

1 Contenedores con Docker

Enlaces de interés

- [Curso "Introducción a Docker"](#)
- [Docker for beginners](#)
- [getting-started-with-docker](#)
- [Docker for Beginners](#)

Es muy común que nos encontremos desarrollando una aplicación, y llegue el momento que decidamos tomar todos sus archivos y migrarlos, ya sea al ambiente de producción, de prueba, o simplemente probar su comportamiento en diferentes plataformas y servicios.

Para este tipo de situaciones existen herramientas que nos facilitan el embalaje y despliegue de la aplicación, es aquí donde entra en juego los contenedores (Por ejemplo Docker o Podman).

Estas herramientas nos permiten crear "contenedores", que son aplicaciones empaquetadas auto-suficientes, muy livianas, capaces de funcionar en prácticamente cualquier ambiente, ya que tiene su propio sistema de archivos, librerías, terminal, etc.

Docker es una tecnología contenedor de aplicaciones construida sobre LXC.

1.1 Instalación

Enlaces de interés:

- [EN - Docker installation on SUSE](#)
- [ES - Curso de Docker en](#)

[vídeos](#) Ejecutar como superusuario:

- `zypper in docker`, instalar docker en OpenSUSE (`apt install docker` en Debian/Ubuntu).
- `systemctl start docker`, iniciar el servicio. NOTA: El comando `docker daemon` hace el mismo efecto.
- `systemctl enable docker`, si queremos que el servicio de inicie automáticamente al encender la máquina.
- `cat /proc/sys/net/ipv4/ip_forward`, consultar el estado de `IP_FORWARD`. Debe estar `activo=1`.

1.2 Primera prueba

Como usuario root:

- Incluir a nuestro usuario (nombre-del-alumno) como miembro del grupo docker. Solamente los usuarios dentro del grupo docker tendrán permiso para usarlo.
- id NOMBRE-ALUMNO, debe mostrar que pertenecemos al grupo docker.
- Cerrar sesión y volver a entrar al sistema con nuestro usuario normal.

Como usuario normal:

- docker version, comprobamos que se muestra la información de las versiones cliente y servidor.

```
dam2@jupiter06:~$ docker version
Client: Docker Engine - Community
 Version:      24.0.6
 API version:  1.43
 Go version:   go1.20.7
 Git commit:   ed223bc
 Built:        Mon Sep  4 12:32:12 2023
 OS/Arch:     linux/amd64
 Context:      default

Server: Docker Engine - Community
 Engine:
  Version:      24.0.6
  API version:  1.43 (minimum version 1.12)
  Go version:   go1.20.7
  Git commit:   1a79695
  Built:        Mon Sep  4 12:32:12 2023
  OS/Arch:     linux/amd64
  Experimental: false
 containerd:
  Version:      1.6.22
  GitCommit:    8165feabfdfe38c65b599c4993d227328c231fca
```

- docker run hello-world, este comando hace lo siguiente:
 - Descarga una imagen "hello-world"
 - Crea un contenedor
 - Ejecuta la aplicación que hay dentro.

```
dam2@jupiter06:~$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
c1ec31eb5944: Pull complete
Digest: sha256:4bd78111b6914a99dbcf560e6a20eab57ff6655aea4a80c50b0c5491968cbc2e6
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
```

- docker images, ahora vemos la nueva imagen "hello-world" descargada en nuestro equipo local.

```
dam2@jupiter06:~$ docker images
REPOSITORY          TAG             IMAGE ID        CREATED         SIZE
mongo                latest          e325fe350a8c   2 weeks ago    757MB
hello-world          latest          d2c94e258dcb   8 months ago    13.3kB
gvenzl/oracle-xe     latest          eccf57eac1b8   11 months ago  3.18GB
mongo                <none>          a440572ac3c1   11 months ago  639MB
```

- docker ps -a, vemos que hay un contenedor en estado 'Exited'.

```
dam2@jupiter06:~$ docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS                    PORTS          NAMES
15f0acb5dd30   hello-world    "/hello"                About a minute ago    Exited (0) About a minute ago           silly_shaw
da7a264e9821   mongo:latest   "docker-entrypoint.s..." 4 days ago      Exited (0) 2 days ago           custommongodb
b7dd0508d6fd   gvenzl/oracle-xe "container-entrypoin..." 11 months ago    Exited (143) 11 months ago         relaxed_dhawan
```

- docker stop IDContainer, parar el contenedor identificado por su IDContainer. Este valor lo obtenemos tras consultar la salida del comando anterior (docker ps -a).

```
dam2@jupiter06:~$ docker stop 15f0acb5dd30
15f0acb5dd30
```

- docker rm IDContainer, eliminar el contenedor.

```
dam2@jupiter06:~$ docker rm 15f0acb5dd30
15f0acb5dd30
dam2@jupiter06:~$ docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS                    PORTS          NAMES
da7a264e9821   mongo:latest   "docker-entrypoint.s..." 4 days ago      Exited (0) 2 days ago           custommongodb
b7dd0508d6fd   gvenzl/oracle-xe "container-entrypoin..." 11 months ago    Exited (143) 11 months ago         relaxed_dhawan
dam2@jupiter06:~$
```

