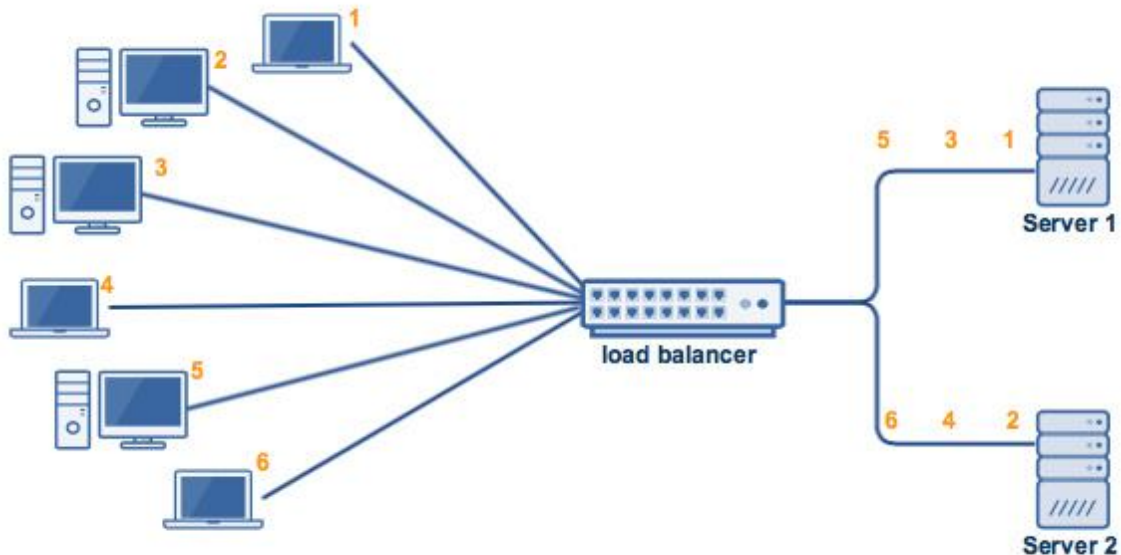


PRÁCTICA 3: BALANCEO DE CARGA DE UN SITIO WEB

Iñaki Melguizo Marcos



1. INTRODUCCIÓN

En esta práctica configuraremos una red entre varias máquinas de tal forma que tengamos un balanceador que reparta la carga entre los servidores finales. Balancearemos los servidores HTTP que tenemos configurados y con esto conseguiremos una infraestructura redundante y de alta disponibilidad. En esta práctica utilizaremos balanceo por software y explicaremos cómo balancear carga con:

- NginX
- HaProxy
- GoBetween
- Zevenet
- Pound

con distintas opciones cada uno de ellos. Posteriormente someteremos a carga a la granja web con la herramienta AB y por último haremos un análisis comparativo de distintos balanceados en base a la carga con AB

Debemos tener dos máquinas servidoras finales (m1 y m2) que las teníamos creadas de prácticas a las que tendremos que eliminar la sincronización de los directorios `/var/www/` que teníamos establecida mediante el demonio cron:

```
m2-imm98 [Corriendo] - Oracle VM VirtualBox
Archivo  Máquina  Ver  Entrada  Dispositivos  Ayuda
# and day of week (dow) or use '*' in these fields (for 'any').#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h dom mon dow   command
# * * * * * rsync -avz -e 'ssh -p 2222' 192.168.56.105:/var/www/ /var/www/
```

Por tanto debemos eliminar esa línea que estaba escrita de la práctica anterior. También debemos de crear una tercera máquina que llamaremos m3-imm 98 en la que no puede haber ningún software que se apropie del puerto 80 ya que esta va a ser la máquina que gestionará el balanceo de carga.

Por último necesitaremos tener nuestra máquina host activa para realizar peticiones HTTP a la máquina balanceadora.

2. NGINX

En primer lugar vamos a utilizar NginX como balanceador. Instalamos NginX en la máquina m3-imm 98 que se encargará de redirigir el tráfico a los servidores finales. Para la instalación ejecutaremos

```
sudo apt-get update && sudo apt-get dist-upgrade && sudo apt-get autoremove
```

```
imm98@m3-imm98:~$ sudo apt-get update && sudo apt-get dist-upgrade && sudo apt-get autoremove
```

```
sudo apt-get install nginx
```

```
imm98@m3-imm98:~$ sudo apt-get install nginx
```

```
sudo systemctl start nginx
```

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

```
imm98@m3-imm98:~$ sudo systemctl start nginx
imm98@m3-imm98:~$ 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: active (running) since Tue 2021-04-06 17:27:16 UTC; 1min 2s ago
     Docs: man:nginx(8)
  Main PID: 15830 (nginx)
    Tasks: 2 (limit: 1107)
   CGroup: /system.slice/nginx.service
           └─15830 nginx: master process /usr/sbin/nginx -g daemon on; master_process on;
             └─15832 nginx: worker process

Apr 06 17:27:16 m3-imm98 systemd[1]: Starting A high performance web server and a reverse proxy serv
Apr 06 17:27:16 m3-imm98 systemd[1]: nginx.service: Failed to parse PID from file /run/nginx.pid: In
Apr 06 17:27:16 m3-imm98 systemd[1]: Started A high performance web server and a reverse proxy serve
lines 1-13/13 (END)
```

y vemos que el servicio efectivamente está ejecutándose.

Para que al lanzarlo se ejecute como balanceador debemos de eliminar la línea que configurar NginX como servidor web en el archivo /etc/nginx/nginx.conf

```
# include /etc/nginx/sites-enabled/*
```

```
imm98@m3-imm98:/etc/nginx$ sudo vi nginx.conf
```

```
include /etc/nginx/conf.d/*.conf;
#include /etc/nginx/sites-enabled/*;
}
```

Dado que hemos modificado el archivo de configuración reiniciamos el servicio:

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

```
imm98@m3-imm98:/etc/nginx$ sudo systemctl restart nginx.service
```

Vamos a definir el grupo de servidores a los que queremos redirigir la carga. Para ello utilizamos la directiva *upstream*.

El fichero de configuración de NginX es **/etc/nginx/conf.d/default.conf** que en nuestro caso no estaba creado por lo que debemos crearlo

```
imm98@m3-imm98:/etc/nginx/conf.d$ sudo touch default.conf
imm98@m3-imm98:/etc/nginx/conf.d$ sudo nano default.conf
```

A modo de recordatorio de la práctica anterior, las IPs de las máquinas m1, m2 y m3 son las siguientes:

m1-imm 98	192.168.56.105
m2-imm 98	192.168.56.104
m3-imm 98	192.168.56.107

Por lo que en la sección upstream debemos escribir dichas IPs teniendo en cuenta también que el puerto de apache de m1-imm 98 estaba cambiado al 8080

```
imm98@m3-imm98:/etc/nginx/conf.d$ cat default.conf
upstream balanceo_imm98{
    server 192.168.56.105:8080;
    server 192.168.56.104;
}
server{
    listen 80;
    server_name balanceador_imm98;

    access_log /var/log/nginx/balanceador_imm98.access.log;
    error_log /var/log/nginx/balanceador_imm98.error.log;
    root /var/www/;

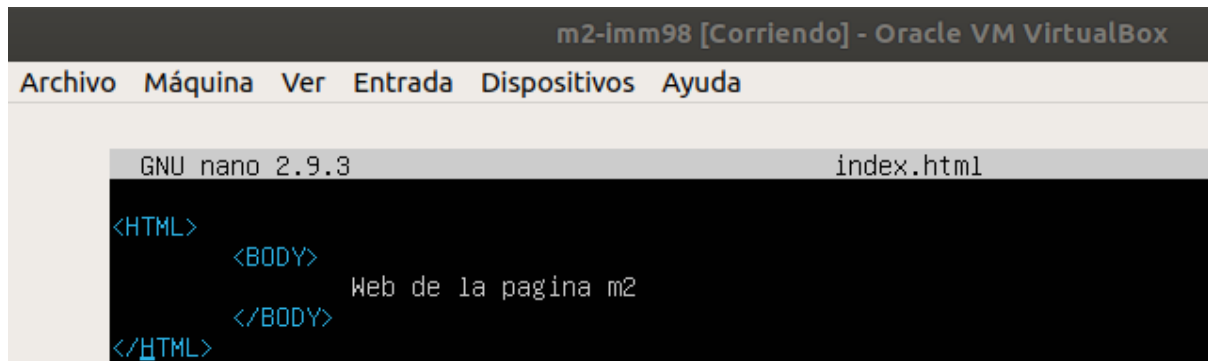
    location /
    {
        proxy_pass http://balanceo_imm98;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_http_version 1.1;
        proxy_set_header Connection "";
    }
}
imm98@m3-imm98:/etc/nginx/conf.d$ sudo systemctl restart nginx.service
[sudo] password for imm98:
```

En el ejemplo de arriba hemos utilizado “round-robin” con la máxima prioridad para todos los servidores.

Volvemos a reiniciar el servicio:

```
imm98@m3-imm98:/etc/nginx$ sudo systemctl restart nginx.service
```

Como nota aclaratoria hemos modificado los archivos *index.html* de los directorios */var/www/html* de m1-imm 98 y m2-imm 98 para mostrar un mensaje de que es el archivo de la máquina 1 y 2 respectivamente. Por ejemplo en el archivo *index.html* de m2-imm 98 tenemos:



```
m2-imm98 [Corriendo] - Oracle VM VirtualBox
Archivo  Máquina  Ver  Entrada  Dispositivos  Ayuda

GNU nano 2.9.3 index.html
<HTML>
  <BODY>
    Web de la pagina m2
  </BODY>
</HTML>
```

Como habíamos modificado el archivo de configuración de NginX de m3-imm 98 para que el método de balanceo fuera por roundrobin vamos a comprobarlo, ejecutando seguidamente la opción:

```
curl http://192.168.56.107
```

ya que 192.168.56.107 es la IP estática de m3-imm98



```
inaki@inaki-N501VW:~$ curl http://192.168.56.107
<HTML>
  <BODY>
    Web de la pagina m2
  </BODY>
</HTML>
inaki@inaki-N501VW:~$ curl http://192.168.56.107
<HTML>
  <BODY>
    Web de la pagina m1
  </BODY>
</HTML>
inaki@inaki-N501VW:~$ curl http://192.168.56.107
<HTML>
  <BODY>
    Web de la pagina m2
  </BODY>
</HTML>
```

y como el método de balanceo es RR vemos que una petición la ejecuta m1 y la siguiente m2 y así sucesivamente.

Ahora queremos que el algoritmo de balanceo sea con ponderación, suponiendo que la máquina m1-imm 98 tenga el doble de capacidad que la de m2-imm98. Para ello utilizaremos un modificador que se llama **weight** y el número que le acompaña indica la

carga que le asignamos por lo que si queremos que la máquina m1-imm98 soporte el doble de carga que m2-imm98 tendremos que asignarle:

- weight 2 a m1-imm98
- weight 1 a m2-imm98

Por lo tanto el archivo /etc/nginx/conf.d/default.conf quedaría:

```
imm98@m3-imm98:/etc/nginx/conf.d$ cat default.conf
upstream balanceo_imm98{
    server 192.168.56.105:8080 weight=2;
    server 192.168.56.104 weight=1;
}
server{
    listen 80;
    server_name balanceador_imm98;

    access_log /var/log/nginx/balanceador_imm98.access.log;
    error_log /var/log/nginx/balanceador_imm98.error.log;
    root /var/www/;

    location /
    {
        proxy_pass http://balanceo_imm98;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_http_version 1.1;
        proxy_set_header Connection "";
    }
}
imm98@m3-imm98:/etc/nginx/conf.d$
```

Vamos a probar que se ejecuta correctamente. Para ejecutamos como antes la orden

curl http://192.168.56.107

seguidamente y vemos que se muestra una vez el index.html de m2-imm98 por cada dos veces que se muestra el index.html de m1-imm98:

```
inaki@inaki-N501VM:~$ curl http://192.168.56.107
<HTML>
    <BODY>
        Web de la pagina m1
    </BODY>
</HTML>
inaki@inaki-N501VM:~$ curl http://192.168.56.107
<HTML>
    <BODY>
        Web de la pagina m1
    </BODY>
</HTML>
inaki@inaki-N501VM:~$ curl http://192.168.56.107
<HTML>
    <BODY>
        Web de la pagina m2
    </BODY>
</HTML>
inaki@inaki-N501VM:~$ curl http://192.168.56.107
<HTML>
    <BODY>
        Web de la pagina m1
    </BODY>
</HTML>
inaki@inaki-N501VM:~$ curl http://192.168.56.107
<HTML>
    <BODY>
        Web de la pagina m1
    </BODY>
</HTML>
```

