5. Redes en docker

Estamos en el módulo 5 y a lo largo de los diferentes módulos anteriores hemos comprobado varias veces que **los contenedores podían conectarse a internet** para **bajarse actualizaciones**, **instalar paquetes** a partir de los repositorios que venían con las distintas distribuciones, **obtenían IP local** y si los contenedores estaban ejecutando servicios, esos servicios **eran accesibles** a través de la IP que tenía el contenedor **o bien desde el exterior a través de un redirección de puertos** en nuestro equipo.

Para que todo esto suceda **docker ha creado sin que nosotros seamos conscientes una "red docker"** y establecido las **reglas iptables necesarias para que esos contenedores tengan conectividad** con el exterior, para que tengan **conectividad entre ellos** si pertenecen a la misma red docker, y para que sean **accesibles desde el exterior** si es que hemos hecho la redirección de puertos pertinente.

En este módulo profundizaremos en este aspecto docker centrándonos en un tipo concreto de red docker, la red **bridge** que es la de uso más común para el desarrollo.

5.1 Tipos de redes en Docker

Cuando nuestro contenedor y los servicios que podemos tener instalados en él tienen algún tipo de conexión nosotros **no hemos tenido que configurar nada** y el contenedor ni sabe ni es consciente del funcionamiento de la red ni de la plataforma sobre la que funciona. El contenedor es **red y sistema agnóstico**.

Esto se consigue con un sistema de red en el que nosotros podemos "enchufar" distintos dispositivos de red a cada contenedor usando distintos drivers que pueden ser de los siguientes tipos:

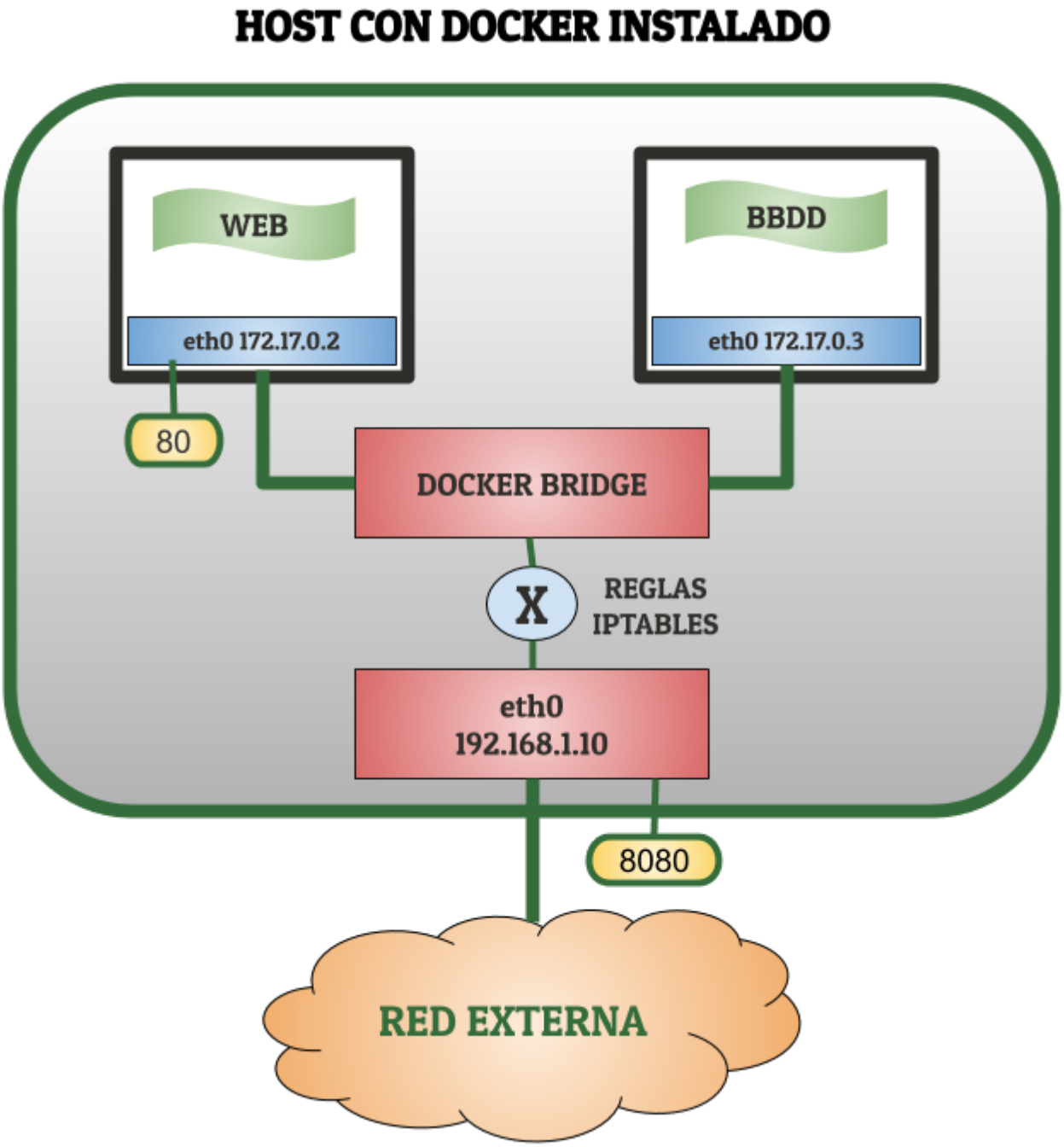
- **Bridge:** Es el driver por defecto. Mi equipo actúa como puente del contenedor con el exterior y como medio de comunicación entre los distintos contenedores que tengo en ejecución dentro de una misma red docker.
- **Host:** El contenedor usa directamente la red de mi máquina (el host).
- **Overlay:** Un sistema que conecta distintos servicios docker de máquinas diferentes. Se utiliza para docker Swarm, que es la tecnología de docker para la orquestación de contenedores.
- **MacVlan:** Que nos permite asignar una MAC a nuestro contenedor que parecerá que es un dispositivo físico en nuestra red.
- **None:** Si queremos que el contenedor no tenga conectividad alguna.

