

Nombre del estudiante: Daniel Alconchel Vázquez

Semestre: 2

Año Académico: 2023/2024



UNIVERSIDAD
DE GRANADA

E.T.S. de Ingenierías Informática y de Telecomunicación

Práctica 2: Balanceo de Carga

Servidores Web de Altas Prestaciones

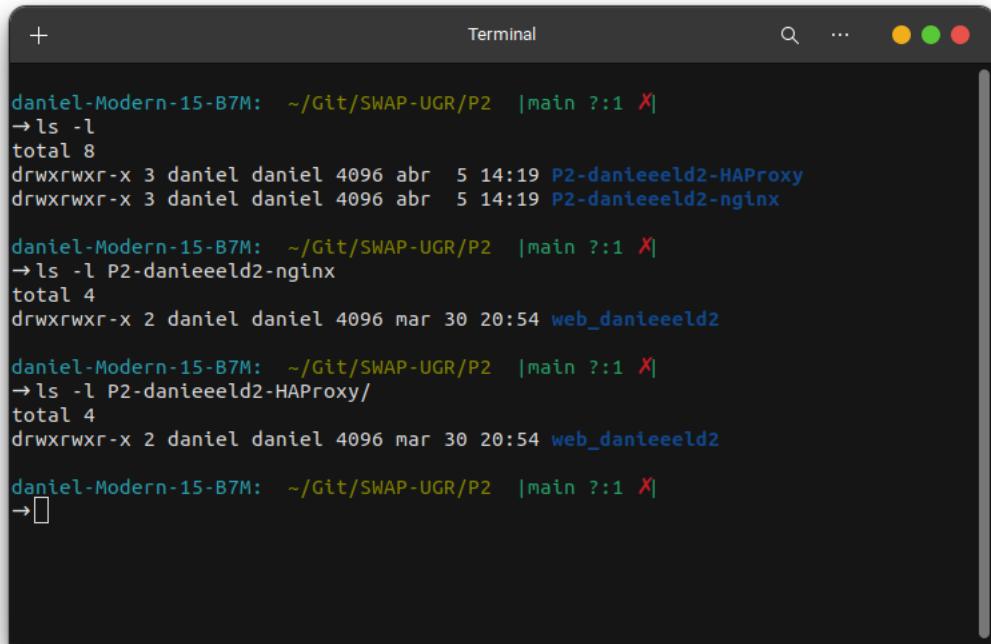
Índice

1. Tareas Básicas	2
1.1. Creación del directorio y espacio de trabajo para cada balanceador	2
1.2. Configuración de Nginx como Balanceador de Carga	2
1.3. Configuración de Nginx con Docker Compose	4
1.4. Verificación y pruebas del balanceador Nginx	8
1.5. Configuración de HAProxy como Balanceador de Carga	9
1.6. Configuración de HAProxy con Docker Compose	10
1.7. Verificación y pruebas del balanceador HAProxy	11
2. Tareas Avanzadas	12
2.1. Configuraciones Avanzadas de Nginx	12
2.1.1. Balanceo basado en menor tiempo de respuesta	12
2.1.2. Balanceo ponderado	13
2.2. Configuraciones Avanzadas de HAProxy	15
2.2.1. Balanceo basado en menor número de conexiones activas	15
2.2.2. Balanceo ponderado	16
2.3. Experimentación con Diferentes Balanceadores de Carga	18
2.3.1. Gobetween	18
2.3.2. Traefik	22
2.4. Investigación y Pruebas de Tolerancia a Fallos	28
2.4.1. Nginx	28
2.4.2. HAProxy	30
2.4.3. Gobetween	31
2.4.4. Traefik	32
2.5. Automatización de escalado de escenario	33
2.5.1. Monitorización de la Carga de los Servidores	33
2.5.2. Script para escalar servicios	36
2.5.3. Automatización del script	48

1. Tareas Básicas

1.1. Creación del directorio y espacio de trabajo para cada balanceador

Para esta parte, simplemente, creamos los directorios tal y como nos dice el guión de prácticas.



```
daniel-Modern-15-B7M: ~/Git/SWAP-UGR/P2 |main ?:1 X
→ls -l
total 8
drwxrwxr-x 3 daniel daniel 4096 abr  5 14:19 P2-danieeeld2-HAProxy
drwxrwxr-x 3 daniel daniel 4096 abr  5 14:19 P2-danieeeld2-nginx

daniel-Modern-15-B7M: ~/Git/SWAP-UGR/P2 |main ?:1 X
→ls -l P2-danieeeld2-nginx
total 4
drwxrwxr-x 2 daniel daniel 4096 mar 30 20:54 web_danieeeld2

daniel-Modern-15-B7M: ~/Git/SWAP-UGR/P2 |main ?:1 X
→ls -l P2-danieeeld2-HAProxy/
total 4
drwxrwxr-x 2 daniel daniel 4096 mar 30 20:54 web_danieeeld2

daniel-Modern-15-B7M: ~/Git/SWAP-UGR/P2 |main ?:1 X
→
```

1.2. Configuración de Nginx como Balanceador de Carga

Para la siguiente parte, simplemente creamos los archivos `DockerfileNginx` y `nginx.conf`¹ y seguimos las instrucciones del guión:

¹Si quisieramos usar Round Robin, no hay que especificar ningún algoritmo de balanceo (Comentamos la línea de `least_conn`; y ya tendríamos la configuración lista)

```

nginx.conf - P2 - Visual Studio Code

File Edit Selection View Go Run Terminal Help

P2
  P2-danieeld2-HAProxy / web_danieeld2
    index.php
    apacheConf_danieeld2
      apache2.conf
      index.php
      docker-compose.yml
      DockerfileApache_danieeld2
      DockerfileNginx
      nginx.conf

nginx.conf
1 events {
2   worker_connections 1024;
3 }
4
5 http {
6   upstream backend_danieeld2 {
7     # Algoritmo de balanceo
8     least_conn;
9     server 192.168.10.2;
10    server 192.168.10.3;
11    server 192.168.10.4;
12    server 192.168.10.5;
13    server 192.168.10.6;
14    server 192.168.10.7;
15    server 192.168.10.8;
16    server 192.168.10.9;
17  }
18
19  server {
20    listen 80;
21    server_name nginx_danieeld2;
22    access_log /var/log/nginx/nginx_danieeld2.access.log;
23    error_log /var/log/nginx/nginx_danieeld2.error.log;
24    location / {
25      proxy_pass http://backend_danieeld2;
26      proxy_set_header Cookie $http_cookie;
27      proxy_hide_header Set-Cookie;
28    }
29    location /estadisticas_danieeld2 {
30      stub_status on;
31    }
32  }
33

```

```

DockerfileNginx
1 FROM nginx:latest

```

Nota: La directiva de `events` la he incluido porque me daba un error posteriormente en el `docker-compose`. La pregunta que le hice a `ChatGPT`:

- **Yo:** Tengo el siguiente error al levantar un servicio de `nginx` con `docker`:

```

balanceador-nginx-1 | /docke-entrypoint.sh:
Configuration complete; ready for start up
balanceador-nginx-1 | 2024/04/05 13:49:23 [emerg] 1#1:
no "events" section in configuration
balanceador-nginx-1 | nginx:
[emerg] no "events" section in configuration

```

- **ChatGPT:** El mensaje de error indica que no hay una sección 'events' en el archivo de configuración de nginx. La sección 'events' es un bloque requerido en los archivos de configuración de nginx donde se definen ajustes globales relacionados con el modelo de eventos utilizado por nginx, como conexiones de trabajadores y otros parámetros relacionados con eventos.

Para resolver este problema, necesitas agregar la sección 'events' a tu archivo `nginx.conf`. Aquí te muestro cómo puedes modificar tu archivo `nginx.conf` para incluir la sección 'events':

```

events {
  worker_connections 1024;
}
...

```

Asegúrate de ajustar el valor de 'worker_connections' en la sección 'events' según los requisitos de tu servidor. Este valor define el número máximo de conexiones simultáneas que puede abrir cada proceso de trabajador.

Análisis de la salida: Me parece una salida adecuada, con una configuración básica para solventar el problema que teníamos anteriormente. Esta directiva indica el número máximo de conexiones simultáneas que un único proceso de **nginx** puede manejar.

1.3. Configuración de Nginx con Docker Compose

Comenzamos copiando el **DockerfileApache_danieeeld2** en el directorio de **nginx**, así como la configuración personalizada que hicimos de **Apache** y crearemos una nueva imagen con el tag **p2**. Para crear esta imagen, ejecutamos:

```
docker build -t danieeeld2-apache-image:p2 -f DockerfileApache_danieeeld2 .
```

Una vez creada esta imagen, podemos rescatar el **docker-compose.yml** de la práctica anterior, para levantar las instancias de **Apache**:

```
version: '2.0'

services:
  web:
    image: danieeeld2-apache-image:p2
    privileged: true
    deploy:
      replicas: 8
    volumes:
      - ./web_danieeeld2:/var/www/html
    networks:
      red_web:
      red_servicios:

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
```

Notad que hemos cambiado el tag de la imagen de **p1** a **p2**.

Si recordamos de la práctica anterior, este **dockerfile** es más escalable, ya que no tenemos que crear un servicio por cada instancia de **Apache**, pero tenía el inconveniente de que desordenaba las **IPs**. Para ello, creamos una script para suplir esta necesidad. Ahora, vamos a intentar crear los contenedores con sus **IPs** ya asignadas antes de crear la instancia de **nginx**, por lo que vamos a necesitar modificar este archivo.

Como no me gusta crear a mano un servicio de **web** por cada instancia de **Apache**, vamos a crear una script de **bash** que acepte como parámetro un entero y nos cree el **docker-compose** con tantos servicios de **webX** como le hayamos indicado. Para ello, vamos a usar la ayuda de **ChatGPT**.

- **Yo:** Quiero crear uns script de bash que me cree el siguiente docker-compose: *Le paso el docker-compose anterior*, pero indicando explícitamente las instancias de webX. la script acepta como parámetro un entero y genera el docker compose con tantos webX como le hayamos indicado. Además, le indicamos directamente a cada webX sus IPs, que serán 192.168.10.((X+1)) y 192.168.20.((X+1))

- **ChatGPT:** Puedes crear un script de Bash para generar el archivo docker-compose.yml con las instancias de webX según el parámetro que le pases. Aquí tienes un ejemplo de cómo hacerlo:

```

#!/bin/bash

if [ "$#" -ne 1 ]; then
    echo "Usage: $0 <number_of_instances>"
    exit 1
fi

num_instances=$1

cat <<EOF >docker-compose.yml
version: '2.0'

services:
EOF

for ((i=1; i<=$num_instances; i++)); do
    cat <<EOF >>docker-compose.yml
web$i:
    image: danieeeld2-apache-image:p2
    privileged: true
    deploy:
        replicas: 8
    volumes:
        - ./web_danieeeld2:/var/www/html
    networks:
        red_web:
            ipv4_address: 192.168.10.$((i+1))
        red_servicios:
            ipv4_address: 192.168.20.$((i+1))

EOF
done

cat <<EOF >>docker-compose.yml

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
EOF

```

Análisis de la salida: La idea está bastante bien, pero tiene algunos fallos. Si nos fijamos, en cada instancia de `webX` nos está generando 8 replicas, por lo que tendríamos que eliminar esas líneas de la script para que no lo genere. Además, le voy a añadir la línea `container_name: web$i` para que ponga el nombre de los contenedores de forma correcta. Por tanto, la script final sería:

```
#!/bin/bash

if [ "$#" -ne 1 ]; then
    echo "Usage: $0 <number_of_instances>"
    exit 1
fi

num_instances=$1

cat <<EOF >docker-compose.yml
version: '2.0'

services:
EOF

for (( i=1; i<=$num_instances; i++)); do
    cat <<EOF >>docker-compose.yml
    web$i:
        image: danieeld2-apache-image:p2
        container_name: web$i
        privileged: true
        volumes:
            - ./web_danieeld2:/var/www/html
        networks:
            red_web:
                ipv4_address: 192.168.10.$((i+1))
            red_servicios:
                ipv4_address: 192.168.20.$((i+1))

    EOF
done

cat <<EOF >>docker-compose.yml

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
EOF
```

Podemos ahora ejecutar `/scripts/generar-docker-compose.sh 8` y tendremos ya el `docker-compose.yml` listo para usar.

```

daniel-Modern-15-B7M: ~/.../P2/P2-danieeeld2-nginx |main ?:1 X
→ ./scripts/generar-docker-compose.sh 8
daniel-Modern-15-B7M: ~/.../P2/P2-danieeeld2-nginx |main ?:1 X
→ docker compose up
[+] Running 10/10
  ✓ Network p2-danieeeld2-nginx_red_web          Created
  ✓ Network p2-danieeeld2-nginx_red_servicios     Created
  ✓ Container web2                                Created
  ✓ Container web7                                Created
  ✓ Container web8                                Created
  ✓ Container web5                                Created
  ✓ Container web4                                Created
  ✓ Container web1                                Created
  ✓ Container web6                                Created
  ✓ Container web3                                Created
Attaching to web1, web2, web3, web4, web5, web6, web7, web8
web2 | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 192.168.10.3. Set the 'ServerName' directive globally to suppress this message
web5 | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 192.168.20.6. Set the 'ServerName' directive globally to suppress this message
web1 | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 192.168.20.2. Set the 'ServerName' directive globally to suppress this message
web6 | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 192.168.20.7. Set the 'ServerName' directive globally to suppress this message
web4 | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 192.168.20.8. Set the 'ServerName' directive globally to suppress this message
web3 | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 192.168.20.4. Set the 'ServerName' directive globally to suppress this message
web8 | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 192.168.20.9. Set the 'ServerName' directive globally to suppress this message
web7 | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 192.168.20.8. Set the 'ServerName' directive globally to suppress this message

```

Antes de continuar, vamos a dejar creada la imagen de `nginx`, para lo que ejecutamos:

```
docker build -t danieeeld2-nginx-image:p2 -f DockerfileNginx .
```

Bien, ahora tenemos que crear un nuevo servicio llamado `balanceador-nginx`. Este servicio depende de las instancias de `webX` y utiliza como base la imagen que acabamos de crear. Además, debe escuchar en el puerto 80 para poder acceder desde `localhost` directamente. Aunque es bastante simple, vamos a preguntarle a ChatGPT:

- **Yo:** danieeeld2-nginx-image:p2 tengo esta imagen de docker y quiero crear un servicio en docker-compose, llamado balanceador-nginx, que use esta imagen, escuche en el puerto 80, cree un volumen para usar `./nginx.conf` que he creado y que dependa de `webX`
- **ChatGPT:** Puedes crear un servicio en docker-compose llamado "balanceador-nginx" que utilice la imagen "danieeeld2-nginx-image:p2", escuche en el puerto 80, cree un volumen para usar un archivo de configuración llamado "nginx.conf" que has creado previamente, y que dependa de un servicio llamado "webX". Aquí tienes un ejemplo de cómo podría ser tu archivo docker-compose.yml:

```

services:
  balanceador-nginx:
    image: danieeeld2-nginx-image:p2
    ports:
      - "80:80"
    volumes:
      - ./nginx.conf:/etc/nginx/nginx.conf
    depends_on:
      - webX
...

```

Análisis de la salida: La salida es buena, pero se me ha olvidado especificar que se conecte también a `red_web` con ip `192.168.10.50`. Vamos a aprovechar y vamos a incluir todo en la script que genera el `docker-compose`:

```
# Generar WebX
...
cat <<EOF >>docker-compose.yml
balanceador-nginx:
  image: danieeld2-nginx-image:p2
  container_name: balanceador-nginx
  ports:
    - "80:80"
  command: [ 'nginx' , '-g' , 'daemon off;' ]
  volumes:
    - ./nginx.conf:/etc/nginx/nginx.conf
  networks:
    red_web:
      ipv4_address: 192.168.10.50
  depends_on:
EOF

for (( i=1; i<=$num_instances; i++)); do
  echo "      - web$i" >>docker-compose.yml
done

# Generar red_web , red_servicios
...
```

De esta forma, hacemos que `balanceador-nginx` dependa de los `webX` existentes. Además, montamos su correspondiente volumen, hacemos que escuche en el puerto 80 y le asignamos la ip correspondiente. Por otro lado, he añadido `command: ['nginx', '-g', 'daemon off;']` para que reinicie el servicio de `nginx` al montar la nueva configuración y todo funcione correctamente.

