

SWAP: Balanceo de carga en un sitio web

David Cabezas Berrido

dxabezas@correo.ugr.es

20 de abril de 2021

Índice

1. Preparativos	2
2. Balanceo de carga con NGINX	2
3. Balanceo de carga con HAProxy	4
4. Estadísticas de HAProxy	5
5. Balanceo de carga con Pound	6
6. Someter a carga la granja web con AB	8
7. Balanceo de carga con Gobetween	9
8. Balanceo de carga con Zevenet	10
9. Análisis comparativo de distintos balanceadores	14

1. Preparativos

Creamos dos archivos `/var/www/html/index.html` básicos en las máquinas 1 y 2, donde se referencia el número de la máquina a la que pertenece el archivo para saber cuál de las dos máquinas atendió la petición.

Creamos una nueva máquina virtual M3 con Ubuntu Server, pero no instalamos los servicios de la práctica 1, ya que no podemos tener a Apache ocupando el puerto 80. Nos limitamos a configurar el doble adaptador de red como hicimos en las otras dos máquinas. Su dirección IP es 192.168.56.103, para hacer peticiones desde la máquina anfitriona.

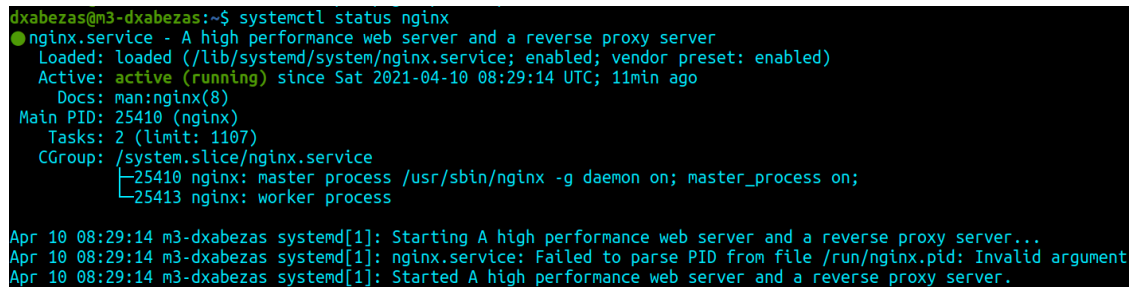
A partir de aquí, todas las órdenes y configuraciones se realizan en M3 a menos que digamos lo contrario.

2. Balanceo de carga con NGINX

Comenzamos instalando y lanzando NGINX:

```
sudo apt-get update && sudo apt-get dist-upgrade && sudo apt-get autoremove
sudo apt-get install nginx
sudo systemctl start nginx
```

Comprobamos que el servicio está en funcionamiento.

A terminal window showing the status of the nginx service. The prompt is 'dxabezas@m3-dxabezas:~\$'. The command 'systemctl status nginx' is executed. The output shows that the service is loaded and active. Below this, the logs for the service are displayed, showing the start of the nginx service and a warning about a failed attempt to parse the PID from the file /run/nginx.pid.

```
dxabezas@m3-dxabezas:~$ systemctl status nginx
● 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 Sat 2021-04-10 08:29:14 UTC; 11min ago
     Docs: man:nginx(8)
   Main PID: 25410 (nginx)
    Tasks: 2 (limit: 1107)
   CGroup: /system.slice/nginx.service
           └─25410 nginx: master process /usr/sbin/nginx -g daemon on; master_process on;
             └─25413 nginx: worker process

Apr 10 08:29:14 m3-dxabezas systemd[1]: Starting A high performance web server and a reverse proxy server...
Apr 10 08:29:14 m3-dxabezas systemd[1]: nginx.service: Failed to parse PID from file /run/nginx.pid: Invalid argument
Apr 10 08:29:14 m3-dxabezas systemd[1]: Started A high performance web server and a reverse proxy server.
```

Figura 1: NGINX está activo.

Escribimos en `/etc/nginx/conf.d/default.conf` la configuración que se indica en el guión para que NGINX funcione como balanceador de carga en lugar de como servidor web.

```
upstream balanceo_dxabezas {
    server 192.168.56.101;
    server 192.168.56.102;
}

server{
    listen 80;
    server_name balanceador_dxabezas;
    access_log /var/log/nginx/balanceador_dxabezas.access.log;
    error_log /var/log/nginx/balanceador_dxabezas.error.log;
    root /var/www/;

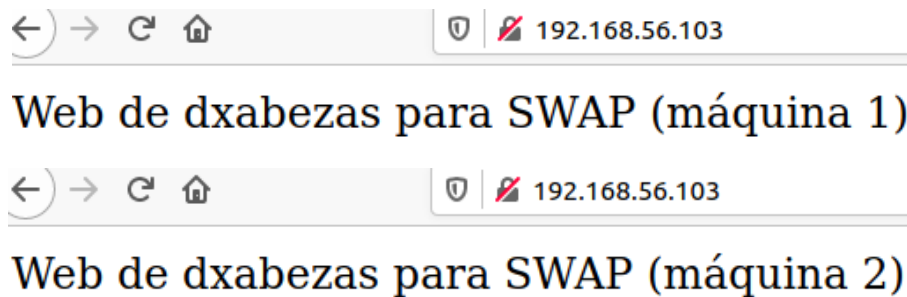
    location /
    {
        proxy_pass http://balanceo_dxabezas;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_http_version 1.1;
        proxy_set_header Connection "";
    }
}
```

Reestauramos el servicio con `sudo service nginx restart`, haremos esto (aunque no lo digamos) cada vez que modifiquemos este fichero. Ahora si visitamos la IP de M3 en el navegador de la máquina anfitriona, observamos que NGINX sigue funcionando como servidor web.



