Balanceo de carga con HAproxy

CDA 2022/23

1 de noviembre de 2022

Índice

1. Descripción **1**

2. Entorno de prácticas **2**

2.1. Software de virtualización VIRTUALBOX 2

2.2. Imágenes a utilizar 2

2.3. Máquinas virtuales y redes creadas 3

3. HAProxy **3**

3.1. Funcionamiento y funcionalidades 3

3.2. Configuración 4

4. EJERCICIO ENTREGABLE: Balanceo de carga en servidores Apache con HAproxy **5**

4.1. Pasos previos 5

4.2. Tarea 1: evaluar rendimiento de un servidor Apache sin balanceo 7

4.3. Tarea 2: configurar y evaluar balanceo de carga con dos servidores Apache 8

4.4. Tarea 3: configurar la persistencia de conexiones Web (*sticky sessions*) 10

5. Documentación a entregar **12**

1. Descripción

Uso de HAproxy como balanceador de carga a nivel de aplicación para servidores web.

- Evaluar la mejora en rendimiento respecto al uso de un único servidor derivada del balanceo de carga
- Comprobar el uso de *sticky cookies* de HAproxy como mecanismo para soportar aplicaciones web "con estado"

Recursos complementarios

- Web de HAproxy
- Manual de la versión 2.2

2. Entorno de prácticas

2.1. Software de virtualización VIRTUALBOX

En estas prácticas se empleará el software de virtualización VIRTUALBOX para simular los equipos GNU/Linux sobre los que se realizarán las pruebas.

- Página principal: `http://virtualbox.org`
- Más información: `http://es.wikipedia.org/wiki/Virtualbox`

2.2. Imágenes a utilizar

1. Scripts de instalación
   - para GNU/Linux: ejercicio-haproxy.sh
     `alumno@pc: $ sh ejercicio-haproxy.sh`
   - para MS windows: ejercicio-haproxy.ps1
     `Powershell.exe -executionpolicy bypass -file ejercicio-haproxy.ps1`

   Notas:
   - Se pedirá un identificador (sin espacios) para poder reutilizar las versiones personalizadas de las imágenes creadas (usad por ejemplo el nombre del grupo de prácticas o el login LDAP)
   - En ambos scripts la variable `$DIR_BASE` especifica donde se descargarán las imágenes y se crearán las MVs.
     Por defecto en GNU/Linux será en `$HOME/CDA2223` y en Windows en `C:/CDA2223`.
     Puede modificarse antes de lanzar los scripts para hacer la instalación en otro directorio más conveniente (disco externo, etc)
   - Es posible descargar las imágenes comprimidas manualmente (o intercambiarlas con USB), basta descargar los archivos con extensión `.vdi.zip` de `http://ccia.esei.uvigo.es/docencia/CDA/2223/practicas/` y copiarlos en el directorio anterior (`$DIR_BASE`) para que el script haga el resto.
   - Si no lo hacen desde el script anterior, se pueden arrancar las instancias VIRTUALBOX desde el interfaz gráfico de VirtualBOX o desde la línea de comandos con `VBoxManage startvm <nombre MV>_<id>`
2. Imágenes descargadas
   - **base_cda.vdi** (1,3 GB comprimida, 4,5 GB descomprimida): Imagen genérica (común a todas las MVs) que contiene las herramientas a utilizar
     Contiene un sistema Debian 11 con herramientas gráficas y un entorno gráfico ligero LXDE (*Lighweight X11 Desktop Environment*) [LXDE].
   - **swap1GB.vdi**: Disco de 1 GB formateado como espacio de intercambio (SWAP)
3. Usuarios configurados e inicio en el sistema
   - Usuarios disponibles

login	password
root	purple
usuario	usuario
(con permisos para `sudo`)	

   - Acceso al entorno gráfico una vez logueado (necesario para poder copiar y pegar desde/hacia el anfitrión)

