Los contenedores

1. Principio

El aislamiento de uno o distintos procesos lo realiza el núcleo Linux usando para ello los espacios de nombres y los grupos de control. Los espacios de nombres aparecieron en 2002 y los grupos de control en 2007. La llegada de los espacios de nombres al núcleo 3.8 en febrero de 2013 abrió la vía al soporte completo de los contenedores.

Todo núcleo Linux integra las funciones por defecto:

Los namespaces, o espacios de nombres, permiten:

- el aislamiento de los procesos y tener un proceso con PID1 como primer proceso en su espacio (PID 1 del primer proceso de un espacio de nombres),
- el aislamiento de la red, con su propia dirección IP y sus propios puertos,
- el aislamiento de los volúmenes de datos, de almacenamiento,
- el aislamiento de los derechos y de los usuarios con respecto al anfitrión: ser root en un contenedor puede corresponder a un usuario sin ese derecho en el anfitrión.

Los cgroups, o grupos de control, son los recursos del sistema que un grupo de procesos puede utilizar, con el objetivo de:

- limitar los recursos de memoria o de procesador de un grupo,
- administrar las prioridades,
- obtener información «contable» sobre el grupo,
- controlar en su conjunto (por ejemplo, hacer que se paren) algunos procesos,
- aislar el grupo, asociándolo a un espacio de nombres,

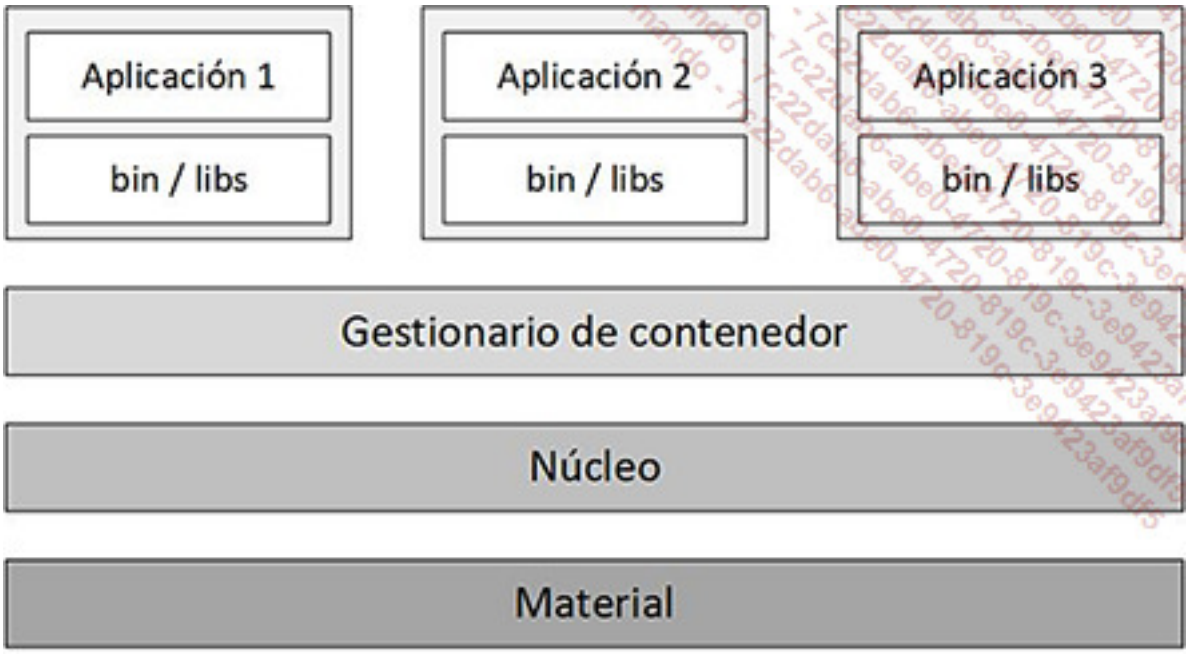
La asociación de esos dos mecanismos es la base del principio del contenedor.

2. Contenedor y Máquina virtual

A veces se comparan los contenedores con las máquinas virtuales, sin embargo se trata de dos tipos de tecnología de virtualización diferentes.