OPCIONES AVANZADAS

También podemos hacer un balanceo por IP, mediante la directiva **ip_hash**. Esto provoca que todo el tráfico proveniente de una IP sus peticiones sean atendidas por el mismo servidor. La manera de implementarlo en el archivo de configuración `/etc/nginx/conf.d/default.conf` sería la siguiente:

```
imm98@m3-imm98:/etc/nginx/conf.d$ cat default.conf
upstream balanceo_imm98{
    ip_hash;
    server 192.168.56.105:8080;
    server 192.168.56.104;
}
server{
    listen 80;
    server_name balanceador_imm98;

    access_log /var/log/nginx/balanceador_imm98.access.log;
    error_log /var/log/nginx/balanceador_imm98.error.log;
    root /var/www/;

    location /
    {
        proxy_pass http://balanceo_imm98;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_http_version 1.1;
        proxy_set_header Connection "";
    }
}
imm98@m3-imm98:/etc/nginx/conf.d$ sudo systemctl restart nginx.service
```

Como en estas pruebas que estamos realizando, todas las peticiones mediante la opción `curl` las estamos lanzando desde la máquina anfitrión, todas las peticiones van a ser redirigidas únicamente a una de las dos máquinas como podemos ver en la imagen siguiente:

```
inaki@inaki-N501VW:~$ curl http://192.168.56.107
<HTML>
  <BODY>
    Web de la pagina m2
  </BODY>
</HTML>
inaki@inaki-N501VW:~$ curl http://192.168.56.107
<HTML>
  <BODY>
    Web de la pagina m2
  </BODY>
</HTML>
inaki@inaki-N501VW:~$ curl http://192.168.56.107
<HTML>
  <BODY>
    Web de la pagina m2
  </BODY>
</HTML>
inaki@inaki-N501VW:~$ curl http://192.168.56.107
<HTML>
  <BODY>
    Web de la pagina m2
  </BODY>
</HTML>
```

donde todas las peticiones HTTP son redirigidas por el balanceador a la máquina m2.

Otra de las directivas que se pueden utilizar para el balanceador de carga de NginX es **keepalive** n. Esta directiva sirve para que a través de una conexión TCP se puedan realizar muchas peticiones HTTP.:

```
imm98@m3-imm98:/etc/nginx/conf.d$ cat default.conf
upstream balanceo_imm98{
    server 192.168.56.105:8080;
    server 192.168.56.104;
    keepalive 3;
}
server{
    listen 80;
    server_name balanceador_imm98;

    access_log /var/log/nginx/balanceador_imm98.access.log;
    error_log /var/log/nginx/balanceador_imm98.error.log;
    root /var/www/;

    location /
    {
        proxy_pass http://balanceo_imm98;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_http_version 1.1;
        proxy_set_header Connection "";
    }
}
imm98@m3-imm98:/etc/nginx/conf.d$ sudo systemctl restart nginx.service
imm98@m3-imm98:/etc/nginx/conf.d$
```

Esto veremos posteriormente que obviamente disminuye el tiempo de ejecución de un número grande de peticiones HTTP ya que no es necesario abrir una conexión por cada petición HTTP.

Otra opción que podemos utilizar es la opción **max_fails**=n en el que le indicas el número de intentos de comunicación fallidos para considerar a un servidor como no operativo. Si no escribimos esta opción se encuentra a 1 por defecto. Si por ejemplo establecemos **max_fails=2** como se indica en la imagen:

```
imm98@m3-imm98:/etc/nginx/conf.d$ cat default.conf
upstream balanceo_imm98{
    server 192.168.56.105:8080 max_fails=2;
    server 192.168.56.104;
}
server{
    listen 80;
    server_name balanceador_imm98;

    access_log /var/log/nginx/balanceador_imm98.access.log;
    error_log /var/log/nginx/balanceador_imm98.error.log;
    root /var/www/;

    location /
    {
        proxy_pass http://balanceo_imm98;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_http_version 1.1;
        proxy_set_header Connection "";
    }
}
imm98@m3-imm98:/etc/nginx/conf.d$ sudo systemctl restart nginx.service
imm98@m3-imm98:/etc/nginx/conf.d$
```


Veremos posteriormente en la sección de comparativas de balanceadores y opciones de balanceadores de carga que es muy ineficiente en cuanto a tiempo de gestionar todas las peticiones ya que necesita comprobar dos veces que un servidor está caído.

La última opción de haproxy que probaremos será **down** que como su nombre indica marca un servidor como offline, es decir, que ninguna de las peticiones HTTP que le lleguen al balanceador haproxy van a ser redirigidas a un servidor marcado como down. Vamos a comprobarlo:

```
imm98@m3-imm98:/etc/nginx/conf.d$ cat default.conf
upstream balanceo_imm98{
    server 192.168.56.105:8080 down;
    server 192.168.56.104;
}
server{
    listen 80;
    server_name balanceador_imm98;

    access_log /var/log/nginx/balanceador_imm98.access.log;
    error_log /var/log/nginx/balanceador_imm98.error.log;
    root /var/www/;

    location /
    {
        proxy_pass http://balanceo_imm98;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_http_version 1.1;
        proxy_set_header Connection "";
    }
}
imm98@m3-imm98:/etc/nginx/conf.d$ sudo systemctl restart nginx.service
imm98@m3-imm98:/etc/nginx/conf.d$
```

Marcamos el servidor m1-imm98 como down, entonces vamos a ver que todas las peticiones HTTP las va a gestionar m2-imm98:

```
inaki@inaki-NS01VW:~$ curl http://192.168.56.107/index.html
<HTML>
  <BODY>
    Web de la pagina m2
  </BODY>
</HTML>
inaki@inaki-NS01VW:~$ curl http://192.168.56.107/index.html
<HTML>
  <BODY>
    Web de la pagina m2
  </BODY>
</HTML>
inaki@inaki-NS01VW:~$ curl http://192.168.56.107/index.html
<HTML>
  <BODY>
    Web de la pagina m2
  </BODY>
</HTML>
inaki@inaki-NS01VW:~$ curl http://192.168.56.107/index.html
<HTML>
  <BODY>
    Web de la pagina m2
  </BODY>
</HTML>
inaki@inaki-NS01VW:~$ curl http://192.168.56.107/index.html
<HTML>
  <BODY>
    Web de la pagina m2
  </BODY>
</HTML>
```

3. HAPROXY

Vamos a instalar el balanceador de carga haproxy. Para ello vamos a ejecutar la orden el comando:

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

```
imm98@m3-imm98:/etc/nginx/conf.d$ sudo apt-get install haproxy
Reading package lists... Done
Building dependency tree
```

Vamos a iniciar el servicio y comprobar que se está ejecutando este servicio mediante las órdenes. También debemos parar la ejecución del balanceador NginX ya que ambos escuchan del puerto 80.

```
sudo systemctl start haproxy
```

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

```
sudo service nginx stop
```

```
imm98@m3-imm98:/etc/nginx/conf.d$ sudo systemctl start haproxy
imm98@m3-imm98:/etc/nginx/conf.d$ sudo systemctl status haproxy.service
● haproxy.service - HAProxy Load Balancer
   Loaded: loaded (/lib/systemd/system/haproxy.service; enabled; vendor preset: e
   Active: active (running) since Wed 2021-04-07 17:10:08 UTC; 52s ago
     Docs: man:haproxy(1)
           file:/usr/share/doc/haproxy/configuration.txt.gz
   Main PID: 2315 (haproxy)
    Tasks: 2 (limit: 1107)
   CGroup: /system.slice/haproxy.service
           └─2315 /usr/sbin/haproxy -Ws -f /etc/haproxy/haproxy.cfg -p /run/hapro
              2316 /usr/sbin/haproxy -Ws -f /etc/haproxy/haproxy.cfg -p /run/hapro

Apr 07 17:10:08 m3-imm98 systemd[1]: Starting HAProxy Load Balancer...
Apr 07 17:10:08 m3-imm98 systemd[1]: Started HAProxy Load Balancer.
lines 1-13/13 (END)
imm98@m3-imm98:/etc/nginx/conf.d$ sudo service nginx stop
```

Una vez instalado y ejecutándose ya haproxy vamos a configurar haproxy como balanceador para que escuche desde el puerto 80 para redirigir las peticiones a las máquinas m1-imm98 y m2-imm98. Lo haremos modificando el archivo de configuración **/etc/haproxy/haproxy.cfg**.

```
imm98@m3-imm98:/etc/haproxy$ cat haproxy.cfg
frontend http-in
    bind *:80
    default_backend balanceo_imm98

backend balanceo_imm98
    server m1 192.168.56.105:8080 maxconn 32
    server m2 192.168.56.104:80 maxconn 32
```

Ya que hemos modificado el archivo de configuración de haproxy tenemos que reiniciar el servicio mediante la orden

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

```
imm98@m3-imm98:/etc/haproxy$ sudo systemctl restart haproxy.service
imm98@m3-imm98:/etc/haproxy$
```

y vemos que se ha reiniciado el servicio correctamente.

Ahora vamos a comprobar que el balanceador funciona correctamente. Como no le hemos indicado pesos este balanceador distribuirá la carga mediante round robin:

```
inaki@inaki-N501VW:~$ curl http://192.168.56.107/index.html
<HTML>
    <BODY>
        Web de la pagina m1
    </BODY>
</HTML>
inaki@inaki-N501VW:~$ curl http://192.168.56.107/index.html
<HTML>
    <BODY>
        Web de la pagina m2
    </BODY>
</HTML>
```

Vemos que se ejecuta correctamente el balanceo de carga.

Ahora vamos a balancear la carga por ponderación suponiendo que M1 tiene el doble de capacidad que M2. para ello utilizamos como en el caso de NginX la opción **weight** asociándole peso 2 a la máquina m1-imm98 y peso 1 a la máquina m2-imm98. Quedaría así:

```
imm98@m3-imm98:/etc/haproxy$ cat haproxy.cfg
frontend http-in
    bind *:80
    default_backend balanceo_imm98

backend balanceo_imm98
    server m1 192.168.56.105:8080 maxconn 32 weight 2
    server m2 192.168.56.104:80 maxconn 32 weight 1
```

Como modo avanzado vamos a hacer que nuestro balanceador redireccione una cierta petición HTTP al servidor que menos conexiones tuviera abiertas en ese momento. Para ello utilizaremos la opción **leastconn**. Además vamos a hacer uso de las cookies que las llamaremos MUSED. El archivo de configuración /etc/haproxy/haproxy.cfg quedaría así:

```
imm98@m3-imm98:/etc/haproxy$ sudo cat haproxy.cfg
frontend http-in
    bind *:80
    default_backend balanceo_imm98

backend balanceo_imm98
    balance leastconn
    cookie MUSED
    server m1 192.168.56.105:8080 maxconn 32 cookie m1
    server m2 192.168.56.104:80 maxconn 32 cookie m2
```

y reiniciamos el servicio ya que hemos modificado el fichero de configuración de haproxy:

```
imm98@m3-imm98:/etc/haproxy$ sudo systemctl restart haproxy.service
imm98@m3-imm98:/etc/haproxy$
```

Posteriormente, en la sección de comparativas de balanceadores compararemos los balanceadores que hemos implementado arriba con sus diferentes opciones y con los balanceadores implementados con otras herramientas.