Podemos ejecutar entonces

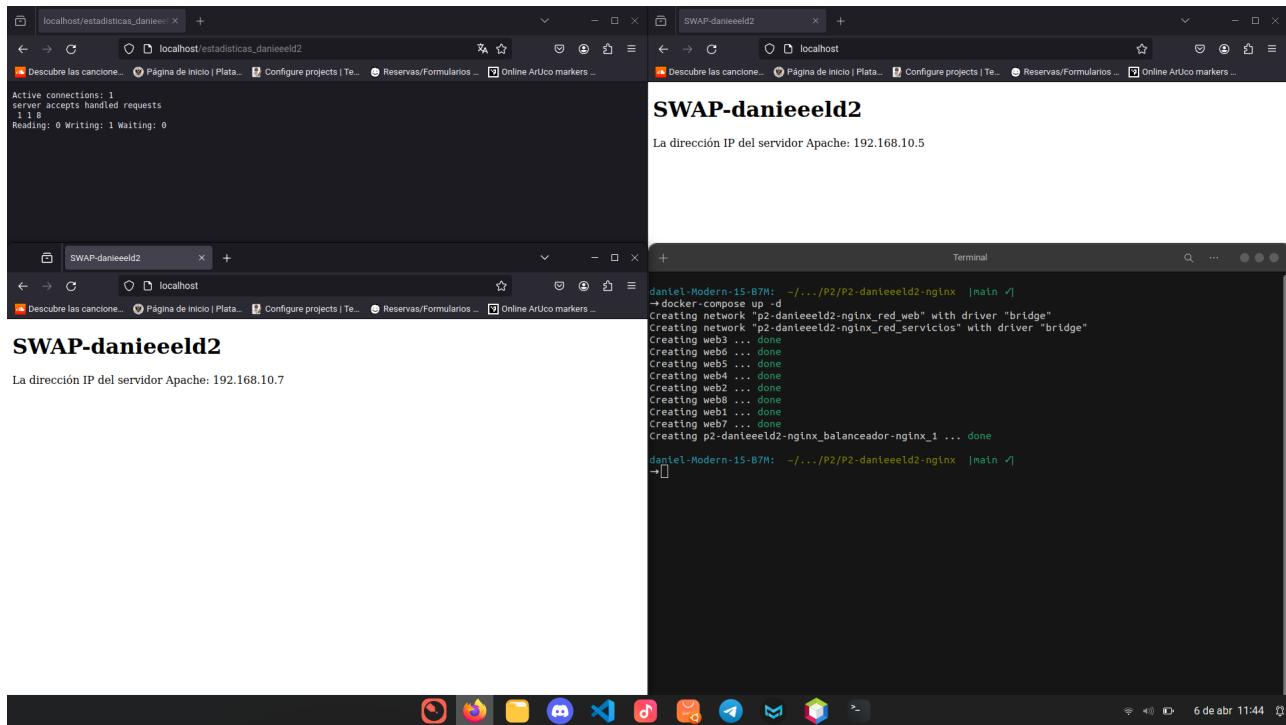
```
./scripts/generar-docker-compose.sh 8
```

para dejar generado el `docker-compose`.

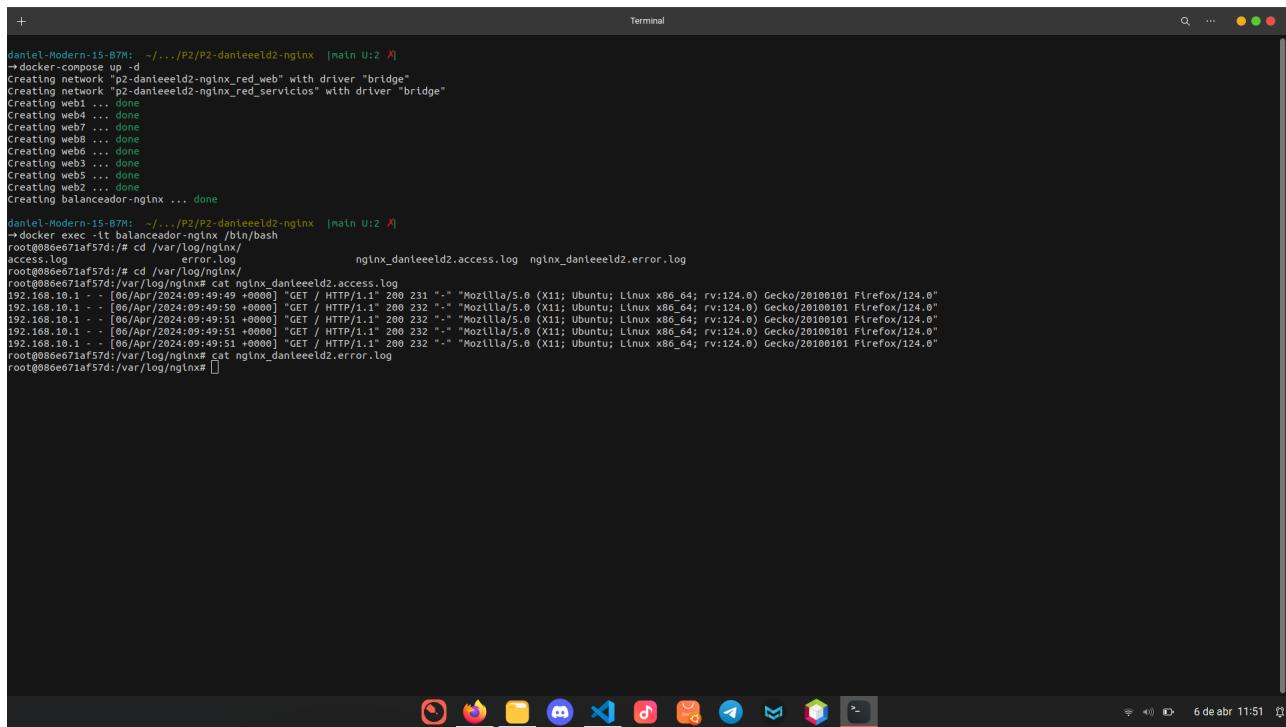
Nota: *Pese al hacer el docker-compose de esta forma, hay veces que no asigna una de las IPs, así que he rescatado las scripts de la P1 de comprobación de conectividad y de asignación manual de IPs para en caso de fallo. Intenté hacer la script así para ver si se solventaba este fallo, pero sigue ocurriendo, así que optaremos por ejecutar la script de asignación de IPs siempre después de arrancar.*

1.4. Verificación y pruebas del balanceador Nginx

Ejecutemos `docker-compose up -d` y verifiquemos que todo funciona correctamente:



También podemos acceder al contenedor de nginx para consultar el sistema de logs:



1.5. Configuración de HAProxy como Balanceador de Carga

Vamos a repetir el proceso anterior, pero configurando ahora HAProxy. Comenzamos creando los ficheros DockerfileHAProxy y haproxy.cfg de la forma en la que se indica en el guión:

```

global
    stats socket /var/lib/haproxy/stats
defaults
    mode http
    timeout connect 5000ms
    timeout client 50000ms
    timeout server 50000ms
frontend frontend_danieeld2
    bind :80
    default_backend backend_danieeld2
backend backend_danieeld2
    balance roundRobin
    server web1 192.168.10.2:80 maxconn 32
    server web2 192.168.10.3:80 maxconn 32
    server web3 192.168.10.4:80 maxconn 32
    server web4 192.168.10.5:80 maxconn 32
    server web5 192.168.10.6:80 maxconn 32
    server web6 192.168.10.7:80 maxconn 32
    server web7 192.168.10.8:80 maxconn 32
    server web8 192.168.10.9:80 maxconn 32
listen stats
    bind *:9000
    mode http
    stats enable
    stats uri /estadisticas.danieeld2
    stats realm Haproxy_Statistics
    stats auth danieeld2:SWAP1234

```

Como podemos ver, la configuración es exactamente la del guión, menos por el detalle de `defaults`. Este detalle surge de un error que me dió al quitar esta directriz:

```

balanceador-haproxy | [ALERT]      (1) : config : parsing
[/usr/local/etc/haproxy/haproxy.cfg:2] : 'listen' or 'defaults' expected.
balanceador-haproxy | [ALERT]      (1) : config : parsing
[/usr/local/etc/haproxy/haproxy.cfg:3] : 'listen' or 'defaults' expected.
balanceador-haproxy | [ALERT]      (1) : config : parsing
[/usr/local/etc/haproxy/haproxy.cfg:24]: Missing LF on last line, file might
have been truncated at position 5.

```

Pese a que el error indica que debe haber `listen` (el cuál existe) o `defaults`, tuve que acabar poniendo los dos.

Antes de proceder, vamos a dejar creada la imagen que usaremos para el balanceador:

```
docker build -t danieeld2-haproxy-image:p2 -f DockerfileHAProxy .
```

1.6. Configuración de HAProxy con Docker Compose

Para este paso, podemos usar exactamente la misma script que en la configuración de `nginx`, pero cambiando la parte de la script que crea el `balanceador-nginx` (1.3). Sera muy semejante, solo que ahora debe escuchar tanto en el puerto 80 como en el 9000 (que es donde hemos habilitado las estadísticas) y usar la imagen y archivo de configuración correspondiente.

En conclusión, quedaría de la siguiente forma:

```

# Generar WebX
...
cat <<EOF >>docker-compose.yml
balanceador-haproxy:

```

```

image: danieeeld2-haproxy-image:p2
container_name: balanceador-haproxy
ports:
  - "80:80"
  - "9000:9000"
command: ["haproxy", "-f", "/usr/local/etc/haproxy/haproxy.cfg", "-d"]
volumes:
  - ./haproxy.cfg:/usr/local/etc/haproxy/haproxy.cfg
networks:
  red_web:
    ipv4_address: 192.168.10.50
depends_on:
EOF

for (( i=1; i<=$num_instances; i++)); do
  echo "      - web$i" >> docker-compose.yml
done

# Generar red_web, red_servicios
...

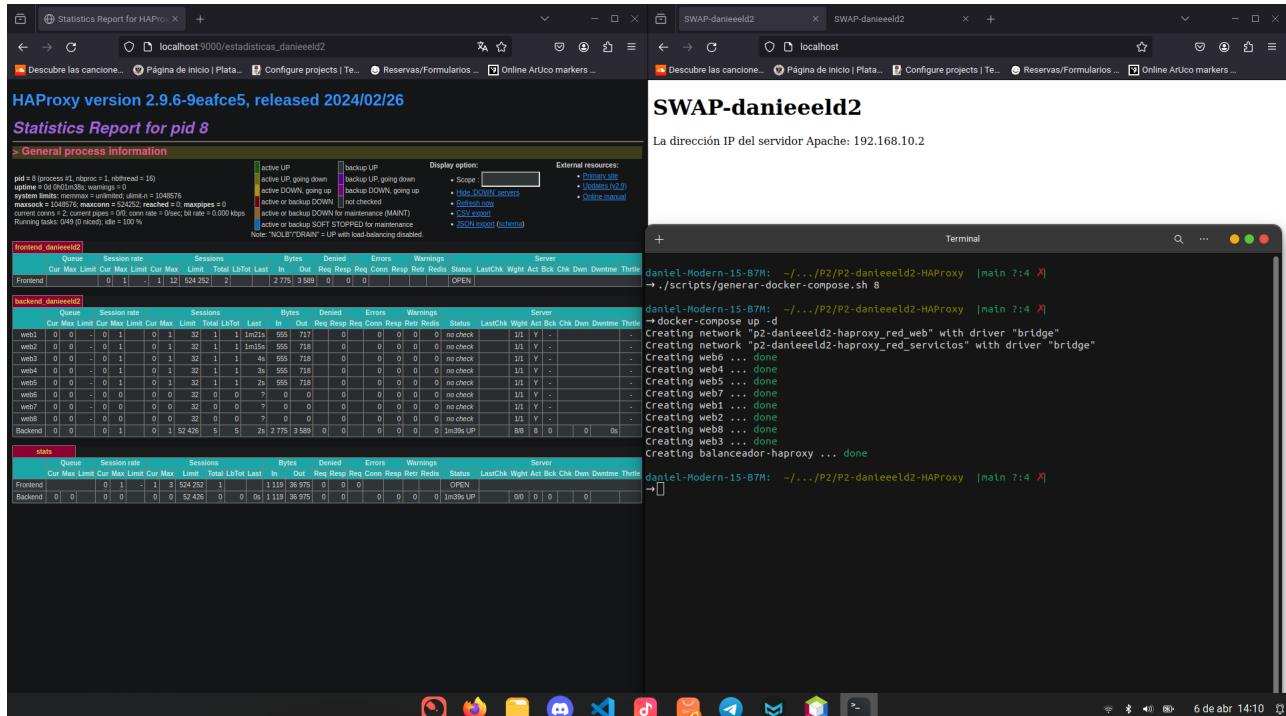
```

Podemos ejecutar ahora `./scripts/generar-docker-compose.sh 8` y tendremos listo el fichero de `docker-compose` para usar HAProxy como balanceador.

Nota: También he incorporado la script de asignación de IPs y comprobación de conectividad, debido al aparente fallo que tiene docker al asignar las IPs.

1.7. Verificación y pruebas del balanceador HAProxy

Ejecutamos `docker-compose up -d` y verificamos que todo funcione correctamente:



Como vemos, todo funciona correctamente².

²Si en la configuración añadimos la directiva `check` a cada servidor del backend, comprobará la disponibilidad del mismo y saldrá en colores en las estadísticas, para identificar fácilmente cuales están caídos y cuales funcionan correctamente.

2. Tareas Avanzadas

2.1. Configuraciones Avanzadas de Nginx

2.1.1. Balanceo basado en menor tiempo de respuesta

Como siempre vamos a empezar preguntando a ChatGPT:

- **Yo:** Tengo el siguiente fichero de configuración de nginx: (*Le paso el fichero de configuración de Nginx 1.2*) ¿Cómo puedo implementar una estrategia de balanceo basada en el menor tiempo de respuesta?
- **ChatGPT:** Para implementar una estrategia de balanceo de carga basada en el menor tiempo de respuesta en nginx, puedes utilizar el módulo `ngx_http_upstream_module` junto con la directiva `least_time`. Aquí te muestro cómo puedes modificar tu configuración para lograrlo:

```
http {
    upstream backend_danieeeld2 {
        least_time header;
        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 {
        listen 80;
        server_name nginx_danieeeld2;
        access_log /var/log/nginx/nginx_danieeeld2.access.log;
        error_log /var/log/nginx/nginx_danieeeld2.error.log;
        location / {
            proxy_pass http://backend_danieeeld2;
            proxy_set_header Cookie $http_cookie;
            proxy_hide_header Set-Cookie;
        }
        location /estadisticas_danieeeld2 {
            stub_status on;
        }
    }
}
```

En esta configuración, he agregado la directiva `least_time header;` dentro del bloque `upstream backend_danieeeld2`. Esto le dice a Nginx que utilice el tiempo de respuesta del servidor como métrica para el balanceo de carga. El tiempo de respuesta se extrae del encabezado HTTP proporcionado por el servidor backend. Nginx seleccionará el servidor con el menor tiempo de respuesta para enviar la solicitud del cliente.

Análisis de la salida: Como vemos, el cambio que nos proporciona en la salida respecto al fichero original es bastante simple. Vamos a probarlo. Incluimos la nueva línea en nuestro fichero de configuración:

```
$ docker compose up
...
balanceador-nginx | /docker-entrypoint.sh: Launching
/docker-entrypoint.d/30-tune-worker-processes.sh
balanceador-nginx | /docker-entrypoint.sh: Configuration complete;
ready for start up
balanceador-nginx | 2024/04/13 21:58:10 [emerg] 1#1: unknown directive
"least_time" in /etc/nginx/nginx.conf:9
balanceador-nginx | nginx: [emerg] unknown directive "least_time"
in /etc/nginx/nginx.conf:9
balanceador-nginx exited with code 1
```

Vemos que falla. Tras investigar durante un buen rato y ver que las diferentes IAs no me daban información útil, encontré que `least_time` solo se puede usar en la versión de pago de Nginx, como se ve en Manual de Nginx, por lo que vamos a probar otro método de balanceo a cambio.

2.1.2. Balanceo ponderado

Vamos a comenzar preguntándole a ChatGPT:

- **Yo:** Tengo el siguiente fichero de configuración de nginx: (*Le paso el fichero de configuración de Nginx 1.2*) ¿Cómo puedo implementar una estrategia de balanceo basada en ponderación?
- **ChatGPT:** Para implementar una estrategia de balanceo ponderado en Nginx, puedes utilizar el módulo `nginx_http_upstream_module`, que proporciona la directiva `weight` para asignar pesos a los servidores en el grupo `upstream`. Aquí te muestro cómo modificar tu configuración para implementar el balanceo ponderado:

```
events {
    worker_connections 1024;
}

http {
    upstream backend_danieeld2 {
        # Algoritmo de balanceo
        # least_conn;
        server 192.168.10.2 weight=3;
        server 192.168.10.3 weight=2;
        server 192.168.10.4 weight=1;
        server 192.168.10.5 weight=1;
        server 192.168.10.6 weight=2;
        server 192.168.10.7 weight=3;
        server 192.168.10.8 weight=2;
        server 192.168.10.9 weight=1;
    }

    server {
        listen 80;
```

```

        server_name nginx_danieeeld2;
        access_log /var/log/nginx/nginx_danieeeld2.access.log;
        error_log /var/log/nginx/nginx_danieeeld2.error.log;
        location / {
            proxy_pass http://backend_danieeeld2;
            proxy_set_header Cookie $http_cookie;
            proxy_hide_header Set-Cookie;
        }
        location /estadisticas_danieeeld2 {
            stub_status on;
        }
    }
}

```

En esta configuración:

- Se han asignado pesos a cada uno de los servidores mediante la sintaxis weight=X, donde X es el peso asignado.
- Los servidores con pesos más altos recibirán una proporción mayor de solicitudes.
- Puedes ajustar los pesos según tus necesidades de balanceo y capacidad de servidor.