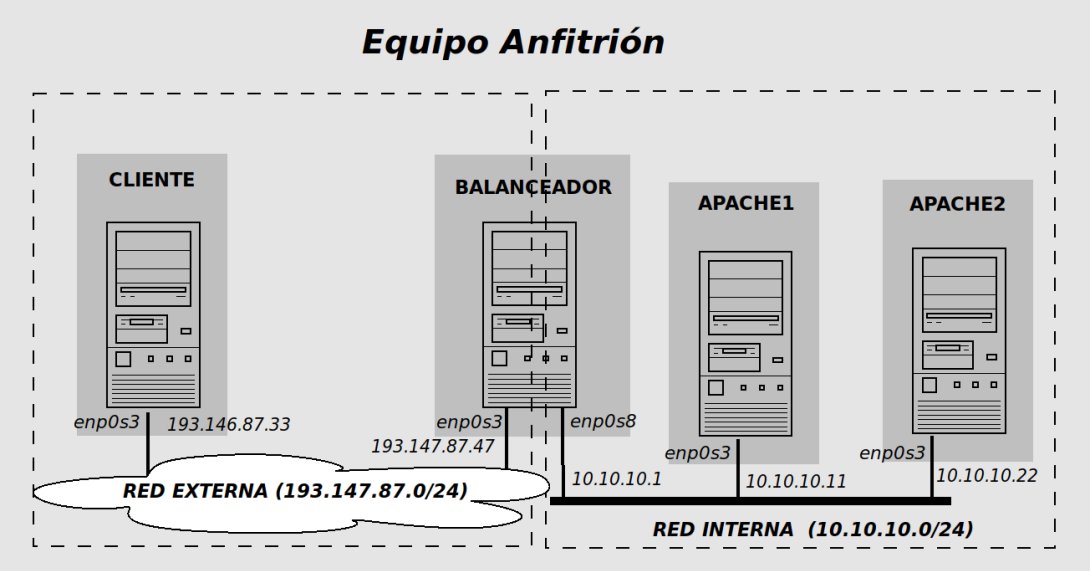
     `root@datos:~# startx`

   - Habilitar copiar y pegar desde/hacia el anfitrión en el menú `Dispositivos -> Portapapeles compartido -> bidir`
     de la ventana de la máquina virtual.

2.3. Máquinas virtuales y redes creadas

- Máquinas virtuales
  - **cliente** (193.147.87.33)
  - **balanceador** (193.147.87.47 en enp0s3 y 10.10.10.1 en enp0s8)
  - **apache1** (10.10.10.11) [para hacer más evidente el efecto del balanceo de carga, la capacidad de su CPU está limitada al 30 %]
  - **apache2** (10.10.10.22) [para hacer más evidente el efecto del balanceo de carga, la capacidad de su CPU está limitada al 30 %] nan

  Nota: para hacer más evidente el efecto del balanceo de carga, la capacidad de uso de la CPU en las dos máquinas del cluster de balanceo de carga (**apache1** y **apache2**) está reducida al 30 %.
- Red externa (193.147.87.0 ... 193.147.87.255): máquina **cliente** (enp0s3) + interfaz enp0s3 de **balanceador**
- Red balanceador (10.10.10.0 ... 10.10.10.255): máquina **apache1** (enp0s3) + máquina **apache2** (enp0s3) + interfaz enp0s8 de **balanceador**



3. HAProxy

HAProxy es un proyecto *open source* que ofrece un balanceador de carga implementado como un proxy inverso que reparte conexiones a nivel HTTP o TCP entre un conjunto de servidores reales.

- Versión *community*: `http://www.haproxy.org/`
- Versión *entreprise*: `https://www.haproxy.com/`
  - ALOHA (balanceador hardware/virtual en formato "enrackable"): `https://www.haproxy.com/products/haproxy-aloha/`
- Algunos servicios que usan/usaron HAproxy: `http://www.haproxy.org/they-use-it.html`

3.1. Funcionamiento y funcionalidades

HAProxy es un balanceador de carga que funciona como un *proxy inverso* (*reverse proxy*) repartiendo las conexiones recibidas de los clientes entre un conjunto de "servidores reales".

