Universitat de Barcelona

Facultat de Matemàtiques i Informàtica

Pràctica 2: Dockers

Sistemes Operatius II

Martí Martínez Gargallo i Víctor Sort Rubio

Professor: Óliver Díaz

Contents

1 Introducció

La virtualització a nivell de SO és un paradigma en el qual el SO permet crear espais d'usuaris aïllats per els processos que s'executin i és de gran utilitat per aspectes com la seguretat o la optimització de temps i recursos en els desenvolupadors, entre d'altres.

En aquesta segona pràctica estudiarem i treballarem sobre els contenidors, una tecnología de virtualització molt eficaç. Ho farem mitjançant Docker, la plataforma líder per gestionar contenidors; i la màquina virtual d'Ubuntu donada a l'assignatura.

2 Respostes i proves realitzades

2.1 Pràctica amb Dockers

2.1.1 Quin error s'obté a l'intentar eliminar la imatge de `hello-world` abans d'eliminar el contenidor asociat a ella? Com podríem solucionar el problema de manera que es pugui eliminar tant la imatge com els contenidors associats?

Si executem la comanda `docker rmi 9c7a54a9a43c` se'ns retorna l'error següent:

```
Error response from daemon: conflict: unable to delete 9c7a54a9a43c (must be forced) -
image is being used by stopped container dc7c4f9d3c36
```

Per solucionar-ho cal primer borrar els contenidors que estiguin associats a l'imatge (`docker rm dc7c4f9d3c36` i `docker rm 755adbac7638`) i després tornar a executar la comanda `docker rmi 9c7a54a9a43c`. S'observa, si usem les comandes `docker ps -a` i `docker images` que s'ha eliminat la imatge juntament amb els seus contenidors associats.

2.1.2 Quina comanda es pot utilitzar per eliminar simultàniament tots els contenidors i imatges no utilitzades?

Es podria usar la comanda `docker system prune`, la qual elimina recursos no utilitzats, com contenidors detinguts, imatges no usades i altres recursos no associats a cap contenidor en execució, com ens avisa:

```
WARNING! This will remove:
  - all stopped containers
    ...
```

També es podria usar la mateixa comanda amb l'opció `all`: `docker system prune --all`, que caldria usar però amb més precaució.

2.1.3 Quina versió de "gcc" té instal·lada en la MV i quina es va instal·lar dins del contenidor? Perquè pot resultar útil tenir dues versiones diferents en la màquina host i dins del contenidor?

A la MV es té la versió de "gcc" `(Ubuntu 11.3.0-1ubuntu1 22.04.1) 11.3.0` del 2021 i al contenidor la versió `(GCC) 13.2.0` del 2023. A més de ser útil per l'aïllament i la portabilitat també facilita la col·laboració amb altres desenvolupadors, ja que tots poden usar així contenidors amb les mateixes versions de llibreries i dependències, tal i com hem vist a teoria, sense afectar al host.

2.1.4 Comenta què succeeix a l'intentar instal·lar l'editor "vi" dins el contenidor statistics. Què pot explicar això?

A l'executar la comanda `apt-get install vim` ens dona l'error:

```
E: Could not open lock file /var/lib/dpkg/lock-frontend - open (13: Permission denied)
E: Unable to acquire the dpkg frontend lock (/var/lib/dpkg/lock-frontend), are you root?
```

El misstage d'error indica que no tenim els privilegis necessaris per instal·lar software dins del contenidor. L'error específic "Permission denied" es refereix a la falta de permisos per realitzar operacions d'administració del sistema.

2.1.5 Has pogut incloure les instruccions per instal·lar "vi" després de la línea `USER appuser`? Què podria explicar això? Finalment, quines instruccions has inclós a l'arxiu DockerFile per instal·lar "vi"?