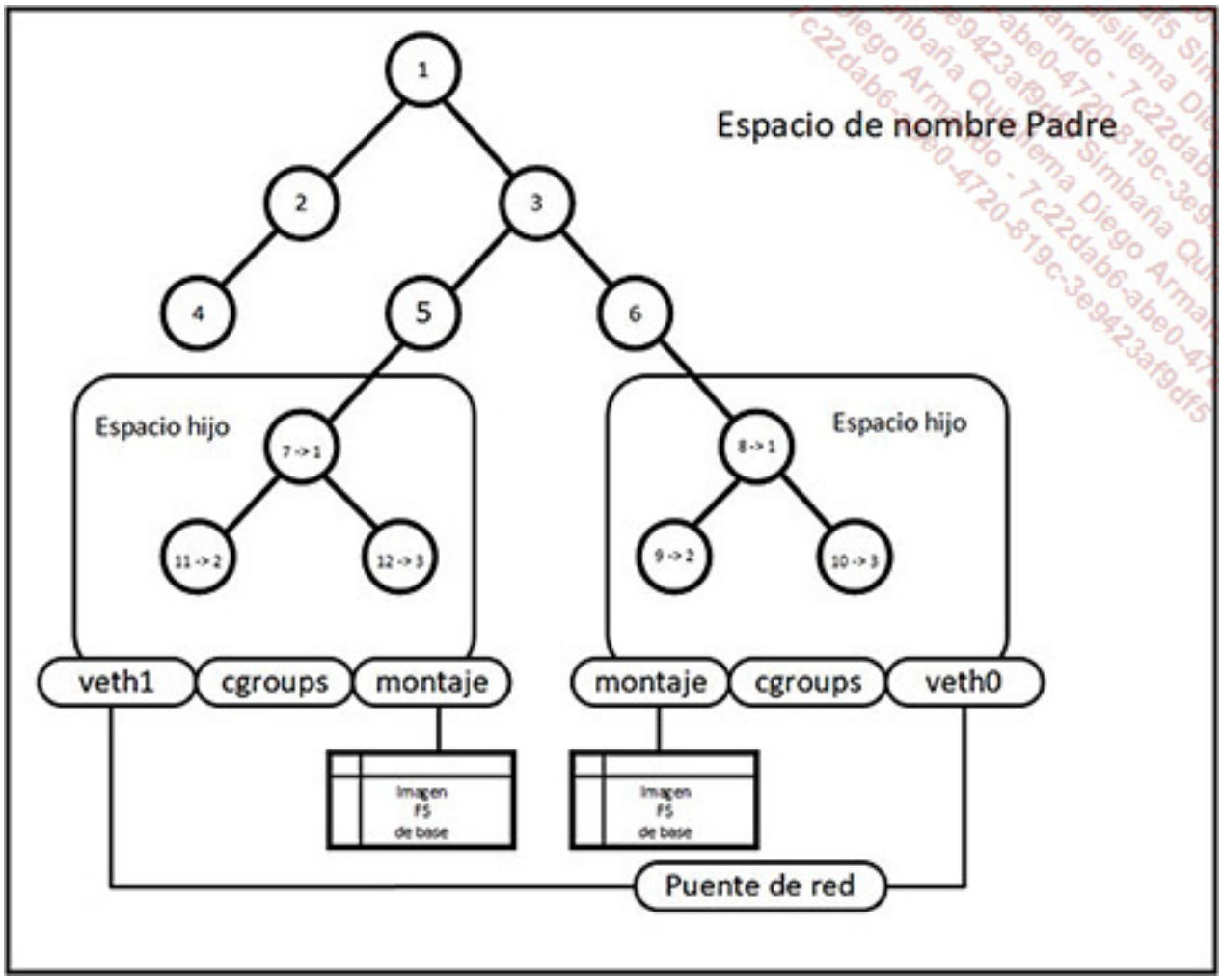
Las máquinas virtuales funcionan en hipervisores. Ya sea el material emulado o no, las máquinas virtuales ejecutan un sistema operativo completo, con núcleo y drivers para los periféricos (incluso los que usan la paravirtualización). Además, las aplicaciones se instalan en el sistema operativo de la máquina virtual. Desde el punto de vista del hipervisor, la máquina virtual es un proceso único.

