

PRÁCTICA 2

Herramientas y utilidades de diagnóstico en red



UNIVERSIDAD
DE GRANADA

Titulación: Ingeniería Informática + ADE

FLORIN EMANUEL TODOR GLIGA

ÍNDICE

1. Tareas Básicas

- a. B1. Preparación del Entorno de Trabajo:
- b. B2. Configuración de Nginx como Balanceador de Carga:
- c. B3. Implementación del escenario de Nginx con Docker Compose:
- d. B4. Verificación y Pruebas del escenario de Nginx:
- e. B5. Configuración de HAProxy como Balanceador de Carga:
- f. B6. Implementación del escenario de HAProxy con Docker Compose
- g. B7. Verificación y Pruebas del escenario de HAProxy

2. Tareas Avanzadas

- a. A1. Configuraciones Avanzadas de Nginx:
- b. A2. Configuraciones Avanzadas de HAProxy
- c. Breve Inciso
- d. A3. Experimentación con Diferentes Balanceadores de Carga:
 - i. Traefik
 - ii. Envoy
- e. A4. Investigación y Pruebas de Tolerancia a Fallos:
- f. A5. Automatización de escalado del escenario

3. Uso de Inteligencia Artificial Generativa

Tareas Básicas

B1. Preparación del Entorno de Trabajo

Para la parte básica de esta práctica, por ahora, el entorno de trabajo es el siguiente:

```
● > tree
.
├── docker-compose_haproxy_balanceador.yaml
├── docker-compose_nginx_balanceador.yaml
└── init.sh
    ├── logs_apache
    │   ├── access.log
    │   ├── apache_monitor_web1.log
    │   ├── apache_monitor_web3.log
    │   ├── apache_monitor_web5.log
    │   ├── apache_monitor_web7.log
    │   └── error.log
    ├── logs_nginx
    │   ├── access_web.log
    │   └── error_web.log
    ├── P2-flotodor-apache
    │   ├── apache_config
    │   ├── apache_monitor.sh
    │   ├── DockerfileApache_florin
    │   └── entrypoint_apache.sh
    ├── P2-flotodor-haproxy
    │   ├── config_balanceador
    │   │   └── haproxy.cfg
    │   └── DockerfileHaproxy_balanceador
    ├── P2-flotodor-nginx
    │   ├── config_balanceador
    │   │   └── nginx.conf
    │   ├── config_webs
    │   │   └── default
    │   │       └── nginx.conf
    │   ├── DockerfileNginx_balanceador
    │   ├── DockerfileNginx_web
    │   └── entrypoint_nginx.sh
    └── web_flotodor
        └── index.php

11 directories, 24 files
```

Como podemos ver, hay diferentes directorios, en el que encontramos la información relevante de la parte básica de esta práctica junto al uso de la parte creada de la práctica anterior (como por ejemplo el script de init.sh)

B2. Configuración de Nginx como Balanceador de Carga

Para esta parte, hago uso de un dockerfile simple para la creación del contenedor del balanceador de carga.

```

1  FROM nginx:latest
2
3  RUN apt-get update && apt-get upgrade -y
4  RUN apt install -y net-tools iputils-ping iptables
5
6  # Configuración de Nginx
7  RUN rm /etc/nginx/nginx.conf
8  COPY ./P2-flotodor-nginx/config_balanceador/nginx.conf /etc/nginx/nginx.conf
9
10 EXPOSE 80
11
12 CMD ["nginx", "-g", "daemon off;"]
13

```

Como podemos ver, además de descargar las herramientas necesarias, eliminamos el nginx.conf predeterminado y enviamos el siguiente nginx.conf que se nos pide para la práctica.

```

## LOGS MODIFICADOS PARA MOSTRAR INFORMACION DETALLADA
log_format detailed_logs_balanceador '[CLIENT $remote_addr] - $remote_user [$time_local] '
    '$request' => $status ($body_bytes_sent bytes)
    'Referer: "$http_referer"'
    'Agent: "$http_user_agent"'
    'RequestTime: $request_time sec';

upstream backend_flotodor {
    # por defecto es round-robin
    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_balanceador;
    root /usr/share/nginx/html;
    index index.php index.html;

    access_log /var/log/nginx/nginx_flotodor_acces_balanceador.log detailed_logs_balanceador;
    error_log /var/log/nginx/nginx_flotodor_balanceador_error.log;
    location / {
        proxy_pass http://backend_flotodor;
        proxy_set_header Cookie $http_cookie;
        proxy_hide_header Set-Cookie;
    }

    location /estadisticas_flotodor{
        stub_status on;
    }
}

sendfile on;
tcp_nopush on;
types_hash_max_size 2048;
# server_tokens off;

```

Vemos el uso del formato de los logs que modifiqué en la práctica 1, por otro lado vemos el uso del balanceador con el algoritmo de round-robin (que es el que está por defecto en nginx, por eso no hace falta indicarlo), además de ver la información relevante que se nos pide que coloquemos en dicho fichero como son los servidores de backend que va a gestionar el balanceador, etc.

B3. Implementación del escenario de Nginx con Docker Compose:

Para esta parte, he de comentar previamente que he creado dos docker-compose.ymal (por ahora son dos debido a que es la parte básica de esta práctica, todavía me quedan implementar diferentes balanceador de la parte avanzada).

Este docker-compose es el mismo que el de la práctica anterior solo que ha tenido algunos cambios como es el uso de las redes de forma externa y la implementación del nginx_balanceador con su dependencia de los servidores web, es decir, primero se deben de crear dichos servidores web y posteriormente el balanceador.

Omito la información irrelevante de las webs, ya que es la misma que la práctica anterior:

```

x-common-nginx_balanceador-config: &common-nginx_balanceador-config
  image: flotodor-nginx_balanceador-image:p2
  restart: always
  volumes:
    - ./logs_nginx:/var/log/nginx

131  ▶ Run Service
132  nginx_balanceador:
133    <<: *common-nginx_balanceador-config
134    container_name: nginx_balanceador
135    environment:
136      - SERVER_NAME=nginx_balanceador
137    networks:
138      red_web:
139        | ipv4_address: 192.168.10.50
140    ports:
141      - "80:80"
142    depends_on:
143      - web1
144      - web2
145      - web3
146      - web4
147      - web5
148      - web6
149      - web7
150      - web8
151
152  networks:
153    red_web:
154      external: true
155    red_servicios:
156      external: true

```

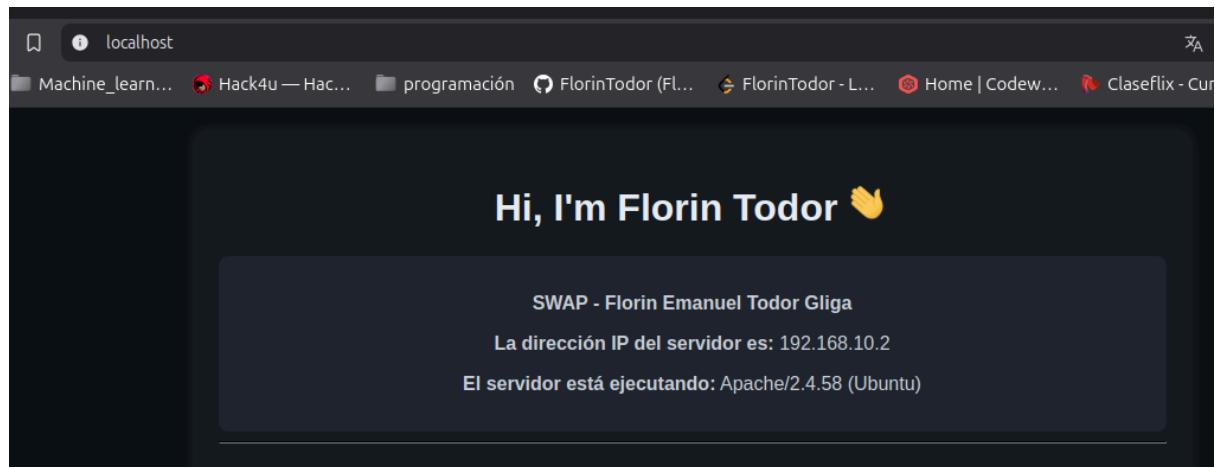
B4. Verificación y Pruebas del escenario de Nginx:

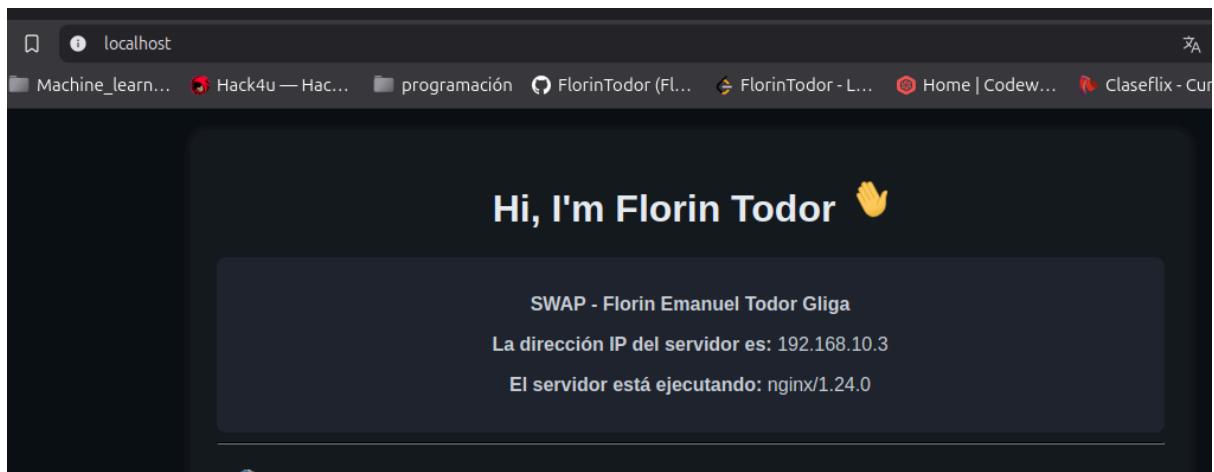
```
└─[~]~/Escritorio/SWAP/P2| on [main] ?2
  ./init.sh -u nginx
```

```
> ./init.sh -u nginx
[✓] Red red_servicios ya existe.
[✓] Red red_web ya existe.
[i] Comprobando puertos 8080 a 8089...
[!] Se detectó haproxy_balanceador corriendo. Deteniéndolo...
[✓] haproxy_balanceador detenido.
[+] Running 9/9
✓ Container web1           Running
✓ Container web2           Running
✓ Container web3           Running
✓ Container web6           Running
✓ Container web8           Running
✓ Container web4           Running
✓ Container web7           Running
✓ Container web5           Running
✓ Container nginx_balanceador Started
[+] Servicios iniciados con Nginx.
```

```
[+] Servicios iniciados con nginx.
• docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
bc446f57d511 flotodor-nginx_balanceador-image:p2 "/docker-entrypoint..." 29 seconds ago Up 29 seconds 0.0.0.0:80->80/tcp, [::]:80->80/tcp nginx_balanceador
f6df2f9f6aa flotodor-apache-image:p2 "/usr/local/bin/entr..." 7 hours ago Up 7 hours 0.0.0.0:8087->80/tcp, [::]:8087->80/tcp web7
75188fdbded7 flotodor-nginx_web-image:p2 "/entrypoint.sh" 7 hours ago Up 7 hours 0.0.0.0:8086->80/tcp, [::]:8086->80/tcp web6
2a9186ff0d64 flotodor-apache-image:p2 "/usr/local/bin/entr..." 7 hours ago Up 7 hours 0.0.0.0:8084->80/tcp, [::]:8084->80/tcp web1
424d01045059 flotodor-nginx_web-image:p2 "/entrypoint.sh" 7 hours ago Up 7 hours 0.0.0.0:8081->80/tcp, [::]:8081->80/tcp web8
7b04923f6c17 flotodor-apache-image:p2 "/usr/local/bin/entr..." 7 hours ago Up 7 hours 0.0.0.0:8088->80/tcp, [::]:8088->80/tcp web3
7e8e52c879cd flotodor-nginx_web-image:p2 "/entrypoint.sh" 7 hours ago Up 7 hours 0.0.0.0:8083->80/tcp, [::]:8083->80/tcp web2
44b5b30133c1 flotodor-apache-image:p2 "/usr/local/bin/entr..." 7 hours ago Up 7 hours 0.0.0.0:8082->80/tcp, [::]:8082->80/tcp web5
```

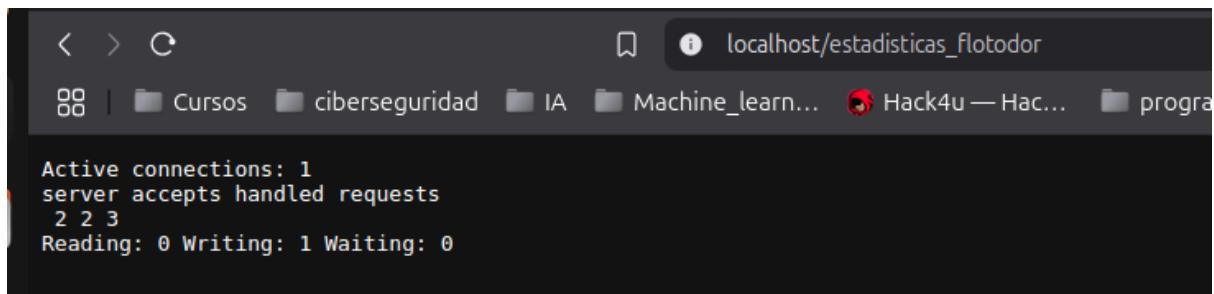
Podemos ver que los servidores web y el balanceador se ejecutan correctamente.





Por otro lado, podemos ver al entrar en localhost (al puerto 80 que es donde se ejecuta el balanceador de nginx, que gestiona de forma correcta el round-robin, debido a que cuando se recarga la página carga la siguiente web).

Por último, podemos ver las estadísticas básicas que nos ofrece nginx:



B5. Configuración de HAProxy como Balanceador de Carga:

En este caso, el dockerfile de HAproxy es el siguiente:

```

1  FROM haproxy:latest
2
3  # Asegura que tengamos permisos de root
4  USER root
5
6  # Instalación básica
7  RUN apt-get update && apt-get upgrade -y
8  RUN apt-get install -y net-tools iputils-ping iptables
9
10 # Copiamos la configuración
11 COPY ./P2-flotodor-haproxy/config_balanceador/haproxy.cfg /usr/local/etc/haproxy/haproxy.cfg
12
13 # Exponemos los puertos necesarios
14 EXPOSE 80
15 EXPOSE 9000
16
17 # Ejecutamos HAProxy en primer plano
18 CMD ["haproxy", "-f", "/usr/local/etc/haproxy/haproxy.cfg", "-db"]
19

```

Como se puede ver, debo de indicar el usuario ROOT debido a que si no hay problemas para la instalación de las herramientas. Por otro lado copiamos el haproxy.cfg y abrimos los puertos 80 y 9000, este último para las estadísticas del balanceador.

