



UNIVERSIDAD DE GRANADA

P2 - Balanceo de carga

Servidores Web de Altas Prestaciones

UGR - ETSIIT

Torres Ramos, Juan Luis

17 abril de 2024

Índice

Índice	1
Tareas Básicas - B1 Preparación del Entorno de Trabajo	1
Tareas Básicas - B2 Configuración de Nginx como Balanceador de Carga	3
Tareas Básicas - B3 Implementación del escenario de Nginx con Docker Compose	7
Tareas Básicas - B4 Verificación y Pruebas del escenario de Nginx	9
Tareas Básicas - B5 Configuración de HAProxy como Balanceador de Carga	12
Tareas Básicas - B6 Implementación del escenario de HAProxy con Docker Compose	15
Tareas Básicas - B7 Verificación y Pruebas del escenario de HAProxy	16
Tareas Avanzada - A1 Configuraciones Avanzadas de Nginx	19
Tareas Avanzada - A2 Configuraciones Avanzadas de HAProxy	21
Tareas Avanzada - A3 Experimentación con Diferentes Balanceadores de Carga	24
Tareas Avanzada - A4 Investigación y Pruebas de Tolerancia a Fallos	28
Tareas Avanzada - A5 Automatización de escalado del escenario	32

Tareas Básicas - B1 Preparación del Entorno de Trabajo

Voy a crear directorios específicos para los archivos de configuración de los balanceadores:

1. *P2-tuusuariougr-nginx* para trabajar con los archivos de configuración de nginx
2. *P2-tuusuariougr-haproxy* para trabajar con los archivos de configuración de HAProxy.

```
mkdir P2-juanluis-nginx  
mkdir P2-juanluis-haproxy
```

Quedaría de la siguiente forma

```
tree  
.  
├── P2-juanluis-haproxy  
└── P2-juanluis-nginx
```

Tareas Básicas - B2 Configuración de Nginx como Balanceador de Carga

Trabajamos desde la carpeta de nginx. Voy a redactar el Dockerfile para crear una imagen personalizada de Nginx a partir de la imagen oficial. Además voy a escribir el archivo de configuración nginx.conf con la estrategia de balanceo round-robin.

Este es el dockerfile para ejecutar la imagen nginx, donde lo expongo en el puerto 80

```
FROM nginx:latest

COPY nginx.conf /etc/nginx/nginx.conf

EXPOSE 80

# en primer plano
CMD [ "nginx", "-g", "daemon off;"]

# en segundo plano
# CMD ["nginx"]
```

Para ejecutarlo

```
docker build -t juanluistr-nginx-image:p2 -f DockerfileNginx_juanluisUGR .
```

Pasamos a crear el archivo de configuración *nginx.conf*. Previamente hemos configurado la imagen nginx anterior solo comentando la parte del copiar nginx.conf. Con ello se generará un nginx.conf por defecto. Ejecutamos la imagen con un *docker run*. Dentro del contenedor, en el bash, vamos a la dirección */etc/nginx/* y hacemos un cat al archivo *nginx.conf* para copiar el de por defecto. Ya teniendo un *nginx.conf* por defecto vamos a modificarlo para que tenga una configuración de balanceo de carga aplicando round-robin, asociamos nuestros servicios habilitamos estadísticas en */estadisticas_juanluis* y aplicamos configuraciones de registro de acceso y errores.

El archivo nginx quedaría como sigue:

```
user  nginx;
worker_processes  auto;
error_log  /var/log/nginx/error.log notice;
pid        /var/run/nginx.pid;

events {
    worker_connections  1024;
}

http {
    upstream backend_juanluis{
        # algoritmo balanceo (least_conn;round-robin; etc.)
        # round-robin se usa por defecto
        least_conn; # para el apartado A1
        server 192.168.10.9;
        server 192.168.10.2;
        server 192.168.10.3 weight=5;
        server 192.168.10.4;
        server 192.168.10.5;
        server 192.168.10.6 weight=10;
        server 192.168.10.7;
        server 192.168.10.8;
        # etc
    }
    server {
        listen 80;
        server_name nginx_juanluis;

        access_log /var/log/nginx/nginx_juanluis.access.log;
        error_log /var/log/nginx/nginx_juanluis.error.log;

        location / {
            proxy_pass http://backend_juanluis;
            proxy_set_header Cookie $http_cookie;
            proxy_hide_header Set-Cookie;
        }

        location /estadisticas_juanluis {
            stub_status on;
        }
    }
}
```

El servidor Nginx escucha en el puerto 80 y actúa como balanceador de carga. Todas las solicitudes que entran son procesadas por el balanceador y luego las enviamos a los servidores backend definidos de la práctica 1 que hemos definido en la sección upstream. Round robin se establece por defecto , en /estadisticas_juanluis hemos configurado un monitor de estadísticas y los errores y los accesos se encuentran en las variables /var/log/nginx/nginx_juanluis.access.log y /var/log/nginx/nginx_juanluis.error.log.

Tareas Básicas - B3 Implementación del escenario de Nginx con Docker Compose

Vamos a reutilizar el DockerfileApache de la práctica 1 para los contenedores de Apache, y tenemos que modificar el dockerfile de la práctica anterior para, añadir a cada web una ip propia (igual a la que definimos en upstream) y luego añadimos el servicio “balanceador-nginx”.

El nuevo archivo dockerfile quedaría de la siguiente forma

```
version: '3.8'

services:
  balanceador-nginx:
    build:
      context: .
      dockerfile: DockerfileNginx_juanluisUGR
    image: juanluistr-nginx-image:p2
    volumes:
      - ./nginx_conf:/etc/nginx/conf.d
    ports:
      - "3000:80"
    networks:
      red_web:
        ipv4_address: 192.168.10.50
    depends_on:
      - web1
      - web2
      - web3
      - web4
      - web5
      - web6
      - web7
      - web8

  web1:
    image: juanluistr-apache-image:p1
    privileged: true
    build:
      context: .
      dockerfile: DockerfileApache_juanluisUGR
    volumes:
      - ./web_usuarioUGR:/var/www/html
    networks:
```

```
    red_web:
      ipv4_address: 192.168.10.9
    red_servicios:
      ipv4_address: 192.168.20.9
  #ports:
  # - "8081:80"

# ... (el resto de webX son similares a web1)

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

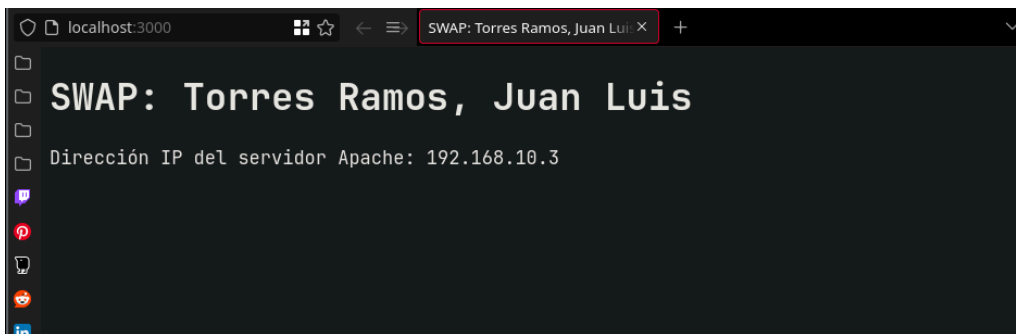
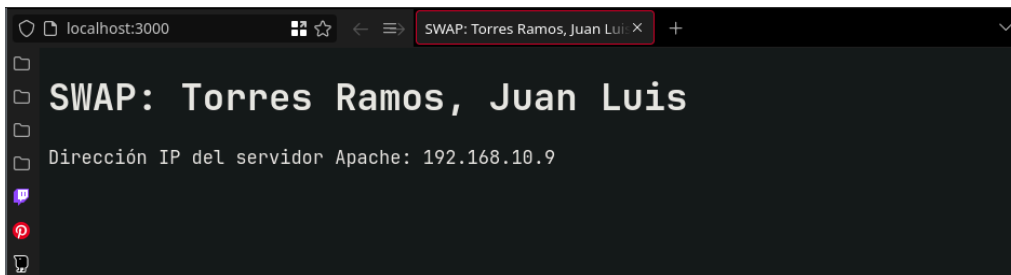
  red_servicios:
    driver: bridge
    ipam:
      driver: default
      config:
        - subnet: 192.168.20.0/24
```

