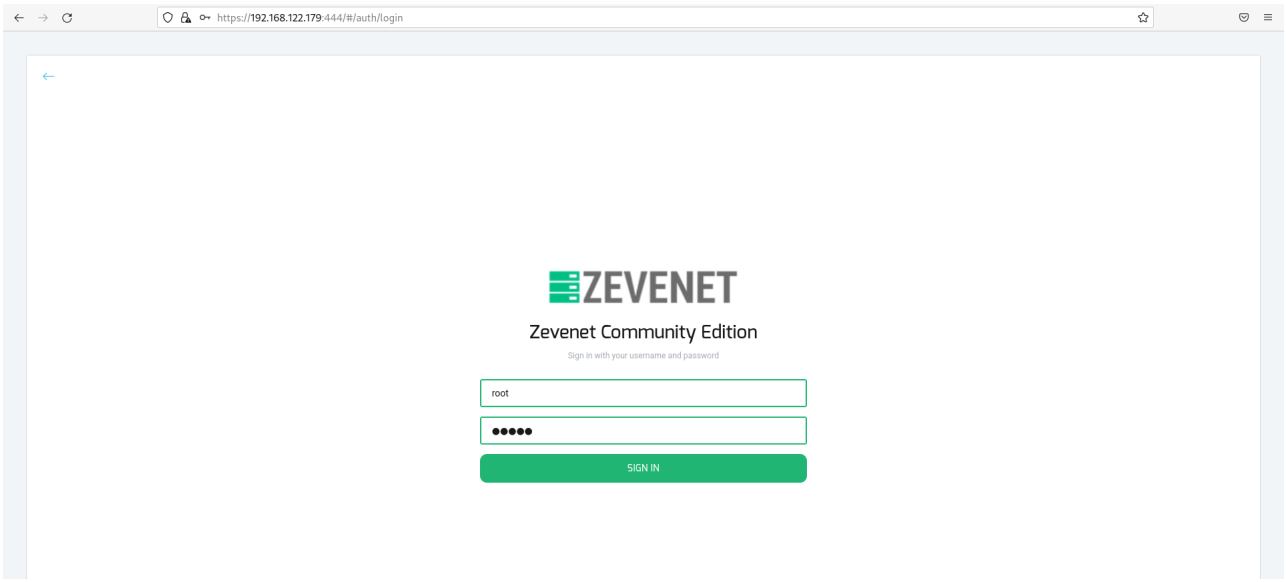

Figura 49: Acceso web.

Finalmente podemos visualizar el panel de control.

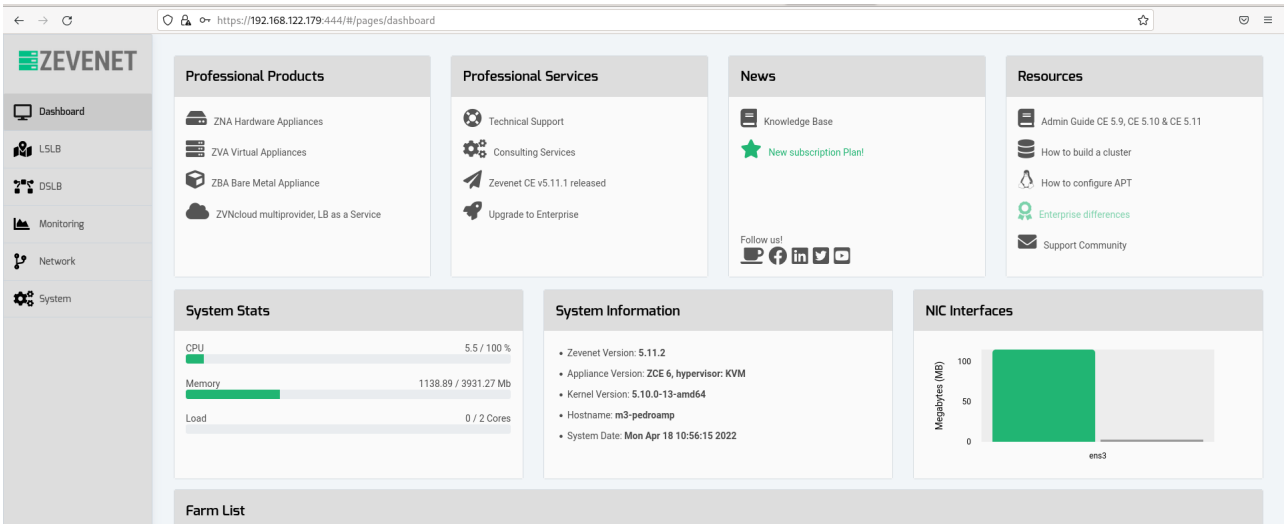

Figura 50: Dashboard.

4.2. Configurando LSLB

Para configurar el modo LSLB necesitamos una configuración previa que es la dirección IPv4 que tiene en su interfaz física. Para ello vamos a Network - NIC.

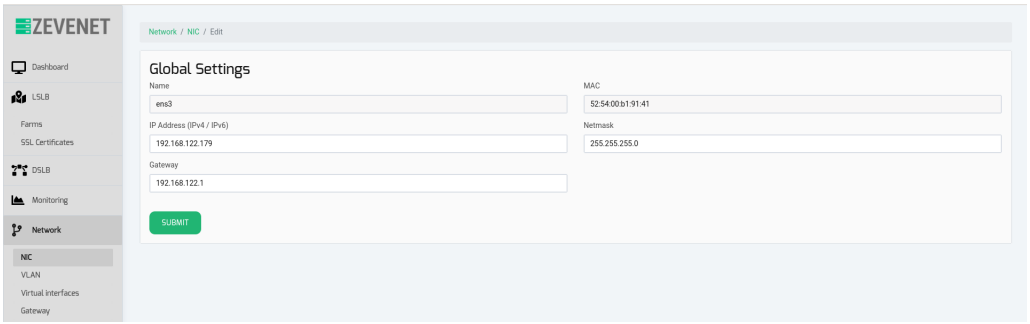

Figura 51: Configurando IPv4 en Zevenet.

