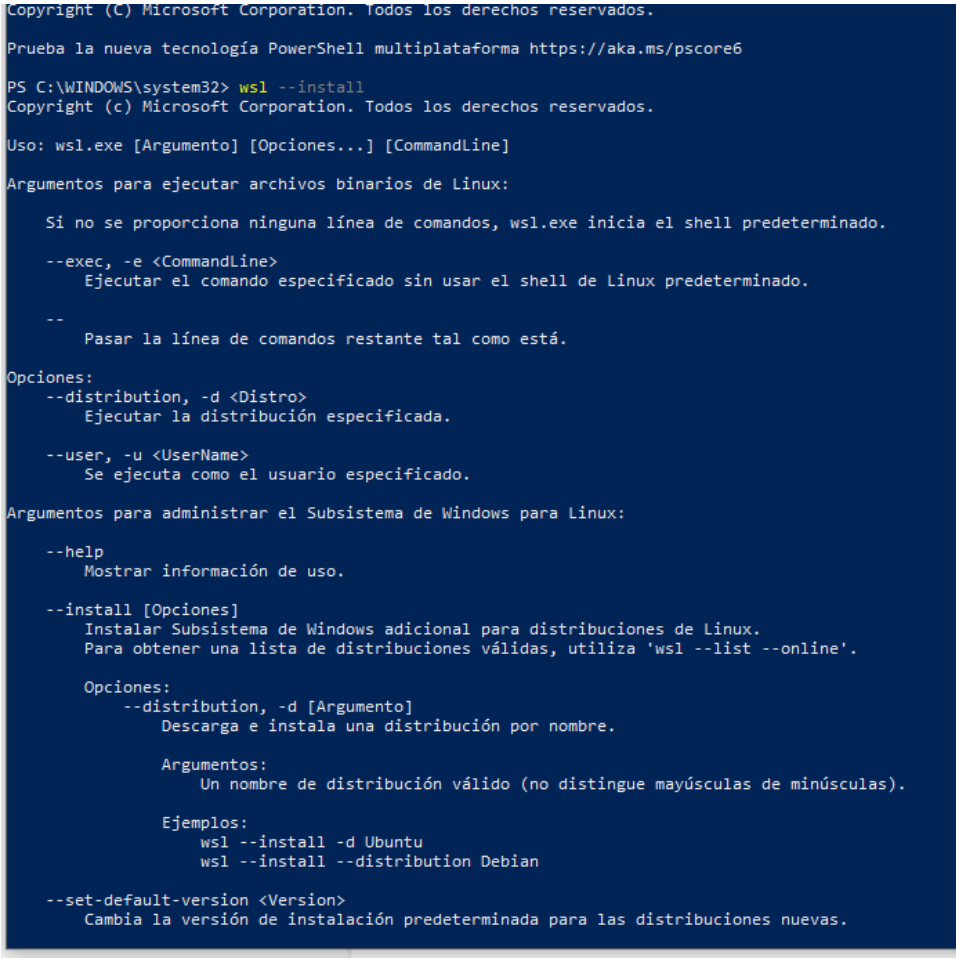
INTRODUCCIÓN A DOCKER

1. Instalación de Docker

1. Revisar la versión de Windows (Botón Windows + R y ejecutar winver).
2. Instalar Windows Terminal (si no lo tenemos ya).
3. Ejecutar Windows Terminal como administrador. Si da error, ejecutar Powershell como administrador.
4. wsl --install (para tener Linux dentro de Windows).



5. Descargar desde https://docs.docker.com/desktop/windows/install/
6. Descargar https://docs.microsoft.com/es-es/windows/wsl/install-manual#step-4---download-the-linux-kernel-update-package

7. Seguir los pasos de la página del punto 6 (con Powershell en admin).
8. Descargar Ubuntu 20 desde Microsoft Store.

9. Instalar Docker (dejas todas las opciones marcadas).

Curso de Docker de OpenWebinars

Características de Docker

Es open source

Docker es una interfaz utilizada para gestionar contenedores.

Un contenedor es un proceso aislado al resto de procesos del sistema, ya que tiene su propio sistema de ficheros, su propio servicio de usuarios, su propia interfaz de red… por lo que está completamente aislado del resto de los procesos. Por esto se dice que es una **máquina virtual ligera.**

- **Portabilidad**: se pueden ejecutar en cualquier dispositivo o máquina, y funcionan de la misma forma.
- **Inmutabilidad:** el paquete de procesos con independencias (contenedor) lo podemos testear en cualquier dispositivo, ya que se comporta siempre de la misma forma.
- **Ligero**, dado que todos los contenedores usan el mismo kernel. Esto nos permite estar ejecutando un programa en una versión en un contenedor y otra distinta en otro sin que interfieran entre sí.