Tareas Básicas - B4 Verificación y Pruebas del escenario de Nginx

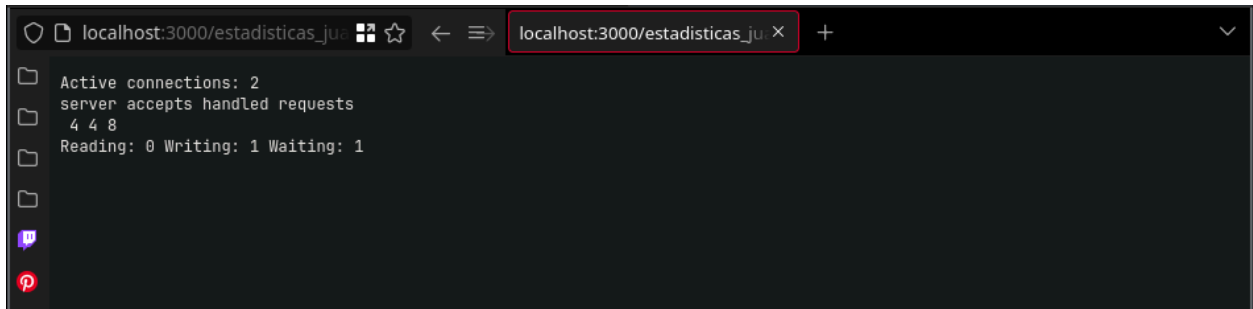
Ahora desplegamos los servicios

```
docker compose up -d
```

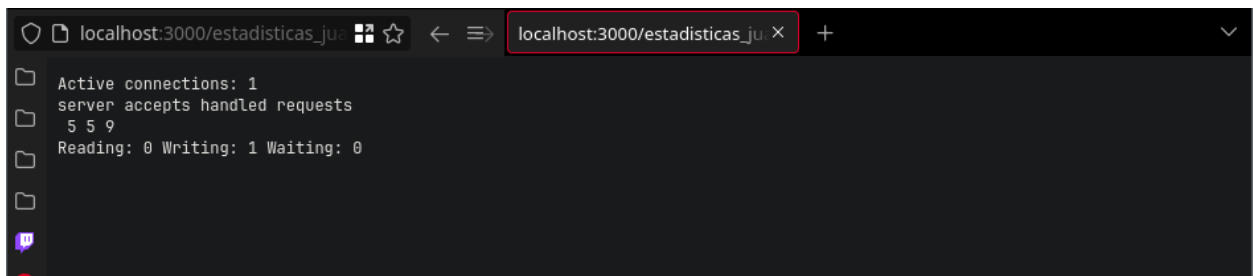
Vamos a comprobar que los servicios están activos y que Nginx distribuye las solicitudes correctamente. Habiendo quitado los puertos de cada WebX, solo podremos acceder al balanceador. Accedemos al balanceador en el <http://localhost:3000/>. Ahí si vamos actualizando la página , con la política round robin iremos recorriendo cada servicio del backend que hayamos asignado al balanceador.



Por último entramos al monitor de estadísticas que hemos configurado en http://localhost:3000/estadisticas_juanluis



Nos indica que, hay 2 conexiones activas del servidor, 8 solicitudes en total han sido procesadas por el servidor y que 4 han sido aceptadas y 4 han sido manejadas por el servidor. El balanceador actualmente está escribiendo una respuesta al cliente y esta esperando 1 conexión del siguiente WebX. Si actualizamos, pasaría a leer el siguiente WebX y quedaría de la forma:



Análisis propuesta IA

¿Qué significa las estadísticas de esta directiva stub_status?

Estas estadísticas son proporcionadas por la directiva stub_status de Nginx y proporcionan información sobre el estado actual del servidor. Aquí está el significado de cada línea:

Active connections: 2: Indica el número total de conexiones activas en el servidor en ese momento. En este caso, hay 2 conexiones activas.

server accepts handled requests: Estas tres métricas indican la cantidad de conexiones que el servidor ha aceptado, la cantidad de conexiones que ha manejado y el número total de solicitudes procesadas.

Accepted connections (aceptadas): 4 conexiones han sido aceptadas por el servidor desde que se inició.

Handled connections (manejadas): 4 conexiones han sido manejadas completamente por el servidor, es decir, se han completado todas las operaciones de lectura y escritura para estas conexiones.

Requests (solicitudes): 8 solicitudes en total han sido procesadas por el servidor.

Reading: 0 Writing: 1 Waiting: 1: Estas tres métricas representan el estado actual de las conexiones en el servidor.

Reading (Leyendo): El número de conexiones que están actualmente leyendo una solicitud del cliente.

Writing (Escribiendo): El número de conexiones que están actualmente escribiendo una respuesta al cliente.

Waiting (Esperando): El número de conexiones que están en estado de espera, por ejemplo, esperando en la cola para ser procesadas.

En resumen, estas estadísticas proporcionan una instantánea del rendimiento del servidor Nginx en términos de conexiones activas, solicitudes procesadas y el estado actual de las conexiones en lectura, escritura y espera. Son útiles para monitorear el rendimiento y la salud del servidor en tiempo real

Con chat gpt me ha ayudado a entender mejor qué es lo que devuelve mi monitoreo en Nginx. Me dice que devuelve cada variable y así me aseguro que mi respuesta es correcta porque viendo directamente desde el monitor de estadísticas puedo intuir que significa cada variable, pero con chat gpt me aseguro que vale cada variable.

Tareas Básicas - B5 Configuración de HAProxy como Balanceador de Carga

Vamos ahora a trabajar desde la carpeta HAProxy para implementar un balanceador de carga con este servicio, vamos a redactar el dockerfile para crear una imagen de HAProxy a partir de la imagen oficial.

Imagen Dockerfile Haproxy

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

# Copia tu configuración personalizada de HAProxy al contenedor
COPY haproxy.cfg /usr/local/etc/haproxy/haproxy.cfg

# Exponer el puerto 80 (o cualquier otro puerto que uses en tu configuración)
EXPOSE 80

# Comando predeterminado para ejecutar HAProxy
CMD ["haproxy", "-f", "/usr/local/etc/haproxy/haproxy.cfg"]
```

Antes de construir vamos a pasar a crear el haproxy.cfg

```
docker build -t juanluistr-haproxy-image:p2 -f
DockerfileHaproxy_juanluisUGR .
```

Análisis propuesta IA

redactar el dockerfile para crear una imagen de HAProxy a partir de la imagen oficial

Aquí tienes un Dockerfile básico para crear una imagen de HAProxy a partir de la imagen oficial:

Dockerfile

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

# Copia tu configuración personalizada de HAProxy al contenedor
COPY haproxy.cfg /usr/local/etc/haproxy/haproxy.cfg
```

```
# Opcional: si necesitas agregar archivos adicionales, como certificados
SSL, puedes copiarlos aquí
# COPY certs/ /usr/local/etc/haproxy/certs/
```

```
# Opcional: si necesitas instalar herramientas adicionales dentro del
contenedor, puedes hacerlo aquí
# RUN apt-get update && apt-get install -y tu-herramienta-adicional
```

```
# Exponer el puerto 80 (o cualquier otro puerto que uses en tu
configuración)
```

```
EXPOSE 80
```

```
# Comando predeterminado para ejecutar HAProxy
```

```
CMD ["haproxy", "-f", "/usr/local/etc/haproxy/haproxy.cfg"]
```

Asegúrate de tener un archivo haproxy.cfg en el mismo directorio que tu Dockerfile, o ajusta la ruta en la instrucción COPY si está en un directorio diferente. Este archivo debe contener tu configuración personalizada de HAProxy.

Después de crear tu Dockerfile, puedes construir la imagen ejecutando el siguiente comando en la misma carpeta que contiene el Dockerfile:

```
bash
```

```
docker build -t mi-haproxy .
```

Esto construirá la imagen de HAProxy según tu Dockerfile y le asignará el nombre mi-haproxy.

Recuerda que debes tener instalado Docker en tu sistema para poder construir y ejecutar contenedores Docker.

Me genera una imagen funcional y no le he tenido que hacer ninguna modificación, me ayuda a saber path importantes a la hora de usar Haproxy. Chat gpt a la hora de crear imágenes sencillas es muy funcional.

Este es el archivo de configuracion haproxy.cfg

```
frontend frontend_juanluisUGR
    bind *:80
    default_backend backend_juanluisUGR
    timeout client 60s

backend backend_juanluisUGR
    balance roundrobin
    timeout connect 5s
    timeout server 30s
    server web1 192.168.10.9:80 maxconn 32 check
    server web2 192.168.10.2:80 maxconn 32 check
    server web3 192.168.10.3:80 maxconn 32 check
    server web4 192.168.10.4:80 maxconn 32 check
    server web5 192.168.10.5:80 maxconn 32 check
    server web6 192.168.10.6:80 maxconn 32 check
    server web7 192.168.10.7:80 maxconn 32 check
    server web8 192.168.10.8:80 maxconn 32 check

listen stats
    bind *:9000
    mode http
    stats enable
    stats uri /estadisticas_juanluisUGR
    stats auth juanluisUGR:SWAP1234
    timeout client 60s
```

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

Tenemos ahora que crear el servicio en docker-compose, tenemos que exponer tanto el frontend que está en el puerto 80 como stats que está en el puerto 9000. Quedaría de la siguiente forma

Lo suyo es que para el frontend exponerlo en el puerto 80:80 en vez de 3000:80 o 8080:80 pero tengo en uso el puerto 80.

