
Servidores Web de Altas Prestaciones.

Práctica 3

Balanceo de carga en un sitio web.

Ricardo Ruiz Fernández de Alba

25/05/2023

Índice

Introducción	2
Descripción de las tareas	2
Tarea 1. Balanceo de carga con NGINX y HAProxy.	3
Balanceo de carga con NGINX	3
Instalación de NGINX.	3
Configuración de NGINX como balanceador de carga	4
Ejemplo de funcionamiento	7
Tarea avanzada: repartir carga en función de pesos	7
Balanceo de carga con HAProxy	11
Instalación de HAProxy	11
Configuración básica de haproxy como balanceador	11
Ejemplo de funcionamiento	13
Tarea Avanzada: repartir carga en función de pesos	14
Tarea Avanzada: Módulo de estadísticas.	15
Tarea 2. Alta carga con Apache Benchmark	16
HAProxy con Round Robin	16
NGINX con Round Robin	20
Tarea Avanzada: Balanceador de carga con Gobetween	21
Instalación de Gobetween	21
Tarea 3. Análisis Comparativo	22
Referencias	23

Introducción

En esta práctica, el objetivo es configurar las máquinas virtuales de forma que dos hagan de servidores web finales mientras que la tercera haga de balanceador de carga por software.

Descripción de las tareas

En esta práctica se llevarán a cabo las **tareas básicas**:

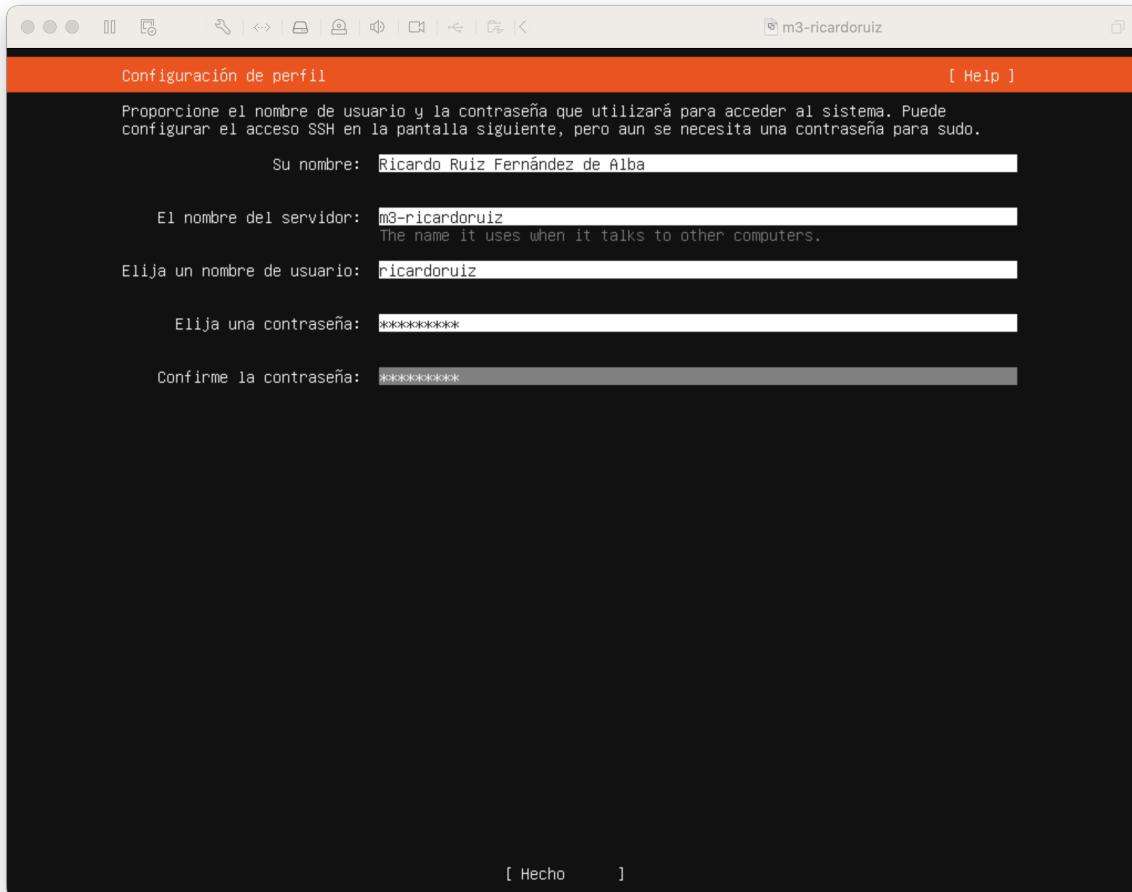
1. Configurar una máquina e instalar nginx y haproxy como balanceadores de carga con el algoritmo round-robin
2. Someter la granja web a una alta carga con la herramienta Apache Benchmark a través de M3, considerando 2 opciones:
 - a) nginx con round-robin
 - b) haproxy con round-robin
3. Realizar un análisis comparativo de los resultados considerando el número de peticiones por unidad de tiempo

Como **opciones avanzadas**:

1. Configurar nginx y haproxy como balanceadores de carga con ponderación, suponiendo que M1 tiene el doble de capacidad que M2.
2. Habilitar el módulo de estadísticas en HAProxy con varias opciones y analizarlo.
3. Instalar y configurar otros balanceadores de carga (Gobetween, Zevenet, Pound, etc.).
4. Someter la granja web a una alta carga con la herramienta Apache Benchmark considerando los distintos balanceadores instalados y configurados.
5. Realizar un análisis comparativo de los resultados considerando el número de peticiones por unidad de tiempo

Tarea 1. Balanceo de carga con NGINX y HAProxy.

Creamos una nueva máquina virtual llamada m3-ricardoruz con Ubuntu Server 22.04 LTS, a la que añadiremos el usuario ricardoruz con contraseña Swap12324.



Balanceo de carga con NGINX

Instalación de NGINX.

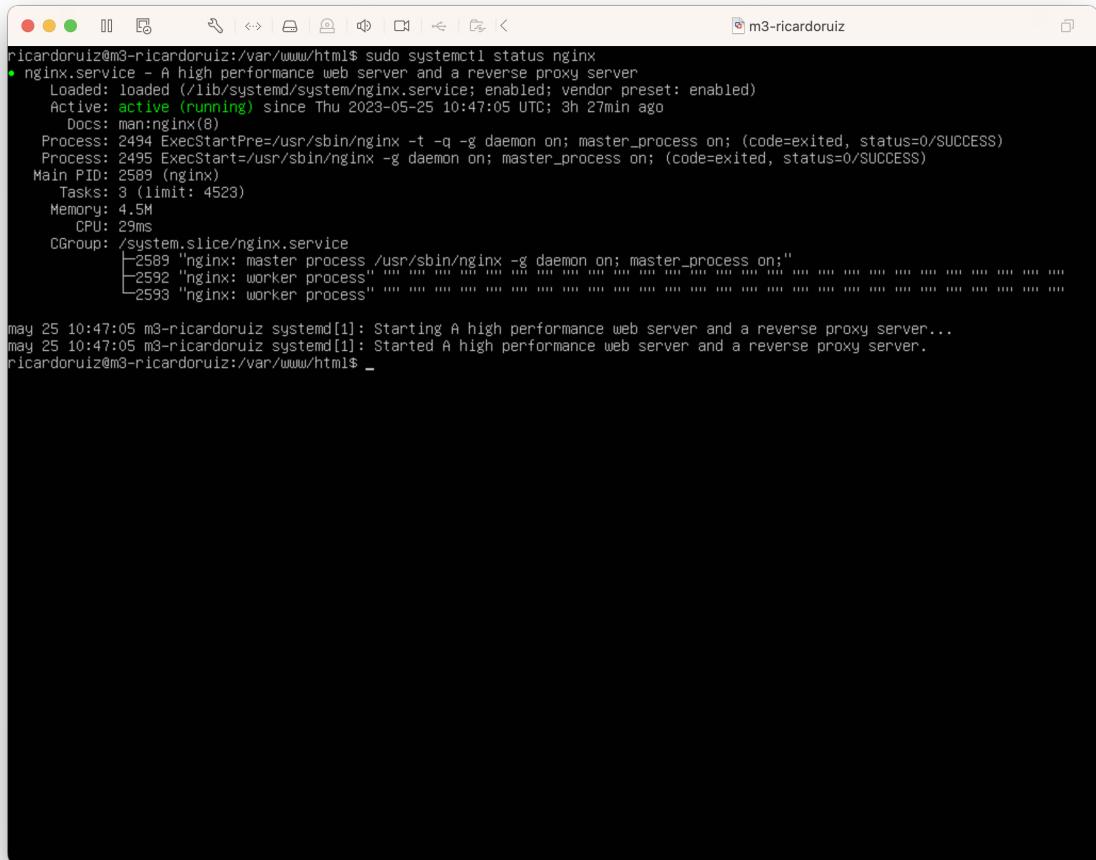
Seguiremos la guia de instalación de nginx para Ubuntu Server 22.04 deDigital Ocean.

```
1 ricardoruz@m3-ricardoruz $ sudo apt update
2 ricardoruz@m3-ricardoruz $ sudo apt install nginx
```

Antes de probar Nginx, es necesario configurar el firewall para permitir el acceso al servicio. Nginx se registra como un servicio en ufw durante la instalación, lo que facilita permitir el acceso a Nginx.

```
1 ricardoruiz@m3-ricardoruiz $ sudo ufw allow 'Nginx HTTP'
```

Comprobamos que nginx está activo con `sudo systemctl status nginx`:



```
ricardoruiz@m3-ricardoruiz:/var/www/html$ sudo 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 Thu 2023-05-25 10:47:05 UTC; 3h 27min ago
    Docs: man:nginx(8)
 Process: 2494 ExecStartPre=/usr/sbin/nginx -t -q -g daemon on; master_process on; (code=exited, status=0/SUCCESS)
 Process: 2495 ExecStart=/usr/sbin/nginx -g daemon on; master_process on; (code=exited, status=0/SUCCESS)
 Main PID: 2589 (nginx)
   Tasks: 3 (limit: 4523)
    Memory: 4.5M
      CPU: 29ms
     CGroup: /system.slice/nginx.service
             ├─2589 "nginx: master process /usr/sbin/nginx -g daemon on; master_process on;"
             ├─2592 "nginx: worker process"
             └─2593 "nginx: worker process"

may 25 10:47:05 m3-ricardoruiz systemd[1]: Starting A high performance web server and a reverse proxy server...
may 25 10:47:05 m3-ricardoruiz systemd[1]: Started A high performance web server and a reverse proxy server.
ricardoruiz@m3-ricardoruiz:/var/www/html$ _
```

Figura 1: Nginx

Configuración de NGINX como balanceador de carga

Debemos deshabilitar la configuración por defecto de nginx como servidor web para que actúe como balanceador.

