



UNIVERSIDAD DE GRANADA

P1 - Servidores web y almacenamiento

Servidores Web de Altas Prestaciones

UGR - ETSIIT

Torres Ramos, Juan Luis

29 marzo de 2024

Índice

Índice	1
Tareas Básicas - B1 Configuración del Entorno	2
Tareas Básicas - B2 Creación del Dockerfile	3
Tareas Básicas - B3 Uso de Docker Compose	8
Tareas Básicas B4 - Despliegue y verificación de Contenedores	10
Tareas Básicas B5 - Pruebas Básicas	13
Tareas Avanzada A1 - Personalización del Dockerfile	14
Tareas Avanzada A2 - Creación de contendores con otros servidores web	18
Tarea Avanzada A3 - Gestión Avanzada de Redes	22
Tareas Avanzada A4 - Automatización con Scripts	25
Tarea Avanzada A5 - Monitoreo y Logging:	29

Tareas Básicas - B1 Configuración del Entorno

Vamos a crear un directorio en nuestra máquina local llamado *web_usuarioUGR*, dentro del directorio crearemos el *index.php* que mostrará una página web sencilla donde muestra nuestro nombre de usuario y la dirección IP del servidor Apache.

```
mkdir web_usuarioUGR
```

Análisis propuesta IA

genera un *index.php* que muestre "SWAP: nombre usuario y la dirección IP del servidor Apache"

index.php

```
<?php
// Obtener el nombre de usuario usando el comando shell 'whoami'
// $usuario = exec('whoami');
$usuario = "Torres Ramos, Juan Luis";

// Obtener la dirección IP del servidor Apache
$ipServidor = $_SERVER['SERVER_ADDR'];
?>

<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>SWAP: <?php echo $usuario; ?></title>
</head>
<body>
    <h1>SWAP: <?php echo $usuario; ?></h1>
    <p>Dirección IP del servidor Apache: <?php echo $ipServidor; ?></p>
</body>
</html>
```

Análisis

Chat gpt genera una página web simple con php, veo que con *\$usuario* y *\$ipServidor* usa php para obtener el nombre y la dirección IP del servidor. Hay que hacer un cambio porque *\$usuario* equivalía a hacer un *whoami* y este devolvió un *root*, yo quiero solamente que muestre mi nombre.

Tareas Básicas - B2 Creación del Dockerfile

En la raíz del proyecto, es decir, todo el proyecto está albergado en una carpeta llamada código y dentro de esta está la carpeta que hemos creado anteriormente. Ahora crearemos un *DockerfileApache_usuarioUGR*, en este caso pongo mi nombre *juanluis* en vez de usuario. Enseñó un tree de como va mi proyecto:

```
$ tree
.
├── DockerfileAPache_juanluisUGR
└── web_usuarioUGR
    └── index.php
```

Análisis propuesta IA

como personalizar el nombre de un dockerfile

Para personalizar el nombre de un archivo Dockerfile, puedes simplemente nombrarlo como desees. Por convención, se utiliza el nombre **"Dockerfile"**, pero si deseas usar un nombre diferente, simplemente renómbralo y asegúrate de hacer referencia al nuevo nombre en los comandos de Docker.

Por ejemplo, si deseas nombrar tu archivo Dockerfile como **"MiDockerfile"**, simplemente renómbralo y luego usa el parámetro `-f`` o `--file`` al construir la imagen Docker para especificar el nombre del archivo:

```
bash
`docker build -t nombre_de_la_imagen -f MiDockerfile .`
```

Aquí, ``nombre_de_la_imagen`` es el nombre que deseas darle a tu imagen Docker, y ``MiDockerfile`` es el nuevo nombre que has elegido para tu archivo Dockerfile. El punto al final indica el contexto de construcción, que generalmente es el directorio actual. Asegúrate de ejecutar el comando ``docker build`` desde el directorio que contiene tu Dockerfile personalizado.

Análisis

Chat gpt nos dice que podemos personalizar el nombre de nuestra imagen, pero a la hora de construirlo hay que usar el flag `-g "file"` para llamar a dicho archivo e indicar dónde se encuentra, en este caso la carpeta actual. Además nos da un ejemplo de build.

¿Qué herramientas de red son importantes para un dockerfile donde ejecuto un php dentro de un contenedor?

Para trabajar con un contenedor Docker que ejecuta una aplicación PHP y diagnosticar problemas de red, puedes utilizar diversas herramientas. Aquí te menciono algunas de las más comunes junto con instrucciones básicas sobre cómo instalarlas:

ping: ping se utiliza para verificar la conectividad de red entre dos hosts. Si no está instalado en tu contenedor, generalmente está disponible en la mayoría de las distribuciones de Linux.

Para instalar ping en una imagen basada en Debian/Ubuntu, puedes usar el siguiente comando en tu Dockerfile:

Dockerfile

```
RUN apt-get update && apt-get install -y iputils-ping
```

Para imágenes basadas en Alpine Linux:

Dockerfile

```
RUN apk add --no-cache iputils
```

curl: curl es una herramienta para transferir datos desde o hacia un servidor, utilizando varios protocolos, incluidos HTTP, HTTPS, FTP, etc.

Puedes instalar curl en una imagen basada en Debian/Ubuntu con el siguiente comando:

Dockerfile

```
RUN apt-get update && apt-get install -y curl
```

Para imágenes basadas en Alpine Linux:

Dockerfile

```
RUN apk add --no-cache curl
```

net-tools: Este paquete incluye herramientas como netstat, ifconfig, entre otras, que son útiles para diagnosticar problemas de red.

Para imágenes basadas en Debian/Ubuntu:

Dockerfile

```
RUN apt-get update && apt-get install -y net-tools
```

Para imágenes basadas en Alpine Linux:

Dockerfile

```
RUN apk add --no-cache net-tools
```

traceroute: traceroute se utiliza para rastrear la ruta que sigue un paquete IP desde tu contenedor hasta un host de destino.

Para imágenes basadas en Debian/Ubuntu:

Dockerfile

```
RUN apt-get update && apt-get install -y traceroute
```

Para imágenes basadas en Alpine Linux:

Dockerfile

```
RUN apk add --no-cache traceroute
```

Netcat: Netcat es una herramienta de red versátil que puede leer y escribir datos a través de conexiones de red utilizando TCP/IP. Puedes usarlo para transferir archivos, realizar escaneos de puertos, establecer conexiones de red, etc.

Para imágenes basadas en Debian/Ubuntu:

Dockerfile

```
RUN apt-get update && apt-get install -y netcat
```

Para imágenes basadas en Alpine Linux:

Dockerfile

```
RUN apk add --no-cache netcat-openbsd
```

Recuerda agregar estas instrucciones **RUN** en tu Dockerfile para instalar estas herramientas dentro del contenedor durante la construcción de la imagen. Esto te permitirá utilizarlas para diagnosticar problemas de red dentro del contenedor Docker que ejecuta tu aplicación PHP

Justificación

Chat pgt nos indica algunas herramientas red básicas a instalar para comenzar con la imagen. Estos paquetes son ip-utils, curl, net-tools, traceroute, netcat. Además nos dice que con RUN es como estaría reflejado en la imagen, además de una versión de sí utilizaremos Alpine Linux. Es una buena solución para comenzar con mi imagen, pero si luego se me presentan problemas específicos, necesitaré añadir otras herramientas e ir actualizando los paquetes que voy a instalar en mi imagen.