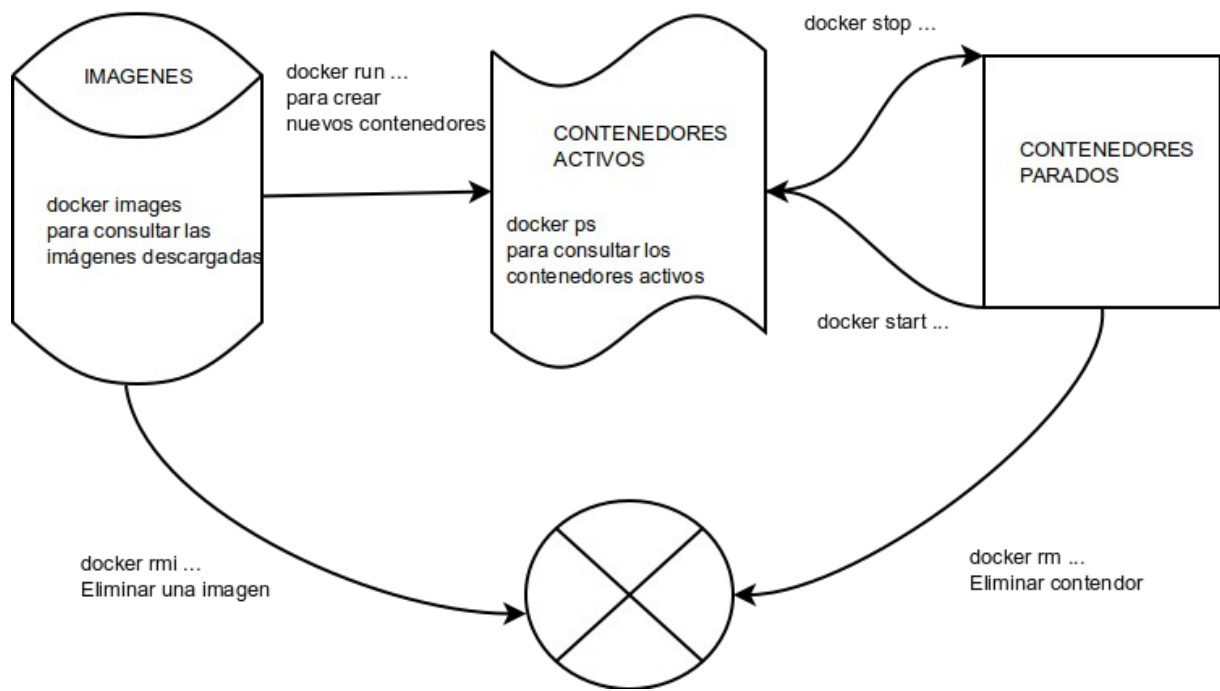
Hemos comprobado que Docker funciona correctamente.

1.3 TEORIA: Sólo para LEER

Tabla de referencia para no perderse:

Software	Base	Sirve para crear	Aplicaciones
VirtualBox	ISO	Máquinas virtuales	N
Vagrant	Box	Máquinas virtuales	N
Docker	Imagen	Contenedores	1

Veamos cómo es el flujo de trabajo con los contenedores Docker:



Comandos útiles de Docker:

Comando	Descripción
<code>docker stop CONTAINERID</code>	Parar un contenedor
<code>docker start CONTAINERID</code>	Iniciar un contenedor
<code>docker attach CONTAINERID</code>	Conectar el terminal actual con el contenedor
<code>docker ps</code>	mostrar los contenedores en ejecución
<code>docker ps -a</code>	mostrar todos los contenedores (en ejecución o no)
<code>docker rm CONTAINERID</code>	Eliminar un contenedor
<code>docker rmi IMAGENAME</code>	Eliminar una imagen

1.4 Alias

Para ayudarnos a trabajar de forma más rápida con la línea de comandos podemos agregar alias `d='docker'` a nuestro fichero `$HOME/.alias`.

Ahora `d ps` equivale a `docker ps`.

2 Creación manual de nuestra imagen

Nuestro SO base es OpenSUSE, pero vamos a crear un contenedor Debian, y dentro instalaremos Nginx.

2.1 Crear un contenedor manualmente

Descargar una imagen

- docker search debian, buscamos en los repositorios de Docker Hub contenedores con la etiqueta debian.

```
dam2@jupiter06:~$ docker search debian
```

NAME	DESCRIPTION	STARS	OFFICIAL	AUTOMATED
ubuntu	Ubuntu is a Debian-based Linux operating sys...	16790	[OK]	
debian	Debian is a Linux distribution that's compos...	4917	[OK]	
neurodebian	NeuroDebian provides neuroscience research s...	106	[OK]	
bitnami/debian-base-buildpack	Debian base compilation image	2		[OK]
kasmweb/debian-bullseye-desktop	Debian Bullseye desktop for Kasm Workspaces	6		
kasmweb/debian-bookworm-desktop	Debian Bookworm desktop for Kasm Workspaces	2		
mirantis/debian-build-ubuntu-xenial		0		
rancher/debianconsole		1		

- docker pull debian, descargamos una imagen en local.

```
dam2@jupiter06:~$ docker pull debian
Using default tag: latest
latest: Pulling from library/debian
1b13d4e1a46e: Pull complete
Digest: sha256:b16cef8cbcb20935c0f052e37fc3d38dc92bfec0bcfb894c328547f81e932d67
Status: Downloaded newer image for debian:latest
docker.io/library/debian:latest
```

- docker images, comprobamos que se ha descargado.

```
dam2@jupiter06:~$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
debian	latest	a6916e41aa87	11 days ago	117MB
mongo	latest	e325fe350a8c	2 weeks ago	757MB
hello-world	latest	d2c94e258dcb	8 months ago	13.3kB
gvenzl/oracle-xe	latest	eccf57eac1b8	11 months ago	3.18GB
mongo	<none>	a440572ac3c1	11 months ago	639MB

Crear un contenedor: Vamos a crear un contenedor con nombre app1debian a partir de la imagen debian, y ejecutaremos el programa /bin/bash dentro del contenedor:

- docker run --name=app1debian -i -t debian /bin/bash

```
dam2@jupiter06:~$ docker run --name=app1debian -i -t debian /bin/bash
root@2852dd28cb4b:/#
root@2852dd28cb4b:/#
```

Parámetro	Descripción
docker run	Crea un contenedor y lo pone en ejecución
-name	Nombre del nuevo contenedor
-i	Abrir una sesión interactiva
-t	Imagen que se usará para crear el contenedor
/bin/bash	Es la aplicación que se va a ejecutar

2.2 Personalizar el contenedor

Ahora estamos dentro del contenedor, y vamos a personalizarlo a nuestro gusto:

Instalar aplicaciones dentro del contenedor

root@IDContenedor:/# cat /etc/motd # Comprobamos que estamos en Debian

```
root@2852dd28cb4b:/# cat /etc/motd

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
root@2852dd28cb4b:/#
```

root@IDContenedor:/# apt update

