



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	16
2.4.1. Mantenimiento	16
2.4.2. Creación de Contenedores y Configuración de Redes	18
2.4.3. Estado de conectividad a las subredes:	20
2.5. Monitoreo y Logging	22

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_danieeeld2/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_danieeeld2 .
$ docker run -d -p 8080:80 --name mi-apache-php debian-apache-php

```

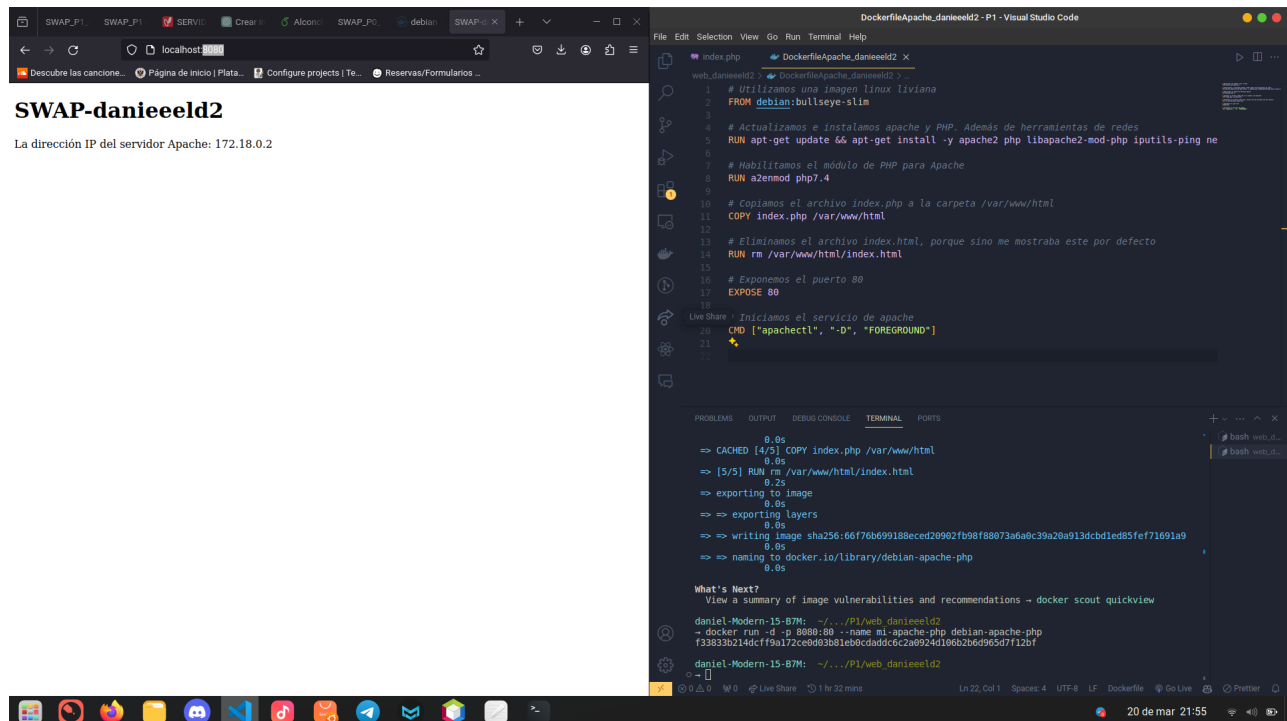
Primero, construimos la imagen (indicando el fichero, ya que este no se llama Dockerfile, por lo que no lo toma por defecto) y la ejecutamos en segundo plano con `-d`. En caso de que quisiéramos pararlo y/o borrarlo:

```

$ 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 dichero 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 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.

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:

```
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
[+]

daniel-Modern-15-B7M: ~
→docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED      STATUS      PORTS
91409cc3b1fe       danieeld2-apache-l... "apache2ctl -D FOREGR..." 6 minutes ago Up 6 minutes 80/tcp
86a0a8e08a01       danieeld2-apache-l... "apache2ctl -D FOREGR..." 6 minutes ago Up 6 minutes 80/tcp
ae354737d187       danieeld2-apache-l... "apache2ctl -D FOREGR..." 6 minutes ago Up 6 minutes 80/tcp
ecd9c5eb9c6c       danieeld2-apache-l... "apache2ctl -D FOREGR..." 6 minutes ago Up 6 minutes 80/tcp
4ee33d57e680       danieeld2-apache-l... "apache2ctl -D FOREGR..." 6 minutes ago Up 6 minutes 80/tcp
6d6f7addb371       danieeld2-apache-l... "apache2ctl -D FOREGR..." 6 minutes ago Up 6 minutes 80/tcp
be25dc889f9a       danieeld2-apache-l... "apache2ctl -D FOREGR..." 6 minutes ago Up 6 minutes 80/tcp
6d3e7b889916       danieeld2-apache-l... "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:

```
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-3 | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using
192.168.10.5. 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 5/0
✔ 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
192.168.10.5. 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
[+]

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 4216 (0.0 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:a8:0a:08 txqueuelen 0 (Ethernet)
    RX packets 0 bytes 0 (0.0 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

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=0.094 ms
64 bytes from 192.168.20.7: icmp_seq=2 ttl=64 time=0.088 ms
64 bytes from 192.168.20.7: icmp_seq=3 ttl=64 time=0.082 ms
^C
--- 192.168.20.7 ping statistics ---
3 packets transmitted, 3 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`.

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"]
```

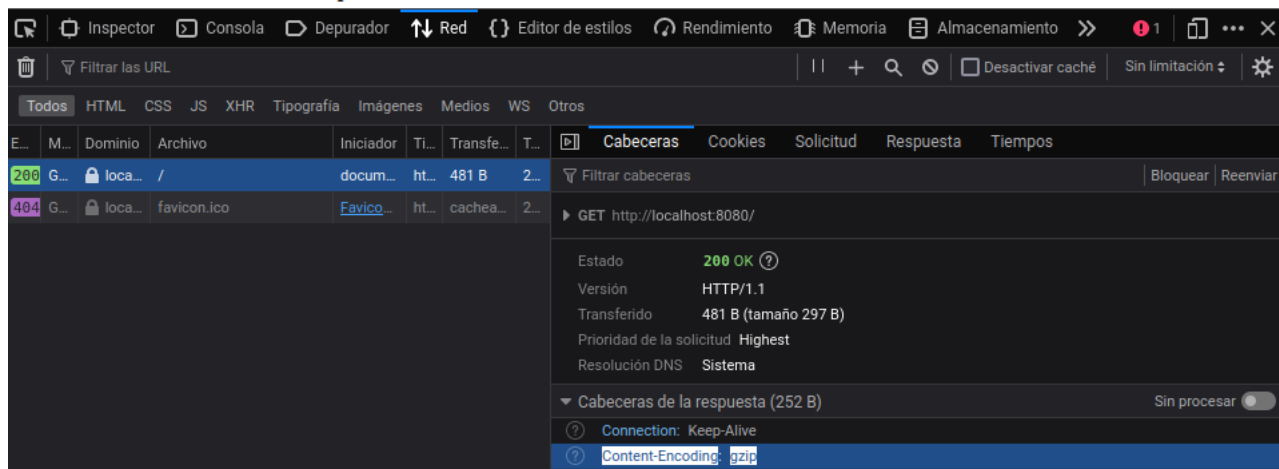
Podemos ahora construir, nuevamente, la imagen y ejecutarla de la siguiente forma:

```
$ 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>
```

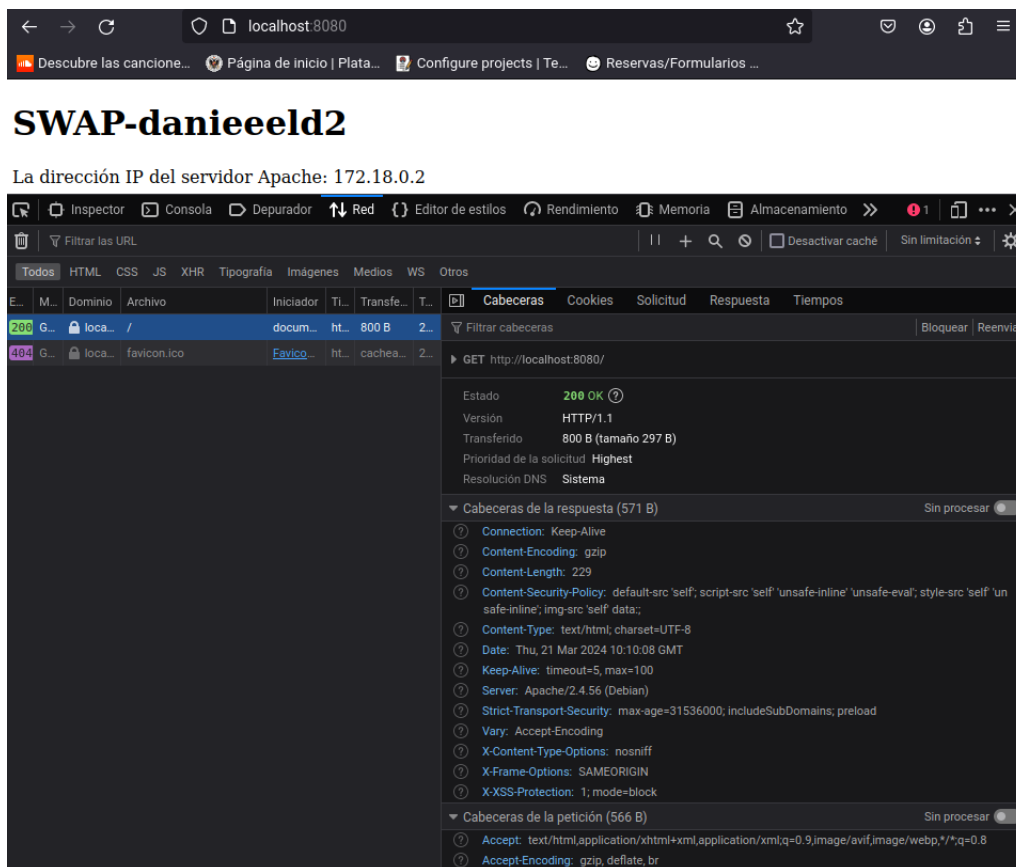
Luego, añade la siguiente línea a tu archivo dockerfile: `RUN a2enmod headers`.

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 indice 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

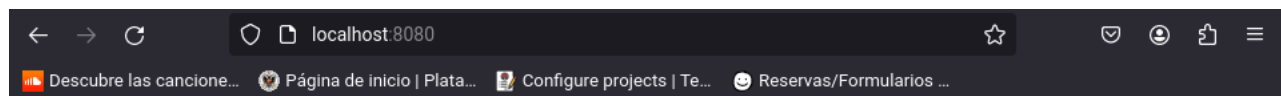
# Exponemos el puerto 80
EXPOSE 80

# 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:

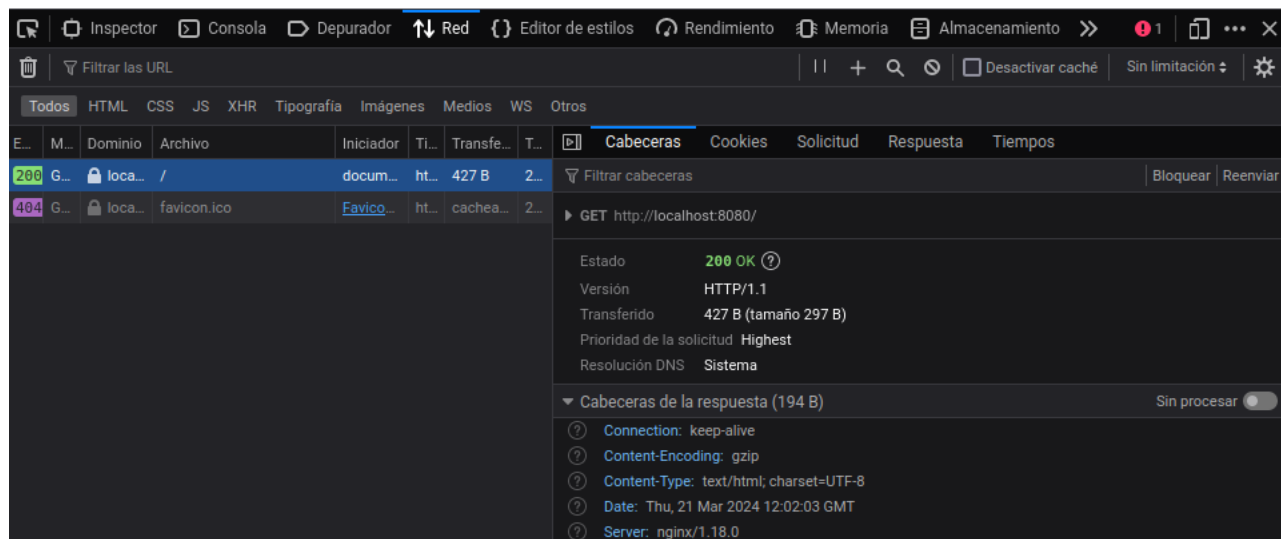
```
$ docker build -t danieeeld2-nginx-image:p1 -f DockerfileNginx-danieeeld2 .
$ docker run -d -p 8080:80 danieeeld2-nginx-image:p1
```

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

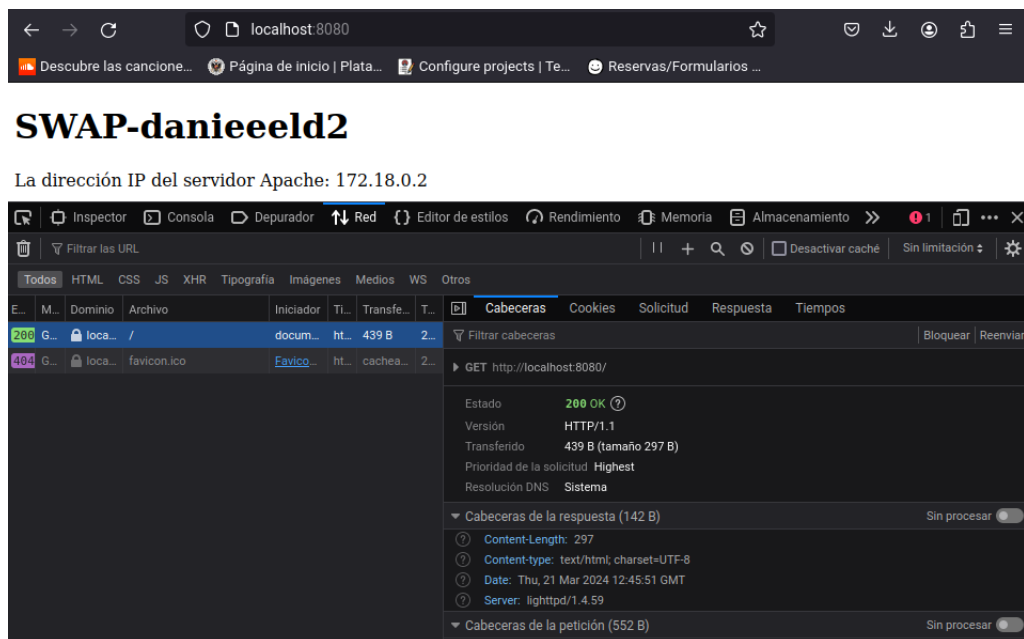
# Copiamos el archivo index.php
COPY web-danieeeld2/index.php /var/www/html

# 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

Pendiente

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

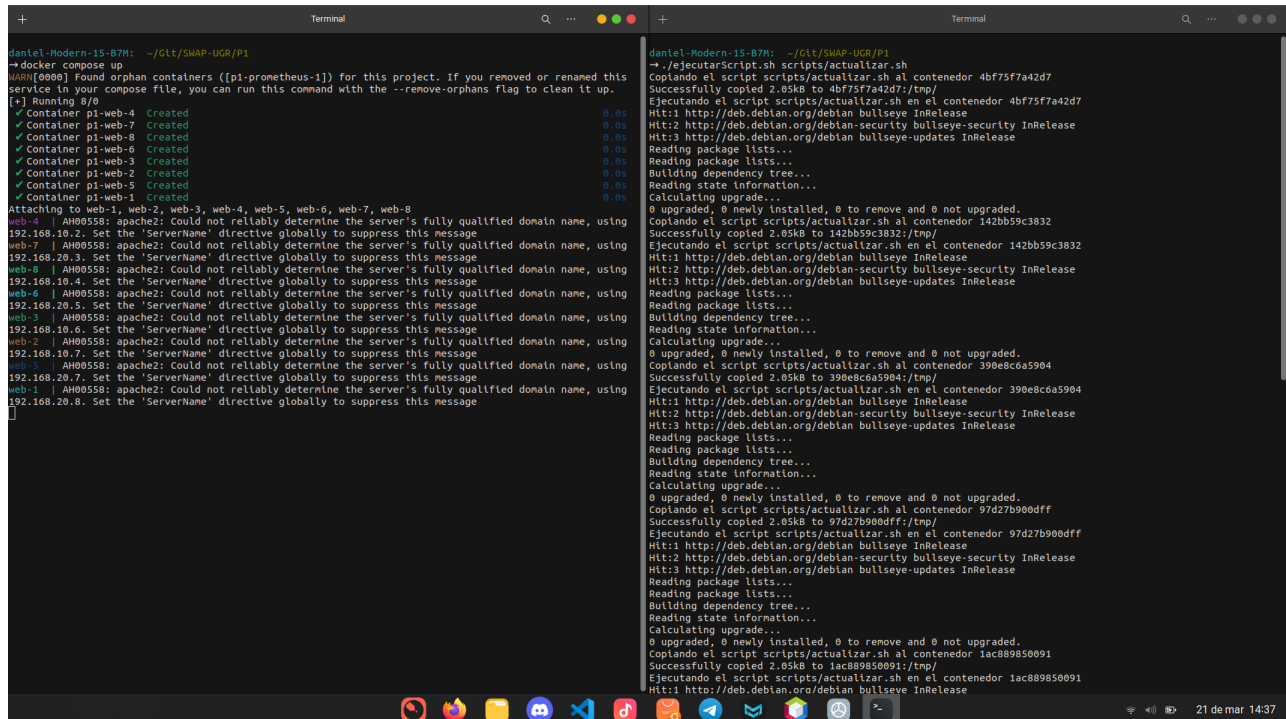
# Verificar que haya 1 argumento
if [ $# -eq 0 ]; then
    echo "Uso: $0 <script-a-ejecutar>"
    exit 1
fi

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

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

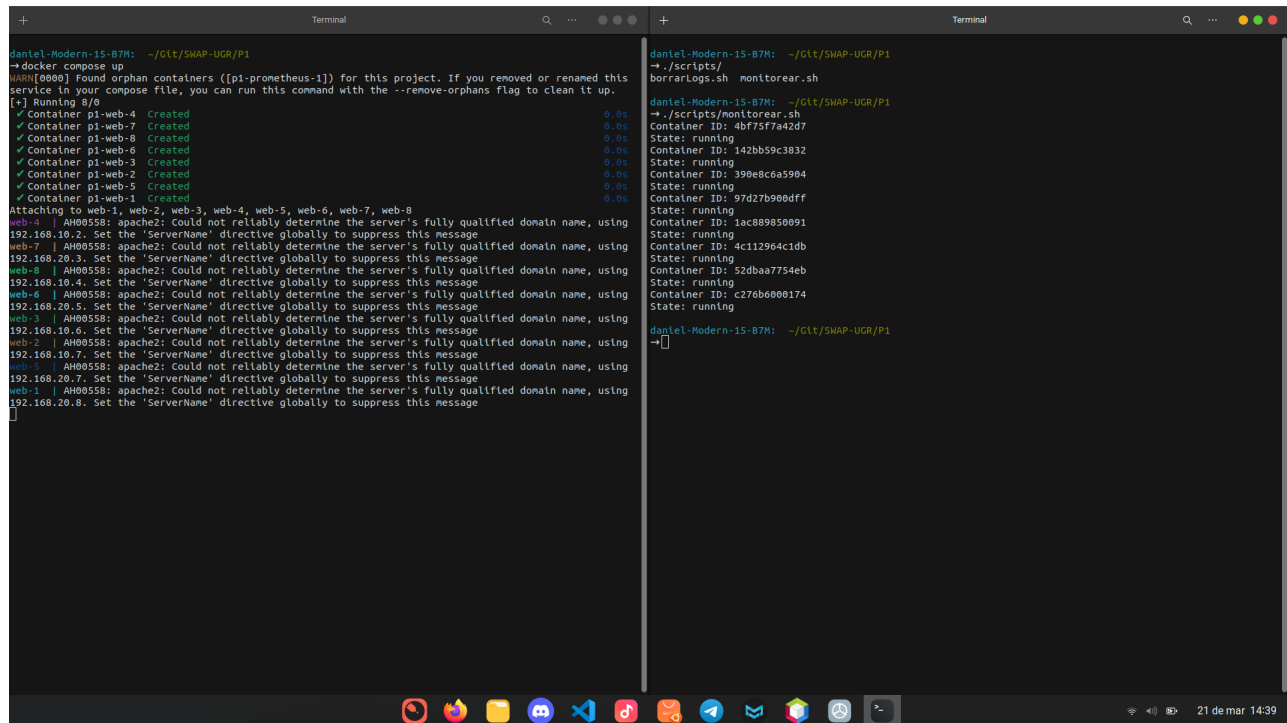
# Iterar sobre los IDs de los contenedores y ejecutar el script en cada uno (copiando la script en /tmp)
for container in $containers; do
    echo "Copiando el script $script al contenedor $container"
    docker cp "$script" "$container:/tmp/"
    echo "Ejecutando el script $script en el contenedor $container"
    docker exec "$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-15-b7m: ~/Glt/SNAP-UGR/P1
→ docker compose up
[+] Running 8/0
✔ Container p1-web-4 Created 0.0s
✔ Container p1-web-7 Created 0.0s
✔ Container p1-web-8 Created 0.0s
✔ Container p1-web-3 Created 0.0s
✔ Container p1-web-6 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-15-b7m: ~/Glt/SNAP-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 97d27b900dff
Successfully copied 2.05kB to 97d27b900dff:/tmp/
Ejecutando el script scripts/actualizar.sh en el contenedor 97d27b900dff
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)



The image shows two terminal windows. The left window displays the output of a Docker Compose command, showing the creation of eight containers (p1-web-1 to p1-web-8) and the execution of a script to set the 'ServerName' directive globally for each container. The right window shows the execution of a monitoring script (monitorrear.sh) which lists the running containers and their IDs.