Ahora para crear el dockerfile, voy a seguir los siguientes pasos, basándome en el ejemplo de las diapositivas:

1. uso imagen debian:buster-slim (la que se usa en las prácticas), versión minimalista de debian
2. instalo apache, php, y algunas herramientas de red básica, borro luego los paquetes instalados para no tener basura en la imagen.
3. copio index.php que he creado antes en la dirección donde se ejecutara apache y borro el html que genera por defecto para que use solamente index.php
4. muestro puerto 80
5. establece el comando predeterminado para iniciar apache al principio

Dockerfile

```
# Establece la imagen base en Debian Buster slim
FROM debian:buster-slim

# Especifica el mantenedor de la imagen (obsoleto en Docker 1.13 y posteriores)
MAINTAINER Juan Luis Torres Ramos

# Actualiza los repositorios de apt-get y luego instala Apache, php y
herramientas de red por último borrar los paquetes que acabo de instalar, buena
práctica para limpiar el contenedor
RUN apt-get update && apt-get install -y \
    apache2\
    php php-cli \
    iputils-ping \
    netcat \
    traceroute \
    && rm -rf /var/lib/apt/lists/*

# Configuro el ServerName a localhost y evito el warning
RUN echo "ServerName localhost" >> /etc/apache2/apache2.conf

# Borro el de por defecto y copio tu código PHP a la carpeta de documentos de
Apache
RUN rm /var/www/html/index.html
COPY ./web_usuarioUGR/index.php /var/www/html/

# Se expone el puerto 80 para que pueda accederse al servidor web
EXPOSE 80

# Inicio Apache por defecto
CMD ["apache2ctl", "-D", "FOREGROUND"]
```

también veo que al ejecutar posteriormente el *dockerfile* me sale el siguiente warning:

```
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
```

En el siguiente enlace de stackoverflow me da una solución para evitar el warning

<https://stackoverflow.com/questions/46266527/ah00558-apache2-could-not-reliably-determine-the-servers-fully-qualified-domain-name>

añado esto a mi dockerfile

```
# Configura el ServerName a localhost y evito el warning
RUN echo "ServerName localhost" >> /etc/apache2/apache2.conf
```

Con el siguiente comando creo la imagen. Uso el flag -f según chat gpt para llamar a un dockerfile con el nombre personalizado

```
docker build -t juanluistr-apache-image:p1 -f
DockerfileApache_juanluisUGR .
```

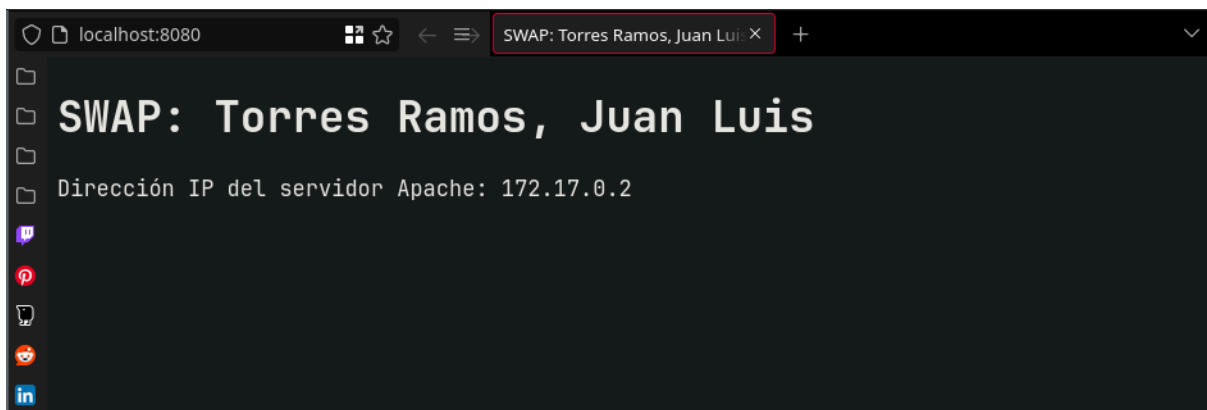
Ahora ejecuto la imagen:

```
docker run -d -p 8080:80 juanluistr-apache-image:p1
```

Para el resto de la práctica voy a trabajar con Docker-desktop, aunque si queremos comprobar que se ha ejecutado correctamente la imagen podemos ejecutar el siguiente comando

```
docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES 2c2bf69b3809
juanluistr-apache-image:p1 "apache2ctl -D FOREG..." 4 seconds ago Up 2
seconds 0.0.0.0:8080->80/tcp modest_satoshi
```

ahora que sé que se está ejecutando entro en <http://localhost:8080/> y obtengo el siguiente resultado



Tareas Básicas - B3 Uso de Docker Compose

Voy a crear el siguiente *docker-compose.yml* con las siguientes características:

1. Mi imagen anterior es *juanluistr-apache-image:p1* la que voy a utilizar en cada web
2. Voy a crear 8 contenedores llamados webX, X es del 1 al 8 con volúmenes donde se monte el directorio *web_usuarioUGR* en el directorio raíz de Apache en el contenedor
3. Tengo que añadir 2 redes a cada contenedor, una red llamada *red_web* con direccion *192.168.56.10.0/24* y otra red llamada *red_servicios* con direccion *192.168.10.0/24*

Análisis propuesta IA

cómo añadir las dos redes al contenedor, una red llamada *red_web* con direccion *192.168.10.0/24* y otra red llamada *red_servicios* con direccion *192.168.10.0/24* en mi *docker-compose.yml*

Para añadir dos redes distintas a un contenedor en tu archivo *`docker-compose.yml`*, debes definir las redes en la sección *`networks`* y luego asignar esas redes al servicio específico en la sección *`services`*. Aquí tienes un ejemplo de cómo podrías hacerlo:

```
`version: '3.8'  services:    tu_servicio:      image:      nombre_de_la_imagen      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: 192.168.11.0/24`
```

En este ejemplo:

- Definimos dos redes distintas, *`red_web`* y *`red_servicios`*, en la sección de *`networks`*.
- Para cada red, especificamos el controlador (*`driver`*) como *`bridge`*, que es uno de los controladores de red más comunes.
- Especificamos la configuración de IP para cada red utilizando *`ipam`*.
- Para cada red, proporcionamos una subred diferente.

Recuerda reemplazar *`tu_servicio`* con el nombre de tu servicio y *`nombre_de_la_imagen`* con el nombre de la imagen Docker que estás utilizando para tu servicio. Además, asegúrate de que las subredes sean diferentes para evitar conflictos de direcciones IP.

Análisis

Usa la versión 3.8 de docker (otra alternativa que he visto es usar la etiqueta “latest”, que viene a ser lo mismo) nos hace un ejemplo con “tu_servicio” y añade las redes red_web y red_servicios con las IP que le he indicado, además los configura como bridge. Nos recuerda asegurarnos de que las subredes que indiquemos sean diferentes para evitar conflictos de direcciones IP.

para definir cada servicio webX lo que hago es lo siguiente

1. especificar la imagen a usar (3.8 o latest)
2. específico el fichero dockerfile que se va a usar
3. veo los volúmenes que voy a usar en la imagen, son ./web_usuarioUGR cómo /var/www/html
4. Asigné el puerto en que se va a escuchar, para web1 sera 8081, para web2 será 8082 y así con el resto
5. para las redes defino como me ha indicado chat gpt y las indico en webX.

