



## UT 011. DOCKER

Sistemas informáticos  
CFGS DAW

Álvaro

Maceda

[a.macedaarranz@edu.gva.es](mailto:a.macedaarranz@edu.gva.es)

2022/2023

Versión:230309.1335

## Licencia



**Atribución - No comercial - CompartirIgual**  
**(por-nc-sa):** No se permite el uso comercial de la obra original ni de ninguna obra derivada. cuya distribución debe realizarse bajo una licencia igual a la que rige la obra original.

## Nomenclatura

A lo largo de esta unidad se utilizarán diferentes símbolos para distinguir elementos importantes dentro del contenido. Estos símbolos son:



Importante



Atención



Interesante

## ÍNDICE

<b>5. Trabajar con imágenes.....</b>	<b>4</b>
5.1 ¿Qué es una imagen Docker?.....	4
5.2 Docker Hub.....	6
5.3 Creación de imágenes .....	8
<b>6. Red.....</b>	<b>11</b>
6.1 Puente por defecto .....	11
6.2 Redes de puentes personalizadas.....	13
<b>7. Material complementario .....</b>	<b>14</b>

## UT 011. DOCKER

### 5. TRABAJAR CON LA IMAGEN S

#### 5.1 Qué es una imagen Docker

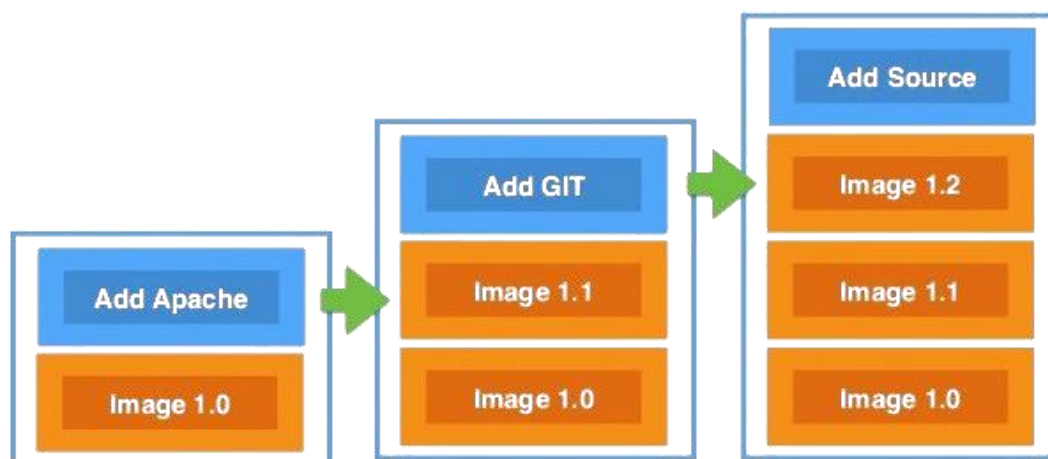
La imagen Docker es como el "disco duro" a partir del cual se crean los contenedores. Está hecha de capas, cada capa representa los cambios en el sistema de archivos de las capas anteriores. **Llamamos imagen a cada una de esas capas**, por lo que una imagen está "compuesta" de otras imágenes.

Por ejemplo, cuando descargamos una imagen:

```
$ docker pull wernight/funbox

Uso de la etiqueta por defecto: latest
último: Pulling from wernight/funbox
f2b6b4884fc8: Pull completo
24876304c826: Pull completo
dc2853569c8e: Pull completo
f1feacc76ece: Pull completo
47b0568134ef: Pull complete
```

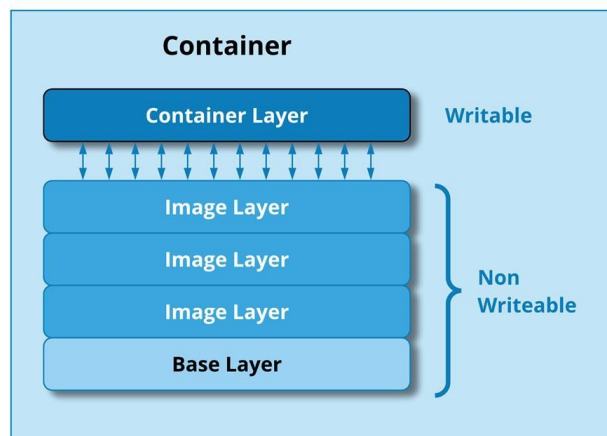
Puede ver que varias capas se descargan de forma independiente. Eso es porque, al crear imágenes, cada vez que se modifica algo se añade una nueva capa. Por ejemplo, si queremos una imagen con Apache, Git y algo de código fuente, el proceso sería más o menos así:



