



E.T.S. de Ingenierías Informática y de Telecomunicación
Práctica 1: Servidores Web y Almacenamiento
Servidores Web de Altas Prestaciones

Índice

1. Tareas Básicas	1
1.1. Configuración del Entorno	1
1.2. Creación y Configuración de Dockerfile	2
1.3. Configuración con Docker Compose	4
1.4. Verificación y Pruebas	5
2. Tareas Avanzadas	7
2.1. Personalización del Dockerfile	7
2.1.1. Activar compresión de archivos de Apache	7
2.1.2. Configuración de encabezados HTTP	9
2.2. Creación de contenedores con otros servidores web	11
2.2.1. Contenedor con nginx	11
2.2.2. Contenedor con lighttpd	13
2.3. Gestión Avanzada de Redes	15
2.4. Automatización con Scripts	20
2.4.1. Mantenimiento	20
2.4.2. Creación de Contenedores y Configuración de Redes	23
2.4.3. Estado de conectividad a las subredes:	25
2.4.4. Comprobar estado de Apache	27
2.4.5. Instalación de paquetes	29
2.5. Monitoreo y Logging	30
2.5.1. Sysdig	30
2.5.2. Estado del servidor	34

1. Tareas Básicas

1.1. Configuración del Entorno

Estoy usando un sistema operativo Linux, con Ubuntu 22. La configuración inicial es bastante simple, puesto que es crear un directorio y el archivo `index.php`.

Para esta sección, no hemos utilizado IA generativa, puesto que no tiene más misterio. Lo único destacable, es el comando de php para consultar la IP del servidor, el cuál es

```
<?php echo $_SERVER['SERVER_ADDR']; ?>
```

1.2. Creación y Configuración de Dockerfile

Para este apartado, ya sí vamos a hacer uso de IA. Comencemos preguntándole a ChatGPT:

- **Yo:** ¿Cómo creo un archivo Dockerfile con una imagen de Linux y le instalo PHP, Apache y herramientas de red, para comprobar conectividad?
- **IA:** Puedes crear un archivo Dockerfile para construir una imagen de Docker que contenga Linux, PHP, Apache y herramientas de red. Aquí te muestro un ejemplo básico de cómo hacerlo:

```
# Utilizamos la imagen base de Ubuntu
FROM ubuntu:latest

# Actualizamos los paquetes e instalamos las herramientas de red
RUN apt-get update && \
    apt-get install -y \
        iputils-ping \
        net-tools \
        curl \
        && apt-get clean

# Instalamos Apache y PHP
RUN apt-get install -y apache2 php libapache2-mod-php && \
    apt-get clean

# Copiamos el index a la carpeta de recursos de Apache
COPY index.php /var/www/html/

# Exponemos el puerto 80 para que podamos acceder a Apache
EXPOSE 80

# Iniciamos Apache al arrancar el contenedor
CMD ["apache2ctl", "-D", "FOREGROUND"]
```

Después, comenta un poco como ejecutar el contenedor y levantarlo, pero no copiaré toda la salida.

Análisis de la salida: Analizando la salida que nos genera ChatGPT, vemos que toma algunas consideraciones, que, a mi parecer, son mejorables:

- La imagen de Linux que toma es la de **Ubuntu**, la cuál es bastante pesada. Sería conveniente tomar una imagen más liviana, como la del ejemplo del seminario introductorio.
- Añade herramientas extras que, a priori, no son necesarias, como la instalación de **CURL**.

Como consecuencia, he decidido hacer las siguientes modificaciones:

- Usar una imagen basada en **Debian** más ligera, como puede ser **debian:bullseye-slim**. Para consultar las imágenes oficiales de **Debian**, podemos consultar aquí.
- Simplificar la instalación de paquetes, limitándonos a las herramientas necesarias.

```

# Utilizamos una imagen linux liviana
FROM debian:bullseye-slim

# Actualizamos e instalamos apache y PHP. Ademas de herramientas de redes
RUN apt-get update && apt-get install -y apache2 php libapache2-mod-php iputils-ping net-tools

# Habilitamos el modulo de PHP para Apache
RUN a2enmod php7.4

# Copiamos el archivo index.php a la carpeta /var/www/html
COPY web_danieeeeld2/index.php /var/www/html

# Eliminamos el archivo index.html, porque sino me mostraba este por defecto
RUN rm /var/www/html/index.html

# Exponemos el puerto 80
EXPOSE 80

# Iniciamos el servicio de apache
CMD ["apachectl", "-D", "FOREGROUND"]

```

Para poder ejecutar el contenedor, hacemos lo siguiente:

```

$ docker build -t debian-apache-php -f DockerfileApache_danieeeeld2 .
$ docker run -d -p 8080:80 --name mi-apache-php debian-apache-php

```

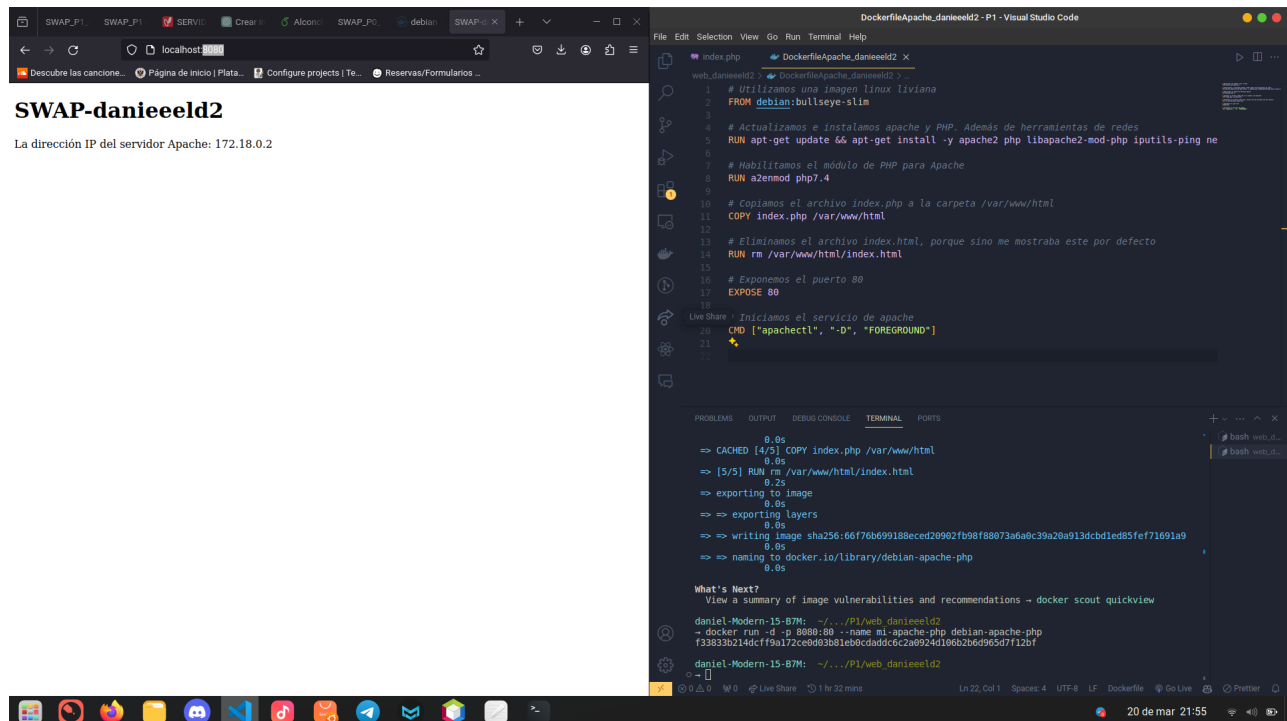
Primero, construimos la imagen (indicando el fichero, ya que este no se llama Dockerfile, por lo que no lo toma por defecto) y la ejecutamos en segundo plano con `-d`. En caso de que quisiéramos pararlo y/o borrarlo:

```

$ docker stop mi-apache-php
$ docker rm mi-apache-php

```

Podemos ver que se ejecuta correctamente en la siguiente captura:



1.3. Configuración con Docker Compose

Ahora tenemos que crear el fichero `docker-compose.yml`. Vamos a comenzar preguntándole a ChatGPT que nos ayude con la tarea:

- **Yo:** Tengo un fichero Dockerfile, llamado `DockerfileApache_danieeeld2`, el cual tiene una imagen debian con apache y php instalados. Deseo crear 8 réplicas del contenedor, las cuales están conectadas a dos redes específicas. Cada contenedor servirá una página web simple desde un directorio montado. Las instrucciones detalladas de la tarea son las siguientes¹:
 - Primero, crea una imagen llamada `danieeeld2-apache-image:p1` a partir del Dockerfile `DockerfileApache_danieeeld2`.
 - Una vez creada la imagen anterior, crea 8 contenedores llamados `webX`, donde X es un número de 1 a 8 con volúmenes donde se monte el directorio “`web_danieeeld2`” en el directorio raíz de Apache en el contenedor.
 - Por último, añade dos redes al contenedor, una red llamada `red_web` con dirección de red `192.168.10.0/24` y otra red llamada `red_servicios` con dirección de red `192.168.20.0/24`. Cada contenedor tiene misma parte de red, pero va cambiando los bits de hosts. La primera copia tendría `192.168.20.2` para conectarse a la red `servicios`, la siguiente copia `192.168.20.3` y así sucesivamente. Lo mismo ocurre con la otra red.
- **ChatGPT:** Puedes utilizar un archivo `docker-compose.yml` para definir y configurar los servicios que necesitas. A continuación, te muestro cómo hacerlo:

```
version: '3'

services:
  web:
    image: danieeeld2-apache-image:p1
    deploy:
      replicas: 8
    volumes:
      - ./web_danieeeld2:/var/www/html
    networks:
      red_web:
        ipv4_address: 192.168.10.{{.Task.Slot}}
      red_servicios:
        ipv4_address: 192.168.20.{{.Task.Slot}}

networks:
  red_web:
    ipam:
      config:
        - subnet: 192.168.10.0/24
  red_servicios:
    ipam:
      config:
        - subnet: 192.168.20.0/24
```

¹A ChatGPT se lo pasé todo como un párrafo, pero aquí lo pongo como `itemize` por estética y legibilidad

En este archivo `docker-compose.yml`, definimos un servicio llamado `web` que utiliza la imagen `danieeeld2-apache-image:p1`. Configuramos este servicio para tener 8 réplicas mediante la clave `replicas` bajo `deploy`.

Montamos el directorio local `./web_danieeeld2` en el directorio raíz de Apache dentro del contenedor. Para las redes, creamos dos redes llamadas `red_web` y `red_servicios` con las subredes especificadas. Luego, asignamos direcciones IP específicas para cada contenedor en estas redes usando la sintaxis `ipv4_address: 192.168.10..Task.Slot` y `ipv4_address: 192.168.20..Task.Slot`, respectivamente.

Análisis de la salida: La propuesta no parece mala, pero tiene algunos puntos a ser comentados:

- Se olvida de como crear la imagen base. Por suerte, sabemos crearla y lo hemos hecho justo en el apartado anterior. Solo tenemos que modificar el nombre. La creación de la imagen seria:

```
$ docker build -t danieeeld2-apache-image:p1 -f DockerfileApache_danieeeld2 .
```

- El uso de `.Task.Slot` no funciona, a parte de ser bastante extraño

Teniendo esto en cuenta, he decidido hacer las siguientes modificaciones:

```
version: '1.0'

services:
  web:
    image: danieeeld2-apache-image:p1
    deploy:
      replicas: 8
    volumes:
      - ./web_danieeeld2:/var/www/html
    networks:
      red_web:
      red_servicios:

networks:
  red_web:
    driver: bridge
    ipam:
      config:
        - subnet: 192.168.10.0/24
  red_servicios:
    driver: bridge
    ipam:
      config:
        - subnet: 192.168.20.0/24
```

Como no dice nada de que el número de la copia tenga que tener el mismo número de host, lo dejamos que lo asigne automáticamente. Esta asignación (al menos en las pruebas que he hecho) está optimizada para que tengan el mismo valor de bits de host en cada subred. Por otro lado, ha hecho un buen uso de **replicas**, pensé qué, por cómo había plantado la pregunta, me lo iba a hacer manual.