Los contenedores funcionan directamente en el anfitrión, directamente en el núcleo. Los procesos que este contiene son iguales que los otros, solamente aislados y limitados en recursos. Si un contenedor tiene 10 procesos, habrá 10 procesos funcionando en el anfitrión.



El contenedor usa el núcleo del anfitrión

3. Los espacios de nombres



Contenedores: espacios de nombres y grupos de control

El primer proceso de un espacio de nombres lleva el PID número 1. Todos los procesos hijos forman parte automáticamente de este espacio de nombres y se encuentran aislados de los otros espacios de nombres. No ven lo que sucede fuera de su espacio: se trata del «PID namespace».

Dentro de un espacio de nombres los procesos se ven como procesos ordinarios, con PIDs «normales». De esta manera, el proceso PID 1 de un espacio podría poseer el PID 12345 desde el punto de vista del anfitrión.

Los mecanismos de espacio de nombres también permiten modificar la raíz «rootfs» del proceso, como si fuera un chroot. Se instala todo lo necesario para el funcionamiento de los procesos en un repertorio, y después se reemplaza el sistema de archivos raíz «/» del espacio de nombres con ese repertorio: se trata del «mount namespace».

Cada espacio de nombres puede disponer de una interfaz virtual. Esta a veces está puenteada en una interfaz red del anfitrión, o sobre otra interfaz gestionada por otro servicio específico. La interfaz virtual recibe su propia dirección IP y asegura la comunicación entre el espacio de nombres y el exterior: se trata del «network namespace».

Por defecto, los procesos de un espacio de nombres comparan las cuentas de los usuarios y los grupos con los del sistema. Sin embargo se pueden aislar, disociar. También se pueden asociar a otras cuentas de usuario. De esta manera, es posible asignar un UID idéntico a cuentas de usuarios diferentes, o incluso tener una cuenta root dentro del espacio que será asociado a un usuario sin privilegios o fuera de este: se trata del «user namespace».

Cada espacio de nombres se puede asociar a un grupo de control «cgroup» con el objetivo de limitar los recursos de los procesos del espacio: se trata del «cgroup namespace».

4. Los grupos de control

Poner procesos dentro de un grupo de control permite controlar los recursos que puede utilizar. Se puede, así, limitar la cantidad de memoria utilizable, por ejemplo 256 MB, y la cantidad de tiempo de procesador asignada, exprimida en cores o milicores, como por ejemplo 500 milicores, para 0,5 cores. Estos límites se aplican al conjunto del espacio de nombres, y no a cada proceso.

Los grupos de control permiten, no solamente limitar el uso de los recursos de un grupo de procesos, sino también evitar que se acaparen todos los recursos del sistema, lo que presenta el riesgo de canibalizar las otras aplicaciones e incluso el propio sistema. De esta manera, si un proceso del grupo intenta usar memoria que no le ha sido asignada, será eliminado por el núcleo Linux del anfitrión.

5. Montaje en unión

El montaje en **unión** «Union mount» reúne distintos sistemas de archivos en uno solo, generalmente una pila ordenada de directorios en un único punto de montaje, creando así

un sistema de archivos virtual, fruto de la unión de todos los otros. Cada capa de la pila añade o suprime archivos. Los sistemas de archivos más conocidos son AUFS y Overlay. Un sistema de archivos de ese tipo se puede implementar usando LVM o las instantáneas « snapshots » propuestas por algunos sistemas de archivos como BTRFS.

Este tipo de sistema de archivos representa una ventaja muy importante cuando está montado como sistema de archivos raíz de un contenedor: todas las capas están en modo lectura, excepto la última, un directorio que está vacío de forma voluntaria y que se encuentra en modo de escritura. El contenedor puede escribir en el punto de montaje sin modificar el sistema de archivos de base. Cuando se destruye el contenedor, las modificaciones se pierden por la destrucción de la última capa.

6. Imagen aplicativa

Para arrancar una aplicación dentro de un contenedor, todas las dependencias tienen que haber sido instaladas en el sistema de archivos de raíz, ya que dicha aplicación está totalmente aislada y no puede acceder a los archivos que se encuentran fuera de su espacio. Un método consiste en copiar en un repertorio una arborescencia Linux estándar que contenga los archivos (binarios, bibliotecas y archivos de configuración) de base, las dependencias de la aplicación que se ejecutarán en el contenedor y la misma aplicación.