Cómo se crean las imágenes y las capas

Los recuadros azules son la nueva información que añadimos sobre la capa anterior. En el primer paso, instalamos Apache sobre el contenido de la Imagen 1.0: esos cambios formarán la capa "Imagen 1.1". Después, creamos la capa "Imagen 1.2" instalando git, y así sucesivamente.

Cuando un contenedor está en funcionamiento, la última capa es la única en la que se puede escribir. Esa capa se destruye cuando destruimos el contenedor.



Capas de imagen en un contenedor Docker

### 5.1.1 Etiquetas de imagen

Las imágenes se identifican con un ID (el número hexadecimal que puedes ver a la izquierda del ejemplo del docker pull). Siempre podemos referenciar una imagen por su ID, pero es más cómodo referenciar una imagen por su etiqueta. Normalmente, sólo se etiqueta la imagen superior.

Una etiqueta es una cadena compuesta por un **nombre** y una **etiqueta**. Por ejemplo, si lista las imágenes que ya tiene en su máquina con `docker image ls`:

REPOSITORIO	TAG	ID IMAGEN	CREADO	TAMAÑO
alpine	última	b2aa39c304c2	2 semanas hace	7.05MB
nginx	última	9eee96112def	3 semanas hace	142 MB
ubuntu	última	58db3edaf2be	4 semanas hace	77,8MB
ubuntu	20.04	e40cf56b4be3	4 semanas hace	72,8 MB
hola-mundo	última	feb5d9fea6a5	hace 17 meses	13,3kB
wernight/funbox	última	538c146646c3	hace 4 años	1,12 GB

La columna REPOSITORIO es el nombre, y la columna ETIQUETA es la versión. Hay una etiqueta especial, `latest`, que se utiliza para identificar la última versión disponible de una imagen. En el ejemplo anterior, se puede ver que tenemos dos imágenes diferentes para ubuntu: la imagen etiquetada con la versión 20.04 y la última versión disponible.

Una etiqueta es sólo un alias para el ID de la imagen, por lo que puede tener diferentes etiquetas apuntando al mismo ID de imagen.

No todas las imágenes están etiquetadas. Las imágenes intermedias no suelen estar etiquetadas.

Puede verlas con

`docker image ls --all:`

REPOSITORIO	TAG	ID IMAGEN	CREADO	TAMAÑO
alpine	última	b2aa39c304c2	2 semanas hace	7.05MB
ubuntu	20.04	e40cf56b4be3	4 semanas hace	72,8 MB
ubuntu	última	58db3edaf2be	4 semanas hace	77,8MB
<ninguno>	<ninguno>	86105d084ebb	3 hace meses	84,2 MB
<ninguno>	<ninguno>	9419c58a6e1e	3 hace meses	84,2 MB
<ninguno>	<ninguno>	116e02c4b147	3 hace meses	76,1MB

<ninguno>	<ninguno>	61968c3a4100	3 hace meses	73,9MB
mariadb	última	6e0162b44a5f	hace 11 meses	414 MB
hola-mundo	última	feb5d9fea6a5	hace 17 meses	13,3kB
wernight/funbox	última	538c146646c3	hace 4 años	1,12 GB

### 5.1.2 Eliminar imágenes

Cuando ya no necesitemos una imagen, podemos eliminarla con `docker image rm <tag o ID>`.

A menudo, queremos eliminar las imágenes que no son necesarias para liberar espacio en nuestro disco duro. Para ello, necesitamos saber la diferencia entre imágenes no utilizadas y colgantes:

- **Imágenes no utilizadas:** Imágenes que no son utilizadas por ningún contenedor. Tenga en cuenta que sólo la última imagen (la última capa) es la utilizada por un contenedor, pero las imágenes utilizadas por éste también se consideran utilizadas.
- **Imágenes colgantes:** Imágenes que no tienen etiqueta y no son necesarias para otra imagen.