Nota: En una de las tareas avanzadas, creo una script que configura las redes para que cada contenedor tenga la parte de host igual a su identificador+1 (el +1 se debe a que, al tener **bridge** habilitado, este reserva esa IP para tenerla en el host, es decir, en nuestro PC como una interfaz de puente)

1.4. Verificación y Pruebas

Para esta sección, he ido entrando en los contenedores y probando la conectividad de cada uno. Adjunto una captura que ejemplifica el trabajo realizado:

```
+ Terminal
daniel-Modern-15-B7M: ~/Glt/SWAP-UGR/P1
→ docker compose up
[+] Running 1/0
✔ Network p1_red_web Created 0.1s
✔ Network p1_red_servicios Created 0.1s
✔ Container p1-web-8 Created 0.1s
✔ Container p1-web-3 Created 0.1s
✔ Container p1-web-1 Created 0.1s
✔ Container p1-web-5 Created 0.1s
✔ Container p1-web-6 Created 0.1s
✔ Container p1-web-7 Created 0.1s
✔ Container p1-web-4 Created 0.1s
Attaching to web-1, web-2, web-3, web-4, web-5, web-6, web-7, web-8
web-7 | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using
192.168.20.2. Set the 'ServerName' directive globally to suppress this message
web-4 | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using
192.168.20.3. Set the 'ServerName' directive globally to suppress this message
web-8 | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using
192.168.20.4. Set the 'ServerName' directive globally to suppress this message
web-3 | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using
192.168.20.5. Set the 'ServerName' directive globally to suppress this message
web-2 | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using
192.168.20.5. Set the 'ServerName' directive globally to suppress this message
web-6 | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using
192.168.20.6. Set the 'ServerName' directive globally to suppress this message
web-5 | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using
192.168.20.7. Set the 'ServerName' directive globally to suppress this message
web-1 | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using
192.168.20.8. Set the 'ServerName' directive globally to suppress this message
+ Terminal
daniel-Modern-15-B7M: ~
→ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS
91409cc3b1fe danieeld2-apache-lmage:p1 "apache2ctl -D FOREGR..." 6 minutes ago Up 6 minutes 80/tcp
86a0a8ee8ba1 danieeld2-apache-lmage:p1 "apache2ctl -D FOREGR..." 6 minutes ago Up 6 minutes 80/tcp
ae354737d187 danieeld2-apache-lmage:p1 "apache2ctl -D FOREGR..." 6 minutes ago Up 6 minutes 80/tcp
ecd9c5eb9c6 danieeld2-apache-lmage:p1 "apache2ctl -D FOREGR..." 6 minutes ago Up 6 minutes 80/tcp
4ee33d57e680 danieeld2-apache-lmage:p1 "apache2ctl -D FOREGR..." 6 minutes ago Up 6 minutes 80/tcp
6d6f7addb371 danieeld2-apache-lmage:p1 "apache2ctl -D FOREGR..." 6 minutes ago Up 6 minutes 80/tcp
be25dc889f9a danieeld2-apache-lmage:p1 "apache2ctl -D FOREGR..." 6 minutes ago Up 6 minutes 80/tcp
6d3e7b889916 danieeld2-apache-lmage:p1 "apache2ctl -D FOREGR..." 6 minutes ago Up 6 minutes 80/tcp
daniel-Modern-15-B7M: ~
→
```

También se ha probado a acceder a cada página web usando su dirección IP. Adjunto captura del proceso de prueba que se ha seguido:

```
+ Terminal
192.168.20.3. Set the 'ServerName' directive globally to suppress this message
web-8 | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using
192.168.20.4. Set the 'ServerName' directive globally to suppress this message
web-3 | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using
192.168.20.5. Set the 'ServerName' directive globally to suppress this message
web-2 | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using
192.168.20.5. Set the 'ServerName' directive globally to suppress this message
web-6 | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using
192.168.20.6. Set the 'ServerName' directive globally to suppress this message
web-5 | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using
192.168.20.7. Set the 'ServerName' directive globally to suppress this message
web-1 | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using
192.168.20.8. Set the 'ServerName' directive globally to suppress this message
^CGracefully stopping... (press Ctrl+C again to force)
[+] Stopping 1/0
✔ Container p1-web-1 Stopped 11.8s
✔ Container p1-web-2 Stopped 12.0s
✔ Container p1-web-7 Stopped 11.8s
✔ Container p1-web-8 Stopped 12.0s
✔ Container p1-web-4 Stopped 11.4s
✔ Container p1-web-3 Stopped 10.9s
✔ Container p1-web-5 Stopped 11.6s
✔ Container p1-web-6 Stopped 11.2s
canceled
daniel-Modern-15-B7M: ~/Glt/SWAP-UGR/P1
→ docker compose up
[+] Running 0/0
✔ Container p1-web-7 Created 0.0s
✔ Container p1-web-4 Created 0.0s
✔ Container p1-web-8 Created 0.0s
✔ Container p1-web-3 Created 0.0s
✔ Container p1-web-2 Created 0.0s
✔ Container p1-web-6 Created 0.0s
✔ Container p1-web-5 Created 0.0s
✔ Container p1-web-1 Created 0.0s
Attaching to web-1, web-2, web-3, web-4, web-5, web-6, web-7, web-8
web-7 | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using
192.168.20.2. Set the 'ServerName' directive globally to suppress this message
web-4 | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using
192.168.20.3. Set the 'ServerName' directive globally to suppress this message
web-8 | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using
192.168.20.4. Set the 'ServerName' directive globally to suppress this message
web-3 | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using
192.168.20.5. Set the 'ServerName' directive globally to suppress this message
web-2 | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using
192.168.20.5. Set the 'ServerName' directive globally to suppress this message
web-6 | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using
192.168.20.6. Set the 'ServerName' directive globally to suppress this message
web-5 | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using
192.168.20.7. Set the 'ServerName' directive globally to suppress this message
web-1 | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using
192.168.20.8. Set the 'ServerName' directive globally to suppress this message
+ Terminal
daniel-Modern-15-B7M: ~
→ docker exec -it p1-web-1 /bin/bash
root@6d3e7b889916:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.20.8 netmask 255.255.255.0 broadcast 192.168.20.255
    ether 02:42:c0:a8:14:08 txqueuelen 0 (Ethernet)
    RX packets 52 bytes 8216 (8.0 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 4 bytes 280 (280.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.10.8 netmask 255.255.255.0 broadcast 192.168.10.255
    ether 02:42:c0:a8:14:08 txqueuelen 0 (Ethernet)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 4 bytes 280 (280.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    loop txqueuelen 1000 (Local Loopback)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
root@6d3e7b889916:/# ping 192.168.10.7
PING 192.168.10.7 (192.168.10.7) 56(84) bytes of data.
64 bytes from 192.168.10.7: icmp_seq=1 ttl=64 time=0.119 ms
64 bytes from 192.168.10.7: icmp_seq=2 ttl=64 time=0.139 ms
^C
--- 192.168.10.7 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1020ms
rtt min/avg/max/ndev = 0.119/0.129/0.139/0.010 ms
root@6d3e7b889916:/# ping 192.168.20.7
PING 192.168.20.7 (192.168.20.7) 56(84) bytes of data.
64 bytes from 192.168.20.7: icmp_seq=1 ttl=64 time=0.088 ms
64 bytes from 192.168.20.7: icmp_seq=2 ttl=64 time=0.082 ms
^C
--- 192.168.20.7 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 2024ms
rtt min/avg/max/ndev = 0.082/0.088/0.094/0.005 ms
root@6d3e7b889916:/#
```

Para acceder a cada página, simplemente tenemos que usar `http://<Ip>:80`. En el caso de la captura, estoy usando `http://192.168.10.2:80`.

Nota: Había ocasiones en las que al hacer `docker compose up` se quedaba uno de los contenedores sin conectividad (parece que los arrancaba en paralelo e intentaba asignar la misma IP a dos y abortaba, pero esto no pasaba siempre, pasa solo en 1 de cada 15-20 ejecuciones). Se puede solventar usando `docker compose scale web=8`

2. Tareas Avanzadas

2.1. Personalización del Dockerfile

Como estoy cursando la asignatura de Tecnologías Web, donde usamos Apache y PHP, he decidido usar algunas configuraciones vistas en esta asignatura, con el fin de personalizar mi dockerfile.

2.1.1. Activar compresión de archivos de Apache

Para esta primera tarea, podemos usar el módulo `deflate` de Apache. También necesitaremos modificar el archivo de configuración de la herramienta. Vamos a empezar preguntándole a ChatGPT como hacerlo:

- **Yo:** Tengo un Dockerfile que usa una imagen de linux con Apache. ¿Como puedo activar y configurar la compresión de Archivos de Apache desde el propio Dockerfile?
- **ChatGPT:** Para activar y configurar la compresión de archivos en Apache desde un Dockerfile, puedes seguir estos pasos:

Abre o crea tu Dockerfile.

Asegúrate de que la imagen base de Linux que estás utilizando ya tenga Apache instalado. Si no es así, asegúrate de instalarlo adecuadamente en tu Dockerfile.

Una vez que tengas Apache instalado, puedes habilitar la compresión de archivos agregando o modificando la configuración del archivo de configuración de Apache (generalmente `httpd.conf` o `apache2.conf` dependiendo de la distribución de Linux) desde el Dockerfile.

Aquí hay un ejemplo de cómo podrías hacerlo en tu Dockerfile:

```
# Usa una imagen base de Linux con Apache instalado
FROM nombre_de_la_imagen_apache

# Copia tu archivo de configuración de Apache con la configuración
personalizada
COPY apache.conf /etc/apache2/apache.conf

# Instala módulos de Apache necesarios para la compresión de archivos
si es necesario
# RUN comando_para_instalar_módulos_de_apache

# Puedes configurar directamente en el archivo de configuración Apache,
o usar sed u otro comando para modificarlo directamente en el Dockerfile
# Ejemplo de configuración de compresión en el archivo apache.conf:
# RUN sed -i 's/#LoadModule deflate_module
modules/mod_deflate.so/LoadModule
deflate_module modules/mod_deflate.so/' /etc/apache2/apache2.conf
# RUN echo "AddOutputFilterByType DEFLATE text/html text/plain
text/xml" >> /etc/apache2/apache2.conf

# Asegúrate de exponer el puerto en el que Apache está escuchando
EXPOSE 80

# Inicia Apache
CMD ["apache2ctl", "-D", "FOREGROUND"]
```

Análisis de la salida: La propuesta en sí no me parece mala, pero vamos a simplificarlo un poco. Como tenemos ya construido la imagen del dockerfile, podemos correr un contenedor con dicha imagen, en modo iterativo, con el objetivo de copiar el contenido del fichero de configuración de **Apache**. De esta forma, podemos crearnos un nuevo archivo de configuración en nuestro equipo local, el cuál, vamos a ir modificando y añadiendo elementos de configuración (esto es simplemente para no cambiar la base del fichero de configuración). Una vez hecho esto, modificaremos el dockerfile, para activar el módulo que se encarga de la compresión, y, copiaremos el nuevo archivo de configuración.

Comenzamos creando una nueva carpeta `apacheConf_danieeeld2`, donde incluiremos el archivo de configuración que hemos copiado al ejecutar el contenedor en modo iterativo y usando el comando `cat`. Hecho esto, modificamos el archivo de configuración de **Apache**, añadiendo en la última línea lo siguiente:

```
<IfModule mod_deflate.c>
    AddOutputFilterByType DEFLATE text/html text/plain text/xml text/css
    text/javascript application/javascript application/x-javascript
    application/json
</IfModule>
```

Por último, modificamos el dockerfile, el cuál queda de la siguiente manera:

```
# Utilizamos una imagen linux liviana
FROM debian:bullseye-slim

# Actualizamos e instalamos apache y PHP. Ademas de herramientas de redes
RUN apt-get update && apt-get install -y apache2 php libapache2-mod-php
    iputils-ping net-tools

# Habilitamos el modulo de PHP y de compresion para Apache
RUN a2enmod php7.4 deflate

# Copiamos el archivo index.php a la carpeta /var/www/html
COPY web_danieeeld2/index.php /var/www/html

# Copiamos el archivo de configuracion de apache a la carpeta /etc/apache2
COPY apacheConf_danieeeld2/apache2.conf /etc/apache2/apache2.conf

# Eliminamos el archivo index.html, porque sino me mostraba este por defecto
RUN rm /var/www/html/index.html

# Exponemos el puerto 80
EXPOSE 80

# Iniciamos el servicio de apache
CMD ["apachectl", "-D", "FOREGROUND"]
```

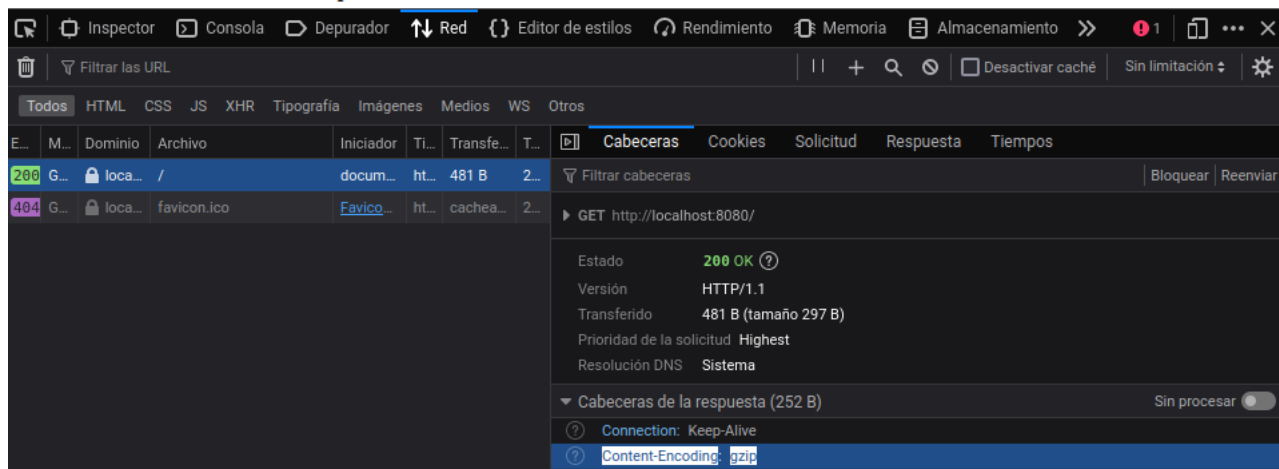
Podemos ahora construir, nuevamente, la imagen y ejecutarla de la siguiente forma:

```
$ docker build -t danieeeld2-apache-image:p1 -f DockerfileApache_danieeeld2 .
$ docker run -d -p 8080:80 danieeeld2-apache-image:p1
```

Para comprobar el correcto funcionamiento, podemos ir a las herramientas de desarrollador del navegador y ver que está activado **Content-Encoding gzip**. Adjunto captura para verificar esta tarea:

SWAP-danieeld2

La dirección IP del servidor Apache: 172.18.0.2



2.1.2. Configuración de encabezados HTTP

Otra configuración de Apache que vamos a hacer es la de las cabeceras HTTP, para lo cuál, empezamos preguntando a ChatGPT:

- **Yo:** Tengo un Dockerfile que usa una imagen de linux con Apache. ¿Como puedo activar y configurar las cabeceras HTTP de Apache desde el propio Dockerfile?
- **ChatGPT:** Para agregar cabeceras HTTP personalizadas en Apache, puedes hacerlo utilizando el archivo de configuración apache.conf dentro de tu contenedor Docker. Aquí te muestro cómo podrías incluir algunas cabeceras HTTP comunes para mejorar la seguridad y la experiencia del usuario:

```
# En el archivo apache.conf
<IfModule mod_headers.c>
    # Configuración de cabeceras HTTP adicionales

    # Evitar ataques de clics de navegador (X-Frame-Options)
    Header always set X-Frame-Options "SAMEORIGIN"

    # Habilitar la política de seguridad de contenido (Content-Security
-Policy)
    Header always set Content-Security-Policy "default-src 'self';
script-src 'self' 'unsafe-inline' 'unsafe-eval'; style-src 'self'
'unsafe-inline'; img-src 'self' data;";

    # Evitar la detección de tipo de contenido (X-Content-Type-Options)
    Header always set X-Content-Type-Options "nosniff"

    # Evitar la inyección de scripts (X-XSS-Protection)
    Header always set X-XSS-Protection "1; mode=block"

    # Habilitar la política de seguridad de transporte estricta
(Strict-Transport-Security)
```

```
Header always set Strict-Transport-Security "max-age=31536000;  
includeSubDomains; preload"  
</IfModule>
```