Para el haproxy.cfg tenemos lo siguiente:

```

1 global
2     stats socket /var/lib/haproxy/stats
3
4 defaults
5     mode http #el tráfico es gestionado por http
6     timeout connect 5000ms #tiempo máximo para conectar con el servidor
7     timeout client  50000ms #tiempo máximo para recibir datos del cliente
8     timeout server  50000ms #tiempo máximo para recibir datos del servidor
9
10 frontend flotodor
11     bind *:80 #puerto de escucha
12     default_backend backend_flotodor #backend por defecto
13
14 backend backend_flotodor
15     #Servidores web con maximo de 32 conexiones
16     #balanceo de carga roundrobin de forma predeterminada
17     server web1 192.168.10.2:80 maxconn 32 check
18     server web2 192.168.10.3:80 maxconn 32 check
19     server web3 192.168.10.4:80 maxconn 32 check
20     server web4 192.168.10.5:80 maxconn 32 check
21     server web5 192.168.10.6:80 maxconn 32 check
22     server web6 192.168.10.7:80 maxconn 32 check
23     server web7 192.168.10.8:80 maxconn 32 check
24     server web8 192.168.10.9:80 maxconn 32 check
25
26 listen stats
27     bind *:9000 #puerto de escucha para las estadísticas
28     mode http #modo http
29     stats enable #habilitar estadísticas
30     stats uri /estadisticas_flotodor #uri para acceder a las estadísticas
31     stats realm HAProxy\ Statistics
32     stats auth flotodor:SWAP1234 #usuario y contraseña para acceder a las estadísticas
33

```

Como podemos ver, contiene lo necesario que se nos pedía para la práctica básica.

B6. Implementación del escenario de HAProxy con Docker Compose

Para este caso, como anteriormente, creo un nuevo docker-compose en el cual incluyo las redes de forma externa y creo el haproxy como balanceador.

```

20  x-common-haproxy_balanceador-config: &common-haproxy_balanceador-config
21    image: flotodor-haproxy_balanceador-image:p2
22    restart: always
23    volumes:
24      - ./logs_haproxy:/var/log/haproxy
25

  ▶ Run Service
haproxy_balanceador:
  <<: *common-haproxy_balanceador-config
  container_name: haproxy_balanceador
  environment:
    - SERVER_NAME=haproxy_balanceador
  networks:
    red_web:
      ipv4_address: 192.168.10.50
  ports:
    - "80:80"
    - 9000:9000
  depends_on:
    - web1
    - web2
    - web3
    - web4
    - web5
    - web6
    - web7
    - web8

  networks:
    red_web:
      external: true
    red_servicios:
      external: true

```

Como he comentado anteriormente, omito la información irrelevante de las webs ya que es el mismo que el de la práctica anterior.

B7. Verificación y Pruebas del escenario de HAProxy

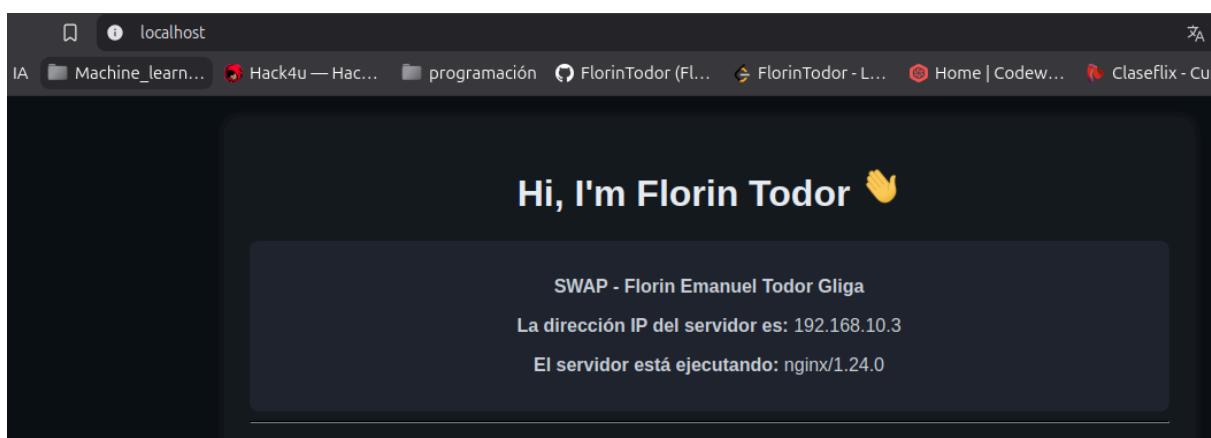
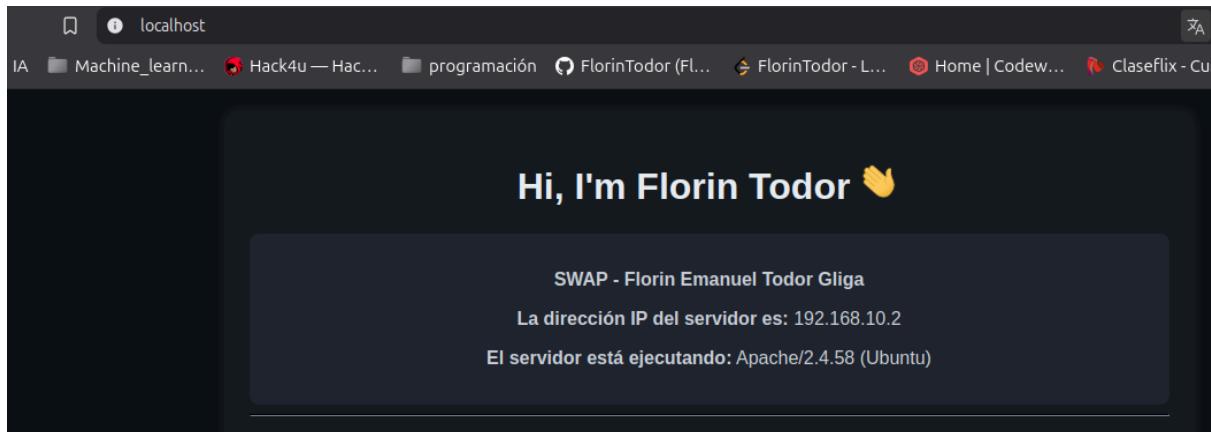
Lo primero que quería comentar es que en el script de init.sh que cree para automatizar todo el proceso en los trabajos, añadí que en el caso de que exista ya un balanceador en ejecución primero lo detecta, lo para y elimina y posteriormente se ejecuta el nuevo balanceador.

```
• > ./init.sh -u haproxy
[✓] Red red_servicios ya existe.
[✓] Red red_web ya existe.
[i] Comprobando puertos 8080 a 8089...
[!] Se detectó nginx_balanceador corriendo. Deteniéndolo...
[✓] nginx_balanceador detenido.
[+] Running 9/9
  ✓ Container web3           Running
  ✓ Container web4           Running
  ✓ Container web2           Running
  ✓ Container web6           Running
  ✓ Container web7           Running
  ✓ Container web8           Running
  ✓ Container web5           Running
  ✓ Container web1           Started
  ✓ Container haproxy_balanceador Started
[+] Servicios iniciados con HAProxy.
```

Podemos ver que todos los contenedores se ejecutan de forma correcta:

Servicios iniciados con HAProxy.					
CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
6aeadd41c0	flotodor-haproxy_balanceador-image:p2	"docker-entrypoint.s..."	About a minute ago	Up About a minute	0.0.0.0:80->80/tcp, [::]:80->80/tcp, 0.0.0.0:9000->9000/tcp, [::]:9000->9000/tcp
2e72887645	flotodor-apache-image:p2	"./usr/local/bin/entr..."	2 minutes ago	Up About a minute	0.0.0.0:8081->80/tcp, [::]:8081->80/tcp
d5f2f9f6aa	flotodor-apache-image:p2	"./usr/local/bin/entr..."	7 hours ago	Up 7 hours	0.0.0.0:8087->80/tcp, [::]:8087->80/tcp
188fdbded7	flotodor-nginx_web-image:p2	"/entrypoint.sh"	7 hours ago	Up 7 hours	0.0.0.0:8086->80/tcp, [::]:8086->80/tcp
9d92d63228	flotodor-nginx_web-image:p2	"/entrypoint.sh"	7 hours ago	Up 7 hours	0.0.0.0:8084->80/tcp, [::]:8084->80/tcp
web4	flotodor-nginx_web-image:p2	"/entrypoint.sh"	7 hours ago	Up 7 hours	0.0.0.0:8088->80/tcp, [::]:8088->80/tcp
4d01045059	flotodor-nginx_web-image:p2	"/entrypoint.sh"	7 hours ago	Up 7 hours	0.0.0.0:8088->80/tcp, [::]:8088->80/tcp
04923f6c17	flotodor-apache-image:p2	"./usr/local/bin/entr..."	7 hours ago	Up 7 hours	0.0.0.0:8083->80/tcp, [::]:8083->80/tcp
web3	flotodor-apache-image:p2	"./usr/local/bin/entr..."	7 hours ago	Up 7 hours	0.0.0.0:8083->80/tcp, [::]:8083->80/tcp
8e52c879cd	flotodor-nginx_web-image:p2	"/entrypoint.sh"	7 hours ago	Up 7 hours	0.0.0.0:8082->80/tcp, [::]:8082->80/tcp
web2	flotodor-apache-image:p2	"./usr/local/bin/entr..."	7 hours ago	Up 7 hours	0.0.0.0:8085->80/tcp, [::]:8085->80/tcp
b5b30133c1	flotodor-apache-image:p2	"/entrypoint.sh"	7 hours ago	Up 7 hours	0.0.0.0:8085->80/tcp, [::]:8085->80/tcp
web5					

Por otro lado, comprobamos la ejecución del balanceador con round-robin (que es el que se ejecuta de forma predeterminada)



Por otro lado vamos a ver las estadísticas que nos ofrece haproxy, que son mejores que las que nos ofrece nginx.

HAProxy version 3.1.6-d929ca2, released 2025/03/20																					
Statistics Report for pid 8																					
General process information																					
pid = 8 (process #1, nbproc: 1, listenfd = 16) optime = 2 (06/06/2025), warnings = 0 maxconn = 1048576, maxqueue = 1048576, maxclients = 1048576 maxsocks = 1048576, maxxconn = 524248, reached = 0, maxpipes = 0 Uptime = 0 days, 0 hours, 0 minutes, 0 seconds (since Mon Mar 20 10:00:00 2025 (MANT)) active or backup DOWN by default (MANT) active or backup SOFT STOPPED for maintenance Running tasks: 0/54 (0 ready; idle = 100 %)																					
Note: "NOBYDRAN" = UP with load-balancing disabled.																					
Display option:																					
• Scope: Primary site • Hide DOWN services • Database now • Cache now • SSL/TLS export (schema)																					
Frontend		Queue	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	In	Bytes	Denied	Errors	Warnings	Server	LastChk	Wight	Act	
Frontend		Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	LbTot	Last	In	Out	Req	Conn	Redis	Bck	Cchk	Dwn	Thrtie
Frontend		0	0	-	0	1	-	0	1	32	3	1m24s	1 706	5 725	0	0	0	0	0	0	0s
Backend		0	0	-	0	1	-	0	1	32	3	1m3s	1 685	8 403	0	0	0	0	0	0	-
Backend		0	0	-	0	1	-	0	1	32	2	1m24s	1 706	5 725	0	0	0	0	0	0	-
Backend		0	0	-	0	1	-	0	1	32	2	1m24s	1 706	5 725	0	0	0	0	0	0	-
Backend		0	0	-	0	1	-	0	1	32	2	1m24s	1 706	5 725	0	0	0	0	0	0	-
Backend		0	0	-	0	1	-	0	1	32	2	1m24s	1 706	5 725	0	0	0	0	0	0	-
Backend		0	0	-	0	1	-	0	1	32	2	1m24s	1 706	5 725	0	0	0	0	0	0	-
Backend		0	0	-	0	1	-	0	1	32	2	1m24s	1 706	5 725	0	0	0	0	0	0	-
Backend		0	0	-	0	1	-	0	1	32	2	1m24s	1 706	5 725	0	0	0	0	0	0	-
Backend		0	0	-	0	1	-	0	1	32	2	1m24s	1 706	5 725	0	0	0	0	0	0	-
Backend		0	0	-	0	1	-	0	1	32	2	1m24s	1 706	5 725	0	0	0	0	0	0	-
Backend		0	0	-	0	1	-	0	1	32	2	1m24s	1 706	5 725	0	0	0	0	0	0	-
Backend		0	0	-	0	1	-	0	1	32	2	1m24s	1 706	5 725	0	0	0	0	0	0	-
Backend		0	0	-	0	1	-	0	1	32	2	1m24s	1 706	5 725	0	0	0	0	0	0	-
Backend		0	0	-	0	1	-	0	1	32	2	1m24s	1 706	5 725	0	0	0	0	0	0	-
Backend		0	0	-	0	1	-	0	1	32	2	1m24s	1 706	5 725	0	0	0	0	0	0	-
Backend		0	0	-	0	1	-	0	1	32	2	1m24s	1 706	5 725	0	0	0	0	0	0	-
Backend		0	0	-	0	1	-	0	1	32	2	1m24s	1 706	5 725	0	0	0	0	0	0	-
Backend		0	0	-	0	1	-	0	1	32	2	1m24s	1 706	5 725	0	0	0	0	0	0	-
Backend		0	0	-	0	1	-	0	1	32	2	1m24s	1 706	5 725	0	0	0	0	0	0	-
Backend		0	0	-	0	1	-	0	1	32	2	1m24s	1 706	5 725	0	0	0	0	0	0	-
Backend		0	0	-	0	1	-	0	1	32	2	1m24s	1 706	5 725	0	0	0	0	0	0	-
Backend		0	0	-	0	1	-	0	1	32	2	1m24s	1 706	5 725	0	0	0	0	0	0	-
Backend		0	0	-	0	1	-	0	1	32	2	1m24s	1 706	5 725	0	0	0	0	0	0	-
Backend		0	0	-	0	1	-	0	1	32	2	1m24s	1 706	5 725	0	0	0	0	0	0	-
Backend		0	0	-	0	1	-	0	1	32	2	1m24s	1 706	5 725	0	0	0	0	0	0	-
Backend		0	0	-	0	1	-	0	1	32	2	1m24s	1 706	5 725	0	0	0	0	0	0	-
Backend		0	0	-	0	1	-	0	1	32	2	1m24s	1 706	5 725	0	0	0	0	0	0	-
Backend		0	0	-	0	1	-	0	1	32	2	1m24s	1 706	5 725	0	0	0	0	0	0	-
Backend		0	0	-	0	1	-	0	1	32	2	1m24s	1 706	5 725	0	0	0	0	0	0	-
Backend		0	0	-	0	1	-	0	1	32	2	1m24s	1 706	5 725	0	0	0	0	0	0	-
Backend		0	0	-	0	1	-	0	1	32	2	1m24s	1 706	5 725	0	0	0	0	0	0	-
Backend		0	0	-	0	1	-	0	1	32	2	1m24s	1 706	5 725	0	0	0	0	0	0	-
Backend		0	0	-	0	1	-	0	1	32	2	1m24s	1 706	5 725	0	0	0	0	0	0	-
Backend		0	0	-	0	1	-	0	1	32	2	1m24s	1 706	5 725	0	0	0	0	0	0	-
Backend		0	0	-	0	1	-	0	1	32	2	1m24s	1 706	5 725	0	0	0	0	0	0	-
Backend		0	0	-	0	1	-	0	1	32	2	1m24s	1 706	5 725	0	0	0	0	0	0	-
Backend		0	0	-	0	1	-	0	1	32	2	1m24s	1 706	5 725	0	0	0	0	0	0	-
Backend		0	0	-	0	1	-	0	1	32	2	1m24s	1 706	5 725	0	0	0	0	0	0	-
Backend		0	0	-	0	1	-	0	1	32	2	1m24s	1 706	5 725	0	0	0	0	0	0	-
Backend		0	0	-	0	1	-	0	1	32	2	1m24s	1 706	5 725	0	0	0	0	0	0	-
Backend		0	0	-	0	1	-	0	1	32	2	1m24s	1 706	5 725	0	0	0	0	0	0	-
Backend		0	0	-	0	1	-	0	1	32	2	1m24s	1 706	5 725	0	0	0	0	0	0	-
Backend		0	0	-	0	1	-	0	1	32	2	1m24s	1 706	5 725	0	0	0	0	0	0	-
Backend		0	0	-	0	1	-	0	1	32	2	1m24s	1 706	5 725	0	0	0	0	0	0	-
Backend		0	0	-	0	1	-	0	1	32	2	1m24s	1 706	5 725	0	0	0	0	0	0	-
Backend		0	0	-	0	1	-	0	1	32	2	1m24s	1 706	5 725	0	0	0	0	0	0	-
Backend		0	0	-	0	1	-	0	1	32	2	1m24s	1 706	5 725	0	0	0	0	0	0	-
Backend		0	0	-	0	1	-	0	1	32	2	1m24s	1 706	5 725	0	0	0	0	0	0	-
Backend		0	0	-	0	1	-	0	1	32	2	1m24s	1 706	5 725	0	0	0	0	0	0	-
Backend		0	0	-	0	1	-	0	1	32	2	1m24s	1 706	5 725	0	0	0	0	0	0	-
Backend		0	0	-	0	1	-	0	1	32	2	1m24s	1 706	5 725	0	0	0	0	0	0	-
Backend		0	0	-	0	1	-	0	1	32	2	1m24s	1 706	5 725	0	0	0	0	0	0	-
Backend		0	0	-	0	1	-	0	1	32	2	1m24s	1 706	5 725	0	0	0	0	0	0	-
Backend		0	0	-	0	1	-	0	1	32	2	1m24s	1 706	5 725	0	0	0	0	0	0	-
Backend		0	0	-	0	1	-														

Tareas Avanzadas

Para esta parte, va a haber varios cambios en los ficheros, script, etc.