- Mantiene dos conexiones, `cliente <-> HAproxy` y `HAproxy <-> servidor real`, sobre las que retransmite los mensajes de petición y respuesta
- El *proxy inverso* puede retransmitir conexiones a nivel de capa de aplicación (capa 7, sólo para el protocolo HTTP) o de la capa de transporte (capa 4, para cualquier tráfico TCP)
- Al recibir una nueva conexión de un `cliente` el balanceador `HAproxy` decide a cúal de los `servidores reales` se retransmitirá el tráfico de esa nueva conexión
  1. aplica el algoritmo de reparto de carga configurado (*round robin*, *least connections*, *static*, etc)
  2. establece una conexión con ese `servidor real`
  3. retransmite todo el tráfico de ambas conexiones (`cliente <-> HAproxy`, `Haproxy <-> servidor real`) en ambos sentidos mientras la conexión con el `cliente` esté establecida

Funcionalidades

- Balanceo de carga en capa 7 (sólo HTTP) o capa 4 (TCP)
- Reescritura de URLs y cabeceras HTTP
- Seguimiento de cookies mediante reescritura de cabeceras HTTP
- Terminación TLS/SSL en el proxy
- Soporte HTTP/2, WebSockets.
- ...

Detalles en `http://docs.haproxy.org/2.2/intro.html3`

3.2. Configuración

Configuración por defecto en `/etc/haproxy/haproxy.cfg`

Fichero de configuración organizado en secciones

global Parámetros globales

- del proceso HAProxy: `user, group, chroot, setenv, ...`
- de configuración general TLS/SSL: `ca-base, ssl-default-xxx-xxx, ...`
- de "tunning": `maxconn, tune.buffers.xxx, ...`

defaults Valores por defecto de parámetros usados en el resto de secciones (pueden sobreescribirse posteriormente)

- ejemplos: `mode` (tipo de proxy `http` ó `tcp`), timeouts, etc

frontend Definición y parámetros de un `frontend` (pueden definirse varios)

- Describe un conjunto de puertos de escucha donde se aceptan las conexiones de los clientes que serán "balanceadas" por HAProxy
- Tiene un nombre y una serie de parámetros
  - `bind [<address>]:<port_range>`: define uno (o varios) puertos de escucha (opcionalmente también dirección IP) de un proxy inverso / balanceador
  - `stats enable|auth|admin|scope|uri|...`: configuración de la página de estadísticas del proxy
  - `default-backend`: nombre del backend al que por defecto se redigirán las conexiones recibidas (excepto las que se controlen mediante `use_backend`)
  - `use_backend <nombre> if|unless <condicion>`: especifica el `backend` que atenderá las conexiones indicadas en las ACL que indique la `<condicion>`

- `acl <nombre> <criterio> <valor>`: declara una lista de control y la condición sobre un criterio (`src, src_port, hdr(...), ...`) que debe cumplir una conexión

backend Definición y parámetros de un `backend` (pueden definirse varios)

- Describe un conjunto de `servidores reales` a donde serán reenviadas las conexiones de los clientes "balanceadas" por HAProxy
- `balance <algorithm>`: algoritmo usado para seleccionar el serividor real a usar en cada nueva conexión (`roundrobin, leastconn, first, source, uri,...`)
- `server <name> <address>[:port] [settings ...]`: definición de cada uno de los servidores reales del `backend`
- `cookie <name> rewrite|insert|prefix ...`: habilita la "persistencia de conexiones/sesiones" basada en el seguimiento de la cookie con el nombre indicado.

listen Sección opcional que combina la definición de un `frontend` y un `backend` para describir un proxy inverso completo.

Detalles en `http://docs.haproxy.org/2.2/configuration.html#2`

4. EJERCICIO ENTREGABLE: Balanceo de carga en servidores Apache con HAproxy

4.1. Pasos previos

1. Arrancar el entorno gráfico en **cliente** [**193.147.87.33**]

   ```
   cliente:~# startx
   ```

2. Ajustar la configuración de las dos máquinas del cluster de balanceo (**apache1** y **apache2**)

   a) Deshabilitar la opción *KeepAlive* en el fichero de configuración `/etc/apache2/apache2.conf` para realizar la evaluación del rendimiento sin la opción de reutilización de conexiones.

