

Introducció a Docker

David Fernàndez

Curs 22-23

Índex

Introducció	1
Instal·lació Docker i comprovació funcionament:	2
Gestió d'imatges	3
Gestió de contenidors.....	4
<i>Creació d'un contenidor.....</i>	<i>4</i>
<i>Engegar i aturar un contenidor</i>	<i>5</i>
<i>Comanda tot en un: docker run</i>	<i>5</i>
<i>Execució comandes dins un contenidor</i>	<i>5</i>
<i>Copia d'arxius entre el contenidor i el sistema operatiu amfitrió.....</i>	<i>6</i>
<i>Altres opcions en la creació de contenidors.....</i>	<i>6</i>
Gestió de contenidors: Xarxa	6
Gestio contenidors : Volumes	7
Altres comandes amb Docker: commit, image	8
Imatges personalitzades.	9
<i>Configuració php per depuració remota.....</i>	<i>11</i>
Docker compose.	13
<i>Introducció.....</i>	<i>13</i>
<i>Instal·lació</i>	<i>13</i>
<i>Fitxer docker-compose.yml.....</i>	<i>13</i>
<i>Comandes docker-compose.....</i>	<i>15</i>
<i>Imatge personalitzada en docker-compose.....</i>	<i>16</i>
Dockeritzar una aplicacio node.js	16
Referències:.....	16

Introducció

Actualment hi ha 2 formes principals de virtualització:

- Màquines virtuals.
- Contenedors

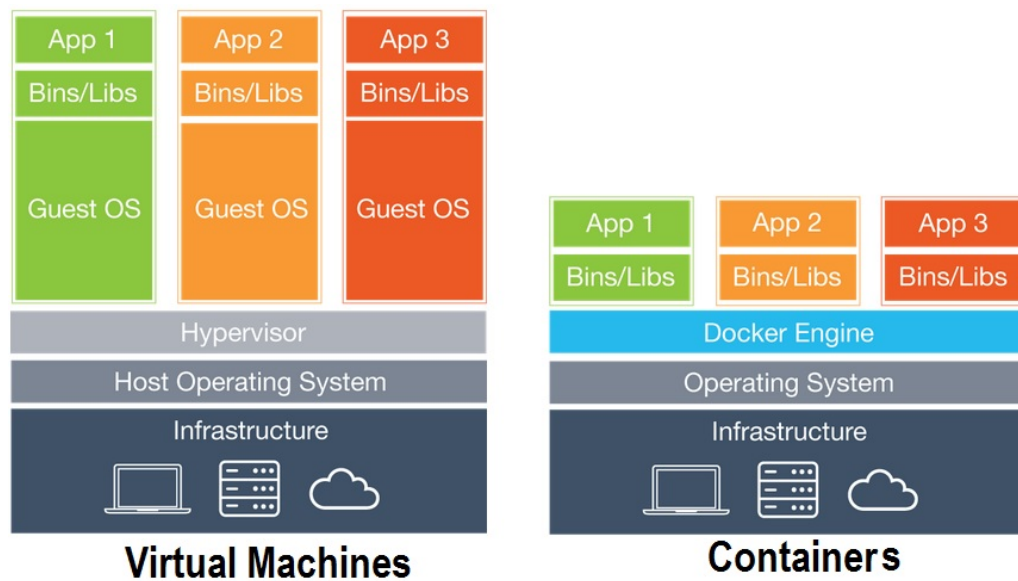
Tenen en comú:

- Tots dos utilitzen imatges com a "plantilla"
- Utilitzant la imatge es poden generar molts sistemes virtualitzats, anomenats màquines virtuals i contenedors, respectivament.
- Les imatges són de només lectura, però el sistema virtualitzat no.

I una diferència principal:

- Un contenidor no té kernel del sistema operatiu, sinó que utilitza el del sistema operatiu que executa el servei de contenedors. És per això que no realitza tota la comprovació del hardware i inicialització del sistema operatiu, com faria una màquina virtual, la qual cosa permet que l'inici de qualsevol servei en un contenidor sigui molt més ràpid que en una màquina virtual.

Actualment, l'estàndard de facto en contenedors és Docker, tot i que hi ha d'altres sistemes com LXC, LXD, etc.



