Introducción a Docker

UD 05. Redes y volúmenes en Docker

Autor: Sergi García Barea

Actualizado Marzo 2022

Licencia

Nomenclatura

A lo largo de este tema se utilizarán distintos símbolos para distinguir elementos importantes dentro del contenido. Estos símbolos son:

📖 **Importante**

❕ **Atención**

💬 **Interesante**

UD05. REDES Y VOLÚMENES EN DOCKER

1. INTRODUCCIÓN

Hasta el momento hemos tratado con contenedores relativamente aislados y con poco contacto con la máquina anfitrión y con otros contenedores. En esta unidad hablaremos de cómo configurar redes entre contenedores y de cómo compartir datos y dotarlos de persistencia mediante volúmenes.

2. GESTIONANDO REDES

En este apartado veremos las redes predefinidas que posee Docker y cómo podemos crear y eliminar nuestras propias redes.

2.1 Redes predefinidas al instalar Docker

Al instalar Docker se establecen de forma predefinida 3 redes internas con las que podemos trabajar. Estas redes no se pueden eliminar y están siempre presentes:

- **Red "bridge":** es la red por defecto de cualquier contenedor, dando una IP propia. Para funcionar utiliza una interfaz de red virtual en la máquina anfitriona llamada "***docker0***".
- **Red "host":** si un contenedor utiliza esta red, estará utilizando la misma configuración de red de la máquina anfitriona.
- **Red "none"**: red no permite acceso a otras redes. Solo permite el acceso a la interfaz de loopback.

Podremos observar en la máquina anfitriona la interfaz "***docker0***" usando:

```
ifconfig docker0
```

o

```
ip a show docker0
```

```
sergi@ubuntu:~$ ifconfig docker0
docker0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.17.0.1  netmask 255.255.0.0  broadcast 172.17.255.255
        ether 02:42:49:d1:62:37  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

sergi@ubuntu:~$ ip a show docker0
5: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    link/ether 02:42:49:d1:62:37 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
sergi@ubuntu:~$
```

2.2 Crear redes internas en Docker

Si lo deseamos, podemos crear distintas redes independientes, ampliando las 3 redes por defecto.

¿Cuándo nos puede ser útil crear redes? En general, en contextos en los que queramos aislar las comunicaciones entre un conjunto de contenedores (por ejemplo, una red para pruebas de test y otra para desarrollo, o simular una red aislada con determinados servicios).

Para crear una red, simplemente usando el comando

```
docker network create redtest
```

Al crear la red, en la máquina anfitriona se creará una red virtual con formato "br-12 primeros dígitos del identificador". Lo vemos con un ejemplo:

```
sergi@ubuntu:~$ docker network create redtest
2e5ffdf1bb17f02f48fe4a4f1b2e057e795f2e16b79fe989774f4e4583d0b069
sergi@ubuntu:~$
sergi@ubuntu:~$ ifconfig br-2e5ffdf1bb17
br-2e5ffdf1bb17: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.18.0.1  netmask 255.255.0.0  broadcast 172.18.255.255
        ether 02:42:50:9a:bb:7c  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

sergi@ubuntu:~$
```

Al crearse la red "***redtest***" se nos devuelve un identificador. Usando los 12 primeros dígitos del identificador, podemos observar la red virtual en la máquina anfitriona llamada "***br-2e5ffdf1bb17***".

En este caso, hemos creado una red simple con los parámetros por defecto. Algunos de los parámetros configurables más interesantes son:

- "***--internal***": para red interna. Restringe el acceso desde el exterior.
- "***--gateway***": para indicar la puerta de enlace de la red.
- "***--ip-range***": delimita el rango de IPs asignables al contenedor.
- "***--ipv6***": habilita el uso de IPV6.
- "***--subnet***": define la subred en formato CIDR.
    - https://es.wikipedia.org/wiki/Classless_Inter-Domain_Routing

Para más información relacionada con el comando "***docker network create***" podéis consultar el siguiente enlace: https://docs.docker.com/engine/reference/commandline/network_create/

2.3 Inspeccionar y eliminar redes

Podemos observar las redes Docker de nuestro sistema con el comando:

```
docker network ls
```

```
sergi@ubuntu:~$ docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
124ec95c1c63   bridge    bridge    local
5dc3d12b1689   host      host      local
e1698d60fc4a   none      null      local
2e5ffdf1bb17   redtest   bridge    local
sergi@ubuntu:~$
```

