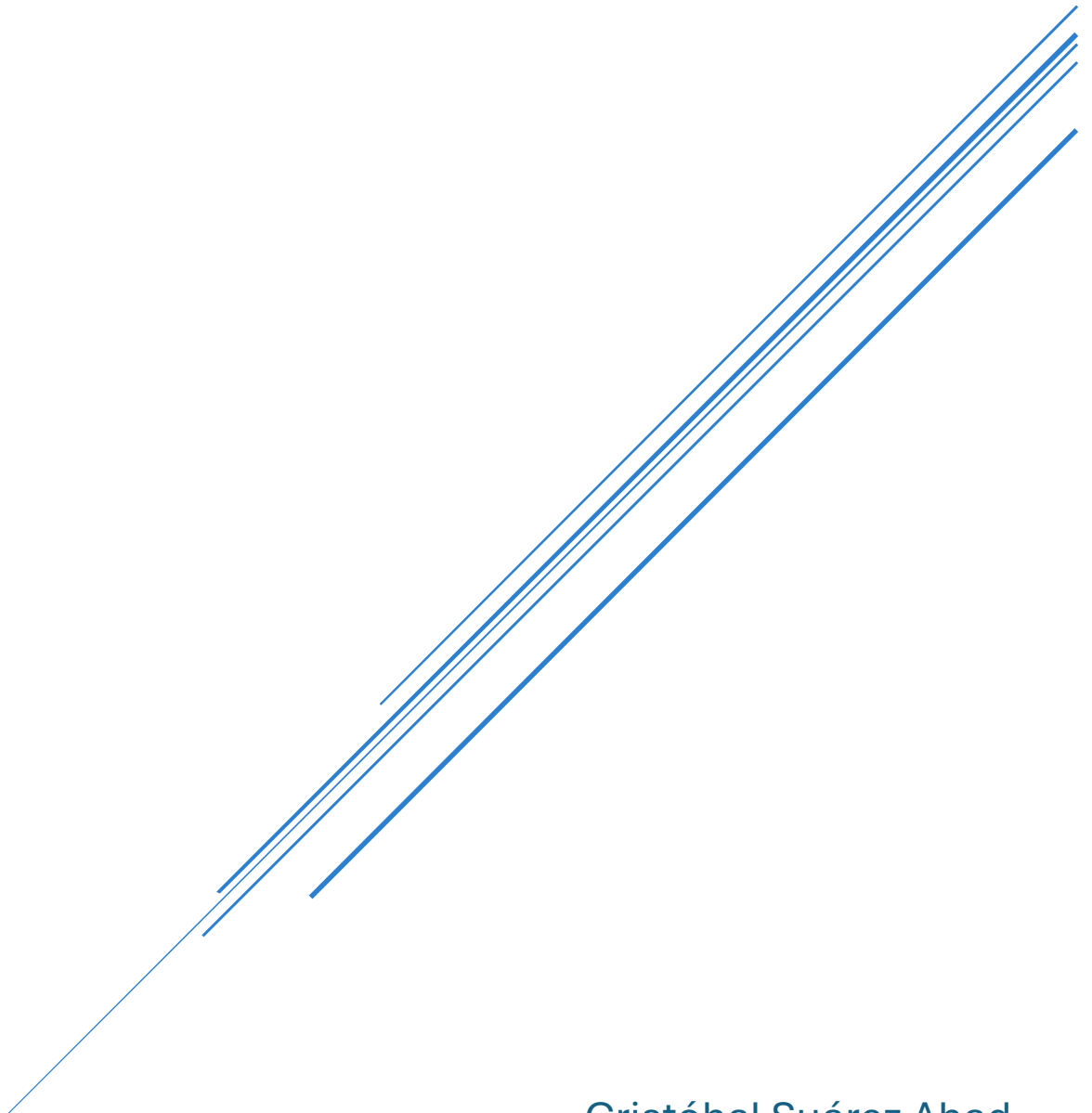


ACTIVIDAD 1

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**

```
23 de sep 16:24
cristobal@debian: ~
root@debian:~# apt install docker.io
Installing:
  docker.io

Installing dependencies:
  containerd          liberror-perl        libsort-naturally-perl
  criu                libintl-perl         libterm-readkey-perl
  docker-buildx       libintl-xs-perl      needrestart
  docker-cli          libip4tc2             patch
  git                 libip6tc2             python3-protobuf
  git-man             libmodule-find-perl  python3-pycriu
  iptables            libnet1               runc
  libcompell          libproc-processtable-perl tini

Paquetes sugeridos:
  containernetworking-plugins  xfsprogs          git-cvs
  docker-doc                   zfs-fuse          git-mediawiki
  aufs-tools                   | zfsutils-linux  git-svn
  btrfs-progs                  git-doc            firewallld
  cgroupfs-mount               git-email          needrestart-session
  debootstrap                  git-gui            | libnotify-bin