Quedaría de la siguiente forma en nuevo servicio balanceador-haproxy.

```
services:
  balanceador-haproxy:
    build:
      context: .
      dockerfile: DockerfileHaproxy_juanluisUGR
    image: juanluistr-haproxy-image:p2
    privileged: true
    volumes:
      - ./haproxy.cfg:/usr/local/etc/haproxy/haproxy.cfg:ro
    ports:
      - "3000:80"
      - "9000:9000"
```

luego podemos confirmar con un logs del contenedor que funciona correctamente, obtendremos:

```
2024-04-18 18:00:37 [NOTICE] (1) : New worker (8) forked
2024-04-18 18:00:37 [NOTICE] (1) : Loading success.
```

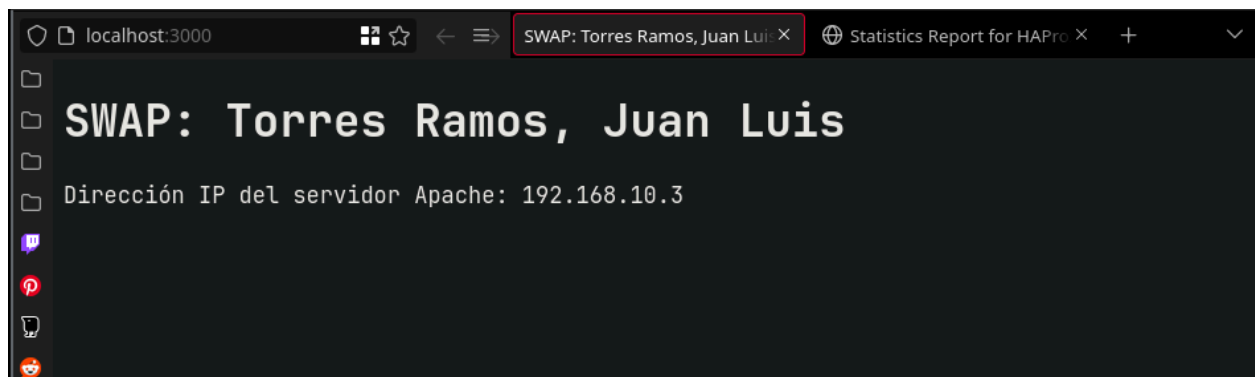
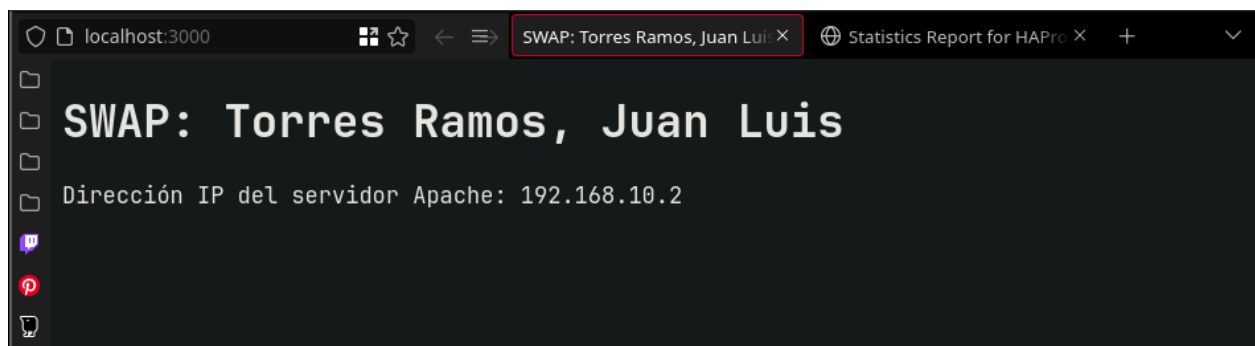
Tareas Básicas - B7 Verificación y Pruebas del escenario de HAProxy

Desplegamos nuestra granja web con el balanceador haproxy

```
docker-compose up -d
```

Ahora para ver nuestros servidores tenemos que acceder al <http://localhost:3000/> y para ver las estadísticas será en http://localhost:9000/estadisticas_juanluisUGR, accedemos con el usuario juanluisUGR y la contraseña SWAP1234.

En <http://localhost:3000/> podemos acceder a los servidores del backend, e irá recorriendo los según la política round robin que hemos definido



Viendo las estadísticas podemos ver que todos los servidores están up y en la variable session last podemos ver la última vez que se ha accedido a dicho servidor. Vamos recorriendo en <http://localhost:3000/> viendo todos los server y se irá actualizando dicha variable de acceso:

HAProxy version 2.9.7-5742051, released 2024/04/05

Statistics Report for pid 7

> General process information

pid = 7 (process #1, nbproc = 1, nbthread = 8)
 uptime = 0d 0h01m12s; warnings = 3
 system limits: memmax = unlimited; ulimit-n = 1048575
 maxsock = 1048575; maxconn = 524260; reached = 0; maxpipes = 0
 current conns = 1; current pipes = 0/0; conn rate = 0/sec; bit rate = 0.000 kbps
 Running tasks: 0/36 (0 niced); idle = 100 %

active UP
 active UP, going down
 active DOWN, going up
 active or backup DOWN
 active or backup DOWN for maintenance (MAINT)
 active or backup SOFT STOPPED for maintenance

backup UP
 backup UP, going down
 backup DOWN, going up
 not checked

Display option:
 Scope :
 Hide 'DOWN' servers
 Refresh now
 CSV export
 JSON export (schema)

External resources:
 Primary site
 Updates (v2.9)
 Online manual

frontend_juanluisUGR

	Queue			Session rate			Sessions				Bytes		Denied	Errors			Warnings		Status	Server									
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last		In	Out	Req	Conn	Resp		Retr	Redis	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
Frontend	0	1	-	0	3	524 260	3			1 521	1 491	0	0	0	0	0	0	0	0	0	OPEN								

backend_juanluisUGR

	Queue			Session rate			Sessions				Bytes		Denied	Errors			Warnings		Status	Server								
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last		In	Out	Req	Conn	Resp		Retr	Redis	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme
web1	0	0	-	0	1	0	1	32	1	1	1m8s	507	497	0	0	0	0	0	0	1m12s UP	L4OK in 0ms	1/1	Y	-	0	0	0s	-
web2	0	0	-	0	1	0	1	32	1	1	31s	507	497	0	0	0	0	0	0	1m12s UP	L4OK in 0ms	1/1	Y	-	0	0	0s	-
web3	0	0	-	0	1	0	1	32	1	1	17s	507	497	0	0	0	0	0	0	1m12s UP	L4OK in 0ms	1/1	Y	-	0	0	0s	-
web4	0	0	-	0	0	0	0	32	0	0	?	0	0	0	0	0	0	0	0	1m12s UP	L4OK in 0ms	1/1	Y	-	0	0	0s	-
web5	0	0	-	0	0	0	0	32	0	0	?	0	0	0	0	0	0	0	0	1m12s UP	L4OK in 0ms	1/1	Y	-	0	0	0s	-
web6	0	0	-	0	0	0	0	32	0	0	?	0	0	0	0	0	0	0	0	1m12s UP	L4OK in 0ms	1/1	Y	-	0	0	0s	-
web7	0	0	-	0	0	0	0	32	0	0	?	0	0	0	0	0	0	0	0	1m12s UP	L4OK in 0ms	1/1	Y	-	0	0	0s	-
web8	0	0	-	0	0	0	0	32	0	0	?	0	0	0	0	0	0	0	0	1m12s UP	L4OK in 0ms	1/1	Y	-	0	0	0s	-
Backend	0	0	-	0	1	52 426	3	3	17s	1 521	1 491	0	0	0	0	0	0	0	0	1m12s UP		8/8	8	0	0	0s	-	

stats

	Queue			Session rate			Sessions				Bytes		Denied	Errors			Warnings		Status	Server								
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last		In	Out	Req	Conn	Resp		Retr	Redis	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme
Frontend	0	1	-	1	1	524 260	1			940	60 068	0	0	0	0	0	0	0	0	OPEN								
Backend	0	0	-	0	0	52 426	0	0	0s	940	60 068	0	0	0	0	0	0	0	0	1m12s UP		0/0	0	0	0	0	0s	-

Cada vez que se despliega la granja web puede ocurrir que algún servidor no sea reconocido por el Haproxy. Lo he comentado ya en clase pero no se ha podido solucionar completamente. Intuyo que puede ser que se ejecute antes el servicio del balanceador antes que algún WebX. Para solucionar este problema, aunque hay veces que puede pasar, hay que añadir más tiempo de timeout client tanto en el frontend como el listen stat, mínimo unos 60s y la mayoría de despliegue deberían realizarse correctamente. Si algún servicio del backend no ha sido detectado por el balanceador, borrar el conjunto de contenedores que genera docker-compose y volver a desplegarlo

Análisis propuesta IA

im accesing form port 9000 but it says that the servers are not checked why

ChatGPT

If HAProxy is reporting that the servers are not checked, even though they are up and running, it suggests that HAProxy is unable to successfully perform health checks on the backend servers. Here are some potential reasons and troubleshooting steps to resolve the issue:

Health Check Configuration: Review your health check configuration in HAProxy to ensure it's correctly configured and appropriate for your backend server setup. Pay attention to parameters such as `inter`, `rise`, and `fall` in your server definitions.

