UNIVERSIDAD DE GRANADA

Servidores Web de Altas Prestaciones

Práctica 2: Balanceo de carga

Miguel Torres Alonso

Abril 2024

Índice

1. Tareas Básicas - B1. Preparación del entorno de trabajo

1.1. Crear directorios específicos para los archivos de configuración de los balanceadores

Considero que no es necesario el uso de la IA generativa para este apartado. Por tanto, creamos los dos directorios **'P2-mtorres26-nginx'** y **'P2-mtorres26-haproxy'** tal cual nos lo dicen en el guión, ambas carpetas con el directorio *'web_mtorres26'* ya creado en la práctica 1.

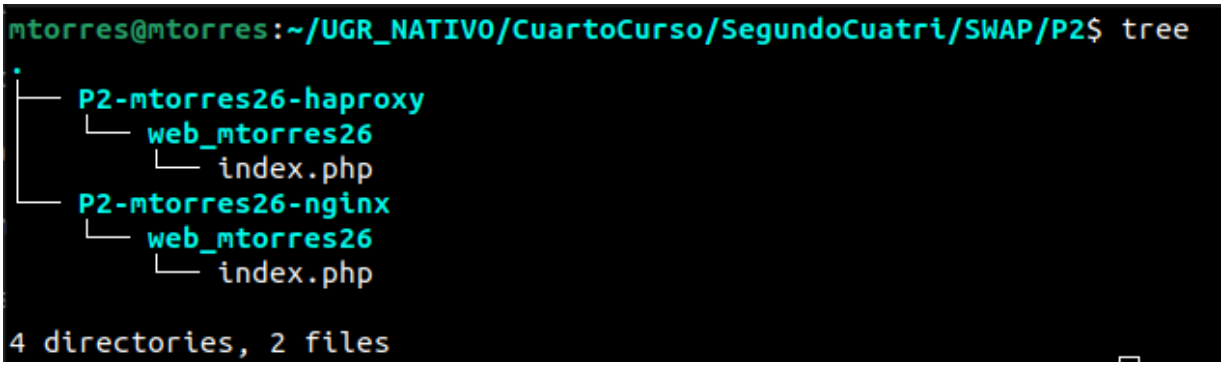

Figura 1: Captura mostrando la estructura en árbol de las carpetas de trabajo.

2. Tareas Básicas - B2. Configuración de Nginx como balanceador de carga

2.1. Redactar el Dockerfile para crear una imagen personalizada de Nginx a partir de la imagen oficial.

2.1.1. Consulta realizada IA:

'Genera el código de un Dockerfile que use la imagen oficial de Nginx y use una configuración personalizada.'

2.1.2. Respuesta proporcionada por la IA:

```
# Usa la imagen oficial de Nginx como base
FROM nginx

# Copia el archivo de configuracion personalizado al contenedor
COPY nginx.conf /etc/nginx/nginx.conf

# Opcional: Copia cualquier otro archivo de configuracion personalizado
# COPY otro_archivo_de_configuracion /ruta/en/el/contenedor

# Opcional: Copia los archivos estaticos de tu aplicacion web
# COPY carpeta_de_archivos_estaticos /ruta/en/el/contenedor

# Opcional: Define cualquier variable de entorno necesaria
# ENV VARIABLE_NAME valor
```

```
# Opcional: Expone los puertos necesarios
# EXPOSE puerto

# Puedes cambiar el comando predeterminado de Nginx
# CMD ["nginx", "-g", "daemon off;"]
```

Listing 1: Dockerfile proporcionado por la IA.

2.1.3. Análisis detallado:

La IA nos proporciona una buena respuesta, aunque vamos a eliminar ciertos parámetros del Dockerfile que considero que no son necesarios, como la copia de carpetas, la definición de variables de entorno, la apertura del puerto 80... La copia del nginx.conf la haremos en el *docker-compose.yml*. El Dockerfile quedaría así:

```
# Usa la imagen oficial de Nginx como base
FROM nginx:latest

# Puedes cambiar el comando predeterminado de Nginx
CMD ["nginx", "-g", "daemon off;"]
```

Listing 2: Código de nuestro 'DockerfileNginx'

2.2. Escribir el archivo de configuración nginx.conf con la estrategia de balanceo round-robin y configuraciones de registro de accesos y errores.

2.2.1. Consulta realizada IA:

'Genera el código necesario para un archivo de configuración nginx.conf con servidores cuyas IP's van desde 192.168.10.2 hasta 192.168.10.9, con la estrategia de balanceo round-robin y configuraciones de registro de accesos y errores.'

2.2.2. Respuesta proporcionada por la IA:

```
# Configuracion de eventos para Nginx
events {
    worker_connections  1024;  # Numero maximo de conexiones por trabajador
}

# Configuracion del servidor HTTP
http {
    # Configuracion de los registros de accesos
    access_log /var/log/nginx/access.log;
    error_log /var/log/nginx/error.log;

    # Configuracion del balanceador de carga
    upstream backend {
        # Se especifican las direcciones IP de los servidores backend
        server 192.168.10.2;
        server 192.168.10.3;
        server 192.168.10.4;
        server 192.168.10.5;
        server 192.168.10.6;
```

```
        server 192.168.10.7;
        server 192.168.10.8;
        server 192.168.10.9;
    }

    # Configuracion del servidor virtual
    server {
        listen 80;  # Puerto en el que Nginx escucha las solicitudes HTTP

        # Configuracion de la ruta de acceso "/"
        location / {
            proxy_pass http://backend;  # Balanceo round-robin entre los servidores backer
            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_set_header X-Forwarded-Proto $scheme;
        }
    }
}
```

Listing 3: Código proporcionado por la IA.

2.2.3. Análisis detallado:

La IA nos imprime un archivo de configuración bastante similar al proporcionado en el guión, pero nos vamos a decantar por este último adaptado a nuestro nombre de usuario de la UGR y añadiendo el apartado *events*, obligatorio para un correcto funcionamiento.