Aunque todos estos tipos de drivers tienen su utilizad en determinadas situaciones lo cierto es que para los objetivos del curso nos vamos a centrar únicamente en los drivers de tipo **BRIDGE** que me van 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.

Todo esto lo veremos con más detalle en los apartados posteriores de este mismo módulo.

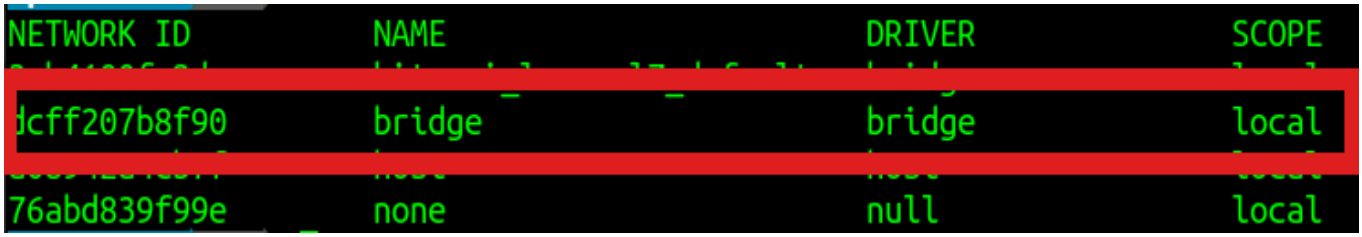
5.2 Gestionando redes

En el apartado anterior comentamos que para usar docker para el desarrollo vamos a tener **más que suficiente** con la creación de **redes** con el driver **bridge**. Sin embargo, vamos a tener que hacer una diferenciación entre **dos tipos de redes "bridged"**: la red creada **por defecto** por docker para que funcionen todos los contenedores y aquellas redes "bridged" **definidas por el usuario**, es decir, por nosotros. Esta red por defecto se llama **bridge** y podemos comprobrar 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
```

Como resultado obtendremos una salida similar a la siguiente, en la que se destaca mediante un recuadro la red bridge por defecto.



Juan Diego Pérez Jiménez. *Red bridge docker por defecto* (Dominio público)

Esta 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 netowkr 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
```

NOTA: NO PUEDO BORRAR UNA RED QUE TENGA CONTENEDORES QUE LA ESTÉN USANDO. DEBERÉ PRIMERO BORRAR LOS CONTENEDORES O DESCONECTAR LA RED.

5.3 Obteniendo información de las redes

De igual manera que con las imágenes y los contenedores, puedo obtener información de las redes docker de maneras diferentes:

- Mediante la orden ***docker network ls,*** que presentamos en el apartado anterior y que además tiene diversas opciones algunas de las cuales veremos posteriormente.
- Mediante la orden ***docker network inspect,*** que nos mostrará una información mucho más detallada con todas las características de la red.

Un ejemplo de la ejecución de la orden inspect lo podemos ver en la siguiente imagen:

```
pekechis ~ docker network inspect bitnami_laravel7_default
[
    {
        "Name": "bitnami_laravel7_default",
        "Id": "3cb4100fe2dc638a787055d6db47c9c4f47f6441903b0a330bb3fe4ec18975ef",
        "Created": "2020-11-30T23:29:30.684400342+01:00",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "172.19.0.0/16",
                    "Gateway": "172.19.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {},
        "Options": {},
        "Labels": {}
    }
]
```

Juan Diego Pérez Jiménez. *Ejemplo de salida de la orden docker inspect* (Dominio público)

También podemos formatear esta salida usando Go Templates, tal y como habíamos hecho en los módulos anteriores cuando inspeccionábamos imágenes o contenedores. Un ejemplo de ello sería lo siguiente:

```
# Mostrar el tipo de driver de una red docker (podríamos usar también el ID de la red)

> docker network inspect mi_red  --format 'El driver de {{.Name}} es {{.Driver}}'
```

```
# Mostrar la dirección de red y la pasarela de una red docker

> docker network inspect mi_red  --format '{{.IPAM.Config}}'
```