Análisis de la salida: Como vemos, una salida bastante simple, por lo que no tengo mucho que comentar. Vamos a probarla en nuestra configuración:

```

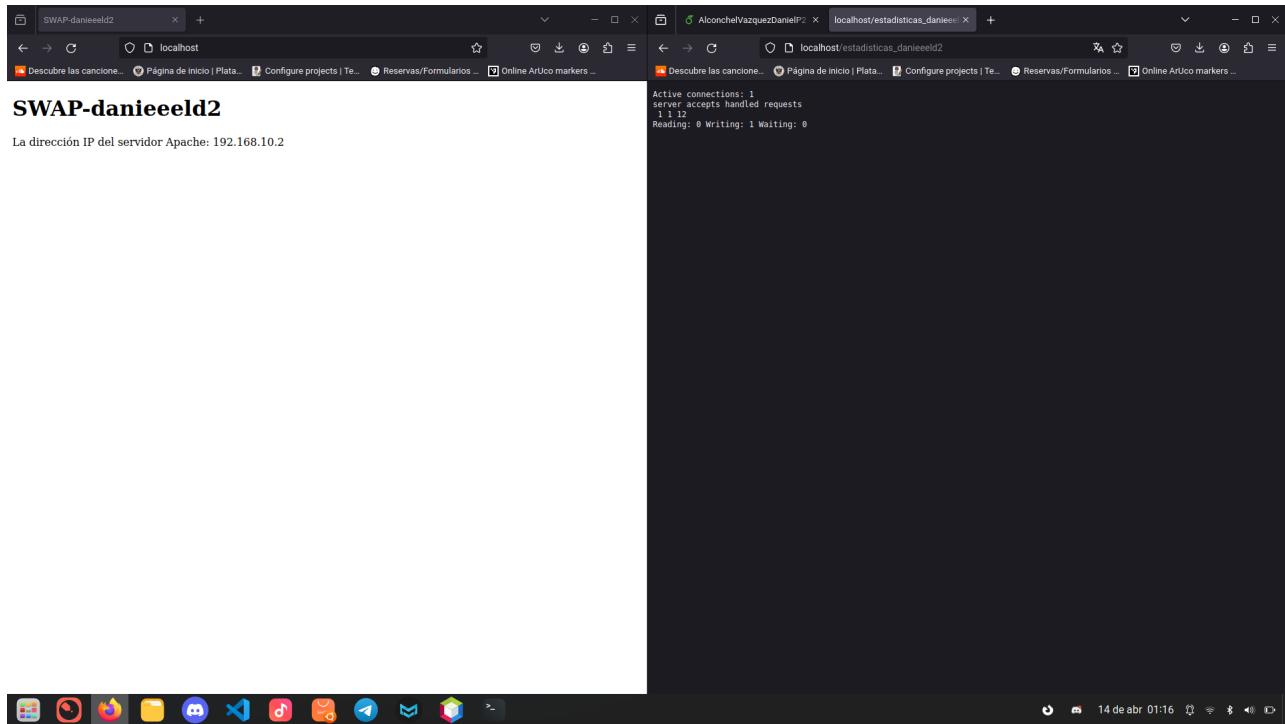
events {
    worker_connections 1024;
}

http {
    upstream backend_danieeeld2 {
        # Algoritmo de balanceo
        # least_conn;
        server 192.168.10.2 weight=10;
        server 192.168.10.3 weight=1;
        server 192.168.10.4 weight=1;
        server 192.168.10.5 weight=1;
        server 192.168.10.6 weight=3;
        server 192.168.10.7 weight=1;
        server 192.168.10.8 weight=1;
        server 192.168.10.9 weight=1;
    }

    server {
        listen 80;
        server_name nginx_danieeeld2;
        access_log /var/log/nginx/nginx_danieeeld2.access.log;
        error_log /var/log/nginx/nginx_danieeeld2.error.log;
        location / {
            proxy_pass http://backend_danieeeld2;
            proxy_set_header Cookie $http_cookie;
            proxy_hide_header Set-Cookie;
        }
        location /estadisticas_danieeeld2 {
            stub_status on;
        }
    }
}

```

Como vemos, he puesto un caso extremo de pesos para apreciar mejor el resultado.



Podemos ver que todo sigue funcionando correctamente. Recargando varias veces se aprecia como nos redirige con bastante frecuencia a la dirección 192.168.10.2.

Nota: Estuve buscando si podía incluir algunas estadísticas más avanzadas para ver la redirección, pero no lo conseguí, así que simplemente dejo comentado la experiencia.

2.2. Configuraciones Avanzadas de HAProxy

2.2.1. Balanceo basado en menor número de conexiones activas

Comenzamos preguntándole a ChatGPT (aunque es bastante simple este apartado):

- **Yo:** como configuro haproxy para que balancee por el menor numero de conexiones, teniendo en cuenta que mi fichero de configuración es el siguiente (*Le paso el fichero de configuración 1.5*).
- **ChatGPT:** Para configurar HAProxy para balancear por el menor número de conexiones, puedes utilizar el algoritmo de balanceo llamado "leastconn" (menor número de conexiones). Aquí tienes un ejemplo básico de cómo configurar HAProxy para usar este algoritmo:

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

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

frontend frontend_danieeed2
```

```

bind *:80
default_backend backend_danieeeld2

backend backend_danieeeld2
    balance leastconn
    server web1 192.168.10.2:80 maxconn 32 check
    server web2 192.168.10.3:80 maxconn 32 check
    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

listen stats
    bind *:9000
    mode http
    stats enable
    stats uri /estadisticas_danieeeld2
    stats realm HAProxy\ Statistics
    stats auth danieeeld2:SWAP1234

```

Como vemos, tampoco tiene mucho misterio. Al probarlo, si no establecemos previamente muchas conexiones a uno de los contenedores, parecerá un **round-robin**, pues solo nosotros estamos habilitando en cada regarga una conexión a uno de los servidores e irá escogiendo el siguiente, por lo que, para apreciar bien el resultado, es conveniente establecer muchas conexiones a unos cuantos contenedores previamente.

2.2.2. Balanceo ponderado

Comenzamos preguntándole a ChatGPT:

- **Yo:** Tengo el siguiente fichero de configuración de HAProxy, el cual uso como balanceador de carga: (*Le paso el fichero de configuración 1.5*) ¿Cómo puedo implementar una estrategia de balanceo basada en ponderación?
- **ChatGPT:** Para implementar una estrategia de balanceo basada en ponderación en HAProxy, puedes asignar un peso a cada servidor en el backend. Esto se hace utilizando la opción "weight" dentro de la definición de cada servidor en el backend.

Aquí te muestro cómo modificar tu configuración para implementar ponderación en el balanceo:

```

global
stats socket /var/lib/haproxy/stats

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

frontend frontend_danieeeld2

```

```

        bind *:80
        default_backend backend_danieeeld2

backend backend_danieeeld2
    balance roundrobin
    server web1 192.168.10.2:80 weight 1 maxconn 32
    server web2 192.168.10.3:80 weight 2 maxconn 32
    server web3 192.168.10.4:80 weight 3 maxconn 32
    server web4 192.168.10.5:80 weight 1 maxconn 32
    server web5 192.168.10.6:80 weight 2 maxconn 32
    server web6 192.168.10.7:80 weight 3 maxconn 32
    server web7 192.168.10.8:80 weight 1 maxconn 32
    server web8 192.168.10.9:80 weight 2 maxconn 32

listen stats
    bind *:9000
    mode http
    stats enable
    stats uri /estadisticas_danieeeld2
    stats realm HAProxy\ Statistics
    stats auth danieeeld2:SWAP1234

```

Análisis de la salida: Como vemos, la respuesta que nos proporciona es muy parecida a la del caso del balanceo ponderado con Nginx (2.1.2). Como es bastante simple no tengo mucho que comentar, así que vamos a ponerla en práctica:

```

global
stats socket /var/lib/haproxy/stats

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

frontend frontend_danieeeld2
    bind *:80
    default_backend backend_danieeeld2

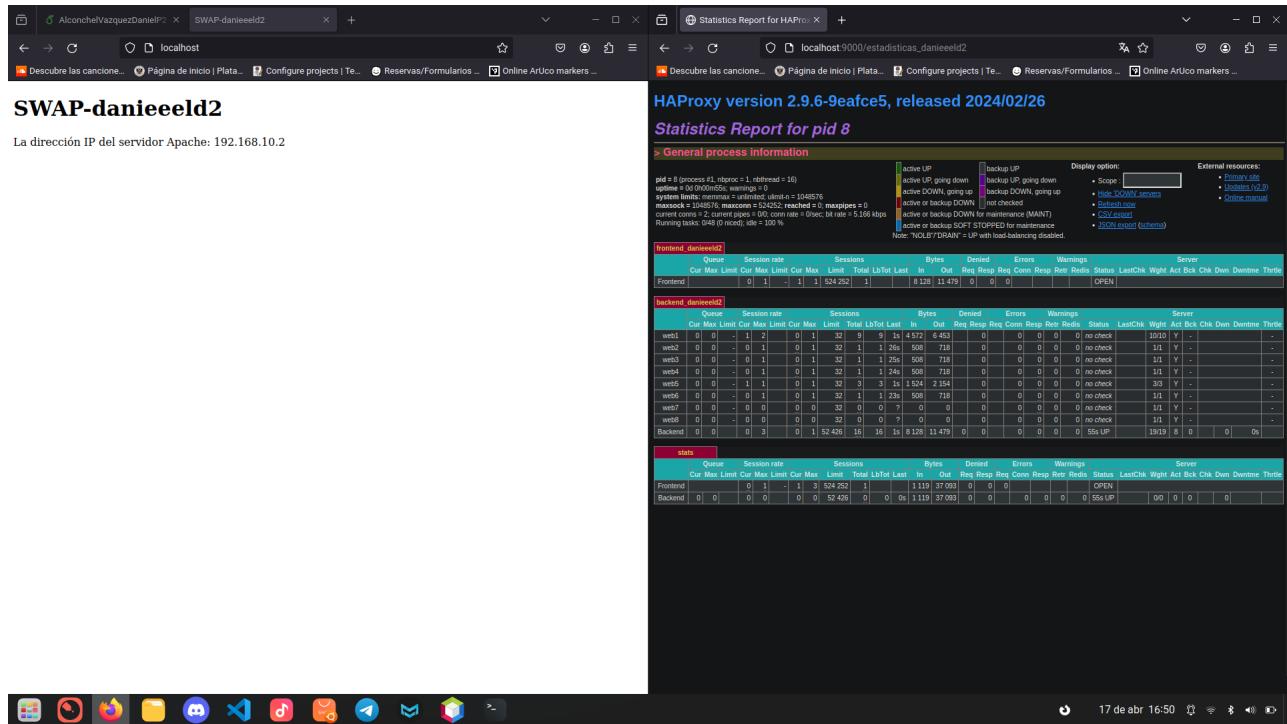
backend backend_danieeeld2
    balance roundrobin
    server web1 192.168.10.2:80 weight 10 maxconn 32
    server web2 192.168.10.3:80 weight 1 maxconn 32
    server web3 192.168.10.4:80 weight 1 maxconn 32
    server web4 192.168.10.5:80 weight 1 maxconn 32
    server web5 192.168.10.6:80 weight 3 maxconn 32
    server web6 192.168.10.7:80 weight 1 maxconn 32
    server web7 192.168.10.8:80 weight 1 maxconn 32
    server web8 192.168.10.9:80 weight 1 maxconn 32

listen stats
    bind *:9000
    mode http
    stats enable
    stats uri /estadisticas_danieeeld2
    stats realm HAProxy\ Statistics

```

```
stats auth danieeeld2:SWAP1234
```

Como vemos, he puesto un caso extremo de pesos para apreciar mejor el resultado.



Se puede apreciar claramente que, tras recargar múltiples veces la página, se redirige con mayor frecuencia a 192.168.10.2.

2.3. Experimentación con Diferentes Balanceadores de Carga

2.3.1. Gobetween

Vamos a comenzar preguntándole a ChatGPT:

- **Yo:** Tengo un escenario docker con 8 contenedores conectados a una misma red que suministran una página web y quiero configurar un nuevo contenedor que sea el balanceador de carga usando gobetween. El despliegue se realiza con docker-compose. Los servicios que suministran la pagina son de la siguiente forma: (*Le paso los servicios web1 y web2*).
- **ChatGPT:** Para configurar un nuevo contenedor como balanceador de carga utilizando GoBetween en tu escenario Docker, primero necesitas agregar un servicio adicional a tu archivo docker-compose.yml que sea responsable de ejecutar GoBetween. Luego, este servicio deberá estar conectado a la misma red que los otros contenedores para que pueda enrutar el tráfico correctamente.

Aquí tienes un ejemplo de cómo podrías modificar tu archivo docker-compose.yml para incluir el balanceador de carga con GoBetween:

```
version: '2.0'

services:
  gobetween:
    image: your-gobetween-image
    container_name: gobetween
```

```

    command: ["gobetween", "-c", "/etc/gobetween/gobetween.toml"]
    volumes:
      - ./gobetween-config:/etc/gobetween
    networks:
      - red_web
      - red_servicios

    # Tus servicios web existentes
    web1:
      image: danieeeld2-apache-image:p2
      container_name: web1
      privileged: true
      volumes:
        - ./web_danieeeld2:/var/www/html
      networks:
        red_web:
          ipv4_address: 192.168.10.2
        red_servicios:
          ipv4_address: 192.168.20.2

    web2:
      image: danieeeld2-apache-image:p2
      container_name: web2
      privileged: true
      volumes:
        - ./web_danieeeld2:/var/www/html
      networks:
        red_web:
          ipv4_address: 192.168.10.3
        red_servicios:
          ipv4_address: 192.168.20.3

    # Agrega más servicios web según sea necesario

    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

```

Análisis de la salida: Como vemos es una salida muy escueta que no aporta gran información, por lo que decidí preguntar directamente cómo crear el servicio de balanceo en docker-compose usando gobetween:

- **Yo:** quiero crear un servicio en docker-compose que se trate de un balanceador de carga con gobetween

- **ChatGPT:** Claro, puedo ayudarte con eso. Docker Compose es una herramienta excelente para definir y ejecutar aplicaciones Docker multi-contenedor. Vamos a crear un archivo docker-compose.yml para configurar un balanceador de carga utilizando GoBetween.

Primero, necesitarás tener Docker y Docker Compose instalados en tu sistema.

Aquí tienes un ejemplo básico de cómo podría ser tu archivo docker-compose.yml:

```
version: '3'

services:
  gobetween:
    image: "gobetween/gobetween"
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - ./gobetween.cfg:/etc/gobetween/gobetween.toml
    restart: always
```

Este archivo define un servicio llamado "gobetween" que utiliza la imagen oficial de GoBetween desde Docker Hub. Exponemos los puertos 80 y 443 para HTTP y HTTPS, respectivamente. Además, montamos un archivo de configuración gobetween.cfg en /etc/gobetween/gobetween.toml dentro del contenedor.

A continuación, necesitas crear el archivo de configuración gobetween.cfg. Aquí tienes un ejemplo básico:

```
# gobetween.cfg

[gobetween]
verbose = false

[api]
bind = "0.0.0.0:8888"

[[backend]]
url = "http://backend1:8080"

[[backend]]
url = "http://backend2:8080"

[[frontend]]
backend = ["backend1", "backend2"]
frontend = ":80,:443"
protocol = "tcp"
```

Análisis de la salida: He puesto una de las salidas que me daba, pero estuve bastante rato preguntándole a ChatGPT y Gemini cómo configurar esta herramienta, pero no me daban una respuesta que funcionase. Para empezar, la imagen que recomiendan usar no existe, mezclan formatos de ficheros de configuración, no realizan los ficheros correctamente, etc...

Finalmente, decidí ir por la solución que más me gusta a mí cuando se trata de configurar cosas, que es ir directamente a la documentación oficial de **gobetween**

Tras un rato leyendo la documentación, acabé realizando la siguiente solución:

```
# docker-compose.yml
# ... servicios web ...
balanceador-gobetween:
image: yyyar/gobetween:latest
container_name: balanceador-gobetween
ports:
- "80:80"
volumes:
- ./gobetween.toml:/etc/gobetween/conf/gobetween.toml
networks:
red_web:
  ipv4_address: 192.168.10.50
depends_on:
- web1
- web2
- web3
- web4
- web5
- web6
- web7
- web8
# ... creacion de redes ....
```

Nota: He modificado la script 1.3 para seguir generando los docker-compose de la misma manera.

```
#gobetween.toml

[servers.sample]
bind = "0.0.0.0:80"
protocol = "tcp"
balance = "roundrobin"

max_connections = 10000
client_idle_timeout = "10m"
backend_idle_timeout = "10m"
backend_connection_timeout = "20s"

[servers.sample.discovery]
kind = "static"
static_list = [
  "192.168.10.2:80",
  "192.168.10.3:80",
  "192.168.10.4:80",
  "192.168.10.5:80",
  "192.168.10.6:80",
  "192.168.10.7:80",
  "192.168.10.8:80",
  "192.168.10.9:80"
]

[servers.sample.healthcheck]
```

```

fails = 1
passes = 1
interval = "2s"
timeout="1s"
kind = "ping"
ping-timeout-duration = "500ms"

```

Vamos a probar el funcionamiento de la misma. (**Nota:** He bindeado la dirección 0.0.0.0:80³, ya que localhost me daba problemas.)

The screenshot shows two terminal windows side-by-side. The left terminal window displays log entries from a Docker container. The right terminal window shows the results of curl requests to three different Apache services running on ports 10.5, 10.6, and 10.8.