Firewall or Network Issues: Check for any firewall rules or network issues that might be blocking HAProxy from communicating with the backend servers on the health check ports. Ensure that the health check ports are accessible from the HAProxy container.

Backend Server Configuration: Verify that the backend servers are properly configured to respond to the health check requests sent by HAProxy. Ensure that the servers are configured to respond to health checks on the correct port and path.

Server Reachability: Ensure that HAProxy can reach the backend servers on the specified IP addresses and ports. You can test this by trying to access the backend servers directly from the HAProxy container or by using tools like `telnet` or `nc`.

Logging and Debugging: Check the HAProxy logs for any error messages or warnings related to the health checks. Increase the logging level if necessary to get more detailed information about what might be going wrong.

Server Load or Performance Issues: If the backend servers are under heavy load or experiencing performance issues, they may not be able to respond to health check requests in a timely manner. Monitor the server's performance and ensure that it can handle the health check load.

By addressing these potential issues and troubleshooting accordingly, you should be able to identify and resolve the problem with HAProxy not successfully checking the backend servers.

Al principio, con el `haproxy.cfg`, no salía que los servidores estuviesen en verde, si se actualizaba la variable de acceso pero no si estaban siendo chequeados o no. Pregunté varias veces a chat gpt pero solo respondía con que revisará la configuración de mis archivos a ver si tenía algún error. Luego se encuentra en esta página web y solo necesita añadir el tag `check` en cada server del backend. Es importante con chatgpt usar bien los prompt y preguntarle cosas en específico, entender que es lo que le preguntas si no tardaras mas tiempo en encontrar la solución.

<https://www.haproxy.com/blog/how-to-enable-health-checks-in-haproxy>

Tareas Avanzada - A1 Configuraciones Avanzadas de Nginx

Vamos a modificar Nginx para aplicar una política de balanceo de conexiones activas con ponderaciones. Seguimos los pasos del siguiente enlace y añadimos que el web3 tenga peso 5 y web6 tenga peso 10 (los pesos también se pueden usar en la política round robin).

<https://docs.nginx.com/nginx/admin-guide/load-balancer/http-load-balancer/>

nginx.conf

```
user  nginx;
worker_processes  auto;

error_log  /var/log/nginx/error.log notice;
pid        /var/run/nginx.pid;


events {
    worker_connections  1024;
}


http {
    upstream backend_juanluis{
        # algoritmo balanceo (least_conn;round-robin; etc.)
        # round-robin se usa por defecto
        least_conn; # para el apartado A1
        server 192.168.10.9;
        server 192.168.10.2;
        server 192.168.10.3 weight=5;
        server 192.168.10.4;
        server 192.168.10.5;
        server 192.168.10.6 weight=10;
        server 192.168.10.7;
        server 192.168.10.8;

        # etc
    }

    server {
        listen 80;
        server_name nginx_juanluis;
```

```

access_log /var/log/nginx/nginx_juanluis.access.log;
error_log /var/log/nginx/nginx_juanluis.error.log;

location / {
    proxy_pass http://backend_juanluis;
    proxy_set_header Cookie $http_cookie;
    proxy_hide_header Set-Cookie;
}

location /estadisticas_juanluis {
    stub_status on;
}
}
}

```

Volvemos a construir la imagen y desplegar los contenedores .Ahora cuando recorremos los WebX nos da el siguiente resultado como ejemplo:

```

192.168.10.6
192.168.10.3
192.168.10.6
192.168.10.9
192.168.10.6
192.168.10.3
192.168.10.2
etc

```

Podemos ver que ahora el comportamiento da más prioridad a acceder a Web6 y luego a Web3 y luego al resto de servidores. Como las conexiones activas de cada servidor son las mismas, actuará como round robin a fin de cuentas(es decir que least_conn no tiene relevancia), pero con los pesos cambia la cosa. Es un balance basado en pesos/ponderaciones con el menor número de conexiones posibles.

Tareas Avanzada - A2 Configuraciones Avanzadas de HAProxy

Vamos a modificar Haproxy para aplicar una política source, donde se distribuye las solicitudes de manera uniforme entre los servidores backend basándose en la dirección IP de origen de la solicitud, todas las solicitudes provenientes de una misma dirección IP serán dirigidas al mismo servidor backend.

El archivo de configuración queda de la siguiente forma

```
frontend frontend_juanluisUGR
  bind *:80
  default_backend backend_juanluisUGR
  timeout client 30s

backend backend_juanluisUGR
  balance source          # Use 'balance source' to achieve similar IP-based
                           hashing
  hash-type consistent
  timeout connect 5s
  timeout server 30s
  server web1 192.168.10.9:80 maxconn 32 check
  server web2 192.168.10.2:80 maxconn 32 check
  server web3 192.168.10.3:80 maxconn 32 check
  server web4 192.168.10.4:80 maxconn 32 check
  server web5 192.168.10.5:80 maxconn 32 check
  server web6 192.168.10.6:80 maxconn 32 check
  server web7 192.168.10.7:80 maxconn 32 check
  server web8 192.168.10.8:80 maxconn 32 check
listen stats
  bind *:9000
  mode http
  stats enable
  stats uri /estadisticas_juanluisUGR
  stats auth juanluisUGR:SWAP1234
  timeout client 10s
```

construimos la imagen y desplegamos la granja con el balanceador Estudiamos ahora un ejemplo de salida:

```
192.168.10.7  
192.168.10.7  
192.168.10.7  
192.168.10.7  
etc
```

Confirmamos lo que hemos dicho anteriormente. Ahora todas las solicitudes provenientes de una misma dirección IP serán dirigidas al mismo servidor backend, en este caso es el 192.168.10.7.

Análisis propuesta IA

IP address.

```
try another similar option like ip hash
```

I see. It appears there might be a difference in the way HAProxy 2.9.7 handles the balance hash directive compared to later versions.

In your version, you might need to use a different approach. Instead of balance hash, you can use balance source to achieve a similar effect. Here's the modified configuration:

```
plaintext
backend backend_juanluisUGR
balance source          # Use 'balance source' to achieve similar IP-based
hashing
    hash-type consistent
    timeout connect 5s
    timeout server 30s
    server web1 192.168.10.9:80 maxconn 32 check
    server web2 192.168.10.2:80 maxconn 32 check
    server web3 192.168.10.3:80 maxconn 32 check
    server web4 192.168.10.4:80 maxconn 32 check
    server web5 192.168.10.5:80 maxconn 32 check
    server web6 192.168.10.6:80 maxconn 32 check
    server web7 192.168.10.7:80 maxconn 32 check
    server web8 192.168.10.8:80 maxconn 32 check
```

With this modification, HAProxy will still use the source IP address for hashing, effectively distributing requests based on the client's IP address. This should resolve the configuration error you encountered.

Al principio chat gpt me indicaba que usase balance IP-hash, lo que sucedía es que al final actuaba como una política round robin , o no se aplicaba la política hash, por lo que le pregunte a chat gpt que me diera otro acercamiento, en este caso funcionó con una política similar *balance source* añadiendo la directiva *hash-type consistent*. Hay veces que la primera solución de chat gpt no es la correcta, por lo que es recomendable probar otros caminos

Tareas Avanzada - A3 Experimentación con Diferentes Balanceadores de Carga

Vamos a aplicar el balanceador gobetween <https://leo.leung.xyz/wiki/Gobetween>

Vamos a usar la imagen directamente en docker

En docker-compose añadimos

```
version: '3.3'
services:
  gobetween:
    image: yyyar/gobetween
    ports:
      - "80:80"
      - "443:443"
    restart: always
    volumes:
      - ./config.toml:/etc/gobetween/conf/gobetween.toml
```

config file sera como config.toml

```
[servers.https]
bind = ":443"
protocol = "tcp"
balance = "weight"

max_connections = 1000
client_idle_timeout = "10m"
backend_idle_timeout = "10m"
backend_connection_timeout = "2s"

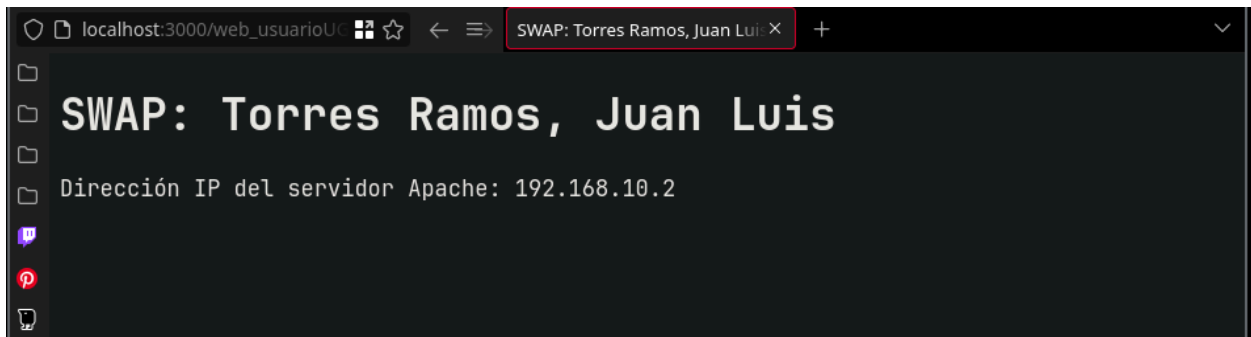
[servers.https.discovery]
kind = "static"
static_list = [
  "192.168.248.5:443 weight=1"
]

[servers.http]
bind = ":80"
protocol = "tcp"
balance = "weight"
```

```
max_connections = 1000
client_idle_timeout = "10m"
backend_idle_timeout = "10m"
backend_connection_timeout = "2s"

[servers.http.discovery]
kind = "static"
static_list = [
    "192.168.248.5:80 weight=1"
]
```