Instal·lació Docker i comprovació funcionament:

Instal·lem Docker i afegim l'usuari al grup Docker

```
sudo apt install docker.io docker-compose
sudo adduser `id -un` docker
```

NOTA: Les imatges es guardaran a `/var/lib/docker`.

Per provar el seu funcionament, podem executar:

```
docker run hello-world
```

Això realitzarà diversos passos tots en un, i mostrarà un missatge com aquest:

```
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
1b930d010525: Pull complete
Digest: sha256:4fe721ccc2e8dc7362278a29dc660d833570ec2682f4e4194f4ee23e415e1064
Status: Downloaded newer image for hello-world:latest

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

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/get-started/
```

Si ens fixem, ens està dient que no ha trobat la imatge localment i

1. El client (la comanda Docker) contacta amb el servei Docker.

D'aquí entenem que tenim un servei o *daemon* Docker funcionant en el nostre sistema, i que és el que s'encarrega de fer tota la feina, i l'usuari, mitjançant el client docker li envia les ordres.

2. Es descarrega la imatge sol·licitada d'Internet.

El lloc on hi ha el repositori de totes les imatges s'anomena Docker Hub, i nosaltres ens podem registrar i pujar les nostres imatges per compartir.

3. El servei, crea un contenidor a partir de la imatge, i aquest executa un programa que produeix el missatge en qüestió.

S'anomena contenidor a un procés que s'executa encapsulat dins el servei de docker. Aquest contenidor es basa en una plantilla (imatge), tot i que com veurem, es pot personalitzar.

4. El servei envia la sortida al client, que la mostra pel terminal.

No serà habitual això, sinó normalment utilitzarem contenidors per executar serveis, i aquests deixen la sortida en arxius de log.

El lloc on estan les imatges s'anomena Docker Hub i el servei Docker les descarrega automàticament.

Recordem la diferència entre imatge i contenidor.

La **imatge** és una plantilla que conté tots els arxius necessaris per executar un procés, per exemple un servidor apache, amb els seus mòduls configurats, o un servidor mysql. No s'executa, però serveix com a base per crear contenidors personalitzats o no, i executar-los.

Els **contenidors** són processos que s'executen aïllats del sistema operatiu que els conté, amb el seu propi sistema de fitxers. Els contenidors executen el software que hi ha dins la imatge, però en un sistema de fitxers propi i aïllat, per tant tots els canvis que realitzin, per exemple, els registres que s'insereixen en un servidor mysql, es queden dins el contenidor, de manera que si s'esborra el contenidor, s'eliminen les dades. Més endavant veurem com preservar les dades i permetre que el contenidor es comuniqui amb la resta de contenidors, o el sistema operatiu.

Gestió d'imatges

Podem veure una llista de les imatges que tenim descarregades en el nostre sistema amb:

```
docker images
```

o Bé

```
root@srv100dfs:[~]>docker image ls
REPOSITORY      TAG          IMAGE ID       CREATED        SIZE
hello-world     latest      feb5d9fea6a5  11 months ago 13.3kB
```

Veurem la imatge del hello-word que s'ha baixat quan hem fet la prova del docker run hello-world

Buscar imatge: docker search: buscarà al Docker Hub imatges amb el nom.
Per exemple: docker search php

```
root@srv100dfs:[~]> docker search php
NAME                                DESCRIPTION                                STARS    OFFICIAL
AUTOMATED
php                                While designed for web development, the PHP ... 6727    [OK]
composer                           Composer is a dependency manager written in ... 905     [OK]
admirer                             Database management in a single PHP file.       748     [OK]
phpmyadmin                         phpMyAdmin - A web interface for MySQL and M... 608     [OK]
mediawiki                          MediaWiki is a free software open source wik... 441     [OK]
webdevops/php-nginx                Nginx with PHP-FPM                            233
[OK]
php-zendserver                     Zend Server - the integrated PHP application... 202     [OK]
yourls                             YOURLS is a set of PHP scripts that will all... 191     [OK]
bitnami/php-fpm                    Bitnami PHP-FPM Docker Image                    150
[OK]
webdevops/php-apache-dev           PHP with Apache for Development (eg. with xd... 147
[OK]
webdevops/php-apache               Apache with PHP-FPM (based on webdevops/php)    127
[OK]
bitnami/phpmyadmin                 Bitnami Docker Image for phpMyAdmin              37
[OK]
circleci/php                       CircleCI images for PHP                          35
webdevops/php-nginx-dev            PHP with Nginx for Development (eg. with xde... 28
[OK]
bitnami/phpbb                      Bitnami Docker Image for phpBB                   25
[OK]
webdevops/php                      PHP (FPM and CLI) service container              24
[OK]
bitnami/phppgadmin                 10
phpcollab/phpcollab               phpcollab is an open source internet-enabled... 7
webdevops/php-dev                 PHP with debugging tools (eg. xdebug)           5
[OK]
newrelic/php-daemon                New Relic's PHP daemon gathers APM data from... 4
cimg/php                           2
pipelinecomponents/php-codesniffer  PHP Codesniffer in a container for gitlab-ci    1
drud/phpmyadmin                   PHPMYADMIN container for ddev                   1
okteto/php                         0
pipelinecomponents/php-linter      PHP parallel linter in a container for gitla... 0
```

