

T5.1 – Redes en Docker

Cada contenedor y los servicios que podemos tener instalados en él tienen conexión de red pero no ha habido que configurar nada.

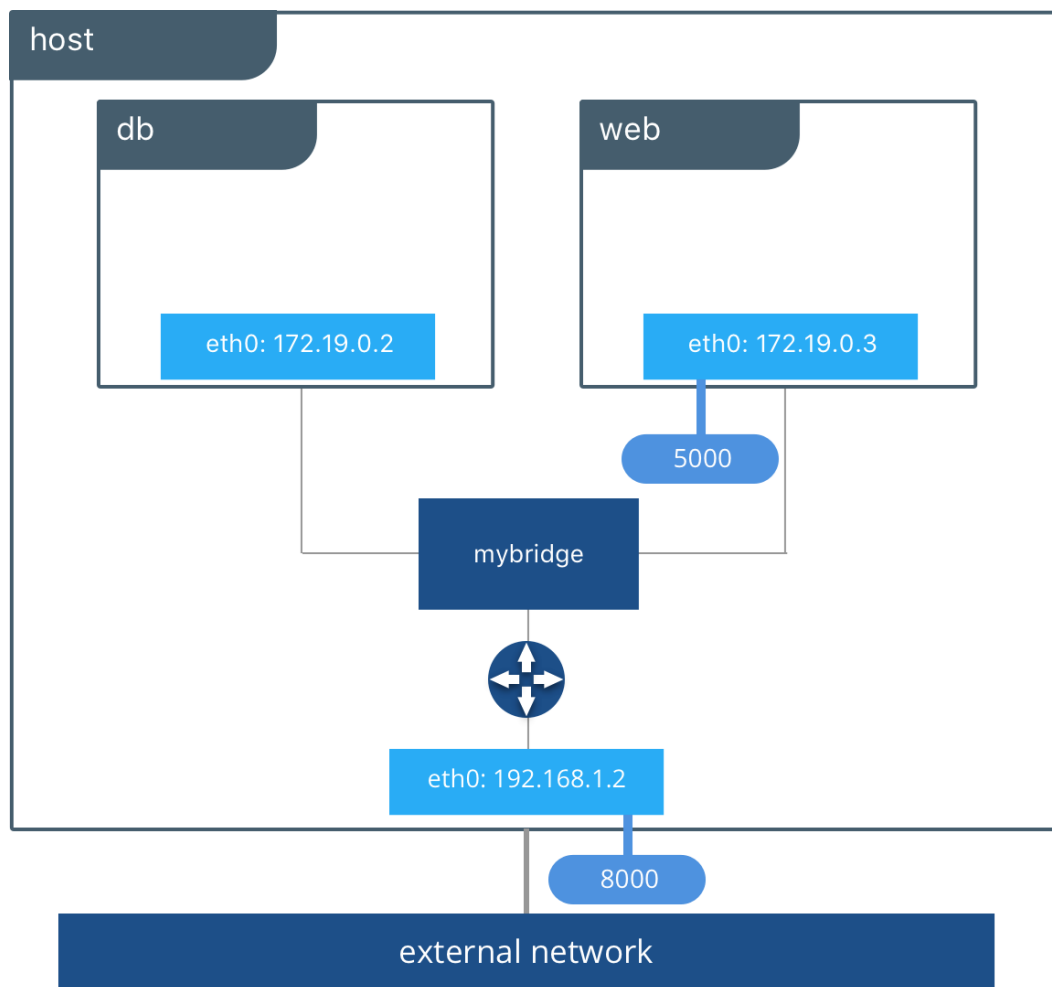
Docker network explained

First, let's understand how the Docker network works. For that we are going to focus on the default bridge network. When you are using Docker, if you don't specify a driver this is the type of network you are using.

The **bridge** network works as a private network internal to the host so containers on it can communicate. External access is granted by exposing ports to containers.

Bridge networks are used when your applications run in standalone containers that need to communicate.