También podemos obtener información de cada red usando

```
docker network inspect ID/NOMBRE-RED
```

```
sergi@ubuntu:~$ docker network inspect redtest
[
    {
        "Name": "redtest",
        "Id": "2e5ffdf1bb17f02f48fe4a4f1b2e057e795f2e16b79fe989774f4e4583d0b069",
        "Created": "2021-03-22T10:45:58.552673201+01:00",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.18.0.0/16",
                    "Gateway": "172.18.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {},
        "Options": {},
        "Labels": {}
    }
]
sergi@ubuntu:~$
```

También podemos eliminar redes mediante el comando "***docker network rm***". Si todo va bien, este comando nos mostrará el nombre de la red eliminada.

```
docker network rm ID/NOMBRE-RED
```

```
sergi@ubuntu:~$ docker network rm redtest
redtest
sergi@ubuntu:~$
```

Atención: para poder eliminar una red, es imprescindible que ningún contenedor en ejecución esté conectado a dicha red.

3. ASIGNANDO REDES A CONTENEDORES

3.1 Asignar la red al crear un contenedor

Cuando creamos un contenedor mediante los comandos "***docker run***" o "***docker create***", podemos especificar de que red va a formar parte **(por defecto, si no se indica, forma parte de "bridge").**

Por ejemplo para lanzar un contenedor conectado a la red "redtest" usamos el comando:

```
docker run -it --network redtest ubuntu /bin/bash
```

Al conectar un contenedor a una red, mediante un DNS interno implementado por Docker se nos permite referenciar a un contenedor usando su nombre de contenedor como un "hostname".

Si a un contenedor con nombre "***miserver***" es parte de una red, si alguien intenta resolver el nombre con "***dig***", "***ping***", etc. "miserver" se corresponderá a la IP de dicho contenedor.

Vamos a ver un ejemplo concreto, donde crearemos dos contenedores y haremos ping sobre ellos.

Creamos un contenedor "***prueba1***" en la red "***redtest***" con:

```
docker run -it --network redtest --name prueba1 alpine
```

Tras ello, salimos con "***exit***". Al salir, se parará el contenedor, así que para nuestra prueba, lo iniciamos de nuevo con el comando:

```
docker start prueba1
```

tras ello lanzamos el segundo contenedor "prueba2" con el comando:

```
docker run -it --network redtest --name prueba2 alpine
```

Ya dentro del contenedor "***prueba2***", lanzamos el comando "***ping***" para comprobar que el contenedor "***prueba1***" es accesible desde la máquina "***prueba2***":

```
ping prueba1
```

Podemos observar el ejemplo completo en esta captura:

```
sergi@ubuntu:~$ docker network create redtest
55f3c45170efff09602815d6e0f47be447ab6a06c2eab32a49685861f831055e
sergi@ubuntu:~$ docker run -it --network redtest --name prueba1 alpine
Unable to find image 'alpine:latest' locally
latest: Pulling from library/alpine
ba3557a56b15: Pull complete
Digest: sha256:a75afd8b57e7f34e4dad8d65e2c7ba2e1975c795ce1ee22fa34f8cf46f96a3be
Status: Downloaded newer image for alpine:latest
/ # exit
sergi@ubuntu:~$ docker start prueba1
prueba1
sergi@ubuntu:~$ docker ps
CONTAINER ID   IMAGE     COMMAND     CREATED          STATUS         PORTS     NAMES
afa38ecf9e96   alpine    "/bin/sh"   17 seconds ago   Up 4 seconds             prueba1
sergi@ubuntu:~$ docker run -it --network redtest --name prueba2 alpine
/ # ping prueba1
PING prueba1 (172.20.0.2): 56 data bytes
64 bytes from 172.20.0.2: seq=0 ttl=64 time=0.188 ms
64 bytes from 172.20.0.2: seq=1 ttl=64 time=0.121 ms
^C
--- prueba1 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 0.121/0.154/0.188 ms
/ #
```

Otro parámetro interesante relacionado con las redes que aplica a "docker run" es el parámetro "***--network-alias***". Este parámetro permite asignar un alias al contenedor en la red, de forma que al resolver ese alias asignado, se resuelva la ip.

