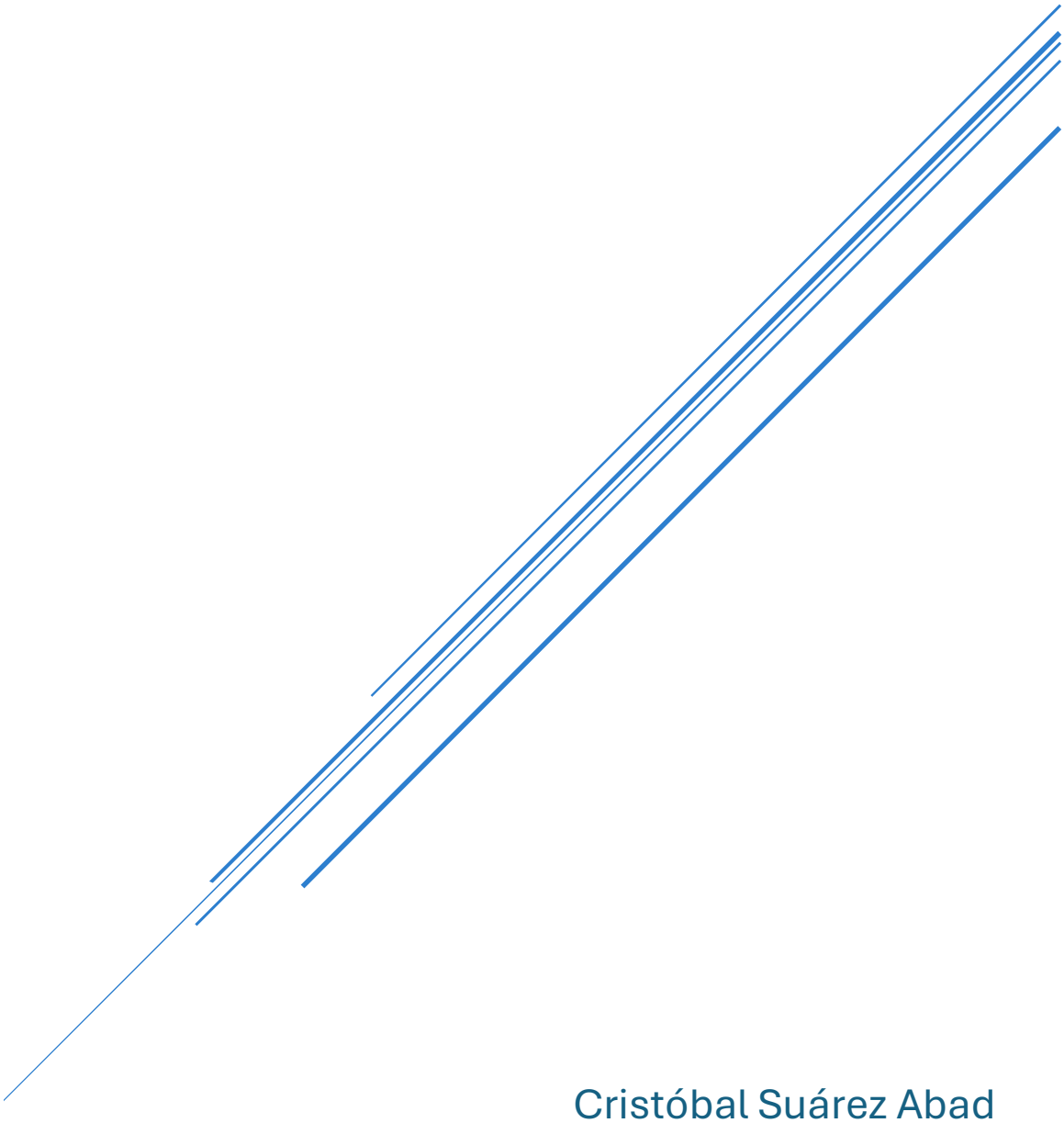
ACTIVIDAD 1

Docker: Instalación y ejemplos



Cristóbal Suárez Abad
Optativa – 2º ASIR

Índice:

1) Instala Docker.
2) Sigue los ejemplos:
    a) Creando un contenedor con un servidor web: https://plataforma.josedomingo.org/pledin/cursos/docker2024/contenido/modulo2/06_web.html
    b) Configuración de un contenedor con la imagen MariaDB: https://plataforma.josedomingo.org/pledin/cursos/docker2024/contenido/modulo2/07_mariadb.html
    c) Limitando los recursos utilizados por un contenedor Docker: https://plataforma.josedomingo.org/pledin/cursos/docker2024/contenido/modulo2/09_limite.html

