

# Pràctica 1: Docker

---

## Introducció

En aquesta pràctica ens introduïrem el món de la Virtualització a nivell d'SO, utilitzant el Docker per manipular l'ús de contenidors, podent controlar l'espai de directoris/fitxers al quals poden accedir, el nombre màxim de CPUs que poden utilitzar, la memòria de CPUs que poden utilitzar i el nombre màxim de processos que es poden executar a un contenidor.

**Pregunta: Com es pot “despertar” un contenidor que ha sigut “aturat” prèviament de forma que es puguin introduir noves instruccions dins del contenidor?**

Un cop has fet exit, per a tornar a despertar un contenidor que ha estat aturat s'han de cridar a les comandes **docker container start <idContenidor>** (es pot saber l'id del contenidor fent **docker container ls -a**) i després la segona comanda és **docker container attach <idContenidor>**

**Pregunta: Quants processos fork-bomb s'estan executant dins del contenidor? com ho compteu? podem fer servir instruccions de la línia de comandes per saber-ho?**

32, ho posa a la dreta del tot a la columna PIDS després d'obrir un nou terminal i cridar a la comanda **ctop**, a més de ser el nombre de forks que havíem limitat amb la comanda **docker run --ulimit nproc=32:64 --cpus 1 -ti fork-bomb**

**Pregunta: Quanta CPU està utilitzant l'ordinador? està utilitzant 1/2 o més CPU's?**

De les 4 CPU's que té l'ordinador, està utilitzant una quantitat que varia amb el temps i que oscil·la entre 94% i 106% , però si fem la mitja trobem que treballa al 100% per la qual cosa està utilitzant 1 CPU. Tal i com ho havíem limitat amb la comanda **docker run --ulimit nproc=32:64 --cpus 1 -ti fork-bomb**

**Pregunta: Observeu que en aquest cas hem executat el contenidor en mode interactiu. És a dir, apareix la línia de comandes que ens permet executar una comanda. Proveu de fer-ho! Per exemple, executeu un ls o ps. Per què dona el bash un missatge d'error?**

Perquè no deixa executar més processos dels que excedeixin el límit que havíem imposat unes quantes comandes abans amb la comanda **docker run --ulimit nproc=32:64 --cpus 1 -ti fork-bomb** per això dona el missatge d'error **bash: fork: retry: Resource temporarily unavailable**

**Pregunta: Proveu d'executar els dos servidors en dos terminals diferents, sense fer servir cap contenidor. Què és el que succeeix en intentar fer-ho?**

El dispositiu no accepta la petició, bind() ha fallat. Prova un altre port.

**Pregunta: Per què succeeix?**

Perquè port 5000 ja està ocupada per l'altre servidor.

**Pregunta: Observeu, al README, que per fer el mapat de ports es fa servir l'opció "-p" per executar el contenidor. Descriviu breument, fent servir la documentació oficial del Docker, què és el que permet fer l'opció "-p". Indiqueu també què passa si no es fa servir l'opció "-p" per executar el contenidor. Quina és la pàgina web del Docker on està descrit el funcionament d'aquesta opció?**

L'opció -p s'utilitza per fer que un port estigui disponible per a serveis fora de Docker o per als contenidors de Docker que no estiguin connectats a la xarxa del contenidor. Això crea un tallafoc que assigna un port de contenidor a un port de l'amfitrió Docker al món exterior.

Si no fem servir l'opció -p per executar el contenidor, aquest es manté aïllat del dispositiu local.

El funcionament d'aquesta opció està descrit a:

<https://docs.docker.com/config/containers/container-networking/>

**Pregunta: Per què, actualment, es fan servir els contenidors per executar els serveis associats a una aplicació més gran en contenidors diferents? Quines avantatges aporta?**

Perquè permeten implementar i ajustar l'escala d'aplicacions, llibreries i altres dependències ràpidament en qualsevol entorn amb la certesa de saber que el codi s'executarà. De tal manera que podem agafar diverses parts d'un programa (modularització), i generar una major seguretat i facilitat per a modificar-les.

