

# ENTORNOS DE DESARROLLO

UD3.3 Gestión de imágenes en  
Docker

# Contenidos

---

- ❖ ¿Qué es Docker y cómo funciona?
- ❖ Introducción a los contenedores y a Docker
- ❖ Instalación de Docker
- ❖ Primeros pasos con Docker
- ❖ **Creación de imágenes personalizadas con Docker.**
- ❖ Redes y volúmenes en Docker.
- ❖ Docker compose.
- ❖ Utilidades para gestionar Docker fácilmente.

# Índice

---

- ❖ Listar imágenes local y para su descarga
- ❖ Descargando y eliminando imágenes (y contenedores) locales
  - `docker pull`
  - Eliminando imágenes con `docker rmi`
  - Eliminando contenedores con `docker rm`
  - Eliminar todo “`docker system prune -a`”
- ❖ Crear nuestras propias imágenes a partir de un contenedor

# Índice

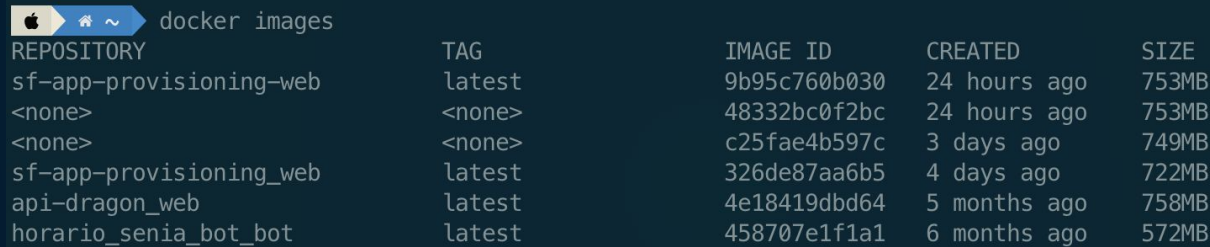
---

- ❖ **Subir nuestras imágenes a DockerHub**
  - **Crear repositorio para almacenar imagen**
  - **Subir imagen local a repositorio en DockerHub**
- ❖ **Generar imágenes a partir de Dockerfile**
  - **Plugins vscode**
  - **Dockerfile**
  - **Otros comandos de Dockerfile**
- ❖ **Mejora de imágenes**

# Listar imágenes locales y para su descarga

Podemos obtener información de qué imágenes tenemos almacenadas localmente usando

**docker images**



REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
sf-app-provisioning-web	latest	9b95c760b030	24 hours ago	753MB
<none>	<none>	48332bc0f2bc	24 hours ago	753MB
<none>	<none>	c25fae4b597c	3 days ago	749MB
sf-app-provisioning_web	latest	326de87aa6b5	4 days ago	722MB
api-dragon_web	latest	4e18419dbd64	5 months ago	758MB
horario_senia_bot_bot	latest	458707e1f1a1	6 months ago	572MB

Podemos utilizar filtros sencillos usando la nomenclatura

`"docker images [REPOSITORIO[:TAG]]"`. `docker images ubuntu:14.04`

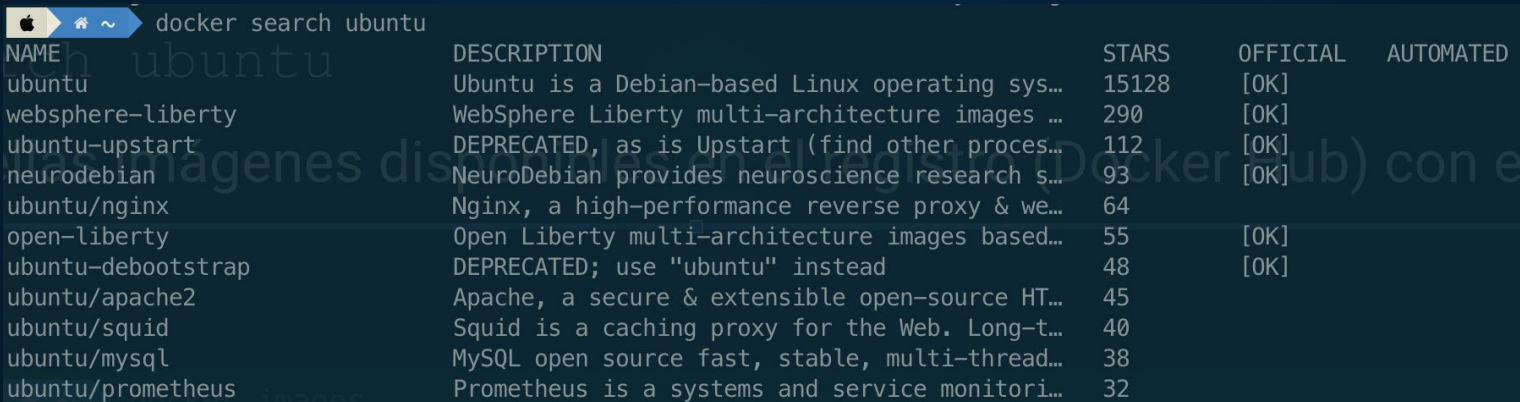
Nos mostrará la imagen del repositorio "ubuntu" en su versión "14.04".