   ```
   apache1:~# nano /etc/apache2/apache2.conf
      ...
      KeepAlive Off
      ...


   apache2:~# nano /etc/apache2/apache2.conf
      ...
      KeepAlive Off
      ...
   ```

   Nota:

   - este ajuste no es estrictemente necesario (y sería desaconsejable en un entorno de producción real), pero facilita las pruebas manueles dado que permite detectar inmediatamente el "cambio" de destino resultado del balanceo de carga
   - manteniendo la opción por defecto, en las pruebas manuales desde el navegador sería necesario esperar 5 segundos (el *time out* de *keep alive*) antes de recargar la página y ver el efecto del reparto de carga

   b) Editar los archivos del sitio web para incluir una indicación del servidor real que está sirviendo una petición, de modo que sea posible "diferenciarlos" en las pruebas manuales con el navegador

   - en **apache1**

```
apache1:~# nano /var/www/html/index.html
   ...
   <h1> Servidor por APACHE_UNO </h1>
   ...

apache1:~# nano /var/www/html/sesion.php
   ...
   <h1> Servido por APACHE_UNO </h1>
   ...
```

- en **apache2**

```
apache2:~# nano /var/www/html/index.html
   ...
   <h1> Servido por APACHE_DOS </h1>
   ...

apache2:~# nano /var/www/html/sesion.php
   ...
   <h1> Servido por APACHE_DOS </h1>
   ...
```

Notas:

- este ajuste es simplemente una herramienta de depuración para verificar que el balanceador funciona correctamente
- en una "granja" de servidores real este comportamiento no tendría sentido, dado que, obviamente, todos los nodos servirían el mismo contenido/aplicaciones (normalmente almacenado de forma compartido en una SAN o soluciones similares)

c) [**Importante**] Corregir en ambas máquinas (**apache1**, **apache2**) el script PHP `sleep.php` usado en las pruebas

```
apache1~:# nano /var/www/html/sleep.php

apache2~:# nano /var/www/html/sleep.php
```

Nuevo contenido (reemplaza el existente)

```
<html>
<title> Página dinámica con retardo </title>
<body>
  <h1> Prueba dinámica con retardo </h1>
  <p> hora de inicio: <?php echo date('h:i:s'); ?> </p>

  <?php
  for ($i=0; $i < 100000; $i++) {       // <- poner 100000 iteraciones
     $str1 = sha1(rand()*rand());       // <- tres llamadas a sha1()
     $str2 = sha1(rand()*rand());
     $str3 = sha1(rand()*rand());
  }
  ?>

  <p> hora de fin: <?php echo date('h:i:s'); ?> </p>
</body>
</html>
```

Comprobación

```
apache1~:# php /var/www/sleep.php

apache2~:# php /var/www/sleep.php
```

4.2. Tarea 1: evaluar rendimiento de un servidor Apache sin balanceo

Se realizarán varias pruebas de carga sobre el servidor Apache ubicado en la máquina `apache1` para determinar el rendimiento que puede llegar a ofrecer una instancia única del servidor Apache.

- Se hará uso de la herramienta *Apache Benchmark* (comando `ab`) incluida en la distribución del servidor Apache
- Manual de `ab`: (página man).

Pasos a realizar

1. Parar el servicio `haproxy` en **balanceador [193.147.87.47]** (por defecto está arrancado, aunque no configurado)

   ```
   balanceador:~# systemctl stop haproxy
   ```

2. Parar el servidor `apache2` en **balanceador [193.147.87.47]**, en caso de estar iniciado

   ```
   balanceador:~# systemctl stop apache2
   ```

   Nota: En ocasiones no tiene efecto la parada del servicio `apache2`, puede comprobarse si el proceso sigue en ejecución con `pidof apache2` y "matar" los posibles procesos supervivientes si fuera necesario.

