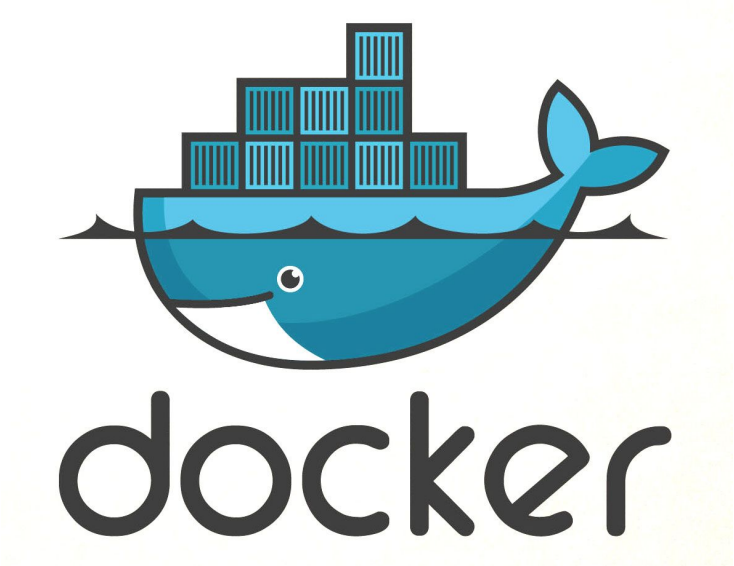
Redes



Repaso - *Redes*

Docker permite crear **redes virtuales.**

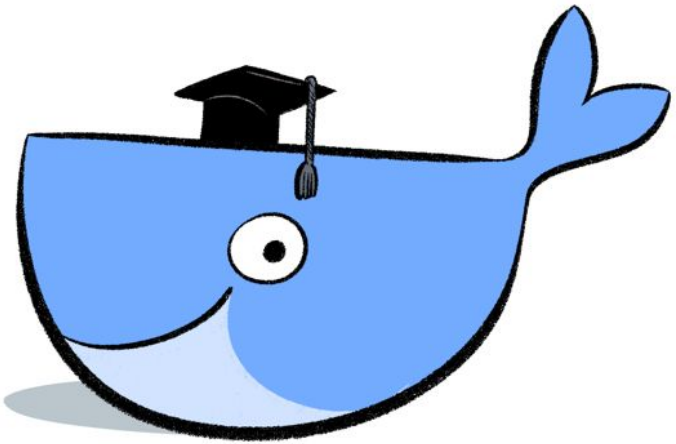
Las redes virtuales sirven para:

- Comunicar contenedores **entre ellos**.
- Comunicar contenedores **con el anfitrión**.
- Comunicar contenedores **con el exterior**.

Contenido

1. Redes por defecto
2. Conectar contenedores a redes
3. Gestión de redes bridge creadas por el usuario

Redes por defecto

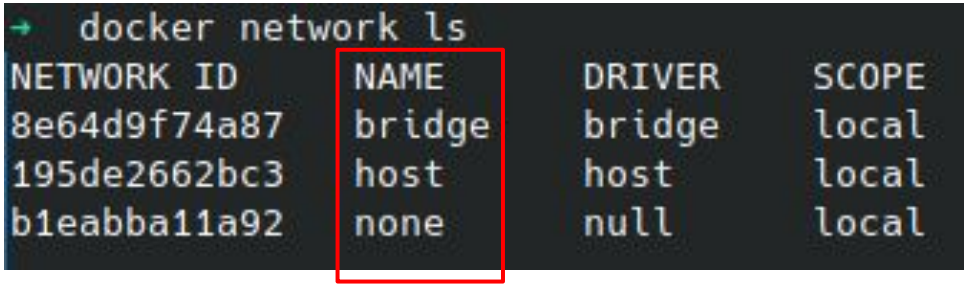


Redes por defecto

Docker crea automáticamente unas redes cuando se instala para que los contenedores puedan comunicarse entre sí, con el host y con el exterior.

Estas redes no se pueden eliminar.

