UNIVERSITAT DE BARCELONA

Sistemes Operatius II

PRÀCTICA 1: Docker

Alumnes:

Javier González Béjar
Alejandro Guzmán Requena

NIUBs:

20329750
20392610

ÍNDEX

1. Introducció:

La virtualització és un mecanisme que té com a objectiu oferir més flexibilitat de la qual disposa un Sistema Operatiu.

Quan es virtualitza un sistema, la seva interfície i recursos es mapejen a la interfície i recursos del sistema real, de manera que aquest simula un altre sistema diferent. D'aquesta manera, hi existeixen eines que permeten a l'usuari controlar l'espai en què s'executen els processos.

A Linux, aquestes eines s'anomenen contenidors, i poden controlar coses com l'espai de directoris/fitxers als quals poden accedir, el nombre màxim de CPUs que poden utilitzar, la memòria RAM màxima que poden utilitzar o el nombre màxim de processos que es poden executar a un contenidor.

Una d'aquestes eines d'alt nivell s'anomena Docker, i és la que utilitzarem a la següent pràctica.

2. Treball realitzat:

2.1. Experiments amb els contenidors.

2.1.1. Docker: hello-world

El primer exercici és molt senzill, l'únic que realitzem és descarregar la imatge de hello-world des de DockerHub i el sistema ens avisa que tot està en ordre després d'executar la comanda

```
docker run hello-world
```

Comanda utilitzada: *docker run hello-world*

```
Hello from Docker!
This message shows that your installation appears to be working correctly.
```

Il·lustració 1: Hello world a la terminal des de Docker

2.1.2. Docker: aplicació statistics

Comandes utilitzades

```
docker build -t statistics . # Construir la imatge del docker
```

```
docker run -ti statistics # Executar la imatge i generar el contenidor
```

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

Resposta: Amb la comanda següent: *docker start [OPTIONS] CONTAINER [CONTAINER...]*

Per saber l'id del contenidor que ens interessa executarem la comanda docker ps –a per llistar els contenidors que s'estan executant i que han sigut aturats recentment.

```
CONTAINER ID   IMAGE          COMMAND                  CREATED       STATUS                          PORTS     NAMES
cc91c384c37b   bindmount2     "bash"                   3 hours ago   Exited (0) About an hour ago              bindmount2
242318c11419   bindmount      "bash"                   3 hours ago   Exited (127) 3 hours ago                  bindmount
098bcc93cf4a   bindmount3     "bash"                   8 hours ago   Exited (127) 3 hours ago                  bindmount3
056255584ea8   e62531de2d91   "/bin/sh -c 'adduser…"   8 hours ago   Exited (1) 8 hours ago                    mystifying_shannon
48d5fd0ce8ec   e62531de2d91   "/bin/sh -c 'useradd…"   8 hours ago   Exited (127) 8 hours ago                  infallible_mendeleev
```

Il·lustració 2: Llista de tots els contenidors que s'estan executant o han sigut aturats recentment

2.1.3. Docker: fork-bomb

Comandes utilitzades

```
docker build -t fork-bomb # Construir la imatge del docker:

docker run --ulimit nproc=32:64 --cpus 1 -ti fork-bomb # Executar la imatge i generar el contenidor (limitant nombre de CPUs i processos màxims)

./fork-bomb & # Executar el programa fork-bomb

ctop # Llistar contenidors actius

docker container ls

docker kill <container> # Matar contenidor conflictiu
```

Pregunta: Quants processos fork-bomb s'estan executant dins del contenidor? Com ho compteu? Podeu fer servir instruccions de la línia de comandes (p.ex.bash) per saber-ho?