   ```
   balanceador:~# pidof apache2
   wwww xxxx yyyy zzzz
   balanceador:~# kill -9 wwww xxxx yyyy zzzz
   ```

3. Habilitar en **balanceador [193.147.87.47]** la redirección de puertos para que sea accesible el servidor Apache de la máquina **apache1 [10.10.10.11]** empleando el siguiente comando `iptables`

   ```
   balanceador:~# echo 1 > /proc/sys/net/ipv4/ip_forward

   balanceador:~# iptables -P FORWARD ACCEPT

   balanceador:~# iptables -t nat -A PREROUTING \
                           --in-interface enp0s3 --protocol tcp --dport 80 \
                           -j DNAT --to-destination 10.10.10.11
   ```

   Notas:

   - La primera regla `iptables` establece una política de aceptar por defecto en la cadena `FORWARD`
     Normalmente esta regla no sería necesaria (por defecto el kernel arranca con una política "aceptar por defecto" en todas las cadenas), pero en este caso la instalación de Docker en la MV estableció un `DROP` por defecto en la cadena `FORWARD` que impediría el tráfico del ejercicio.
   - La segunda regla `iptables` sí es necesaria en cualquier caso y es la que establece la redirección del puerto 80 de la máquina `balanceador` al mismo puerto de la máquina `apache1` para el tráfico procedente de la red externa (interfaz de entrada `enp0s3`).

4. Arrancar (o reiniciar) en **apache1 [10.10.10.11]** el servidor web Apache

   ```
   apache1:~# systemctl restart apache2      #(ó     systemctl start apache2)
   ```

   Comprobación:

- En la máquina **cliente** [**193.147.87.33**] abrir la URL `http://193.147.87.47` en el navegador web *Falkon* o en el navegador en modo texto `lynx`.

  ```
  cliente:~#  lynx 193.147.87.47
  ```

  Usar la tecla `Q` o `[Control]+C` para salir de `lynx`
- La respuesta indicará que proviene del servidor `APACHE_uno`.

5. Lanzar las pruebas de carga iniciales sobre **balanceador** [**193.147.87.47**] usando el herramienta *Apache Benchmark*

   - **Prueba 1:** Contenido estático

     Pruebas a realizar:

     ```
     cliente:~# ab -n 2000 -c 10 http://193.147.87.47/index.html
     cliente:~# ab -n 2000 -c 50 http://193.147.87.47/index.html
     ```

     Envía 2000 peticiones HTTP (opción `-n`) sobre la URI "estática" `index.html`, realizando, respectivamente, 10 y 50 conexiones concurrentes (opción `-c`). *(< 1 minuto)*
   - **Prueba 2:** Scripts PHP
     - Se usará un script PHP (`sleep.php`) que introduce un retardo mediante un bucle "activo" de 100000 iteraciones que busca forzar el uso de CPU con cálculos de hashes SHA1 y concatenaciones de cadenas.
     - Ver código en el fichero `/var/www/html/sleep.php` de las máquinas **apache1** [**10.10.10.11**] o **apache2** [**10.10.10.22**]

     Pruebas a realizar:

     ```
     cliente:~# ab -n 250 -c 10 http://193.147.87.47/sleep.php
     cliente:~# ab -n 250 -c 30 http://193.147.87.47/sleep.php
     ```

     Envía 250 peticiones HTTP sobre la URI "dinámica", realizando, respectivamente, 10 y 30 conexiones concurrentes. *(aprox 3-4 minutos)*

   En cada ejecución del comando `ab` se muestran las estadísticas obtenidas. Para el tipo de prueba informal que se realiza en este ejemplo, basta prestar atención a los parámetros `Requests per second` (num. peticiones por segundo) ó `Time per request` (tiempo en milisegundos para procesar cada petición).

4.3. Tarea 2: configurar y evaluar balanceo de carga con dos servidores Apache

1. Deshabilitar la redirección del puerto 80 de la máquina **balanceador** [**193.147.87.47**] con el siguiente comando `iptables` (HAproxy se encargará de retransmitir ese tráfico sin necesidad de redireccionar los puertos)

   ```
   balanceador:~# iptables -t nat -F
   balanceador:~# iptables -t nat -Z
   ```

2. Parar el servidor `apache2` en **balanceador** [**193.147.87.47**], en caso de estar iniciado

   ```
   balanceador:~# systemctl stop apache2
   ```

3. Arrancar/reiniciar los servidores Apache de **apache1** [**10.10.10.11**] y **apache2** [**10.10.10.22**], en caso de no estar iniciados

