CFGS ADMINISTRACIÓN DE SISTEMAS INFORMÁTICOS EN RED

BIENVENIDOS!

Manuel Domínguez.

mftienda@gmail.com

@mafradoti

www.linkedin.com/in/mftienda

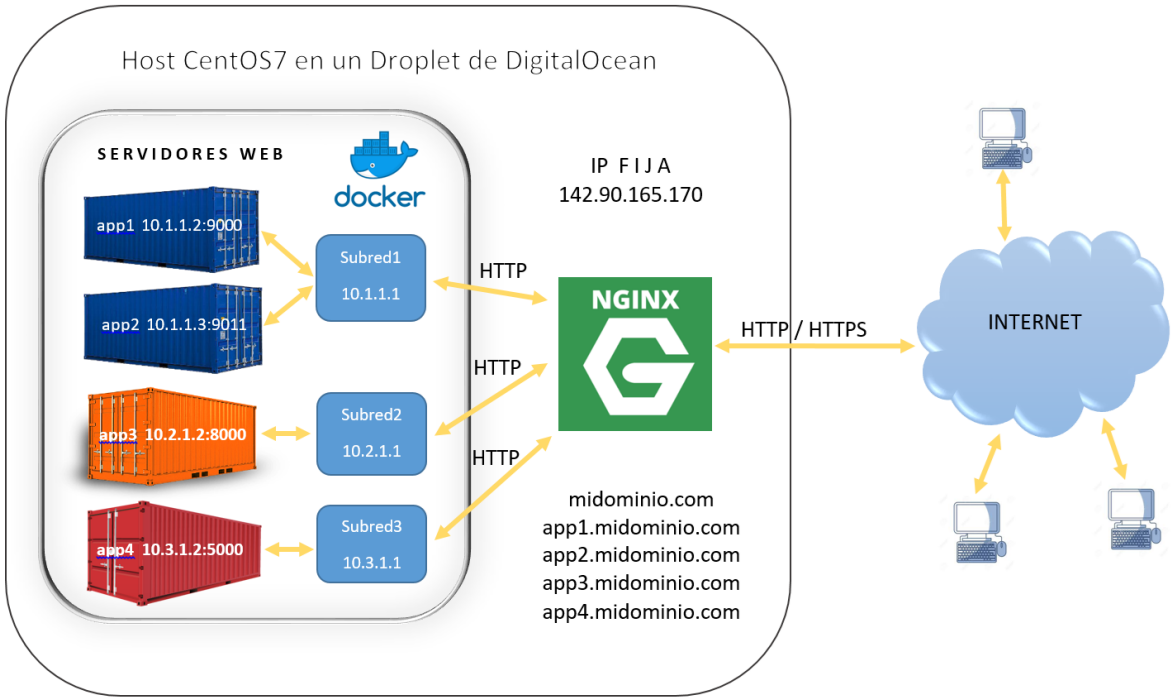
https://github.com/mftienda

Ud.- Comunicar contenedores

Índice

1.- Introducción

¿Cómo podemos conectar un contenedor con el mundo exterior?



2.- Mapear puertos

El mapeo de puertos nos va a permitir **comunicar nuestros contenedores con el mundo exterior.**

Por defecto los puertos de un contenedor son privados y no pueden ser accedidos.

Debemos hacerlos públicos (Expose)

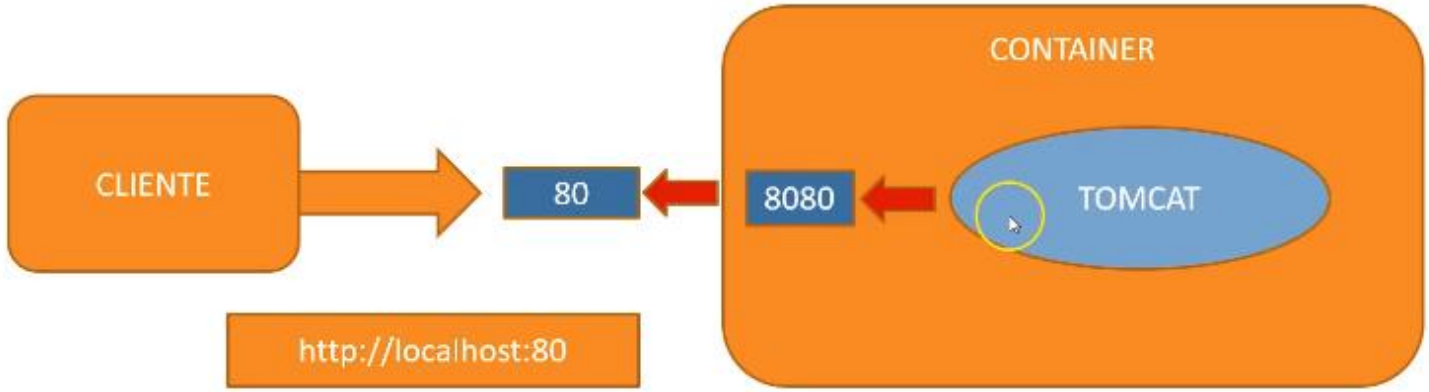
Podemos mapear los puertos del contenedor con puertos de la máquina host.