En concreto, se crean 3 redes:



Redes por defecto - Red bridge

Es la red a la que se conectan los contenedores **por defecto**.

Utiliza la interfaz virtual “**docker0**” del anfitrión.

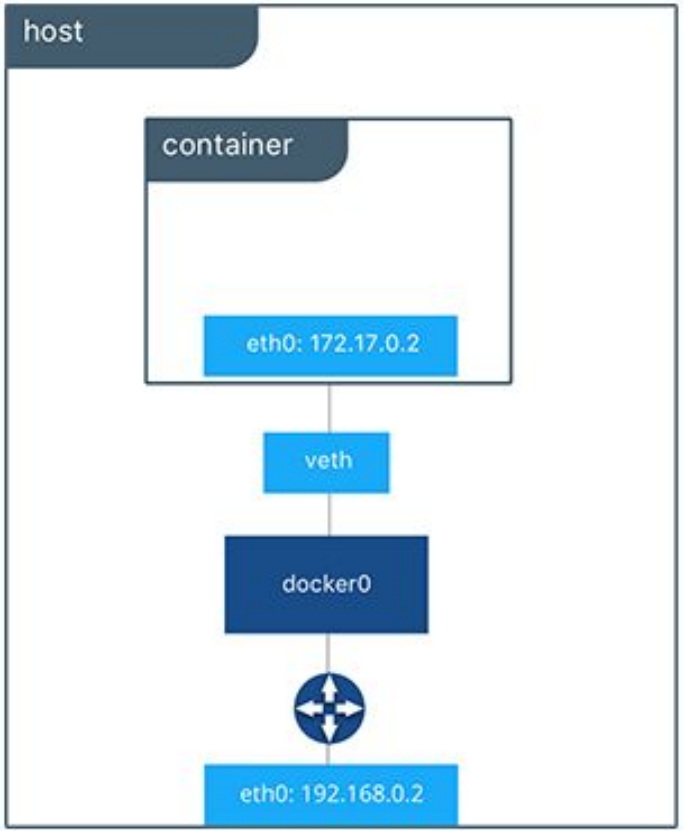
```
→  ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eno1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 50:65:f3:28:df:84 brd ff:ff:ff:ff:ff:ff
    altname enp0s25
    inet 10.2.2.48/24 brd 10.2.2.255 scope global dynamic noprefixroute eno1
       valid_lft 40403sec preferred_lft 40403sec
    inet6 fe80::728a:9daa:a9:1d34/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    link/ether 02:42:93:55:99:9b 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
```

Redes por defecto - Red bridge

El direccionamiento por defecto es: **172.17.0.0/16**.

Los contenedores conectados a esta red que quieren exponer puertos al exterior tienen que usar la opción **-p**.

Docker crea las reglas **iptables** necesarios para gestionarla.



Redes por defecto - Red bridge

```
➜  docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED         STATUS         PORTS     NAMES
0f6102f1f077   ubuntu    "bash"    8 minutes ago   Up 8 minutes             wizardly_shirley
```

```
➜  docker inspect -f '{{json .NetworkSettings.Networks}}' wizardly_shirley | json_pp
{
   "bridge" : {
      "Aliases" : null,
      "DriverOpts" : null,
      "EndpointID" : "0034bb7eee7a1f3bb840506e0347c3f8ccd6100545259ee39bddf4de73256cca",
      "Gateway" : "172.17.0.1",
      "GlobalIPv6Address" : "",
      "GlobalIPv6PrefixLen" : 0,
      "IPAMConfig" : null,
      "IPAddress" : "172.17.0.2",
      "IPPrefixLen" : 16,
      "IPv6Gateway" : "",
      "Links" : null,
      "MacAddress" : "02:42:ac:11:00:02",
      "NetworkID" : "1745b4d8250d7fe9cdc5223815441123bd3e64d20a4516ea8502c01eb6637ca4"
   }
}
```

Redes por defecto - Red host

Si un conectanedor se conecta a la red host es como si utilizara directamente la **red del anfitrión**.