Entonces, nuestro archivo de configuración quedaría así:

```
events {
    worker_connections 1024; # Numero maximo de conexiones por trabajador
}
http {
    upstream backend_mtorres26 {
        # nuestros 8 contenedores
        server 192.168.10.2;
        server 192.168.10.3;
        server 192.168.10.4;
        server 192.168.10.5;
        server 192.168.10.6;
        server 192.168.10.7;
        server 192.168.10.8;
        server 192.168.10.9;
    }
    server{
        # puerto 80 para http
        listen 80;
        server_name nginx_mtorres26;
        access_log /var/log/nginx/nginx_mtorres26.access.log;
        error_log /var/log/nginx/nginx_mtorres26.error.log;
        location / {
            proxy_pass http://backend_mtorres26;
            proxy_set_header Cookie $http_cookie;
```

```
            proxy_hide_header Set-Cookie;
        }
        location /estadisticas_mtorres26 {
            stub_status on;
        }
    }
}
```

Listing 4: Archivo de configuración personalizada.

3. Tareas Básicas - B3. Implementación del escenario de Ningx con Docker Compose

3.1. Reutilizar o adaptar el DockerfileApache de la Práctica 1 para los contenedores de Apache con el nuevo tag p2.

Para este apartado no es necesario la consulta a la IA, pues vamos a utilizar el *DockerfileApache_mtorres26* de la P1. Al tenerlo escrito con los requisitos de las tareas avanzadas de la P1, vamos a realizar un par de modificaciones para tener una versión "básica".

Se quedaría con el siguiente aspecto:

```
# Usa una imagen base de Debian con una version ligera
FROM debian:buster-slim

# Actualiza el repositorio e instala Apache, PHP y herramientas de red
RUN apt-get update && \
    apt-get install -y apache2 php libapache2-mod-php \
    net-tools iputils-ping && \
    apt-get clean && \
    rm -rf /var/lib/apt/lists/*

# Expone el puerto 80 para Apache
EXPOSE 80

# Inicia Apache en el inicio del contenedor
CMD ["apachectl", "-D", "FOREGROUND"]
```

Listing 5: El Dockerfile de Apache que vamos a usar.

3.2. Desarrollar el docker-compose.yml para configurar el servicio para cada contenedor, el volumen para el directorio y el balanceador de carga Nginx con las características correspondientes.

3.2.1. Consulta realizada IA:

'Genera el código para un docker-compose en el que se creen 8 contenedores denominados 'web' con direcciones IP desde la 192.168.10.2 hasta la 192.168.10.9, también un servicio llamado 'balanceador-nginx' que use una imagen de Nginx a partir del Dockerfile 'DockerfileNginx', monte con un volumen el archivo nginx.conf en la ruta '/etc/nginx' del contenedor y use la dirección 192.168.10.50'

3.2.2. Respuesta proporcionada por la IA:

```
services:
  balanceador-nginx:
    build:
      context: .
      dockerfile: DockerfileNginx
    volumes:
      - ./nginx.conf:/etc/nginx/nginx.conf
    ports:
      - "192.168.10.50:80:80"
    networks:
      default:
        ipv4_address: 192.168.10.50
```

Listing 6: Servicio de balanceador en el docker-compose generado por IA.

3.2.3. Análisis detallado:

El *docker-compose.yml* generado por la IA es muy similar al que ya tenemos hecho de la P1, únicamente nos faltaba la configuración del servicio *balanceador-nginx*, por tanto vamos a usar lo que nos dice la IA incluyendo el manejo de puertos lo cual es necesario y las dependencias, pues necesitamos que el *balanceador* se ejecute siempre después de los *contenedores web*.

El *docker-compose.yml* queda de la siguiente manera:

```
services:
  web1:
    image: mtorres26-apache-image:p2
    container_name: web1
    build:
      context: .
      dockerfile: DockerfileApache_mtorres26
    volumes:
      - ./web_mtorres26:/var/www/html
    networks:
      red_web:
        ipv4_address: 192.168.10.2
      red_servicios:
        ipv4_address: 192.168.20.2

  web2:
    image: mtorres26-apache-image:p2
    container_name: web2
    volumes:
      - ./web_mtorres26:/var/www/html
    networks:
      red_web:
        ipv4_address: 192.168.10.3
      red_servicios:
        ipv4_address: 192.168.20.3

  web3:
    image: mtorres26-apache-image:p2
    container_name: web3
    volumes:
```

```
    - ./web_mtorres26:/var/www/html
  networks:
    red_web:
      ipv4_address: 192.168.10.4
    red_servicios:
      ipv4_address: 192.168.20.4

web4:
  image: mtorres26-apache-image:p2
  container_name: web4
  volumes:
    - ./web_mtorres26:/var/www/html
  networks:
    red_web:
      ipv4_address: 192.168.10.5
    red_servicios:
      ipv4_address: 192.168.20.5

web5:
  image: mtorres26-apache-image:p2
  container_name: web5
  volumes:
    - ./web_mtorres26:/var/www/html
  networks:
    red_web:
      ipv4_address: 192.168.10.6
    red_servicios:
      ipv4_address: 192.168.20.6

web6:
  image: mtorres26-apache-image:p2
  container_name: web6
  volumes:
    - ./web_mtorres26:/var/www/html
  networks:
    red_web:
      ipv4_address: 192.168.10.7
    red_servicios:
      ipv4_address: 192.168.20.7

web7:
  image: mtorres26-apache-image:p2
  container_name: web7
  volumes:
    - ./web_mtorres26:/var/www/html
  networks:
    red_web:
      ipv4_address: 192.168.10.8
    red_servicios:
      ipv4_address: 192.168.20.8

web8:
```

```
    image: mtorres26-apache-image:p2
    container_name: web8
    volumes:
      - ./web_mtorres26:/var/www/html
    networks:
      red_web:
        ipv4_address: 192.168.10.9
      red_servicios:
        ipv4_address: 192.168.20.9

  balanceador-nginx:
    image: mtorres26-nginx-image:p2
    container_name: balanceador-nginx
    build:
      context: .
      dockerfile: DockerfileNginx
    volumes:
      - ./nginx.conf:/etc/nginx/nginx.conf
    networks:
      red_web:
        ipv4_address: 192.168.10.50
    ports:
      - "80:80"
    depends_on:
      - web1
      - web2
      - web3
      - web4
      - web5
      - web6
      - web7
      - web8

networks:
  red_web:
    driver: bridge
    ipam:
      config:
        - subnet: 192.168.10.0/24

  red_servicios:
    driver: bridge
    ipam:
      config:
        - subnet: 192.168.20.0/24
```