Componentes de Docker

Docker tiene una estructura modular

- Cliente: Generalmente suele ser la línea de comandos de Docker, aunque puede ser cualquier programa que interactúe con el Docker Daemon
- Daemon: Gestiona y construye los contenedores. Donde está toda la lógica. Está instalado en un host (no tiene por qué ser el mismo del cliente). Pone una API que el cliente consume.
    - Imagen: Sistema de ficheros con las dependencias, variables de entorno...
    - Contenedor: Proceso que corre una de esas imágenes.

- Registro: Donde se almacenan los contenedores.

Si el cliente está en una máquina remota, se le manda una petición al Daemon para que construya esa imagen.

Fichero **docker-compose.yml**: Nos permite definir una serie de servicios y otras entidades como volúmenes o redes. Es un fichero declarativo donde definimos cómo se comporta un contenedor (dependiendo de la imagen que se vaya a utilizar y otros metadatos como variables de entorno)

Ciclo de desarrollo

Tenemos tres contenedores: nginx (imágenes), la de nuestra aplicación y la de la base de datos (sql)

Simplemente, puedes trabajar desde local, realizar push al registro y un compañero podría pullear esas imágenes sin importar la versión en la que esté trabajando en local.

Más productividad, más felicidad :D

Instalación de Docker

Se descarga desde https://docs.docker.com/desktop/windows/install/

Podemos ejecutar el comando ***docker version*** para comprobar la versión que tenemos instalada en nuestro ordenador.

Desde DockerHub podremos descargarnos distintas imágenes para correr nuestros contenedores

Docker para desarrolladores

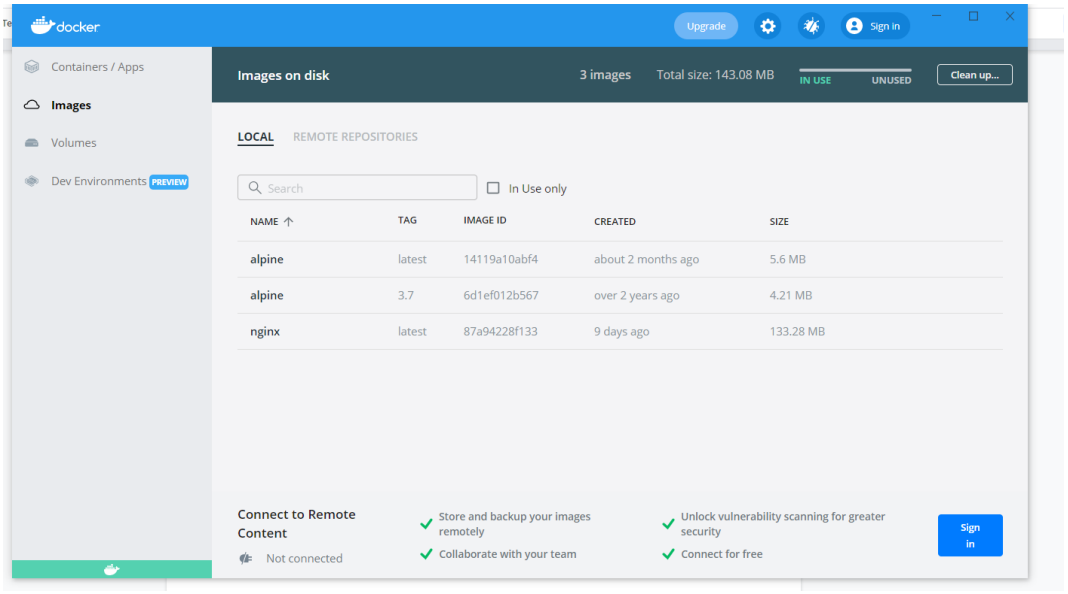
- *docker version*: da información sobre la versión de docker que estamos corriendo.
- *docker info*: da información acerca de la cantidad de contenedores e imágenes que está gestionando la máquina actual, así como los plugins actualmente instalados.
- *docker run (-d)(--name + nombre)(-p + puerto) + imagen*: crea un contenedor a partir de una imagen. Este comando permite multitud de parámetros, que son actualizados para cada versión del Docker Engine, por lo que para su documentación lo mejor es hacer referencia a la página oficial.

