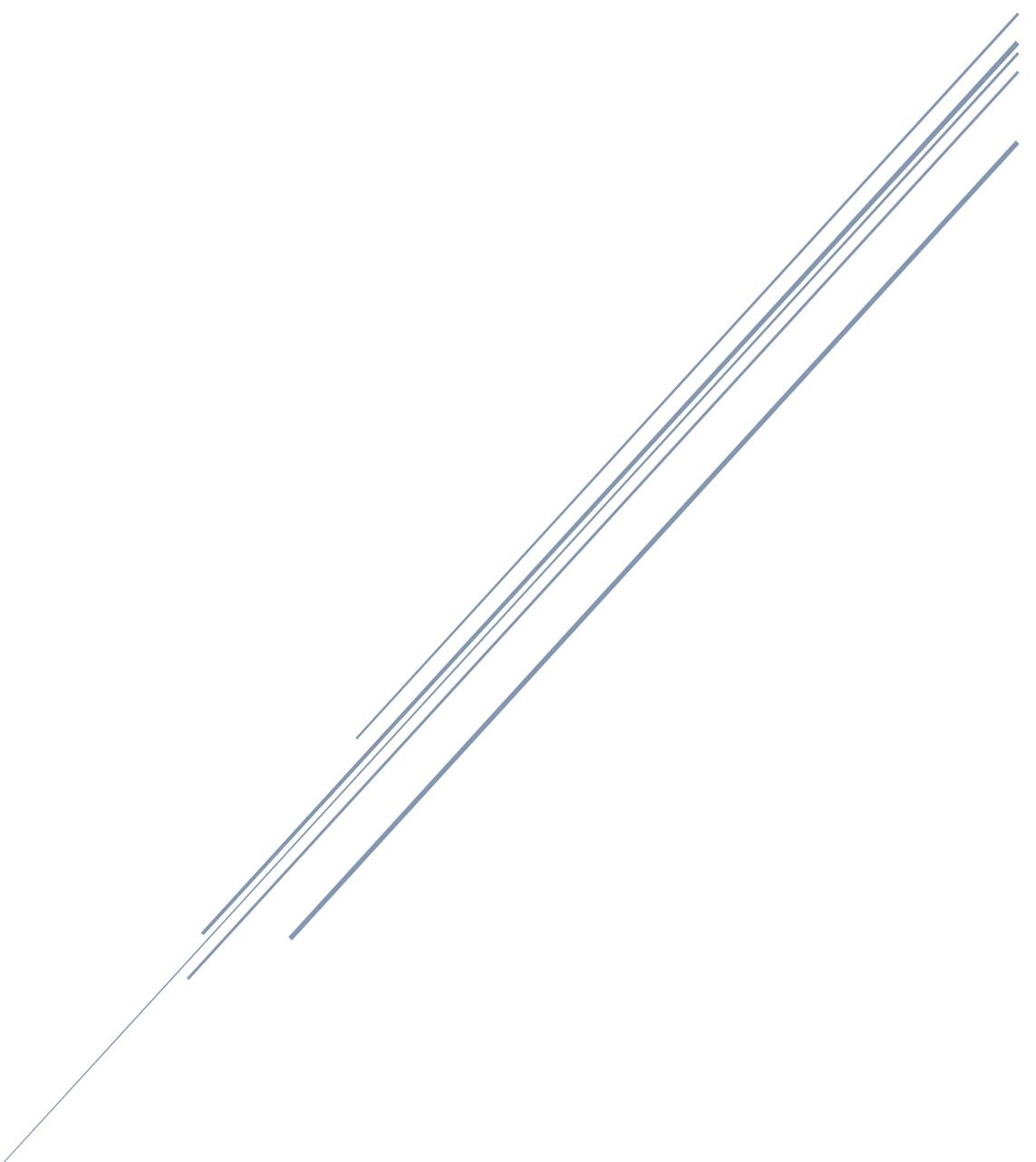


DOCKER - REDES

Actividad 4



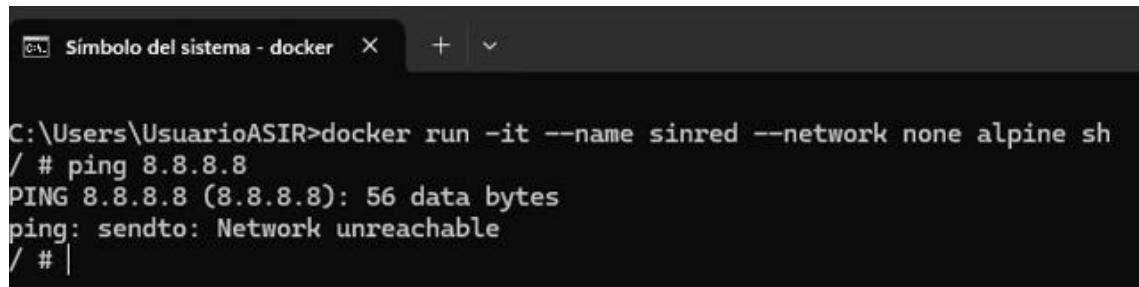
Cristóbal Suárez Abad
Optativa – 2º ASIR

Parte 1

1. Crea un contenedor con tipo de red None

docker run -it --name sinred --network none alpine sh

2. Intenta hacer ping a google.com. ¿Qué ocurre y explica por qué?



```
C:\Users\UsuarioASIR>docker run -it --name sinred --network none alpine sh
/ # ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8): 56 data bytes
ping: sendto: Network unreachable
/ # |
```

No puede hacer ping porque con la opción “--network none” se aísla al contenedor: no configurada una IP (sin acceso a Internet ni a otros contenedores alojados en el mismo host).

Parte 2

1. Levantar un contenedor de nginx con puerto publicado. Accede desde el navegador para verificar que está funcionando.

```
docker run -d --name nginx02 -p 8080:80 nginx
```

```
C:\Users\UsuarioASIR>docker run -d --name nginx02 -p 8080:80 nginx
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
250b90fb2b9a: Pull complete
b459da543435: Pull complete
5d8ea9f4c626: Pull complete
54e822d8ee0c: Pull complete
8da8ed3552af: Pull complete
58d144c4badd: Pull complete
Digest: sha256:f79cde317d4d172a392978344034eed6dff5728a8e6d7a42f507504c23ecf8b8
Status: Downloaded newer image for nginx:latest
9d682ea330705b207a2de1b4ecd713d289dc4000c4d355b5dde2da3d8e969c76
```

2. Intenta hacer ping a google.com. ¿Qué ocurre y explica por qué?