Docker-compose.yml (lo creamos en la raíz del proyecto)

```
version: '3.8'
services:
  web1:
    image: juanluistr-apache-image:p1
    build:
      context: .
      dockerfile: DockerfileApache_usuarioUGR
    volumes:
      ./web_usuarioUGR:/var/www/html
    networks:
      - red_web
      - red_servicios
    ports: "8082:80"

# .... (se repite la misma estructura para resto webX)

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

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

Tareas Básicas B4 - Despliegue y verificación de Contenedores

Ejecuto le docker-compose.yml que he creado anteriormente, en background con el flag -d

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

Los contenedores se llaman *codigo_webX_1* porque en la carpeta en la que se encuentra se llama codigo. Además pasa lo mismo para las redes, las redes se pasan a llamar *codigo_red_web* y *codigo_red_servicio*.

Compruebo que los contenedores están en ejecución:

```
docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED
STATUS	PORTS	NAMES	
3908b30fe285	juanluistr-apache-image:p1	"apache2ctl -D FOREG..."	About a
minute ago	Up About a minute	0.0.0.0:8087->80/tcp	codigo_web7_1
ab2ed7e82fbe	juanluistr-apache-image:p1	"apache2ctl -D FOREG..."	About a
minute ago	Up About a minute	0.0.0.0:8081->80/tcp	codigo_web1_1
f583635ffc7e	juanluistr-apache-image:p1	"apache2ctl -D FOREG..."	About a
minute ago	Up About a minute	0.0.0.0:8082->80/tcp	codigo_web2_1
a5e6892b4404	juanluistr-apache-image:p1	"apache2ctl -D FOREG..."	About a
minute ago	Up About a minute	0.0.0.0:8086->80/tcp	codigo_web6_1
6738d38e7a93	juanluistr-apache-image:p1	"apache2ctl -D FOREG..."	About a
minute ago	Up About a minute	0.0.0.0:8088->80/tcp	codigo_web8_1
6f7aaa02623d	juanluistr-apache-image:p1	"apache2ctl -D FOREG..."	About a
minute ago	Up About a minute	0.0.0.0:8085->80/tcp	codigo_web5_1
9cb3a275601d	juanluistr-apache-image:p1	"apache2ctl -D FOREG..."	About a
minute ago	Up About a minute	0.0.0.0:8084->80/tcp	codigo_web4_1
8e1f58ca61e2	juanluistr-apache-image:p1	"apache2ctl -D FOREG..."	About a
minute ago	Up About a minute	0.0.0.0:8083->80/tcp	codigo_web3_1

Voy a verificar que cada contenedor tiene una IP asignada para red_web y red_servicios

```
docker network ls
```

NETWORK ID	NAME	DRIVER	SCOPE
fa9889c21915	bridge	bridge	local
a5dafdddfc30e	codigo_red_servicios	bridge	local
ccb50156733e	codigo_red_web	bridge	local
a8f66e803b44	host	host	local
8d8746967d21	none	null	local
5cc2343d50cf	red1	bridge	local
ba205e2db5f3	red2	bridge	local

En docker-desktop, si vamos al contenedor -> view details -> inspect, podemos ver las IP que tiene asignada el contenedor

compruebo que las redes están bien asociadas para eso hago un *inspect* de cada una,

devolverá un JSON con la información de red asociada a cada contenedor

```
docker network inspect codigo_red_web
[
  {
    "Name": "codigo_red_web",
    "Id": "e67301b7307b5255efd3cc1fc11dc5a09ff53099f19d6fab9c7a7588dae7d050",
    "Created": "2024-03-29T10:26:31.185995849Z",
    "Scope": "local",
    "Driver": "bridge",
    "EnableIPv6": false,
    "IPAM": {
      "Driver": "default",
      "Options": null,
      "Config": [
        {
          "Subnet": "192.168.10.0/24"
        }
      ]
    },
    "Internal": false,
    "Attachable": true,
    "Ingress": false,
    "ConfigFrom": {
      "Network": ""
    },
    "ConfigOnly": false,
    "Containers": {
      "3908b30fe285866526eed49d98cf555e29a71b67cd5b287d44b28be85ae7c446": {
        "Name": "codigo_web7_1",
        "EndpointID": "a0dcf0431f6eeb5da46de9a5d5087ee5cb203ed4579a3ba8e20bdae95eb7479f",
        "MacAddress": "02:42:c0:a8:0a:03",
        "IPv4Address": "192.168.10.3/24",
        "IPv6Address": ""
      },
      "6738d38e7a93d38d90e57b86d70e9fa7516377024420ef5ac09298526a6c0972": {
        "Name": "codigo_web8_1",
        "EndpointID": "65a393c302d1ebe306b86ffdf6c1a51951ad1de4171fea6988b31a8fadaeeb0e",
        "MacAddress": "02:42:c0:a8:0a:08",
        "IPv4Address": "192.168.10.8/24",
        "IPv6Address": ""
      },
      "6f7aaa02623d01709d00391c39b974dfde8a135d04d5d34a317c5cf164170265": {
        "Name": "codigo_web5_1",
        "EndpointID": "f431fd554c89e9bf89c9e85afbc47e75de69ff3d1a66758b470c9474824e5bdf",
        "MacAddress": "02:42:c0:a8:0a:06",
        "IPv4Address": "192.168.10.6/24",
        "IPv6Address": ""
      }
    },
    "Options": {},
    "Labels": {
      "com.docker.compose.network": "red_web",
      "com.docker.compose.project": "codigo",
      "com.docker.compose.version": "1.29.2"
    }
  }
]
```

```

docker network inspect codigo_red_servicio
[
  {
    "Name": "codigo_red_servicios",
    "Id": "6934a5f21ef7177c06760337c184a8e5b7391014cbec4d19b5b87054ae026d95",
    "Created": "2024-03-29T10:26:31.23715611Z",
    "Scope": "local",
    "Driver": "bridge",
    "EnableIPv6": false,
    "IPAM": {
      "Driver": "default",
      "Options": null,
      "Config": [
        {
          "Subnet": "192.168.20.0/24"
        }
      ]
    },
    "Internal": false,
    "Attachable": true,
    "Ingress": false,
    "ConfigFrom": {
      "Network": ""
    },
    "ConfigOnly": false,
    "Containers": {
      "3908b30fe285866526eed49d98cf555e29a71b67cd5b287d44b28be85ae7c446": {
        "Name": "codigo_web7_1",
        "EndpointID": "d1d1056bb8dc912abdce59d0fd60946bb83bc12a1efa9941dd0067dbb7bfaf84",
        "MacAddress": "02:42:c0:a8:14:09",
        "IPv4Address": "192.168.20.9/24",
        "IPv6Address": ""
      },
      "6738d38e7a93d38d90e57b86d70e9fa7516377024420ef5ac09298526a6c0972": {
        "Name": "codigo_web8_1",
        "EndpointID": "7cdf92947fd7f8086dae5bfb2d0f38172b25d9c341c6566e1004f24aaedad449",
        "MacAddress": "02:42:c0:a8:14:06",
        "IPv4Address": "192.168.20.6/24",
        "IPv6Address": ""
      },
      ...
    },
    "Options": {},
    "Labels": {
      "com.docker.compose.network": "red_servicios",
      "com.docker.compose.project": "codigo",
      "com.docker.compose.version": "1.29.2"
    }
  }
]

```

veo que puedo ver la información de cada red generada y que está asociada a cada contenedor. Por ejemplo el contenedor `codigo_web7_1` tiene la ip de red_servicios 192.168.20.9/24 y red_web será 192.168.10.8/24.