- *docker ps*: muestra los contenedores que están corriendo en la máquina. Con el flag -a muestra también los contenedores que están parados.
- *docker inspect + nombre_contenedor*: muestra información detallada de un contenedor en formato json. Se puede acceder a un campo particular con el comando docker inspect -f '{{.campo}}' + nombre_contenedor.
- *docker stop*: para la ejecución de un contenedor.
- *docker start*: reanuda la ejecución de un contenedor.
- *docker rm*: elimina un contenedor. Para borrar todos los contenedores de una máquina se puede ejecutar el comando docker rm -fv $(docker ps -aq).
  memoria utilizada, la CPU, el disco…
- *docker exec*: ejecuta un comando en un contenedor. Útil para depurar contenedores en ejecución con las opciones docker exec -it contenedor bash.

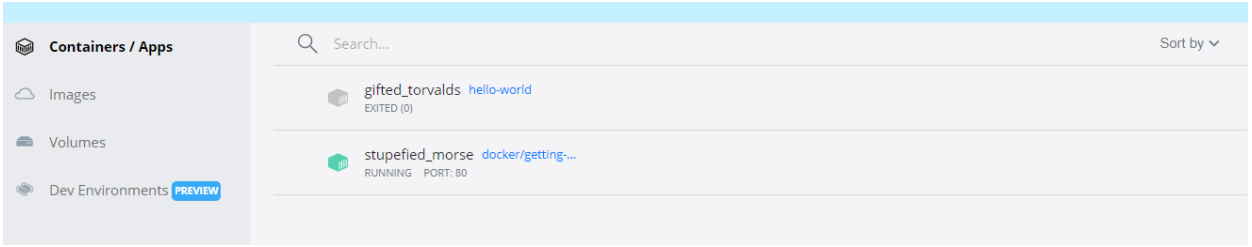
  Con **docker exec -it + nombre_contenedor sh** accedemos al shell del contenedor y ejecutar comandos dentro del mismo.
- *docker cp*: copia archivos entre el host y un contenedor.
- *docker logs*: muestra los logs de un contenedor.
- *docker stats*: muestra las estadísticas de ejecución de un contenedor.
- *docker system prune*: utilidad para eliminar recursos que no están siendo usados en este momento.

Practicando los comandos

```
PS C:\Users\dominguez.vamar21> docker -v
Docker version 20.10.8, build 3967b7d
PS C:\Users\dominguez.vamar21> docker pull alpine
Using default tag: latest
latest: Pulling from library/alpine
a0d0a0d46f8b: Pull complete
Digest: sha256:e1c082e3d3c45cccac829840a25941e679c25d438cc8412c2fa221cf1a824e6a
Status: Downloaded newer image for alpine:latest
docker.io/library/alpine:latest
PS C:\Users\dominguez.vamar21> docker pull alpine:3.7
3.7: Pulling from library/alpine
5d20c808ce19: Pull complete
Digest: sha256:8421d9a84432575381bfabd248f1eb56f3aa21d9d7cd2511583c68c9b7511d10
Status: Downloaded newer image for alpine:3.7
docker.io/library/alpine:3.7
PS C:\Users\dominguez.vamar21> docker pull nginx
Using default tag: latest
latest: Pulling from library/nginx
b380bbd43752: Pull complete
fca7e12d1754: Pull complete
745ab57616cb: Pull complete
a4723e260b6f: Pull complete
1c84ebdff681: Pull complete
858292fd2e56: Pull complete
Digest: sha256:644a70516a26004c97d0d85c7fe1d0c3a67ea8ab7ddf4aff193d9f301670cf36
Status: Downloaded newer image for nginx:latest
docker.io/library/nginx:latest
PS C:\Users\dominguez.vamar21>
```



```
PS C:\Users\dominguez.vamar21> docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
2db29710123e: Pull complete
Digest: sha256:37a0b92b08d4919615c3ee023f7ddb068d12b8387475d64c622ac30f45c29c51
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/

PS C:\Users\dominguez.vamar21> docker run -d -p 80:80 docker/getting-started
Unable to find image 'docker/getting-started:latest' locally
latest: Pulling from docker/getting-started
540db60ca938: Pull complete
0ae30075c5da: Pull complete
9da81141e74e: Pull complete
b2e41dd2ded0: Pull complete
7f40e809fb2d: Pull complete
758848c48411: Pull complete
23ded5c3e3fe: Pull complete
38a847d4d941: Pull complete
Digest: sha256:10555bb0c50e13fc4dd965ddb5f00e948ffa53c13ff15dcdc85b7ab65e1f240b
Status: Downloaded newer image for docker/getting-started:latest
c05d99f45723d9fb6567f5511bbcbd95496df6adced4d332c507a13860f77fb7
PS C:\Users\dominguez.vamar21>
```




```
PS C:\Users\dominguez.vamar21> docker stop stupefied_morse
stupefied_morse
PS C:\Users\dominguez.vamar21>
```

