Introducción a Docker

UD 04. Gestión de imágenes en Docker

Autor: Sergi García Barea

Actualizado Febrero 2025

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

También podemos utilizar filtros sencillos usando la nomenclatura ***"docker images [REPOSITORIO[:TAG]]"***.

```
docker images ubuntu:22.04
```

Nos mostrará la imagen del repositorio "ubuntu" en su versión "22.04".

```
alumno@alumno-virtualbox:~/Desktop$ docker images
REPOSITORY    TAG       IMAGE ID       CREATED       SIZE
ubuntu        latest    6d79abd4c962   12 days ago   78.1MB
ubuntu        22.04     b1dc6972547a   4 weeks ago   77.9MB
hello-world   latest    1b44b5a3e06a   6 weeks ago   10.1kB
alumno@alumno-virtualbox:~/Desktop$ docker images ubuntu:22.04
REPOSITORY   TAG       IMAGE ID       CREATED       SIZE
ubuntu       22.04     b1dc6972547a   4 weeks ago   77.9MB
alumno@alumno-virtualbox:~/Desktop$
```

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"
```

```
alumno@alumno-virtualbox:~/Desktop$ docker images -f=reference="u*:*04"

REPOSITORY   TAG       IMAGE ID       CREATED       SIZE
ubuntu       22.04     b1dc6972547a   4 weeks ago   77.9MB
alumno@alumno-virtualbox:~/Desktop$
```

> **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:

```
alumno@alumno-virtualbox:~/Desktop$ 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
alumno@alumno-virtualbox:~/Desktop$
```

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:

```
alumno@alumno-virtualbox:~/Desktop$ docker history nginx
IMAGE          CREATED       CREATED BY                                      SIZE      COMMENT
41f689c20910   5 weeks ago   CMD ["nginx" "-g" "daemon off;"]                0B        buildkit.dockerfile.v0
<missing>      5 weeks ago   STOPSIGNAL SIGQUIT                              0B        buildkit.dockerfile.v0
<missing>      5 weeks ago   EXPOSE map[80/tcp:{}]                           0B        buildkit.dockerfile.v0
<missing>      5 weeks ago   ENTRYPOINT ["/docker-entrypoint.sh"]            0B        buildkit.dockerfile.v0
<missing>      5 weeks ago   COPY 30-tune-worker-processes.sh /docker-ent…   4.62kB    buildkit.dockerfile.v0
<missing>      5 weeks ago   COPY 20-envsubst-on-templates.sh /docker-ent…   3.02kB    buildkit.dockerfile.v0
<missing>      5 weeks ago   COPY 15-local-resolvers.envsh /docker-entryp…   389B      buildkit.dockerfile.v0
<missing>      5 weeks ago   COPY 10-listen-on-ipv6-by-default.sh /docker…   2.12kB    buildkit.dockerfile.v0
<missing>      5 weeks ago   COPY docker-entrypoint.sh / # buildkit          1.62kB    buildkit.dockerfile.v0
<missing>      5 weeks ago   RUN /bin/sh -c set -x     && groupadd --syst…   118MB     buildkit.dockerfile.v0
<missing>      5 weeks ago   ENV DYNPKG_RELEASE=1~bookworm                   0B        buildkit.dockerfile.v0
<missing>      5 weeks ago   ENV PKG_RELEASE=1~bookworm                      0B        buildkit.dockerfile.v0
<missing>      5 weeks ago   ENV NJS_RELEASE=1~bookworm                      0B        buildkit.dockerfile.v0
<missing>      5 weeks ago   ENV NJS_VERSION=0.9.1                           0B        buildkit.dockerfile.v0
<missing>      5 weeks ago   ENV NGINX_VERSION=1.29.1                        0B        buildkit.dockerfile.v0
<missing>      5 weeks ago   LABEL maintainer=NGINX Docker Maintainers <d…   0B        buildkit.dockerfile.v0
<missing>      5 weeks ago   # debian.sh --arch 'amd64' out/ 'bookworm' '…   74.8MB    debuerreotype 0.16
alumno@alumno-virtualbox:~/Desktop$
```

3.3 Eliminando imágenes con "docker rmi"

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