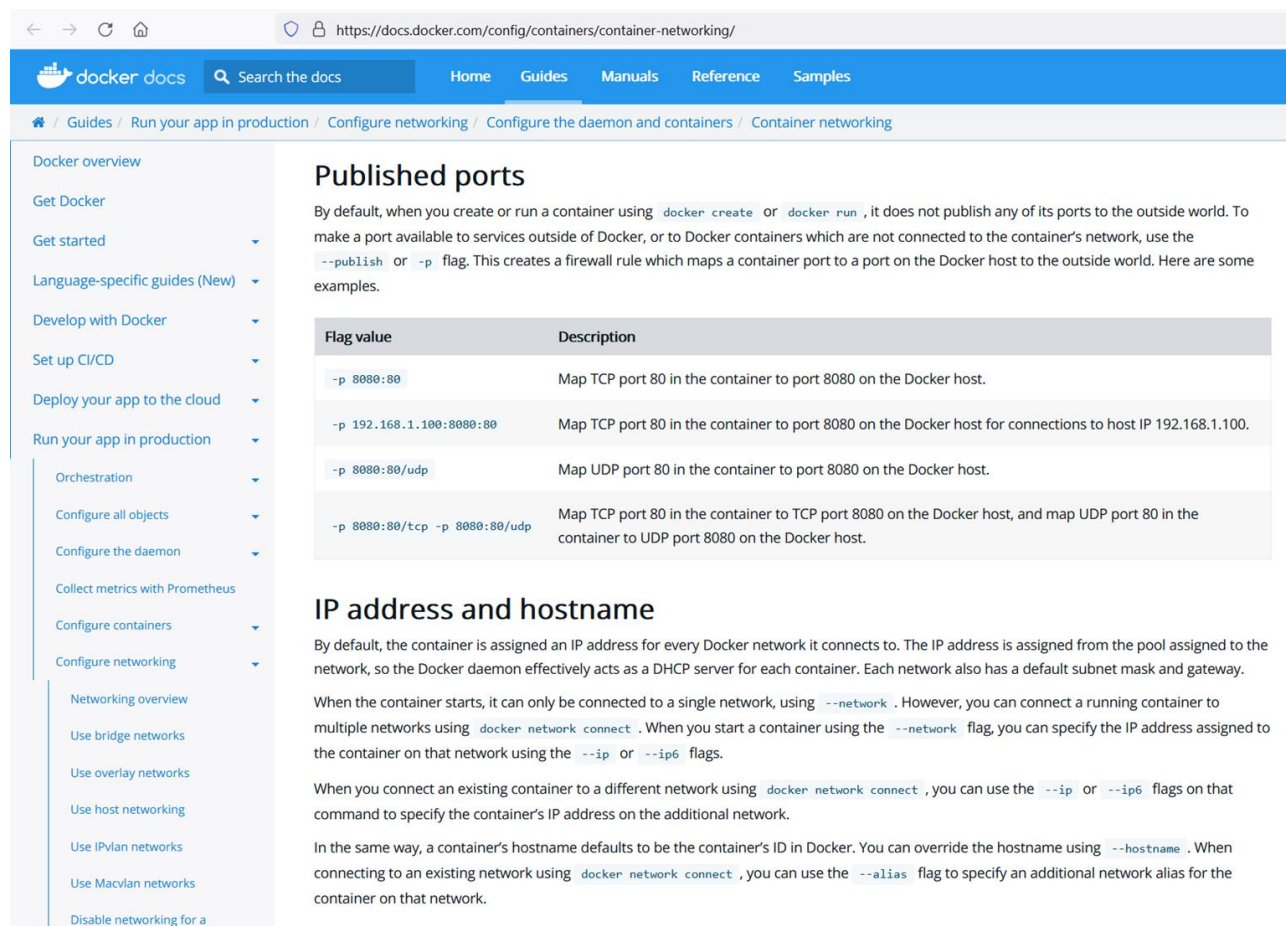
In the picture above db and web can communicate with each other on a user created bridge network called mybridge.



Docker Container IP Address

By default, the container is assigned an IP address for every Docker network it connects to. And each network is created with a default subnet mask, using it as a pool later on to give away the IP addresses.

Usually Docker uses the default **172.17. 0.0/16** subnet for container networking.



The screenshot shows the Docker documentation website. The browser address bar displays `https://docs.docker.com/config/containers/container-networking/`. The page has a blue header with the Docker logo and navigation links: Home, Guides, Manuals, Reference, and Samples. A search bar is also present. Below the header, a breadcrumb trail reads: / Guides / Run your app in production / Configure networking / Configure the daemon and containers / Container networking. The left sidebar contains a table of contents with links to Docker overview, Get Docker, Get started, Language-specific guides (New), Develop with Docker, Set up CI/CD, Deploy your app to the cloud, Run your app in production, and various configuration options. The main content area is titled "Published ports" and explains that by default, containers do not publish ports. It provides examples of the `--publish` or `-p` flag, such as `-p 8080:80` to map container port 80 to host port 8080. A table lists these flag values and their descriptions. Below this, the "IP address and hostname" section explains that containers are assigned IP addresses from a pool and can be connected to specific networks using the `--network` flag. It also mentions that IP addresses can be specified using `--ip` or `--ip6` flags.

