

# Manual práctico de Docker

## Manual práctico de Docker

### ¿Qué es Docker?

1. Docker Images
  - 1.1. Imágenes oficiales
  - 1.2. Creando imágenes personalizadas
  - 1.3. Introducción a Dockerfile
  - 1.4. Aplicando lo aprendido
  - 1.5. Buenas prácticas
  - 1.6. Cambiando Dockerfile predeterminado
  - 1.7. Dangling Images
  - 1.8. Ejercicios
2. Docker Containers
  - 2.1. Listar/mapear puertos
  - 2.2. Iniciar/reiniciar/detener
  - 2.3. Variables de entorno
  - 2.4. Comprobar recursos usados
  - 2.5. Administración de usuarios
  - 2.6. Limitar recursos de un contenedor
  - 2.7. Copiar ficheros a un contenedor
  - 2.8. Convertir un contenedor en una imagen
  - 2.9. Ejercicios
3. Docker Volumes
  - 3.1. Host volumes
  - 3.2. Anonymous volumes
  - 3.3. Named volumes
  - 3.4. Dangling volumes
  - 3.5. Ejercicios
4. Docker Network
  - 4.1. Red bridge
  - 4.2. Crear y eliminar una red propia
  - 4.3. Cambiar la red de un contenedor
  - 4.4. Añadir nueva interfaz de red
  - 4.5. Asignar IP a un contenedor
  - 4.6. Red host
  - 4.7. Configurando IPs en servicios
  - 4.8. Ejecicios
5. Docker Registry
  - 5.1. Crear nuestro propio registry
  - 5.2. Subida y descarga de imágenes

# ¿Qué es Docker?

Docker es un software de código abierto que nos permite generar contenedores, encapsulando todo un entorno de trabajo de forma sencilla. De esta manera facilita el trabajo tanto a desarrolladores, asegurándoles un entorno de pruebas concreto, como a administradores de sistemas por su facilidad de administración y de despliegue.

Estos contenedores son más ligeros que una máquina virtual ya que no necesitamos esa capa de virtualización de hardware, sino que utiliza directamente los recursos del servidor de Docker. Adicionalmente aclarar que un contenedor no “virtualiza” un sistema operativo completo, sino solo aquellas partes que necesitamos para funcionar en nuestro entorno de trabajo ahorrando una cantidad de recursos considerable.

Otra ventaja a tener en cuenta de los contenedores es que son totalmente portables, pudiendo desplegarse en cualquier máquina con Docker instalada independientemente del sistema operativo que esté utilizando.

## Docker Images

Una imagen es una “plantilla” mediante la cual crearemos nuestros contenedores. Estas plantillas contendrán todos los elementos que necesitaremos para poder funcionar.

Un ejemplo sencillo sería un entorno con un servidor web donde tendremos los elementos básicos del sistema operativo, un servidor web Apache, la correspondiente configuración del mismo y una aplicación web que hayamos desarrollado

Las imágenes pueden ser las ya predefinidas con Docker en su repositorio público o podremos crear las nuestras personalizadas

### 1.1 Imágenes oficiales

Las imágenes oficiales, nos las proporciona directamente Docker y sin imágenes preconfiguradas con software específico como puede ser Apache, MongoDB, etc. Las podemos encontrar en:

<https://hub.docker.com>

Para descargarlas, podremos hacerlo desde nuestro Docker host con el comando:

```
docker pull nombre_imagen
```

Ejemplo:

```

dockeruser@vps382733:~$ docker pull httpd
Using default tag: latest
latest: Pulling from library/httpd
a5a6f2f73cd8: Pull complete
ac13924397e3: Pull complete
91b81769f14a: Pull complete
fec7170426de: Pull complete
992c7790d5f3: Pull complete
Digest: sha256:9753aabc6b0b8cd0a39733ec13b7aad59e51069ce96d63c6617746272752738e
Status: Downloaded newer image for httpd:latest

```

Si nos fijamos la primera línea nos muestra "Using default tag: latest" y esto nos está indicando que se ha descargado la última versión disponible de dicha imagen. Pero en ocasiones deberemos descargarnos otra versión diferente y para ello deberemos indicar en el propio comando la versión concreta a descargar:

*docker pull nombre:imagen\_version*

Ejemplo:

```

dockeruser@vps382733:~$ docker pull httpd:2.4.37-alpine
2.4.37-alpine: Pulling from library/httpd
4fe2ade4980c: Pull complete
42101a4e4c4e: Pull complete
73eadb9961ff: Pull complete
830983fb5ec2: Pull complete
78fb51bbfc27: Pull complete
Digest: sha256:b875793145fe613aa2d1f73f1cd8ec09b775abcd6980024bc60a6a9c73d644fc
Status: Downloaded newer image for httpd:2.4.37-alpine

```

Para poder saber cual es el tag concreto de cada versión, los podremos visualizar dentro de [hub.docker.com](https://hub.docker.com). Concretamente dentro de la sección correspondiente a la imagen que nos queramos descargar:

httpd ☆  
Last pushed: 7 days ago

Repo Info Tags

Short Description  
The Apache HTTP Server Project

Full Description  
Supported tags and respective Dockerfile links

- 2.4.37, 2.4, 2, latest ([2.4/Dockerfile](#))
- 2.4.37-alpine, 2.4-alpine, 2-alpine, alpine ([2.4/alpine/Dockerfile](#))

Docker Pull Command  
docker pull httpd

Una vez descargados, podremos ver las versiones disponibles:

```
dockeruser@vps382733:~$ docker images | grep httpd
httpd                2.4.37-alpine        11fc0c2a2dfa        9 days ago
91.8MB
httpd                latest               2a51bb06dc8b        9 days ago
132MB
```

En ocasiones, cuando descarguemos una imagen, veremos que nos indicará que algunas capas ya existen (Already exist) y esto es debido a que si Docker detecta una capa similar en alguna de las versiones que ya tengamos de un software, simplemente la copiará del local y no la descargará.

Recordemos que cuando descargábamos una imagen, se le asociaba un tag y en el caso de las últimas versiones disponibles se descargaba con el tag "latest". Puede suceder que pasen unos días, saquen una nueva versión de una imagen y queramos descargar de nuevo la última versión sacada. ¿Qué sucedería? pues que esta última versión también se descargaría con el tag "latest" y la anterior se quedaría sin ningún tag asociado (<none>).

Si queremos eliminar una imagen, simplemente tendremos que utilizar el siguiente comando:

```
docker rmi nombre_imagen
```

Si queremos eliminar una versión concreta:

```
docker rmi nombre_imagen:version
```

## 1.2 Creando imágenes personalizadas

Para crear una imagen deberemos crear un fichero de texto normal con el nombre "Dockerfile", en la carpeta donde queramos almacenar nuestras imágenes personalizadas.

A continuación se indican las instrucciones que deben componer dichas imágenes y el orden correcto en el que deben estar. En el ejemplo crearemos una imagen con Apache y PHP:

1ª Capa FROM: sistema operativo instalado en nuestra imagen. Si el SO ya dispone de imagen oficial, podremos hacer una llamada a dicha imagen en los repositorios de Docker.

2ª Capa RUN: en esta capa indicaremos los comandos que utilizaremos para poder instalar nuestro software, en este caso Apache y PHP.

```
Archivo Editar Ver Buscar Terminal Ayuda
GNU nano 2.9.3 Dockerfile Modified
FROM centos:latest
RUN yum install httpd
```

Una vez tengamos el fichero creado, podremos crear una imagen a partir de nuestro Dockerfile con el comando:

*docker build -t nombre\_imagen .*

También podremos crear tags con diferentes versiones:

*docker build -t nombre\_imagen:version .*

\*-t indica el nombre de la imagen resultante. Y puede utilizarse también "--tag".

\*Ojo, por que el punto al final del comando debe escribirse también.

Si hemos seguido los pasos indicados, al crear la imagen veremos que nos devuelve el siguiente error:

```
Transaction Summary
=====
Install 1 Package (+5 Dependent packages)

Total download size: 24 M
Installed size: 31 M
Is this ok [y/d/N]: Exiting on user command
Your transaction was saved, rerun it with:
yum load-transaction /tmp/yum_save_tx.2018-11-25.16-03.I_1WTM.yumtx
The command '/bin/sh -c yum install httpd' returned a non-zero code: 1
```

Esto es debido a que cuando se crea una imagen no podemos tener interacción con el proceso de creación, por lo que hay que especificar también aquellos pasos donde se necesite interacción por parte del usuario. Como por ejemplo el típico paso de introducir "y" o "n" para confirmar una instalación.