Lo único cuando vayamos a <http://localhost:3000> nos saldrá un index (no he podido configurar esta parte), allí tendremos que acceder a web_usuario y luego otra vez a web_usuario. Allí ya nos aparecerá nuestro index.php y cada vez que actualicemos, irá mediante la política round robin, visitando nuestros servidores backend.



Análisis propuesta IA

crea un dockerfile que use la imagen de gobetween para implementar un balanceador de carga, no uses alpine

Entendido, si prefieres no usar Alpine como base, podemos utilizar una imagen más genérica. Aquí tienes un Dockerfile que utiliza la imagen oficial de GoBetween desde Docker Hub:

Dockerfile

Usa la imagen base de Debian

FROM debian:latest

Instala dependencias necesarias

RUN apt-get update && apt-get install -y --no-install-recommends \
ca-certificates \
&& rm -rf /var/lib/apt/lists/*

Descarga y copia el archivo binario de GoBetween

ADD

https://github.com/yyyar/gobetween/releases/download/0.8.0/gobetween_0.8.0_linux_amd64 /usr/local/bin/gobetween

Asigna permisos de ejecución al archivo de GoBetween

RUN chmod +x /usr/local/bin/gobetween

Puerto en el que escuchará el balanceador de carga

EXPOSE 80 443

Comando por defecto al iniciar el contenedor

CMD ["gobetween", "--api=0.0.0.0:8888"]

Este Dockerfile descarga el archivo binario de GoBetween versión 0.8.0 para Linux de la URL proporcionada y lo coloca en /usr/local/bin/gobetween. Luego, asigna permisos de ejecución y expone los puertos 80 y 443 que serán utilizados para el balanceo de carga. Finalmente, el comando por defecto al iniciar el contenedor es gobetween --api=0.0.0.0:8888, lo que significa que GoBetween escuchará en todas las interfaces en el puerto 8888. Puedes modificar el comando según tus necesidades.

Cuidado con la aproximación que le damos. En este caso este no es el método que se recomiendan en gobetween. Recomendamos usar directamente la imagen dockerfile que les proporcionan, no crear la tuya misma. Yo he tenido problemas aquí a la hora de intentar usar mi propia imagen de gobetween. Este es un balanceador nuevo y hay que tener en cuenta que el chat gpt a veces no está actualizado.

Configura el archivo config.toml anterior para aplicar round robin
Para configurar el balanceo de carga como round-robin, debes cambiar el parámetro de balance a "roundrobin" en lugar de "weight". Aquí tienes cómo hacerlo:

```
toml
[servers.http]
bind = ":80"
protocol = "tcp"
balance = "roundrobin" # Cambiado a roundrobin

max_connections = 1000
client_idle_timeout = "10m"
backend_idle_timeout = "10m"
backend_connection_timeout = "2s"

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

Con esta modificación, el balanceador de carga distribuirá las solicitudes de manera equitativa entre los servidores backend en un ciclo round-robin. Cada solicitud se enviará al siguiente servidor en la lista, comenzando desde el principio después de alcanzar el último servidor.

No sabía cómo aplicar la política. Chat gpt en este caso si acierta con la respuesta y aplica la política correcta. Para preguntas simples y directas chat gpt suele ser muy útil.

Tareas Avanzada - A4 Investigación y Pruebas de Tolerancia a Fallos

Vamos a iniciar la granja web con el balanceador haproxy, y allí vamos a pausar uno de los contenedores y a estudiar la reacción en el balanceador

Desplegamos *con docker compose up -d* y nos vamos directamente a las estadísticas del balanceador. Como describí anteriormente, se encuentra en http://localhost:9000/estadisticas_juanluisUGR con usuario *juanluisUGR* y contraseña *SWAP1234*.

Estadísticas del balanceador haproxy

HAProxy version 2.9.7-5742051, released 2024/04/05

Statistics Report for pid 7

> General process information

pid = 7 (process #1, nbproc = 1, nbthread = 8)
uptime = 0d 0h00m11s; warnings = 0
system limits: memmax = unlimited; ulimit-n = 1048575
maxsock = 1048575; maxconn = 524260; reached = 0; maxpipes = 0
current conns = 2; current pipes = 0/0; conn rate = 0/sec; bit rate = 0.000 kbps
Running tasks: 0/38 (0 niced); idle = 100 %

active UP, active UP, going down, active DOWN, going up, active or backup DOWN, active or backup DOWN for maintenance (MAINT), active or backup SOFT STOPPED for maintenance, backup UP, backup UP, going down, backup DOWN, going up, not checked

Display option: Scope:
Hide 'DOWN' servers, Refresh now, CSV export, JSON export (schema)

External resources: Primary site, Updates (v2.9), Online manual

frontend_juanluisUGR

	Queue			Session rate			Sessions				Bytes		Denied	Errors		Warnings		Server													
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last		In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
Frontend				0	1	-	1	1		524	260	1			0	0	0	0					OPEN								

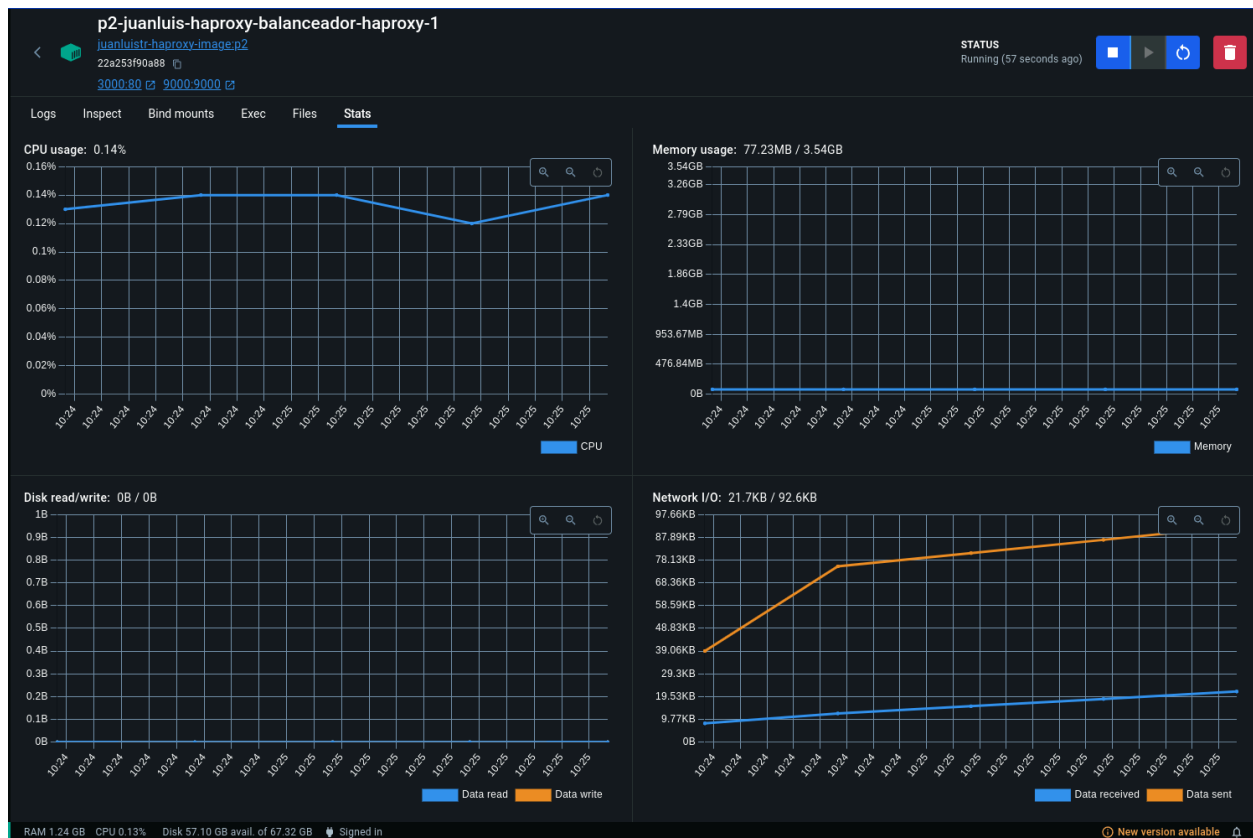
backend_juanluisUGR

	Queue			Session rate			Sessions				Bytes		Denied	Errors		Warnings		Server													
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last		In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
web1	0	0	-	0	1		1	1		32	1	1	3s	0	0	0	0	0	0	0	0	0	12s UP	L4OK in 0ms	1/1	Y	-	0	0	0s	-
web2	0	0	-	0	0		0	0		32	0	0	?	0	0	0	0	0	0	0	0	0	12s UP	L4OK in 0ms	1/1	Y	-	0	0	0s	-
web3	0	0	-	0	0		0	0		32	0	0	?	0	0	0	0	0	0	0	0	0	12s UP	L4OK in 0ms	1/1	Y	-	0	0	0s	-
web4	0	0	-	0	0		0	0		32	0	0	?	0	0	0	0	0	0	0	0	0	12s UP	L4OK in 0ms	1/1	Y	-	0	0	0s	-
web5	0	0	-	0	0		0	0		32	0	0	?	0	0	0	0	0	0	0	0	0	12s UP	L4OK in 0ms	1/1	Y	-	0	0	0s	-
web6	0	0	-	0	0		0	0		32	0	0	?	0	0	0	0	0	0	0	0	0	12s UP	L4OK in 0ms	1/1	Y	-	0	0	0s	-
web7	0	0	-	0	0		0	0		32	0	0	?	0	0	0	0	0	0	0	0	0	12s UP	L4OK in 0ms	1/1	Y	-	0	0	0s	-
web8	0	0	-	0	0		0	0		32	0	0	?	0	0	0	0	0	0	0	0	0	12s UP	L4OK in 0ms	1/1	Y	-	0	0	0s	-
Backend	0	0		0	1		1	1		52	426	1	1	3s	0	0	0	0	0	0	0	0	12s UP		8/8	8	0	0	0	0s	

stats

	Queue			Session rate			Sessions				Bytes		Denied	Errors		Warnings		Server														
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last		In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle	
Frontend				0	1	-	1	2		524	260	1			470	29	902	0	0	0			OPEN									
Backend	0	0		0	0		0	0		52	426	0	0	0s	470	29	902	0	0	0	0	0	0	12s UP		0/0	0	0	0	0	0s	

Las estadísticas actuales del balanceador las podemos ver directamente en docker-desktop



o podemos hacer `docker stats <nombre contenedor>`

