Introducción a Docker

UD 03. Principales acciones con Docker

Autor: Sergi García Barea

Actualizado Marzo 2021

Licencia

Nomenclatura

A lo largo de este tema se utilizarán distintos símbolos para distinguir elementos importantes dentro del contenido. Estos símbolos son:

📖 **Importante**

❕ **Atención**

💬 **Interesante**

UD03. PRINCIPALES ACCIONES CON DOCKER

1. INTRODUCCIÓN

En esta unidad explicaremos algunas de las principales acciones básicas que podemos realizar con Docker. Al finalizar la unidad ya estaremos listos para usar Docker con cierta soltura.

2. ¿GESTIONAREMOS DOCKER MEDIANTE INTERFAZ GRÁFICA?

Existen distintas herramientas para gestionar Docker desde una interfaz gráfica, haciendo la tarea más visual e intuitiva. Aunque estas herramientas pueden ser muy útiles, en el momento de aprender a trabajar con Docker, pueden hacer que se nos escape la comprensión de determinados mecanismos del funcionamiento de Docker. Por ese motivo en el curso no gestionaremos Docker mediante interfaz gráfica y realizaremos todas las operaciones mediante la línea de comandos.

3. IMÁGENES Y CONTENEDORES

3.1 ¿Qué es una imagen y un contenedor?

Antes de comenzar, es importante aclarar algunos conceptos sobre qué son imágenes y contenedores y cuales son sus características Algunos conceptos a tener en cuenta son:

- **Imágenes**:
  - La imagen es una plantilla de solo lectura que se utiliza para crear contenedores. A partir de una imágen pueden crearse múltiples contenedores.
  - Las imágenes, además de tener sus sistema de ficheros predefinido, tienen una serie de parámetros predefinidos (comandos, de variables de entorno, etc.) con valores por defecto y que se pueden personalizar en el momento de crear el contenedor.
  - Docker permite crear nuevas imágenes basándose en imágenes anteriores. Se podría decir que una imagen puede estar formada por un conjunto de "capas" que han modificado una imagen base.
    - Al crear una nueva imagen, simplemente estamos añadiendo una capa a la imagen anterior, la que actúa como base.
- **Contendores**:
  - Son instancias de una imágen.
  - Pueden ser arrancados, parados y ejecutados.
  - Cada contenedor Docker posee un **identificador único de 64 caracteres**, pero habitualmente se utiliza una **versión corta con los primeros 12 caracteres**.
    - Los comandos Docker habitualmente soportan ambas versiones.

Un símil para entender estos conceptos: una instalación de una distribución de Linux mediante un DVD. Ese DVD sería nuestra imagen y el sistema operativo instalado sería el contenedor.

Detallaremos conceptos relacionados sobre la creación de imágenes en futuras unidades.

3.2 ¿Dónde se almacenan imágenes, contenedores y datos?

El lugar donde se almacenan contenedores e imágenes puede variar según distribución/sistema operativo, driver de almacenamiento y versión de Docker. Normalmente mediante el siguiente comando de Docker, podemos ver información del sistema, incluyendo el directorio de Docker.

```
docker info
```

Ese comando nos ofrece información sobre el estado de Docker. Para conocer donde se almacena la información, dos datos son importantes: directorio de Docker y driver de almacenamiento.

Driver de almacenamiento utilizado por Docker:

```
Storage Driver: overlay2
 Backing Filesystem: extfs
 Supports d_type: true
 Native Overlay Diff: true
```

En el ejemplo se nos indica el driver de almacenamiento utilizado (“***overlay2***”) y sobre qué sistema de archivos está funcionando (“***extfs***”, en concreto “***ext4***”).

Para saber más sobre los distintos drivers de almacenamiento de Docker, podéis consultar https://docs.docker.com/storage/storagedriver/select-storage-driver/

Directorio de Docker:

```
Docker Root Dir: /var/lib/docker
```

Directorio donde se almacena todo lo relacionado con Docker.

