

<b>T�tulo</b>	4. Docker. Gesti�n de im�genes y contenedores
<b>Destinatario</b>	1� DAW – Semipresencial
<b>Autor</b>	Pascual Mart�nez
<b>Correo</b>	<a href="mailto:pmartinez@florida-uni.es">pmartinez@florida-uni.es</a>

## 4. Docker. Gesti n de im genes y contenedores

1.	Introducci�n.....	2
2.	Creaci�n de una imagen incluyendo nuestra aplicaci�n. ....	3
3.	Integraci�n en un registro.....	9
4.	Ejecuci�n de contenedores .....	12
5.	Ciclo de vida de los contenedores .....	16
6.	Referencias. ....	17

T�tulo	4. Docker. gesti�n de im�genes y contenedores
Destinatario	1� DAW – Semipresencial
Autor	Pascual Mart�nez
Correo	<a href="mailto:pmartinez@florida-uni.es">pmartinez@florida-uni.es</a>

## 1. Introducci n

Llegados a este punto, vamos a recapitular un poco la informaci n desplegada hasta la fecha, para tratar de resumir o sintetizar conceptos. **El objetivo principal de este documento va a ser entender el proceso para generar un contenedor personalizado, en el que podremos incluir nuestra propia aplicaci n** o microservicio y, conocer cada uno los pasos del proceso para poder conseguirlo. Nuestra aplicaci n puede consistir en algo muy sencillo del tipo “Hola mundo”, o algo m s complejo si lo prefieres. Ese aspecto no es relevante en este momento. Quien quiera, es libre para buscar otros retos y generar un contenedor con otro tipo de aplicaciones, es todo un mundo...

Hasta ahora hemos hablado de contenedores de una forma conceptual y abstracta, porque es importante tener claro lo que son, el motivo por el que han surgido y el objetivo que persiguen, antes de comenzar a “mancharnos las manos” con ellos. De este modo hemos ido rellenando progresivamente nuestro conocimiento mediante “capas incrementales e interconectadas”, que nos van a permitir llegar a realizar “muchas cosas” con s lo varios comandos o varias acciones mediante interfaz gr fica. En este documento vamos a manejar im genes y contenedores, desde su creaci n o puesta en marcha hasta su detenci n o eliminaci n.

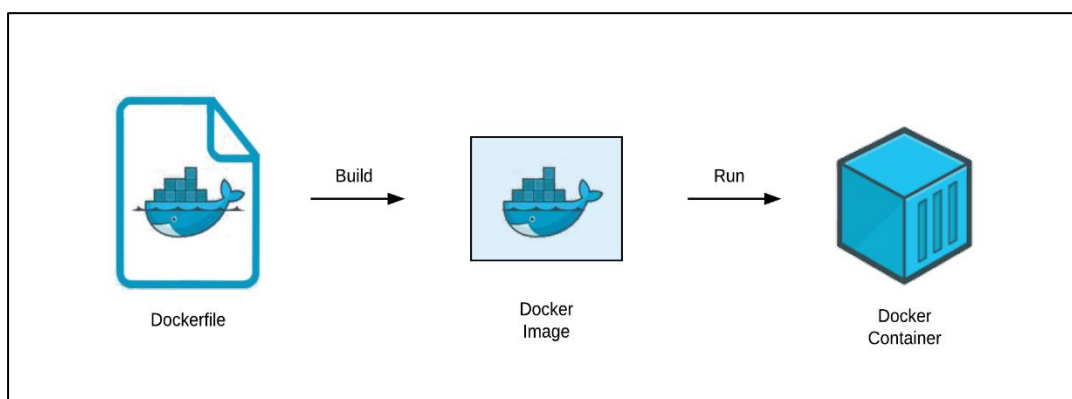


<b>T�tulo</b>	4. Docker. gesti�n de im�genes y contenedores
<b>Destinatario</b>	1� DAW – Semipresencial
<b>Autor</b>	Pascual Mart�nez
<b>Correo</b>	<a href="mailto:pmartinez@florida-uni.es">pmartinez@florida-uni.es</a>

Es importante que no nos perdamos o atasquemos en los detalles. Es decir, durante el proceso vamos a ver c mo generar una imagen y un posterior contenedor, en base a unos factores concretos definidos. No es relevante memorizar los par metros de un fichero de configuraci n o de un comando. Son muchas las opciones y posibilidades a explorar y es necesaria una amplia experiencia para poder generarlo todo “de memoria” y conocer todos los escenarios posibles. Lo m s importante es entender los procedimientos y los conceptos clave.

## 2. Creaci n de una imagen incluyendo nuestra aplicaci n.

Si recordamos contenidos anteriores, una imagen de Docker es una especie de plantilla o molde, en base al cual podremos generar tantos contenedores como necesitemos. Todos los contenedores generados en base a una misma plantilla ser n iguales. Por lo tanto, se puede deducir que una plantilla debe proporcionar la **informaci n** necesaria para poder generar un contenedor. Esta informaci n, a su vez, viene definida desde un **fichero descriptivo llamado Dockerfile** que se utiliza para generar la plantilla o imagen.



Por lo tanto, vamos a ver a trav s de un ejemplo c mo el contenido de un fichero **Dockerfile** permite encapsular en un contenedor, una **aplicaci n web personalizada** junto con un SO, un servidor web y la configuraci n necesaria para que  sta responda. Este resultado se logra mediante la indicaci n en el fichero de ciertas sentencias descriptivas que van a posibilitar la generaci n de un entorno de ejecuci n y/o desarrollo, en cuesti n de poco tiempo.