Por ello, comentar que si hay algo que no se parece tal cual a las imágenes anteriores se debe a estos cambios. Igualmente, los iré comentando.

A1: Configuraciones Avanzadas de Nginx

Para esta parte, voy a modificar el script de init.sh para tener de forma predeterminada (sin incluir argumento) round-robin y en el caso de que se añada el parámetro de “pd” (ponderado), se hará con estrategia basada en ponderación. También crearé el parámetro “rb” de round-robin.

Posteriormente analizaré el impacto de esta configuración (la estrategia de tiempo de respuesta la implementaré para HAProxy). He estado buscado implementar la estrategia de menor tiempo de respuesta pero se solicita NGINX PLUS, podría implementar el de menor número de conexiones (least_conn).

Por lo tanto, para el **balanceador nginx** voy a usar **round-robin y ponderación**.

Para eso he creado dos archivos de configuración de nginx (aunque luego se crea el nginx.conf para seguir la nomenclatura).

Para la **estrategia de round-robin**, tenemos este fichero de configuración (**nginx_rb.conf**):

```
upstream backend_flotodor {
    # por defecto es round-robin
    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;
}
```

Al ejecutar el script con esta estrategia:

```
• > ./init.sh -u nginx rb
[✓] Red red_servicios ya existe.
[✓] Red red_web ya existe.
[i] Estrategia de balanceo: round-robin (por defecto)
[i] Comprobando puertos 8080 a 8089...
[+] Running 9/9
✓ Container web1           Started
✓ Container web6           Started
✓ Container web7           Started
✓ Container web8           Started
✓ Container web4           Started
✓ Container web3           Started
✓ Container web5           Started
✓ Container web2           Started
✓ Container nginx_balanceador Started
[+] Servicios iniciados con Nginx.
```

Posteriormente hacemos una prueba de acceso al balanceador:

```
• > for i in {1..2000}; do curl -s http://localhost | grep "192.168" >> log.txt; done
cat log.txt | sort | uniq -c
250      <p><strong>La dirección IP del servidor es:</strong> 192.168.10.2</p>
250      <p><strong>La dirección IP del servidor es:</strong> 192.168.10.3</p>
250      <p><strong>La dirección IP del servidor es:</strong> 192.168.10.4</p>
250      <p><strong>La dirección IP del servidor es:</strong> 192.168.10.5</p>
250      <p><strong>La dirección IP del servidor es:</strong> 192.168.10.6</p>
250      <p><strong>La dirección IP del servidor es:</strong> 192.168.10.7</p>
250      <p><strong>La dirección IP del servidor es:</strong> 192.168.10.8</p>
250      <p><strong>La dirección IP del servidor es:</strong> 192.168.10.9</p>
```

Como podemos ver, se ejecutan todas las webs de forma igualitaria con el round-robin.

Para el caso de **Ponderación** tenemos:

En su fichero de configuración (**nginx_pd.conf**):

```
upstream backend_flotodor {
    server 192.168.10.2 weight=5;
    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=1;
    server 192.168.10.7 weight=1;
    server 192.168.10.8 weight=1;
    server 192.168.10.9 weight=1;
}
```

Al ejecutarlo tenemos :

```
● > ./init.sh -u nginx pd
[✓] Red red_servicios ya existe.
[✓] Red red_web ya existe.
[i] Estrategia de balanceo: ponderación con pesos
[i] Comprobando puertos 8080 a 8089...
[+] Running 9/9
✓ Container web5           Started
✓ Container web2           Started
✓ Container web4           Started
✓ Container web8           Started
✓ Container web1           Started
✓ Container web6           Started
✓ Container web3           Started
✓ Container web7           Started
✓ Container nginx_balanceador Started
[+] Servicios iniciados con Nginx.
```

```
● > for i in {1..2000}; do curl -s http://localhost | grep "192.168" >> log.txt; done
cat log.txt | sort | uniq -c
 264      <p><strong>La dirección IP del servidor es:</strong> 192.168.10.2</p>
 248      <p><strong>La dirección IP del servidor es:</strong> 192.168.10.3</p>
 248      <p><strong>La dirección IP del servidor es:</strong> 192.168.10.4</p>
 248      <p><strong>La dirección IP del servidor es:</strong> 192.168.10.5</p>
 248      <p><strong>La dirección IP del servidor es:</strong> 192.168.10.6</p>
 248      <p><strong>La dirección IP del servidor es:</strong> 192.168.10.7</p>
 248      <p><strong>La dirección IP del servidor es:</strong> 192.168.10.8</p>
 248      <p><strong>La dirección IP del servidor es:</strong> 192.168.10.9</p>
```

Como podemos comprobar, el servidor web con ip .2, que es el que tiene peso 5, es el que tiene más accesos, a diferencia de los demás que tienen los mismos accesos.

A2: Configuraciones Avanzadas de HAProxy

Para este balanceador voy a implementar la estrategia **menor número de conexiones** (la cual la he llamado con “lc”).

Como sabemos, el menor número de conexiones, es el que normalmente es el más rápido en ese momento en responder (comento esto porque no encuentro en ningún lado la estrategia de menor tiempo de respuesta).

Para esta estrategia tenemos el siguiente archivo de configuración (**haproxy_lc.cfg**)

```
backend backend_flotodor
    #Servidores web con maximo de 32 conexiones
    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
```

Al ejecutar el script tenemos:

```
[!] Estrategia desconfigurada: rr. Usd cc 0 10 .
• > ./init.sh -u haproxy lc
[✓] Red red_servicios ya existe.
[✓] Red red_web ya existe.
[i] Estrategia de balanceo: menor número de conexiones
[i] Comprobando puertos 8080 a 8089...
[!] haproxy_balanceador ya está corriendo. Reiniciándolo para aplicar nueva estrategia...
[✓] haproxy_balanceador reiniciado con la estrategia lc.
[+] Running 9/9
✓ Container web4          Running
✓ Container web3          Running
✓ Container web5          Running
✓ Container web6          Running
✓ Container web2          Running
✓ Container web7          Running
✓ Container web8          Running
✓ Container web1          Running
✓ Container haproxy_balanceador Started
[+] Servicios iniciados con HAProxy.
```

Vemos en este caso que se ejecuta en este caso con una salida similar a la de round-robin

```
• > for i in {1..2000}; do curl -s http://localhost | grep "192.168" >> log.txt; done
cat log.txt | sort | uniq -c
250      <p><strong>La dirección IP del servidor es:</strong> 192.168.10.2</p>
250      <p><strong>La dirección IP del servidor es:</strong> 192.168.10.3</p>
250      <p><strong>La dirección IP del servidor es:</strong> 192.168.10.4</p>
250      <p><strong>La dirección IP del servidor es:</strong> 192.168.10.5</p>
250      <p><strong>La dirección IP del servidor es:</strong> 192.168.10.6</p>
250      <p><strong>La dirección IP del servidor es:</strong> 192.168.10.7</p>
250      <p><strong>La dirección IP del servidor es:</strong> 192.168.10.8</p>
250      <p><strong>La dirección IP del servidor es:</strong> 192.168.10.9</p>
```

Por otro lado tenemos, el archivo de configuración de round-robin (**haproxy_rb.cfg**):

```
backend backend_flotodor
    #Servidores web con maximo de 32 conexiones
    #balanceo de carga roundrobin de forma predeterminada
    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
```

Al ejecutarlo tenemos:

```
• > ./init.sh -u haproxy rb
[✓] Red red_servicios ya existe.
[✓] Red red_web ya existe.
[i] Estrategia de balanceo: round-robin (por defecto)
[i] Comprobando puertos 8080 a 8089...
[!] haproxy_balanceador ya está corriendo. Reiniciándolo para aplicar nueva estrategia...
[✓] haproxy_balanceador reiniciado con la estrategia rb.
[+] Running 9/9
  ✓ Container web3           Running
  ✓ Container web8           Running
  ✓ Container web7           Running
  ✓ Container web1           Running
  ✓ Container web2           Running
  ✓ Container web4           Running
  ✓ Container web6           Running
  ✓ Container web5           Running
  ✓ Container haproxy_balanceador Started
[+] Servicios iniciados con HAProxy.
```

La salida es la misma que en el caso anterior.

```
• > for i in {1..2000}; do curl -s http://localhost | grep "192.168" >> log.txt; done
cat log.txt | sort | uniq -c
  250 <p><strong>La dirección IP del servidor es:</strong> 192.168.10.2</p>
  250 <p><strong>La dirección IP del servidor es:</strong> 192.168.10.3</p>
  250 <p><strong>La dirección IP del servidor es:</strong> 192.168.10.4</p>
  250 <p><strong>La dirección IP del servidor es:</strong> 192.168.10.5</p>
  250 <p><strong>La dirección IP del servidor es:</strong> 192.168.10.6</p>
  250 <p><strong>La dirección IP del servidor es:</strong> 192.168.10.7</p>
  250 <p><strong>La dirección IP del servidor es:</strong> 192.168.10.8</p>
  250 <p><strong>La dirección IP del servidor es:</strong> 192.168.10.9</p>
```

Breve inciso

He de comentar que he realizado varias modificaciones en mi script principal init.sh:

Método añadido para crear las redes de forma externa, para evitar errores comentados en clase:

```
# Función para crear redes si no existen, para evitar conflictos de IP
# Se crean dos redes: red_web y red_servicios
function ensure_networks() {
    declare -A redes=
    ["red_web"]="192.168.10.0/24"
    ["red_servicios"]="192.168.20.0/24"
}
for red in "${!redes[@]}"; do
    if ! docker network ls --format '{{.Name}}' | grep -q "^${red}$"; then
        docker network create --driver bridge --subnet "${redes[$red]}" "$red" &>/dev/null
        echo -e "${greenColour}[\u2713] Red ${red} creada correctamente.${endColour}"
    else
        echo -e "${blueColour}[\u2713] Red ${red} ya existe.${endColour}"
    fi
done
}
```

Funciones para establecer las estrategias de balanceo en los balanceadores de nginx y haproxy

```
# Función para establecer la estrategia de balanceo de carga en HAProxy
# Se copia el archivo de configuración correspondiente a la estrategia seleccionada
# y se muestra un mensaje indicando la estrategia utilizada
# Se espera que el archivo de configuración esté en la ruta ./P2-flotodor-haproxy/config_balanceador/
#Las estrategias disponibles son: menor tiempo de respuesta (tr) y round-robin (rb) y por defecto se usa round-robin
function set_haproxy_strategy() {
    local strategy=$1
    case "$strategy" in
        tr)
            cp ./P2-flotodor-haproxy/config_balanceador/haproxy_tr.cfg ./P2-flotodor-haproxy/config_balanceador/haproxy.cfg
            echo -e "${blueColour}[i] Estrategia de balanceo: menor tiempo de respuesta${endColour}"
            ;;
        rb|*)
            cp ./P2-flotodor-haproxy/config_balanceador/haproxy_rb.cfg ./P2-flotodor-haproxy/config_balanceador/haproxy.cfg
            echo -e "${blueColour}[i] Estrategia de balanceo: round-robin (por defecto)${endColour}"
            ;;
        *)
            echo -e "${redColour}(!) Estrategia desconocida: $strategy. Usa 'tr' o 'rb'.${endColour}"
            return 1
            ;;
    esac
}

# Función para establecer la estrategia de balanceo de carga en Nginx
# Se copia el archivo de configuración correspondiente a la estrategia seleccionada
# y se muestra un mensaje indicando la estrategia utilizada
# Se espera que el archivo de configuración esté en la ruta ./P2-flotodor-nginx/config_balanceador/
#Las estrategias disponibles son: ponderación (pd) y round-robin (rb) y por defecto se usa round-robin
function set_nginx_strategy() {
    local strategy=$1
    case "$strategy" in
        pd)
            cp ./P2-flotodor-nginx/config_balanceador/nginx_pd.conf ./P2-flotodor-nginx/config_balanceador/nginx.conf
            echo -e "${blueColour}[i] Estrategia de balanceo: ponderación con pesos${endColour}"
            ;;
        rb|*)
            cp ./P2-flotodor-nginx/config_balanceador/nginx_rb.conf ./P2-flotodor-nginx/config_balanceador/nginx.conf
            echo -e "${blueColour}[i] Estrategia de balanceo: round-robin (por defecto)${endColour}"
            ;;
        *)
            echo -e "${redColour}(!) Estrategia desconocida: $strategy. Usa 'rb' o 'pd'.${endColour}"
            return 1
            ;;
    esac
}
```

Método para poder cambiar de balanceador en mitad de la ejecución (para no tener que hacerlo de forma manual)

```
# Función para detener balanceadores en conflicto
# Se verifica si hay un balanceador de tipo Nginx o HAProxy en ejecución y se detiene en el caso de que el usuario quiera cambiar de balanceador con este script
function stop_conflicting_balanceador() {
    local type=$1
    if [ "$type" == "nginx" ] && docker ps --format '{{.Names}}' | grep -q "^haproxy_balanceador$"; then
        echo -e "${yellowColour}(!) Se detectó haproxy_balanceador corriendo. Deteniéndolo...${endColour}"
        docker stop haproxy_balanceador &>/dev/null
        docker rm haproxy_balanceador &>/dev/null
        echo -e "${greenColour}[+] haproxy_balanceador detenido.${endColour}"
    elif [ "$type" == "haproxy" ] && docker ps --format '{{.Names}}' | grep -q "^nginx_balanceador$"; then
        echo -e "${yellowColour}(!) Se detectó nginx_balanceador corriendo. Deteniéndolo...${endColour}"
        docker stop nginx_balanceador &>/dev/null
        docker rm nginx_balanceador &>/dev/null
        echo -e "${greenColour}[+] nginx_balanceador detenido.${endColour}"
    fi
}
```