```
docker stats p2-juanluis-haproxy-balanceador-haproxy-1
```

CONTAINER ID	NAME	CPU %	MEM
22a253f90a88	p2-juanluis-haproxy-balanceador-haproxy-1	0.14%	77.22MiB / 3.541GiB
		2.13%	57.2kB / 158kB
		0B / 0B	9

Podemos ver que el uso de CPU es muy bajo, de un 0,14%.

Ahora vamos a parar web6 con comandos o con docker desktop

```
docker stop p2-juanluis-haproxy-web6-1
```

Nos vamos a las estadísticas del balanceador (para la segunda imagen hemos de actualizar las estadísticas nuevamente:

localhost:9000/estadisticas_ju

Statistics Report for HAProxy

SWAP: Torres Ramos, Juan Luis

HAProxy version 2.9.7-5742051, released 2024/04/05

Statistics Report for pid 7

> General process information

pid = 7 (process #1, nbproc = 1, nbthread = 8)

uptime = 0d 0h04m21s; warnings = 0

system limits: memmax = unlimited; ulimit-n = 1048575

maxsock = 1048575; maxconn = 524260; reached = 0; maxpipes = 0

current conns = 1; current pipes = 0/0; conn rate = 1/sec; bit rate = 0.000 kbps

Running tasks: 0/37 (0 niced); idle = 100 %

active UP

active UP, going down

active DOWN, going up

active or backup DOWN

active or backup DOWN for maintenance (MAINT)

active or backup SOFT STOPPED for maintenance

backup UP

backup UP, going down

backup DOWN, going up

not checked

not checked

not checked

Display option:

Scope :

Hide 'DOWN' servers

Refresh now

CSV export

JSON export (schema)

External resources:

Primary site

Updates (v2.x)

Online manual

frontend_juanluisUGR

	Queue	Session rate	Sessions	Bytes	Denied	Errors	Warnings	Server
	Cur Max Limit	Cur Max Limit	Cur Max Limit	Total LbTot Last	In Out	Req Resp	Req Conn Resp Retr Redis	Status LastChk Wght Act Bck Chk Dwn Downtme Thrtle
Frontend	0 1 - 0 1	524 260	1		456 497	0 0 0		OPEN

backend_juanluisUGR

	Queue	Session rate	Sessions	Bytes	Denied	Errors	Warnings	Server
	Cur Max Limit	Cur Max Limit	Cur Max Limit	Total LbTot Last	In Out	Req Resp	Req Conn Resp Retr Redis	Status LastChk Wght Act Bck Chk Dwn Downtme Thrtle
web1	0 0 - 0 0	0 0	32 1	4m12s	456 497	0 0 0	0 0 0 0	4m21s UP L4OK in 0ms 1/1 Y - 0 0
web2	0 0 - 0 0	0 0	32 0	0 0	0 0	0 0 0	0 0 0 0	4m21s UP L4OK in 0ms 1/1 Y - 0 0
web3	0 0 - 0 0	0 0	32 0	0 0	0 0	0 0 0	0 0 0 0	4m21s UP L4OK in 0ms 1/1 Y - 0 0
web4	0 0 - 0 0	0 0	32 0	0 0	0 0	0 0 0	0 0 0 0	4m21s UP L4OK in 0ms 1/1 Y - 0 0
web5	0 0 - 0 0	0 0	32 0	0 0	0 0	0 0 0	0 0 0 0	4m21s UP L4OK in 0ms 1/1 Y - 0 0
web6	0 0 - 0 0	0 0	32 0	0 0	0 0	0 0 0	0 0 0 0	4m21s UP L4OK in 0ms 1/1 Y - 0 0
web7	0 0 - 0 0	0 0	32 0	0 0	0 0	0 0 0	0 0 0 0	4m21s UP L4OK in 0ms 1/1 Y - 0 0
web8	0 0 - 0 0	0 0	32 0	0 0	0 0	0 0 0	0 0 0 0	4m21s UP L4OK in 0ms 1/1 Y - 0 0
Backend	0 0	0 1	0 1	52 426	1 4m12s	456 497	0 0 0 0	4m21s UP 8/8 8 0 0 0

stats

	Queue	Session rate	Sessions	Bytes	Denied	Errors	Warnings	Server
	Cur Max Limit	Cur Max Limit	Cur Max Limit	Total LbTot Last	In Out	Req Resp	Req Conn Resp Retr Redis	Status LastChk Wght Act Bck Chk Dwn Downtme Thrtle
Frontend	1 1 - 1 2	524 260	2		940 59 936	0 0 0		OPEN
Backend	0 0	0 0	0 0	52 426	0 0	0 0	0 0 0 0	4m21s UP 0/0 0 0 0 0

localhost:9000/estadisticas_ju

Statistics Report for HAProxy

SWAP: Torres Ramos, Juan Luis

HAProxy version 2.9.7-5742051, released 2024/04/05

Statistics Report for pid 7

> General process information

pid = 7 (process #1, nbproc = 1, nbthread = 8)

uptime = 0d 0h04m32s; warnings = 1

system limits: memmax = unlimited; ulimit-n = 1048575

maxsock = 1048575; maxconn = 524260; reached = 0; maxpipes = 0

current conns = 1; current pipes = 0/0; conn rate = 0/sec; bit rate = 0.000 kbps

Running tasks: 0/37 (0 niced); idle = 100 %

active UP

active UP, going down

active DOWN, going up

active or backup DOWN

active or backup DOWN for maintenance (MAINT)

active or backup SOFT STOPPED for maintenance