  rinse                         gitk                ed
  rootlesskit                  gitweb              diffutils-doc

Summary:
  Upgrading: 0, Installing: 25, Removing: 0, Not Upgrading: 103
  Download size: 93,4 MB
  Space needed: 392 MB / 110 GB available

Continue? [S/n] S
```

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.



```
cristobal@debian: ~  
root@debian:~# docker run -d --name my-apache-app -p 8080:80 httpd:2.4  
Unable to find image 'httpd:2.4' locally  
2.4: Pulling from library/httpd  
cel261c6d567: Pull complete  
aeb6d226161f: Pull complete  
4f4fb700ef54: Pull complete  
56926e6ce68f: Pull complete  
4938babf7b43: Pull complete  
307fcc49c641: Pull complete  
Digest: sha256:027c678f36d3cd3dd2b44ad1e963e81be66f9eba065381c1126d3019ff  
feb01a  
Status: Downloaded newer image for httpd:2.4  
d918557b6e3d8bd8453a735176ff15b598e0b6fcfc7764867a8f7278d3d0f5d0  
root@debian:~#
```

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



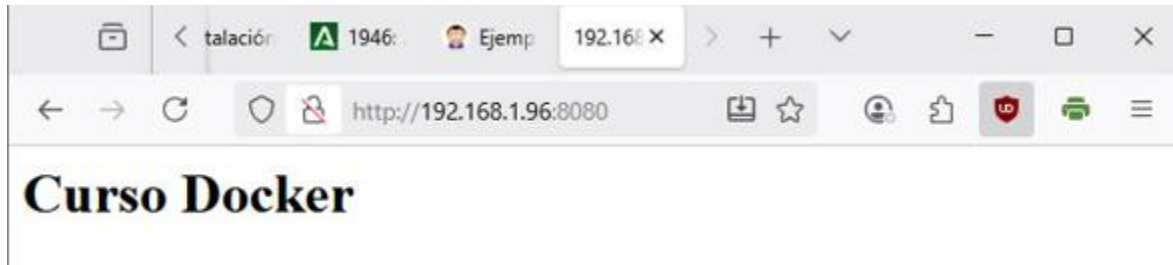
```
root@debian:~# docker port my-apache-app  
80/tcp -> 0.0.0.0:8080  
80/tcp -> [::]:8080  
root@debian:~#
```