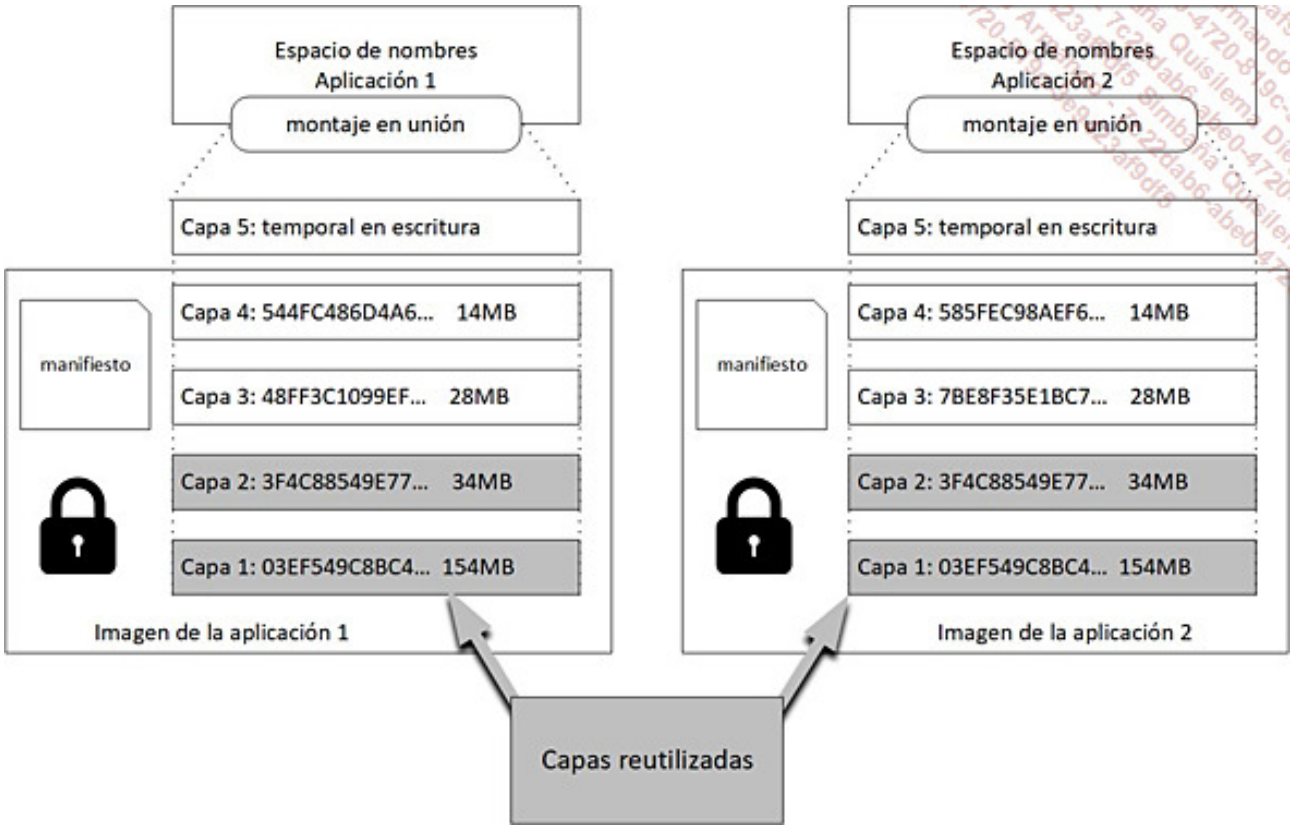
La arborescencia Linux puede ser la de cualquier distribución.

Se puede crear una copia de esta arborescencia para hacer una imagen de base reutilizable que se puede almacenar, replicar y distribuir. Podremos, de esta manera, crear catálogos, repositorios, derivados de versiones diferentes dependiendo de lo que necesitemos.