   ```
   apache1:~# systemctl restart apache2      #(ó     systemctl start apache2)

   apache2:~# systemctl restart apache2      #(ó     systemctl start apache2)
   ```

4. Instalar *HAproxy* en **balanceador** [**193.147.87.47**] [**ya está hecho**]

   ```
   balanceador:~# apt-get update
   balanceador:~# apt-get install haproxy
   ```

5. Configurar *HAproxy* en **balanceador [193.147.87.47]** (de momento sin soporte de sesiones persistentes)

```
balanceador:~# cd /etc/haproxy
balanceador:/etc/haproxy/# mv haproxy.cfg haproxy.cfg.original
balanceador:/etc/haproxy/# nano haproxy.cfg
```

Contenido a incluir:

```
global
        daemon
        maxconn 256
        user    haproxy
        group   haproxy

defaults
        mode    http
        log     global
        timeout connect 5000ms
        timeout client  50000ms
        timeout server  50000ms

frontend web_cda
        bind 193.147.87.47:80
        mode http
        stats enable
        stats auth  cda:cda
        default_backend servidores_cda

backend servidores_cda
        balance roundrobin
        server uno 10.10.10.11:80 maxconn 128
        server dos 10.10.10.22:80 maxconn 128
```

Define (en la sección `frontend`) un "proxy inverso" de nombre `web_cda` que:

a) trabajará en modo `http` (la otra alternativa es el modo `tcp`, pero no analiza las peticiones/respuestas HTTP, sólo retransmite paquetes TCP a nivel de capa de transporte)

b) atendiendo peticiones en el puerto `80` de la dirección `193.147.87.47`

c) las peticiones se repartirán en el *cluster* `servidores_cda`

d) adicionalmente, habilita la consola Web de estadísticas, accesible con las credenciales `cda:cda`

Define (en la sección `backend`) la lista de servidores reales `servidores_cda`

a) con balanceo `round-robin`

b) formada por dos servidores reales (de nombres `uno` y `dos`) en el puerto `80` de las direcciones `10.10.10.11` y `10.10.10.22`

Más detalles en Opciones de configuración HAPproxy 2.2

6. Iniciar *HAproxy* en **balanceador [193.147.87.47]**

Antes de hacerlo es necesario habilitar en `/etc/default/haproxy` el arranque de HAproxy desde los scripts de inicio de Debian, estableciendo la variable `ENABLED=1`

```
balanceador:/etc/haproxy/# nano  /etc/default/haproxy

...
ENABLED=1
```

Inicio empleando el script de arranque de HAproxy (preferente)

```
balanceador:/etc/haproxy/# systemctl stop  haproxy    # ya estaba arrancado
balanceador:/etc/haproxy/# systemctl start haproxy
```

- **Nota:** Es posible iniciar manualmente HAproxy desde línea de comandos (las opciones `-d` y `-V` habilitan los mensajes de DEBUG) [**no necesario** en este ejemplo]

  ```
  balanceador:/etc/haproxy/# haproxy -d -V -f /etc/haproxy/haproxy.cfg
  ```

7. **Comprobar funcionamiento del balancenador:** desde la máquina **cliente** [**193.147.87.33**] abrir la URL `http://193.147.87.47` en el navegador web en modod texto `lynx` y recargar varias veces con `[Control]+R` para comprobar como cambia el servidor real que responde las peticiones.

   ```
   cliente:~#  lynx 193.147.87.47
   ```

   Nota: El navegador gráfico *Falkon* hace caché de las páginas cargadas y no permite comprobar el balanceo de carga directamente recargando la página.

   Nota: Si no se ha deshabilitado la opción *KeepAlive* de Apache, es necesario esperar 5 segundos entre las recargas para que se agote el tiempo de espera para cerrar completamente la conexión HTTP y que pase a ser atendida por otro servidor.

