Docker

Setembre 2021

Índex

1 Introducció

La Virtualització a nivell d'SO (OS-level virtualization) és un paradigma en què el sistema operatiu permet crear espais d'usuaris aïllats per als processos que s'hi executen. Una de les primeres eines que es va desenvolupar és el chroot (any 1979). Aquesta eina permetia que un procés només pogués accedir als fitxers que hi havia dintre d'un determinat subdirectori. Aquestes eines han rebut recentment un interès més alt en desenvolupar-se mecanismes de forma en què es pot controlar l'espai d'usuari en què s'executen els processos.

A Linux es va incorporar al 2013 la funcionalitat necessària per fer-ho factible, adoptant el nom de "contenidor" a cadascuna d'aquestes instàncies. Entre altres coses, es poden controlar

- L'espai de directoris/fitxers als quals poden accedir
- El nombre màxim de CPUs que poden utilitzar
- La memòria RAM màxima que poden utilitzar
- El nombre màxim de processos que es poden executar a un contenidor

- Moltes altres coses...

Els contenidors estan gestionats internament pel sistema operatiu. A l'actualitat existeixen múltiples eines que permeten utilitzar els contenidors. Una d'aquestes eines és el Docker, que és el que farem servir en aquesta pràctica.

Docker és una eina d'alt nivell que permet manipular de forma senzilla els contenidors. Aquesta aplicació té una gran versatilitat per manipular els contenidors. Aquí veurem algunes de les coses que es pot fer amb aquesta aplicació.

2 Experiments amb els contenidors

Per fer els següents experiments farà falta connexió a la xarxa ja que el Docker baixarà del repositori les imatges dels contenidors necessàries.

2.1 Docker hello-world

Comencem per comprovar que l'aplicació funciona correctament. Executeu la següent comanda:

```
docker run hello-world
```

Aquesta comanda farà que es baixin del repositori les imatges necessàries per crear el contenidor en què s'executarà una aplicació. Aquesta aplicació imprimirà un missatge per pantalla de "Hello world". Si us apareix aquest missatge tot funciona bé.

Podeu provar d'executar múltiples vegades l'anterior comanda. Veureu a partir de la segona vegada s'executa més ràpid que la primera vegada ja que la imatge a partir de la qual es crea el contenidor ja s'ha baixat.

2.2 Docker: aplicació statistics

A continuació executarem un codi senzill dintre del Docker. El codi associat és el fitxer `statistics.c`. Abans d'executar el codi dintre del Docker, comproveu el bon funcionament del codi. Trobareu el codi dintre del directori fitxers. Compileu i executeu el codi per assegurar el bon funcionament del codi.

A continuació compilarem i executarem aquest codi dintre d'una imatge Docker. Per construir la imatge s'acostuma a fer servir l'anomenat fitxer `Dockerfile`. Aquest fitxer inclou les regles per construir una nova imatge a partir d'una imatge base, la qual està especificada al principi d'aquest fitxer. La imatge base pot tenir dependències amb altres imatges, les quals es baixaran automàticament.[1].

Per construir la nova imatge, que anomenarem `statistics`, cal executar

```
docker build -t statistics .
```

El primer cop que s'executi aquesta instrucció pot trigar una mica en construir la nova imatge ja que Docker haurà de baixar del repositori les imatges necessàries per poder executar el codi.

Un cop construïda la nova imatge la podem executar. És en aquest moment en què es crea un contenidor! Intuïtivament, podeu interpretar que una imatge i un contenidor són conceptualment

[1] Els detalls de les instruccions que es poden fer servir al fitxer Dockerfile es poden trobar a https://docs.docker.com/engine/reference/builder/

equivalents a un programa i un procés, respectivament. El programa és el fitxer executable que hi ha a disc, i el contenidor és un programa en execució.

Executarem la nova imatge de forma que puguem executar el contenidor de forma interactiva

```
docker run -ti statistics
```

Observar que ara podem executar instruccions com `pwd`, `whoami`, ... Podeu comprovar també quina és la versió del compilador gcc que hi ha al contenidor amb `gcc --version`. Compareu-ho amb la versió del compilador que hi ha instal·lada per defecte a la màquina en què esteu executant el Docker.