Borrar todas las imágenes

docker rmi $(docker images -aq)

Se puede inicializar un contenedor con un comando de linux a través del archivo Dockerfile de la imagen:

```
FROM scratch ①
ADD alpine-minirootfs-3.10.3-x86_64.tar.gz /
CMD ["/bin/sh"] ③
```

docker run --rm → Elimina el contenedor tras crearlo, lo que nos sirve para ejecutar contenedores que no vamos a seguir usando (por ejemplo: para practicar).

```
PS C:\Users\dominguez.vamar21> docker run --rm alpine cat /etc/os-release
NAME="Alpine Linux"
ID=alpine
VERSION_ID=3.14.2
PRETTY_NAME="Alpine Linux v3.14"
HOME_URL="https://alpinelinux.org/"
BUG_REPORT_URL="https://bugs.alpinelinux.org/"
PS C:\Users\dominguez.vamar21>
```

docker run -d nginx → Ejecuta la imagen en segundo plano.

A través de Docker Desktop, si pulsamos sobre un contenedor inicializado, podremos ver su pantalla de log.

Podemos hacer lo mismo desde el terminal con **docker logs nombre_contenedor** o **docker logs id_contenedor**

```
PS C:\Users\dominguez.vamar21> docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS         PORTS     NAMES
226649df8cc0   nginx     "/docker-entrypoint.…"   2 minutes ago   Up 2 minutes   80/tcp    bold_colden
PS C:\Users\dominguez.vamar21> docker logs bold_colden
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2021/10/27 07:21:31 [notice] 1#1: using the "epoll" event method
2021/10/27 07:21:31 [notice] 1#1: nginx/1.21.3
2021/10/27 07:21:31 [notice] 1#1: built by gcc 8.3.0 (Debian 8.3.0-6)
2021/10/27 07:21:31 [notice] 1#1: OS: Linux 5.10.16.3-microsoft-standard-WSL2
2021/10/27 07:21:31 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2021/10/27 07:21:31 [notice] 1#1: start worker processes
2021/10/27 07:21:31 [notice] 1#1: start worker process 31
2021/10/27 07:21:31 [notice] 1#1: start worker process 32
2021/10/27 07:21:31 [notice] 1#1: start worker process 33
2021/10/27 07:21:31 [notice] 1#1: start worker process 34
2021/10/27 07:21:31 [notice] 1#1: start worker process 35
2021/10/27 07:21:31 [notice] 1#1: start worker process 36
PS C:\Users\dominguez.vamar21>
```

docker logs -f (o --follow) nombre_contenedor → Muestra log sin que se detenga.

Para que un contenedor no se detenga al ejecutarse debemos indicarle que queremos iniciarlo en modo interactivo. Para ello lo haremos con el flag **-it**.

```
PS C:\Users\dominguez.vamar21> docker run -it --name alpinec alpine
/ #
/ #
```

- El **parámetro -i** nos permite interaccionar con el contenedor a través de la entrada estándar STDIN.
- El **parámetro -t** nos asigna un terminal dentro del contenedor.
- Los dos parámetros -it nos permiten usar un contenedor como si fuese una máquina virtual tradicional.

exec → Nos permite ejecutar un comando en un contenedor que está en ejecución indicando su nombre o su ID. **exec** ejecuta el comando en un proceso nuevo, asignándonos un nuevo terminal.

Esto significa que si salimos del contenedor con **exit**, el contenedor no detendrá su ejecución.