Método para inicializar los dos balanceadores (de momento):

```
# Función para iniciar el balanceador de carga
# Se utiliza docker compose para levantar el balanceador de carga especificado, esto se utiliza para cuando el usuario quiere cambiar de tipo de balanceador en mitad de la ejecución
function start_balanceador() {
    local type=$1
    case $type in
        nginx)
            docker compose -f docker-compose_nginx_balanceador.yaml up -d --remove-orphans
            echo -e "${greenColour}[+] Servicios iniciados con Nginx.${endColour}"
            ;;
        haproxy)
            docker compose -f docker-compose_haproxy_balanceador.yaml up -d --remove-orphans
            echo -e "${greenColour}[+] Servicios iniciados con HAProxy.${endColour}"
            ;;
        *)
            echo -e "${redColour}(!) Especifica el tipo de balanceador: nginx o haproxy${endColour}"
            return 1
    esac
}
```

Métodos para cambiar de estrategia de balanceo en mitad de la ejecución:

```
# Función para reiniciar el balanceador de carga Nginx si ya está en ejecución
# Se detiene y elimina el contenedor existente y se inicia uno nuevo con la nueva estrategia
# Se utiliza para cambiar de estrategia de balanceo, con balanceador nginx.
# Por ello primero tenemos que pararlo y eliminarlo, para luego volver a levantarlo, con la nueva estrategia
function force_restart_balanceador_si_nginx() {
    local type=$1
    local strategy=$2

    if [ $strategy == "" ]; then
        strategy="rb"
    fi
    if [ "$type" == "nginx" ] && docker ps --format '{{.Names}}' | grep -q "^nginx_balanceador$"; then
        echo -e "${yellowColour}(!) nginx_balanceador ya está corriendo. Reiniciándolo para aplicar nueva estrategia...${endColour}"
        docker stop nginx_balanceador &>/dev/null
        docker rm nginx_balanceador &>/dev/null
        echo -e "${greenColour}[+] nginx_balanceador reiniciado con la estrategia ${strategy}.${endColour}"
    fi
}

# Función para reiniciar el balanceador de carga HAProxy si ya está en ejecución
# Se detiene y elimina el contenedor existente y se inicia uno nuevo con la nueva estrategia
# Se utiliza para cambiar de estrategia de balanceo, con balanceador haproxy.
# Por ello primero tenemos que pararlo y eliminarlo, para luego volver a levantararlo, con la nueva estrategia
function force_restart_balanceador_si_haproxy() {
    local type=$1
    local strategy=$2

    if [ $strategy == "" ]; then
        strategy="rb"
    fi
    if [ "$type" == "haproxy" ] && docker ps --format '{{.Names}}' | grep -q "^haproxy_balanceador$"; then
        echo -e "${yellowColour}(!) haproxy_balanceador ya está corriendo. Reiniciándolo para aplicar nueva estrategia...${endColour}"
        docker stop haproxy_balanceador &>/dev/null
        docker rm haproxy_balanceador &>/dev/null
        echo -e "${greenColour}[+] haproxy_balanceador reiniciado con la estrategia ${strategy}.${endColour}"
    fi
}
```

Por último, tenemos el método principal para ejecutar el docker compose up:

```
#Función principal para ejecutar docker compose up
# Se asegura de que las redes necesarias estén creadas y disponibles
# Se comprueba la disponibilidad de los puertos y se detienen平衡adores en conflicto
# Se inicia el balanceador de carga especificado (nginx o haproxy)
# Se utiliza para levantar el balanceador de carga y los contenedores web
function compose_up() {
    local type=$1
    local strategy=$2

    ensure_networks

    if [ "$type" == "nginx" ]; then
        set_nginx_strategy "$strategy" || return
    fi

    if [ "$type" == "haproxy" ]; then
        set_haproxy_strategy "$strategy" || return
    fi

    check_ports_availability || {
        echo -e "${redColour}[X] Algunos puertos están ocupados. Aborta ejecución de docker compose up.${endColour}"
        return
    }

    stop_conflicting_balanceador "$type"
    force_restart_balanceador_si_nginx "$type" "$strategy"
    force_restart_balanceador_si_haproxy "$type" "$strategy"
    start_balanceador "$type"
}
```

He realizado este breve inciso debido a que me ha parecido relevante tener toda esta información modularizada para evitar posibles errores en un futuro, además de mejorar la legibilidad.

Sin embargo, seguramente **seguirán habiendo más cambios** en este script en esta práctica y en las siguientes, por lo que se recomienda leer el script y sus comentarios.

En resumen actualmente podemos hacer lo siguiente:

```
• ➤ ./init.sh
[+] Uso:
    -c Limpiar archivos dentro de logs_apache y logs_nginx
    -s Detener y eliminar contenedores (apache/nginx/nginx_balanceador/haproxy_balanceador/all)
    -b Crear imagen docker (apache/nginx/nginx_balanceador/haproxy_balanceador/all)
    -u Ejecutar docker compose up para el balanceador (nginx/haproxy)
        Puedes indicar una estrategia de balanceo para los siguientes balanceadores:
            nginx:
                pd = ponderación con pesos
                rb = round-robin (por defecto)
            haproxy:
                lc = menor número de conexiones
                rb = round-robin (por defecto)
    -p Actualizar paquetes dentro de los contenedores activos
    -h Mostrar este panel de ayuda
```

A3.Experimentación con Diferentes Balanceadores de Carga

En primer lugar voy a implementar el balanceador de carga de Traefik.

Traefik: <https://doc.traefik.io/traefik/routing/overview/>

Traefik

Debemos de saber que se divide en varias partes:

- **EntryPoints:** Se trata de los puertos que se encuentran en escucha, es decir, indicamos por donde van a escuchar tanto el balanceador como el dashboard.

```
entryPoints:
  web:
    address: ":80"
  traefik:
    address: ":8080"
```

- **Routers:** Se encarga de dirigir cada petición a uno de los servicios, puede utilizar los middlewares.

```
root@flotodor-traefik: ~ ! dynamic.yaml
1 http:
2   routers:
3     flotodor-router:
4       rule: "PathPrefix(`/`) && !PathPrefix(`/dashboard`) && !PathPrefix(`/api`)"
5       entryPoints:
6         - web
7       service: flotodor-service
8
9     traefik-dashboard:
10      rule: "PathPrefix(`/dashboard`) || PathPrefix(`/api`)"
11      entryPoints:
12        - traefik
13      service: api@internal
14      middlewares:
15        - auth
16
17   middlewares:
```

- **Services:** Son el grupo de servidores/contenedores a los que se reenvía el tráfico

```
services:
  flotodor-service:
    loadBalancer:
      passHostHeader: true
    servers:
      - url: "http://192.168.10.2"
      - url: "http://192.168.10.3"
      - url: "http://192.168.10.4"
      - url: "http://192.168.10.5"
      - url: "http://192.168.10.6"
      - url: "http://192.168.10.7"
      - url: "http://192.168.10.8"
      - url: "http://192.168.10.9"
```

- **Middlewares:** Son un tipo de “filtro” que se ejecuta entre el router y los servidores/servicios.

```
middlewares:
  auth:
    basicAuth:
      users:
        - "flotodor:$2y$05$x/9oT3Amok6W8YXmt05HX0yhxQ5uEoMdqBd.Er2NkTITeQpToj1b."
```

En este caso he usado un middleware para colocar una autenticación para entrar en el dashboard (igual que hicimos en haproxy). El usuario y contraseña es lo mismo que en HAproxy.

- **Providers:** Es la forma en la que traefik consigue su configuración dinámica

```
providers:
  file:
    filename: /etc/traefik/dynamic.yaml
    watch: true
```

Esto se debe a que tenemos un fichero que es estático (traefik.yaml) y otro que es dinámico (dynamic.yaml). El primero se carga en la imagen y el otro va variando.

Hay que indicar que he creado un dockerfile para este balanceador:

```
1 FROM traefik:latest
2
3 # Herramientas útiles para pruebas dentro del contenedor (opcional)
4 RUN apk add --no-cache iputils net-tools iproute2 curl
5
6 # Exponer puertos necesarios
7 EXPOSE 80
8 EXPOSE 8080
9
10 # Ejecutar Traefik (el archivo será montado)
11 CMD ["traefik", "--configFile=/etc/traefik/traefik.yaml"]
12
```

Y un docker-compose que es como los demás pero con la información de este balanceador

```
x-common-traefik-config: &common-traefik-config
  build:
    context: .
    dockerfile: ./P2-flotodor-traefik/DockerfileTraefik_balanceador
    restart: always
  volumes:
    - ./logs_traefik:/var/log/traefik
    - /var/run/docker.sock:/var/run/docker.sock
    - ./P2-flotodor-traefik/traefik.yaml:/etc/traefik/traefik.yaml:ro
    - ./P2-flotodor-traefik/dynamic.yaml:/etc/traefik/dynamic/dynamic.yaml:ro
```

```

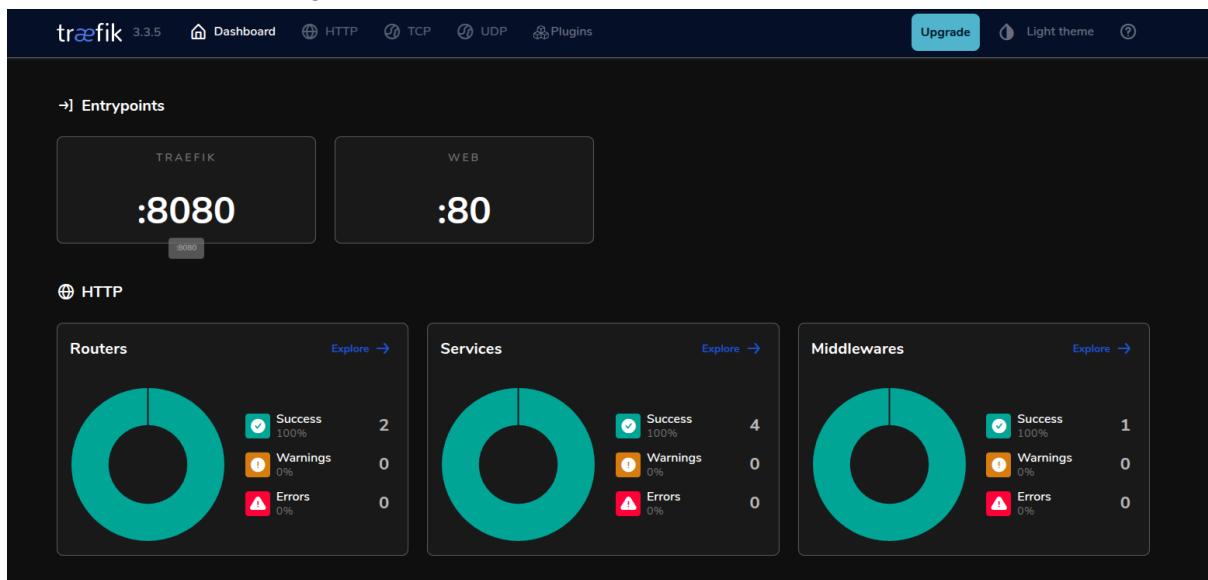
traefik_balanceador:
  <<: *common-traefik-config
  container_name: traefik_balanceador
  networks:
    red_web:
      ipv4_address: 192.168.10.50
  ports:
    - "80:80"
    - "8080:8080"
  depends_on:
    - web1
    - web2
    - web3
    - web4
    - web5
    - web6
    - web7
    - web8

```

Por último, podemos comprobar la información que nos ofrece el dashboard de este balanceador.

Para ello accedemos a <http://localhost:8080>

En el podemos ver lo siguiente:



Solo muestro la información del http porque no hacemos uso del resto de protocolos.

También podemos ver que podemos instalar plugins, los cuales estarían muy interesantes para la gestión de los servidores webs.

Status	TLS	Rule	Entrypoints	Name	Service	Provider	Priority
		PathPrefix('') && !PathPrefix('/dashboard') && !PathPrefix('/api')	web	flotodor-router@file	flotodor-service		67
		PathPrefix('/dashboard') PathPrefix('/api')	traefik	traefik-dashboard@file	api@internal		46

Envoy

Envoy: <https://gobetween.io/documentation.html#Configuration>

También he decidido preparar otro balanceador de carga, al principio había utilizado gobetween pero no tenía interfaz gráfica.

Aunque Envoy tampoco tiene tal cual una UI para representar las gráficas como en el caso de Traefik, permite coger las estadísticas en formato de prometheus para poder usar posteriormente Grafana.

Aunque no implemente ahora el uso de Prometheus y Grafana, me parece interesante tenerlo preparado por si algún día me pongo a hacerlo.

Para esta parte lo mismo, creación de docker-file para envoy, docker-compose, etc.

El archivo de configuración para envío es el siguiente:

```
1 static_resources:
2   listeners:
3     - name: listener_http
4       address:
5         socket_address: { address: 0.0.0.0, port_value: 80 }
6       filter_chains:
7         - filters:
8           - name: envoy.filters.network.http_connection_manager
9             typed_config:
10               "@type": type.googleapis.com/envoy.extensions.filters.network.http_connection_manager.v3.HttpConnectionManager
11               stat_prefix: ingress_http
12               access_log:
13                 - name: envoy.access_loggers.stdout
14                   typed_config:
15                     "@type": type.googleapis.com/envoy.extensions.access_loggers.stream.v3.StdoutAccessLog
16               route_config:
17                 name: local_route
18                 virtual_hosts:
19                   - name: flotodor_service
20                     domains: [""]
21                     routes:
22                       - match: { path: "/" }
23                         route:
24                           cluster: flotodor_cluster
25             http_filters:
26               - name: envoy.filters.http.router
27                 typed_config:
28                   "@type": type.googleapis.com/envoy.extensions.filters.http.router.v3.Router
29
30 clusters:
31   - name: flotodor_cluster
32     connect_timeout: 0.25s
33     type: strict_dns
34     lb_policy: round_robin
35     load_assignment:
36       cluster_name: flotodor_cluster
37       endpoints:
38         - lb_endpoints:
39           - endpoint: { address: { socket_address: { address: 192.168.10.2, port_value: 80 } } }
40           - endpoint: { address: { socket_address: { address: 192.168.10.3, port_value: 80 } } }
41           - endpoint: { address: { socket_address: { address: 192.168.10.4, port_value: 80 } } }
42           - endpoint: { address: { socket_address: { address: 192.168.10.5, port_value: 80 } } }
43
44
45
46
47
48 admin:
49   access_log_path: "/dev/null"
50   address:
51     socket_address:
52       address: 0.0.0.0
53       port_value: 9901
```

Como podemos ver, defino un listener, el cual escucha en todas las interfaces y por el puerto 80 y para el cual configuro el filtro de http con ayuda de la IA.

Por otro lado defino los clusters, que conforman los servidores web de la práctica.

Por último, habilito la interfaz administrativa en el puerto 9901.