Podemos montar esta arborescencia en sólo lectura uniéndolo con un directorio vacío en escritura y utilizarla como sistema de archivos raíz en el espacio de nombres.

Crear un contenedor, es instanciar una imagen de sistema de archivos y arrancar uno o distintos procesos, dentro de un espacio de nombres.

7. Las capas de imágenes



Imágenes en capas y montaje en unión

Cuando se construye una imagen siempre se empieza desde un mínimo utilizable.

Encima de esta primera capa, se pueden añadir las actualizaciones de los paquetes e instalar después las dependencias de la aplicación que queremos arrancar, y después la aplicación en sí.

En cada etapa, se añade una capa (layer) suplementaria:

- Capa 1: imagen de base
- Capa 2: actualizaciones
- Capa 3: dependencias de la aplicación
- Capa 4: la aplicación

Para añadir una capa encima de la anterior, se montan en unión las capas anteriores en lectura y un nuevo directorio en escritura será creado encima que se utilizará como una nueva capa. Cuando esta nueva capa se modifique, se archivará. Si queremos crear una nueva imagen para una nueva aplicación, no es necesario volver a crear las capas 1 y 2

porque ya existen. La construcción de la nueva imagen se hará a partir de estas dos capas.

Se archivará cada una de las capas. Una imagen es un apilamiento de capas sucesivas y reutilizables, montadas en unión. Un manifiesto no modificable describe la lista ordenada. Este manifiesto ya no es modificable, la imagen es atómica.

Para disponer de una imagen utilizable, recuperamos su manifiesto y cada una de las capas que están descomprimidas en su propio directorio. Esos directorios tienen que estar montados y unificados para que puedan ser usados como sistema de archivos raíz.

El tamaño de las diferentes capas debería de ser lo más pequeña posible, con el fin de disminuir su huella en el anfitrión, la red, la velocidad de tratamiento y la biblioteca.

En último lugar, el mecanismo de etiqueta o tag permite conservar diferentes versiones de una imagen. La imagen por defecto lleva la etiqueta «latest», es la que será recuperada si no se especifica la versión.

8. El proyecto OCI

El proyecto OCI, «*Open Container Initiative*», es un proyecto de la Fundación Linux en el que se publican estándares abiertos sobre los contenedores. Fue iniciado por Docker, es soportado por todos los grandes actores del mercado de los contenedores. Se han publicado dos especificaciones:

- Especificación imagen «image-spec»: es la definición de un formato de imagen estandardizado del sistema de archivos del contenedor y de los métodos para crearlos, almacenarlos y distribuirlos.
- Especificación runtime «runtime-spec»: es la definición de la configuración, del entorno de ejecución y del ciclo de vida de un contenedor.

9. Docker

Linux propone la posibilidad de crear contenedores a través de su núcleo y usando

distintos comandos, pero se trata de una tarea compleja. Lo ideal es disponer de una solución.

Docker apareció en marzo de 2013, fruto del trabajo de Solomon Hykes. Creó en 2008 la empresa dotCloud, cuyas oficinas se encuentran en Montrouge (Francia), apostando por el Cloud y el PaaS. En 2010 esta empresa empezó a trabajar con el proyecto Docker. Los inversores franceses no entendieron el potencial del proyecto y dotCloud se mudó en 2011 a la Silicon Valley. Solomon liberó el código en 2013, cambió el nombre de la empresa y del producto y delegó el puesto de director general a otra persona, para poder dedicarse al puesto de director técnico.