# Listar imágenes locales y para su descarga

Podemos obtener información de imágenes que podemos descargar en el registro (por defecto, Docker Hub) utilizando el comando “docker search”. Por ejemplo con el siguiente comando:

```
docker search ubuntu
```

Nos aparecen aquellas imágenes disponibles en el registro (Docker Hub) con esa palabra.



A terminal window on a macOS system showing the command `docker search ubuntu` and its output. The output is a table listing Docker images related to 'ubuntu'.

NAME	DESCRIPTION	STARS	OFFICIAL	AUTOMATED
ubuntu	Ubuntu is a Debian-based Linux operating sys...	15128	[OK]	
websphere-liberty	WebSphere Liberty multi-architecture images ...	290	[OK]	
ubuntu-upstart	DEPRECATED, as is Upstart (find other proces...	112	[OK]	
neurodebian	NeuroDebian provides neuroscience research s...	93	[OK]	
ubuntu/nginx	Nginx, a high-performance reverse proxy & we...	64		
open-liberty	Open Liberty multi-architecture images based...	55	[OK]	
ubuntu-debootstrap	DEPRECATED; use "ubuntu" instead	48	[OK]	
ubuntu/apache2	Apache, a secure & extensible open-source HT...	45		
ubuntu/squid	Squid is a caching proxy for the Web. Long-t...	40		
ubuntu/mysql	MySQL open source fast, stable, multi-thread...	38		
ubuntu/prometheus	Prometheus is a systems and service monitori...	32		

# Descargando y eliminado imágenes (y contenedores) locales

---

Podemos almacenar imágenes localmente desde el registro sin necesidad de crear un contenedor mediante el comando “docker pull”

```
docker pull alpine:3.10
```

Este comando nos descarga la imagen “alpine” con el tag “3.10”, como vemos aquí:

```
🍏 ➤ 🏠 ~ ➤ docker pull alpine:3.10
3.10: Pulling from library/alpine
396c31837116: Pull complete
Digest: sha256:451eee8bedcb2f029756dc3e9d73bab0e7943c1ac55cff3a4861c52a0fdd3e98
Status: Downloaded newer image for alpine:3.10
docker.io/library/alpine:3.10
```

# Descargando y eliminado imágenes (y contenedores) locales

---

Podemos almacenar imágenes localmente desde el registro sin necesidad de crear un contenedor mediante el comando “docker pull”

```
docker pull alpine:3.10
```

Este comando nos descarga la imagen “alpine” con el tag “3.10”, como vemos aquí:

A terminal window with a dark background and light blue text. The prompt shows a terminal icon, an Apple logo, and a tilde symbol. The command 'docker pull alpine:3.10' has been entered. The output shows the image being pulled from the library, the pull is complete, the digest is sha256:451eee8bedcb2f029756dc3e9d73bab0e7943c1ac55cff3a4861c52a0fdd3e98, the status is 'Downloaded newer image for alpine:3.10', and the source is 'docker.io/library/alpine:3.10'.

```
docker pull alpine:3.10
3.10: Pulling from library/alpine
396c31837116: Pull complete
Digest: sha256:451eee8bedcb2f029756dc3e9d73bab0e7943c1ac55cff3a4861c52a0fdd3e98
Status: Downloaded newer image for alpine:3.10
docker.io/library/alpine:3.10
```