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

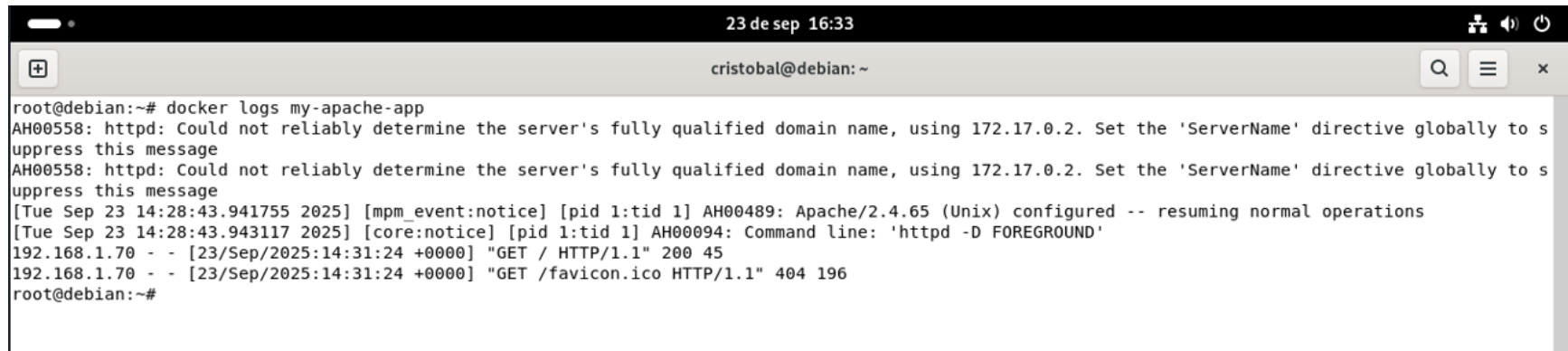
Ahora podemos comprobar desde un navegador web, que Apache está funcionando, ya sea desde el propio Host o desde otro equipo.

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)**.



A terminal window titled "23 de sep 16:33" with a user prompt "cristobal@debian: ~". The terminal shows the command "root@debian:~# docker logs my-apache-app" and its output. The output includes two warning messages from httpd about the server's fully qualified domain name, followed by a notice from the Apache mpm_event module, and two GET requests from 192.168.1.70.

```
root@debian:~# docker logs my-apache-app
AH00558: httpd: Could not reliably determine the server's fully qualified domain name, using 172.17.0.2. Set the 'ServerName' directive globally to suppress this message
AH00558: httpd: Could not reliably determine the server's fully qualified domain name, using 172.17.0.2. Set the 'ServerName' directive globally to suppress this message
[Tue Sep 23 14:28:43.941755 2025] [mpm_event:notice] [pid 1:tid 1] AH00489: Apache/2.4.65 (Unix) configured -- resuming normal operations
[Tue Sep 23 14:28:43.943117 2025] [core:notice] [pid 1:tid 1] AH00094: Command line: 'httpd -D FOREGROUND'
192.168.1.70 - - [23/Sep/2025:14:31:24 +0000] "GET / HTTP/1.1" 200 45
192.168.1.70 - - [23/Sep/2025:14:31:24 +0000] "GET /favicon.ico HTTP/1.1" 404 196
root@debian:~#
```

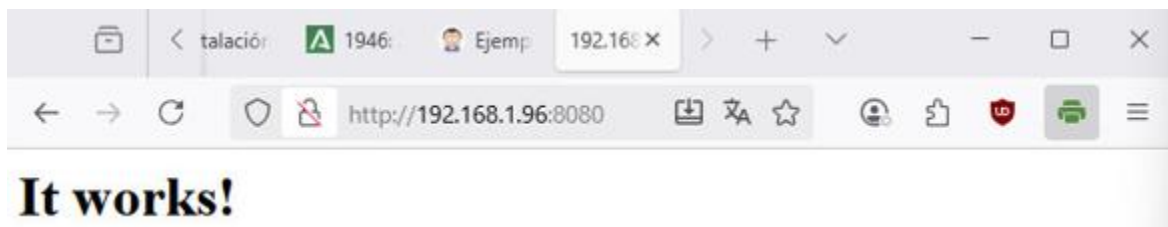
Ahora vamos a modificar el archivo “index.html” que se encuentra dentro del contenedor. Para ello hay varias maneras. La que hemos usado en esta ocasión es:

```
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.

A terminal window titled 'cristobal@debian: ~' showing the command: `root@debian:~# docker exec my-apache-app bash -c 'echo "<h1>Curso Docker</h1>" > /usr/local/apache2/htdocs/index.html'`

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”

```
root@debian:~# docker run -d --name mimariadb -e MARIADB_ROOT_PASSWORD=12345 mariadb:10.5
```

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

```
root@debian:~# docker exec -it mimariadb env
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
HOSTNAME=4aa7eab17ef8
TERM=xterm
MARIADB_ROOT_PASSWORD=12345
GOSU_VERSION=1.17
LANG=C.UTF-8
MARIADB_MAJOR=10.5
MARIADB_VERSION=1:10.5.29+maria~ubu2004
HOME=/root
root@debian:~#
```


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@4aa7eab17ef8:~# 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@4aa7eab17ef8:~# 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.