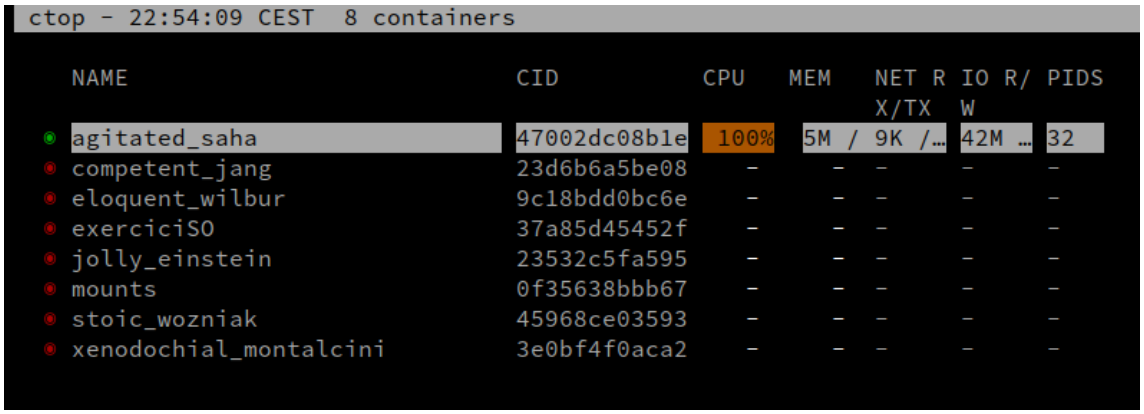
El nombre de processos que s'estan executant dins del contenidor és de 32.

Ho podem saber gràcies a la darrera columna (PIDS) de la taula de contenidors que apareix en executar la comanda ctop- També és el nombre màxim de processos que es poden executar, tal que s'indica amb *–ulimit nproc=32:64.*

Amb la comanda *docker top [NOM DEL DOCKER]* també podem veure també el nombre de processos que s'estan executant.

Pregunta: Quanta CPU està utilitzant el contenidor? Està utilitzant 1, 2 o més CPUs? Per mirar-ho tingueu en compte que en un ordinador amb 4 CPUs el màxim de CPU que es pot ocupar és d'un 400%.

Està emprant el 100% de CPU, de manera que només 1 CPU, tal i com havíem especificat amb *docker run --ulimit nproc=32:64 --cpus 1 -ti fork-bomb*



Il·lustració 3: Número de contenidors creats, amb els processos actius

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?

Aquestes comandes donen errors perquè s'està utilitzant ja el 100% de la CPU en aquest contenidor, i el nombre de processos màxim tal com s'ha especificat, de manera que ja no es pot executar cap procés més.

2.1.4. Docker: connexió via socket

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? Per què succeeix ?

El que succeeix és que dona error al fer bind(), i això succeeix perquè el servidor s'intenta obrir al mateix port que l'anterior (un port ja ocupat) i cada port només pot ser utilitzat per una aplicació o un servidor en particular. Només un únic procés pot escoltar en un port determinat per a la mateixa adreça IP i el mateix protocol, perquè en cas contrari el sistema operatiu no sabria a quin procés enviar el paquet rebut. Llavors l'error ocorre perquè el port 5000 ja està ocupat per l'altre server.

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?

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

Si no es fa servir la opció -p, la terminal espera que s'especifiqui la imatge d'un servidor (el que acabem de crear en el nostre cas), de manera que envia un error.

La pàgina web on trobem la documentació del Docker és: *https://docs.docker.com*

El subapartat on trobem la descripció del funcionament d'aquesta opció és:

https://docs.docker.com/engine/reference/commandline/run/

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 és beneficiós que aquests serveis associats estiguin aïllats entre sí, és a dir, que siguin independents. D'aquesta manera és menys probable que es produeixin errors que comprometin l'aplicació pare, ja que els serveis es troben en entorns predictibles.

A més, aquests contenidors admeten dependències de software que els serveis puguin necessitar-ne, com per exemple biblioteques.

Així, en estar els serveis aïllats es garanteixen espais d'execució independents i segurs.

2.2. Exercicis: Pautes pel desenvolupament d'aplicacions

2.2.1. Compilació i execució sense contenidor:

Comandes utilitzades

```
Make # Compilar codi

./mainSave # Executar arxius

./mainRecc 1 2207774
```

2.2.2. Etapa de desenvolupament: Compilació i execució en un contenidor

Exercici: 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 des 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.