```
root@2852dd28cb4b:/# apt update
Get:1 http://deb.debian.org/debian bookworm InRelease [151 kB]
Get:2 http://deb.debian.org/debian bookworm-updates InRelease [52.1 kB]
Get:3 http://deb.debian.org/debian-security bookworm-security InRelease [48.0 kB]
Get:4 http://deb.debian.org/debian bookworm/main amd64 Packages [8787 kB]
Get:5 http://deb.debian.org/debian bookworm-updates/main amd64 Packages [12.7 kB]
Get:6 http://deb.debian.org/debian-security bookworm-security/main amd64 Packages [134 kB]
Fetched 9185 kB in 2s (3690 kB/s)
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
All packages are up to date.
root@2852dd28cb4b:/#
```

root@IDContenedor:/# apt install -y nginx # Instalamos nginx en el contenedor

```
root@2852dd28cb4b:/# apt install -y nginx
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  iproute2 krb5-locales libatml libbpf1 libbsd0 libcap2-bin libelf1 libgssapi-krb5-2 libk5crypto3 libkeyutils1 libkrb5-3 libkrb5support0
  liblmnl0 libpam-cap libssl3 libtirpc-common libtirpc3 libxtables12 nginx-common
Suggested packages:
  iproute2-doc python3:any krb5-doc krb5-user fcgiwrap nginx-doc ssl-cert
The following NEW packages will be installed:
  iproute2 krb5-locales libatml libbpf1 libbsd0 libcap2-bin libelf1 libgssapi-krb5-2 libk5crypto3 libkeyutils1 libkrb5-3 libkrb5support0
  liblmnl0 libpam-cap libssl3 libtirpc-common libtirpc3 libxtables12 nginx nginx-common
```

root@IDContenedor:/# apt install -y nano # Instalamos editor nano en el contenedor

```
root@2852dd28cb4b:/# apt install -y nano
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  libgpm2 libncursesw6
Suggested packages:
  gpm hunspell
The following NEW packages will be installed:
  libgpm2 libncursesw6 nano
0 upgraded, 3 newly installed, 0 to remove and 0 not upgraded.
Need to get 837 kB of archives.
After this operation, 3339 kB of additional disk space will be used.
Get:1 http://deb.debian.org/debian bookworm/main amd64 libncursesw6 amd64 6.4-4 [134 kB]
Get:2 http://deb.debian.org/debian bookworm/main amd64 nano amd64 7.2-1 [689 kB]
Get:3 http://deb.debian.org/debian bookworm/main amd64 libgpm2 amd64 1.20.7-10+b1 [14.2 kB]
```

Crear un fichero HTML holamundo1.html.

root@IDContenedor:/# echo "<p>Hola nombre-del-alumno</p>" > /var/www/html/holamundo1.html

```
root@2852dd28cb4b:/# echo "<p>Hola nombre-del-alumno</p>" > /var/www/html/holamundo1.html
root@2852dd28cb4b:/# █
```

Crear un script /root/server.sh con el siguiente contenido:

```
#!/bin/bash
echo "[INFO] Iniciando Nginx!"
/usr/sbin/nginx &

echo "[INFO] No cierras esta terminal y abre una
nueva" while(true) do
sleep 60
done
```

```
root@2852dd28cb4b:/# cd /root/
root@2852dd28cb4b:~# nano server.sh
root@2852dd28cb4b:~#
```

```
GNU nano 7.2 server.sh *
#!/bin/bash
echo "[INFO] Iniciando Nginx!"
/usr/sbin/nginx &

echo "[INFO] No cierras esta terminal y abre una nueva" while(true) do
sleep 60 done
```

Recordatorio:

- Hay que poner permisos de ejecución al script para que se pueda ejecutar (chmod
- `+x /root/server.sh`).

```
root@2852dd28cb4b:~# chmod +x /root/server.sh
```

- La primera línea de un script, siempre debe comenzar por `#!/`, sin espacios.
- Este script inicia el programa/servicio y entra en un bucle, para mantener el contenedor activo y que no se cierre al terminar la aplicación.

2.3 Crear una imagen a partir del contenedor

Ya tenemos nuestro contenedor auto-suficiente de Nginx, ahora vamos a crear una nueva imagen que incluya los cambios que hemos hecho.

- Abrir otra ventana de terminal.
- `docker commit app1debian nombre-del-alumno/nginx1`, a partir del contenedor modificado vamos a crear la nueva imagen que se llamará "nombre-del-alumno/nginx1".

```
dam2@jupiter06:~$ docker commit app1debian ruben/nginx1
sha256:04312a1d84566b036a88b4a826630127a215b67c17420a5eb818ce0acfa217a8
```


NOTA:

- Los estándares de Docker estipulan que los nombres de las imágenes deben seguir el formato `nombreusuario/nombreimagen`.
- Todo cambio realizado que no se acompañe de un commit a la imagen, se perderá en cuanto se cierre el contenedor.
- `docker images`, comprobamos que se ha creado la nueva imagen.

```
dam2@jupiter06:~$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
ruben/nginx1	latest	04312a1d8456	44 seconds ago	156MB
debian	latest	a6916e41aa87	11 days ago	117MB
mongo	latest	e325fe350a8c	2 weeks ago	757MB
hello-world	latest	d2c94e258dc	8 months ago	13.3kB
gvenzl/oracle-xe	latest	eccf57eac1b8	11 months ago	3.18GB
mongo	<none>	a440572ac3c1	11 months ago	639MB

- Ahora podemos parar el contenedor, `docker stop appldebian` 🏹
- Eliminar el contenedor, `docker rm appldebian`.

```
dam2@jupiter06:~$ docker stop appldebian
appldebian
```

```
dam2@jupiter06:~$ docker rm appldebian
appldebian
```

3 Crear contenedor a partir de nuestra nueva imagen

3.1 Crear contenedor con Nginx

Ya tenemos una imagen "nombre-alumno/nginx" con Nginx preinstalado dentro.