backup UP

backup UP, going down

backup DOWN, going up

not checked

not checked

not checked

Display option:

Scope :

Hide 'DOWN' servers

Refresh now

CSV export

JSON export (schema)

External resources:

Primary site

Updates (v2.x)

Online manual

frontend_juanluisUGR

	Queue	Session rate	Sessions	Bytes	Denied	Errors	Warnings	Server
	Cur Max Limit	Cur Max Limit	Cur Max Limit	Total LbTot Last	In Out	Req Resp	Req Conn Resp Retr Redis	Status LastChk Wght Act Bck Chk Dwn Downtme Thrtle
Frontend	0 1 - 0 1	524 260	1		456 497	0 0 0		OPEN

backend_juanluisUGR

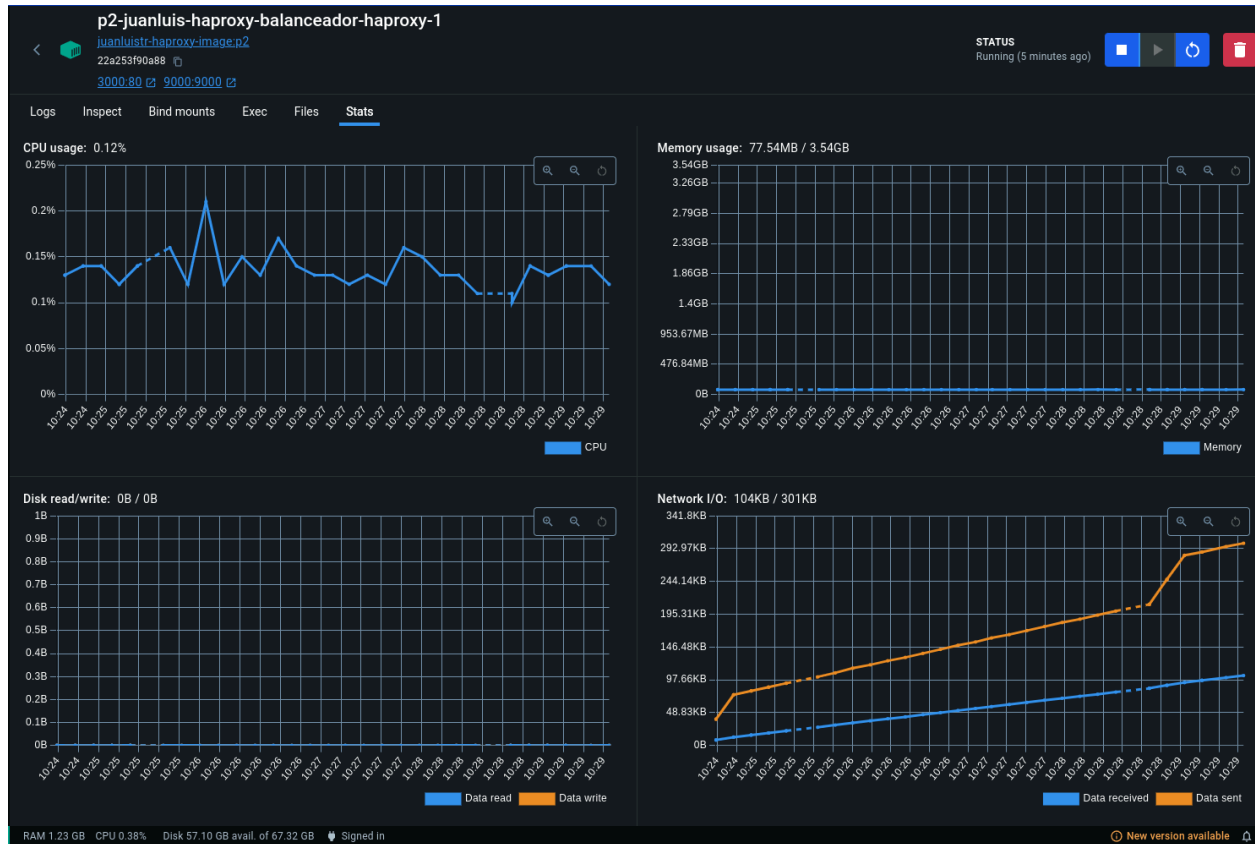
	Queue	Session rate	Sessions	Bytes	Denied	Errors	Warnings	Server
	Cur Max Limit	Cur Max Limit	Cur Max Limit	Total LbTot Last	In Out	Req Resp	Req Conn Resp Retr Redis	Status LastChk Wght Act Bck Chk Dwn Downtme Thrtle
web1	0 0 - 0 0	0 1	32 1	4m23s	456 497	0 0 0	0 0 0 0	4m32s UP L4OK in 0ms 1/1 Y - 0 0 0s
web2	0 0 - 0 0	0 0	32 0	0 0	0 0	0 0 0	0 0 0 0	4m32s UP L4OK in 0ms 1/1 Y - 0 0 0s
web3	0 0 - 0 0	0 0	32 0	0 0	0 0	0 0 0	0 0 0 0	4m32s UP L4OK in 0ms 1/1 Y - 0 0 0s
web4	0 0 - 0 0	0 0	32 0	0 0	0 0	0 0 0	0 0 0 0	4m32s UP L4OK in 0ms 1/1 Y - 0 0 0s
web5	0 0 - 0 0	0 0	32 0	0 0	0 0	0 0 0	0 0 0 0	4m32s UP L4OK in 0ms 1/1 Y - 0 0 0s
web6	0 0 - 0 0	0 0	32 0	0 0	0 0	0 0 0	0 0 0 0	9s DOWN L4TOUT in 2001ms 1/1 Y - 3 1 9s
web7	0 0 - 0 0	0 0	32 0	0 0	0 0	0 0 0	0 0 0 0	4m32s UP L4OK in 0ms 1/1 Y - 0 0 0s
web8	0 0 - 0 0	0 0	32 0	0 0	0 0	0 0 0	0 0 0 0	4m32s UP L4OK in 0ms 1/1 Y - 0 0 0s
Backend	0 0	0 1	0 1	52 426	1 4m23s	456 497	0 0 0 0	4m32s UP 7/7 7 0 0 0s

stats

	Queue	Session rate	Sessions	Bytes	Denied	Errors	Warnings	Server
	Cur Max Limit	Cur Max Limit	Cur Max Limit	Total LbTot Last	In Out	Req Resp	Req Conn Resp Retr Redis	Status LastChk Wght Act Bck Chk Dwn Downtme Thrtle
Frontend	0 1 - 1 2	524 260	2		1 410 90 172	0 0 0		OPEN
Backend	0 0	0 0	0 0	52 426	0 0	0 0	0 0 0 0	4m32s UP 0/0 0 0 0 0

Vemos que pasa de amarillo a rojo, de active UP, going down a active or backup DOWN.

Vamos a volver a ver las estadísticas a ver cómo ha afectado a nuestro balanceador



Al ser el uso en un principio del balanceador muy bajo, el impacto al quitar el servidor web6 es poco significativo, ha pasado de un 0,14% a un 0,12%.

Tareas Avanzada - A5 Automatización de escalado del escenario

Vamos a implementar una lógica de escalado automático. En el caso planteado he creado un script donde al inicializarlo, primero comprueba si el uso de CPU es menor del 10%, suspende todos los servicios del backend a excepción de 1 y del balanceador para no usar toda la granja web, luego se pone a comprobar hasta 5 veces si no supera la carga de cpu de un 70%, si lo supera levantamos uno de los servidores web que hemos suspendido antes, el script es de la forma:

```
#!/bin/bash
# Escalado dinámico

# Contador máximo de contenedores web
max=1
times=0

# Definir el nombre del balanceador
balanceador="balanceador-haproxy-1"

$cpu2=$(docker stats --no-stream --format "{{.Name}} {{.CPUPerc}}" | grep
"balanceador" | awk '{print $2}' | sed 's/%//')
if ((cpu2 < 10)); then
    echo "Cerrando contenedores web por poca carga..."
    docker ps -q --filter "name=web" | grep -v "$(docker ps -q --filter
"name=${balanceador}")" | tail -n +2 | xargs -I {} docker stop {} >/dev/null
2>&1
fi

# Bucle principal
while true; do
    # Mostrar mensaje de comprobación
    echo "Realizando comprobación..."

    # Contadores web activos
    web=$(docker ps --format '{{.Names}}' --filter status=running | grep web)
    num_containers=$(echo "$web" | wc -w)

    # Monitorizar la CPU del balanceador
    cpu=$(docker stats --no-stream --format "{{.Name}} {{.CPUPerc}}" | grep
"balanceador" | awk '{print $2}' | sed 's/%//')

