Introducció a Docker

UD 01 (Part 04). Gestió d'imatges en Docker

Autor: Sergi García Barea

Actualitzat setembre 2021

Llicència

Nomenclatura

Al llarg d'aquest tema s'utilitzaran diferents símbols per distingir elements importants dins del contingut. Aquests símbols són:

📖 **Important**

❗ **atenció**

💬 **Interessant**

UD01 (PART 04). GESTIÓ D'IMATGES EN DOCKER

1. INTRODUCCIÓ

Fins ara, hem vist com descarregar i treballar amb imatges de tercers en Docker. En aquesta unitat explicarem com gestionar les imatges de contenidors Docker (llistat, eliminació, història, etc.) així com la seua creació tant de forma manual com utilitzant el comandament ***"docker build"*** amb els anomenats "Dockerfiles".

2. LLISTANT IMATGES LOCALS I PER A BAIXAR

2.1 Llistant imatges locals

Podem obtenir informació de quines imatges tenim emmagatzemades localment utilitzant

```
docker images
```

Obtenint un resultat similar a el següent, on veiem informació sobre les imatges

```
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:~$
```

Podem utilitzar filtres senzills usant la nomenclatura ***"docker images [REPOSITORIO[:TAG]]".***

```
docker images ubuntu:14.04
```

Ens mostrarà la imatge del repositori "ubuntu" en la seua versió "14.04".

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

Si volem utilitzar algun filtre avançat, podem usar l'opció "-f". Aquí un exemple, filtrant les imatges que comencen per "o" i acabe la seua etiqueta a "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
```

> **Atenció:** no confondre aquest comandament amb ***"docker image"*** (sense la s final).

Més informació a https://docs.docker.com/engine/reference/commandline/images/

2.2 Llistant imatges per a baixar

Podem obtenir informació d'imatges que podem descarregar en el registre (per defecte, Docker Hub) utilitzant el comandament "docker search". Per exemple amb la següent comanda:

```
docker search Ubuntu
```

Ens apareixen aquelles imatges disponibles en el registre (Docker Hub) amb aquesta paraula.

3. DESCARREGANT I ELIMINANT IMATGES (I CONTENIDORS) LOCALS

3.1 descarregant imatges amb "docker pull"

Podem emmagatzemar imatges localment des del registre sense necessitat de crear un contenidor mitjançant la comanda ***"docker pull"***, clarament inspirat en sistemes de control de versions com ***"git".*** Per conéixer els seus noms i versions, podem usar la comanda "docker search" explicat anteriorment o visitar https://hub.docker.com/.

```
docker pull alpine:3.10
```

Aquesta comanda ens descarrega la imatge ***"alpine"*** amb el tag ***"3.10",*** com veiem 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 l'historial d'una imatge descarregada

Podeu observar l'historial d'una imatge descarregada, és a dir, en quines versions es basa, utilitzant la comanda ***"docker history".*** Per exemple amb:

```
docker history nginx
```

Obtenim el següent:

```
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 Eliminant imatges amb "docker rmi"

Amb la comanda ***"docker rmi"*** podem eliminar imatges emmagatzemades localment.

```
docker rmi ubuntu:14.04
```

Elimina la imatge “ubuntu” amb l'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 manera d'eliminar **totes** les imatges locals, que no estiguin sent usades per un contenidor, combinant ***"docker images -q"*** per obtenir la llista i ***"docker rmi"*** és la següent :

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

Aquí s'observa l'esborrat, excepte d'aquelles usades per un contenidor:

```
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 Eliminant contenidors amb "docker rm"

Aprofitant que tractem l'esborrat d'imatges, comentem com esborrar contenidors aturats (si un contenidor està en marxa, ha de ser aturat abans de l'esborrat).

Amb la següent ordre es pot esborrar un contenidor per identificador o nom

```
docker rm IDENTIFICADOR/NOM
```

Així mateix, una forma d'esborrar tots els contenidors (que estiguen aturats), de manera similar a com vam veure en l'anterior punt, és la següent:

Pas 1 (opcional): parem tots els contenidors:

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

Pas 2: esborrem tots els contenidors:

```
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 Eliminant totes les imatges i contenidors amb "docker system prune -a

Una manera de fer les operacions anteriors de cop, és usant "docker system prune -a", que elimina tota imatge i contenidor aturat.