<b>T�tulo</b>	4. Docker. Gesti�n de im�genes y contenedores
<b>Destinatario</b>	1� DAW – Semipresencial
<b>Autor</b>	Pascual Mart�nez
<b>Correo</b>	<a href="mailto:pmartinez@florida-uni.es">pmartinez@florida-uni.es</a>

Poniendo un ejemplo un tanto extremo, pensad en lo que puede suponer a nivel de coste econ mico: comprar un servidor, ensamblarlo, integrarlo en la sala de servidores, integrarlo en red, instalarle antivirus, configurar copia de seguridad, instalarle un entorno de desarrollo (como m nimo) y comenzar a desarrollar.... Y adem s de todo eso, no nos olvidemos de lo m s importante, lo irrecuperable, el tiempo necesario para atender y llevar a cabo todas esas tareas.

**Una imagen de Docker consta de varias capas de s lo lectura, cada una de las cuales viene representada por una instrucci n en el fichero de Dockerfile, por orden de aparici n. Las capas se apilan entre s  y cada capa va adicionando novedades a lo existente en capas anteriores, es decir un delta, un incremental.**

```
FROM ubuntu
MAINTAINER ARSTECH arstech@e-mail
RUN apt-get update && apt-get upgrade -y
RUN apt-get install -y apt-utils htop
CMD ["echo","It's my Docker Image "]
```



Las instrucciones de un fichero Dockerfile son variables y m ltiples. Depender n del servicio que se pretenda generar, en funci n del escenario y las circunstancias. Por ejemplo, nos puede interesar poner en marcha un servidor web, o un servidor de bases de datos, o un entorno de desarrollo.

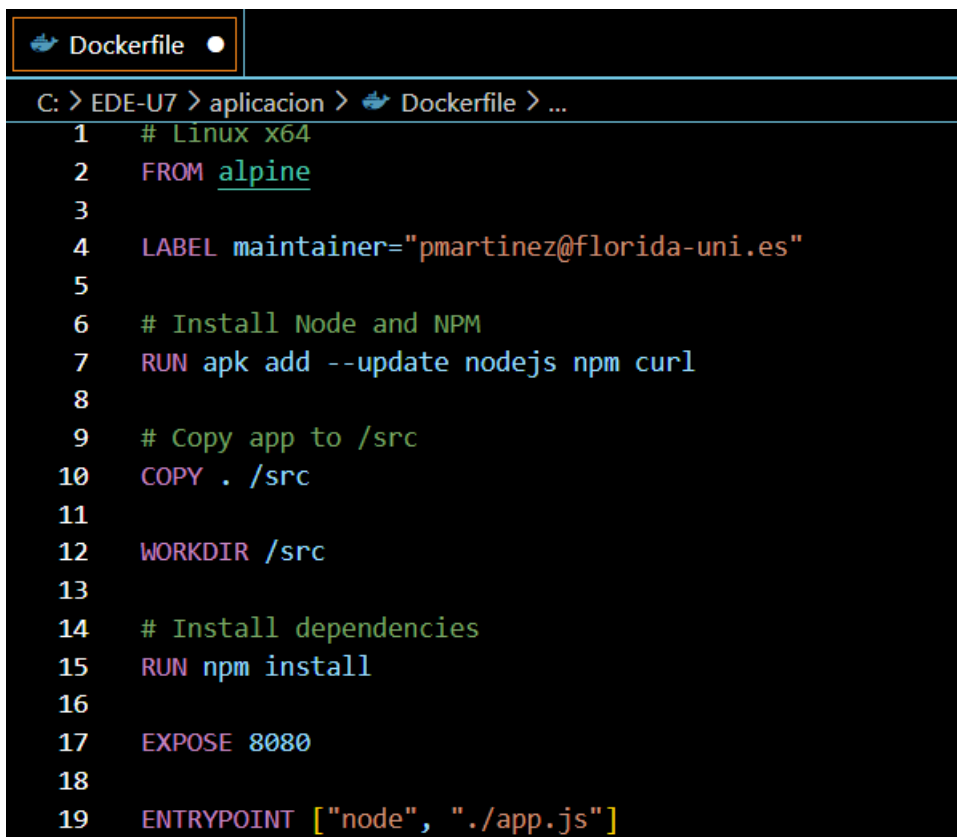
Aunque no es el objetivo en estos momentos, si tuvi ramos que generar un fichero Dockerfile desde cero y utilizarlo para montar otros escenarios, disponemos de informaci n oficial en el portal de Docker:

<https://docs.docker.com/reference/dockerfile/>

En nuestro caso de ejemplo, vamos a suponer que disponemos del fichero Dockerfile predefinido, con las instrucciones de la siguiente imagen.

<b>T�tulo</b>	4. Docker. Gesti�n de im�genes y contenedores
<b>Destinatario</b>	1� DAW – Semipresencial
<b>Autor</b>	Pascual Mart�nez
<b>Correo</b>	<a href="mailto:pmartinez@florida-uni.es">pmartinez@florida-uni.es</a>

El nombre de la cada instrucci n se indica en may sculas, pero el contenido o par metros asociados a cada instrucci n se indican en min sculas:



```

1  # Linux x64
2  FROM alpine
3
4  LABEL maintainer="pmartinez@florida-uni.es"
5
6  # Install Node and NPM
7  RUN apk add --update nodejs npm curl
8
9  # Copy app to /src
10 COPY . /src
11
12 WORKDIR /src
13
14 # Install dependencies
15 RUN npm install
16
17 EXPOSE 8080
18
19 ENTRYPOINT ["node", "./app.js"]

```

Donde:

- **#** : texto informativo, comentarios.
- **FROM**: es el inicio de la imagen. Crear  una capa base, a partir de otra imagen en Docker hub del SO que usaremos, en este caso “alpine” (Linux). El resto de las capas se “montar n encima”.
- **LABEL maintainer**: se trata de una **etiqueta**. Las etiquetas pueden personalizar un contenedor, mediante metadatos. En este caso, se refiere al **contacto de la persona** que se va a encargar del **mantenimiento** de ese contenedor. Es meramente informativo.