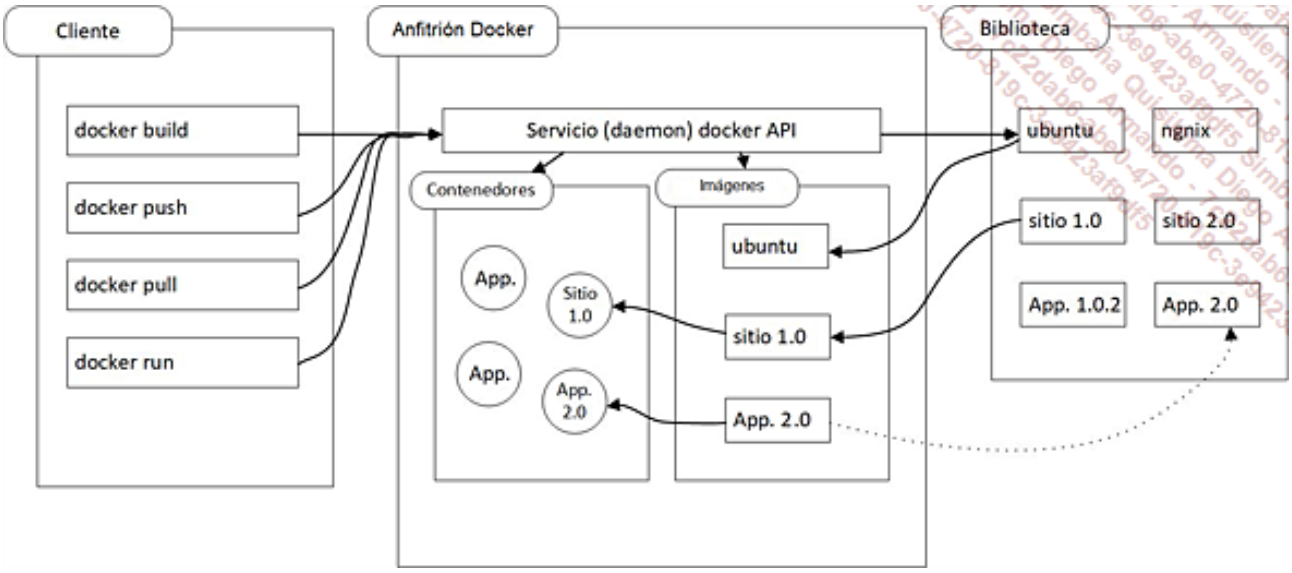
Docker suscitó el interés tanto de programadores y administradores como de grandes empresas. El éxito y la adopción del producto fueron inmediatos.

Docker dispone de tres grandes funcionalidades:

- La creación facilitada de imágenes aplicativas gracias a un archivo descriptivo simple, basado en un número reducido pero eficaz de comandos. Se trata del archivo Dockerfile.
- La gestión de una biblioteca (repository) de diferentes versiones de imágenes basada en una API REST. Esta biblioteca permite disponer de un conjunto de imágenes aplicativas reutilizables. El Docker Hub propone un catálogo de más de 10000 imágenes de libre acceso.
- Gestionar el ciclo de vida de un contenedor, desde su arranque hasta su destrucción. Docker administra simplemente los espacios de nombres y los grupos de control.

Para todo esto se utiliza un comando único con una sintaxis simple: docker.

La fuerza de Docker es haber hecho simple y evidente el uso de una tecnología compleja. El contenedor se considera una aplicación clásica, que puede ser construido, almacenado, arrancado, expuesto, controlado, parado y suprimido.



Los principios básicos de Docker

Docker ha sabido escuchar los consejos de su comunidad de usuarios, ha fundado y seguido las especificaciones OCI. Propone una API simple, y kits de desarrollo en diversos lenguajes como Go o Python, hecho que ha acelerado su difusión.

10. Un ejemplo completo

Aunque Docker se encuentra en la mayoría de las distribuciones, algunas de ellas han optado por ofrecer solo herramientas y comandos que vienen de OCI o desarrollados por ellos mismos como podman, buildah, skopeo o runc. Aquí se pierde la ventaja de la herramienta única y completa, pero se trata de herramientas ligeras.

Para descargar e instalar la versión que corresponda a su distribución o a su sistema operativo, tendremos que visitar el sitio web de Docker Community. Docker también está disponible en Windows y macOS: https://docs.docker.com/engine/install/

a. Crear una imagen

Vamos a crear una imagen que arrancará un servidor web nginx y mostrará una página estática hecha por nosotros. Para ello, podemos ir a Docker Hub, sitio que proporciona una lista de imágenes preparadas, a veces puestas a disposición por el propio editor en la

dirección https://hub.docker.com/_/nginx

➔ Cree una arborescencia como la siguiente:

```
mkdir -p Docker/html
```

➔ En el directorio Docker/html, cree un archivo index.html con las líneas siguientes:

```
<html>
<head>
 <title>My Page</title>
</head>
<body>
 <h1>My title</h1>
Welcome to our test page
</body>
</html>
```

➔ En el directorio Docker, cree el archivo Dockerfile con las directivas siguientes:

```
FROM nginx
COPY html /usr/share/nginx/html
```

Este archivo contiene dos instrucciones:

- Recuperar la imagen nginx desde Docker Hub. Se trata de una imagen que ya existe y que contiene todo lo necesario para hacer funcionar una versión de base de nginx (dependencias e instrucciones de arranque).
- Copie un sitio web estático en el directorio del futuro contenedor.

