Introducción a Docker

UD 04. Gestión de imágenes en Docker

Autor: Sergi García Barea

Actualizado Marzo 2023

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**

UD04. GESTIÓN DE IMÁGENES EN DOCKER

1. INTRODUCCIÓN

Hasta ahora, hemos visto cómo descargar y trabajar con imágenes de terceros en Docker. En esta unidad explicaremos cómo gestionar las imágenes de contenedores Docker (listado, eliminación, historia, etc.) así como su creación tanto de forma manual como utilizando el comando "***docker build***" con los llamados "***Dockerfiles***".

2. LISTANDO IMÁGENES LOCALES Y PARA SU DESCARGA

2.1 Listando imágenes locales

Podemos obtener información de qué imágenes tenemos almacenadas localmente usando

```
docker images
```

Obteniendo un resultado similar al siguiente, donde vemos información acerca de las imágenes

```
sergi@ubuntu:~$ docker images
REPOSITORY     TAG       IMAGE ID       CREATED         SIZE
hello-world    latest    d1165f221234   8 days ago      13.3kB
ubuntu         latest    4dd97cefde62   10 days ago     72.9MB
busybox        latest    491198851f0c   3 weeks ago     1.23MB
nginx          latest    35c43ace9216   3 weeks ago     133MB
theasp/novnc   latest    58d7a8345d26   2 months ago    531MB
ubuntu         14.04     df043b4f0cf1   5 months ago    197MB
sergi@ubuntu:~$
```

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".

```
sergi@ubuntu:~$ docker images ubuntu:14.04
REPOSITORY   TAG       IMAGE ID       CREATED         SIZE
ubuntu       14.04     df043b4f0cf1   5 months ago    197MB
```

Si queremos utilizar algún filtro avanzado, podemos usar la opción "-f". Aquí un ejemplo, filtrando las imágenes que empiecen por "u" y acabe su etiqueta en "04".

```
docker images -f=reference="u*:*04"
```

```
sergi@ubuntu:~$ docker images -f=reference="u*:*04"
REPOSITORY   TAG       IMAGE ID       CREATED         SIZE
ubuntu       14.04     df043b4f0cf1   5 months ago    197MB
```

> **Atención:** no confundir este comando con ***"docker image"*** (sin la s final).

Más información en https://docs.docker.com/engine/reference/commandline/images/

2.2 Listando imágenes 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.

3. DESCARGANDO Y ELIMINANDO IMÁGENES (Y CONTENEDORES) LOCALES

3.1 Descargando imágenes con "docker pull"

Podemos almacenar imágenes localmente desde el registro sin necesidad de crear un contenedor mediante el comando ***"docker pull"***, claramente inspirado en sistemas de control de versiones como ***"git"***. Para conocer sus nombres y versiones, podemos usar el comando "docker search" explicado anteriormente o visitar https://hub.docker.com/.

```
docker pull alpine:3.10
```

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

```
sergi@ubuntu:~$ docker pull alpine:3.10
3.10: Pulling from library/alpine
8464c5956bbe: Pull complete
Digest: sha256:0b4d282d7ae7cf5ed91801654a918aea45d6c1de6df0db6a29d60619141fb8de
Status: Downloaded newer image for alpine:3.10
docker.io/library/alpine:3.10
sergi@ubuntu:~$
```

3.2 Observar el historial de una imagen descargada

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
```

Obtenemos lo siguiente:

```
sergi@ubuntu:~$ docker history nginx
IMAGE          CREATED       CREATED BY                                      SIZE      COMMENT
35c43ace9216   3 weeks ago   /bin/sh -c #(nop)  CMD ["nginx" "-g" "daemon…   0B
<missing>      3 weeks ago   /bin/sh -c #(nop)  STOPSIGNAL SIGQUIT           0B
<missing>      3 weeks ago   /bin/sh -c #(nop)  EXPOSE 80                    0B
<missing>      3 weeks ago   /bin/sh -c #(nop)  ENTRYPOINT ["/docker-entr…   0B
<missing>      3 weeks ago   /bin/sh -c #(nop) COPY file:c7f3907578be6851…   4.62kB
<missing>      3 weeks ago   /bin/sh -c #(nop) COPY file:0fd5fca330dcd6a7…   1.04kB
<missing>      3 weeks ago   /bin/sh -c #(nop) COPY file:0b866ff3fc1ef5b0…   1.96kB
<missing>      3 weeks ago   /bin/sh -c #(nop) COPY file:65504f71f5855ca0…   1.2kB
<missing>      3 weeks ago   /bin/sh -c set -x     && addgroup --system -…   63.8MB
<missing>      3 weeks ago   /bin/sh -c #(nop)  ENV PKG_RELEASE=1~buster      0B
<missing>      3 weeks ago   /bin/sh -c #(nop)  ENV NJS_VERSION=0.5.1         0B
<missing>      3 weeks ago   /bin/sh -c #(nop)  ENV NGINX_VERSION=1.19.7      0B
<missing>      4 weeks ago   /bin/sh -c #(nop)  LABEL maintainer=NGINX Do…   0B
<missing>      4 weeks ago   /bin/sh -c #(nop)  CMD ["bash"]                  0B
<missing>      4 weeks ago   /bin/sh -c #(nop) ADD file:d5c41bfaf15180481…   69.2MB
```

3.3 Eliminando imágenes con "docker rmi"

Con el comando ***"docker rmi"*** podemos eliminar imágenes almacenadas localmente.

```
docker rmi ubuntu:14.04
```

Elimina la imagen ubuntu con la etiqueta 14.04

```
sergi@ubuntu:~$ docker rmi ubuntu:14.04
Untagged: ubuntu:14.04
Untagged: ubuntu@sha256:63fce984528cec8714c365919882f8fb64c8a3edf23fdfa0b218a2756125456f
Deleted: sha256:df043b4f0cf196749a9a426080f433b76cabf6b37dde2edefef317ba54c713c7
Deleted: sha256:d67d0461b8453209ccf455d0e50e1e096e1622b95448392f0381682ebcdd60ea
Deleted: sha256:873ef23ebe5a4cba2880a40f95d5b8c823524ae1192f067d86c5611dcc9ea154
Deleted: sha256:f2fa9f4cf8fd0a521d40e34492b522cee3f35004047e617c75fadeb8bfd1e6b7
sergi@ubuntu:~$
```

Una forma de eliminar **todas** las imágenes locales, que no estén siendo usadas por un contenedor,

combinando ***"docker images -q"*** para obtener la lista y ***"docker rmi"*** es la siguiente:

```
docker rmi $(docker images -q)
```

Aquí se observa el borrado, excepto de aquellas usadas por un contenedor:

```
sergi@ubuntu:~$ docker rmi $(docker images -q)
Untagged: alpine:3.10
Untagged: alpine@sha256:0b4d282d7ae7cf5ed91801654a918aea45d6c1de6df0db6a29d60619141fb8de
Deleted: sha256:5641e297651005e256c7e27f8363dcacc560fabdc635f1254cd5a41354485dfd
Deleted: sha256:483b65c07faaf8ee7f1f57c6d7de0eda9dfd61a34f03337926650b8178db5286
Untagged: theasp/novnc:latest
Untagged: theasp/novnc@sha256:cd5210a86611bc2dc3ea6eb96a2bfe91237983f8fbc1ab02175142e63e461c40
Deleted: sha256:58d7a8345d269148b19d58a6a7e89ef7b469b2d16a91ce6a0033f5d9c3d9ab66
Deleted: sha256:1a190cda9e95cf2cbd2be33aefca0896da455c0b9d0d0ffd9779305127697db3
Deleted: sha256:06a568747e469240b65bdb0610ef420f9ef912d0e7f00d8ba48c7e63cda3ea7f
Deleted: sha256:f0e10b20de190c7cf4ea7ef410e7229d64facdc5d94514a13aa9b58d36fca647
Error response from daemon: conflict: unable to delete d1165f221234 (must be forced) - image is being used by stopped container e20ffb22edb5
Error response from daemon: conflict: unable to delete 4dd97cefde62 (must be forced) - image is being used by stopped container 904f3a9fd3ff
Error response from daemon: conflict: unable to delete 491198851f0c (must be forced) - image is being used by stopped container 683c18d67f24
Error response from daemon: conflict: unable to delete 35c43ace9216 (must be forced) - image is being used by stopped container 6c1a1916ab0a
sergi@ubuntu:~$
```

3.4 Eliminando contenedores con "docker rm"

Aprovechando que tratamos el borrado de imágenes, comentamos cómo 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)
```

```
sergi@ubuntu:~$ docker stop $(docker ps -a -q)
a1b6d1b3f8a5
ce37400c9d08
91a7e729c8be
6c1a1916ab0a
afd5b07cd223
683c18d67f24
e20ffb22edb5
cfe2fd201282
36c74bb5c568
872aa285fb22
80feb0808621
904f3a9fd3ff
sergi@ubuntu:~$
sergi@ubuntu:~$
sergi@ubuntu:~$ docker rm $(docker ps -a -q)
a1b6d1b3f8a5
ce37400c9d08
91a7e729c8be
6c1a1916ab0a
afd5b07cd223
683c18d67f24
e20ffb22edb5
cfe2fd201282
36c74bb5c568
872aa285fb22
80feb0808621
904f3a9fd3ff
```