Listing 7: El docker-compose que vamos a usar

4. Tareas Básicas - B4. Verificación y Pruebas del escenario de Nginx

Para esta tarea considero que no es necesario consultar con la IA generativa, pues solamente debemos seguir un par de comandos de lanzamiento de los servidores y verificación de los mismos.

4.1. Desplegar los servicios con docker-compose up -d.

Desplegamos los servicios:

```
mtorres@mtorres:~/UGR_NATIVO/CuartoCurso/SegundoCuatri/SWAP/P2/P2-mtorres26-nginx$ docker compose up -d
[+] Running 9/11
 ⠸ Network p2-mtorres26-nginx_red_servicios  Created
 ⠸ Network p2-mtorres26-nginx_red_web        Created
 ✔ Container web7                            Started
 ✔ Container web4                            Started
 ✔ Container web8                            Started
 ✔ Container web6                            Started
 ✔ Container web1                            Started
 ✔ Container web3                            Started
 ✔ Container web5                            Started
 ✔ Container web2                            Started
 ✔ Container balanceador-nginx               Started
```

Figura 2: Captura mostrando el despliegue de los servicios.

4.2. Verificar que los servicios están activos y que Nginx distribuye correctamente las solicitudes.

Ejecutamos un par de comandos para comprobar lo que se pide, como podemos observar en las figuras 3 y 4.

```
mtorres@mtorres:~/UGR_NATIVO/CuartoCurso/SegundoCuatri/SWAP/P2/P2-mtorres26-nginx$ docker ps
CONTAINER ID   IMAGE                          COMMAND                  CREATED         STATUS         PORTS                                   NAMES
54bf424db4d6   mtorres26-nginx-image:p2       "/docker-entrypoint.…"   5 seconds ago   Up 1 second    0.0.0.0:80->80/tcp, :::80->80/tcp       balanceador-ngi
nx
fcc05b8920b0   mtorres26-apache-image:p2      "apachectl -D FOREGR…"   6 seconds ago   Up 2 seconds   80/tcp                                  web4
c3a7050ac8ac   mtorres26-apache-image:p2      "apachectl -D FOREGR…"   6 seconds ago   Up 2 seconds   80/tcp                                  web3
7824381b751f   mtorres26-apache-image:p2      "apachectl -D FOREGR…"   6 seconds ago   Up 2 seconds   80/tcp                                  web6
0d3bc072e765   mtorres26-apache-image:p2      "apachectl -D FOREGR…"   6 seconds ago   Up 2 seconds   80/tcp                                  web1
b6dcdae8ae99   mtorres26-apache-image:p2      "apachectl -D FOREGR…"   6 seconds ago   Up 2 seconds   80/tcp                                  web5
3bad65461ea7   mtorres26-apache-image:p2      "apachectl -D FOREGR…"   6 seconds ago   Up 3 seconds   80/tcp                                  web7
d83321a77cd3   mtorres26-apache-image:p2      "apachectl -D FOREGR…"   6 seconds ago   Up 2 seconds   80/tcp                                  web2
f10f069c7fba   mtorres26-apache-image:p2      "apachectl -D FOREGR…"   6 seconds ago   Up 2 seconds   80/tcp                                  web8
mtorres@mtorres:~/UGR_NATIVO/CuartoCurso/SegundoCuatri/SWAP/P2/P2-mtorres26-nginx$
```

Figura 3: Captura mostrando los servicios en funcionamiento con *docker ps*.

4.3. Acceder a la página de estadísticas de Nginx para observar el rendimiento del balanceador.

La figura 5 muestra lo que pasa cuando accedemos a la URL de estadisticas de nuestro balanceador, *estadisticas_mtorres26* y nos muestra: conexiones activas, peticiones aceptadas, manejadas... etc.

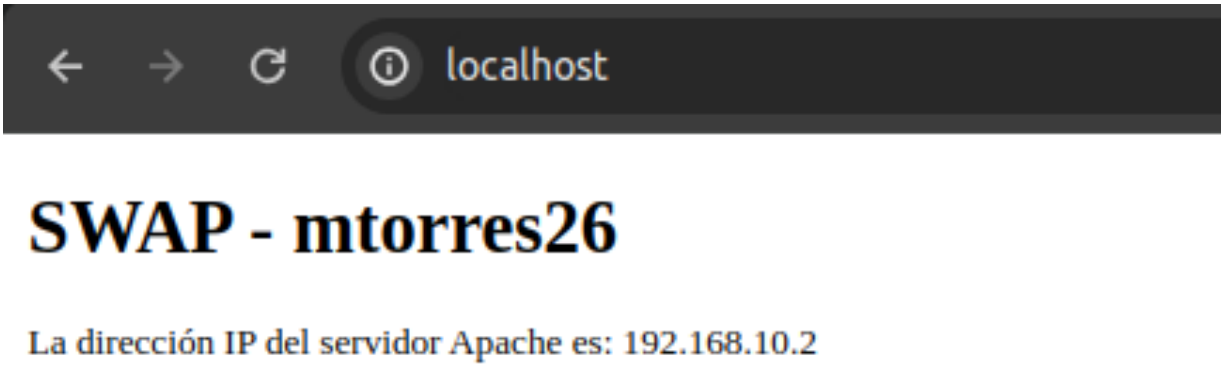

Figura 4: Captura mostrando el acceso a un servidor.