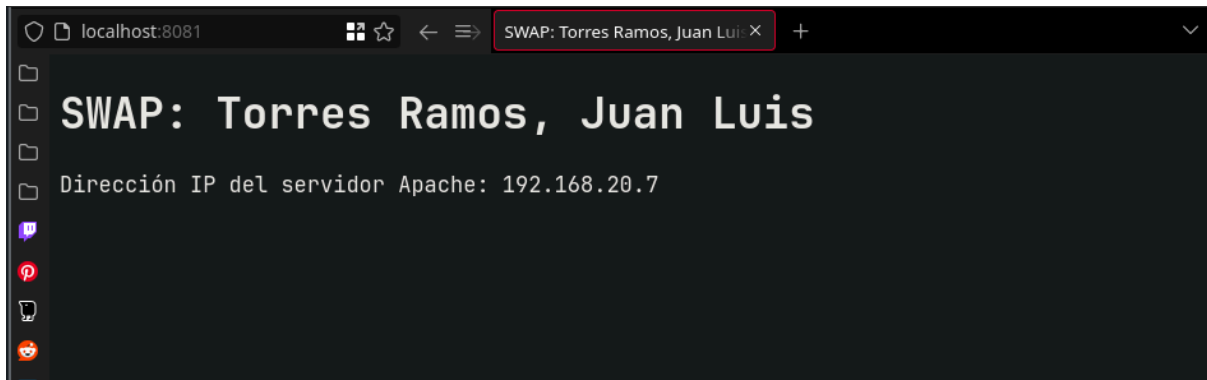
Los comandos para ver la información de las redes y ejecutar están en el guión *P0:contenerización*.

Tareas Básicas B5 - Pruebas Básicas

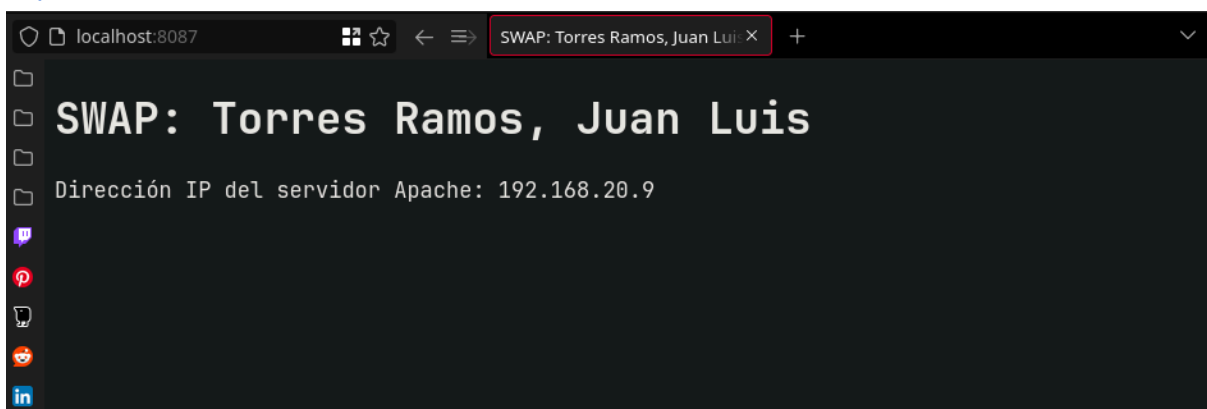
Tras haber ejecutado el docker-compose.yml, voy a acceder a la página web de cada contenedor usando su dirección IP y veo qué información muestra

Recuerdo que para cada contenedor he de acceder a localhost808x, al cual le haya asignado

por ejemplo para el web1 tendre que acceder a <http://localhost:8081/>



para el ejemplo que hemos visto tras inspeccionar web7 me conectare a <http://localhost:8087/>



veo que la IP de red_servicios coincide con la IP del servidor Apache. Pienso que red_servicio funcionará para cosas del backend, a la hora de que los contenedores se hablen entre sí y red_web será para poder conectarnos desde fuera, por ejemplo de la máquina local tal que <http://192.168.10.8:8081> (habria que habilitarlo previamente),

Tareas Avanzada A1 - Personalización del Dockerfile

Voy a modificar el Dockerfile para incluir configuraciones personalizadas de Apache o PHP

Para personalizar Apache vamos a modificar Apache2.conf y para php crearemos un archivo php-config.ini que luego pasaremos a nuestro contenedor.

Anteriormente habíamos ya personalizado apache configurando el ServerName a localhost de la siguiente manera en nuestro docker image:

```
RUN echo "ServerName localhost" >> /etc/apache2/apache2.conf
```

Para añadir otra configuración personalizada voy a seguir las indicaciones de esta página web que me han parecido interesantes sobre seguridad de Apache.

<https://www.unixmen.com/how-to-disable-server-signature-using-htaccess-or-by-editing-a-pache/>

Normalmente, Apache muestra su versión y el nombre del sistema operativo en los errores.

Un atacante puede utilizar esta información para lanzar un ataque utilizando las vulnerabilidades disponibles públicamente en el servidor particular o la versión del sistema operativo.

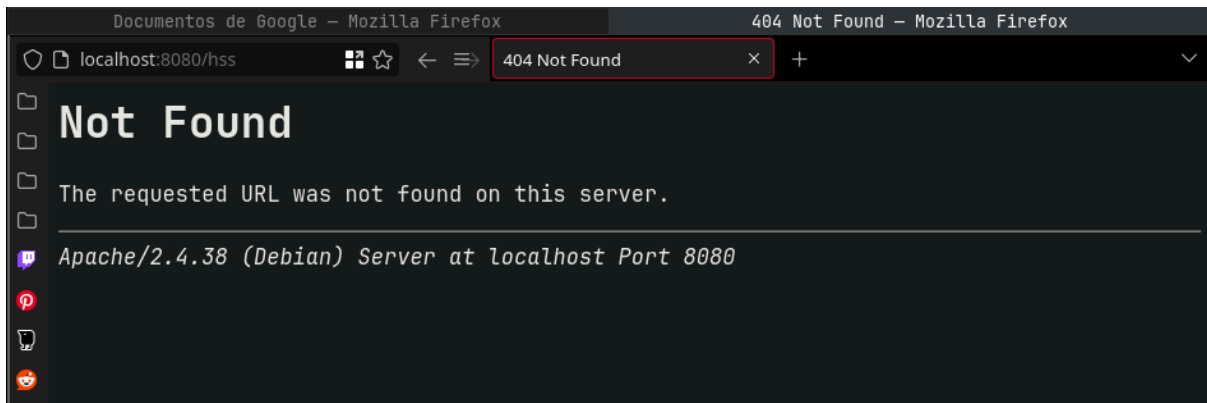
Para evitar que el servidor web Apache muestre esta información, hay que cambiar el parámetro “firma del servidor”, opción disponible en el archivo de configuración de apache (apache2.conf) . Por defecto está en “ON”, debe ponerlo en “OFF”.