Figura 2: NGINX está funcionando como servidor web.

Como se nos indica en el guión, comentamos la línea `/etc/nginx/sites-enabled/*;` del fichero `/etc/nginx/nginx.conf`. Ahora accedemos a la IP de la máquina 3, y cada vez que refrestamos la página se turnan las máquinas 1 y 2 para servirnos su `index.html`.



Ahora añadimos el parámetro `weight` en el fichero `/etc/nginx/conf.d/default.conf` para que la máquina 2 reciba el doble de peticiones que la 1.

```
upstream balanceo_dxabezas {
    server 192.168.56.101 weight=1;
    server 192.168.56.102 weight=2;
}
```

Si ahora vamos refrescando la página, la máquina que nos atiende en cada momento es: M1, M2, M2, M1, M2, M2, M1, M2, M2, ...

Seguidamente, probamos a cambiar el algoritmo de Round-Robin (por defecto) a IP-HASH para que siempre nos atienda la misma máquina, lo conseguimos añadiendo la directiva `ip_hash` en el fichero de configuración.

```
upstream balanceo_dxabezas {
    ip_hash;
    server 192.168.56.101;
    server 192.168.56.102;
}
```

Ahora, por más que refresquemos la página, nos atiende siempre la misma máquina, en nuestro caso la 2.

Finalmente, activamos las conexiones con `keepalive` the 3 segundos.

```
upstream balanceo_dxabezas {
    server 192.168.56.101;
    server 192.168.56.102;
```

```
    keepalive 3;
}
```

Aunque no es sencillo comprobar que funciona correctamente, ya que recargar la página abre una conexión nueva. Probamos algunas opciones avanzadas más. De las propuestas en el guión, elegiremos aquellas cuyo funcionamiento podamos comprobar más fácilmente.

```
upstream balanceo_dxabezas {
    ip_hash;
    server 192.168.56.101;
    server 192.168.56.102 down;
}
```

Marcando M2 como down con `ip_hash`, comprobamos que ahora nos atiende M1 en lugar de M2.

```
upstream balanceo_dxabezas {
    server 192.168.56.101;
    server 192.168.56.102 backup;
}
```

Ahora marcamos M2 como backup y siempre nos atiende M1. Si desactivamos el servicio de M1 con `sudo systemctl stop apache2`, pasa a atendernos todo el rato M2.

3. Balanceo de carga con HAProxy

Primero apagamos NGINX para que libere el puerto 80 con `sudo systemctl stop nginx`

Instalamos el balanceador con `sudo apt install haproxy`. Añadimos la siguiente configuración al fichero `/etc/haproxy/haproxy.cfg`, y hacemos `sudo systemctl restart haproxy.service` (lo haremos tras cada modificación del fichero de configuración).

```
frontend http-in
    bind *:80
    default_backend balanceo_dxabezas

backend balanceo_dxabezas
    server m1 192.168.56.101:80 maxconn 32
    server m2 192.168.56.102:80 maxconn 32
```

Si ahora hacemos `status`, vemos que el servicio está activo y nuestro balanceador iniciado.

```
dxabezas@m3-dxabezas:~$ sudo systemctl status haproxy
● haproxy.service - HAProxy Load Balancer
   Loaded: loaded (/lib/systemd/system/haproxy.service; enabled; vendor preset: enabled)
   Active: active (running) since Mon 2021-04-12 18:11:06 UTC; 1min 32s ago
     Docs: man:haproxy(1)
           file:/usr/share/doc/haproxy/configuration.txt.gz
   Process: 3366 ExecStartPre=/usr/sbin/haproxy -f $CONFIG -c -q $EXTRA_OPTS (code=exited, status=0/SUCCESS)
  Main PID: 3376 (haproxy)
    Tasks: 2 (limit: 1107)
   CGroup: /system.slice/haproxy.service
           └─3376 /usr/sbin/haproxy -Ws -f /etc/haproxy/haproxy.cfg -p /run/haproxy.pid
             └─3377 /usr/sbin/haproxy -Ws -f /etc/haproxy/haproxy.cfg -p /run/haproxy.pid

Apr 12 18:11:06 m3-dxabezas systemd[1]: Starting HAProxy Load Balancer...
Apr 12 18:11:06 m3-dxabezas haproxy[3376]: Proxy http-in started.
Apr 12 18:11:06 m3-dxabezas haproxy[3376]: Proxy http-in started.
Apr 12 18:11:06 m3-dxabezas haproxy[3376]: Proxy balanceo_dxabezas started.
Apr 12 18:11:06 m3-dxabezas haproxy[3376]: Proxy balanceo_dxabezas started.
Apr 12 18:11:06 m3-dxabezas systemd[1]: Started HAProxy Load Balancer.
```

Si ahora accedemos a `192.168.56.103` desde el navegador, observamos que las dos máquinas se turnan para servirnos su `index.html`.