```
docker rmi ubuntu:22.04
```

Elimina la imagen ubuntu con la etiqueta 22.04

```
alumno@alumno-virtualbox:~/Desktop$ docker rmi ubuntu:22.04
Untagged: ubuntu:22.04
Untagged: ubuntu@sha256:4e0171b9275e12d375863f2b3ae9ce00a4c53ddda176bd55868df97ac6f21a6e
Deleted: sha256:b1dc6972547a6fd2bd691a8d37cfecb7f66f3b27279018ca61657c77c8c32b3c
Deleted: sha256:dc6eb6dad5f9e332f00af553440e857b1467db1be43dd910cdb6830ba0898d50
alumno@alumno-virtualbox:~/Desktop$
```

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:

```
alumno@alumno-virtualbox:~/Desktop$ docker rmi $(docker images -q)

Untagged: nginx:latest
Untagged: nginx@sha256:d5f28ef21aabddd098f3dbc21fe5b7a7d7a184720bc07da0b6c9b9820e97f25e
Deleted: sha256:41f689c209100e6cadf3ce7fdd02035e90dbd1d586716bf8fc6ea55c365b2d81
Deleted: sha256:6d03e4aefb16ea9e0d73cacb9a9fcb8f7fb3a806c41606600cab179aa381550f
Deleted: sha256:0951af45805a90677b83c3e3ae0ff7c1d6114b796206381914d1976df19d5af7
Deleted: sha256:dfa09858601e1678c6300924eeb880cbe0070bd41b3cc1bd8245f39104f1c8d7
Deleted: sha256:0502b463a609459d9b7bbd8044fd58e77f4309818da8841029dbf638a3a1581e
Deleted: sha256:39ffa72f273801580dbcd954e125f78fd5bb430896c32ab303667c5bbf154f39
Deleted: sha256:e57750b7e7ad7cb2436b3e9d3aafa338e89ee68f50d7e63bc09af37166ebc68e
Deleted: sha256:36f5f951f60a9fa1d51878e76fc16ba7b752f4d464a21b758a8ac88f0992c488
Untagged: alpine:3.10
Untagged: alpine@sha256:451eee8bedcb2f029756dc3e9d73bab0e7943c1ac55cff3a4861c52a0fdd3e98
Deleted: sha256:e7b300aee9f9bf3433d32bc9305bfdd22183beb59d933b48d77ab56ba53a197a
Deleted: sha256:9fb3aa2f8b8023a4bebbf92aa567caf88e38e969ada9f0ac12643b2847391635
Error response from daemon: conflict: unable to delete 6d79abd4c962 (must be forced) - image is being used by stopped container f95e77f187cb
Error response from daemon: conflict: unable to delete 1b44b5a3e06a (must be forced) - image is being used by stopped container b7b835e43279
alumno@alumno-virtualbox:~/Desktop$
```

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)
```

```
alumno@alumno-virtualbox:~/Desktop$ docker stop $(docker ps -a -q)

b7b835e43279
f95e77f187cb
alumno@alumno-virtualbox:~/Desktop$ docker rm $(docker ps -a -q)

b7b835e43279
f95e77f187cb
alumno@alumno-virtualbox:~/Desktop$
```

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:

```
alumno@alumno-virtualbox:~/Desktop$ 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: ubuntu:latest
untagged: ubuntu@sha256:353675e2a41babd526e2b837d7ec780c2a05bca0164f7ea5dbbd433d21d166fc
deleted: sha256:6d79abd4c96299aa91f5a4a46551042407568a3858b00ab460f4ba430984f62c
deleted: sha256:f9f52dc133e2af9188960e5a5165cafaa51657ef740ff20219e45a561d78c591
untagged: hello-world:latest
untagged: hello-world@sha256:54e66cc1dd1fcb1c3c58bd8017914dbed8701e2d8c74d9262e26bd9cc1642d31
deleted: sha256:1b44b5a3e06a9aae883e7bf25e45c100be0bb81a0e01b32de604f3ac44711634
deleted: sha256:53d204b3dc5ddbc129df4ce71996b8168711e211274c785de5e0d4eb68ec3851

Total reclaimed space: 78.13MB
alumno@alumno-virtualbox:~/Desktop$
```