```

ain name, using 192.168.20.5. Set the 'ServerName' directive globally to suppress this message
balanceador-gobetween | 2024-04-18 20:58:47 [INFO ] (manager): Initializing...
balanceador-gobetween | 2024-04-18 20:58:47 [INFO ] (server): creating 'sample': 0.0.0.0:80 roundrobin
static ping
balanceador-gobetween | 2024-04-18 20:58:47 [INFO ] (scheduler): Starting scheduler sample
balanceador-gobetween | 2024-04-18 20:58:47 [INFO ] (manager): Initialized
balanceador-gobetween | 2024-04-18 20:58:47 [INFO ] (metrics): Metrics disabled
balanceador-gobetween | 2024-04-18 20:58:47 [INFO ] (api): API disabled
balanceador-gobetween | 2024-04-18 20:58:47 [INFO ] (healthcheck-worker): Sending to scheduler: [{192.1
68.10.2 80} 2]
balanceador-gobetween | 2024-04-18 20:58:47 [INFO ] (healthcheck-worker): Sending to scheduler: [{192.1
68.10.6 80} 2]
balanceador-gobetween | 2024-04-18 20:58:47 [INFO ] (healthcheck-worker): Sending to scheduler: [{192.1
68.10.5 80} 2]
balanceador-gobetween | 2024-04-18 20:58:47 [INFO ] (healthcheck-worker): Sending to scheduler: [{192.1
68.10.8 80} 2]
balanceador-gobetween | 2024-04-18 20:58:47 [INFO ] (healthcheck-worker): Sending to scheduler: [{192.1
68.10.9 80} 2]
balanceador-gobetween | 2024-04-18 20:58:48 [INFO ] (healthcheck-worker): Sending to scheduler: [{192.1
68.10.3 80} 1]
balanceador-gobetween | 2024-04-18 20:58:48 [INFO ] (healthcheck-worker): Sending to scheduler: [{192.1
68.10.7 80} 1]
balanceador-gobetween | 2024-04-18 20:51:10 [INFO ] (healthcheck-worker): Sending to scheduler: [{192.1
68.10.5 80} 1]
balanceador-gobetween | 2024-04-18 20:51:10 [INFO ] (healthcheck-worker): Sending to scheduler: [{192.1
68.10.9 80} 1]
balanceador-gobetween | 2024-04-18 20:51:10 [INFO ] (healthcheck-worker): Sending to scheduler: [{192.1
68.10.2 80} 1]
balanceador-gobetween | 2024-04-18 20:51:10 [INFO ] (healthcheck-worker): Sending to scheduler: [{192.1
68.10.8 80} 1]
balanceador-gobetween | 2024-04-18 20:51:10 [INFO ] (healthcheck-worker): Sending to scheduler: [{192.1
68.10.6 80} 1]
balanceador-gobetween | 2024-04-18 20:51:10 [INFO ] (healthcheck-worker): Sending to scheduler: [{192.1
68.10.3 80} 1]
balanceador-gobetween | 2024-04-18 20:51:11 [INFO ] (healthcheck-worker): Sending to scheduler: [{192.1
68.10.4 80} 2]
balanceador-gobetween | 2024-04-18 20:51:11 [INFO ] (healthcheck-worker): Sending to scheduler: [{192.1
68.10.2 80} 2]
balanceador-gobetween | 2024-04-18 20:51:11 [INFO ] (healthcheck-worker): Sending to scheduler: [{192.1
68.10.6 80} 2]
balanceador-gobetween | 2024-04-18 20:51:11 [INFO ] (healthcheck-worker): Sending to scheduler: [{192.1
68.10.3 80} 2]
balanceador-gobetween | 2024-04-18 20:51:11 [INFO ] (healthcheck-worker): Sending to scheduler: [{192.1
68.10.8 80} 2]
balanceador-gobetween | 2024-04-18 20:51:13 [INFO ] (healthcheck-worker): Sending to scheduler: [{192.1
68.10.9 80} 2]
balanceador-gobetween | 2024-04-18 20:51:13 [INFO ] (healthcheck-worker): Sending to scheduler: [{192.1
68.10.6 80} 2]

daniel.Modern-15-B7M: ~/.../P2/P2-danleeeld2-gobetween |main U:3 ?:1 X
→curl http://192.168.10.50:80/
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>SWAP-danleeeld2</title>
</head>
<body>
  <h1>SWAP-danleeeld2</h1>
  <p>La dirección IP del servidor Apache: 192.168.10.5</p>
</body>
</html>
daniel.Modern-15-B7M: ~/.../P2/P2-danleeeld2-gobetween |main U:3 ?:1 X
→curl http://192.168.10.50:80/
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>SWAP-danleeeld2</title>
</head>
<body>
  <h1>SWAP-danleeeld2</h1>
  <p>La dirección IP del servidor Apache: 192.168.10.6</p>
</body>
</html>
daniel.Modern-15-B7M: ~/.../P2/P2-danleeeld2-gobetween |main U:3 ?:1 X
→curl http://192.168.10.50:80/
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>SWAP-danleeeld2</title>
</head>
<body>
  <h1>SWAP-danleeeld2</h1>
  <p>La dirección IP del servidor Apache: 192.168.10.7</p>
</body>
</html>
daniel.Modern-15-B7M: ~/.../P2/P2-danleeeld2-gobetween |main U:3 ?:1 X
→curl http://192.168.10.50:80/
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>SWAP-danleeeld2</title>
</head>
<body>
  <h1>SWAP-danleeeld2</h1>
  <p>La dirección IP del servidor Apache: 192.168.10.8</p>
</body>
</html>

```

Podemos ver como se realiza tanto el **round-robin** como el **healthcheck** de los servicios.

Nota: Estuve un rato más intentando habilitar la API y métricas avanzadas, pero no lo conseguí, ya que la documentación es bastante escueta y las IAs no me proporcionaban información útil.

2.3.2. Traefik

Nota: En esta parte he usado bastante IA, aunque adelanto que no ha servido de mucho.

Vamos a comenzar preguntándole a ChatGPT:

- **Yo:** Tengo dos contenedores (*Simplifiqué un poco el escenario para que fuese más fácil para la IA*), creados con docker-compose que suministran una misma página web. Quiero crear un nuevo servicio que sirva como balanceador de carga, usando traefik, conectado a la misma red que los dos servicios anteriores, con IP 192.168.10.50.

- **ChatGPT:**

1. Crear un archivo de configuración Traefik: Crea un archivo de configuración `traefik.toml` que contenga la configuración de Traefik. Aquí hay un ejemplo básico:

³Esta dirección hace referencia a todas las interfaces de red disponibles con puerto de escucha 80

```

# traefik.toml
[entryPoints]
[entryPoints.web]
    address = ":80"

[api]
    dashboard = true

2. Crear un archivo Docker Compose: Crea un archivo docker-compose.yml para definir tus servicios y configurar Traefik como un servicio adicional:

version: '3.8'

services:
    traefik:
        image: traefik:v2.6
        container_name: traefik
        command:
            - "--api.insecure=true"
            - "--providers.docker=true"
            - "--entrypoints.web.address=:80"
            - "--log.level=DEBUG"
            - "--providers.docker.exposedbydefault=false"
        volumes:
            - /var/run/docker.sock:/var/run/docker.sock
            - ./traefik.toml:/traefik.toml
        ports:
            - "80:80"
            - "8080:8080"

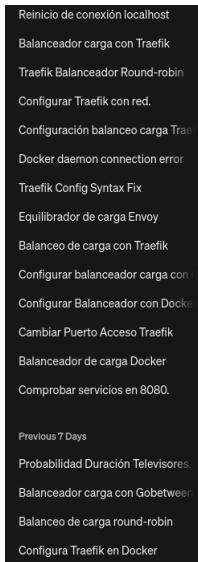
    webapp1:
        image: your_webapp_image1
        labels:
            - "traefik.enable=true"
            - "traefik.http.routers.webapp1.rule=
                Host('webapp1.example.com')"

    webapp2:
        image: your_webapp_image2
        labels:
            - "traefik.enable=true"
            - "traefik.http.routers.webapp2.rule=
                Host('webapp2.example.com')"

```

Análisis de la salida: En una primera instancia, la salida no me parecía mala, pero me daba fallos (*Obviamente, cambiando los parámetros necesarios por los de mi práctica*) y en los logs no me indicaba nada, por lo que no tenía mucha idea de por donde tirar (*El principal error que obtenía era que me salía 404 page not found, es decir, no sabe que página suministrar. Tampoco conseguí sacar las estadísticas*). Decidí entonces mirar la documentación oficial de la imagen (ver aquí) y vemos que, para empezar, los

ficheros de configuración que usa son distintos. Las versiones pertenecientes a las 2.x usan .yml, mientras que las derivadas de la 1.7 usan .toml. Posiblemente, esto se deba a que la base de datos que usa ChatGPT-3.5 es del 2022 y se haya quedado desactualizada.



Nota: *Le estuve bastante rato preguntando, pero tengo demasiadas conversaciones con la IA como para incluir todas, así que me quedo con la más significativa y aporto una captura para ver todas las que tuve.*

Como ChatGPT-3.5 aparentemente no nos proporciona información de gran calidad, decidí preguntarle a otra inteligencia artificial más moderada, como Gemini.

Vamos a preguntarle a Gemini (*Nuevamente, tuve bastantes conversaciones con la IAs así que voy a poner la más significativa*):

- **Yo:** Tengo estos dos servicios de docker-compose que se encargan de servir una misma página web. Quiero crear un nuevo servicio de traefik que se encargue de balancear la carga entre los dos servicios web, usando round-robin. Para ello, el servicio de traefik está conectado a la misma red con ip 192.168.10.50.
- **Gemini:** Para balancear la carga entre tus dos servicios web "web1" y "web2" utilizando Traefik en Docker Compose, puedes seguir estos pasos:

1. Crear un archivo docker-compose.yml:

```
version: '2.0'

services:
  # ... (Tus servicios web1 y web2 existentes)

  traefik:
    image: traefik:latest
    container_name: traefik
    restart: always
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - ./traefik.yml:/etc/traefik/traefik.yml
    networks:
      - red_web
  depends_on:
    - web1
```

- web2

2. Crear el archivo de configuración de Traefik (traefik.yml):

```
# traefik.yml

static_files:
  - path: /web
    directory: ./web_danieeeld2

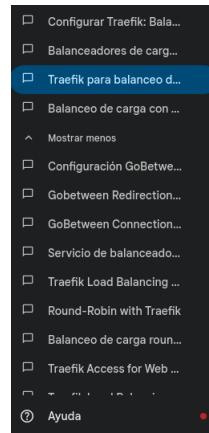
providers:
  file:
    source: /etc/traefik/dynamic.yml

routers:
  web:
    entryPoints:
      - http
      - https
    middlewares:
      - default-headers
      - add-host
    tls:
      options:
        - certsDir: /etc/traefik/certs
    service:
      loadbalancer:
        servers:
          - address: 192.168.10.2:80
          - address: 192.168.10.3:80
        sticky:
          cookieName: my-app-cookie
        cookie:
          maxAge: 3600
        condition:
          request.headers.Host:
            - "web1.example.com"
            - "web2.example.com"
```

Después me daba una explicación muy extensa de como funciona cada parámetro, etc...

Análisis de la salida: Esta salida, al menos, utiliza ya la extensión del fichero de configuración correcto. El problema de la IA en estos ejercicios de configuración tan específica es que te saca una configuración genérica (muchas veces desactualizada) y no tienes una gran explicación de lo que está ocurriendo. Pese a que Traefik no es una herramienta excesivamente compleja, si que utiliza muchos conceptos en su configuración, que cuentan con pequeños detalles de configuración, como `entrypoints`, `routers`, `services`, `providers`... Además, dependiendo de la versión la configuración cambia bastante o existen múltiples posibilidades, como configurarlo a través de `labels` en `docker-compose` o mediante el fichero de configuración, etc...

Conclusión, al final para este tipo de cuestiones, como son configuraciones específicas de una herramienta muy específica, lo mejor es seguir los tutoriales y documentación que proveen los creadores del servicio. Para este caso, acabé usando la documentación oficial de **Traefik**, que es bastante extensa y me llevó un buen rato de leer y entender.



Finalmente, plantee la siguiente solución final:

```
# docker-compose.yml

# ... creacion de servicio web ...
balanceador-traefik:
  image: traefik:latest
  container_name: balanceador-traefik
  ports:
    - "80:80"
    - "8080:8080"
  volumes:
    - ./traefik.yml:/etc/traefik/traefik.yml
    - ./dynamic.yml:/etc/traefik/conf.d/dynamic.yml
    - /var/run/docker.sock:/var/run/docker.sock
  networks:
    red_web:
      ipv4_address: 192.168.10.50
  depends_on:
    - web1
    - web2
    - web3
    - web4
    - web5
    - web6
    - web7
    - web8
# ... creacion de redes ...
```

Nota: He modificado la script 1.3 para seguir generando los docker-compose de la misma manera.

```
# traefik.yml

# API and dashboard configuration
api:
  insecure: true

# Entry points configuration
entryPoints:
  web:
    address: ":80"
```

```

providers:
  file:
    directory: /etc/traefik/conf.d

http:
  routers:
    to-app:
      rule: "Path('')"
      service: app

  services:
    app:
      weighted:
        services:
          - name: web1
            weight: 1
          - name: web2
            weight: 1
          - name: web3
            weight: 1
          - name: web4
            weight: 1
          - name: web5
            weight: 1
          - name: web6
            weight: 1
          - name: web7
            weight: 1
          - name: web8
            weight: 1

    web1:
      loadBalancer:
        servers:
          - url: "http://192.168.10.2:80/"

    web2:
      loadBalancer:
        servers:
          - url: "http://192.168.10.3:80/"

    web3:
      loadBalancer:
        servers:
          - url: "http://192.168.10.4:80/"

    web4:
      loadBalancer:
        servers:
          - url: "http://192.168.10.5:80/"

    web5:
      loadBalancer:
        servers:
          - url: "http://192.168.10.6:80/"

    web6:
      loadBalancer:
        servers:
          - url: "http://192.168.10.7:80/"

```

```

web7:
  loadBalancer:
    servers:
      - url: "http://192.168.10.8:80/"

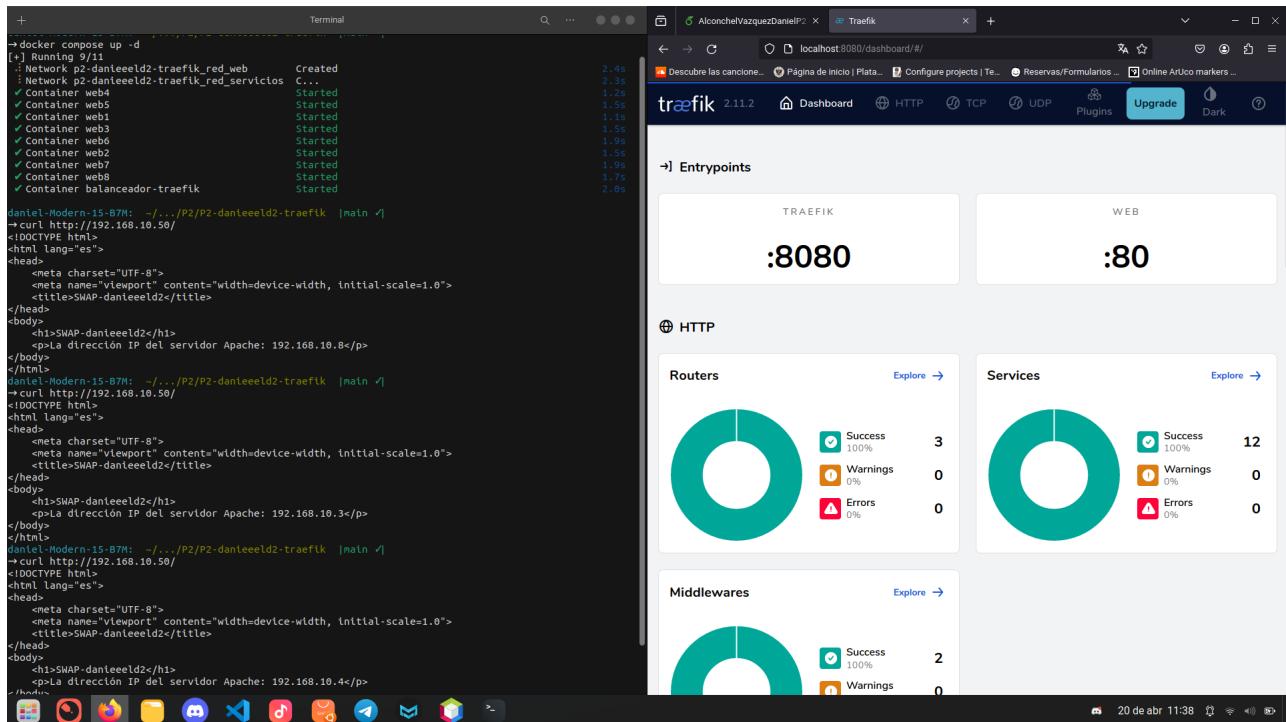
```

```

web8:
  loadBalancer:
    servers:
      - url: "http://192.168.10.9:80/"

```

Podemos comprobar que todo funciona correctamente:



2.4. Investigación y Pruebas de Tolerancia a Fallos

Para esta sección, en cada herramienta que hemos configurado, vamos a desconectar algunos de los contenedores de la `red_web` para comprobar la tolerancia a fallos.

2.4.1. Nginx

Vamos a desconectar la mitad de los contenedores:

```

daniel-Modern-15-B7M: ~/.../P2/P2-daneeeld2-nginx [main ✓]
→ docker compose up -d
[+] Running 9/11
  . Network p2-daneeeld2-nginx_red_servicios Created
  . Network p2-daneeeld2-nginx_red_web Created
  ✓ Container web3 Started
  ✓ Container web8 Started
  ✓ Container web7 Started
  ✓ Container web5 Started
  ✓ Container web2 Started
  ✓ Container web1 Started
  ✓ Container web6 Started
  ✓ Container balanceador-nginx Started
daniel-Modern-15-B7M: ~/.../P2/P2-daneeeld2-nginx [main ✓]
→ docker network disconnect p2-daneeeld2-nginx_red_web web3
daniel-Modern-15-B7M: ~/.../P2/P2-daneeeld2-nginx [main ✓]
→ docker network disconnect p2-daneeeld2-nginx_red_web web8
daniel-Modern-15-B7M: ~/.../P2/P2-daneeeld2-nginx [main ✓]
→ docker network disconnect p2-daneeeld2-nginx_red_web web7
daniel-Modern-15-B7M: ~/.../P2/P2-daneeeld2-nginx [main ✓]
→ docker network disconnect p2-daneeeld2-nginx_red_web web5
daniel-Modern-15-B7M: ~/.../P2/P2-daneeeld2-nginx [main ✓]
→ docker network disconnect p2-daneeeld2-nginx_red_web web2
daniel-Modern-15-B7M: ~/.../P2/P2-daneeeld2-nginx [main ✓]
→ curl http://192.168.10.50/
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>SWAP-daneeeld2</title>
</head>
<body>
  <h1>SWAP-daneeeld2</h1>
  <p>La dirección IP del servidor Apache: 192.168.10.5</p>
</body>
</html>
daniel-Modern-15-B7M: ~/.../P2/P2-daneeeld2-nginx [main ✓]
→ curl http://192.168.10.50/
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>SWAP-daneeeld2</title>
</head>
<body>
  <h1>SWAP-daneeeld2</h1>
  <p>La dirección IP del servidor Apache: 192.168.10.2</p>
</body>
</html>
daniel-Modern-15-B7M: ~/.../P2/P2-daneeeld2-nginx [main ✓]
→ curl http://192.168.10.50/
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>SWAP-daneeeld2</title>
</head>
<body>
  <h1>SWAP-daneeeld2</h1>
  <p>La dirección IP del servidor Apache: 192.168.10.6</p>
</body>
</html>
daniel-Modern-15-B7M: ~/.../P2/P2-daneeeld2-nginx [main ✓]
→ curl http://192.168.10.50/
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>SWAP-daneeeld2</title>
</head>
<body>
  <h1>SWAP-daneeeld2</h1>
  <p>La dirección IP del servidor Apache: 192.168.10.2</p>
</body>
</html>
daniel-Modern-15-B7M: ~/.../P2/P2-daneeeld2-nginx [main ✓]
→ curl http://192.168.10.50/
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>SWAP-daneeeld2</title>
</head>
<body>
  <h1>SWAP-daneeeld2</h1>
  <p>La dirección IP del servidor Apache: 192.168.10.7</p>
</body>
</html>
real    0m18.42s
user    0m0.004s
sys     0m0.006s
daniel-Modern-15-B7M: ~/Imágenes/Capturas de pantalla
→

```

Vemos que sigue funcionando correctamente, aunque hay veces que se satura un poco si recargo muy rápido. Vamos a probar ahora a dejar un único contenedor:

```

daniel-Modern-15-B7M: ~/.../P2/P2-daneeeld2-nginx [main ✓]
→ docker compose up -d
[+] Running 9/11
  . Network p2-daneeeld2-nginx_red_servicios Created
  . Network p2-daneeeld2-nginx_red_web Created
  ✓ Container web3 Started
  ✓ Container web8 Started
  ✓ Container web7 Started
  ✓ Container web5 Started
  ✓ Container web2 Started
  ✓ Container web1 Started
  ✓ Container web6 Started
  ✓ Container balanceador-nginx Started
daniel-Modern-15-B7M: ~/.../P2/P2-daneeeld2-nginx [main ✓]
→ docker network disconnect p2-daneeeld2-nginx_red_web web3
daniel-Modern-15-B7M: ~/.../P2/P2-daneeeld2-nginx [main ✓]
→ docker network disconnect p2-daneeeld2-nginx_red_web web8
daniel-Modern-15-B7M: ~/.../P2/P2-daneeeld2-nginx [main ✓]
→ docker network disconnect p2-daneeeld2-nginx_red_web web7
daniel-Modern-15-B7M: ~/.../P2/P2-daneeeld2-nginx [main ✓]
→ docker network disconnect p2-daneeeld2-nginx_red_web web5
daniel-Modern-15-B7M: ~/.../P2/P2-daneeeld2-nginx [main ✓]
→ docker network disconnect p2-daneeeld2-nginx_red_web web2
daniel-Modern-15-B7M: ~/.../P2/P2-daneeeld2-nginx [main ✓]
→ curl http://192.168.10.50/
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>SWAP-daneeeld2</title>
</head>
<body>
  <h1>SWAP-daneeeld2</h1>
  <p>La dirección IP del servidor Apache: 192.168.10.7</p>
</body>
</html>
real    0m18.42s
user    0m0.004s
sys     0m0.006s
daniel-Modern-15-B7M: ~/Imágenes/Capturas de pantalla
→

```

Podemos ver que sigue funcionando, pero el tiempo de respuesta se vuelve mucho más alto. Evidentemente, si desconectamos todos recibimos un 502 Bad Gateway, porque no sabe a donde redireccionar y, por otro lado, si vamos conectando de nuevo los contenedores, irá funcionando correctamente y cada vez el tiempo de respuesta será menor, ya que cada contenedor estará menos saturado.

2.4.2. HAProxy

Vamos a comenzar desconectando la mitad de los contenidos.

The screenshot shows a Linux desktop environment with a terminal window and a browser window. The terminal window displays system monitoring information for the HAProxy server. The browser window shows the HAProxy Statistics Report for pid 8, which includes detailed information about the frontends, backends, and their performance metrics.

Esta vez podemos verlo de una forma gráfica, gracias a las estadísticas avanzadas que ofrece HAProxy. Vemos también como sigue funcionando perfectamente pese a desconectar la mitad de contenidos. Vamos a probar ahora a dejar solo un contenedor:

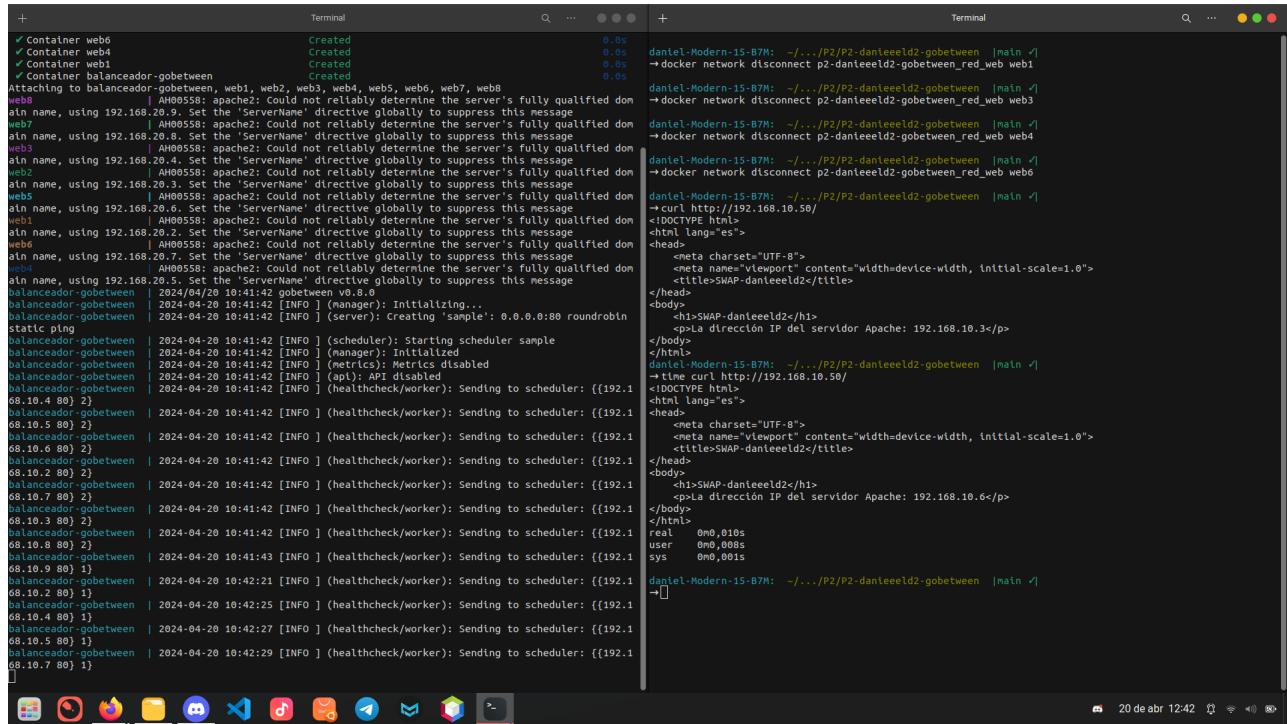
This screenshot shows the same Linux desktop setup as the previous one, but with a modified HAProxy configuration. Only one backend, 'web1', is active. The HAProxy statistics report reflects this change, showing the single frontend 'danieeeld2' handling requests to the single backend 'web1'. The terminal window still displays system monitoring data.

Podemos apreciar que la eficiencia a la hora de redirigir la carga y las peticiones de HAProxy ante fallos (como que se caigan "los servicios") es muy superior a la de Nginx, solo hay que ver los tiempos de respuesta

de curl. Por último comentar que, al igual que Nginx, si vamos reconectadno los contenedores podremos ver en las estadísticas como se van coloreando de verde y HAProxy los volverá a considerar para balancear la carga.

2.4.3. Gobetween

Como siempre, vamos a comenzar desconectando 4 contenedores:



The screenshot shows two terminal windows side-by-side. The left terminal displays the Docker logs for the 'balanceador-gobetween' container. It shows several log entries from Apache2 and the Gobetween scheduler. Apache2 logs mention 'ServerName' directives and connection attempts from various IP addresses (192.168.10.2, 192.168.10.3, 192.168.10.4, 192.168.10.5, 192.168.10.6, 192.168.10.7). The Gobetween scheduler logs show it sending healthcheck requests to these servers. The right terminal shows the curl command being run against the Apache2 service at 192.168.10.5, which returns a standard HTML response page.

```

+-----+-----+
| Terminal | Terminal |
+-----+-----+
[Container web6]   [Created]          0.05
[Container web4]   [Created]          0.05
[Container web1]   [Created]          0.05
[Container balanceador-gobetween] [Created]          0.05
Attaching to balanceador-gobetween, web1, web2, web3, web4, webs, web6, web7, web8
web8  | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 192.168.20.8. Set the 'ServerName' directive globally to suppress this message
web7  | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 192.168.20.8. Set the 'ServerName' directive globally to suppress this message
web3  | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 192.168.20.8. Set the 'ServerName' directive globally to suppress this message
web1  | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 192.168.20.8. Set the 'ServerName' directive globally to suppress this message
webs  | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 192.168.20.8. Set the 'ServerName' directive globally to suppress this message
web6  | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 192.168.20.8. Set the 'ServerName' directive globally to suppress this message
web5  | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 192.168.20.8. Set the 'ServerName' directive globally to suppress this message
balanceador-gobetween | 2024/04/20 10:41:42 gobetween v0.8.0
balanceador-gobetween | 2024-04-20 10:41:42 [INFO ] (manager): Initializing...
balanceador-gobetween | 2024-04-20 10:41:42 [INFO ] (server): Creating 'sample': 0.0.0.0:80 roundrobin
static ping
balanceador-gobetween | 2024-04-20 10:41:42 [INFO ] (scheduler): Starting scheduler sample
balanceador-gobetween | 2024-04-20 10:41:42 [INFO ] (manager): Initialized
balanceador-gobetween | 2024-04-20 10:41:42 [INFO ] (metrics): Metrics disabled
balanceador-gobetween | 2024-04-20 10:41:42 [INFO ] (api): API disabled
balanceador-gobetween | 2024-04-20 10:41:42 [INFO ] (healthcheck): Sending to scheduler: [{192.168.10.4 80} 2]
balanceador-gobetween | 2024-04-20 10:41:42 [INFO ] (healthcheck): Sending to scheduler: [{192.168.10.5 80} 2]
balanceador-gobetween | 2024-04-20 10:41:42 [INFO ] (healthcheck): Sending to scheduler: [{192.168.10.6 80} 2]
balanceador-gobetween | 2024-04-20 10:41:42 [INFO ] (healthcheck): Sending to scheduler: [{192.168.10.7 80} 1]
balanceador-gobetween | 2024-04-20 10:42:21 [INFO ] (healthcheck): Sending to scheduler: [{192.168.10.2 80} 1]
balanceador-gobetween | 2024-04-20 10:42:25 [INFO ] (healthcheck): Sending to scheduler: [{192.168.10.4 80} 1]
balanceador-gobetween | 2024-04-20 10:42:27 [INFO ] (healthcheck): Sending to scheduler: [{192.168.10.5 80} 1]
balanceador-gobetween | 2024-04-20 10:42:29 [INFO ] (healthcheck): Sending to scheduler: [{192.168.10.7 80} 1]
]

daniel-Modern-15-B7M: ~/.../P2/P2-daneeeld2-gobetween [main ↵
→ docker network disconnect p2-daneeeld2-gobetween_red_web web1
daniel-Modern-15-B7M: ~/.../P2/P2-daneeeld2-gobetween [main ↵
→ docker network disconnect p2-daneeeld2-gobetween_red_web web3
daniel-Modern-15-B7M: ~/.../P2/P2-daneeeld2-gobetween [main ↵
→ docker network disconnect p2-daneeeld2-gobetween_red_web web4
daniel-Modern-15-B7M: ~/.../P2/P2-daneeeld2-gobetween [main ↵
→ docker network disconnect p2-daneeeld2-gobetween_red_web web6
daniel-Modern-15-B7M: ~/.../P2/P2-daneeeld2-gobetween [main ↵
→ curl http://192.168.10.50/
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>SWAP-daneeeld2</title>
</head>
<body>
  <h1>SWAP-daneeeld2</h1>
  <p>La dirección IP del servidor Apache: 192.168.10.3</p>
</body>
</html>
daniel-Modern-15-B7M: ~/.../P2/P2-daneeeld2-gobetween [main ↵
→ time curl http://192.168.10.50/
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>SWAP-daneeeld2</title>
</head>
<body>
  <h1>SWAP-daneeeld2</h1>
  <p>La dirección IP del servidor Apache: 192.168.10.6</p>
</body>
</html>
real    0m0.010s
user    0m0.008s
sys     0m0.001s
daniel-Modern-15-B7M: ~/.../P2/P2-daneeeld2-gobetween [main ↵
]
```

Podemos ver que sigue funcionando correctamente. Además, en los logs de la parte izquierda podemos apreciar como se va realizando el **healthcheck** de los servicios. Vamos ahora a dejar solo un contenedor activo:

```

+ Terminal
68.10.9 80} 2}
balanceador-gobetween | 2024-04-20 10:46:28 [INFO ] (healthcheck-worker): Sending to scheduler: [{192.1
68.10.4 80} 2}
balanceador-gobetween | 2024-04-20 10:46:28 [INFO ] (healthcheck-worker): Sending to scheduler: [{192.1
68.10.8 80} 2}
balanceador-gobetween | 2024-04-20 10:46:29 [INFO ] (healthcheck-worker): Sending to scheduler: [{192.1
68.10.6 80} 1}
balanceador-gobetween | 2024-04-20 10:46:29 [INFO ] (healthcheck-worker): Sending to scheduler: [{192.1
68.10.2 80} 1}
balanceador-gobetween | 2024-04-20 10:46:33 [INFO ] (healthcheck-worker): Sending to scheduler: [{192.1
68.10.7 80} 1}
balanceador-gobetween | 2024-04-20 10:46:33 [INFO ] (healthcheck-worker): Sending to scheduler: [{192.1
68.10.8 80} 1}
balanceador-gobetween | 2024-04-20 10:46:33 [INFO ] (healthcheck-worker): Sending to scheduler: [{192.1
68.10.4 80} 1]
balanceador-gobetween | 2024-04-20 10:46:33 [INFO ] (healthcheck-worker): Sending to scheduler: [{192.1
68.10.8 80} 1]
balanceador-gobetween | 2024-04-20 10:46:33 [INFO ] (healthcheck-worker): Sending to scheduler: [{192.1
68.10.3 80} 1]
balanceador-gobetween | 2024-04-20 10:46:34 [INFO ] (healthcheck-worker): Sending to scheduler: [{192.1
68.10.5 80} 1]
balanceador-gobetween | 2024-04-20 10:46:35 [INFO ] (healthcheck-worker): Sending to scheduler: [{192.1
68.10.9 80} 1]
balanceador-gobetween | 2024-04-20 10:46:36 [INFO ] (healthcheck-worker): Sending to scheduler: [{192.1
68.10.9 80} 2]
balanceador-gobetween | 2024-04-20 10:46:36 [INFO ] (healthcheck-worker): Sending to scheduler: [{192.1
68.10.8 80} 2]
balanceador-gobetween | 2024-04-20 10:46:36 [INFO ] (healthcheck-worker): Sending to scheduler: [{192.1
68.10.5 80} 2]
balanceador-gobetween | 2024-04-20 10:46:36 [INFO ] (healthcheck-worker): Sending to scheduler: [{192.1
68.10.6 80} 2]
balanceador-gobetween | 2024-04-20 10:46:37 [INFO ] (healthcheck-worker): Sending to scheduler: [{192.1
68.10.2 80} 1]
balanceador-gobetween | 2024-04-20 10:46:36 [INFO ] (healthcheck-worker): Sending to scheduler: [{192.1
68.10.7 80} 2]
balanceador-gobetween | 2024-04-20 10:46:36 [INFO ] (healthcheck-worker): Sending to scheduler: [{192.1
68.10.4 80} 2]
balanceador-gobetween | 2024-04-20 10:46:36 [INFO ] (healthcheck-worker): Sending to scheduler: [{192.1
68.10.5 80} 1]
balanceador-gobetween | 2024-04-20 10:46:51 [INFO ] (healthcheck-worker): Sending to scheduler: [{192.1
68.10.4 80} 1]
balanceador-gobetween | 2024-04-20 10:46:53 [INFO ] (healthcheck-worker): Sending to scheduler: [{192.1
68.10.5 80} 1]
balanceador-gobetween | 2024-04-20 10:46:55 [INFO ] (healthcheck-worker): Sending to scheduler: [{192.1
68.10.7 80} 1]
balanceador-gobetween | 2024-04-20 10:46:57 [INFO ] (healthcheck-worker): Sending to scheduler: [{192.1
68.10.8 80} 1]
balanceador-gobetween | 2024-04-20 10:46:59 [INFO ] (healthcheck-worker): Sending to scheduler: [{192.1
68.10.9 80} 1}