3.5 Eliminando todas las imágenes y contenedores con "docker system prune -a"

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
```

Obteniendo algo similar a esto:

```
sergi@ubuntu:~$ docker system prune -a
WARNING! This will remove:
  - all stopped containers
  - all networks not used by at least one container
  - all images without at least one container associated to them
  - all build cache

Are you sure you want to continue? [y/N] y
Deleted Images:
untagged: hello-world:latest
untagged: hello-world@sha256:89b647c604b2a436fc3aa56ab1ec515c26b085ac0c15b0d105bc475be15738fb
deleted: sha256:d1165f2212346b2bab48cb01c1e39ee8ad1be46b87873d9ca7a4e434980a7726
deleted: sha256:f22b99068db93900abe17f7f5e09ec775c2826ecfe9db961fea68293744144bd
untagged: nginx:latest
untagged: nginx@sha256:f3693fe50d5b1df1ecd315d54813a77afd56b0245a404055a946574deb6b34fc
deleted: sha256:35c43ace9216212c0f0e546a65eec93fa9fc8e96b25880ee222b7ed2ca1d2151
deleted: sha256:61f2666cb67e4572a31412367fa44567e6ac238226385762ea65670ed39034a8
deleted: sha256:622fb7fb6a35078e3a2d446bb0e74c6a0cd500e3a211fd17ecbbcea5377ded38
deleted: sha256:69a8591f1aaa7d694fa79a187886f6690e6e51e8c2bc91727be01a9e87daacd2
deleted: sha256:8a451c701633832102e10093db7545eada8e5639a1b35bb14afaf48601948802
deleted: sha256:2edbde38832e9e0e07d113df74817dc736fd49ea2f9c0d7ce8e40e3446b49b82
deleted: sha256:9eb82f04c782ef3f5ca25911e60d75e441ce0fe82e49f0dbf02c81a3161d1300
untagged: busybox:latest
untagged: busybox@sha256:c6b45a95f932202dbb27c31333c4789f45184a744060f6e569cc9d2bf1b9ad6f
deleted: sha256:491198851f0ccdd0882cb9323f3856043d4e4c65b773e8eac3e0f6bc979a2ae7
deleted: sha256:84009204da3f70b09d2be3914e12844ae9db893aa85ef95df83604f95df05187
untagged: ubuntu:latest
untagged: ubuntu@sha256:b4f9e18267eb98998f6130342baacaeb9553f136142d40959a1b46d6401f0f2b
deleted: sha256:4dd97cefde62cf2d6bcfd8f2c0300a24fbcddbe0ebcd577cc8b420c29106869a
deleted: sha256:95bc1f83306cc7ebaa959492929d6624b0cc1bb6ba61be1cd04fed7d39b002fc
deleted: sha256:a0fcf305193749a4fe8c9da074c4781a0f1e63f2c5b5a979a88597ada5c74645
deleted: sha256:aeb3f02e937406fb402a926ce5cebc7da79b14dbcb4f85a5ce0e3855623cec80

Total reclaimed space: 207.2MB
sergi@ubuntu:~$
```

4. CREANDO NUESTRAS PROPIAS IMÁGENES A PARTIR DE UN CONTENEDOR EXISTENTE

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]
```

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 "Sergi" -m "Ubuntu modificado" IDCONTENEDOR
sergi/ubuntumod:2021
```

y tras ello, comprobamos las imágenes con

```
docker images
```

observamos lo siguiente:

```
sergi@ubuntu:~$ docker commit -a "Sergi" -m "Ubuntu modificado" ubuntumod sergi/ubuntumod:2021
sha256:c997cf91fb33b5c5a769bc8f4a2d4dbe4f9d6bb7fbf82e7b1435c8b77f2d828d
sergi@ubuntu:~$ docker images
REPOSITORY        TAG       IMAGE ID       CREATED         SIZE
sergi/ubuntumod   2021      c997cf91fb33   3 seconds ago   102MB
ubuntu            latest    4dd97cefde62   13 days ago     72.9MB
sergi@ubuntu:~$
```

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

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

```
docker run -it sergi/ubuntumod:2021
```

Si quisiéramos añadir una nueva etiqueta a la imagen, como "latest", podemos usar el comando "***docker tag***", teniendo en cuenta que una misma imagen puede tener varias etiquetas:

```
docker tag sergi/ubuntumod:2021 sergi/ubuntumod:latest
```

Obtendremos algo similar a:

```
sergi@ubuntu:~$ docker tag sergi/ubuntumod:2021 sergi/ubuntumod:latest
sergi@ubuntu:~$
sergi@ubuntu:~$ docker images
REPOSITORY        TAG       IMAGE ID       CREATED         SIZE
sergi/ubuntumod   2021      c997cf91fb33   6 minutes ago   102MB
sergi/ubuntumod   latest    c997cf91fb33   6 minutes ago   102MB
ubuntu            latest    4dd97cefde62   13 days ago     72.9MB
sergi@ubuntu:~$
```

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/ubuntumod:2021
```

quedaría así:

```
sergi@ubuntu:~$ docker rmi sergi/ubuntumod:2021
Untagged: sergi/ubuntumod:2021
sergi@ubuntu:~$ docker images
REPOSITORY        TAG       IMAGE ID       CREATED          SIZE
sergi/ubuntumod   latest    c997cf91fb33   10 minutes ago   102MB
ubuntu            latest    4dd97cefde62   13 days ago      72.9MB
sergi@ubuntu:~$
```

Más información de los comandos en:

- Docker commit https://docs.docker.com/engine/reference/commandline/commit/
- Docker tag https://docs.docker.com/engine/reference/commandline/tag/

5. EXPORTANDO/IMPORTANDO IMÁGENES LOCALES A/DESDE FICHEROS

Una vez tengamos una imagen local en nuestro sistema, podemos hacer una copia de la misma, ya sea como copia de seguridad o como forma de transportarla a otros sistemas mediante el comando "***docker save"***. Por ejemplo, se puede hacer de estas dos formas:

```
docker save -o copiaSeguridad.tar sergi/ubuntumod
```

o de forma alternativa

```
docker save sergi/ubuntumod > copiaSeguridad.tar
```

Si queremos importar el fichero para crear una imagen en nuestra máquina, podemos usar "***docker load***". Por ejemplo, se puede hacer de estas dos formas:

```
docker load -i copiaSeguridad.tar
```

o de forma alternativa

```
docker load < copiaSeguridad.tar
```

Más información sobre los comandos:

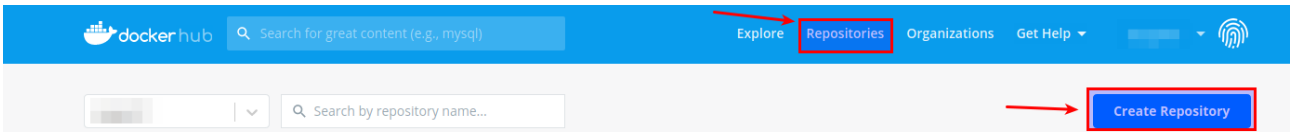
- Docker save: https://docs.docker.com/engine/reference/commandline/save/
- Docker load: https://docs.docker.com/engine/reference/commandline/load/

6. SUBIENDO NUESTRAS PROPIAS IMÁGENES A UN REPOSITORIO (DOCKER HUB)

Podemos subir una imagen a un repositorio (por defecto Docker Hub). Para ello, debemos realizar los siguientes pasos:

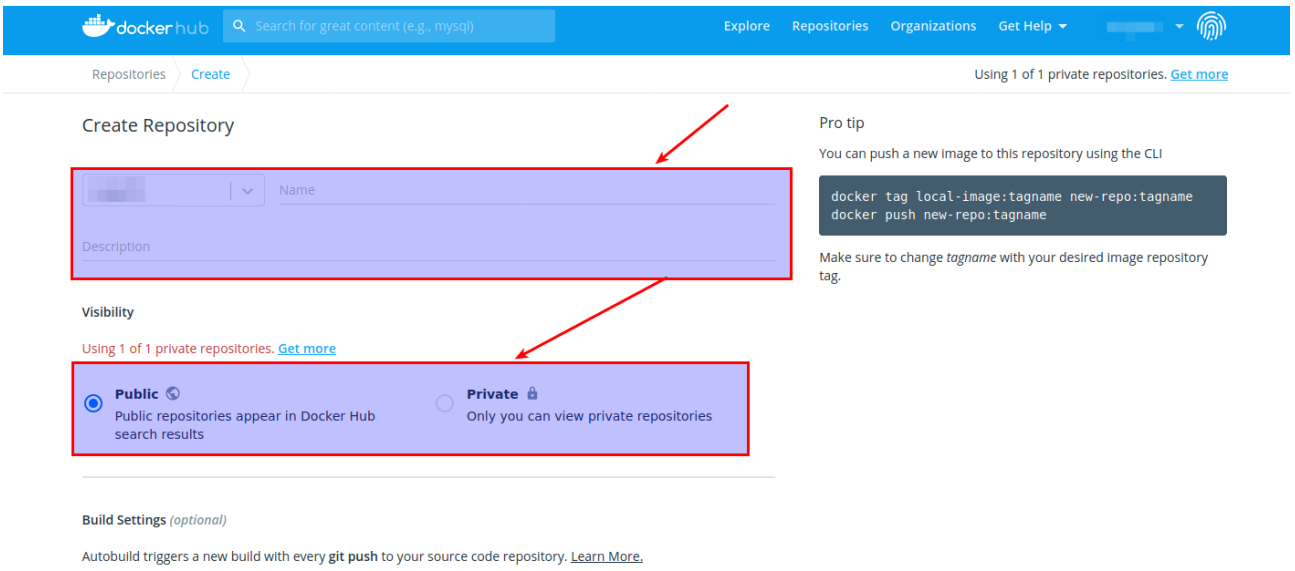
6.1 Paso 1: creando repositorio para almacenar la imagen en Docker Hub

En primer lugar, debéis crearos una cuenta en https://hub.docker.com e iniciar sesión. Una vez iniciada sesión, debéis acceder a "***Repositories***" y ahí a "***Create repository***" de forma similar a como se ve en la imagen siguiente:



Tras ello, podréis quedar un repositorio con vuestra cuenta y elegir si dicho repositorio es público (cualquiera puede acceder) o privado (solo puede acceder dueño o autorizados).

La pantalla de creación del repositorio tiene un aspecto similar a este:



Una vez creado, si tu usuario es "***sergi***" y la imagen se llama "***prueba"***, podremos referenciarla en distintos contextos como "***sergi/prueba***"