Para ello primero tenemos que iniciar el contenedor con **docker start nombre_contenedor**

```
PS C:\Users\dominguez.vamar21> docker start alpinec
alpinec
PS C:\Users\dominguez.vamar21> docker exec -it alpinec /bin/sh
/ #
```

Para instalar paquetes en un contenedor utilizaremos el comando **apk**

```
PS C:\Users\dominguez.vamar21> docker exec -it alpinec /bin/sh
/ # apk update
fetch https://dl-cdn.alpinelinux.org/alpine/v3.14/main/x86_64/APKINDEX.tar.gz
fetch https://dl-cdn.alpinelinux.org/alpine/v3.14/community/x86_64/APKINDEX.tar.gz
v3.14.2-116-gea9b052f30 [https://dl-cdn.alpinelinux.org/alpine/v3.14/main]
v3.14.2-112-g40db7c160b [https://dl-cdn.alpinelinux.org/alpine/v3.14/community]
OK: 14943 distinct packages available
/ # apk add nano
(1/4) Installing libmagic (5.40-r1)
(2/4) Installing ncurses-terminfo-base (6.2_p20210612-r0)
(3/4) Installing ncurses-libs (6.2_p20210612-r0)
(4/4) Installing nano (5.7-r2)
Executing busybox-1.33.1-r3.trigger
OK: 13 MiB in 18 packages
/ # nano
/ #
```

```
PS C:\Users\dominguez.vamar21>  docker run -it --rm --name prueba alpine
/ #
```

Crea un contenedor con Ubuntu en modo desconectado, asigna un nombre e incluye el parámetro --rm.

```
PS C:\Users\dominguez.vamar21>  docker run -dit --rm --name otro alpine
e5909ea45182f9c38d705413666a25324cf078d1e1bc00c7b995d68c59adf5ff
```

- Si salimos pulsando **CTRL + C** el contenedor se detendrá y finalizará su ejecución.
- Si salimos pulsando **CTRL + P + Q** el contenedor seguirá ejecutándose en background.

La diferencia entre **exec** y **attach** es que **exec** crea un nuevo terminal o ventana, mientras que **attach** utilizará el terminal existente (si es que lo hay).

$ docker rm $(docker ps -aq) → El comando **rm** puede recibir la salida de otro comando. De esta forma primero evalúa los contenedores que están detenidos y luego los elimina con **rm**. Esto realmente es de Linux y no de Docker ⇒ Parecido a los pipes. Esto quiere decir que podemos utilizar esto con otros comandos de Linux.

```
PS C:\Users\dominguez.vamar21> docker rm $(docker ps -aq)
bafb95303221
c05d99f45723
52ada1e094bf
Error response from daemon: You cannot remove a running container 828804eb004fb3e5815b265a5ae4a05571ea0ae7f07fd22cd7
35e20fbed3. Stop the container before attempting removal or force remove
Error response from daemon: You cannot remove a running container e5909ea45182f9c38d705413666a25324cf078d1e1bc00c7b9
8c59adf5ff. Stop the container before attempting removal or force remove
Error response from daemon: You cannot remove a running container a47856c7ec30fc0e89e7d7c58cc23c669a493e7ab73a830e35
Error response from daemon: You cannot remove a running container 35b41e27aa88b808d08c90b0bec2930fe9eb55631d11044488
8c7e987af2. Stop the container before attempting removal or force remove
Error response from daemon: You cannot remove a running container 226649df8cc068c352ac3057a5abeade9a28767d5d6da011e7
a78b8d1fec. Stop the container before attempting removal or force remove
```

Con este comando podemos borrar hasta los contenedores que se están ejecutando:

```
PS C:\Users\dominguez.vamar21> docker rm -f $(docker ps -aq)
828804eb004f
e5909ea45182
a47856c7ec30
35b41e27aa88
226649df8cc0
PS C:\Users\dominguez.vamar21>
```

docker inspect nombre_contenedor → Aporta información sobre el contenedor.

PUERTOS

Consiste en reservar un puerto del servidor de Docker con el objetivo de redirigir las peticiones a un puerto específico de un contenedor.

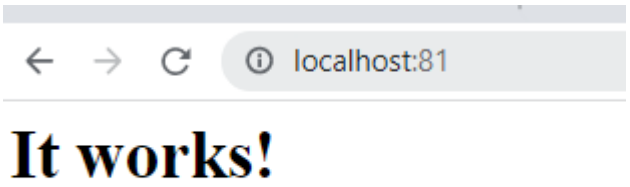
Para cambiar el puerto de un contenedor añadimos la anotación **-p**

```
PS C:\Users\fatuarte.feern21> docker run -d --rm --name alpinec -p 81:80 alpine
```