Un ejemplo de uso:

```
docker run -it --network redtest --network-alias miservidor --name
prueba3 alpine
```

En este ejemplo, vemos cómo se asigna el alias "***miservidor***", por lo cual, si se intenta resolver el host "***miservidor***", obtendremos la ip de este contenedor.

3.2 Conectar y desconectar un contenedor de una red

Mediante el comando "***docker network connect/disconnect***" podemos conectar o desconectar un contenedor de una red. Un contenedor puede estar conectado a más de una red.

Por ejemplo con el comando:

```
docker network connect IDRED IDCONTENEDOR
```

Conectaremos el contenedor a una red existente. A continuación comentamos algunas opciones interesantes de "***docker network connect***":

- De forma similar a la comentada en el punto anterior, podemos establecer un alias en la red de este contenedor con el parámetro "***--alias***". Esto permitirá que al resolver el nombre DNS del alias, indique la ip del contenedor.
- En el ejemplo planteado, al contenedor se le asignará una ip entre las disponibles en el rango de la red. Si se quiere intentar a que se le asigne una ip fija, se puede usar la opción "***--ip***" para una IPV4 o "***--ip6***" para una IPV6.

Para desconectar, podemos usar simplemente la opción "***disconnect***", como vemos aquí:

```
docker network disconnect IDRED IDCONTENEDOR
```

Para más información relacionada con el comando "***docker network connect***" podéis consultar el siguiente enlace: https://docs.docker.com/engine/reference/commandline/network_connect/

4. PERSISTENCIA DE DATOS EN DOCKER

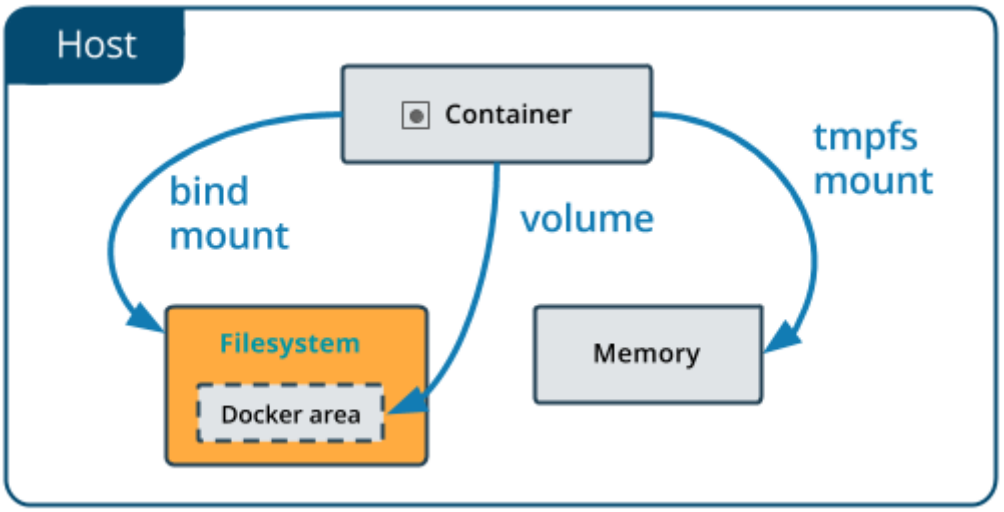
En este apartado veremos cómo gestionar la persistencia de datos en Docker usando distintos tipos de herramientas, conociendo sus casos de uso más típicos. También hablaremos de estrategias no orientadas a la persistencia como "***tmpfs***".

4.1 ¿Cuáles son las principales herramientas de persistencia en Docker?

Los principales métodos de persistencia en Docker son "***binding mount***", volúmenes y "***tmpfs***".