6.2 Paso 2: almacenando imagen local en repositorio Docker Hub

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

```
docker login
```

Una vez iniciada sesión, debemos hacer un "commit" local de la imagen, siguiendo la estructura vista en puntos anteriores. Un ejemplo podría ser:

```
docker commit -a "Sergi" -m "Ubuntu modificado" IDCONTENEDOR
sergi/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"***.

Más información de los comandos:

- Docker login https://docs.docker.com/engine/reference/commandline/login/
- Docker push https://docs.docker.com/engine/reference/commandline/push/

7. GENERAR AUTOMÁTICAMENTE NUESTRAS PROPIAS IMÁGENES MEDIANTE DOCKERFILE

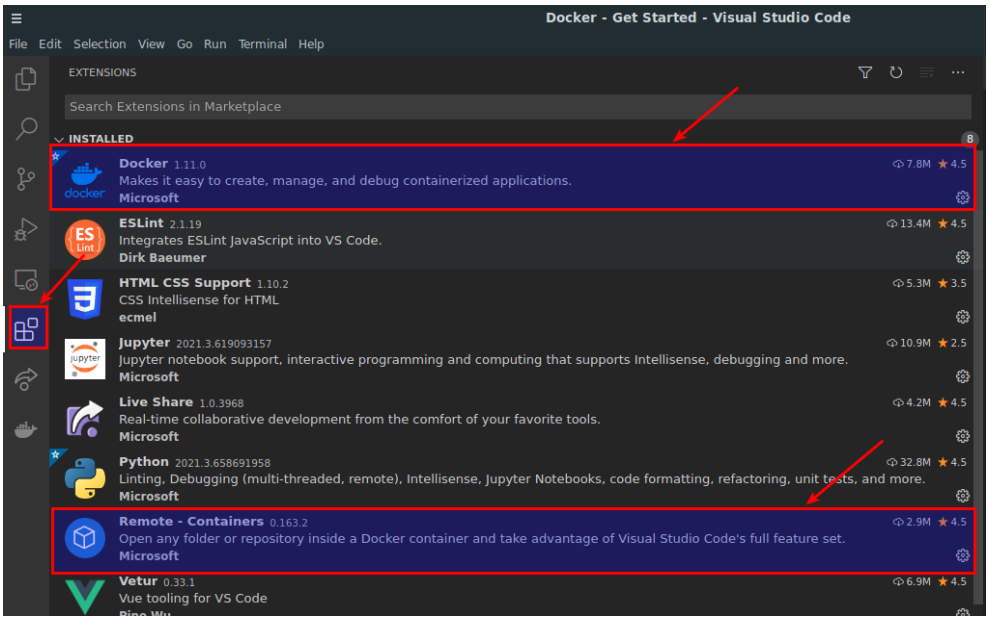
Docker nos permite generar de forma automática nuestras propias imágenes usando "***docker build***" y los llamados "***Dockerfile"***.

7.1 Editor Visual Studio Code y plugins asociados a Docker

Los ficheros "***Dockerfile***" pueden crearse con cualquier editor de texto, pero desde aquí recomendamos el editor multiplataforma "***Visual Studio Code***" https://code.visualstudio.com/

Para saber más sobre cómo usar este editor podéis usar https://code.visualstudio.com/learn

Al instalarlo, si detecta Docker instalado en el sistema, el propio editor nos sugerirá una serie de plugins. Merece la pena instalarlos. Si no, siempre podéis buscar en plugins manualmente. Yo personalmente, os recomiendo estos dos que podéis ver en la imagen:



7.2 Creando nuestro primer Dockerfile

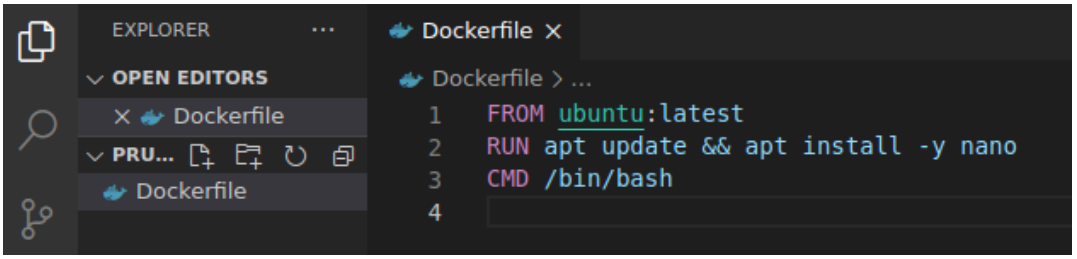
Empezaremos creando un sencillo ***"Dockerfile"*** donde crearemos una imagen de Ubuntu con el editor de texto "nano" instalado. Para ello indicaremos:

- De qué imagen base partiremos.
- Qué 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
```

En el editor quedará de una forma similar a:



Si ahora usamos el comando "*docker build*" de la siguiente forma:

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

Obtendremos algo similar a

```
sergi@ubuntu:~/Desktop/pruebaDockerFile$ docker build -t ubuntunano ./
Sending build context to Docker daemon  2.048kB
Step 1/3 : FROM ubuntu:latest
 ---> 4dd97cefde62
Step 2/3 : RUN apt update && apt install -y nano
 ---> Running in 2d6bf70874ae

WARNING: apt does not have a stable CLI interface. Use with caution in scripts.

Get:1 http://security.ubuntu.com/ubuntu focal-security InRelease [109 kB]
Get:2 http://archive.ubuntu.com/ubuntu focal InRelease [265 kB]
Get:3 http://archive.ubuntu.com/ubuntu focal-updates InRelease [114 kB]
Get:4 http://archive.ubuntu.com/ubuntu focal-backports InRelease [101 kB]
Get:5 http://security.ubuntu.com/ubuntu focal-security/main amd64 Packages [683 kB]
Get:6 http://archive.ubuntu.com/ubuntu focal/restricted amd64 Packages [33.4 kB]
Get:7 http://archive.ubuntu.com/ubuntu focal/universe amd64 Packages [11.3 MB]
Get:8 http://security.ubuntu.com/ubuntu focal-security/restricted amd64 Packages [187 kB]
Get:9 http://security.ubuntu.com/ubuntu focal-security/universe amd64 Packages [681 kB]
Get:10 http://security.ubuntu.com/ubuntu focal-security/multiverse amd64 Packages [21.6 kB]
Get:11 http://archive.ubuntu.com/ubuntu focal/multiverse amd64 Packages [177 kB]
Get:12 http://archive.ubuntu.com/ubuntu focal/main amd64 Packages [1275 kB]
Get:13 http://archive.ubuntu.com/ubuntu focal-updates/main amd64 Packages [1089 kB]
Get:14 http://archive.ubuntu.com/ubuntu focal-updates/universe amd64 Packages [938 kB]
Get:15 http://archive.ubuntu.com/ubuntu focal-updates/multiverse amd64 Packages [29.6 kB]
Get:16 http://archive.ubuntu.com/ubuntu focal-updates/restricted amd64 Packages [220 kB]
Get:17 http://archive.ubuntu.com/ubuntu focal-backports/universe amd64 Packages [4305 B]
Fetched 17.3 MB in 2s (8866 kB/s)
Reading package lists...
Building dependency tree...
Reading state information...
1 package can be upgraded. Run 'apt list --upgradable' to see it.

WARNING: apt does not have a stable CLI interface. Use with caution in scripts.
```

Lo que hemos hecho es "ejecutar" lo que marca él "***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).

> ! **Atención:** el fichero debe llamarse exactamente "***Dockerfile***", respetando mayúsculas y minúsculas.

Si queréis especificar un nombre de fichero distinto a buscar en el directorio, puede usarse la opción "-f", como en este ejemplo:

```
docker build -t ubuntunano -f Dockerfile2 ./
```

Podemos observar la historia de la imágen que hemos creado con "***docker history***"

```
sergi@ubuntu:~/Desktop/pruebaDockerFile$ docker history ubuntunano
IMAGE          CREATED         CREATED BY                                      SIZE      COMMENT
25bdaae2cdf4   7 minutes ago   /bin/sh -c #(nop)  CMD ["/bin/sh" "-c" "/bin…   0B
602193ec2673   7 minutes ago   /bin/sh -c apt update && apt install -y nano    28.6MB
4dd97cefde62   13 days ago     /bin/sh -c #(nop)  CMD ["/bin/bash"]            0B
<missing>      13 days ago     /bin/sh -c mkdir -p /run/systemd && echo 'do…   7B
<missing>      13 days ago     /bin/sh -c [ -z "$(apt-get indextargets)" ]     0B
<missing>      13 days ago     /bin/sh -c set -xe   && echo '#!/bin/sh' > /…   811B
<missing>      13 days ago     /bin/sh -c #(nop) ADD file:c77338d21e6d1587d…   72.9MB
sergi@ubuntu:~/Desktop/pruebaDockerFile$
```

Donde observamos, que nuestra imagen ha crecido en "28.6MB" al hacer "***apt update && apt install -y nano***". Aunque en el anterior "Dockerfile" hemos usado un solo RUN, podríamos haber utilizado varios RUN en lugar de uno, escrito de una forma secuencial, como vemos en:

```
FROM ubuntu:latest
RUN apt update
RUN apt install -y nano
CMD /bin/bash
```

Aquí, simplemente, habría más capas intermedias, como se observa en "docker history" si generamos la imagen con la secuencia anterior

```
sergi@ubuntu:~/Desktop/pruebaDockerFile$ docker history ubuntunano
IMAGE          CREATED          CREATED BY                                      SIZE      COMMENT
95dee6b6df30   3 seconds ago    /bin/sh -c #(nop)  CMD ["/bin/sh" "-c" "/bin…   0B
b4cc983223d9   4 seconds ago    /bin/sh -c apt install -y nano                  1.23MB
c3d9ab68cdbc   10 seconds ago   /bin/sh -c apt update                           27.4MB
4dd97cefde62   13 days ago      /bin/sh -c #(nop)  CMD ["/bin/bash"]            0B
<missing>      13 days ago      /bin/sh -c mkdir -p /run/systemd && echo 'do…   7B
<missing>      13 days ago      /bin/sh -c [ -z "$(apt-get indextargets)" ]     0B
<missing>      13 days ago      /bin/sh -c set -xe   && echo '#!/bin/sh' > /…   811B
<missing>      13 days ago      /bin/sh -c #(nop) ADD file:c77338d21e6d1587d…   72.9MB
sergi@ubuntu:~/Desktop/pruebaDockerFile$
```