Dins del directori veurem el codi font `statistics.c`. Podem compilar el codi executant `make`. Finalment, podem executar el codi. També podem executar altres instruccions com, per exemple, `wc`, o també podem editar el codi amb `vi`. Tot dintre del contenidor, en cap moment podem "veure" què és el que hi ha fora del contenidor. Més endavant veurem com podem donar accés al contenidor per poder escriure fora del contenidor.

Es pot sortir del contenidor executant la comanda `exit`. Aquesta instrucció surt del contenidor i no el destrueix. Dit d'altra forma, el contenidor queda "aturat".

Executem de nou la instrucció

```
docker run -ti statistics
```

Observeu si queda alguna cosa del que heu fet abans. Hi ha l'executable, per exemple? Veureu que no queda res. Això és perquè la instrucció `run` crea un nou contenidor cada cop que es fa servir la comanda `run` del `docker`.

Per "despertar" contenidors "aturats" consulteu la documentació del `docker`. Per això podeu mirar la documentació associada als contenidors que docker està gestionant

```
docker container
```

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

2.3 Docker: fork-bomb

Ara executarem el fork-bomb dintre d'un contenidor. El fork-bomb s'executarà limitant el nombre de processos que es poden crear dintre del contenidor, així com el nombre de processadors que es podran utilitzar per executar el fork-bomb.

Tots els fitxers necessaris es troben dins del director `fork-bomb`. Per generar la imatge necessària executem

```
docker build -t fork-bomb .
```

per executar la imatge i generar el contenidor fem

```
docker run --ulimit nproc=32:64 --cpus 1 -ti fork-bomb
```

En executar la imatge `fork-bomb` especifiquem el nombre màxim de CPUs que es faran servir per executar el contenidor així com el nombre màxim de processos que es poden executar dins del contenidor. Mireu la documentació d'ulimit!

En executar la comanda anterior executem en mode interactiu. Podeu executar qualsevol comanda disponible dintre del contenidor. Ara es tracta d'executar el fork-bomb! Per això compileu el codi i executeu fent

```
./fork-bomb &
```

Anem a veure quantes CPUs així com quants processos s'estan executant al contenidor. Per fer-ho es farà servir l'aplicació `ctop`, una aplicació equivalent a `top` que està dissenyada per analitzar la càrrega dels contenidors que s'executen al nostre ordinador. El codi font està disponible a https://github.com/bcicen/ctop/releases. A la màquina virtual disposeu del codi compilat corresponent (només fa falta que executeu `ctop`).

Obriu un altre terminal (fora del contenidor) i executeu l'aplicació `ctop`. Es demana

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

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%.

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 dóna el bash un missatge d'error?

Per tal d'aturar el contenidor es pot procedir de la següent forma. Primer de tot, llistem els contenidors actius (i.e. aquells que s'estan executant en aquell moment)

```
docker container ls
```

Un cop identificat el contenidor "conflictiu", el podem matar amb la instrucció

```
docker kill <container>
```

on hem d'indicar a `<container>` l'identificador del contenidor a matar. Observar que aquest cop es mata el procés principal (el que va iniciar el fork-bomb).

Com es pot veure, Docker permet executar aplicacions de forma segura limitant l'espai d'usuari que li assignem.

2.4 Docker: connexió via socket

Els contenidors s'utilitzen sovint en servidors per tal d'executar cadascuna de les aplicacions (i.e. serveis) en un contenidor diferent. D'aquesta forma s'assegura que els serveis no s'interfereixin entre sí.

Execució sense contenidors

Comencem amb un exemple de client-servidor que no utilitza contenidors. En aquest exemple no és objectiu que entengueu com es programa la comunicació entre el client i el servidor, sinó veure la utilitat dels contenidors per executar diferents serveis en diferents contenidors.

El codi està emmagatzemat al directori `socket-wo-docker`. Obriu un terminal i compileu el client i el servidor.

```
gcc socket_server.c -o socker_server
gcc socket_client.c -o socket_client
```

Executeu el servidor en un terminal. Veureu que apareix un missatge per pantalla en què el servidor indica que està escoltant connexions dels clients al port 5000.