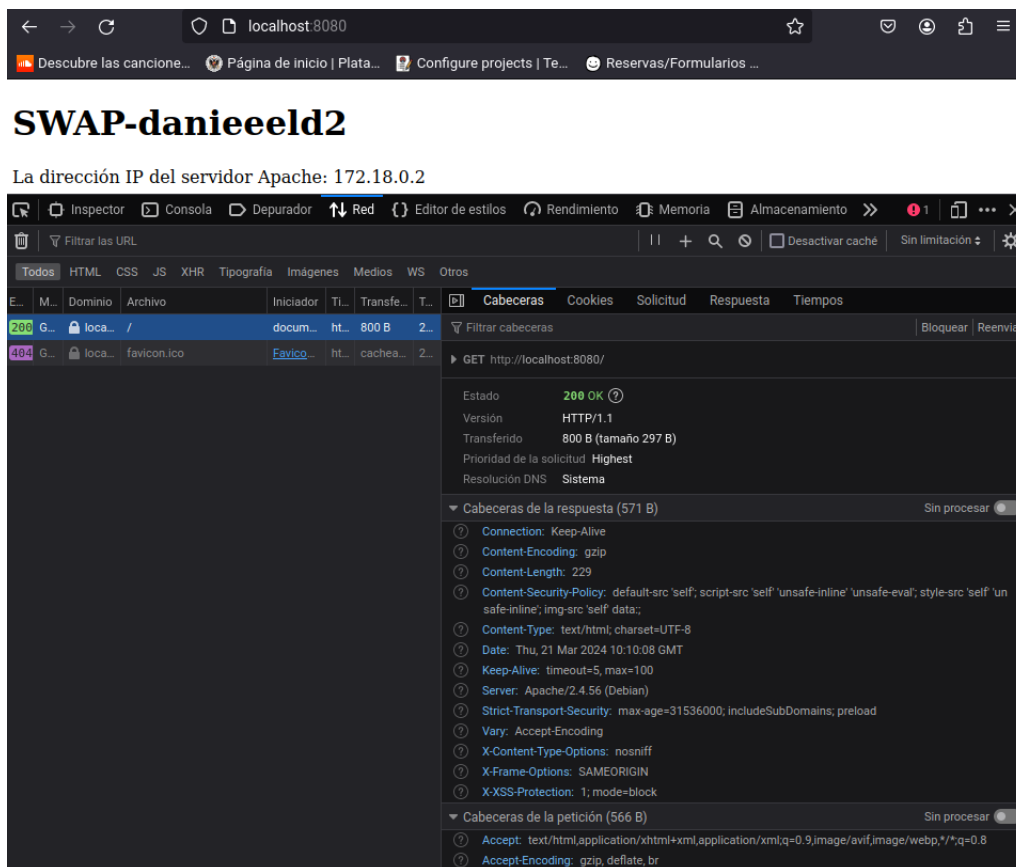
Luego, añade la siguiente línea a tu archivo dockerfile: `RUN a2enmod headers`.

Nota: He acertado un poco la salida, porque volvía a explicar cómo se copia el archivo de configuración, además de extenderse en la explicación de las cabeceras que usa.

Análisis de la salida: Esta vez, parece bastante acertado, por lo que vamos a hacerle caso. Copiamos la configuración de las cabeceras al final del archivo de configuración de Apache que creamos previamente y habilitamos el módulo `headers` en el dockerfile. El aspecto final del dockerfile es el siguiente:

```
# Utilizamos una imagen linux liviana  
FROM debian:bullseye-slim  
  
# Actualizamos e instalamos apache y PHP. Ademas de herramientas de redes  
RUN apt-get update && apt-get install -y apache2 php libapache2-mod-php iputils-ping  
net-tools  
  
# Habilitamos el modulo de PHP, compresion para Apache y uso de cabeceras HTTP  
RUN a2enmod php7.4 deflate headers  
  
# Copiamos el archivo index.php a la carpeta /var/www/html  
COPY web_danieeeld2/index.php /var/www/html  
  
# Copiamos el archivo de configuracion de apache a la carpeta /etc/apache2  
COPY apacheConf_danieeeld2/apache2.conf /etc/apache2/apache2.conf  
  
# Eliminamos el archivo index.html, porque sino me mostraba este por defecto  
RUN rm /var/www/html/index.html  
  
# Exponemos el puerto 80  
EXPOSE 80  
  
# Iniciamos el servicio de apache  
CMD ["apachectl", "-D", "FOREGROUND"]
```

Podemos volver a construir la imagen y ejecutarla, como acabamos de ver en la tarea anterior. Abriendo las herramientas para desarrolladores del navegador, podemos ver que, efectivamente, funciona:



2.2. Creación de contenedores con otros servidores web

2.2.1. Contenedor con nginx

Como este ejercicio es un poco más fácil, voy a ir más rápido que en el resto. Comenzamos preguntándole a ChatGPT:

- **Yo:** Dado el siguiente fichero dockerfile: *Le pasé el fichero que teníamos creado con Apache. ¿Cómo replico el proceso, pero usando un servidor Nginx?*
- **ChatGPT:** Para lograr lo mismo utilizando Nginx en lugar de Apache, puedes seguir una estructura similar en tu Dockerfile. Aquí tienes una versión modificada del Dockerfile que utiliza Nginx en lugar de Apache:

```
# Utilizamos una imagen Linux liviana con Nginx
FROM nginx:latest

# Actualizamos el índice de paquetes e instalamos PHP y otras herramientas
RUN apt-get update && apt-get install -y php-fpm php-mysql
iputils-ping net-tools

# Copiamos el archivo index.php a la carpeta /usr/share/nginx/html
COPY web_danieeeld2/index.php /usr/share/nginx/html

# Eliminamos el archivo index.html para evitar que se muestre por defecto
RUN rm /usr/share/nginx/html/index.html
```

```

# Copiamos el archivo de configuracion de Nginx personalizado
COPY nginxConf_danieeeeld2/default.conf /etc/nginx/conf.d/default.conf

# Exponemos el puerto 80
EXPOSE 80

# Iniciamos el servicio de Nginx
CMD ["nginx", "-g", "daemon off;"]

```

Análisis de la salida: La idea no parece mala, pero es cierto qué podemos ahorrarnos instalar algunos paquetes, como `net-tools`, ya que esta vez solo quiero que me muestre la página. Además, no nos proporciona el fichero de configuración de `Nginx`, por lo que tendremos que crearlo nosotros mismos:

```

# Fichero básico de configuración
server {
    listen 80;
    server_name localhost;

    root /var/www/html;
    index index.php index.html index.htm;

    location / {
        try_files $uri $uri/ /index.php?$query_string;
    }

    location ~ \.php$ {
        include snippets/fastcgi-php.conf;
        fastcgi_pass unix:/var/run/php/php7.4-fpm.sock;
        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
        include fastcgi_params;
    }

    location ~ /\.ht {
        deny all;
    }
}

```

Este fichero lo he sacado de un seminario que hice en la asignatura Tecnologías Web hace poco. Normalmente, los recursos de `Nginx` se administran desde una carpeta en `\usr\share`, pero he decidido que busque en el mismo sitio que `Apache` para hacerlo lo más semejante posible a lo que teníamos (*Debido a esto, no tenemos que eliminar el archivo que usa por defecto, ya que este se encuentra en otra ubicación*). Además, pese a que la solución de `ChatGPT` es buena, he decidido estructurar el `dockerfile` de la siguiente forma:

```

# Utilizamos una imagen Linux liviana
FROM debian:bullseye-slim

# Actualizamos el índice de paquetes e instalamos Nginx y PHP
RUN apt-get update && \
    apt-get install -y nginx php-fpm

```

```
# Copiamos el archivo index.php a la carpeta /var/www/html
COPY web-danieeeld2/index.php /var/www/html

# Copiamos el archivo de configuracion de Nginx personalizado
COPY nginxConf-danieeeld2/default.conf /etc/nginx/sites-available/default

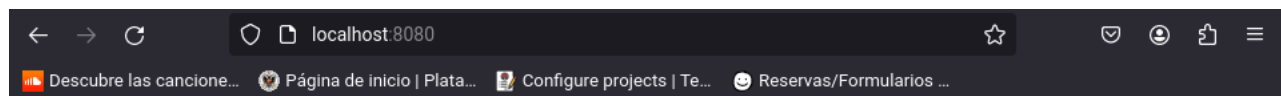
# Exponemos el puerto 80
EXPOSE 80

# Iniciamos el servicio de Nginx y PHP-FPM
CMD service php7.4-fpm start && nginx -g 'daemon off;'
```

Podemos levantarlo siguiendo los comandos de siempre:

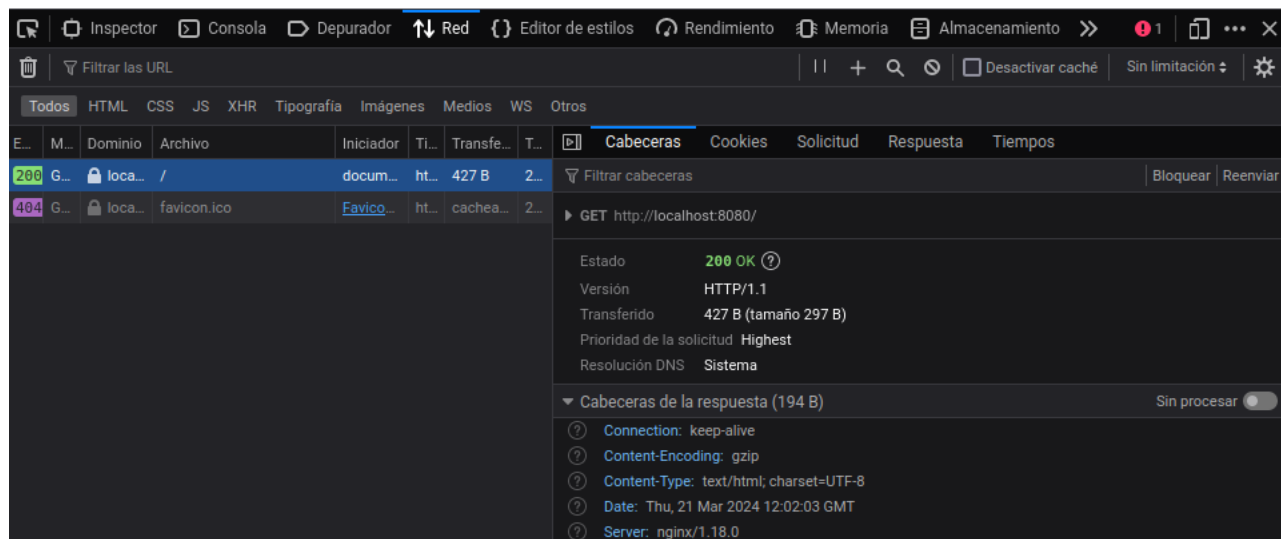
```
$ docker build -t danieeeld2-nginx-image:p1 -f DockerfileNginx-danieeeld2 .
$ docker run -d -p 8080:80 danieeeld2-nginx-image:p1
```

y, al entrar en el servidor e inspeccionar con la herramienta para desarrolladores, vemos que teníamos lo mismo de antes, pero, esta vez, corriendo en un servidor Nginx.



SWAP-danieeeld2

La dirección IP del servidor Apache: 172.18.0.2



2.2.2. Contenedor con lighttpd

Repetimos un poco el procedimiento hecho con el ejercicio anterior:

- **Yo:** Dado el siguiente fichero dockerfile: *Le pasé el fichero que teníamos creado con Apache. ¿Cómo replico el proceso, pero usando un servidor lighttpd?*
- **ChatGPT:** Para adaptar el Dockerfile para usar lighttpd en lugar de Apache, puedes seguir un proceso similar al que has utilizado con Apache. Aquí tienes una versión modificada del Dockerfile que utiliza lighttpd en lugar de Apache:

```

# Utilizamos una imagen linux liviana
FROM debian:bullseye-slim

# Actualizamos e instalamos lighttpd y PHP
RUN apt-get update && apt-get install -y lighttpd php7.4-cgi

# Habilitamos el módulo FastCGI para PHP en lighttpd
RUN lighty-enable-mod fastcgi-php

# Copiamos el archivo index.php a la carpeta /var/www/html
COPY web_danieeeld2/index.php /var/www/html

# Copiamos el archivo de configuración de lighttpd a la carpeta /etc/lighttpd
COPY lighttpdConf_danieeeld2/lighttpd.conf /etc/lighttpd/lighttpd.conf

# Exponemos el puerto 80
EXPOSE 80

# Iniciamos el servicio de lighttpd
CMD ["lighttpd", "-D", "-f", "/etc/lighttpd/lighttpd.conf"]

```

Análisis de la salida: Esta vez parece que ha optado por respetar la estructura original que teníamos, en vez de usar una imagen nueva. Por otro lado, sigue sin indicarnos el archivo de configuración, por lo que tendremos que hacerlo nosotros. Por suerte, también tengo uno de la asignatura Tecnologías Web hecha en esta herramienta.

```

server.modules = (
    "mod_access",
    "mod_alias",
    "mod_compress",
    "mod_redirect",
    "mod_fastcgi",
    "mod_rewrite"
)

server.document-root = "/var/www/html"
server.upload-dirs = ( "/var/cache/lighttpd/uploads" )
server.errorlog = "/var/log/lighttpd/error.log"
server.pid-file = "/var/run/lighttpd.pid"
server.username = "www-data"
server.groupname = "www-data"

index-file.names = ( "index.php", "index.html",
                    "index.htm", "default.htm" )

static-file.exclude-extensions = ( ".php", ".pl", ".fcgi" )

compress.cache-dir = "/var/cache/lighttpd/compress/"
compress.filetype = ( "application/javascript", "text/css",