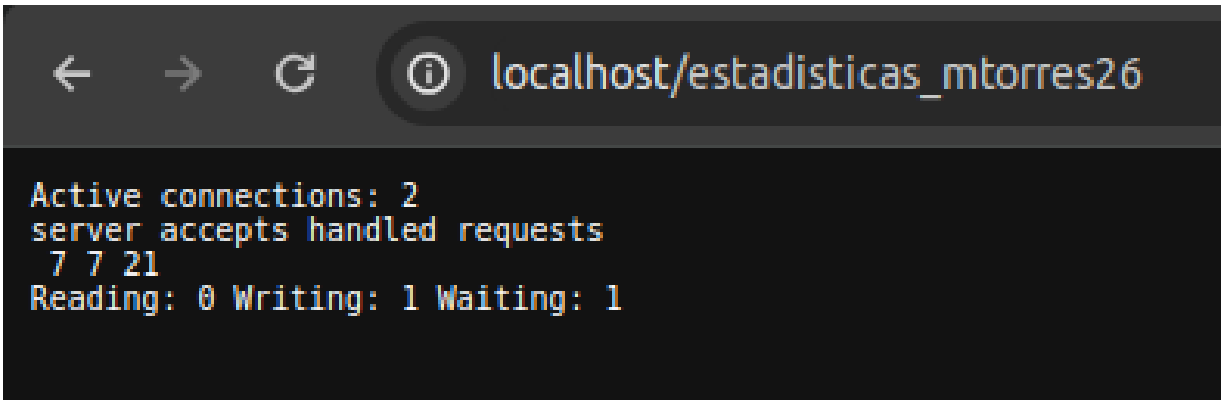

Figura 5: Captura mostrando las estadísticas de nuestro balanceador.

5. Tareas Básicas - B5. Configuración de HAProxy como Balanceador de Carga

5.1. Redactar el Dockerfile para crear una imagen de HAProxy a partir de la imagen oficial.

5.1.1. Consulta realizada IA:

'Genera el código necesario para crear un Dockerfile que use HAProxy como imagen base y copie un archivo de configuración personalizado dentro de la imagen'

5.1.2. Respuesta proporcionada por la IA:

```
# Usa la imagen oficial de HAProxy como base
FROM haproxy:latest

# Copia el archivo de configuracion personalizado al contenedor
COPY haproxy.cfg /usr/local/etc/haproxy/haproxy.cfg

# Opcional: Copia cualquier otro archivo de configuracion personalizado
```

```
# COPY otro_archivo_de_configuracion /ruta/en/el/contenedor
```

Listing 8: Dockerfile de HAProxy sugerido por IA.

5.1.3. Análisis detallado:

La IA nos ha generado correctamente lo que buscábamos, por tanto nuestro *DockerfileHAProxy* quedará tal cual nos lo propone[1].

5.2. Crear el archivo de configuración haproxy.cfg incluyendo las estrategias de balanceo de carga y la configuración de las estadísticas.

5.2.1. Consulta realizada IA:

'Genera el código de un archivo de configuración de HAProxy para utilizar algoritmo de balanceo de carga round-robin y activar las estadísticas de HAProxy para el monitoreo en el puerto 9000'

5.2.2. Respuesta proporcionada por la IA:

```
# Configuracion global
global
    log /dev/log    local0
    log /dev/log    local1 notice
    stats socket /var/run/haproxy.sock mode 660 level admin
    stats timeout 30s
    user haproxy
    group haproxy
    daemon

# Configuracion de defaults
defaults
    log     global
    mode    http
    option  httplog
    option  dontlognull
    timeout connect 5000
    timeout client  50000
    timeout server  50000

# Configuracion de los backends y frontends
frontend http_front
    bind *:80
    stats uri /haproxy?stats
    default_backend http_back

backend http_back
    balance roundrobin
    server server1 192.168.1.101:80 check
    server server2 192.168.1.102:80 check
    # Agrega mas servidores segun sea necesario
```

[1]Exceptuando las dos últimas líneas de código que son simples comentarios opcionales

```
# server serverN IP:PUERTO check
```

Listing 9: Archivo de configuración de HAProxy sugerido por IA

5.2.3. Análisis detallado:

La IA proporciona un archivo de configuración decente, pero vamos a usar principalmente el proporcionado en el guión añadiendo el 'check' para comprobar periódicamente la salud de los servidores en el apartado de backend.

También será necesario:

- Añadir 'mode http' a la parte de backend.
- Añadir un 'stats refresh 10s' en la parte de 'listen stats'.

6. Tareas Básicas - B6. Implementación del escenario de HAProxy con Docker Compose

6.1. Reutilizar o adaptar el DockerfileApache de la Práctica 1 para los contenedores de Apache con el nuevo tag p2.

Simplemente vamos a copiar el *DockerfileApache_mtorres26* usado en la carpeta de Nginx en la carpeta de HAProxy.

6.2. Detallar en docker-compose.yml la configuración de cada servicio Apache, el volumen para el directorio, y el servicio para el balanceador de carga HAProxy.

6.2.1. Consulta realizada IA:

'Genera el código para un docker-compose en el que se creen 8 contenedores denominados 'web' con direcciones IP desde la 192.168.10.2 hasta la 192.168.10.9, también un servicio llamado 'balanceador-haproxy' que use una imagen de haproxy a partir del Dockerfile 'DockerfileHAProxy', monte con un volumen el archivo haproxy.cfg en la ruta '/usr/local/etc/haproxy/haproxy.cfg' del contenedor y use la dirección 192.168.10.50'

6.2.2. Respuesta proporcionada por la IA:

Lo importante de la respuesta de la IA es lo siguiente:

```
services:
  balanceador-haproxy:
    build:
      context: .
      dockerfile: DockerfileHAProxy
    volumes:
      - ./haproxy.cfg:/usr/local/etc/haproxy/haproxy.cfg
    ports:
      - "192.168.10.50:80:80"
    networks:
      default:
        ipv4_address: 192.168.10.50
```

Listing 10: Servicio de balanceador en el docker-compose sugerido por IA.