Baixar una imatge: docker pull *nomimatge*

Esborrar una imatge: `docker rmi nomimatge`

Cal destacar de les imatges que: totes tenen un identificador, en el cas del hello-world és el **feb5d9fea6a5** el qual es pot utilitzar en comptes del nom, per crear contenidors o esborrar imatges.

També ens fixem que cada nom d'imatge pot anar precedida del símbol `:` i una etiqueta (tag). Per exemple, si descarreguem la imatge de php, amb la comanda pull:

```
root@srv100dfs: [~]> docker pull php
Using default tag: latest
```

Ens estem descarregant la última versió de php que estigui publicada (Default tag: latest), però si anem a la pàgina web on està la informació de la imatge al dockerhub: https://hub.docker.com/_/php ens explicarà quines són les etiquetes alternatives que hi ha, a la secció tags, o bé en la pròpia descripció de la imatge. Per exemple, si volem una versió específica de php sobre una distribució de linux

```
docker pull php:8.1.5-apache-bullseye
```

Després de descarregar-la, podem consultar la llista d'imatges amb `docker images`

```
root@srv100dfs: [/opt]> docker images
REPOSITORY      TAG          IMAGE ID          CREATED          SIZE
php              latest       4d6ad64a864a     2 weeks ago     484MB
php              8.1.5-apache-bullseye 9c92679c1244     3 months ago     458MB
hello-world      latest       feb5d9fea6a5     11 months ago   13.3kB
```

I podem esborrar la del php:latest, posant el nom o el Image ID a la comanda: `docker rmi nomimage o idimage`

```
root@srv100dfs: [/opt]> docker rmi 4d6ad64a864a
Untagged: php:latest
Untagged:
php@sha256:ebd3bdb934e7bbd8343cb5a15ebab85fe299215d29d658a6dcf4ac8705c300cc
Deleted: sha256:4d6ad64a864a4b33455091d2579dcb684ec8824ce23e85354f1f24e3a0a556f8
Deleted: sha256:618245f5a722170c9907ae7598810b394fc3e555f4e7d5b611c6f07d5863059f
Deleted: sha256:9e9dd822af16f8484fbfdaf9a2271cd646371503a0772fe932cd0221e81026aa
Deleted: sha256:d5823abab8d0901d6b1667a41cb89bb15adedf9517ad5d47f2af147c45a2fb95
Deleted: sha256:9e07532c0cc4c1e2c91afb4d731df8ec609a758e02ac20ba3a4ac3ed9dc87550
Deleted: sha256:df1a99d61a55f46cfa9a0c9b6d09040d88c172c1763610d40bd552a8a0c6b211
Deleted: sha256:a3ea16fde0bb3b7126e77919b0a917c93d97a8e59cbbcf82316ff44e93cf2321
Deleted: sha256:8a59ccc406220985199db9b70ec6a38989d7b59761144514727feba05384c937
Deleted: sha256:2cdbe54f2f3a6a2241cde9fba238ff6de3ce1c519b9e3c080825933f8dfb26cd
Deleted: sha256:92a4e8a3140f7a04a0e5a15793adef2d0e8889ed306a8f95a6cfb67cecb5f212
```

Gestió de contenidors

Creació d'un contenidor.