```
daniel-Modern-15-B7M: ~/Glt/SWAP-UGR/P1
→docker compose up
[+] Running 8/0
 ✓ 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-15-B7M: ~/Glt/SWAP-UGR/P1
→ ./scripts/monitorrear.sh
daniel-Modern-15-B7M: ~/Glt/SWAP-UGR/P1
→ ./scripts/monitorrear.sh
Container ID: 4b775f7a42d7
State: running
Container ID: 142bb59c3832
State: running
Container ID: 399e8c6a5904
State: running
Container ID: 97d27b90dfff
State: running
Container ID: 1ac889850091
State: running
Container ID: 4c112964c1db
State: running
Container ID: 52dbaa7754eb
State: running
Container ID: c276b6000174
State: running
daniel-Modern-15-B7M: ~/Glt/SWAP-UGR/P1
→
```

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 est n 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:

The image shows two terminal windows side-by-side. The left window shows the output of 'docker compose up', which successfully creates a network 'pl-red-web' and several containers including 'pl-red-servicios', 'pl-red-web', and multiple 'pl-web' instances. It then shows the output of 'python3 scripts/conectividad.py', which reports that all containers are correctly connected to the specified subnets. The right window shows the output of 'docker network disconnect -f pl-red-web pl-web-1', which successfully disconnects the 'pl-web-1' container from the 'pl-red-web' network. A subsequent run of 'python3 scripts/conectividad.py' shows an error: 'Container 'pl-web-1' is missing connectivity to subnet(s): pl-red-web'.