1) **Instalación de Docker.**

Siguiendo la guía de https://josedom24.github.io/curso_docker_2022/sesion1/instalacion.html instalamos Docker: **apt install docker.io**

2) **Sigue los ejemplos:**

a) Creando un contenedor con un servidor web: *https://plataforma.josedomingo.org/pledin/cursos/docker2024/contenido/modulo2/06_web.html*

Ejecutamos: **docker run -d --name my-apache-app -p 8080:80 httpd:2.4**

Este comando crea y pone en funcionamiento un contenedor del servidor web Apache.

La opción "**-d**" hace que se ejecute en segundo plano (no mostrará todos los procesos en pantalla, solo los importantes).

"**--name**" establece un nombre para el contenedor, en vez de dejar que Docker le otorgue uno aleatorio.

"**-p 8080:80**": esto hace que se publique en el puerto 8080 del host, el 80 del contenedor (el 80 es el predeterminado del tráfico HTTP).

"httpd:2.4" es el nombre la imagen oficial de Apache.

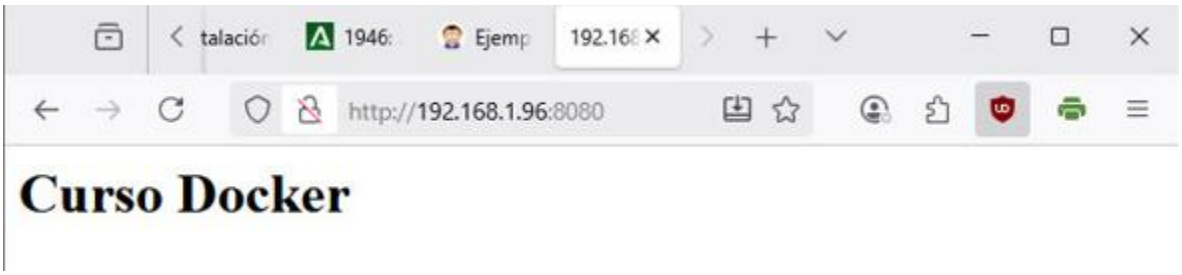
Comprobamos el mapeo de puertos: **docker port my-apache-app**

También podemos usar: **docker inspect --format='{{range $p, $conf := .NetworkSettings.Ports}} {{(index $conf 0).HostPort}} -> {{$p}} {{end}}' my-apache-app**

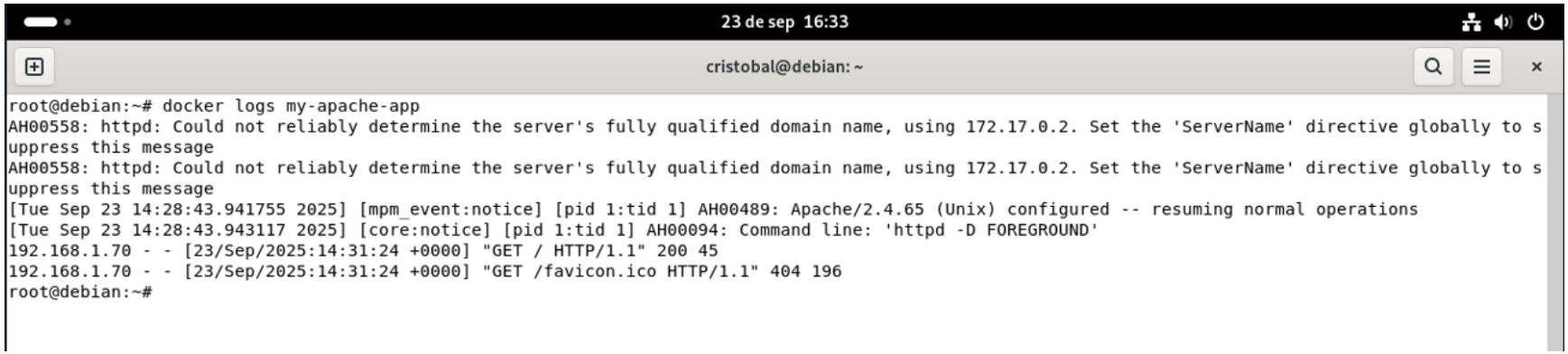
Ahora podemos comprobar desde un navegador web, que Apache está funcionando, ya sea desde el propio Host o desde otro equipo.