Una vez indicado a Zevenet cuál es su dirección IP, podemos ir a crear la Granja Web sobre la cual nos va a balancear la carga. Para ello vamos a LSLB - Farms - Create LSLB Farm

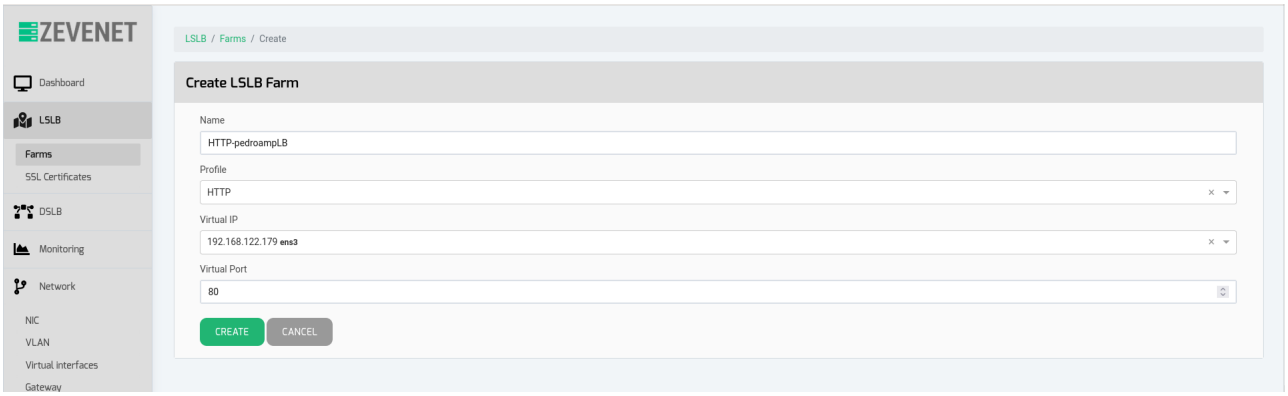

Figura 52: Creando la granja de LSLB.

Una vez creada nos saldrá en el panel. Que aparece con un Status en color negro, eso indica que está en estado crítico, que según la documentación está activo pero no existe el backend disponible. Para crearlo tenemos que ir a un icono lapiz que nos permita editar la configuración de backend.

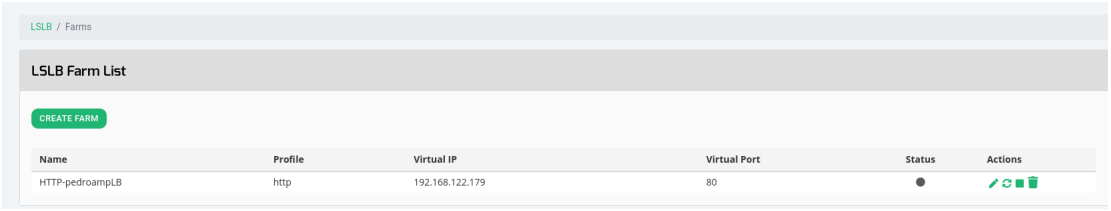

Figura 53: Creada la granja de LSLB.

Ahora tenemos que ir al final de la configuración y crear un servicio. Que lo llamaré igual que el VirtualHost al que queramos redirigir las peticiones.
Los pasos creados son:

- Creamos el nombre del VirtualHost sobre el que responde el balanceador en mi caso mswap-pedroamp. Recordemos que el VirtualHost sirve para cuando bajo se alojen varios servicios HTTP en la misma dirección IP pública, el VirtualHost se lee en la cabecera HTTP de Host o en TLS en SNI se incluye el Host al que se trata de conectar. Permitiendo la ejecución de varios servicios HTTP de distinto uso bajo la misma IP y Puerto TCP.

- Persistence: Lo pongo como IP client Address, como el IP_hash de las otras configuraciones ya explicadas. Existen más modos de persistencia como las cookies de sesión. Eso implica que podemos mandar una cookie que puede caducar, alojada en el navegador del cliente para que en el caso de deconexión pueda restaurar el carro de compra y el servidor en el que estaba haciendo la transacción. Liberando de la carga de los servidores.

- Health Checks: Permite mas opciones aparte del ICMP (echo PING), permite además el check por HTTP o HTTPs. El check http es que envía peticiones HTTP al backend y comprueba que obtiene una respuesta 200.

- En cada backend se indica un weight siendo el más alto el más prioritario, donde lo configuramos como en Nginx y los demas balanceadores. Si lo ponemos igual es RR.

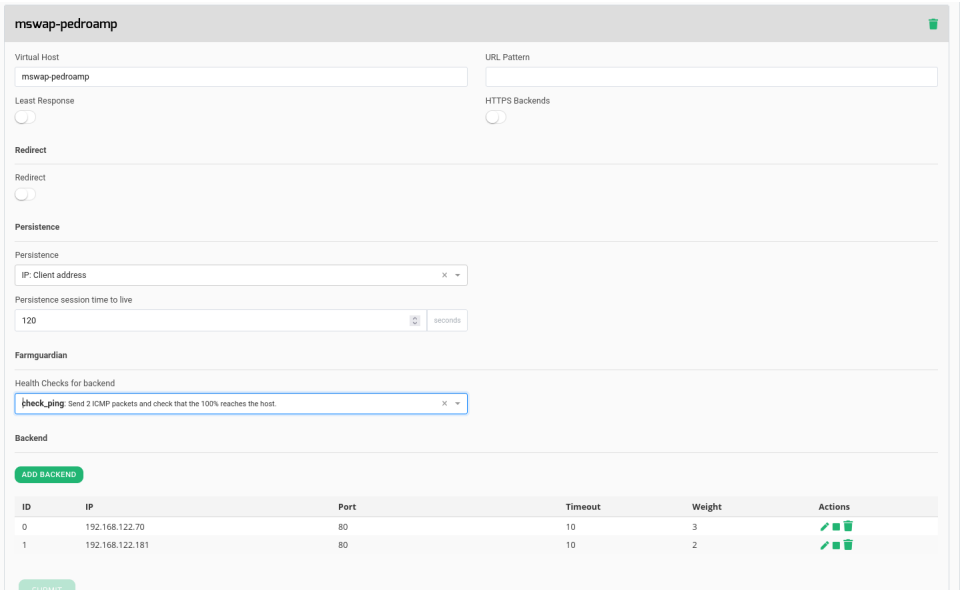

Figura 54: Creando el servicio de mswap-pedroamp configuración.

Finalmente el icono del status cambia a verde indicando que el servicio está funcionando.

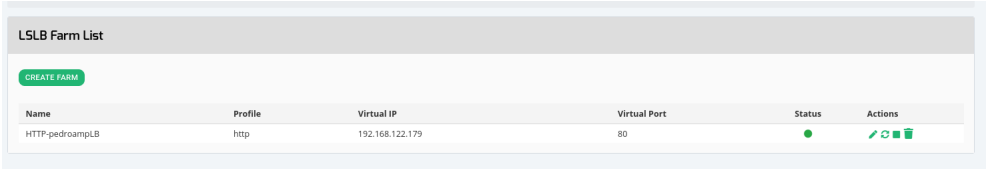

Figura 55: Status de la granja.