Ahora configuramos la ponderación.

```
frontend http-in
    bind *:80
    default_backend balanceo_dxabezas

backend balanceo_dxabezas
    server m1 192.168.56.101:80 maxconn 32 weight 2
    server m2 192.168.56.102:80 maxconn 32 weight 1
```

Si refrescamos la página nos atiende M1, M1, M2, M1, M1, M2, M1, M1, M2, ...

Ahora buscaremos algunas opciones avanzadas. Seleccionamos balanceo por IP-HASH.

```
frontend http-in
    bind *:80
    default_backend balanceo_dxabezas

backend balanceo_dxabezas
    balance source
    hash-type consistent
    server m1 192.168.56.101:80 maxconn 32
    server m2 192.168.56.102:80 maxconn 32
```

Ahora nos atiende siempre M1.

Finalmente, ponemos M1 en modo **backup**. Debemos añadir **check**, ya que no se activa el servidor de backup si **check** no devuelve **DOWN**. Si no añadimos la comprobación, las peticiones no son atendidas.

```
frontend http-in
    bind *:80
    default_backend balanceo_dxabezas

backend balanceo_dxabezas
    server m1 192.168.56.101:80 maxconn 32 check backup
    server m2 192.168.56.102:80 maxconn 32 check
```

Nos atiende siempre M2, pero cuando apagamos el servicio Apache2 en M2, pasa a atendernos siempre M1.

4. Estadísticas de HAProxy

Dejamos la configuración con Round-Robin básico, pero mantenemos los **check**.

Ahora añadimos la siguiente configuración (a HAProxy) para habilitar las estadísticas: En **global** escribimos **stats socket /var/lib/haproxy/stats**, sustituyendo la anterior línea de **stats socket**; y añadimos el siguiente bloque:

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

Reseteamos el servicio. Ahora, en la ruta **/stats** del puerto 9999 podemos ver las estadísticas, logueandonos primero con **dxabezas** tanto como usuario como contraseña.

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

Statistics Report for pid 2375

> General process information

pid = 2375 (process #1, nproc = 1, nbthread = 1)
uptime = 0d 0h26m04s
system limits: memmax = unlimited; ulimit-n = 4032
maxsock = 4032; maxconn = 2000; maxpipes = 0
current conns = 1; current pipes = 0/0; conn rate = 1/sec
Running tasks: 1/6; idle = 100 %

active UP
active UP, going down
active DOWN, going up
active or backup DOWN
active or backup DOWN for maintenance (MAINT)
active or backup SOFT STOPPED for maintenance
Note: "NOLB"/"DRAIN" = UP with load-balancing disabled.

Display option:

- Scope :
- Hide **DOWN** servers
- Refresh now
- CSV export

External resources:

- Primary site
- Updates (v1.8)
- Online manual

stats		Queue		Session rate			Sessions						Bytes		Denied		Errors		Warnings		Server											
		Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle	
Frontend					1	2	-	1	1	2 000	23			18 883	522 228	0	0	0					OPEN									
Backend		0	0		0	2		0	1	200	18	0	0s	18 883	522 228	0	0		18	0	0	0	26m4s UP		0	0	0		0			

http-in		Queue		Session rate			Sessions						Bytes		Denied		Errors		Warnings		Server											
		Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle	
Frontend					0	1	-	0	1	2 000	10			34 385	32 200	0	0	0					OPEN									

balanceo_dxabezas		Queue		Session rate			Sessions						Bytes		Denied		Errors		Warnings		Server										
		Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
m1		0	0	-	0	3		0	1	32	40	40	35s	16 962	16 089		0		0	0	0	0	no check				1	Y	-		-
m2		0	0	-	0	4		0	1	32	40	40	35s	17 423	16 111		0		0	0	0	0	no check		1	Y	-			-	
Backend		0	0		0	7		0	1	200	80	80	35s	34 385	32 200	0	0		0	0	0	0	26m4s UP		2	2	0		0	0s	

Figura 3: Estadísticas de HAProxy, podemos ver los pesos de cada servidor, las sesiones totales, las sesiones actuales de cada servidor (Cur=0), el máximo (1) y el límite de sesiones concurrentes (maxconn 32); también el tiempo que llevan funcionando.

Añadiendo estas dos líneas a `listen stats` y restaurando el servicio, hacemos que los datos se refresquen cada 15 segundos, y que podamos modificar marcar los servidores como DOWN o MAINTENANCE, o cortar sus sesiones actuales.

```
stats refresh 15s
stats admin if TRUE
```

balanceo_dxabezas																																
<input type="checkbox"/>		Queue			Session rate			Sessions				Bytes		Denied		Errors			Warnings		Server											
		Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle	
<input type="checkbox"/>	m1	0	0	-	0	0		0	0	32	0	0	?	0	0	0	0	0	0	0	0	0	41s UP	L4OK in 1ms	1	Y	-	0	0	0s	-	
<input checked="" type="checkbox"/>	m2	0	0	-	0	0		0	0	32	0	0	?	0	0	0	0	0	0	0	0	41s UP	L4OK in 1ms	1	Y	-	0	0	0s	-		
	Backend	0	0		0	0		0	0	200	0	0	?	?	0	0	0	0	0	0	0	41s UP		2	2	0		0	0s			

Choose the action to perform on the checked servers :

~

Apply

Figura 4: Podemos elegir acciones que aplicar a los servidores que marcamos.