Desde el Host es: http://localhost:8080

Desde otro equipo: http://IP_DEL_HOST:8080



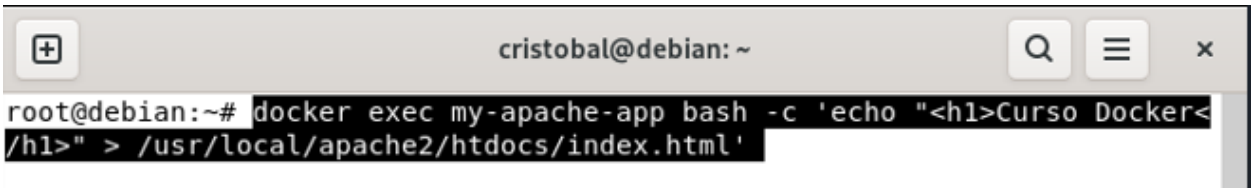
Ahora vamos a acceder al log del contenedor con: **docker logs (nombre del contenedor)**.



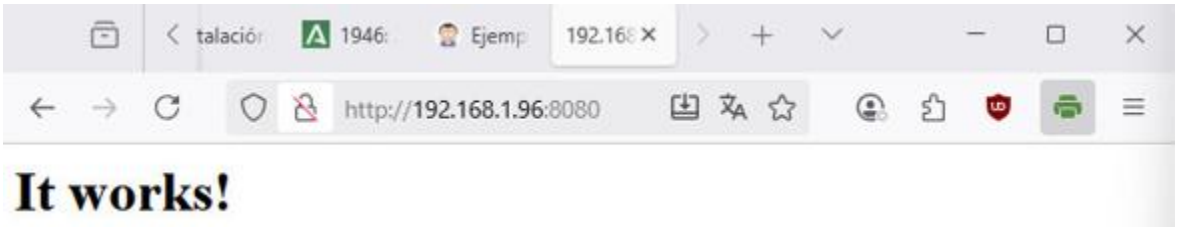
Ahora vamos a modificar el archivo “index.html” que se encuentra dentro del contenedor. Para ello hay varias maneras. La que hemos usado en esta ocasión es:

docker exec my-apache-app bash -c 'echo "<h1>Curso Docker</h1>" > /usr/local/apache2/htdocs/index.html'

Así podemos modificar el archivo en el interior del contenedor.



Y funciona:



Hay otras dos maneras:

- Accediendo al contenedor y modificando el archivo:

$ docker exec -it my-apache-app bash

root@cf3cd01a4993:/usr/local/apache2# cd /usr/local/apache2/htdocs/

root@cf3cd01a4993:/usr/local/apache2/htdocs# echo "<h1>Curso Docker</h1>" > index.html

root@cf3cd01a4993:/usr/local/apache2/htdocs# exit

- Creando un archivo nuevo fuera y copiándolo dentro con “docker cp”:

$ echo "<h1>Curso Docker</h1>" > index.html

$ docker cp index.html my-apache-app:/usr/local/apache2/htdocs/

b) *Configuración de un contenedor con la imagen MariaDB:*

https://plataforma.josedomingo.org/pledin/cursos/docker2024/contenido/modulo2/07_mariadb.html

Ahora vamos a probar variables de entorno durante la creación y puesta en marcha de contenedores. Para ello se puede consultar la documentación que existe de cada imagen, para saber que variables podemos usar. En este caso, la imagen "**mariadb**" nos permite establecer la contraseña del "**root**" con "**-e MARIADB_ROOT_PASSWORD=contraseñaquequeramos**"

Para comprobar las variables que se han creado, usamos: **docker exec -it mimariadb env**

Ahora para acceder podemos usar la variable. Primero entramos en el contenedor y vamos directamente a bash: **docker exec -it mimariadb bash**

“docker exec” ejecuta un comando dentro del contenedor.

“-it” lo abre en modo interactivo con terminal.

“mimariadb" nombre del contenedor.

“bash” el comando que se va a ejecutar.