Para ello, comentamos la línea

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

del fichero de configuración `/etc/nginx/nginx.conf`.

Creamos una nueva configuración en `/etc/nginx/conf.d/default.conf`

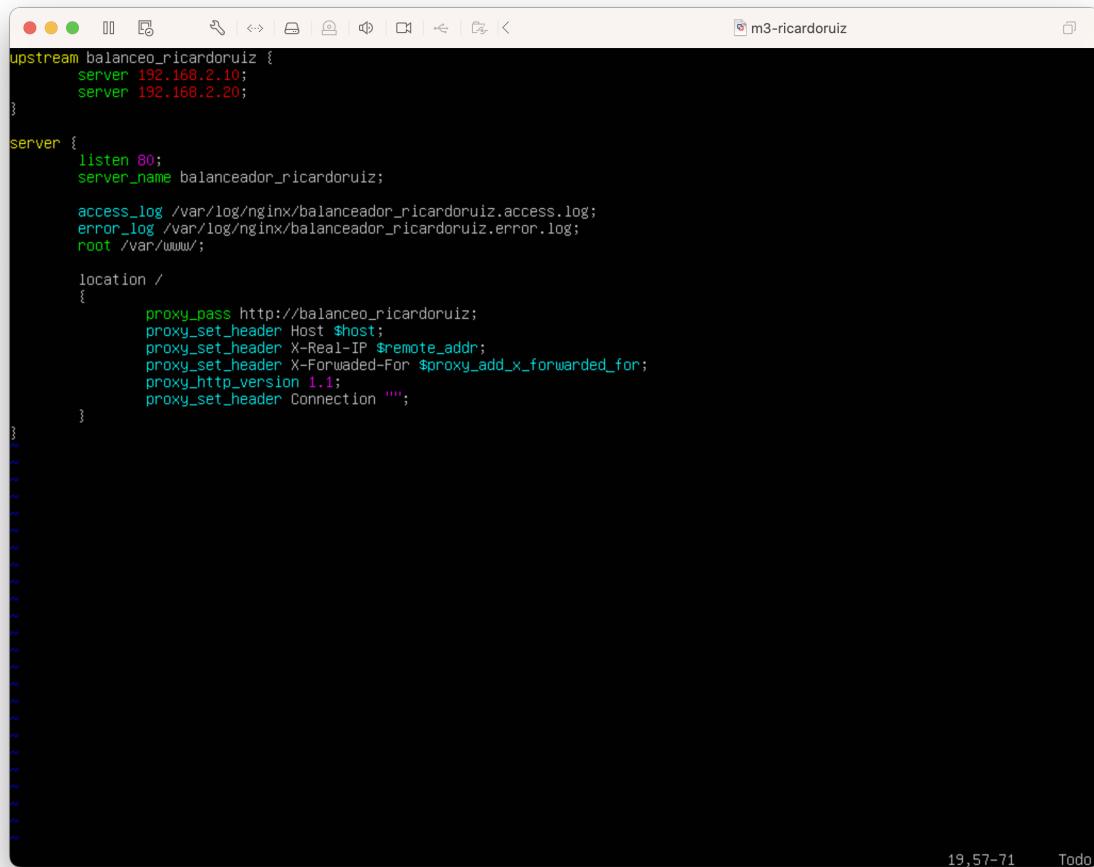
Para definir la granja web de servidores apache escribimos la sección upstream con la IP de las M1 y M2. Es importante que este al principio del archivo de configuración, fuera de la sección server.

```
1 upstream balanceo_ricardoruiz {  
2     server 192.168.2.10;  
3     server 192.168.2.20;  
4 }
```

Debemos definir ahora la sección server para indicar a nginx que use el grupo definido anteriormente en upstream. Para que el proxy_pass funcione correctamente , debemos indicar una conexión de tipo HTTP 1.1 asi como eliminar la cabecera `Connection` para evitar que se pase al servidor final la cabecera que indica el usuario.

```
1 [...]  
2 server {  
3     listen 80;  
4     server_name balanceador_ricardoruiz;  
5     access_log /var/log/nginx/balanceador_ricardoruiz.access.log;  
6     error_log /var/log/nginx/balanceador_ricardoruiz.error.log;  
7     root /var/www/;  
8     location / {  
9         proxy_pass http://balanceo_ricardoruiz;  
10        proxy_set_header Host $host;  
11        proxy_set_header X-Real-IP $remote_addr;  
12        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;  
13        proxy_http_version 1.1;  
14        proxy_set_header Connection "";  
15    }  
16 }
```

Luego la configuración completa quedaría como sigue:



```
upstream balanceo_ricardoruez {
    server 192.168.2.10;
    server 192.168.2.20;
}

server {
    listen 80;
    server_name balanceador_ricardoruez;

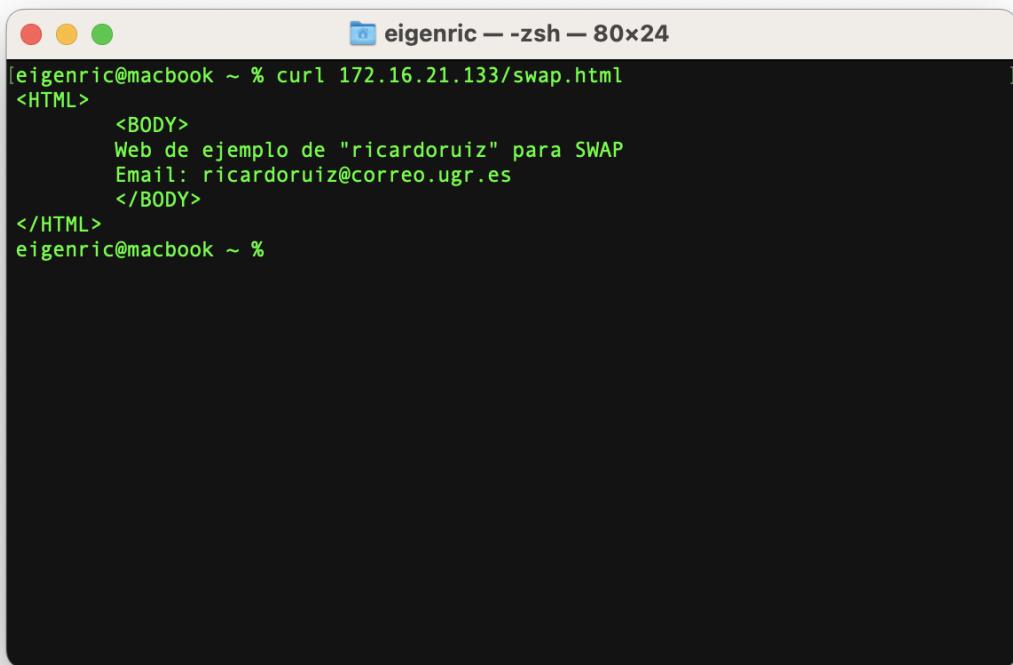
    access_log /var/log/nginx/balanceador_ricardoruez.access.log;
    error_log /var/log/nginx/balanceador_ricardoruez.error.log;
    root /var/www/;

    location /
    {
        proxy_pass http://balanceo_ricardoruez;
        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 "";
    }
}
```

Ejemplo de funcionamiento

La IP accesible desde el Sistema Operativo a M3 es 172.16.21.133.