8. Desde la máquina **cliente** [**193.147.87.33**] repetir las pruebas de carga con `ab`

   Pruebas a realizar:

   ```
   cliente:~# ab -n 2000 -c 10 http://193.147.87.47/index.html
   cliente:~# ab -n 2000 -c 50 http://193.147.87.47/index.html

   cliente:~# ab -n 250 -c 10 http://193.147.87.47/sleep.php
   cliente:~# ab -n 250 -c 30 http://193.147.87.47/sleep.php
   ```

   Los resultados deberían de ser mejores que con la prueba anterior con un servidor Apache único (al menos en el caso del script `sleep.php`)

9. Desde la máquina **cliente** [**193.147.87.33**] abrir en el navegador web *Falkon* la URL `http://193.147.87.47/haproxy?sta` para inspeccionar las estadísticas del balanceador HAProxy (pedirá un usuario y un password, ambos `cda`)

10. Desde uno de los servidores (**apache1** ó **apache2**), verificar los logs del servidor Apache

    ```
    apache1:~# tail /var/log/apache2/error.log
    apache1:~# tail /var/log/apache2/access.log
    ```

    ```
    apache2:~# tail /var/log/apache2/error.log
    apache2:~# tail /var/log/apache2/access.log
    ```

11. **Cuestión 1.** En todos los casos debería figurar como única dirección IP cliente la IP interna de la máquina **balanceador** [**10.10.10.1**]. **¿Por qué?**

4.4. Tarea 3: configurar la persistencia de conexiones Web (*sticky sessions*)

Comprobar el contenido del fichero PHP `/var/www/html/sesion.php` disponible en las máquinas **apache1** [**10.10.10.11**] y **apache2** [**10.10.10.22**].

Se puede verificar el funcionamiento "defectuoso" del balanceo de carga por defecto (*round robin* sin seguimiento de `cookies`), accediendo desde la máquina **cliente** a la URL `http://193.147.87.47/sesion.php` con el navegador en modo texto `lynx` y recargando la página varias veces con `[Control]+R`.

```
cliente:~# lynx -accept-all-cookies  http://193.147.87.47/sesion.php
```

- Como consecuencia del reparto de conexiones que realiza el balanceador de carga entre ambos servidores (con *round robin*), se estarán manteniendo realmente dos sesiones, una en cada servidor del cluster, con un contador de accesos diferente en cada una de ellas.

- El comportamiento deseado en este caso sería:
  - una vez establecida la sesión por parte del servidor del cluster designado por el balanceador de carga (enviando una respuesta HTTP en la que el parámetro `SET_COOKIE` donde se establece el nombre y el valor de la Cookie)
  - las sucesivas peticiones HTTP enviadas desde el cliente y marcadas con esa COOKIE (parámetro `COOKIE` de la cabecera de la petición HTTP) serán enviadas únicamente el servidor que estableció esa Cookie, "desactivando" en ese caso el reparto de carga que realizaría el balanceador

Para solucionarlo es necesario configurar HAProxy para que haga el **seguimiento de las cookies de sesión**. De tal modo que no se aplique el balanceo de carga, asignado el mismo servidor real que estableció una cookie determinada en un parámetro `SET_COOKIE` de una `HTTP response`.

1. Detener HAproxy en la máquina **balanceador** [**193.147.87.47**]

   ```
   balanceador:/etc/haproxy/# systemctl stop haproxy
   ```

2. Añadir las opciones de persistencia de conexiones HTTP (*sticky cookies*) al fichero de configuración

   ```
   balanceador:~# nano /etc/haproxy/haproxy.cfg
   ```

   Contenido a incluir: (añadidos marcados con `<- aqui`)

   ```
   global
           daemon
           maxconn 256
           user    haproxy
           group   haproxy

   defaults
           mode    http
           log     global
           timeout connect 10000ms
           timeout client  50000ms
           timeout server  50000ms

   frontend web_cda
           bind 193.147.87.47:80
           mode http
           stats enable
           stats auth  cda:cda
           default_backend servidores_cda

   backend servidores_cda
           balance roundrobin
           cookie PHPSESSID prefix                              # <- aqui
           server uno 10.10.10.11:80 cookie ZIPI maxconn 128    # <- aqui
           server dos 10.10.10.22:80 cookie ZAPE maxconn 128    # <- aqui
   ```