**AVANTATGES:**

- Permet treballar zones aïllades
- Alta seguretat
- Sistema Operatiu independent de l'aplicació
- Major facilitat per al desenvolupament i l'execució
- Millor escalabilitat i flexibilitat
- No està fix a un dispositiu local (independent de la màquina)

**Exercici 3.2: Crear un contenidor que tingui el compilador (gcc). En executar el contenidor es farà un "bind mount" de forma que internament hi ha un director /home/appuser/practica4 del qual es pot accedir a un directori extern al Docker el qual contindrà el codi font i les dades de la pràctica 4. Provar de compilar i executar el codi des de l'interior del contenidor.**

**Comproveu que els executables així com les dades associades que es llegeixen i s'escriuen en executar les aplicacions són fora del contenidor.**

Contenidor =>

```
oslab:~/Desktop/practica4> docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS        NAMES
b79ed17f4b81   practica4 "bash"                4 minutes ago Up 4 minutes             epic_benz
```

Codi =>

```
docker run -it --mount  
type=bind,source=/home/oslab/Desktop/practica4/,target=/home/appu  
ser/practica4 practica4
```

```
docker run -it --mount type=bind,source='<Directori  
absolut>',target=<Directori de Contenidor> <contenidor>
```

Demostració =>

```
oslab:~/Desktop/practica4> docker run -it --mount type=bind,source=/home/oslab/Desktop/practica4/,target=/home/appuser/practica4 practica4  
appuser@31b8fc35c54a:~$ ls  
practica4  
appuser@31b8fc35c54a:~$ cd practica4/  
appuser@31b8fc35c54a:~/practica4$ ls  
Dockerfile Makefile Readme.txt data dataStructures hash.bin mainRecc mainRecc.c mainRecc.o mainSave mainSave.c mainSave.o matrix.bin meta.bin model nginx-image  
appuser@31b8fc35c54a:~/practica4$
```

**Pregunta: Observar que per poder escriure al directori extern al Docker cal permisos per poder-ho fer. Com ho solucioneu?**

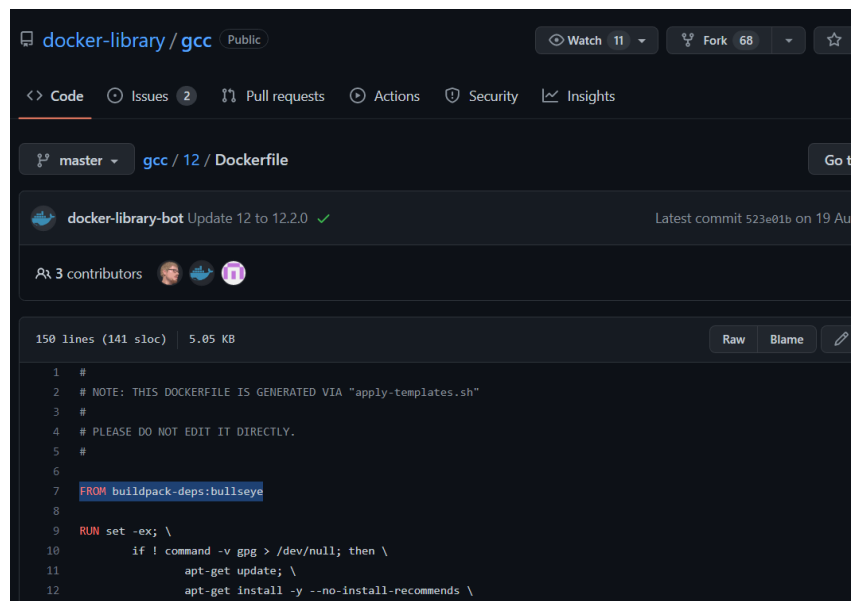
Com el contenidor no té el permís per a executar/modificar, variem --uid a 1000 per a que pertanyin al mateix grup (el de l'oslab) per a que tinguin els mateixos permisos de lectura i escriptura.

