Prácticas Docker

1. Configurar puertos y conectar contenedores

- Vamos a crear dos contenedores de MariaDB, uno como cliente y otro como servidor
- Descargamos la imagen de MariaDB

```
docker pull mariadb
```

- Vamos a averiguar los puertos por los que escucha. Podemos usar el comando inspect sobre la imagen para localizarlos. Por supuesto, también podemos comprobar la documentación de Docker Hub para averiguarlo, donde además normalmente tenemos ejemplos

```
docker inspect --format='{{.Config.ExposedPorts}}' mariadb
map[3306/tcp:{} 33060/tcp:{}]
```

- Comprobamos que lo hace por los puertos 3306 y 33060
- Creamos un contenedor llamado mariadb1 que escuche por el puerto 4000 en el host. Por supuesto debemos asegurarnos de que ese puerto no esté ya en uso por alguna otra herramienta porque de lo contrario recibiríamos un error. Recordemos también que Mariadb necesita una variable de entorno para configurar la contraseña del usuario administrador

```
docker run -d -p 4000:3306 --name mariadb1 -e
MARIADB_ROOT_PASSWORD=lepanto mariadb
1ef4d55897a2478aead8f4a7aecd85bcce5a805c61c1e2fd423170eb94
642e79
```

- Comprobamos que funciona y los puertos por los que escucha

```
docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED
STATUS          PORTS
NAMES
1ef4d55897a2   mariadb     "docker-entrypoint.s…"   54
seconds ago   Up 51 seconds   33060/tcp, 0.0.0.0:4000-
>3306/tcp, :::4000->3306/tcp   mariadb1
```

- También podemos usar el comando PORT

```
docker port mariadb1
3306/tcp -> 0.0.0.0:4000
3306/tcp -> :::4000
```

- Ahora vamos a comprobar las redes que tenemos

```
docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
3d8689b8a3ea        bridge              bridge              local
81ce05a3ba16        host                host                local
17052d6bd175        none                null                local
```

- Comprobamos los contenedores que tiene la red Bridge. Al final , mas o menos tenemos nuestro contenedor mongo2, con la IP que se le ha asignado

```
docker network inspect bridge
….
…
..
"Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {

"1ef4d55897a2478aead8f4a7aecd85bcce5a805c61c1e2fd423170eb94642e79": {
                "Name": "mariadb1",
                "EndpointID":
"3b21df4ac2a74f0da0cc7520fa556c2c21e642371c54e45fe98fa4b2e5a0adac",
                "MacAddress": "02:42:ac:11:00:02",
                "IPv4Address": "172.17.0.2/16",
                "IPv6Address": ""
            }
…
…
…
```

- Podemos probar que llegamos al contenedor con un ping. (NOTA: en Windows no se puede hacer ping contra un contenedor de tipo Linux Container. Información en https://docs.docker.com/desktop/networking/)

```
ping 172.17.0.2
PING 172.17.0.2 (172.17.0.2) 56(84) bytes of data.
64 bytes from 172.17.0.2: icmp_seq=1 ttl=64 time=0.062 ms
64 bytes from 172.17.0.2: icmp_seq=2 ttl=64 time=0.060 ms
```

- También aparece el rango de IPs que tiene para asignar

```
"IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "172.17.0.0/16",
                    "Gateway": "172.17.0.1"
                }
            ]
```

- Podemos comprobarlo también a través del contenedor
- Entre la información que nos aparece, debe estar la de la RED

```
docker inspect mariadb1
….
….


},
        "NetworkSettings": {
            "Bridge": "",
            "SandboxID":
"2661e600959e2d6435ba92cd8ec3382d346560c0ba3c3321da43e831f4593588",
            "HairpinMode": false,
            "LinkLocalIPv6Address": "",
            "LinkLocalIPv6PrefixLen": 0,
            "Ports": {
                "3306/tcp": [
                    {
                        "HostIp": "0.0.0.0",
```

```
                        "HostPort": "4000"
                    },
                    {
                        "HostIp": "::",
                        "HostPort": "4000"
                    }
                ],
                "33060/tcp": null
            },
…..
..
..
```