+ Terminal
daniel-Modern-15-B7M: ~.../P2/P2-danleeeld2-gobetween [main ↵]
→ docker network disconnect p2-danleeeld2-gobetween_red_web web1
daniel-Modern-15-B7M: ~.../P2/P2-danleeeld2-gobetween [main ↵]
→ docker network disconnect p2-danleeeld2-gobetween_red_web web2
daniel-Modern-15-B7M: ~.../P2/P2-danleeeld2-gobetween [main ↵]
→ docker network disconnect p2-danleeeld2-gobetween_red_web web3
daniel-Modern-15-B7M: ~.../P2/P2-danleeeld2-gobetween [main ↵]
→ docker network disconnect p2-danleeeld2-gobetween_red_web web4
daniel-Modern-15-B7M: ~.../P2/P2-danleeeld2-gobetween [main ↵]
→ docker network disconnect p2-danleeeld2-gobetween_red_web web5
daniel-Modern-15-B7M: ~.../P2/P2-danleeeld2-gobetween [main ↵]
→ curl http://192.168.10.50/
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>SWAP-danleeeld2</title>
</head>
<body>
    <h1>SWAP-danleeeld2</h1>
    <p>La dirección IP del servidor Apache: 192.168.10.6</p>
</body>
</html>
real    0m0.018s
user   0m0.003s
sys    0m0.005s

daniel-Modern-15-B7M: ~.../P2/P2-danleeeld2-gobetween [main ↵]
→ curl http://192.168.10.50/
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>SWAP-danleeeld2</title>
</head>
<body>
    <h1>SWAP-danleeeld2</h1>
    <p>La dirección IP del servidor Apache: 192.168.10.6</p>
</body>
</html>
real    0m0.009s

```

Vemos que sigue funcionando muy bien, mostrando una buena tolerancia a fallos. Como siempre, si vamos reconectando los contenidos, irá teniéndolo en cuenta para el balanceo de la carga gracias al **healthcheck** que tenemos habilitado.

2.4.4. Traefik

Vamos ya con la última herramienta y vamos a comenzar desconectando la mitad de los contenidos:

```

+ Terminal
daniel-Modern-15-B7M: ~.../P2/P2-danleeeld2-traefik [main ↵]
→ docker compose up -d
[+] Running 9/11
  Network p2-danleeeld2-traefik_red_web     Created
  Network p2-danleeeld2-traefik_red_servicios  C...
  Container web3                            Started
  Container web4                            Started
  Container web5                            Started
  Container web7                            Started
  Container web1                            Started
  Container web3                            Started
  Container web8                            Started
  Container web6                            Started
  Container balanceador-traefik            Started
daniel-Modern-15-B7M: ~.../P2/P2-danleeeld2-traefik [main ↵]
→ docker
docker                  dockerd           docker-index
docker-compose          dockerd-rootless-setuptool.sh docker-proxy
docker-credential-desktop dockerd-rootless.sh
daniel-Modern-15-B7M: ~.../P2/P2-danleeeld2-traefik [main ↵]
→ docker n
network node
daniel-Modern-15-B7M: ~.../P2/P2-danleeeld2-traefik [main ↵]
→ docker network disconnect p2-danleeeld2-traefik_red_servicios p2-danleeeld2-traefik_red_web
daniel-Modern-15-B7M: ~.../P2/P2-danleeeld2-traefik [main ↵]
→ docker network disconnect p2-danleeeld2-traefik_red_web web1
daniel-Modern-15-B7M: ~.../P2/P2-danleeeld2-traefik [main ↵]
→ docker network disconnect p2-danleeeld2-traefik_red_web web3
daniel-Modern-15-B7M: ~.../P2/P2-danleeeld2-traefik [main ↵]
→ docker network disconnect p2-danleeeld2-traefik_red_web web5
daniel-Modern-15-B7M: ~.../P2/P2-danleeeld2-traefik [main ↵]
→ docker network disconnect p2-danleeeld2-traefik_red_web web7
daniel-Modern-15-B7M: ~.../P2/P2-danleeeld2-traefik [main ↵]
→ curl http://192.168.10.50/
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>SWAP-danleeeld2</title>
</head>
<body>
    <h1>SWAP-danleeeld2</h1>
    <p>La dirección IP del servidor Apache: 192.168.10.9</p>
</body>
</html>
real    0m0.011s
user   0m0.003s
sys    0m0.006s

daniel-Modern-15-B7M: ~.../P2/P2-danleeeld2-traefik [main ↵]
→ curl http://192.168.10.50/
Bad Gateway
real    0m3.065s
user   0m0.005s
sys    0m0.009s

daniel-Modern-15-B7M: ~.../P2/P2-danleeeld2-traefik [main ↵]
→ curl http://192.168.10.50/
Bad Gateway
real    0m3.065s
user   0m0.005s
sys    0m0.009s

daniel-Modern-15-B7M: ~.../P2/P2-danleeeld2-traefik [main ↵]
→ curl http://192.168.10.50/
Bad Gateway
real    0m3.065s
user   0m0.005s
sys    0m0.009s

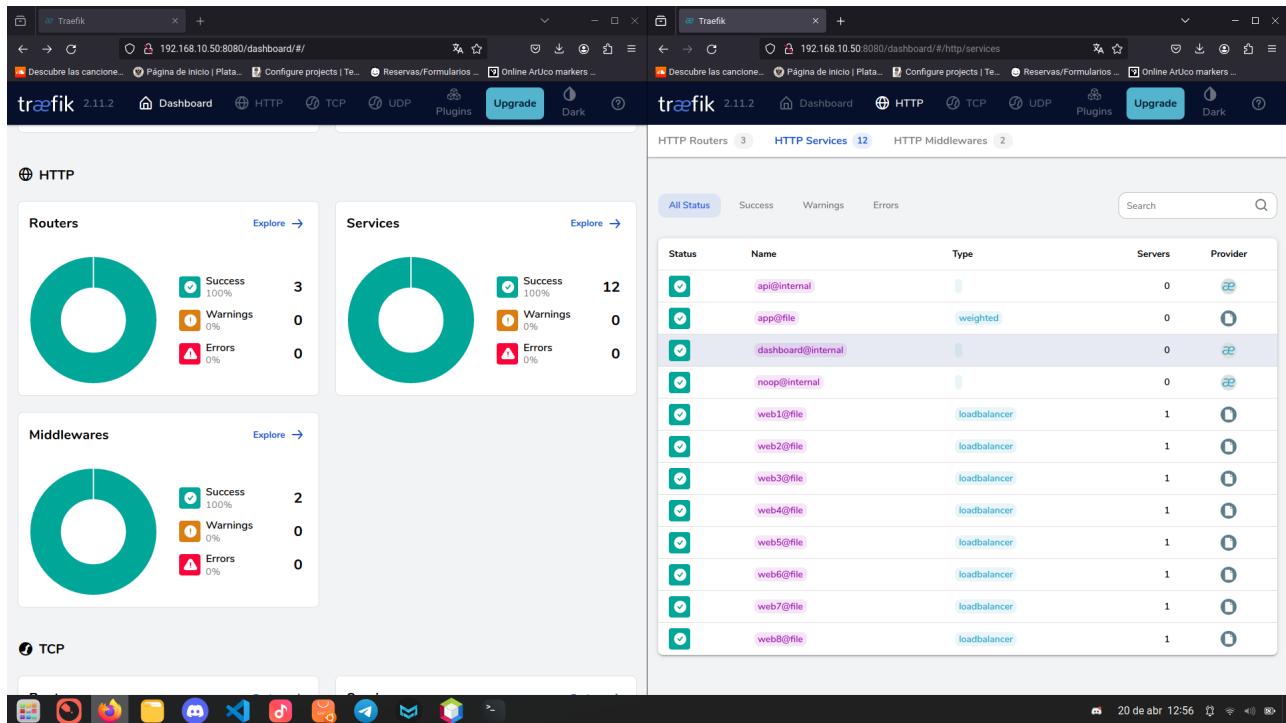
daniel-Modern-15-B7M: ~.../P2/P2-danleeeld2-traefik [main ↵]
→ curl http://192.168.10.50/
Bad Gateway
real    0m3.065s
user   0m0.005s
sys    0m0.009s

daniel-Modern-15-B7M: ~.../P2/P2-danleeeld2-traefik [main ↵]
→ curl http://192.168.10.50/
Bad Gateway
real    0m3.065s
user   0m0.005s
sys    0m0.009s

daniel-Modern-15-B7M: ~.../P2/P2-danleeeld2-traefik [main ↵]
→ curl http://192.168.10.50/
Bad Gateway
real    0m3.065s
user   0m0.005s
sys    0m0.009s

```

Vemos que algunas veces falla y nos devuelve un **Bad GateAway**. Si observamos las estadísticas:



Vemos como los servicios no aparecen desconectados. Esto se debe a que a la hora de configurar la herramienta no incluimos checkeo de la salud de los servicios⁴.

2.5. Automatización de escalado de escenario

2.5.1. Monitorización de la Carga de los Servidores

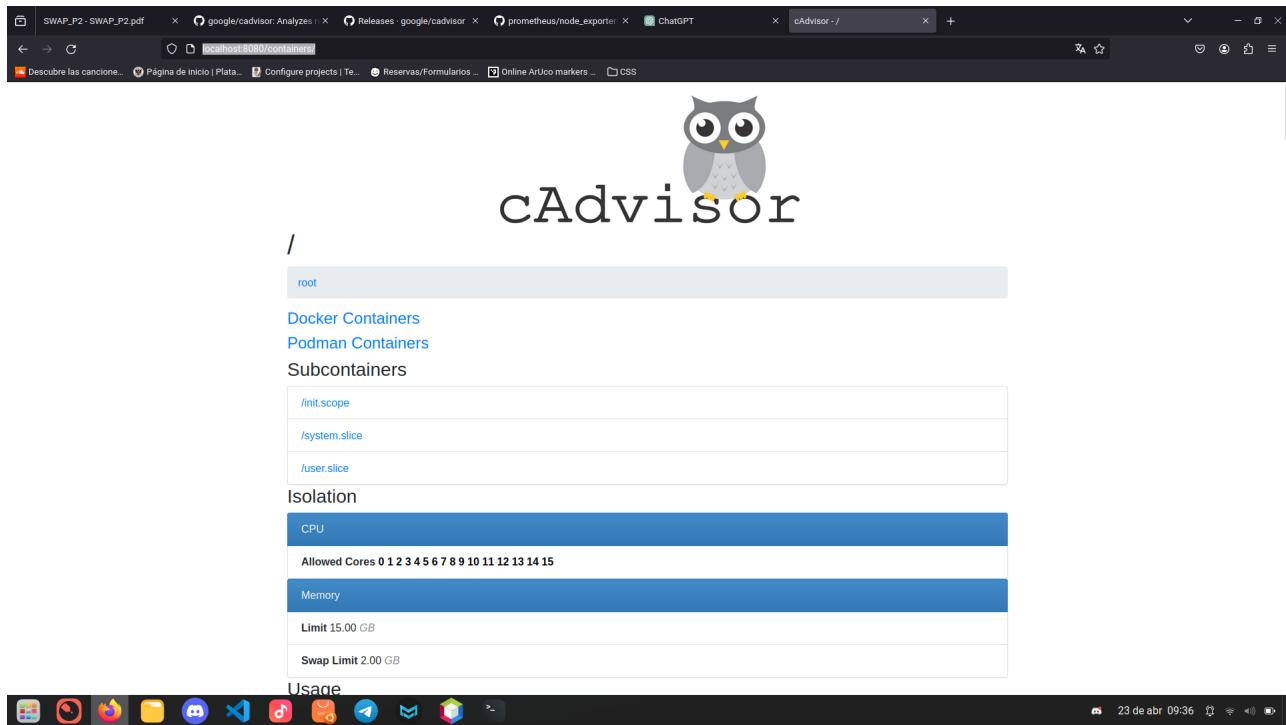
Para este apartado vamos a comenzar configurando la herramienta `cAdvisor`. Como últimamente la IA no nos está proporcionando buenas soluciones de configuración, he decidido ir directamente al repositorio de `cAdvisor`, donde podemos ver como levantar un contenedor con este servicio de forma bastante simple (ver aquí).

He adaptado esos comandos para crear la versión de `docker-compose`, por lo que tendríamos lo siguiente:

```
cAdvisor:
  image: gcr.io/cadvisor/cadvisor:v0.49.1
  container_name: cAdvisor
  privileged: true
  ports:
    - "8080:8080"
  volumes:
    - /:/rootfs:ro
    - /var/run:/var/run:ro
    - /sys:/sys:ro
    - /var/lib/docker/:/var/lib/docker:ro
  devices:
    - /dev/kmsg
```

Con esto ya tendríamos levantado el servicio (**Nota:** *Lo he incluido en la script de generación del docker compose de HAProxy 1.3*). Podemos comprobar su funcionamiento accediendo a `http://localhost:8080`, como observamos a continuación:

⁴Si quisiéramos añadir esta utilidad a la herramienta, podemos consultar la documentación oficial aquí.



Además, podemos usar la API que tiene habilitada para filtrar y descargar datos, por ejemplo, si queremos las stats de un contenedor en específico, usamos <http://localhost:8080/api/v1.3/docker/<ID>>.

Ahora, vamos a preguntarle a Gemini como extraer este consumo de CPU del JSON:

- **Yo:** (*Le paso una captura de como está organizado el json*) ¿Cómo puedo extraer los valores de cpu total? Con esto me refiero a, en la cadena `cpu": {"usage": {"total": 9175310000, "user": 1352929000, "system": 7822380000}}`, el valor numérico a la derecha de "total".
- **Gemini:** Para extraer el valor numérico a la derecha de "total" en la estructura JSON proporcionada, puedes usar expresiones regulares nuevamente para buscar el patrón específico dentro del JSON. Aquí tienes cómo podrías hacerlo:

```

import re

# JSON proporcionado
json_data = '''
{
    "cpu": {
        "usage": {
            "total": 8212439000,
            "user": 1250299000,
            "system": 6962140000
        }
    }
}

# Expresión regular para encontrar el valor numérico después de "total"
pattern = r'"total": (\d+)"
```

```

# Buscar todas las coincidencias de la expresión regular en el JSON
match = re.search(pattern, json_data)

# Si se encuentra una coincidencia, extraer el valor numérico
if match:
    total_value = int(match.group(1))
    print("Valor total de CPU:", total_value)
else:
    print("No se encontró ningún valor total de CPU.")

```

Análisis de la salida: La idea de usar expresiones regulares no es mala, pero vamos a mejorarlo un poco. Vamos a hacer una script más genérica que calcule el promedio de consumo de `cpu` de cada contenedor `web` en el último minuto y lo almacene en un fichero:

```

import requests
import json
import re
import docker
from datetime import datetime

def obtener_promedio_cpu(container_id):
    # Obtener las metricas de CPU de un contenedor desde la API de cAdvisor
    url = "http://localhost:8080/api/v1.3/docker/" + container_id
    # Realizar la petición GET
    response = requests.get(url)

    if response.status_code == 200:
        json_data = response.text # Obtener el contenido de la respuesta
        try:
            data = json.loads(json_data) # Convertir el contenido a un objeto JSON
            # Obtener las metricas de CPU del ultimo minuto (ultimos 30 registros)
            pattern = r'"cpu": {"usage": {"total": (\d+)'
            match = re.findall(pattern, json_data)
            cpu_metrics = [int(x) for x in match[-30:]]
            # Calcular el promedio de uso de CPU
            cpu_avg = sum(cpu_metrics) / len(cpu_metrics)
            return cpu_avg
        except json.JSONDecodeError as e:
            print("Error decoding JSON:", e)
    else:
        print("Error:", response.status_code)

def main():
    # Obtener todos los contenedores con nombre webX
    docker_client = docker.from_env()
    containers = docker_client.containers.list()
    web_containers = [container for container in containers if container.name.
    startswith("web")]
    cpu_general_avg = 0

    # Calcular el promedio de uso de CPU para cada contenedor
    for container in web_containers:
        container_id = container.id
        cpu_avg = obtener_promedio_cpu(container_id)
        if cpu_avg is not None:
            cpu_general_avg += cpu_avg
            # Escribir el promedio en un archivo

```