También puede establecer “ServerTokens Prod”, que indica al servidor web que devuelva sólo Apache y que suprima las versiones mayores y menores del sistema operativo. Para hacer lo indicado añadido esto en el dockerfile

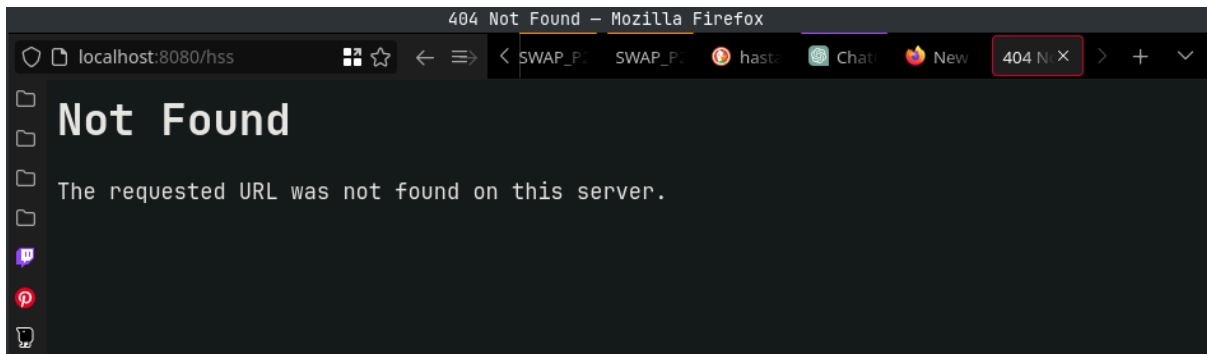
```
# A.1: Oculto la versión de Apache y la información del sistema operativo
RUN echo "ServerTokens Prod" >> /etc/apache2/apache2.conf && \
    echo "ServerSignature Off" >> /etc/apache2/apache2.conf
```

luego cuando entro en <http://localhost/hss> luego de haber ejecutado la imagen, no aparece la información de Apache y el Sistema Operativo que uso.

Antes



Después



Para personalizar PHP he creado el siguiente archivo:

php-config.ini

```
memory_limit = 256M
upload_max_filesize = 20M
post_max_size = 25M
```

- memory_limit: límite máximo de memoria que un script PHP puede utilizar durante su ejecución
- Upload_max_filesize: tamaño máximo permitido para carga de archivos a través de formatos HTML en PHP
- post_max_size: tamaño máximo permitido para los datos POST enviados a través de formularios HTML en PHP

el dockerfile tendríamos que añadir

```
# A.1: Copio la configuración personalizada de PHP
COPY ./web_usuarioUGR/php-config.ini
/etc/php/7.3/apache2/conf.d/custom.ini
```

ahora construyo la imagen de nuevo e inicializo el contenedor sin problemas. Entrando en el bash del contenedor podría ver que los cambios se han aplicado correctamente.

Análisis propuesta IA

que tengo que modificar para tener personalizada apache y php

Para personalizar Apache y PHP en tu Dockerfile, necesitarás realizar los siguientes pasos:

Crear archivos de configuración personalizados: Como se mencionó anteriormente, necesitas crear archivos de configuración personalizados para Apache y PHP según tus necesidades. Los ejemplos proporcionados son solo básicos, pero puedes agregar o modificar configuraciones según lo requiera tu aplicación.

Copiar los archivos de configuración personalizados: En el Dockerfile, asegúrate de copiar los archivos de configuración personalizados al directorio adecuado dentro del contenedor. Esto se realiza utilizando la instrucción COPY. Asegúrate de que los nombres de los archivos coincidan con los que proporcionas y ajusta las rutas según sea necesario.

Reiniciar Apache (si es necesario): Dependiendo de los cambios que realices en la configuración de Apache, es posible que necesites

reiniciar el servidor web dentro del contenedor. Puedes hacer esto agregando comandos al final de tu Dockerfile para reiniciar Apache.

genera los archivos `apache-config-conf` y `php-config.ini`

Claro, puedo darte ejemplos básicos de cómo podrían ser estos archivos de configuración:

`apache-config.conf`:

`apache`

`<VirtualHost *:80>`

`ServerAdmin webmaster@localhost`

`DocumentRoot /var/www/html`

`ErrorLog ${APACHE_LOG_DIR}/error.log`

`CustomLog ${APACHE_LOG_DIR}/access.log combined`

`</VirtualHost>`

Este es un ejemplo básico de un archivo de configuración de Apache para un sitio virtual. Define el administrador del servidor, la ruta del documento raíz y los registros de errores y acceso.

`php-config.ini`:

`ini`

`memory_limit = 256M`

`upload_max_filesize = 20M`

`post_max_size = 25M`

Este es un ejemplo básico de un archivo de configuración de PHP. Define límites de memoria, tamaño máximo de carga de archivos y tamaño máximo de publicación.

Ajusta estos archivos según tus necesidades específicas de configuración. Luego, guárdalos en el mismo directorio que tu Dockerfile o ajusta la ruta en la instrucción `COPY` dentro del Dockerfile para que se copien correctamente dentro del contenedor.

Análisis

Con estas dos preguntas Chat gpt me ha podido ayudar en el apartado A1. La parte de configurar apache no me ha llegado a convencer ya que los cambios que aplican, en mi opinión no son muy significativos a primera vista. He optado por una configuración básica de seguridad de apache para deshabilitar “server signature”. Mientras que la configuración de ph me parece bien, un archivo básico y fácil de entender donde define el límite de memoria, tamaño máximo de carga de archivos y tamaño máximo de publicación. Si tuviésemos un archivo php con un uso de memoria mayor que el propio contenedor tendríamos problemas.

Tareas Avanzada A2 - Creación de contenedores con otros servidores web

Para este apartado, he decidido usar node para crear contenedores con otros servidores web.

1. necesito crear un server.js, que se ejecutara dentro del contenedor para hacer un deploy de un servidor web, uso el puerto 8080.
 - En node.js, para crear servicios web se suele usar el paquete express. Defino 2 rutas: para enseñar el index.html, para obtener la ip y luego lo pongo a escuchar, para obtener la ip me he basado en <https://stackoverflow.com/questions/10849687/express-js-how-to-get-remote-client-address>

```
const express = require('express');

const app = express();
const port = 8080;

app.get('/', (req, res) => {
  res.sendFile(__dirname + '/index.html');
});

// ruta para obtener la ip del cliente
app.get('/ip', (req, res) => {
  let ip = req.headers['x-forwarded-for'] ||
  req.connection.remoteAddress;

  // Eliminar la parte "::ffff:" si está presente
  if (ip.substr(0, 7) === '::ffff:') {
    ip = ip.substr(7);
  }
  res.send(ip);
});

app.listen(port, () => {
  console.log(`Servidor Node.js escuchando en
  http://localhost:${port}`);
});
```

tras tener el server.js podemos pasar al index.html (con node no puedo trabajar con php, mejor trabajar con javascript) Lo que cambia es que con fetch hago un GET a la ruta /ip, con response.text obtengo lo que muestra /ip y luego lo enseño en el html , hago un catch en el caso de que hubiese algún error.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
initial-scale=1.0">
  <title>Información de IP</title>
</head>
<body>
  <h1>SWAP: Torres Ramos, Juan Luis </h1>
  <p>La dirección IP del servidor Node es: <span id="ip"></span></p>

  <script>
    fetch('/ip')
      .then(response => response.text())
      .then(data => {
        document.getElementById('ip').innerText = data;
      })
      .catch(error => console.error('Error al obtener la IP:',
error));
  </script>
</body>
</html>
```

Ahora podemos pasar al dockerfile, que será el siguiente

```
FROM node:latest
WORKDIR /app

COPY ./web_usuarioUGR/index.html .
COPY ./web_usuarioUGR/server.js .