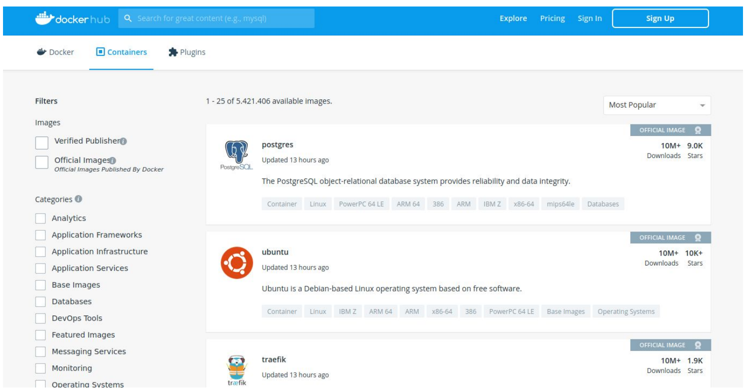
Al utilizar el driver “***overlay2***” sabemos que

- La información de las imágenes se encuentra en “***/var/lib/docker/overlay2***”.
  - Recordemos que las imágenes se basan en capas, una imagen puede estar creada por un conjunto de capas.
  - El almacenamiento temporal de contenedores (es decir, pequeños cambios en los contenedores), se realiza como “versiones de las imágenes”, es decir, una capa más de la imagen base.
- La configuración de los contenedores se almacena en “***/var/lib/docker/containers***”.
- Para acceso a datos compartidos, persistencia, etc, etc. existe la figura de los volúmenes, la cual detallaremos en posteriores unidades.

4. Registro: Docker Hub

Docker Hub es una “plataforma de registro” de Docker. Los servicios básicos son gratuitos y nos permite registrar imágenes Docker, haciéndolas públicas o privadas.

Contiene un gran ecosistema de imágenes ya creadas, usualmente con instrucciones de instalación y uso, además de un buscador que nos permite encontrar imágenes según distintos parámetros. Enlace al buscador https://hub.docker.com/search?q=&type=image



Por defecto, Docker utiliza esta plataforma registro como “registro por defecto”, aunque es posible si se requiere, elegir otro servicio de registro, e incluso montar un servicio privado de registro.

Más información sobre cómo crear un registro privado en

- https://www.digitalocean.com/community/tutorials/how-to-set-up-a-private-docker-registry-on-ubuntu-18-04-es

5. Creando y arrancando contenedores con “docker run”

5.1 ¿Que hace el comando “docker run”?

Es posiblemente el comando “docker run” más utilizado. Podríamos decir que **este comando crea un contenedor a partir de una imagen y lo arranca.**

Importante: un error común es creer que ***“docker run”*** solo arranca contenedores. Si haces varios ***“docker run”***, estás creando varios contenedores, no arrancando varias veces un contenedor.

La descripción completa del comando “docker run” la podéis encontrar en https://docs.docker.com/engine/reference/commandline/run/

Comentaremos a lo largo de esta unidad algunas de sus opciones básicas más importantes. Además en futuras unidades, ampliaremos los conocimientos sobre este comando.

5.2 Creando contenedores sin arrancarlos

Para crear un contenedor sin arrancarlo (recordamos, ***“docker run”*** crea y arranca), existe el comando ***“docker create”***. La descripción completa del comando ***“docker create”*** la podéis encontrar en https://docs.docker.com/engine/reference/commandline/create/

5.3 Repasando caso práctico “Hello World”

En anteriores unidades propusimos un sencillo caso práctico para comprobar que funcionaba Docker usando el siguiente comando:

```
docker run hello-world
```

5.3.1 Repaso parte 1: obteniendo la imagen

Al ejecutar este comando por primera vez, obtenemos un resultado similar a este:

```
sergi@ubuntu:~$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
b8dfde127a29: Pull complete
Digest: sha256:89b647c604b2a436fc3aa56ab1ec515c26b085ac0c15b0d105bc475be15738fb
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/
```

La documentación en Docker Hub del contenedor que estamos lanzando la tenemos disponible en https://hub.docker.com/_/hello-world

En primer lugar, nos fijamos en el comienzo de la información mostrada, concretamente en:

```
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
b8dfde127a29: Pull complete
Digest: sha256:89b647c604b2a436fc3aa56ab1ec515c26b085ac0c15b0d105bc475be15738fb
Status: Downloaded newer image for hello-world:latest
```

Ahí se nos indica que la imagen ***"hello-word:latest"*** no está localmente en nuestro sistema. Al no estar, se descarga del registro por defecto (normalmente Docker Hub) y se almacena localmente.

De hecho, si volvemos a hacer el comando "docker run hello-world", al tener la imagen ya en el sistema, **no nos aparecerá este texto, ya que la imagen la tenemos almacenada localmente.**

Otro aspecto a destacar, es que pese a que solo hemos escrito ***"hello-world"***, nos ha descargado una imagen llamada ***"hello-world:latest"***. Esto es porque cada imagen creada tiene un nombre de versión. Si no indicamos nada o indicamos "***latest"***, nos instala la última versión. Si queremos instalar una versión concreta de una imagen se indica de la forma ***"imagen:nombreversion"***.