Las imágenes colgantes se eliminan con `docker image prune`. Las imágenes no utilizadas y colgantes se eliminan con `docker image prune --all`.

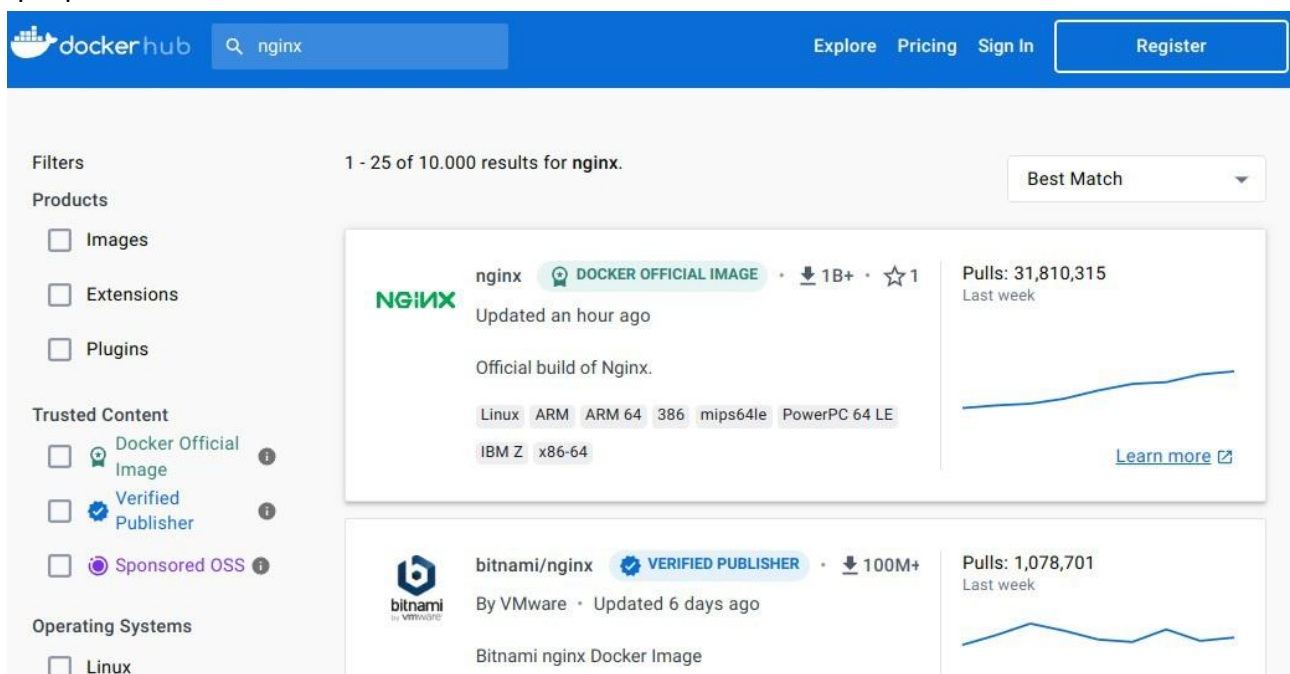
Normalmente es seguro eliminar imágenes porque siempre puedes descargarlas o generarlas de nuevo. Veremos cómo hacerlo dentro de un momento.

## 5.2 Docker Hub

La forma más sencilla de obtener una imagen para crear un contenedor es utilizar un Hub. Un Hub es un lugar en Internet con un directorio de imágenes. El Hub público más usado es Docker Hub (no necesitas registrarte para usar el sitio):

<https://hub.docker.com/>

Allí puede buscar imágenes por una palabra clave. Normalmente encontrará más de una imagen que preste ese servicio:



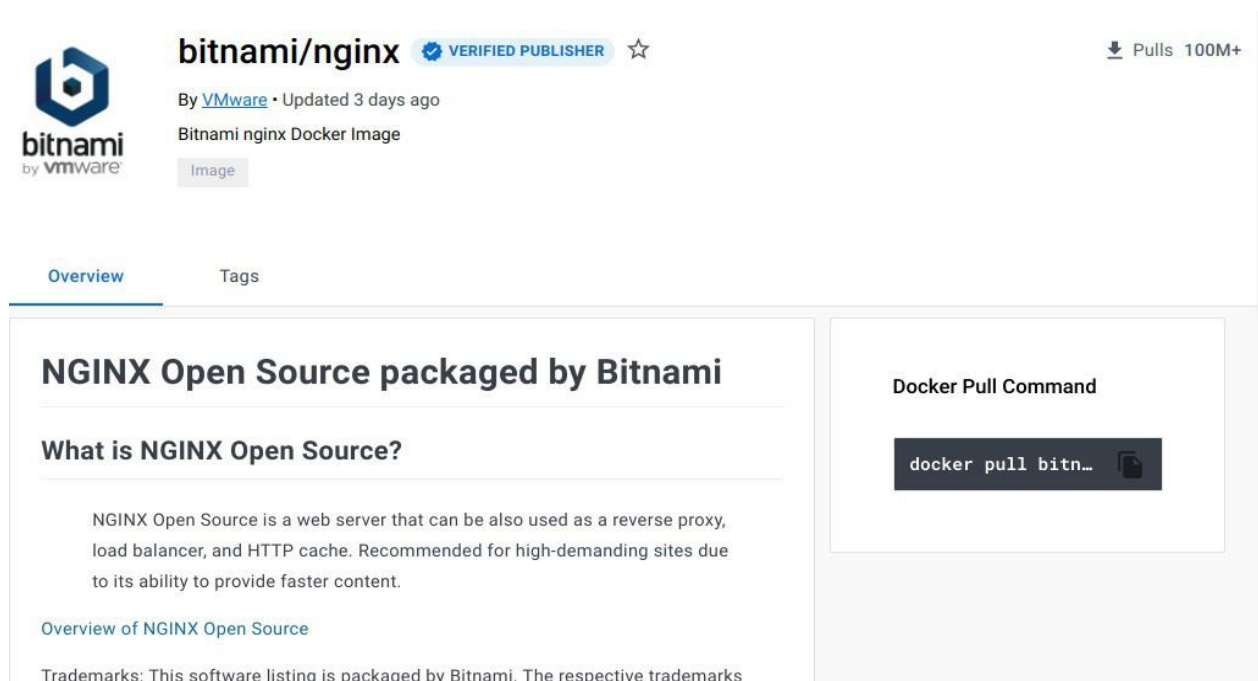
The screenshot shows the Docker Hub search results for the keyword 'nginx'. The interface includes a search bar at the top with 'nginx' entered, and navigation links for 'Explore', 'Pricing', 'Sign In', and 'Register'. On the left, there are filters for 'Products' (Images, Extensions, Plugins) and 'Trusted Content' (Docker Official Image, Verified Publisher, Sponsored OSS). The main area displays two search results. The first result is for 'nginx', marked as a 'DOCKER OFFICIAL IMAGE', with 1B+ downloads and 31,810,315 pulls. It includes a line graph showing pull trends over the last week. The second result is for 'bitnami/nginx', marked as a 'VERIFIED PUBLISHER', with 100M+ downloads and 1,078,701 pulls, also featuring a pull trend graph. The results are sorted by 'Best Match'.

Resultados de la búsqueda en Docker Hub

Cualquiera puede publicar una imagen en Docker Hub, pero las imágenes pueden contener malware o puertas traseras, por lo que son más o menos fiables dependiendo de quién las haya publicado. Según el publicador, las imágenes pueden clasificarse en:

- **Imágenes oficiales:** Esas imágenes son revisadas por un equipo dedicado de Docker. Tienen buena documentación y están bien soportadas. Son las más fiables.
- **Editor verificado:** El editor de la imagen ha sido verificado por Docker, por lo que puedes confiar en esas imágenes
- **Resto de imágenes:** Imágenes que no han sido verificadas. A menudo son inofensivas, pero debe utilizarlas con cuidado en su sistema de producción.