4.3. Pruebas del backend

Para realizar las pruebas he debido quitar la persistencia por IP para que las peticiones de los servidores se repartan con el modo weighted y el modo RR de manera indiscriminada sin asignar un servidor del backend. De lo contrario esta prueba es poco fiable ya que todas las peticiones irán a m1-pedroamp por su mayor peso y la persistencia de sesión por IP.
Pruebas con weighted:

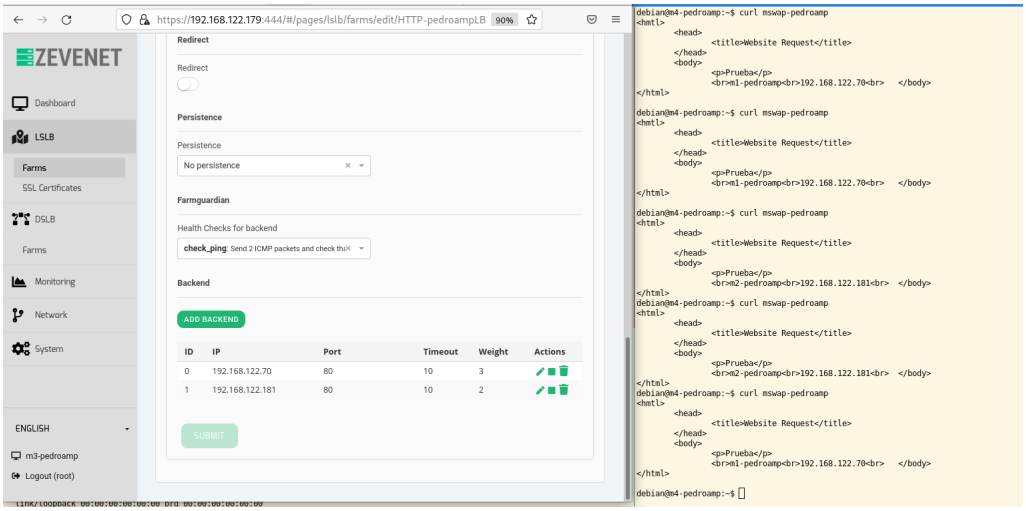

Figura 56: Weighted prueba con Zevenet.

Pruebas en modo RR:

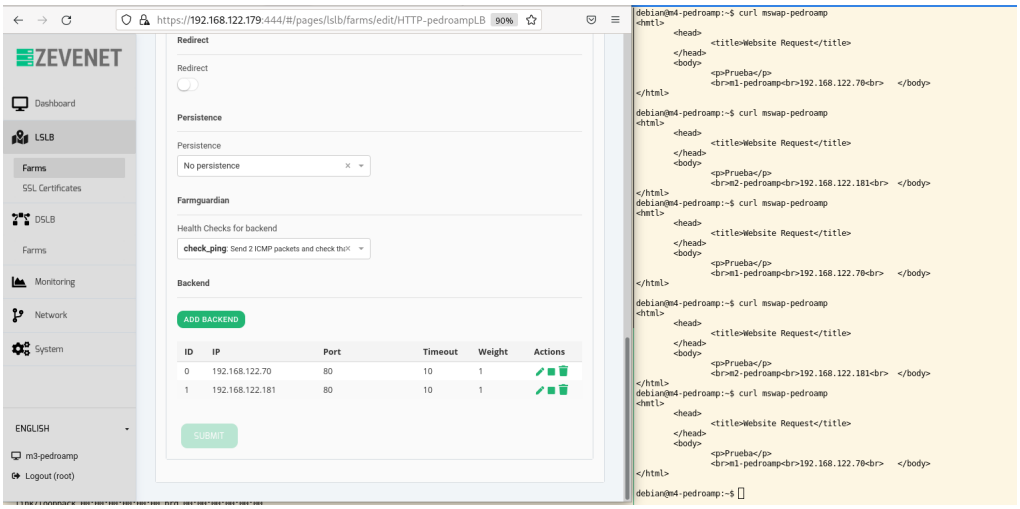

Figura 57: RR Zevenet.

5. AB Apache Benchmark

Es una utilidad incluida en un paquete llamado apache2-utils para instalarlo debemos poner:

```
sudo apt-get install apache2-utils
```

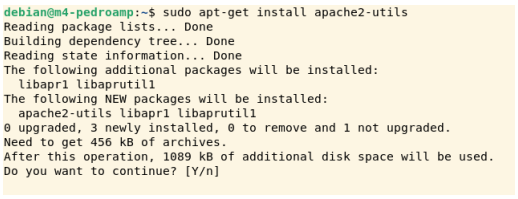

Figura 58: Instalación de la utilidad apache benchmark

Utilizo el siguiente comando para testear todo el sistema:

```
ab -l -k -n 10000 -c 100 -g datos.tsv
        http://mswap-pedroamp/
```

1. -l – durante las pruebas he tenido fallos de peticiones en las que ab asumía falsamente que como ha variado el tamaño de la respuesta es que está mal. Pero como mi contenido es alojado con PHP de manera "dinámica"donde lo que variaba es que cambia la variable interna de qué servidor y qué IP es. He considerado oportuno incluirlo, porque generaba un error mucho más grande en el tiempo de reacción de las peticiones al tener que rehacerla y añadiendo casi 0.3 segundos a la prueba sintética total.
2. -k – keepalive
3. -n [n request] - n de peticiones a realizar.
4. -c [n concurrent] - n de clientes concurrentes a imitar.
5. -g datos.tsv - datos en tab separated values, donde se usa luego a continuación para gnuplot para generar gráficos.

Para procesar el fichero .tsv en una gráfica. Tenemos que generar un fichero datosrr.p que es para que gnuplot genere el png de la gráfica a partir de los datos de ambos tsv para compararlos.
He tenido un error de lectura donde he identificado que el -g generaba un fichero CSV, en vez de uno TSV en realidad. Entonces a pesar de que ponga .csv en las capturas son .tsv

```
set terminal png size 600
set output "resultadosRR.png"
set title "10000 peticiones, 100 peticiones concurrentes"
set size ratio 0.6
set grid y
set xlabel "Nro Peticiones"
set ylabel "Tiempo de respuesta (ms)"
plot "nginxdatosrr.csv" using 9 smooth sbezier with lines
title "http://mswap-pedroamp/ nginx as load balancer",
"haproxydatosrr.csv" using 9 smooth sbezier with lines
title "http://mswap-pedroamp/ haproxy as load balancer"
```

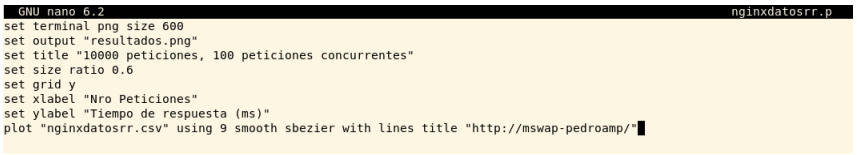

Figura 59: Fichero .p generado

Finalmente tenemos que ejecutar el siguiente comando para generar el gráfico:

```
gnuplot nginxdatosrr.p
```

5.1. Configuración round-robin

Vamos a testear el comportamiento de los servidores y los balanceadores (en nginx es proxy inverso). Para ver su diferencia de rendimiento. Pero antes muestro la configuración establecida tanto para nginx y haproxy en m3-pedroamp.
Configuración de nginx con round-robin.