5.3.2 Repaso parte 2: el contenedor se crea y ejecuta un comando

Una vez descargada la imagen, se crea el contenedor, se inicia y ejecuta un proceso. Este proceso lo podemos proporcionar dentro de la orden "docker run" o si como en el caso concreto de este ejemplo, no lo hemos proporcionado, ejecutará un comando predefinido por la propia imagen.

En este caso concreto, al no haber especificado ningún comando, al iniciarse el contenedor lanza un programa por defecto llamado "hello" y nos muestra por la salida estándar información de como Docker ha generado este mensaje:

```
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.
```

Si tenéis curiosidad por ver el código fuente del programa "hello", está disponible en https://github.com/docker-library/hello-world/blob/master/hello.c

Este programa genera un texto que básicamente nos explica que el cliente de Docker se ha conectado con el servicio de Docker, este se ha descargado la imagen de Docker Hub (o localmente si ya estaba en nuestro sistema), se ha creado un contenedor, que por defecto tenía un comando que generaba la salida que estamos leyendo y finalmente, el servicio de Docker lo ha enviado a la terminal.

6. LISTAR CONTENEDORES DISPONIBLES EN EL SISTEMA CON "DOCKER PS"

Mediante el comando ***"docker ps"*** podemos lista contenedores en el sistema, tanto parados como en ejecución. Si ejecutamos el siguiente:

```
docker ps
```

Nos aparecerá un listado como este si no tenemos ningún contenedor en ejecución:

```
sergi@ubuntu:~$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
```

O si tenemos contenedores en ejecución, podemos obtener algo similar a esto:

```
sergi@ubuntu:~$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED         STATUS         PORTS     NAMES
434d318b3771   ubuntu    "bash"    9 seconds ago   Up 7 seconds             stupefied_colden
```

Si lanzamos el comando

```
docker ps -a
```

Obtendremos un listado de todos los contenedores, tanto aquellos en funcionamiento como aquellos que están parados.

```
sergi@ubuntu:~$ docker ps -a
CONTAINER ID   IMAGE         COMMAND       CREATED            STATUS                     PORTS     NAMES
434d318b3771   ubuntu        "bash"        About a minute ago Up About a minute                    stupefied_colden
b0619341b82c   ubuntu        "/bin/bash"   3 minutes ago      Exited (0) 2 minutes ago             intelligent_ritchie
3fb53852ce99   hello-world   "/hello"      37 minutes ago     Exited (0) 37 minutes ago            recursing_meninsky
77dc9f7f80e5   hello-world   "/hello"      41 minutes ago     Exited (0) 41 minutes ago            priceless_pare
```

La información que obtenemos de los contenedores es la siguiente:

- **CONTAINER_ID**: identificador único del contenedor (versión 12 primeros caracteres).
- **IMAGE**: imagen utilizada para crear el contenedor.
- **COMMAND**: comando que se lanza al arrancar el contenedor.
- **CREATED**: cuando se creó el contenedor.
- **STATUS**: si el contenedor está en marcha o no (indicando cuánto lleva en marcha o cuánto hace que se paró).
- **PORTS**: redirección de puertos del contenedor (lo veremos más adelante en la unidad).
- **NAMES**: nombre del contenedor. Se puede generar como parámetro al crear el

contenedor, o si no se indica nada, el propio Docker genera un nombre aleatorio.

La descripción completa del comando ***"docker ps"*** la podéis encontrar en https://docs.docker.com/engine/reference/commandline/ps/

7. Parando y arrancando contenedores existentes con "docker start/stop/restart"

Para arrancar/parar un contenedor ya creado (recordamos, ***"docker run"*** crea y arranca), existen los comandos ***"docker start", "docker stop" y "docker restart"***.