```

container_name = container.name
container_number = re.findall(r'\d+', container_name)[-1]
file_path = f'estadisticas/web{container_number}.data'
now = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
with open(file_path, 'a') as f:
    f.write(f'{now}: {cpu_avg}\n')
    print(f'Promedio de uso de CPU para {container_name}: {cpu_avg}')
# Calcular el promedio general de uso de CPU
cpu_general_avg /= len(web_containers)
file_path = f'estadisticas/general.data'
now = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
with open(file_path, 'a') as f:
    f.write(f'{now}: {cpu_general_avg}\n')
print(f'Promedio general de uso de CPU: {cpu_general_avg}')

if __name__ == "__main__":
    main()

```

De esta forma, tenemos el promedio de consumo de `cpu` en el último minuto de cada contenedor.

Nota: Añado también la fecha porque si tuviéramos un escenario real, sería interesante buscar patrones de consumo de `cpu` en franjas horarias, para aumentar los recursos en esos momentos.

2.5.2. Script para escalar servicios

Vamos a hacer una script que escalado. Para ello, tenemos que empezar definiendo un umbral de consumo de `cpu`. Tras un rato estudiando las gráficas y valores de consumo, he decidido que el umbral será `100000000.0`. Por debajo de este valor, apaga el contenedor que menos `cpu` esté consumiendo y, por encima de este valor, levanta un nuevo contenedor y lo introduce en la configuración de `HAProxy` para el balanceo.

Vamos a comenzar preguntándole a ChatGPT:

- **Yo:** Tengo un fichero que contiene datos de esta forma:

```

2024-04-23 11:46:41: 498373766.6666666
2024-04-23 11:50:41: 498373766.6666666
2024-04-23 11:55:41: 498373766.6666666
2024-04-23 11:57:41: 498373766.6666666

```

quiero una script de python que se quede con el valor numérico de la ultima linea después de los :

- **ChatGPT:**

```

# Abre el archivo en modo lectura
with open('tu_archivo.txt', 'r') as archivo:
    # Lee todas las líneas del archivo
    lineas = archivo.readlines()

    # Obtiene la última línea del archivo
    ultima_linea = lineas[-1]

    # Divide la línea en dos partes usando el carácter ':' como separador
    partes = ultima_linea.split(':')

    # Extrae y limpia el valor numérico de la última parte
    valor_numérico = partes[1].strip()

```

```

valor_numerico = partes[-1].strip()

print("Valor numérico de la última línea:", valor_numerico)

```

Análisis de la salida: Con esto podemos plantear el siguiente esquema:

```

def get_last_cpu_usage(filename):
    with open(filename, 'r') as file:
        lines = file.readlines()
        last_line = lines[-1]
        partes = last_line.split(':')
        last_value = partes[-1].strip()
    return last_value

def escalado(cpu_average, umbral):
    if cpu_average > umbral:
        return True
    else:
        return False

def main():
    cpu = get_last_cpu_usage('estadisticas/general.data')
    umb = 100000000.0
    v = escalado(float(cpu), umb)
    if v:
        print('Escalando')
    else:
        print('No escalando')

if __name__ == '__main__':
    main()

```

Vamos a empezar con el caso de que $cpu_average < umbral$, en cuyo caso, podemos parar algún contenedor. Para ello, tenemos que hacer lo siguiente:

1. Comprobar que haya 2 o más `webX` activos.
2. Parar el servicio `webX` con mayor valor de X de los activos.
3. Eliminar la IP correspondiente del servicio de balanceo.
4. Hacer un soft-reload del servicio.

Vamos a pedirla ayuda a ChatGPT:

- **Yo:** User Hazme una función de python que haga lo siguiente:
 1. Comprueba que haya más o dos servicios de docker-compose activos llamados `webX`, donde X es un identificador
 2. En caso afirmativo, toma el `webX` de mayor indice X y lo para mediante `docker-compose down webX`
 3. Modifica el archivo de configuración `haproxy.cfg` y elimina la última linea cuya primera palabra empiece por `server web...`
- **ChatGPT:** Claro, puedo ayudarte con eso. Aquí tienes una función de Python que hace lo que has solicitado utilizando la biblioteca `docker` para interactuar con Docker y manipulando el archivo `haproxy.cfg` para eliminar la última línea que empiece por "server web...":

```

import os
import re
import docker

def stop_highest_web_service():
    # Inicializa el cliente de Docker
    client = docker.from_env()

    # Busca todos los contenedores activos
    containers = client.containers.list()

    # Filtra los contenedores que contienen "web" en su nombre
    web_containers = [container for container in containers if
                      'web' in container.name]

    # Filtra los contenedores de docker-compose
    web_compose_containers = [container for container in web_containers
                               if 'docker-compose' in container.name]

    # Comprueba si hay al menos dos contenedores activos
    if len(web_compose_containers) < 2:
        print("No hay suficientes servicios de docker-compose activos
              llamados 'webX'.")
        return

    # Ordena los contenedores de docker-compose por su identificador X
    sorted_web_compose_containers = sorted(web_compose_containers,
                                             key=lambda x: int(re.search(r'web(\d+)', x.name).group(1)))

    # Toma el último contenedor de docker-compose de la lista ordenada
    highest_web_service = sorted_web_compose_containers[-1]

    # Detiene el servicio de docker-compose
    print(f"Deteniendo el servicio '{highest_web_service.name}'...")
    highest_web_service.stop()

def modify_haproxy_config():
    # Ruta al archivo de configuración haproxy.cfg
    haproxy_config_path = '/ruta/a/haproxy.cfg'

    # Lee el contenido del archivo
    with open(haproxy_config_path, 'r') as file:
        lines = file.readlines()

    # Elimina la última línea que empiece por "server web"
    for i in range(len(lines) - 1, -1, -1):
        if lines[i].strip().startswith('server web'):
            del lines[i]
            break

```

```

# Escribe el contenido modificado de nuevo al archivo
with open(haproxy_config_path, 'w') as file:
    file.writelines(lines)

# Ejecuta las funciones
stop_highest_web_service()
modify_haproxy_config()

```

Análisis de la salida: La salida es bastante buena, aunque voy a hacer algunos ajustes para adaptarlo al código que ya teníamos. De esta forma, quedaría:

```

import os
import re
import docker

def get_last_cpu_usage(filename):
    with open(filename, 'r') as file:
        lines = file.readlines()
        last_line = lines[-1]
        partes = last_line.split(':')
        last_value = partes[-1].strip()
    return last_value

def stop_highest_webX():
    # Inicializamos el cliente de docker y obtenemos los contenedores webX
    client = docker.from_env()
    containers = client.containers.list()
    web_containers = [container for container in containers if 'web' in container.name]
    # Ordenamos los contenedores por el numero de webX
    sorted_web_compose_containers = sorted(web_containers, key=lambda x: int(re.search(r'web(\d+)', x.name).group(1)))
    # Detenemos el contenedor con mayor numero en caso de haber mas de dos
    if len(sorted_web_compose_containers) <= 2:
        print('No se pueden detener mas contenedores')
        return False
    else:
        container_to_stop = sorted_web_compose_containers[-1]
        container_to_stop.stop()
        print(f'Se ha detenido el contenedor {container_to_stop.name}')
        return True

def modify_haproxy_config(bool):
    haproxy_config_path = './haproxy.cfg'
    if bool:
        # Lee el contenido del archivo
        with open(haproxy_config_path, 'r') as file:
            lines = file.readlines()
        # Elimina la ultima linea cuya primera palabra empiece por "server web"
        for i in range(len(lines) - 1, -1, -1):
            if lines[i].strip().startswith('server web'):
                del lines[i]
                break
        # Escribe el contenido modificado de nuevo al archivo
        with open(haproxy_config_path, 'w') as file:
            file.writelines(lines)
    else:
        pass

```

```

def restart_haproxy(bool):
    if bool:
        # Acceder al contenedor balanceador-haproxy
        client = docker.from_env()
        haproxy_container = client.containers.get('balanceador-haproxy')
        # Hacer un soft reload del servicio
        haproxy_container.exec_run('haproxy -f /usr/local/etc/haproxy/haproxy.cfg -p /var/run/haproxy.pid -sf $(cat /var/run/haproxy.pid)')
    else:
        pass

def escalado(cpu_average, umbral):
    if cpu_average > umbral:
        return True
    else:
        bool = stop_highest_webX()
        bool2 = modify_haproxy_config(bool)
        restart_haproxy(bool2)
        return False

def main():
    cpu = get_last_cpu_usage('estadisticas/general.data')
    umb = 100000000.0
    v = escalado(float(cpu), umb)
    if v:
        print('Escalando')
    else:
        print('Desescalando')

if __name__ == '__main__':
    main()

```

He hecho algunos ajustes para que modifique el fichero de HAProxy y reinicie el servicio en función o no de si hay contenedores disponibles para des-escalarar.

Nota: No he conseguido encontrar el .pid del servicio para hacer el soft-reload. Buscando en internet dí con esta solución, así que voy a cambiar el comando a ejecutar para que todo funcione por el nuevo.

```

haproxy_container.exec_run('haproxy -f /usr/local/etc/haproxy/haproxy.cfg -p /var/run/haproxy.pid -sf $(cat /var/run/haproxy.pid)')
# Lo cambiamos por
command = "docker kill -s HUP balanceador-haproxy"
subprocess.run(command, shell=True)

```

Vamos a comprobar que funciona, para ello, empezamos desde el punto de partida, con todo levantado y la configuración habitual:

He colocado un valor adecuado en los ficheros `estadisticas/general.data` para que ejecute esta parte de la script:

Terminal

```
daniel-Modern-15-87M: ~ /.../P2/P2-danieleeld2-HAProxy |main U:3 ?:3 X|  
→python3 scripts/escalado.py  
Se ha detenido el contenedor web8  
balanceador-haproxy  
Desescalando  
daniel-Modern-15-87M: ~ /.../P2/P2-danieleeld2-HAProxy |main U:3 ?:3 X|  
→
```

SWAP_P2 - SWAP_F ChatGPT cAdvisor - / AlconchoVazquez Statistics Report X + ⌂

localhost:9000/estadisticas_danieleeld2
Describe las canciones... Página de inicio | Plata... Configure projects | Te... Reservas/Formularios ... onlineArUco markers ... >>

HAProxy version 2.9.6-9eafce5, released 2024/02/26

Statistics Report for pid 26

> General process information

pid = 26 (process #1, listening = 1) uptime = 0d 00m00s, warnings = 0 system memory: memfree = 1071648k, freehigh = 6582656k process memory: total = 124248k, reached = 0k, maxpipes = 0 current conn = 1, current pipes = 0/0, core rate = 1/sec, bit rate = 0.000 kbps running tasks: Usr2 (0 nice), dso = 100 %	active UP active UP, going down active DOWN, going up active or backup DOWN active or backup DOWN for maintenance (MAINT)	backup UP backup UP, going down backup DOWN, going up backup DOWN for maintenance (MAINT)	Display option: Scope: Local servers • Refresh now • CSV export Note: NO LB DRAM - UP with load-balancing disabled.	External resources: • Python API • Metrics (V2.9) • Online manual
---	---	--	---	--

Frontend danieleeld2

Queue	Session rate			Sessions			Bytes	Denied	Errors	Warnings	Reqs	Replies	Status	LastClik	Wight	Act Bck	Chk	Dwn	Downtime	Server
	Cur Max	Limit	Cur Max	Limit	Cur Max	Limit														
Frontend	0	0	-	0	0	524 248	0	0	0	0	0	0	0	0	0	0	0	0	0	OPEN

Backend danieleeld2

Queue	Session rate			Sessions			Bytes	Denied	Errors	Warnings	Reqs	Replies	Status	LastClik	Wight	Act Bck	Chk	Dwn	Downtime	Server						
	Cur Max	Limit	Cur Max	Limit	Cur Max	Limit															Total	LbTot	Last In	Out	Req	Req Conn
web1	0	-	0	0	0	32	0	0	7	0	0	0	0	0	0	45	UP	LACK in Dens	1/1	Y	-	0	0	0s		
web2	0	-	0	0	0	32	0	0	7	0	0	0	0	0	0	45	UP	LACK in Dens	1/1	Y	-	0	0	0s		
web3	0	-	0	0	0	32	0	0	7	0	0	0	0	0	0	45	UP	LACK in Dens	1/1	Y	-	0	0	0s		
web4	0	-	0	0	0	32	0	0	7	0	0	0	0	0	0	45	UP	LACK in Dens	1/1	Y	-	0	0	0s		
web5	0	-	0	0	0	32	0	0	7	0	0	0	0	0	0	45	UP	LACK in Dens	1/1	Y	-	0	0	0s		
web6	0	-	0	0	0	32	0	0	7	0	0	0	0	0	0	45	UP	LACK in Dens	1/1	Y	-	0	0	0s		
web7	0	-	0	0	0	32	0	0	7	0	0	0	0	0	0	45	UP	LACK in Dens	1/1	Y	-	0	0	0s		
Backend	0	0	0	0	0	0	52425	0	0	7	0	0	0	0	0	0	0	0	45	UP	7/7	7	0	0	0	0s

stats

Queue	Session rate			Sessions			Bytes	Denied	Errors	Warnings	Reqs	Replies	Status	LastClik	Wight	Act Bck	Chk	Dwn	Downtime	Server					
	Cur Max	Limit	Cur Max	Limit	Cur Max	Limit															Total	LbTot	Last In	Out	Req
Frontend	1	1	-	1	1	524 248	1	0	0	0	0	0	0	0	0	0	0	0	OPEN						
Backend	0	0	0	0	0	0	52425	0	0	0	0	0	0	0	0	0	0	45	UP	0/0	0	0	0	0	0s

23 de abr 15:46

Como vemos, hasta aquí, la script funciona correctamente. El problema está en que si nosotros ahora pasamos el servicio (`docker-compose down`) y volvemos a levantarla, el fichero de configuración de HAProxy sigue desconfigurado, ya que lo hemos sobre-escrito. Abarcaremos este inconveniente después de la siguiente parte de la script.

Ahora tenemos que abarcar el caso contrario, que es el de levantar contenedores. Para ello, necesitamos:

1. Comprobar si tenemos contenedores suficientes para levantar, en caso contrario, llamar a la script de creación del `docker-compose` con $X+1$ contenedores.
2. Levantar dicho contenedor.
3. Agregar su IP al archivo de configuración, para que se tome en cuenta para el balanceo.
4. Recargar el servicio de balanceo.

Nos encontramos con un inconveniente y es, como sabemos que cantidad de servicios `webX` tenemos en el `docker-compose.yml`, para saber si podemos levantar un nuevo servicio o llamamos a uno nuevo. Para ello y en relación con el problema comentado en el apartado anterior, se me ha ocurrido manejar un fichero `.env` que indique el número de `webX` que tenemos definidos en el `docker-compose.yml`. Para ello, en la script de generación 1.3 tenemos que añadir que escriba la variable en este fichero:

```
echo "$num_instances" > instancias.env
```

Ahora, podemos añadir en la función `stop_highest_webX` la siguiente línea:

```
...
else:
    container_to_stop = sorted_web_compose_containers[-1]
    numero_contenedor = int(container_to_stop.name.split("web")[-1])
    command = "./scripts/generar-docker-compose.sh " + str(numero_contenedor -1)
    subprocess.run(command, shell=True)
    container_to_stop.stop()
...
...
```

Siguiendo un poco la dinámica anterior, podemos declarar la siguiente función:

```
def start_highest_webX():
    # Inicializamos el cliente de docker y obtenemos los contenedores webX
    client = docker.from_env()
    containers = client.containers.list()
    web_containers = [container for container in containers if 'web' in container.name]
    # Ordenamos los contenedores por el número de webX
    sorted_web_compose_containers = sorted(web_containers,
                                             key=lambda x: int(re.search(r'web(\d+)', x.name).group(1)))
    last_container = sorted_web_compose_containers[-1]
    numero_contenedor = int(last_container.name.split("web")[-1])
    # Leemos el numero del .env
    with open('instancias.env', 'r') as archivo:
        valor = archivo.read().strip()
        valor_numerico = int(valor)

    if numero_contenedor >= valor_numerico:
        command = "./scripts/generar-docker-compose.sh " + str(numero_contenedor +1)
        subprocess.run(command, shell=True)

    command = "docker-compose up -d web" + str(numero_contenedor+1)
    subprocess.run(command, shell=True)
    print(f'Se ha iniciado el contenedor web{numero_contenedor+1}')
    return numero_contenedor+1
```

La idea es muy parecida a la anterior, mira si tiene definidos servicios suficientes en el `docker-compose.yml` para levantar y si no, llama a la script de generación 1.3.

El siguiente paso es modificar el fichero de configuración de HAProxy para que añada la nueva línea. Para esto, vamos a pedirle ayuda a ChatGPT:

- **Yo:** Tengo un fichero de configuración de HAProxy de esta forma: (*Le paso el fichero de configuración de HAProxy*) quiero que crees una función de python que añada debajo de la última línea que empiece por server web una nueva línea que sea exactamente igual que la de arriba, pero que sea `webX+1` y `192.168.10.X+1`.
- **ChatGPT:** Puedes lograrlo fácilmente con una función de Python que lea el archivo de configuración, encuentre la última línea que empiece por "server web" luego añada una nueva línea con la información deseada. Aquí te dejo un ejemplo de cómo podrías hacerlo:

```
def agregar_servidor(config_file):  
    with open(config_file, 'r') as f:  
        lines = f.readlines()  
  
        # Encontrar la última línea que empiece por "server web"  
        last_server_line_index = -1  
        for i in range(len(lines) - 1, -1, -1):  
            if lines[i].startswith("server web"):  
                last_server_line_index = i  
                break  
  
        if last_server_line_index == -1:  
            # No se encontraron líneas que empiecen por "server web"  
            return False  
  
        # Extraer el número del último servidor  
        last_server_number = int(lines[last_server_line_index].split()[1])  
  
        # Construir la nueva línea  
        new_server_line = f"    server web{last_server_number + 1}  
                           192.168.10.{last_server_number + 1}:80 maxconn 32 check\n"  
  
        # Insertar la nueva línea debajo de la última línea  
        lines.insert(last_server_line_index + 1, new_server_line)  
  
    with open(config_file, 'w') as f:  
        f.writelines(lines)  
  
    return True  
  
# Uso de la función  
archivo_configuracion = "haproxy.cfg"  
exito = agregar_servidor(archivo_configuracion)  
if exito:  
    print("Se ha agregado el nuevo servidor correctamente.")  
else:
```

```
        print("No se encontró ninguna línea que empiece por 'server web'.")
```

Análisis de la salida: Vamos a realizarle algunos cambios para adaptarla a lo que tenemos, pero es una buena solución.