Hay que instalar “ping”.

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

```
C:\Users\UsuarioASIR>docker exec -it nginx02 bash
root@9d682ea33070:/# apt-get update -y
root@9d682ea33070:/# apt-get install -y iputils-ping
```

Pero si, tiene acceso a internet. Esto ocurre, porque por defecto, si no se especifica con la opción “--network”, los contenedores se crean con un bridge, el cual les otorga una IP (dentro de este 172.17.0.0/16), la cual le permite comunicarse con otros contenedores y con Internet.

```
root@9d682ea33070:/# ping google.com
PING google.com (142.250.184.14) 56(84) bytes of data.
64 bytes from 142.250.184.14: icmp_seq=1 ttl=63 time=30.4 ms
64 bytes from 142.250.184.14: icmp_seq=2 ttl=63 time=25.8 ms
64 bytes from 142.250.184.14: icmp_seq=3 ttl=63 time=25.4 ms
64 bytes from 142.250.184.14: icmp_seq=4 ttl=63 time=112 ms
64 bytes from 142.250.184.14: icmp_seq=5 ttl=63 time=27.2 ms
```

3. Ver puertos mapeados

docker port nginx02

```
C:\Users\UsuarioASIR>docker port nginx02
80/tcp -> 0.0.0.0:8080
80/tcp -> [::]:8080
```

O

docker container ls --format "table {{.ID}}\t{{.Names}}\t{{.Ports}}" -a

```
C:\Users\UsuarioASIR>docker container ls --format "table {{.ID}}\t{{.Names}}\t{{.Ports}}" -a
CONTAINER ID NAMES PORTS
9d682ea33070 nginx02 0.0.0.0:8080->80/tcp, [::]:8080->80/tcp
996728e2f996 sinred
22c1127f880b mariadb-server
f93f8395b1d2 nextcloud-server
80f654bd7dd7 mediawiki3
8c1d00fe6f09 mediawiki2
a9a4c52cfe73 mediawiki1
```