A continuació obriu un altre terminal i executeu el client indicant-li la direcció IP i el port (el 5000) al qual us voleu connectar

```
./socket_client 127.0.0.1 5000
```

La direcció IP 127.0.0.1 significa "aquest ordinador", i 5000 és el port al qual s'ha de connectar. El client us demanarà introduir una cadena. Aquesta cadena s'enviarà al servidor, el qual transformarà les minúscules a majúscules, i retornarà la cadena al client, el qual imprimirà el cadena per pantalla. Ja hem fet el primer experiment sense contenidors!

Ara farem un segon experiment en què hi hauran dos servidors escoltant al port 5000.

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?

Pregunta Per què succeeix?

Per poder gestionar el problema anterior podríem editar el codi font dels dos servidors i fer que cadascun es connecti a un port diferent. Suposem, però, que treballem amb aplicacions de les quals només disposem de l'executable i no es pot (o és complicat) canviar els ports en què les les aplicacions escolten.

Execució amb contenidors

A continuació s'executaran dos contenidors, un per a cada servidor. Tots dos escoltaran, dins del contenidor, en el port 5000. Mitjançant els contenidors es maparà, en el primer contenidor, el port 5000 (intern) al port 3000 a l'exterior. Al segon contenidor es maparà el port 5000 (intern) al port 4000 a l'exterior. Des de l'exterior del contenidor ens podrem connectar al port 3000 o 4000 per accedir a un o altre servidor.

Teniu els fitxers necessaris al directori `socket-docker`. Al seu interior trobareu els dos servidors amb els fitxers README corresponents per generar i executar cadascun dels servidors. Observar que, en executar cadascun dels contenidors, es fa el mapat de ports corresponents.

Finalment, executeu el client. La direcció IP és la mateixa d'abans, 127.0.0.1. Si en connectar-vos indiqueu el port 3000, us connectareu al `server1` (que transforma les minúscules a majúscules). En canvi, si indiqueu el port 4000, us connecteu al `server2` (que transforma les majúscules a minúscules).

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?

El Docker permet també que dos contenidors es puguin connectar entre sí fent servir la connexió de xarxa. En aquesta pràctica no s'entrarà en els detalls associats (ho podeu trobar a la documentació!). Aquesta darrera característica és pot fer servir perquè diferents serveis (e.g. base de dades, web, ...)

d'una aplicació més gran s'executin cadascuna en un contenidor diferent (en comptes d'executar-les totes fora del contenidor o dintre d'un únic contenidor). Els serveis es comuniquen entre sí fent servir la connexió.

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?

3 Exercicis: pautes pel desenvolupament d'aplicacions

A l'actualitat el Docker es fa servir de forma que cada aplicació s'executa en un contenidor diferent. En desenvolupar una aplicació (en C, Java, Python, ...) Docker recomana seguir unes pautes per tal de fer-ho (https://docs.docker.com/develop/dev-best-practices/).

En aquesta secció seguirem aquestes pautes per executar la pràctica 4 de Sistemes Operatius 1 (curs 2020/2021) en un contenidor. S'han de tenir en comptes aspectes com

- L'aplicació s'ha de compilar fent servir el compilador que ofereix el contenidor. D'aquesta forma es genera un executable que es podrà executar dins del contenidor atès que és "compatible" amb totes les llibreries que hi ha dins del contenidor.
- En generar l'executable, on deixem l'executable i els fitxers corresponents que es generen? Per exemple, suposem que volem executar en un contenidor l'aplicació que llegeix la base de dades i que genera la matriu de recomanació. Per altra banda, volem executar en un altre contenidor l'aplicació que permet fer recomanacions. És convenient doncs que els dos contenidors comparteixin les dades. O, per què no deixem els executables i totes les dades que es llegeixen i es generen fora dels contenidors on s'executen?
- Un cop haguem generat els executables fa falta que el contenidor on l'executem tingui el compilador? L'ideal seria tenir un contenidor que contingui el mínim necessari per executar-lo. Com ho fem?