5. Balanceo de carga con Pound

Descargamos el programa de aquí: <https://packages.ubuntu.com/xenial-updates/amd64/pound/download>.

Ahora nos vamos al fichero de configuración `/etc/pound/pound.cfg` y escribimos lo siguiente:

```
ListenHTTP
    Address 192.168.56.103
    Port    80
End

Service
    Backend
        Address 192.168.56.101
        Port    80
        Priority 2
    End

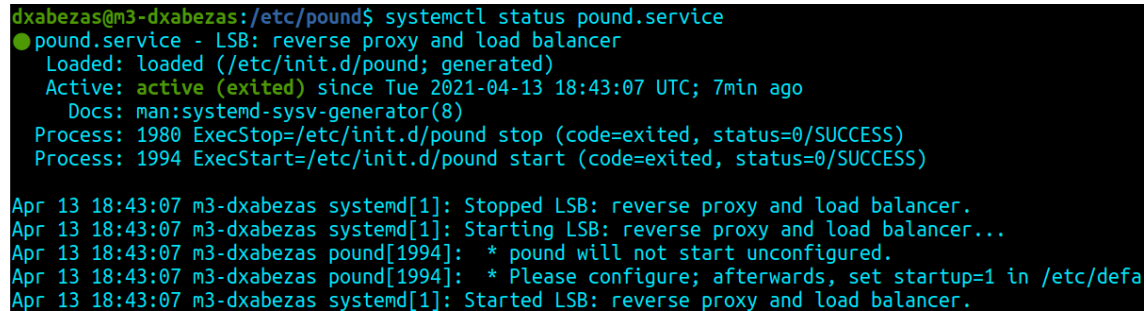
    Backend
        Address 192.168.56.102
```

```

        Port      80
        Priority    1
    End
End

```

Además, hay que cambiar `startup=0` por `startup=1`, en el fichero `/etc/default/pound` como se indica en el siguiente mensaje de estado (sale cortado):



```

dxabezas@m3-dxabezas:/etc/pound$ systemctl status pound.service
● pound.service - LSB: reverse proxy and load balancer
   Loaded: loaded (/etc/init.d/pound; generated)
   Active: active (exited) since Tue 2021-04-13 18:43:07 UTC; 7min ago
     Docs: man:systemd-sysv-generator(8)
  Process: 1980 ExecStop=/etc/init.d/pound stop (code=exited, status=0/SUCCESS)
  Process: 1994 ExecStart=/etc/init.d/pound start (code=exited, status=0/SUCCESS)

Apr 13 18:43:07 m3-dxabezas systemd[1]: Stopped LSB: reverse proxy and load balancer.
Apr 13 18:43:07 m3-dxabezas systemd[1]: Starting LSB: reverse proxy and load balancer...
Apr 13 18:43:07 m3-dxabezas pound[1994]: * pound will not start unconfigured.
Apr 13 18:43:07 m3-dxabezas pound[1994]: * Please configure; afterwards, set startup=1 in /etc/defa
Apr 13 18:43:07 m3-dxabezas systemd[1]: Started LSB: reverse proxy and load balancer.

```

Figura 5: El servicio no arrancará hasta que no se marque como configurado.

Hemos aprovechado para poner ponderaciones con `Priority`. Nos conectamos a la IP de M3 desde el navegador y nos M1 atiende el doble de peticiones que M2.

Se pueden poner más opciones como `Timeout 10`, para fijar un tiempo de `Timeout` de 10 segundos antes de que Pound considere que el servidor no va a responder.

Como opción avanzada, introducimos `Emergency`, para que sólo use el servidor cuando el resto falle.

```

Service
    BackEnd
        Address 192.168.56.101
        Port     80
    End

    Emergency
        Address 192.168.56.102
        Port     80
    End
End

```

Ahora nos atiende siempre M1, pero si apagamos el servicio Apache2 en M1 pasa a atendernos M2.

Finalmente, Pound tiene una funcionalidad similar al IP-Hash llamada `Session`, que hace que las peticiones provenientes de la misma IP sean atendidas por la misma máquina dentro de un tiempo, a partir del cual se descarta la sesión. Dejamos la configuración de esta forma:

```

Service
    BackEnd
        Address 192.168.56.101
        Port     80
    End

    BackEnd
        Address 192.168.56.102
        Port     80
    End

    Session
        Type IP

```

TTL 60

End

End

Las peticiones de la misma IP siempre son atendidas por al misma máquina. Tras 60 segundos sin peticiones, se descarta la sesión.

6. Someter a carga la granja web con AB

Apache Benchmark se instala con el servicio Apache2, lo instalo en mi máquina anfitriona, desde la que lanzaré peticiones a M3.