4. ESTADÍSTICAS EN HAPROXY

Aparte, vamos a habilitar las estadísticas del balanceador haproxy para que podamos ver mediante una petición http al puerto 9999 autentiicándonos en este caso con nuestro usuario de la UGR como usuario y contraseña. En mi caso imm98:imm98. Para ello debemos añadir en el archivo de configuración /etc/haproxy/haproxy.cfg lo siguiente:

```
imm98@m3-imm98:/etc/haproxy$ cat haproxy.cfg
frontend http-in
    bind *:80
    default_backend balanceo_imm98

backend balanceo_imm98
    server m1 192.168.56.105:8080 maxconn 32
    server m2 192.168.56.104:80 maxconn 32

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 listeners
    stats socket /var/lib/haproxy/stats
    stats timeout 30s
    user haproxy
    group haproxy
    daemon

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

    # Default ciphers to use on SSL-enabled listening sockets.
    # For more information, see ciphers(1SSL). This list is from:
    # https://hynek.me/articles/hardening-your-web-servers-ssl-ciphers/
    # An alternative list with additional directives can be obtained from
    # https://mozilla.github.io/server-side-tls/ssl-config-generator/?server=haproxy
    ssl-default-bind-ciphers ECDH+AESGCM:DH+AESGCM:ECDH+AES256:DH+AES256:ECDH+AES128:DH+AES:RSA+AESGCM
:RSA+AES:!aNULL:!MD5:!DSS
    ssl-default-bind-options no-ssl3

listen stats
    bind *:9999
    mode http
    stats enable
    stats uri /stats
    stats realm HAProxy\ Statistics
    stats auth imm98:imm98
```

Concretamente añadimos en la sección global *stats socket /var/lib/haproxy/stats* y la sección **listen stats** con la información comentada previamente. Reiniciamos el servicio haproxy ya que hemos modificado su archivo de configuración:

```
imm98@m3-imm98:/etc/haproxy$ sudo systemctl restart haproxy.service
imm98@m3-imm98:/etc/haproxy$
```

← → ⓘ No es seguro | 192.168.56.107:9999/stats

Statistics Report for pid 2789

Haciendo zoom:

Una de las opciones avanzadas que he implementado ha sido **stats refresh** que hace que se actualice la página de las estadísticas con frecuencia (en segundos) el número que acompañe a stats refresh. En nuestro caso vamos a poner para que la página se refresque cada 10 segundos:

```
listen stats
    bind *:9999
    mode http
    stats refresh 10s
    stats enable
    stats uri /stats
    stats realm HAProxy\ Statistics
    stats auth imm98:imm98
```

Y comprobaremos que funciona correctamente:



The screenshot shows a web browser window with the address bar displaying '192.168.56.107:9999/stats'. The page title is 'HAProxy version 1.8.8-1ubuntu0.11, released 2020/06/22' and the subtitle is 'Statistics Report for pid 4926'. A section titled '> General process information' is expanded, showing the following details: pid = 4926 (process #1, nbproc = 1, nbthread = 1), uptime = 0d 0h00m56s, system limits: memmax = unlimited; ulimit-n = 4042, maxsock = 4042; maxconn = 2000; maxpipes = 0, current conns = 2; current pipes = 0/0; conn rate = 2/sec, and Running tasks: 1/7; idle = 100 %.

← → ↻ ⓘ No es seguro | 192.168.56.107:9999/stats

Aplicaciones Spotify – Inicio

HAProxy version 1.8.8-1ubuntu0.11, released 2020/06/22

Statistics Report for pid 4926

> General process information

pid = 4926 (process #1, nbproc = 1, nbthread = 1)
uptime = 0d 0h00m56s
system limits: memmax = unlimited; ulimit-n = 4042
maxsock = 4042; maxconn = 2000; maxpipes = 0
current conns = 2; current pipes = 0/0; conn rate = 2/sec
Running tasks: 1/7; idle = 100 %

Vemos que el uptime es de 56 segundos. Y la próxima vez que se actualizan las estadísticas es:



The screenshot shows the same web browser window as before, but the uptime has increased to 0d 0h01m06s. All other statistics remain the same.

← → ↻ ⓘ No es seguro | 192.168.56.107:9999/stats

Aplicaciones Spotify – Inicio

HAProxy version 1.8.8-1ubuntu0.11, released 2020/06/22

Statistics Report for pid 4926

> General process information

pid = 4926 (process #1, nbproc = 1, nbthread = 1)
uptime = 0d 0h01m06s
system limits: memmax = unlimited; ulimit-n = 4042
maxsock = 4042; maxconn = 2000; maxpipes = 0
current conns = 2; current pipes = 0/0; conn rate = 2/sec
Running tasks: 1/7; idle = 100 %

al minuto y 6 segundos. Por lo tanto han pasado exactamente 10 segundos entre dos actualizaciones consecutivas de la página como queríamos.

5. GO-BETWEEN

Otro balanceador de carga software que existe se llama Go-Between. Vamos a instalarlo y configurarlo en nuestra máquina virtual que funciona como balanceadora (m3-imm98).

Para su instalación primero actualizaremos el sistema mediante la orden:

```
sudo apt update
```

```
imm98@m3-imm98:/etc/haproxy$ sudo apt update
```

Posteriormente instalaremos el paquete snapd mediante la orden:

```
sudo apt install snapd
```

```
imm98@m3-imm98:/etc/haproxy$ sudo apt install snapd
Reading package lists... Done
Building dependency tree
Reading state information... Done
snapd is already the newest version (2.48.3+18.04).
snapd set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 5 not upgraded.
```

Y una vez hemos instalado snapd instalamos el balanceador gobetween con la orden

```
sudo snap install gobetween --edge
```

```
imm98@m3-imm98:/etc/haproxy$ sudo snap install gobetween --edge
2021-04-08T08:37:34Z INFO Waiting for automatic snapd restart...
Download snap "core18" (1997) from channel "stable"
```

E iniciamos el servicio mediante la orden

```
sudo snap start gobetween
```

```
imm98@m3-imm98:/etc/haproxy$ sudo snap start gobetween
Started.
```

Y vemos que se inicia correctamente. Con la orden

```
snap services gobetween
```

Vemos si el servicio gobetween está activo

```
imm98@m3-imm98:/etc/haproxy$ snap services gobetween
Service      Startup Current Notes
gobetween.gobetween enabled active -
imm98@m3-imm98:/etc/haproxy$ sudo snap stop gobetween
Stopped.
imm98@m3-imm98:/etc/haproxy$ snap services gobetween
Service      Startup Current Notes
gobetween.gobetween enabled inactive -
imm98@m3-imm98:/etc/haproxy$
```


Para parar el proceso, tenemos la orden:

```
snap stop between
```

como se muestra en la imagen previa.

Para reiniciar el servicio ejecutaremos las orden:

```
sudo snap restart gobetween
```

```
imm98@m3-imm98:/var/snap/gobetween/common$ sudo snap restart gobetween  
Restarted.
```

Una vez que tenemos instalado Go-Between vamos a pasar a configurar el balanceador de carga. El archivo de configuración de Go-Between es **/var/snap/gobetween/common/gobetween.toml**.

```
imm98@m3-imm98:/var/snap/gobetween/common$ ls  
gobetween.toml
```

Vamos a configurar primero el balanceador de carga con el protocolo **Round Robin**. Para ello escribimos en dicho archivo de configuración en la sección llamada [servers] el nombre de nuestro servidor que actuará como balanceador de carga, en mi caso **servers.imm98** y el puerto al que debe recibir la carga que debe balancear, en mi caso he puesto el puerto 3000:

```
[servers.imm98]  
protocol = "tcp"  
bind = "0.0.0.0:3000"  
  
[servers.imm98.discovery]  
kind = "static"  
static_list = [  
    "192.168.56.105:8080 weight=1 priority=1",  
    "192.168.56.104:80 weight=1 priority=1"  
]
```

Como he puesto `weight=1` en ambas direcciones IP estáticas distribuirá la carga por Round Robin. Comprobamos su correcta ejecución mandándole peticiones al puerto 3000:

```
inaki@inaki-N501VW:~$ curl http://192.168.56.107:3000/index.html
<HTML>
  <BODY>
    Web de la pagina m1
  </BODY>
</HTML>
inaki@inaki-N501VW:~$ curl http://192.168.56.107:3000/index.html
<HTML>
  <BODY>
    Web de la pagina m2
  </BODY>
</HTML>
inaki@inaki-N501VW:~$ curl http://192.168.56.107:3000/index.html
<HTML>
  <BODY>
    Web de la pagina m1
  </BODY>
</HTML>
inaki@inaki-N501VW:~$ curl http://192.168.56.107:3000/index.html
<HTML>
  <BODY>
    Web de la pagina m2
  </BODY>
</HTML>
inaki@inaki-N501VW:~$ curl http://192.168.56.107:3000/index.html
<HTML>
  <BODY>
    Web de la pagina m1
  </BODY>
</HTML>
```

Si por el contrario deseamos que distribuya carga por ponderación, al igual que en otros balanceadores previos, lo haremos con la opción **weight**. En este caso, como queremos que la máquina m1-imm98 reciba el doble de carga que m2-imm98 escribiremos:

```
[servers.imm98]
protocol = "tcp"
bind = "0.0.0.0:3000"

[servers.imm98.discovery]
kind = "static"
static_list = [
    "192.168.56.105:8080 weight=2 priority=1",
    "192.168.56.104:80 weight=1 priority=1"
]
```

Posteriormente compararemos los tiempos de ejecución de una carga grande de peticiones al balanceador.

Para hacer un balanceo por IP, mediante la directiva **iphash**. Esto hace que todo el tráfico proveniente de una IP sus peticiones sean atendidas por el mismo servidor:

```
[servers.imm98]
protocol = "tcp"
bind = "0.0.0.0:3000"

[servers.imm98.discovery]
kind = "static"
static_list = [
    "iphash",
    "192.168.56.105:8080 priority=1",
    "192.168.56.104:80 priority=1"
]
```

Por último para que las peticiones se redirijan a la máquina que menos conexiones TCP tenga abiertas en ese momento se utiliza la directiva **leastconn** como se puede ver en la imagen de abajo:

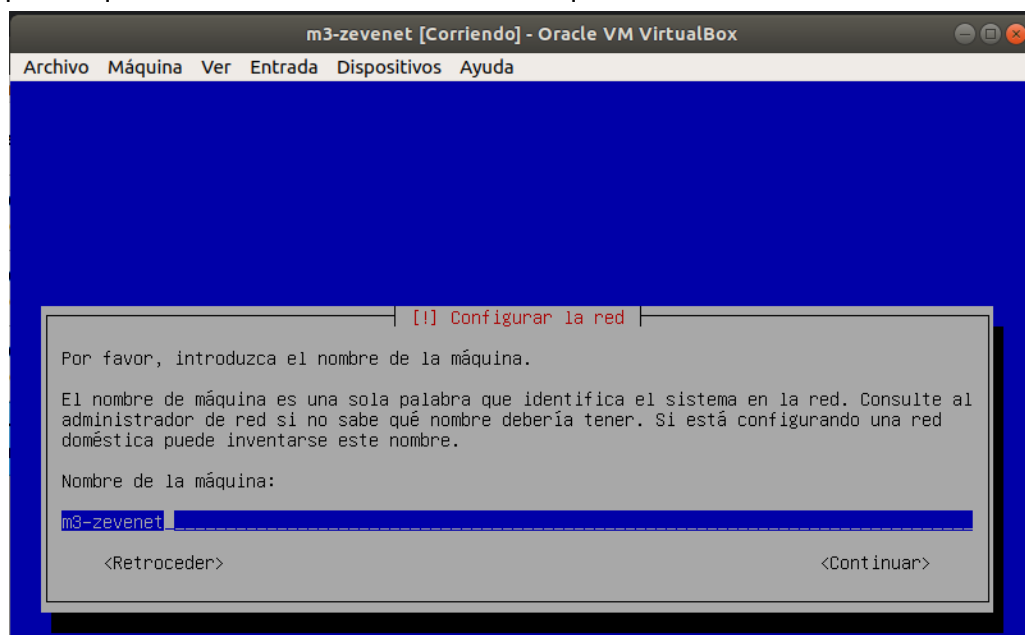
```
[servers.imm98]
protocol = "tcp"
bind = "0.0.0.0:3000"

[servers.imm98.discovery]
kind = "static"
static_list = [
    "leastconn",
    "192.168.56.105:8080 priority=1",
    "192.168.56.104:80 priority=1"
]
```

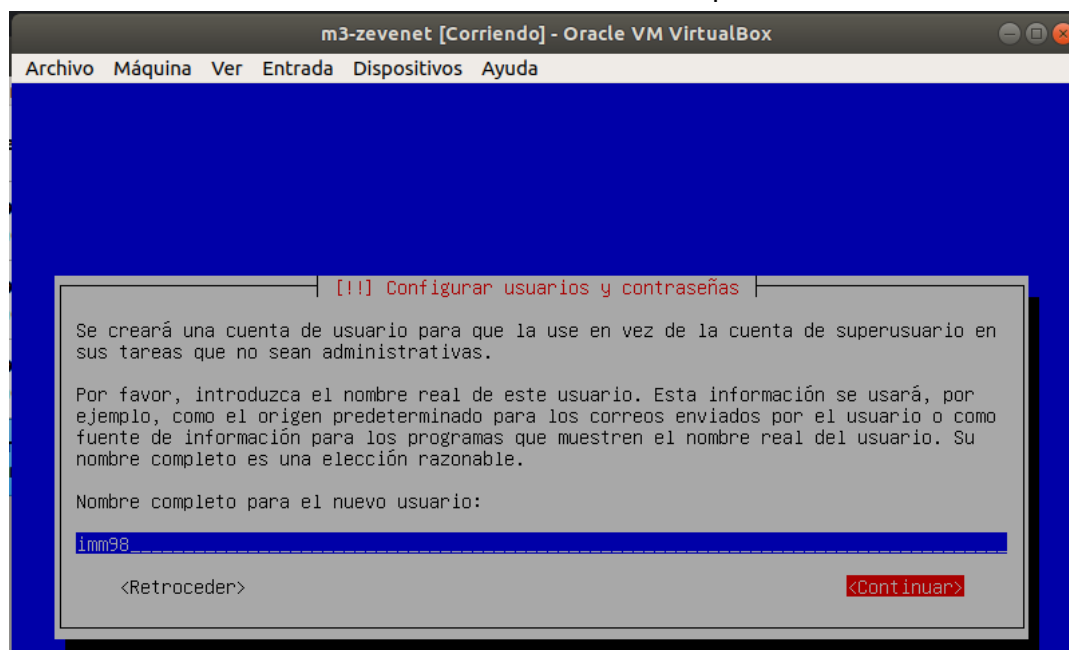
6. ZEVENET

Zevenet es otro balanceador de carga. Para instalárnoslo tendremos que crear una nueva máquina virtual que la he llamado **m3-zevenet** y hacer lo mismo que hicimos al crear las otras máquinas virtuales, es decir, configurar dos adaptadores de red, uno tipo NAT y otro tipo solo anfitrión. Cuando iniciemos la máquina virtual tendremos que ejecutar una iso para instalar zevenet. Concretamente he utilizado la iso **zevenet-ce_v5.11-amd64-v1.iso**