Figura 60: Configuración de nginx round robin

Configuración de HAProxy con round robin

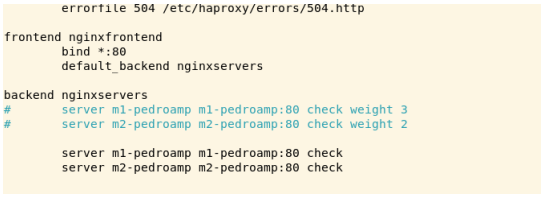

Figura 61: Configuración de HAProxy round robin

Configuración de gobetween con round robin.

```
# ---------- tcp example ----------- #

[servers.mswap-pedroamp]
protocol = "tcp"
bind = "192.168.122.179:80"
balance="roundrobin"

  [servers.mswap-pedroamp.discovery]
  kind = "static"
  static_list = [
      "192.168.122.70:80",
      "192.168.122.181:80",
     # "localhost:8002 weight=100 priority=0"
  ]

# ---------- udp example ----------- #
```

Figura 62: Gobetween en roundrobin.

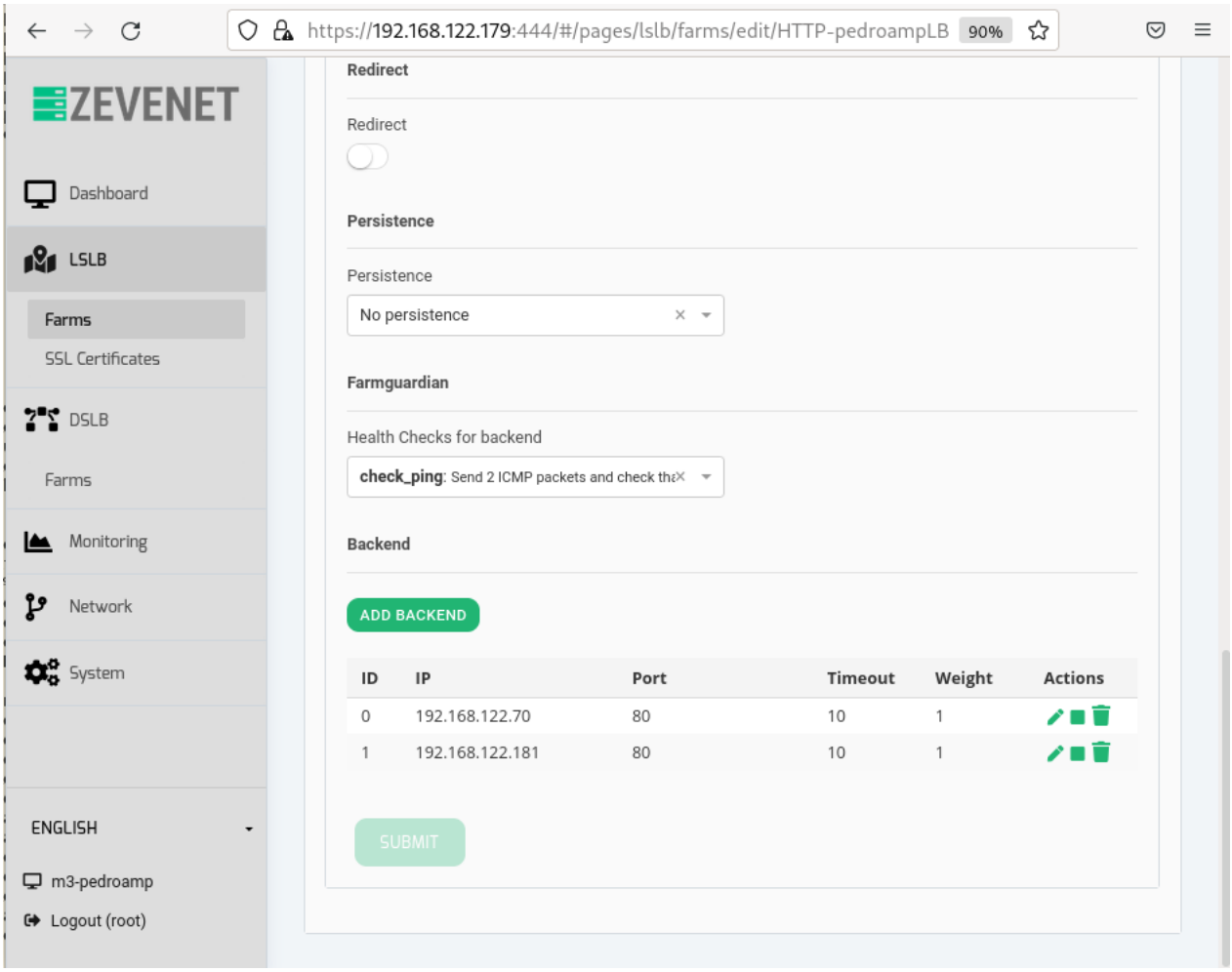

Figura 63: Zevenet en roundrobin.

5.1.1. Prueba de estrés con nginx.

```
debian@m4-pedroamp:~$ ab -k -n 10000 -c 100 -g nginxdatosrr.csv http://mswap-pedroamp/
This is ApacheBench, Version 2.3 <$Revision: 1879490 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking mswap-pedroamp (be patient)
Completed 1000 requests
Completed 2000 requests
Completed 3000 requests
Completed 4000 requests
Completed 5000 requests
Completed 6000 requests
Completed 7000 requests
Completed 8000 requests
Completed 9000 requests
Completed 10000 requests
Finished 10000 requests


Server Software:        nginx/1.18.0
Server Hostname:        mswap-pedroamp
Server Port:            80

Document Path:          /
Document Length:        213 bytes

Concurrency Level:      100
Time taken for tests:   1.681 seconds
Complete requests:      10000
Failed requests:        0
Keep-Alive requests:    9904
Total transferred:      3989520 bytes
HTML transferred:       2130000 bytes
Requests per second:    5949.41 [#/sec] (mean)
Time per request:       16.808 [ms] (mean)
Time per request:       0.168 [ms] (mean, across all concurrent requests)
Transfer rate:          2317.90 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.4      0       5
Processing:     5   17   2.4     17      38
Waiting:        2   17   2.4     16      38
Total:          5   17   2.4     17      40

Percentage of the requests served within a certain time (ms)
  50%     17
  66%     17
  75%     17
  80%     18
  90%     18
  95%     19
  98%     23
  99%     31
 100%     40 (longest request)
```