La sintaxis para indicar los puertos será **puerto_local:puerto_contenedor**

Comprobación de que funciona:

```
PS C:\Users\dominguez.vamar21> docker run -d --rm --name webserver2 -p 81:80 httpd
ee0f16f5dfe3248d62d9878ddd4b014c7a5e11271dd586a05a4fa3bc3dbaf24e
PS C:\Users\dominguez.vamar21> docker ps
CONTAINER ID   IMAGE     COMMAND              CREATED          STATUS          PORTS                  NAMES
ee0f16f5dfe3   httpd     "httpd-foreground"   7 seconds ago    Up 5 seconds    0.0.0.0:81->80/tcp     webserver2
be0344724ee1   httpd     "httpd-foreground"   12 minutes ago   Up 12 minutes   80/tcp                 httpdc
PS C:\Users\dominguez.vamar21>
```



Se mostrará este error si intentamos conectar el mismo puerto a otro contenedor diferente.

```
C:\Users\chamorro.gaser21> docker run -d --rm --name webserver2 -p 81:80 httpd
6c628a301ca475af979f5b7456b5667c1cd04520285d93b33bbaf79aecefdb1a
docker: Error response from daemon: driver failed programming external connectivity on endpoint webserver2 (f2158c8eca35
acb6f103da220bc24521b565ef78eee724934c6032d96f7ac5e1): Bind for 0.0.0.0:81 failed: port is already allocated.
```

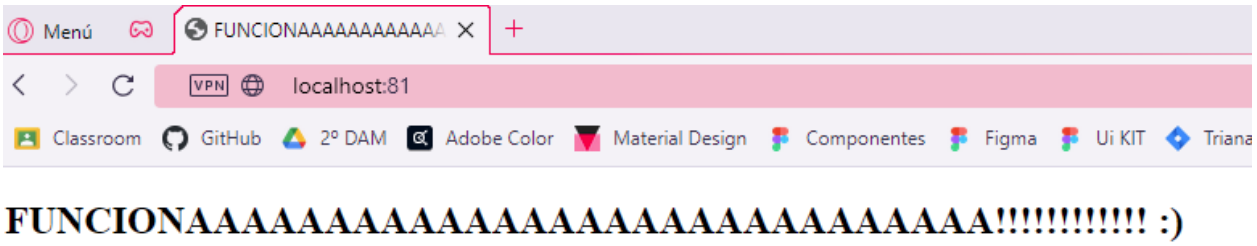
//Con este comando, no tenemos que especificar el puerto por el que queremos acceder al contenedor. Docker se lo asigna solo.

```
PS C:\Users\dominguez.vamar21> docker run -d --rm --name webserver3 -P httpd
ef5a1d6b30becc8edddb25e7f40c03da7f653ac14987f31e721d2a58bf8001d1
PS C:\Users\dominguez.vamar21>
```

```
CONTAINER ID   IMAGE     COMMAND              CREATED              STATUS              PORTS                     NAMES
ef5a1d6b30be   httpd     "httpd-foreground"   About a minute ago   Up About a minute   0.0.0.0:49153->80/tcp     webserver
3
ee0f16f5dfe3   httpd     "httpd-foreground"   7 minutes ago        Up 7 minutes        0.0.0.0:81->80/tcp        webserver
2
be0344724ee1   httpd     "httpd-foreground"   20 minutes ago       Up 20 minutes       80/tcp                    httpdc
PS C:\Users\dominguez.vamar21>
```

Para hacerlo con **Nginx**:

Editamos el fichero /usr/share/nginx/html/index.html




```
PS C:\Users\fatuarte.feern21> docker cp webService:/usr/share/nginx/html/index.html .\Desktop\
```

El comando sería:

contenedor → ordenador ⇒ docker cp contenedor:ruta_contenedor ruta_mi_equipo

ordenador → contenedor ⇒ docker cp ruta_local contenedor:ruta_contenedor

```
PS C:\Users\efatu\Desktop\web> docker cp C:\Users\efatu\Desktop\web\index.html nginx_prueba:/usr/share/nginx/html/index.html
PS C:\Users\efatu\Desktop\web>
```

Volúmenes

-v ruta_local:ruta_contenedor

Almacenamiento en Docker

- **Bind amount**: pueden estar almacenados en cualquier directorio del sistema de archivos de la máquina host. Pueden ser consultados o modificados tanto por otros procesos de la máquina host como por otros contenedores Docker.
- **Volúmenes**: se almacenan en la máquina host dentro del área del sistema de archivos que gestiona Docker. Solo pueden ser modificados por Docker.

```
PS C:\Users\dominguez.vamar21\desktop> docker run --name webserver-st -d -p 80:80 -v C:\Users\domi
nguez.vamar21\Desktop\prueba_docker\:/usr/share/nginx/html --rm nginx
3b0f117631a189f1afdc0f8603298542194477262e9d526922238d4e1287fd6a
```