6.2.3. Análisis detallado:

La IA esta vez nos proporciona un fragmento de código no muy útil, pues por ejemplo carece de directiva *image*. Por tanto nos vamos a fijar en el *docker-compose.yml* usado en el apartado 3.2.3 para Nginx y análogamente lo vamos a modificar para HAProxy, con la importante apertura del puerto 9000 que veremos especificada a continuación.

Para no repetir código, voy a mostrar únicamente el servicio *balanceador-haproxy* del *docker.compose.yml* para HAProxy, pues los demás servicios web no presentan modificaciones con respecto del *docker-compose.yml* del apartado 3.2.3.

```
balanceador-haproxy:
    image: mtorres26-haproxy-image:p2
    container_name: balanceador-haproxy
    build:
      context: .
      dockerfile: DockerfileHAProxy
    volumes:
      - ./haproxy.cfg:/usr/local/etc/haproxy/haproxy.cfg
    networks:
      red_web:
        ipv4_address: 192.168.10.50
    ports:
      - "80:80"
      - "9000:9000"
    depends_on:
      - web1
      - web2
      - web3
      - web4
      - web5
      - web6
      - web7
      - web8
```

Listing 11: Estructura final del servicio balanceador de HAProxy en el docker-compose.

7. Tareas Básicas - B7. Verificación y Pruebas del escenario de HAProxy

Para esta tarea considero que no es necesario consultar con la IA generativa debido al uso de comandos sencillos.

7.1. Iniciar los servicios con *'docker-compose up -d'* asegurando que HAProxy esté operativo.

7.2. Comprobar la correcta distribución de solicitudes por parte de HAProxy.

Primeramente comprobamos con **'docker ps'** que nuestros servicios están ejecutándose correctamente como vemos en la figura 7.

Ahora accedemos a *localhost* desde el navegador y recargamos la página varias veces para provocar un cambio de servidor[2].

[2]Al tener round-robin debería alternar servidores con cada solicitud.

```
mtorres@mtorres:~/UGR_NATIVO/CuartoCurso/SegundoCuatri/SWAP/P2/P2-mtorres26-haproxy$ docker compose up -d
[+] Running 9/11
 ⠿ Network p2-mtorres26-haproxy_red_web        Created                                                                    3.0s
 ⠿ Network p2-mtorres26-haproxy_red_servicios  Created                                                                    2.9s
 ✔ Container web1                              Started                                                                    1.5s
 ✔ Container web6                              Started                                                                    1.6s
 ✔ Container web5                              Started                                                                    1.7s
 ✔ Container web7                              Started                                                                    2.1s
 ✔ Container web3                              Started                                                                    2.1s
 ✔ Container web8                              Started                                                                    2.2s
 ✔ Container web4                              Started                                                                    2.3s
 ✔ Container web2                              Started                                                                    1.3s
 ✔ Container balanceador-haproxy               Started                                                                    2.6s
```

Figura 6: Captura mostrando el despliegue de los servicios web y balanceador.

```
mtorres@mtorres:~/UGR_NATIVO/CuartoCurso/SegundoCuatri/SWAP/P2/P2-mtorres26-haproxy$ docker ps
CONTAINER ID   IMAGE                          COMMAND                  CREATED         STATUS         PORTS
                          NAMES
237d7694cf5e   mtorres26-haproxy-image:p2   "docker-entrypoint.s…"   7 seconds ago   Up 3 seconds   0.0.0.0:80->80/tcp, :::80->80/tcp, 0.0.0.0:9000->9
000/tcp, :::9000->9000/tcp   balanceador-haproxy
c81bde8f3a46   mtorres26-apache-image:p2    "apachectl -D FOREGR…"   7 seconds ago   Up 4 seconds   80/tcp
                          web7
8e9a9319ec19   mtorres26-apache-image:p2    "apachectl -D FOREGR…"   7 seconds ago   Up 4 seconds   80/tcp
                          web3
a0e45ba274e7   mtorres26-apache-image:p2    "apachectl -D FOREGR…"   7 seconds ago   Up 4 seconds   80/tcp
                          web2
fc0f6348707c   mtorres26-apache-image:p2    "apachectl -D FOREGR…"   7 seconds ago   Up 3 seconds   80/tcp
                          web8
fc9c35cb5b95   mtorres26-apache-image:p2    "apachectl -D FOREGR…"   7 seconds ago   Up 3 seconds   80/tcp
                          web4
2fa6edbc8e34   mtorres26-apache-image:p2    "apachectl -D FOREGR…"   7 seconds ago   Up 4 seconds   80/tcp
                          web5
558ef6932a4d   mtorres26-apache-image:p2    "apachectl -D FOREGR…"   7 seconds ago   Up 4 seconds   80/tcp
                          web6
073e16bd3636   mtorres26-apache-image:p2    "apachectl -D FOREGR…"   7 seconds ago   Up 4 seconds   80/tcp
                          web1
```

Figura 7: Captura mostrando la ejecución de los servicios web y balanceador.

7.3. Observar el rendimiento a través de la interfaz de estadísticas configurada.

Lanzamos el *docker-compose.yml* y accedemos a *localhost*, donde veremos la página servida por alguno de los servidores web. Si recargamos, cada recarga nos muestra una dirección IP distinta (en eso consiste roundrobin) por lo que por ahora nos va bien la cosa.

Accedemos a la interfaz de estadísticas (figura 8) donde podemos ver el estado de los servidores y la información sobre las solicitudes que le hemos hecho.