Podemos comprobar el funcionamiento del balanceador con `curl 172.16.21.133/swap.html`

A screenshot of a macOS terminal window titled "eigenric — -zsh — 80x24". The window shows the command "curl 172.16.21.133/swap.html" being run, and the resulting HTML response. The response contains a title "Web de ejemplo de "ricardoruiz" para SWAP" and an email address "Email: ricardoruiz@correo.ugr.es".

```
[eigenric@macbook ~ % curl 172.16.21.133/swap.html
<HTML>
  <BODY>
    Web de ejemplo de "ricardoruiz" para SWAP
    Email: ricardoruiz@correo.ugr.es
  </BODY>
</HTML>
eigenric@macbook ~ %
```

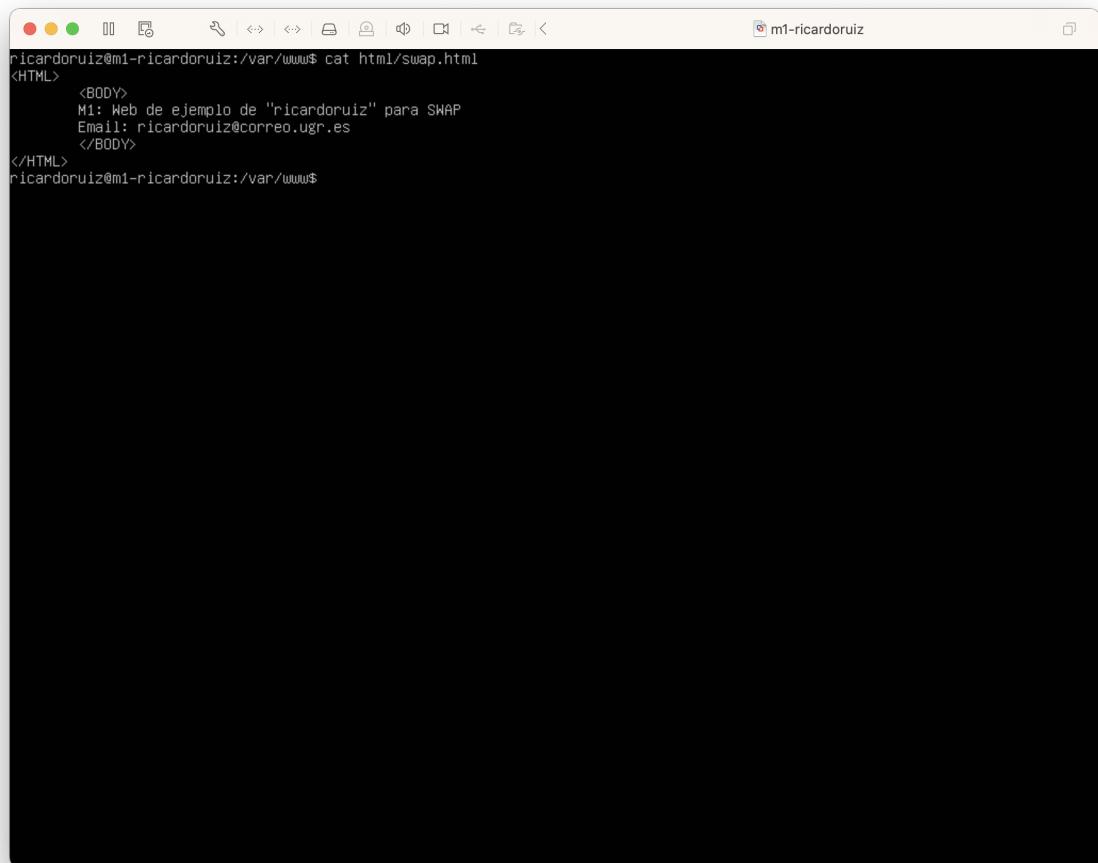
Tarea avanzada: repartir carga en función de pesos

En caso de saber que alguna de las máquinas finales es más potente, podemos modificar la definición del “upstream” para pasarle más tráfico que al resto. Para ello, asignamos un valor numero al modificador “weight”.

Realizamos la tarea avanzada haciendo que cada tres peticiones que lleguen al balanceador, la máquina M1 reciba dos y la M2 una. Para ello, modificamos el fichero de configuración de nginx como sigue:

```
1 upstream balanceo_ricardoruiz {
2   server 192.168.2.10 weight=2;
3   server 192.168.2.20 weight=1;
4 }
```

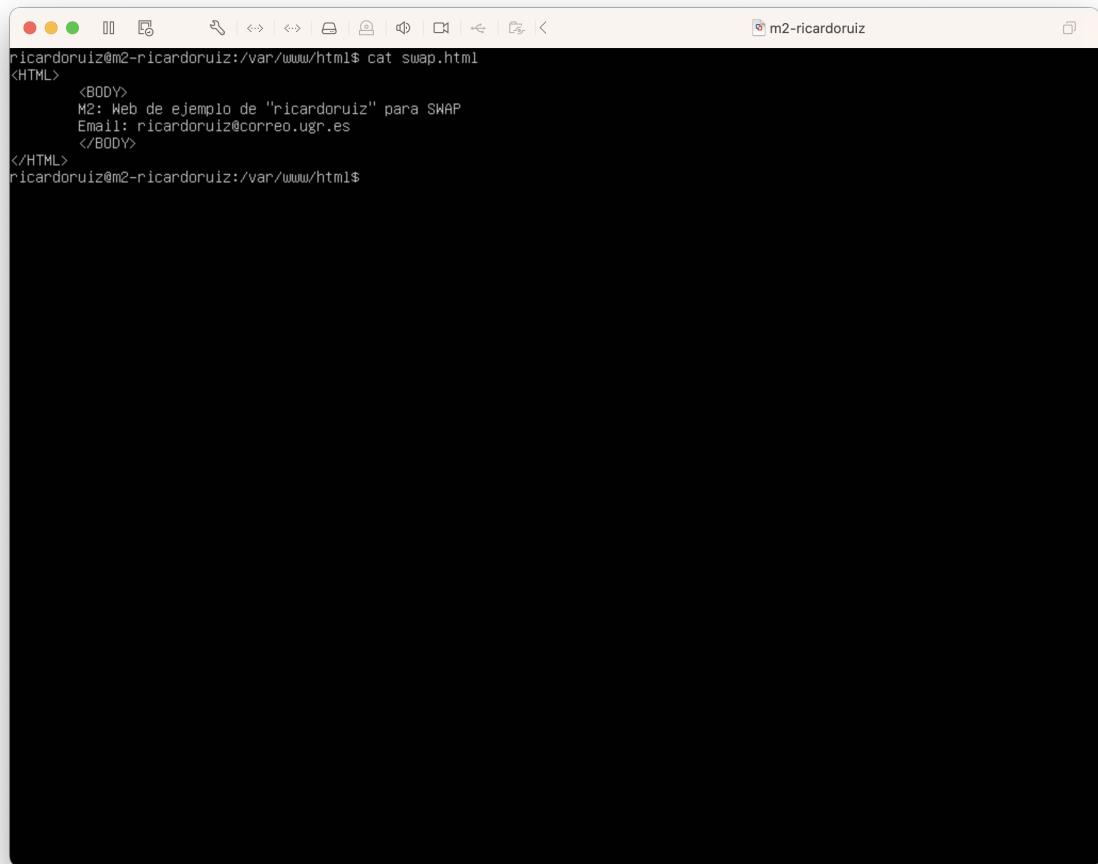
Para comprobarlo, modificamos `swap.html` las máquinas finales para identificarlas.



A screenshot of a terminal window titled "m1-ricardoruez". The window shows the command `cat html/swap.html` and its output. The output is an HTML document with the following content:

```
<HTML>
  <BODY>
    M1: Web de ejemplo de "ricardoruez" para SWAP
    Email: ricardoruez@correo.ugr.es
  </BODY>
</HTML>
```

Figura 2: swap.html en M1



A screenshot of a terminal window with a dark background. The window title is "m2-ricardoruez". The terminal shows the following text:

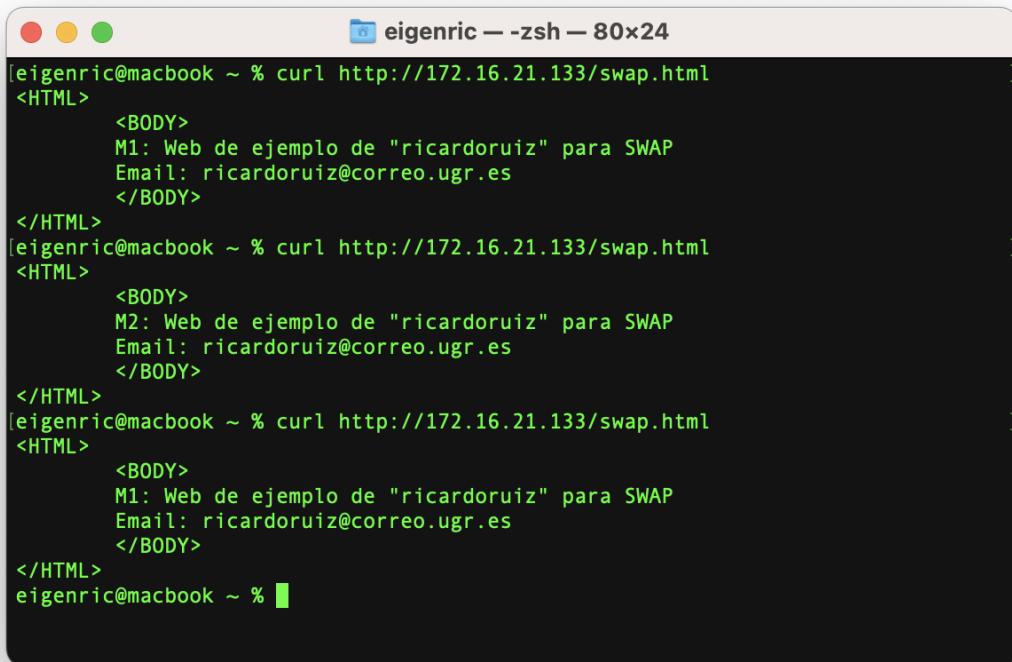
```
ricardoruez@m2-ricardoruez:~$ cat swap.html
<HTML>
  <BODY>
    M2: Web de ejemplo de "ricardoruez" para SWAP
    Email: ricardoruez@correo.ugr.es
  </BODY>
</HTML>
ricardoruez@m2-ricardoruez:~$
```

Figura 3: swap.html en M2

Desactivamos también la tarea cron de sincronización con rsync para evitar que se sobreesciban los cambios.