Figura 64: Configuración de nginx round robin prueba

La tabla informativa de los connection times medido en ms funciona de la siguiente manera:

1. connect: es el tiempo que tarda ab en establecer una conexión TCP three-way-handshake y empezar con su HTTP request.
2. Processing: Es el tiempo está la conexión abierta despues de haber sido creada.
3. Waiting: Es el tiempo que está esperando ab después de enviar su HTTP request y que el servidor envíe una respuesta HTTP response.
4. Total: Es el tiempo total desde que pasa en los 3 estados anteriores AB.

Suele tener una desviación típica de 2.4ms, es decir que los datos están muy centrados. En el caso con el balanceador nginx. Sus máximos son 38ms en el tiempo en el que tarda el servidor en responder a una petición con 100 usuarios concurrentes.
Los datos más importantes aparte de la tabla son:

1. Time taken for test - Tiempo tomado para realizar el test.
2. Complete request - indica las peticiones completadas.

3. Failed request - Peticiones fallidas, suele darse en el caso de que el servidor llegue al punto del colapso.

4. Time per request media.

5. Time per request sobre todos los grupos concurrentes su media.

```
debian@m4-pedroamp:~$ ab -k -n 10000 -c 100 -g nginxdatosrr.csv http://mswap-pedroamp/
This is ApacheBench, Version 2.3 <$Revision: 1879490 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking mswap-pedroamp (be patient)
Completed 1000 requests
Completed 2000 requests
Completed 3000 requests
Completed 4000 requests
Completed 5000 requests
Completed 6000 requests
Completed 7000 requests
Completed 8000 requests
Completed 9000 requests
Completed 10000 requests
Finished 10000 requests


Server Software:        nginx/1.18.0
Server Hostname:        mswap-pedroamp
Server Port:            80

Document Path:          /
Document Length:        213 bytes

Concurrency Level:      100
Time taken for tests:   1.681 seconds
Complete requests:      10000
Failed requests:        0
Keep-Alive requests:    9904
Total transferred:      3989520 bytes
HTML transferred:       2130000 bytes
Requests per second:    5949.41 [#/sec] (mean)
Time per request:       16.808 [ms] (mean)
Time per request:       0.168 [ms] (mean, across all concurrent requests)
Transfer rate:          2317.90 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.4      0       5
Processing:     5   17   2.4     17      38
Waiting:        2   17   2.4     16      38
Total:          5   17   2.4     17      40

Percentage of the requests served within a certain time (ms)
  50%     17
  66%     17
  75%     17
  80%     18
  90%     18
  95%     19
  98%     23
  99%     31
 100%     40 (longest request)
```

Figura 65: Prueba con nginx

5.1.2. Prueba de estrés con HAProxy.

Ejecuto los mismos parámetros indicados en la subsección de nginx, aquí no me voy a parar a analizar lo que es cada cosa. Voy a los datos concretos.

```
debian@m4-pedroamp:~$ ab -l -k -n 10000 -c 100 -g haproxydatosrr.csv http://mswap-pedroamp/
This is ApacheBench, Version 2.3 <$Revision: 1879490 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking mswap-pedroamp (be patient)
Completed 1000 requests
Completed 2000 requests
Completed 3000 requests
Completed 4000 requests
Completed 5000 requests
Completed 6000 requests
Completed 7000 requests
Completed 8000 requests
Completed 9000 requests
Completed 10000 requests
Finished 10000 requests


Server Software:        nginx/1.18.0
Server Hostname:        mswap-pedroamp
Server Port:            80

Document Path:          /
Document Length:        Variable

Concurrency Level:      100
Time taken for tests:   1.294 seconds
Complete requests:      10000
Failed requests:        0
Keep-Alive requests:    0
Total transferred:      2760000 bytes
HTML transferred:       1390000 bytes
Requests per second:    7729.65 [#/sec] (mean)
Time per request:       12.937 [ms] (mean)
Time per request:       0.129 [ms] (mean, across all concurrent requests)
Transfer rate:          2083.38 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    1   0.8      0       8
Processing:     2   12   4.7     12      33
Waiting:        1   12   4.7     12      33
Total:          2   13   4.7     12      38
WARNING: The median and mean for the initial connection time are not within a normal deviation
        These results are probably not that reliable.

Percentage of the requests served within a certain time (ms)
  50%     12
  66%     14
  75%     16
  80%     16
  90%     19
  95%     21
  98%     25
  99%     27
 100%     38 (longest request)
```

Figura 66: Prueba con haproxy.

Los datos nos indican que la el tiempo total, la conexión, procesado y espera son mucho mas bajos que en nginx, pero son más inestables según lo que dice la standard deviation porque está más disperso.

5.1.3. Prueba de estrés con gobetween

El resultado es el siguiente. Ha tardado bastante más que otros balanceadores como HAProxy o nginx por mucho.

```
debian@m4-pedroamp:~$ ab -k -n 10000 -c 100 -g gobetweendatosrr.csv http://mswap-pedroamp/
This is ApacheBench, Version 2.3 <$Revision: 1879490 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking mswap-pedroamp (be patient)
Completed 1000 requests
Completed 2000 requests
Completed 3000 requests
Completed 4000 requests
Completed 5000 requests
Completed 6000 requests
Completed 7000 requests
Completed 8000 requests
Completed 9000 requests
Completed 10000 requests
Finished 10000 requests


Server Software:        nginx/1.18.0
Server Hostname:        mswap-pedroamp
Server Port:            80

Document Path:          /
Document Length:        140 bytes

Concurrency Level:      100
Time taken for tests:   2.834 seconds
Complete requests:      10000
Failed requests:        5000
   (Connect: 0, Receive: 0, Length: 5000, Exceptions: 0)
Keep-Alive requests:    0
Total transferred:      2760000 bytes
HTML transferred:       1390000 bytes
Requests per second:    3528.87 [#/sec] (mean)
Time per request:       28.338 [ms] (mean)
Time per request:       0.283 [ms] (mean, across all concurrent requests)
Transfer rate:          951.14 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    1   1.0      0       7
Processing:     3   28   7.3     27      57
Waiting:        3   27   7.3     27      56
Total:          4   28   7.3     28      57
WARNING: The median and mean for the initial connection time are not within a normal deviation
        These results are probably not that reliable.

Percentage of the requests served within a certain time (ms)
  50%     28
  66%     30
  75%     33
  80%     34
  90%     38
  95%     41
  98%     45
  99%     47
 100%     57 (longest request)
```

Figura 67: Prueba con gobetween.