<b>T�tulo</b>	4. Docker. Gesti�n de im�genes y contenedores
<b>Destinatario</b>	1� DAW – Semipresencial
<b>Autor</b>	Pascual Mart�nez
<b>Correo</b>	<a href="mailto:pmartinez@florida-uni.es">pmartinez@florida-uni.es</a>

- **RUN: ejecuta comandos y crea capas.** En la 1  aparici n, RUN genera una capa con el entorno de ejecuci n para JavaScript, NodeJS, y un par de utilidades (npm, un gestor de paquetes para las dependencias de proyectos y curl, una herramienta para lanzar peticiones web desde la l nea de comandos del contenedor). En la 2  aparici n, RUN genera una capa con las dependencias indicadas en el fichero “package.json” del directorio de trabajo.

- **COPY: agrega “nuestros ficheros” dentro de la imagen,** como una **nueva capa**. Se define un **origen** y un **destino**.

Es interesante resaltar que, al hacerlo de este modo, la imagen se crear  con nuestra aplicaci n en el estado en que est  en ese momento. Si hacemos cambios posteriormente en nuestra aplicaci n en la m quina f sica, no se actualizar n en el contenedor, y viceversa tampoco. Es como una foto de lo que hay en ese momento en la m quina anfitriona. Para variar esa situaci n, existen mecanismos que permiten conectar un contenedor con tu sistema de archivos en tiempo real. A este mecanismo se le denomina **volumen** y hablaremos de  l posteriormente.

- **WORKDIR:** establece el **directorio de trabajo**, para las siguientes instrucciones.
- **EXPOSE:** indica el **puerto en el que escuchar  el contenedor** en tiempo de ejecuci n.
- **ENTRYPOINT:** indica la acci n que se ejecutar  cuando arranque un contenedor.

Una vez disponemos del fichero Dockerfile, la sintaxis del comando para generar una imagen es la siguiente:

**docker image build [OPTIONS] PATH | URL | -**

Donde:

- La palabra “image” es **opcional**.
- **[OPTIONS]:** son las opciones o par metros del comando.
- **PATH | URL | - :** ruta o direcci n donde encontrar el contexto necesario para crear la imagen (fichero Dockerfile, ...).

<b>T�tulo</b>	4. Docker. gesti�n de im�genes y contenedores
<b>Destinatario</b>	1� DAW – Semipresencial
<b>Autor</b>	Pascual Mart�nez
<b>Correo</b>	<a href="mailto:pmartinez@florida-uni.es">pmartinez@florida-uni.es</a>

Ejemplo gen rico:

**docker image build -t nombre\_imagen:tag .**

Donde:

- “-t”, es **una de las opciones del comando**, indica que la imagen se identificar  con un nombre y una etiqueta (“tag”). La **etiqueta** es  til para indicar la **versi n** de la imagen.
- “**nombre\_imagen:tag**” es el nombre que queramos ponerle a la imagen y una etiqueta.

**\*\*NOTA:** En cuanto al nombre, se recomienda usar la siguiente estructura con el objetivo de facilitar la ejecuci n del siguiente paso:

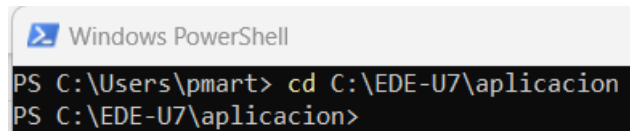
**nombre\_imagen = nombre\_repositorio\_docker\_hub/nombre\_que\_yo\_quiera**

Es decir, deber amos establecer como prefijo en el nombre de la imagen: **nuestro repositorio Docker Hub**.

- “.” En este caso, ser  el PATH, que indica que se incluyan los ficheros de contexto del directorio actual (fichero Dockerfile, ...).

**Ejemplo:**

Accedemos a la carpeta donde est  ubicado el fichero Dockerfile y la aplicaci n:



```
Windows PowerShell
PS C:\Users\pmart> cd C:\EDE-U7\aplicacion
PS C:\EDE-U7\aplicacion>
```

Ejecutamos el comando para generar la imagen:

**docker image build -t pmartinezflorida/imagen\_u7:v0 .**

<b>Título</b>	4. Docker. Gestión de imágenes y contenedores
<b>Destinatario</b>	1º DAW – Semipresencial
<b>Autor</b>	Pascual Martínez
<b>Correo</b>	<a href="mailto:pmartinez@florida-uni.es">pmartinez@florida-uni.es</a>

```

Windows PowerShell
PS C:\EDE-U7\aplicacion> docker image build -t pmartinezflorida/imagen_u7:v0 .
[+] Building 18.5s (11/11) FINISHED
=> [internal] load build definition from Dockerfile                                0.0s
=> => transferring dockerfile: 301B                                              0.0s
=> [internal] load .dockerignore                                                  0.1s
=> => transferring context: 2B                                                  0.0s
=> [internal] load metadata for docker.io/library/alpine:latest                 1.4s
=> [auth] library/alpine:pull token for registry-1.docker.io                   0.0s
=> [internal] load build context                                                 0.2s
=> => transferring context: 1.28kB                                              0.0s
=> [1/5] FROM docker.io/library/alpine@sha256:ff6bdca1701f3a8a67e328815ff2346b0e4067d32ec36b7992c1fdc001dc8517  0.0s
=> CACHED [2/5] RUN apk add --update nodejs npm curl                          0.0s
=> [3/5] COPY . /src                                                            0.7s
=> [4/5] WORKDIR /src                                                          0.1s
=> [5/5] RUN npm install                                                         15.6s
=> exporting to image                                                           0.4s
=> => exporting layers                                                         0.4s
=> => writing image sha256:d8e761fb6be954d91f0ffa65ecb6c405fb25f32c3aac823a4d43054ecd48acd7  0.0s
=> => naming to docker.io/pmartinezflorida/imagen_u7:v0                      0.0s
PS C:\EDE-U7\aplicacion>