Pas 1 (opcional): parem tots els contenidors:

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

Pas 2: esborrem tots els contenidors:

```
docker system prune -a
```

Obtenint una cosa similar a això:

```
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. CREANT LES NOSTRES PRÒPIES IMATGES A PARTIR D'UN CONTENIDOR EXISTENT

El sistema d'imatges de Docker funciona com un control de versions per capes, de manera similar a l'eina ***"git"*** per a control de versions. Podem entendre que un contenidor és com una "capa temporal" d'una imatge, per la qual cosa, podem fer un ***"commit"*** i convertir aquesta "capa temporal" en una imatge. La sintaxi més habitual és la següent

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

Per exemple, si tenim un contenidor amb nom***"ubuntumod"*** que simplement és un contenidor basat en la imatge ***"ubuntu"*** en què s'ha instal·lat un programa i fem:

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

i després d'això, vam comprovar les imatges amb

```
docker images
```

observem el següent:

```
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:~$
```

Hem obtingut el següent: una nova imatge, amb nom***"sergi/ubuntumod***"amb tag***"2021",***on "sergi" actua com a nom d'usuari per usar-lo en un repositori remot (recordem novament, que per defecte és "Docker Hub").

Ara ja podríem crear nous contenidors amb aquesta imatge, usant per exemple:

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

Si volguérem afegir una nova etiqueta a la imatge, com "latest", podem usar el comandament ***"docker tag",*** tenint en compte que una mateixa imatge pot tenir diverses etiquetes:

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

Obtindrem alguna cosa 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:~$
```

Per eliminar una etiqueta, simplement haurem d'esborrar la imatge amb "docker rmi". La imatge es mantindrà mentre almenys tinga una etiqueta. Per exemple amb:

```
docker rmi sergi/ubuntumod:2021
```

quedaria així:

```
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:~$
```

Més informació dels comandaments en:

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

5. EXPORTANT/IMPORTANT IMATGES LOCALS A/DES DE FITXERS

Un cop tinguem una imatge local en el nostre sistema, podem fer una còpia d'aquesta, sigui com a còpia de seguretat o com a forma de transportar-la a altres sistemes mitjançant el comandament ***"docker save".*** Per exemple es pot fer d'aquestes dues formes:

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

o de forma alternativa

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

Si volem importar el fitxer per crear una imatge a la nostra màquina, podem usar "docker import". Per exemple es pot fer d'aquestes dues formes:

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

o de forma alternativa

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

Més informació sobre els comandaments:

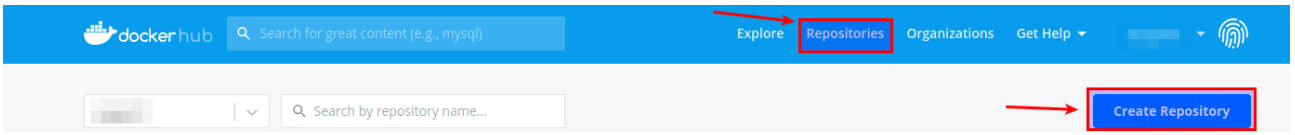
- Docker save: https://docs.docker.com/engine/reference/ commandline/save/
- Docker load: https://docs.docker.com/engine/reference/commandline/load/

6. PUJANT LES NOSTRES PRÒPIES IMATGES A UN REPOSITORI (DOCKER HUB)

Podem pujar una imatge a un repositori (per defecte Docker Hub). Per a això, hem de realitzar els següents passos:

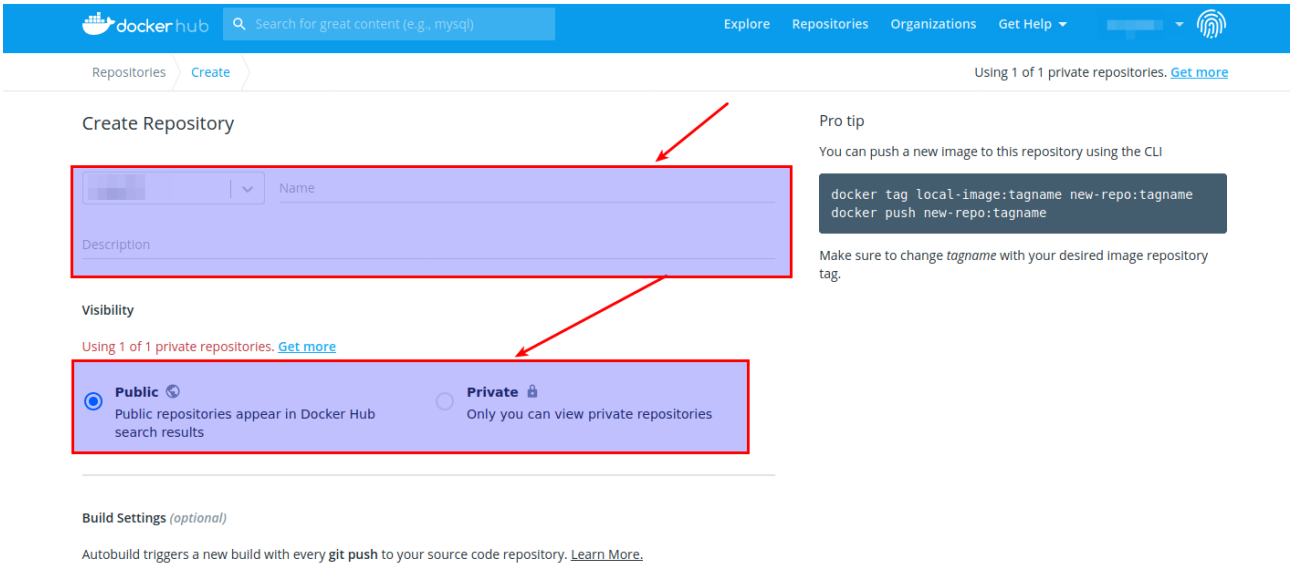
6.1 Pas 1: creant repositori per emmagatzemar la imatge en Docker Hub

En primer lloc, heu de crear-vos un compte a https://hub.docker.com i logear-vos. Una vegada loguejats, heu d'accedir a ***"Repositories"*** i ací a ***"Create repository"*** de manera similar a com es veu en la imatge següent:



Després d'això, podreu quedar un repositori amb el vostre compte i triar si aquest repositori és públic (qualsevol pot accedir) o privat (només pot accedir amo o autoritzats).

La pantalla de creació del repositori té un aspecte similar a aquest:



Un cop creat, si el teu usuari és ***"sergi"*** i la imatge es diu ***"prova",*** podrem referenciar en diferents contextos com ***"sergi/prova***"

6.2 Pas 2: emmagatzemant imatge local en repositori Docker Hub

En primer lloc, haurem de loguearnos mitjançant consola al repositori mitjançant la comanda

```
docker login
```

Un cop loguejat, hem de fer un "commit" local de la imatge, seguint l'estructura vista en punts anteriors. Un exemple podria ser:

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

Fet aquest commit local, hem de pujar-lo usant "docker push"

```
docker push sergi/prueba
```

Un cop fet això, si la imatge és pública (o privada amb permisos), qualsevol podrà descarregar-la i crear contenidors usant ***"docker pull"*** o ***"docker run***".