<https://stackoverflow.com/questions/49821358/how-to-list-exposed-port-of-all-containers>

4. Lista redes disponibles

docker network ls

```
C:\Users\UsuarioASIR>docker network ls
NETWORK ID     NAME      DRIVER      SCOPE
d18bff22427f  bridge    bridge      local
5cc043015edb  host      host       local
2b3789c1e06f  none      null       local
```

5. Inspeccionar la red bridge. Explica lo que ves

docker network inspect bridge

```
C:\Users\UsuarioASIR>docker network inspect bridge
[
    {
        "Name": "bridge",
        "Id": "d18bff22427fc41499d8f4260d4a8b28c0f091401d8c572ea943320ef8eab05",
        "Created": "2025-10-08T08:49:09.069060599Z",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv4": true,
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "172.17.0.0/16",
                    "Gateway": "172.17.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "9d682ea330705b207a2de1b4ecd713d289dc4000c4d355b5dde2da3d8e969c76": {
                "Name": "nginx02",
                "EndpointID": "8fd664beb5aedec5c9327c05d09935a74e5317cfbd270206068677bb35d3d557",
                "MacAddress": "4e:eb:e4:6b:d7:33",
                "IPv4Address": "172.17.0.2/16",
                "IPv6Address": ""
            }
        },
        "Options": {
            "com.docker.network.bridge.default_bridge": "true",
            "com.docker.network.bridge.enable_icc": "true",
            "com.docker.network.bridge.enable_ip_masquerade": "true",
            "com.docker.network.bridge.host_binding_ipv4": "0.0.0.0",
            "com.docker.network.bridge.name": "docker0",
            "com.docker.network.driver.mtu": "1500"
        },
        "Labels": {}
    }
]
```

Se muestra:

- Nombre de la red.
- ID
- Fecha de creación.
- Alcance (“scope”) = “local” indica que la red solo existe en el host.
- IPv4 habilitado e IPv6 deshabilitado.
- IPAM: (IP Address Management).
 - o Driver: default → el gestor de direcciones IP que usa Docker por defecto.
 - Subnet: la red donde se asignan las IP de los contenedores (172.17.0.0/16).
 - Gateway.
- Internal: false → la red puede comunicarse con el exterior (no está aislada).
- Attachable: false → los contenedores no se pueden conectar manualmente a esta red con docker network connect (solo a redes personalizadas).
- Ingress: false → no es una red de entrada especial para balanceo de carga de Swarm.
- Containers: Aquí Docker lista los **contenedores conectados** a la red:
 - **Name:** el contenedor se llama nginx02.
 - **IPv4Address:** IP asignada al contenedor (172.17.0.2).
 - **MacAddress:** dirección MAC de la interfaz virtual dentro del contenedor.
- **Opciones:**
 - o default_bridge: true → es la red bridge por defecto del sistema.
 - o enable_icc: true → permite comunicación entre contenedores dentro de la misma red.
 - o enable_ip_masquerade: true → activa el NAT (enmascaramiento de IP), permitiendo que los contenedores salgan a Internet usando la IP del host.
 - o host_binding_ipv4: 0.0.0.0 → el host puede escuchar en cualquier interfaz.
 - o bridge.name: docker0 → es la interfaz de red del host que actúa como puente virtual.
 - o mtu: 1500 → tamaño máximo de los paquetes (igual que una red Ethernet estándar).

Parte 3

1. Crear una red personalizada

```
docker network create mired03
```

```
docker network ls
```

```
C:\Users\UsuarioASIR>docker network create mired03
b4115719e2b754cf31db64ac739bcd015e73b0bfa06f0ef0e6bcc1e95654016

C:\Users\UsuarioASIR>dockert network ls
"dockert" no se reconoce como un comando interno o externo,
programa o archivo por lotes ejecutable.

C:\Users\UsuarioASIR>docker network ls
NETWORK ID      NAME      DRIVER      SCOPE
d18bff22427f    bridge    bridge      local
5cc043015edb    host      host       local
b4115719e2b7    mired03   bridge      local
2b3789c1e06f    none      null       local
```