La forma más habitual de usar estos comandos, es usar el nombre del comando, seguido del identificador único o nombre asignado al contendor. Por ejemplo con identificador:

```
docker start 434d318b3771
```

o con nombre del contenedor

```
docker start stupefied_colden
```

La descripción completa de estos comandos la podéis encontrar en

- https://docs.docker.com/engine/reference/commandline/start/
- https://docs.docker.com/engine/reference/commandline/stop/
- https://docs.docker.com/engine/reference/commandline/restart/

8. Inspeccionando contenedores con "docker inspect"

El comando "***docker inspect***" es un comando que nos proporciona diversos detalles de la configuración de un contenedor. Ofrece distintos datos, entre ellos, identificador único (versión 64 caracteres), almacenamiento, red, imagen en que se basa, etc. Su sintaxis es:

```
docker inspect IDENTIFICADOR/NOMBRE
```

La descripción completa del comando ***"docker inspect"*** la podéis encontrar en https://docs.docker.com/engine/reference/commandline/inspect/.

9. Ejecutando comandos en un contenedor con "docker exec"

El comando ***"docker exec"*** nos permite ejecutar un comando dentro de un contenedor que esté en ese momento en ejecución. La forma sintaxis habitual para utilizar este comando es la siguiente

```
docker exec [OPCIONES] IDENTIFICADOR/NOMBRE COMANDO [ARGUMENTOS]
```

Algunos ejemplos de uso, suponiendo un contenedor en marcha llamando "contenedor":

```
docker exec -d contenedor touch /tmp/prueba
```

Ejemplo que se ejecuta ejecuta en "background", gracias al parámetro "-d". Este ejemplo simplemente crea mediante el comando "touch" un fichero "prueba" en "/tmp".

```
docker exec -it contenedor bash
```

Orden que ejecutará la "shell" bash en nuestra consola (gracias al parámetro "-it" se enlaza la entrada y salida estándar a nuestra terminal). A efectos prácticos, con esta orden accederemos a una "shell" bash dentro del contenedor.

```
docker exec -it -e VAR=1 contenedor bash
```

Comando que establece un variable de entorno con el parámetro **"-e"**. Se enlaza la entrada y salida de la ejecución del comando con **"-it"**. A efectos prácticos, en esa "shell" estará disponible la variable de entorno **"VAR1"** con valor 1. Lo podemos probar con ***"echo $VAR1"***.

La descripción completa del comando ***"docker exec"*** la podéis encontrar en https://docs.docker.com/engine/reference/commandline/exec/.

10. Copiando ficheros entre anfitrión y contenedores con "docker cp"

El comando ***"docker cp"*** es un comando que nos permite copiar ficheros y directorios del anfitrión a un contenedor o viceversa. No se permite actualmente la copia de fichero entre contenedores.

Algunos ejemplos de uso:

```
docker cp idcontainer:/tmp/prueba ./
```

Copia el fichero "***/tmp/prueba***" del contenedor con identificador o nombre "idcontainer" al directorio actual de la máquina que ejerce como anfitrión.

```
docker cp ./miFichero idcontainer:/tmp
```

Copia el fichero "***miFichero***" del directorio actual al directorio "/tmp" del contenedor.

La descripción completa del comando ***"docker cp"*** la podéis encontrar en https://docs.docker.com/engine/reference/commandline/cp/.

11. ACCEDIENDO A UN PROCESO EN EJECUCIÓN CON "DOCKER ATTACH"

En algunos casos, deseamos enlazar la entrada o salida estándar de nuestra terminal a un contenedor que está ejecutando un proceso en segundo plano, de forma similar a la siguiente

```
docker attach [OPCIONES] IDENTIFICADOR/NOMBRE
```

Para probarlo utilizaremos el siguiente ejemplo:

El ejemplo consiste en crear un contenedor que lanza un proceso que genera texto (imprimiendo la fecha) por la salida estándar de forma indefinida. El comando llama a "sh" con el parámetro -c (que indica que la siguiente cadena es algo a procesar por la "shell" sh), seguido de una cadena con un "shell script". Aquí vemos el comando, que podríamos lanzar en cualquier terminal.

```
sh -c "while true; do $(echo date); sleep 1; done"
```

Aplicamos este comando a nuestro ejemplo creando un contenedor:

```
docker run -d --name=muchotexto busybox sh -c "while true; do $(echo date); sleep 1; done"
```

❕ **Atención:** Los parámetros de este "docker run" son explicados más adelante en el documento.

Con ese contenedor en marcha, ya podemos probar ***"docker attach".*** Podremos enlazar la entrada y salida del proceso en ejecución a nuestra terminal y observar el texto generado usando:

```
docker attach muchotexto
```

La descripción completa del comando ***"docker attach"*** la podéis encontrar en https://docs.docker.com/engine/reference/commandline/attach/.