8. Tareas Avanzadas - A1. Configuraciones avanzadas de Nginx.

8.1. Implementar estrategias de balanceo de carga avanzadas en nginx.conf, como el balanceo basado en menor tiempo de respuesta o ponderado y analiza el impacto de esas configuraciones en el escenario de balanceo.

8.1.1. Consulta realizada IA:

'Genera el código de un archivo de configuración de Nginx "nginx.conf"que utilice balanceo de carga basado en menor tiempo de respuesta'

8.1.2. Respuesta proporcionada por la IA:

```
    worker_processes  1;

events {
    worker_connections  1024;
}
```

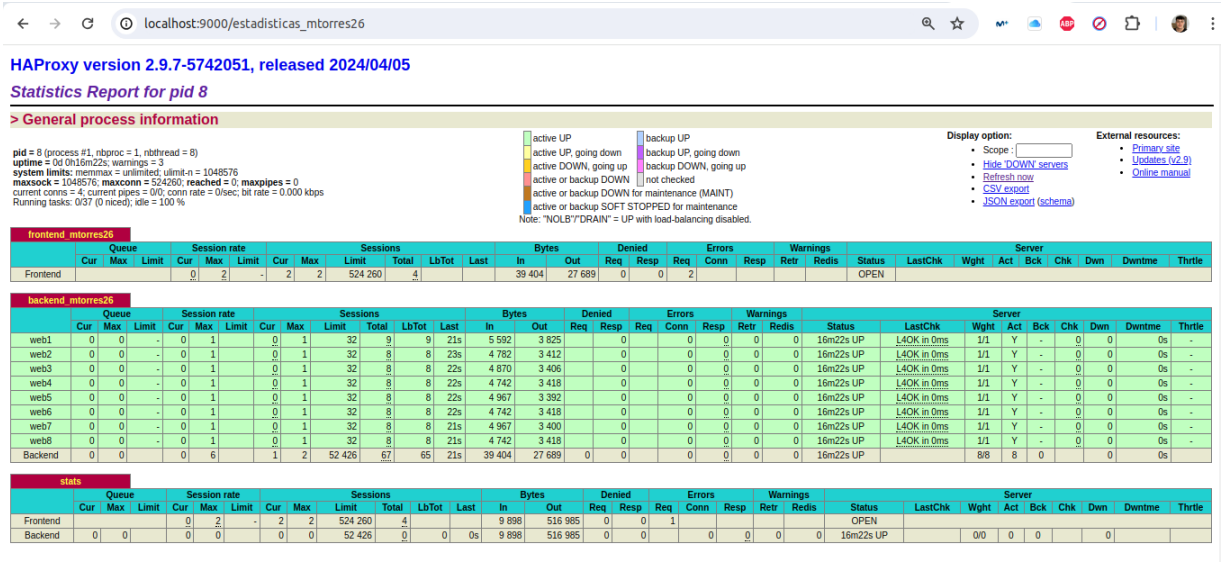

Figura 8: Captura que muestra la interfaz de estadísticas.

```
http {
    upstream backend {
        least_conn;
        server backend1.example.com;
        server backend2.example.com;
        server backend3.example.com;
    }

    server {
        listen 80;

        location / {
            proxy_pass http://backend;
        }
    }
}
```

Listing 12: Archivo de configuración avanzada con menor tiempo de respuesta generado por IA.

8.1.3. Análisis detallado:

Para usar menor tiempo de respuesta como algoritmo de balanceo de carga, la IA nos propone usar *least-conn* en el apartado del backend. Sin embargo esto no es correcto pues si observamos la documentación oficial de Nginx nos indica que el código sería *least-time header;*. El problema está en que esta directiva solo es posible con Nginx Plus y no con la versión normal.

Vamos a probar con algoritmos de balanceo de carga ponderados:

8.1.4. Consulta realizada IA:

'Genera el código para un archivo de configuración de Nginx que use el algoritmo de balanceo de carga ponderado'

8.1.5. Respuesta proporcionada por la IA:

```
upstream backend {
    server backend1.example.com weight=3;
    server backend2.example.com weight=2;
    server backend3.example.com weight=1;
}
```

Listing 13: Código con ponderación de servidores generado por IA.

8.1.6. Análisis detallado:

Introducimos a nuestro archivo *nginx.conf* lo que nos dice la IA y vamos a comprobar si los servidores con mayor ponderación reciben más solicitudes que los que tienen la ponderación por defecto.

```
upstream backend_mtorres26 {
        # nuestros 8 contenedores
        server 192.168.10.2;
        server 192.168.10.3;
        server 192.168.10.4 weight=3;
        server 192.168.10.5;
        server 192.168.10.6;
        server 192.168.10.7 weight=2;
        server 192.168.10.8;
        server 192.168.10.9;
}
```

Listing 14: Backend del 'nginx.conf'

Accedemos a *localhost* y efectivamente tras recargar la página repetidas veces, los servidores con más peso aparecen n veces más que los demás, siendo n el peso asignado en la configuración.

9. Tareas Avanzadas - A2. Configuraciones avanzadas de HA-Proxy.

9.1. Implementar estrategias de balanceo de carga avanzadas en haproxy.cfg, como el balanceo basado en menor tiempo de respuesta o ponderado y analiza el impacto de esas configuraciones en el escenario de balanceo.

9.1.1. Consulta realizada IA:

'Genera el código para un archivo de configuración de HAProxy que use el algoritmo de balanceo de carga ponderado'

9.1.2. Respuesta proporcionada por la IA:

```
backend my_backend
    balance roundrobin
    server server1 192.168.0.101:80 weight 3 check
    server server2 192.168.0.102:80 weight 2 check
    server server3 192.168.0.103:80 weight 1 check
```

Listing 15: Propuesta de ponderación de servidores generado por IA.