No me sale pero apunto aquí el comando por si acaso:

```
C:\Users\dominguez.vamar21\Desktop> docker run -d --rm --name apache_php -p 8080:80 -v C:\Users\do
minguez.vamar21\Desktop\prueba2\:var/www/html php:7.2-apache
Unable to find image 'php:7.2-apache' locally
7.2-apache: Pulling from library/php
6ec7b7d162b2: Pull complete
db606474d60c: Pull complete
afb30f0cd8e0: Pull complete
3bb2e8051594: Pull complete
4c761b44e2cc: Pull complete
c2199db96575: Pull complete
1b9a9381eea8: Pull complete
fd07bbc59d34: Pull complete
72b73ab27698: Pull complete
983308f4f0d6: Pull complete
6c13f026e6da: Pull complete
e5e6cd163689: Pull complete
5c5516e56582: Pull complete
154729f6ba86: Pull complete
Digest: sha256:4dc0f0115acf8c2f0df69295ae822e49f5ad5fe849725847f15aa0e5802b55f8
Status: Downloaded newer image for php:7.2-apache
docker: Error response from daemon: invalid volume specification: '/run/desktop/mnt/host/c/Users/d
ominguez.vamar21/Desktop/prueba2:var/www/html': invalid mount config for type "bind": invalid moun
t path: 'var/www/html' mount path must be absolute.
See 'docker run --help'.
```

VARIABLES DE ENTORNO Y CONTENEDORES

Cuando creamos un contenedor docker con una imagen de MySQL, no pregunta qué contraseña queremos establecer, por lo que debemos establecerla manualmente.

Para eso usamos añadimos una variable de entorno a la hora de la creación del contenedor, lo haremos usando el flag **-e**, a la cual añadiremos la variable **MYSQL_ROOT_PASSWORD=pwd**

Usaremos las variables de entorno de esta forma para establecer contraseñas y otros datos confidenciales necesarios para la aplicación, de forma que no queden expuestos en el código fuente.

```
PS C:\Users\dominguez.vamar21> docker run -d --rm --name mysqlc -e MYSQL_ROOT_PASSWORD=root
 -e MYSQL_DATABASE=dam -p 3306:3306 mysql:5.7.28
3249531bd2a55421a17bab2c7f0b1c29805a3fe199055e7c3db3a63f697a520e
```

Las variables de entorno deben ir separadas, es decir, por cada una habrá que poner -e delante.

```
PS C:\Users\dominguez.vamar21> docker exec -it mysqlc /bin/bash
root@3249531bd2a5:/# mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 5.7.28 MySQL Community Server (GPL)

Copyright (c) 2000, 2019, Oracle and/or its affiliates. All rights reserved.

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.

mysql>
```

```
mysql> use dam;
Database changed
```

```
mysql> create table dummy(message varchar(100));
Query OK, 0 rows affected (0.02 sec)

mysql> select * from dummy;
Empty set (0.00 sec)
```

```
mysql> INSERT INTO dam VALUES('Hola Mundo');
ERROR 1146 (42S02): Table 'dam.dam' doesn't exist
mysql> INSERT INTO dummy VALUES('Hola Mundo');
Query OK, 1 row affected (0.01 sec)

mysql> select * from dummy;
+------------+
| message    |
+------------+
| Hola Mundo |
+------------+
1 row in set (0.00 sec)

mysql>
```

```
PS C:\Users\dominguez.vamar21> docker run -d --rm --name mysqlc -e MYSQL_ROOT_PASSWORD=root
 -e MYSQL_DATABASE=dam -p 3306:3306 -v mysql_data/var/lib/mysql mysql:5.7.28
03f7f7f7bb258377e2e07894d6546feec5ae782ee09b075592b8ee5228f3c613
```

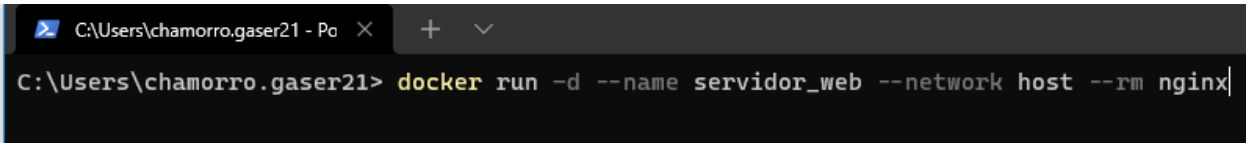
Redes

Cuando instalamos Docker Engine se crean tres redes:

• **bridge**: Es la red que usarán por defecto todos los contenedores que se ejecutan en el mismo host. También se les conoce como las default bridge network. En Linux durante la instalación se crea una nueva interfaz de red virtual llamada docker0. Cuando ejecutamos un contenedor, esta es la red que utilizará por defecto a no ser que indiquemos lo contrario.

