Variables en Docker
En esta parte de la práctica vamos a aprender como enviar variables a nuestro contenedor docker.
Para enviar variables al contenedor usamos la opción “-e” a la hora de crear nuestro contenedor. En la documentación en Docker Hub de las imágenes nos pueden indicar que variables hay que presentar a la hora de crear un contenedor, es por ello que hay revisar la documentación de la imagen descargada.
A continuación, lo aplicaremos en una imagen de Postgres, base de datos.

1. Descargamos una imagen, por ejemplo “postgres” y comprobamos que se ha descargado:

$docker pull postgres
$docker images | grep postgres

```
[root@localhost ~]# docker pull postgres
Using default tag: latest
latest: Pulling from library/postgres
b0a0cf830b12: Pull complete
dda3d8fbd5ed: Pull complete
283a477db7bb: Pull complete
91d2729fa4d5: Pull complete
9739ced65621: Pull complete
ae3bb1b347a4: Pull complete
f8406d9c00ea: Pull complete
c199bff16b05: Pull complete
e0d55fdb4d15: Pull complete
c1cb13b19080: Pull complete
873532e5f8c7: Pull complete
050d9f8c3b1c: Pull complete
710e142705f8: Pull complete
cb628c265f09: Pull complete
Digest: sha256:4aea012537edfad80f98d870a36e6b90b4c09b27be7f4b4759d72db863baeebb
Status: Downloaded newer image for postgres:latest
docker.io/library/postgres:latest
```

2. Si intentamos crear un contenedor de la imagen postgres sin pasar ninguna variable nos ocurre lo siguiente:

$docker run -it --name postgres1 postgres

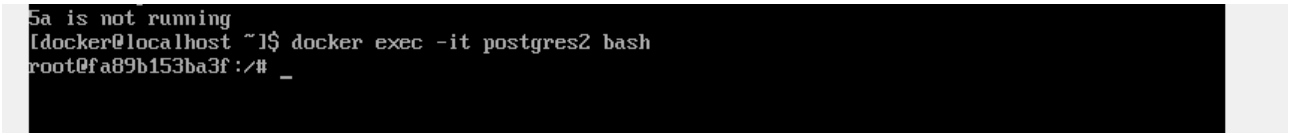
```
[root@localhost ~]# docker pull postgres
Using default tag: latest
latest: Pulling from library/postgres
Digest: sha256:4aea012537edfad80f98d870a36e6b90b4c09b27be7f4b4759d72db863baeebb
Status: Image is up to date for postgres:latest
docker.io/library/postgres:latest
[root@localhost ~]# docker images | grep postgres
postgres      latest    8e4fc9e18489   7 weeks ago     431MB
[root@localhost ~]#
```

3. Creamos nuevamente el contenedor indicando un usuario, contraseña y una base de datos:

$docker run -it --name postgres2 -e POSTGRES_PASSWORD=luisabenitez -e POSTGRES_USER=usuario1 -e POSTGRES_DB=bd2 postgres


4. Entramos en la base de datos y comprobamos que podemos ver que se ha creado correctamente el usuario y la base de datos:

$docker exec -it postgres1 bash




$psql -U usuario1 bd1 -> ( Comando para acceder a la base de datos)

$\l -> (Comando para listar las bases de datos de postgres)

$\q -> (Comando para salir de la base de datos postgres)

$exit

5. Con docker inspect podemos ver las variables enviadas al contenedor. (Poner captura de las variables enviadas al contenedor con el comando “docker inspect”)

Puertos y Redes

Ahora veremos como mapear puertos entre los contenedores y también a crear redes.
Para mapear puertos se usará la opción “-p puerto_Host:puerto_Contenedor” cuando se crea un contenedor.
Una forma de saber que puerto usa una imagen o contenedor, por defecto, se usa “docker inspect Nombre_Imagen_o_Nombre_Contenedor”.
Para saber que puertos está usando un contenedor se puede usar, a parte del anterior, el comando “docker port NombreContenedor”.

1. En el siguiente ejemplo veremos como con la imagen de base de datos MYSQL mapeamos para que escuche por el puerto 4000. Mysql también necesita que le pasemos una variable de entorno para que inicie correctamente:

$docker run -d -p 4000:3306 --name mysql1 -e
MYSQL_ROOT_PASSWORD=secret1 mysql

Así haces una conexión a mysql, le pones un nombre, contraseña y que en vez de escuchar el puerto 3306, escuche el 4000

$docker ps

```
[root@localhost ~]# docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS         PORTS
                                NAMES
e56f9d538712   mysql     "docker-entrypoint.s■"   4 minutes ago   Up 4 minutes   33060/tcp, 0.0.0.0:
4000->3306/tcp, :::4000->3306/tcp   mysql1
[root@localhost ~]# _
```