| Flag value | Description |
|--|---|
| <code>-p 8080:80</code> | Map TCP port 80 in the container to port 8080 on the Docker host. |
| <code>-p 192.168.1.100:8080:80</code> | Map TCP port 80 in the container to port 8080 on the Docker host for connections to host IP 192.168.1.100. |
| <code>-p 8080:80/udp</code> | Map UDP port 80 in the container to port 8080 on the Docker host. |
| <code>-p 8080:80/tcp -p 8080:80/udp</code> | Map TCP port 80 in the container to TCP port 8080 on the Docker host, and map UDP port 80 in the container to UDP port 8080 on the Docker host. |

Las conexiones de red pueden ser de los siguientes tipos:

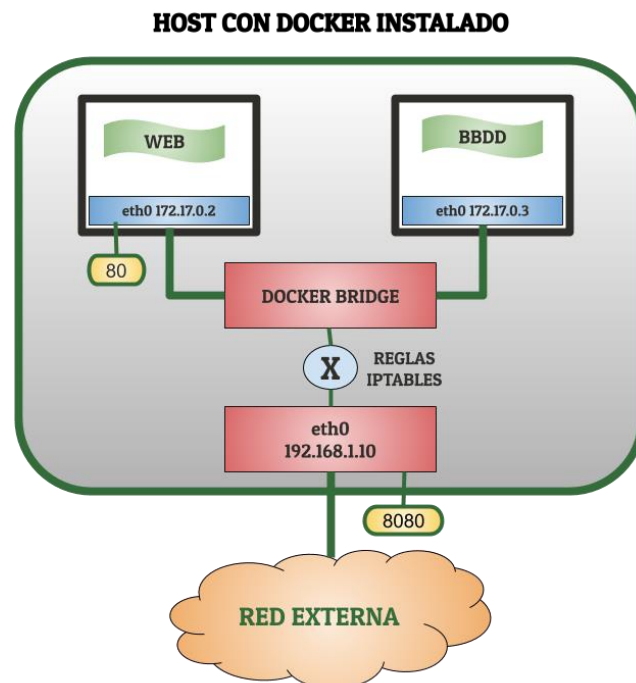
- **Bridge:** Es el driver por defecto. El equipo actúa como puente del contenedor.
- **Host:** El contenedor usa directamente la red de mi máquina (el host).
- **Overlay:** Un sistema que conecta distintos servicios docker de máquinas diferentes.
- **MacVlan:** Que nos permite asignar una MAC a nuestro contenedor que parecerá que es un dispositivo físico en nuestra red.
- **None:** El contenedor no tenga conectividad alguna.

Aunque todos estos tipos de drivers tienen su utilidad en determinadas situaciones, nos vamos a centrar únicamente en los drivers de tipo **BRIDGE** que va a permitir, siempre dentro nuestra máquina local:

- **Aislar los distintos contenedores** que tengo en distintas subredes docker, de tal manera que desde cada una de las subredes solo podremos acceder a los equipos de esa misma subred.

- **Aislar los contenedores del acceso exterior.**
- **Publicar servicios** que tengamos en los contenedores **mediante redirecciones** que docker implementará con las pertinentes reglas de ip tables.

Un ejemplo de una configuración de una **RED CONSTRUIDA CON EL DRIVER BRIDGE** podría ser la siguiente:



Juan Diego Pérez Jiménez. *Ejemplo red docker con driver bridge* (Dominio público)

Este esquema representa una aplicación típica compuesta por dos servicios, un servidor web y un servidor de base de datos, cada uno de ellos en un contenedor diferente y haciendo accesible al exterior mi servidor web en el puerto 8080.

Para que todo esto funcione docker creará de manera automática los interfaces virtuales y los puentes de red necesarios para cada uno de los dispositivos y configurará las reglas necesarias para que esos interfaces tengan acceso a internet, para aislar los contenedores del resto de las redes y para establecer las redirecciones de puertos necesarias.