12. OBTENIENDO INFORMACIÓN DE LOS LOGS CON "DOCKER LOGS"

Podemos consultar la información generada con el comando "***docker logs***"

```
docker logs [OPCIONES] IDENTIFICADOR/NOMBRE
```

Este uso es similar a "docker attach", solo que tiene opciones específicas para tratar la información obtenida como un log. Partiendo del mismo ejemplo usado en "***docker attach***".

```
docker run -d --name=muchotexto busybox sh -c "while true; do $(echo date); sleep 1; done"
```

Un ejemplo de uso para obtener logs podría ser

```
docker logs -f --until=2s muchotexto
```

Con este ejemplo, te mostraría los logs generados (realmente la salida estándar y de error), incluyendo aquellos que se fueran generando, parando a los dos segundos.

La descripción completa del comando ***"docker logs"*** la podéis encontrar en https://docs.docker.com/engine/reference/commandline/logs/.

13. Renombrando contenedores con "docker rename"

El comando ***"docker rename"*** nos permite cambiar el nombre asociado a un contenedor.

```
docker rename contenedor1 contenedor2
```

Cambia el nombre de "***contenedor1***" a "***contenedor2***".

La descripción completa del comando ***"docker rename"*** la podéis encontrar en https://docs.docker.com/engine/reference/commandline/rename/.

14. Principales parámetros del comando "docker run"

Anteriormente hemos indicado que el comando ***"docker run"*** es de gran importancia en el uso de Docker y que este nos permite crear contenedores a partir de una imagen y arrancarlos.

La estructura principal del comando es la siguiente

```
docker run [PARAMETROS] IMAGEN [COMANDO AL ARRANCAR] [ARGUMENTOS]
```

A continuación mostramos algunos ejemplos de ***"docker run"***.

14.1 Ejemplo 1: lanzando Ubuntu y accediendo a una terminal

Utilizando el comando

```
docker run -it --name=nuestroUbuntu1 ubuntu /bin/bash
```

Estamos creando un nuevo contenedor a partir de la imagen "***ubuntu***". Al crear este contenedor hemos especificado los siguientes parámetros:

- **Parámetro "-i":** indica que el proceso lanzado en el contenedor docker estará en modo interactivo, es decir, enlaza la entrada estándar cuando se asigna un proceso a una terminal.
- **Parámetro "-t"**: asigna al proceso lanzado al arrancar el contenedor una pseudo terminal, facilitando el acceso al mismo desde nuestra terminal.
- **Parámetro "--name"**: nos permite establecer un nombre a nuestro contenedor. Si no indicamos este parámetro, nos creará un nombre aleatorio.

Por último, el comando ejecutado al lanzarse el contenedor es ***"/bin/bash"***. Esto combinado con los parámetros ***"-it"*** (que entraban en modo interactivo y asociaban una pseudoterminal), nos hace que justo después de lanzar el comando, estemos en una "shell" dentro del contenedor creado. Al finalizar dicha "shell" (con "exit", "control+c", etc.) el contenedor se parará.

Los cambios que hayamos hecho con la "shell", como por ejemplo, crear un directorio, se almacenarán como una imagen temporal (veremos en profundidad las imágenes en otra unidad) y a efectos prácticos, los cambios serán permanentes al arrancar de nuevo este contenedor.

14.2 Ejemplo 1 EXTRA: accediendo a terminal desde el contenedor parado

El anterior ejemplo nos permitía acceder crear un contenedor y acceder de forma interactiva a dicha "shell", pero al salir de la shell, simplemente se paraba el contenedor.

Entonces ¿Cómo podríamos volver a ese contenedor y a dicha "shell"? Usaremos ***"docker start"***.

Este comando nos permitirá arrancar el contenedor parado. Al arrancar no especificaremos un comando a lanzar, ya que se lanza el comando que hayamos especificado (o por defecto de la imagen si no hemos especificado nada) al hacer **"docker run" o "docker create".**

El comando ***"docker start"*** sigue la siguiente estructura:

```
docker start [PARAMETROS] IDENTIFICADOR/NOMBRE
```

Podemos obtener identificador único o nombre usando el comando:

```
docker ps -a
```

Tras ello, lanzamos el contenedor de la siguiente forma:

```
docker start -ai IDENTIFICADOR
```

Los parámetros especificados a ***"docker start"*** son los siguientes:

- **Parámetro "-a"**: al arrancar el contenedor, enlaza la salida estándar y de error del contenedor a nuestra terminal.
- **Parámetro "-i":** al arrancar el contenedor, lo hace en modo interactivo, es decir enlazando la entrada estándar del contenedor a nuestra terminal.