```
root@debian:~# docker rm -f mimariadb
mimariadb
root@debian:~#
```

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

```
root@4aa7eab17ef8: /
root@debian:~# docker run -d -p 3306:3306 --name mimariadb -e MARIADB_ROOT_PASSWORD=12345 mariadb:10.5
0f953bf9d5923f0622c0cbd1e795eace8bd887c3e44043afd179e1619deddae6
root@debian:~#
```

Comprobación de mapeado de puertos:

```
root@debian:~# docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED        STATUS        PORTS                               NAMES
0f953bf9d592   mariadb:10.5 "docker-entrypoint.s..." 11 seconds ago Up 9 seconds  0.0.0.0:3306->3306/tcp, :::3306->3306/tcp  mimariadb
d918557b6e3d   httpd:2.4  "httpd-foreground"       19 minutes ago Up 19 minutes  0.0.0.0:8080->80/tcp, :::8080->80/tcp  my-apache-app
```

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**



cristobal@debian: ~

```
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:~#
```

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
```

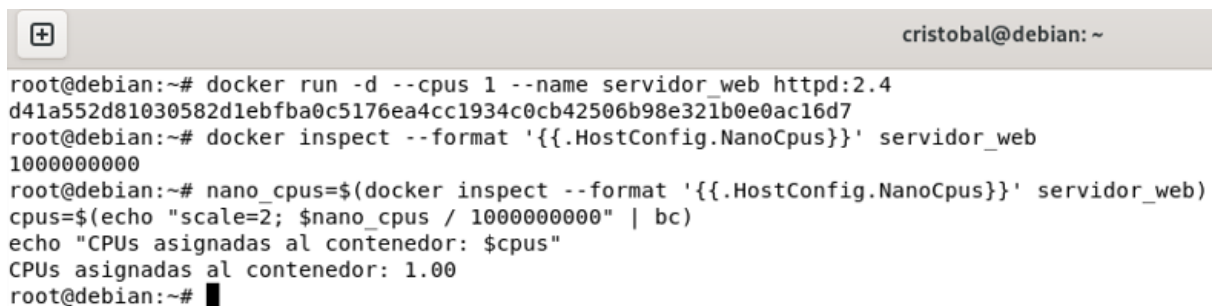
Podemos ver este límite con: **docker inspect --format '{{.HostConfig.NanoCpus}}' servidor_web**