Figura 4: Desactivación de la tarea cron

Realizamos tres peticiones y comprobamos que se sigue el **Algoritmo Round Robin**, acabando dos de ellas en M2:



```
[eigenric@macbook ~ % curl http://172.16.21.133/swap.html
<HTML>
<BODY>
M1: Web de ejemplo de "ricardoruiz" para SWAP
Email: ricardoruiz@correo.ugr.es
</BODY>
</HTML>
[eigenric@macbook ~ % curl http://172.16.21.133/swap.html
<HTML>
<BODY>
M2: Web de ejemplo de "ricardoruiz" para SWAP
Email: ricardoruiz@correo.ugr.es
</BODY>
</HTML>
[eigenric@macbook ~ % curl http://172.16.21.133/swap.html
<HTML>
<BODY>
M1: Web de ejemplo de "ricardoruiz" para SWAP
Email: ricardoruiz@correo.ugr.es
</BODY>
</HTML>
eigenric@macbook ~ %
```

Balanceo de carga con HAProxy

HAProxy es un software de balanceo de carga y proxy inverso de alta disponibilidad que se utiliza para distribuir el tráfico de red a varios servidores backend y mejorar la escalabilidad y la fiabilidad de las aplicaciones web.

Instalación de HAProxy

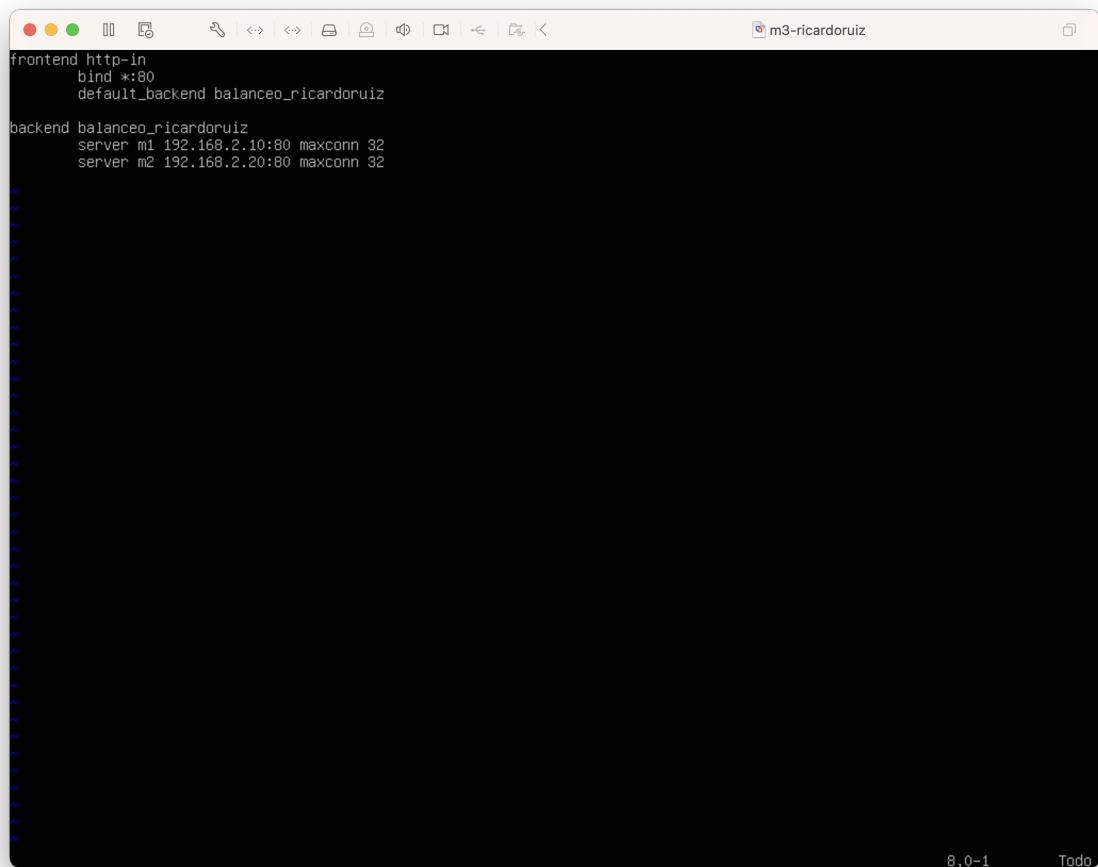
Instalamos HAProxy con `sudo apt install haproxy`.

Configuración básica de haproxy como balanceador

La configuración de HAProxy se encuentra en el fichero `/etc/haproxy/haproxy.cfg`. Debemos modificarlo para indicarle cuales son nuestros servidores (backend) y qué peticiones balancear.

La siguiente configuración hace que HAProxy escuche en el puerto 80 y redirige el tráfico a las máquinas M1 y M2.

```
1 frontend http-in
2   bind *:80
3   default_backend balanceo_ricardoruz
4
5 backend balanceo_ricardoruz
6   balance roundrobin
7   server m1 192.168.2.10:80 maxconn 32
8   server m2 192.168.2.20:80 maxconn 32
```



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

backend balanceo_ricardoruz
  server m1 192.168.2.10:80 maxconn 32
  server m2 192.168.2.20:80 maxconn 32
```

Ejemplo de funcionamiento

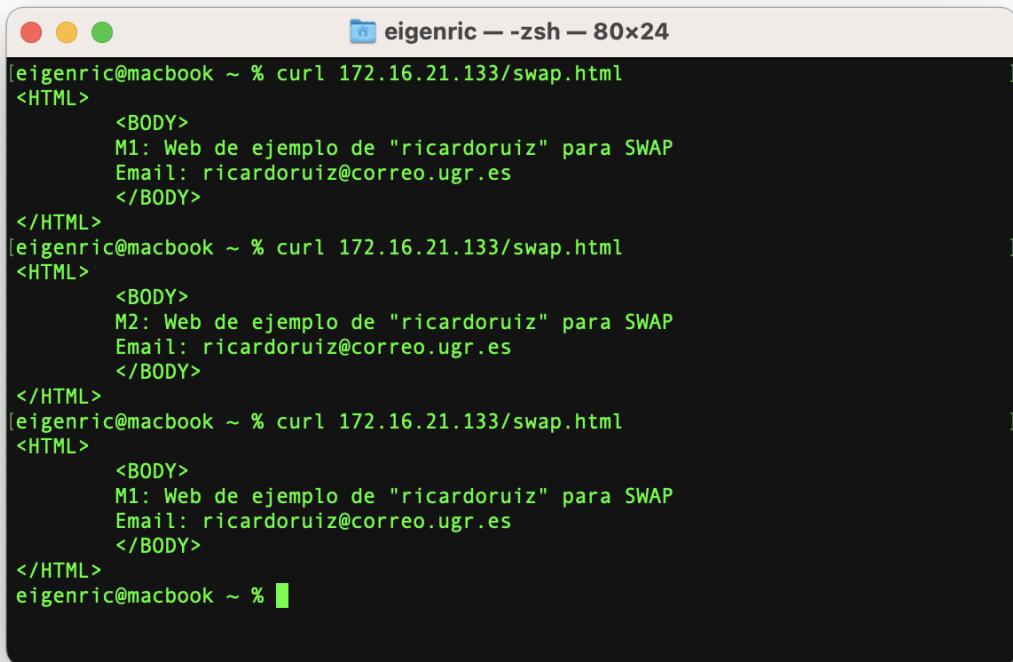
En primer lugar, debemos desactivar el servicio de NGINX para que no haya conflictos con el puerto 80.

```
1 ricardoruiz@m3-ricardoruiz $ sudo systemctl stop nginx
```

Y lanzamos HAProxy con

```
1 ricardoruiz@m3-ricardoruiz $ sudo haproxy -f /etc/haproxy/haproxy.cfg
2 ricardoruiz@m3-ricardoruiz $ sudo service haproxy restart
```

En efecto, comprobamos que se balancea el tráfico entre M1 y M2 siguiendo el algoritmo Round Robin:



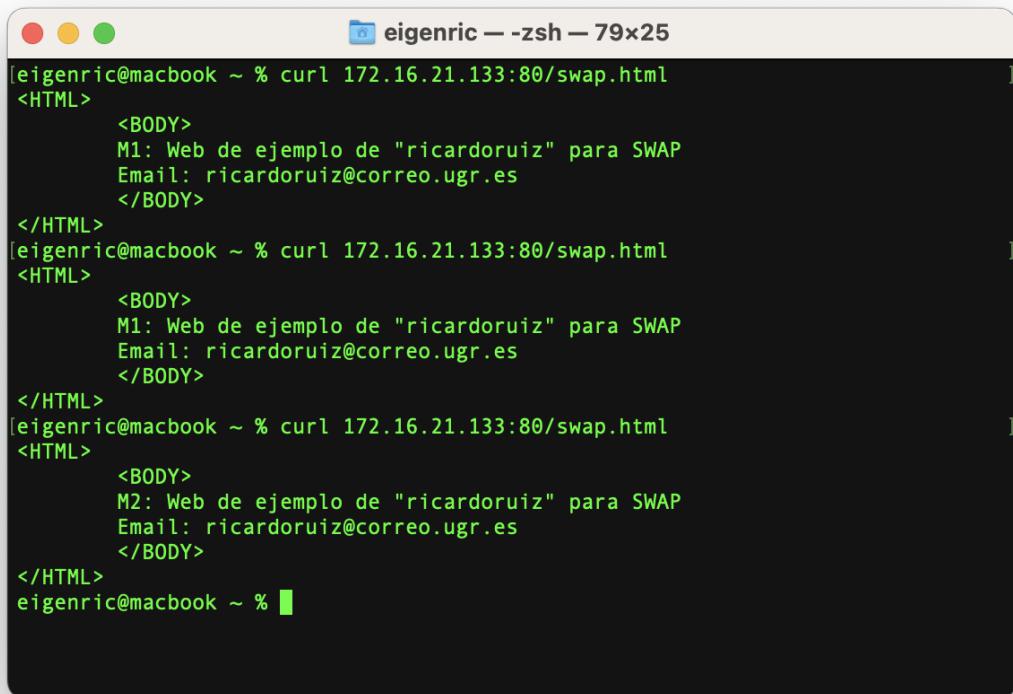
```
[eigenric@macbook ~ % curl 172.16.21.133/swap.html
<HTML>
<BODY>
M1: Web de ejemplo de "ricardoruiz" para SWAP
Email: ricardoruiz@correo.ugr.es
</BODY>
</HTML>
[eigenric@macbook ~ % curl 172.16.21.133/swap.html
<HTML>
<BODY>
M2: Web de ejemplo de "ricardoruiz" para SWAP
Email: ricardoruiz@correo.ugr.es
</BODY>
</HTML>
[eigenric@macbook ~ % curl 172.16.21.133/swap.html
<HTML>
<BODY>
M1: Web de ejemplo de "ricardoruiz" para SWAP
Email: ricardoruiz@correo.ugr.es
</BODY>
</HTML>
eigenric@macbook ~ %
```

Tarea Avanzada: repartir carga en función de pesos

Para configurar HAProxy para que distribuya la carga de manera que M1 reciba el doble de peticiones que M2, debemos modificar el fichero de configuración de HAProxy para que quede como sigue:

```
1 frontend http-in
2   bind *:80
3   default_backend balanceo_ricardoruez
4
5 backend balanceo_ricardoruez
6   balance roundrobin
7   server m1 192.168.2.10:80 weight 2 maxconn 32
8   server m2 192.168.2.20:80 weight 1 maxconn 32
```

Relanzamos HAProxy como se hizo anteriormente y realizamos tres peticiones, comprobando que se sigue el **Algoritmo Round Robin** recibiendo M1 el doble que M2:



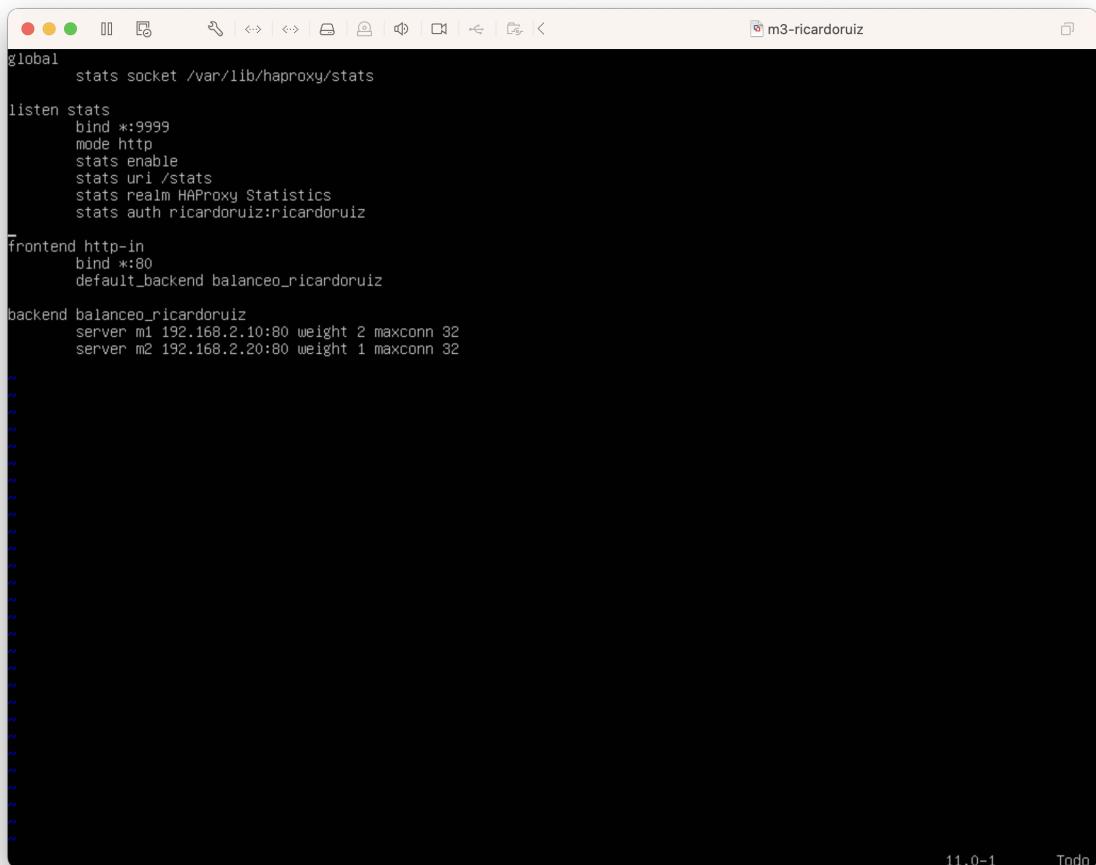
```
[eigenric@macbook ~ % curl 172.16.21.133:80/swap.html
<HTML>
  <BODY>
    M1: Web de ejemplo de "ricardoruez" para SWAP
    Email: ricardoruez@correo.ugr.es
  </BODY>
</HTML>
[eigenric@macbook ~ % curl 172.16.21.133:80/swap.html
<HTML>
  <BODY>
    M1: Web de ejemplo de "ricardoruez" para SWAP
    Email: ricardoruez@correo.ugr.es
  </BODY>
</HTML>
[eigenric@macbook ~ % curl 172.16.21.133:80/swap.html
<HTML>
  <BODY>
    M2: Web de ejemplo de "ricardoruez" para SWAP
    Email: ricardoruez@correo.ugr.es
  </BODY>
</HTML>
eigenric@macbook ~ %
```

Figura 5: Distribución con pesos en HAProxy

Tarea Avanzada: Módulo de estadísticas.

Una opción interesante es habilitar el módulo de estadísticas del balanceador. Se puede habilitar añadiendo la configuración en el archivo </etc/haproxy/haproxy.cf>

```
1 global
2     stats socket /var/lib/haproxy/stats
3
4 listen stats
5     bind *:9999
6     mode http
7     stats enable
8     stats uri /stats
9     stats realm HAProxy Statistics
10    stats auth ricardoruiz:ricardoruiz
```



The screenshot shows a terminal window with the following content:

```
global
  stats socket /var/lib/haproxy/stats

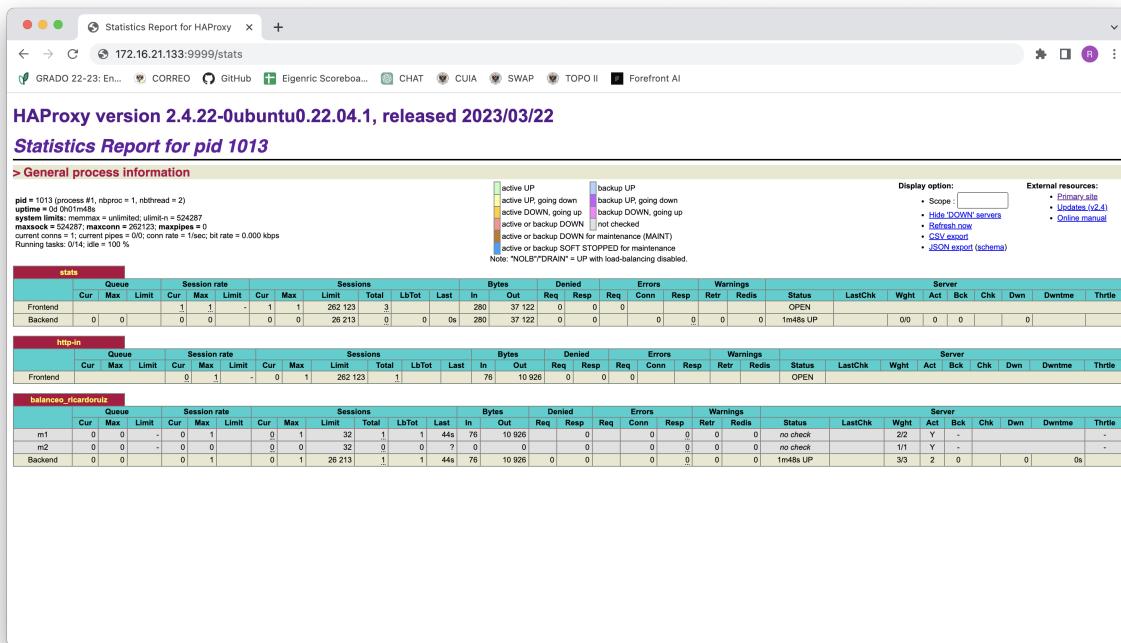
listen stats
  bind *:9999
  mode http
  stats enable
  stats uri /stats
  stats realm HAProxy Statistics
  stats auth ricardoruiz:ricardoruiz