En este caso se solucionaría añadiendo un "-y" al comando de instalación de Apache, quedando de la siguiente manera:

```
Archivo Editar Ver Buscar Terminal Ayuda
GNU nano 2.9.3

FROM centos:latest

RUN yum install httpd -y
```

Y ahora si, se creará sin problema:

```
Complete!
Removing intermediate container a7f074690674
--> dc244580a990
Successfully built dc244580a990
Successfully tagged centos-apache:latest
```

\*Habrá instalaciones en las que necesitemos establecer otros parámetros como puede ser una zona geográfica (en una instalación de PHP por ejemplo). Esto se solucionaría añadiendo a nuestro Dockerfile una variable de entorno. Ejemplo:

```
ENV TZ="Europe/Madrid"
RUN ln -snf /usr/share/zoneinfo/$TZ /etc/localtime && echo $TZ > /etc/timezone
```

Podemos ver más detalles sobre el proceso de creación con:

```
docker history -H nombre_imagen
```

\*-H devuelve el resultado del comando de forma legible

```
dockeruser@vps382733:~/docker-images$ docker history -H centos-apache
```

IMAGE	CREATED	CREATED BY	SIZE	COMMENT
dc244580a990	8 minutes ago	/bin/sh -c yum install httpd -y	132MB	
75835a67d134	6 weeks ago	/bin/sh -c #(nop) CMD ["/bin/bash"]	0B	
<missing>	6 weeks ago	/bin/sh -c #(nop) LABEL org.label-schema.sc...	0B	
<missing>	6 weeks ago	/bin/sh -c #(nop) ADD file:fbe9badfd2790f074...	200MB	

Si ahora ejecutamos el contenedor que hemos creado con el comando:

```
docker run -d centos-apache
```

Y acto seguido ejecutamos un comando:

```
docker ps -a
```

Vamos a ver que el contenedor no se está ejecutando:



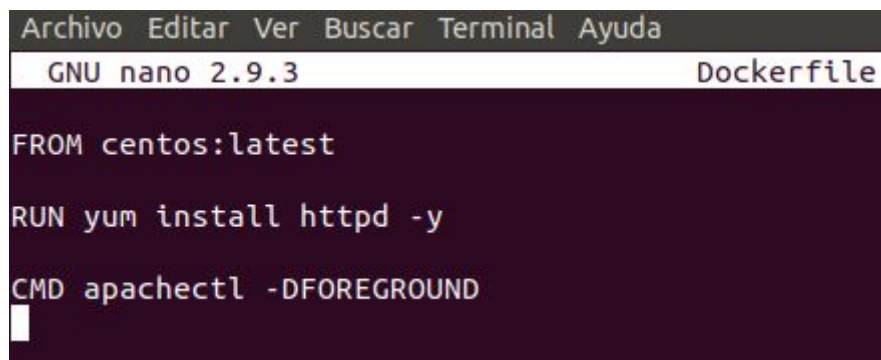
```

dockeruser@vps382733:~$ docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
98021faa114a       centos-apache      "/bin/bash"        About a minute ago  Exited (0)          About a minute ago  adoring_bell

```

Y eso es debido a que en el fichero Dockerfile nos falta una última capa.

3ª capa CMD: En dicha capa le diremos que ejecute el servicio Apache en primer plano. Podemos encontrar los comandos correspondientes a cada servicio buscando “nombre\_servicio run in foreground” en cualquier buscador. Por lo que en el caso de Apache quedaría de la siguiente manera:



```

Archivo Editar Ver Buscar Terminal Ayuda
GNU nano 2.9.3 Dockerfile
FROM centos:latest
RUN yum install httpd -y
CMD apachectl -DFOREGROUND

```

Una vez guardados los cambios si volvemos a ejecutar la imagen no nos va a funcionar. Recordemos que las imágenes son de solo lectura y que no se pueden modificar por lo que deberemos crearla de nuevo:

*docker build -t apache-centos:apache-cmd .*

```

dockeruser@vps382733:~/docker-images$ docker build -t apache-centos:apache-cmd .
Sending build context to Docker daemon 2.048kB
Step 1/3 : FROM centos:latest
----> 75835a67d134
Step 2/3 : RUN yum install httpd -y
----> Using cache
----> dc244580a990
Step 3/3 : CMD apachectl -DFOREGROUND
----> Running in 94c54b286942
Removing intermediate container 94c54b286942
----> 15a0de679520
Successfully built 15a0de679520
Successfully tagged apache-centos:apache-cmd

```

Y ahora si, si creamos el contenedor y comprobamos si está funcionando veremos que está UP:

```

dockeruser@vps382733:~/docker-images$ docker run -d --name apache apache-centos:apache-cmd
80b8a029b59c6a1c31396a33c72cbe550f8af020b48c650bb0494bdb42f94812
dockeruser@vps382733:~/docker-images$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
80b8a029b59c       apache-centos:apache-cmd  "/bin/sh -c 'apachec..."  5 seconds ago      Up 4 seconds              apache

```



Lo siguiente correspondería a la sección de contenedores, pero vamos a ir avanzándolo. Si ahora mismo accedemos por nuestro navegador a la ip del servidor donde tengamos el contenedor levantado debería mostrarse la típica página de Apache recién instalado, pero ahora mismo no funcionaría por que nos falta levantar el contenedor con un parámetro concreto:

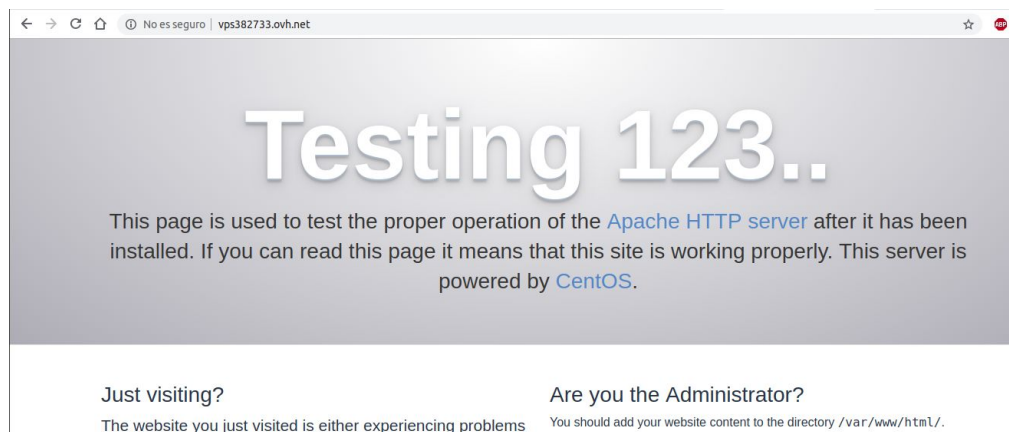
```
docker run -d --name apache2 -p 80:80 apache-centos:apache-cmd
```

El parámetro “-p” permite que el contenedor sea accesible a través de dicho puerto en nuestra máquina. Por lo que volvemos a lanzar un contenedor:

```
dockeruser@vps382733:~/docker-images$ docker run -d --name apache -p 80:80 apache-centos:apache-cmd
52e55ef67323793be8eb4c5a6a7b655d32fbfc02067155208f5484e923b89c5d
dockeruser@vps382733:~/docker-images$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
52e55ef67323	apache-centos:apache-cmd	"/bin/sh -c 'apachec..."	3 seconds ago	Up 2 seconds	0.0.0.0:80->80/tcp	apache

Y ahora si, si accedemos a través de nuestro navegador al servidor VPS podremos ver que Apache está funcionando:



## 1.3 Introducción a Dockerfile

Dockerfile es el fichero donde definimos la configuración de nuestras imágenes y está compuesto por las siguientes capas:

**FROM :** especificamos el sistema operativo o incluso una imagen como ya hemos hecho anteriormente con la oficial de centos.

**RUN:** instrucciones que vamos a ejecutar en la terminal, como por ejemplo una instalación de Apache. También podemos hacer acciones como agregar usuarios, crear directorios, archivos, etc.

**COPY:** copia archivos de nuestra máquina hacia la imagen, siempre que se encuentren en la misma carpeta que el fichero Dockerfile. Esta opción la usaremos cuando tengamos que modificar ficheros de configuración de un servicio (por ejemplo al modificar el virtualhost de un Apache para que funcione con un certificado SSL). Ejemplo:

Hemos creado una carpeta llamada "prueba" y dentro hemos introducido un fichero index.html sencillo con un hola mundo. Para crear una nueva imagen añadiendo "copy" quedaría de la siguiente manera:

```
Archivo Editar Ver Buscar Terminal Ayuda
GNU nano 2.9.3

FROM centos:7

RUN yum install httpd -y

COPY prueba /var/www/html

CMD apachectl -DFOREGROUND
```

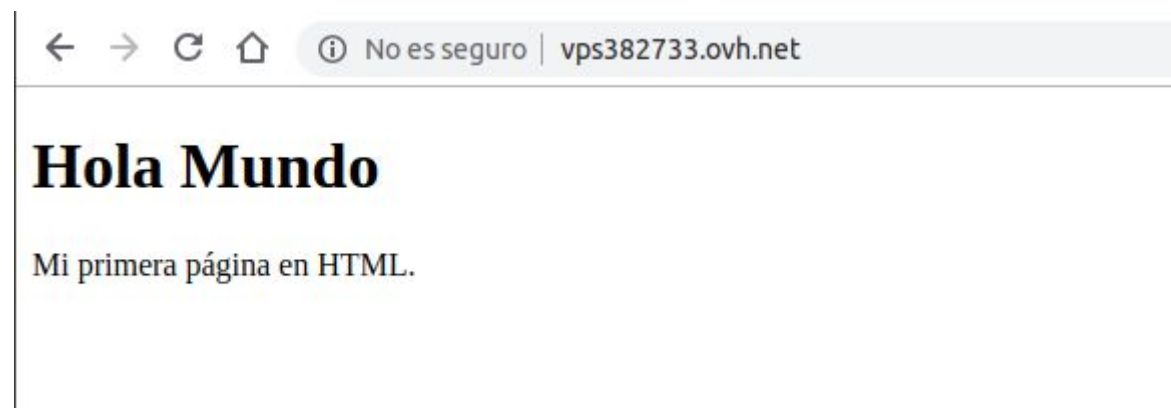
Como podemos observar, le hemos indicado la carpeta a copiar y la ruta "/var/www/html". Esta ruta, es donde se guardan los sitios webs en Apache. Generamos la nueva imagen:

```
dockeruser@vps382733:~/docker-images$ docker build -t apache .
```

Y creamos nuestro contenedor:

```
dockeruser@vps382733:~/docker-images$ docker run -d -p 80:80 apache
```

Y ahora al acceder a nuestro navegador, podremos visualizar el contenido del fichero html:



**ADD:** Esta instrucción es similar a COPY, con la diferencia de que si por ejemplo especificamos un fichero comprimido al ejecutarse el guión lo descomprimirá de manera automática. También se puede añadir una URL, que descargará los ficheros que contenga para añadirlos al destino que le indiquemos.

ENV: permite añadir variables de entorno a nuestro fichero Dockerfile y poder configurar así los diferentes servicios. Ejemplo:

La instrucción ENV estará compuesto por dos valores. El primero será el nombre de la variable y el segundo el valor. En nuestro caso "contenido" es la variable y "prueba" es el valor de dicha variable.

Para poder ejecutar dicha variable deberemos utilizar una instrucción RUN y en este caso lo que haremos será meter el valor de la variable contenido en un fichero html dentro del contenedor.

```
Archivo Editar Ver Buscar Terminal Ayuda
GNU nano 2.9.3 Dockerfile
FROM centos:7
RUN yum install httpd -y
COPY prueba /var/www/html
ENV contenido prueba
RUN echo "$contenido" > /var/www/html/prueba.html
CMD apachectl -DFOREGROUND
```

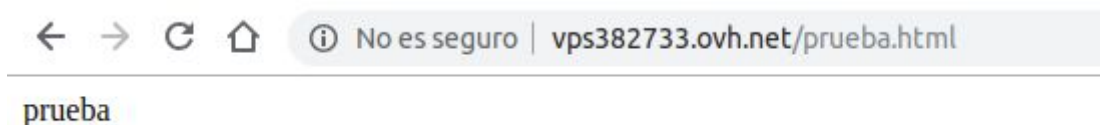
Creamos la imagen de nuevo:

```
dockeruser@vps382733:~/docker-images$ docker build -t apache .
```

Construimos el contenedor:

```
dockeruser@vps382733:~/docker-images$ docker run -d -p 80:80 apache
```

Y si accedemos a la URL de nuestro servidor seguido del fichero "prueba.html" podremos ver como se ha ejecutado correctamente:



The screenshot shows a web browser window with the address bar displaying "vps382733.ovh.net/prueba.html". The page content is the word "prueba".

WORKDIR: esta instrucción nos permitirá decirle a Docker, desde que directorio queremos partir a la hora de realizar una instrucción posterior. Ejemplo:

Hemos modificado el fichero Dockerfile añadiendo:

1º- WORKDIR /var/www/html

2º- COPY prueba .

```
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
GNU nano 2.9.3 Dockerfile

FROM centos:7

RUN yum install httpd -y

WORKDIR /var/www/html

COPY prueba .

ENV contenido prueba

RUN echo "$contenido" > /var/www/html/prueba.html

CMD apachectl -DFOREGROUND
```

Primero le hemos dicho la ruta del sistema de ficheros donde nos encontramos y después con COPY, indicamos el directorio a copiar. Si os fijáis, al final hemos añadido un punto y que va a hacer referencia al directorio donde nos encontramos.

De igual manera, podríamos hacer la copia del fichero desde la carpeta "www" de la siguiente manera:

```
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
GNU nano 2.9.3 Dockerfile

FROM centos:7

RUN yum install httpd -y

WORKDIR /var/www

COPY prueba html/

ENV contenido prueba

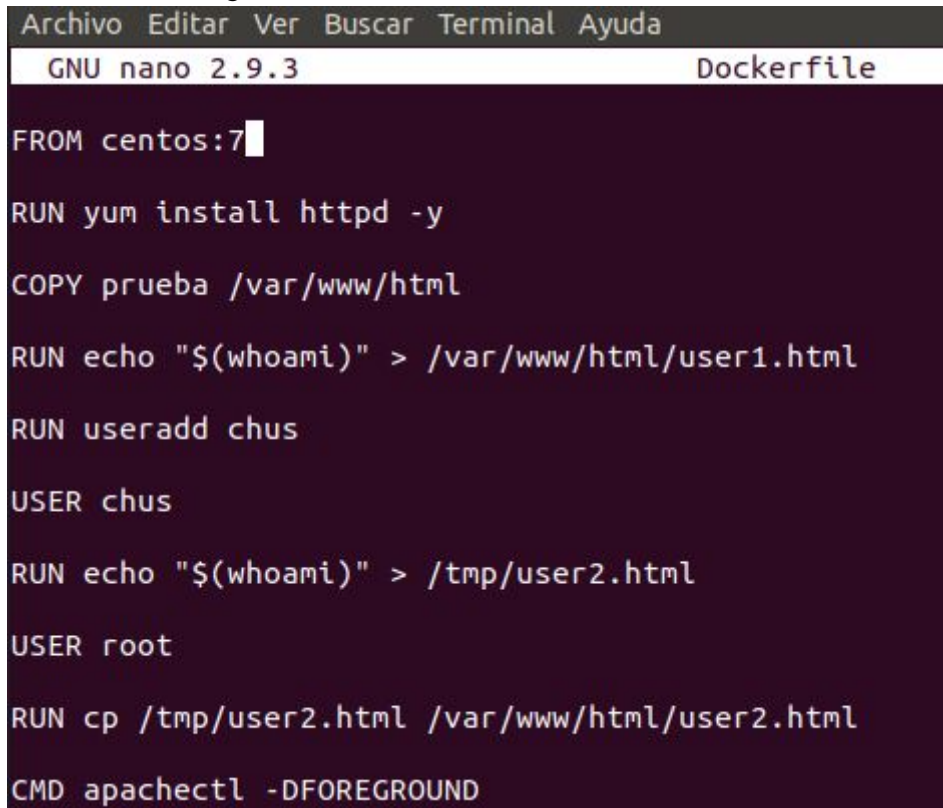
RUN echo "$contenido" > /var/www/html/prueba.html