$docker port mysql1

```
[root@localhost ~]# docker port mysql1
3306/tcp -> 0.0.0.0:4000
3306/tcp -> [::]:4000
[root@localhost ~]# _
```

2. Hacemos lo mismo con una imagen del servidor apache (httpd):

$docker run -d -p 8080:80 --name apache1 httpd

```
[root@localhost ~]# docker run -d -p 8080:80 --name apache1 httpd
Unable to find image 'httpd:latest' locally
latest: Pulling from library/httpd
b0a0cf830b12: Already exists
851c52adaa9b: Pull complete
4f4fb700ef54: Pull complete
39d9f60535a6: Pull complete
943a2b3cf551: Extracting  262.1kB/26.03MB
ea83e81966d6: Download complete
```

$docker ps

```
[root@localhost ~]# docker ps
CONTAINER ID   IMAGE     COMMAND              CREATED         STATUS         PORTS
                                  NAMES
f36c950d64cd   httpd     "httpd-foreground"   55 seconds ago   Up 54 seconds   0.0.0.0:8080->80/
tcp, :::8080->80/tcp                apache1
```

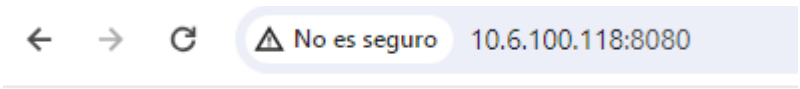
$docker port apache1

```
[root@localhost ~]# docker port apache1
80/tcp -> 0.0.0.0:8080
80/tcp -> [::]:8080
```

3. Abrimos un navegador y accedemos al puerto 8080 y podemos ver el resultado:

Usamos ifconfig | less para saber mi ip

Ponemos en el navegador del pc 10.6.100.118:8080, que es mi ip y el nombre del puerto




# It works!

4. Pasemos ahora a comprobar las redes que tenemos. Para comprobar las redes que tenemos en docker usamos el siguiente comando:

$docker network ls

```
[root@localhost ~]# docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
8ac610627671   bridge    bridge    local
26b6129cd8d6   host      host      local
b23006dec668   none      null      local
[root@localhost ~]#
```

5. Por defecto los contenedores se añaden a la red BRIDGE. Los dos contenedores anteriores iniciados los podremos verlos en dicha red si la inspeccionamos, incluso podemos ver las IPs que se les ha asignado:

$docker network inspect bridge


También podemos ver a la red que pertenece cada contenedor con el comando “docker inspect”.

docker inspect apache1

# Crear redes y asociarlas a contenedores

Ahora pasaremos a crear nuestras propias redes y asociarlas a contenedores. Comando de ayuda:

$docker network --help

1. Creamos nuestra propia red y la listamos para ver que se ha creado correctamente:

$docker network create Nombre_Red

```
[root@localhost ~]# docker network create Nombre_Red
31bd124e455a535e3d770d13b6bd19b0f22f67eae45b11ccaa4389cd07879cfa
```

$docker network ls

```
[root@localhost ~]# docker network ls
NETWORK ID     NAME         DRIVER    SCOPE
31bd124e455a   Nombre_Red   bridge    local
8ac610627671   bridge       bridge    local
26b6129cd8d6   host         host      local
b23006dec668   none         null      local
```

2. Inspeccionamos la red creada para ver, por ejemplo, el rango de direcciones creada:

$docker network inspect Nombre_Red

```
        "Created": "2024-04-13T03:59:34.937964891+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": {}
    }
]
[root@localhost ~]# docker network inspect Nombre_Red_
```

3. A nivel físico se ha tenido que crear una red cuyo nombre coincide con el ID de la red creada.

$nmcli

```
br-31bd124e455a: conectado to br-31bd124e455a
        "br-31bd124e455a"
        bridge, 02:42:99:7D:5A:DB, sw, mtu 1500
        inet4 172.18.0.1/16
        route4 172.18.0.0/16

docker0: sin gestión
        "docker0"
        bridge, 02:42:C8:92:BD:45, sw, mtu 1500

veth055d007: sin gestión
        "veth055d007"
        ethernet (veth), 7E:DC:3D:CC:30:D4, sw, mtu 1500

vetha1683e8: sin gestión
        "vetha1683e8"
        ethernet (veth), FE:D4:E3:DD:34:5A, sw, mtu 1500

lo: sin gestión
        "lo"
        loopback (unknown), 00:00:00:00:00:00, sw, mtu 65536

Use «nmcli device show» para obtener información completa sobre dispositivos conocidos y
«nmcli connection show» para obtener un resumen de los perfiles de las conexiones activas.

Consulte las páginas del manual nmcli(1) y nmcli-examples(7) para detalles de uso completos.
lines 2-37/37 (END)
```