Més informació de les ordres:

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

7. GENERAR AUTOMÀTICAMENT LES NOSTRES PRÒPIES IMATGES MITJANÇANT DOCKERFILE

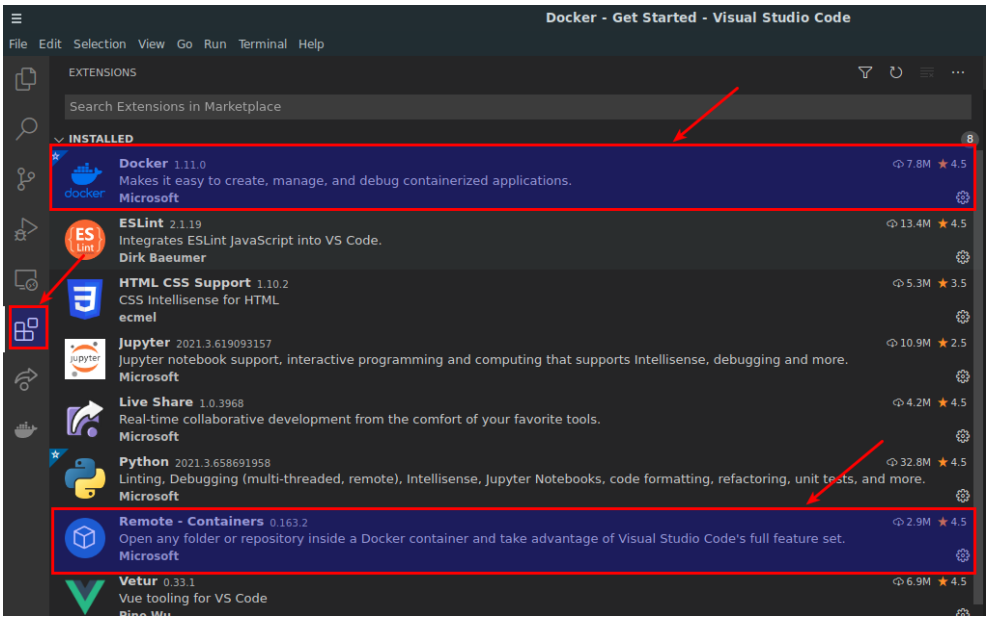
Docker ens permet generar de forma automàtica les nostres pròpies imatges usant ***"docker build"*** i els anomenats "Dockerfile".

7.1 Editor Visual Studio Code i connectors associats a Docker

Els fitxers ***"Dockerfile"*** poden crear-se amb qualsevol editor de text, però des d'aquí recomanem l'editor multiplataforma ***"Visual Studio Code***" https://code.visualstudio.com/

Per saber més sobre com utilitzar aquest editor podeu fer servir https://code.visualstudio.com/learn

En instal·lar-lo, si detecta Docker instal·lat en el sistema, el mateix editor ens suggerirà una sèrie de connectors. Val la pena instal·lar-los. Si no, sempre podeu buscar a connectors manualment. Jo personalment, us recomane aquests dos que podeu veure a la imatge:



7.2 Creant el nostre primer Dockerfile

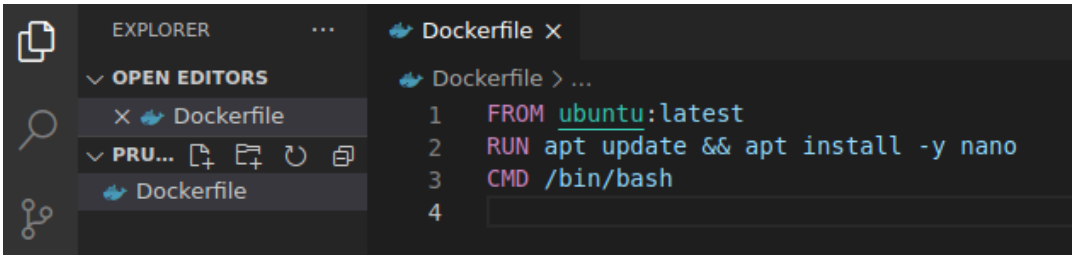
Començarem creant un senzill ***"Dockerfile"*** on crearem una imatge d'Ubuntu amb l'editor de text "nano" instal·lat. Per a això indicarem:

- De quina imatge base partirem.
- Que ordres llançarem sobre la imatge base, per crear la nova imatge
- Què comanda s'associarà per defecte en llançar un contenidor amb la nova imatge