Podéis observar el historial de una imagen descargada, es decir, en qué versiones se basa, usando el comando “docker history”. Por ejemplo con:

```
docker history nginx
```



# Descargando y eliminado imágenes (y contenedores) locales

---

Con el comando “docker rmi” podemos eliminar imágenes almacenadas localmente.

```
docker rmi alpine:3.10
```

Elimina la imagen alpine con la etiqueta 3.10

```
❯ docker rmi alpine:3.10
Untagged: alpine:3.10
Untagged: alpine@sha256:451eee8bedcb2f029756dc3e9d73bab0e7943c1ac55cff3a4861c52a0fdd3e98
Deleted: sha256:e7b300aee9f9bf3433d32bc9305bfdd22183beb59d933b48d77ab56ba53a197a
Deleted: sha256:9fb3aa2f8b8023a4bebbf92aa567caf88e38e969ada9f0ac12643b2847391635
```

# Descargando y eliminado imágenes (y contenedores) locales

---

Para borrar contenedores parados (si un contenedor está en marcha, debe ser parado antes del borrado).

Con la siguiente orden se puede borrar un contenedor por identificador o nombre

```
docker rm IDENTIFICADOR/NOMBRE
```

Asimismo, una forma de borrar todos los contenedores (que estén parados), de forma similar a como vimos en el anterior punto, es la siguiente:

**Paso 1 (opcional):** paramos todos los contenedores:

```
docker stop $(docker ps -a -q)
```

**Paso 2:** borramos todos los contenedores:

```
docker rm $(docker ps -a -q)
```

# Descargando y eliminado imágenes (y contenedores) locales

---

Una forma de realizar las operaciones anteriores de golpe, es usando

`"docker system prune -a",`

que elimina toda imagen y contenedor parado.

**Paso 1 (opcional):** paramos todos los contenedores:

```
docker stop $(docker ps -a -q)
```

**Paso 2:** borramos todos los contenedores:

```
docker system prune -a
```

Ejecutamos los comandos de borrado y observamos la información que nos devuelve.

# Crear nuestras propias imágenes a partir de un contenedor

---

El sistema de imágenes de Docker funciona como un control de versiones por capas, de forma similar a la herramienta “git” para control de versiones.

Podemos entender que un contenedor es como una “capa temporal” de una imagen, por lo cual, podemos hacer un “commit” y convertir esa “capa temporal” en una imagen.

La sintaxis más habitual es la siguiente

```
docker commit -a "autor" -m "comentario" ID/NOMBRE-CONTENEDOR  
usuario/imagen:[version]
```

# Crear nuestras propias imágenes a partir de un contenedor

---

Por ejemplo, si tenemos un contenedor con nombre “ubuntumod” que simplemente es un contenedor basado en la imagen “ubuntu” en el que se ha instalado un programa y hacemos:

```
docker commit -a "Antonio" -m "Ubuntu modificado" IDCONTENEDOR  
buig/ubuntumod:2021
```

y tras ello, comprobamos las imágenes con

```
docker images
```

Hemos obtenido lo siguiente: una nueva imagen, con nombre “buig/ubuntumod” con tag “2021”, donde “buig” actúa como nombre de usuario para usarlo en un repositorio remoto (recordamos nuevamente, que por defecto es “Docker Hub”).

# Crear nuestras propias imágenes a partir de un contenedor

---

Ahora ya podríamos crear nuevos contenedores con esa imagen, usando por ejemplo:

```
docker run -it buig/ubuntu:2021
```

Si quisiéramos añadir una nueva etiqueta a la imagen, como “latest”, podemos usar el comando “docker tag”:

```
docker tag buig/ubuntu:2021 buig/ubuntu:latest
```

Ejecutamos de nuevo docker images, para ver el resultado del etiquetaje de las imágenes.