```

Terminal
daniel-Modern-15-B7M: ~/Glt/SWAP-UGR/P1 [main 7:11]
→ docker compose up
[+] Running 1/1
✔ Network pl-red-servicios Created 0.1s
✔ Network pl-red-web Created 0.1s
[+] Running 10/10 orphan containers ([pl-network-configurator-1 pl-prometheus-1]) for this project. If yo
✔ Network pl-red-servicios Created 0.1s
✔ Network pl-red-web Created 0.1s
✔ Container pl-web Created 0.1s
✔ Container pl-web-1 Created 0.1s
✔ Container pl-web-2 Created 0.1s
✔ Container pl-web-3 Created 0.1s
✔ Container pl-web-4 Created 0.1s
✔ Container pl-web-5 Created 0.1s
✔ Container pl-web-6 Created 0.1s
✔ Container pl-web-7 Created 0.1s
✔ Container pl-web-8 Created 0.1s
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 7:11]
→ python3 scripts/conectividad.py
Todos los contenedores están correctamente conectados a las subredes especificadas.
daniel-Modern-15-B7M: ~/Glt/SWAP-UGR/P1 [main 7:11]
→ docker network disconnect -f pl-red-web pl-web-1
daniel-Modern-15-B7M: ~/Glt/SWAP-UGR/P1 [main 7:11]
→ python3 scripts/conectividad.py
Container 'pl-web-1' is missing connectivity to subnet(s): pl-red-web
daniel-Modern-15-B7M: ~/Glt/SWAP-UGR/P1 [main 7:11]
→

```

2.5. Monitoreo y Logging

Pendiente