➔ Ejecute la construcción de la imagen llamada my-nginx:

```
$ docker build -t my-nginx .
Sending build context to Docker daemon   12.8kB
Step 1/2 : FROM nginx
latest: Pulling from library/nginx
```

```
8ec398bc0356: Pull complete
465560073b6f: Pull complete
f473f9fd0a8c: Pull complete
Digest: sha256:b2d89d0a210398b4d1120b3e3a7672c16a4ba09c2c4a0395f18b9f7999b768f2
Status: Downloaded newer image for nginx:latest
---> f7bb5701a33c
Step 2/2 : COPY html /usr/share/nginx/html
---> c24885ab201e
Successfully built c24885ab201e
Successfully tagged my-nginx:latest
```

Haga un listado de sus imágenes, encontrará dos: la imagen nginx de base, de Docker Hub, y la suya:

```
$ docker images
REPOSITORY      TAG       IMAGE ID       CREATED             SIZE
my-nginx        latest    c24885ab201e   About a minute ago  126MB
nginx           latest    f7bb5701a      5 days ago          126MB
```

b. Iniciar un contenedor

Para iniciar un contenedor:

```
$ docker run --name test1 -d my-nginx
70f74bf4b9522ba77f344163a215844d0f40f911acf734c30ddb155a8fd69250
```

Para saber si funciona:

```
$ docker ps
CONTAINER ID      IMAGE             COMMAND
CREATED           STATUS            PORTS             NAMES
70f74bf4b952      my-nginx          "nginx -g 'daemon of...
50 seconds ago    Up 49 seconds     80/tcp            test1
```

¿Cómo se puede acceder? El comando **docker ps** le muestra que el contenedor escucha en el puerto 80. Pero cuidado, se trata del puerto 80 del mismo contenedor, ¡no del anfitrión desde el que lo ha iniciado! Hay que vincular el puerto de red del anfitrión al puerto del contenedor para publicarlo desde el anfitrión.

c. Paro del contenedor

→ Empezaremos forzando el paro y la destrucción del contenedor actual usando su identificador único:

```
$ docker stop 70f74bf4b952 && docker rm 70f74bf4b952
70f74bf4b952
70f74bf4b952
```

→ O más simplemente:

```
$ docker rm -f 70f74bf4b952
```

d. Publicación del contenedor

→ Solicitaremos a Docker que arranque el contenedor asociando el puerto local 8080 del anfitrión al puerto 80 del contenedor, así:

```
$ docker run --name test1 -p 8080:80 -d my-nginx
9ee19cc937f287a592ec22940051cda562e0ade309c0c03f65e8037ab387c5bb
```

El comando **docker ps** nos arroja esta salida:

```
0.0.0.0:8080->80/tcp
```

→ Desde un navegador, o con un comando **curl**, intente acceder al servidor nginx en el puerto 8080:

```
$ curl  http://127.0.0.1:8080
<html>
<head>
 <title>My Page</title>
</head>
<body>
 <h1>My title</h1>
Welcome to our test page
</body>
</html>
```

e. Dinamismo

Al crear nuestra imagen, sobrescribimos la arborescencia HTML de manera estática. Pero podemos pedir a Docker que monte cualquier tipo de arborescencia dentro de un contenedor.

→ Destruya y vuelva a ejecutar de nuevo el contenedor, pero especificando el directorio del anfitrión:

```
$ docker run --name test1 -p 8080:80 -v
/Users/seb/Docker/html:/usr/share/
nginx/html:ro -d my-nginx
```

→ Ahora, modifique el contenido del archivo index.html:

```
<body>
 <h1>My title</h1>
Welcome to our test page, with a modification
</body>
```

Si usa de nuevo el comando **curl** obtendrá el nuevo texto.