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
sergi/ubuntu	2021	c997cf91fb33	6 minutes ago	102MB
sergi/ubuntu	latest	c997cf91fb33	6 minutes ago	102MB
ubuntu	latest	4dd97cefde62	13 days ago	72.9MB

Para eliminar una etiqueta, simplemente deberemos borrar la imagen con “docker rmi”. La imagen se mantendrá mientras al menos tenga una etiqueta. Por ejemplo con:

```
docker rmi sergi/ubuntu:2021
```

# Subir nuestras imágenes a DockerHub

## 1. Crear repositorio

Iniciamos sesión en nuestra cuenta de DockerHub. Una vez logueados, debéis acceder a “Repositories” y ahí a “Create repository”



Podemos crearlo como **público** o **privado**. En caso de ser privado solo pueden usarlo su dueño y autorizados (otros usuarios de la plataforma con los que decidimos compartir la imagen).

Una vez creado, si tu usuario es “buig” y la imagen se llama “prueba”, podremos referenciarla en distintos contextos como “buig/prueba”.

# Subir nuestras imágenes a DockerHub

---

## 2. Subir imagen local al repositorio Docker Hub

En primer lugar, deberemos iniciar sesión mediante consola al repositorio mediante el comando

```
docker login
```

Una vez logueado, debemos hacer un “commit” local de la imagen, siguiendo la estructura vista en puntos anteriores. Un ejemplo podría ser:

```
docker commit -a "Antonio" -m "Ubuntu modificado" IDCONTENEDOR buig/prueba
```

Hecho este commit local, debemos subirlo usando “docker push”

```
docker push sergi/prueba
```

Una vez hecho eso, si la imagen es pública (o privada con permisos), cualquiera podrá descargarla y crear contenedores usando “docker pull” o “docker run”.



# Generar imágenes a partir de Dockerfile

---

## 1. Plugins VS Code

Docker nos permite generar de forma automática nuestras propias imágenes usando “docker build” y los llamados “Dockerfile”.

Para trabajar con los ficheros Dockerfile usaremos VS Code y dos de sus extensiones más comunes para la gestión de Docker.

- [Download Visual Studio Code - Mac, Linux, Windows](#)
- [Docker - Visual Studio Marketplace](#)
- [Visual Studio Code Dev Containers](#)

# Generar imágenes a partir de Dockerfile

---

## 2. Dockerfile

Empezaremos creando un sencillo “Dockerfile” donde crearemos una imagen de Ubuntu con el editor de texto “nano” instalado. Para ello indicaremos:

- De qué imagen base partiremos.
- Que comandos lanzaremos sobre la imagen base, para crear la nueva imagen
- Qué comando se asociará por defecto al lanzar un contenedor con la nueva imagen

Creamos el fichero “Dockerfile” (Visual Studio Code le pondrá un icono de la ballena) y añadimos:

```
FROM ubuntu:latest
```

```
RUN apt update && apt install -y nano
```

```
# Aquí un comentario
```

```
CMD /bin/bash
```

# Generar imágenes a partir de Dockerfile

---

## 2. Dockerfile

Ahora usamos el comando “docker build” de la siguiente forma:

```
docker build -t ubuntunano ./
```

Lo que hemos hecho es “ejecutar” lo que marca el “Dockerfile”. El resultado, se ha guardado en una nueva imagen local cuyo nombre hemos especificado con la opción “-t”. El lugar donde se encontraba el “Dockerfile” se ha indicado mediante “./” (directorio actual).

Podemos observar la historia de la imagen que hemos creado con “`docker history`”

Los comandos vistos (`FROM`, `RUN` y `CMD`) admiten distintos formatos. Para saber más podemos visitar su ayuda: <https://docs.docker.com/engine/reference/builder/>

# Generar imágenes a partir de Dockerfile

---

## 3. Otros comandos de Dockerfile

### EXPOSE

Diferencia entre exponer y publicar puertos en Docker:

- **Si no se expone ni publica** un puerto, este sólo es accesible desde el interior del contenedor.
- **Exponer** un puerto, indica que ese puerto es accesible tanto dentro del propio contenedor como por otros contenedores, pero no desde fuera (incluido el anfitrión).
- **Publicar** un puerto, el puerto es accesible desde fuera del contenedor, debe asociarse a un puerto del anfitrión.