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
usuariodockerhub/ubuntumod:2025
```

y tras ello, comprobamos las imágenes con

```
docker images
```

observamos lo siguiente:

```
alumno@alumno-virtualbox:~/Desktop$ docker ps
CONTAINER ID   IMAGE     COMMAND              CREATED         STATUS         PORTS     NAMES
be20e36e8229   ubuntu    "tail -f /dev/null"   5 seconds ago   Up 4 seconds             wonderful_williams
alumno@alumno-virtualbox:~/Desktop$ docker commit -a "Sergi" -m "Ubuntu modificado" be20 usuariodockerhub/ubuntumod:2025
sha256:6463f10ad7f514a1d8e6da0f870f2f6cc40aa7609a9a9696926ae1d10daff48f
alumno@alumno-virtualbox:~/Desktop$
```

Hemos obtenido lo siguiente: una nueva imagen, con nombre "***usuariodockerhub/ubuntumod***" con tag "***2025***", donde "usuariodockerhub" 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 usuariodockerhub/ubuntumod:2025
```

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 usuariodockerhub/ubuntumod:2025
usuariodockerhub/ubuntumod:latest
```

Obtendremos algo similar a:

```
alumno@alumno-virtualbox:~/Desktop$ docker tag usuariodockerhub/ubuntumod:2025 usuariodockerhub/ubuntumod:latest

alumno@alumno-virtualbox:~/Desktop$ docker images
REPOSITORY                   TAG       IMAGE ID       CREATED              SIZE
usuariodockerhub/ubuntumod   2025      6463f10ad7f5   About a minute ago   78.1MB
usuariodockerhub/ubuntumod   latest    6463f10ad7f5   About a minute ago   78.1MB
ubuntu                       latest    6d79abd4c962   12 days ago          78.1MB
alumno@alumno-virtualbox:~/Desktop$
```

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 usuariodockerhub/ubuntumod:2025
```

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 usuariodockerhub/ubuntumod
```

o de forma alternativa

```
docker save usuariodockerhub/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:

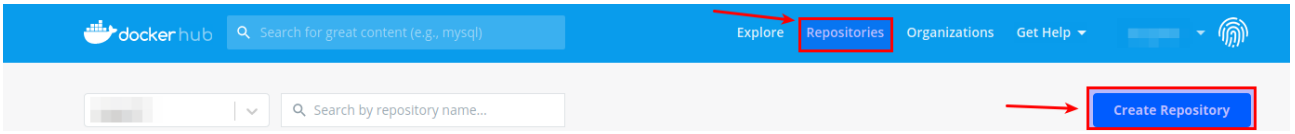
- Docker save: https://docs.docker.com/engine/reference/commandline/save/
- Docker load: https://docs.docker.com/engine/reference/commandline/load/

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:

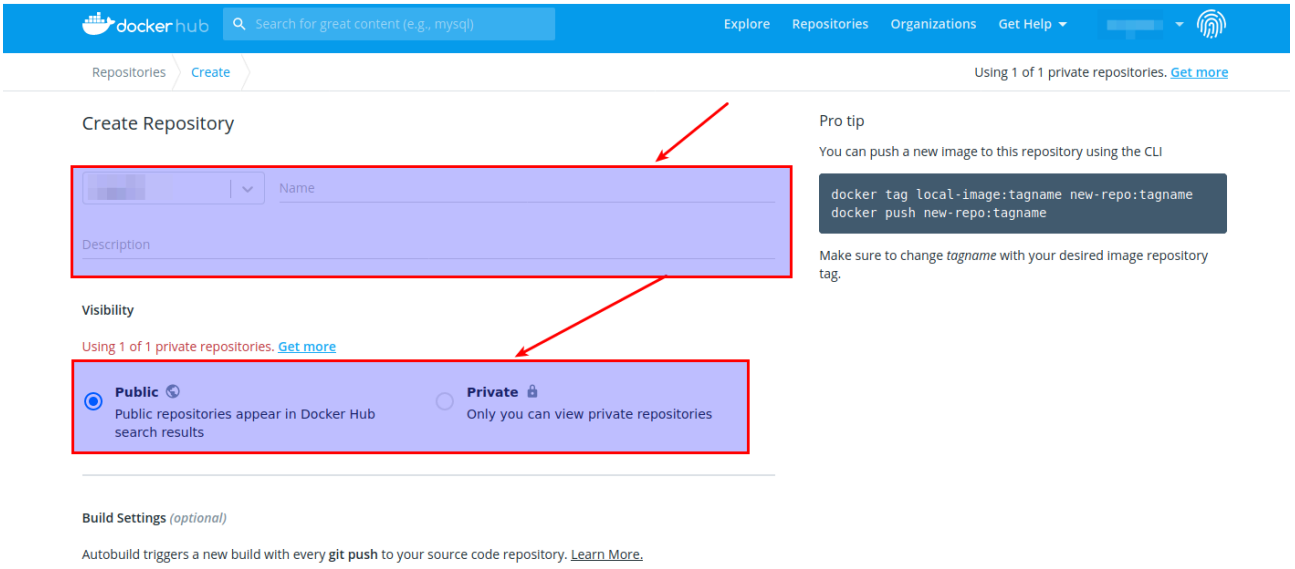
6.1 Paso 1: creando repositorio para almacenar la imagen en Docker Hub

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 "***usuariodockerhub***" y la imagen se llama "***prueba"***, podremos referenciarla en distintos contextos como "***usuariodockerhub/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 "usuariodockerhub" -m "Ubuntu modificado" IDCONTENEDOR usuariodockerhub/prueba
```

Hecho este "commit" local, debemos subirlo usando "docker push"

```
docker push usuariodockerhub/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

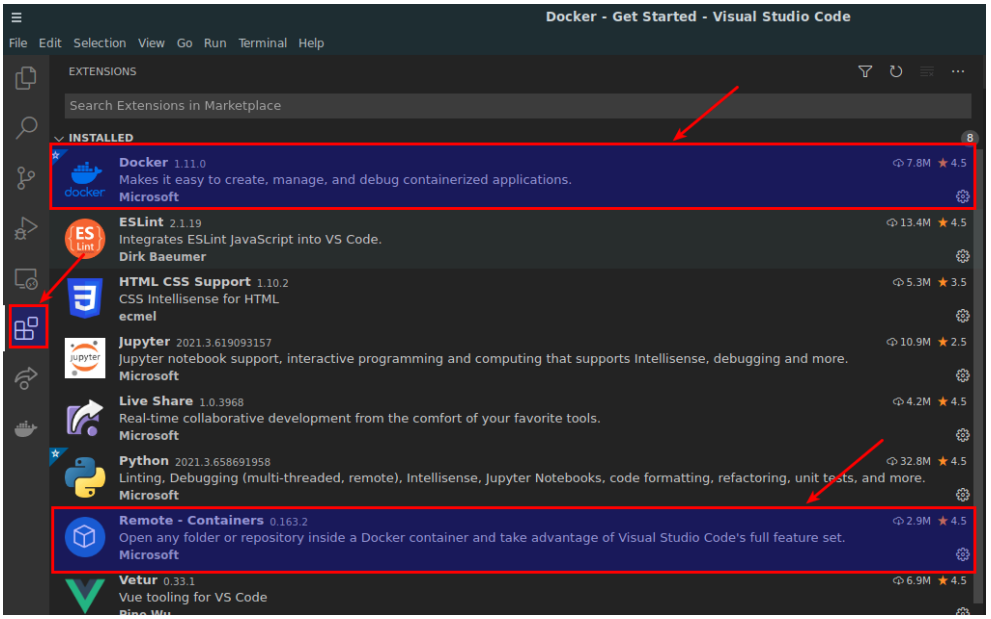
Docker nos permite generar de forma automática nuestras propias imágenes usando "***docker build***" y los llamados "***Dockerfile"***.