9.1.3. Análisis detallado:

Análogamente realizamos lo mismo que hemos hecho con Nginx. Vamos a cambiar simplemente los servidores ponderados que serán los contenedores *web1* y *web2*.

```
backend backend_mtorres26
    mode http
    server web1 192.168.10.2:80 maxconn 32 check weight 2
    server web2 192.168.10.3:80 maxconn 32 check weight 3
    server web3 192.168.10.4:80 maxconn 32 check
    server web4 192.168.10.5:80 maxconn 32 check
    server web5 192.168.10.6:80 maxconn 32 check
    server web6 192.168.10.7:80 maxconn 32 check
    server web7 192.168.10.8:80 maxconn 32 check
    server web8 192.168.10.9:80 maxconn 32 check
```

Listing 16: Estructura final de nuestro backend de haproxy.cfg

Efectivamente, tras recargar el *localhost* múltiples veces y posteriormente visitar la URL de estadísticas (figura 9) podemos observar que el contenedor *web1* tiene el doble de sesiones (weight 2) y el contenedor *web2* el triple de sesiones (weight 3).

backend_mtorres26

	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
web1	0	0	-	0	2		0	1	32	12	12	3s	7 281	5 112		0		0	0	0	0	1m UP	L4OK in 0ms	2/2	Y	-	0	0	0s	-
web2	0	0	-	0	2		0	1	32	18	18	3s	11 065	7 656		0		0	0	0	0	1m UP	L4OK in 0ms	3/3	Y	-	0	0	0s	-
web3	0	0	-	0	1		0	1	32	6	6	5s	3 647	2 556		0		0	0	0	0	1m UP	L4OK in 0ms	1/1	Y	-	0	0	0s	-
web4	0	0	-	0	1		0	1	32	6	6	4s	3 634	2 556		0		0	0	0	0	1m UP	L4OK in 0ms	1/1	Y	-	0	0	0s	-
web5	0	0	-	0	1		0	1	32	6	6	4s	3 709	2 544		0		0	0	0	0	1m UP	L4OK in 0ms	1/1	Y	-	0	0	0s	-
web6	0	0	-	1	1		0	1	32	6	6	2s	3 634	2 556		0		0	0	0	0	1m UP	L4OK in 0ms	1/1	Y	-	0	0	0s	-
web7	0	0	-	0	1		0	1	32	5	5	6s	3 093	2 125		0		0	0	0	0	1m UP	L4OK in 0ms	1/1	Y	-	0	0	0s	-
web8	0	0	-	0	1		0	1	32	5	5	5s	3 018	2 131		0		0	0	0	0	1m UP	L4OK in 0ms	1/1	Y	-	0	0	0s	-
Backend	0	0		0	6		1	2	52 426	65	64	2s	39 081	27 236	0	0		0	0	0	0	1m UP		11/11	8	0		0	0s	

Figura 9: Captura mostrando las estadísticas de HAProxy.

10. Tareas Avanzadas - A4. : Investigación y Pruebas de Tolerancia a Fallos.

10.1. Realizar pruebas de tolerancia a fallos apagando intencionadamente instancias de servidor web para observar la reacción y la reasignación de carga de los balanceadores.

10.1.1. Pruebas en Nginx:

Vamos a lanzar los servidores y con el comando *'docker compose stop web[1-9]'* vamos a ir parando manualmente los contenedores para ver cómo se distribuyen las solicitudes (figura 10). Visitamos *localhost* y recargamos varias veces la página. Se puede denotar cierto tiempo de espera entre recarga y recarga debido a que casi todos los servidores se encuentran apagados.

10.1.2. Pruebas en HAProxy:

Análogamente a Nginx, vamos a lanzar los servidores y con el comando *'docker compose stop web[1-9]'* vamos a ir parando manualmente los contenedores para ver cómo se distribuyen las solicitudes (figura 11).

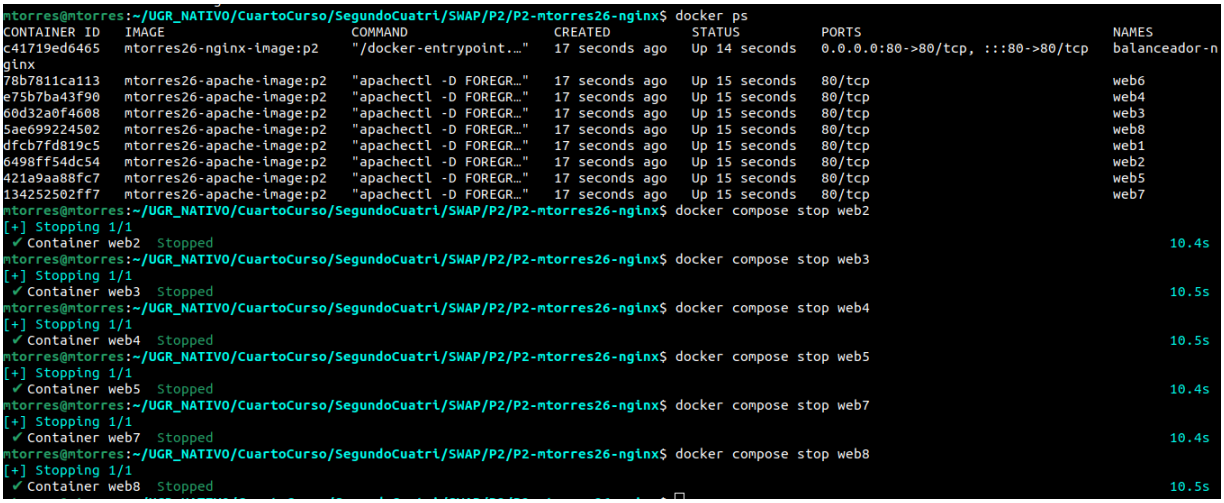

Figura 10: Captura mostrando el estado de los contenedores y su apagado manualmente.

Ahora, cada vez que recargamos *localhost* sólo actúan los contenedores *web1* y *web6*, que son los que hemos dejado activos como podemos ver en la figura 12. Al contrario que en el apartado anterior (apartado 10.1.1), podemos denotar que el tiempo de espera entre recarga y recarga es casi despreciable.