El contenedor **no tiene IP** propia, sino que usa la del anfitrión.

Los puertos utilizados son directamente los del anfitrión, no hay que mapearlos.

Es como si los servicios se ejecutaran directamente los del anfitrión.

Redes por defecto - Red host

Por ejemplo lanzamos un nginx conectado a la red host, u no es necesario mapear el puerto 80, sino que es accesible directamente desde el anfitrión.

```
→  docker run -d --network host nginx
e3dbeb129668dec19ddc91b2928fd5e68f75b4982a7d7d9a063822c7403dc782
```

Redes por defecto - Red host

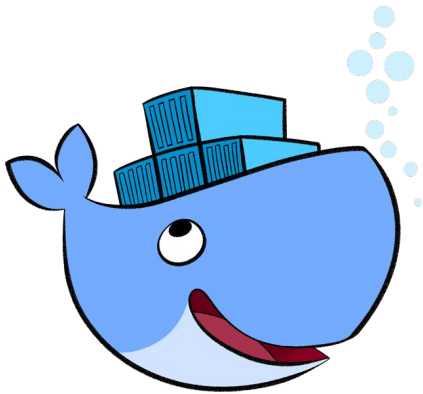
```
➜  docker inspect -f '{{json .NetworkSettings.Networks}}' xenodochial_noether | json_pp
{
   "host" : {
      "Aliases" : null,
      "DriverOpts" : null,
      "EndpointID" : "96c0f6bd0f548b137cb010dbfe24e1f492f9d24698e6e3ec7bc8d75c7d14d86c",
      "Gateway" : "",
      "GlobalIPv6Address" : "",
      "GlobalIPv6PrefixLen" : 0,
      "IPAMConfig" : null,
      "IPAddress" : "",
      "IPPrefixLen" : 0,
      "IPv6Gateway" : "",
      "Links" : null,
      "MacAddress" : "",
      "NetworkID" : "195de2662bc330d032c8a93b0f0c20269e2786b065accb2546a5798af1c3f8e7"
   }
}
```

Redes por defecto - Red none

Si un conectanedor se conecta a la red none, no tiene conexión a ninguna red.

La única interfaz de red que tiene es *loopback.*

Conectar contenedores a redes



Conectar contenedores a redes

1. Al crear el contenedor.
2. Con contenedor ya creado a una red.

Conectar redes al crear un contenedor

docker **run** y docker **create** admiten el parámetro **--network** para especificar la red a la que conectar el contenedor.

Por defecto se conecta a la red **“bridge”**.

Dentro de la misma red definida por el usuario, los contenedores se pueden referenciar por sus **nombres**, ya que Docker tienen un DNS que resuelve la IP.

En el caso de la red **“bridge”**, para que haya resolución de nombres se tiene que usar **--link**.

Conectar redes al crear un contenedor

```
➜  docker network create RedPrueba
d34c748a55fdcdd53b62714969ce95530b37cb7b5156854047c2eaa8b81821fc
```

```
➜  docker run -d --name webserver --network RedPrueba nginx
5c89b6ec509f2c90f381227b5b9aeef8a8e40ab8d43758cedd508441557ea68e
```

```
➜  docker run -it --network RedPrueba bash
bash-5.1# ping webserver
PING webserver (172.18.0.2): 56 data bytes
64 bytes from 172.18.0.2: seq=0 ttl=64 time=0.185 ms
64 bytes from 172.18.0.2: seq=1 ttl=64 time=0.074 ms
^C
--- webserver ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 0.074/0.129/0.185 ms
bash-5.1# exit
exit
```