```

Una vez finalizada la ejecución del comando, podemos confirmar que la imagen se ha generado correctamente, listando las imágenes existentes desde nuestro directorio de trabajo, o bien listando concretamente la que acabamos de crear, indicando su nombre:tag, mediante el comando:

### **docker image ls nombre\_imagen:tag**

Donde:

- “**nombre\_imagen:tag**” es opcional, lo usamos para buscar directamente la imagen que hemos generado. Si no lo ponemos, saldrían todas las imágenes.

```

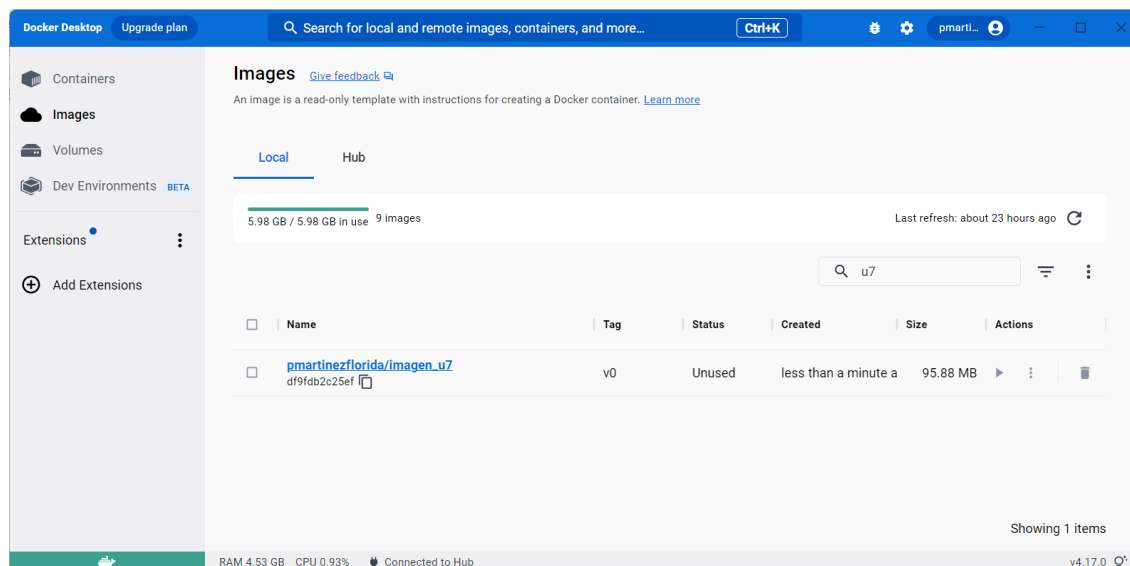
Windows PowerShell
PS C:\EDE-U7\aplicacion> docker image ls pmartinezflorida/imagen_u7:v0
REPOSITORY          TAG          IMAGE ID       CREATED        SIZE
pmartinezflorida/imagen_u7  v0          d8e761fb6be9  7 minutes ago  95.9MB
PS C:\EDE-U7\aplicacion>

```



<b>Título</b>	4. Docker. Gestión de imágenes y contenedores
<b>Destinatario</b>	1º DAW – Semipresencial
<b>Autor</b>	Pascual Martínez
<b>Correo</b>	<a href="mailto:pmartinez@florida-uni.es">pmartinez@florida-uni.es</a>

Una vez hayamos generado la imagen, podemos acceder a ella desde la aplicación gráfica, Docker Desktop:

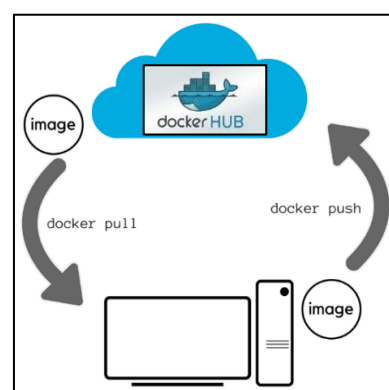


### 3. Integración en un registro.

Llegados a este punto, ya disponemos de una imagen Docker, donde se define todo el entorno necesario, empaquetado o encapsulado. La imagen está lista para generar y usar contenedores con las especificaciones explicadas anteriormente.

Vamos a aprovechar este momento para subir nuestra imagen a un registro de repositorios público. En este caso, usaremos el registro oficial de imágenes Docker, **Docker Hub**.

**Lo que conseguimos con este paso es que la imagen sea accesible desde otros hosts para generar o arrancar contenedores en base a ella.** Para poder utilizarla desde nuestro propio host, no sería necesario o imprescindible guardarla en un repositorio.



<b>T�tulo</b>	4. Docker. gesti�n de im�genes y contenedores
<b>Destinatario</b>	1� DAW – Semipresencial
<b>Autor</b>	Pascual Mart�nez
<b>Correo</b>	<a href="mailto:pmartinez@florida-uni.es">pmartinez@florida-uni.es</a>

Utilizaremos el siguiente comando, con la sintaxis:

**docker image push [OPTIONS] NAME[:TAG]**

Ejemplo gen rico:

**docker image push nombre\_imagen:tag**

**\*\*Nota:** recuerda que hay que estar validado en Docker Hub para realizar un push.

Si no hubi ramos usado como prefijo del nombre de la imagen, el nombre de nuestro repositorio de Docker Hub, al ejecutar este comando podemos obtener un error de permisos en el repositorio.