2. Correr dos contenedores en esa red. Nginx como servidor y alpine como cliente.

```
docker run -d --name nginxserver03 --network mired03 nginx
```

```
docker run -it --name alpinecliente03 --network mired03 alpine sh
```

```
C:\Users\UsuarioASIR>docker run -d --name nginxserver03 --network mired03 nginx
21bafe17263ded8e9f61a95a317d814a3e54264ca45785fade5058e0a6164158

C:\Users\UsuarioASIR>docker run -it --name alpinecliente03 --network mired03 alpine sh
```

3. Intenta hacer ping a google.com. ¿Qué ocurre y explica por qué?

```
C:\Users\UsuarioASIR>docker run -it --name alpinecliente03 --network mired03 alpine sh
/ # ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8): 56 data bytes
64 bytes from 8.8.8.8: seq=0 ttl=63 time=13.001 ms
64 bytes from 8.8.8.8: seq=1 ttl=63 time=17.051 ms
64 bytes from 8.8.8.8: seq=2 ttl=63 time=14.632 ms
64 bytes from 8.8.8.8: seq=3 ttl=63 time=19.529 ms
64 bytes from 8.8.8.8: seq=4 ttl=63 time=12.953 ms
64 bytes from 8.8.8.8: seq=5 ttl=63 time=16.157 ms
64 bytes from 8.8.8.8: seq=6 ttl=63 time=55.626 ms
64 bytes from 8.8.8.8: seq=7 ttl=63 time=50.782 ms
64 bytes from 8.8.8.8: seq=8 ttl=63 time=19.479 ms
64 bytes from 8.8.8.8: seq=9 ttl=63 time=35.956 ms
^C
--- 8.8.8.8 ping statistics ---
10 packets transmitted, 10 packets received, 0% packet loss
round-trip min/avg/max = 12.953/25.516/55.626 ms
```

Hace ping correctamente. Porque a la hora de crear la red personalizada, si no especificamos parámetros, toma de referencia el bridge por defecto de Docker y esos parámetros funcionan.

4. Desde el cliente, probar la conectividad con servidor. ¿Qué ocurre?

```
/ # ping nginxserver03
PING nginxserver03 (172.18.0.2): 56 data bytes
64 bytes from 172.18.0.2: seq=0 ttl=64 time=0.277 ms
64 bytes from 172.18.0.2: seq=1 ttl=64 time=0.148 ms
64 bytes from 172.18.0.2: seq=2 ttl=64 time=0.106 ms
64 bytes from 172.18.0.2: seq=3 ttl=64 time=0.150 ms
64 bytes from 172.18.0.2: seq=4 ttl=64 time=0.157 ms
```

Si, se puede acceder perfectamente. Ambos están en la misma subnet. 172.18.0.0/16

```
19 packets transmitted, 19 packets received, 0% packet loss
round-trip min/avg/max = 0.098/0.127/0.277 ms
/ # ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: eth0@if8: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue state UP
    link/ether 2a:7d:d6:76:68:b1 brd ff:ff:ff:ff:ff:ff
    inet 172.18.0.3/16 brd 172.18.255.255 scope global eth0
        valid_lft forever preferred_lft forever
/ # |
```

5. Añade otro cliente alpine a la misma red

```
docker run -it --name alpinecliente03_b --network mired03 alpine sh
```

6. Desde cliente2, conectarse a servidor1. ¿Qué ocurre?

```
C:\Users\UsuarioASIR>docker run -it --name alpinecliente03_b --network mired03 alpine sh
/ # ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: eth0@if24: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue state UP
    link/ether 32:60:5d:1d:15:4b brd ff:ff:ff:ff:ff:ff
    inet 172.18.0.4/16 brd 172.18.255.255 scope global eth0
        valid_lft forever preferred_lft forever
/ # ping nginxserver03
PING nginxserver03 (172.18.0.2): 56 data bytes
64 bytes from 172.18.0.2: seq=0 ttl=64 time=0.266 ms
64 bytes from 172.18.0.2: seq=1 ttl=64 time=0.105 ms
64 bytes from 172.18.0.2: seq=2 ttl=64 time=0.083 ms
64 bytes from 172.18.0.2: seq=3 ttl=64 time=0.149 ms
```

Hace ping perfectamente.

7. ¿Puedes acceder desde el navegador al servidor que hemos creado Nginx ? Si no es así, que tendríamos que modificar

Servidor nginxserver03 (172.18.0.2): No, porque no hay ningún puerto mapeado.