CMD apachectl -DFOREGROUND
```

**EXPOSE:** permite modificar el puerto por el que escucha el contenedor. Por ejemplo, que un servicio Apache en vez de por el puerto 80, sea accesible a través del 81. Como es una instrucción más ligada a contenedores que a la construcción de imágenes lo veremos más adelante.

**USER:** permite modificar el usuario que ejecutará las siguientes capas de la imagen. Es importante que previamente lo hayamos creado previamente. Ejemplo:

Observad el siguiente Dockerfile:



```
FROM centos:7
RUN yum install httpd -y
COPY prueba /var/www/html
RUN echo "$(whoami)" > /var/www/html/user1.html
RUN useradd chus
USER chus
RUN echo "$(whoami)" > /tmp/user2.html
USER root
RUN cp /tmp/user2.html /var/www/html/user2.html
CMD apachectl -DFOREGROUND
```

Vamos a utilizar el comando "whoami" de Linux para poder mostrar como sería el funcionamiento de USER:

1-Vamos a crear el servidor Apache y con COPY vamos a mover el contenido de prueba a la carpeta por defecto del servidor web.

2-A continuación vamos a decirle que nos meta en una variable el contenido del comando "whoami" en el fichero "user1.html" dentro de la carpeta por defecto de Apache.  
Este fichero contendrá el usuario "root" que es el que por defecto ejecuta todas las instrucciones.

3-Después vamos a crear el usuario "chus", vamos a iniciar sesión con el y vamos a hacer que haga lo mismo que "root" pero con el fichero "user2.html" dentro del directorio "/tmp". Le he indicado que lo cree dentro de "/tmp" debido a que por defecto el usuario creado, no tendrá permisos sobre la

carpeta por defecto de Apache asique para simplificar el ejemplo vamos a hacerlo tal y como está.

4-Por último, volvemos a iniciar sesión con el usuario "root" y movemos el contenido de "/tmp" a la carpeta por defecto de Apache.

5-Creamos la imagen de nuevo:

```
dockeruser@vps382733:~/docker-images$ docker build -t apache .
```

6-Construimos el contenedor:

```
dockeruser@vps382733:~/docker-images$ docker run -d -p 80:80 apache
```

7-Y si accedemos a las URL:



Podremos ver el contenido de los ficheros y como cambia el contenido en función del usuario que lo ha ejecutado.

**VOLUME:** nos permite utilizar en el contenedor una ubicación de nuestro host para poder almacenar datos de manera permanente y el volumen creado, será un volumen anónimo (más adelante veremos con mayor detalle qué significa). Los volúmenes de los contenedores siempre son accesibles en el host en la ubicación "/var/lib/docker/volumes". Esta instrucción la veremos más adelante con mayor detalle pero os pongo un ejemplo de cómo se declararía en nuestro Dockerfile:



```
Archivo Editar Ver Buscar Terminal Ayuda
GNU nano 2.9.3

FROM centos:7

RUN yum install httpd -y

COPY prueba /var/www/html

RUN echo "$(whoami)" > /var/www/html/user1.html

RUN useradd chus

USER chus

RUN echo "$(whoami)" > /tmp/user2.html

VOLUME /var/www/html

USER root

RUN cp /tmp/user2.html /var/www/html/user2.html

CMD apachectl -DFOREGROUND
```

Simplemente habrá que indicarle a VOLUME la ruta de la carpeta que queremos guardar, para que al borrar el contenedor el contenido no se pierda. Más adelante lo veremos con mayor profundidad.

**CMD:** con esta instrucción indicaremos que proceso debe ejecutar el contenedor y debe ser única. En el caso de contener varias la última es la que predominará. Hasta ahora hemos estado utilizando comandos concretos para mantener Apache levantado, pero también se pueden utilizar script. Ejemplo:

Primero vamos a crear un script que se llama run.sh para iniciar el servicio de Apache:

```
Archivo Editar Ver Buscar Terminal Ayuda
GNU nano 2.9.3 run.sh

#!/bin/bash
echo "Iniciando contenedor..."
apachectl -DFOREGROUND
```

Ahora, le diremos a nuestro fichero DockerFile que tome dicho fichero para ejecutar el contenedor. En nuestro caso lo hemos guardado en la misma carpeta donde tenemos el fichero Dockerfile.

Y modificaremos nuestro fichero añadiendo un COPY para copiar el script en nuestro contenedor y a continuación, en el CMD, le diremos que ejecute dicho script.

```
Archivo Editar Ver Buscar Terminal Ayuda
GNU nano 2.9.3

FROM centos:7

RUN yum install httpd -y

COPY prueba /var/www/html

RUN echo "$(whoami)" > /var/www/html/user1.html

RUN useradd chus

USER chus

RUN echo "$(whoami)" > /tmp/user2.html

VOLUME /var/www/html

USER root

RUN cp /tmp/user2.html /var/www/html/user2.html

COPY run.sh /run.sh

CMD sh /run.sh
```

Creamos la imagen de nuevo:

```
dockeruser@vps382733:~/docker-images$ docker build -t apache .
```

Construimos el contenedor:

```
dockeruser@vps382733:~/docker-images$ docker run -d -p 80:80 apache
```

Y ahora podremos observar al comprobar el estado de los contenedores que el apartado COMMAND nos indica que se está ejecutando a partir de dicho script:

```
dockeruser@vps382733:~/docker-images$ docker ps --no-trunc
CONTAINER ID        NAMES               IMAGE               COMMAND             CREATED             STATUS
5b87ede24b73ded8f4b1ee7ae796e90e5ecf44b52a499c7283d1160f5fd99c43    heuristic_brattain  apache             "/bin/sh -c 'sh /run.sh'"  24 seconds ago     Up 22 seconds
```

Y si comprobamos los logs, veremos los pasos hasta iniciarse:

```
dockeruser@vps382733:~/docker-images$ docker logs -f heuristic_brattain
Iniciando contenedor...
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 thi
s message
```

**.dockerignore:** este fichero se encuentra oculto y lo creamos para ignorar ficheros que se encuentren dentro del directorio actual donde esté nuestro Dockerfile. Por defecto, cuando creamos una imagen el constructor tomará todos los ficheros que haya en el directorio. Añadiendo los ficheros que no queramos tener en cuenta a este fichero se ignorarán con el correspondiente ahorro de espacio en la imagen.

## 1.4 Aplicando lo aprendido

A continuación os expongo un ejemplo utilizando lo aprendido hasta ahora:

```
Archivo Editar Ver Buscar Terminal Ayuda
GNU nano 2.9.3

FROM nginx

RUN useradd chus

COPY prueba /usr/share/nginx/html

ENV archivo docker

WORKDIR /usr/share/nginx/html

RUN echo "$archivo" > /usr/share/nginx/html/env.html

EXPOSE 90

LABEL version=1

USER chus

RUN echo "Yo soy $(whoami)" > /tmp/yo.html

USER root

RUN cp /tmp/yo.html /usr/share/nginx/html/docker.html

VOLUME /var/log/nginx

CMD nginx -g 'daemon off;'
```

Utilizaremos la imagen oficial de Nginx del hub de Docker donde podemos obtener más información sobre la misma (como el directorio principal del servicio) en la URL de la imagen:



How to use this image

## Hosting some simple static content

```
$ docker run --name some-nginx -v /some/content:/usr/share/nginx/html:ro -d nginx
```

Alternatively, a simple `Dockerfile` can be used to generate a new image that includes the necessary content (which is a much cleaner solution than the bind mount above):

```
FROM nginx
```

```
COPY static-html-directory /usr/share/nginx/html
```

Place this file in the same directory as your directory of content ("static-html-directory"), run `docker build -t some-content-nginx .`, then start your container:

```
$ docker run --name some-nginx -d some-content-nginx
```

En la imagen creada pasamos el contenido de la carpeta prueba al directorio principal de Nginx y creamos la variable de entorno "archivo" con el valor "docker", que una vez nos hemos posicionado en el directorio principal enviamos su contenido al fichero "env.html".

Cambiamos el puerto al 90, le asignamos una etiqueta con la versión y con el usuario "chus" enviamos la información de la sesión a la carpeta /tmp.

## 1.5 Buenas prácticas

- Servicio efímero.
- Un servicio por contenedor
- Si tenemos archivos pesados en el directorio de Dockerfile, añadirlos al fichero .dockerignore.
- Optimizar el uso de capas y usar las menos posibles.
- Simplicidad en los argumentos.
- No instalar paquetes innecesarios.
- Utilizar labels para aplicar los metadatos (versiones, etc).

## 1.6 Cambiando Dockerfile predeterminado

En ocasiones queremos tener diferentes Dockerfiles creados con diferentes nombres. Para construir una imagen en base a un fichero Dockerfile por defecto deberemos añadir el parámetro "-f" al comando:

```
docker build -t nombre_imagen -f nombre_fichero .
```

## 1.7 Dangling Images

Según vayamos creando imágenes en base a un mismo Dockerfile al sacar un listado de las que tengamos creadas podremos ver que tenemos algunas con el tag "<none>". Estas imágenes son las que se han quedado "huerfanas" por que hemos vuelto a crear una nueva. Y podemos listarlas con el siguiente comando:

```
docker images -f dangling=true
```

```
dockeruser@vps382733:~/docker-images$ docker images -f dangling=true
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
<none>              <none>             c769621fddf3       About an hour ago
206MB
<none>              <none>             1db3ee46ec78       4 hours ago
201MB
<none>              <none>             907de531a878       4 hours ago
201MB
```

Lo normal es que no vuelvan a ser utilizadas por lo que podemos eliminarlas todas con el siguiente comando:

```
docker images -f dangling=true -q | xargs docker rmi
```

## 1.8 Ejercicios

**Ejercicio 1** - Creación de imagen Apache+PHP+TLS/SSL en Centos 7, cargando una plantilla html por defecto y un fichero .php (con un simple php\_info).

**Ejercicio 2** - Creación de imagen Apache+PHP+TLS/SSL en Ubuntu 18.04 cargando una plantilla html por defecto y un fichero .php (con un simple php\_info).

**Ejercicio 3** - Creación de imagen Nginx+PHP-FPM en Centos 7 cargando una plantilla html por defecto y un fichero .php (con un simple php\_info).

**Ejercicio 4** - Creación de imagen Nginx+PHP-FPM en Ubuntu 18.04 cargando una plantilla html por defecto y un fichero .php (con un simple php\_info).



# Docker Containers

Son una instancia de ejecución de una imagen.

Son temporales por lo que los cambios los debemos definir en el fichero Dockerfile.

Tienen una capa de lectura/escritura.

Podemos crear varios contenedores partiendo de una sola imagen.

## 2.1 Listar/mapear puertos

Cuando creamos un contenedor mediante “docker run” debemos recordar que para poder acceder a el, deberemos indicar el puerto por el que vamos a acceder en nuestra máquina y el puerto que tendrá el contenedor accesible. Si no lo hacemos no podremos acceder al mismo. Ejemplo:

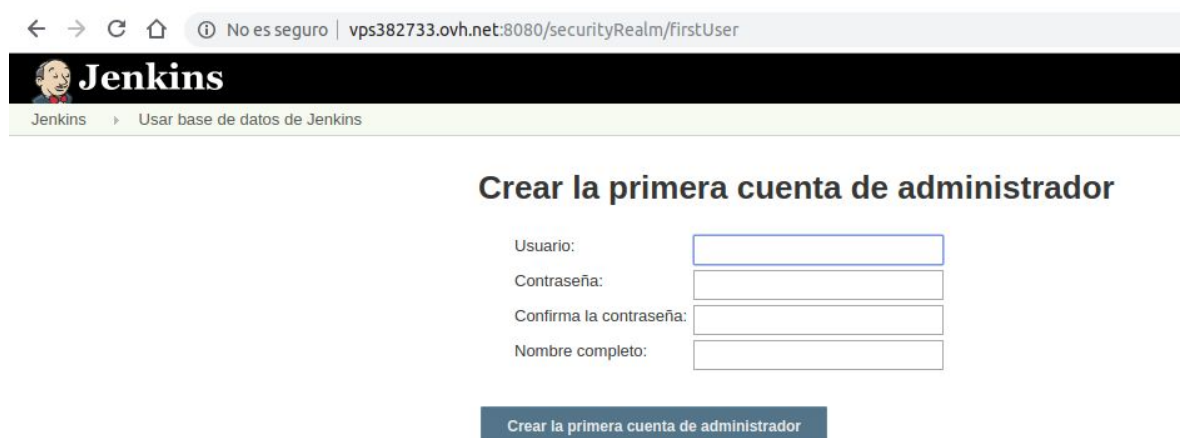
1-Descargamos la imagen oficial de Jenkins:

```
dockeruser@vps382733:~$ docker pull jenkins
```

2-Y levantamos un contenedor:

```
dockeruser@vps382733:~$ docker run -d -p 8080:8080 jenkins
```

Utilizaremos -d para indicar que funcione en segundo plano y con “-p” le diremos que acceda al puerto 8080 por el 8080 de nuestra máquina. Y comprobamos que es accesible el contenedor:



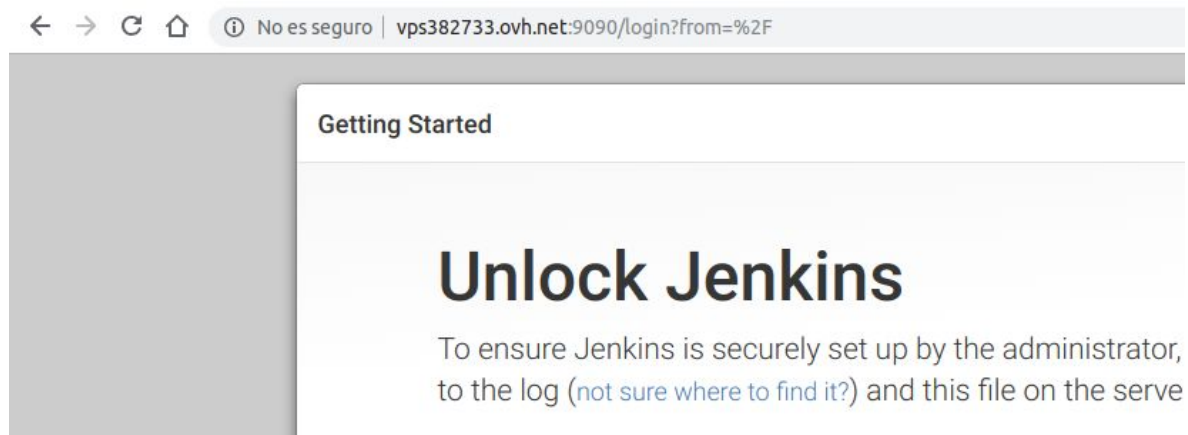
The screenshot shows a web browser window with the address bar displaying "vps382733.ovh.net:8080/securityRealm/firstUser". The page header includes the Jenkins logo and the text "Usar base de datos de Jenkins". The main heading is "Crear la primera cuenta de administrador". Below this, there are four input fields labeled "Usuario:", "Contraseña:", "Confirma la contraseña:", and "Nombre completo:". At the bottom of the form is a blue button labeled "Crear la primera cuenta de administrador".

Anteriormente hablábamos de que una misma imagen podría servir para levantar varios contenedores. En estos casos obviamente no podremos acceder a todos a través del mismo puerto, por lo que simplemente

deberíamos modificar el puerto mediante el que accedemos por nuestra máquina:

```
dockeruser@vps382733:~$ docker run -d -p 9090:8080 jenkins
```

Y al acceder al puerto 9090, podréis comprobar que accedemos a este segundo contenedor que hemos levantado:



## 2.2 Iniciar/reiniciar/detener

-Detener contenedor:

```
docker stop nombre_contenedor
```

-Iniciar contenedor:

```
docker start nombre_contenedor
```

-Reiniciar contenedor

```
docker restart nombre_contenedor
```

Para poder acceder a un contenedor e interactuar con el utilizaremos:

```
docker exec -ti nombre_contenedor bash
```

```
dockeruser@vps382733:~$ docker exec -ti competent_sammet bash
jenkins@b57058397f56:/$
```

Utilizamos -t para que sea mediante terminal y -i para que sea interactivo. Una vez ejecutado podremos ver como cambia el prompt al del propio contenedor y accederemos con el usuario "jenkins" del sistema.

Pero...¿y si queremos acceder con el usuario root del contenedor?. Lo haremos con una modificación del comando anterior:

```
docker exec -u root -ti nombre_contenedor bash
```

```
dockeruser@vps382733:~$ docker exec -u root -ti competent_sammet bash
root@b57058397f56:/#
```

Esto es muy útil para aplicaciones como Jenkins en las que nos pide acceder a un fichero concreto y pegarle el valor del mismo:

## Unlock Jenkins

To ensure Jenkins is securely set up by the administrator, a password has been written to the log ([not sure where to find it?](#)) and this file on the server:

```
/var/jenkins_home/secrets/initialAdminPassword
```

Please copy the password from either location and paste it below.

Administrator password

```
root@b57058397f56:/# cat /var/jenkins_home/secrets/initialAdminPassword
0e7da41ff69c4375813936c6a069a6a5
root@b57058397f56:/#
```

## 2.3 Variables de entorno

Las variables de entorno se pueden definir tanto en el Dockerfile como vimos anteriormente con la instrucción ENV o con el comando con el que levantamos los contenedores, añadiendo un “-e”:

```
docker run -dti -e "variable=valor" nombre_imagen
```

Una buena práctica para ver con mayor detalle como funcionan las variables de entorno, sería crear un contenedor con la imagen oficial de MySQL y pasarle parámetros como por ejemplo la creación de una base de datos:

# Environment Variables

When you start the `mysql` image, you can adjust the configuration of the MySQL instance by passing one or more environment variables on the `docker run` command line. Do note that none of the variables below will have any effect if you start the container with a data directory that already contains a database: any pre-existing database will always be left untouched on container startup.

See also <https://dev.mysql.com/doc/refman/5.7/en/environment-variables.html> for documentation of environment variables which MySQL itself respects (especially variables like `MYSQL_HOST`, which is known to cause issues when used with this image).

## MYSQL\_ROOT\_PASSWORD

This variable is mandatory and specifies the password that will be set for the MySQL `root` superuser account. In the above example, it was set to `my-secret-pw`.

## MYSQL\_DATABASE

This variable is optional and allows you to specify the name of a database to be created on image startup. If a user/password was supplied (see below) then that user will be granted superuser access ([corresponding to GRANT ALL](#)) to this database.

Una imagen similar y que también sería buena práctica sería la de MongoDB

## 2.4 Comprobar recursos usados

Más adelante, veremos como limitar los recursos que nos puede consumir un contenedor. Mientras tanto si quereis ver los recursos que está utilizando, lo podreis hacer con el siguiente comando:

```
docker stats nombre_contenedor
```

Y nos devolverá el siguiente resultado:

CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT
MEM %	NET I/O	BLOCK I/O	PIDS
4939ed95dce0	my-db1	0.47%	371.1MiB / 1.9GiB
19.08%	23.4kB / 3.9kB	8.88MB / 526MB	38

## 2.5 Administración de usuarios

En ocasiones querremos crear un contenedor que sea accesible. Anteriormente hemos podido ver que mediante el comando:

```
docker exec -ti nombre_contenedor bash
```

Nos permitía acceder con el usuario por defecto que nos permitía la imagen usada o incluso decir con que usuario queremos entrar, pero habrá veces en la que queramos definir nosotros mismos el usuario con el que se debe acceder.

Si creamos un contenedor con la imagen de CentOS, podremos ver que por defecto nos deja acceder con el usuario "root":

```
dockeruser@vps382733:~/docker-images$ docker exec -ti prueba bash
[root@fbc45b49d349 /]# exit
```

¿Cómo crearíamos nuestro usuario?. Muy sencillo:

```
FROM centos:7
ENV prueba 1234
RUN useradd chus
USER chus
```

En nuestro fichero Dockerfile, utilizaremos "RUN" para crear el usuario y mediante "USER" le indicaremos el usuario por defecto con el que accederemos al contenedor:

Creamos de nuevo el contenedor:

```
dockeruser@vps382733:~/docker-images$ docker run -d -ti --name prueba centos:prueba
3c5d1197ba71ae9ebb01a0dcd35c206f953f929dd0745dde460d7a7e5bd057f0
```

Y al acceder, podremos confirmar que efectivamente entramos con el usuario que le hemos definido por defecto:

```
dockeruser@vps382733:~/docker-images$ docker exec -ti prueba bash
[chus@3c5d1197ba71 /]$ exit
```

Como indicamos en la anterior lección, si queremos acceder con un usuario distinto al predefinido por defecto, bastará con utilizar “-u” en el comando de acceso:

```
docker exec -u usuario -ti nombre_contenedor bash
```

## 2.6 Limitar recursos de un contenedor

Para esta lección vamos a utilizar la imagen de MongoDB:

```
dockeruser@vps382733:~/docker-images$ docker run -d --name mongo mongo
```

Si os acordais de anteriores lecciones, hablamos de que con el comando:

```
docker stats nombre_contenedor
```

Podíamos ver cuantos recursos estaba consumiendo un contenedor:

CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM %
dadf1950889a	mongo	0.34%	40.6MiB / 1.9GiB	2.09%

Si os fijais, nos muestra el % de CPU usada junto a la memoria (MEM USAGE) y también el límite que tiene de la misma para usar (LIMIT, en este caso 1,9GB). Por defecto los contenedores van a coger el máximo de memoria que tenga la máquina donde se esté ejecutando.

En Docker podemos limitar este uso con la opción memory y estas son las opciones que permite:

```
dockeruser@vps382733:~/docker-images$ docker run --help | grep memo
--kernel-memory bytes      Kernel memory limit
-m, --memory bytes         Memory limit
--memory-reservation bytes  Memory soft limit
--memory-swap bytes        Swap limit equal to memory plus swap: '-1' to enable unlimited swap
--memory-swappiness int     Tune container memory swappiness (0 to 100) (default -1)
```

Vamos a hacer una prueba, creando otro contenedor de MongoDB pero limitándole la memoria RAM a 500 Mb:

```
docker run -d -m "500mb" --name mongo2 mongo
```

Una vez creado, si ejecutamos el comando “docker stats”, podremos ver que el límite de memoria del contenedor es de 500Mb.

En el caso de la CPU tenemos varias opciones disponibles para limitarlo:



```
dockeruser@vps382733:~/docker-images$ docker run --help | grep cpu
--cpu-period int          Limit CPU CFS (Completely Fair Scheduler) period
--cpu-quota int           Limit CPU CFS (Completely Fair Scheduler) quota
--cpu-rt-period int       Limit CPU real-time period in microseconds
--cpu-rt-runtime int      Limit CPU real-time runtime in microseconds
-c, --cpu-shares int       CPU shares (relative weight)
--cpus decimal            Number of CPUs
--cpuset-cpus string       CPUs in which to allow execution (0-3, 0,1)
--cpuset-mems string       MEMs in which to allow execution (0-3, 0,1)
```

Pero primero deberemos saber cuantas CPUs tenemos disponibles:

```
dockeruser@vps382733:~/docker-images$ grep "model name" /proc/cpuinfo
model name      : Intel Core Processor (Haswell, no TSX)
```

Podremos verlo de forma más simple con:

```
dockeruser@vps382733:~/docker-images$ grep "model name" /proc/cpuinfo | wc -l
1
```

Vamos a crear un contenedor nuevo de MongoDB limitando el número de CPUs mediante:

```
docker run -d -m "1gb" --cpuset-cpus 0-1 --name mongo mongo
```

En mi caso solo tengo una CPU en la máquina donde estoy desarrollando todas las explicaciones, pero imaginemos que tenemos un servidor con 4 CPUs.

Si nos fijamos en la sintaxis de "cpuset-cpus" nos indica que podemos pasarle un valor entre 0-3 y esto quiere decir los cores que le permitiremos usar al contenedor.

Por lo que si queremos usar únicamente 2 CPUs deberemos pasarle al comando los parámetros "0-1" y si quisiésemos permitirle usar 3, el parámetro cambiaría a "0-2".

## 2.7 Copiar ficheros a un contenedor

Vamos a crearnos un contenedor con la imagen de Apache oficial:

```
docker run -d --name apache -p 80:80 httpd
```

Una situación normal para este tipo de servicio sería querer copiar nuestra propia página web, por lo que vamos a crearnos un fichero index.html sencillo para hacer esta práctica:

```
echo "Mi primer fichero copiado a un contenedor" > index.html
```

Para copiarlo sería tan sencillo como usar el comando “docker cp” que nos permitirá copiar ficheros desde fuera hacia dentro del contenedor y viceversa. Vamos a copiar nuestro primer fichero directamente en la ruta por defecto que tiene Apache. En este caso añadimos “index.html” para que sobrescriba el fichero actual:

```
docker cp index.html apache:/usr/local/apache2/htdocs/index.html
```

La sintaxis es muy sencilla:

*docker cp fichero nombre\_contenedor:ruta*

Ahora si accedemos a nuestro navegador, podremos ver que efectivamente el fichero se ha copiado correctamente:

---

### Mi primer fichero copiado a un contenedor

Si el escenario fuese el contrario, es decir, que necesitamos sacar un fichero del contenedor por ejemplo un log sería igual de sencillo pero la sintaxis del comando cambiaría:

*docker cp nombre\_contenedor:ruta\_fichero ruta\_destino*

Por ejemplo:

```
docker cp apache:/var/log/dpkg.log .
```

En este caso añadimos un “.” al final por que queremos que nos lo descargue en la misma ruta donde ejecutamos el comando.

## 2.8 Convertir un contenedor en una imagen

Siguiendo el escenario anterior, como bien sabemos todos los cambios que realicemos en un contenedor son efímeros y una vez lo eliminemos se perderán y no se guardarán en la imagen. Por lo que si queremos crear una imagen para ser utilizada en futuras ocasiones lo haremos con “docker commit”:

*docker commit nombre\_contenedor nueva\_imagen*

Siguiendo con el ejemplo anterior, sería algo así:

```
docker commit apache apache_modificado
```

De forma que si mostramos el listado de imagenes con “docker images” podremos ver que se encuentra entre ellas:

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
apache_modificado	latest	97d645827459	About a minute ago	132MB

Y si levantamos un nuevo contenedor utilizando esa imagen, veremos que directamente el index que mostrará será el que creamos en el anterior apartado.

Hay que tener cuidado con esto ya que si creamos un fichero dentro de un volumen del contenedor que hayamos definido en la imagen, no se guardarán. Pero más adelante veremos con mayor detalle los volúmenes en Docker.

## 2.9 Modificar directorio de contenedores

Por defecto Docker almacena sus contenedores en el siguiente directorio:

```
/var/lib/docker
```

Pero en ocasiones quizás necesitaremos cambiar dicho directorio y para ello deberemos modificar el fichero “daemon.json” que se encuentra ubicado en:

```
/etc/docker/daemon.json
```

Y añadir:

```
{
  "graph": "/midirectorio/dockerhome"
}
```

## 2.10 Ejercicios

**Ejercicio 1** - Crear un contenedor con las siguientes características:

Necesitamos un servidor Apache con PHP que tenga:

- 500Mb de memoria RAM.
- Acceso a una sola CPU.
- Dos variables de entorno (a elegir)
- El servidor debe ser accesible por el puerto 6666 del navegador.

**Ejercicio 2** - Crear un contenedor con las siguientes características:

De nuevo otro contenedor con Apache y PHP:

- 200Mb de memoria RAM.

- Acceso a dos CPUs.
- Una sola variable de entorno.
- El servidor web debe ser accesible por el puerto 8181

Para comprobar que el ejercicio esté correcto deberemos tener ambos contenedores funcionando y deben ser accesible por los puertos indicados en cada ejercicio.

## Docker Volumes

Los volúmenes permiten almacenar datos de manera persistente en un contenedor en nuestra máquina local, de forma que si eliminamos un contenedor el contenido del volumen se va a mantener. Hay tres tipos:

-Host: volúmenes que se almacenan en nuestro servidor de Docker (Docker host) y se ubican en una carpeta dentro del sistema de ficheros que definamos.

-Anonymous: son los volúmenes en los que no definimos una carpeta, pero Docker la genera de manera automática.

-Named volumes: son volúmenes que nosotros creamos pero no están en nuestro sistema de ficheros, son administradas por Docker y a diferencia de los Anonymous si que tienen un nombre concreto.

Para empezar a entender como funcionan los volúmenes, vamos a utilizar contenedores con MySQL con los que entenderemos los tres tipos de volúmenes.

### 3.1 Host volumes

Por defecto, si no utilizamos los volúmenes e introducimos datos en la base de datos creada (docker-db) una vez borremos el contenedor perderemos los datos y es información importante obviamente no queremos perderla. El contenedor de MySQL, tal y como podremos ver en la ficha de Docker Hub de esa imagen, guarda toda la configuración y bases de datos en el directorio "/var/lib/mysql".

-Primero crearemos una carpeta en el Docker host que destinaremos a guardar los ficheros del volumen. Por ejemplo:

```
mkdir /opt/mysql
```

-Después levantaremos un contenedor MySQL, especificándole que utilice como volumen la carpeta creada anteriormente. Para ello añadiremos “-v ruta\_dockerhost:ruta\_contenedor” al comando para levantarlo el contenedor :

```
docker run -d --name my-db1 -p 3306:3306 -e  
"MYSQL_ROOT_PASSWORD=12345678" -v /opt/mysql:/var/lib/mysql mysql:5.7
```

-Una vez ejecutado, podremos ver como en “/opt/mysql” se habrán copiado los datos de “/var/lib/mysql”:

auto.cnf	client-cert.pem	ibdata1	ibtmp1	private_key.pem	server-key.pem
ca-key.pem	client-key.pem	ib_logfile0	mysql	public_key.pem	sys
ca.pem	ib_buffer_pool	ib_logfile1	performance_schema	server-cert.pem	

-Cuando se haya creado el contenedor y arrancado el servicio, vamos a probar a acceder al contenedor:

```
mysql -u root -h 127.0.0.1 -p12345678
```

Y creamos dos bases de datos:

```
mysql> create database prueba;  
Query OK, 1 row affected (0.01 sec)  
  
mysql> create database prueba1;  
Query OK, 1 row affected (0.00 sec)
```

Si volvemos a comprobar el contenido de “/opt/mysql” en nuestro Docker Host podremos ver como están las dos bases de datos que acabamos de crear:

auto.cnf	client-cert.pem	ibdata1	ibtmp1	private_key.pem	public_key.pem	sys
ca-key.pem	client-key.pem	ib_logfile0	mysql	prueba	server-cert.pem	
ca.pem	ib_buffer_pool	ib_logfile1	performance_schema	prueba1	server-key.pem	

-Si eliminamos el contenedor actual y volvemos a crearlo, al acceder a MySQL y comprobar las bases de datos podreis confirmar que se encuentran las creadas anteriormente “prueba” y “prueba1”.

## 3.2 Anonymous volumes

Con los volúmenes anónimos el funcionamiento es similar, pero con una ligera diferencia a la hora de ejecutar el comando con el que levantaremos el contenedor. Esta vez, no le diremos en que carpeta queremos que nos guarde el contenido de “/var/lib/mysql”:

```
docker run -d --name my-db1 -p 3306:3306 -e  
"MYSQL_ROOT_PASSWORD=12345678" -e "MYSQL_DATABASE=prueba-db" -v
```

```
/var/lib/mysql mysql:5.7
```

Pero, ¿donde guarda ese volumen?. Para comprobarlo ejecutamos el siguiente comando:

```
dockeruser@vps382733:~$ docker info | grep -i root
Docker Root Dir: /var/lib/docker
```

Y al dirigirnos a ese directorio veremos que dentro de la carpeta “volumes” se encuentra el volumen anónimo creado:

```
root@vps382733:/var/lib/docker/volumes# ls
225b40d400864aa863966f928530a268a42d626b15c5d9629589b943fdb96e74
```

Si eliminamos el contenedor, podremos comprobar que no se eliminan esos volúmenes. Pero cuidado, hasta ahora a la hora de utilizar “docker rm” añadíamos “-fv” para eliminarlo y si lo hacemos con un contenedor que contenga volúmenes anónimos Docker los eliminará también, por lo que únicamente especificaremos “-f” para eliminarlos.

### 3.3 Named volumes

Los volúmenes nombrados se crean en el directorio root de Docker, el mismo directorio donde vimos que se creaban los anónimos, y nosotros mismos especificaremos el nombre con:

```
docker volume create nombre_volumen
```

Para asignarlo a un contenedor, es más sencillo que con los host ya que únicamente deberemos hacer referencia al nombre del volumen creado y no será necesario hacer referencia a la ruta completa. Como ejemplo vamos a tomar el comando utilizado para crear MySQL:

```
docker run -d --name my-db1 -p 3306:3306 -e  
"MYSQL_ROOT_PASSWORD=12345678" -v nombre_contenedor:/var/lib/mysql  
mysql:5.7
```

Al contrario de lo que sucedía con los volúmenes anónimos, si utilizamos “-fv” en el comando “docker rm” el volumen nombrado no será eliminado.

### 3.4 Dangling volumes

Al igual que sucedía con las imágenes, en ocasiones nos encontraremos con volúmenes que se han quedado “huerfanos” y no tienen ningún contenedor



asociado. Para listarlos y poder eliminarlos después, podremos hacerlo con el siguiente comando:

```
docker volume ls -f dangling=true -q
```

Si queremos listarlos y eliminarlos en un solo comando usaremos:

```
docker volume ls -f dangling=true -q | xargs docker volume rm
```

## 3.5-Ejercicios

**Ejercicio 1**-Para esta lección el ejercicios propuesto será similar a la anterior, pero con una ligera modificación:

Necesitamos un servidor Apache con PHP que tenga:

- 500Mb de memoria RAM.
- Acceso a una sola CPU.
- Dos variables de entorno (a elegir).
- El servidor debe ser accesible por el puerto 6666 del navegador.
- Crearemos un volumen host en la carpeta "/opt/web" donde irá nuestro contenido web que deberá seguir cuando eliminemos el contenedor.

# Docker Network

## 4.1 Red bridge

En Docker, podemos encontrar varios tipos de redes, pero por defecto cada vez que creamos un contenedor lo crea en la red "bridge".

Una vez levantado un contenedor podremos ver que le asigna una subred y para comprobarlo debemos utilizar:

```
docker inspect nombre_contenedor
```

En la línea "IPAddress" viene especificada dicha dirección. Podemos hacer una prueba rápida levantando un contenedor que tenga ping habilitado, como puede ser la imagen de CentOS:

```
docker run -dti centos
```

Y realizamos la prueba:

```
docker exec nombre_contenedor bash -c "ping dir_ip"
```

\*Importante: el ping lo podremos realizar con la dirección IP, pero no por nombre de contenedor.

## 4.2 Crear y eliminar una red propia

Las redes se crean mediante:

```
docker network
```

Si ejecutamos el comando, obtendremos la ayuda y las posibilidades que ofrece. Para la creación usamos:

```
docker network create nombre_red
```

Podemos comprobar que se ha creado correctamente con:

```
docker network ls | grep nombre_red
```

Si quisiésemos crearla con nuestra propia subred y nuestra puerta de enlace usaríamos:

```
docker network create -d bridge --subnet bloque_ip --gateway gateway_ip nombre_red
```

“-d” es para definir el driver de red, que por defecto es “bridge”.

Para consultar toda la información de una red de Docker, lo podemos hacer con el siguiente comando:

```
docker network inspect nombre_red|less
```

Si quisiésemos eliminar una red, usaríamos:

```
docker network rm nombre_red
```

Es importante saber que si tenemos algún contenedor conectado a esa red no nos va a permitir borrarla, por lo que deberemos o eliminar el contenedor o desconectarlo de la red (más adelante explicaremos como).

## 4.3 Cambiar la red de un contenedor

Como ya hemos visto anteriormente, Docker asigna por defecto un contenedor a la red “bridge” pero lo normal es querer definir nuestras propias redes. Para crear y asignar un contenedor a una red concreta usamos:

```
docker run --network nombre_red -d --name nombre_contenedor -ti imagen
```

Anteriormente, vimos como en la red por defecto “bridge” nose podía hacer ping de un contenedor a otro utilizando el nombre del mismo. En las redes propias si que lo permite siempre y cuando estén en la misma red.

## 4.4 Añadir nueva interfaz de red

Por defecto los contenedores que se encuentren en distintas redes no se pueden ver entre ellos. Pero al igual que sucede con las máquinas virtuales, podemos añadirle otro interfaz de red a los contenedores para que formen parte de una red distinta con:

```
docker network connect nombre_red nombre_contenedor
```

Podremos comprobar ambos interfaces de nuevo con:

```
docker inspect nombre_contenedor
```

Si quisiésemos eliminar dicho interfaz:

```
docker network disconnect nombre_red nombre_contenedor
```

## 4.5 Asignar IP a un contenedor

Para asignar una IP concreta a un contenedor:

```
docker run --network nombre_red --ip dir_ip -d --name nombre_contenedor -ti nombre_imagen
```

Para que funcione correctamente, debemos haber creado previamente la red y haberle asignado un bloque de IPs y una puerta de enlace por defecto.

## 4.6 Red host

La red host es la red del propio servidor y viene creada con ese nombre dentro de Docker. Para crear un contenedor en esa red:

```
docker run --network host -d --name nombre_contenedor -ti imagen
```

Y creará un contenedor duplicando la configuración de red del host.

## 4.7 Configurando IPs en servicios

Cuando levantamos un contenedor por ejemplo con Apache, este será accesible por el puerto 80 a través de localhost, 127.0.0.1 o a través de la dirección IP pública o de la red que tenga asignada.

Pero, quizás queramos que el servicio sea accesible únicamente desde una de las interfaces de red que tenga y para ello usaremos:

```
docker run -d -p dirección_ip:puerto:puerto nombre_imagen
```

Ejemplo:

```
docker run -d -p 192.168.100.2:8080:80 nginx
```

Docker registry

## 4.8 Ejercicios

**Ejercicio 1**-Deberemos crear dos contenedores que serán uno con Wordpress llamado "wordpress" y otro con MySQL llamado "mysql".

Ambos deben estar en la misma red y tenemos que tener en cuenta que hay que modificar el fichero de configuración de Wordpress para que se conecte con la base de datos del contenedor "mysql".

# Docker Registry

## 5.1 Crear nuestro propio registry

Un servicio de registry, es el servicio mediante el cual podremos crear nuestro propio repositorio de imágenes de Docker para poder descargarlas y usarlas para desplegar contenedores.

Cuando usamos "docker pull" para descargar imágenes, por defecto nos conectamos al registry oficial de Docker.

El registry que vamos a crear es un contenedor y lo haremos mediante:

```
docker run -d -p 5000:5000 --name nombre_registry -v $PWD/data:/var/lib/registry registry:2
```

## 5.2 Subida y descarga de imágenes

Para subir una imagen a nuestro registry, primero debemos "taggearla" para poder especificar la versión del software que tengamos dentro. Ejemplo:

```
docker run -tag nombre_imagen:version  
ip_registry:5000/nombre_imagen_subida
```

Una vez hecho, la subimos:

```
docker push ip_registry:5000/nombre_imagen
```

Si quisiéramos descargarla directamente de nuestro registry:

```
docker pull ip_registry:5000/nombre_imagen
```

Podemos usar "localhost" en vez de la dirección IP, pero si usamos esto último nos va a dar un error de autenticación por que no lo tenemos cifrado con un certificado SSL.

Si queremos deshabilitarlo para que no solicite ningún certificado, debemos editar el fichero "docker.service" cuya ruta es:

```
/lib/systemd/system/docker.service
```

Y modificar la línea "ExecStartd=/usr/bin/dockerd" para que quede de la siguiente manera:

```
ExecStartd=/usr/bin/dockerd --insecure-registry direccion_IP:puerto
```