Conectar redes al crear un contenedor

También se puede asignar un alias al contendor dentro de la red con la opción **--network-alias**:

```
➜  docker run -d --name webserver --network RedPrueba --network-alias nginxserver nginx
ec2d06c772d08abf8f56e7b0de1745b4685c674a9612c7329bc0d10f11d2df77
```

```
➜  docker run -it --network RedPrueba bash
bash-5.1# ping nginxserver
PING nginxserver (172.18.0.2): 56 data bytes
64 bytes from 172.18.0.2: seq=0 ttl=64 time=0.119 ms
64 bytes from 172.18.0.2: seq=1 ttl=64 time=0.078 ms
64 bytes from 172.18.0.2: seq=2 ttl=64 time=0.168 ms
^C
--- nginxserver ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 0.078/0.121/0.168 ms
bash-5.1# ping webserver
PING webserver (172.18.0.2): 56 data bytes
64 bytes from 172.18.0.2: seq=0 ttl=64 time=0.098 ms
```

Conectar redes a un contenedor ya creado.

docker connect/disconnect red contenedor

Algunas opciones interesantes son:

- **--alias**: para especificar un alias del contenedor en la red.
- **--ip**: si se quiere especificar una ip4 fija.
- **--ip6**: si se quiere especificar una ip6 fija.

Conectar redes a un contenedor ya creado.

```
→  docker run -d --name webserver nginx
70c481a16c3fe25cc2759a8de2744e183fbc88460b1909f223f1c70eb59d8d1e
```

```
→  docker run --rm -it --network RedPrueba bash
bash-5.1# ping webserver
ping: bad address 'webserver'
```

```
→  docker network connect RedPrueba webserver
```

```
→  docker run --rm -it --network RedPrueba bash
bash-5.1# ping webserver
PING webserver (172.18.0.2): 56 data bytes
64 bytes from 172.18.0.2: seq=0 ttl=64 time=0.129 ms
```

Gestión de redes bridge creadas por el usuario

Gestión de redes bridge creadas por el usuario

1. Redes bridge.
2. Red “bridge” por defecto vs redes bridge.
3. Gestión de redes bridge.

Redes bridge

Además de las 3 redes que se crean por defecto, se pueden **crear** más redes de tipo **bridge**.

Su uso es para implementar redes **internas/privadas** entre varios contenedores.

Estas redes bridge creadas **no** son exactamente iguales que la red “brigde” por defecto.

En sistemas **en producción**, se **aconseja** su uso en lugar de la red “bridge” por defecto.

Red “bridge” por defecto vs redes bridge

Red “*bridge*” por defecto

- Se crea automáticamente al instalar Docker.
- No se puede eliminar.
- Se asigna por defecto a los contenedores si no se explicita una red.
- Solo proporciona resolución de nombres si se usa --link, que está depreciado. Además es resolución estática.

Redes *bridge*

- Tienen que crearse de forma explícita.
- Se pueden eliminar.
- No se asignan por defecto a los contenedores. Debe explicitarse.
- Proporcionan resolución DNS.

Red “bridge” por defecto vs redes bridge

Red “*bridge*” por defecto

- No permite conexión en caliente de contenedores. Tienen que apagarse antes.
- Al ser la red por defecto, proporciona menos aislamiento y control.
- Los contenedores comparten ciertas valiables de entorno, lo que puede ser problemático.

Redes *bridge*

- Permiten conexión en caliente de contenedores.
- Proporcionan más aislamiento y control.
- Los contenedores no comparten variables de entorno.

Red “bridge” por defecto vs redes bridge

```
➜  docker run -d --name webserver -e NGINX_PASS=12345 nginx
878c4f67c36e4efb2a985a402a952b9766b928aefd8229b426726e990b119cfa
```

Se observa:

- Resolución de nombres estática
- Compartición de variables de entorno

```
➜  docker run --rm -it --link webserver bash
bash-5.1# cat /etc/hosts
127.0.0.1       localhost
::1     localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
172.17.0.2      webserver 878c4f67c36e
172.17.0.3      b595f3d92bc1
```

```
bash-5.1# env
HOSTNAME=b595f3d92bc1
WEBSERVER_PORT_80_TCP_ADDR=172.17.0.2
WEBSERVER_PORT_80_TCP_PORT=80
WEBSERVER_ENV_NGINX_VERSION=1.21.6
PWD=/
WEBSERVER_PORT_80_TCP=tcp://172.17.0.2:80
_BASH_BASELINE_PATCH=16
WEBSERVER_PORT=tcp://172.17.0.2:80
HOME=/root
_BASH_VERSION=5.1.16
_BASH_BASELINE=5.1.16
WEBSERVER_PORT_80_TCP_PROTO=tcp
_BASH_LATEST_PATCH=16
TERM=xterm
SHLVL=1
WEBSERVER_ENV_NGINX_PASS=12345
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin
WEBSERVER_ENV_NJS_VERSION=0.7.2
WEBSERVER_ENV_PKG_RELEASE=1~bullseye
WEBSERVER_NAME=/upbeat_brown/webserver
_=/usr/bin/env
```