```

```
"text/html", "text/plain" )

# FastCGI module settings
fastcgi.server = ( ".php" =>
    (( "bin-path" => "/usr/bin/php-cgi",
       "socket" => "/tmp/php.socket" )) )
```

Se trata de un fichero de configuración muy básico para que ejecute PHP. En cuanto al dockerfile final, el resultado es:

```
# Utilizamos una imagen linux liviana
FROM debian:bullseye-slim

# Actualizamos e instalamos lighttpd y PHP
RUN apt-get update && apt-get install -y lighttpd php7.4-cgi

# Habilitamos el modulo FastCGI para PHP en lighttpd
RUN lighty-enable-mod fastcgi-php

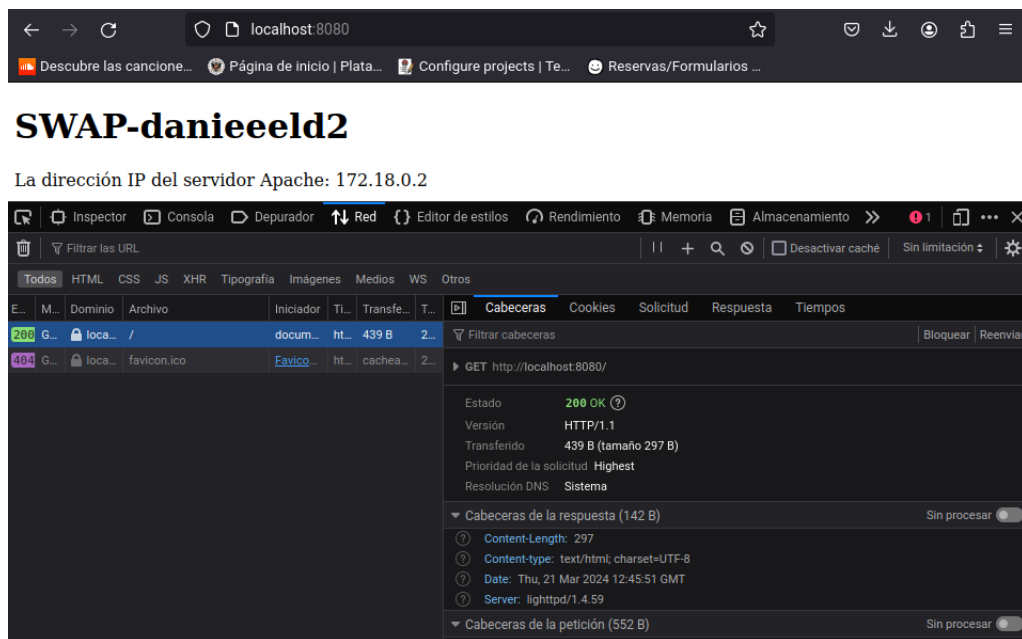
# Copiamos el archivo index.php
COPY web_danieeeld2/index.php /var/www/html

# Copiamos el archivo de configuracion de lighttpd
COPY lighttpdConf_danieeeld2/lighttpd.conf /etc/lighttpd/lighttpd.conf

# Exponemos el puerto 80
EXPOSE 80

# Iniciamos el servicio de lighttpd
CMD [ "lighttpd", "-D", "-f", "/etc/lighttpd/lighttpd.conf" ]
```

Repitiendo el mismo proceso de ejecución que los ejercicios anteriores, obtenemos el siguiente resultado:



2.3. Gestión Avanzada de Redes

Como no se me ocurría nada en primera instancia, comencé preguntándole directamente a ChatGPT por algo más genérico:

- **Yo:** Tengo contenedores con imágenes linux, en las cuales he instalado Apache y PHP. Cada contenedor esta conectado a dos subredes llamadas `p1_red_web` y `p1_red_servicios`. Me piden Configurar reglas específicas de enrutamiento o restricciones de acceso entre las dos redes `red_web` y `red_servicios`.
ChatGPT: Para configurar reglas específicas de enrutamiento o restricciones de acceso entre las dos subredes `p1_red_web` y `p1_red_servicios`, puedes utilizar herramientas como `iptables` en Linux. Aquí te proporciono un ejemplo básico de cómo podrías configurar estas reglas en tus contenedores.

Supongamos que quieres permitir que los contenedores en la subred `p1_red_web` puedan acceder a los servicios en la subred `p1_red_servicios`, pero no al revés. Puedes hacerlo con `iptables` de la siguiente manera:

1. Permitir el tráfico saliente desde `p1_red_web` hacia `p1_red_servicios`:

```
iptables -A FORWARD -i p1_red_web -o p1_red_servicios -j ACCEPT
```

2. Denegar el tráfico entrante desde `p1_red_servicios` hacia `p1_red_web`:

```
iptables -A FORWARD -i p1_red_servicios -o p1_red_web -j REJECT
```

Estas reglas permitirán que los contenedores en `p1_red_web` se comuniquen con los servicios en `p1_red_servicios`, pero bloquearán cualquier intento de comunicación desde `p1_red_servicios` hacia `p1_red_web`.

Análisis de la salida: Como primer fallo, encontramos que no te dice el paquete necesario para usar esta herramienta (Tenemos que descargar `iptables`). La salida es un poco escueta, ya que el prompt introducido es muy general, aunque, a raíz de esto, se me ha ocurrido una buena idea. Vamos a hacer una script que permita denegar o permitir el tráfico de una red a otra.

Nota: *Estuve bastante rato intentando hacer esta gestión a nivel de contenedor, pero el kernel de la imagen está protegido y, pese a ser root, no me dejaba hacer configuraciones de red avanzadas, tanto con `iptables` como con `ufw`. Además, tampoco contaba con `sudo` instalado. Estuve muchísimo rato preguntándole a `ChatGPT`, pero no me daba una respuesta correcta. Me repetía una y otra vez usa `sudo` o revisa el kernel, etc... Al final, tuve que investigar bastante por mi cuenta y acabé dando con la clave, que es usar `privileged: true` en el `docker compose`.*

Al final modifiqué el fichero del `docker compose` y quedó de la siguiente manera:

```
...
web:
  image: danieeeld2-apache-image:p1
  privileged: true
  deploy:
    replicas: 8
  ...
```

Por lo que, ahora, ya podemos hacer ciertas gestiones interesantes a nivel de redes.

Si usamos el comando `ifconfig`, veremos que tenemos las interfaces de puente creada por `docker-compose` en nuestro host. La salida debería ser algo tal que así:


```
br-0c9b3ca932c9: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.10.1 netmask 255.255.255.0 broadcast 192.168.10.255
    inet6 fe80::42:4bff:fee5:e182 prefixlen 64 scopeid 0x20<link>
    ether 02:42:4b:e5:e1:82 txqueuelen 0 (Ethernet)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 24 bytes 3393 (3.3 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

br-f877a6a1bf01: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.20.1 netmask 255.255.255.0 broadcast 192.168.20.255
    inet6 fe80::42:4ff:fedc:6013 prefixlen 64 scopeid 0x20<link>
    ether 02:42:04:dc:60:13 txqueuelen 0 (Ethernet)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 29 bytes 3987 (3.9 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

Esto se debe al uso del **driver:bridge** a la hora de crear las redes. Vamos a tratar de crear una tercera interfaz para que las dos subredes tengan conexión entre ellas.

- **Yo:** Tengo dos redes creadas con docker compose usando driver:bridge. Quiero conectar dichas dos interfaces en mi host mediante el uso de una tercera interfaz, para que así las dos subredes tengan comunicación **ChatGPT:** Crear la tercera interfaz de red en el host: Puedes hacerlo utilizando herramientas como ip o ifconfig, dependiendo de tu sistema operativo. Por ejemplo, si estás utilizando Linux:

```
sudo ip link add name br3 type bridge
sudo ip addr add 192.168.100.1/24 dev br3
sudo ip link set dev br3 up
```

Conectar las redes Docker a la tercera interfaz del host: Utiliza los comandos docker network connect para conectar las redes Docker a la nueva interfaz de red en tu host. Por ejemplo:

```
sudo docker network connect --ip 192.168.100.2 br1 br3
sudo docker network connect --ip 192.168.100.3 br2 br3
```

Análisis de la salida: La primera parte, que es la creación de la nueva interfaz, me parece casi correcta, porque el dispositivo que crea es de tipo **bridge** y no se pueden conectar directamente dos dispositivos de este tipo, aunque intentaremos seguir sus instrucciones, con el fin de ver donde llegamos (Por otro lado, usaremos la 198.168.30.1/24. Por otro lado, interpreta mal mi idea e intenta conectar los contenedores a esta nueva red, cuando lo que queremos es conectar solo las interfaces en el host. Para ello, le volvemos a preguntar:

- **Yo:** Quiero conectar en mi host las dos interfaces de bridge a esta nueva, para que se comuniquen entre ellas siguiendo un enrutamiento por el host. Además, dichas interfaces cuentan con Ips 198.168.10.1/24 y 198.168.20.1/24. Además, el nombre de las interfaces son br-f877a6a1bf01 y br-f877a6a1bf01, respectivamente

- **ChatGPT:** Entiendo, parece que tienes tres interfaces de puente en tu sistema, br-0c9b3ca932c9, br-f877a6a1bf01 y br3, y deseas conectar las dos primeras a la tercera (br3).

Puedes hacerlo agregando las interfaces br-0c9b3ca932c9 y br-f877a6a1bf01 a br3. Aquí tienes cómo hacerlo en Linux:

```
sudo brctl addif br3 br-0c9b3ca932c9
sudo brctl addif br3 br-f877a6a1bf01
```

Esto conectará las interfaces br-0c9b3ca932c9 y br-f877a6a1bf01 a br3. Ahora, cualquier tráfico que llegue a br3 será transmitido a través de estas interfaces también.

Ten en cuenta que brctl es una herramienta para configurar los puentes de red en Linux. Asegúrate de tenerla instalada en tu sistema si aún no lo está. Además, este enlace es temporal y no persistirá después de reiniciar el sistema. Si deseas que esta configuración persista, deberás agregar estos comandos a tus scripts de inicio o configurarlos mediante otros medios dependiendo de tu distribución de Linux.

Análisis de la respuesta: Esta vez me convence la salida, pero seguramente de fallo como consecuencia de la decisión primera. Vamos a probarlo:

```
daniel-Modern-15-B7M: ~/Git/SWAP-UGR/P1 |main U:2 X|
→sudo brctl addif br3 br-0c9b3ca932c9
device br-0c9b3ca932c9 is a bridge device itself; can't add a bridge device to a bridge device.
daniel-Modern-15-B7M: ~/Git/SWAP-UGR/P1 |main U:2 X|
→
```

Efectivamente, da fallo.

Nota: Como este apartado era más corto, quería introducir algunas conversaciones extras con la IA, para ver como muchas veces nos da respuestas erróneas como consecuencia de que le pasamos muy poco contexto, no expresamos del todo clara la idea o pasamos información innecesaria. Vamos a eliminar la interfaz creada con `sudo ip link delete br3` e intentaremos este nuevo planteamiento.

Vamos a probar a simplificar el ejercicio al máximo (Parte de la culpa es mía, porque no estaba plantando bien el ejercicio):

- **Yo:** Tengo dos redes, la red_web (192.168.10.0/24) y la red_servicios (192.168.20.0/24). Por otro lado, tengo dos contenedores docker conectados a ambas redes y, en mi máquina anfitrión, dos puentes (interfaces) con IP 192.168.10.1 y 192.168.20.1. Quiero que, al desconectar el contenedor 1 de la subred red_web y el contenedor 2 de la subred red_servicios, sigan teniendo conexión.
- **ChatGPT:** Para lograr que los contenedores mantengan la conexión incluso después de desconectarlos de sus respectivas subredes, puedes utilizar Docker y configurar enrutamiento y reglas de iptables en tu máquina anfitriona. Aquí te muestro los pasos generales para lograr esto:

Conectar los contenedores a las subredes deseadas inicialmente. Configurar las reglas de enrutamiento y iptables en la máquina anfitriona para permitir la comunicación entre los contenedores. Desconectar los contenedores de sus subredes originales. Actualizar las reglas de iptables para que los contenedores todavía puedan comunicarse.

Aquí tienes un ejemplo básico de cómo podrías lograr esto en un sistema Linux:

Supongamos que los contenedores Docker están conectados a las subredes red_web y red_servicios, y que las interfaces de puente en la máquina anfitriona son br-red_web y br-red_servicios con IPs 192.168.10.1 y 192.168.20.1 respectivamente.

```
# Conectar los contenedores a las subredes:
docker network connect red_web container1
docker network connect red_servicios container2
# Configurar reglas de enrutamiento y iptables en la máquina anfitriona:
# Permitir el enrutamiento IP
echo 1 > /proc/sys/net/ipv4/ip_forward

# Añadir reglas iptables para permitir la comunicación entre contenedores
iptables -I FORWARD -i br-red_web -o br-red_servicios -j ACCEPT
iptables -I FORWARD -i br-red_servicios -o br-red_web -j ACCEPT
# Desconectar los contenedores de sus subredes originales:
docker network disconnect red_web container1
docker network disconnect red_servicios container2
```

Análisis de la salida: Ahora que he planteado mejor el ejercicio, la salida es más correcta y se entiende mejor la idea de ejecución que pretendo. Vamos a ponerlo a prueba para dos contenedores. Primero, permitiremos el tráfico entre los puentes de la máquina anfitrión:

The image shows two terminal windows side-by-side. The left window shows the configuration of iptables rules to allow forwarding between two bridges, and the right window shows the execution of a traceroute command to verify connectivity between two containers.

