

Prácticas Docker

1. Crear redes y asociarlas a contenedores

- Comprobamos las redes que tenemos en este momento:

```
docker network ls
```

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

- Creamos una nueva red de tipo bridge

```
docker network create net1
```

```
8e83268b846d8f6937d13383afa1791bf30faccb1a3b1a02248736ff3ee14c1c
```

```
network ls
```

NETWORK ID	NAME	DRIVER	SCOPE
3d8689b8a3ea	bridge	bridge	local
81ce05a3ba16	host	host	local
8e83268b846d	net1	bridge	local
17052d6bd175	none	null	local

- La inspeccionamos

```
docker network inspect net1
```

```
[
  {
    "Name": "net1",
    "Id":
    "8e83268b846d8f6937d13383afa1791bf30faccb1a3b1a02248736ff3ee14c1c",
    "Created": "2018-03-22T23:40:59.678173713+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": {}
}
]
```

- Vemos que tiene el rango de direcciones "Subnet": "172.18.0.0/16",
- A nivel físico del sistema operativo ha debido crear una nueva conexión para la red. La que tiene el nombre de docker0 es la que corresponde a la red bridge predeterminada de Docker
- (NOTA. El comando "nmcli" es Red Hat-Fedora, en vuestro Linux puede variar. Podemos usar ifconfig por ejemplo para ver la información)

nmcli con

NOMBRE DISPOSITIVO	UUID	TIPO
br-8e83268b846d br-8e83268b846d	edc8f727-36ec-472e-95c3-6eeacb28b657	bridge
docker0 docker0	5770a886-453c-4836-bef6-aaeb077788c8	bridge
static-ens37 ens37	daf9e2af-7ccb-420e-b42e-f1ffe41fc58d	802-3-ethernet
virbr0 virbr0	2b4ecf59-343a-4820-bd2f-c557f9ccdefe	bridge

- El nombre de dispositivo que se ha puesto a la nueva conexión coincide con el ID que Docker ha asignado a la red

```
docker network ls
```

NETWORK ID	NAME	DRIVER	SCOPE
3d8689b8a3ea	bridge	bridge	local
81ce05a3ba16	host	host	local
8e83268b846d	net1	bridge	local
17052d6bd175	none	null	local

- Creamos un contenedor mongodb con esa red.

```
docker run -d -p 27017:27017 --network net1 --name mongo1 mongo
e909eff322a201a20e61d9a1dc821df2f10f493a8ba0e7a95e011ca613f2a804
```

- Comprobamos que funciona y los puertos por los que escucha

```
docker ps
CONTAINER ID        IMAGE               COMMAND
CREATED            STATUS             PORTS
NAMES
e909eff322a2        mongo              "docker-entrypoint.s..." 59
seconds ago        Up 58 seconds      0.0.0.0:27017->27017/tcp    mongo1
```

- Comprobamos en la red que efectivamente se ha unido el contenedor a la red y podemos comprobar también la IP que le ha facilitado

```
docker network inspect net1
"Containers": {
  "e909eff322a201a20e61d9a1dc821df2f10f493a8ba0e7a95e011ca613f2a804": {
    "Name": "mongo1",
    "EndpointID":
"f6d2880394e239f57a7f63da5d364c1fc29684ce008cc6a21c01732fb13bf751",
    "MacAddress": "02:42:ac:12:00:02",
    "IPv4Address": "172.18.0.2/16",
    "IPv6Address": ""
  }
},
```

- Vamos a comprobar ahora que cuando creamos una red personalizada, Docker genera un DNS interno que nos permite acceder a los contenedores con el nombre que le pusimos con la opción `--name` al crearlos

- Si entramos dentro del contenedor “mongo1” (tu debes usar el nombre que has usado para crearlo)

```
docker exec -it mongo1 bash
```

- Instalamos ping

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

- Si probamos a realizar un ping a “mongo1” (y recordemos que no hemos configurado nada en la red para poner ese nombre). Vemos que funciona bien y localiza la IP

```
ping mongo1  
  
PING mongo1 (172.18.0.2) 56(84) bytes of data.  
64 bytes from e909eff322a2 (172.18.0.2): icmp_seq=1 ttl=64  
time=0.055 ms  
64 bytes from e909eff322a2 (172.18.0.2): icmp_seq=2 ttl=64  
time=0.044 ms
```

- Desde otro terminal, vamos a crear un Segundo contenedor en esa red

```
docker run -d -p 27018:27017 --name mongo2 --network net1  
mongo  
  
c1b5a5882d0db3554927b3f022ff7f83c67d313ef9eee81d9eed4ef6f5  
b3fb27  
  
docker ps  
  
CONTAINER ID          IMAGE          COMMAND  
CREATED              STATUS        PORTS  
NAMES  
  
c1b5a5882d0d         mongo         "docker-  
entrypoint.s..."  40 seconds ago  Up 38 seconds  
0.0.0.0:27018->27017/tcp  mongo2  
  
e909eff322a2         mongo         "docker-  
entrypoint.s..."  27 minutes ago  Up 27 minutes  
0.0.0.0:27017->27017/tcp  mongo1
```

- Desde la bash en la que estamos dentro de mongo1 hacemos un ping a mongo2

- Debe funcionar perfectamente, ya que la red personalizada se encarga de gestionar los nombre de los contenedores. Y como hemos visto, nosotros no hemos tenido que configurar nada.

```
ping mongo2
```

```
PING mongo2 (172.18.0.3) 56(84) bytes of data.  
64 bytes from mongo2.net1 (172.18.0.3): icmp_seq=1 ttl=64  
time=0.150 ms  
64 bytes from mongo2.net1 (172.18.0.3): icmp_seq=2 ttl=64  
time=0.061 ms  
64 bytes from mongo2.net1 (172.18.0.3): icmp_seq=3 ttl=64  
time=0.060 ms
```