Una vez seleccionada una imagen, en su página puede encontrar información relevante como la forma en que debe utilizar la imagen. Si no sabes cómo utilizar una imagen, probablemente encontrarás esa información en esta página:



The screenshot shows the Docker Hub page for the 'bitnami/nginx' image. At the top, there's a Bitnami logo and the repository name 'bitnami/nginx' with a 'VERIFIED PUBLISHER' badge and a star icon. Below this, it says 'By VMware • Updated 3 days ago' and 'Bitnami nginx Docker Image'. The 'Overview' tab is selected, showing a description of NGINX Open Source packaged by Bitnami. A 'Docker Pull Command' box displays 'docker pull bitnami/nginx:latest'.

También tienes una pestaña "Etiquetas" con las diferentes etiquetas de esa imagen. Para usar una imagen debes referenciarla por su nombre y su etiqueta. Por ejemplo, si quieres usar la versión Debian de nginx 1.22.1 puedes crear un contenedor con esa imagen con:

```
docker create --name nignx bitnami/nginx:1.22.1-debian-11-r46
```

Las imágenes se extraen automáticamente de Docker Hub si no las tienes en tu ordenador, pero puedes extraer una imagen antes con `docker pull <imagen>`.

Si no especifica una etiqueta al utilizar una imagen, se utilizará la imagen **más reciente**. Por ejemplo, `docker pull nginx` será el mismo que `docker pull nginx:latest`.



Overview

Tags

Sort by

Newest

Filter Tags

TAG

[latest](#)

Last pushed 3 days ago by [bitnamibot](#)

docker pull bitnami/nginx:la...

DIGEST	OS/ARCH	SCANNED	COMPRESSED SIZE
<a href="#">56fd7ff483c8</a>	linux/amd64	---	36.33 MB
<a href="#">44e4e8bed0ec</a>	linux/arm64	---	34.84 MB

TAG

[1.22.1-debian-11-r45](#)

Last pushed 3 days ago by [bitnamibot](#)

docker pull bitnami/nginx:1...

DIGEST	OS/ARCH	SCANNED	COMPRESSED SIZE
<a href="#">8a2621aef557</a>	linux/amd64	---	36.33 MB
<a href="#">8e68f674a0cd</a>	linux/arm64	---	34.84 MB

TAG

[1.22.1](#)

Last pushed 3 days ago by [bitnamibot](#)

docker pull bitnami/nginx:1...

DIGEST	OS/ARCH	SCANNED	COMPRESSED SIZE
--------	---------	---------	-----------------

Lista de etiquetas de una imagen Docker

## 5.3 Creación de imágenes

A veces te bastará con las imágenes publicadas pero, cuando no es así, puedes crear tu propia imagen. Docker proporciona una forma de hacerlo, el `Dockerfile` y el comando `docker build`.

### 5.3.1 Dockerfile y construcción

Un Dockerfile es un archivo de texto plano con un conjunto de instrucciones para construir la imagen. Un ejemplo de un Dockerfile simple podría ser:

```
DESDE python:3.8
RUN mkdir /scripts
COPIAR ./hello.py /scripts
CMD ["python", "/scripts/hello.py"]
```

Veremos el significado de esas líneas en la siguiente sección. Con el Dockerfile podemos construir una imagen con este comando:

```
docker build --tag <nombre de la imagen> <directorio del Dockerfile>.
```

El directorio que pasamos a `docker build` se llama contexto. Podemos hacer referencia a cualquier archivo de ese directorio en las instrucciones de compilación. Con la opción `--tag` asignaremos un nombre a la imagen generada.

### 5.3.2 Comandos Dockerfile

En este curso, veremos sólo un subconjunto de todas las opciones disponibles para un Dockerfile. Puedes consultar todas las opciones disponibles en la documentación:

<https://docs.docker.com/engine/reference/builder/>

#### DESDE

Esta instrucción se utiliza para establecer la imagen base para crear la nueva. Por ejemplo, si queremos construir una imagen basada en la imagen de `ubuntu` usaremos:

```
DESDE ubuntu
```

Como la primera instrucción en nuestro Dockerfile. Esta instrucción sólo puede existir una vez por etapa de compilación y debe aparecer como la primera instrucción (excepto la instrucción `ARG`, que puede aparecer antes)

#### RUN

Cuando utilizamos la directiva `RUN` en Docker, ejecutamos una instrucción de línea de comandos dentro del "contenedor" mientras construimos la imagen. Los cambios resultantes se colocan encima de la imagen existente como una capa nueva.