Es decir, un puerto local lo apuntamos a un puerto del contenedor. De esa forma podemos acceder al contenedor desde el exterior.

2.- Mapear puertos

Ejemplo:
El tomcat escucha por el puerto 8080
Debo hacerlo público.
Debo mapearlo con un puerto de la máquina real. En este caso lo hemos mapeado con el puerto 80.

El puerto 8080 del contenedor va a ser accesible a través del puerto 80 del cliente. Haciendo localhost:80 estamos accediendo al contenedor a través del puerto 8080.



2.- Mapear puertos

Puerto aleatorio

nginx: Consultando la documentación, podemos ver que utiliza el puerto 80.

-P → Esta opción convierte todos los puertos del contenedor en público y asigna uno aleatorio.

#docker run -d -P nginx
-d → Para que se ejecute en segundo plano y nos deje libre la consola.

#docker ps

```
PORTS
0.0.0.0:32768->80/tcp
```

0.0.0.0 → Desde cualquier dirección IP de la máquina principal (Puede ser que nuestra máquina tenga dos IPs)
32768 → Puerto de la máquina principal
80 → Puerto al que accedemos dentro del contenedor.

2.- Mapear puertos

Puerto aleatorio

Para visualizar el servidor web nginx:

1.- localhost:32768

o bien

2.- 172.17.0.2:80, siendo 172.17.0.2, la IP del contenedor.

2.- Mapear puertos

Puerto fijo

```
-p <puerto_host>:<puerto_contenedor>
```

httpd: Consultando la documentación, podemos ver que utiliza el puerto 80.

docker run -p 81:80 httpd:latest

El puerto 81 de la máquina host apunta al puerto 80 del contenedor.

Podemos acceder al contenedor escribiendo en el navegador: **localhost:81** .

También: 127.17.0.2:80 (suponiendo que esa es la IP asignada al contenedor)

2.- Mapear puertos

Conocer puertos:

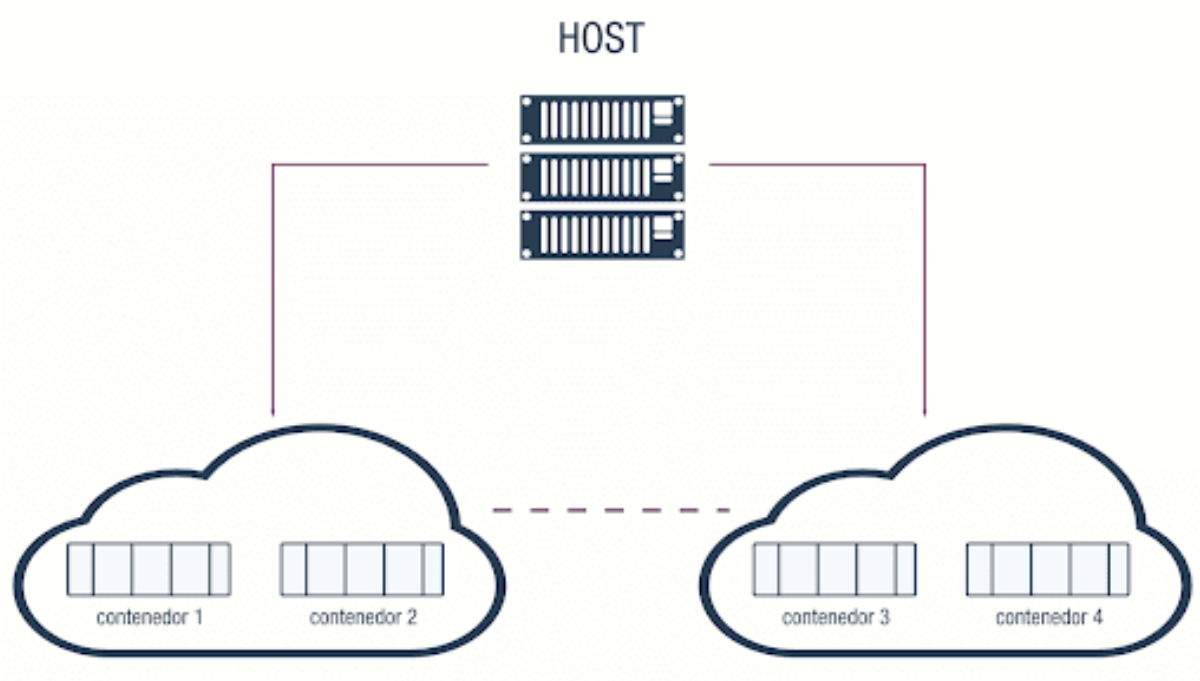
 # Para ver puertos abiertos en linux: netstat -lntu