Escrivint la comanda `RUN apt-get update && apt-get install -y vim` a la última línea (després de `USER appuser`) i intentar tornar a crear la imatge ens dona el següent error per pantalla (resumit):

```
 => ERROR [8/8] RUN apt-get update && apt-get install -y vim
 > [8/8] RUN apt-get update && apt-get install -y vim:
0.447 E: List directory /var/lib/apt/lists/partial is missing. - Acquire (13: Permission
denied)
...
not complete successfully: exit code: 100
```

Això succeeix perquè amb l'anterior comanda s'ha canviat l'usuari a appuser, el qual no té els permisos suficients per instal·lar software al sistema, com succeïa de manera semblant a la pregunta anterior. Per solucionar-ho n'hi ha prou en escriure la comanda en el DockerFile abans de canviar d'usuari. Un cop fet, si creem la imatge i l'executem creant un contenidor veiem que ja podem usar "vi" correctament.

2.1.6 Com es pot despertar un contenidor que ha sigut parat ("STATUS" = "Exited") de manera que es puguin introduir noves instruccions dins seu o usar el que va quedar grabat al seu interior? Escriu les instruccions concretes per el contenidor statistics.

Per despertar un contenidor usem la funció `docker start id-container`. En el nostre cas usem les següents comandes:

```
docker run -ti statistics
vi prova.tx
exit
docker ps -a
docker start 7b5bfa65fe8f
docker exec -it 7b5bfa65fe8f /bin/bash
ls
```

Primer creem el contenidor i en ell un arxiu "vi". Sortim del contenidor i busquem el seu id, usant la comanda que hem vist en preguntes anteriors. Usem aquest id per despertar la comanda usant `start` i usant `exec` accedim de nou al contenidor. Lúltima comanda `ls` serveix per comprovar que realment està l'arxiu `prova.txt`, com podem veure satisfactòriament.

2.1.7 Quants processos fork-bomb s'executen dins del contenidor després d'executar-se `./fork-bomb`? Com es pot comprovar el número?

Notem inicialment que al haver executat la imatge usant `--ulimit nproc=32:64` el màxim nombre de processos permesos és 32 (i 64 fils). Executant `./fork-bomb` i `ctop` en una altre terminal amb l'usuari oslab observem a la penúltima columna del contenidor que aquest té 32 PIDS, és a dir, 32 processos, el màxim permés.

2.1.8 Quanta CPU està usant el contenidor a l'executar-se `./fork-bomb`? Utilitza 1, 2 o més CPUs?

De manera similar a l'anterior pregunta notem inicialment que al haver executat la imatge usant `--cpus 1` es limita el contenidor a usar un sol nucli de CPU. Executant `./fork-bomb` i `ctop` en una altre terminal amb l'usuari oslab observem a la segona columna del contenidor que aquest usa un 200% de CPU aproximadament. Ens podria sobtar que no fos un 100%, però cal tenir en compte que a Linux amb `ctop` es conta cada fil com a CPU.

2.1.9 Podem executar comandes vía línea de comandes dins del contenidor després de que s'executi `fork-bomb`? Perquè bash dona un missatge d'error?

Si ho intentem amb qualsevol comanda, per exemple `ls`, sen's retorna el següent missatge d'error:

```
bash: fork: retry: Resource temporarily unavailable     ...
bash: fork: Resource temporarily unavailable
```

Com es pot deduïr de les dues anteriors preguntes aquest missatge indica que bash no pot crear un nou procés fill doncs `fork-bomb` ha esgotat els recursos del sistema.

2.1.10 Què succeeix a l'intentar executar ambdos servidors en dos terminals diferents sense utilitzar cap contenidor?

Quan intentem executar el segon servidor ens dóna un error sobre el port que s'està utilitzant i es suggereix utilitzar un altre port. Això és perquè el mateix servidor executat primer escolta el port ja, i per tant amb el mateix codi sense contenidors ens resulta impossible fer la connexió.

2.1.11 Descriu breument què permet fer la opció *-p*. Què succeeix si no se utiliza la opció *-p* a l'executar el contenidor. Proporciona l'enllaç de la pàgina web de Docker en la que basses la resposta.

La opció `-p` permet mapejar el port del contenidor (en el nostre cas 5000) a un altre port de l'amfitrió. En el cas del servidor 1 es mapeja al port 3000 i en el servidor 2 al port 4000.

Si no s'usa la opció `-p` la terminal espera que s'especifique la imatge d'un servidor (en el nostre cas el que s'acaba de crear), així que s'envia un error.