Siempre podemos volver a etiquetar o renombrar la imagen con el comando:

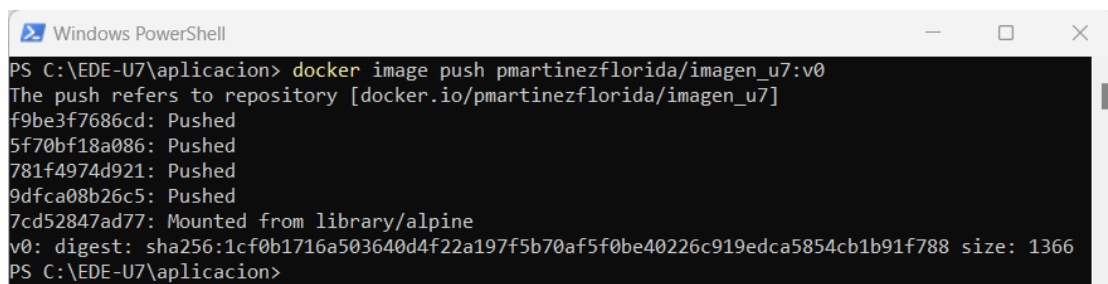
**docker tag SOURCE\_IMAGE[:TAG] TARGET\_IMAGE[:TAG]**

Ejemplo gen rico:

**docker tag nombre\_imagen:tag nombre\_repositorio\_docker\_hub /nombre\_que\_yo\_quiera:tag**

Ejemplo:

**docker image push pmartinezflorida/imagen\_u7:v0**



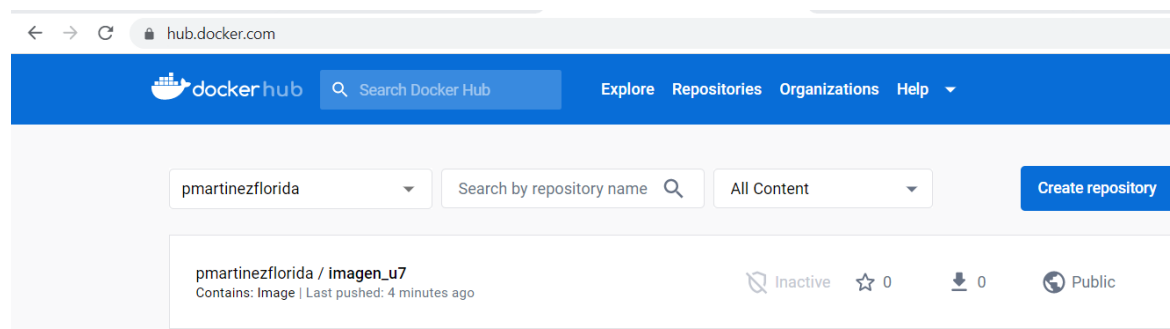
```

Windows PowerShell
PS C:\EDE-U7\aplicacion> docker image push pmartinezflorida/imagen_u7:v0
The push refers to repository [docker.io/pmartinezflorida/imagen_u7]
f9be3f7686cd: Pushed
5f70bf18a086: Pushed
781f4974d921: Pushed
9dfca08b26c5: Pushed
7cd52847ad77: Mounted from library/alpine
v0: digest: sha256:1cf0b1716a503640d4f22a197f5b70af5f0be40226c919edca5854cb1b91f788 size: 1366
PS C:\EDE-U7\aplicacion>

```

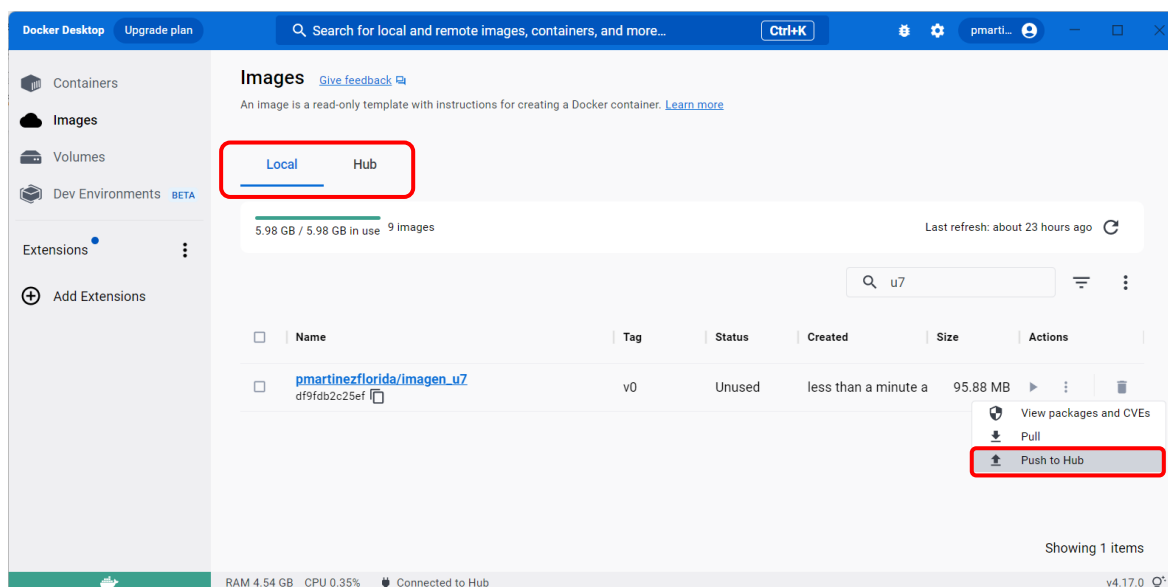
<b>T�tulo</b>	4. Docker. Gesti�n de im�genes y contenedores
<b>Destinatario</b>	1� DAW – Semipresencial
<b>Autor</b>	Pascual Mart�nez
<b>Correo</b>	<a href="mailto:pmartinez@florida-uni.es">pmartinez@florida-uni.es</a>

Una vez finalizada la ejecuci n del comando, dispondremos de nuestra imagen personalizada en nuestro repositorio del registro Docker Hub.