- "***Binding mount***": básicamente este tipo de persistencia consiste en "montar" un fichero o directorio de la máquina anfitrión en un fichero o directorio del contenedor. Este montaje se hace en el momento de crear el contenedor.
  - El fichero o directorio se indica en ambos casos con una **ruta absoluta** no tiene por qué existir en el contenedor (si no existe, se creará).
  - El rendimiento de este tipo de persistencia, a efectos prácticos, depende del sistema de ficheros y de las características del hardware de la máquina real. Una buena configuración según las necesidades del contenedor influirá en el rendimiento.
  - Estos volúmenes pueden ser usados por **varios contenedores simultáneamente**.
  - En sistemas Linux no suele haber diferencias de rendimiento respecto a volúmenes, pero en sistemas Windows y Mac el rendimiento es peor.
- ***Volúmenes Docker***: similar al "***binding mount***", solo que no especificamos en qué lugar está el directorio a montar en la máquina anfitrión, sino que solo damos un nombre del volumen para identificarlo. Esto nos proporciona algunas ventajas respecto al anterior:
  - Nos permite abstraernos de "donde" está realmente el volumen. Esta abstracción no es únicamente a nivel de directorio de la máquina anfitrión, sino incluso pudiendo estar el volumen en servidores remotos.
  - Obtienen mejor rendimiento que "***binding mount***" en sistemas Windows y Mac.
- "***Tmpfs***": este tipo de volumen utiliza el sistema de ficheros https://es.wikipedia.org/wiki/Tmpfs el cual se aloja en memoria y no en el disco, por lo cual ***no tiene persistencia***. A cambio, aumentaremos el rendimiento de entrada y salida.
  - Tiene algunas limitaciones:
    - Solo funciona en sistemas Linux.
    - No permite compartir el volumen entre contenedores.

Para entender mejor estas definiciones, vamos a observar y comentar esta imagen que ilustra su funcionamiento:



Fuente imagen: https://github.com/docker/docker.github.io/blob/master/storage/images/types-of-mounts-bind.png

- Observamos que "***Binding mount***" y volumen, apuntan al sistema de ficheros. La principal diferencia es que "***Binding mount"*** puede ir a cualquier parte del sistema y volumen apunta a un área concreta de Docker (donde se almacenan los volúmenes).
- En el caso de "***tmpfs***", vemos que va directamente a la memoria del sistema.

💬 **Interesante:** al igual que con el comando "mount" de sistemas Linux, si montamos un volumen en un directorio existente, lo que hará es "solapar" el contenido de ese directorio con el volumen montado. Esto puede ser útil, para por ejemplo, probar una nueva versión de una aplicación en un contenedor sin tener que rehacer la imagen.

4.2 Utilizando "Binding mount"

Este tipo de persistencia (al igual que el resto) es montada en el momento de crear el contenedor. Como veremos a la hora de montar todo tipo de contenedores, podemos usar dos parámetros distintos: "**-v**" que es más simple y "**--mount**" que es más explícito. Al utilizar estos comandos, debemos utilizar **rutas absolutas.**

```
docker run -d  -it  --name appcontainer   -v /home/sergi/target:/app
nginx:latest
```

o

```
docker run -d  -it --name appcontainer  --mount
type=bind,source=/home/sergi/target,target=/app nginx:latest
```

En ambos casos, este comando crea un contenedor llamado "***appcontainer***" donde la ruta del anfitrión "***/home/sergi/target***" se montará en el contenedor en "***/app***".

Más información en: https://docs.docker.com/storage/bind-mounts/

4.3 Creando volúmenes Docker

En estos apartados comentamos las principales acciones con volúmenes Docker. No obstante, si quieres saber más puedes consultar https://docs.docker.com/storage/volumes/.

4.3.1 Creando volúmen al crear contenedor

De manera similar a "***binding mount***", es posible crear volúmenes Docker usando "***-v***" o "***--mount***". Aquí vemos un ejemplo:

```
docker run -d  -it  --name appcontainer   -v mivolumen:/app
nginx:latest
```

o

```
docker run -d  -it --name appcontainer  --mount
source=mivolumen,target=/app nginx:latest
```

En este ejemplo, se ha montado el volumen gestionado por Docker "***mivolumen***" en el directorio "***/app***" del contenedor. Si el volumen "***mivolumen***" no existía previamente, lo crea.

4.3.2 Gestionando volúmenes

Este tipo de volúmenes, pueden crearse de forma separada, sin crear un contenedor asociado al mismo. Esto lo podemos hacer con el comando "***docker volume***":

- Con "***docker volume create mivolumen***" podemos crear el volumen vacío.
- Con "***docker volume ls***" podemos observar los volúmenes existentes.
- Con "***docker volume rm mivolumen***" puedes borrar un volumen, siempre que todo contenedor que lo utilice esté parado.

4.3.3 Poblando volúmenes

Si creamos un volumen directamente al lanzar un contenedor (no aplicable al caso de un contenedor vacío, pero creado previamente) y lo asociamos a un directorio que no está vacío, el contenido de ese directorio se copiará al volúmen. Por ejemplo si lanzamos

```
docker run -d  -it --name appcontainer  --mount
source=mivolumen,target=/app nginx:latest
```

Si el directorio "***/app***" tenía información, esta se copiará al volumen "***mivolumen***".