Comandes utilitzades

```
docker build -t bindmounts # Construir la imatge del docker amb el compilador de gcc inclòs com a capa pare.
```

```
docker run -ti --name mounts –mount type=bind,source=/home/oslab/Desktop/practica4,target=/home/appuser/practica4 bindmounts # Per a executar el contenidor fent un bind mount de manera que enllacem directori host i extern
```

```
appuser@debd0debaa3d:~/practica4$ ls
Makefile  Readme.txt  data  dataStructures  mainRecc.c  mainSave.c  model
appuser@debd0debaa3d:~/practica4$ make
gcc -O0 -Wall -c mainSave.c
gcc -O0 -Wall -c model/CSVReader.c -Imodel -o model/CSVReader.o
gcc -O0 -Wall -c model/Movie.c -Imodel -o model/Movie.o
gcc -O0 -Wall -c model/User.c -Imodel -o model/User.o
gcc -O0 -Wall -c model/RecommendationMatrix.c -Imodel -o model/RecommendationMat
rix.o
gcc -O0 -Wall -c model/HashTableDiskService.c -Imodel -o model/HashTableDiskServ
ice.o
gcc -O0 -Wall -c model/MatrixDiskService.c -Imodel -o model/MatrixDiskService.o
gcc -O0 -Wall -c dataStructures/set.c -IdataStructures -o dataStructures/set.o
gcc -O0 -Wall mainSave.o model/CSVReader.o model/Movie.o model/User.o model/Reco
mmendationMatrix.o model/HashTableDiskService.o model/MatrixDiskService.o dataSt
ructures/set.o -o mainSave -lm
gcc -O0 -Wall -c mainRecc.c
gcc -O0 -Wall mainRecc.o model/CSVReader.o model/Movie.o model/User.o model/Reco
mmendationMatrix.o model/HashTableDiskService.o model/MatrixDiskService.o dataSt
ructures/set.o -o mainRecc -lm
appuser@debd0debaa3d:~/practica4$ ./mainSave
appuser@debd0debaa3d:~/practica4$ ./mainRecc 2 1
The number of users that have seen the movie 1 is 547
appuser@debd0debaa3d:~/practica4$
```

Il·lustració 4: Compilació i execució del codi des de l'interior

```
# Dockerfile reference
# https://docs.docker.com/engine/reference/builder/#format
# Es fa servir un usuari amb ID 500, diferent de oslab, per evitar
# que es mostri el missatge d'error "resource temporarily unavailable"
# Veure https://docs.docker.com/engine/reference/commandline/run/

FROM gcc
RUN addgroup --gid 1000 appgroup
RUN useradd -r --uid 1000 -g appgroup appuser
RUN mkdir /home/appuser
RUN mkdir /home/appuser/practica4
RUN chown appuser:appgroup /home/appuser
WORKDIR /home/appuser
USER appuser
```

Il·lustració 5: Dockerfile emprat per a aquest exercici

Com es comprova a la imatge anterior, és possible compilar des de l'interior del contenidor.

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

Això ho solucionem canviant el uid del user de manera que coincideixi amb el del grup, per tal de que posseeixi permisos d'escriptura, tal i com es pot veure en el *Dockerfile* emprat per a aquest exercici (veure arxius adjunts).

2.2.3. Etapa de producció: execució en un contenidor petit

Exercici: 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?

Se'ns demana compilar dins d'un contenidor amb una imatge de recursos mínims el codi executable prèviament compilat de la pràctica 4. Llavors em cercat sobre les imatges amb recursos mínims i partint del *Dockerfile gcc* buscant la capa pare de la capa pare successivament fins a obtenir el *Dockerfile* amb *FROM scratch*, on *scratch* és una imatge buida. El fil conductor que hem seguit ha sigut el següent:

FROM gcc → FROM buildpack-deps:bullseye → FROM buildpack-deps:bullseye-scm → FROM buildpack-deps:bullseye-curl → FROM debian:bullseye → FROM scratch