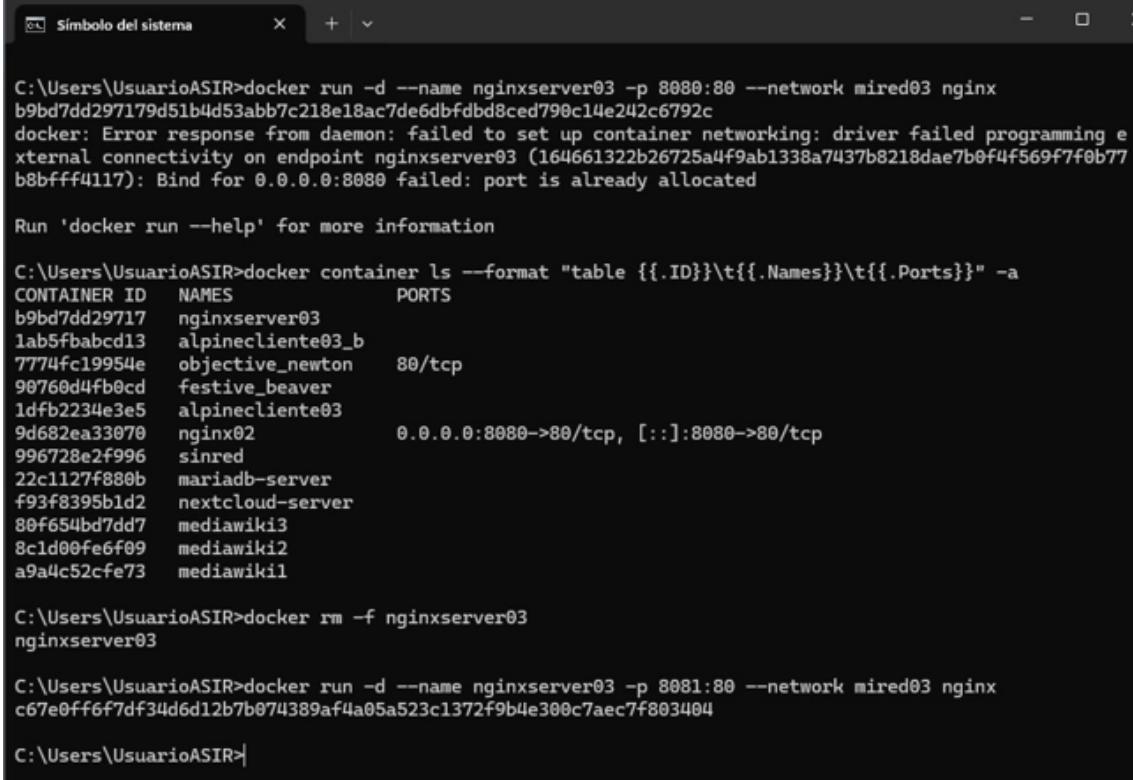
En vez de modificarlo, lo cual se puede hacer, es mucho más fácil borrarlo y hacerlo de nuevo, sobre todo porque no tenemos configurado nada aún en este contenedor.

Borrarlo: **docker rm -f nginxserver03**

Nuevo:

docker run -d --name nginxserver03 -p 8081:80 --network mired03 nginx

Usamos el Puerto 8081 porque el 8080 ya está siendo usado en el anfitrión por “nginx02”.



```
C:\Users\UsuarioASIR>docker run -d --name nginxserver03 -p 8080:80 --network mired03 nginx
b9bd7dd297179d51b4d53abb7c218e18ac7de6dbfdbd8ced790c14e242c6792c
docker: Error response from daemon: failed to set up container networking: driver failed programming e
xternal connectivity on endpoint nginxserver03 (164661322b26725a4f9ab1338a7437b8218dae7b0f4f569f7f0b77
b8bfff4117): Bind for 0.0.0.0:8080 failed: port is already allocated

Run 'docker run --help' for more information

C:\Users\UsuarioASIR>docker container ls --format "table {{.ID}}\t{{.Names}}\t{{.Ports}}" -a
CONTAINER ID NAMES PORTS
b9bd7dd29717 nginxserver03
1ab5fbabcd13 alpinecliente03_b
7774fc19954e objective_newton 80/tcp
90760d4fb0cd festive_beaver
1dfb2234e3e5 alpinecliente03
9d682ea33070 nginx02 0.0.0.0:8080->80/tcp, [::]:8080->80/tcp
996728e2f996 sinred
22c1127f880b mariadb-server
f93f8395b1d2 nextcloud-server
80f654bd7dd7 mediawiki3
8c1d00fe6f09 mediawiki2
a9a4c52cfe73 mediawiki1

C:\Users\UsuarioASIR>docker rm -f nginxserver03

C:\Users\UsuarioASIR>docker run -d --name nginxserver03 -p 8081:80 --network mired03 nginx
c67e0ff6f7df34d6d12b7b074389af4a05a523c1372f9b4e300c7aec7f803404

C:\Users\UsuarioASIR>
```