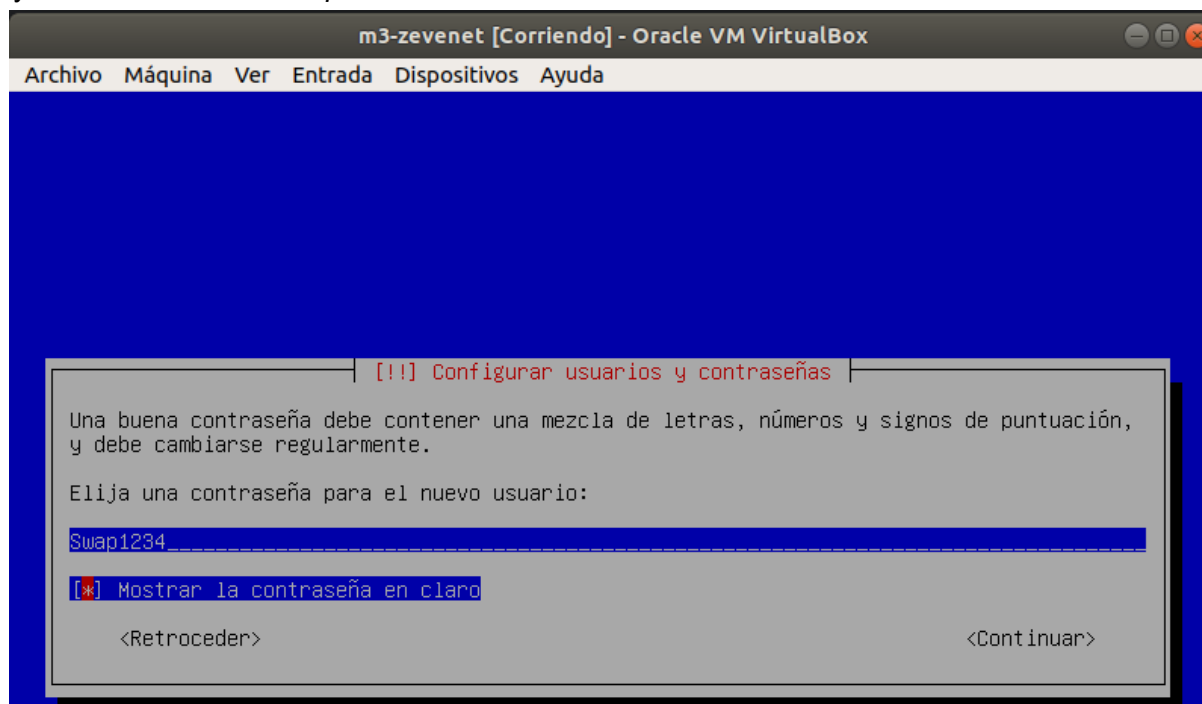
Una vez que hayamos seleccionado la imagen ISO descrita anteriormente nos pedirá que escribamos el nombre de la máquina:



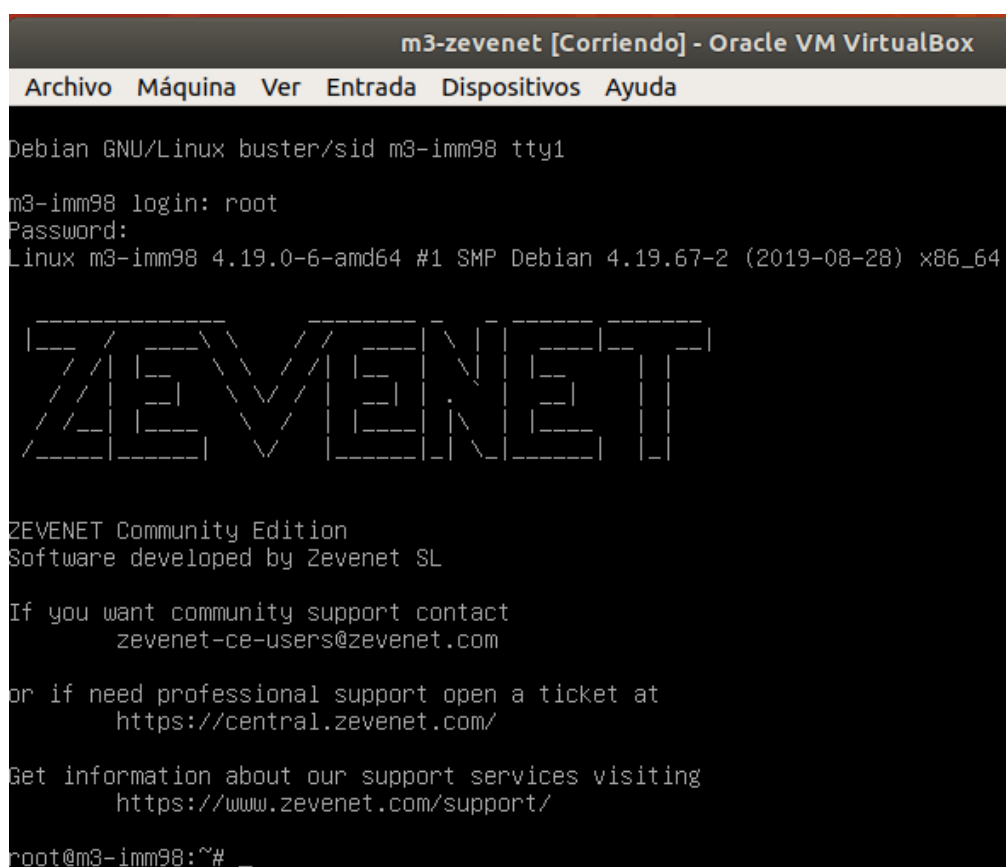
Como nombre de usuario escribiremos como en otras prácticas el usuario de la UGR:



y como contraseña *Swap1234*:



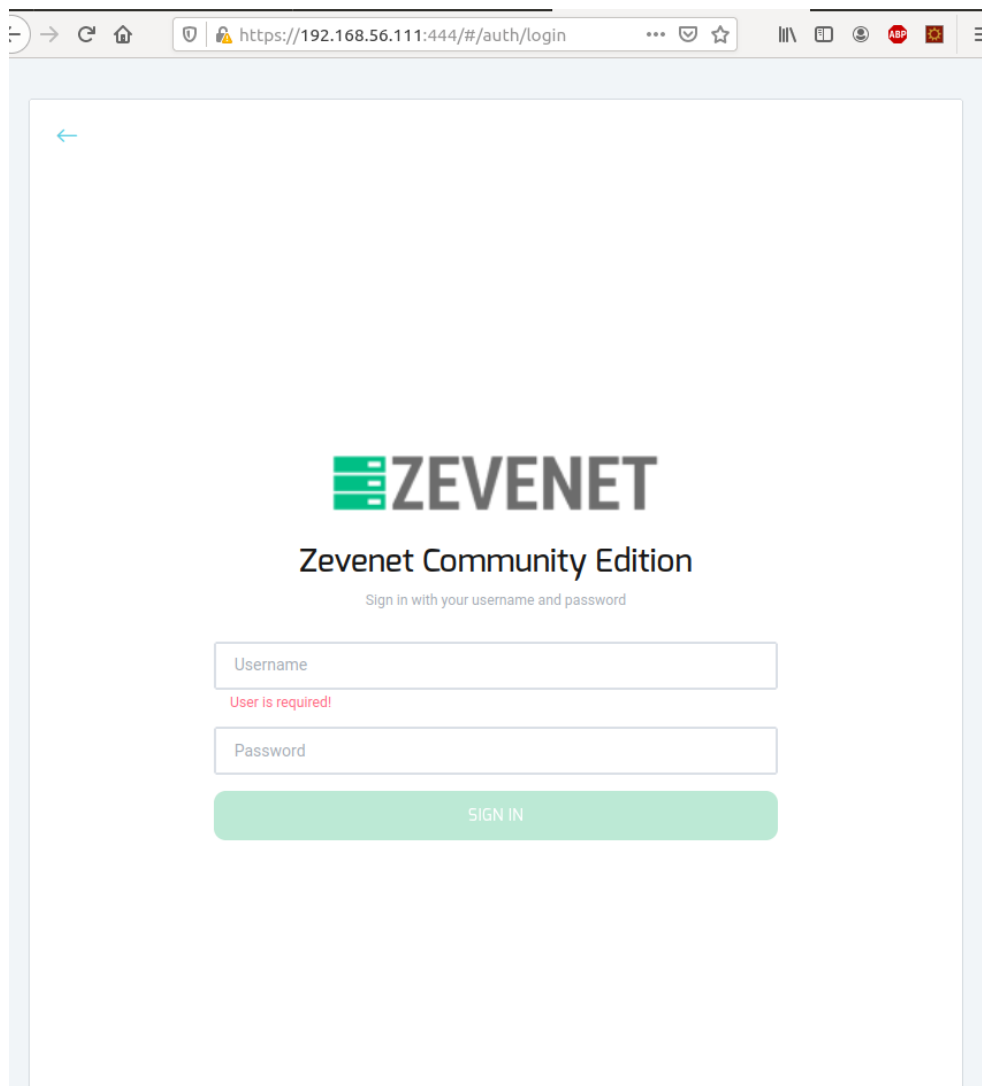
Una vez que lo tengamos instalado nos introduciremos como usuarios root con la contraseña descrita anteriormente:



La IP de esta nueva máquina vemos que es la **192.168.56.111**:

```
root@m3-zevenet:/etc/netplan# ip add
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 08:00:27:14:4a:13 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 brd 10.0.2.255 scope global dynamic enp0s3
        valid_lft 86159sec preferred_lft 86159sec
    inet6 fe80::a00:27ff:fe14:4a13/64 scope link
        valid_lft forever preferred_lft forever
3: enp0s8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 08:00:27:9a:34:d3 brd ff:ff:ff:ff:ff:ff
    inet 192.168.56.111/24 brd 192.168.56.255 scope global enp0s8
        valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fe9a:34d3/64 scope link
        valid_lft forever preferred_lft forever
```

Y tras configurar con netplan las interfaces de red pruebo a conectarme mediante una petición HTTP desde el navegador:

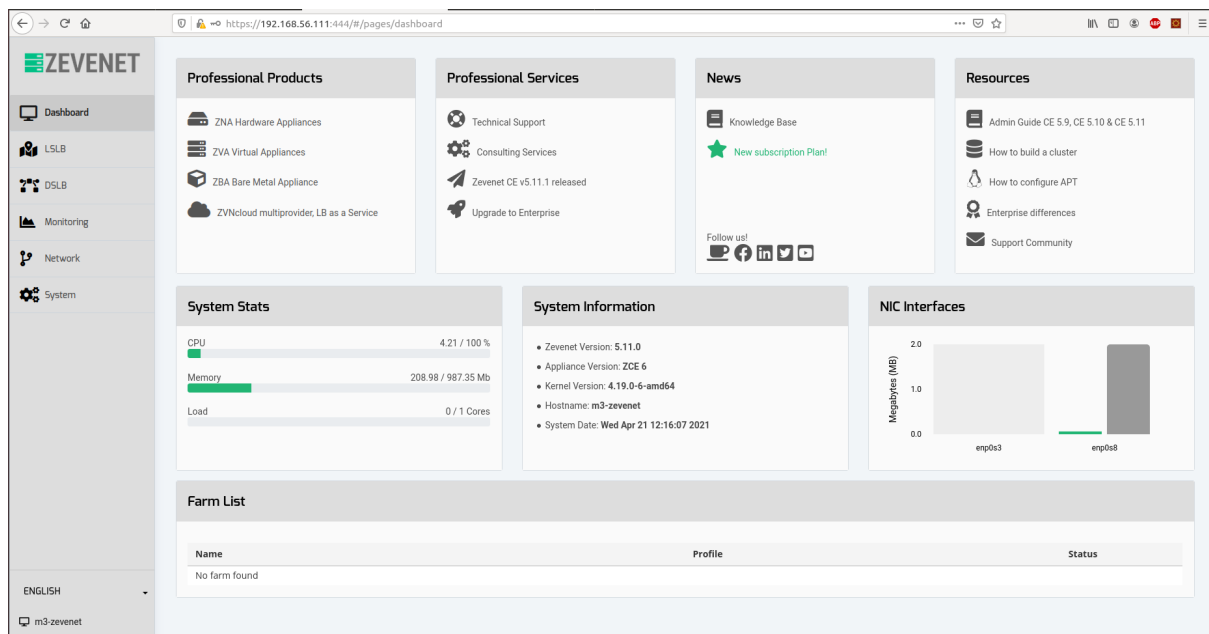


Me introduzco como usuario root con la contraseña *Swap1234*:



The login page features the Zevenet logo at the top, followed by the text "Zevenet Community Edition". Below this is a prompt "Sign in with your username and password". There are two input fields: the first contains the username "root", and the second contains masked characters ".....". A green "SIGN IN" button is positioned below the password field.