frontend http-in
  bind *:80
  default_backend balanceo_ricardoruiz

backend balanceo_ricardoruiz
  server m1 192.168.2.10:80 weight 2 maxconn 32
  server m2 192.168.2.20:80 weight 1 maxconn 32
```

The terminal window has a dark background and light-colored text. It includes standard OS X window controls (red, yellow, green buttons) and a menu bar with "m3-ricardoruiz". The bottom right corner shows "11,0-1" and "Todo".

Y tras ingresar el usuario y contraseña ricardoruiz / ricardoruiz, accedemos a la página de estadísticas:



Tarea 2. Alta carga con Apache Benchmark

Para medir el rendimiento de un servidor necesitaremos una herramienta que ejecutar en los clientes para crear una carga HTTP específica.

Dado que el numero de usuarios puede ser alto lo recomendable es usar programas de línea de comandos que sobrecarguen lo mínimo posible las máquinas que estamos usando.

En esta tarea, usaremos la herramienta Apache Benchmark para someter a la granja web a una alta carga en dos escenarios: nginx con round-robin y haproxy con round-robin.

Es conveniente ejecutar los benchmark en otra máquina distinta a las involucradas en la granja web (servidores web o balanceador), para que el consumo de recursos no afecte al rendimiento

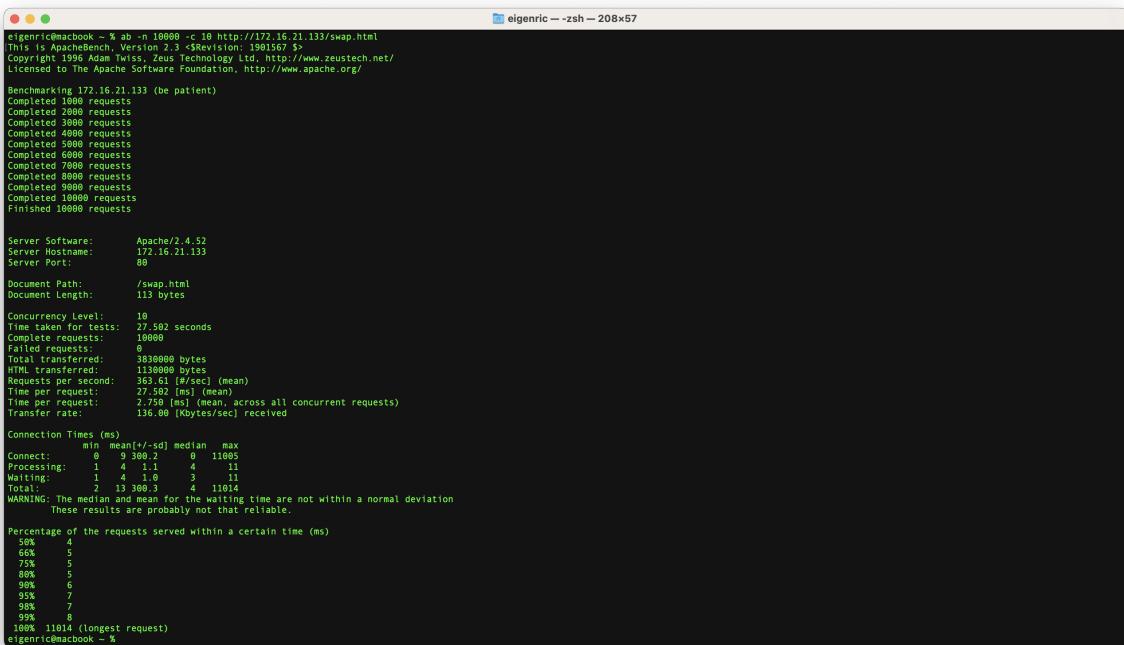
HAProxy con Round Robin

Realizamos el benchmark con la siguiente orden:

```
1 ab -n 10000 -c 10 http://172.16.21.133/swap.html
```

La opción **-n** indica el número de peticiones y **-c** el número de peticiones concurrentes.

Obtenemos los siguientes resultados:



```
eigenric@macbook ~ % ab -n 10000 -c 10 http://172.16.21.133/swap.html
This is ApacheBench, Version 2.3 <Revision: 1901567 $>
copyright 1996 Adam Twiss, Zeos Technology Ltd: http://www.zeosoft.net/
Licensed to The Apache Software Foundation: http://www.apache.org/
Benchmarking 172.16.21.133 (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/2.4.52
Server Hostname:     172.16.21.133
Server Port:          80

Document Path:        /swap.html
Document Length:      113 bytes

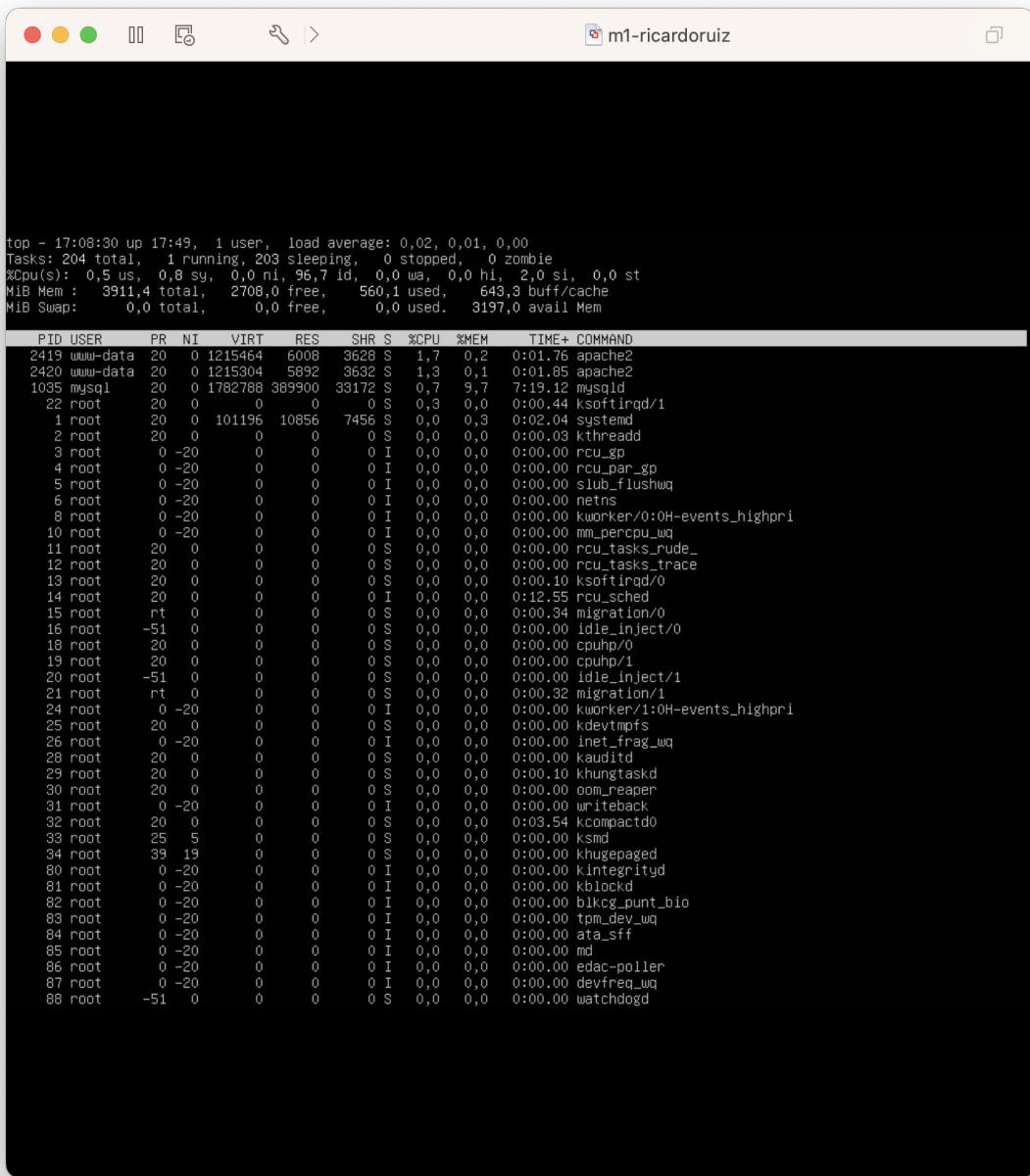
Concurrency Level:    10
Time taken for tests: 27.582 seconds
Complete requests:   10000
Failed requests:    0
Total transferred:  3830000 bytes
HTML transferred:   1130000 bytes
Requests per second: 363.00 [#/sec] (mean)
Time per request:   27.582 [ms] (mean)
Time per request:   2.758 [ms] (mean, across all concurrent requests)
Transfer rate:       136.80 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median  max
Connect:        0    9 380.2    0  11085
Processing:     1    4  11.1    4   11
Waiting:        1    4  10.0    3   11
Total:          2   13 380.3    4  11014
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%    4
  66%    5
  75%    5
  80%    5
  90%    6
  95%    7
  98%    8
  100%  11014 (longest request)
eigenric@macbook ~ %
```

Figura 6: Resultado Apache Benchmark

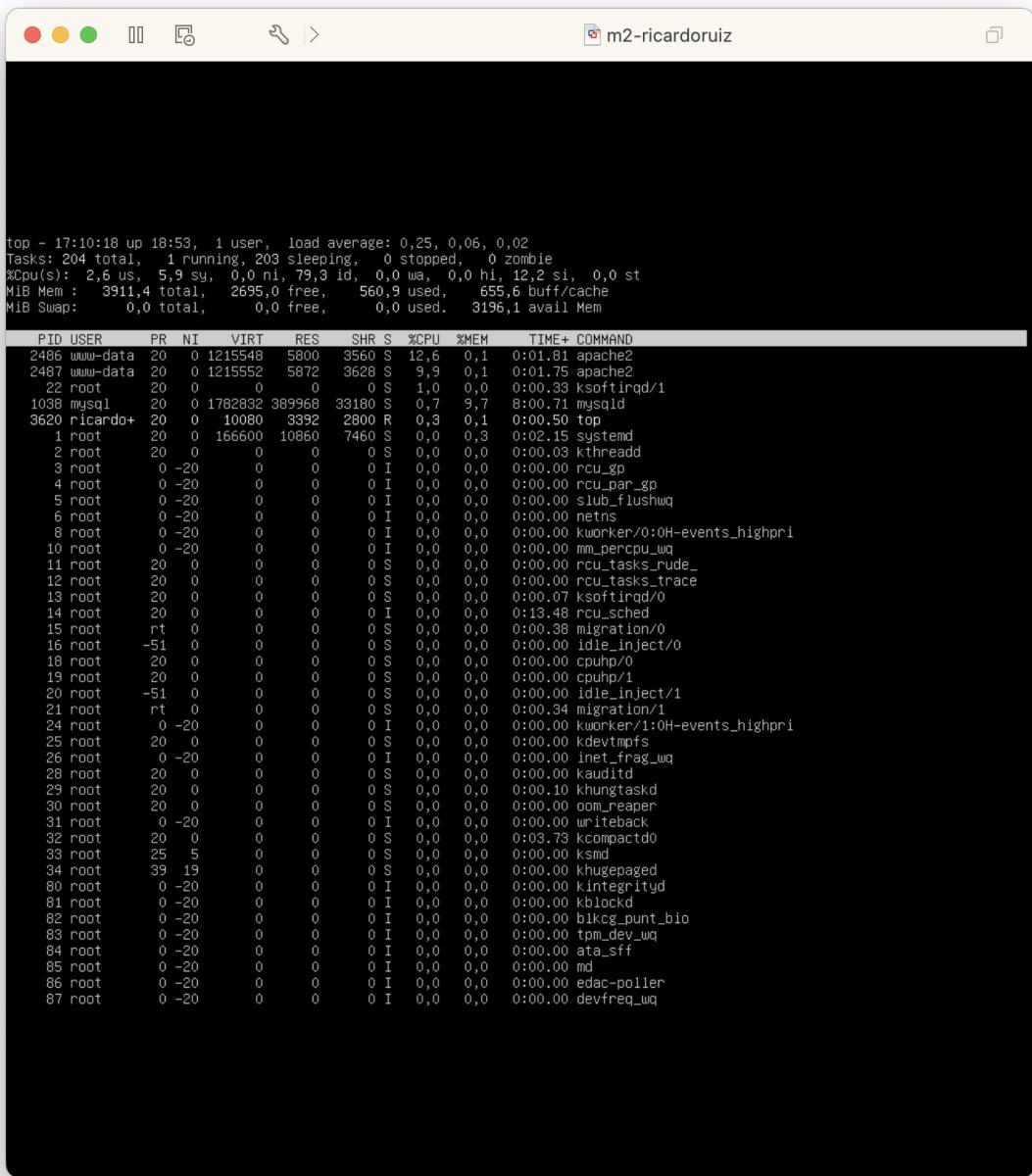
De hecho, ejecutando el comando `top` en las máquinas virtuales M1 y M2, podemos comprobar que se están recibiendo peticiones de forma alternada:



```
top - 17:08:30 up 17:49,  1 user,  load average: 0,02, 0,01, 0,00
Tasks: 204 total,   1 running, 203 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0,5 us,  0,8 sy,  0,0 ni, 96,7 id,  0,0 wa,  0,0 hi,  2,0 si,  0,0 st
Mem: 3911,4 total, 2708,0 free, 560,1 used, 643,3 buff/cache
Swap: 0,0 total, 0,0 free, 0,0 used. 3197,0 avail Mem

PID USER      PR  NI    VIRT    RES    SHR %CPU %MEM TIME+ COMMAND
2419 www-data  20   0 1215464  6008  3628 S  1,7  0,2 0:01,76 apache2
2420 www-data  20   0 1215304  5892  3632 S  1,3  0,1 0:01,85 apache2
1035 mysql     20   0 1782788 389900 33172 S  0,7  9,7 7:19,12 mysqld
 22 root      20   0      0      0      0 S  0,3  0,0 0:00,44 ksoftirqd/1
  1 root      20   0 101196 10856  7456 S  0,0  0,3 0:02,04 systemd
  2 root      20   0      0      0      0 S  0,0  0,0 0:00,03 kthreadd
  3 root      0 -20      0      0      0 I  0,0  0,0 0:00,00 rcu_gp
  4 root      0 -20      0      0      0 I  0,0  0,0 0:00,00 rcu_par_gp
  5 root      0 -20      0      0      0 I  0,0  0,0 0:00,00 slub_fflushwq
  6 root      0 -20      0      0      0 I  0,0  0,0 0:00,00 netns
  8 root      0 -20      0      0      0 I  0,0  0,0 0:00,00 kworker/0:0H-events_highpri
 10 root      0 -20      0      0      0 I  0,0  0,0 0:00,00 mm_percpu_wq
 11 root      20   0      0      0      0 S  0,0  0,0 0:00,00 rcu_tasks_rude_
 12 root      20   0      0      0      0 S  0,0  0,0 0:00,00 rcu_tasks_trace
 13 root      20   0      0      0      0 S  0,0  0,0 0:00,10 ksoftirqd/0
 14 root      20   0      0      0      0 I  0,0  0,0 0:12,55 rcu_sched
 15 root      rt   0      0      0      0 S  0,0  0,0 0:00,30 migration/0
 16 root      -51   0      0      0      0 S  0,0  0,0 0:00,00 idle_inject/0
 18 root      20   0      0      0      0 S  0,0  0,0 0:00,00 cpuhp/0
 19 root      20   0      0      0      0 S  0,0  0,0 0:00,00 cpuhp/1
 20 root      -51   0      0      0      0 S  0,0  0,0 0:00,00 idle_inject/1
 21 root      rt   0      0      0      0 S  0,0  0,0 0:00,32 migration/1
 24 root      0 -20      0      0      0 I  0,0  0,0 0:00,00 kworker/1:0H-events_highpri
 25 root      20   0      0      0      0 S  0,0  0,0 0:00,00 kdevtmpfs
 26 root      0 -20      0      0      0 I  0,0  0,0 0:00,00 inet_frag_wq
 28 root      20   0      0      0      0 S  0,0  0,0 0:00,00 kauditd
 29 root      20   0      0      0      0 S  0,0  0,0 0:00,10 khungtaskd
 30 root      20   0      0      0      0 S  0,0  0,0 0:00,00 oom_reaper
 31 root      0 -20      0      0      0 I  0,0  0,0 0:00,00 writeback
 32 root      20   0      0      0      0 S  0,0  0,0 0:03,50 kcompactd0
 33 root      25   5      0      0      0 S  0,0  0,0 0:00,00 ksmd
 34 root      39   19      0      0      0 S  0,0  0,0 0:00,00 khugepaged
 80 root      0 -20      0      0      0 I  0,0  0,0 0:00,00 kintegrityd
 81 root      0 -20      0      0      0 I  0,0  0,0 0:00,00 kblockd
 82 root      0 -20      0      0      0 I  0,0  0,0 0:00,00 blkcg_punt_bio
 83 root      0 -20      0      0      0 I  0,0  0,0 0:00,00 tpm_dev_wq
 84 root      0 -20      0      0      0 I  0,0  0,0 0:00,00 ata_sff
 85 root      0 -20      0      0      0 I  0,0  0,0 0:00,00 md
 86 root      0 -20      0      0      0 I  0,0  0,0 0:00,00 edac-poller
 87 root      0 -20      0      0      0 I  0,0  0,0 0:00,00 devfreq_wq
 88 root      -51   0      0      0      0 S  0,0  0,0 0:00,00 watchdogd
```

Figura 7: Comando top en M1



```

top - 17:10:18 up 18:53,  1 user,  load average: 0,25, 0,06, 0,02
Tasks: 204 total,  1 running, 203 sleeping,  0 stopped,  0 zombie
%Cpu(s): 2,6 us, 5,9 sy, 0,0 ni, 79,3 id, 0,0 wa, 0,0 hi, 12,2 si, 0,0 st
Mem: 3911,4 total, 2695,0 free, 560,9 used, 655,6 buff/cache
Swap: 0,0 total, 0,0 free, 0,0 used. 3196,1 avail Mem

PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
2486 www-data  20   0 1215548  5800  3560 S 12,6  0,1 0:01:81 apache2
2487 www-data  20   0 1215552  5872  3628 S  9,9  0,1 0:01:75 apache2
 2  root      20   0      0      0      0 S  1,0  0,0 0:00:38 ksoftirqd/1
1038 mysql     20   0 1782832 389968 33180 S  0,7  9,7 8:00:71 mysqld
3620 ricardo+ 20   0 10080  3392  2800 R  0,3  0,1 0:00:50 top
 1 root      20   0 166600 10860  7460 S  0,0  0,3 0:02:15 systemd
 2 root      20   0      0      0      0 S  0,0  0,0 0:00:03 kthreadd
 3 root      0 -20      0      0      0 I  0,0  0,0 0:00:00 rcu_gp
 4 root      0 -20      0      0      0 I  0,0  0,0 0:00:00 rcu_par_gp
 5 root      0 -20      0      0      0 I  0,0  0,0 0:00:00 slab_flushwq
 6 root      0 -20      0      0      0 I  0,0  0,0 0:00:00 netns
 8 root      0 -20      0      0      0 I  0,0  0,0 0:00:00 kworker/0:0H-events_highpri
10 root      0 -20      0      0      0 I  0,0  0,0 0:00:00 mm_percpu_wq
11 root      20   0      0      0      0 S  0,0  0,0 0:00:00 rcu_tasks_rude_
12 root      20   0      0      0      0 S  0,0  0,0 0:00:00 rcu_tasks_trace
13 root      20   0      0      0      0 S  0,0  0,0 0:00:07 ksoftirqd/0
14 root      20   0      0      0      0 I  0,0  0,0 0:13:48 rcu_sched
15 root      rt   0      0      0      0 S  0,0  0,0 0:00:38 migration/0
16 root      -51   0      0      0      0 S  0,0  0,0 0:00:00 idle_inject/0
18 root      20   0      0      0      0 S  0,0  0,0 0:00:00 cpuhp/0
19 root      20   0      0      0      0 S  0,0  0,0 0:00:00 cpuhp/1
20 root      -51   0      0      0      0 S  0,0  0,0 0:00:00 idle_inject/1
21 root      rt   0      0      0      0 S  0,0  0,0 0:00:30 migration/1
24 root      0 -20      0      0      0 I  0,0  0,0 0:00:00 kworker/1:0H-events_highpri
25 root      20   0      0      0      0 S  0,0  0,0 0:00:00 kdevtmpfs
26 root      0 -20      0      0      0 I  0,0  0,0 0:00:00 inet_frag_wq
28 root      20   0      0      0      0 S  0,0  0,0 0:00:00 kaudited
29 root      20   0      0      0      0 S  0,0  0,0 0:00:10 khungtaskd
30 root      20   0      0      0      0 S  0,0  0,0 0:00:00 oom_reaper
31 root      0 -20      0      0      0 I  0,0  0,0 0:00:00 writeback
32 root      20   0      0      0      0 S  0,0  0,0 0:03:73 kcompactd0
33 root      25   5      0      0      0 S  0,0  0,0 0:00:00 ksmd
34 root      39   19     0      0      0 S  0,0  0,0 0:00:00 khugepaged
80 root      0 -20      0      0      0 I  0,0  0,0 0:00:00 kintegrityd
81 root      0 -20      0      0      0 I  0,0  0,0 0:00:00 kblockd
82 root      0 -20      0      0      0 I  0,0  0,0 0:00:00 blkcg_punt_bio
83 root      0 -20      0      0      0 I  0,0  0,0 0:00:00 tpm_dev_wq
84 root      0 -20      0      0      0 I  0,0  0,0 0:00:00 ata_sff
85 root      0 -20      0      0      0 I  0,0  0,0 0:00:00 md
86 root      0 -20      0      0      0 I  0,0  0,0 0:00:00 edac-poller
87 root      0 -20      0      0      0 I  0,0  0,0 0:00:00 devfreq_wq

```

Figura 8: Comando top en M2

Podemos también comprobar que se balancea la carga entre M1 y M2 con el archivo `getload.php` proporcionado en prado:

```
1 ricardoruiz@m3-ricardoruiz $ ab -n 10000 -c 10 http://172.16.21.133/getload.php
```

Y comprobamos mediante la carga de CPU que M1 (IP 172.16.21.132) recibe el doble de peticiones que M2 (172.16.21.130):

Figura 9: getload.php mientras se ejecuta ab

NGINX con Round Robin

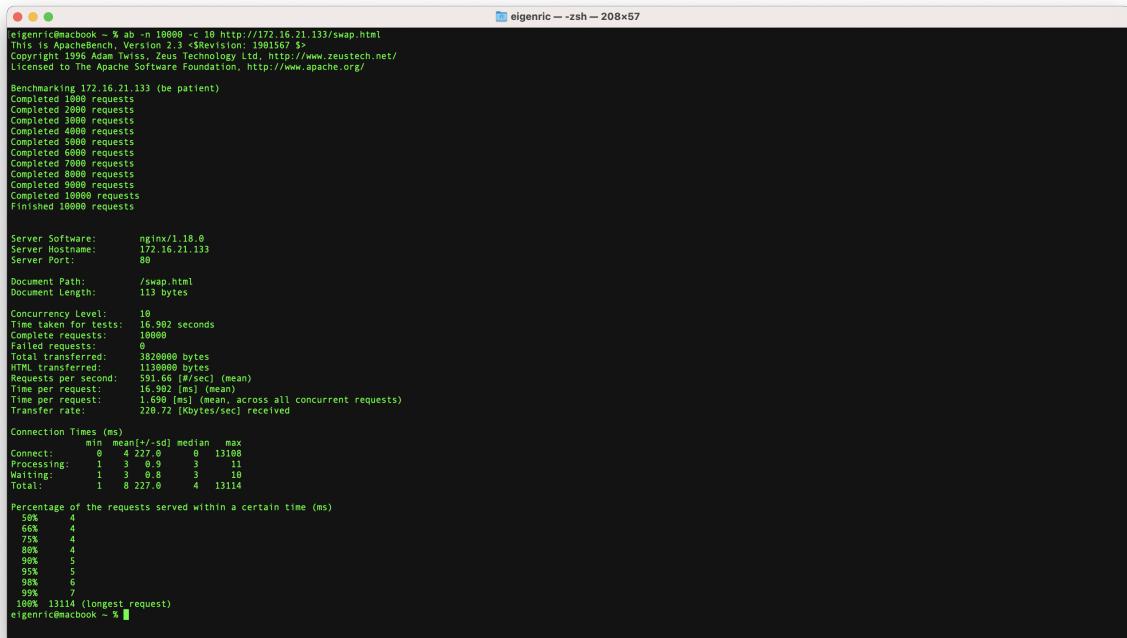
Desactivamos el servicio de HAProxy y activamos el de NGINX:

```
1 ricardoruiz@m3-ricardoruiz $ sudo systemctl stop haproxy  
2 ricardoruiz@m3-ricardoruiz $ sudo systemctl start nginx
```

Y realizamos el benchmark con la siguiente orden:

```
1 ricardoruiz@m3-ricardoruiz $ ab -n 10000 -c 10 http://172.16.21.133/
  swap.html
```

obteniendo los siguientes resultados:



```
eigenric@macbook ~ % ab -n 10000 -c 10 http://172.16.21.133/swap.html
This is ApacheBench, Version 2.3 <$Revision: 1901567 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/
Benchmarking 172.16.21.133 (be patient)
Completed 1000 requests
Completed 2000 requests
Completed 3000 requests
Completed 4000 requests
Completed 5000 requests
Completed 6000 requests
Completed 7000 requests
Completed 8000 requests
Completed 9000 requests
Completed 10000 requests
Finished 10000 requests

Server Software:      nginx/1.18.0
Server Hostname:     172.16.21.133
Server Port:          80

Document Path:        /swap.html
Document Length:      113 bytes

Concurrency Level:    10
Time taken for tests: 16.982 seconds
Complete requests:   10000
Failed requests:     0
Total transferred:   3820000 bytes
HTML transferred:   1130000 bytes
Requests per second: 591.66 [#/sec] (mean)
Time per request:   16.982 [ms] (mean)
Time per request:   1.698 [ms] (min)
Time per request:   13.114 [ms] (max) across all concurrent requests
Transfer rate:        228.72 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median  max
Connect:        0  227.0    0  13114
Processing:     1    3  0.9    3   11
Waiting:        1    3  0.8    3   10
Total:          1    8 227.0    4  13114

Percentage of the requests served within a certain time (ms)
  50%   4
  66%   4
  75%   4
  80%   4
  90%   5
  95%   5
  99%   6
  99.9% 7
  100% 13114 (longest request)
eigenric@macbook ~ %
```

Figura 10: Resultado Apache Benchmark

Tarea Avanzada: Balanceador de carga con Gobetween

Gobetween es un balanceador de carga y proxy inverso de alta disponibilidad escrito en Go. Es una alternativa ligera a HAProxy y NGINX.

Instalación de Gobetween

Compilamos de repositorio fuente:

```
1 $ git clone git@github.com:yyyar/gobetween.git
2 $ make
3 $ make run
```

Tarea 3. Análisis Comparativo

En esta tarea, realizaremos un análisis comparativo de los resultados obtenidos en la tarea anterior, considerando el número de peticiones por unidad de tiempo.

Peticiones por segundo	
NGINX	591.66
HAProxy	363.61

El resultado de Apache Benchmark revela diferencias significativas en el rendimiento entre NGINX y HAProxy en términos de peticiones por segundo. Según los resultados obtenidos, NGINX alcanza un promedio de 591.66 peticiones por segundo, mientras que HAProxy logra un promedio de 363.61 peticiones por segundo.

Esto indica que NGINX muestra un rendimiento superior en comparación con HAProxy en términos de capacidad para manejar un mayor número de peticiones por unidad de tiempo. La diferencia de aproximadamente 228 peticiones por segundo entre ambas soluciones destaca la eficiencia y escalabilidad de NGINX en la gestión de la carga de trabajo.

Si se requiere un alto rendimiento y una mayor capacidad de procesamiento de peticiones, NGINX se presenta como una opción más sólida en comparación con HAProxy. Sin embargo, es importante tener en cuenta que los resultados pueden variar en función de los escenarios y configuraciones específicas de implementación.

En resumen, NGINX supera a HAProxy en términos de rendimiento y capacidad de manejo de peticiones, lo que lo convierte en una elección preferida en entornos que requieren una alta carga y distribución eficiente del tráfico.

Referencias

- **Apache Benchmark** <http://httpd.apache.org/docs/2.2/programs/ab.html>
- **HAProxy** <https://www.haproxy.org/>
- **HAProxy Documentation** <https://cbonte.github.io/haproxy-dconv/>
- **NGINX** <https://nginx.org/>
- **NGINX como balanceador de carga** <https://www.nginx.com/resources/glossary/load-balancing/>