La red por defecto se llama bridge y podemos comprobar que se ha creado ejecutando la siguiente orden que nos muestra todas las redes docker que tengamos:

```
# Mostrar todas las redes docker creadas
> docker network ls
```

La salida, además del nombre de cada una de las redes creadas recoge la siguiente información:

- El **NETWORK ID** que me sirve para identificar una red y que se puede usar indistintamente con el nombre para cualquiera de las operaciones de gestión de redes (crear, borrar, obtener información etc...)
- El **DRIVER**, que como ya dijimos en el apartado anterior me define el tipo de red que voy a "conectar" a los contenedores. Podía tomar los valores bridge, none, host, macvlan y overlay.
- El **SCOPE** que nos indica el ámbito de nuestras redes y que en este caso son redes locales dentro de nuestra propia máquina.

Esta red "bridged" por defecto, que es la usada por defecto por los contenedores, se diferencia en varios aspectos de las redes "bridged" que creamos nosotros. Estos aspectos son los siguientes:

- Las redes que nosotros definamos proporcionan **resolución DNS entre los contenedores** cosa que la red por defecto no hace a no ser que usemos opciones que ya se consideran "deprectated".
- Puedo **conectar en caliente a los contenedores redes "bridged" definidas por el usuario**. Si uso la red por defecto tengo que parar previamente el contenedor.
- Me permite gestionar de manera más segura el **aislamiento de los contenedores** ,ya que si no indico una red al arrancar un contenedor éste se incluye en la red por defecto donde pueden convivir servicios que no tengan nada que ver.
- Tengo más control sobre la configuración de las redes si las defino yo. Los contenedores de la red por defecto comparten todos la misma configuración de red (MTU, reglas ip tables etc...).
- Los contenedores dentro de la red "bridge" comparten todos ciertas variables de entorno lo que puede provocar ciertos conflictos.

Una vez nos hemos situado vamos a ver cómo realizamos las operaciones más comunes para gestionar y trabajar con redes en docker. Estas operaciones son:

- Listado de las redes (ya visto, *docker network ls*)
- Creación de las redes. (**docker network create**)
- Borrado de las redes. (**docker network rm / docker network prune**)

Una descripción más detallada de lo todas las opciones la podemos ver en la referencia completa de redes en docker pero, tal y como acostumbramos en este curso, vamos a ilustrar su funcionamiento con distintos ejemplos.

EJEMPLOS DE CREACIÓN DE REDES

```
# Crear una red. Al no poner nada más coge las opciones por defecto, red bridge local y el mismo docker elige la dirección de red y la máscara
```

```
> docker network create red1
```

```
# Crear una red (la red2) dándole explícitamente el driver bridge (-d) , una dirección y una máscara de red (--subnet) y una gateway (--gateway)
```

```
> docker network create -d bridge --subnet 172.24.0.0/16 --gateway 172.24.0.1 red2
```

La orden **docker network create** tiene más opciones para las redes de tipo bridge y muchas más para redes de otro tipo. Pero como estamos en un curso de docker aplicado al desarrollo estas opciones son más que suficientes para poder montar nuestros entornos y los de nuestros alumnos.

NOTA: CADA RED DOCKER QUE CREO CREA UN PUENTE DE RED ESPECÍFICO PARA CADA RED QUE PODEMOS VER CON `ifconfig` / `ip a`

ELIMINACIÓN DE REDES

```
# Eliminar la red red1
```

```
> docker network rm red1
```

```
# Eliminar una red con un determinado ID
```

```
> docker network rm 3cb4100fe2dc
```

```
# Eliminar todas la redes que no tengan contenedores asociados
```

```
> docker network prune
```

```
# Eliminar todas las redes que no tengas contenedores asociados sin preguntar confirmación (-f o --force)
```

```
> docker network prune -f
```

```
# Eliminar todas las redes que no tengan contenedores asociados y que fueron creadas hace más de 1 hora (--filter)
```

```
> docker network prune --filter until=60m
```

Capturas a entregar: (incluir capturas de todos los comandos necesarios)

- Comprueba los caso de volume y bind mount anteriormente indicados
- Crea un volumen llamado **volumen_web1** y volumen_http
- Comprueba la información del volumen creado.
- Elimina volumen_http y compruebalo.
- Crea un contenedor sobre la imagen **php:7.4-apache** a la vez que montas un volumen que has creado anteriormente volumen_web con su directorio de trabajo y accesible por el puerto 8787 (consulta la documentación sobre dockerhub). Y lista los volúmenes y los contenedores creados.
- Inspecciona la diferencia entre los volúmenes y big mount creados creados.