Y tras pulsar en el botón *SIGN IN* esto es lo que se nos muestra:

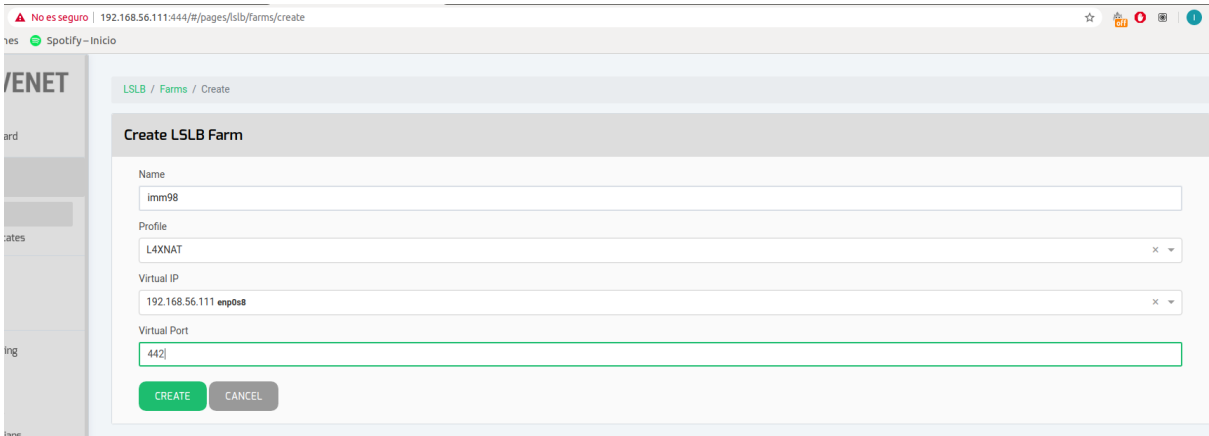


The dashboard is divided into several sections. On the left is a sidebar with navigation links: Dashboard, LSLB, DSLB, Monitoring, Network, and System. The main content area includes:

- Professional Products:** ZNA Hardware Appliances, ZVA Virtual Appliances, ZBA Bare Metal Appliance, and ZVncloud multiprovider, LB as a Service.
- Professional Services:** Technical Support, Consulting Services, Zevenet CE v5.11.1 released, and Upgrade to Enterprise.
- News:** Knowledge Base and a "New subscription Plan!" announcement.
- Resources:** Admin Guide CE 5.9, CE 5.10 & CE 5.11, How to build a cluster, How to configure APT, Enterprise differences, and Support Community.
- System Stats:** CPU (4.21 / 100 %), Memory (208.98 / 987.35 Mb), and Load (0 / 1 Cores).
- System Information:** Zevenet Version: 5.11.0, Appliance Version: ZCE 6, Kernel Version: 4.19.0-6-amd64, Hostname: m3-zevenet, and System Date: Wed Apr 21 12:16:07 2021.
- NIC Interfaces:** A bar chart showing memory usage for em0s3 and em0s8.
- Farm List:** A table with columns Name, Profile, and Status, currently showing "No farm found".

At the bottom left, there is a language selector set to "ENGLISH" and a user profile icon labeled "m3-zevenet".

Para configurar nuestro balanceador de carga, primeramente crearemos la una Granja LSLB (la creamos pulsando en el botón LSLB de la izquierda). La llamaré **imm98** como mi usuario de la UGR y le asociaré la IP de la máquina m3-zevenet, es decir, la **192.168.56.111** El puerto de la granja será el 442 como se puede ver en la captura de abajo:



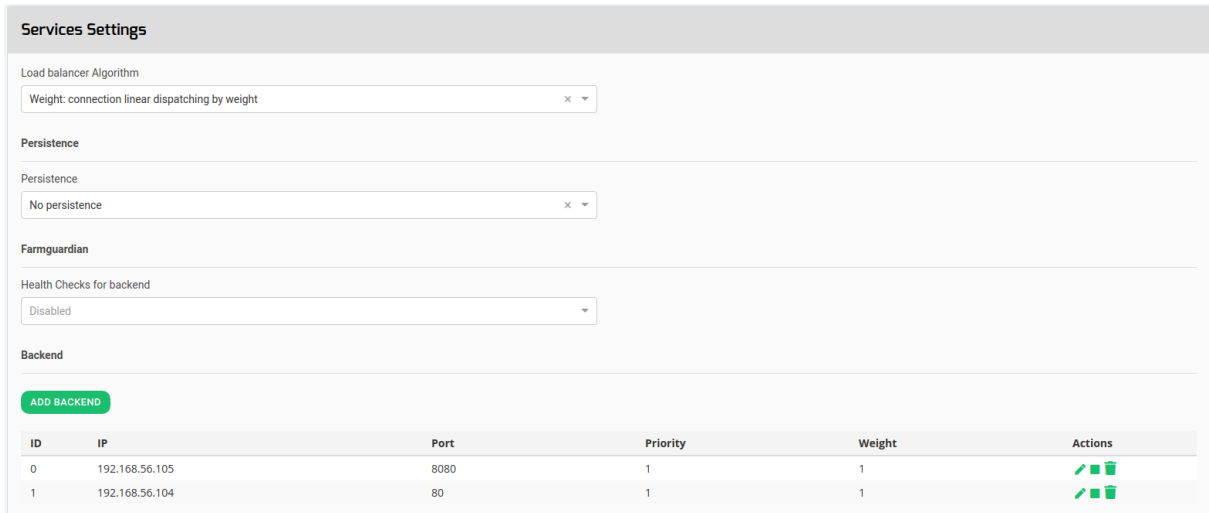
Vemos que se ha creado correctamente:

Name	Profile	Virtual IP	Virtual Port	Status	Actions
imm98	l4xnat	192.168.56.111	442		

Para configurar la granja pulsaremos el botón que tiene un dibujo de un lápiz, en la columna *Actions*. Ahora ya introduciremos las IPs de las máquinas a las que queremos balancear la carga que llega a la granja, es decir, la IP de **m1-imm98** y de **m2-imm98** que son las que se indican en la tabla de abajo:

m1-imm98	192.168.56.105
m2-imm98	192.168.56.104

siendo 8080 el puerto HTTP de m1-imm98 y el puerto 80 el de m2-imm98. También, en la parte superior, seleccionaremos el algoritmo de balanceo:



ID	IP	Port	Priority	Weight	Actions
0	192.168.56.105	8080	1	1	
1	192.168.56.104	80	1	1	

Por tanto ya tenemos configurado nuestro balanceador. Vamos a probarlo:

```
inaki@inaki-N501VW:~$ curl http://192.168.56.111:442/index.html
<HTML>
  <BODY>
    Web de la pagina m1
  </BODY>
</HTML>
inaki@inaki-N501VW:~$ curl http://192.168.56.111:442/index.html
<HTML>
  <BODY>
    Web de la pagina m1
  </BODY>
</HTML>
inaki@inaki-N501VW:~$ curl http://192.168.56.111:442/index.html
<HTML>
  <BODY>
    Web de la pagina m1
  </BODY>
</HTML>
inaki@inaki-N501VW:~$ curl http://192.168.56.111:442/index.html
<HTML>
  <BODY>
    Web de la pagina m2
  </BODY>
</HTML>
inaki@inaki-N501VW:~$ curl http://192.168.56.111:442/index.html
<HTML>
  <BODY>
    Web de la pagina m2
  </BODY>
</HTML>
```

Vemos que funciona correctamente. Por tanto vamos a proceder a que balancee siguiendo el algoritmo Round Robin, que es lo que nos pedía la práctica:

Round Robin: Sequential backend selection x ▾

Persistence

Persistence

No persistence x ▾







Farmguardian

Health Checks for backend

Disabled ▾

Backend

ADD BACKEND

ID	IP	Port	Priority	Weight	Actions
0	192.168.56.105	8080	1	1	  
1	192.168.56.104	80	1	1	  

Vemos que distribuye la carga entre las dos máquinas siguiendo el algoritmo Round Robin como era de prever:

```
inaki@inaki-N501VW:~$ curl http://192.168.56.111:442/index.html
<HTML>
  <BODY>
    Web de la pagina m1
  </BODY>
</HTML>
inaki@inaki-N501VW:~$ curl http://192.168.56.111:442/index.html
<HTML>
  <BODY>
    Web de la pagina m2
  </BODY>
</HTML>
inaki@inaki-N501VW:~$ curl http://192.168.56.111:442/index.html
<HTML>
  <BODY>
    Web de la pagina m1
  </BODY>
</HTML>
inaki@inaki-N501VW:~$ curl http://192.168.56.111:442/index.html
<HTML>
  <BODY>
    Web de la pagina m2
  </BODY>
</HTML>
inaki@inaki-N501VW:~$ curl http://192.168.56.111:442/index.html
<HTML>
  <BODY>
    Web de la pagina m1
  </BODY>
</HTML>
```

También he configurado la granja para que distribuya la carga por ponderación de forma que el servidor m1-imm98 reciba el doble de carga que m2-imm98. Para ello a m1-imm98 le asocio **Weight 2** como se puede ver en la imagen:

Services Settings

Load balancer Algorithm

Weight: connection linear dispatching by weight

Persistence

Persistence

No persistence





Farmguardian

Health Checks for backend

Disabled

Backend

ADD BACKEND

ID	IP	Port	Priority	Weight	Actions
0	192.168.56.105	8080	1	2	 
1	192.168.56.104	80	1	1	 

SUBMIT

Y vemos mediante el comando curl que parece que hace la distribución de carga con este criterio:

```
inaki@inaki-N501VW:~$ curl http://192.168.56.111:442/index.html
<HTML>
  <BODY>
    Web de la pagina m1
  </BODY>
</HTML>
inaki@inaki-N501VW:~$ curl http://192.168.56.111:442/index.html
<HTML>
  <BODY>
    Web de la pagina m1
  </BODY>
</HTML>
inaki@inaki-N501VW:~$ curl http://192.168.56.111:442/index.html
<HTML>
  <BODY>
    Web de la pagina m2
  </BODY>
</HTML>
inaki@inaki-N501VW:~$ curl http://192.168.56.111:442/index.html
<HTML>
  <BODY>
    Web de la pagina m1
  </BODY>
</HTML>
inaki@inaki-N501VW:~$ curl http://192.168.56.111:442/index.html
<HTML>
  <BODY>
    Web de la pagina m1
  </BODY>
</HTML>
inaki@inaki-N501VW:~$ curl http://192.168.56.111:442/index.html
<HTML>
  <BODY>
    Web de la pagina m2
  </BODY>
</HTML>
inaki@inaki-N501VW:~$ curl http://192.168.56.111:442/index.html
<HTML>
  <BODY>
    Web de la pagina m1
  </BODY>
</HTML>
inaki@inaki-N501VW:~$ curl http://192.168.56.111:442/index.html
<HTML>
  <BODY>
    Web de la pagina m1
  </BODY>
</HTML>
```

Las distintos algoritmos de balanceo que podemos utilizar con zevenet son todos los que se indican a continuación:

Round Robin: Sequential backend selection

Weight: connection linear dispatching by weight

Source Hash: Hash per Source IP and Source Port

Simple Source Hash: Hash per Source IP only

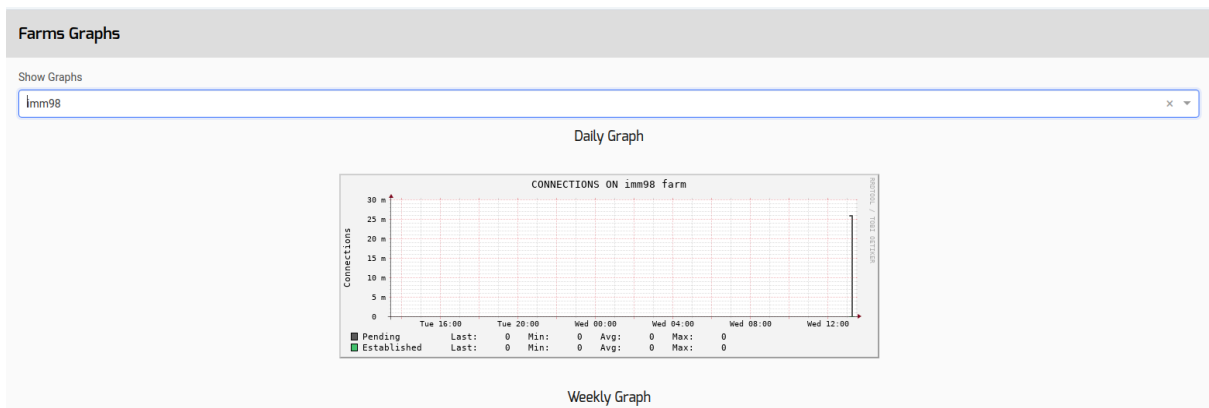
Symmetric Hash: Round trip hash per IP and Port

Round Robin: Sequential backend selection

además de poder hacer todas las combinaciones de prioridades y pesos de las máquinas que queramos:

ID	IP	Port	Priority	Weight	Actions
0	192.168.56.105	8080	2	1	 
1	192.168.56.104	80	1	1	 

También podemos ver estadísticas diarias, mensuales y anuales de las conexiones que se realizan a la granja Web



7. POUND

Otro balanceador de carga que vamos a utilizar se llama Pound. Ya que el balanceador de carga Pound ya no se encuentra en los repositorios de ubuntu vamos a descargarlo de un repositorio de github. Descargamos el repositorio con la orden:

```
wget http://kr.archive.ubuntu.com/ubuntu/pool/universe/p/pound/pound_2.6-6.1_amd64.deb
```

```
imm98@m3-imm98:~$ wget http://kr.archive.ubuntu.com/ubuntu/pool/universe/p/pound/pound_2.6-6.1_amd64.deb
--2021-04-14 10:57:03-- http://kr.archive.ubuntu.com/ubuntu/pool/universe/p/pound/pound_2.6-6.1_amd64.deb
Resolving kr.archive.ubuntu.com (kr.archive.ubuntu.com)... 91.189.91.39, 91.189.88.152, 91.189.91.38, ...
Connecting to kr.archive.ubuntu.com (kr.archive.ubuntu.com)|91.189.91.39|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 92604 (90K) [application/x-debian-package]
Saving to: 'pound_2.6-6.1_amd64.deb'

pound_2.6-6.1_amd64 100%[=====] 90.43K 329KB/s in 0.3s

2021-04-14 10:57:04 (329 KB/s) - 'pound_2.6-6.1_amd64.deb' saved [92604/92604]
```

Lo instalamos mediante la orden:

```
sudo dpkg -i pound_2.6-6.1_amd64.deb
```

```
imm98@m3-imm98:~$ sudo dpkg -i pound_2.6-6.1_amd64.deb
[sudo] password for imm98:
Selecting previously unselected package pound.
(Reading database ... 105165 files and directories currently installed.)
Preparing to unpack pound_2.6-6.1_amd64.deb ...
Unpacking pound (2.6-6.1) ...
Setting up pound (2.6-6.1) ...
Processing triggers for man-db (2.8.3-2ubuntu0.1) ...
Processing triggers for systemd (237-3ubuntu10.45) ...
Processing triggers for ureadahead (0.100.0-21) ...
```

Una vez que lo tenemos instalado probamos a iniciarlo:

```
pound
```

```
imm98@m3-imm98:~$ pound
starting...
imm98@m3-imm98:~$
```

El archivo de configuración de Pound va a ser el archivo `/etc/pound/pound.cfg` como podemos ver en la captura:

```
imm98@m3-imm98:/etc/pound$ ls
pound.cfg
```

Podemos reiniciar el servicio mediante la orden:

```
sudo systemctl restart pound
```

```
imm98@m3-imm98:/etc/default$ sudo systemctl restart pound
```

y vemos si el servicio de Pound está activo con el comando

```
sudo systemctl status pound.service
```

```
imm98@m3-imm98:/etc/default$ sudo systemctl status pound.service
● pound.service - LSB: reverse proxy and load balancer
   Loaded: loaded (/etc/init.d/pound; generated)
   Active: active (running) since Wed 2021-04-14 11:26:39 UTC; 5s ago
     Docs: man:systemd-sysv-generator(8)
  Process: 1953 ExecStop=/etc/init.d/pound stop (code=exited, status=0/SUCCESS)
  Process: 1974 ExecStart=/etc/init.d/pound start (code=exited, status=0/SUCCESS)
    Tasks: 132 (limit: 1107)
   CGroup: /system.slice/pound.service
           └─1980 /usr/sbin/pound
             1981 /usr/sbin/pound

Apr 14 11:26:39 m3-imm98 systemd[1]: Stopped LSB: reverse proxy and load balance
Apr 14 11:26:39 m3-imm98 systemd[1]: Starting LSB: reverse proxy and load balance
Apr 14 11:26:39 m3-imm98 pound[1974]: * Starting reverse proxy and load balance
Apr 14 11:26:39 m3-imm98 pound[1974]: starting...
Apr 14 11:26:39 m3-imm98 pound[1974]: ...done.
Apr 14 11:26:39 m3-imm98 systemd[1]: Started LSB: reverse proxy and load balance
lines 1-17/17 (END)
```

Ahora vamos a modificar el archivo de configuración /etc/pound/pound.cfg para que Pound distribuya carga a las máquinas m1-imm98 y m2-imm98. Primeramente Pound escuchará peticiones HTTP desde el puerto 80 y las redistribuirá mediante el algoritmo Round Robin a las máquinas m1-imm98 y m2-imm98. Para ello escribimos:

```
ListenHTTP
    Address 192.168.56.107
    Port    80

    ## allow PUT and DELETE also (by default only GET, POST and HEAD)?:
    xHTTP   0

    Service
        BackEnd
            Address 192.168.56.105
            Port    8080
        End
        BackEnd
            Address 192.168.56.104
            Port    80
        End
    End
End
```

y vemos con la orden curl que efectivamente distribuye la carga entre las máquinas m1-imm98 y m2-imm98:

```
inaki@inaki-N501VW:~$ curl http://192.168.56.107/index.html
<HTML>
  <BODY>
    Web de la pagina m1
  </BODY>
</HTML>
inaki@inaki-N501VW:~$ curl http://192.168.56.107/index.html
<HTML>
  <BODY>
    Web de la pagina m2
  </BODY>
</HTML>
inaki@inaki-N501VW:~$ curl http://192.168.56.107/index.html
<HTML>
  <BODY>
    Web de la pagina m2
  </BODY>
</HTML>
inaki@inaki-N501VW:~$ curl http://192.168.56.107/index.html
<HTML>
  <BODY>
    Web de la pagina m1
  </BODY>
</HTML>
```

También podríamos haber escrito que escuchara en un puerto distinto al puerto 80. Por ejemplo vamos a poner que escuche las peticiones HTTP por el puerto 8081:

```
ListenHTTP
  Address 192.168.56.107
  Port    8081

  ## allow PUT and DELETE also (by default only
  xHTTP    0

  Service
    BackEnd
      Address 192.168.56.105
      Port    8080
    End
    BackEnd
      Address 192.168.56.104
      Port    80
    End
  End
End
```

Como habíamos hecho antes, vamos a ver mediante la orden curl que efectivamente distribuye bien la carga que le llega por el puerto 8081:

```
inaki@inaki-N501VW:~$ curl http://192.168.56.107:8081/index.html
<HTML>
  <BODY>
    Web de la pagina m2
  </BODY>
</HTML>
inaki@inaki-N501VW:~$ curl http://192.168.56.107:8081/index.html
<HTML>
  <BODY>
    Web de la pagina m1
  </BODY>
</HTML>
inaki@inaki-N501VW:~$ curl http://192.168.56.107:8081/index.html
<HTML>
  <BODY>
    Web de la pagina m2
  </BODY>
</HTML>
```

En la siguiente sección analizaremos los resultados en términos de tiempo al someter a carga a este balanceador Pound.

Una de las opciones que nos da Pound es dar una prioridad a cada máquina a la que queremos distribuir la carga. Esto se hace con la opción **Priority**. Le vamos a dar una prioridad de 2 a la máquina m1-imm98 y una prioridad de 1 a m2-imm98

```
ListenHTTP
  Address 192.168.56.107
  Port    80

  ## allow PUT and DELETE also (by default only GET, POST and HEAD)?:
  xHTTP    0

  Service
    BackEnd
      Address 192.168.56.105
      Port    8080
      Priority 2
    End
    BackEnd
      Address 192.168.56.104
      Port    80
      Priority 1
    End
  End
End
```


Como la máquina m2-imm98 tiene mayor prioridad, la mayoría de peticiones que le llegan al balanceador las distribuirá a la máquina m2-imm98 como vamos a comprobar con la orden curl:

```
inaki@inaki-N501VW:~$ curl http://192.168.56.107/index.html
<HTML>
  <BODY>
    Web de la pagina m1
  </BODY>
</HTML>
inaki@inaki-N501VW:~$ curl http://192.168.56.107/index.html
<HTML>
  <BODY>
    Web de la pagina m1
  </BODY>
</HTML>
inaki@inaki-N501VW:~$ curl http://192.168.56.107/index.html
<HTML>
  <BODY>
    Web de la pagina m2
  </BODY>
</HTML>
inaki@inaki-N501VW:~$ curl http://192.168.56.107/index.html
<HTML>
  <BODY>
    Web de la pagina m1
  </BODY>
</HTML>
inaki@inaki-N501VW:~$ curl http://192.168.56.107/index.html
<HTML>
  <BODY>
    Web de la pagina m2
  </BODY>
</HTML>
inaki@inaki-N501VW:~$ curl http://192.168.56.107/index.html
<HTML>
  <BODY>
    Web de la pagina m1
  </BODY>
</HTML>
inaki@inaki-N501VW:~$ curl http://192.168.56.107/index.html
<HTML>
  <BODY>
    Web de la pagina m1
  </BODY>
</HTML>
inaki@inaki-N501VW:~$ curl http://192.168.56.107/index.html
<HTML>
  <BODY>
    Web de la pagina m1
  </BODY>
</HTML>
inaki@inaki-N501VW:~$
```

Otra opción adicional que tenemos con el balanceador de carga Pound es **Client** que indica el número de segundos que Pound va a esperar a una determinada petición a un cliente. Cuando pasen ese número de segundos sin respuesta de una máquina, que por defecto son 10, Pound cerrará la conexión con ese cliente. Lo hemos puesto a 2 segundos ese tiempo de espera y funciona correctamente como se ve en la captura de abajo:

```
ListenHTTP
  Address 192.168.56.107
  Port    80
  Client  2

  ## allow PUT and DELETE also (by default only GET, POST and HEAD)?:
  xHTTP   0

  Service
    BackEnd
      Address 192.168.56.105
      Port    8080
      Priority 2
    End
    BackEnd
      Address 192.168.56.104
      Port    80
      Priority 1
    End
  End
End
```

8. Someter a carga la granja web con AB

Para someter a carga HTTP a la máquina m3-imm98 hemos utilizado la herramienta Apache Benchmark **ab**. Para el análisis comparativo que hemos hecho de los distintos balanceadores hemos utilizado las opciones **-n** y **-c** que indican respectivamente el número de peticiones HTTP y con **-c** el número de conexiones concurrentes. Por lo que un ejemplo sería:

```
ab -n 10000 -c 10 http://192.168.56.107/index.html
```

```
lnaki@lnaki-N501VW:~$ ab -n 10000 -c 10 http://192.168.56.107/index.html
This is ApacheBench, Version 2.3 <$Revision: 1807734 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 192.168.56.107 (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:      Apache/2.4.29
Server Hostname:      192.168.56.107
Server Port:          80

Document Path:        /index.html
Document Length:      54 bytes

Concurrency Level:    10
Time taken for tests:  3.974 seconds
Complete requests:    10000
Failed requests:       0
Total transferred:    3000000 bytes
HTML transferred:     540000 bytes
Requests per second:  2516.36 [#/sec] (mean)
Time per request:      3.974 [ms] (mean)
Time per request:      0.397 [ms] (mean, across all concurrent requests)
Transfer rate:         737.21 [Kbytes/sec] received


Connection Times (ms)
              min      mean[+/-sd] median   max
Connect:        0        0   0.2      0      5
Processing:      1        4   1.4      4     21
Waiting:         1        4   1.3      3     15
Total:           1        4   1.4      4     22


Percentage of the requests served within a certain time (ms)
 50%      4
 66%      4
 75%      5
 80%      5
 90%      6
 95%      6
 98%      8
 99%      9
100%     22 (longest request)
```

que vemos que tarda en ejecutarse 3.974 segundos

Con la opción `-t seconds` indicamos que realice el máximo número de peticiones en ese número de segundos que le indicamos. Por ejemplo:

`ab -c 10 -t 5 http://192.168.56.107/index.html`

mandará todas las peticiones HTTP que “pueda” en 5 segundos

```
inaki@inaki-NS01VW:~$ ab -c 10 -t 5 http://192.168.56.107/index.html
This is ApacheBench, Version 2.3 <$Revision: 1807734 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 192.168.56.107 (be patient)
Completed 5000 requests
Completed 10000 requests
Completed 15000 requests
Completed 20000 requests
Finished 22812 requests


Server Software:      nginx/1.14.0
Server Hostname:      192.168.56.107
Server Port:          80

Document Path:        /index.html
Document Length:      182 bytes

Concurrency Level:    10
Time taken for tests:  5.000 seconds
Complete requests:    22812
Failed requests:       0
Non-2xx responses:    22812
Total transferred:    7824516 bytes
HTML transferred:     4151784 bytes
Requests per second:  4562.32 [#/sec] (mean)
Time per request:     2.192 [ms] (mean)
Time per request:     0.219 [ms] (mean, across all concurrent requests)
Transfer rate:        1528.20 [Kbytes/sec] received
```

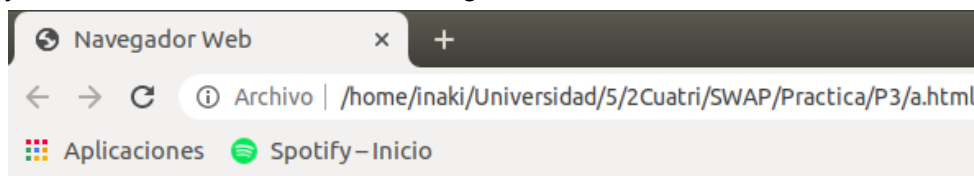
Con la opción **-w** reproduce los datos en formato HTML. Aquí un ejemplo de la ejecución de la orden:

```
ab -c 10 -t 5 -w http://192.168.56.107/index.html
```

```
inaki@inaki-N501VW:~$ ab -c 10 -t 5 -w http://192.168.56.107/index.html
<p>
This is ApacheBench, Version 2.3 <i>&lt;$Revision: 1807734 $&gt;</i><br>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/<br>
Licensed to The Apache Software Foundation, http://www.apache.org/<br>
</p>
<p>
Completed 5000 requests
Completed 10000 requests
Completed 15000 requests
Completed 20000 requests
Finished 21758 requests

<table>
<tr><th colspan=2 bgcolor=white>Server Software:</th><td colspan=2 bgcolor=white>nginx/1.14.0</td></tr>
<tr><th colspan=2 bgcolor=white>Server Hostname:</th><td colspan=2 bgcolor=white>192.168.56.107</td></tr>
<tr><th colspan=2 bgcolor=white>Server Port:</th><td colspan=2 bgcolor=white>80</td></tr>
<tr><th colspan=2 bgcolor=white>Document Path:</th><td colspan=2 bgcolor=white>/index.html</td></tr>
<tr><th colspan=2 bgcolor=white>Document Length:</th><td colspan=2 bgcolor=white>182 bytes</td></tr>
<tr><th colspan=2 bgcolor=white>Concurrency Level:</th><td colspan=2 bgcolor=white>10</td></tr>
<tr><th colspan=2 bgcolor=white>Time taken for tests:</th><td colspan=2 bgcolor=white>5.000 seconds</td></tr>
<tr><th colspan=2 bgcolor=white>Complete requests:</th><td colspan=2 bgcolor=white>21758</td></tr>
<tr><th colspan=2 bgcolor=white>Failed requests:</th><td colspan=2 bgcolor=white>0</td></tr>
<tr><th colspan=2 bgcolor=white>Non-2xx responses:</th><td colspan=2 bgcolor=white>21758</td></tr>
<tr><th colspan=2 bgcolor=white>Total transferred:</th><td colspan=2 bgcolor=white>7462994 bytes</td></tr>
<tr><th colspan=2 bgcolor=white>HTML transferred:</th><td colspan=2 bgcolor=white>3959956 bytes</td></tr>
<tr><th colspan=2 bgcolor=white>Requests per second:</th><td colspan=2 bgcolor=white>4351.57</td></tr>
<tr><th colspan=2 bgcolor=white>Transfer rate:</th><td colspan=2 bgcolor=white>1457.61 kb/s received</td></tr>
<tr><th colspan=4>Connection Times (ms)</th></tr>
<tr><th colspan=4>min</th><th colspan=4>avg</th><th colspan=4>max</th></tr>
<tr><th colspan=4>Connect:</th><td colspan=2 bgcolor=white>0</td><td colspan=2 bgcolor=white>0</td><td colspan=2 bgcolor=white>1</td></tr>
<tr><th colspan=4>Processing:</th><td colspan=2 bgcolor=white>1</td><td colspan=2 bgcolor=white>2</td><td colspan=2 bgcolor=white>3076</td></tr>
<tr><th colspan=4>Total:</th><td colspan=2 bgcolor=white>1</td><td colspan=2 bgcolor=white>2</td><td colspan=2 bgcolor=white>3077</td></tr>
</table>
```

y el resultado mostrado en un navegador web sería:



Server Software: nginx/1.14.0
Server Hostname: 192.168.56.107
Server Port: 80
Document Path: /index.html
Document Length: 182 bytes
Concurrency Level: 10
Time taken for tests: 5.000 seconds
Complete requests: 21758
Failed requests: 0
Non-2xx responses: 21758
Total transferred: 7462994 bytes
HTML transferred: 3959956 bytes
Requests per second: 4351.57
Transfer rate: 1457.61 kb/s received
Connection Times (ms)

	min	avg	max
Connect:	0	0	1
Processing:	1	2	3076
Total:	1	2	3077

9. Análisis comparativo de distintos balanceados en base a la carga con AB

Para el análisis comparativo en base a la carga con AB hemos realizado, como se ha comentado en el apartado anterior, el lanzamiento a la máquina m3-imm98 de una serie de peticiones HTTP desde nuestra máquina host mediante la orden AB. Esta máquina m3-imm98 redistribuirá la carga entre las máquinas m1-imm98 y m2-imm98 según el algoritmo que hayamos seleccionado (Round Robin, ponderación....)

Para poder compararlos, hemos realizado siempre las peticiones a m3-imm98 con el mismo número de peticiones:

- 10.000 peticiones
- 50.000 peticiones

y con la misma concurrencia (-c 10) para poder compararlas con cierta coherencia . He realizado entre 2 y 4 mediciones para cada opción de balanceo ya que la media es más precisa que una simple medición

a. NGINX

i. Round Robin

1. 10.000 peticiones

```
inaki@inaki-NS01W:~$ ab -n 10000 -c 10 http://192.168.56.107/index.html
This is ApacheBench, Version 2.3 <$Revision: 1807734 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 192.168.56.107 (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/1.14.0
Server Hostname:      192.168.56.107
Server Port:          80

Document Path:        /index.html
Document Length:      54 bytes

Concurrency Level:    10
Time taken for tests:  4.701 seconds
Complete requests:    10000
Failed requests:       0
Total transferred:    2990000 bytes
HTML transferred:     540000 bytes
Requests per second:  2127.17 [#/sec] (mean)
Time per request:     4.701 [ms] (mean)
Time per request:     0.470 [ms] (mean, across all concurrent requests)
Transfer rate:        621.12 [Kbytes/sec] received

Connection Times (ms)
  min   mean[+/-sd] median   max
Connect:    0      0   0.1      0     4
Processing:  1      5   0.9      4    16
Waiting:    1      5   0.9      4    16
Total:      1      5   0.9      5    17
WARNING: The median and mean for the processing time are not within a normal deviation
These results are probably not that reliable.
WARNING: The median and mean for the waiting 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%    5
 66%    5
 75%    5
 80%    5
 90%    6
 95%    6
```

```
Time taken for tests: 4.730 seconds
```

```
Time taken for tests: 4.787 seconds
```

10.000

4.730
4.787
4.701
Media: 4.74

2. 50.000 peticiones

```
Time taken for tests: 23.522 seconds  
Complete requests: 50000
```

```
Time taken for tests: 23.615 seconds  
Complete requests: 50000
```

50.000

23.522
23.615
Media: 23.66

ii. Ponderación

1. 10.000 peticiones

```
lnaki@lnaki-N501VM:~$ ab -n 10000 -c 10 http://192.168.56.107/index.html
This is ApacheBench, Version 2.3 <$Revision: 1807734 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 192.168.56.107 (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/1.14.0
Server Hostname:      192.168.56.107
Server Port:          80

Document Path:        /index.html
Document Length:      54 bytes

Concurrency Level:    10
Time taken for tests:  4.602 seconds
Complete requests:    10000
Failed requests:      0
Total transferred:    2990000 bytes
HTML transferred:     540000 bytes
Requests per second:  2173.13 [#/sec] (mean)
Time per request:     4.602 [ms] (mean)
Time per request:     0.460 [ms] (mean, across all concurrent requests)
Transfer rate:        634.54 [Kbytes/sec] received

Connection Times (ms)
              min    mean[+/-sd] median    max
Connect:      0      0   0.1      0      4
Processing:   1      4   0.9      4     26
Waiting:      1      4   0.9      4     26
Total:        1      5   0.9      4     26
WARNING: The median and mean for the total 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%    4
 66%    5
 75%    5
 80%    5
 90%    5
 95%    6
 98%    7
```

```
Concurrency Level:      10
Time taken for tests:    4.616 seconds
Complete requests:      10000
```

10.000

4.602
4.616
Media: 4.609

2. 50.000 peticiones

```
Time taken for tests:    23.479 seconds
Complete requests:      50000
```

```
Time taken for tests:    24.441 seconds
Complete requests:      50000
```

50.000

23.479

24.441

iii. Keepalive

1. 10.000 peticiones

```
inaki@inaki-N501VM:~$ ab -n 10000 -c 10 http://192.168.56.107/index.html
This is ApacheBench, Version 2.3 <$Revision: 1807734 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 192.168.56.107 (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/1.14.0
Server Hostname:      192.168.56.107
Server Port:          80


Document Path:        /index.html
Document Length:      54 bytes


Concurrency Level:    10
Time taken for tests:  3.192 seconds
Complete requests:    10000
Failed requests:      0
Total transferred:    2990000 bytes
HTML transferred:     540000 bytes
Requests per second:  3132.49 [#/sec] (mean)
Time per request:     3.192 [ms] (mean)
Time per request:     0.319 [ms] (mean, across all concurrent requests)
Transfer rate:        914.66 [Kbytes/sec] received


Connection Times (ms)
              min      mean[+/-sd] median   max
Connect:        0        0   0.2      0      5
Processing:      1        3   0.8      3     16
Waiting:         1        3   0.8      3     16
Total:           1        3   0.8      3     16


Percentage of the requests served within a certain time (ms)
 50%    3
 66%    3
 75%    3
 80%    3
 90%    4
 95%    4
 98%    5
 99%    7
100%   16 (longest request)
```

10.000

3.192

2. 50.000 peticiones

```
Time taken for tests: 15.340 seconds
Complete requests:   50000
```

50.000

3.192

b. HAPROXY

i. Round Robin

1. 10.000 peticiones

```
lnaki@lnaki-N501VW:~$ ab -n 10000 -c 10 http://192.168.56.107/index.html
This is ApacheBench, Version 2.3 <$Revision: 1807734 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 192.168.56.107 (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:      Apache/2.4.29
Server Hostname:      192.168.56.107
Server Port:          80

Document Path:        /index.html
Document Length:      54 bytes

Concurrency Level:     10
Time taken for tests:  3.651 seconds
Complete requests:     10000
Failed requests:       0
Total transferred:     3000000 bytes
HTML transferred:     540000 bytes
Requests per second:   2738.74 [#/sec] (mean)
Time per request:      3.651 [ms] (mean)
Time per request:      0.365 [ms] (mean, across all concurrent requests)
Transfer rate:         802.36 [Kbytes/sec] received

Connection Times (ms)
  min   mean[+/-sd] median   max
Connect:    0      0   0.1      0     5
Processing:  1      4   1.1      3    12
Waiting:    1      3   1.0      3    11
Total:      1      4   1.1      3    13

Percentage of the requests served within a certain time (ms)
 50%    3
 66%    4
 75%    4
 80%    4
 90%    5
 95%    6
 98%    6
 99%    7
100%   13 (longest request)
```

```
Concurrency Level:      10
Time taken for tests:    3.694 seconds
Complete requests:      10000
```