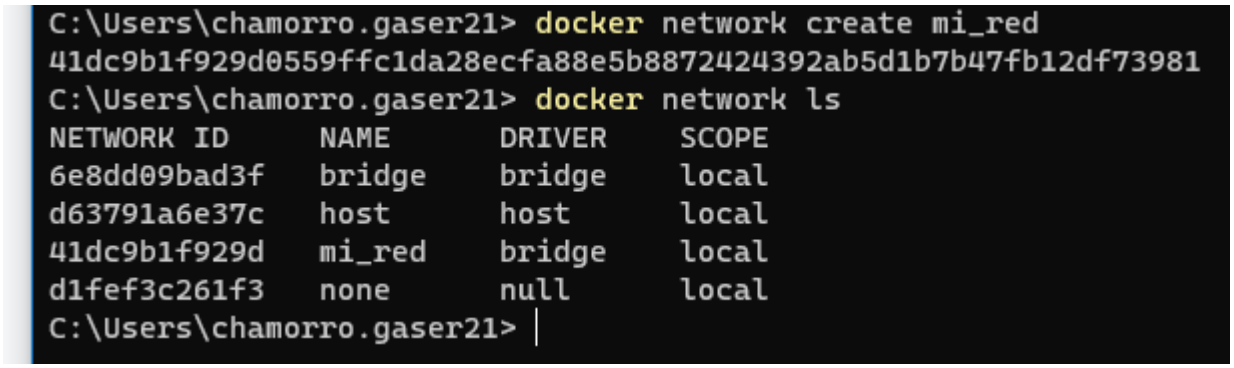
• **none**: En esta red el contenedor no tendrá asociada ninguna interfaz de red, sólo tendrá la de loopback (lo).

• **host**: En esta red el contenedor tendrá la misma configuración que el servidor Docker Engine donde se esté ejecutando.



Mediante este comando, docker le asignará al contenedor una IP de la misma red que nuestro ordenador.

docker network create nombre_red → Crear una nueva red. Se creará como una red de tipo *bridge* por defecto.



```
PS C:\Users\dominguez.vamar21> docker network create mired
504c0df71a53e4aef97a46d475203f1c7c812efefd998efacf7d937d5de6c107
PS C:\Users\dominguez.vamar21> docker network inspect mired
[
    {
        "Name": "mired",
        "Id": "504c0df71a53e4aef97a46d475203f1c7c812efefd998efacf7d937d5de6c107",
        "Created": "2021-11-03T08:58:09.44467482",
        "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": {}
    }
]
```

```
PS C:\Users\dominguez.vamar21> docker container run -dit --name container-a --network mired alpine ash
c94ae682d68292d8429b5f0ec25a8ed98c0c471d7127371a195fe8de865594fd
PS C:\Users\dominguez.vamar21> docker container run -dit --name container-b --network mired alpine ash
fbdcb3cfa6a237da156a3ef31b2e76aa05b19b5c7ba9b148c9f65bdb299563cc
PS C:\Users\dominguez.vamar21> docker attach container-a
/ # ping -c 3 container-b
PING container-b (172.18.0.3): 56 data bytes
64 bytes from 172.18.0.3: seq=0 ttl=64 time=0.090 ms
64 bytes from 172.18.0.3: seq=1 ttl=64 time=0.058 ms
64 bytes from 172.18.0.3: seq=2 ttl=64 time=0.140 ms

--- container-b ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 0.058/0.096/0.140 ms
/ #
```

Wordpress y SQL

```
PS C:\Users\dominguez.vamar21> docker run -d --rm --name mysqlc --network wordpress-net -p
 8080:8080 -e MYSQL_ROOT_PASSWORD=root -e MYSQL_DATABASE=wp_database -e MYSQL_USER=wp_user
 -e MYSQL_PASSWORD=wp_password -e WORDPRESS_DB_HOST=mysql -v wordpress_mysql_data:/var/lib
/mysql mysql:5.7.28
24d655fc82e2c9cfe11c87eda1e7f04b760f7dd474e4b43867622a578ea64a1b
PS C:\Users\dominguez.vamar21>
```