Red “bridge” por defecto vs redes bridge

Ambos contenedores **comparten DNS con el host**.

bash

```
→  docker run --rm -it --link webserver bash
bash-5.1# cat /etc/resolv.conf
nameserver 10.2.1.254
search lliurex
```

webserver

```
→  docker exec webserver cat /etc/resolv.conf
nameserver 10.2.1.254
search lliurex
```

Red “bridge” por defecto vs redes bridge

host

```
→  resolvectl status
Global
       Protocols: -LLMNR -mDNS -DNSOverTLS DNSSEC=no/unsupported
resolv.conf mode: stub

Link 2 (enp2s0)
    Current Scopes: DNS
         Protocols: +DefaultRoute +LLMNR -mDNS -DNSOverTLS DNSSEC=no/unsupported
Current DNS Server: 10.2.1.254
       DNS Servers: 10.2.1.254
        DNS Domain: lliurex
```

Red “bridge” por defecto vs redes bridge

```
→  docker run -d --name webserver -e NGINX_PASS=12345 --network RedPrueba nginx
75e9f71d20d79273e67f1a909566f50e04f8dbe28aed0d04ad7af5c4c11cda28
```

Se observa:

- Resolución de nombres dinámica
- No hay compartición de variables de entorno

```
→  docker run --rm -it --network RedPrueba bash
bash-5.1# cat /etc/hosts
127.0.0.1       localhost
::1     localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
172.18.0.3      948b37a58169
```

```
bash-5.1# ping webserver
PING webserver (172.18.0.2): 56 data bytes
64 bytes from 172.18.0.2: seq=0 ttl=64 time=0.297 ms
```

```
bash-5.1# env
HOSTNAME=948b37a58169
PWD=/
_BASH_BASELINE_PATCH=16
HOME=/root
_BASH_VERSION=5.1.16
_BASH_BASELINE=5.1.16
_BASH_LATEST_PATCH=16
TERM=xterm
SHLVL=1
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin
_=/usr/bin/env
```

Red “bridge” por defecto vs redes bridge

Ambos contenedores **comparten un DNS** que es **diferente al del host**.

bash

```
bash-5.1# cat /etc/resolv.conf
search lliurex
nameserver 127.0.0.11
options edns0 trust-ad ndots:0
```

webserver

```
→  docker exec webserver cat /etc/resolv.conf
search lliurex
nameserver 127.0.0.11
options edns0 trust-ad ndots:0
```

Este DNS hace forward por defecto al del host, a no ser que se especifique otro con el parámetro **--dns** de docker create o run.

Red “bridge” por defecto vs redes bridge

host

```
→  resolvectl status
Global
       Protocols: -LLMNR -mDNS -DNSOverTLS DNSSEC=no/unsupported
resolv.conf mode: stub

Link 2 (enp2s0)
    Current Scopes: DNS
         Protocols: +DefaultRoute +LLMNR -mDNS -DNSOverTLS DNSSEC=no/unsupported
Current DNS Server: 10.2.1.254
       DNS Servers: 10.2.1.254
        DNS Domain: lliurex
```

Gestión de redes bridge - Comandos

docker network *comando*

create *nombre*	Crea una red
ls	Lista todas las redes
rm *nombre*	Elimina una red
prune	Elimina las redes no usados por ningún contenedor
inspect	Proporciona información sobre una red
connect / disconnect	Conecta/Desconecta un contenedor a una red