- Ahora, vamos a crear una segunda red. Además le vamos a poner dos características:
 - Que su subred sea la 172.30.0.0/16
 - Que los contenedores se les asocie una IP que comience a partir de la 172.30.10.0/24

```
docker network create net2 --subnet=172.30.0.0/16 --ip-  
range=172.30.10.0/24
```

```
31ed5d42621519c775729042167d62ff4b34cb76af13a4007168a57b18  
2e8704
```

- Podemos comprobar haciendo un inspect a la red

```
docker network inspect red2
```

```
...  
...  
...  
"IPAM": {  
    "Driver": "default",  
    "Options": {},  
    "Config": [  
        {  
            "Subnet": "172.30.0.0/16",  
            "IPRange": "172.30.10.0/24"  
        }  
    ]  
}
```

```
    },
    "Internal": false,
    ...
    ...
```

- Vamos a crear un contenedor mongo asociado a esa red

```
docker run -d --name mongo3 --network net2 mongo
f03d25490e1765e62d8b955cceb563844e7df8a7dc13f0ed2ee1c36e2a
32af35
```

docker ps

CONTAINER ID	IMAGE	COMMAND
CREATED	STATUS	PORTS
NAMES		
f03d25490e17 entrypoint.s..." 27017/tcp	mongo 38 seconds ago mongo3	"docker- Up 37 seconds
c1b5a5882d0d entrypoint.s..." 0.0.0.0:27018->27017/tcp	mongo 9 minutes ago mongo2	"docker- Up 9 minutes
e909eff322a2 entrypoint.s..." 0.0.0.0:27017->27017/tcp	mongo 36 minutes ago mongo1	"docker- Up 36 minutes

- Comprobamos con inspect la red que tiene al contenedor. Mandamos la información a un fichero porque salen bastantes cosas

```
docker inspect mongo3 > mongo3.json
```

- Editamos el fichero y buscamos la zona de la red. Podemos ver que está asociado a la net2 y con la dirección IP adecuada, a partir de la 10

```
"Networks": {
    "net2": {
        "IPAMConfig": null,
        "Links": null,
        "Aliases": [
            "f03d25490e17"
        ],
    },
}
```

```

        "NetworkID":
        "31ed5d42621519c775729042167d62ff4b34cb76af13a4007168a57b1
        82e8704",
        "EndpointID":
        "85f58410558a16e1523875874ff9942d655d4a30eeaea71ae5defb426
        544e4f5",
        "Gateway": "172.30.10.0",
        "IPAddress": "172.30.10.1",
        "IPPrefixLen": 16,
        "IPv6Gateway": "",
        "GlobalIPv6Address": "",
        "GlobalIPv6PrefixLen": 0,
        "MacAddress": "02:42:ac:1e:0a:01",
        "DriverOpts": null
    }

```

- Si intentamos acceder desde la bash que tenemos abierta en “mongo1” a mongo3 no la encuentra porque no están en la misma subred
- Por último, vamos a asociar este contenedor a la red “net1”

```
docker network connect net1 mongo3
```

- Volvemos a hacer un inspect del contenedor. Podemos comprobar que ahora tiene asociadas la net1 y la net2 y que tiene una IP de cada una de ellas

```

"net1": {
    "IPAMConfig": {},
    "Links": null,
    "Aliases": [
        "f03d25490e17"
    ],
    "NetworkID":
    "8e83268b846d8f6937d13383afa1791bf30facb1a3b1a02248736ff3
    ee14c1c",
    "EndpointID":
    "55ba7b5b56569f456b8bc8796467b89ac1018e4efcabd3230ce330690
    0a39981",
    "Gateway": "172.18.0.1",
    "IPAddress": "172.18.0.4",

```

```

        "IPPrefixLen": 16,
        "IPv6Gateway": "",
        "GlobalIPv6Address": "",
        "GlobalIPv6PrefixLen": 0,
        "MacAddress": "02:42:ac:12:00:04",
        "DriverOpts": null
    },
    "net2": {
        "IPAMConfig": null,
        "Links": null,
        "Aliases": [
            "f03d25490e17"
        ],
        "NetworkID":
"31ed5d42621519c775729042167d62ff4b34cb76af13a4007168a57b1
82e8704",
        "EndpointID":
"85f58410558a16e1523875874ff9942d655d4a30eeaea71ae5defb426
544e4f5",
        "Gateway": "172.30.10.0",
        "IPAddress": "172.30.10.1",
        "IPPrefixLen": 16,
        "IPv6Gateway": "",
        "GlobalIPv6Address": "",
        "GlobalIPv6PrefixLen": 0,
        "MacAddress": "02:42:ac:1e:0a:01",
        "DriverOpts": null
    }

```

- Si volvemos a la bash que teníamos abierta en mongo1, debemos poder acceder ahora a mongo3

ping mongo3

PING mongo3 (172.18.0.4) 56(84) bytes of data.

64 bytes from mongo3.net1 (172.18.0.4): icmp_seq=1 ttl=64
time=0.178 ms

64 bytes from mongo3.net1 (172.18.0.4): icmp_seq=2 ttl=64
time=0.059 ms


```
64 bytes from mongo3.net1 (172.18.0.4): icmp_seq=3 ttl=64  
time=0.058 ms
```

- Desconectamos mongo3 de la red “net1”

```
docker network disconnect net1 mongo3
```