Un cop arribem a la imatge buida, l'únic que hem fet ha sigut modificar el nostre *Dockerfile* perquè enlloc d'importar des de *gcc*, importi des de *debian:bullseye* els recursos mínims per tal de poder executar el codi.

```
# Dockerfile reference
# https://docs.docker.com/engine/reference/builder/#format
# Es fa servir un usuari amb ID 500, diferent de oslab, per evitar
# que es mostri el missatge d'error "resource temporarily unavailable"
# Veure https://docs.docker.com/engine/reference/commandline/run/

FROM debian:bullseye
RUN addgroup --gid 1000 appgroup
RUN useradd -r --uid 1000 -g appgroup appuser
RUN mkdir /home/appuser
RUN mkdir /home/appuser/practica4
RUN chown appuser:appgroup /home/appuser
WORKDIR /home/appuser
USER appuser
```

Il·lustració 6: Dockerfile emprat per a aquest exercici

Anem a comprovar que efectivament funciona correctament el Dockerfile modificat. Llavors construïm un nou Docker a partir utilitzant la comanda corresponent, i realitzem el bind mount de la mateixa forma que a l'apartat anterior utilitzant docker build -t bindmount .

```
docker run -it --name bindmount --mount
type=bind,source="$(pwd)"/practica4,target="/home/appuser/practica4"
bindmount
```

Una altre opció és crear una imatge basada en el sistema operatiu més senzill a DockerHub, es a dir, *alpine* amb el tag 3.14 que pesa uns 5MB. El problema es que alpine no conté les llibreries necessàries per a executar el codi així que s'han d'afegir diverses dependències extres.

2.2.4. Etapa de producció: execució en un contenidor fent servir volums

En l'últim exercici de la pràctica compilarem i executarem el codi que hem estat utilitzant en els exercicis d'aquest apartat però amb la particularitat que ho farem dins d'un volum.

El que a continuació farem és crear un volum i copiar tot el necessari (executable, dades, ...) del directori local dins del volum.

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

```
docker volume create vol_practica4 # Creem el volume en el docker  (un directori gestionat per Docker) amb el nom de vol_practica4.

cd practica4 # Anem al directori de la practica4

docker container create –name temp –v vol –practica4:/data busybox

# Creem un container temporal a partir de la imatge de busybox (Una distribució lleugera de Linux)

docker cp –a . temp:/data # Copiem el contingut de practica4 a data (el volume) i com aquestes dades persisteixen el volum ja conté les dades que voliem

docker rm temp # Esborrem el container temporal
```

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 muntant el volum amb la comanda

```
docker run \ --mount type=volume,src=vol-practica4,target=/home/appuser/practica4 \ practica4 [comanda] [paràmetres de la comanda]
```

Per exemple per executar mainRecc posarem:

```
docker run \ --mount type=volume,src=vol-practica4,target=/home/appuser/practica4 \ practica4 ./mainRecc 1 2207774
```

En aquest directori es troba el contingut de practica4 que hem muntat com a *volume*. El propietari d'aquests arxius es el *root* però es pot executar el codi amb altres usuaris (appuser).

3. Conclusions:

Per a concloure, és important destacar-ne les avantatges tècniques que comporta Docker. Principalment, permet crear entorns de treball aïllats per a serveis, evitant comprometre i afectar altres processos en cas d'error.

Els Dockers actualment ens permeten separar les aplicacions de la infraestructura per poder oferir programari de forma eficaç. Amb Docker, podem gestionar la infraestructura de la mateixa manera que gestionem qualsevol aplicació. Aprofitant les metodologies de Docker, tenim l'avantatge de poder enviar, provar i desplegar codi ràpidament, podem reduir significativament el retard entre escriure codi i executar-lo a la producció, tal i com hem vist en els darrers exercicis de la pràctica.

Finalment, podem extreure que Docker és essencialment un conjunt d'eines que permet als desenvolupadors construir, executar i aturar contenidors mitjançant ordres simples i automatitzades mitjançant una única API.

A nivell personal, pensem que la pràctica ha estat interessant per a descobrir aquesta eina tant significativa, tot i que algunes preguntes ens han presentat dificultats a l'hora d'entendre-les sobretot a l'inici. Per sort, a classe de pràctiques el professor no trigava a resoldre-les.