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

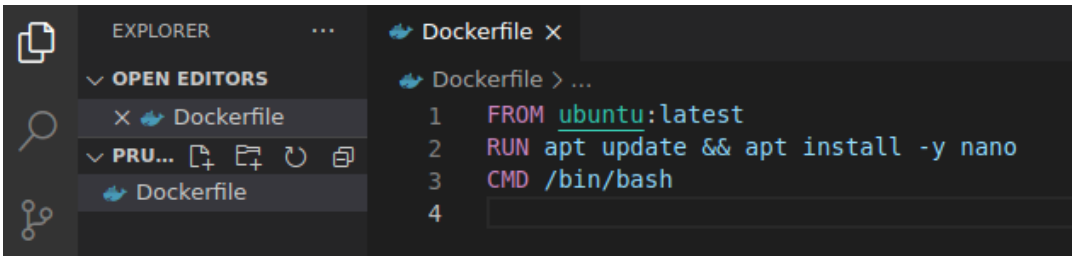
Empezaremos creando un sencillo ***"Dockerfile"*** donde crearemos una imagen de Ubuntu con el editor de texto "nano" instalado. Para ello indicaremos:

- 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

```
alumno@alumno-virtualbox:~/Desktop/pruebaDockerFile$ docker build -t ubuntunano ./
[+] Building 18.6s (6/6) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 130B
 => [internal] load metadata for docker.io/library/ubuntu:latest
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [1/2] FROM docker.io/library/ubuntu:latest
 => [2/2] RUN apt update && apt install -y nano
 => exporting to image
 => => exporting layers
 => => writing image sha256:c4726d9873465d507d90c5f060fc00439c913b0176c8bc876fbe4c59b7cf0b7d
 => => naming to docker.io/library/ubuntunano

 1 warning found (use docker --debug to expand):
 - JSONArgsRecommended: JSON arguments recommended for CMD to prevent unintended behavior related to OS signals (line 4)
alumno@alumno-virtualbox:~/Desktop/pruebaDockerFile$
```

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 imagen que hemos creado con "***docker history***"

```
alumno@alumno-virtualbox:~/Desktop/pruebaDockerFile$ docker history ubuntunano
IMAGE          CREATED          CREATED BY                                      SIZE      COMMENT
c4726d987346   42 seconds ago   CMD ["/bin/sh" "-c" "/bin/bash"]                0B        buildkit.dockerfile.v0
<missing>      42 seconds ago   RUN /bin/sh -c apt update && apt install -y …   55.3MB    buildkit.dockerfile.v0
<missing>      12 days ago      /bin/sh -c #(nop)  CMD ["/bin/bash"]            0B
<missing>      12 days ago      /bin/sh -c #(nop) ADD file:dafefa97de6dc66a6…   78.1MB
<missing>      12 days ago      /bin/sh -c #(nop)  LABEL org.opencontainers.…   0B
<missing>      12 days ago      /bin/sh -c #(nop)  LABEL org.opencontainers.…   0B
<missing>      12 days ago      /bin/sh -c #(nop)  ARG LAUNCHPAD_BUILD_ARCH     0B
<missing>      12 days ago      /bin/sh -c #(nop)  ARG RELEASE                  0B
alumno@alumno-virtualbox:~/Desktop/pruebaDockerFile$
```

Donde observamos, que nuestra imagen ha crecido en tamaño 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

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 imagen.

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/#/