-l: Permite ver exclusivamente los puertos de escucha
-n: Permite ver el número de puerto
-t: Visualiza los puertos TCP
-u: Visualiza los puertos UDP

Nota: netstat → net-tools

#Para ver puertos abiertos en el contenedor:
docker inspect ID|grep -i tcp

3.- Redes en Docker



3.- Redes en Docker

root@debian:~# **docker network --help**

Usage: docker network COMMAND

Manage networks

Commands:
  connect   Connect a container to a network
  create   Create a network
  disconnect Disconnect a container from a network
  inspect   Display detailed information on one or more networks
  ls   List networks
  prune   Remove all unused networks
  rm   Remove one or more networks

Run 'docker network COMMAND --help' for more information on a command.

3.- Redes en Docker

Redes que implementan

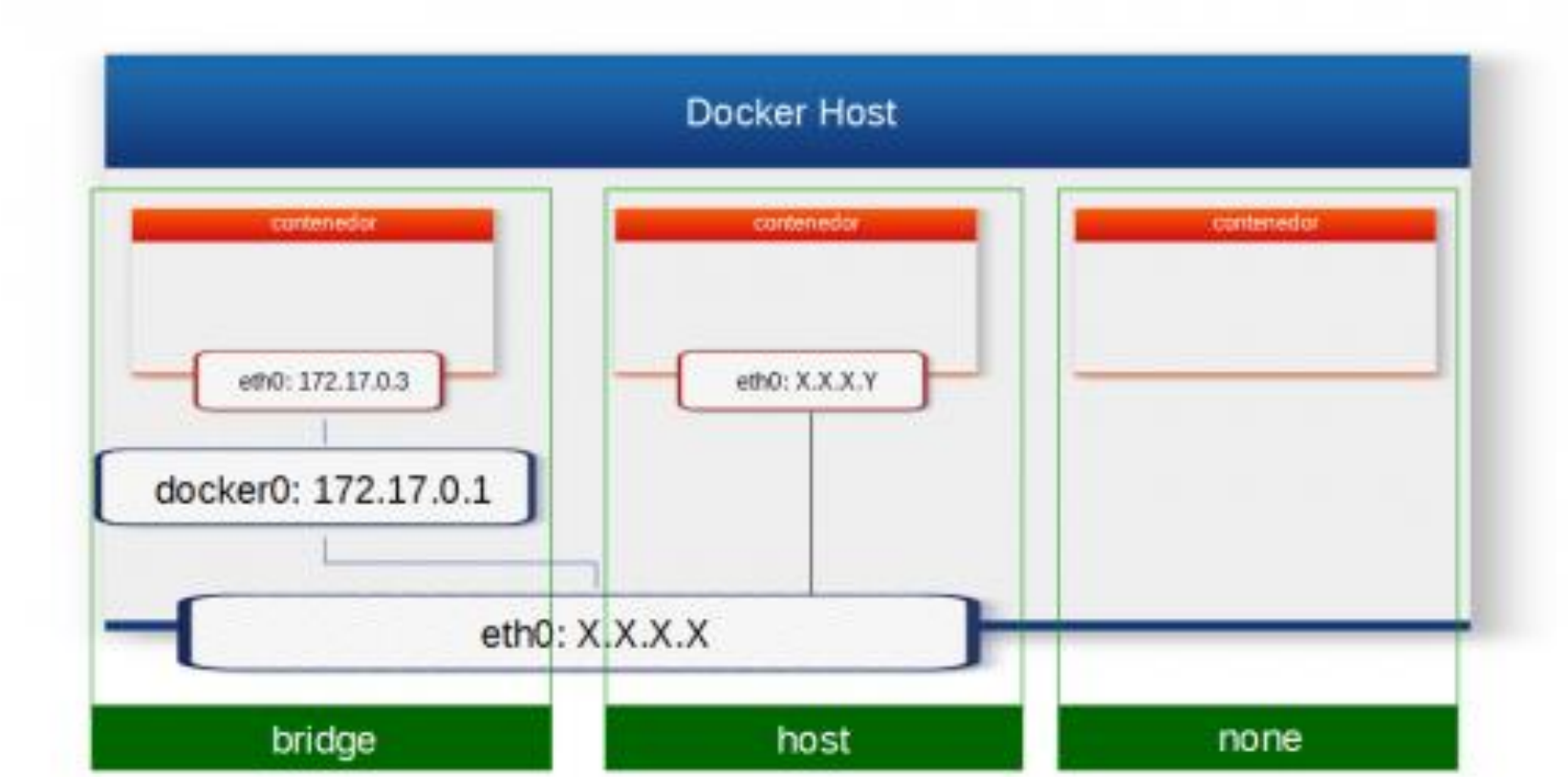
root@debian:~# **docker network ls**

NETWORK ID	NAME	DRIVER	SCOPE
385e9c504f6b	bridge	bridge	local
431da480179b	host	host	local
5ae4db105586	none	null	local

bridge: conecta el contenedor con el interfaz docker0 del host, utilizando una red privada y un servicio DHCP que asigna las direcciones IP a los contenedores. Es la red a la que se conectan por defecto.
host: conecta el contenedor con la red del host, de forma que no hay aislamiento entre el host y el contenedor. **Los contenedores no se ven entre sí.**
none: en este caso el contenedor no tiene red.

3.- Redes en Docker

Redes que implementan



3.- Redes en Docker

Redes que implementan

En la instalación por defecto de Docker, se crea el interfaz **docker0 en el host.**

Podemos ver las interfaz: # ip a

```
3: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:70:e4:82:31 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::42:70ff:fee4:8231/64 scope link
       valid_lft forever preferred_lft forever
```

3.- Redes en Docker

Ejemplo:

1.- Creamos un contenedor. #docker run -d nginx

2.- Para ver la Ip asignada al contenedor: #docker inspect ID|grep IPAd

3.- Hacemos un ping para ver que hay conectividad:

ping -c 4 172.17.0.2

4.- Podemos ver el tipo de red: #docker inspect ID →

```
"NetworkSettings": {
    "Bridge": "",
```

3.- Redes en Docker

Inspeccionar una red

 # docker network inspect bridge

Name
Driver
Config
subnet
Gateway

Containers → Podemos ver los contenedores **arrancados** asignados a esa red.

4.-Crear una red

Crear una red

Bridge por defecto → Los contenedores deben utilizar **link** para unirse.

Redes bridges

Bridge creadas → Los contenedores asociados a esa red exponen todos sus puertos (Se van a ver).

Es decir, las redes bridge que creamos tienen propiedades adicionales a la de defecto.

RECOMENDACIONES: Crear tus propias redes.

4.-Crear una red

Crear una red

 # docker network create --help

Podemos especificar:

Tipo de driver: bridge, host, null, overlay (cluster), etc

Subnet

#docker network create red1 → Por defecto le asigna, tipo bridge

#docker network ls

 # docker network create --subnet=192.168.3.0/24 red2

4.-Crear una red

Redes que implementan

Cuando creamos una red, en el host se crea una nueva interfaz para esa nueva red.

Podemos verlas, escribiendo:

#ip a
#nmcli con → Conexiones

docker0: Asignada a la red por defecto, bridge.
br-XXXXXX: Asignada a la nueva red bridge, que hemos creado.

5.- Asociar un contenedor a una red.

Asociar a un contenedor una red en la creación

 # docker run -it --name mi-debian --network red1 debian

Podemos comprobar la IP asignada y la red:

#docker inspect mi-debian → IP asignada

Asociar a otro contenedor la misma red en la creación

 # docker run -d --name mi-nginx --network red1 nginx

Comprobación:

mi-debian → ping a mi-nginx

5.- Asociar un contenedor a una red.