```
def add_haproxy_config():
    haproxy_config_path = './haproxy.cfg'
    with open(haproxy_config_path, 'r') as f:
        lines = f.readlines()

    # Encontrar la última línea que empiece por "server web"
    for i in range(len(lines) - 1, -1, -1):
        if lines[i].strip().startswith('server web'):
            last_server_line_index = i
            break

    if last_server_line_index == -1:
        # No se encontraron líneas que empiecen por "server web"
        return False

    # Extraer el número del último servidor
    last_server = lines[last_server_line_index].split()[1]
    last_server_number = last_server.split("web")[-1]

    # Construir la nueva línea
    new_server_line = f"    server web{int(last_server_number) + 1}\n        192.168.10.{int(last_server_number) + 2}:80 maxconn 32 check\n"

    # Insertar la nueva línea debajo de la última línea
    lines.insert(last_server_line_index + 1, new_server_line)

    with open(haproxy_config_path, 'w') as f:
        f.writelines(lines)

    return True
```

Nuestra script finalmente quedaría como:

```
import os
import re
import docker
import subprocess

def get_last_cpu_usage(filename):
    with open(filename, 'r') as file:
        lines = file.readlines()
        last_line = lines[-1]
        partes = last_line.split(':')
        last_value = partes[-1].strip()
    return last_value

def stop_highest_webX():
    # Inicializamos el cliente de docker y obtenemos los contendores webX
```

```

client = docker.from_env()
containers = client.containers.list()
web_containers = [container for container in containers if 'web' in container.name]
]
# Ordenamos los contenedores por el numero de webX
sorted_web_compose_containers = sorted(web_containers, key=lambda x: int(re.search(r'web(\d+)', x.name).group(1)))
# Detenemos el contenedor con mayor numero en caso de haber mas de dos
if len(sorted_web_compose_containers) <= 2:
    print('No se pueden detener mas contenedores')
    return False
else:
    container_to_stop = sorted_web_compose_containers[-1]
    numero_contenedor = int(container_to_stop.name.split("web")[-1])
    command = "./scripts/generar-docker-compose.sh " + str(numero_contenedor - 1)
    subprocess.run(command, shell=True)
    container_to_stop.stop()
    command = "docker rm web" + str(numero_contenedor)
    print(f'Se ha detenido el contenedor {container_to_stop.name}')
    return True

def start_highest_webX():
    # Inicializamos el cliente de docker y obtenemos los contenedores webX
    client = docker.from_env()
    containers = client.containers.list()
    web_containers = [container for container in containers if 'web' in container.name]
    ]
    # Ordenamos los contenedores por el numero de webX
    sorted_web_compose_containers = sorted(web_containers, key=lambda x: int(re.search(r'web(\d+)', x.name).group(1)))
    last_container = sorted_web_compose_containers[-1]
    numero_contenedor = int(last_container.name.split("web")[-1])
    # Leemos el numero del .env
    with open('instancias.env', 'r') as archivo:
        valor = archivo.read().strip() # Lee el contenido y elimina espacios en blanco al inicio y al final
        valor_numerico = int(valor)

    if numero_contenedor >= valor_numerico:
        command = "./scripts/generar-docker-compose.sh " + str(numero_contenedor + 1)
        subprocess.run(command, shell=True)

    command = "docker-compose up -d web" + str(numero_contenedor+1)
    subprocess.run(command, shell=True)
    print(f'Se ha iniciado el contenedor web{numero_contenedor+1}')

def modify_haproxy_config(bool):
    haproxy_config_path = './haproxy.cfg'
    if bool:
        # Lee el contenido del archivo
        with open(haproxy_config_path, 'r') as file:
            lines = file.readlines()
        # Elimina la ultima linea cuya primera palabra empiece por "server web"
        for i in range(len(lines) - 1, -1, -1):
            if lines[i].strip().startswith('server web'):
                del lines[i]
                break
        # Escribe el contenido modificado de nuevo al archivo
        with open(haproxy_config_path, 'w') as file:
            file.writelines(lines)

```

```

        return True
    else:
        return False

def add_haproxy_config():
    haproxy_config_path = './haproxy.cfg'
    with open(haproxy_config_path, 'r') as f:
        lines = f.readlines()

    # Encontrar la ultima linea que empiece por "server web"
    for i in range(len(lines) - 1, -1, -1):
        if lines[i].strip().startswith('server web'):
            last_server_line_index = i
            break

    if last_server_line_index == -1:
        # No se encontraron lineas que empiecen por "server web"
        return False

    # Extraer el numero del ultimo servidor
    last_server = lines[last_server_line_index].split()[1]
    last_server_number = last_server.split("web")[-1]

    # Construir la nueva linea
    new_server_line = f"    server web{int(last_server_number) + 1} 192.168.10.{int(last_server_number) + 2}:80 maxconn 32 check\n"

    # Insertar la nueva linea debajo de la ultima linea que empiece por "server web"
    lines.insert(last_server_line_index + 1, new_server_line)

    with open(haproxy_config_path, 'w') as f:
        f.writelines(lines)

    return True

def restart_haproxy(bool):
    if bool:
        # Reiniciar el servicio de HAProxy
        command = "docker kill -s HUP balanceador-haproxy"
        subprocess.run(command, shell=True)
    else:
        pass

def escalado(cpu_average, umbral):
    if cpu_average > umbral:
        start_highest_webX()
        bool = add_haproxy_config()
        restart_haproxy(bool)
        return True
    else:
        bool = stop_highest_webX()
        bool2 = modify_haproxy_config(bool)
        restart_haproxy(bool2)
        return False

def main():
    cpu = get_last_cpu_usage('estadisticas/general.data')
    umb = 100000000.0

```

```

v = escalado( float(cpu) , umbr)
if v:
    print('Escalando')
else:
    print('Desescalando')

if __name__ == '__main__':
    main()

```

Nota: En la primera parte, además de parar el contenedor, también lo elimino para que no dé problemas de ejecución la script.

Vamos a probarlo. He vuelto a distribuir todo como en el caso base y he colocado los datos para que esta vez el consumo sea mayor que el umbral:

The terminal window shows the following output:

```

daniel-Modern-15-B7M: ~/.../P2/P2-daneeeld2-HAProxy |main U:3 7:4 X|
→ docker container up -d
[...] Running 19/13
Network p2-daneeeld2-haproxy_default Created 2.25
Network p2-daneeeld2-haproxy_red_web Created 2.15
Network p2-daneeeld2-haproxy_red_serviceclos C... 2.05
Container CADvisor Started 0.75
Container web5 Started 1.25
Container web3 Started 1.15
Container web6 Started 1.05
Container web2 Started 0.65
Container web8 Started 1.45
Container web1 Started 0.75
Container web4 Started 1.05
Container web7 Started 1.05
Container balanceador-haproxy Started 1.95

daniel-Modern-15-B7M: ~/.../P2/P2-daneeeld2-HAProxy |main U:3 7:4 X|
→ docker ps --format '{{.Names}}'
balanceador-haproxy
web3
web1
web5
web7
web6
web5
web4
cadvisor
web8
web2

daniel-Modern-15-B7M: ~/.../P2/P2-daneeeld2-HAProxy |main U:3 7:4 X|
→

```

The HAProxy statistics report shows the following details:

HAProxy version 2.9.6-9eacef5, released 2024/02/26

Statistics Report for pid 8

General process information

Process	State	Uptime	Memory	Threads
pid # 8 (process #4, stepid = 1, birthpid = 16)	active UP	00:00:00	0	0
system limits: memmax = unlimited, ulimit-n = 1048576	active UP, going down	00:00:00	0	0
maxfiles: open files = 102400, maxfd = 102400, reached fd: 0, mapsize = 3	active DOWN	00:00:00	0	0
current tasks: 1, current queue: 0, current rate: 0 b/sec, 0 error/s, 0.000000 kbps	active or backup DOWN for maintenance (MAINT)	00:00:00	0	0
Running tasks: 0/53 (0 nice), idle = 100 %	active or backup SOFT STOPPED for maintenance	00:00:00	0	0

Note: NOBODY/DRAINY = UP with load-balancing disabled.

Frontend: daneeeld2

Queue	Session rate			Sessions			Bytes			Denied			Errors			Warnings			Server							
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTotal	LastIn	Out	Req	Conn	Resp	Rej	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Downtime
Frontend	0	0	-	0	0	-	0	0	0	524248	0	0	0	0	0	0	0	OPEN								

Backend: daneeeld2

Queue	Session rate			Sessions			Bytes			Denied			Errors			Warnings			Status			LastChk			Wght			Act			Bck			Chk			Dwn			Downtime			Throttle		
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTotal	LastIn	Out	Req	Conn	Resp	Rej	Redis	Server	Act	Bck	Chk	Dwn	Downtime	Throttle																				
Backend	0	0	-	0	0	-	0	0	0	52425	0	0	0	0	0	0	0	OPEN	88	8	0	0	0	0	0	0	0	0	0	0	0	0													

stats

Queue	Session rate			Sessions			Bytes			Denied			Errors			Warnings			Status			LastChk			Wght			Act			Bck			Chk			Dwn			Downtime			Throttle		
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTotal	LastIn	Out	Req	Conn	Resp	Rej	Redis	Server	Act	Bck	Chk	Dwn	Downtime	Throttle																				
Frontend	0	0	-	1	1	-	7	534248	3	486	27120	0	0	0	0	0	0	OPEN	88	8	0	0	0	0	0	0	0	0	0	0	0														
Backend	0	0	-	0	0	-	0	0	0	52425	0	0	0	0	0	0	0	OPEN	0	0	0	0	0	0	0	0	0	0	0	0	0														

```

daniel-Modern-15-B7M: ~/.../P2/P2-danteelld2-HAProxy [main U:3 ?:4 X]
→ docker compose up -d
[+] Running 10/13
  ⚡ Network p2-danteelld2-haproxy_default      Created          2.25
  ⚡ Network p2-danteelld2-haproxy_red_web       Created          2.15
  ⚡ Network p2-danteelld2-haproxy_red_servicios C...
  ✓ Container cAdvisor                         Started         2.05
  ✓ Container web1                            Started         0.75
  ✓ Container web2                            Started         1.25
  ✓ Container web3                            Started         1.45
  ✓ Container web4                            Started         1.55
  ✓ Container web5                            Started         1.65
  ✓ Container web6                            Started         0.65
  ✓ Container web7                            Started         1.45
  ✓ Container web8                            Started         0.75
  ✓ Container web9                            Started         1.65
  ✓ Container balanceador-haproxy            Started         1.95

daniel-Modern-15-B7M: ~/.../P2/P2-danteelld2-HAProxy [main U:3 ?:4 X]
→ docker ps -format '{{.Names}}'
balanceador-haproxy
web3
web4
web5
web6
web7
web8
cAdvisor
web9
web2

daniel-Modern-15-B7M: ~/.../P2/P2-danteelld2-HAProxy [main U:3 ?:4 X]
→ python3 scripts/escalado.py
Creating web9 ... done
Se ha iniciado el contenedor web9
balanceador-haproxy
Escalando
Escalando
daniel-Modern-15-B7M: ~/.../P2/P2-danteelld2-HAProxy [main U:3 ?:4 X]
→

```

Statistics Report for pid 26

General process information

Frontend demarcated																				
Queue	Session rate			Sessions			Bytes			Denied Errors Warnings										
Cur Max Limit	Cur Max	Cur Limit	Cur Max	Cur Max Limit	Cur Max	Cur Max Limit	Total	LstFor	Last In	Out Req	Req Conn	Req Redis	Status	LastChk	Wght	Act	Bck Chk	Dwn	Downtime	Thrthr
Frontend	0	0	-	0	0	0	524,247	0	0	0	0	0	0	OPEN						

Backend demarcated																				
Queue	Session rate			Sessions			Bytes			Denied Errors Warnings										
Cur Max Limit	Cur Max	Cur Limit	Cur Max	Cur Max Limit	Cur Max	Cur Max Limit	Total	LstFor	Last In	Out Req	Req Conn	Req Redis	Status	LastChk	Wght	Act	Bck Chk	Dwn	Downtime	Thrthr
Backend	0	0	-	0	0	0	524,255	0	0	0	0	0	0	OPEN						

stats

Frontend demarcated																				
Queue	Session rate			Sessions			Bytes			Denied Errors Warnings										
Cur Max Limit	Cur Max	Cur Limit	Cur Max	Cur Max Limit	Cur Max	Cur Max Limit	Total	LstFor	Last In	Out Req	Req Conn	Req Redis	Status	LastChk	Wght	Act	Bck Chk	Dwn	Downtime	Thrthr
Frontend	0	0	-	0	0	0	524,247	1	0	0	0	0	0	OPEN						

Backend demarcated																				
Queue	Session rate			Sessions			Bytes			Denied Errors Warnings										
Cur Max Limit	Cur Max	Cur Limit	Cur Max	Cur Max Limit	Cur Max	Cur Max Limit	Total	LstFor	Last In	Out Req	Req Conn	Req Redis	Status	LastChk	Wght	Act	Bck Chk	Dwn	Downtime	Thrthr
Backend	0	0	-	0	0	0	524,255	0	0	0	0	0	0	OPEN						

Como vemos, funciona perfectamente. Si ahora volvemos a poner un valor por debajo del umbral y volvemos a ejecutar:

```

daniel-Modern-15-B7M: ~/.../P2/P2-danteelld2-HAProxy [main U:3 ?:4 X]
→ docker ps -format '{{.Names}}'
balanceador-haproxy
web1
web2
cAdvisor
web3
web4
web5
web6
web7
web8
web9

daniel-Modern-15-B7M: ~/.../P2/P2-danteelld2-HAProxy [main U:3 ?:4 X]
→ python3 scripts/escalado.py
Creating web9 ... done
Se ha iniciado el contenedor web9
balanceador-haproxy
Escalando
Escalando
daniel-Modern-15-B7M: ~/.../P2/P2-danteelld2-HAProxy [main U:3 ?:4 X]
→

```

Statistics Report for pid 44

General process information

Frontend demarcated																				
Queue	Session rate			Sessions			Bytes			Denied Errors Warnings										
Cur Max Limit	Cur Max	Cur Limit	Cur Max	Cur Max Limit	Cur Max	Cur Max Limit	Total	LstFor	Last In	Out Req	Req Conn	Req Redis	Status	LastChk	Wght	Act	Bck Chk	Dwn	Downtime	Thrthr
Frontend	0	0	-	0	0	0	524,247	0	0	0	0	0	0	OPEN						

Backend demarcated																				
Queue	Session rate			Sessions			Bytes			Denied Errors Warnings										
Cur Max Limit	Cur Max	Cur Limit	Cur Max	Cur Max Limit	Cur Max	Cur Max Limit	Total	LstFor	Last In	Out Req	Req Conn	Req Redis	Status	LastChk	Wght	Act	Bck Chk	Dwn	Downtime	Thrthr
Backend	0	0	-	0	0	0	524,255	0	0	0	0	0	0	OPEN						

stats

Frontend demarcated																				
Queue	Session rate			Sessions			Bytes			Denied Errors Warnings										
Cur Max Limit	Cur Max	Cur Limit	Cur Max	Cur Max Limit	Cur Max	Cur Max Limit	Total	LstFor	Last In	Out Req	Req Conn	Req Redis	Status	LastChk	Wght	Act	Bck Chk	Dwn	Downtime	Thrthr
Frontend	0	1	-	1	4	524,247	1	496	37,119	0	0	0	OPEN							

Backend demarcated																				
Queue	Session rate			Sessions			Bytes			Denied Errors Warnings										
Cur Max Limit	Cur Max	Cur Limit	Cur Max	Cur Max Limit	Cur Max	Cur Max Limit	Total	LstFor	Last In	Out Req	Req Conn	Req Redis	Status	LastChk	Wght	Act	Bck Chk	Dwn	Downtime	Thrthr
Backend	0	0	-	0	0	0	524,255	0	0	0	0	0	0	OPEN						

Vemos que lo deja como estaba.

2.5.3. Automatización del script

Finalmente, para configurar la automatización de las scripts, podemos usar Crontab. Por un lado, vamos a configurar que se ejecute la script de monitorización cada minuto y, por otro lado, que se ejecute la de

escalado cada dos minutos. Para ello, es tan simple como:

```
$ crontab -e
# Con el comando anterior entramos en modo edición
# Para que se ejecute cada dos minutos
*/2 * * * * cd /ruta/al/directorio && /ruta/al/directorio/tu_script.sh
# Para que se ejecute cada minuto
* * * * * cd /ruta/al/directorio && ./tu_script.sh
```

En mi caso sería:

```
*/2 * * * * cd /home/daniel/Git/SWAP-UGR/P2 && python3 /scripts/escalado.py
* * * * * cd /home/daniel/Git/SWAP-UGR/P2 && python3 /scripts/promedio.py
```

Para ver si funciona correctamente, podemos ver los logs en `/var/log/cron` o ejecutar el comando `$ grep CRON /var/log/syslog`