El codi emprat:

```
RUN addgroup --gid 1000 appgroup  
RUN useradd -r -uid 1000 -g appgroup appuser
```

**Exercici 3.3: L'objectiu és utilitzar una imatge que contingui el mínim necessari per poder executar l'aplicació (accedint a les dades externes amb el "bind mount"). Quin és el Dockerfile corresponent? Com heu arribat a trobar-lo?**

Primer de tot entrem al Dockerfile oficial de gcc de Docker, com demostra la primera imatge, i si llegim atentament el Dockerfile podem trobar que el gcc prové d'una imatge que es diu buildpack-deps:bullseye, i si llegim la documentació de buildpack-deps podem arribar la conclusió que: la imatge mínima necessària per executar el codi és buildpack-deps:bullseye-curl



The screenshot shows the Dockerfile for gcc in the docker-library repository. The file is located at gcc / 12 / Dockerfile. It contains the following content:

```
1 #  
2 # NOTE: THIS DOCKERFILE IS GENERATED VIA "apply-templates.sh"  
3 #  
4 # PLEASE DO NOT EDIT IT DIRECTLY.  
5 #  
6  
7 FROM buildpack-deps:bullseye  
8  
9 RUN set -ex; \  
10     if ! command -v gpg > /dev/null; then \  
11         apt-get update; \  
12         apt-get install -y --no-install-recommends \  
13
```

Documentació de buildpack-deps:

## What is `buildpack-deps` ?

In spirit, `buildpack-deps` is similar to Heroku's `stack images`. It includes a large number of "development header" packages needed by various things like Ruby Gems, PyPI modules, etc. For example, `buildpack-deps` would let you do a `bundle install` in an arbitrary application directory without knowing beforehand that `ssl.h` is required to build a dependent module.

### `curl`

This variant includes just the `curl`, `wget`, and `ca-certificates` packages. This is perfect for cases like the Java JRE, where downloading JARs is very common and necessary, but checking out code isn't.

El Dockerfile és el següent:

```
FROM buildpack-deps:bullseye-curl
RUN addgroup --gid 1000 appgroup
RUN useradd -r --uid 1000 -g appgroup appuser
RUN mkdir /home/appuser/
WORKDIR /home/appuser
USER appuser
```

```
oslab:~/Desktop/practica4> docker build -t small .
Sending build context to Docker daemon 3.373GB
Step 1/6 : FROM buildpack-deps:bullseye-curl
--> fle2c1abde86
Step 2/6 : RUN addgroup --gid 1000 appgroup
--> Using cache
--> dcf043a2f2dd
Step 3/6 : RUN useradd -r --uid 1000 -g appgroup appuser
--> Using cache
--> dab417bc74c4
Step 4/6 : RUN mkdir /home/appuser/
--> Using cache
--> da336fead1fb
Step 5/6 : WORKDIR /home/appuser
--> Running in b7510166e23a
Removing intermediate container b7510166e23a
--> 77b8893fe3ba
Step 6/6 : USER appuser
--> Running in 5a22fbf4cd76
Removing intermediate container 5a22fbf4cd76
--> e81534516dc6
Successfully built e81534516dc6
Successfully tagged small:latest
```