Las estadísticas las podemos ver en este interfaz, las cuales son las siguientes:

Command	Description
certs	print certs on machine
clusters	upstream cluster status
config_dump	dump current Envoy configs (experimental)
<input type="text"/>	The resource to dump
<input type="text"/>	The mask to apply. When both resource and mask are specified, the mask is applied to ev
<input type="text"/>	Dump only the currently loaded configurations whose names match the specified regex. C
<input type="checkbox"/>	Dump currently loaded configuration including EDS. See the response definition for more
contention	dump current Envoy mutex contention stats (if enabled)
cpuprofiler	enable/disable the CPU profiler
<input checked="" type="checkbox"/>	enables the CPU profiler
drain_listeners	drain listeners
<input type="checkbox"/>	When draining listeners, enter a graceful drain period prior to closing listeners. This behav
<input type="checkbox"/>	When draining listeners, do not exit after the drain period. This must be used with gracefu
<input type="checkbox"/>	Drains all inbound listeners. traffic_direction field in envoy_v3_api_msg_config.listener.v3.
healthcheck/fail	cause the server to fail health checks
healthcheck/ok	cause the server to pass health checks
heap_dump	dump current Envoy heap (if supported)
heapprofiler	enable/disable the heap profiler
<input checked="" type="checkbox"/>	enable/disable the heap profiler
help	print out list of admin commands
hot_restart_version	print the hot restart compatibility version
init_dump	dump current Envoy init manager information (experimental)
<input type="text"/>	The desired component to dump unready targets. The mask is parsed as a ProtobufWkt::I
listeners	print listener info
<input type="text"/> <input checked="" type="button"/>	File format to use
logging	query/change logging levels
<input type="text"/>	Change multiple logging levels by setting to <logger_name1>:<desired_level1>,<logger_r
<input type="text"/>	desired logging level
memory	print current allocation/heap usage
quitquitquit	exit the server

Indicar como antes que he realizado modificaciones en el script principal de init.sh, por ello se recomienda verificarlo.

A4: Investigación y Pruebas de Tolerancia a Fallos

Para estas pruebas primero voy a probar el algoritmo de menor número de conexiones para el balanceador de haproxy y posteriormente probaré el algoritmo de round-robin para el balanceador de traefi.

He elegido solamente estos dos平衡adores por la interfaz gráfica que proporcionan.

```
> ./init.sh
[+] Uso:
  -c Limpiar archivos dentro de logs_apache y logs_nginx
  -s Detener y eliminar contenedores (apache/nginx/nginx_balanceador/haproxy_balanceador/all)
  -b Crear imagen docker (apache/nginx/nginx_balanceador/haproxy_balanceador/all)
  -u Ejecutar docker compose up para el balanceador (nginx/haproxy/traefik/envoy)
    Puedes indicar una estrategia de balanceo para los siguientes balanceadores:
      nginx:
        pd = ponderación con pesos
        rb = round-robin (por defecto)
      haproxy:
        lc = menor número de conexiones
        rb = round-robin (por defecto)
        traefik/envoy: sin parámetro = round-robin (por defecto)
  -p Actualizar paquetes dentro de los contenedores activos
  -h Mostrar este panel de ayuda

C ~ /Escritorio/SWAP/P2 | on ~ main !16
./init.sh -u haproxy lc
```

```
> ./init.sh -u haproxy lc
[✓] Red red_servicios ya existe.
[✓] Red red_web ya existe.
[i] Estrategia de balanceo: menor número de conexiones
[!] Se detectó nginx_balanceador corriendo. Deteniéndolo...
[✓] nginx_balanceador detenido.
[i] Comprobando puertos 8080 a 8089...
[+] Running 9/9
✓ Container web4           Running
✓ Container web3           Running
✓ Container web6           Running
✓ Container web8           Running
✓ Container web5           Running
✓ Container web2           Running
✓ Container web7           Running
✓ Container web1           Started
✓ Container haproxy_balanceador Started
[+] Servicios iniciados con HAProxy.

C ~ /Escritorio/SWAP/P2 | on ~ main !15
```

Vemos actualmente todos los servidores web activos

HAProxy version 3.1.6-d929ca2, released 2025/03/20

Statistics Report for pid 8

> General process information

		Frontend												Backend												Display option:		External resources:																			
		Queue			Session rate			Sessions			Bytes			Denied			Errors			Warnings			Redis			Status		LastChk		Wght		Act		Server		Bck		Chk		Dwn		Downtime		Thrtile			
		Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Conn	Retr	Req	Conn	Retr	Redis	OPEN																							
		Frontend	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Conn	Retr	Req	Conn	Retr	OPEN																							
		backend rotodor												Sessions												Display option:		External resources:																			
		Queue			Session rate			Sessions			Bytes			Denied			Errors			Warnings			Redis			Status		LastChk		Wght		Act		Server		Bck		Chk		Dwn		Downtime		Thrtile			
		Cur			Max			Limit			Total			LbTot			Last			In			Out			Status		LastChk		Wght		Act		Server		Bck		Chk		Dwn		Downtime		Thrtile			
		Frontend												Sessions												Display option:		External resources:																			
		Cur			Max			Limit			Total			LbTot			Last			In			Out			Status		LastChk		Wght		Act		Server		Bck		Chk		Dwn		Downtime		Thrtile			
		Backend												Sessions												Display option:		External resources:																			
		Cur			Max			Limit			Total			LbTot			Last			In			Out			Status		LastChk		Wght		Act		Server		Bck		Chk		Dwn		Downtime		Thrtile			
		Backend												Sessions												Display option:		External resources:																			
		Cur			Max			Limit			Total			LbTot			Last			In			Out			Status		LastChk		Wght		Act		Server		Bck		Chk		Dwn		Downtime		Thrtile			

Voy a hacer actualmente unas pruebas con todas las webs, para ello usaré un bucle de peticiones.

```
> for i in {1..500}; do curl -s http://localhost | grep "192.168" >> log.txt; done
cat log.txt | sort | uniq -c
 63 <p><strong>La dirección IP del servidor es:</strong> 192.168.10.2</p>
 62 <p><strong>La dirección IP del servidor es:</strong> 192.168.10.3</p>
 62 <p><strong>La dirección IP del servidor es:</strong> 192.168.10.4</p>
 62 <p><strong>La dirección IP del servidor es:</strong> 192.168.10.5</p>
 62 <p><strong>La dirección IP del servidor es:</strong> 192.168.10.6</p>
 63 <p><strong>La dirección IP del servidor es:</strong> 192.168.10.7</p>
 63 <p><strong>La dirección IP del servidor es:</strong> 192.168.10.8</p>
 63 <p><strong>La dirección IP del servidor es:</strong> 192.168.10.9</p>
```

Como podemos ver no existe ningún problema, voy a parar ahora los contenedores de nginx (los que son con ip impar dentro de la red 192.168.10.0/24)

		Frontend												Backend												Display option:		External resources:																			
		Queue			Session rate			Sessions			Bytes			Denied			Errors			Warnings			Redis			Status		LastChk		Wght		Act		Server		Bck		Chk		Dwn		Downtime		Thrtile			
		Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Conn	Retr	Req	Conn	Retr	Redis	OPEN																							
		Frontend												Sessions												Display option:		External resources:																			
		Cur			Max			Limit			Total			LbTot			Last			In			Out			Status		LastChk		Wght		Act		Server		Bck		Chk		Dwn		Downtime		Thrtile			
		Backend												Sessions												Display option:		External resources:																			
		Cur			Max			Limit			Total			LbTot			Last			In			Out			Status		LastChk		Wght		Act		Server		Bck		Chk		Dwn		Downtime		Thrtile			
		Backend												Sessions												Display option:		External resources:																			
		Cur			Max			Limit			Total			LbTot			Last			In			Out			Status		LastChk		Wght		Act		Server		Bck		Chk		Dwn		Downtime		Thrtile			

Vemos que funciona de forma correcta:

```
> for i in {1..500}; do curl -s http://localhost | grep "192.168" >> log.txt; done
cat log.txt | sort | uniq -c
 188      <p><strong>La dirección IP del servidor es:</strong> 192.168.10.2</p>
  62      <p><strong>La dirección IP del servidor es:</strong> 192.168.10.3</p>
187      <p><strong>La dirección IP del servidor es:</strong> 192.168.10.4</p>
  62      <p><strong>La dirección IP del servidor es:</strong> 192.168.10.5</p>
187      <p><strong>La dirección IP del servidor es:</strong> 192.168.10.6</p>
  63      <p><strong>La dirección IP del servidor es:</strong> 192.168.10.7</p>
188      <p><strong>La dirección IP del servidor es:</strong> 192.168.10.8</p>
  63      <p><strong>La dirección IP del servidor es:</strong> 192.168.10.9</p>
```

Las otras conexiones de 62-63 son de las pruebas de antes sobre los servidores nginx que hemos desactivado.

Ahora voy a realizar las pruebas para round-robin con traefik

```
> ./init.sh -u traefik
[✓] Red red_servicios ya existe.
[✓] Red red_web ya existe.
[!] Se detectó haproxy_balanceador corriendo. Deteniéndolo...
[✓] haproxy_balanceador detenido.
[i] Comprobando puertos 8080 a 8089...
[+] Running 9/9
✓ Container web2          Started
✓ Container web7          Running
✓ Container web4          Started
✓ Container web5          Running
✓ Container web3          Running
✓ Container web8          Started
✓ Container web6          Started
✓ Container web1          Started
✓ Container traefik_balanceador Started
[+] Servicios iniciados con Traefik.
```

El resultado es el siguiente:

Servers

	http://192.168.10.2
	http://192.168.10.3
	http://192.168.10.4
	http://192.168.10.5
	http://192.168.10.6
	http://192.168.10.7
	http://192.168.10.8

Podemos ver que todos los contenedores están corriendo correctamente.

```
> for i in {1..500}; do curl -s http://localhost | grep "192.168" >> log.txt; done
cat log.txt | sort | uniq -c
   63      <p><strong>La dirección IP del servidor es:</strong> 192.168.10.2</p>
   62      <p><strong>La dirección IP del servidor es:</strong> 192.168.10.3</p>
   63      <p><strong>La dirección IP del servidor es:</strong> 192.168.10.4</p>
   63      <p><strong>La dirección IP del servidor es:</strong> 192.168.10.5</p>
   62      <p><strong>La dirección IP del servidor es:</strong> 192.168.10.6</p>
   62      <p><strong>La dirección IP del servidor es:</strong> 192.168.10.7</p>
   63      <p><strong>La dirección IP del servidor es:</strong> 192.168.10.8</p>
   62      <p><strong>La dirección IP del servidor es:</strong> 192.168.10.9</p>
```

C | ~ /Escritorio/SWAP/P2 | on main !14 ?1

Si volvemos a parar los servidores de nginx y realizamos la prueba volvemos a obtener el mismo resultado que antes.

Realmente no sé qué pruebas más realizar, debido a que para haproxy para el de menor número de conexiones debería de tener muchos más dispositivos conectandome para comprobar una carga real.

Por ello decido seguir con el siguiente apartado que en mi opinión es más relevante.

A5. Automatización de escalado del escenario

Como sabemos, tenemos 8 servidores web, pero quería controlar los 8 servidores haciendo uso de node exporter y viendo la información a través de por ejemplo Grafana. Quiero que al hacer pruebas de cargas, tanto en servidores nginx como en apache, si la carga de la cpu de un servidor por ejemplo es más del 40-50% se cree un nuevo servidor web y se balancee la carga.

Para ello, haciendo uso de la IA, he planteado lo siguiente:

Uso de docker compose, node exporter, prometheus, grafana, Balanceador de carga (HAProxy) y el script para el escalado.

El cliente realiza peticiones al balanceador de carga y este a los servidores web

El Node exporter monitorea los servidores, dicha información la recopila prometheus y posteriormente se visualiza en grafana.

El script de escalado, consulta prometheus, el cual realiza una petición al docker compose API y realiza el escalado automático.

Para esta parte lo primero que he realizado es:

- Crear un nuevo fichero docker compose, el cual es el mismo que el de haproxy pero conteniendo los servicios de prometheus, qrafana y node exporter:

```
prometheus:
  image: prom/prometheus:latest
  container_name: prometheus
  restart: always
  volumes:
    - ./prometheus.yml:/etc/prometheus/prometheus.yml
    - prometheus-storage:/prometheus
    - ./file_sd:/etc/prometheus/file_sd      # <- esta linea es clave
  ports:
    - "9090:9090"
  networks:
    red_web:
      ipv4_address: 192.168.10.100

▷ Run Service
node-exporter:
  image: prom/node-exporter:latest
  container_name: node-exporter
  restart: always
  command:
    - '--path.rootfs=/host'
  volumes:
    - '/:/host:ro,rslave'
  networks:
    red_web:
      ipv4_address: 192.168.10.101
  ports:
    - "9100:9100"

▷ Run Service
grafana:
  image: grafana/grafana:latest
  container_name: grafana
  restart: always
  volumes:
    - grafana-storage:/var/lib/grafana

  networks:
    red_web:
      ipv4_address: 192.168.10.102
  ports:
    - "3000:3000"
  depends_on:
    - prometheus
```

Posteriormente, he instalado el node exporter dentro de las imágenes tanto de apache como nginx:

```
RUN apt update && apt install -y wget \
  && wget https://github.com/prometheus/node_exporter/releases/download/v1.8.1/node_exporter-1.8.1.linux-amd64.tar.gz \
  && tar xzf node_exporter-* .tar.gz \
  && mv node_exporter-* /node_exporter /usr/local/bin/ \
  && rm -rf node_exporter*
```

Además de exportar el puerto 9100 por el que cogerá las métricas prometheus.

Respecto a prometheus, debemos de crear un fichero de configuración el cual es: "prometheus.yml"

```
1 > ! prometheus.yml
1   global:
2     |   scrape_interval: 15s
3
4   scrape_configs:
5     |     - job_name: 'web_servers'
6       |       file_sd_configs:
7         |           - files:
8           |             |   - /etc/prometheus/file_sd/web_servers.json
9           |             refresh_interval: 10s
10
```

En esta configuración vemos la parte peculiar que es la de file_sd, que es la que usamos para la gestión automática de los servidores web que vamos a utilizar para obtener las métricas. Es decir, cuando se crea o se eliminan webs, se va actualizando dicho fichero json.

Al tener ya los ficheros de configuración debemos de crear los directorios de grafana_data y prometheus_data (esto lo realiza el script automáticamente, pero es para evitar los errores).

Al tener todos los ficheros, solamente nos quedaría ejecutar el docker compose y comprobar que todo está en orden.

CONTAINER ID	IMAGE NAMES	COMMAND	CREATED	STATUS	PORTS
ca752d01811f	flotodor-apache-image:p2	"/usr/local/bin/entr..."	About an hour ago	Up About an hour	9100/tcp, 0.0.0.0:8085->80/tcp, [::]:8085->80/tcp
16233fbfe737	flotodor-apache-image:p2	"/usr/local/bin/entr..."	About an hour ago	Up About an hour	9100/tcp, 0.0.0.0:8081->80/tcp, [::]:8081->80/tcp
b2e63a87efa0	flotodor-apache-image:p2	"/usr/local/bin/entr..."	About an hour ago	Up About an hour	9100/tcp, 0.0.0.0:8083->80/tcp, [::]:8083->80/tcp
02dce55a6839	flotodor-apache-image:p2	"/usr/local/bin/entr..."	About an hour ago	Up About an hour	9100/tcp, 0.0.0.0:8087->80/tcp, [::]:8087->80/tcp
863481a2665d	flotodor-haproxy_balanceador-image:p2	"docker-entrypoint.s..."	About an hour ago	Up 9 minutes	0.0.0.0:80->80/tcp, [::]:80->80/tcp, 0.0.0.0:9000->9000/tcp, [::]:9000->9000/tcp
78a42fe8e7d4	flotodor-nginx_web-image:p2	"/entrypoint.sh"	About an hour ago	Up About an hour	9100/tcp, 0.0.0.0:8088->80/tcp, [::]:8088->80/tcp
43e98c5523f3	flotodor-nginx_web-image:p2	"/entrypoint.sh"	About an hour ago	Up About an hour	9100/tcp, 0.0.0.0:8084->80/tcp, [::]:8084->80/tcp
7a22b98b4b11	flotodor-nginx_web-image:p2	"/entrypoint.sh"	About an hour ago	Up About an hour	9100/tcp, 0.0.0.0:8086->80/tcp, [::]:8086->80/tcp
a65d306e3e4a	flotodor-nginx_web-image:p2	"/entrypoint.sh"	About an hour ago	Up About an hour	9100/tcp, 0.0.0.0:8082->80/tcp, [::]:8082->80/tcp
a4fcfd54006fc	grafana/grafana:latest	"/run.sh"	About an hour ago	Up About an hour	0.0.0.0:3000->3000/tcp, [::]:3000->3000/tcp
6dfedlace02	prom/node-exporter:latest	"/bin/node_exporter ..."	About an hour ago	Up About an hour	0.0.0.0:9100->9100/tcp, [::]:9100->9100/tcp
20610db0b070	prom/prometheus:latest	"/bin/prometheus --c..."	About an hour ago	Up About an hour	0.0.0.0:9090->9090/tcp, [::]:9090->9090/tcp

Al comprobar que todo está funcionando correctamente, vamos a comprobar que funciona node exporter, prometheus y grafana.

