

ENTORNOS DE DESARROLLO

UD3.4 Redes y volúmenes

Contenidos

- ❖ ¿Qué es Docker y cómo funciona?
- ❖ Introducción a los contenedores y a Docker
- ❖ Instalación de Docker
- ❖ Primeros pasos con Docker
- ❖ Creación de imágenes personalizadas con Docker.
- ❖ **Redes y volúmenes en Docker.**
- ❖ Docker compose.
- ❖ Utilidades para gestionar Docker fácilmente.

Índice

❖ Gestión de redes

- Redes predefinidas al instalar Docker
- Crear redes internas en Docker
- Inspeccionar y eliminar redes

❖ Asignar redes a contenedores

- Asignar red al crear contenedor
- Conectar y desconectar un contenedor de una red

Índice

- ❖ **Persistencia de datos en Docker**
 - **Herramientas de persistencia**
 - **Uso de “Binding mount”**
 - **Crear volúmenes Docker**
 - **Crear volumen al crear contenedor**
 - **Gestión de volúmenes**
 - **Poblar volumen**
 - **Copia de seguridad de un volumen**

Gestión de redes

❖ 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”**: esta 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 0 ip a show docker0
```

Gestión de redes

❖ 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:

```
docker network create redtest
```

```
ifconfig br-12 primeros dígitos del identificador
```

Gestión de redes

❖ Crear redes internas en Docker

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:

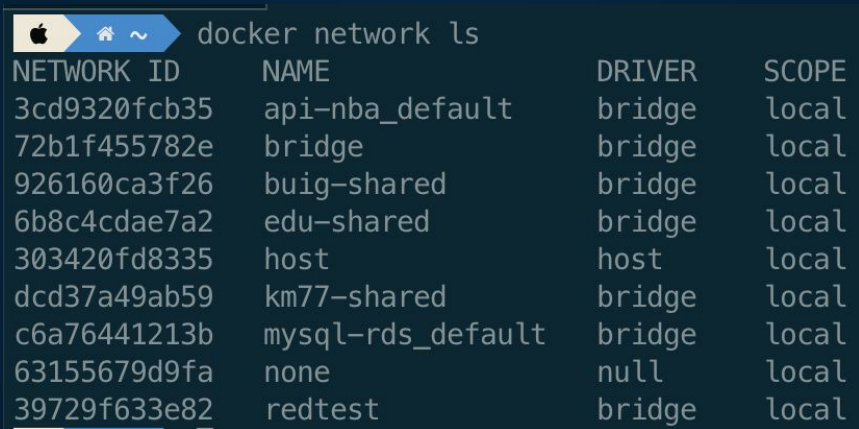
https://docs.docker.com/engine/reference/commandline/network_create/

Gestión de redes

❖ Inspeccionar y eliminar redes

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

`docker network ls`



```
docker network ls
```

NETWORK ID	NAME	DRIVER	SCOPE
3cd9320fcb35	api-nba_default	bridge	local
72b1f455782e	bridge	bridge	local
926160ca3f26	buig-shared	bridge	local
6b8c4cdae7a2	edu-shared	bridge	local
303420fd8335	host	host	local
dcd37a49ab59	km77-shared	bridge	local
c6a76441213b	mysql-rds_default	bridge	local
63155679d9fa	none	null	local
39729f633e82	redtest	bridge	local

Gestión de redes

❖ Inspeccionar y eliminar redes

También podemos obtener información de cada red usando

docker network inspect ID/NOMBRE-RED

```
docker network inspect redtest
[
  {
    "Name": "redtest",
    "Id": "39729f633e8287dfe95b1304376103ad0a5948c2205cd62d884c810eaa9dc685",
    "Created": "2022-11-15T08:41:57.854998144Z",
    "Scope": "local",
    "Driver": "bridge",
    "EnableIPv6": false,
    "IPAM": {
      "Driver": "default",
      "Options": {},
      "Config": [
        {
          "Subnet": "172.22.0.0/16",
          "Gateway": "172.22.0.1"
        }
      ]
    },
    "Internal": false,
    "Attachable": false,
    "Ingress": false,
    "ConfigFrom": {
      "Network": ""
    },
    "ConfigOnly": false,
    "Containers": {},
    "Options": {},
    "Labels": {}
  }
]
```

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

```
docker network rm redtest
redtest
```

Asignar redes a contenedores

❖ Asignar la red al crear un contenedor

Con “`docker run`” o “`docker create`”, podemos especificar de qué red va a formar parte (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.

Asignar redes a contenedores

❖ Asignar la red al crear un contenedor

Veamos un ejemplo:

Creamos dos contenedor “prueba1” y “prueba2” ambos en la red “redtest” con:

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

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

Tras crear el primero de ellos, salimos con “exit”. Al salir, se para el contenedor, así que lo iniciamos y ejecutamos el segundo comando:

```
docker start prueba1
```

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

```
ping prueba1
```

Asignar redes a contenedores

❖ Asignar la red al crear un contenedor