Els contenidors són còpies de les imatges, preparades per executar-se, o bé en execució.

El primer pas per executar una imatge és crear el contenidor amb **docker container create**. Podem crear un contenidor a partir del servidor web apache anterior, amb php:

```
root@srv100dfs: [/opt]> docker container create --name server1 php:8.1.5-apache-bullseye
e425d6218af99e8aacde9f0b33e2b2babf52e339be21f07ed0be8d5c8ec98ab4
```

Els contenidors estan creats, però no estan engegats. podem veure una llista amb **docker container list**, però això només ens ensenya els contenidors engegats. Per veure la llista de tots els contenidors, és **docker container list -a**:

```
root@srv100dfs: [/opt]> docker container list
CONTAINER ID   IMAGE          COMMAND          CREATED   STATUS    PORTS    NAMES
root@srv100dfs: [/opt]> docker container list -a
CONTAINER ID   IMAGE          COMMAND          CREATED          STATUS    PORTS    NAMES
e425d6218af9   php:8.1.5-apache-bullseye  "docker-php-entri...  About a minute ago   Created          server1
```

```
765818b4efce  hello-world  "/hello"  5 months ago
Exited (0) 5 months ago  busy_heyrovsky
```

En aquesta llista podem veure el contenidor que hem creat amb nom `server1`. Si no especifiquéssim un nom amb el modificador `--name`, el sistema ens en crearia un aleatòriament.

Engagar i aturar un contenidor

Per engagar un contenidor ja creat utilitzem la comanda **`docker container start`** i el nom del contenidor o el seu id. Un cop engegat podem comprovar si està funcionant amb un `Docker container list`.

```
root@srv100dfs: [/opt]> docker container start server1
server1
root@srv100dfs: [/opt]> docker container list
```

CONTAINER ID	IMAGE	COMMAND	CREATED
e425d6218af9	php:8.1.5-apache-bullseye	"docker-php-entrypoi..."	7 minutes ago
Up 6 seconds	80/tcp	server1	

Per aturar un contenidor utilitzem la comanda **`docker container stop`**

Esborrar un contenidor: **`docker container rm nom_del_contenidor`**

```
root@srv100dfs: [/opt]> docker container rm busy_heyrovsky
busy_heyrovsky
root@srv100dfs: [/opt]> docker container list -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED
e425d6218af9	php:8.1.5-apache-bullseye	"docker-php-entrypoi..."	19 minutes ago
Up 12 minutes	80/tcp	server1	

Comanda tot en un: `docker run`

Amb **`docker run`** podem crear el contenidor i executar-lo, tot en un. Veureu aquesta comanda molt sovint a Internet, però s'ha de ser conscient dels passos que fa. És mes, si la imatge no la té, també la descarregarà automàticament.

Execució comandes dins un contenidor

Com que un contenidor conté els programes executables que hi ha en la seva imatge, és possible que alguns contenidors continguin un intèrpret de comandes `bash`, `csh` o `ksh`. Podem "entrar" dins d'un contenidor executant la seva shell, amb la comanda `docker exec` (que serveix per executar quelcom dins el contenidor) amb el modificador `-it` per a que creï una consola i l'executi interactivament. Executarem el programa `bash`:

```
root@srv100dfs: [/opt]> docker exec -it server1 bash
root@e425d6218af9: /var/www/html#
```

Ens retorna un "prompt" que correspon al `bash` dins el contenidor. Allà som l'usuari `root`, el nom de l'ordinador és el seu id i per defecte ens situa dins la carpeta `/var/www/html`. Recordem que és un servidor apache.

Dins el contenidor, podem executar les comandes que hi hagi en la imatge, per exemple, no funciona el **ping**, doncs no està instal·lat.