Una vez dentro del contenedor usamos: **mysql -u root -p"$MARIADB_ROOT_PASSWORD"**


```
root@debian:~# docker exec -it mimariadb bash
root@4aa7eab17ef8:/# mysql -u root -p"$MARIADB_ROOT_PASSWORD"
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 3
Server version: 10.5.29-MariaDB-ubu2004 mariadb.org binary distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]>
```

Si queremos entrar directamente: docker exec -it mimariadb mysql -u root -p -h 127.0.0.1


```
root@debian:~# docker exec -it mimariadb mysql -u root -p -h 127.0.0.1
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 4
Server version: 10.5.29-MariaDB-ubu2004 mariadb.org binary distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]>
```

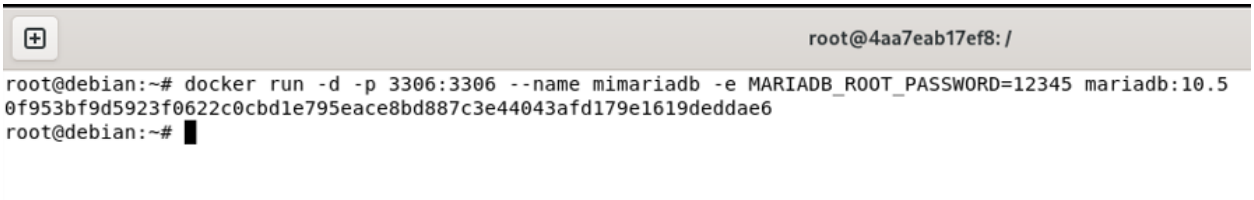
- *Accediendo a servidor de base de datos desde el exterior.*

Primero vamos a borrar el contenedor: **docker rm -f mimariadb**

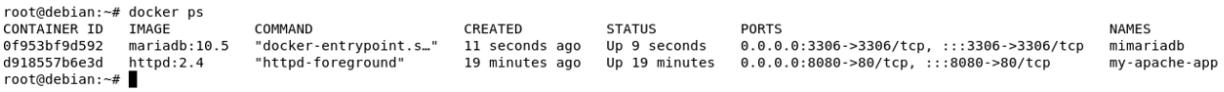
La opción "-f" permite borrarlo incluso si se está ejecutando.

Ahora vamos a crear otro contenedor, pero esta vez usando los puertos 3306 del Host y del contenedor.

docker run -d -p 3306:3306 --name mimariadb -e MARIADB_ROOT_PASSWORD=my-secret-pw mariadb:10.5



Comprobación de mapeado de puertos:



Ahora, desde el host, vamos a conectarnos: **mysql -u root -p -h 127.0.0.1**

Recuerda que el equipo cliente debe tener instalado "mariadb-client".

En este punto puede producirse un error debido a requisitos del SSL (Seguridad).Para solucionarlo hay que ejecutar el anterior comando de esta manera: **mysql -h 127.0.0.1 -P 3306 -u root -p --ssl=0**


```
root@debian:~# mysql -u root -p -h 127.0.0.1
Enter password:
ERROR 2026 (HY000): TLS/SSL error: SSL is required, but the server does not support it
root@debian:~# mysql -h 127.0.0.1 -P 3306 -u root -p --ssl=0
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 6
Server version: 10.5.29-MariaDB-ubu2004 mariadb.org binary distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]>
```

c) *Limitando los recursos utilizados por un contenedor Docker:*

https://plataforma.josedomingo.org/pledin/cursos/docker2024/contenido/modulo2/09_limite.html

- Limitando el uso de la CPU:

Primero vamos a aprender a limitar el uso de CPU. Se comprueba la cantidad de núcleos del Host: **nproc --all**

```
root@debian:~# nproc --all
2
root@debian:~#
```

ç

A la hora de crear un contenedor, podemos especificar límites de núcleos que puede usar. Para ello usamos la opción “--cpus".