Podem trobar informació al respecte a la web https://docs.docker.com/engine/reference/run, a la secció ENTRYPOINT (default command to execute at runtime).

2.1.12 Perquè actualment s'utilitzen contenidors per executar els serveis associats amb una aplicació gran en diferents contenidors? ¿Quins aventatges aporta?

En primer lloc, els contenidors permeten l'aïllament dels serveis, la qual cosa millora la seguretat, ja que cada servei està contingut en un entorn separat. Això redueix els riscos d'interferències i conflictes entre els serveis.

A més a més, els contenidors simplifiquen la gestió de dependències. Cada contenidor pot contenir totes les llibreries i recursos necessaris per al servei, eliminant així les complicades configuracions de dependències que sovint es produeixen en entorns compartits.

Això també millora la portabilitat, ja que els contenidors poden ser desplegats de manera consistent en diferents entorns, com ara equips de desenvolupament, etapes de prova i producció. A més a més, la gestió de recursos és més eficient, ja que els contenidors comparteixen el nucli del sistema operatiu i només consumeixen els recursos que necessiten.

2.2 Ús de dades externes i càrrega/descàrrega del nostre Docker a Docker Hub

2.2.1 Quina ha sigut la instrucció usada per executar el contenidor de manera que pogués accedir a la carpeta local des de l'interior del contenidor?

Una vegada carregada la imatge gcc des del Docker Hub, la hem executat mitjançant aquesta comanda per a poder accedir a la carpeta local:

```
docker run -ti -v /media/sf_sharedfoldervm/practica4/practica4:/home/appuser/practica4/ gcc
```

Comprovem que la comanda s'ha executat correctament, navegant per la carpeta local des del contenidor:

```
> docker run -ti -v /media/sf_sharedfoldervm/practica4/practica4:/home/appuser/practica4/ gcc
> root@ab0b9da0f698:/# ls
bin   dev  home lib32 libx32 mnt proc  run  srv tmp var
boot  etc  lib  lib64 media  opt root  sbin sys usr
```

Cal destacar que hem pogut compilar i executar correctament des de l'interior del compilador, i una vegada hem sortit hem pogut executar els fitxers que havíem compilat des de dins amb els resultats desitjats.

2.2.2 Quant espai en disc estan usant els contenidors amb i sense gcc? Quines comandes s'utilitzen per descobrir aquesta informació?

En primer lloc veurem com hem pogut executar el codi compilat en el primer contenidor des del segon:

```
> docker run -ti -v /media/sf_sharedfoldervm/practica4/practica4:/home/appuser/practica4/ ubuntu
root@9f0981236388:/home/appuser/practica4# ./mainRecc 1 2207774
The number of movies seen by the user 2207774 is 57
```

Podem veure, amb la comanda `docker images` l'espai de les imatges corresponents als dos contenidors, però no ens interessen les imatges, sinó els propis contenidors,

```
oslab@oslab:/media/sf_sharedfoldervm/practica4/practica4$ docker images
REPOSITORY   TAG       IMAGE ID       CREATED       SIZE
gcc          latest    2b82e1a0f207   6 days ago    1.38GB
ubuntu       latest    e4c58958181a   13 days ago   77.8MB
```

A continuació podem veure l'espai que estan usant els contenidors en total amb `docker system df` i l'espai individual de cada contenidor amb `docker ps -s -a`:

```
oslab@oslab:/media/sf_sharedfoldervm/practica4/practica4$ docker system df
TYPE            TOTAL     ACTIVE    SIZE      RECLAIMABLE
Images          2         2         1.457GB   0B (0%)
Containers      2         1         247B      67B (27%)
oslab@oslab:/media/sf_sharedfoldervm/practica4/practica4$ docker ps -s -a
CONTAINER ID   IMAGE     COMMAND         ...                  SIZE
9f0981236388   ubuntu    "/bin/bash"     ...               67B (virtual 77.8MB)
c3ac462a12a3   gcc       "bash"          ...               180B (virtual 1.38GB)
```

180B correspon al contenidor amb gcc, i 67B correspon al que no disposa de gcc.