Al entrar en <https://localhost:9090>, es decir, a la configuración de prometheus, podemos comprobar directamente si recoge los datos configuradores por el node exporter, por ejemplo con la métrica básica de cpu seconds total.

The screenshot shows the Prometheus interface with the query `>_ node_cpu_seconds_total`. Below the query bar are three tabs: Table (selected), Graph, and Explain. A message box indicates "Formatting turned off" due to displaying more than 1000 series. The results list several metrics, all with mode="idle":

```

node_cpu_seconds_total{cpu="0", instance="node-exporter:9100", job="node-exporter-host", mode="idle"}
node_cpu_seconds_total{cpu="0", instance="node-exporter:9100", job="node-exporter-host", mode="iowait"}
node_cpu_seconds_total{cpu="0", instance="node-exporter:9100", job="node-exporter-host", mode="irq"}
node_cpu_seconds_total{cpu="0", instance="node-exporter:9100", job="node-exporter-host", mode="nice"}
node_cpu_seconds_total{cpu="0", instance="node-exporter:9100", job="node-exporter-host", mode="softirq"}

```

Vemos que si hay datos.

Ahora vamos a comprobar la parte de target para comprobar que todos los nodos (todas las webs) funcionan correctamente.

Select scrape pool	Filter by target health	Filter by endpoint or labels	8 / 8 up
web_servers			
Endpoint	Labels	Last scrape	State
http://192.168.10.5:9100/metrics	instance="192.168.10.5:9100" job="node_exporter"	14.567s ago	130ms UP
http://192.168.10.9:9100/metrics	instance="192.168.10.9:9100" job="node_exporter"	733ms ago	107ms UP
http://192.168.10.2:9100/metrics	instance="192.168.10.2:9100" job="node_exporter"	8.112s ago	105ms UP
http://192.168.10.3:9100/metrics	instance="192.168.10.3:9100" job="node_exporter"	10.476s ago	93ms UP
http://192.168.10.6:9100/metrics	instance="192.168.10.6:9100" job="node_exporter"	13.759s ago	108ms UP
http://192.168.10.7:9100/metrics	instance="192.168.10.7:9100" job="node_exporter"	401ms ago	104ms UP
http://192.168.10.8:9100/metrics	instance="192.168.10.8:9100" job="node_exporter"	11.183s ago	119ms UP
http://192.168.10.4:9100/metrics	instance="192.168.10.4:9100" job="node_exporter"	2.146s ago	110ms UP

Como podemos comprobar, funciona todo perfectamente.

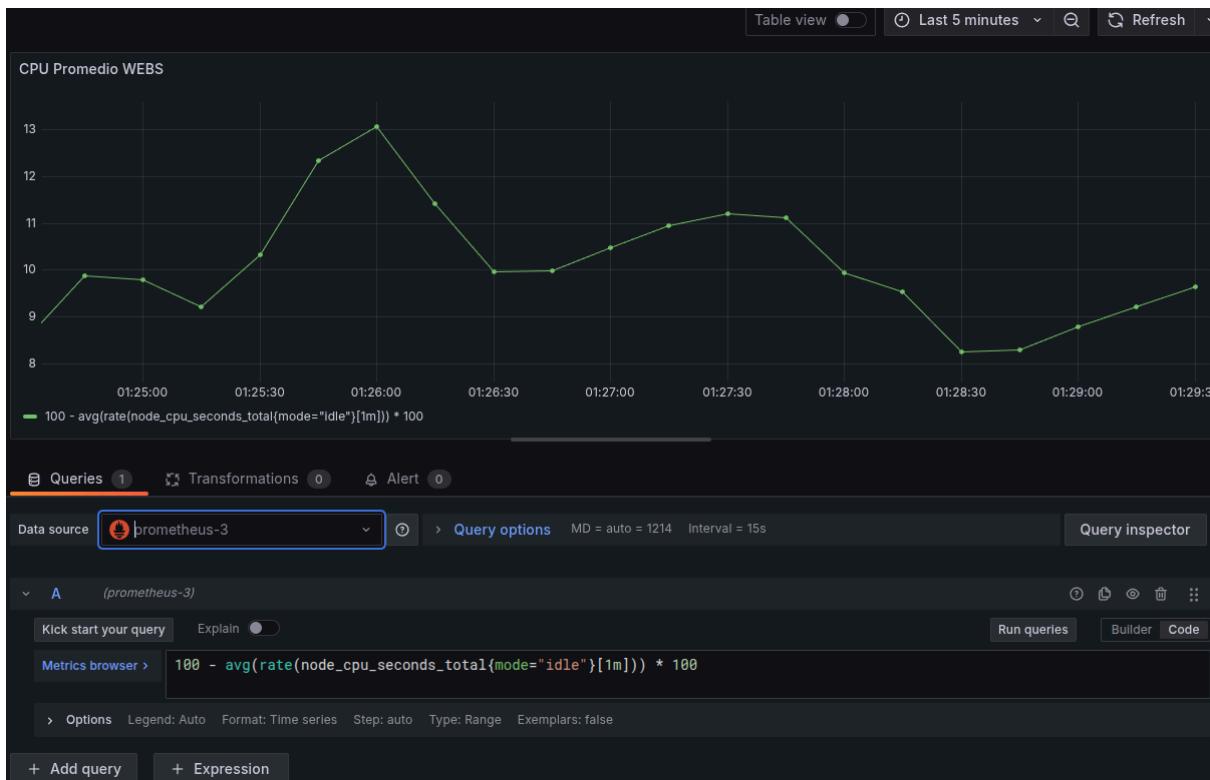
Ahora nos quedaría comprobar la parte de Grafana, para ello primero tenemos que configurar que se recopile la información de prometheus.

Para acceder a Grafana accedemos a través de <http://localhost:3000>, user y password son admin.

Al entrar vamos a la parte de Dashboard y vamos a Configure a new data source y buscamos prometheus.

en connection tenemos que colocar la ip de prometheus, en este caso como se trata de la misma red podemos colocar como ip <http://prometheus:9090>, debido a que se encuentran en la misma red en docker.

Posteriormente, vamos de nuevo a dashboard y seleccionamos la configuración de prometheus que hemos creado y ya entramos y en métricas colocamos la siguiente:



Como podemos observar, vemos el uso de cpu promedio de los servidores web cada 5 minutos. Esto lo veremos con el uso de potenciar la carga de los servidores web y ver posteriormente como se crean nuevas webs como se nos pide en el ejercicio.

Este dashboard lo he exportado en formato JSON, “dashboard.json”. Al importarlo en grafana saldrá la misma gráfica.

Toda la parte de la configuración actual lo he introducido en el script principal haciendo uso del parámetro -u escalado.

```
> ./init.sh -u escalado
[✓] Red red_servicios ya existe.
[✓] Red red_web ya existe.
[i] Estrategia de balanceo: round-robin (por defecto)
[!] Se detectó haproxy_balanceador corriendo. Deteniéndolo...
[✓] haproxy_balanceador detenido.
[i] Comprobando puertos 8080 a 8089...
[+] Running 12/12
✓ Container web7           Running
✓ Container web4           Running
✓ Container web2           Running
✓ Container web8           Running
✓ Container web3           Running
✓ Container web6           Running
✓ Container web1           Running
✓ Container node-exporter   Running
✓ Container prometheus     Running
✓ Container web5           Running
✓ Container grafana         Running
✓ Container haproxy_balanceador Started
[+] Servicios iniciados con monitorización y escalado automático.
[✓] Escalador iniciado en nueva terminal (gnome-terminal)
```

Tras comprobar que las configuraciones básicas funcionan correctamente y automatizar dentro del script el docker compose, me quedaría la parte más relevante del ejercicio, comprobar el monitoreo (haciendo uso de cargas en los servidores), escalar servicios y automatizar el script de escalado dentro del propio script inicial.

Para esta parte, primero voy a comentar el script que he creado para que se ejecute.

Dicho script se llama “escalador.sh”, el cual se ejecuta dentro del script de init.sh cuando usamos el escalado, es decir, ./init.sh -u escalado.

En este script tenemos las siguientes limitaciones:

```
12
13 # Configuraciones
14 #Limitaciones de escalado
15 THRESHOLD_UP=50 # CPU alta para escalar, 50%
16 THRESHOLD_DOWN=20 # CPU baja para desescalar, 20%
17 MAX_CONTAINERS=20 # Máximo de contenedores
18 MIN_CONTAINERS=8 # Mínimo de contenedores, las 8 webs iniciales
19
20 PROMETHEUS_URL="http://localhost:9090" # URL de Prometheus
21 FILE_SD_CONFIG="./file_sd/web_servers.json" # Archivo de configuración de file_sd para Prometheus
22 STOP_FILE="./stop_escalador.flag" # Archivo de parada para el escalador
```

Como podemos ver, he puesto de límite que se cree 20 webs (alternando entre nginx y apache) y que el mínimo de webs sean las 8 que se crean desde la práctica 1. Por otro lado, vemos las limitaciones respecto al uso de cpu, si el promedio de carga de las cpus de los servidores web superan el 50%, se crearán nuevas webs de forma automática hasta el límite de 20 y si dicho promedio es menor que el 20% se irán eliminando hasta llegar al mínimo de 8.

Dentro de este script tengo los siguientes métodos:

```
# Método para crear el archivo de log de las acciones del escalador
log() {
    mkdir -p ./logs_escalado
    echo "[${(date '+%Y-%m-%d %H:%M:%S')}] $1" >> ./logs_escalado/escalador.log
}
```

Método para comprobar los logs de cuando se han creado o eliminado las webs.

```
# Método para actualizar la configuración de HAProxy con las instancias activas
actualizar_haproxy_cfg() {
    local cfg=".P2-flotodor-haproxy/config_balanceador/haproxy.cfg"
    cat > "$cfg" <<EOF
global
    stats socket /var/lib/haproxy/stats

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

frontend flotodor
    bind *:80
    default_backend backend_flotodor

backend backend_flotodor
    option httpchk GET /
EOF

    for web in ${get_active_webs}; do
        ip=$(docker inspect -f '{{.NetworkSettings.Networks.red_web.IPAddress}}' "$web")
        echo "    server $web $ip:80 maxconn 32 check" >> "$cfg"
    done

    cat >> "$cfg" <<EOF

listen stats
    bind *:9000
    mode http
    stats enable
    stats uri /estadisticas_flotodor
    stats realm HAProxy\\ Statistics
    stats auth flotodor:SWAP1234
EOF
    echo -e "[i] Configuración de HAProxy actualizada con instancias activas"
}
```

Método para actualizar el fichero de configuración del balanceador de HAProxy con las nuevas webs que se van creando/eliminando.

```

1 # Método para actualizar el archivo de configuración de file_sd
2 # con las instancias activas
3 # Este archivo es utilizado por Prometheus para descubrir las instancias
4 # de Node Exporter que están corriendo en los contenedores web.
5 update_file_sd_config() {
6     echo "[" > "$FILE_SD_CONFIG"
7     echo " {" >> "$FILE_SD_CONFIG"
8     echo '     "targets": [' >> "$FILE_SD_CONFIG"
9     for web in $(get_active_webs); do
10         ip=$(docker inspect -f '{{.NetworkSettings.Networks.red_web.IPAddress}}' "$web")
11         echo "         \"$ip:9100\", " >> "$FILE_SD_CONFIG"
12     done
13     sed -i '$ s/,//}' "$FILE_SD_CONFIG"
14     echo "     ]," >> "$FILE_SD_CONFIG"
15     echo '     "labels": { "job": "node_exporter" }' >> "$FILE_SD_CONFIG"
16     echo " }" >> "$FILE_SD_CONFIG"
17     echo "]" >> "$FILE_SD_CONFIG"
18     echo -e "[i] Archivo file_sd actualizado con targets activos"
19 }
```

Como he comentado anteriormente, para prometheus iba a usar configuración automática para obtener las métrica de los node exporter, por ello tenemos que ir actualizando el fichero de file_sd que nos permite colocar las ips de los servidores que vamos a usar para obtener sus métricas de forma dinámica.

```

1
2 # Método para obtener los nombres de los contenedores web activos
3 # y ordenarlos por su índice numérico
4 get_active_webs() {
5     docker ps --format '{{.Names}}' | grep -E '^web[0-9]+$' | sort -V
6 }
7
8 # Método para obtener el siguiente índice disponible para crear una nueva web
9 # Se basa en los nombres de los contenedores activos
10 get_next_index() {
11     get_active_webs | sed 's/web//' | sort -n | tail -n1 | awk '{print $1 + 1}'
12 }
```

En este caso tenemos dos métodos para obtener los nombres de los contenedores y los índices. Esto se utiliza para saber que web crear, qué tipo (nginx/apache) etc.

```

102
103 # Método para crear una nueva web
104 # Se basa en el índice pasado como argumento
105 # Se asigna un tipo de web (nginx o apache) basado en el índice
106 # Se asigna una IP y un puerto basado en el índice
107 # Se asignan volúmenes para los logs y el contenido web
108 # Se conecta la web a las redes red_web y red_servicios
109 crear_web() {
110     local id=$1
111     local tipo=$(( $(($id % 2)) -eq 0 )) && echo "nginx" || echo "apache"
112     local puerto=$((8080 + id))
113     local ip_web="192.168.10.$((1 + id))"
114     local ip_srv="192.168.20.$((1 + id))"
115
116     if [ "$tipo" == "apache" ]; then
117         volume1="-v $(pwd)/web_flotodor:/var/www/html"
118         volume2="-v $(pwd)/logs_apache:/var/log/apache2"
119         imagen="flotodor-apache-image:p2"
120     else
121         volume1="-v $(pwd)/web_flotodor:/usr/share/nginx/html:ro"
122         volume2="-v $(pwd)/logs_nginx:/var/log/nginx"
123         imagen="flotodor-nginx_web-image:p2"
124     fi
125
126     docker network disconnect red_web "web$id" 2>/dev/null
127     docker network disconnect red_servicios "web$id" 2>/dev/null
128
129     docker run -d --name "web$id" \
130         --network red_web \
131         --ip "$ip_web" \
132         -e SERVER_NAME=web$id \
133         $volume1 \
134         $volume2 \
135         -p "$puerto:80" \
136         "$imagen"
137
138     docker network connect --ip "$ip_srv" red_servicios "web$id"
139     echo -e "[+] web$id (${tipo}) creado y arrancado"
140     log "[+] web$id creado"
141 }