4. Ahora asignaremos la red creada a nuestros contenedores. Para asignar un contenedor a una red especifica usaremos la opción “--network Nombre_Red” en el momento de crear el contenedor, haremos las pruebas con apache (httpd):

$docker run -d -p 8080:80 --network red1 --name apache1 httpd

NO FUNCIONA, ME DICE QUE EL CONTENEDOR apache1 esta actualmente en uso y que TENGO QUE BORRAR O CAMBIARLE EL NOMBRE

Lo hice con:
$docker run -d -p 8080:80 --network red1 --name apache3 httpd

5. Comprobamos que el contenedor se ha unido a la red creada inspeccionando la nueva red o inspeccionando el contenedor. En el ejemplo se hace sobre la red creada:

$docker network inspect red1

6. Pasamos ahora a entrar en nuestro contenedor y a instalar ping para poder hacer pruebas con la red:

$docker exec -it apache1 bash
$apt-get update
$apt-get install -y iputils-ping

```
docker@localhost ~]$ docker exec -it apache3 bash
oot@d5101e980efa:/usr/local/apache2# apt-get update
et:1 http://deb.debian.org/debian bookworm InRelease [151 kB]
et:2 http://deb.debian.org/debian bookworm-updates InRelease [55.4 kB]
et:3 http://deb.debian.org/debian-security bookworm-security InRelease [48.0 kB]
et:4 http://deb.debian.org/debian bookworm/main amd64 Packages [8786 kB]
eading package lists... Done
: Release file for http://deb.debian.org/debian/dists/bookworm-updates/InRelease is not valid yet (
nvalid for another 5d 17h 28min 16s). Updates for this repository will not be applied.
: Release file for http://deb.debian.org/debian-security/dists/bookworm-security/InRelease is not v
lid yet (invalid for another 5d 20h 53min 53s). Updates for this repository will not be applied.
oot@d5101e980efa:/usr/local/apache2# apt-get install -y iputils-ping_
```

7. Creamos ahora otro contenedor de apache (paso 4 teniendo en cuenta los parámetros) añadiéndolo a nuestra red creada anteriormente e instalamos, inspeccionamos que se ha unido correctamente a la red (paso 5) e instalamos ping (paso 6).

PRIMERO SALIMOS DEL BASH

```
root@d5101e980efa:/usr/local/apache2# exit
exit
[docker@localhost ~]$
```

Ahora tenemos 2 servidores webescuchando en dos puertos diferentes

```
exit
[docker@localhost ~]$ docker run -d -p 8081:80 --network Nombre_Red --name alienDAM httpd
5b3705e7e3e83bda55934299a51a2d9cd9cb35adab8fed599204f0516e434e8a
[docker@localhost ~]$
```

```
[docker@localhost ~]$ docker exec -it alienDAM bash
root@5b3705e7e3e8:/usr/local/apache2# apt-get update
Get:1 http://deb.debian.org/debian bookworm InRelease [151 kB]
Get:2 http://deb.debian.org/debian bookworm-updates InRelease [55.4 kB]
Get:3 http://deb.debian.org/debian-security bookworm-security InRelease [48.0 kB]
Get:4 http://deb.debian.org/debian bookworm/main amd64 Packages [8786 kB]
Reading package lists... Done
E: Release file for http://deb.debian.org/debian/dists/bookworm-updates/InRelease is not valid yet (
invalid for another 5d 17h 19min 13s). Updates for this repository will not be applied.
E: Release file for http://deb.debian.org/debian-security/dists/bookworm-security/InRelease is not v
alid yet (invalid for another 5d 20h 44min 50s). Updates for this repository will not be applied.
root@5b3705e7e3e8:/usr/local/apache2# apt-get -y iputils-ping_
```

8. Como los dos contenedores creados anteriormente están en la misma red, se pueden hacer ping entre ellos solo con el nombre de cada uno de ellos. Entramos en cada contenedor y lanzamos un ping al otro contenedor:

9. Procedemos ahora a crear otra red con las siguientes especificaciones: a. Que la subred sea 172.42.0.0/16 b. Que las IPs de los contenedores asignados sean a partir de la IP 172.42.10.0/24.

$docker network create red2 --subnet= 172.42.0.0/16 --ip-range=172.42.10.0/24

10. Hacer una inspección de la red nueva para ver que ha asignado correctamente los valores de red mencionados anteriores.

11. Creamos un nuevo contenedor apache (httpd) y lo agregamos a la nueva red creada en el paso 9.

12. Comprobamos ahora que si entramos en este último apache de la nueva red creada no nos funciona ping a cualquiera de los 2 nombres de los primeros contenedores creados porque no están en la misma red:

13. Podemos asociar un contenedor a otra red con el siguiente comando:

$docker network connect NombreRed NombreContenedor