La opción **EXPOSE** indica los puertos por defecto expuestos que tendrá un contenedor basado en esta imagen. Es similar a la opción “**--expose**” de “**docker run**” (y de paso, recordamos que “**docker run**” con “**-p**” los publica). Por ejemplo, para exponer 80 443 y 8080 indicaremos:

```
EXPOSE 80 443 8080
```

# Generar imágenes a partir de Dockerfile

---

## 3. Otros comandos de Dockerfile

### ADD/COPY

ADD y COPY son comandos para copiar ficheros de la máquina anfitrión al nuevo contenedor. Se recomienda usar COPY, excepto que queramos descomprimir un “zip”, que ADD permite su descompresión.

#### Uso de ADD:

```
ADD ./mifichero.zip /var/www/html
```

Descomprimirá el contenido de “mifichero.zip” en el directorio destino de la nueva imagen.

#### Uso de COPY:

```
COPY ./mifichero.zip /var/www/html
```

# Generar imágenes a partir de Dockerfile

---

## 3. Otros comandos de Dockerfile

### ENTRYPOINT

Por defecto, los contenedores Docker están configurados para que ejecuten los comandos que se lancen mediante `"/bin/sh -c"`. Dicho de otra forma, los comandos que lanzábamos, eran parámetros para `"/bin/sh -c"`. Podemos cambiar qué comando se usa para esto con `ENTRYPOINT`.

Por ejemplo:

```
ENTRYPOINT ["cat"]
```

```
CMD ["/etc/passwd"]
```

Indicaremos que los comandos sean lanzados con `"cat"`. Al lanzar el comando `"/etc/passwd"`, realmente lo que haremos es que se lanzará `"cat /etc/passwd"`.

# Generar imágenes a partir de Dockerfile

---

## 3. Otros comandos de Dockerfile

### WORKDIR

Cada vez que expresamos el comando WORKDIR, estamos cambiando el directorio de la imagen donde ejecutamos los comandos. Si este directorio no existe, se crea. Por ejemplo:

```
WORKDIR /root
```

```
CMD mkdir tmp
```

```
WORKDIR /var/www/html
```

```
CMD mkdir tmp
```

Crearé la carpeta “tmp” tanto en “/root” como en “/var/www/html”. Si los directorios “/root” o “/var/www/html” no hubieran existido, los habría creado.

# Generar imágenes a partir de Dockerfile

---

## 3. Otros comandos de Dockerfile

### ENV

El comando ENV nos permite definir variables de entorno por defecto en la imagen.

```
ENV v1="valor1" v2="valor2"
```



# Generar imágenes a partir de Dockerfile

---

## 3. Otros comandos de Dockerfile

### ARG, VOLUME, LABEL, HEALTHCHECK

- **ARG**: permite enviar parámetros al propio “Dockerfile” con la opción “`--build-arg`” del comando “`docker build`”.
- **VOLUME**: permite establecer volúmenes por defecto en la imagen. Hablaremos de los volúmenes más adelante en el curso.
- **LABEL**: permite establecer metadatos dentro de la imagen mediante etiquetas. Uno de los casos más típicos, sustituyendo al comando MAINTAINER, que esta “deprecated” es:
  - **LABEL** maintainer="sergi.profesor@gmail.com"
- **HEALTHCHECK**: permite definir cómo se comprobará si ese contenedor está funcionando correctamente o no. Útil para sistemas orquestadores como “Docker swarm”, aunque otros como “Kubernetes” incorporan su propio sistema

# Mejora y optimización de imágenes

---

Al crear imágenes, es habitual aumentar el tamaño de las imágenes base. Algunos consejos para dentro de lo posible, aumentar el tamaño lo menos posible:

- Usar imágenes **base ligeras**, tipo “**Alpine**” y no instalar **programas innecesarios** (Vim, Nano).
- **Minimizar capas** en “**Dockerfile**”.
- **Mejor usar** “`RUN apt-get install -y <packageA> <packageB>`” que usar “`RUN apt-get install -y <packageA>`” y tras ello “`RUN apt-get install -y <packageB>`”.
- Limpieza tras instalaciones con “apt”, tales como:
  - “`rm -rf /var/lib/apt/lists/*`”.
  - “`apt-get purge --auto-remove && apt-get clean`”.
- Al usar “`apt install`” usar la opción “`no-install-recommend`”.
- Analiza tus “**Dockerfile**” con [FromLatest.io](https://fromlatest.io) y sigue sus consejos.