```

Este método se utiliza para crear las webs, entre nginx o apache y usando la información relevante que tenemos que tener en cuenta, es decir, la información que tenemos en los docker compose para la creación de las 8 primeras webs.

```

3 | # Método para eliminar la última web dinámica
4 | # Se basa en el nombre del contenedor y se ordena por su índice numérico
5 | eliminar_ultimo_web() {
6 |     last_web=$(get_active_webs | grep -E '^web([9-9]|1-9)[0-9]+)$' | sort -V | tail -n 1)
7 |     if [ ! -z "$last_web" ]; then
8 |         docker stop "$last_web" && docker rm "$last_web"
9 |         echo -e "[+] $last_web eliminado (dinámica)"
10 |        log "[+] $last_web eliminado (dinámica)"
11 |    else
12 |        echo "[i] No hay webs dinámicas por encima de web8 para eliminar"
13 |        log "[i] No hay webs dinámicas por encima de web8 para eliminar"
14 |    fi
15 }
16

```

De forma análoga, creé este método para eliminar las webs.

```

# Método para obtener el uso de CPU de los contenedores
# Se basa en la consulta a Prometheus para obtener el uso de CPU
# Se utiliza la métrica node_cpu_seconds_total para calcular el uso de CPU
get_cpu_usage() {
    local response
    response=$(curl -sG "$PROMETHEUS_URL/api/v1/query" \
        --data-urlencode 'query=100 - (avg by (instance) (rate(node_cpu_seconds_total{mode="idle"}[1m])) * 100)')

    if echo "$response" | jq -e '.data.result | length > 0' &>/dev/null; then
        echo "$response" | jq -r '.data.result[] | "\(.metric.instance) \(.value[1])"'
        return 0
    else
        return 1
    fi
}

```

Este método es para obtener el uso de CPU promedio de las webs haciendo uso de la API de prometheus.

```

# Método para recargar el balanceador de carga
recargar_balanceador() {
    if docker ps --format '{{.Names}}' | grep -q "haproxy_balanceador"; then
        docker restart haproxy_balanceador
        echo -e "[v] Balanceador reiniciado"
        log "[v] haproxy_balanceador reiniciado"
    fi
}

```

Cuando actualizamos el fichero de configuración de haproxy debemos de reiniciar el servicio del balanceador.

Para terminar con este script, me queda comentar el método principal:

```

# BUCLE PRINCIPAL
# =====
# LÓGICA DE BUCLE CON FLAG
# =====
rm -f "$STOP_FILE"
echo -e "${greenColour}[i] Escalador iniciado. Esperando señal de parada...${endColour}"

# =====
# Esperar hasta obtener un valor de CPU válido, esto se debe a que
# el contenedor de Prometheus puede tardar un poco en iniciar y
# devolver métricas válidas.
# =====
while true; do
    # Comprobar si hay señal de parada
    if [[ -f "$STOP_FILE" ]]; then
        echo "[i] Señal de parada detectada. Deteniendo escalador..."
        log "[!] Señal de parada detectada. Deteniendo escalador..."
        rm -f "$STOP_FILE"
        exit 0
    fi

    # Obtener datos de CPU
    cpu_data=$(get_cpu_usage)
    # Comprobar si la consulta fue exitosa y si hay datos
    # Si no hay datos, esperar 15 segundos y volver a intentar
    if [ $? -ne 0 ] || [ -z "$cpu_data" ]; then
        sleep 15
        continue
    fi

```

En esta primera parte, debemos de tener en cuenta que he creado un fichero, que utilizo como flag, debido a que he hecho que este script de escalador se ejecute cada 15 segundos, pero he tenido que tener en cuenta dos “problemas”:

- Tener en cuenta que se ejecuta en la misma terminal que ejecuto el script inicial, por lo tanto he tenido que colocar que este script se ejecute en una nueva terminal (gnome-terminal).
- También he tenido que tener en cuenta que si yo paro el programa con init.sh -s all, ese bucle no se para si no se utiliza por ejemplo un flag, cuando el flag existe, este bucle no se ejecuta.

Por otro lado, cuando iniciamos por primera vez los servicios, se va a tardar unos segundos en obtener información relevante de las métricas de prometheus, por ello he utilizado la comprobación, todo ello para evitar tener información basura en la terminal.

```

# Obtener datos de CPU
cpu_data=$(get_cpu_usage)
# Comprobar si la consulta fue exitosa y si hay datos
# Si no hay datos, esperar 15 segundos y volver a intentar
if [ $? -ne 0 ] || [ -z "$cpu_data" ]; then
    sleep 15
    continue
fi

# Obtener el valor de CPU promedio
avg_cpu=$(echo "$cpu_data" | awk '{print $2}' | head -n1)
# Comprobar si el valor de CPU es un número entero
avg_cpu_int=${avg_cpu%.*}
# Si no es un número entero, esperar 15 segundos y volver a intentar
# Esto puede ocurrir si la consulta a Prometheus no devuelve un valor válido
if ! [[ "$avg_cpu_int" =~ ^[0-9]+$ ]]; then
    sleep 15
    continue
fi

# Obtener el número total de webs activas
total_webs=$(get_active_webs | wc -l)
echo -e "${blueColour}[1] CPU Promedio: $avg_cpu_int% ${endColour}"

```

En esta parte vemos cómo obtener el valor de cpu, comprobar que sea un valor en número entero y obtener el total de webs que están activas.

Los siguientes métodos los he comentado de forma detallada con uso de la IA su uso, por eso solo voy a dejar la captura.

```

#
# Este bloque de código verifica si el promedio de uso de CPU (avg_cpu_int) es mayor o igual al umbral definido (THRESHOLD_UP)
# y si el número total de contenedores web (total_webs) es menor que el máximo permitido (MAX_CONTAINERS).
# Si ambas condiciones se cumplen:
# 1. Obtiene el siguiente índice disponible para un nuevo contenedor web mediante la función get_next_index.
# 2. Crea un nuevo contenedor web utilizando la función crear_web con el índice obtenido.
# 3. Actualiza la configuración del balanceador de carga HAProxy llamando a actualizar_haproxy_cfg.
# 4. Actualiza el archivo de configuración de service discovery llamando a update_file_sd_config.
# 5. Recarga el balanceador de carga para aplicar los cambios llamando a recargar_balanceador.
if [ "$avg_cpu_int" -ge "$THRESHOLD_UP" ] && [ "$total_webs" -lt "$MAX_CONTAINERS" ]; then
    index=$(get_next_index)
    crear_web "$index"
    actualizar_haproxy_cfg
    update_file_sd_config
    recargar_balanceador

    # 1. Si el uso promedio de CPU es menor que el umbral inferior (THRESHOLD_DOWN) y el número
    #     total de webs es mayor que el mínimo permitido (MIN_CONTAINERS):
    #     - Se elimina el último contenedor web.
    #     - Se actualiza la configuración de HAProxy.
    #     - Se actualiza el archivo de configuración de service discovery.
    #     - Se recarga el balanceador de carga.
    elif [ "$avg_cpu_int" -lt "$THRESHOLD_DOWN" ] && [ "$total_webs" -gt "$MIN_CONTAINERS" ]; then
        eliminar_ultimo_web
        actualizar_haproxy_cfg
        update_file_sd_config
        recargar_balanceador

    # 2. Si el uso promedio de CPU es mayor o igual al umbral superior (THRESHOLD_UP) y el número
    #     total de webs ya ha alcanzado el máximo permitido (MAX_CONTAINERS):
    #     - Se muestra un mensaje indicando que la CPU está alta, pero no se puede escalar más
    #     - porque ya se alcanzó el límite máximo de contenedores.
    elif [ "$avg_cpu_int" -ge "$THRESHOLD_UP" ] && [ "$total_webs" -ge "$MAX_CONTAINERS" ]; then
        echo -e "${redColour}[1] CPU alta, pero ya hay $MAX_CONTAINERS webs. No se escala más. ${endColour}"
    # 3. Si el uso promedio de CPU es menor que el umbral inferior (THRESHOLD_DOWN) y el número
    #     total de webs ya está en el mínimo permitido (MIN_CONTAINERS):
    #     - Se muestra un mensaje indicando que la CPU está baja, pero no se puede desescalar más
    #     - porque ya se alcanzó el límite mínimo de contenedores.
    elif [ "$avg_cpu_int" -lt "$THRESHOLD_DOWN" ] && [ "$total_webs" -le "$MIN_CONTAINERS" ]; then
        echo -e "${yellowColour}[1] CPU baja, pero ya están las $MIN_CONTAINERS webs mínimas. No se desescala más. ${endColour}"
    # 4. En cualquier otro caso (cuando la CPU está estable y no se cumplen las condiciones
    #     anteriores):
    #     - Se muestra un mensaje indicando que la CPU está estable y no se realizará ninguna
    #     acción de escalado o desescalado.
    else
        echo -e "${turquoiseColour}[1] CPU estable. No se escala ni desescala. ${endColour}"
    fi

    sleep 15
done

```

Tras haber comentado toda la información relevante sobre el script inicial (init.sh) y este script para el escalado, voy a mostrar su funcionamiento.

Al ejecutar ./init.sh -u escalado tenemos la terminal del escalado:

```
[i] Escalador iniciado. Esperando señal de parada...
[i] CPU Promedio: 12%
[i] CPU baja, pero ya están las 8 webs mínimas. No se desescalda más.

[+] Escalador detenido (flag creada)
> ./init.sh -s all
[!] No se encontraron contenedores activos para eliminar.
[i] Escalador detenido (flag creada)
> ./init.sh -u escalado
[!] Red red_servicios ya existe.
[!] Red red_web ya existe.
[i] Estrategia de balanceo: round-robin (por defecto)
[i] Comprobando puertos 8080 a 8089...
[+] Running 12/12
✓ Container web3          Started
✓ Container web8          Started
✓ Container web5          Started
✓ Container web6          Started
✓ Container web4          Started
✓ Container web7          Started
✓ Container node-exporter Started
✓ Container web2          Started
✓ Container web1          Started
✓ Container prometheus    Started
✓ Container grafana       Started
✓ Container haproxy_balanceador Started
[+] Servicios iniciados con monitorización y escalado automático
[!] Escalador iniciado en nueva terminal (gnome-terminal)
```

Para su funcionamiento voy a hacer uso de la siguiente prueba de carga:

```
wrk -t100 -c800 -d250s http://localhost:80
```

Para ello hay que tener instalado wrk, la prueba se realiza al propio balanceador de carga.

Esta prueba envía 100 hebras con 800 conexiones durante 250 segundos (estos datos han sido a base de prueba y error para poder comprobar que se crean las 20 webs y no se pasan del límite y que al eliminarse ocurre lo mismo).

Por ello, voy a ejecutar la carga y mostrar las diferentes capturas de pantalla de su funcionamiento.

En pocos segundos podemos ver con docker stats lo siguiente:

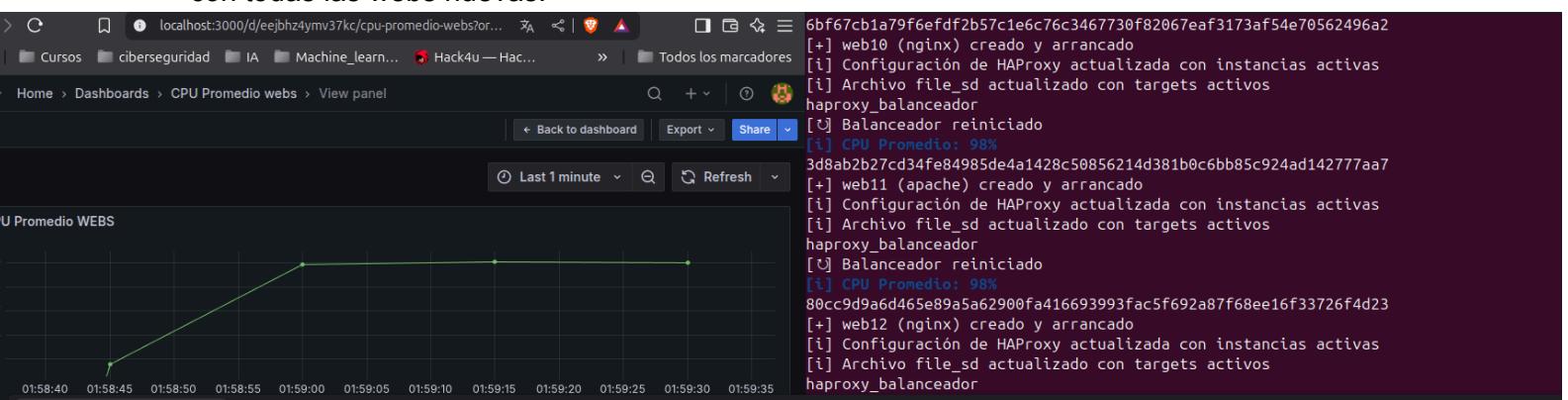
CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM %	NET I/O	BLOCK I/O	PIDS
11e469525ad3	node-exporter	0.00%	2.996MiB / 15.04GiB	0.02%	45.3kB / 126B	0B / 0B	5
b19a7964393b	web6	102.95%	35.04MiB / 15.04GiB	0.23%	5.53MB / 241MB	0B / 12.3kB	28
36f86d007fe2	web2	100.09%	35.32MiB / 15.04GiB	0.23%	5.61MB / 246MB	0B / 77.8kB	27
27c16c530df8	web4	103.84%	35.68MiB / 15.04GiB	0.23%	5.42MB / 237MB	0B / 12.3kB	28
a9ab1ad5039b	prometheus	0.55%	73.34MiB / 15.04GiB	0.48%	22.2MB / 4.25MB	3.37MB / 74.1MB	22
3316a2edb6a5	web3	107.82%	50.22MiB / 15.04GiB	0.33%	4.17MB / 181MB	0B / 2.93MB	49
cbf2cd4efee6	web1	101.19%	48.96MiB / 15.04GiB	0.32%	3.86MB / 167MB	0B / 729KB	48
c781988fd0b	web8	99.66%	35.86MiB / 15.04GiB	0.23%	5.65MB / 247MB	0B / 9.63MB	28
3c85f5f554d	grafana	0.21%	72.55MiB / 15.04GiB	0.47%	359KB / 377KB	303KB / 459KB	21
30258a901fa7	web5	102.06%	49.18MiB / 15.04GiB	0.32%	3.81MB / 164MB	0B / 500kB	49
d290c49f892c	web7	103.17%	49.56MiB / 15.04GiB	0.32%	3.64MB / 156MB	0B / 504KB	50
f2839ea8dc33	haproxy_balanceador	336.11%	96.07MiB / 15.04GiB	0.62%	1.64GB / 1.65GB	0B / 0B	17

Podemos ver como aumenta el uso de la CPU y como se van creando las nuevas webs:

```
[i] CPU Promedio: 0%
[i] CPU baja, pero ya están las 8 webs mínimas. No se desescalá más.
[i] CPU Promedio: 35%
[i] CPU estable. No se escala ni desescalá.
[i] CPU Promedio: 66%
0b4867b809c9a4d107650d3f0086d3c22a027423d77c936fb5c53e6bcdcc1fc84
[+] web9 (apache) creado y arrancado
[i] Configuración de HAProxy actualizada con instancias activas
[i] Archivo file_sd actualizado con targets activos
haproxy_balanceador
[!] Balanceador reiniciado
[i] CPU Promedio: 95%
6bf67cb1a79f6efdf2b57c1e6c76c3467730f82067eaf3173af54e70562496a2
[+] web10 (nginx) creado y arrancado
[i] Configuración de HAProxy actualizada con instancias activas
[i] Archivo file_sd actualizado con targets activos
haproxy_balanceador
[!] Balanceador reiniciado
```