2.2.3 Mostra el contingut del teu DockerFile.

Hem creat nostre DockerFile a partir del Dockerfile vist anteriorment a *experiments* eliminant la còpia del contingut, afegint la línia corresponent a la instal·lació de vim i python2.7 i utilitzant el paquet d'Ubuntu.

```
FROM ubuntu:latest
RUN addgroup --gid 1000 appgroup
RUN useradd -r --uid 1000 -g appgroup appuser
RUN mkdir /home/appuser
RUN chown appuser:appgroup /home/appuser
RUN apt-get update -y && apt-get install -y vim python2.7
WORKDIR /home/appuser
USER appuser
```

Podríem haver fet un DockerFile més senzill:

```
FROM ubuntu:latest
WORKDIR /home/appuser
RUN apt-get update && apt-get install -y python2 vim
```

2.2.4 Quines instruccions has usat per 1) generar la imatge, 2) carregar-la a Docker Hub i 3) comprovar si "Python 2" i "vi" s'han instal·lat?

Un cop hem tingut el Dockerfile preparat hem realitzat les següents comandes:

(1) `docker build -t myp2image .`: amb aquesta hem generat la imatge a partir del Dockerfile.

```
oslab@oslab:/media/sf_sharedfoldervm/docker$ docker build -t myp2image .
...
oslab@oslab:/media/sf_sharedfoldervm/docker$ docker images
REPOSITORY   TAG       IMAGE ID       CREATED         SIZE
myp2image    latest    eacbd373ccb1   2 minutes ago   205MB
...
```

(2) `docker tag myp2image mmartiga209/myp2image`: primer de tot hem assignat una etiqueta i amb `docker login` iniciem sessió. Per acabar, amb `docker push mmartiga209/myp2image` ja tenim la imatge pujada.

(3) `python2 --version, vi --version`: amb aquestes dues comandes podrem comprovar les respectives instal·lacions (havent executat una imatge amb `docker run -it myp2image`).

```
appuser@78e2f1a8bd09:~$ python2.7 --version
Python 2.7.18
appuser@78e2f1a8bd09:~$ vim --version
VIM - Vi IMproved 8.2 (2019 Dec 12, compiled Oct 06 2023 07:49:43)
...
```

2.2.5 Mostra les instruccions que has usat per descarregar i executar la teva imatge `myp2image` des de Docker Hub. Quines són les possibles aventatges de tenir les nostres imatges en repositoris com Docker Hub?

Per descarregar i executar la nostra imatge hem executat les comandes `docker pull mmartiga209/imatge, docker run -it mmartiga209/imatge`:

```
oslab@oslab:/media/sf_sharedfoldervm/docker$ docker pull mmartiga209/myp2image
...
docker.io/mmartiga209/myp2image:latest
oslab@oslab:/media/sf_sharedfoldervm/docker$ docker images
REPOSITORY              TAG       IMAGE ID       CREATED          SIZE
mmartiga209/myp2image   latest    eacbd373ccb1   25 minutes ago   205MB
oslab@oslab:/media/sf_sharedfoldervm/docker$ docker run -ti mmartiga209/myp2image
appuser@54ed50acb9d3:~$ python2.7 --version
Python 2.7.18
```

Les possibles avantatges de tenir les nostres imatges en repositoris com aquest són l'accés general, la gestió centralitzada i el versionat de les imatges així com la fàcil implementació i escalabilitat.

3 Conclusions i valoracions personals

Aquesta pràctica ens ha semblat molt útil per consolidar els continguts que hem après a classe. Hem après totes les comandes bàsiques i la informació més rellevant sobre Docker, i hem explorat aplicacions pràctiques que han millorat la nostra comprensió. Hem après a utilitzar la documentació de Docker i ara tenim una eina addicional que segurament utilitzarem en el futur.

Creiem a més a més que el nivell de dificultat de la pràctica no és excessiu i el temps invertit en la seva realització és correcte. On hem tingut més dificultats ha sigut en la resposta de l'exercici 2.2.1.

4 Contribució dels membres del grup

Per acabar aquest informe, comentar que hem treballat molt bé en equip, ens hem ajudat mútuament i ens hem repartit les tasques a fer de manera equitativa, mentres anàvem realitzant els exercicis proposats al mateix temps.