Tot això són preguntes que es ens podem fer i que es portaran a terme a continuació.

3.1 Compilació i execució sense contenidors

Començarem per compilar i executar el codi de la pràctica 4 de sistemes operatius 1 del curs 2020/2021 sense fer servir contenidors. En aquesta pràctica no és necessari entendre el codi de la pràctica. Per compilar el codi farem

```
make
```

A continuació executem el codi que llegeix el fitxer CSV i genera la matriu de recomanació matrix.bin, de metadades meta.bin, així com el fitxer de hash.bin.

```
./mainSave
```

Finalment comprovem que el codi funciona correctament

```
./mainRecc 1 2207774
```

El resultat de l'execució hauria de ser "The number of movies seen by the user 2207774 is 57".

3.2 Etapa de desenvolupament: compilació i execució en un contenidor

En aquesta secció ens centrarem en que anomenarem etapa de desenvolupament: l'objectiu és compilar i executar el codi dins del contenidor fent que les dades (codi font, fitxer CSV, fitxers bin) es puguin compartir entre diversos contenidors.

Per fer-ho es faran servir els anomenats "bind mounts" (https://docs.docker.com/storage/bind-mounts/). La idea és fer que el codi font i les dades estiguin en un directori local de l'ordinador i que és fora del contenidor. El contenidor es configura de forma que es permet l'accés al directori on està tot el necessari per fer anar l'aplicació.

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.

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

3.3 Etapa de producció: execució en un contenidor petit

Tenim el codi font compilat i no és necessari tornar-lo a compilar. Podem executar el codi en un contenidor que contingui el mínim necessari per executar el codi. Per altra banda, en comptes de fer servir "bind mounts" es recomana fer servir "volums", un sistema que permet compatir fitxers i que no depèn del sistema operatiu (i.e. sistema de fitxers) en què s'executa l'aplicació.

Comencem pel primer punt. Es tracta de generar una imatge que permeti executar el codi i que no contingui el compilador ja que no és necessari. Per fer-ho hem de tenir en compte l'arquitectura del Docker: en construir una imatge ho fa per "capes". Hi ha una capa "pare" i sobre aquesta capa s'instal·len totes les eines necessàries per executar la imatge. En el nostre cas hi ha una capa "pare" a sobre de la qual s'han instal·lat totes les eines necessàries per poder compilar codi, que ara ja no és necessari ja que el codi ha sigut compilat i no el tornarem a compilar.

Els Dockerfile's es fan servir per construir les diverses capes que formen una imatge. Al fitxer Dockerfile es fa servir la instrucció "FROM" per indicar quina és la capa anterior a partir de la qual es construiran les següents capes. En el nostre cas hi ha posat "FROM gcc". Al web de Docker podrem veure els Dockerfile's corresponents que permeten construir la capa "gcc" (veure https://hub.docker.com/_/gcc).

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?

3.4 Etapa de producció: execució en un contenidor fent servir volums

Docker recomana l'ús de "volums" a l'etapa de producció. Un volum és un sistema d'emmagatzematge amb una funcionalitat superior als "bind mounts", veure https://docs.docker.com/storage/volumes/ .