Asociar a un contenedor una red en CALIENTE

 # docker network connect red2 mi-debian

Comprobaciones:

#docker network inspect red2 → containers
#docker inspect mi-debian

Desconectar de una red:

```
~# docker network disconnect red2 mi-debian
~# docker inspect mi-debian |grep IPA
"SecondaryIPAddresses": null,
"IPAddress": "",
        "IPAMConfig": null,
        "IPAddress": "172.18.0.2",
```

6.- Enlazar contenedores con link

Cuando los contenedores están unidos a través de la **red bridge por defecto.**

Enlazamos los contenedores mediante la opción **-link**, se crea un «tunel» que enlaza los dos contenedores, sin la necesidad de exponer puertos.

Al enlazar los contenedores, también hacemos que las variables de entorno de un contenedor, sean accesibles desde el otro.

Hay que tener en cuenta que los links **son unidimensionales.**

IMPORTANTE: Cuando voy a enlazar contenedores, es mejor asignarle un nombre al contenedor.

6.- Enlazar contenedores con link

Creación de link entre contenedores:

Imagen **BusyBox** (Imagen pequeño, utilizado para probar componentes)

1.- # docker run -it --name mi-busybox -1 busybox → Contenedor-1

2.- # docker run -it --name mi-busybox-2 busybox → Contenedor-2

Comprobaciones:

.- Desde un terminal: #docker network inspect bridge → Se encuentra los contenedores
.- Desde cada contenedor: #ip a → Vemos las ip asignadas
.- Desde cada contenedor: ping 172.17.0.x → Responde. Están en la misma red.

6.- Enlazar contenedores con link

Creación de link entre contenedores:

Desde el contenedor1: ping mi-busybox-2, NO responde

Solución: Crear un enlace

#docker run -it --name mi-busybox-3 --link mi-busybox-1:busy-1 busybox

El nuevo contenedor enlaza (conoce) a mi-busybox-1 (alias, busy-1)

Comprobaciones:

Desde el contenedor-3: cat /etc/hosts
ping mi-busybox-1
ping busy-1

Son enlaces unidimensionales.

7.- Enlazar contenedores de redes personalizadas.

Creación de contenedores:

mysql:

Consultamos la documentación en Docker Hub. Esta imagen, nos crea una BD a la que hay que pasarle una variables de entorno: Password del ROOT.

How to use this image

Start a `mysql` server instance

Starting a MySQL instance is simple:

```
$ docker run --name some-mysql -e MYSQL_ROOT_PASSWORD=my-secret-pw -d mysql:tag
```

... where `some-mysql` is the name you want to assign to your container, `my-secret-pw` is the password to be set for the MySQL root user and `tag` is the tag specifying the MySQL version you want. See the list above for relevant tags.

7.- Enlazar contenedores de redes personalizadas.

Ejemplo: Al crear dos contenedores y asociarlos una red personalizada, automáticamente se van a ver sin tener que hacer ningún link.

1.- Creamos el contenedor-1: mysql-server

docker run -d --name mysql-server --network red1 -e MYSQL_ROOT_PASSWORD=peque mysql

2.- Comprobamos que podemos conectarnos a ese contenedor.

docker exec -it mysql-server /bin/bash

mysql -u root -p → nos conectamos a mysql

mysql> show databases;

7.- Enlazar contenedores de redes personalizadas.

3.-. Creamos el contenedor-2: mysql-client

Este contenedor lo vamos a utilizar como cliente para conectarnos a mysql-server.

docker run -it --name mysql-client --network red1 mysql /bin/bash

-it → Queremos utilizar una consola
No indicamos Password, porque no queremos crear una nueva BD, queremos solo utilizarlo para conectarnos la servidor.

4.- Nos conectamos al servidor mysql de forma remota:

 # mysql -h mysql-server -u root -p

mysql>show databases;

7.- Enlazar contenedores de redes personalizadas.

CONCLUSIÓN:

Cuando utilizamos redes personalizadas, los contenedores asociadas a esas redes se ven automáticamente.

Resumen

1.- ¿Qué es mapear puertos?

2.- Tipos de redes que soporta Docker.

3.- Diferencia entre la red bridge por defecto y la red bridge personalizada.
¿Es necesario hacer un link para conectar dos contenedores?

Sugerencias/mejoras del tema

Sugerencias /mejoras del tema

Dudas

Referencias

- Albert Coronado
- robertoorayen.eu