- `docker run --name=app2nginx1 -p 80 -t nombre-alumno/nginx1 /root/server.sh`, iniciar el contenedor a partir de la imagen anterior.

```
dam2@jupiter06:~$ docker run --name=app2nginx1 -p 80 -t ruben/nginx1 /root/server.sh
[INFO] Iniciando Nginx!
```

El argumento `-p 80` le indica a Docker que debe mapear el puerto especificado del contenedor, en nuestro caso el puerto 80 es el puerto por defecto sobre el cual se levanta Nginx.

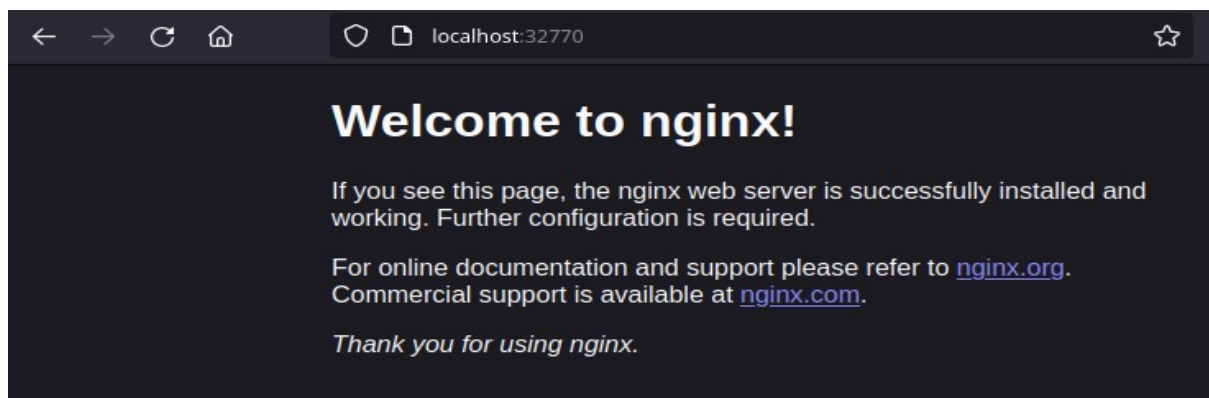
- No cierres la terminal. El contenedor ya está en ejecución y se queda esperando a que nos conectemos a él usando un navegador.
- Seguimos.

3.2 Comprobamos

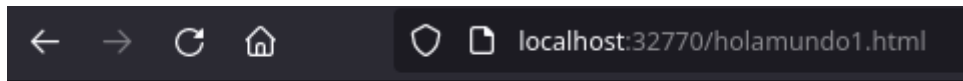
- Abrimos una nueva terminal.
- `docker ps`, nos muestra los contenedores en ejecución. Podemos apreciar que la última columna nos indica que el puerto 80 del contenedor está redireccionado a un puerto local `0.0.0.0:PORT -> 80/tcp`.

```
dam2@jupiter06:~$ docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS                               NAMES
3b319f524e50   ruben/nginx1   "/root/server.sh"       20 seconds ago Up 20 seconds  0.0.0.0:32770->80/tcp, :::32770->80/tcp  app2nginx1
dam2@jupiter06:~$
```

- Abrir navegador web y poner URL `0.0.0.0:PORT`. De esta forma nos conectaremos con el servidor Nginx que se está ejecutando dentro del contenedor.



- Comprobar el acceso al fichero HTML. Abrir navegador web y poner URL 0.0.0.0.:PORT/holamundo1.html.



Hola nombre-del-alumno

- Paramos el contenedor app2nginx1 y lo eliminamos.

```
dam2@jupiter06:~$ docker stop app2nginx1
docker rm app2nginx1
dam2@jupiter06:~$ docker rm app2nginx1
app2nginx1
```

Como ya tenemos una imagen docker con Nginx (Servidor Web), podremos crear nuevos contenedores cuando lo necesitemos.

3.3 Migrar la imagen a otra máquina

¿Cómo puedo llevar los contenedores Docker a un nuevo servidor?

Enlaces de interés

- <https://www.odooargentina.com/forum/ayuda-1/question/migrar-todo-a-otro-servidor-imagenes-docker-397>
- <http://linuxide.com/linux-how-to/backup-restore-migrate-containers-docker/>

Exportar imagen Docker a fichero tar:

- `docker save -o alumnoXXdocker.tar nombre-alumno/nginx1`, guardamos la imagen "nombre-alumno/nginx1" en un fichero tar.

Intercambiar nuestra imagen exportada con la de un compañero de clase.

Importar imagen Docker desde fichero:

- Coger la imagen de un compañero de clase.
- Nos llevamos el tar a otra máquina con docker instalado, y restauramos.
- `docker load -i alumnoXXdocker.tar`, cargamos la imagen docker a partir del fichero tar. Cuando se importa una imagen se muestra en pantalla las capas que tiene. Las capas las veremos en un momento.
- `docker images`, comprobamos que la nueva imagen está disponible.
- Probar a crear un contenedor (`app3tar`), a partir de la nueva imagen.

3.4 Capas

Teoría sobre las capas. Las imágenes de docker están creadas a partir de capas que van definidas en el fichero Dockerfile. Una de las ventajas de este sistema es que esas capas son cacheadas y se pueden compartir entre distintas imágenes, esto es que si por ejemplo la creación de nuestra imagen consta de 10 capas, y modificamos una de esas capas, a la hora de volver a construir la imagen solo se debe ejecutar esta nueva capa, el resto permanecen igual.

Estas capas, aparte de ahorrarnos peticiones de red al bajarnos una nueva versión de una imagen también ahorra espacio en disco, ya que las capas que no se hayan cambiado entre versiones no se descargarán.

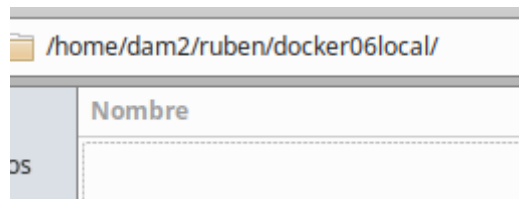
- `docker image history nombre_imagen:latest`, para consultar las capas de la imagen del compañero.