- Si arrancamos otro contenedor Mariadb, vamos a ver los datos que les indica. Recordemos que debemos poner otro puerto distinto para el host, por ejemplo el 4001. Le llamamos por ejemplo “mariadb2”

```
docker run -d -p 4001:3306 --name mariadb2 -e
MARIADB_ROOT_PASSWORD=lepanto mariadb
8001f59cf78943cd98c02aa7116471ca8c3009f6200274ac3a71b8205e9c7d56
```

- Si inspeccionamos la red, debemos tener los dos contenedores, con sus direcciones IPs correspondientes

```
"Containers": {

"1ef4d55897a2478aead8f4a7aecd85bcce5a805c61c1e2fd423170eb94642e79": {
                "Name": "mariadb1",
                "EndpointID":
"3b21df4ac2a74f0da0cc7520fa556c2c21e642371c54e45fe98fa4b2e5a0adac",
                "MacAddress": "02:42:ac:11:00:02",
                "IPv4Address": "172.17.0.2/16",
                "IPv6Address": ""
            },

"8001f59cf78943cd98c02aa7116471ca8c3009f6200274ac3a71b8205e9c7d56": {
                "Name": "mariadb2",
                "EndpointID":
"053fa4f9932888771bb14108f53a774d3826cc7128ca2e5ec8dded2399762388",
                "MacAddress": "02:42:ac:11:00:03",
                "IPv4Address": "172.17.0.3/16",
                "IPv6Address": ""
```

```
            }
```

- Vamos a probar que llegamos desde un contendor a otro
- Abrimos una bash contra mariadb2

```
docker exec -it mariadb2 bash
root@8001f59cf789:/#
```

- Instalamos el ping

```
apt-get update
apt-get install -y iputils-ping
```

- Hacemos un ping contra mariadb1 . En mi caso es el 172.17.0.2, tal y como me ha indicado la red

```
ping 172.17.0.2
PING 172.17.0.2 (172.17.0.2) 56(84) bytes of data.
64 bytes from 172.17.0.2: icmp_seq=1 ttl=64 time=0.099 ms
64 bytes from 172.17.0.2: icmp_seq=2 ttl=64 time=0.074 ms
64 bytes from 172.17.0.2: icmp_seq=3 ttl=64 time=0.057 ms
```

- Como prueba final, vamos a conectarnos como cliente, desde mariadb2 a la base de datos Mariadb que tenemos en el contenedor mariadb1. Es decir, vamos a comprobar que los contenedores de una red concreta se pueden conectar entre sí sin problemas.
- Desde la Shell que hemos abierto en mariadb1 ponemos el siguiente comando, que es la Shell de mariadb

```
mariadb -u root -p -h 172.17.0.2
Enter password:
Welcome to the Mariadb monitor.  Commands end with ; or \g.
Your Mariadb connection id is 8
Server version: 8.0.28 Mariadb Community Server - GPL

Copyright (c) 2000, 2022, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.
```

```
Type 'help;' or '\h' for help. Type '\c' to clear the current input
statement.


mariadb>
```

- Evidentemente, la IP y el puerto pertenecen a “mariadb1”. NOTA: ¿por qué no hemos usado el puerto 4000 para conectarnos, ya que es el que hemos configurado al crear el contenedor?. Al no poner ningún puerto, se ha conectado por el estándar qué es el 3306. En el video siguiente explicaremos este concepto para que os quede claro. No es lo mismo conectarse desde fuera de la red de docker a conectarse desde dentro de la red de docker qué es lo que estamos haciendo en este caso ya que estamos accediendo de un contenedor a otro
- Debería conectarme sin problemas
- Ya estamos conectados. Podemos probar por ejemplo viendo las Bases de datos de mariadb1

```
mariadb> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mariadb              |
| performance_schema |
| sys                |
+--------------------+
4 rows in set (0.01 sec)
```

- Cerramos los dos contenedores