```
🍏 ~$ docker network create redtest
ed6589c34d0ab820e6a49db99a78e1b05411f4bc8935bb0bb3f79095decc38cc
🍏 ~$ docker run -it --network redtest --name prueba1 alpine
/ # exit
🍏 ~$ docker start prueba 1
Error response from daemon: No such container: prueba
Error response from daemon: Failed to inspect container 1: Error response from daemon: Multiple IDs found with provided prefix: 1
Error: failed to start containers: prueba, 1
🍏 ~$ docker start prueba1
prueba1
🍏 ~$ docker run -it --network redtest --name prueba2 alpine
/ # ping prueba1
PING prueba1 (172.25.0.2): 56 data bytes
64 bytes from 172.25.0.2: seq=0 ttl=64 time=0.216 ms
64 bytes from 172.25.0.2: seq=1 ttl=64 time=0.077 ms
^C
--- prueba1 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 0.077/0.146/0.216 ms
/ # exit
```

Asignar redes a contenedores

❖ Conectar y desconectar de una red

Con 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.

Para desconectar, podemos usar simplemente la opción “disconnect”, como vemos aquí:

```
docker network disconnect IDRED IDCONTENEDOR
```

Persistencia de datos

❖ Herramientas de persistencia

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 que no tiene porqué 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.
- 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**.

Persistencia de datos

◆ Herramientas de persistencia

Los principales métodos de persistencia en Docker son “`binding mount`”, volúmenes y “`tmpfs`”.

- **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.

Persistencia de datos

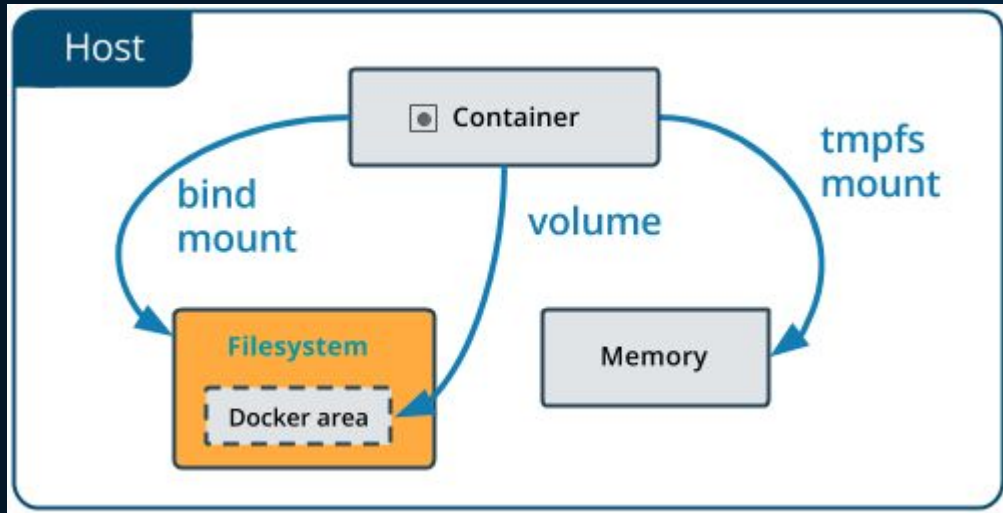
◆ Herramientas de persistencia

Los principales métodos de persistencia en Docker son “`binding mount`”, volúmenes y “`tmpfs`”.

- “**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.

Persistencia de datos

◆ Herramientas de persistencia



Observamos que “**Binding mount**” y **volume**, apuntan al sistema de ficheros.

- La principal diferencia es que “**Binding mount**” puede ir a cualquier parte del sistema y **volume** 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.

Persistencia de datos

❖ Uso de 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  
0
```

```
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”.

Persistencia de datos

❖ Crear volúmenes Docker

➤ Al crear un 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  
0
```

```
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.

Persistencia de datos

❖ Crear volúmenes Docker

➤ Gestión de 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.

Persistencia de datos

❖ Crear volúmenes Docker

➤ Poblar 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 volumen. 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”**.

Persistencia de datos

❖ 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.

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/(dam/daw)/backup:/backup ubuntu bash -c "cd /datos && tar cvf  
/backup/copiaseguridad.tar ."
```

Persistencia de datos

❖ Copia de seguridad de un volumen

Explicación del comando:

Este comando lanza un contenedor temporal (se borra al acabar con “`--rm`”).

Éste contenedor monta los volúmenes existentes en “`contenedor1`” (usando “`--volumes-from`”) y realiza un “binding mount” del directorio del anfitrión “`/home/ (dam/daw) /backup`” con el directorio “`/backup`” del contenedor.

Tras ello, entra en la carpeta “`/datos`” (donde se monta el volumen dentro del contenedor) y guarda todo el contenido empaquetado en un fichero “`.tar`” en “`/backup`”.

Al acabar la ejecución, como “`/backup`” está montada en “`/home/ (dam/daw) /backup`”, ahí tendremos disponible la copia de seguridad.