14.3 Ejemplo 2: ejecutando una versión de una imagen y auto-eliminando el contenedor

Lanzando el siguiente comando

```
docker run -it --rm  ubuntu:14.04 /bin/bash
```

Estamos creando un contenedor con la versión de la imagen ***"ubuntu"*** etiquetada como ***"14.04"*** en Docker Hub y arrancandolo de forma similar al ejemplo anterior.

Los parámetros nuevos incluidos en esta orden son:

- **Parámetro "--rm":** este parámetro hará que nada más el contenedor se pare, se borre el contenedor del sistema.

14.4 Ejemplo 3: lanzando un servidor web en background y asociando sus puertos

Lanzando el siguiente comando

```
docker run  -d -p 1200:80 nginx
```

Estamos creando un contenedor con la versión de la imagen ***"nginx:latest"***, la cual contiene un servidor web Nginx en funcionamiento en el puerto 80 del contenedor y al que podremos acceder en nuestra máquina como ***"localhost:1200"***.

Los parámetros nuevos incluidos en esta orden son:

- **Parámetro "-d":** parámetro "***deatached***", que indica que lanza el contenedor en segundo plano. Al lanzarlo con esta opción, no se nos muestra ninguna información de la entrada/salida del contenedor. La única información que se nos muestra es el ID del contenedor lanzado.
- **Parámetro "-p":** siguiendo el estilo ***"pAnf:pCont"*** nos indica que en el puerto de la máquina anfitrión "***pAnf***" está enlazado con el puerto interno del contenedor "***pCont***".
    - Si solo se indica un puerto, algo del estilo ***"-p 80"***, el sistema tomará dicho puerto como el puerto interno del contenedor y asociará un puerto aleatorio libre de la máquina anfitrión. Podremos consultar los puertos expuestos de un contenedor mediante el comando ***"docker ps"*** o el específico para esta tarea ***"docker port"*** .

❕ **Atención:** el mapeo de puertos **solo puede realizarse en el momento de crear el contenedor.** No se puede modificar el mapeo de puertos con el contenedor ya creado.

Para saber más sobre la imagen que hemos utilizado, en este caso "nginx" podemos consultar su página en Docker Hub https://hub.docker.com/_/nginx

14.5 Ejemplo 3 EXTRA: cambiando el "index.html" y consultando logs

Observando la documentación que nos ofrece sobre la imagen https://hub.docker.com/_/nginx, observamos que la ruta donde se encuentra la página que sirve ***"nginx"*** se encuentra en ***"/usr/share/nginx/html"***. Accediendo a esa ruta, podríamos modificar el "***index.html***" que se ve cuando nos conectamos al puerto 1200 en nuestra máquina.

Con las herramientas que tenemos, tenemos varias acciones para modificarlo

- Accede con una "shell" con ***"docker exec"***, instalar un editor de texto (por ejemplo con ***"apt update; apt install nano"***) y editar el fichero desde la consola.
- Copiar ***"index.html"*** desde nuestra máquina anfitrión con ***"docker cp"***.

También podemos acceder a los logs que nos va generando durante su ejecución. Si por ejemplo queremos acceder a las últimas 10 líneas de logs generados, podemos utilizar

```
docker logs -n 10 busy_kapitsa
```

14.6 Ejemplo 4: estableciendo variables de entorno

Vamos a ver un sencillo ejemplo donde vamos a establecer una variable de entorno e imprimir su valor en pantalla. Ejecutamos el siguiente comando

```
docker run  -it -e MENSAJE=HOLA ubuntu bash
```

Con este ejemplo, al crear el contenedor hemos establecido la variable de entorno "MENSAJE" y lanzado una terminal. Podemos probar que la variable se ha establecido correctamente usando:

```
echo $MENSAJE
```

Los parámetros nuevos incluidos en esta orden son:

- **Parámetro "-e":** simplemente nos permite establecer una o varias "variables de entorno".

Este simple ejemplo nos indica cómo establecer variables de entorno al construir un contenedor.

También, en el momento de la creación de imágenes, se puede establecer variables de entorno con valores por defecto de cada imagen. Estos valores se mantendrán, salvo que sean sobreescritos con el parámetro ***"-e"***.

15. BIBLIOGRAFÍA

[1] Docker Docs https://docs.docker.com/