Por ejemplo, si queremos instalar `cowsay` en un contenedor base de ubuntu:

```
DESDE ubuntu
EJECUTAR apt
update
```

Es decir, ejecutamos los mismos comandos que ejecutaríamos en el terminal para instalar esa aplicación. Después de construir la imagen, tendremos el comando `cowsay` disponible en todos los contenedores creados a partir de esa imagen.

El ejemplo anterior utiliza la notación shell: los comandos se envían a un shell, que los ejecutará utilizando sustituciones, etc. Existe otra notación, la notación `exec`. Este es el mismo ejemplo usando la notación `exec` (el `#` al principio de la línea significa que la línea es un comentario y no será procesada):

```
DESDE ubuntu
# RUN apt update
# RUN apt install -y cowsay
# Esto es casi equivalente:
RUN ["apt", "update"]
RUN ["apt", "install", "-y", "cowsay"]
```

Al construir imágenes, Docker utilizará la caché y no regenerará una capa si ya ha sido generada anteriormente. Así, por ejemplo, si construimos el Dockerfile anterior dos veces, la segunda construcción será mucho más rápida que la primera.

#### COPIA

La instrucción `COPY` añade ficheros y directorios a la imagen. El origen será el contexto de compilación (el directorio proporcionado al comando de compilación) Por ejemplo, si queremos que el archivo `mi_script.sh` y el directorio `./datos/archivos` estén en la imagen haremos:

```
COPIAR mi_script.sh /bin/mi_script.sh
```

```
COPIAR datos/archivos /destino
```

Ten en cuenta que no puedes copiar archivos desde fuera del contexto de la compilación. También tendrás que asignar los permisos correspondientes a los archivos del contenedor.

## CMD

**CMD** (abreviatura de "Command") es una instrucción en un Dockerfile que especifica el comando predeterminado que se ejecutará cuando se inicie un contenedor basado en la imagen. Si el usuario no proporciona ningún argumento de línea de comandos, se ejecutará el comando predeterminado especificado en la instrucción **CMD**.

Por ejemplo, supongamos que quieres crear una imagen Docker que ejecute un script de Python que imprima "¡Hola, mundo!". Podrías crear un Dockerfile con el siguiente contenido:

```
DE python:3
CMD python -c "print('¡Hola, mundo!')"
```

Cuando construyas esta imagen Docker e inicies un contenedor basado en ella, el comando python se ejecutará por defecto:

```
docker build -t ejemplo-cmd .
docker run --rm ejemplo-cmd

Hola, mundo.
```

El **CMD** puede ser anulado por parámetros de línea de comandos. Así, por ejemplo, podemos ejecutar

```
docker run --rm ejemplo-cmd echo "Patata"
Patata
```

El comando `echo "Potato"` se ejecutará en lugar del **CMD** en el Dockerfile.

## PUNTO DE ENTRADA

Usando **ENTRYPOINT** puedes definir una orden que se ejecutará cuando se inicie el contenedor, como **CMD**. Sin embargo, **ENTRYPOINT** no puede ser anulado y se ejecutará siempre. Este es el comando utilizado por los contenedores que ejecutan un servicio, como servidores web o servidor de base de datos.

Por ejemplo, si creamos

```
DE python:3
ENTRYPOINT python -c "print('Sobreviviré')"
```

No podremos

```
docker build -t ejemplo-punto-de-entrada .
docker run --rm ejemplo-punto-de-entrada echo
"Patata" Sobreviviré
```

Si utiliza la notación exec para **ENTRYPOINT**, los parámetros de la línea de comandos se añadirán como parámetros al punto de entrada. Por ejemplo:

```
DE python:3
ENTRYPOINT ["python", "-c"]
```

Añadirá los parámetros de la línea de comandos al `ENTRYPOINT`, ejecutando ese código (las instrucciones después de `-c` en python son interpretadas y ejecutadas):

```
docker run --rm ejemplo-punto-deentrada print('Desde la línea de comandos')
Desde la línea de comandos
```

Aquí tienes una tabla comparativa entre `CMD` y `ENTRYPOINT`:

CMD	PUNTO DE ENTRADA
Se ejecutará si no se especifica cmd al ejecutar el contenedor	Ejecutará siempre
Puede anularse mediante parámetros de la línea de comandos	No se puede anular. Los parámetros se añaden a la instrucción <code>ENTRYPOINT</code> si se definen en formato <code>exec</code> .