5.1.4. Prueba de estrés con Zevenet

Zevenet no he podido realizar la prueba de estrés por factores desconocidos. He incluido hasta la opción -r para poder continuar la prueba ignorando errores de sockets. Pero al llegar a las 9000 se atasca y cierra la conexión.
Hice la sugerencia del profesor donde reduzco el n de peticiones de AB y quitando la concurrencia además pero no he logrado nada. El indicador de Zevenet web se pone en azul que indica que hay problemas para llegar al backend, por lo que no se cual podría ser el problema ya que con los demás benchmark no ha ocurrido ningún problema.

```
Total of 190 requests completed
debian@m4-pedroamp:~$ ab -k -r -n 10000 -c 100 -g zevenetdatosrr.csv http://mswap-pedroamp/
This is ApacheBench, Version 2.3 <$Revision: 1879490 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking mswap-pedroamp (be patient)
Completed 1000 requests
Completed 2000 requests
Completed 3000 requests
Completed 4000 requests
Completed 5000 requests
Completed 6000 requests
Completed 7000 requests
Completed 8000 requests
Completed 9000 requests
apr_pollset_poll: The timeout specified has expired (70007)
Total of 9970 requests completed
```

Figura 68: Prueba con Zevenet.

5.1.5. GNUPlot comparativo de Round Robin

El balanceador HAProxy tiene menor tiempo de respuesta o latencia. Gobetween es el peor balanceador de carga.
Gobetween es un balanceador de carga que está menos testeado que los otros dos balanceadores dando un peor rendimiento.
Nginx, al principio es el segundo peor balanceador, pero

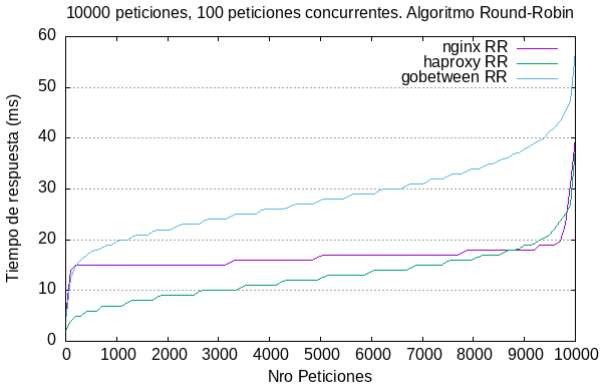

Figura 69: Gráfico comparativo de nginx vs haproxy vs gobetween.

5.2. Configuración ponderada (weighted)

Aquí vamos a estresar más el servidor m1 para ver como escalan los balanceadores la carga del benchmark ab.
Las configuraciones establecidas son:

```
http {
        log_format upstreamlog 'Client_addr: $remote_addr -Server_addr $upstream_addr

        upstream mswap-pedroamp {
                server m1-pedroamp weight=3;
                server m2-pedroamp weight=2;
                #server m5-pedroamp backup;
        }
        server {
                listen 80;
                location / {
                        proxy_pass http://mswap-pedroamp;
                }

                access_log /var/log/nginx/upstream.log upstreamlog;

        }

        access_log /var/log/nginx/access.log;
        error_log /var/log/nginx/error.log;
```

Figura 70: nginx weighted.

```
        errorfile 502 /etc/haproxy/errors/502.http
        errorfile 503 /etc/haproxy/errors/503.http
        errorfile 504 /etc/haproxy/errors/504.http

frontend nginxfrontend
        bind *:80
        default_backend nginxservers

backend nginxservers
        server m1-pedroamp m1-pedroamp:80 check weight 3
        server m2-pedroamp m2-pedroamp:80 check weight 2

#       server m1-pedroamp m1-pedroamp:80 check
#       server m2-pedroamp m2-pedroamp:80 check
```

Figura 71: HAProxy weighted.

```
# ---------- tcp example ----------- #

[servers.mswap-pedroamp]
protocol = "tcp"
bind = "192.168.122.179:80"
#balance="roundrobin"
balance="weight"

  [servers.mswap-pedroamp.discovery]
  kind = "static"
  static_list = [
        "192.168.122.70:80 weight=3",
        "192.168.122.181:80 weight=2",
     # "localhost:8002 weight=100 priority=0"
  ]

  [servers.mswap-pedroamp.healthcheck]
  kind="ping"
  interval = "2s"
  ping_timeout_duration = "500ms"
  fails = 1
  passes = 1

# ---------- udp example ----------- #
```

Figura 72: Gobetween weighted.