En cuanto a la orden ***docker network ls,*** vamos a ver con distintos ejemplos algunas de las opciones que podemos usar:

```
# Mostrar solo el ID de las redes (-q o --quiet)

> docker network ls -q

# Mostrar las redes de driver=bridge y nombre=brigde( la red por defecto) (-f o --filter)

> docker network ls -f driver=bridge -f name="bridge"

# Mostrar lo mismo que en el anterior caso pero formateando con Go Templates

> docker network ls -f driver=bridge -f name=bridge --format 'La red por de defecto tiene el siguiente ID {{.ID}}'
```

5.4 Asociando redes a contenedores

En los tres apartados anteriores hemos hablado de aspectos relativos a las redes docker pero **no hemos dicho nada de cómo "conectamos" esas redes a nuestros contenedores** que es el paso fundamental para que dichos contenedores puedan conectarse entre ellos , puedan ofrecer servicios al exterior y puedan conectarse a Internet para poder actualizarse y/o instalar cualquier dependencia que necesitemos. Para realizar esta "conexión" debemos de tener en cuenta los siguientes aspectos:

- Al arrancar un contenedor podemos **especificar a qué red** está conectado inicialmente usando el **flag --network** seguido del nombre de la red a la que queremos conectarlo.
- Si al arrancar un contenedor **no especificamos una red**, el contenedor se conectará a la red por defecto, la **red "bridge" que usa el driver "bridge"**.
- **Al arrancar** un contenedor **no** puedo **"conectarlo"** inicialmente **a más de una red.**
- **Tras crear el contenedor** puedo **conectarlo a más redes o desconectarlo de alguna red.** Dependiendo de si he elegido la red por defecto o no, podré o no podré hacer esa conexión o desconexión en caliente (con el contenedor un funcionando).

Para ilustrar todas estar afirmaciones vamos a realizar distintos ejemplos:.

```
# Arrancar un contenedor de Apache sin especificar red y habilitando la conexión desde el exterior a través del puerto 80. Se conectará por defecto a la red bridge.
```

```
> docker run -d --name web -p 80:80 httpd

# Arranchar un contenedor de Apache conectándose a la red red1 que es una red bridge definida por el usuario y habilitando la conexión desde el exterior a través del puerto 8080

> docker run -d --name web2 --network red1 -p 8080:80 httpd

# Arranchar un contenedor de Apache conectándose a la red red1 dándole una ip (que debe pertenecer a esa red)

> docker run -d --name web2 --network red1 --ip 172.18.0.5 -p 8181:80 httpd

# Conectar una nueva red, la red2  al contenedor web2.

> docker network connect red2 web2

# Conectar una red, la red2 al contenedor web y darle una ip (que debe pertenecer a esa red)

> docker network connect --ip 172.28.0.3 red2 web

# Desconectar la red1 del contenedor web2. Debe estar funcionando para poder desconectarse

> docker network disconnect red1 web2
```

Por supuesto la orden docker connect tienen más opciones que podemos consultar en la referencia y , adicionalmente, hay varios flags de la orden docker run que están relacionados con redes y que pueden resultar de interés:

- **--dns** para establecer unos servidores DNS predeterminados.
- **--ip6** para establecer la dirección de red ipv6
- **--hostname o -h** para establecer el nombre de host del contenedor. Si no lo establezco será el ID del mismo.

5.5 Iptables en contenedores

En ocasiones en algunos de los módulos (más relacionados con redes normalmente) puede que queramos montar ciertos entornos, compuestos por varias máquinas, en los que sea necesario que tengamos nosotros el control de las reglas **iptables** de las distintas máquinas que los conforman.

Normalmente esto se hace con distintas máquinas virtuales pero en vista de las posibilidades de red que nos dan los contenedores, ¿podría montar entornos de ese estilo con contenedores?. **Por defecto esto no es posible en los contenedores**, pero podemos **conseguirlo** de una de estas dos maneras:

- Arrancar el contenedor con el flag **--cap-add=NET_ADMIN**. Así le damos al contenedor la "capacidad" Linux de administrar la red.

- Arrancar el contenedor de con flag **--privileged** que le da todas las capacidades al contenedor que puede hacer lo mismo que se puede hacer desde el host. Esto, además de habilitar el uso de iptables, me permitiría cosas como "instalar docker dentro de un contenedor docker". Debemos de tener mucho cuidado con esta característica porque puede presentar varios problemas al levantar ciertas limitaciones a los cgroups..

Independientemente de la opción que elijamos, antes de poder usar iptables, y dado las pocas cosas que incluye por defecto un contenedor, deberemos:

- **apt update** (para recargar los repositorios que no vienen cargados por defecto).
- **apt install -y iptables** (para instalar la herramienta de firewall para Linux).