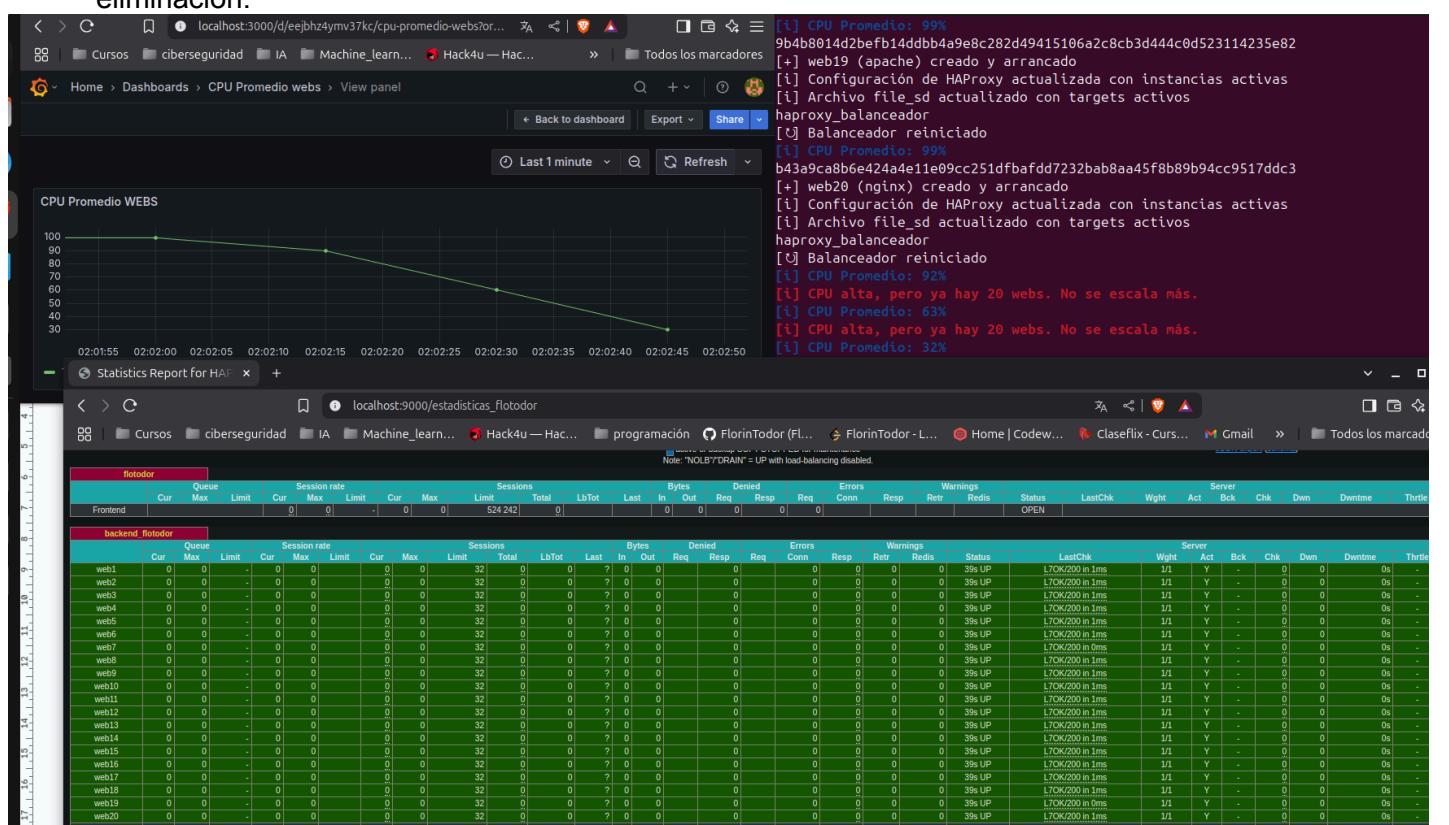
A su vez, podemos ver toda la información relevante:

En esta imagen vemos como crece el uso de la cpu en grafana, en la terminal vemos como no para de crearse nuevas webs y en la parte inferior vemos la información del balanceador con todas las webs nuevas.



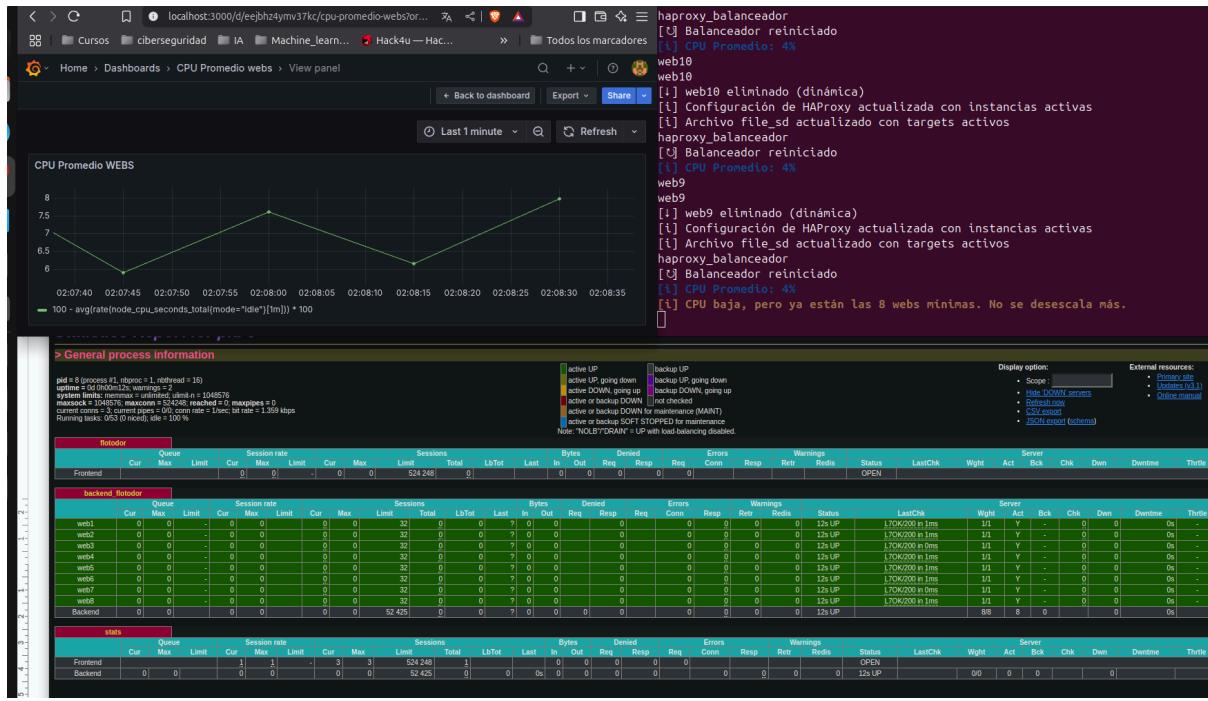
General process information														Display option:		External resources:			
pid = 8 (process #1, nbproc = 1, nbthread = 16)	uptime = 0d 00h00m05s; warnings = 2	active UP	backup UP	Scope:	Primary site														
sysctl = 0.0.0.0:8080; conntrack = 1048576; maxconn = 1048576; maxconn = 524248; reached = 0; maxpipes = 0	active UP, going down	backup UP, going down	Hide DOWN servers	Updates(V1)															
current conn = 801; current pipes = 0; conn rate = 0/sec; bit rate = 1.355 Gbps	active DOWN, going up	backup DOWN, going up	Refresh now	Online manual															
Running tasks: 479/1973 (0 need); idle = 16 %	active or backup UP	not checked	CSV export													JSON export (schema)			
Note: "NOLB" DRAIN = UP with load-balancing disabled.																			
flotodor		Queue			Session rate			Sessions			Bytes			Denied			Errors		
		Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	In	Out	Req	Conn	Resp	Retr	Redis	Status	
Frontend		0	800	-	0	800	800	524	246	800	7 050 855	1 163 361 729	0	0	0	0	0	OPEN	
backend flotodor		Queue			Session rate			Sessions			Bytes			Denied			Errors		
		Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	In	Out	Req	Conn	Resp	Retr	Redis	Status	
web1		0	0	-	1 653	2 018	23	32	32	12 163	546	0s	594 860	98 285 440	0	0	0	0	7s UP
web2		0	0	-	1 603	2 012	25	32	32	12 062	526	0s	599 813	97 174 701	0	0	0	0	0s UP
web3		0	0	-	1 646	2 039	25	32	32	12 142	537	0s	593 733	98 099 232	0	0	0	0	0s UP
web4		0	0	-	1 624	2 024	22	32	32	11 933	520	0s	583 639	96 157 503	0	0	0	0	0s UP
web5		0	0	-	1 654	1 974	18	32	32	12 096	551	0s	591 822	97 783 488	0	0	0	0	0s UP
web6		0	0	-	1 603	1 954	17	32	32	11 908	527	0s	582 659	99 956 043	0	0	0	0	0s UP
web7		0	0	-	1 643	2 009	17	32	32	12 148	536	0s	594 419	98 122 576	0	0	0	0	0s UP
web8		0	0	-	1 575	2 025	21	32	32	11 922	504	0s	583 149	96 076 773	0	0	0	0	0s UP
web9		0	0	-	1 654	2 047	17	32	32	12 131	554	0s	593 586	98 087 058	0	0	0	0	0s UP
web10		0	0	-	1 584	1 943	22	32	32	11 822	525	0s	578 200	95 273 200	0	0	0	0	0s UP
web11		0	0	-	1 652	1 999	24	32	32	12 152	533	0s	594 272	98 200 416	0	0	0	0	0s UP
web12		0	0	-	1 584	1 960	17	32	32	11 926	503	0s	583 541	96 153 266	0	0	0	0	0s UP
Backend		162	416	-	10 277	34 086	561	800	561 476	8 562	0s	7 074 260	1 165 917 818	0	0	0	0	0s UP	

Para finalizar con toda esta parte avanzada, voy a mostrar como llega al límite superior e inferior y como se van eliminando las webs que se acaban de crear mientras disminuye la gráfica de grafana. Por último mostraré los logs que se han guardado de la creación y eliminación.



Como vemos, la prueba de carga ha terminado y comienza a disminuir el uso de la cpu. En la terminal podemos ver como llega al límite superior y que ya no se crean más.

Podemos ver ahora como llegamos al límite inferior y como se han ido eliminando las webs:



La parte de comprobar que funciona el balanceador la he obviado debido a que la comprobación es obvia sobre su funcionamiento y debido a que ya es relativamente extensa esta documentación.

Para finalizar con esta parte voy a mostrar los logs como he comentado:

La última eliminación:

[2025-04-19 02:08:26] [↓] web9 eliminado (dinámico)

[2025-04-19 02:08:29] [↻] haproxy_balanceador reiniciado

Y la primera creación:

[2025-04-19 01:58:33] [+] web9 creado

[2025-04-19 01:58:35] [↻] haproxy_balanceador reiniciado

Análisis propuesta IA

Enlace: <https://chatgpt.com/share/67edc653-5c70-800d-9d07-93ac04798081>

Como comenté en la práctica anterior, el uso de la IA debería de ser obligatoria en todas las asignaturas debido a que es una herramienta que nos ayuda a aumentar la eficiencia.

Respecto al análisis de la propuesta:

Lo primero que tengo que comentar es que yo debería de ser más eficientemente en explicar los errores, como debería de darme la salida, en que se debería de centrar, etc. Sin embargo, hay momentos en los que un error “tonto” hace que pierdas el norte en vez de reflexionar.

La IA lo he utilizado en la mayor parte de esta práctica, simplemente para facilitar la creación de las distintas partes y tener una base por donde empezar.

El principal conflicto que he tenido con la IA respecto a la propuesta que me ha proporcionado ha sido por el uso de las imágenes de docker, debido a que pretendía que utilizara en las órdenes de CMD y ENTRYPOINT dos o más comandos distintos de bash para ejecutar los distintos servicios, aunque el pensamiento tendría sentido debido a que utilizaba la ejecución de forma paralela con el uso del &, y que a su vez concatenaba ambas ejecuciones con el operador &&, nunca se ejecutaban ambos servicios. Por ello decidí crear scripts para ejecutar todos los servicios que hicieran falta y que con el CMD o ENTRYPOINT solamente ejecuten dicho script.

Por otro lado, conflictos con sus propuestas respecto a la creación de logs, ya que aunque su lógica era coherente, no tenía en cuenta que hay que desactivar la creación de logs automáticos estándar de los servicios como apache o nginx. Para solucionar esto se lo comenté y busqué más información en internet y ya desactivé la creación automática.

Respecto al propio código de programación y lógica a utilizar me ha dejado muy sorprendido, debido a que apenas ha tenido algún error o que no se ejecuten las cosas a la primera (cuando salió la IA había muchos más fallos y sabemos todos que tardábamos más en buscar como arreglar el propio código de la IA que en hacer la práctica por nuestra parte). En esta parte no tengo mucha autoevaluación porque he utilizado en su mayoría su propio código ya que como he comentado antes, tendría directamente una base por donde comenzar.

Por otro lado, le comenté respecto al uso de balanceadores que me estaba ofreciendo tener dos servicios de balanceador en un mismo docker compose y eso no tenía sentido, por lo tanto le comenté que tendría que comenzar a crear distintos docker compose.

También tuve conflictos respecto al uso de balanceadores, debido a que yo quería utilizar algunos más visuales y los que me ofrecía eran más “”mediocres”” respecto a la estética.

O simplemente estaban muy limitadas. Aunque también me ha ofrecido implementar algunos balanceadores más complejos.

Me di cuenta tarde, pero le envié un gran chat con fácilmente más de 1000 líneas de error que me salía. Esto es una de las cosas que tengo que mejorar en el momento de hablar con la IA.

Por último donde más útil me ha sido ha sido para la tarea A5. Debido a que al tener nociones de este tema debido a la asignatura de ISE (la cual me gustó mucho y me pareció interesante esta parte de monitoreo). Me ha sido más simple entender que hacer y qué hacía la IA en cada momento.

Por ello, siempre comento que debe de ser una herramienta y entender la finalidad que pretende implementar la IA y la nuestra propia para solucionar los problemas que se nos plantea.

Respecto a este último ejercicio avanzado, he utilizado más lo que me ha ido comentado la IA debido a que tenía sentido toda la ejecución que me estaba planteando, aunque al principio estaba creando “objetos” de más, por ejemplo el uso de más exporters, no tenía en cuenta algunos errores o límites que hay que tener en cuenta para que se sobrepase por ejemplo el escalado de las webs y convertirlo todo en un bucle infinito (aunque también depende del tipo de carga que probemos).

Con esto doy por finalizada esta documentación y la práctica.

Tiempo en la práctica

Realmente, para la parte básica se tarda 2 horas en realizar, la parte avanzada quitando el A5 se tarda unas 3 horas y el A5 si no tienes nociones sobre node exporter, prometheus y grafana se puede complicar. Pero teniendo las nociones básicas se tarda en el A5 unas 3-4 horas (si realmente estás enfocado ese tiempo).

Por lo tanto he tardado en realizar la práctica, junto a la documentación que las hacía a la vez, unas 9-10 horas.