Del mismo modo, podr amos haber hecho la integraci n en el registro (push) desde Docker Desktop.

Una vez realizada la integraci n, la imagen aparecer  tambi n en la pesta a Hub.



T�tulo	4. Docker. gesti�n de im�genes y contenedores
Destinatario	1� DAW – Semipresencial
Autor	Pascual Mart�nez
Correo	<a href="mailto:pmartinez@florida-uni.es">pmartinez@florida-uni.es</a>

#### 4. Ejecuci n de contenedores

Como  ltimo paso, estamos en disposici n de generar, poner en marcha, o mover un contenedor, mediante el comando con la siguiente sintaxis:

**docker container run [OPTIONS] IMAGE [COMMAND] [ARG...]**

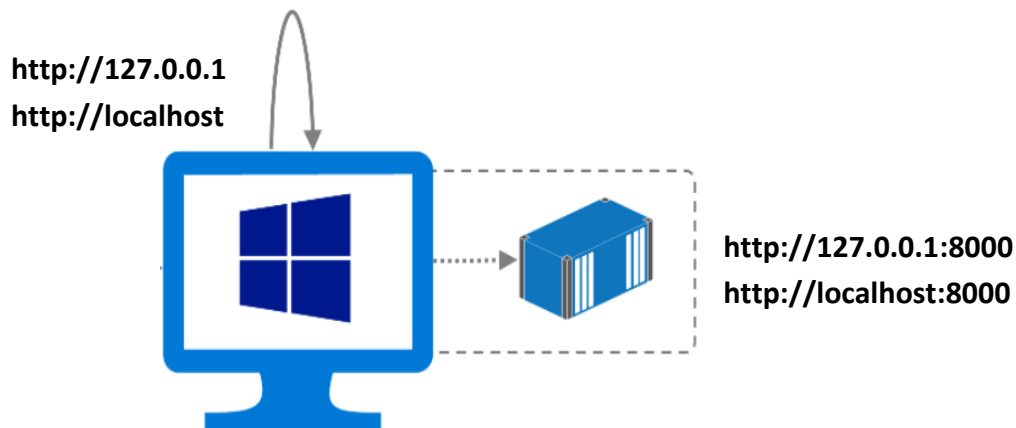
Ejemplo gen rico:

**docker container run -d --name nombre\_contenedor -p 8000:8080 nombre\_imagen:tag**

Donde:

- La palabra “**container**” es **opcional**.
- **-d** indica que el contenedor, en este caso, se va a ejecutar en segundo plano.
- **--name** indica su nombre: **nombre\_contenedor**.
- **-p 8000:8080** indica c mo se publican o mapean los puertos **entre el contenedor y el host** (m quina f sica anfitriona). En este caso, la solicitud de servicio entrar  por el puerto **8000 del host**, que la enviar  al puerto **8080 del contenedor**. Es decir, cuando queramos acceder por navegador al contenedor, lo haremos a trav s de “**localhost:8000**” o bien “**127.0.0.1:8000**” y el host lo reenviar  al puerto 8080 del contenedor.  
Pensad que, si no se indica el puerto, las peticiones HTTP van por el puerto por defecto y, en ese caso, responder  el host (m quina f sica anfitriona), Es decir: “**localhost**” o bien “**127.0.0.1**”
- **nombre\_imagen:tag** indica el nombre de la imagen en base a la cual se generar  el contenedor.

<b>Título</b>	4. Docker. Gestión de imágenes y contenedores
<b>Destinatario</b>	1º DAW – Semipresencial
<b>Autor</b>	Pascual Martínez
<b>Correo</b>	<a href="mailto:pmartinez@florida-uni.es">pmartinez@florida-uni.es</a>