4 Dockerfile

Ahora vamos a conseguir el mismo resultado del apartado anterior, pero usando un fichero de configuración. Esto es, vamos a crear un contenedor a partir de un fichero Dockerfile.

4.1 Preparar ficheros

- Crear directorio /home/nombre-alumno/dockerXXlocal.
- Entrar al directorio anterior.



- Crear fichero holamundo2.html con el siguiente contenido:

Proyecto : dockerXXlocal

Autor : Nombre del

alumno Fecha :

Fecha actual

- Crear el fichero Dockerfile con el siguiente contenido:

FROM debian

MAINTAINER nombre-del-alumnoXX

1.0 RUN apt update

RUN apt install -y apt-

utils RUN apt install -y

nginx

COPY holamundo2.html /var/www/html

RUN chmod 666 /var/www/html/holamundo2.html

EXPOSE 80

CMD ["/usr/sbin/nginx", "-g", "daemon off;"]

```
GNU nano 4.8 Dockerfile
FROM debian
MAINTAINER nombre-del-alumnoXX 1.0 RUN apt update
RUN apt install -y apt-utils RUN apt install -y nginx

COPY holamundo2.html /var/www/html
RUN chmod 666 /var/www/html/holamundo2.html EXPOSE 80
CMD ["/usr/sbin/nginx", "-g", "daemon off;"]
```

- Enlace de interés:

<https://docs.nginx.com/nginx/admin-guide/installing-nginx/installing-nginx-docker/>

Descripción de los parámetros del Dockerfile:

Parámetro	Descripción
FROM	Imagen a partir de la cual se creará el contenedor
MAINTAINER	Información del autor
RUN	Comando que se ejecutará dentro del contenedor
COPY	Copiar un fichero dentro del contenedor
EXPOSE	Puerto de contenedor que será visible desde el exterior
CMD	Comando que se ejecutará al iniciar el contenedor

Ahora no nos hace falta el script /root/server.sh que mantenía la aplicación "despierta" porque estamos invocando (Instrucción CMD) al servidor Nginx con los parámetros "-g" y "daemon off;" que mantienen el servicio activo.

4.2 Crear imagen a partir del Dockerfile

El fichero Dockerfile contiene toda la información necesaria para construir el contenedor, veamos:

- `cd dockerXXlocal`, entramos al directorio con el Dockerfile.
- `docker build -t nombre-alumno/nginx2 .`, construye una nueva imagen a partir del Dockerfile. **OJO: el punto final es necesario.**

```
dam2@jupiter06:~/ruben/docker06local$ docker build -t ruben/nginx2 .
[+] Building 12.1s (11/11) FINISHED                                docker:default
=> [internal] load .dockerignore                                  0.0s
=> => transferring context: 2B                                     0.0s
=> [internal] load build definition from Dockerfile               0.0s
=> => transferring dockerfile: 298B                               0.0s
=> [internal] load metadata for docker.io/library/debian:latest  0.0s
=> CACHED [1/6] FROM docker.io/library/debian                    0.0s
=> [internal] load build context                                  0.0s
=> => transferring context: 36B                                     0.0s
=> [2/6] RUN apt update                                          4.4s
=> [3/6] RUN apt install -y apt-utils                            2.3s
=> [4/6] RUN apt install -y nginx                               4.8s
=> [5/6] COPY holamundo2.html /var/www/html                     0.0s
=> [6/6] RUN chmod 666 /var/www/html/holamundo2.html            0.4s
=> exporting to image                                           0.2s
=> => exporting layers                                           0.2s
=> => writing image sha256:c9a8a154c0260cc97295da643987bc51b1ee208bac6ae 0.0s
=> => naming to docker.io/ruben/nginx2                           0.0s
dam2@jupiter06:~/ruben/docker06local$
```

- *docker images*, ahora debe aparecer nuestra nueva imagen.

```
dam2@jupiter06:~/ruben/docker06local$ docker images
REPOSITORY          TAG             IMAGE ID        CREATED         SIZE
ruben/nginx2        latest         c9a8a154c026   About a minute ago  155MB
ruben/nginx1        latest         3a362579e19d   19 minutes ago    156MB
debian              latest         a6916e41aa87   11 days ago       117MB
mongo               latest         e325fe350a8c   2 weeks ago       757MB
hello-world         latest         d2c94e258dcb   8 months ago      13.3kB
gvenzl/oracle-xe    latest         eccf57eac1b8   11 months ago     3.18GB
mongo               <none>         a440572ac3c1   11 months ago     639MB
dam2@jupiter06:~/ruben/docker06local$
```

4.3 Crear contenedor y comprobar

A continuación vamos a crear un contenedor con el nombre `app4nginx2`, a partir de la imagen `nombre-alumno/nginx2`. Probaremos con:

```
docker run --name=app4nginx2 -p 8082:80 -t nombre-alumno/nginx2
```

- El terminal se ha quedado "bloqueado" porque el comando anterior no ha terminado y lo hemos lanzado en primer plano (foreground). Vamos a abrir otro terminal.

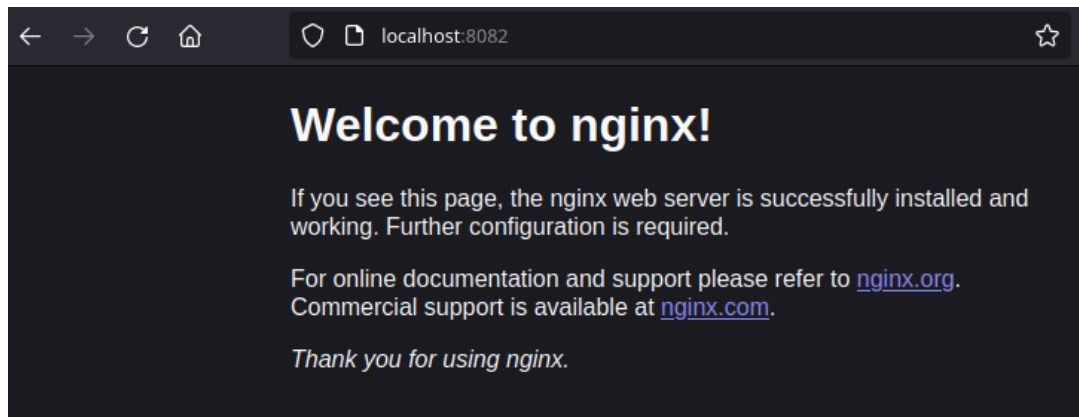
```
dam2@jupiter06:~/ruben/docker06local$ docker run --name=app4nginx2 -p 8082:80 -t
ruben/nginx2
```