De esta manera, imagine ahora lo siguiente:

- Puede arrancar tantos servidores web (nginx, apache, u otro) como quiera, bajo

la forma de un contenedor.

- Actualizar un servidor web, se trata solamente de reemplazar una imagen con otra.
- Puede ejecutar varias versiones de un servidor web de manera simultánea, sin generar conflictos.
- Puede crear bibliotecas locales o remotas de sus imágenes.
- Sus imágenes funcionarán en cualquier anfitrión.

f. Acceder al contenedor

La mayoría de los contenedores presenta un comando shell dentro de la imagen que les permite acceder como cualquier sistema Linux. Aquí la imagen no presenta el comando ps, utilizaremos un truco para listar los procesos:

```
docker exec -ti 203e16732619 bash
root@203e16732619:/# ls -d /proc/[1-9]*
/proc/1 /proc/7 /proc/8
root@203e16732619:~# cat /proc/1/comm
nginx
```

El proceso de PID 1 del contenedor es el comando **nginx**.

El Dockerfile de la imagen nginx de base está aquí: https://github.com/dockerfile/nginx/blob/master/Dockerfile

Observe una línea CMD que define qué comando se ejecuta cuando se instancia la imagen, es decir, cuando se arranca el contenedor:

```
CMD ["nginx"]
```

g. Registros

Los registros son las salidas de STDOUT o STDERR por los procesos del contenedor.

➜ Se accede así:

```
$ docker logs 203e16732619
172.17.0.1 - - [02/Jan/2021:15:48:49 +0000] "GET / HTTP/1.1" 200 136 "-" "curl/7.64.1" "-"
```

h. Suprimir el contenedor y la imagen

Se usan los comandos **stop**, **rm** y **rmi**:

```
$ docker stop 203e16732619
$ docker rm 203e16732619
$ docker rmi f7bb5701a33c c24885ab201e
Untagged: my-nginx:latest
Deleted: sha256:c24885ab201ee4ae9abbe9ea7e2808465033f38c9d267f3f8efce32769371075
Deleted: sha256:6089ed9f16f30b3fe9b08e66321194bd3d4c246c728601375c05707c9862c9fa
Untagged: nginx:latest
Untagged: nginx@sha256:b2d89d0a210398b4d1120b3e3a7672c16a4ba09c2c4a0395f18b9f7999b768f2
Deleted: sha256:f7bb5701a33c0e572ed06ca554edca1bee96cbbc1f76f3b01c985de7e19d0657
Deleted: sha256:8f1984cf0de0461043fcabd6b2a2040325ac001d9897a242e0d80486fe71575e
Deleted: sha256:59d91a36ba4b720eadfbf346ddb825c2faa2d66cc7a915238eaf926c4b4b40ee
Deleted: sha256:556c5fb0d91b726083a8ce42e2faaed99f11bc68d3f70e2c7bbce87e7e0b3e10
```

11. Seguridad

Desde el anfitrión, el acceso a los contenedores está permitido si se es miembro del grupo docker, y se prohíbe el acceso a los demás. Docker trabaja usando un socket. Cualquiera que tenga acceso a este podrá interactuar con los contenedores.

No se puede acceder al anfitrión, ni a los demás contenedores, desde un contenedor, excepto si se arranca un contenedor en modo privilegiado y se monta el sistema de archivos del anfitrión dentro de este. Se encuentran aislados del anfitrión y entre ellos mismos.

Por defecto, todos los procesos dentro de un contenedor se ejecutan con los UID y GID 0, es decir root. Esto no representa ningún problema, mientras no se encuentre en el caso

anterior. Sin embargo, a parte de algunos casos puntuales, no existe ninguna razón de peso para ejecutar una aplicación como root dentro de un contenedor. Debería, al crear las imágenes, usar un usuario anónimo para ejecutar sus aplicaciones y tener acceso a los archivos.

Aunque sea posible, no publique los contenedores directamente en la dirección IP del anfitrión. Utilice más bien puertos específicos o un proxy.

De manera general, las buenas prácticas están expuestas aquí: https://docs.docker.com/develop/develop-images/dockerfile_best-practices/