Per a veure que s'hagi creat bé:

```
oslab:~/Desktop/practica4> docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
small	latest	e81534516dc6	49 minutes ago	154MB
buildpack-deps	bullseye-curl	f1e2c1abde86	16 hours ago	154MB
practica4	latest	e6b65b24dc36	6 days ago	1.27GB

**Exercici 3.4: Pregunta: Què és el que fan cadascuna de les instruccions anteriors?**

**docker volume create vol-practica4** ➔ Crea un nou volum que anomenarem com a vol-practica4

**docker volume ls** ➔ Llista tots els volums coneguts per Docker

**cd practica4** ➔ Per a moure'ns al directori anomenat practica4

**docker container create --name temp -v vol-practica4:/data busybox** ➔ Creem un contenidor anomenat temp i indiquem el camí on volem crear-lo

**docker cp -a . temp:/data** ➔ amb docker cp -a copiem tot el contingut de /data al contenidor

**docker rm temp** ➔ Eliminem el contenidor temp

**Exercici: Comprovem que s'ha copiat tot correctament. Per això es demana executar el contenidor petit muntant el volum vol-practica4 en el directori /home/appuser/practica4. Què hi ha en aquest directori? Quin és l'usuari propietari dels arxius? Es pot executar el codi que hi ha?**

Executem el contenidor petit muntant el volum vol-practica4 en el directori /home/appuser/practica4:

```
oslab:~/Desktop/practica4> docker run -dit -u root --name volTest -v vol-practica4:/home/appuser/prac
ca4 small
cfe03baf6fe1f0f783ff657ec2f5cd92eddf25f16013a969a0ff96e80a5960ef
```

Mirem què hi ha al directori (i qui és el propietari):

```
oslab:~/Desktop/practica4> ls -l
total 3112860
drwxr-xr-x 2 oslab users      30 Sep 29 17:37 data
drwxr-xr-x 2 oslab users      45 Sep 29 17:37 dataStructures
-rwxrwx--- 1 oslab users    459 Oct  5 18:04 Dockerfile
drwxr-xr-x 2 oslab users      24 Oct  5 17:49 dockerfile 3.2
-rwxrwx--- 1 oslab users 3556000 Oct  5 18:24 hash.bin
-rwxrwx--- 1 oslab users   28000 Sep 22 18:08 mainRecc
-rwxrwx--- 1 oslab users    4838 Sep  1  2021 mainRecc.c
-rwxrwx--- 1 oslab users    5696 Sep 22 18:08 mainRecc.o
-rwxr-xr-x 1 oslab users   27816 Oct  5 18:23 mainSave
-rwxrwx--- 1 oslab users    2967 Sep  1  2021 mainSave.c
-rwxrwx--- 1 oslab users    3248 Sep 22 18:08 mainSave.o
-rwxrwx--- 1 oslab users    2370 Sep  2  2021 Makefile
-rwxrwx--- 1 oslab users 3183905592 Oct  5 18:24 matrix.bin
-rwxrwx--- 1 oslab users      8 Oct  5 18:24 meta.bin
drwxr-xr-x 2 oslab users     4096 Sep 29 17:38 model
drwxr-xr-x 3 oslab users      19 Sep 29 17:38 nginx-image
-rwxrwx--- 1 oslab users    1075 Sep  1  2021 README.txt
```

Comprovem que es pot executar el codi que hi ha:

```
oslab:~/Desktop/practica4> ./mainSave
oslab:~/Desktop/practica4> ./mainRecc 1 2207774
The number of movies seen by the user 2207774 is 57
oslab:~/Desktop/practica4> ./mainRecc 2 1
The number of users that have seen the movie 1 is 547
oslab:~/Desktop/practica4> ./mainRecc 3 2207774 1
U:2207774 - M:1 - The forecasted rating was 4.101167
oslab:~/Desktop/practica4> ./mainRecc 4 2207774
The recommended movie for user 2207774 is 1418
```

## CONCLUSIONS:

Amb aquesta pràctica ens hem pogut adonar de la importància i la utilitat de Docker, que permet lliurar codi amb més rapidesa, estandarditzar les operacions de les aplicacions, transferir el codi amb facilitat i estalviar diners en millorar l'ús de recursos. Amb Docker, es pot obtenir un sol objecte que es pot executar de manera fiable a qualsevol lloc. I tot i que nosaltres l'haguem utilitzat en el nivell més baix possible, el fet de poder crear un contenidor i que el codi es pugui seguir executant sense necessitat d'haver de descarregar llibreries ni res que requerís, ja ens permet imaginar lo útil que pot arribar a ser en un major nivell de programació.