El contenedor *web1* tiene el doble de sesiones que el *web6* porque estamos utilizando la configuración avanzada con ponderaciones realizada en el apartado 9.

```
mtorres@mtorres:~/UGR_NATIVO/CuartoCurso/SegundoCuatri/SWAP/P2/P2-mtorres26-haproxy$ docker ps
CONTAINER ID   IMAGE                          COMMAND                  CREATED          STATUS          PORTS
                                NAMES
1f84892b8693   mtorres26-haproxy-image:p2   "docker-entrypoint.s…"   21 minutes ago   Up 21 minutes   0.0.0.0:80->80/tcp, :::80->80/tcp, 0.0.0.0:9000-
>9000/tcp, :::9000->9000/tcp   balanceador-haproxy
3243cb9c2f61   mtorres26-apache-image:p2    "apachectl -D FOREGR…"   21 minutes ago   Up 21 minutes   80/tcp
                                web4
f79e3ffa5860   mtorres26-apache-image:p2    "apachectl -D FOREGR…"   21 minutes ago   Up 21 minutes   80/tcp
                                web5
80a83c30be81   mtorres26-apache-image:p2    "apachectl -D FOREGR…"   21 minutes ago   Up 21 minutes   80/tcp
                                web1
2ab7a2f2c802   mtorres26-apache-image:p2    "apachectl -D FOREGR…"   21 minutes ago   Up 21 minutes   80/tcp
                                web6
969281ab8a62   mtorres26-apache-image:p2    "apachectl -D FOREGR…"   21 minutes ago   Up 21 minutes   80/tcp
                                web3
a0ef47b70f0b   mtorres26-apache-image:p2    "apachectl -D FOREGR…"   21 minutes ago   Up 21 minutes   80/tcp
                                web8
19bfc1fbbe2f   mtorres26-apache-image:p2    "apachectl -D FOREGR…"   21 minutes ago   Up 21 minutes   80/tcp
                                web7
e965ced40710   mtorres26-apache-image:p2    "apachectl -D FOREGR…"   21 minutes ago   Up 21 minutes   80/tcp
                                web2
mtorres@mtorres:~/UGR_NATIVO/CuartoCurso/SegundoCuatri/SWAP/P2/P2-mtorres26-haproxy$ docker compose stop web3
[+] Stopping 1/1
 ✔ Container web3  Stopped                                                                                                    10.4s
mtorres@mtorres:~/UGR_NATIVO/CuartoCurso/SegundoCuatri/SWAP/P2/P2-mtorres26-haproxy$ docker compose stop web2
[+] Stopping 1/1
 ✔ Container web2  Stopped                                                                                                    10.4s
mtorres@mtorres:~/UGR_NATIVO/CuartoCurso/SegundoCuatri/SWAP/P2/P2-mtorres26-haproxy$ docker compose stop web4
[+] Stopping 1/1
 ✔ Container web4  Stopped                                                                                                    10.5s
mtorres@mtorres:~/UGR_NATIVO/CuartoCurso/SegundoCuatri/SWAP/P2/P2-mtorres26-haproxy$ docker compose stop web5
[+] Stopping 1/1
 ✔ Container web5  Stopped                                                                                                    10.4s
mtorres@mtorres:~/UGR_NATIVO/CuartoCurso/SegundoCuatri/SWAP/P2/P2-mtorres26-haproxy$ docker compose stop web7
[+] Stopping 1/1
 ✔ Container web7  Stopped                                                                                                    10.3s
mtorres@mtorres:~/UGR_NATIVO/CuartoCurso/SegundoCuatri/SWAP/P2/P2-mtorres26-haproxy$ docker compose stop web8
[+] Stopping 1/1
 ✔ Container web8  Stopped                                                                                                    10.4s
```

Figura 11: Captura mostrando el estado de los contenedores y su apagado manualmente.

backend_mtorres26

	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
web1	0	0	-	0	4		0	1	32	50	50	6m38s	29 670	21 382		0		0	0	0	0	32m33s UP	L4OK in 0ms	2/2	Y	-	0	0	0s	-
web2	0	0	-	0	2		0	1	32	26	26	9m49s	15 846	11 074		0		0	0	0	0	9m19s DOWN	L4TOUT in 2002ms	3/3	Y	-	3	1	9m19s	-
web3	0	0	-	0	2		0	2	32	14	8	9m7s	4 817	2 990		0		2	0	6	0	9m51s DOWN	L4CON in 1070ms	1/1	Y	-	3	1	9m51s	-
web4	0	0	-	0	1		0	1	32	9	9	9m49s	5 371	3 843		0		0	0	0	0	9m6s DOWN	* L4TOUT in 2003ms	1/1	Y	-	3	1	9m6s	-
web5	0	0	-	0	1		0	1	32	9	9	9m49s	5 521	3 823		0		0	0	0	0	8m50s DOWN	* L4TOUT in 2003ms	1/1	Y	-	3	1	8m50s	-
web6	0	0	-	0	2		0	1	32	24	24	6m39s	14 356	10 254		0		0	0	0	0	32m33s UP	L4OK in 0ms	1/1	Y	-	0	0	0s	-
web7	0	0	-	0	1		0	1	32	8	8	9m50s	4 892	3 406		0		0	0	0	0	8m34s DOWN	* L4TOUT in 2003ms	1/1	Y	-	3	1	8m34s	-
web8	0	0	-	0	1		0	1	32	8	8	9m50s	4 892	3 406		0		0	0	0	0	8m22s DOWN	* L4TOUT in 2003ms	1/1	Y	-	3	1	8m22s	-
Backend	0	0		0	8		0	4	52 426	145	142	6m38s	85 365	60 178	0	0		2	0	6	0	32m33s UP		3/3	2	0		0	0s	

Figura 12: Captura mostrando la página de estadísticas y las solicitudes realizadas a los servidores.