```
root@e425d6218af9: /var/www/html# ping 8.8.8.8
bash: ping: command not found
```

Però la comanda **apt** sí funciona, i per tant podem fer un `apt update` o un `apt install`. Com que es tracta d'un servidor web amb php, probablement hi hagi les eines de gestió de servei web, i d'instal·lació de mòduls d'apache i de php. Cal consultar la documentació al `docker hub`.

Més endavant, quan estudiem la creació d'imatges, veurem com podem personalitzar la nostra instal·lació d'apache amb php per incloure o excloure mòduls, de manera que no calgui fer la instal·lació a cada contenidor que creem.

Aquesta comanda també pot ser útil si volem canviar algun arxiu de configuració, i reiniciar el contenidor per veure els canvis. Compte! si fem això, i esborrem el contenidor, els canvis en els arxius es perdran!, veurem més endavant com preservar aquests canvis i automatitzar les modificacions.

Si no necessitem interactivitat, però volem executar una comanda, podem utilitzar el `-d` en comptes del `-it`. Per exemple, la següent comanda crea un arxiu `test.txt` en la carpeta `tmp`

```
docker exec -d server1 touch /tmp/test.txt
```

Copia d'arxius entre el contenidor i el sistema operatiu amfitrió

De la mateixa manera que podem executar comandes, també podem transferir arxius. Si tenim una pagina web `test.html`, podem copiar-la dins el contenidor:

Arxiu `test.html`:

```
root@srv100dfs:[/opt]>cat test.html
<html>
<body>
<h1>Pagina dins el contenidor</h1>
</body>
</html>
```

Copiar dins el contenidor:

```
docker cp test.html server1:/var/www/html
```

També podem extraure arxius invertint origen i destí:

```
docker cp server1:/var/log/apache2/access.log /tmp/
```

Altres opcions en la creació de contenidors

Cal destacar el modificador

Gestió de contenidors: Xarxa

Fins ara hem vist com operar amb el contenidor, però si és un servidor web, com es comunica amb l'exterior?, que passa si ja tenim un altre servidor apache instal·lat en el sistema operatiu amfitrió?, o si volem utilitzar un altre port?

La documentació en Docker hub ens hauria de dir quins ports exposa la imatge, és a dir, quins ports estarien disponibles dins el contenidor. Tractant-se d'un servidor web, amb una configuració per defecte, hi haurà el port 80 i el 443.

Per tal que el sistema operatiu amfitrió pugui obrir un port i redirigir-lo cap al contenidor, cal "mapejar" un port de amfitrió, a un port del contenidor amb el modificador `-p` o `--publish` en el moment de creació del contenidor.

La sintaxi és `[ip:]hostport:containerport`

Per exemple:

```
docker container create -p 8080:80 --name server2 php:8.1.5-apache-bullseye
```

Crea el contenidor anomenat `server2`, i mapeja el port 8080 del amfitrió, al port 80 del contenidor.

Si fem un `docker start server2` i un `docker container list` veurem el servidor en execució i el port obert.

```
root@srv100dfs:[/opt]> docker container create -p 8080:80 --name server2 php:8.1.5-apache-bullseye
b933a3c150c9082bb517a363ca75af4fef5c5f1874f9be6dad4c08d53062e3c2
root@srv100dfs:[/opt]> docker start server2
server2
root@srv100dfs:[/opt]> docker container list
```


CONTAINER ID	IMAGE	COMMAND	CREATED
STATUS	PORTS	NAMES	
b933a3c150c9	php:8.1.5-apache-bullseye	"docker-php-entrypoi..."	10 seconds ago
Up 4 seconds	0.0.0.0:8080->80/tcp, :::8080->80/tcp	server2	
e425d6218af9	php:8.1.5-apache-bullseye	"docker-php-entrypoi..."	About an hour ago
Up 57 minutes	80/tcp	server1	

Veurem que el server1 no havíem mapejar ports i en canvi al server2 ens indica que el port 8080 és accessible des de qualsevol ip i redirigit al port 80. Podeu consultar la documentació del modificador -p per veure totes les possibilitats de restringir una ip, o bé mapejar una llista de ports

Per accedir al servidor web només cal anar a una de les ip's del sistema operatiu amfitrió al port 8080. Compte!, la imatge vé sense cap arxiu html, ni tan sols el de prova, per tant podem copiar l'arxiu anterior test.html dins el contenidor i accedir-hi amb un navegador:

```
docker cp test.html server2:/var/www/html/
```



Anem a l'adreça del servidor, port 8080 pagina test.html

Com sabem en quina carpeta esta configurat l'apache dins el contenidor? Doncs ho ha d'explicar la documentació de la imatge en el Docker Hub.

Gestio contenidors : Volums

De la mateixa manera que mapejem ports del sistema operatiu al contenidor, també podem mapejar carpetes de dins el contenidor. En concret, la carpeta del contenidor /var/www/html, en mapejarla a una altra carpeta de l'amfitrió, el contenidor la muntarà automàticament. Per tant podem col·locar la nostra pagina web i fer que el servidor web la serveixi.

D'aquesta manera, en estar muntada en el sistema de fitxers de l'amfitrió, en eliminar un contenidor no es perden aquestes dades.

El modificador -v munta un volum del host al contenidor. Per exemple, creem una carpeta i hi situem els arxius index.html i test.html:

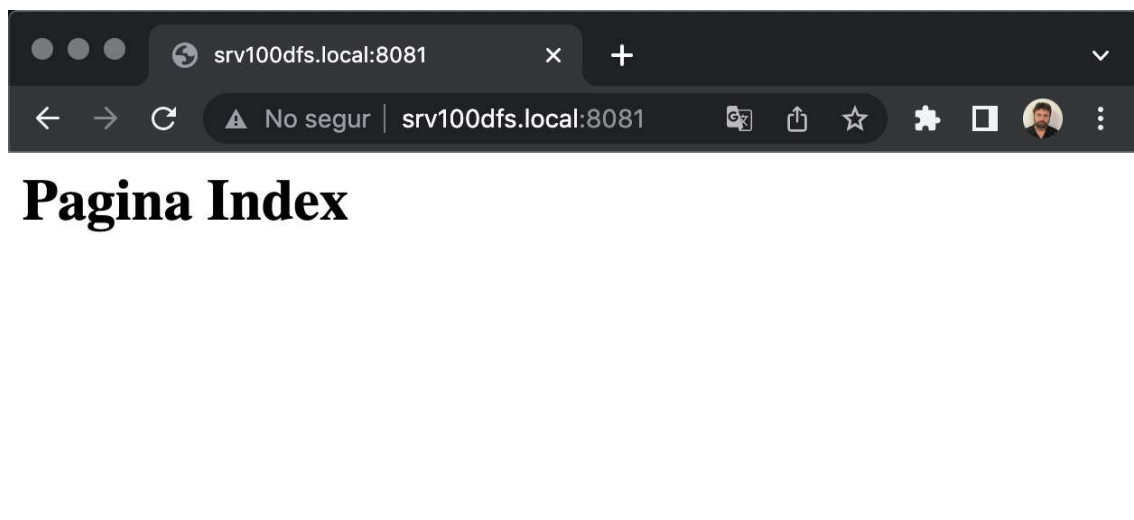
```
root@srv100dfs: [/opt/dockervolumes/server1]>cat index.html
<html>
<body>
<h1> Pagina Index</h1>
</body>
</html>
```

la carpeta estarà a `/opt/dockervolumes/server1` en el sistema operatiu host

Executem la comanda següent per a que docker la munti dins el contenidor: Cal fer-ho en la creació del contenidor, per tant crearem un tercer servidor en un altre port:

`docker container create -p 8081:80 -v /opt/dockervolumes/server1:/var/www/html --name server3 php:8.1.5-apache-bullseye`