```
daniel@Modern-15-B7M: ~/Glt/SWAP-UGR/P1 [main ~]
→ cat /proc/sys/net/ipv4/ip_forward
1

daniel@Modern-15-B7M: ~/Glt/SWAP-UGR/P1 [main ~]
→ sudo iptables -I FORWARD -i br-53be3fe58d98 -o br-1097b844e129 -j ACCEPT

daniel@Modern-15-B7M: ~/Glt/SWAP-UGR/P1 [main ~]
→ sudo iptables -I FORWARD -i br-1097b844e129 -o br-53be3fe58d98 -j ACCEPT

daniel@Modern-15-B7M: ~/Glt/SWAP-UGR/P1 [main ~]
→ docker network disconnect
bridge      host      p1_red_servicios red1
dai_default none      p1_red_web       red2

daniel@Modern-15-B7M: ~/Glt/SWAP-UGR/P1 [main ~]
→ docker network disconnect p1_re
p1_red_servicios p1_red_web

daniel@Modern-15-B7M: ~/Glt/SWAP-UGR/P1 [main ~]
→ docker network disconnect p1_red_web p1-web-1

daniel@Modern-15-B7M: ~/Glt/SWAP-UGR/P1 [main ~]
→ docker network disconnect p1_red_servicios p1-web-2

daniel@Modern-15-B7M: ~/Glt/SWAP-UGR/P1 [main ~]
→ docker exec -it p1-web-1 /bin/bash
root@bac65948dde:/# ifconfig
eth3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.20.2 netmask 255.255.255.0 broadcast 192.168.20.255
    inet6 fe80::42:coff:feab:1402 prefixlen 64 scopeid 0x20<link>
    ether 02:42:co:ab:14:02 txqueuelen 0 (Ethernet)
    RX packets 40 bytes 4562 (4.4 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 12 bytes 936 (936 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@bac65948dde:/# ping 192.168.10.3
PING 192.168.10.3 (192.168.10.3) 56(84) bytes of data:
64 bytes from 192.168.10.3: icmp_seq=1 ttl=63 time=0.250 ms
64 bytes from 192.168.10.3: icmp_seq=2 ttl=63 time=0.161 ms
64 bytes from 192.168.10.3: icmp_seq=3 ttl=63 time=0.185 ms
^C
--- 192.168.10.3 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2053ms
```

```
daniel@Modern-15-B7M: ~/Glt/SWAP-UGR/P1 [main ~]
→ docker exec -it p1-web-1 traceroute 192.168.10.3
traceroute to 192.168.10.3 (192.168.10.3), 30 hops max, 60 byte packets
 1 daniel@Modern-15-B7M (192.168.20.1)  0.102 ms  0.015 ms  0.013 ms
 2 192.168.10.3 (192.168.10.3)  0.058 ms  0.024 ms  0.034 ms
→
```

Como vemos, el tráfico llega correctamente. Vamos a probar ahora a impedir el tráfico:

```
Terminal
daniel-Modern-15-B7M: ~/Glt/SWAP-UGR/P1 [main ✓]
→ sudo iptables -I FORWARD -i br-53be3fe58d98 -o br-1097b844e129 -j DROP
daniel-Modern-15-B7M: ~/Glt/SWAP-UGR/P1 [main ✓]
→ sudo iptables -I FORWARD -i br-1097b844e129 -o br-53be3fe58d98 -j DROP
daniel-Modern-15-B7M: ~/Glt/SWAP-UGR/P1 [main ✓]
→ docker exec -it pi-web-1 /bin/bash
root@0ac656948dde:/# ping 192.168.10.3
PING 192.168.10.3 (192.168.10.3) 56(84) bytes of data.
^C
--- 192.168.10.3 ping statistics ---
22 packets transmitted, 0 received, 100% packet loss, time 21505ms
root@0ac656948dde:/#

Terminal
daniel-Modern-15-B7M: ~/Glt/SWAP-UGR/P1 [main ✓]
→ docker exec -it pi-web-1 traceroute 192.168.10.3
traceroute to 192.168.10.3 (192.168.10.3), 30 hops max, 60 byte packets
 1 daniel-Modern-15-B7M (192.168.20.1)  0.096 ms  0.019 ms  0.017 ms
 2 * * *
 3 * * *
 4 * * *
 5 * * *
 6 * * *
 7 * * *
 8 * * *
 9 * * *
10 * * *
11 * * *
12 * * *
13 * * *
14 * * *
15 * * *
16 * * *
17 * * *
18 * * *
19 * * *
20 * * *
21 * * *
22 * * *
23 * * *
24 * * *
25 * * *
26 * * *
27 * * *
28 * * *
29 * * *
30 * * *
```

Ahora, podemos observar, claramente con `traceroute`, como el tráfico es denegado. Esto podríamos hacerlo para cualquier par de contenedores o, plantear escenarios donde, por ejemplo; los contenedores pares estuvieran conectados solo a una de las dos redes y los impares a la otra.

Este ejemplo que acabamos de ver puede ser muy interesante para la comunicación entre micro-servicios. Por ejemplo, acceder desde contenedores, que cuentan con redes públicas de comunicación, a la red privada de, por ejemplo, una base de datos.

2.4. Automatización con Scripts

2.4.1. Mantenimiento

Hasta ahora, he tratado siempre de usar la misma imagen base. Esto nos permitirá que las scripts que hagamos sean válidas para todos nuestros contenedores. Para ello, vamos a preguntarle a **ChatGPT** por ideas y que nos las desarrolle:

- **Yo:** Estoy trabajando con dockerfile. ¿Puedes darme ideas para scripts que realicen tareas de mantenimiento automatizado, como limpieza de logs, monitoreo de la salud del contenedor, o actualizaciones automáticas de paquetes?
- **ChatGPT:** Para crear scripts para tareas de mantenimiento automatizado, como limpieza de logs, monitoreo de la salud del contenedor o actualizaciones automáticas de paquetes, puedes utilizar varios lenguajes de scripting como Bash, Python, PowerShell, entre otros. Aquí te proporcionaré un enfoque utilizando Bash y Python como ejemplos:

```
#!/bin/bash
```

```
# Limpiar logs que tengan más de 7 días de antigüedad
find /ruta/a/logs -type f -mtime +7 -exec rm {} \;
```

```
#####
```

```
#!/bin/bash

# Obtener el ID y estado de todos los contenedores
containers_info=$(docker ps --format "{{.ID}}:{{.State}}")

# Iterar sobre cada contenedor
while IFS= read -r line; do
    container_id=$(echo "$line" | cut -d ':' -f1)
    container_state=$(echo "$line" | cut -d ':' -f2)

    echo "Container ID: $container_id"
    echo "State: $container_state"

    # Aquí puedes agregar lógica para tomar acciones basadas en el
    # estado del contenedor
    # Por ejemplo, enviar una notificación si el estado no es 'running'

done <<< "$containers_info"

#####

#!/bin/bash

# Actualizar paquetes del sistema
apt-get update
apt-get upgrade -y
```

Después comenzó a hablar de cómo copiar estos ficheros al contenedor, etc...

Análisis de la salida: La verdad, que las ideas que da son bastante buenas, pero la ejecución no me parece la correcta. Aprovechando que tenemos la misma imagen base en todos los contenedores, lo suyo sería ejecutar estas scripts desde fuera del contenedor. De esta forma, podemos programar tareas con **crontab** para que ejecuten una determinada script periódicamente o hacer una script que ejecute las otras scripts sobre los contenedores que tenemos levantados. Por simplicidad, voy a hacer lo segundo ahora como ejemplo, para lo que usaré las scripts en **bash** de antes (en **python** son más versátiles, pero tienes que tener en cuenta las dependencias)

```
#!/bin/bash

# Verificar que haya 1 argumento
if [ $# -eq 0 ]; then
    echo "Uso: $0 <script_a_ejecutar>"
    exit 1
fi

# Obtener el script a ejecutar
script="$1"

# Obtener IDs de los contenedores activos
containers=$(docker ps -q)
```

```
# Iterar sobre los IDs de los contenedores y ejecutar el script en cada uno (copiando
la script en /tmp)
for container in $containers; do
    echo "Copiando el script $script al contenedor $container"
    docker cp "$script" "$container:/tmp/"
    echo "Ejecutando el script $script en el contenedor $container"
    docker exec "$container" sh -c "chmod +x /tmp/${basename $script} && /tmp/${
basename $script}"
done
```

Veamos un ejemplo de uso sobre el entorno que tenemos montado con **docker-compose** y la script de actualización de paquetes:

```
daniel-Modern-35-B7M: ~/Glt/SWAP-UGR/P1
→ docker compose up
WARN[0000] Found orphan containers ([p1-prometheus-1]) for this project. If you removed or renamed this
service in your compose file, you can run this command with the --remove-orphans flag to clean it up.
(1) Running 0/0
✓ Container p1-web-4 Created 0.0s
✓ Container p1-web-7 Created 0.0s
✓ Container p1-web-8 Created 0.0s
✓ Container p1-web-6 Created 0.0s
✓ Container p1-web-3 Created 0.0s
✓ Container p1-web-2 Created 0.0s
✓ Container p1-web-5 Created 0.0s
✓ Container p1-web-1 Created 0.0s
Attaching to web-1, web-2, web-3, web-4, web-5, web-6, web-7, web-8
web-4 | AH00558: apache2: could not reliably determine the server's fully qualified domain name, using
192.168.10.2. Set the 'ServerName' directive globally to suppress this message
web-7 | AH00558: apache2: could not reliably determine the server's fully qualified domain name, using
192.168.20.3. Set the 'ServerName' directive globally to suppress this message
web-8 | AH00558: apache2: could not reliably determine the server's fully qualified domain name, using
192.168.10.4. Set the 'ServerName' directive globally to suppress this message
web-6 | AH00558: apache2: could not reliably determine the server's fully qualified domain name, using
192.168.20.5. Set the 'ServerName' directive globally to suppress this message
web-3 | AH00558: apache2: could not reliably determine the server's fully qualified domain name, using
192.168.10.6. Set the 'ServerName' directive globally to suppress this message
web-2 | AH00558: apache2: could not reliably determine the server's fully qualified domain name, using
192.168.10.7. Set the 'ServerName' directive globally to suppress this message
web-5 | AH00558: apache2: could not reliably determine the server's fully qualified domain name, using
192.168.20.7. Set the 'ServerName' directive globally to suppress this message
web-1 | AH00558: apache2: could not reliably determine the server's fully qualified domain name, using
192.168.20.8. Set the 'ServerName' directive globally to suppress this message

daniel-Modern-35-B7M: ~/Glt/SWAP-UGR/P1
→ ./ejecutarScript.sh scripts/actualizar.sh
Copiando el script scripts/actualizar.sh al contenedor 4bf75f7a42d7
Successfully copied 2.05kB to 4bf75f7a42d7:/tmp/
Ejecutando el script scripts/actualizar.sh en el contenedor 4bf75f7a42d7
Hit:1 http://deb.debian.org/debian bullseye InRelease
Hit:2 http://deb.debian.org/debian-security bullseye-security InRelease
Hit:3 http://deb.debian.org/debian bullseye-updates InRelease
Reading package lists...
Building dependency tree...
Reading state information...
Calculating upgrade...
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
Copiando el script scripts/actualizar.sh al contenedor 142bb59c3832
Successfully copied 2.05kB to 142bb59c3832:/tmp/
Ejecutando el script scripts/actualizar.sh en el contenedor 142bb59c3832
Hit:1 http://deb.debian.org/debian bullseye InRelease
Hit:2 http://deb.debian.org/debian-security bullseye-security InRelease
Hit:3 http://deb.debian.org/debian bullseye-updates InRelease
Reading package lists...
Building dependency tree...
Reading state information...
Calculating upgrade...
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
Copiando el script scripts/actualizar.sh al contenedor 390e8c6a5904
Successfully copied 2.05kB to 390e8c6a5904:/tmp/
Ejecutando el script scripts/actualizar.sh en el contenedor 390e8c6a5904
Hit:1 http://deb.debian.org/debian bullseye InRelease
Hit:2 http://deb.debian.org/debian-security bullseye-security InRelease
Hit:3 http://deb.debian.org/debian bullseye-updates InRelease
Reading package lists...
Building dependency tree...
Reading state information...
Calculating upgrade...
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
Copiando el script scripts/actualizar.sh al contenedor 97d27b90d0ff
Successfully copied 2.05kB to 97d27b90d0ff:/tmp/
Ejecutando el script scripts/actualizar.sh en el contenedor 97d27b90d0ff
Hit:1 http://deb.debian.org/debian bullseye InRelease
Hit:2 http://deb.debian.org/debian-security bullseye-security InRelease
Hit:3 http://deb.debian.org/debian bullseye-updates InRelease
Reading package lists...
Building dependency tree...
Reading state information...
Calculating upgrade...
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
Copiando el script scripts/actualizar.sh al contenedor 1ac889850091
Successfully copied 2.05kB to 1ac889850091:/tmp/
Ejecutando el script scripts/actualizar.sh en el contenedor 1ac889850091
Hit:1 http://deb.debian.org/debian bullseye InRelease
```

Ahora la script para monitorizar (**Ojo:** Esta ha de ejecutarse en nuestro sistema, no en cada contenedor y se podría ampliar para que se ejecute periódicamente con **Crontab** y que si algún contenedor pasa de estar activo a inactivo entre ejecuciones, mande una notificación o una alarma al administrador)

2.4.2. Creación de Contenedores y Configuración de Redes

Ahora vamos a realizar una automatización para que las IPs salgan ordenadas, como en el esquema. Para ello, comenzamos preguntando a ChatGPT:

- **Yo:** Tengo el siguiente docker-compose: *Le pasé el docker-compose entero*. Crea una script que elimine las ips asignadas en cada subred a cada contenedor y le asigne la ip cuya parte de host coincide con su identificador. Cada contenedor se llama p1-web-X donde X es un entero que es su identificador
- **ChatGPT:** Puedes lograrlo utilizando Docker CLI y un script de shell. Aquí tienes un script de shell que podría hacer el trabajo:

```
#!/bin/bash

# Eliminar las IPs asignadas a los contenedores en la subred red_web
for ((i=1; i<=8; i++)); do
    container_name="p1-web-$i"
    container_ip="192.168.10.$i"
    echo "Eliminando IP $container_ip del contenedor $container_name en la subred red_web"
    docker network disconnect -f red_web $container_name
    docker network connect --ip $container_ip red_web $container_name
done

# Eliminar las IPs asignadas a los contenedores en la subred red_servicios
for ((i=1; i<=8; i++)); do
    container_name="p1-web-$i"
    container_ip="192.168.20.$i"
    echo "Eliminando IP $container_ip del contenedor
```

```

        $container_name en la subred red_servicios"
        docker network disconnect -f red_servicios $container_name
        docker network connect --ip $container_ip
        red_servicios $container_name
    done

```

Análisis de la salida: La idea que nos da no es mala, pero no tiene en cuenta que, en cada iteración, puede estar tratando de asignar alguna IP que ya esté en uso por otro contenedor, lo cual nos va a generar un error. Además, vamos a hacer que las IPs empiecen con valor de la parte de host 2, para que quede como en el esquema. Por ello, vamos a realizar algunos cambios en la script:

```

#!/bin/bash

# Eliminar las IPs asignadas a los contenedores en la subred red_web
for ((i=1; i<=8; i++)); do
    container_name="p1-web-$i"
    echo "Eliminando IP del contenedor $container_name en la subred red_web"
    docker network disconnect -f p1-red-web $container_name
done

# Eliminar las IPs asignadas a los contenedores en la subred red_servicios
for ((i=1; i<=8; i++)); do
    container_name="p1-web-$i"
    echo "Eliminando IP del contenedor $container_name en la subred red_servicios"
    docker network disconnect -f p1-red-servicios $container_name
done

# Asignar las IPs a los contenedores en la subred red_web
for ((i=1; i<=8; i++)); do
    container_name="p1-web-$i"
    container_ip="192.168.10.$((i+1))"
    echo "Asignando IP $container_ip al contenedor $container_name en la subred
red_web"
    docker network connect --ip $container_ip p1-red-web $container_name
done

# Asignar las IPs a los contenedores en la subred red_servicios
for ((i=1; i<=8; i++)); do
    container_name="p1-web-$i"
    container_ip="192.168.20.$((i+1))"
    echo "Asignando IP $container_ip al contenedor $container_name en la subred
red_servicios"
    docker network connect --ip $container_ip p1-red-servicios $container_name
done

```

En la siguiente imagen podemos ver como se ejecuta correctamente y, además, entro en el contenedor p1-web-8, el cuál tenía IP 192.168.20.3 y pasa a tener la IP 192.168.20.9.


```
daniel-Modern-15-B7M: ~/Glt/SWAP-UGR/P1 [main 7:1]
→ docker compose up
[+] Running 1/1
✓ Network p1_red_web Created 0.1s
[+] Running 10/2 orphan containers ([p1-network-configurator-1 p1-prometheus-1]) for this project. If yo
✓ Network p1_red_web Created 0.1s
✓ Network p1_red_servicios Created 0.1s
✓ Container p1-web-8 Created 0.0s
✓ Container p1-web-3 Created 0.0s
✓ Container p1-web-5 Created 0.0s
✓ Container p1-web-1 Created 0.0s
✓ Container p1-web-4 Created 0.0s
✓ Container p1-web-7 Created 0.0s
✓ Container p1-web-2 Created 0.0s
Attaching to web-1, web-2, web-3, web-4, web-5, web-6, web-7, web-8
web-1 | AH00558: apache2: could not reliably determine the server's fully qualified domain name, using
192.168.20.2. Set the 'ServerName' directive globally to suppress this message
web-8 | AH00558: apache2: could not reliably determine the server's fully qualified domain name, using
192.168.20.3. Set the 'ServerName' directive globally to suppress this message
web-3 | AH00558: apache2: could not reliably determine the server's fully qualified domain name, using
192.168.20.4. Set the 'ServerName' directive globally to suppress this message
web-2 | AH00558: apache2: could not reliably determine the server's fully qualified domain name, using
192.168.20.5. Set the 'ServerName' directive globally to suppress this message
web-4 | AH00558: apache2: could not reliably determine the server's fully qualified domain name, using
192.168.20.6. Set the 'ServerName' directive globally to suppress this message
web-6 | AH00558: apache2: could not reliably determine the server's fully qualified domain name, using
192.168.20.7. Set the 'ServerName' directive globally to suppress this message
web-7 | AH00558: apache2: could not reliably determine the server's fully qualified domain name, using
192.168.20.8. Set the 'ServerName' directive globally to suppress this message
web-5 | AH00558: apache2: could not reliably determine the server's fully qualified domain name, using
192.168.20.9. Set the 'ServerName' directive globally to suppress this message
[+]

daniel-Modern-15-B7M: ~/Glt/SWAP-UGR/P1 [main 7:1]
→ ./scripts/ordenarIPs.sh
Eltnando IP del contenedor p1-web-1 en la subred red_web
Eltnando IP del contenedor p1-web-2 en la subred red_web
Eltnando IP del contenedor p1-web-3 en la subred red_web
Eltnando IP del contenedor p1-web-4 en la subred red_web
Eltnando IP del contenedor p1-web-5 en la subred red_web
Eltnando IP del contenedor p1-web-6 en la subred red_web
Eltnando IP del contenedor p1-web-7 en la subred red_web
Eltnando IP del contenedor p1-web-8 en la subred red_web
Eltnando IP del contenedor p1-web-1 en la subred red_servicios
Eltnando IP del contenedor p1-web-2 en la subred red_servicios
Eltnando IP del contenedor p1-web-3 en la subred red_servicios
Eltnando IP del contenedor p1-web-4 en la subred red_servicios
Eltnando IP del contenedor p1-web-5 en la subred red_servicios
Eltnando IP del contenedor p1-web-6 en la subred red_servicios
Eltnando IP del contenedor p1-web-7 en la subred red_servicios
Eltnando IP del contenedor p1-web-8 en la subred red_servicios
Asignando IP 192.168.10.2 al contenedor p1-web-1 en la subred red_web
Asignando IP 192.168.10.3 al contenedor p1-web-2 en la subred red_web
Asignando IP 192.168.10.4 al contenedor p1-web-3 en la subred red_web
Asignando IP 192.168.10.5 al contenedor p1-web-4 en la subred red_web
Asignando IP 192.168.10.6 al contenedor p1-web-5 en la subred red_web
Asignando IP 192.168.10.7 al contenedor p1-web-6 en la subred red_web
Asignando IP 192.168.10.8 al contenedor p1-web-7 en la subred red_web
Asignando IP 192.168.10.9 al contenedor p1-web-8 en la subred red_web
Asignando IP 192.168.20.2 al contenedor p1-web-1 en la subred red_servicios
Asignando IP 192.168.20.3 al contenedor p1-web-2 en la subred red_servicios
Asignando IP 192.168.20.4 al contenedor p1-web-3 en la subred red_servicios
Asignando IP 192.168.20.5 al contenedor p1-web-4 en la subred red_servicios
Asignando IP 192.168.20.6 al contenedor p1-web-5 en la subred red_servicios
Asignando IP 192.168.20.7 al contenedor p1-web-6 en la subred red_servicios
Asignando IP 192.168.20.8 al contenedor p1-web-7 en la subred red_servicios
Asignando IP 192.168.20.9 al contenedor p1-web-8 en la subred red_servicios

daniel-Modern-15-B7M: ~/Glt/SWAP-UGR/P1 [main 7:1]
→ docker exec -it p1-web-8 /bin/bash
root@e9c550a0808b:~# ifconfig
eth2: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
inet 192.168.10.9 netmask 255.255.255.0 broadcast 192.168.10.255
ether 02:42:c0:a8:0a:09 txqueuelen 0 (Ethernet)
RX packets 27 bytes 3276 (3.1 KiB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 0 bytes 0 (0.0 B)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

eth3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
inet 192.168.20.9 netmask 255.255.255.0 broadcast 192.168.20.255
ether 02:42:c0:a8:14:09 txqueuelen 0 (Ethernet)
RX packets 26 bytes 3109 (3.0 KiB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 0 bytes 0 (0.0 B)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

De esta forma, se refleja de forma más limpia el esquema y podemos saber que IP tiene cada contenedor, sin necesidad de entrar al mismo y realizar ifconfig.

2.4.3. Estado de conectividad a las subredes:

Vamos a diseñar una script que nos permita visualizar si todos los contenedores tienen conectividad a ambas subredes. Como siempre, empezamos preguntándole a ChatGPT:

- **Yo:** Diseña una script que compruebe que todos los contenedores llamados p1-web-* tienen conectividad a la subred p1_red_web y p1_red.servicios.
- **ChatGPT:** Aquí tienes un ejemplo de un script en Python que utiliza la biblioteca docker para verificar la conectividad de los contenedores con las subredes especificadas:

```
import docker

def check_connectivity(container_name_prefix, subnets):
    client = docker.from_env()
    containers = client.containers.list()

    for container in containers:
        if container.name.startswith(container_name_prefix):
            container_info = container.attrs
            container_networks =
            container_info['NetworkSettings']['Networks']

            connected_subnets = []
            for network in container_networks:
                if container_networks[network]['IPAddress']:
                    connected_subnets.append(network)
```

```

        if all(subnet in connected_subnets for subnet in subnets):
            print(f"Container '{container.name}' has
              connectivity to all specified subnets.")
        else:
            print(f"Container '{container.name}' does not have
              connectivity to all specified subnets.")

# Nombre del prefijo de los contenedores que deseas comprobar
container_prefix = "p1-web-"

# Subredes que se deben verificar
subnets_to_check = ["p1_red_web", "p1_red_servicios"]

check_connectivity(container_prefix, subnets_to_check)

```

Análisis de la salida: La idea es bastante correcta, pero vamos a hacer algunos ajustes para que solo indique si el contenedor tiene conectividad total o, en caso de no tenerla, que lo imprima de un color más característico. Por lo que, finalmente quedaría:

```

import docker
from colorama import init, Fore

# Inicializa colorama
init()

def check_connectivity(container_name_prefix, subnets):
    client = docker.from_env()
    containers = client.containers.list()

    containers_with_no_connectivity = []

    for container in containers:
        if container.name.startswith(container_name_prefix):
            container_info = container.attrs
            container_networks = container_info['NetworkSettings']['Networks']

            connected_subnets = []
            for network in container_networks:
                if container_networks[network]['IPAddress']:
                    connected_subnets.append(network)

            missing_subnets = [subnet for subnet in subnets if subnet not in
connected_subnets]
            if missing_subnets:
                containers_with_no_connectivity.append((container.name,
missing_subnets))

    if containers_with_no_connectivity:
        for container_name, missing_subnets in containers_with_no_connectivity:
            print(f"{Fore.RED}Container '{container_name}' is missing connectivity to
subnet(s): {', '.join(missing_subnets)}{Fore.RESET}")
        else:
            print("Todos los contenedores estan correctamente conectados a las subredes
especificadas.")

# Nombre del prefijo de los contenedores que deseas comprobar

```

```

container_prefix = "pl-web-"

# Subredes que se deben verificar
subnets_to_check = ["pl-red-web", "pl-red-servicios"]

check_connectivity(container_prefix, subnets_to_check)

```

Vamos a hacer una prueba de la misma. Para ello, levantamos los contenedores con docker compose. En primera instancia todos tendrán conectividad. Después, le quitaremos la conectividad a alguno de ellos y volveremos a probar:

```

Terminal
daniel-Modern-15-B7M: ~/Glt/SWAP-UGR/P1 [main 7:1 A]
→ docker compose up
[+] Building 1/5
✓ Network pl_red_servicios Created 0.15
✓ Network pl_red_web Creating 0.15
[+] Building 2/5 Orphan containers ([pi-network-configurator-1 pi-prometheus-1]) for this project. If yo
✓ Network pl_red_servicios Created 0.15
✓ Network pl_red_web Created 0.15
✓ Container pi-web-4 Created 0.15
✓ Container pi-web-1 Created 0.15
✓ Container pi-web-7 Created 0.15
✓ Container pi-web-5 Created 0.15
✓ Container pi-web-2 Created 0.15
✓ Container pi-web-3 Created 0.15
✓ Container pi-web-8 Created 0.15
✓ Container pi-web-6 Created 0.15
Attaching to web-1, web-2, web-3, web-4, web-5, web-6, web-7, web-8
web-6 | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using
192.168.10.2. Set the 'ServerName' directive globally to suppress this message
web-3 | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using
192.168.20.3. Set the 'ServerName' directive globally to suppress this message
web-2 | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using
192.168.20.4. Set the 'ServerName' directive globally to suppress this message
web-7 | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using
192.168.20.5. Set the 'ServerName' directive globally to suppress this message
web-8 | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using
192.168.20.6. Set the 'ServerName' directive globally to suppress this message
web-5 | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using
192.168.20.7. Set the 'ServerName' directive globally to suppress this message
web-4 | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using
192.168.10.8. Set the 'ServerName' directive globally to suppress this message
web-1 | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using
192.168.20.9. Set the 'ServerName' directive globally to suppress this message
[+]

Terminal
daniel-Modern-15-B7M: ~/Glt/SWAP-UGR/P1 [main 7:1 A]
→ python3 scripts/conectividad.py
Todos los contenedores están correctamente conectados a las subredes especificadas.
daniel-Modern-15-B7M: ~/Glt/SWAP-UGR/P1 [main 7:1 A]
→ docker network disconnect -f pl_red_web pi-web-1
daniel-Modern-15-B7M: ~/Glt/SWAP-UGR/P1 [main 7:1 A]
→ python3 scripts/conectividad.py
Container 'pi-web-1' is missing connectivity to subnet(s): pl_red_web
daniel-Modern-15-B7M: ~/Glt/SWAP-UGR/P1 [main 7:1 A]
→

```

2.4.4. Comprobar estado de Apache

Se me ha ocurrido también crear una script, la cual verifique que, al acceder a **http://<IP>**, reciba un **status OK 200** por parte del protocolo HTTP, lo cual, nos garantiza de que Apache está corriendo correctamente. Para ello, comenzamos preguntando a ChatGPT:

- **Yo:** Tengo un fichero docker compose que me despliega múltiples contenedores de una imagen que cuenta con un servidor Apache instalado. Cada contenedor tiene dos IPs, una es 192.168.10.X y otra 192.168.20.X, donde X es el numero del contenedor más 1. ¿Puedo crear alguna script que compruebe que al entrar desde el navegador en dicha IP recibe la página de Apache correctamente?
- Sí, puedes escribir un script en un lenguaje como Python o Bash para comprobar si puedes acceder a la página de Apache en cada una de las direcciones IP de tus contenedores Docker. Aquí tienes un ejemplo de cómo podrías hacerlo en Python utilizando el módulo requests para hacer solicitudes HTTP:

```

import requests

def check_apache(ip_address):
    url = f"http://{ip_address}/"
    try:

```

```

        response = requests.get(url)
        if response.status_code == 200:
            print(f"Apache server is running at {ip_address}")
        else:
            print(f"Failed to access Apache server at {ip_address}")
    except requests.ConnectionError:
        print(f"Failed to connect to {ip_address}")

# Lista de direcciones IP de los contenedores
ip_addresses = [
    "192.168.10.2",
    "192.168.20.2",
    # Agrega aquí las demás direcciones IP de tus contenedores
]

for ip_address in ip_addresses:
    check_apache(ip_address)

```

Análisis de la salida: Me ha venido bien para ver un poco la idea de como hacer el código en Python, pero creo que se podría generalizar bastante, ya que introducir a mano las direcciones de todos los contenedores es bastante tedioso y difícil de escalar. Debido a esto, he decidido generalizar un poco la script para que compruebe las interfaces de red de todos los contenedores y vaya ejecutando este código de comprobación. Además, comprueba si el servidor al que realiza la petición HTTP es Apache o no:

```

import docker
import requests

def get_container_ips(container_name):
    client = docker.from_env()
    container_ips = []
    containers = client.containers.list()
    for container in containers:
        if container.name in container.name:
            container_info = container.attrs
            network_settings = container_info["NetworkSettings"]
            for network, config in network_settings["Networks"].items():
                container_ips.append(config["IPAddress"])
    return container_ips

def check_apache(ip_address):
    url = f"http://{ip_address}/"
    try:
        response = requests.get(url, timeout=5)
        if response.status_code == 200:
            server_header = response.headers.get('Server')
            if server_header and 'Apache' in server_header:
                print(f"Apache server is running at {ip_address}")
            else:
                print(f"Server at {ip_address} is not Apache")
        else:
            print(f"Failed to access Apache server at {ip_address}")
    except requests.ConnectionError:
        print(f"Failed to connect to {ip_address}")

# Nombre del contenedor
container_name = "pl-web"

```

```
# Obtiene todas las direcciones IP de los contenedores con el nombre especificado
container_ips = get_container_ips(container_name)

# Verifica el servidor Apache en cada direccion IP obtenida
for ip_address in container_ips:
    check_apache(ip_address)
```

De esta forma, si decidimos aumentar el número de contenedores, podemos seguir utilizando la script sin problemas. Vamos a comprobar su uso:

```
daniel@Modern-15-B7M: ~/Glt/SWAP-UGR/P1 [main U:2 A]
→ docker exec -it p1-web-1 /bin/bash
root@6bb5a90e2c03:/# ufw deny 80
Rule updated
Rule updated (v6)
root@6bb5a90e2c03:/# exit
exit

daniel@Modern-15-B7M: ~/Glt/SWAP-UGR/P1 [main U:2 A]
→ python3 scripts/comprobacionApache.py
Apache server is running at 192.168.20.2
Failed to connect to 192.168.20.3
Apache server is running at 192.168.10.4
Apache server is running at 192.168.20.4
Apache server is running at 192.168.10.5
Apache server is running at 192.168.20.5
Apache server is running at 192.168.20.6
Apache server is running at 192.168.10.6
Apache server is running at 192.168.10.7
Apache server is running at 192.168.20.7

daniel@Modern-15-B7M: ~/Glt/SWAP-UGR/P1 [main U:2 A]
→

daniel@Modern-15-B7M: ~/Glt/SWAP-UGR/P1 [main U:2 A]
→ docker compose up
[+] Building 1/1
✓ Network p1_red_web Created 0.2s
✓ Network p1_red_servicios Creating 0.0s
[+] Running 1/1 Orphan containers (p1-network-configurator-1 p1-prometheus-1) for this project. If yo
✓ Network p1_red_web Created 0.1s
✓ Network p1_red_servicios Created 0.1s
✓ Container p1-web-8 Created 0.1s
✓ Container p1-web-3 Created 0.1s
✓ Container p1-web-5 Created 0.1s
✓ Container p1-web-4 Created 0.1s
✓ Container p1-web-6 Created 0.1s
✓ Container p1-web-1 Created 0.1s
✓ Container p1-web-2 Created 0.1s
✓ Container p1-web-7 Created 0.1s
Attaching to web-1, web-2, web-3, web-4, web-5, web-6, web-7, web-8
web-1 | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using
192.168.20.2. Set the 'ServerName' directive globally to suppress this message
web-1 | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using
192.168.20.3. Set the 'ServerName' directive globally to suppress this message
web-7 | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using
192.168.10.4. Set the 'ServerName' directive globally to suppress this message
web-5 | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using
192.168.20.4. Set the 'ServerName' directive globally to suppress this message
web-2 | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using
192.168.20.5. Set the 'ServerName' directive globally to suppress this message
web-6 | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using
192.168.20.6. Set the 'ServerName' directive globally to suppress this message
web-8 | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using
192.168.10.7. Set the 'ServerName' directive globally to suppress this message
web-4 | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using
192.168.20.7. Set the 'ServerName' directive globally to suppress this message
```

2.4.5. Instalación de paquetes

Para poder realizar próximos ejercicios, como puede ser el caso de 2.5.2, necesito instalar varias herramientas en cada contenedor, por lo que he creado una script adicional que recibe como parámetro un paquete a instalar y lo instala en todos los contenedores de la práctica:

```
#!/bin/bash

# Verificar si se proporciona un argumento
if [ $# -eq 0 ]; then
    echo "Error: Debes proporcionar el nombre de la herramienta a instalar."
    echo "Uso: $0 <nombre_herramienta>"
    exit 1
fi

# Nombre de la herramienta a instalar
herramienta=$1

# Lista de contenedores que siguen el patron p1-web-
contenedores=$(docker ps --format '{{.Names}}' | grep 'p1-web-')

# Funcion para instalar la herramienta en un contenedor
instalar_herramienta_contenedor() {
    echo "Instalando $herramienta en el contenedor $1..."
```

```

    docker exec $1 apt-get update
    docker exec $1 apt-get install -y $herramienta
}

# Iterar sobre los contenedores y ejecutar la funcion de instalacion
for contenedor in $contenedores; do
    instalar_herramienta_contenedor $contenedor
done

```

No le dedicaré más tiempo a esta script, ya que es bastante sencilla.

2.5. Monitoreo y Logging

Para este apartado comencé preguntándole a ChatGPT por herramientas de monitoreo y logging para rastrear el rendimiento y los eventos de los contenedores y me dió algunas opciones como Prometheus y Grafana, ELK Stack (Elasticsearch, Logstash, Kibana)... Decidí quedarme con Sysdig, ya que Prometheus y Grafana ya lo había probado en otro proyecto personal, así como las opciones integradas Docker Stats y Docker Log.

2.5.1. Sysdig

Comenzamos preguntando a ChatGPT que opciones de configuración tenemos disponibles:

- **Yo:** Tengo un escenario con varios contenedores, los cuales usan una imagen de debian slim y tienen instalados Apache con PHP. El profesor me pide configurar herramientas de monitoreo y logging para rastrear el rendimiento y los eventos de los contenedores. He decido usar la herramienta sysdig, ¿qué configuraciones interesantes puedo hacer para mi escenario?
- **ChatGPT:** Sysdig es una poderosa herramienta de monitoreo y troubleshooting que te permite capturar y analizar datos del sistema en tiempo real. Aquí hay algunas configuraciones interesantes que podrías considerar para tu escenario con contenedores que ejecutan Apache con PHP en una imagen Debian Slim:
 1. Captura de eventos de red: Puedes utilizar Sysdig para capturar eventos de red, como conexiones entrantes y salientes a tus contenedores. Esto te ayudará a entender el tráfico que llega a tus aplicaciones y cómo responden.
 2. Monitoreo de uso de CPU y memoria: Sysdig puede proporcionar estadísticas detalladas sobre el uso de CPU y memoria de tus contenedores. Puedes configurar alertas para notificar cuando estos recursos alcancen ciertos umbrales, lo que te permite tomar medidas proactivas para evitar problemas de rendimiento.
 3. Análisis de tráfico HTTP: Sysdig puede analizar el tráfico HTTP que llega a tus contenedores, lo que te permite identificar patrones de acceso a tus aplicaciones web y detectar posibles ataques o problemas de rendimiento.
 4. Monitoreo de archivos abiertos: Puedes utilizar Sysdig para monitorear los archivos que tus contenedores abren y leen, lo que puede ayudarte a identificar problemas de acceso a archivos o a detectar actividades sospechosas.
 5. Captura de eventos de sistema: Sysdig puede capturar una amplia variedad de eventos de sistema, como llamadas al sistema, cambios de permisos de archivos y actividad de red. Esto te permite obtener una visión completa del comportamiento de tus contenedores y de cualquier actividad anómala que pueda ocurrir.

6. Creación de dashboards personalizados: Sysdig te permite crear dashboards personalizados para visualizar tus métricas de rendimiento de la manera que desees. Puedes crear paneles que muestren gráficos de CPU, memoria, tráfico de red y cualquier otra métrica que consideres importante.
7. Integración con alertas: Sysdig se puede integrar con sistemas de alerta como Prometheus o Grafana, lo que te permite recibir notificaciones cuando se detecten problemas en tus contenedores. Puedes configurar reglas de alerta basadas en umbrales predefinidos o en patrones de comportamiento anormal.

Análisis de la salida: No hay mucho que comentar en este caso, ya que solo estamos preguntando por opciones.

Trás ver las opciones que me ha sugerido ChatGPT he decidido realizar una **captura de eventos de red, monitoreo de uso de CPU y memoria y análisis del tráfico de HTTP**. Vamos a volver a preguntarle a ChatGPT sobre cómo podemos realizar estas configuraciones:

- **Yo:** Tengo un escenario con varios contenedores, los cuales usan una imagen de debian slim y tienen instalados Apache con PHP. Usando la herramienta Sysdig, quiero capturar las conexiones entrantes y salientes de los contenedores, ver estadísticas sobre el uso de CPU y memoria y analizar el tráfico HTTP que llega a los mismos. Hay varios contenedores, p1-web-1, p1-web-2, p1-web-3... Quiero crear una script que al ejecutarla realice las acciones que te pido previamente y lo almacene en un fichero de logs distinto para cada contenedor.
- **ChatGPT:** Puedes crear un script de shell que utilice las herramientas sysdig y docker para realizar las acciones que desees y almacenar los resultados en archivos de registro separados para cada contenedor. Aquí tienes un ejemplo de cómo podrías hacerlo:

```
#!/bin/bash

# Obtener la lista de contenedores que contienen "p1-web"
containers=$(docker ps --format '{{.Names}}' | grep 'p1-web')

# Loop sobre cada contenedor
for container in $containers; do
    echo "Capturando datos para el contenedor $container..."

    # Crear el archivo de log para el contenedor actual
    log_file="${container}_logs.txt"

    # Capturar conexiones entrantes y salientes del contenedor
    sysdig -z -w $log_file "container.name=$container and
    (fd.sip != container.ip or fd.dip != container.ip)"

    # Obtener estadísticas de uso de CPU y memoria
    docker stats --no-stream $container >> $log_file

    # Analizar el tráfico HTTP entrante al contenedor
    sysdig -r $log_file -c spy_users=http "container.name=$container"
    >> $log_file

    echo "Datos capturados para el contenedor $container,
    almacenados en $log_file"
```

done

Este script iterará sobre todos los contenedores que tengan "p1-web.^{en} su nombre, capturará las conexiones entrantes y salientes, las estadísticas de uso de CPU y memoria, y analizará el tráfico HTTP para cada contenedor. Los resultados se almacenarán en archivos de registro separados para cada contenedor, con el nombre del contenedor seguido de _logs.txt.