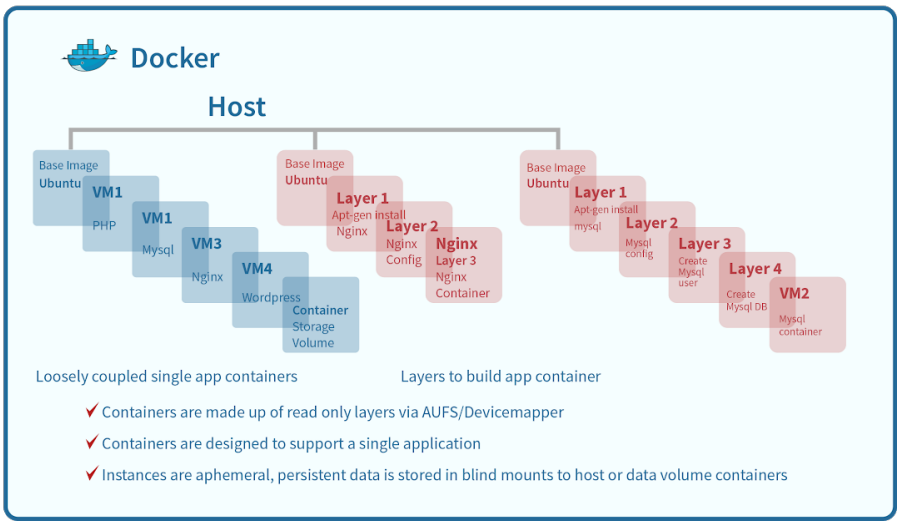

Figura 1: Disseny per capes del Docker, veure https://docs.docker.com/develop/develop-images/dockerfile_best-practices/.

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

```
docker volume create vol-practica4
```

Podem comprovar que el volum ha estat creat

```
docker volume ls
```

El que hem de fer ara es copiar tots els fitxers necessaris de la carpeta `practica4` dintre del volum. Per això es proposa esborrar primer tots els fitxers que no fan falta del directori practica 4 i després es fa una còpia recursiva de tots els fitxers del directori `practica4` dintre del volum `vol-practica4`. Un cop ho hem fet obrim un terminal i executem les següents instruccions:

```
cd practica4
docker container create --name temp -v vol-practica4:/data busybox
docker cp -a .  temp:/data
docker rm temp
```

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

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?

4 Entrega de la pràctica

Es demana entregar un fitxer ZIP que inclogui un informe sobre la feina feta incloent les respostes a les preguntes realitzades al document. Si ho desitgeu, podeu incloure les solucions als exercicis

proposats (els Dockerfile i les instruccions d'execució) de la Secció 3 en fitxers individuals. En tot cas, es convenient que els inclogueu també a l'informe.

El nom del fitxer ha d'indicar el número del grup, seguit del nom i cognom dels integrants del grup (e.g. **P1_Grup07_nom1_cognom1_nom2_cognom2.zip**). L'informe a entregar ha d'estar en format `PDF` o equivalent (no s'admeten formats com `odt`, `docx`, ...).

Un informe està estructurat generalment en una introducció, treball realitzat i conclusions. A la part d'introducció es descriu breument el problema solucionat (i.e. resum del que proposa en aquesta pràctica). A la part de del treball realitzat es responen les preguntes realitzades a la pràctica i es descriuen les proves que s'han realitzat. Per respondre les preguntes podeu seguir un fil conductor que us permeti descriure el treball realitzat. També podeu respondre de forma explícita a cadascuna de les preguntes (en aquest cas indiqueu clarament a quina pregunta esteu responent en cada moment). Sigueu breus i clars en els comentaris i experiments realitzats, no cal que us esteneu en el text. Finalment, a la part de conclusions es descriuen unes conclusions tècniques de les proves realitzades. Per acabar, es poden incloure conclusions personals.

Inclogueu, preferentment en format text, les comandes que heu executat i algun comentari breu descrivint el resultat si ho creieu necessari. En cas que preferiu incloure captures de pantalla en comptes d'incloure el resultat en format text, assegureu-vos que el text de la captura es pot llegir bé (és a dir, que tingui una mida similar a la resta del text del document) i que totes les captures siguin uniformes (és a dir, que totes les captures tinguin la mateixa mida de text).

Adjunteu amb el fitxer ZIP els fitxers Dockerfile necessaris per construir els contenidors associats als exercicis de la Secció 3. Indiqueu a l'informe, de forma resumida, com es construeix el contenidor i com s'ha d'executar el contenidor corresponent. *Eviteu incloure al ZIP el codi de la pràctica 4 atès que el professorat disposa d'ella.*

El document ha de tenir una llargada màxima de 6 pàgines (sense incloure la portada). El document s'avaluarà amb els següents pesos: proves realitzades i comentaris associats, un 60%; escriptura sense faltes d'ortografia i/o expressió, un 20%; paginació del document feta de forma neta i uniforme, un 20%.