   El parámetro `cookie` especifica el nombre de la *cookie* que se usa como identificador único de la sesión del cliente (en el caso de aplicaciones web PHP se suele utilizar por defecto el nombre `PHPSESSID`)

   - Para cada "servidor real" se especifica una etiqueta identificativa exclusiva mediante el parámetro `cookie` (`ZIPI` y `ZAPE` en el ejemplo)
   - Con esa información HAproxy reescribirá las cabeceras HTTP de peticiones y respuestas para seguir la pista de las sesiones establecidas en cada "servidor real" usando el nombre de cookie especificado (`PHPSESSID`)

- Se establece una estrategia de seguimiento de Cookies de tipo `prefix`
  - HAproxy maneja 2 valores para la Cookie sobre la que se hace el seguimiento, uno en la conexión "interna" y otro en la conexión "externa"
  - En la respuesta HTTP donde el párámetro `SET_COOKIE` establece la Cookie de sesión, el cliente recibirá como valor de la Cookie el valor de la Cookie original precedido de la etiqueta del servidor que la haya establecido
    - valor Cookie en conexión `cliente ->balanceador HAproxy : [etiqueta servidor]~[cookie original]`
    - valor Cookie en conexión `balanceador HAproxy ->servidor : [cookie original]`
  - Las sucesivas peticiones HTTP recibidas del cliente, vendrán "'marcadas" con el valor de Cookie "externo", HAproxy identifica esas peticiones como pertenecientes a una sesión abierta y usa la etiqueta de servidor para identificar al servidor dónde está definida esa sesión.
  - HAproxy reescribe esa petición HTTP, reemplazando el valor "externo" de la Cookie por el valor "interno" original, y la retransmite al servidor real que estableció dicha Cookie.

3. Iniciar HAproxy en la máquina **balanceador** [**193.147.87.47**]

   ```
   balanceador:/etc/haproxy/# systemctl start haproxy
   ```

4. En la máquina **cliente** [**193.147.87.33**], arrancar el *sniffer* de red `whireshark` y ponerlo en escucha sobre el interfaz *enp0s3* (puede fijarse como **filtro** la cadena `http` para que solo muestre las peticiones y respuestas HTTP)

   ```
   cliente:~# wireshark &
   ```

5. En la máquina **cliente** [**193.147.87.33**]

   - Desde el navegador web en modo texto acceder varias veces a la URL `http://193.147.87.47/sesion.php` (comprobar que el incremento del contador de visitas [variable de sesión] es el correcto)

     ```
     cliente:~# lynx -accept-all-cookies  http://193.147.87.47/sesion.php
     ```

   (recarga con `[control]+R`)

6. Detener la captura de tráfico en `wireshark` y comprobar las peticiones/respuestas HTTP capturadas

   Verificar la estructura y valores de las *cookies* `PHPSESSID` intercambiadas

   - En la primera respuesta HTTP (inicio de sesión), se establece su valor con un parámetro HTTP `SetCookie` en la cabecera de la respuesta
   - Las sucesivas peticiones del cliente incluyen el valor de esa cookie (parámetro HTTP `Cookie` en la cabecera de las peticiones)

5. Documentación a entregar

- Descripción **breve** del ejercicio realizado.
- Describir cómo sería el flujo de mensajes HTTP (tanto peticiones como respuestas) entre las 3 máquinas implicadas en el caso de las peticiones sobre `http://193.147.87.47/index.html` realizadas en la *Tarea 2* una vez que está configurado y en uso el balanceador HAProxy.
- Explicar el porqué de la **Cuestión 1**, planteada al final de la *Tarea 2*
- Detallar los resultados obtenidos en las pruebas de rendimiento realizadas con `ab` en la *Tarea 1* y en la *Tarea 2*. Comentar brevemente los resultados obtenidos y el porqué de las diferencias (o de la ausencia de diferencias)
- Detallar las capturas de tráfico realizadas con Wireshark en la *Tarea 3* donde se muestre el funcionamiento del *seguimiento de conexiones* (*sticky cookies*) de HAproxy

Entrega: MOOVI (práctica individual)

Fecha límite: hasta el domingo 27/11/2022