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"
```



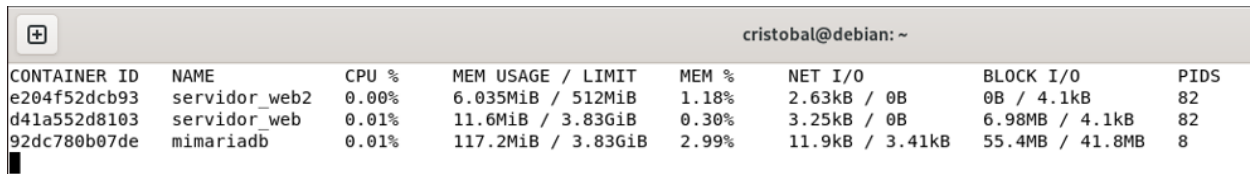
```
crisobal@debian: ~
root@debian:~# docker run -d --cpus 1 --name servidor_web httpd:2.4
d41a552d81030582d1ebfba0c5176ea4cc1934c0cb42506b98e321b0e0ac16d7
root@debian:~# docker inspect --format '{{.HostConfig.NanoCpus}}' servidor_web
1000000000
root@debian:~# nano_cpus=$(docker inspect --format '{{.HostConfig.NanoCpus}}' servidor_web)
cpus=$(echo "scale=2; $nano_cpus / 1000000000" | bc)
echo "CPUs asignadas al contenedor: $cpus"
CPUs asignadas al contenedor: 1.00
root@debian:~#
```

- 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**



CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM %	NET I/O	BLOCK I/O	PIDS
e204f52dcb93	servidor_web2	0.00%	6.035MiB / 512MiB	1.18%	2.63kB / 0B	0B / 4.1kB	82
d41a552d8103	servidor_web	0.01%	11.6MiB / 3.83GiB	0.30%	3.25kB / 0B	6.98MB / 4.1kB	82
92dc780b07de	mimariadb	0.01%	117.2MiB / 3.83GiB	2.99%	11.9kB / 3.41kB	55.4MB / 41.8MB	8

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"

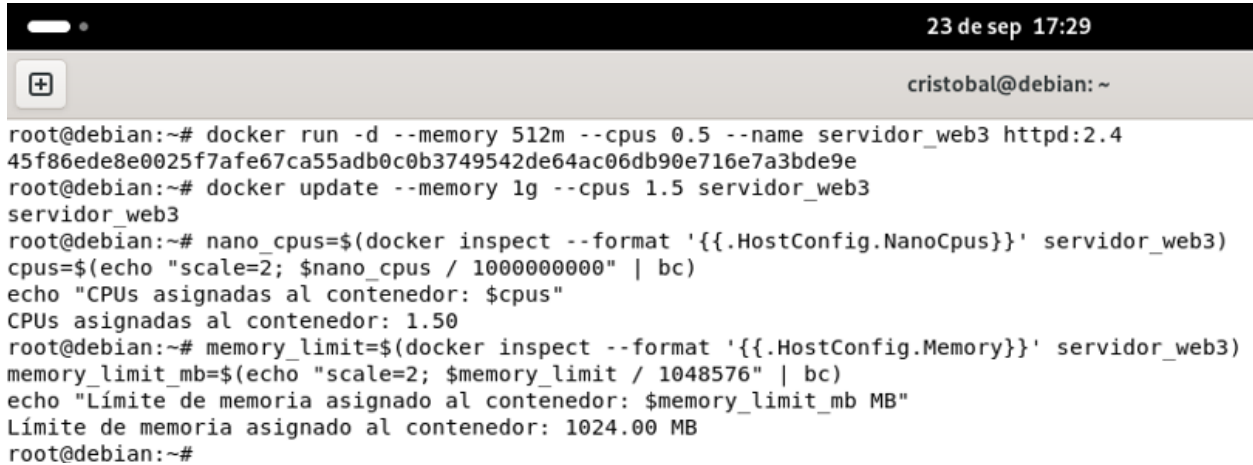


```
root@debian:~# docker inspect --format '{{.HostConfig.Memory}}' servidor_web2
536870912
root@debian:~# memory_limit=$(docker inspect --format '{{.HostConfig.Memory}}' servidor_web2)
memory_limit_mb=$(echo "scale=2; $memory_limit / 1048576" | bc)
echo "Límite de memoria asignado al contenedor: $memory_limit_mb MB"
Límite de memoria asignado al contenedor: 512.00 MB
root@debian:~#
```

- 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



A terminal window titled "23 de sep 17:29" with the user "cristobal@debian: ~". The terminal shows the following commands and output:

```
root@debian:~# docker run -d --memory 512m --cpus 0.5 --name servidor_web3 httpd:2.4
45f86ede8e0025f7afe67ca55adb0c0b3749542de64ac06db90e716e7a3bde9e
root@debian:~# docker update --memory 1g --cpus 1.5 servidor_web3
servidor_web3
root@debian:~# nano_cpus=$(docker inspect --format '{{.HostConfig.NanoCpus}}' servidor_web3)
cpus=$(echo "scale=2; $nano_cpus / 1000000000" | bc)
echo "CPUs asignadas al contenedor: $cpus"
CPUs asignadas al contenedor: 1.50
root@debian:~# memory_limit=$(docker inspect --format '{{.HostConfig.Memory}}' servidor_web3)
memory_limit_mb=$(echo "scale=2; $memory_limit / 1048576" | bc)
echo "Límite de memoria asignado al contenedor: $memory_limit_mb MB"
Límite de memoria asignado al contenedor: 1024.00 MB
root@debian:~#
```