Creem el fitxer***"Dockerfile"***(Visual Studio Code li posarà una icona de la balena) i afegim:

```
FROM ubuntu:latest
RUN apt update && apt install -y nano
# Aquí un comentario
CMD /bin/bash
```

a l'editor quedarà d'una manera similar a:



Si ara fem servir la comanda "docker build" de la següent manera:

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

Obtindrem alguna cosa 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.
```

el que hem fet és "executar" el que marca el ***"Dockerfile".*** El resultat s'ha guardat en una nova imatge local amb el nom que hem especificat amb l'opció ***"-t".*** El lloc on es trobava el ***"Dockerfile"*** s'ha indicat mitjançant "***./"*** (directori actual).

> **Atenció:** el fitxer ha de cridar exactament "Dockerfile***",*** respectant majúscules i minúscules.

Si voleu especificar un nom de fitxer diferent a buscar en el directori, pot usar-se l'opció "-f", com en aquest exemple:

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

Podem observar la història de la imatge que hem creat amb "***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 "$(apt-get indextargets)" ]     0B
<missing>      13 days ago     /bin/sh -c set -xe   && echo '#!/bin/sh' > /…   811B
<missing>      13 days ago     /bin/sh -c #(nop) ADD file:c77338d21e6d1587d…   72.9MB
sergi@ubuntu:~/Desktop/pruebaDockerFile$
```

On observem, que la nostra imatge ha crescut en "28.6MB" al fer "***apt update && apt install -i nano***". Encara que en l'anterior "Dockerfile" hem fet servir un sol RUN, podríem haver utilitzat diversos RUN en lloc d'un, escrit d'una manera seqüencial com veiem en:

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

aquí simplement, hi hauria més capes intermèdies, com s'observa en "docker history" si generem la imatge amb la seqüència 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 "$(apt-get indextargets)" ]     0B
<missing>      13 days ago      /bin/sh -c set -xe   && echo '#!/bin/sh' > /…   811B
<missing>      13 days ago      /bin/sh -c #(nop) ADD file:c77338d21e6d1587d…   72.9MB
sergi@ubuntu:~/Desktop/pruebaDockerFile$
```

Els comandaments vistos (FROM, RUN i CMD) admeten diferents formats. Per saber-ne més podem visitar la seua ajuda: https://docs.docker.com/engine/reference/builder/

7.3 Altres ordres importants de Dockerfile

En crear un "Dockerfile" hi ha multitud d'ordres. A continuació expliquem les ordres més importants a utilitzar:

7.3.1 Comandament EXPOSE

En primer lloc, **repassem la diferència entre exposar i publicar ports en Docker**:

- Si no s'exposa ni publica un port, aquest Només des de l'interior del contenidor.
- Exposar un port, indica que aquest port és accessible tant dins el mateix contenidor com per altres contenidors, però no des de fora (inclòs l'amfitrió).
- Publica un port, indica que el port accessible des de fora del contenidor, per la qual cosa s'ha d'associar a un port de l'amfitrió.

L'opció Expose ens permet indicar els ports per defecte exposats que tindrà un contenidor basat en aquesta imatge. És similar a l'opció ***"--expose"*** de ***"docker run"*** (i de pas, recordem que "docker run" amb ***"-p"*** els publica). Per exemple, per a exposar 80, 443 i 8080 indicarem:

```
EXPOSE 80 443 8080
```

7.3.2 Comandament ADD/COPY

ADD i **COPY** són ordres per copiar fitxers de la màquina amfitrió al nou contenidor. Es recomana usar **COPY,** excepte que vulguem descomprimir un "zip", que **ADD** permet la seua descompressió. Més informació sobre la diferència entre **ADD** i **COPY:**

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

Exemple d'ús de ADD:

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

Descomprimirà el contingut de***"mifichero.zip"*** al directori destí de la nova imatge.

Exemple d'ús de COPY:

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

o fins i tot accedint des del web.

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

En aquest cas, copiarà el fitxer ***"mifichero.zip"*** al directori destí de la nova imatge.

7.3.3 Comando ENTRYPOINT

Per defecte, els contenidors Docker estan configurats perquè executen les ordres que es llancen mitjançant **"*/bin/sh* -c". Dit** d'una altra manera, les ordres que llançàvem, eren paràmetres per **"*/bin/sh* -c".** Podem canviar que comandament s'usa per això amb **ENTRYPOINT. Per** exemple:

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

Indicarem que les ordres siguen llançats amb ***"cat".*** En llançar el comandament "***/etc/passwd***", realment el que farem és que es llançarà ***"cat /etc/passwd***".

7.3.4 Comando USER

Per defecte, tots els comandaments llançats en la creació de la imatge s'executen amb l'usuari root (usuari amb UID=0). Per poder canviar això, podem usar la comanda **USER,** indicant el nom d'usuari o UID amb el qual volem que s'execute el comandament. Per exemple:

```
USER sergi
CMD id
```

Mostrarà amb la comanda "id" el uid i altra informació de l'usuari "sergi".

7.3.5 Comando WORKDIR

Cada vegada que expressem la comanda **WORKDIR,**estem canviant el directori de la imatge on executem les ordres. ***Si aquest directori no existeix, es crea.*** Per exemple:

```
WORKDIR /root
CMD mkdir tmp
WORKDIR /var/www/html
CMD mkdir tmp
```

Crearà la carpeta ***"tmp"*** tant en "***/root"*** com a "***/var/www/*html". Si els directoris***"/root"*** o ***"/var/www/html***" no haguessen existit, els haguera creat.

7.3.6 Comando ENV

La comanda ENV ens permet definir variables d'entorn per defecte en la imatge.

```
ENV v1=”valor1” v2=”valor2”
```

Això definirà les variables d'entorn "v1" i "v2" amb els valors "valor1" i "valor2".

7.3.7 Altres ordres útils: ARG, VOLUME, LABEL, HEALTHCHECK

Aquí comentem ordres útils:

- **ARG:** permet enviar paràmetres al propi ***"Dockerfile"*** amb l'opció ***"--build-arg"*** del comandament ***"docker build***".
- **VOLUME:** permet establir volums per defecte en la imatge. Parlarem dels volums més endavant en el curs.
- **LABEL:** permet establir metadades dins de la imatge mitjançant etiquetes. Un dels casos més típics, substituint a el comandament Maintainer, que aquesta "deprecated" és:
  - **LABEL** maintainer="sergi.profesor@gmail.com"
- **HEALTHCHECK:** permet definir com es comprovarà si aquest contenidor està funcionant correctament o no. Útil per sistemes orquestradors com***"Dockerswarn",*** encara que altres com***"Kubernetes"*** incorporen el seu propi sistema

8. TRUCS PER FER LES NOSTRES IMATGES MÉS LLEUGERES

En crear imatges, és habitual augmentar la mida de les imatges base. Alguns consells per la mesura del possible, augmentar la mida el menys possible:

- Utilitza imatges base lleugeres, tipus "Alpine".
- No instal·lar programes innecessaris, fins i tot evitant eines tipus Vim, Nano, etc.
- Minimitzar capes en "Dockerfile"
  - Millor usar ***"RUN apt-get install -i <packageA> <packageB>"*** que fer servir ***"RUN apt-get install -i*** <packageA>" i després d'això ***"RUN apt-get install -i <packageB>***"
- Utilitza ordres de neteja després instal·lacions amb "***apt***", com ara: "***rm -rf /var/lib/apt/lists/****" després de crear una imatge per esborrar les llistes generades a realitzar "apt update".
  - "***apt-get purge --auto-remove && apt-get clean***" per eliminar temporals d'apt.
- En utilitzar **"apt install"** usar l'opció ***"no-install-recommend",*** perquè no instal·le paquets recomanats associats al paquet instal·lat.

- Analitza els teus "Dockerfile" amb https://www.fromlatest.io/#/ i segueix els seus consells.

Més informació a:

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

9. BIBLIOGRAFIA

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

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

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