<http://localhost:8081>

The screenshot shows a web browser window with the URL `http://localhost:8081` in the address bar. The main content is a white page with the heading **Welcome to nginx!**. Below it, a message states: "If you see this page, the nginx web server is successfully installed and working. Further configuration is required." It also includes links to [nginx.org](#) for online documentation and support, and [nginx.com](#) for commercial support. A footer message says "Thank you for using nginx."

8. ¿Puedes acceder al contenedor que creamos en la parte 1 desde estos contenedores clientes y servidor? Explica porque

No, porque ese contenedor no tiene asignada una IP.

The screenshot shows the Docker interface for a container named **sinred**. The container ID is `996728e2f996` and it is based on the `alpine:latest` image. The **Exec** tab is selected, showing a terminal session. The terminal output displays the command `/ # ip a` followed by the output of the `ip` command, which shows the configuration of the `lo` interface (loopback). A purple banner at the top of the terminal area reads: "Docker Debug brings the tools you need to debug your container with one click." and "Requires a paid Docker subscription. [Learn more](#)".

```
/ # ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
/ #
```

Parte 4

1. Levantar un servidor nginx con una IP específica

En este caso, vamos a crear primero una “network” donde estableceremos la Subnet donde se encontrará la IP estática que estableceremos más adelante¹.

```
docker network create --subnet=172.20.0.0/16 mired04
```

```
C:\Users\UsuarioASIR>docker network create --subnet=172.20.0.0/16 mired04
05c5705d18e1846462669334c2ffd73a6853b6bba6e448c850be1c3d07ad6446
```

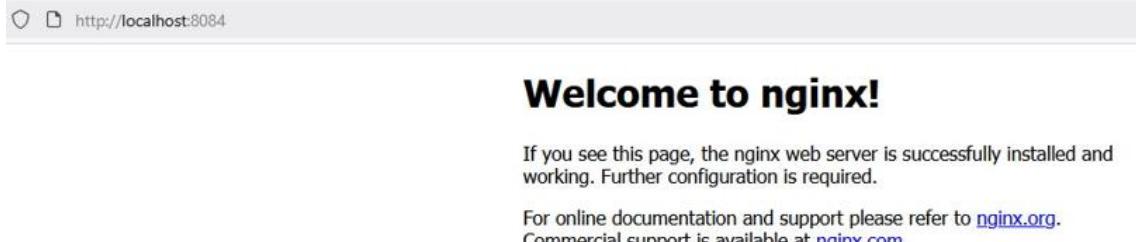
```
docker run -d --name nginxserver04 --network mired04 --ip 172.20.0.10 nginx
```

```
C:\Users\UsuarioASIR>docker run -d --name nginxserver04 --network mired04 --ip 172.20.0.10 nginx
692fa8751cc101c9d0ec2f688604a113e6e986ele81ca6f2ca7949c205c99bb8
```

2. Levanta un nuevo contenedor nginx y mapea múltiples puertos. Comprueba desde tu navegador que está bien configurado

```
docker run -d --name nginxmultiport2 -p 8084:80 -p 8445:443 nginx
```

```
C:\Users\UsuarioASIR>docker run -d --name nginxmultiport2 -p 8084:80 -p 8445:443 nginx
0d72364ed96ccbd0883bbcacf3c709a405b0a14a10ea97da43c28f7feae7710
```



¹ <https://stackoverflow.com/questions/27937185/assign-static-ip-to-docker-container>

Parte 5

1. Desconecta y elimina todas las redes

<https://docs.docker.com/reference/cli/docker/network/prune/>

docker network prune

Puedes usar opción de filter y force (force hace lo mismo, pero no pregunta).