Desde otra terminal:

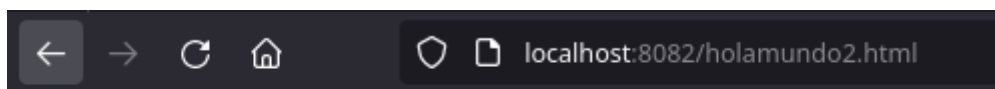
- `docker ps`, para comprobar que el contenedor está en ejecución y en escucha por el puerto deseado.

```
dam2@jupiter06:~/ruben/docker06local$ docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED
STATUS        PORTS         NAMES
03cdf5e7eb28   ruben/nginx2   "/usr/sbin/nginx -g ..." 29 seconds ago
Up 28 seconds  0.0.0.0:8082->80/tcp, :::8082->80/tcp  app4nginx2
dam2@jupiter06:~/ruben/docker06local$
```

- Comprobar en el navegador:
 - URL `http://localhost:PORTNUMBER`



- URL `http://localhost:PORTNUMBER/holamundo2.html`



Proyecto : dockerXXlocal Autor : Ruben Fecha : 22/01/24

Ahora que sabemos usar los ficheros Dockerfile, vemos que es más sencillo usar estos ficheros para intercambiar con nuestros compañeros que las herramientas de exportar/importar que usamos anteriormente.

4.4 Usar imágenes ya creadas

El ejemplo anterior donde creábamos una imagen Docker con Nginx, pero esto se puede simplificar aún más si aprovechamos las imágenes oficiales que ya existen.

Enlace de interés:

[Nginx - Docker Official Images] https://hub.docker.com/_/nginx

- Crea el directorio dockerXXweb. Entrar al directorio.
- Crear fichero holamundo3.html con:
 - Proyecto: dockerXXb
 - Autor: Nombre del alumno
 - Fecha: Fecha actual

```
GNU nano 4.8 holamundo3.html
Proyecto: docker06b
Autor: Ruben
Fecha: 22/01/24
```

- Crea el siguiente

Dockerfile FROM nginx

COPY holamundo3.html /usr/share/nginx/html

RUN chmod 666 /usr/share/nginx/html/holamundo3.html

```
GNU nano 4.8 Dockerfile
FROM nginx

COPY holamundo3.html /usr/share/nginx/html

RUN chmod 666 /usr/share/nginx/html/holamundo3.html
```

- Poner en el directorio dockerXXb los ficheros que se requieran para construir el contenedor.
- docker build -t nombre-alumno/nginx3 . , crear la imagen.

```
dam2@jupiter06:~/ruben/docker06web$ docker build -t ruben/nginx3 .
[+] Building 12.8s (8/8) FINISHED docker:default
=> [internal] load .dockerignore 0.0s
=> => transferring context: 2B 0.0s
=> [internal] load build definition from Dockerfile 0.0s
=> => transferring dockerfile: 145B 0.0s
=> [internal] load metadata for docker.io/library/nginx:latest 2.0s
=> [internal] load build context 0.0s
=> => transferring context: 91B 0.0s
=> [1/3] FROM docker.io/library/nginx@sha256:4c0fd8a8b6341bfdeca5f18f78 10.0s
```

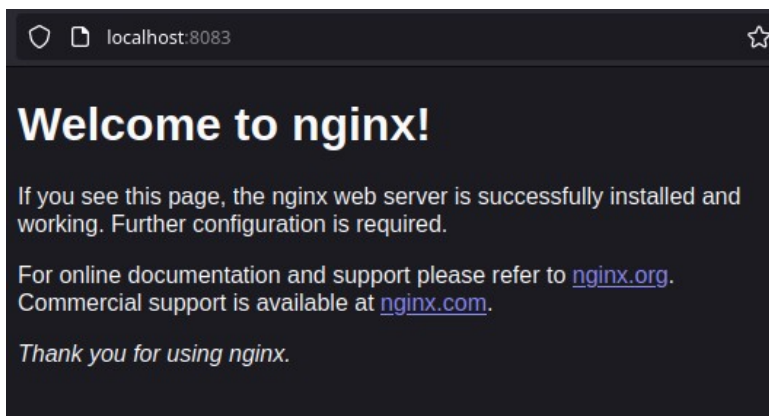
- `docker run -d --name=app5nginx3 -p 8083:80 nombre-alumno/nginx3`, crear contenedor. En este caso hemos añadido la opción "-d" que sirve para ejecutar el contenedor en segundo plano (background).

```
dam2@jupiter06:~/ruben/docker06web$ docker run -d --name=app5nginx3 -p 8083:80 ruben/nginx3
43568b17b50fe91757f799d4dd882c5a95ac9a85cbad995ebb132b8cb4042727
dam2@jupiter06:~/ruben/docker06web$
```

Parámetro	Descripción
<code>docker run -d</code>	Crea un contenedor y lo ejecuta en segundo plano
<code>--name</code>	Nombre del nuevo contenedor
<code>-p</code>	Redirección de puertos
	Se expone el puerto 80 del contenedor por el puerto 8083 de la máquina anfitrión
<code>nombre-alumno/nginx3</code>	Imagen que se usará para crear el contenedor

- Comprobar el acceso a "holamundo3.html".

```
dam2@jupiter06:~/ruben/docker06web$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED    STATUS    PORTS                               NAMES
43568b17b50f   ruben/nginx3   "/docker-entrypoint..." 21 seconds ago Up 20 seconds 0.0.0.0:8083->80/tcp, :::8083->80/tcp app5nginx3
03cdf5e7eb28   ruben/nginx2   "/usr/sbin/nginx -g ..." 8 minutes ago Up 8 minutes 0.0.0.0:8082->80/tcp, :::8082->80/tcp app4nginx2
dam2@jupiter06:~/ruben/docker06web$
```



Proyecto: docker06b Autor: Ruben Fecha: 22/01/24

5 Docker Hub

Ahora vamos a crear un contenedor "holamundo" y subirlo a Docker Hub.