Gestión de redes bridge - Comandos

docker network create *nombre*

```
→  docker network create RedPrueba
970c0f93c3dcdb580a9f60c5eba53e0733108e12e2f7c473f9899b085ffc1a36
```

```
→  docker network create RedIntranet
bc3c6e9170a1b592828308000042f917f246bf2b4e6321e7ac8c58e6ad7a49d09
```

Gestión de redes bridge - Comandos

docker network ls

```
➜  docker network ls
NETWORK ID     NAME          DRIVER    SCOPE
bc3c6e9170a1   RedIntranet   bridge    local
970c0f93c3dc   RedPrueba     bridge    local
fbac97951f5c   bridge        bridge    local
195de2662bc3   host          host      local
b1eabba11a92   none          null      local
```

Gestión de redes bridge - Comandos

docker network inspect name

```
→ docker inspect RedIntranet
[
    {
        "Name": "RedIntranet",
        "Id": "bc3c6e9170a1b59282830800042f917f246bf2b4e6321e7ac8c58e6ad7a49d09",
        "Created": "2022-04-13T17:38:09.086288056+02:00",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.19.0.0/16",
                    "Gateway": "172.19.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {},
        "Options": {},
        "Labels": {}
    }
]
```

```
→ docker network inspect RedPrueba
[
    {
        "Name": "RedPrueba",
        "Id": "970c0f93c3dcdb580a9f60c5eba53e0733108e12e2f7c473f9899b085ffc1a36",
        "Created": "2022-04-13T17:37:35.662840335+02:00",
        "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": {}
    }
]
```

Gestión de redes bridge - Comandos

docker network rm name

```
➜  docker network rm RedPrueba
RedPrueba
```

```
➜  docker network ls
NETWORK ID     NAME          DRIVER    SCOPE
bc3c6e9170a1   RedIntranet   bridge    local
fbac97951f5c   bridge        bridge    local
195de2662bc3   host          host      local
b1eabba11a92   none          null      local
```

Gestión de redes bridge - Comandos

docker network prune

```
➜  docker network prune
WARNING! This will remove all custom networks not used by at least one container.
Are you sure you want to continue? [y/N] y
Deleted Networks:
RedIntranet
```

```
➜  docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
fbac97951f5c   bridge    bridge    local
195de2662bc3   host      host      local
b1eabba11a92   none      null      local
```

Ejercicio 1

Ejercicio 1

Cuando hicimos el ejercicio 2 del tema 04 Contenedores, en el que se montaba un Wordpress con MySQL y PhpMyAdmin utilizamos la red “bridge” por defecto y tuvimos que usar la opción --link para que el contenedor de PhpMyAdmin pudiera resolver el nombre del contenedor de la base de datos.

Dado que usar la red “bridge” no está recomendado en producción, y --link está depreciado, se pide hacer el ejercicio usando una red bridge normal para la comunicación entre los contenedores.

Ejercicio 1 - Solución

Creamos la red:

```
→  docker network create RedWordpress
fc5248d71fe23e35454e1042049904l1b70c08ad0834ca8579ca38593fe9cd1c
```

Ejecutamos el contenedor con MySQL con el nombre que espera PhpMyAdmin:

```
→  docker run -d --name db --network RedWordpress -e MYSQL_ROOT_PASSWORD=pass -e MYSQL_DATABASE=wordpress mysql:8
e4d216eb0bfa93148e7cb074f5b09593ee7b08332d44919b1467f95c2d5afa65
```

Ejecutamos el contenedor con Wordpress:

```
→  docker run -d --name wordpress --network RedWordpress -p 8080:80 wordpress
f367eab6e511355f667f46aee3da756b8f3cac33446d7ac5cea8837dc799d2fc
```

Ejercicio 1 - Solución

Ejercicio 1 - Solución

Ejecutamos el contenedor con PhpMyAdmin:

```
➜  docker run -d --name dbadmin --network RedWordpress -p 8081:80 phpmyadmin
a9b2a7b4ebcd16d89db27a1a8f69964254ff49ce27abd821184031dc18903684
```