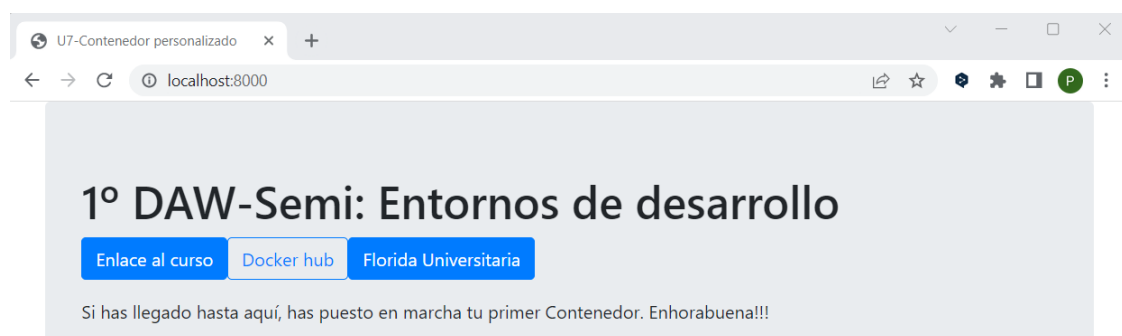


### Ejemplo:

**docker container run -d --name contenedor\_u7 -p 8000:8080 pmartinezflorida/imagen\_u7:v0**

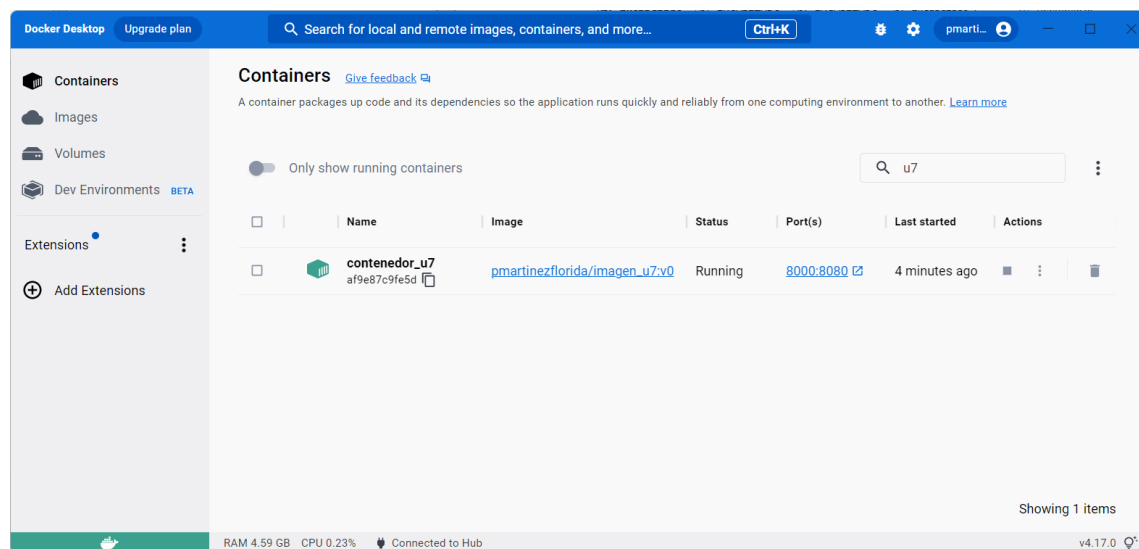
```
Windows PowerShell
PS C:\EDE-U7\aplicacion> docker container run -d --name contenedor_u7 -p 8000:8080 pmartinezflorida/imagen_u7:v0
af9e87c9fe5db937af15d0f3653f397c9bdf5d225b4bfc59ea93a3dc10511e22
PS C:\EDE-U7\aplicacion>
```

Una vez tengamos el contenedor en ejecución, podemos realizarle una petición, a través de navegador web mediante protocolo HTTP, dado que nuestro contenedor tiene en su interior un sistema operativo y un servidor web con una pequeña aplicación y responde a través del puerto 8000 del host ( <http://localhost:8000> o bien <http://127.0.0.1:8000> ).



<b>T�tulo</b>	4. Docker. Gesti�n de im�genes y contenedores
<b>Destinatario</b>	1� DAW – Semipresencial
<b>Autor</b>	Pascual Mart�nez
<b>Correo</b>	<a href="mailto:pmartinez@florida-uni.es">pmartinez@florida-uni.es</a>

Desde Docker Desktop, tambi n podemos acceder a las im genes y contenedores para realizar operaciones con ellos, as  como configurar ciertas opciones.



Por supuesto, tambi n disponemos de un comando para parar un contenedor o iniciar un contenedor existente:

**docker container stop/start [OPTIONS] nombre\_contenedor**

Tambi n podr amos realizar estas acciones desde Docker Desktop.

<b>T�tulo</b>	4. Docker. gesti�n de im�genes y contenedores
<b>Destinatario</b>	1� DAW – Semipresencial
<b>Autor</b>	Pascual Mart�nez
<b>Correo</b>	<a href="mailto:pmartinez@florida-uni.es">pmartinez@florida-uni.es</a>

Tal y como hemos comentado anteriormente, existe un mecanismo que permite conectar una carpeta de la m quina f sica con una del contenedor, denominado **volumen**.

Un volumen es un **montaje entre una ruta en el sistema de archivos del host anfitri n, y una ruta dentro del contenedor**.

Dicho de otro modo, es un mapeo entre una carpeta del anfitri n y otra del contenedor.

Se puede definir un volumen tanto a nivel de Dockerfile, como a nivel de ejecuci n de un contenedor, con una de las opciones de Docker container run:

**docker container run [OPTIONS] IMAGE [COMMAND] [ARG...]**  
**[Options]   →   -v /ruta/en/host:/ruta/en/contenedor**

#### Ejemplo:

```
docker container run -d --name contenedor_u7 -p 8000:8080 -v /ruta/host:/ruta/contenedor pmartinezflorida/imagen_u7:v0
```