5.1 Creamos los ficheros necesarios

Crear nuestra imagen "holamundo":

- Crear carpeta dockerXXpush. Entrar en la carpeta.
- Crear un script (holamundoXX.sh) con lo siguiente:

```
#!/bin/sh
echo "Hola Mundo!"
echo "nombre-del-alumnoXX"
echo "Proyecto
dockerXXpush" date
```



```
GNU nano 4.8 holamundo06.s
#!/bin/sh
echo "Hola Mundo!"
echo "nombre-del-alumnoXX"
echo "Proyecto dockerXXpush" date
```

Este script muestra varios mensajes por pantalla al ejecutarse.

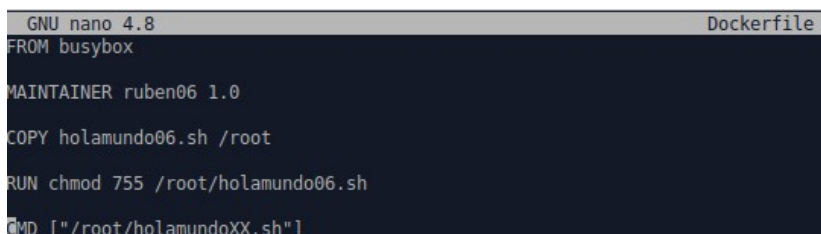
- Crear fichero

```
Dockerfile FROM busybox
MAINTAINER nombre-del-alumnoXX 1.0
```

```
COPY holamundoXX.sh /root
RUN chmod 755
```

```
/root/holamundoXX.sh CMD
```

```
["/root/holamundoXX.sh"]
```



```
GNU nano 4.8 Dockerfile
FROM busybox
MAINTAINER ruben06 1.0
COPY holamundo06.sh /root
RUN chmod 755 /root/holamundo06.sh
CMD ["/root/holamundoXX.sh"]
```

- A partir del Dockerfile anterior crearemos la imagen nombre-alumno/holamundo.

```
dam2@jupiter06:~/ruben/docker06push$ docker build -t ruben/holamundo .
[+] Building 0.5s (8/8) FINISHED
=> [internal] load .dockerignore
=> => transferring context: 28
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 170B
=> [internal] load metadata for docker.io/library/busybox:latest
=> [1/3] FROM docker.io/library/busybox@sha256:6d9ac9237a84afe1516540f40a0fafdc86859b2141954b4d643af7066d598b74
=> [internal] load build context
=> => transferring context: 35B
=> CACHED [2/3] COPY holamundo06.sh /root
=> CACHED [3/3] RUN chmod 755 /root/holamundo06.sh
=> exporting to image
=> => exporting layers
=> => writing image sha256:f191a069d99425359227986fb9823052f17708e1989d0baf64eeb57d5a28b120
=> => naming to docker.io/ruben/holamundo
dam2@jupiter06:~/ruben/docker06push$
```

- Comprobar que docker run nombre-alumno/holamundo se crea un contenedor que ejecuta el script. Eliminar el contenedor si todo va bien.

```
dam2@jupiter06:~/ruben/docker06push$ docker run ruben/holamundo
Hola Mundo!
nombre-del-alumnoXX
Proyecto dockerXXpush date
dam2@jupiter06:~/ruben/docker06push$
```

5.2 Subir la imagen a Docker Hub

- Registrarse en Docker Hub.
- docker login -u USUARIO-DOCKER, para abrir la conexión.

```
dam2@jupiter06:~/ruben/docker06push$ docker login -u poybengs
Password:
WARNING! Your password will be stored unencrypted in /home/dam2/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
dam2@jupiter06:~/ruben/docker06push$
```

- docker tag nombre-alumno/holamundo:latest USUARIO-DOCKER/holamundo:version1, etiquetamos la imagen con "version1".

```
dam2@jupiter06:~/ruben/docker06push$ docker tag ruben/holamundo:latest poybengs/holamundo:version1
dam2@jupiter06:~/ruben/docker06push$
```

- docker push USUARIO-DOCKER/holamundo:version1, para subir la imagen (version1) a los repositorios de Docker.

```
dam2@jupiter06:~/ruben/docker06push$ docker push poybengs/holamundo:version1
The push refers to repository [docker.io/poybengs/holamundo]
986b8757f96b: Pushed
cdebabf01d9cb: Pushed
2e112031b4b9: Mounted from library/busybox
version1: digest: sha256:c0c5e1794fd260112a27c3d2e472594afd405156466a0b9a433fac95b521282d size: 942
dam2@jupiter06:~/ruben/docker06push$
```

5.3 Analizar y entender

Analizar y entender el siguiente Dockerfile (<https://github.com/pausoft/docker>)

- Explicar qué utilidad tiene y cómo lo hace.

