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
```

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

```
$docker run -it --name postgres1 postgres
```

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

```
$docker run -it --name postgres1 -e POSTGRES_PASSWORD=luisabenitez -e
POSTGRES_USER=usuario1 -e POSTGRES_DB=bd1 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
$docker ps
$docker port mysql1
```

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

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

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

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

```
$docker network ls
```

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".

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
$docker network ls
```

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

```
$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
```

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
```

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
```

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).

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
```

14. Ahora nuestro último apache creado podrá hacer ping por el nombre de los otros contenedores sin problema.

15. Para desconectar un contenedor de una red se usa el siguiente comando. Realizamos el ejemplo quitando el último contenedor creado de la red asignada anteriormente.

```
$docker network disconnect NombreRed NombreContenedor
```

16. Para eliminar una red utilizamos el siguiente comando:

```
$docker network rm NombreRed
```

Copiar ficheros entre Host y contenedores

En ocasiones tendremos que pasar información de nuestro host a un contenedor o viceversa. Para ello se usa el siguiente comando:

```
$docker cp ORIGEN DESTINO
```

Si queremos copiar del Host a un contenedor haremos lo siguiente:

```
$docker cp Ruta_Origen_fichero_Host Nombre_Contenedor:Ruta_destino_fichero
```

Si queremos copiar del contenedor al Host haremos lo siguiente:

```
$docker cp Nombre_Contenedor:Ruta_Origen_fichero Ruta_destino_fichero_Host
```

La clave está en que para indicar el contenedor hay que poner el **nombre_del_contenedor:**ruta_del_fichero, y cuando solo es en el host es suficiente con poner la ruta del fichero.

1. Vamos a crear un contenedor apache (httpd) y vamos a pasar un fichero del equipo Host a dicho contenedor y viceversa. Creamos el contenedor, con la opción "--rm" hacemos que el contenedor se elimine en el momento en el que se cierre:

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

2. En otro terminal accedemos al contenedor y nos ubicamos en una carpeta cualquiera para recibir un fichero del equipo Host, por ejemplo, en "/tpm" que no contiene ficheros:

```
$docker exec -it apache1 bash
```

3. Ahora desde un terminal del propio Host ejecutamos el comando "docker cp" para copiar un fichero al contenedor:

```
$docker cp pruebas/pruebaCopiadoDesdeHost.txt apache1:/tmp/
```

Listamos la carpeta /tmp del contenedor y debería estar el fichero que hemos copiado desde el Host:

4. Hacer la copia de un fichero del contenedor a una carpeta del equipo Host.