## 6. RED EN G

Las redes Docker son un tema complejo. Hay múltiples modelos de red disponibles, y también puedes instalar plugins de terceros para obtener opciones adicionales. No obstante, en este curso trabajaremos únicamente con el controlador de red **bridge**. Si quieres aprender más sobre redes Docker puedes consultar la documentación oficial: <https://docs.docker.com/network/>

### 6.1 Puente por defecto

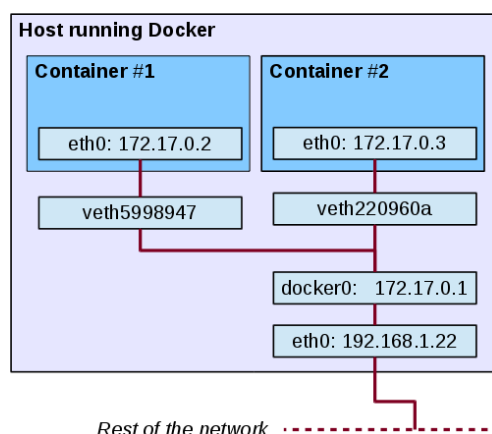
Cuando instalas Docker en tu máquina, se crea una nueva interfaz `docker0`. Puedes mostrarla ejecutando `ifconfig`:

```
docker0: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
    inet 172.17.0.1 netmask 255.255.0.0 broadcast 172.17.255.255 ether
    02:42:90:9b:6f:51 txqueuelen 0 (Ethernet)
    RX paquetes 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX paquetes 0 bytes 0 (0.0 B)
    Errores de transmisión 0 caídas 0 desbordamientos 0 portadora 0
```

Todos los contenedores se conectarán a esa interfaz a menos que se especifique lo contrario. Funciona de la siguiente manera:

El puente por defecto de docker0

- Los contenedores tendrán una IP en la red `docker0`



- Un contenedor puede hablar con otros contenedores, utilizando la dirección IP interna del contenedor. Puedes obtener esa dirección ejecutando un comando dentro del contenedor o usando `docker inspect`.
- Los contenedores pueden conectarse a sitios externos. Tienen `docker0` como puerta de enlace: todas las conexiones pasan por él. Implementa NAT, como el router en una instalación de red doméstica. La interfaz `docker0` será como el router de tu casa, y los contenedores como los dispositivos que conectas al router de tu casa.
- Su contenedor no puede ser accesible desde el exterior.
- Si quieres que tu contenedor esté disponible desde el exterior, tienes que utilizar la asignación de puertos para que esté disponible en uno de los puertos de tu host.

### 6.1.1 Por ejemplo:

En nuestro host, tenemos la interfaz `docker0`:

```
4: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:90:9b:6f:51 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
        valid_lft forever preferred_lft forever
    inet6 fe80::42:90ff:fe9b:6f51/64 scope link
        valid_lft forever preferred_lft forever
```

Si creamos un contenedor usando la imagen `busybox` (esa imagen tiene instaladas utilidades de red y abre un shell por defecto):

```
docker run --rm -ti --name contenedorA busybox
```

Podemos ver que tiene una interfaz `eth0` con la IP `172.17.0.2`, en la misma red que la interfaz `docker0` de nuestro host:

```
/ # ip a
...
28: eth0@if29: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue
    link/ether 02:42:ac:11:00:02 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.2/16 brd 172.17.255.255 scope global eth0
        valid_lft forever preferred_lft forever
```

Y la ruta por defecto es a la IP de `docker0` en nuestro host:

```
/ # route -n
Tabla de enrutamiento IP del núcleo
Destino      Pasarela      Genmask      Bande Métric Ref      Utilizar Iface
ras         a
0.0.0.0      172.17.0.1    0.0.0.0      UG      0      0      0 eth0
172.17.0.0   0.0.0.0      255.255.0.0   U      0      0      0 eth0
```

Podemos llegar al exterior desde nuestro contenedor, prueba por ejemplo a `hacer ping` a `portal.edu.gva.es`. Si creamos otro contenedor:

```
docker run --rm --name contenedorB busybox
```