4.4 Creando volúmenes "tmpfs"

Como hemos comentado antes, los volúmenes "tmpfs" no tiene persistencia (se alojan en memoria) y tienen algunas limitaciones (solo pueden ser usados en sistemas Linux y no pueden ser compartidos entre contenedores). Para crear un volumen de este tipo podemos usar dos parámetros distintos: "**--tmpfs**" que es más simple y "**--mount**" que es más explícito.

Al utilizar estos comandos, debemos usar **rutas absolutas.**

```
docker run -d  -it --tmpfs /app nginx
```

o

```
docker run -d -it --mount type=tmpfs,destination=/app nginx
```

Este tipo de contenedores son útiles para:

- Almacenar ficheros que por algún motivo no quieres que se guarden ni en el contenedor ni en otros volúmenes (por ejemplo, datos sensibles).
- Al operar en memoria, puede ser útil utilizar este tipo de ficheros para acelerar operaciones de lectura/escritura. Por ejemplo, podemos almacenar unos juegos de prueba que vayan a ser leídos muchas veces, copiar contenido de una página a servir, etc.
    - Esto también puede hacerse usando binding mounts y una correcta configuración del sistema de ficheros en el anfitrión, usando el mismo "***tmpfs***" o "***ramfs***". Más información https://www.jamescoyle.net/how-to/943-create-a-ram-disk-in-linux

Para más información sobre "***tmpfs***" https://docs.docker.com/storage/tmpfs/

4.5 Copia de seguridad de un Volumen

Realizar una copia de seguridad de volúmenes Docker, es tan sencillo como realizar una copia de seguridad en el sistema anfitrión en los directorios pertinentes (o implementar en el sistema de ficheros elementos tales como discos espejo, etc.).

Aun así, si se desea realizar una copia de seguridad, por ejemplo, en un fichero "***.tar***" es posible realizarlo mediante comandos. Supongamos tengamos un "***contenedor1***", que utiliza el volumen "***misdatos***" montados en "***/datos***". Para realizar la copia de seguridad, seguiremos estos pasos:

En primer lugar, pararemos el contenedor.

```
docker stop contenedor1
```

y tras ello, lanzando la siguiente orden:

```
docker run --rm --volumes-from contenedor1 -v
/home/sergi/backup:/backup ubuntu bash -c "cd /datos && tar cvf
/backup/copiaseguridad.tar ."
```

Este comando, lanza un contenedor temporal (se borra al acabar con "***--rm***"), que monta los contenedores existentes en "***contenedor1***" (usando "***--volumes-from***") y realiza un "***binding mount***" del directorio del anfitrión "***/home/sergi/backup***" con el directorio "***/backup***" del contenedor. Tras ello, entra en la carpeta "***/datos***" (donde se monta el volumen) y guarda todo el contenido empaquetado en un fichero "***.tar***" en "***/backup***". Al acabar la ejecución, como "***/backup***" está montada en "***/home/sergi/backup***", ahí tendremos disponible la copia de seguridad.

4.6 Plugins para volúmenes en Docker

La funcionalidad de Docker puede extenderse mediante el uso de plugins. Entre ellos existe una gran multitud de plugins para ampliar los tipos de volúmenes soportados. Como curiosidad, decir que los plugins son distribuidos como imágenes en Docker Hub. Una forma de buscar plugins alojados en Docker Hub es mediante https://hub.docker.com/search?q=&type=plugin

Aquí https://docs.docker.com/engine/extend/ se muestra como ejemplo de uso, la instalación y configuración del plugin "***sshfs***" plugin, alojado en https://hub.docker.com/r/vieux/sshfs.

5. Bibliografía

[1] Docker Docs https://docs.docker.com/

[2] Understanding Docker Volumes

https://medium.com/bb-tutorials-and-thoughts/understanding-docker-volumes-with-an-example-d898cb5e40d7

[3] Docker Volumes: how to understand and get started

https://phoenixnap.com/kb/docker-volumes

6. Licencias de elementos externos utilizados

Figura 1: Imagen con licencia Apache 2.0. Fuente: https://github.com/docker/docker.github.io/blob/master/storage/images/types-of-mounts-bind.png