Lanzamos el benchmark desde la máquina anfitriona. Lanzamos 10000 peticiones con un nivel de concurrencia de 10.

```
ab -n 10000 -c 10 http://192.168.56.103/index.html
```

Observamos con `top` como Apache2 se convierte en el proceso que más recursos consume de M1 Y M2, y el balanceador el que más consume de M3.

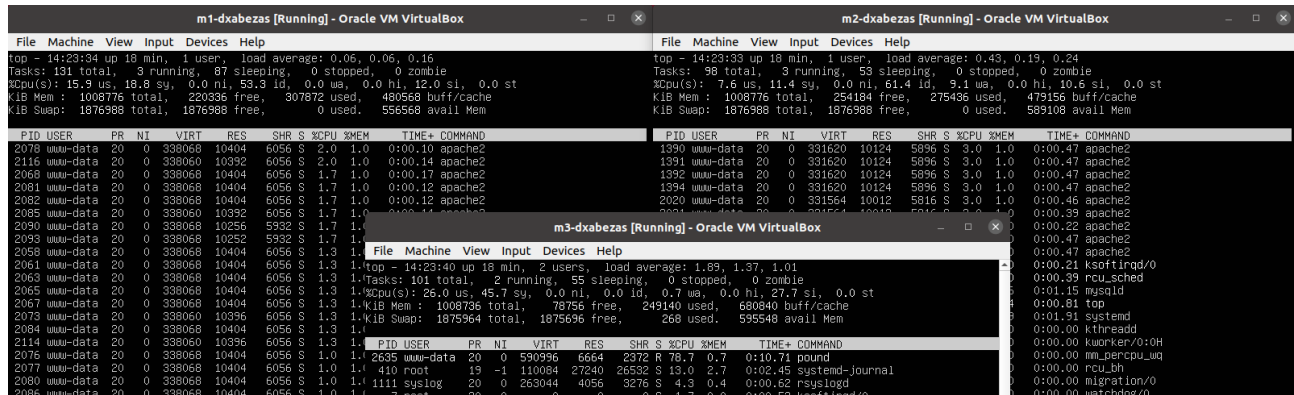


Figura 6: Ante la llegada masiva de peticiones, los servicios empiezan a consumir más recursos.

Esperamos unos segundos, y AB nos genera un informe sobre el rendimiento del servidor.

```
dcabezas@Lenovo:~$ ab -n 10000 -c 10 http://192.168.56.103/index.html
This is ApacheBench, Version 2.3 <$Revision: 1843412 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 192.168.56.103 (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	Connection Times (ms)				
Server Hostname:	192.168.56.103		min	mean[+/-sd]	median	max
Server Port:	80	Connect:	0	0 0.1	0	4
Document Path:	/index.html	Processing:	2	8 1.9	8	23
Document Length:	121 bytes	Waiting:	1	6 1.7	6	20
		Total:	2	8 1.9	8	23
Concurrency Level:	10	Percentage of the requests served within a certain time (ms)				
Time taken for tests:	7.943 seconds	50%	8			
Complete requests:	10000	66%	8			
Failed requests:	0	75%	9			
Total transferred:	3910000 bytes	80%	9			
HTML transferred:	1210000 bytes	90%	10			
Requests per second:	1259.00 [#/sec] (mean)	95%	11			
Time per request:	7.943 [ms] (mean)	98%	13			
Time per request:	0.794 [ms] (mean, across all concurrent requests)	99%	14			
Transfer rate:	480.73 [Kbytes/sec] received	100%	23 (longest request)			

Figura 6: Salida de Apache Benchmark. In la primera imagen nos informa del progreso de benchmarking. En la segunda imagen tenemos información sobre el servidor y el fichero solicitado (`index.html`), así como del número de peticiones y bytes transferidos y el tiempo tomado. En la tercera imagen tenemos estadísticas sobre los tiempos que toma la conexión, procesamiento y espera de las peticiones, y también los milisegundos bajo los cuales se han atendido distintos porcentajes de peticiones.

Aunque vayamos a utilizar el criterio peticiones/segundo (en este caso 1259, segunda imagen) para comparar los balanceadores, también son muy interesantes las medias de la tercera imagen. Nos permiten “asegurar” que nuestro servidor atiende cierto porcentaje de las peticiones en cierto tiempo. Ej: “Generalmente, ninguna petición tarda en atenderse más de 23 ms”, “El 99 % de las peticiones que recibe son atendidas en 14 segundos o menos”.

Por último, introduciremos algunas opciones y parámetros de Apache Benchmark que pueden ser útiles.

- `-e archivo.csv` genera un archivo CSV con los datos de la segunda mitad de la tercera imagen.
- `-q` elimina los mensajes de progreso.
- `-t 60` indica el número máximo de segundos que durará el benchmark aunque no se completen todas las peticiones, en este caso 60.
- `-p archivo` para hacer peticiones de POST de un archivo.
- `-u archivo` para hacer peticiones de PUT de un archivo.
- `-T content-type` indica el header para POST y PUT, es obligatorio si se usa alguna de las dos opciones anteriores.

7. Balanceo de carga con Gobetween

Instalamos Gobetween desde la Snap Store de Ubuntu, que viene instalada y habilitada por defecto.

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

Como lo hemos instalado desde Snap, tenemos que invocarlo con