RUN npm install express ip

EXPOSE 80

CMD ["node", "server.js"]
```

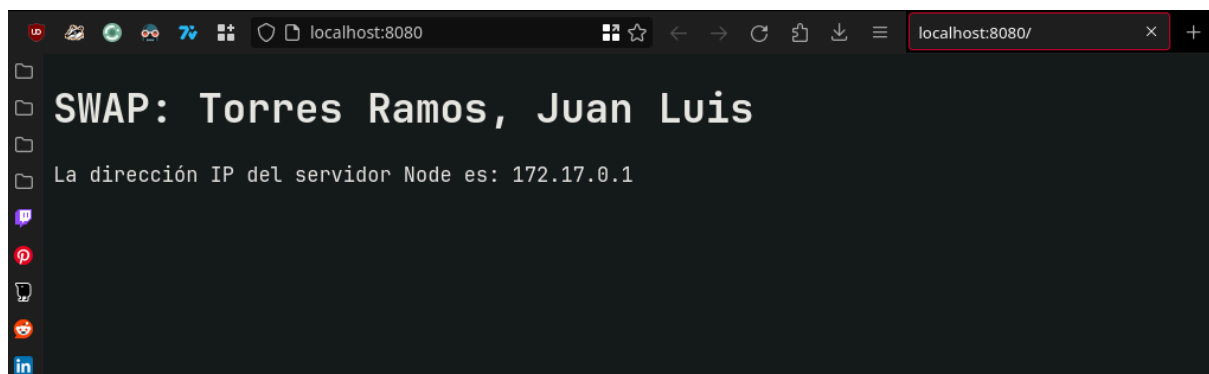
Cambiar el nombre de index2.html a index.html

```
docker build -t juanluistr-apache-image-node:p1 -f
DockerfileApache_juanluisUGR_node .
docker run -d -p 8080:80 juanluistr-apache-image-node:p1
```

Me baso en una imagen de node (latest), creo una carpeta donde voy a pegar tanto el index como server, instalo express y ip que es lo que uso en server.js, y ejecutar node server.js.

```
docker build -t juanluistr-apache-image-node:p1 -f
DockerfileApache_juanluisUGR_node .
docker run -d -p 8080:8080 juanluistr-apache-image-node:p1
```

El resultado es el siguiente entrando en <http://localhost:8080> :



Análisis propuesta IA

no quiero que muestre la parte de ffff, sino solo muestre la ip

Entiendo. Para obtener solo la dirección IP de versión 4 sin la parte "::ffff:", puedes hacer una pequeña modificación en el servidor Node.js para extraer solo la parte de la dirección IPv4. Aquí está cómo puedes hacerlo:

```
javascript
const express = require('express');

const app = express();
const port = 8080;

app.get('/', (req, res) => {
  res.sendFile(__dirname + '/index.html');
});

app.get('/ip', (req, res) => {
  let ip = req.headers['x-forwarded-for'] ||
  req.connection.remoteAddress;
  // Eliminar la parte "::ffff:" si está presente
  if (ip.substr(0, 7) === '::ffff:') {
    ip = ip.substr(7);
  }
  res.send(ip);
});

app.listen(port, () => {
  console.log(`Servidor Node.js escuchando en
  http://localhost:${port}`);
});
```

Con esta modificación, el servidor verificará si la dirección IP comienza con "::ffff:". Si es así, eliminará esa parte y devolverá solo la dirección IPv4. Esto debería mostrar solo la dirección IP sin la parte "::ffff:".

cuando obtenía la ip me mostraba delante ::ffff:, no soy muy experimentado con javascript,. Entonces chat gpt me genero una función para eliminar dicha parte de forma sencilla usando la función substr.

Tarea Avanzada A3 - Gestión Avanzada de Redes

Vamos a configurar las redes de tal manera que el contenedor 1 no se pueda comunicar con el contenedor 7, para ello vamos a hacer uso de *iptables*

1. apuntamos la ip de red_web del contenedor 7
lo vemos como hemos dicho antes: 192.168.10.4
en mi caso el contenedor 1 tiene que ser root , para ello le aplico el tag `privileged: true` en el docker-compose

2. en el contenedor 1 ejecutamos el siguiente comando

```
docker exec -it codigo_web1_1 iptables -A OUTPUT -d 192.168.10.4 -j DROP
```

3. compruebo que se ha aplicado los cambio listando las reglas de iptables dentro del contenedor

```
# iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source                destination

Chain FORWARD (policy ACCEPT)
target     prot opt source                destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source                destination
DROP      all  --  anywhere             codigo_web7_1.codigo_red_web
```

4. hacemos un ping de comprobacion
contenedor 4: 192.168.10.9
hacemos un ping al contenedor 7

```
# ping 192.168.10.4
PING 192.168.10.4 (192.168.10.4) 56(84) bytes of data.
ping: sendmsg: Operation not permitted
ping: sendmsg: Operation not permitted
ping: sendmsg: Operation not permitted
ping: sendmsg: Operation not permitted
ping: sendmsg: Operation not permitted
ping: sendmsg: Operation not permitted
^C
--- 192.168.10.4 ping statistics ---
6 packets transmitted, 0 received, 100% packet loss, time 116ms
```

hacemos un ping al contenedor 4 (este caso es 192.168.10.9)

```
# ping 192.168.10.9
PING 192.168.10.9 (192.168.10.9) 56(84) bytes of data.
64 bytes from 192.168.10.9: icmp_seq=1 ttl=64 time=0.277 ms
64 bytes from 192.168.10.9: icmp_seq=2 ttl=64 time=0.127 ms
64 bytes from 192.168.10.9: icmp_seq=3 ttl=64 time=0.125 ms
64 bytes from 192.168.10.9: icmp_seq=4 ttl=64 time=0.124 ms
^C
```

Análisis propuesta IA

ahora lo que quiero hacer lo siguiente, configura las redes de tal manera que el contenedor 1 no se pueda comunicar con el contenedor 7, para ello vamos a hacer uso de iptables. Luego se ven dentro de los contenedores que no puedo hacer ping entre los contenedores

Para configurar las redes de modo que el contenedor 1 no pueda comunicarse con el contenedor 7 usando iptables, puedes realizar lo siguiente:

Accede al contenedor 1 y ejecuta el siguiente comando para listar las interfaces de red:

```
bash
```

```
docker exec -it <nombre_contenedor_1> ifconfig
```

Anota la dirección IP asignada a la interfaz de red en la red red_web. Supongamos que es 192.168.10.2.

Accede al contenedor 7 y ejecuta el mismo comando para obtener la dirección IP de la interfaz de red en la misma red red_web. Supongamos que es 192.168.10.7.

Ahora, ejecuta el siguiente comando en el contenedor 1 para configurar una regla de iptables que bloqueará todo el tráfico hacia el contenedor 7:

```
sql
```

```
docker exec -it <nombre_contenedor_1> iptables -A OUTPUT -d 192.168.10.7 -j DROP
```

Este comando añade una regla al final de la cadena de salida (OUTPUT) que descarta cualquier paquete dirigido a la dirección IP del contenedor 7 (192.168.10.7).

Con esta configuración, el contenedor 1 no podrá comunicarse con el contenedor 7 a través de la red red_web. Para verificarlo, puedes intentar hacer ping desde el contenedor 1 al contenedor 7 y viceversa. Deberías ver que los intentos de ping fallan.

Es importante tener en cuenta que esta configuración bloqueará todo el tráfico entre los contenedores 1 y 7 en la red `red_web`. Si necesitas una configuración más específica, como permitir ciertos tipos de tráfico o comunicación en un solo sentido, deberás ajustar las reglas de iptables en consecuencia.

Análisis

Tuve un problema porque no sabía si usar el tag `OUTPUT` o `FORWARD`. Le he especificado el caso a chat gpt y me recomienda usar el tag `output`, luego con buscar la ip del contenedor que quiero bloquear, este caso el contenedor 7, de la ip de la `red_web`, al aplicar el comando `iptables` compruebo luego que no puedo hacer ping desde el contenedor 1 a la dirección que he hecho `DROP`. Esto de primeras solo funciona cuando arrancas el contenedor y miras las ip que se han asociado, ya que estas cambian cada vez que arrancas el contenedor. Se podría automatizar con un script y modificando el `docker-compose` creando un nuevo servicio que tuviese como objetivo ejecutar el script (reconocer las ip + aplicar iptables)

Tareas Avanzada A4 - Automatización con Scripts

En este apartado, voy a crear varios script que pueden ser interesantes a la hora de interactuar con Docker

1. script.sh, este script se encarga, primero de listar los contenedores existentes, luego inicia el conjunto de contenedores del apartado B4, verifica que todos los contenedores se han inicializado correctamente y maneja los casos en los que los contenedores pueden no iniciarse correctamente. Me he basado este script en este: <https://github.com/rcaciocamacho/DockerMonitoring/blob/main/docker-monitoring.sh>

`check_and_retry` es una función que intenta iniciar un contenedor específico, si no tiene éxito lo intenta nuevamente un número determinado de veces

```
#!/bin/bash
# Comando para listar todos Los contenedores (nombre y estado)
list_cmd="docker ps -a --format '{{.Names}}:{{.State}}'"
# Comando para iniciar un contenedor
start_cmd="docker start"

# Función para verificar y reintentar contenedores problemáticos
function check_and_retry_container {
    local container_name=$1
    local attempts=0
    local max_attempts=2
    local wait_time=5 # Tiempo de espera en segundos entre intentos
    while [ $attempts -lt $max_attempts ]; do
        ((attempts++))
        echo "Intentando iniciar $container_name, intento $attempts"
        $start_cmd $container_name > /dev/null
        sleep $wait_time # Esperar antes de verificar el estado
        # Verificar el estado después del intento de inicio
        local status=$(docker inspect --format '{{.State.Running}}' $container_name)
        if [ "$status" == "true" ]; then
            echo "$container_name iniciado exitosamente."
            return 0 # Éxito, el contenedor está corriendo
        fi
    done

    echo "Error: No se pudo iniciar $container_name después de $max_attempts intentos."
    return 1 # Fracaso después de reintentos
}

# Inicializar la variable para guardar contenedores con problemas persistentes
problem_containers=""

# Obtener y procesar La lista de contenedores antes de inicializar Docker Compose
echo "Listando contenedores antes de inicializar Docker Compose:"
eval $list_cmd
```

```

# Iniciar Docker Compose
echo "Inicializando Docker Compose..."
docker-compose up -d

# Esperar unos segundos para que los contenedores se inicien completamente
sleep 10

# Comprobar que todos los 7 contenedores se hayan ejecutado correctamente
echo "Comprobando el estado de los contenedores:"
for i in {1..7}; do
    container_name="codigo_web${i}_1"
    container_state=$(docker inspect --format '{{.State.Running}}' $container_name)
    if [ "$container_state" != "true" ]; then
        problem_containers+="$container_name "
    fi
done

# Notificar sobre contenedores con problemas
if [ -n "$problem_containers" ]; then
    echo "Error: Los siguientes contenedores no se han ejecutado correctamente: $problem_containers"
else
    echo "Todos los 7 contenedores se han ejecutado correctamente."
fi

```

Resultado

```

./script.sh
Listando contenedores antes de inicializar Docker Compose:
codigo_web1_1:running
codigo_web4_1:running
codigo_web5_1:running
codigo_web2_1:running
codigo_web3_1:running
codigo_web7_1:running
codigo_web8_1:running
codigo_web6_1:running
busy_noether:exited
kind_tu:exited
youthful_wilson:exited
Inicializando Docker Compose...
codigo_web8_1 is up-to-date
codigo_web2_1 is up-to-date
codigo_web3_1 is up-to-date
codigo_web5_1 is up-to-date
codigo_web6_1 is up-to-date
codigo_web4_1 is up-to-date
codigo_web7_1 is up-to-date
codigo_web1_1 is up-to-date
Comprobando el estado de los contenedores:
Todos los 7 contenedores se han ejecutado correctamente.

```

2. *script_mantenimiento.sh*. En la imagen, he añadido un script que limpia el cache, actualiza los paquetes y comprueba el estado del contenedor. Luego, he

configurado un cron job mediante crontab para ejecutar este script cada minuto dentro del contenedor.

```
#!/bin/bash

# Limpiando caché y temporales
echo "Limpiando caché y temporales..."
apt-get clean
apt-get autoclean
apt-get autoremove -y

# Actualizar paquetes del sistema
echo "Actualizando paquetes del sistema..."
apt-get update
apt-get upgrade -y

service apache2 status

echo "Tareas de mantenimiento completadas dentro del contenedor."
```

he añadido esto al dockerfile , además de descargar cron previamente en el dockerfile:

```
COPY ./web_usuarioUGR/script_mantenimiento.sh /root/script_mantenimiento.sh
RUN chmod +x /root/script_mantenimiento.sh
COPY ./web_usuarioUGR/crontab /etc/cron.d/crontab
RUN chmod 0644 /etc/cron.d/crontab && crontab /etc/cron.d/crontab
```

En el archivo crontab que he necesitado crear he escrito lo siguiente, para que se ejecute cada minuto:

```
* * * * * /root/script_mantenimiento.sh
```

Compruebo dentro del contenedor que se ha aplicado el cron job

```
# crontab -l
* * * * * /root/script_mantenimiento.sh
```

Al final, dentro del contenedor podemos comprobar que el script se ejecuta correctamente:

```
Limpiando caché y temporales...
Reading package lists... Done
Building dependency tree
Reading state information... Done
Reading package lists... Done
Building dependency tree
Reading state information... Done
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
Actualizando paquetes del sistema...
Get:1 http://deb.debian.org/debian buster InRelease [122 kB]
Get:2 http://deb.debian.org/debian-security buster/updates InRelease
[34.8 kB]
Get:3 http://deb.debian.org/debian buster-updates InRelease [56.6 kB]
Get:4 http://deb.debian.org/debian buster/main amd64 Packages [7909 kB]
Get:5 http://deb.debian.org/debian-security buster/updates/main amd64
Packages [590 kB]
Get:6 http://deb.debian.org/debian buster-updates/main amd64 Packages
[8788 B]
Fetched 8721 kB in 3s (2628 kB/s)
Reading package lists... Done
Reading package lists... Done
Building dependency tree
Reading state information... Done
Calculating upgrade... Done
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
[ ok ] apache2 is running.
Tareas de mantenimiento completadas dentro del contenedor.
```

Para esta sección me he basado en

<https://www.howtogeek.com/devops/how-to-use-cron-with-your-docker-containers/>
<https://cronitor.io/guides/running-cron-jobs-inside-a-docker-container>

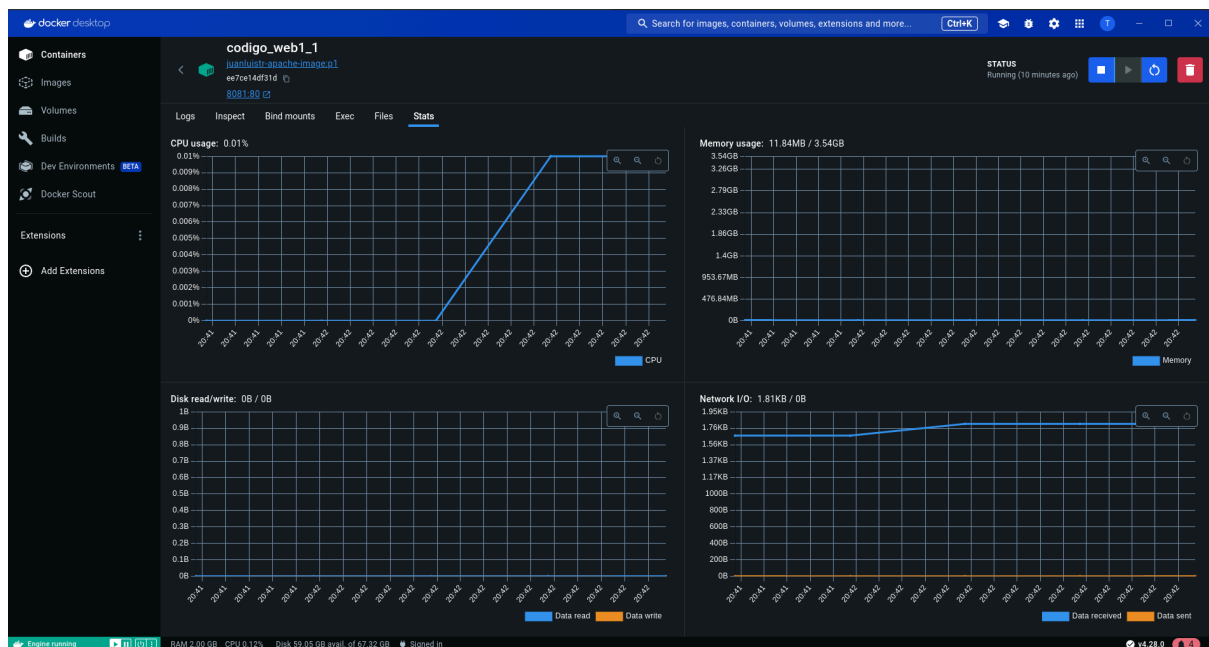
y

Tarea Avanzada A5 - Monitoreo y Logging:

Con el mismo docker-desktop podemos hacer un monitoreo básico de nuestros contenedores. Entramos en el contenedor -> en los 3 puntos -> view details stats.

- *Stat*: se encuentra las gráficas sobre la información de monitoreo del contenedor, como el uso de CPU, memoria, almacenamiento y actividad de red
- *Logs*: Muestra la salida de registro del contenedor, incluye mensajes de error y salida estándar generados por los procesos en ejecución dentro del contenedor
- *Inspect*: Muestra información del contenedor en forma de JSON.
- *mounts*: Enseña los directorios montados desde el host en el contenedor Docker y viceversa.
- *Exec*: Puedes entrar en el bash dentro del contenedor
- *Files*: Muestra el árbol de archivos del contenedor

Stats

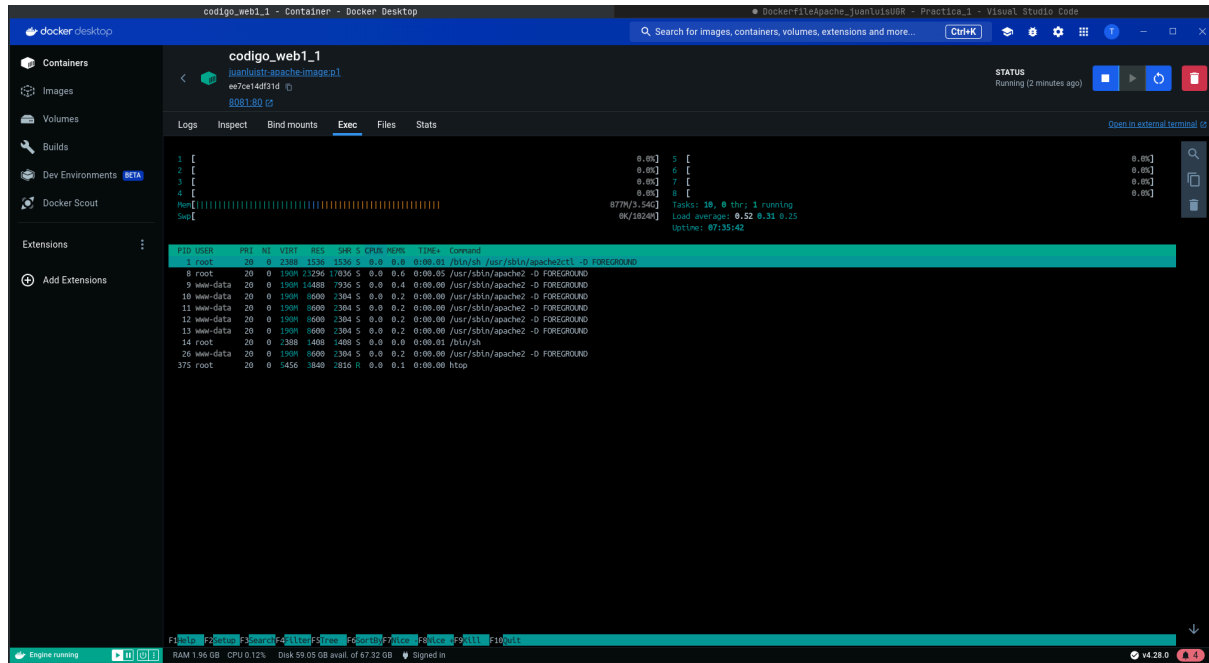


análisis

Podemos ver que el uso de la CPU es muy bajo, sobre un 0,01%, usan además poca memoria sobre 11,84MB/3,54 GB, la lectura de disco también es baja sobre 0,8KB/s y la red igual, alrededor de 1,81KB/s. Podemos concluir que el contenedor funciona de manera eficiente y no usa muchos recursos del sistema (es ligero).

Primero hemos de instalarlos de la forma

Resultado Htop



Resultado netcast

Proto	Recv-Q	Send-Q	Local Address	Foreign Address	State
tcp	0	0	ee7ce14df31d:40164	151.101.134.132:http	
TIME_WAIT					
tcp	0	0	ee7ce14df31d:46428	151.101.134.132:http	
TIME_WAIT					
tcp	0	0	ee7ce14df31d:40154	151.101.134.132:http	
TIME_WAIT					

Proto	RefCnt	Flags	Type	State	I-Node	Path
-------	--------	-------	------	-------	--------	------

30