docker run -d --cpus 1 --name servidor_web httpd:2.4

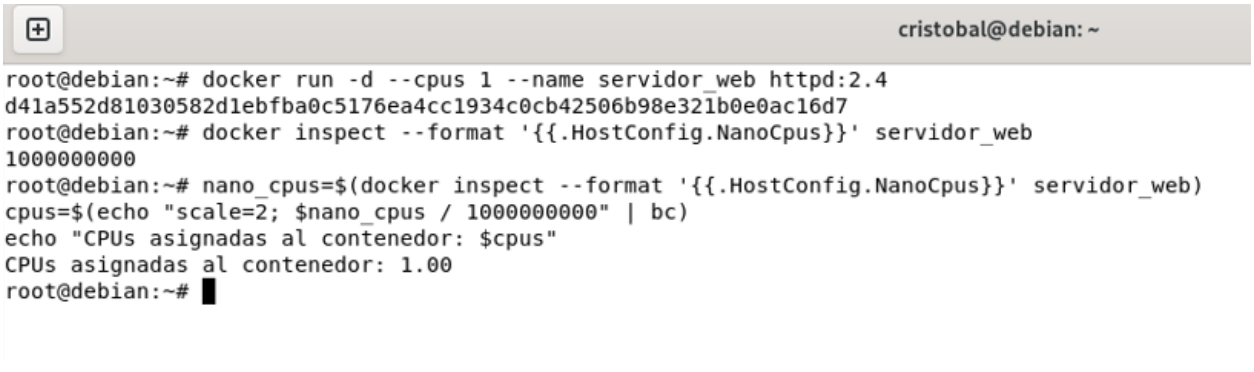
Podemos ver este límite con: **docker inspect --format '{{.HostConfig.NanoCpus}}' servidor_web**

Aunque nos dará un resultado un poco difícil de interpretar. Para visualizarlo mejor, usamos:

nano_cpus=$(docker inspect --format '{{.HostConfig.NanoCpus}}' servidor_web)

cpus=$(echo "scale=2; $nano_cpus / 1000000000" | bc)

echo "CPUs asignadas al contenedor: $cpus"

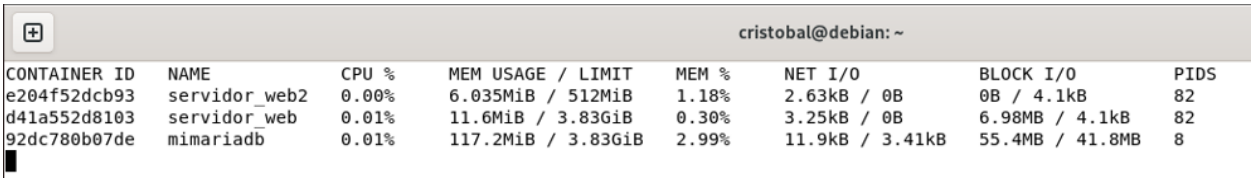


- Limitando el uso de memoria.

Para ello usamos la opción **“--memory**". En este caso hemos puesto el límite en 512 Megabytes.

docker run -d --memory 512m --name servidor_web httpd:2.4

Podemos comprobar los recursos que están consumiendo los contenedores con el comando: **docker stats**



De igual manera que con los anteriores comandos de comprobación de límites, el primer nos dará un valor en bruto y el segundo uno que es más fácil de entender:

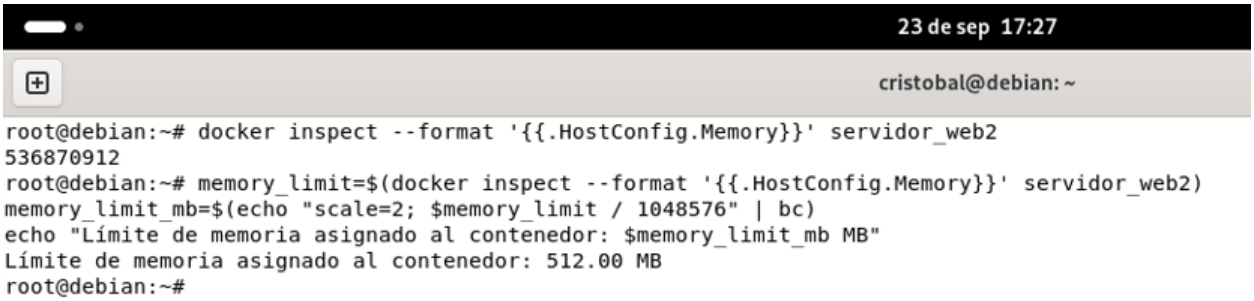
En bruto: **docker inspect --format '{{.HostConfig.Memory}}' servidor_web**

Elaborado:

memory_limit=$(docker inspect --format '{{.HostConfig.Memory}}' servidor_web)

memory_limit_mb=$(echo "scale=2; $memory_limit / 1048576" | bc)

echo "Límite de memoria asignado al contenedor: $memory_limit_mb MB"



- Modificando los límites en tiempo de ejecución.

Estos límites pueden ser modificados incluso durante la ejecución de los contenedores:

docker update --memory 1g --cpus 1.5 servidor_web