```
/snap/gobetween/current/bin/gobetween
```

Creamos un fichero de configuración en `/snap/gobetween/gobetween-cfg.toml`.

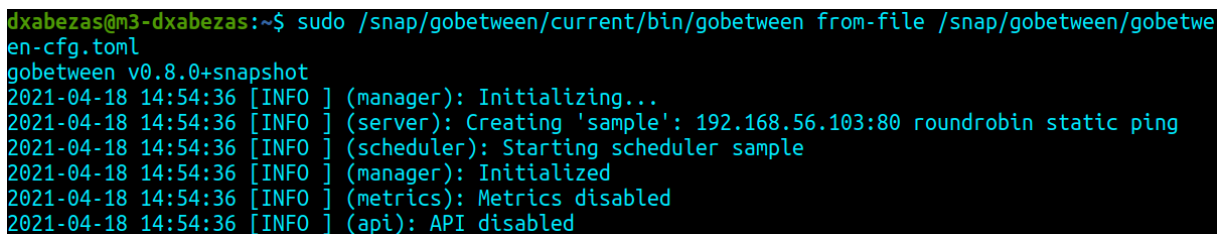
```
[servers.sample]
bind = "192.168.56.103:80"
protocol = "tcp"
balance = "roundrobin"

max_connections = 10000
client_idle_timeout = "10m"
backend_idle_timeout = "10m"
backend_connection_timeout = "2s"
```

```
[servers.sample.discovery]
kind = "static"
static_list = [
    "192.168.56.101:80",
    "192.168.56.102:80"
]

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

Ahora lo lanzamos con el siguiente comando, y efectivamente funciona. Hemos necesitado lanzarlo con `sudo` porque se le denegaba el acceso de escucha en el puerto 80.



```
dxabezas@m3-dxabezas:~$ sudo /snap/gobetween/current/bin/gobetween from-file /snap/gobetween/gobetween-cfg.toml
gobetween v0.8.0+snapshot
2021-04-18 14:54:36 [INFO ] (manager): Initializing...
2021-04-18 14:54:36 [INFO ] (server): Creating 'sample': 192.168.56.103:80 roundrobin static ping
2021-04-18 14:54:36 [INFO ] (scheduler): Starting scheduler sample
2021-04-18 14:54:36 [INFO ] (manager): Initialized
2021-04-18 14:54:36 [INFO ] (metrics): Metrics disabled
2021-04-18 14:54:36 [INFO ] (api): API disabled
```

Figura 7: Lanzamos Gobetween indicando el fichero de configuración.

El parámetro `balance` indica el algoritmo para distribuir la carga, también puede ser `iphash` (IP-Hash), `leastconn` (menor número de conexiones) o `weight` (ponderaciones). Este último nos obliga a escribir la lista de servidores con el siguiente formato.

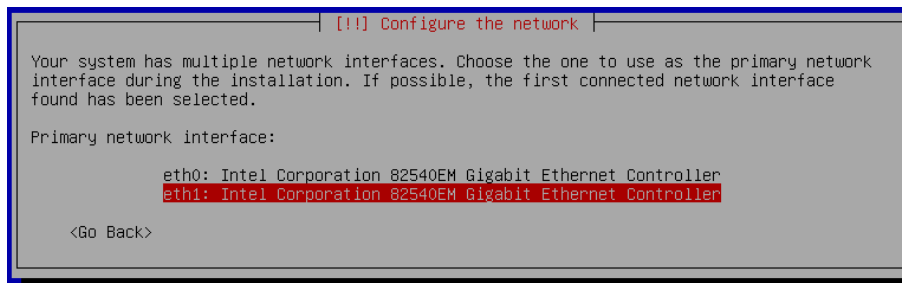
```
[servers.sample.discovery]
kind = "static"
static_list = [
    "192.168.56.101:80 weight=2",
    "192.168.56.102:80 weight=1"
]
```

En el último bloque del fichero de configuración sirve para hacer comprobaciones del estado de los servidores. En este caso, el balanceador hará `ping` cada dos segundos, y se espera una respuesta del servidor en 500 milisegundos. Tras un fallo (parámetro `fails`), se marca el servidor como `DOWN`, pero se siguen enviando pings por si en algún momento se recibe respuesta. Con una respuesta (parámetro `passes`) se vuelve a marcar el servidor como `OK`.

8. Balanceo de carga con Zevenet

Descargamos la ISO de Zevenet Community Edition desde la [web oficial](#) y lo instalamos en una máquina virtual con Ubuntu 64b, también con doble adaptador de red.

Durante la instalación, es importante marcar la interfaz correspondiente a la red local.