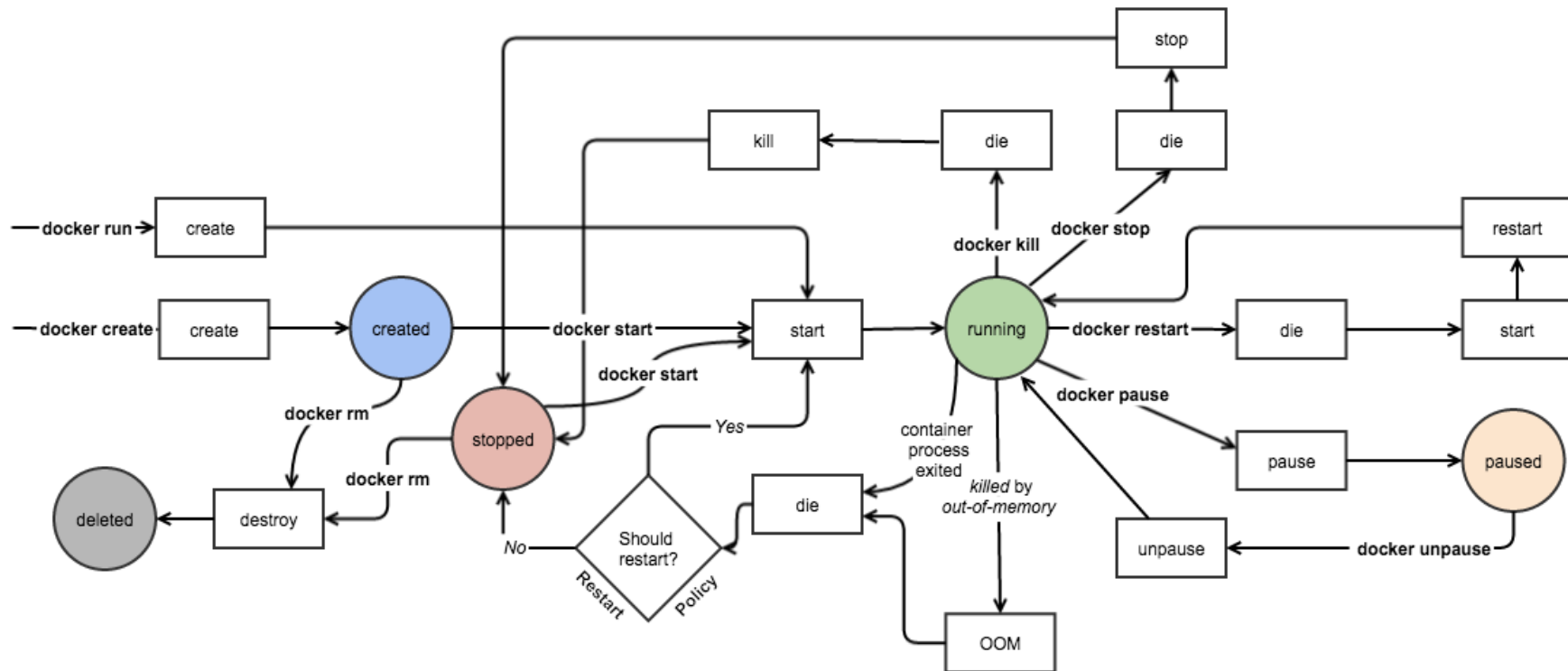
El uso de vol menes nos va a permitir:

- Que un contenedor ejecute una aplicaci n actualizada en tiempo real desde una m quina f sica.
- Guardar informaci n que deba ser persistente, de forma independiente a la vida del contenedor.
- Facilitar la comunicaci n entre contenedores.
- Realizar copias de seguridad.
- ...

Enlace a la documentaci n oficial: <https://docs.docker.com/storage/>

<b>T�tulo</b>	4. Docker. gesti�n de im�genes y contenedores
<b>Destinatario</b>	1� DAW – Semipresencial
<b>Autor</b>	Pascual Mart�nez
<b>Correo</b>	<a href="mailto:pmartinez@florida-uni.es">pmartinez@florida-uni.es</a>

## 5. Ciclo de vida de los contenedores

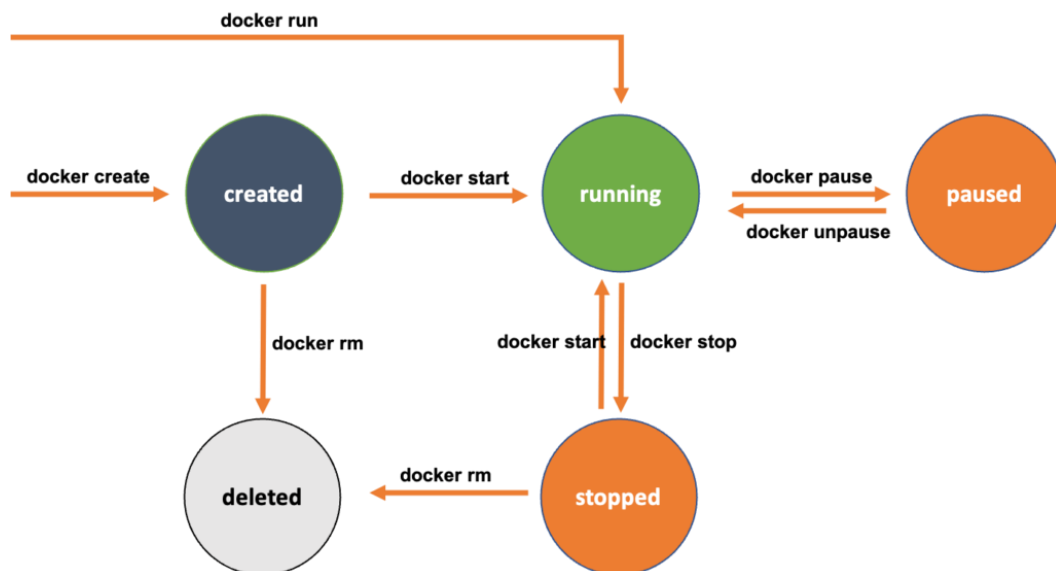


**\*\* Este esquema no es para memorizar, sino para usarlo como gu a de los estados y acciones posibles con un contenedor.**



<b>T�tulo</b>	4. Docker. gesti�n de im�genes y contenedores
<b>Destinatario</b>	1� DAW – Semipresencial
<b>Autor</b>	Pascual Mart�nez
<b>Correo</b>	<a href="mailto:pmartinez@florida-uni.es">pmartinez@florida-uni.es</a>

### Ciclo de vida simplificado:



## 6. Referencias.

- **Docker.**
  - <https://www.docker.com/>
  - <https://docs.docker.com/>
- **Pluralsight.**
  - Nigel Poulton. Docker Deep Dive.
  - Nigel Poulton. Docker and Kubernetes, the big picture.
  - Dan Wahlin. Docker for Web Developers.
- **NodeJS.**
  - <https://nodejs.org/>