Ese contenedor tendrá una dirección en la misma red que el anterior:

```
14: eth0@if15: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue
```

```
link/ether 02:42:ac:11:00:03 brd ff:ff:ff:ff:ff:ff
inet 172.17.0.3/16 brd 172.17.255.255 scope global eth0
valid_lft forever preferred_lft forever
```

Y pueden comunicarse:

```
ping 172.17.0.2
PING 172.17.0.2 (172.17.0.2): 56 bytes de datos
64 bytes desde 172.17.0.2: seq=0 ttl=64 time=0.299 ms
64 bytes desde 172.17.0.2: seq=1 ttl=64 time=0.142 ms
64 bytes desde 172.17.0.2: seq=2 ttl=64 time=0.142 ms
```

La red `172.17.0.0/16` es una red privada, por lo que no se puede acceder a ella desde el exterior.

## 6.2 Puente personalizado networks

Las redes personalizadas se utilizan para aislar un grupo de contenedores que sólo deben comunicarse entre ellos. Funcionan igual que la red por defecto, con estas diferencias:

- Sólo pueden llegar a los contenedores de la misma red
- Pueden llegar a los contenedores de la misma red por su nombre
- Puede conectar y desconectar un contenedor de una red personalizada en tiempo de ejecución.

### 6.2.1 Ejemplo

Vamos a crear una red personalizada en docker:

```
docker network create red-uno
```

Después de crear la red, tendremos otra interfaz en nuestra máquina. Su nombre será algo así como `br-XXXX`:

```
18: br-61ce1d3b85de: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc
noqueue state DOWN grupo por defecto
link/ether 02:42:2b:c5:99:78 brd ff:ff:ff:ff:ff:ff
inet 172.20.0.1/16 brd 172.20.255.255 scope global br-61ce1d3b85de
valid_lft forever preferred_lft forever
```

Podemos lanzar un contenedor en esa red con el parámetro `--network`:

```
docker run --rm -ti --name container-A --network network-one busybox
```

El contenedor tendrá una IP de esa nueva red:

```
23: eth0@if24: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc
noqueue
link/ether 02:42:ac:14:00:02 brd ff:ff:ff:ff:ff:ff
inet 172.20.0.2/16 brd 172.20.255.255 scope global eth0
valid_lft forever preferred_lft forever
```

Si lanzamos otro contenedor en la misma red:

```
docker run --rm -ti --name container-B --network network-one busybox
```

Podrá comunicarse con los demás de la misma red por IP y por nombre:

```
/ # ping 172.20.0.2
```



```
PING 172.20.0.2 (172.20.0.2): 56 bytes de datos
64 bytes desde 172.20.0.2: seq=0 ttl=64 time=0.249 ms
64 bytes desde 172.20.0.2: seq=1 ttl=64 time=0.144 ms

/ # ping contenedor-A
PING contenedor-A (172.20.0.2): 56 bytes de datos
64 bytes desde 172.20.0.2: seq=0 ttl=64 time=0.303 ms
64 bytes desde 172.20.0.2: seq=1 ttl=64 time=0.145 ms
```

Si lanzamos un contenedor en otra red (la predeterminada, por ejemplo):

```
docker run --rm -ti --name contenedor-C busybox
```

No podrá comunicarse con los contenedores de la red personalizada:

```
# ping 172.20.0.2
PING 172.20.0.2 (172.20.0.2): 56 bytes de datos
^C
--- 172.20.0.2 ping statistics ---
3 paquetes transmitidos, 0 paquetes recibidos, 100% de pérdida de paquetes
```

### 6.2.2 Eliminar redes

Puedes eliminar una red cuando no la necesites con: `docker network rm <nombre de la red>.`

Si desea eliminar todas las redes no utilizadas, puede ejecutar `docker network prune`. Esto eliminará todas las redes que no están siendo utilizadas por un contenedor.

## 7. MATERIA COMPLEMENTARIA L

Redes de puentes personalizadas:

<https://blog.devgenius.io/custom-docker-bridge-networks-how-to-run-containers-b8d40c51bab2>

DockerFile

<https://www.ionos.com/digitalguide/server/know-how/dockerfile/>