HaProxy

10.000

3.651

3.694

Media: 3.672

2. 50.000 peticiones

```
Time taken for tests: 24.379 seconds
Complete requests: 50000
```

```
Time taken for tests: 24.256 seconds
Complete requests: 50000
```

50.000

24.379

24.256

ii. Ponderación

1. 10.000 peticiones

```
lnakt@lnaktl-N501VM:~$ ab -n 10000 -c 10 http://192.168.56.107/index.html
This is ApacheBench, Version 2.3 <$Revision: 1807734 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 192.168.56.107 (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: Apache/2.4.29
Server Hostname: 192.168.56.107
Server Port: 80

Document Path: /index.html
Document Length: 54 bytes

Concurrency Level: 10
Time taken for tests: 3.974 seconds
Complete requests: 10000
Failed requests: 0
Total transferred: 3000000 bytes
HTML transferred: 540000 bytes
Requests per second: 2516.36 [#/sec] (mean)
Time per request: 3.974 [ms] (mean)
Time per request: 0.397 [ms] (mean, across all concurrent requests)
Transfer rate: 737.21 [Kbytes/sec] received

Connection Times (ms)
  min   mean[+/-sd] median   max
Connect:  0    0  0.2   0    5
Processing:  1    4  1.4   4   21
Waiting:  1    4  1.3   3   15
Total:  1    4  1.4   4   22

Percentage of the requests served within a certain time (ms)
 50%    4
 66%    4
 75%    5
 80%    5
 90%    6
 95%    6
 98%    8
 99%    9
100%   22 (longest request)
```

```
Time taken for tests: 3.694 seconds
Complete requests: 10000
```

10.000

3.974

3.694

Media: 3.834

2. 50.000 peticiones

```
Concurrency Level:      10
Time taken for tests:    20.396 seconds
Complete requests:      50000
```

```
Time taken for tests:    19.237 seconds
Complete requests:      50000
```

50.000

20.396
19.237

iii. Leastconn (10.000 peticiones)

```
Time taken for tests:    3.886 seconds
Complete requests:      10000
```

```
Time taken for tests:    4.029 seconds
Complete requests:      10000
```

10.000

3.886
4.029
Media: 3.957

c. Go-Between

Con Go-Between no he probado a lanzar 50.000 peticiones con ab ya que los tiempos de ejecución de 10.000 peticiones eran bastante malos:

i. Round Robin

```
lnaki@lnaki-NS01VM:~$ ab -n 10000 -c 10 http://192.168.56.107:3000/index.html
This is ApacheBench, Version 2.3 <$Revision: 1807734 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 192.168.56.107 (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:      Apache/2.4.29
Server Hostname:      192.168.56.107
Server Port:          3000

Document Path:        /index.html
Document Length:      54 bytes

Concurrency Level:     10
Time taken for tests:   7.439 seconds
Complete requests:     10000
Failed requests:        0
Total transferred:     3000000 bytes
HTML transferred:     540000 bytes
Requests per second:   1344.25 [#/sec] (mean)
Time per request:      7.439 [ms] (mean)
Time per request:      0.744 [ms] (mean, across all concurrent requests)
Transfer rate:         393.82 [Kbytes/sec] received

Connection Times (ms)
  min   mean[+/-sd] median   max
Connect:    0      0   0.2      0      7
Processing:  1      7   3.0      7     24
Waiting:    1      7   3.0      7     23
Total:      1      7   3.0      7     24

Percentage of the requests served within a certain time (ms)
 50%    7
 66%    8
 75%    9
 80%   10
 90%   12
 95%   13
 98%   15
 99%   17
100%   24 (longest request)
```

```
Time taken for tests:    7.625 seconds
Complete requests:      10000
```

10.000

7.439

7.625

Media: 7.532

ii. Ponderación

```
Time taken for tests: 7.740 seconds
Complete requests: 10000
```

```
# Time taken for tests: 7.916 seconds
rttComplete requests: 10000
```

10.000

7.740
7.916
Media: 7.828

d. ZEVENET

Con Zevenet he probado a lanzar 10.000 peticiones con ab y los tiempos de ejecución han sido los mejores de entre todos los balanceadores software que hemos probado en esta práctica.

i. Round Robin

```
inaki@inaki-N501VW:~$ ab -n 10000 -c 10 http://192.168.56.111:442/index.html
This is ApacheBench, Version 2.3 <$Revision: 1807734 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 192.168.56.111 (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:      Apache/2.4.29
Server Hostname:      192.168.56.111
Server Port:          442


Document Path:        /index.html
Document Length:      54 bytes


Concurrency Level:    10
Time taken for tests:  2.098 seconds
Complete requests:    10000
Failed requests:      0
Total transferred:    3000000 bytes
HTML transferred:     540000 bytes
Requests per second:  4766.52 [#/sec] (mean)
Time per request:     2.098 [ms] (mean)
Time per request:     0.210 [ms] (mean, across all concurrent requests)
Transfer rate:        1396.44 [Kbytes/sec] received


Connection Times (ms)
              min   mean[+/-sd] median   max
Connect:        0    0   0.3      0      5
Processing:      0    2   0.9      2     14
Waiting:         0    2   0.9      2     14
Total:           0    2   0.9      2     16


Percentage of the requests served within a certain time (ms)
 50%    2
 66%    2
 75%    3
 80%    3
 90%    3
 95%    4
 98%    4
 99%    5
100%   16 (longest request)
```

```

Concurrency Level:      10
Time taken for tests:   2.046 seconds
Complete requests:      10000
Failed requests:        0

```

10.000

2.098
2.046
Media: 2.074

ii. Ponderación

```

inaki@inaki-N501VM:~$ ab -n 10000 -c 10 http://192.168.56.111:442/index.html
This is ApacheBench, Version 2.3 <$Revision: 1807734 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 192.168.56.111 (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:      Apache/2.4.29
Server Hostname:      192.168.56.111
Server Port:          442


Document Path:         /index.html
Document Length:       54 bytes


Concurrency Level:     10
Time taken for tests:   2.559 seconds
Complete requests:      10000
Failed requests:        0
Total transferred:      3000000 bytes
HTML transferred:       540000 bytes
Requests per second:    3908.18 [#/sec] (mean)
Time per request:       2.559 [ms] (mean)
Time per request:       0.256 [ms] (mean, across all concurrent requests)
Transfer rate:          1144.97 [Kbytes/sec] received


Connection Times (ms)
              min   mean[+/-sd] median   max
Connect:        0     0   0.2      0      3
Processing:      0     2   1.4      3     17
Waiting:        0     2   1.4      3     17
Total:          0     3   1.4      3     18


Percentage of the requests served within a certain time (ms)
 50%      3
 66%      3
 75%      4
 80%      4
 90%      4
 95%      4
 98%      5
 99%      5
100%     18 (longest request)

```

```
Concurrency Level:      10
Time taken for tests:    2.606 seconds
Complete requests:      10000
Failed requests:         0
```

10.000

2.559
2.606
Media: 2.577

e. POUND

Para el balanceador de carga Pound los resultados han sido bastante desilusionantes, siendo el peor de los balanceadores software que hemos probado como se podrá ver de forma más clara posteriormente en una gráfica:

i. Round Robin

```
inaki@inaki-N501VW:~$ ab -n 10000 -c 10 http://192.168.56.107/index.html
This is ApacheBench, Version 2.3 <$Revision: 1807734 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/
```

```
Benchmarking 192.168.56.107 (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:      Apache/2.4.29
Server Hostname:      192.168.56.107
Server Port:          80
```

```
Document Path:        /index.html
Document Length:       54 bytes
```

```
Concurrency Level:     10
Time taken for tests:   8.097 seconds
Complete requests:      10000
```

```
Concurrency Level:     10
Time taken for tests:   8.051 seconds
Complete requests:      10000
```

10.000

8.097
8.051
Media: 8.073

ii. Ponderación

```
Concurrency Level:      10
Time taken for tests:    7.757 seconds
Complete requests:      10000
```

```
Concurrency Level:      10
Time taken for tests:    7.683 seconds
Complete requests:      10000
```

10.000

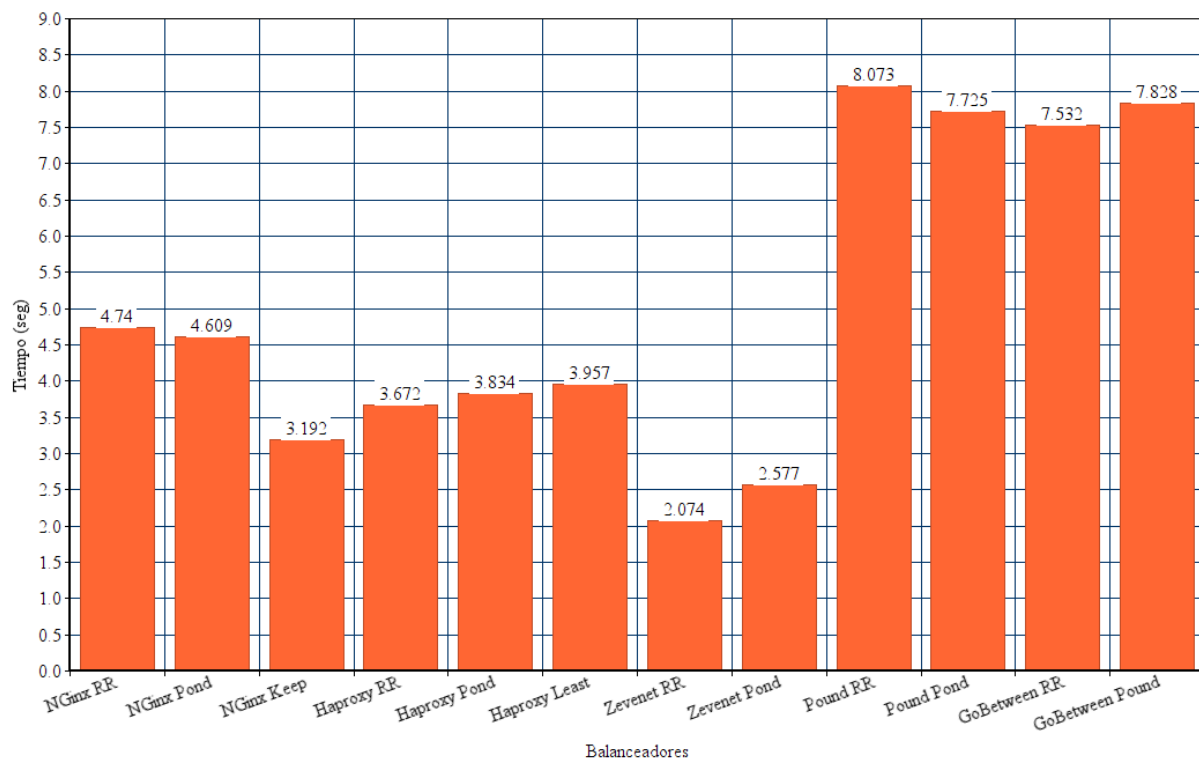
7.757
7.693
Media: 7.725

Una vez que hemos obtenido los tiempos de ejecución de ese sometimiento a carga a la granja web, vamos a comparar a todos los balanceadores software que hemos estudiado. Para ello y para poder compararlos vamos a utilizar los tiempos medios calculados anteriormente, que son los tiempos en segundos de ejecutar la orden

```
ab -n 10000 -c 10 http://IP/index.html
```

Es decir, el tiempo que tarda el balanceador en distribuir 10000 peticiones HTTP.

Gráfica comparativa balanceadores



Como podemos ver en la gráfica, los tiempos de ejecución de cada balanceador con el algoritmo de Round Robin y por el algoritmo de ponderación dándole mayor carga a la máquina m1-imm98 es más o menos similar. Donde si que varían más los tiempos es a la hora de elegir el balanceador software que distribuirá la carga. El mejor balanceador sin ninguna duda es el balanceador **Zevenet** concretamente el balanceador Zevenet con el algoritmo de Round Robin que es más del doble de rápido que uno tan famoso como NGinx. Destaca también la lentitud de los balanceadores GoBetween y Pound y también es curioso como el balanceador NGinx con el algoritmo de Keepalive es bastante más rápido que con Round Robin y Pound ya que no tiene que establecer una conexión TCP por cada petición HTTP.