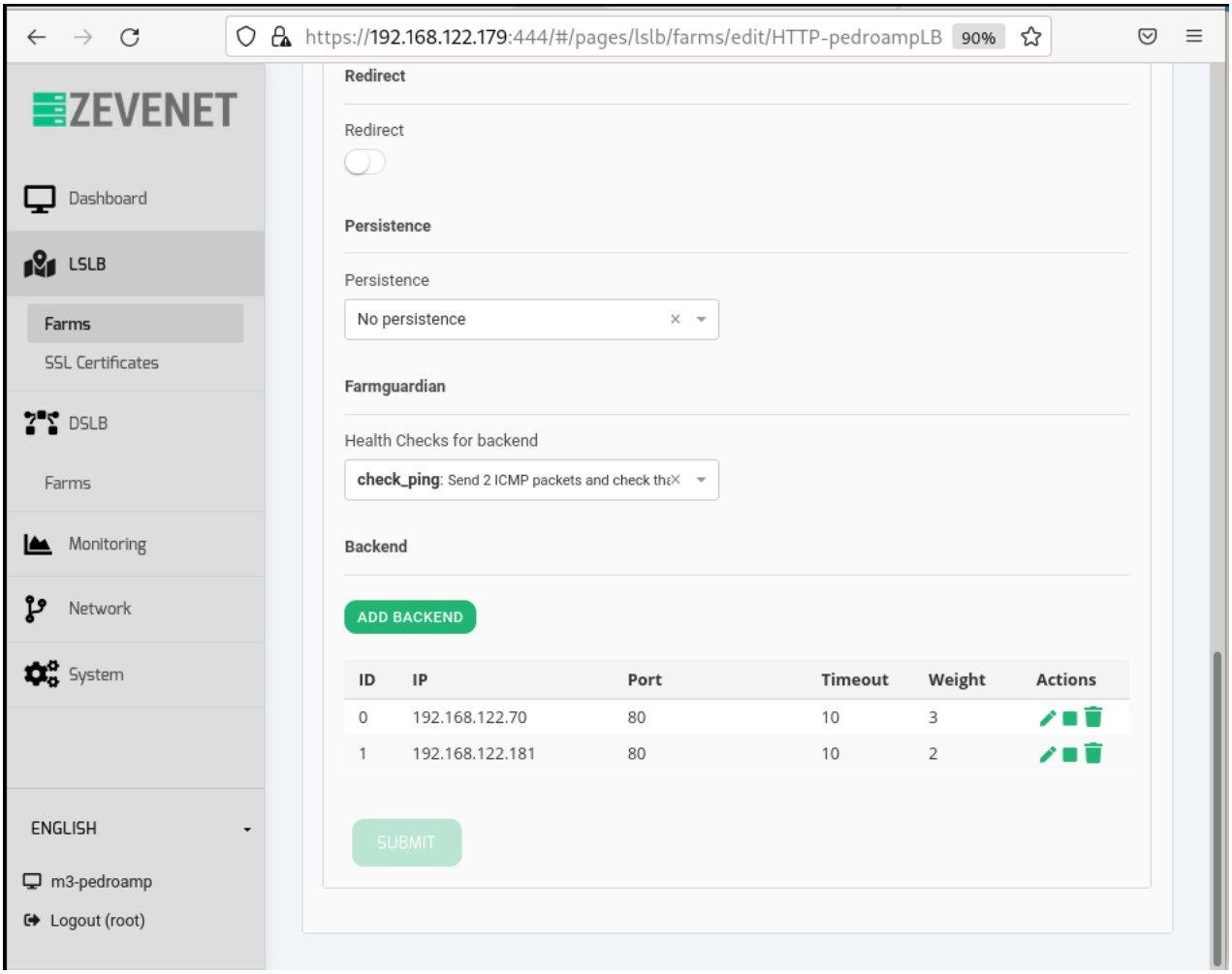

Figura 73: Zevenet weighted.

5.2.1. Prueba de estrés con nginx

```
debian@m4-pedroamp:~$ ab -l -k -n 10000 -c 100 -g nginxdatosrrweighted.csv http://mswap-pedroamp/
This is ApacheBench, Version 2.3 <$Revision: 1879490 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking mswap-pedroamp (be patient)
Completed 1000 requests
Completed 2000 requests
Completed 3000 requests
Completed 4000 requests
Completed 5000 requests
Completed 6000 requests
Completed 7000 requests
Completed 8000 requests
Completed 9000 requests
Completed 10000 requests
Finished 10000 requests


Server Software:        nginx/1.18.0
Server Hostname:        mswap-pedroamp
Server Port:            80

Document Path:          /
Document Length:        Variable

Concurrency Level:      100
Time taken for tests:   1.277 seconds
Complete requests:      10000
Failed requests:        0
Keep-Alive requests:    0
Total transferred:      2762000 bytes
HTML transferred:       1392000 bytes
Requests per second:    7830.96 [#/sec] (mean)
Time per request:       12.770 [ms] (mean)
Time per request:       0.128 [ms] (mean, across all concurrent requests)
Transfer rate:          2112.22 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.7      0       5
Processing:     1   12   7.4     13      34
Waiting:        0   12   7.4     13      34
Total:          1   13   7.4     14      34

Percentage of the requests served within a certain time (ms)
  50%     14
  66%     17
  75%     19
  80%     19
  90%     21
  95%     24
  98%     25
  99%     26
 100%     34 (longest request)
```

Figura 74: Prueba de estrés con nginx weighted.

Los resultados nos indican que ha tenido un total de 34 ms máximo en lo que es el tiempo total de respuesta para completar la petición HTTP realizada con AB. El tiempo de espera es muy poco.
Solamente se ve el problema de que hay mucho margen de tiempo con la sd standard deviation, no está concentrado.

5.2.2. Prueba de estrés con HAProxy

```
debian@m4-pedroamp:~$ ab -l -k -n 10000 -c 100 -g haproxydatosrrweighted.csv http://mswap-pedroamp/
This is ApacheBench, Version 2.3 <$Revision: 1879490 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking mswap-pedroamp (be patient)
Completed 1000 requests
Completed 2000 requests
Completed 3000 requests
Completed 4000 requests
Completed 5000 requests
Completed 6000 requests
Completed 7000 requests
Completed 8000 requests
Completed 9000 requests
Completed 10000 requests
Finished 10000 requests


Server Software:        nginx/1.18.0
Server Hostname:        mswap-pedroamp
Server Port:            80

Document Path:          /
Document Length:        Variable

Concurrency Level:      100
Time taken for tests:   1.430 seconds
Complete requests:      10000
Failed requests:        0
Keep-Alive requests:    0
Total transferred:      2762000 bytes
HTML transferred:       1392000 bytes
Requests per second:    6990.89 [#/sec] (mean)
Time per request:       14.304 [ms] (mean)
Time per request:       0.143 [ms] (mean, across all concurrent requests)
Transfer rate:          1885.63 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    1   1.0      0      10
Processing:     1   14   8.3     12      80
Waiting:        1   13   8.3     12      80
Total:          1   14   8.3     13      80
WARNING: The median and mean for the initial connection time are not within a normal deviation
        These results are probably not that reliable.

Percentage of the requests served within a certain time (ms)
  50%     13
  66%     16
  75%     18
  80%     19
  90%     22
  95%     25
  98%     36
  99%     44
 100%     80 (longest request)
```

Figura 75: Prueba de estrés con HAProxy weighted.

A primera vista puedo ver que hay un tiempo de respuesta de media más bajo pero tiene mayor sd. Pero el tiempo total es mucho mayor que en el caso de realizarlo con nginx.

5.2.3. Prueba de estrés con gobetween

```
debian@m4-pedroamp:~$ ab -k -n 10000 -c 100 -g gobetweendatosweighted.csv http://mswap-pedroamp/
This is ApacheBench, Version 2.3 <$Revision: 1879490 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking mswap-pedroamp (be patient)
Completed 1000 requests
Completed 2000 requests
Completed 3000 requests
Completed 4000 requests
Completed 5000 requests
Completed 6000 requests
Completed 7000 requests
Completed 8000 requests
Completed 9000 requests
Completed 10000 requests
Finished 10000 requests


Server Software:        nginx/1.18.0
Server Hostname:        mswap-pedroamp
Server Port:            80

Document Path:          /
Document Length:        140 bytes

Concurrency Level:      100
Time taken for tests:   2.987 seconds
Complete requests:      10000
Failed requests:        3969
   (Connect: 0, Receive: 0, Length: 3969, Exceptions: 0)
Keep-Alive requests:    0
Total transferred:      2762062 bytes
HTML transferred:       1392062 bytes
Requests per second:    3347.86 [#/sec] (mean)
Time per request:       29.870 [ms] (mean)
Time per request:       0.299 [ms] (mean, across all concurrent requests)
Transfer rate:          903.03 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    1   1.0      0       7
Processing:     5   29   7.7     29      66
Waiting:        2   29   7.6     29      66
Total:          8   30   7.6     29      72
WARNING: The median and mean for the initial connection time are not within a normal deviation
        These results are probably not that reliable.

Percentage of the requests served within a certain time (ms)
  50%     29
  66%     33
  75%     35
  80%     36
  90%     40
  95%     43
  98%     46
  99%     48
 100%     72 (longest request)
```

Figura 76: Prueba de estrés con gobetween weighted.