Análisis de la salida: La propuesta no es mala, pero tiene algunos inconvenientes. No sabe cuando abortar la captura de conexiones entrantes o el tráfico HTTP, por lo que, en cada iteración del bucle, se quedará en espera hasta que nosotros indiquemos que aborte. Podemos solucionarlo haciendo unas pequeñas modificaciones, como crear la carpeta de logs y establecer un tiempo para abortar. Además, vamos a hacer un formateo de la salida y una script con diferentes opciones.

```
#!/bin/bash

# Funcion para obtener la fecha y hora actual formateada
get_timestamp() {
    date +"%Y-%m-%d %T"
}

# Pedir el numero de contenedor a monitorear
read -p "Introduce el numero del contenedor a monitorear (X en p1-web-X): "
container_number
container_name="p1-web-$container_number"

# Verificar si el contenedor existe
docker inspect "$container_name" > /dev/null 2>&1
if [ $? -ne 0 ]; then
    echo "El contenedor $container_name no existe."
    exit 1
fi

# Pedir la accion a realizar
echo "Selecciona la accion a realizar:"
select action in "Estadisticas de CPU y memoria" "Conexiones entrantes" "Tráfico HTTP"
; do
    case $action in
        "Estadisticas de CPU y memoria" )
            action_command="docker stats --no-stream $container_name"
            action_option="Estadisticas de CPU y memoria"
            break
        ;;
        "Conexiones entrantes" )
            action_command="sysdig -pc -A -s 2000 -z -c echo_fds 'evt.type=accept and container.name=$container_name'"
            action_option="Conexiones entrantes"
            break
        ;;
        "Tráfico HTTP" )
            number=$((container_number + 1))
            action_command="sysdig -s 2000 -A -c echo_fds fd.sip=192.168.20.$number or fd.sip=192.168.10.$number and fd.port=80"
            action_option="Tráfico HTTP"
            break
        ;;
        * ) echo "Por favor, selecciona una opcion valida.";;
    esac
done

# Si la accion es estadisticas de CPU y memoria, no solicitar el tiempo maximo de captura
```



```

if [ "$action_option" == "Estadísticas de CPU y memoria" ]; then
    max_time=0
else
    # Pedir el tiempo maximo antes de abortar
    read -p "Introduce el tiempo maximo de captura (en segundos): " max_time
fi

# Pedir si se desea almacenar los datos en un archivo de logs
read -p "Deseas almacenar los datos en un archivo de logs? (s/n): " store_logs

# Crear la carpeta de logs si es necesario
if [ "$store_logs" = "s" ]; then
    mkdir -p ./logs
fi

# Ejecutar la accion
if [ "$store_logs" = "s" ]; then
    # Obtener el nombre del archivo de logs
    log_file="./logs/${container_name}.log"

    # Ejecutar la accion y almacenar los resultados en el archivo de logs
    echo -e "\e[33m$(printf '%80s\n' | tr ' ' '-')\e[0m" >> "$log_file"
    echo -e "\e[33m$(get_timestamp) - Opciones: $action_option\e[0m" >> "$log_file"
    echo -e "\e[33m$(printf '%80s\n' | tr ' ' '-')\e[0m" >> "$log_file"

    $action_command >> "$log_file" &
else
    $action_command
fi

# Esperar al tiempo maximo antes de abortar si no es 0 (caso de estadísticas de CPU y memoria)
if [ $max_time -ne 0 ]; then
    sleep $max_time

    # Finalizar la ejecucion de la accion
    kill $!
fi

echo "Accion finalizada."

```

Se podría hacer scripts más generales, para programar un funcionamiento periódico posterior con herramientas como **Crontab**, pero he decidido hacerlo así para que se vea más claro el uso de la herramienta **Sysdig**.

Vamos a ponerlo a prueba:

```
daniel-Modern-15-B7M: ~/Git/SWAP-UGR/P1 [main 7:1]
→ sudo ./scripts/capturarDatos.sh
Introduce el número del contenedor a monitorear (X en p1-web-X): 1
Selecciona la acción a realizar:
1) Estadísticas de CPU y memoria
2) Conexiones entrantes
3) Tráfico HTTP
#? 1
¿Deseas almacenar los datos en un archivo de logs? (s/n): n
CONTAINER ID   NAME      CPU %     MEM USAGE / LIMIT   MEM %     NET I/O     BLOCK I/O     PID
5              Sfl0af31e364   p1-web-1   0.01%      13.79MiB / 15GiB     0.09%      62.1kB / 75.8kB   319kB / 77.8kB   9
Acción finalizada.

daniel-Modern-15-B7M: ~/Git/SWAP-UGR/P1 [main 7:1]
→ sudo ./scripts/capturarDatos.sh
Introduce el número del contenedor a monitorear (X en p1-web-X): 1
Selecciona la acción a realizar:
1) Estadísticas de CPU y memoria
2) Conexiones entrantes
3) Tráfico HTTP
#? 1
¿Deseas almacenar los datos en un archivo de logs? (s/n): s
Acción finalizada.

daniel-Modern-15-B7M: ~/Git/SWAP-UGR/P1 [main 7:2]
→ sudo ./scripts/capturarDatos.sh
Introduce el número del contenedor a monitorear (X en p1-web-X): 1
Selecciona la acción a realizar:
1) Estadísticas de CPU y memoria
2) Conexiones entrantes
3) Tráfico HTTP
#? 3
Introduce el tiempo máximo de captura (en segundos): 5
¿Deseas almacenar los datos en un archivo de logs? (s/n): s
Acción finalizada.

daniel-Modern-15-B7M: ~/Git/SWAP-UGR/P1 [main 7:2]
→
```

```
SWAP-danieeld2
La dirección IP del servidor Ancho es 192.168.20.2

daniel-Modern-15-B7M: ~/Git/SWAP-UGR/P1 [main 7:1]
→ cat logs/p1-web-1.log
-----
2024-04-02 14:07:25 - Opciones: Estadísticas de CPU y memoria
CONTAINER ID   NAME      CPU %     MEM USAGE / LIMIT   MEM %     NET I/O     BLOCK I/O     PIDs
5              Sfl0af31e364   p1-web-1   0.01%      13.79MiB / 15GiB     0.09%      62.1kB / 75.8kB   319kB / 77.8kB   9
-----
2024-04-02 14:07:31 - Opciones: Tráfico HTTP
-----
..... Write 365B to 192.168.20.1:43418->192.168.20.2:80 (socket)
GET / HTTP/1.1
Host: 192.168.20.2
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:123.0) Gecko/20100101 Firefox/123.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
..... Read 365B from 192.168.20.1:43418->192.168.20.2:80 (apache2)
GET / HTTP/1.1
Host: 192.168.20.2
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:123.0) Gecko/20100101 Firefox/123.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
..... Write 802B to 192.168.20.1:43418->192.168.20.2:80 (apache2)
HTTP/1.1 200 OK
Date: Tue, 02 Apr 2024 12:07:33 GMT
Server: Apache/2.4.56 (Debian)
X-Frame-Options: SAMEORIGIN
Content-Security-Policy: default-src 'self'; script-src 'self' 'unsafe-inline' 'unsafe-eval'; style-src 'self' 'unsafe-inline'; img-src 'self' data:;
```

2.5.2. Estado del servidor

Para este apartado vamos a usar las herramientas `htop`, `netstat` y `apache2ctl` dentro de los contenedores para monitorear y diagnosticar el estado del servidor. Vamos a comenzar preguntándole a ChatGPT:

- **Yo:** Tengo un escenario con varios contenedores, los cuales usan una imagen de `debian slim` y tienen instalados `Apache` con `PHP`. Usando las herramientas `htop`, `netstat`, o `apache2ctl` dentro de los contenedores, monitorear y diagnosticar el estado del servidor. Se te ocurre alguna script que podamos hacer para ver el estado de los servidores de los contenedores? los contenedores se llaman `p1-web-1`, `p2-web-2`,...
- **ChatGPT:** Claro, puedes crear un script Bash para automatizar la tarea de verificar el estado de los servidores en tus contenedores. Aquí tienes un ejemplo de cómo podrías hacerlo:

```
#!/bin/bash

# Obtener lista de contenedores que coinciden con el patrón p1-web-
contenedores=$(docker ps --format '{{.Names}}' | grep 'p1-web-')

# Funcion para verificar el estado de Apache dentro del contenedor
verificar_estado_apache() {
    echo "Estado de Apache en el contenedor $1:"
    docker exec $1 apache2ctl status
}

# Funcion para verificar el uso de recursos del contenedor
verificar_recursos_contenedor() {
    echo "Uso de recursos en el contenedor $1:"
    docker exec $1 htop -bn1
}
```

```

# Funcion para verificar las conexiones de red del contenedor
verificar_conexiones_red() {
    echo "Conexiones de red en el contenedor $1:"
    docker exec $1 netstat -tuln
}

# Iterar sobre la lista de contenedores y ejecutar las
funciones de verificacion
for contenedor in $contenedores; do
    echo "-----"
    verificar_estado_apache $contenedor
    verificar_recursos_contenedor $contenedor
    verificar_conexiones_red $contenedor
done

```

Análisis de la salida: Como vemos, la solución propuesta es bastante razonable, pero, como siempre; vamos a hacer algunas modificaciones de formateo y almacenado para que funcionen de forma más correcta. También, es preciso recordar que tenemos que instalar los paquetes requeridos. Para ello disponemos de la script de instalación realizada previamente. Basta con usar `./scripts/instalacion.sh <nombre-paquete>`. Los paquetes a instalar son: `httpd`, `net-tools`, `apache2-utils` y `lynx` (*Voy a dejar hecha una script para que sea más fácil reinstalarlos cada vez que eliminemos los contenedores. Este archivo se llama `instalacionAUX.sh` u ha de ejecutarse desde la raíz del proyecto*).

Por otro lado, la herramienta `httpd` dará problemas, ya que usa una terminal gráfica. Por lo que usaremos el comando `top` en su lugar:

```

#!/bin/bash

# Crear la carpeta de logs si no existe
mkdir -p logs

# Obtener lista de contenedores que coinciden con el patron p1-web-
contenedores=$(docker ps --format '{{.Names}}' | grep 'p1-web-')

# Funcion para verificar el estado de Apache dentro del contenedor
verificar_estado_apache() {
    echo -e "\e[34mEstado de Apache en el contenedor $1:\e[0m"
    docker exec $1 apache2ctl status
}

# Funcion para verificar el uso de recursos del contenedor
verificar_recursos_contenedor() {
    echo -e "\e[34mUso de recursos en el contenedor $1:\e[0m"
    docker exec $1 top -b -n 1
}

# Funcion para verificar las conexiones de red del contenedor
verificar_conexiones_red() {
    echo -e "\e[34mConexiones de red en el contenedor $1:\e[0m"
    docker exec $1 netstat -tuln
}

# Obtener la hora actual
hora_actual=$(date "+%Y-%m-%d %H:%M:%S")

# Iterar sobre los contenedores y ejecutar las funciones de verificacion
for contenedor in $contenedores; do

```

```

echo -e "\e[33m$(printf '%80s\n' | tr ' ' '-' )\e[0m" >> logs/serverStatus-${
contenedor##*-}.log
echo -e "\e[33m$hora_actual\e[0m" >> logs/serverStatus-${contenedor##*-}.log
echo -e "\e[33m$(printf '%80s\n' | tr ' ' '-' )\e[0m" >> logs/serverStatus-${
contenedor##*-}.log
verificar_estado_apache $contenedor >> logs/serverStatus-${contenedor##*-}.log
verificar_recursos_contenedor $contenedor >> logs/serverStatus-${contenedor##*-}.
log
verificar_conexiones_red $contenedor >> logs/serverStatus-${contenedor##*-}.log
done

```

La script sigue un poco la idea del sub-apartado anterior. Podríamos haber hecho scripts más específicas para que se ejecuten periódicamente y filtren la salida y, en caso de ver algo sospechoso, que envíen una notificación o warning; pero he preferido hacer varias tareas simples para ver el uso general de las herramientas. En este caso, simplemente ejecuta las herramientas y las almacena (con formato) en el fichero de logs. Vamos a probarlo:

```

daniel-Modern-15-B7M: ~/Glt/SWAP-UGR/P1 [main 7:3 A]
→ ./scripts/serverStatus.sh

daniel-Modern-15-B7M: ~/Glt/SWAP-UGR/P1 [main 7:3 A]
→ ./scripts/serverStatus.sh

daniel-Modern-15-B7M: ~/Glt/SWAP-UGR/P1 [main 7:3 A]
→ ls logs/
serverStatus-1.log serverStatus-3.log serverStatus-5.log serverStatus-7.log
serverStatus-2.log serverStatus-4.log serverStatus-6.log serverStatus-8.log

daniel-Modern-15-B7M: ~/Glt/SWAP-UGR/P1 [main 7:3 A]
→ cat logs/serverStatus-1.log
-----
Estado de Apache en el contenedor pi-web-1:
Apache Server Status for localhost (via ::1)

Server Version: Apache/2.4.56 (Debian)
Server MPM: prefork
Server Built: 2023-04-02T03:06:01

Current Time: Tuesday, 02-Apr-2024 13:09:44 UTC
Restart Time: Tuesday, 02-Apr-2024 12:21:41 UTC
Parent Server Config. Generation: 1
Parent Server MPM Generation: 0
Server uptime: 48 minutes 3 seconds
Server load: 0.72 0.67 0.73
Total accesses: 6 - Total Traffic: 24 kB - Total Duration: 4
CPU Usage: 0.06 s.13 cu0 cs0 - .00659% CPU load
.00208 requests/sec - 8 B/second - 4096 B/request - .66667 ms/request
1 requests currently being processed, 4 idle workers

-----
U-----
Scoreboard Key:
" " Waiting for Connection, "S" Starting up, "R" Reading Request,
"W" Sending Reply, "K" Keepalive (read), "D" DNS Lookup,
"C" Closing connection, "L" Logging, "G" Gracefully finishing,
"I" Idle cleanup of worker, "." Open slot with no current process
-----
Uso de recursos en el contenedor pi-web-1:
top - 13:09:45 up 2:57, 0 users, load average: 0.72, 0.67, 0.73
Tasks: 8 total, 1 running, 7 sleeping, 0 stopped, 0 zombie
%CPU(s): 0.4 us, 0.0 sy, 0.0 ni, 99.2 id, 0.0 wa, 0.0 hi, 0.4 st, 0.0 st
Mem Mem: 15356.2 total, 7559.0 free, 4081.3 used, 3715.9 buff/cache
Mem Swap: 2048.0 total, 2048.0 free, 0.0 used, 10865.7 avail Mem

-----
PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
1 root 20 0 2480 1280 1280 S 0.0 0.0 0:00.01 apache2
8 root 20 0 193884 20516 16156 S 0.0 0.1 0:00.21 apache2
9 www-data 20 0 193964 7532 3072 S 0.0 0.0 0:00.00 apache2
10 www-data 20 0 193964 7404 2944 S 0.0 0.0 0:00.00 apache2
11 www-data 20 0 193964 7404 2944 S 0.0 0.0 0:00.00 apache2
12 www-data 20 0 193964 7532 3072 S 0.0 0.0 0:00.00 apache2
13 www-data 20 0 193964 7532 3072 S 0.0 0.0 0:00.00 apache2
2408 root 20 0 6860 2944 2560 R 0.0 0.0 0:00.01 top

-----
Conexiones de red en el contenedor pi-web-1:
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address Foreign Address State
tcp 0 0 127.0.0.1:45517 0.0.0.0:* LISTEN
tcp6 0 0 :::80 ::: LISTEN
udp 0 0 127.0.0.1:49200 0.0.0.0:*

-----
2024-04-02 15:10:58
Estado de Apache en el contenedor pi-web-1:

```