Este Dockerfile está diseñado para crear una imagen de Docker que permite la descarga y reproducción de videos de YouTube, específicamente en formato de audio (mp3). A continuación, explico cada una de las instrucciones del Dockerfile y su propósito:

1. FROM library/debian:9:
 - Esta línea establece la imagen base para el Dockerfile, que en este caso es Debian versión 9. Es un sistema operativo ligero y estable, adecuado para muchas aplicaciones.
2. LABEL maintainer="pau.tome@iescarlesvallbona.cat":
 - Define metadatos para la imagen. Aquí se está especificando el mantenedor del Dockerfile.
3. RUN apt-get -y update && apt-get install -y python ffmpeg curl mplayer:
 - Actualiza la lista de paquetes y luego instala Python, FFmpeg, curl y mplayer:
 - **Python**: Es necesario para ejecutar scripts, probablemente el script temazo.sh.
 - **FFmpeg**: Es una suite de software libre para la grabación, conversión y transmisión de audio y video.
 - **curl**: Una herramienta para transferir datos desde o hacia un servidor.
 - **mplayer**: Un reproductor multimedia.
4. RUN curl -L https://yt-dl.org/downloads/latest/youtube-dl -o /usr/local/bin/youtube-dl && chmod a+rx /usr/local/bin/youtube-dl && mkdir /mp3:
 - Descarga la última versión de youtube-dl, una herramienta de línea de comandos para descargar videos de YouTube y otros sitios, lo instala en /usr/local/bin, cambia los permisos para que sea ejecutable, y crea un directorio /mp3.
5. COPY temazo.sh /usr/local/temazo.sh:
 - Copia el script temazo.sh desde el contexto de construcción local al contenedor en /usr/local/temazo.sh.
6. RUN chmod a+rx /usr/local/temazo.sh:
 - Cambia los permisos del script temazo.sh para hacerlo ejecutable.
7. ENTRYPOINT ["/usr/local/temazo.sh"]:
 - Define el comando que se ejecutará al iniciar el contenedor. En este caso, se ejecutará el script temazo.sh.
8. CMD ["https://www.youtube.com/watch?v=e4Ao-iNPPUc"]:
 - Proporciona un valor por defecto para temazo.sh. Parece ser un enlace a un video de YouTube. Este valor puede ser sobrescrito al iniciar el contenedor con un comando diferente.

En resumen, este Dockerfile crea una imagen que puede usarse para descargar y reproducir el audio de videos de YouTube, utilizando youtube-dl para la descarga y mplayer para la reproducción. El comportamiento específico depende del contenido del script temazo.sh, que no está incluido aquí.

6 Limpiar contenedores e imágenes

Cuando terminamos con los contenedores, y ya no lo necesitamos, es buena idea pararlos y/o destruirlos.

- `docker ps -a`, identificar todos los contenedores que tenemos.
- `docker stop ...`, parar todos los contenedores.
- `docker rm ...`, eliminar los contenedores.

```
dam2@jupiter06:~$ docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED
a01c1729c09e   ruben/holamundo  "/root/holamundo06.sh"  13 minutes ago
43568b17b50f   ruben/nginx3    "/docker-entrypoint...." 46 hours ago
03cdf5e7eb28   ruben/nginx2    "/usr/sbin/nginx -g ..." 47 hours ago
bcbe7c88814c   ruben/nginx1    "/bin/bash"              47 hours ago
da7a264e9821   mongo:latest    "docker-entrypoint.s..." 6 days ago
b7dd0508d6fd   gvenzl/oracle-xe "container-entrypoin..." 11 months ago
dam2@jupiter06:~$ docker rm a01c1729c09e
a01c1729c09e
dam2@jupiter06:~$ docker rm 43568b17b50f
43568b17b50f
dam2@jupiter06:~$ docker rm 03cdf5e7eb28
03cdf5e7eb28
dam2@jupiter06:~$ docker rm bcbe7c88814c
bcbe7c88814c
dam2@jupiter06:~$ docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED
da7a264e9821   mongo:latest    "docker-entrypoint.s..." 6 days ago
b7dd0508d6fd   gvenzl/oracle-xe "container-entrypoin..." 11 months ago
dam2@jupiter06:~$
```

Hacemos lo mismo con las imágenes. Como ya no las necesitamos las eliminamos:

- `docker images`, identificar todas las imágenes.
- `docker rmi ...`, eliminar las imágenes.

```
dam2@jupiter06:~$ docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
poybengs/holamundo  version1  f191a069d994  18 minutes ago  4.26MB
ruben/holamundo    latest   f191a069d994  18 minutes ago  4.26MB
ruben/nginx3      latest   ee7095090674  47 hours ago   187MB
ruben/nginx2      latest   c9a8a154c026  47 hours ago   155MB
ruben/nginx1      latest   3a362579e19d  47 hours ago   156MB
debian            latest   a6916e41aa87  13 days ago    117MB
mongo             latest   e325fe350a8c  2 weeks ago    757MB
hello-world       latest   d2c94e258dcb  8 months ago   13.3kB
gvenzl/oracle-xe   latest   eccf57eac1b8  11 months ago  3.18GB
mongo             <none>   a440572ac3c1  11 months ago  639MB
dam2@jupiter06:~$ docker rmi f191a069d994 f191a069d994 ee7095090674 c9a8a154c026 3a362579e19d a6916e41aa87
Untagged: ruben/nginx3:latest
Deleted: sha256:ee7095090674c5a035aea6085adda39b0b5a2174b32ae70be3953c3b6b07d92a
Untagged: ruben/nginx2:latest
Deleted: sha256:c9a8a154c0260cc97295da643987bc51b1ee208bac6aefc5f2cdc453494ead8f
Untagged: ruben/nginx1:latest
Deleted: sha256:3a362579e19d0daa83d83a716c687945f373c5efa237db435e9cacf131552b81
Deleted: sha256:91ce500055386442743df1ce98609c132d36849e468fa4dd8bac7b28c1088405
Untagged: debian:latest
Untagged: debian@sha256:b16cef8cbcb20935c0f052e37fc3d38dc92bfec0bcfb894c328547f81e932d67
Deleted: sha256:a6916e41aa871b09260f8949fe3c34570719f1c92ffe87d515b7eb702ebaebb1
Error response from daemon: conflict: unable to delete f191a069d994 (must be forced) - image is referenced
Error response from daemon: conflict: unable to delete f191a069d994 (must be forced) - image is referenced
dam2@jupiter06:~$ docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
poybengs/holamundo  version1  f191a069d994  20 minutes ago  4.26MB
ruben/holamundo    latest   f191a069d994  20 minutes ago  4.26MB
mongo             latest   e325fe350a8c  2 weeks ago    757MB
hello-world       latest   d2c94e258dcb  8 months ago   13.3kB
gvenzl/oracle-xe   latest   eccf57eac1b8  11 months ago  3.18GB
mongo             <none>   a440572ac3c1  11 months ago  639MB
```