2. Elimina todos los contenedores

<https://stackoverflow.com/questions/35240278/how-to-remove-all-docker-containers>

sudo docker rm -f \$(sudo docker ps -a -q)

ATENCIÓN: la opción “-f” fuerza el borrado. Incluso si se está usando.

3. Elimina todas las imágenes

4. <https://stackoverflow.com/questions/35240278/how-to-remove-all-docker-containers>

sudo docker image remove -f \$(sudo docker images -a -q)

ATENCIÓN: la opción “-f” fuerza el borrado. Incluso si se está usando.

5. ¿Existe algún comando para hacer estos últimos tres pasos en un solo comando? Explica cómo funciona

<https://www.digitalocean.com/community/tutorials/how-to-remove-docker-images-containers-and-volumes>

docker system prune (lo borra todo)

docker system prune -a (borra incluso más).

Parte 6

1. *¿Qué diferencia hay entre usar la red bridge por defecto y una red personalizada?*

Las redes personalizadas (“custom bridges”) permiten tener DNS integrado, lo que permite que los contenedores se comuniquen con sus nombres, sin tener que especificar IPs, algo que con el bridge por defecto de Docker no se puede hacer.

Se pueden aislar los contenedores por grupos con las redes personalizadas, evitando que se comuniquen con los contenedores de otros bridges, lo cual aumenta la seguridad.

Al contrario que en el bridge por defecto, en las redes personalizadas, los contenedores se pueden añadir y quitar sin necesidad de detenerlos.

En las redes personalizadas se pueden ajustar parámetros que no se pueden modificar en el bridge por defecto.

<https://docs.docker.com/engine/network/drivers/bridge/>

2. *¿Por qué es importante el aislamiento de redes en entornos productivos?*

Porque evita comunicaciones no autorizadas entre contenedores.

Permiten crear entornos separados para diferentes aplicaciones y proyectos, aumentando el control durante el proceso de desarrollo.

Al tener los contenedores separados en diferentes redes, se mejora la gestión del tráfico, evitando congestiones.

<https://labex.io/es/tutorials/docker-how-to-isolate-networks-between-docker-containers-417536>

3. *¿Qué sucede si dos contenedores tienen servicios en el mismo puerto? Haz un ejemplo práctico y explica que ocurre*

ERROR: Bind for 0.0.0.0:8083 failed: port is already allocated

```
C:\Users\UsuarioASIR>docker run -d --name nginxserver03 -p 8080:80 --network mired03 nginx
b9bd7dd297179d51b4d53abb7c218e18ac7de6dbfdbd8ced790c14e242c6792c
docker: Error response from daemon: failed to set up container networking: driver failed programming e
xternal connectivity on endpoint nginxserver03 (164661322b26725a4f9ab1338a7437b8218dae7b0f4f569f7f0b77
b8bfff4117): Bind for 0.0.0.0:8080 failed: port is already allocated

Run 'docker run --help' for more information

C:\Users\UsuarioASIR>docker container ls --format "table {{.ID}}\t{{.Names}}\t{{.Ports}}" -a
CONTAINER ID      NAMES          PORTS
b9bd7dd29717    nginxserver03
lab5fbabcd13   alpinecliente03_b
7774fc19954e   objective_newton    80/tcp
90760d4fb0cd   festive_beaver
1dfb2234e3e5   alpinecliente03
9d682ea33070   nginx02           0.0.0.0:8080->80/tcp, [::]:8080->80/tcp
996728e2f996   sinred
22c1127f880b   mariadb-server
f93f8395b1d2   nextcloud-server
80f654bd7dd7   mediawiki3
8c1d00fe6f09   mediawiki2
a9a4c52cf73   mediawiki1

C:\Users\UsuarioASIR>docker rm -f nginxserver03
nginxserver03

C:\Users\UsuarioASIR>docker run -d --name nginxserver03 -p 8081:80 --network mired03 nginx
c67e0ff6f7df34d6d12b7b074389af4a05a523c1372f9b4e300c7aec7f803404
```

El contenedor se crea, pero no puede usar el puerto especificado, porque está siendo usado por el anterior.