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

7.3 Otros comandos importantes de Dockerfile

Al crear un "Dockerfile" hay multitud de comandos. A continuación explicamos los comandos más importantes a utilizar:

7.3.1 Comando EXPOSE

En primer lugar, **repasamos la diferencia entre exponer y publicar puertos en Docker**:

- Si no se expone ni publica un puerto, este solo 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, indica que el puerto es accesible desde fuera del contenedor, por lo cual debe asociarse a un puerto del anfitrión.

La opción EXPOSE nos permite indicar 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 los puertos 80, 443 y 8080 indicaremos:

```
EXPOSE 80 443 8080
```

7.3.2 Comando ADD/COPY

ADD y **COPY** son comandos para copiar ficheros de la máquina anfitriona al nuevo contenedor. Se recomienda usar **COPY**, excepto que queramos descomprimir un "zip", que **ADD** permite su descompresión. Más información sobre la diferencia entre **ADD** y **COPY:**

https://nickjanetakis.com/blog/docker-tip-2-the-difference-between-copy-and-add-in-a-dockerile

Ejemplo de uso de **ADD**:

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

Descomprimirá el contenido de "***mifichero.zip***" en el directorio destino de la nueva imagen.

Ejemplo de uso de **COPY**:

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

O incluso accediendo desde la web.

```
COPY http://miservidor.commifichero.zip /var/www/html
```

En este caso, copiará el fichero "***mifichero.zip***" en el directorio destino de la nueva imágen.

7.3.3 Comando 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***".

7.3.4 Comando USER

Por defecto, todos los comandos lanzados en la creación de la imagen se ejecutan con el usuario root (usuario con UID=0). Para poder cambiar esto, podemos usar el comando **USER**, indicando el nombre de usuario o UID con el que queremos que se ejecute el comando. Por ejemplo:

```
USER sergi
CMD id
```

Mostrará con el comando "id" el uid y otra información del usuario "sergi".

7.3.5 Comando 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
```

Creará la carpeta "***tmp***" tanto en "***/root***" como en "***/var/www/html***". Si los directorios "***/root***" o "***/var/www/html***" no hubieran existido, los hubiera creado.

7.3.6 Comando ENV

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

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

Esto definirá las variables de entorno "v1" y "v2" con los valores "valor1" y "valor2".

7.3.7 Otros comandos útiles: ARG, VOLUME, LABEL, HEALTHCHECK

Aquí comentamos comandos útiles:

- **ARG**: permite enviar parámetros al propio "***Dockerfile***" con la opción "***--build-arg***" del comando "***docker build***".
- **VOLUME**: permite establecer volúmenes por defecto en la imágen. 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 swarn"***, aunque otros como "***Kubernetes***" incorporan su propio sistema

8. TRUCOS PARA HACER NUESTRAS IMÁGENES MÁS LIGERAS

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".
- No instalar programas innecesarios, incluso evitando herramientas tipo Vim, Nano, etc.
- 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>***"
- Utiliza comandos de limpieza tras instalaciones con "***apt***", tales como:

    - "***rm -rf /var/lib/apt/lists/****" tras crear una imagen para borrar las listas generadas al realizar "apt update".
    - "***apt-get purge --auto-remove && apt-get clean***" para eliminar temporales de apt.
- Al usar "**apt install**" usar la opción "***no-install-recommend***", para que no instale paquetes recomendados asociados al paquete instalado.
- Analiza tus "Dockerfile" con https://www.fromlatest.io/#/ y sigue sus consejos.

Más información en:

- https://hackernoon.com/tips-to-reduce-docker-image-sizes-876095da3b34
- https://medium.com/sciforce/strategies-of-docker-images-optimization-2ca9cc5719b6

9. BIBLIOGRAFÍA

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

[2] Visual Studio Code Learn https://code.visualstudio.com/learn

[3] FROM:latest https://www.fromlatest.io/#/