    # Valor crítico para la CPU
```

```

valor_critico_cpu=70

# Mostrar información de la CPU
echo "Uso de CPU: $cpu"
echo "Valor Critico: 70"

# Comprobar si la CPU es críticamente alta
if (( $(echo "$cpu > $valor_critico_cpu" | bc -l) )); then
    echo "Necesito abrir un nuevo contenedor debido a la sobrecarga de la CPU..."
    # docker compose up -d web_3
    #echo "docker-compose up -d web$((num_containers + 1))"
    #docker compose up -d web$((num_containers))

    if docker ps --format '{{.Names}}' | grep -q "web$num_containers"; then
        echo "El servicio web$num_containers ya está en ejecución.
Aumentando la escala del servicio..."
        docker-compose up -d web=$((num_containers + 1))
    else
        echo "El servicio web$num_containers no está en ejecución. Iniciando
un nuevo contenedor..."
        docker-compose up -d web$((num_containers + 0))
    fi

    times=0
fi

# Incrementar el contador de comprobaciones
times=$((times + 1))

# Si se han realizado suficientes comprobaciones, salir del bucle
if ((times >= 5)); then
    break
fi

# Esperar antes de realizar la siguiente comprobación
sleep 10
done

echo "Escalado dinámico completado."

```


Para las métricas de monitoreo de CPU siempre me baso en docker Desktop, dentro del script hago llamadas a docker para ver el uso de la cpu por parte del balanceador.

Por último este sería un ejemplo de ejecución del script
tenemos la granja web levantada

HAProxy version 2.9.7-5742051, released 2024/04/05

Statistics Report for pid 8

> General process information

pid = 8 (process #1, nbproc = 1, nbthread = 8)
 uptime = 0d 0h00m26s; warnings = 0
 system limits: memmax = unlimited; ulimit-n = 1048575
 maxsock = 1048575; maxconn = 524280; reached = 0; maxpipes = 0
 current conns = 1; current pipes = 0/0; conn rate = 1/sec; bit rate = 0.000 kbps
 Running tasks: 0/37 (0 niced); idle = 100 %

active UP
 active UP, going down
 active DOWN, going up
 active or backup DOWN
 active or backup DOWN for maintenance (MAINT)
 active or backup SOFT STOPPED for maintenance

backup UP
 backup UP, going down
 backup DOWN, going up
 not checked

Display option:
 • Scope :
 • Hide 'DOWN' servers
 • Refresh now
 • CSV export
 • JSON export (schema)

External resources:
 • Primary site
 • Updates (v2)
 • Online manu

frontend_juanluisUGR

	Queue			Session rate			Sessions			Bytes	Denied	Errors	Warnings	Server																		
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit						Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk
Frontend				0	0	-	0	0		524	260	0											OPEN									

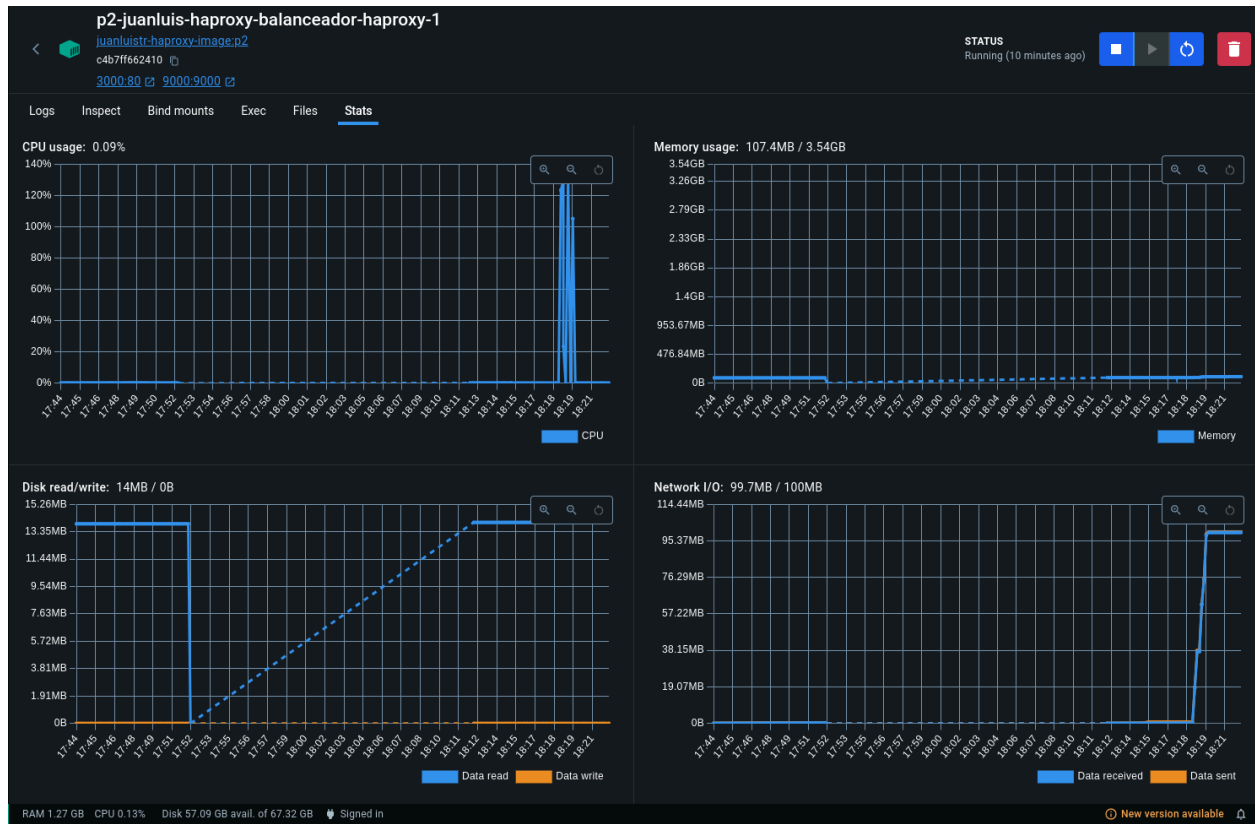
backend_juanluisUGR

	Queue			Session rate			Sessions			Bytes	Denied	Errors	Warnings	Server																	
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit						Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck
web1	0	0	-	0	0		0	0		32	0	0	?	0	0	0	0	0	0	0	0	0	26s UP	L4OK in 0ms	1/1	Y	-	0	0	0s	-
web2	0	0	-	0	0		0	0		32	0	0	?	0	0	0	0	0	0	0	0	26s UP	L4OK in 0ms	1/1	Y	-	0	0	0s	-	
web3	0	0	-	0	0		0	0		32	0	0	?	0	0	0	0	0	0	0	0	26s UP	L4OK in 0ms	1/1	Y	-	0	0	0s	-	
web4	0	0	-	0	0		0	0		32	0	0	?	0	0	0	0	0	0	0	0	26s UP	L4OK in 0ms	1/1	Y	-	0	0	0s	-	
web5	0	0	-	0	0		0	0		32	0	0	?	0	0	0	0	0	0	0	0	26s UP	L4OK in 0ms	1/1	Y	-	0	0	0s	-	
web6	0	0	-	0	0		0	0		32	0	0	?	0	0	0	0	0	0	0	0	26s UP	L4OK in 0ms	1/1	Y	-	0	0	0s	-	
web7	0	0	-	0	0		0	0		32	0	0	?	0	0	0	0	0	0	0	0	26s UP	L4OK in 0ms	1/1	Y	-	0	0	0s	-	
web8	0	0	-	0	0		0	0		32	0	0	?	0	0	0	0	0	0	0	0	26s UP	L4OK in 0ms	1/1	Y	-	0	0	0s	-	
Backend	0	0		0	0		0	0		52	426	0	0	?	0	0	0	0	0	0	0	26s UP		8/8	8	0	0	0	0s		

procedemos a ejecutar script (tenemos que darle permisos antes), lo primero que hace es, como el porcentaje de uso de cpu siempre va a ser bajo, va a suspender todos los servidores excepto el balanceador. En este caso queda up web5

Valor Critico: 70

Este



Los picos que vemos de la cpu es cuando hemos ejecutado el comando de sobrecarga al balanceador.

cron a intervalo regulares para el script

1. ejecutamos el comando

```
crontab -e
```

2. agregamos el trabajo de nuestro script, para que se ejecute cada 2 minutos

```
*/2 * * * *
```

```
/home/torres/Documents/Github/Practicas_SWAP/Practica_2/P2-juanluis-haproxy  
/script.sh
```

Análisis propuesta IA

Configura un trabajo cron que ejecute el script a intervalos regulares, por ejemplo, cada 2 minutos.

ChatGPT

Para configurar un trabajo cron que ejecute tu script a intervalos regulares, como cada 2 minutos, sigue estos pasos:

Abre tu terminal o consola de comandos.

Ejecuta el siguiente comando para editar el archivo de crontab:

```
crontab -e
```

Si es la primera vez que ejecutas este comando, se te pedirá que elijas un editor de texto. Selecciona tu editor preferido.

Agrega la siguiente línea al final del archivo de crontab para ejecutar tu script cada 2 minutos:

```
javascript
```

```
*/2 * * * * /ruta/a/tu/script.sh
```

Asegúrate de reemplazar `/ruta/a/tu/script.sh` con la ruta completa hacia tu script.

Guarda y cierra el editor de texto.

Con esta configuración, el script se ejecutará automáticamente cada 2 minutos. Asegúrate de que el script tenga permisos de ejecución adecuados para que cron pueda ejecutarlo correctamente. Puedes modificar el intervalo de tiempo cambiando el valor 2 en la línea de cron por el intervalo deseado en minutos.

Para configurar el crontab solo le he tenido que preguntar a chat gpt para que me lo configure, una tarea que no requiere de mucho tiempo. Ahora para el script también he usado chat gpt pero he usado varios prompt para crear dicho script, además de las indicaciones que nos han dado en clases de prácticas. Con ello y a base de prueba y error he podido conseguir una solución funcional. He usado muchos prompt ya que no estoy muy informado con el temas de script y uso de docker. Si hubiese estado más informado, con chat gpt habría obtenido una mejor solución en un menor tiempo.