5.2.4. Prueba de estrés con Zevenet

Es imposible realizar la prueba por factores desconocidos donde a partir de los 9000 se atasca la prueba al igual que en RR. Consultar la sección de RR para Zevenet.
Hice la sugerencia del profesor donde reduzco el n de peticiones de AB y quitando la concurrencia además pero no he logrado nada. El indicador de Zevenet web se pone en azul que indica que hay problemas para llegar al backend, por lo que no se cual podría ser el problema ya que con los demás benchmark no ha ocurrido ningún problema.

5.2.5. GNUPlot comparativo weighted

Nginx cuando es ponderado tiene mejor tiempo de respuesta hasta 3500 peticiones. Luego es mas o menos lo mismo hasta las 9500 peticiones donde HAProxy reacciona peor. Gobetween es el que peor resultado da.
Creo que gobetween al estar escrito en un lenguaje que quizás da peor rendimiento de partida rinde muchísimo peor que ambos balanceadores. Aparte de que nginx y haproxy, son balanceadores que se han ido optimizando mucho más por su uso mas extendido en todo el mundo, al contrario que gobetween que está mas verde.

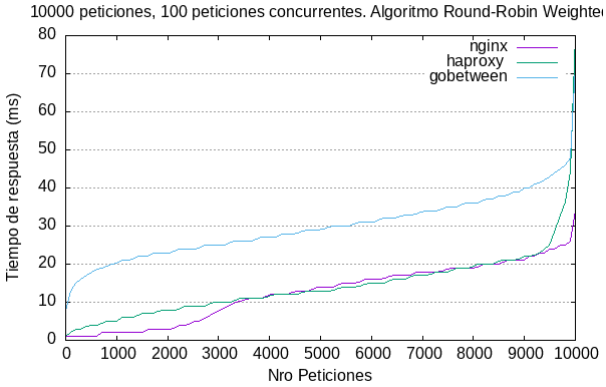

Figura 77: Gráfico comparativo de nginx vs haproxy.

6. Bibliografía

Nginx configuration:
[1] https://nginx.org/en/docs/http/load_balancing.html
Log personalizado para upstream:
[2] nginx.org/en/docs/http/ngx_http_log_module.html#log_format
[3] http://nginx.org/en/docs/http/ngx_http_log_module.html#access_log
[4] https://nginx.org/en/docs/http/ngx_http_upstream_module.html#variables
[5] https://stackoverflow.com/questions/35604530/how-to-keep-track-of-an-upstream-when-nginx-has-multiple-upstreams
HAProxy configuration:
[6] https://cbonte.github.io/haproxy-dconv/2.5/configuration.html
[7] https://www.haproxy.com/documentation/hapee/latest/
[8] https://www.haproxy.com/blog/haproxy-configuration-basics-load-balance-your-servers/
[9] https://www.haproxy.com/documentation/hapee/latest/configuration/servers/syntax/
[10] https://www.haproxy.com/blog/exploring-the-haproxy-stats-page/
[11] https://www.haproxy.com/blog/how-to-enable-health-checks-in-haproxy/
Gobetween configuration:
[12] https://gobetween.io/documentation.html#Static-balancing
[13] https://gobetween.io/documentation.html#Balancing
[14] https://gobetween.io/documentation.html#REST-API
[15] https://gobetween.io/documentation.html#Healthchecks
[16] https://github.com/yyyar/gobetween/issues/152
Configuración de Zevenet
[17] https://www.zevenet.com/knowledge-base/community-edition/community-edition-v5-9-administration-guide/ce-v5-9-login/

[1] Configuración de Nginx
[2] Configuración de log_format
[3] Configuración de access_log
[4] Configuración de Embedded Variables ngx_http_upstream_module
[5] Tracking upstream REQUEST
[6] HAProxy biblia de consulta de configuraciones
[7] HAProxy biblia de consulta de configuraciones - Enterprise version - Tratar con cuidado no es nuestra versión pero el 90 % coincide
[8] HAProxy basic load balance
[9] HAProxy basic load balance - Configuración de servidores y su weight
[10] HAProxy Configuración de la web de estadísticas
[11] HAProxy Configuración de la web de estadísticas - Activando la comprobación de salud
[12] Configuración básica de un balanceador de carga
[13] Algoritmos de balanceo de carga disponibles
[14] Uso de la api REST para información de estado de servidores
[15] Variables para comprobación de Health de servidores
[16] Límites de gobetween
[17] Login con Zevenet

[18] https://githubhot.com/repo/zevenet/zlb/issues/108
[19] https://www.zevenet.com/knowledge-base/community-edition/community-edition-v5-9-administration-guide/ce-v5-9-lslb-farms-create-farm/
[20] https://www.zevenet.com/knowledge-base_category/community-edition-v5-9-administration-guide/
AB Configuration and GNUPlot
[21] https://httpd.apache.org/docs/2.4/programs/ab.html
[22] https://juantrucupei.wordpress.com/2015/11/10/pruebas-con-ab-apache-benchmark/
[23] https://www.datadoghq.com/blog/apachebench/
Configuración de un servicio de SystemD propio
[24] https://www.shubhamdipt.com/blog/how-to-create-a-systemd-service-in-linux/
[25] https://www.freedesktop.org/software/systemd/man/systemd.service.html
[26] https://tuttlem.github.io/2018/02/03/create-a-systemd-daemon.html

[18] Error de acceso a web con Zevenet
[19] Creando granja con Zevenet
[20] Creando granja con Zevenet
[21] Comando de Apache Benchmark
[22] AB con GNUPlot tutorial encontrado
[23] Descripción de qué es cada cosa que sale en AB y cómo analizarlo
[24] Plantilla de configuración básica - Esta está pensada para scripts
[25] Manual de configuración de un fichero .service de SystemD
[26] Otra configuración de un .service. De aquí he extraido el After