También nos obliga a configurar la red. Nos pide IP, gateway, mask, server name y domain name. Esto no es importante, lo configuramos con Netplan una vez finalizada la instalación.

```
network:
  version: 2
  ethernets:
    enp0s3:
      dhcp4: true
    enp0s8:
      dhcp4: true
```

Ahora comprobamos la dirección asignada con `ifconfig`.

```
root@m4-zevenet-ubuntu:~# ifconfig
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.0.2.15 netmask 255.255.255.0 broadcast 10.0.2.255
    inet6 fe80::a00:27ff:fe5b:b129 prefixlen 64 scopeid 0x20<link>
    ether 08:00:27:5b:b1:29 txqueuelen 1000 (Ethernet)
    RX packets 2 bytes 1180 (1.1 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 17 bytes 1840 (1.7 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

enp0s8: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.56.106 netmask 255.255.255.0 broadcast 192.168.56.255
    inet6 fe80::a00:27ff:feb8:6828 prefixlen 64 scopeid 0x20<link>
    ether 08:00:27:b8:68:28 txqueuelen 1000 (Ethernet)
    RX packets 1422 bytes 212689 (207.7 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 2153 bytes 2612809 (2.4 MiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 2 bytes 100 (100.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 2 bytes 100 (100.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

Figura 8: 192.168.56.106

Nos vamos al puerto 444 para hacer las configuraciones. Aquí nos logueamos como root con la contraseña de root que elegimos durante la instalación, en mi caso *Swap1234*. Nos encontramos el Dashboard.

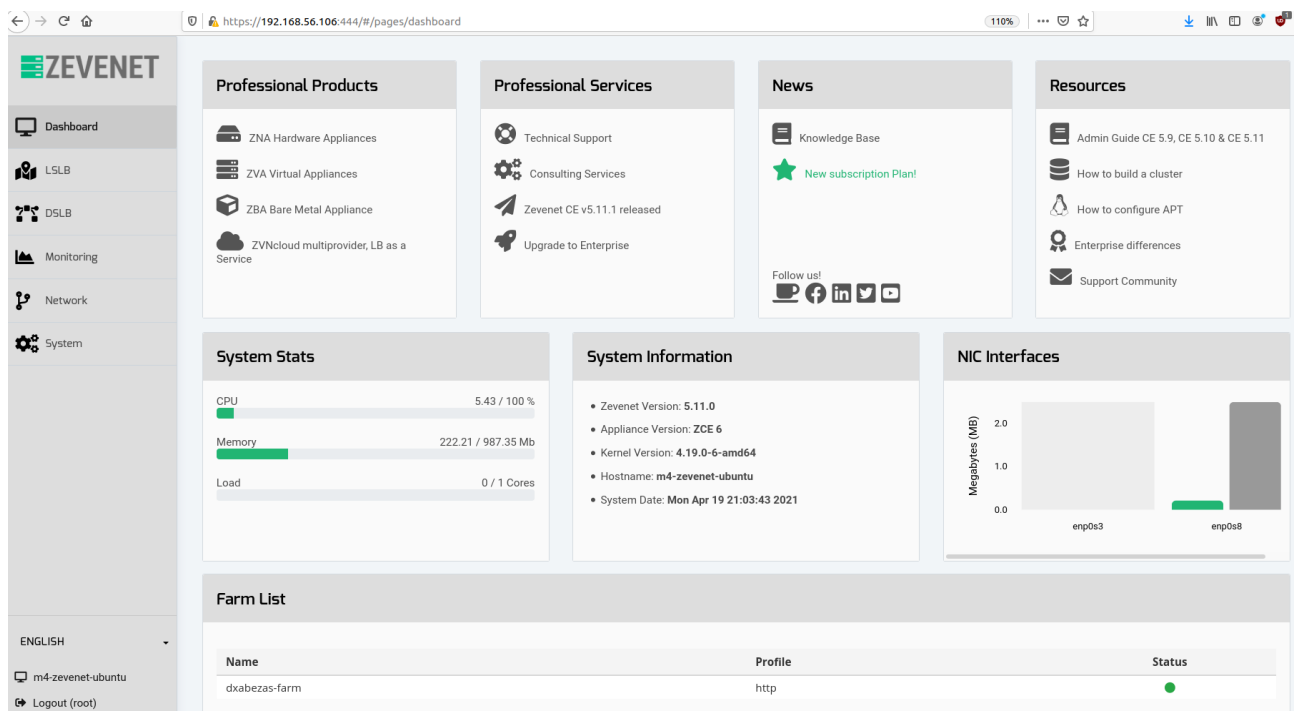


Figura 9: Dashboard de Zevenet. En la parte inferior ya aparece una granja configurada, a continuación explicamos cómo hacerlo.

Primero ponemos una VIP para la red doméstica. Vamos a **Network** -> **NIC** y editamos **enp0s8** para que quede de esta forma.

Network / NIC

NIC List

Name	IP	MAC	Netmask	Gateway	Status	Actions
enp0s3		08:00:27:5b:b1:29			●	✎ 📄 🔄
enp0s8	192.168.56.106	08:00:27:b8:68:28	255.255.255.0	192.168.56.1	●	✎ 📄 🔄

Seguidamente, en **LSLB** -> **Farms** creamos una nueva granja web con la siguiente configuración. Aquí se pueden modificar los timeouts y los mensajes de error entre otras opciones.

LSLB / Farms / Edit

Global Settings

Name:

Virtual IP and Port:

Listener:

Advanced settings

Rewrite location headers:

Backend connection timeout: FORM.seconds

Frequency to check resurrected backends: FORM.seconds

Message Error 414:

Message Error 501:

HTTP verbs accepted:

Backend response timeout: FORM.seconds

Client request timeout: FORM.seconds

Message Error 500:

Message Error 503:

Finalmente, creamos el servicio `index` con M1 y M2 como backends. Aquí se asignan también las ponderaciones.

Services Settings

[NEW SERVICE](#)

index

Virtual Host:

URL Pattern:

Least Response: ☐

HTTPS Backends: ☐

Redirect

Redirect: ☐

Persistence





Persistence:

Farmguardian

Health Checks for backend:

Backend

[ADD BACKEND](#)

ID	IP	Port	Timeout	Weight	Actions
0	192.168.56.101	80	20	1	 
1	192.168.56.102	80	20	1	 

Ya tenemos el balanceador funcionando. Para activar la persistencia según la IP del cliente lo marcamos en la pestaña del servicio, funciona de igual modo que Session en Pound. En Farmguardian tenemos comprobaciones del estado de los backends.

9. Análisis comparativo de distintos balanceadores

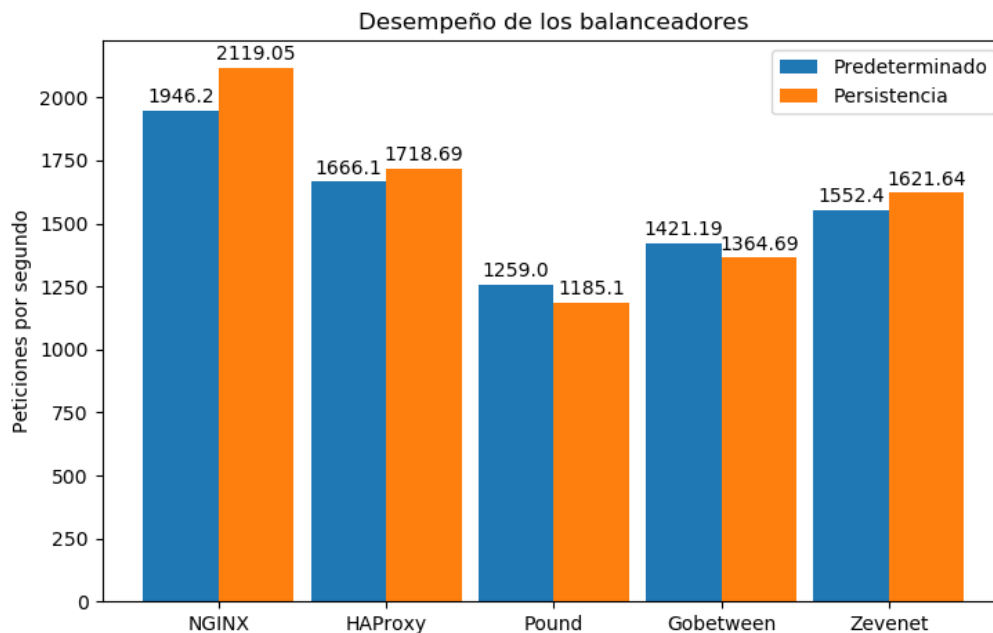
En la siguiente tabla recogemos las peticiones por segundo de cada balanceador con cada algoritmo para un benchmark como el anterior, con 10000 peticiones y concurrencia 10.

Cabe destacar que no estamos seguros de que Pound haga Round Robin por defecto. De hecho, a veces nos atiende dos veces seguidas la misma máquina. Lo mismo le ocurre a Zevenet. Por tanto, el nombre de la columna será Predeterminado en lugar de Round Robin.

Para la persistencia, utilizamos Session en Pound (que no es lo mismo que IP-Hash), y marcamos **Persistence** -> IP: **Client address** para Zevenet, que funciona de la misma manera. Para el resto utilizamos IP-Hash.

peticiones/s	Predeterminado	Persistencia
NGINX	1946.2	2119.05
HAProxy	1666.1	1718.69
Pound	1259	1185.1
Gobetween	1421.19	1364.69
Zevenet	1552.4	1621.64

Representamos los resultados en un gráfico de barras para facilitar la comparación.



Antes de comentar las diferencias en las configuraciones, está claro que para el benchmark realizado lo que más destaca es la diferencia entre los propios balanceadores. NGINX consigue las mejores prestaciones con diferencia, HAProxy es el siguiente más rápido seguido por Zevenet. Después tenemos a Gobetween, algo más lento. Y finalmente a Pound, el más lento con diferencia.

Respecto a las configuraciones, la persistencia obtiene mejores resultados en los más rápidos: NGINX y HAProxy, que usan IP-Hash, NGINX experimenta la mayor mejora. En el caso de Gobetween, la persistencia con IP-Hash deteriora ligeramente la rapidez del balanceador. Las sesiones, sin embargo, empeoran ligeramente a Pound y mejora ligeramente a Zevenet.

Esto no permite sacar conclusiones infalibles sobre la idoneidad de la persistencia en forma de IP-Hash o sesiones, ya que dependen del balanceador. Sin embargo, basándonos en los resultados de este benchmark, la persistencia supone una mejora en el desempeño de los balanceadores más rápidos, especialmente IP-Hash con NGINX. Por

tanto, si tuviésemos que seleccionar un balanceador para nuestra granja basándonos únicamente en los resultados sobre este benchmark, elegiríamos NGINX sin duda, y confiaríamos en IP-Hash como algoritmo de reparto de la carga.