```
root@srv100dfs: [/opt/dockervolumes/server3]> docker container create -p 8081:80 -v
/opt/dockervolumes/server1:/var/www/html --name server3 php:8.1.5-apache-bullseye
2f7a457c74dcbf230c010f368b09e83e3d63a4187ae7111b3511a6892515ea69
root@srv100dfs: [/opt/dockervolumes/server3]> docker container start server3
server3
root@srv100dfs: [/opt/dockervolumes/server3]> docker container list
CONTAINER ID   IMAGE                                COMMAND                                CREATED
STATUS        PORTS
2f7a457c74dc   php:8.1.5-apache-bullseye          "docker-php-entrypoi..."           17 seconds ago
Up 8 seconds   0.0.0.0:8081->80/tcp, :::8081->80/tcp server3
b933a3c150c9   php:8.1.5-apache-bullseye          "docker-php-entrypoi..."           41 minutes ago
Up 41 minutes  0.0.0.0:8080->80/tcp, :::8080->80/tcp server2
e425d6218af9   php:8.1.5-apache-bullseye          "docker-php-entrypoi..."           2 hours ago
Up 2 hours    80/tcp                             server1
```



Comprovem que les pàgines se serveixen en el nou servidor web:

Una practica habitual és muntar les carpetes on hi ha els arxius de configuració, de manera que el servei que s'executi, es carregarà a partir d'un arxiu que podem modificar directament, sense entrar dins el contenidor, i els canvis que realitzem es quedaran guardats encara que destruïm el contenidor.

Altres comandes amb Docker: commit, image.

Hem dit que si eliminem un contenidor, esborrem totes les dades que conté. Tot i que hem vist que es poden muntar volums per preservar els canvis, pot ser útil guardar el contenidor, per exemple si hem fet algun canvi manual en la configuració.

El que podem fer és convertir el contenidor en una imatge, per després, tornar-la a desplegar. Després, quan veiem el dockerfile, veurem com podem modificar la imatge per crear-ne una de nova amb la nostra configuració, sense haver de fer-ho manualment.

Per convertir un contenidor a imatge: **docker commit**

Per crear un arxiu tar a partir d'una imatge: **docker image save**

Per crear una imatge a partir d'un arxiu tar: **docker image load**

També important, en el moment de creació d'un contenidor, se li pot especificar la política de reinici, això és, si la aplicació falla, el contenidor es reinicia. La opció és --restart unless-stopped

Imatges personalitzades.

Fins ara hem creat contenidors a partir d'imatges ja fetes per algú altre, i com a molt, ens hem connectat al contenidor, un cop aquest ja està engegat, per modificar algun fitxer dins.

També hem après com muntar volums per guardar les modificacions que va fent el contenidor, i també com podem muntar volums per canviar la configuració d'un contenidor.

Però totes aquestes modificacions, les fem quan el contenidor ja està creat.

En aquest apartat, aprendrem a crear les nostres pròpies imatges ja modificades, llestes per executar la nostra aplicació, amb configuracions personalitzades ja incorporades a la imatge.

Del procés de crear una imatge s'anomena **build**, i la comanda base és `docker build`. Aquesta comanda utilitzarà un arxiu, anomenat **Dockerfile**, que contindrà totes les instruccions per personalitzar la imatge.

Comencem amb un exemple per a crear un servidor apache amb php, amb la nostra web ja desplegada, i els mòduls apache o php instalats i habilitats.

Creem una carpeta `myapp` i la següent estructura:

```
myapp/
├── conf
│   └── myapp.conf
├── Dockerfile
└── src
    ├── index.html
    ├── script.js
    └── style.css
```

La carpeta **conf** contindrà la configuració del nostre site en apache

La carpeta **src** contindrà el codi html, php, javascript etc, de la nostra aplicació.

L'arxiu **Dockerfile** contindrà les següents instruccions per construir la nostra imatge:

```
FROM php:7.4-apache
COPY conf/myapp.conf /etc/apache2/sites-available/myapp.conf
WORKDIR /var/www/html
COPY src/ ./
RUN a2enmod rewrite\
&&a2dissite 000-default \
&&a2ensite myapp
```

Per crear la imatge, executem la comanda (dins la carpeta on està el Dockerfile):

```
docker build -t myapp .
```

**Compte amb el . final, doncs indica on està la carpeta "arrel " on s'executaran les comandes que hi ha al Dockerfile, com ara el COPY.*

Amb aquesta comanda `build`, `docker` buscarà l'arxiu `Dockerfile` dins la carpeta actual, i crearà una imatge amb l'etiqueta `myapp`. Dins l'arxiu `Dockerfile` observem:

La primera línia és el `FROM`: indica de quina imatge base crearem la nostra. D'aquesta imatge base heretarem tota la configuració. Podem consultar al Docker Hub aquesta imatge i quines opcions de personalització tenim.

La ordre COPY serveix per copiar arxius a dins la imatge, de manera que tots els contenidors que creem amb la nostra imatge ja contindran aquests arxius. Cal conèixer l'estructura de la imatge base, llegint la documentació per saber com està configurat i quines carpetes hi ha per defecte.

La ordre WORKDIR situa, dins el contenidor, la carpeta on s'executaran les següents ordres.

L'ordre RUN executa, dins el contenidor, l'ordre que s'especifica. També hem de consultar la documentació al Docker Hub de quines comandes estan disponibles. També és possible que el creador de la imatge hagi creat alguns scripts per realitzar tasques habituals dins la imatge.

En el nostre cas, la imatge php té els scripts **docker-php-ext-configure**, **docker-php-ext-install**, i **docker-php-ext-enable**.

En executar el build veurem:

```
root@srv100dfs: [/opt/dockerbuild/myapp] > docker build -t myapp .
Sending build context to Docker daemon 8.192kB
Step 1/5 : FROM php:7.4-apache
--> 30857d53af4d
Step 2/5 : COPY conf/myapp.conf /etc/apache2/sites-available/myapp.conf
--> Using cache
--> 3e1c3233c88c
Step 3/5 : WORKDIR /var/www/html
--> Using cache
--> efe8ef28d2fa
Step 4/5 : COPY src/ ./
--> Using cache
--> 585d9d99c3c4
Step 5/5 : RUN a2enmod rewrite      && a2dissite 000-default      && a2ensite myapp
--> Running in 87b785d905cf
Enabling module rewrite.
To activate the new configuration, you need to run:
    service apache2 restart
Site 000-default disabled.
To activate the new configuration, you need to run:
    service apache2 reload
Enabling site myapp.
To activate the new configuration, you need to run:
    service apache2 reload
Removing intermediate container 87b785d905cf
--> 95010e6972aa
Successfully built 95010e6972aa
Successfully tagged myapp:latest
```

I creem un contenidor basat en la nostra imatge myapp:

```
docker run -d -p 8008:80 --name myappserver1 myapp
```

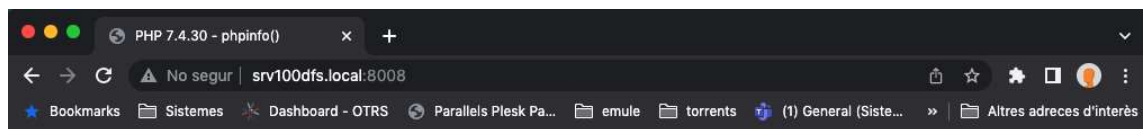
Compte amb aquesta comanda, hem creat i engegat directament el contenidor, anomenat myappserver1, basat en la imatge myapp, i amb el port 8008 mapejat al port 80 del contenidor. Un modificador que no havíem vist fins ara, és el -d. Significa que el contenidor s'engegarà en segon pla (detached). Això no era necessari si fèiem el create i l'start per separat.

Comprovem que el contenidor està funcionant: docker container list

```
root@srv100dfs: [/opt/dockerbuild/myapp] > docker container list
```

CONTAINER ID	IMAGE	COMMAND	CREATED
dda52aa15399	myapp	"docker-php-entrypoi..."	8 seconds ago
Up 6 seconds	0.0.0.0:8008->80/tcp, :::8008->80/tcp	myappserver1	
e425d6218af9	php:8.1.5-apache-bullseye	"docker-php-entrypoi..."	38 hours ago
Up About an hour	80/tcp	server1	

I que els ports estan correctament mapejats i podem accedir a la nostra app a través del port 8008:



Pagina Index MyAPP

PHP Version 7.4.30	
System	Linux e9721fdc79d4 5.4.0-124-generic #140-Ubuntu SMP Thu Aug 4 02:23:37 UTC 2022 x86_64
Build Date	Aug 23 2022 15:23:16
Configure Command	'./configure' '--build=x86_64-linux-gnu' '--with-config-file-path=/usr/local/etc/php' '--with-config-file-scan-dir=/usr/local/etc/php/conf.d' '--enable-option-checking=fatal' '--with-mhash' '--with-pic' '--enable-ftp' '--enable-mbstring' '--enable-mysqld' '--with-password-argon2' '--with-sodium=shared' '--with-pdo-sqlite=/usr' '--with-sqlite3=/usr' '--with-curl' '--with-iconv' '--with-openssl' '--with-readline' '--with-zlib' '--disable-phpdbg' '--with-pear' '--with-libdir=lib/x86_64-linux-gnu' '--disable-cgi' '--with-apxs2' 'build_alias=x86_64-linux-gnu'
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/usr/local/etc/php
Loaded Configuration File	(none)

Configuració php per depuració remota

Podem esborrar el contenidor i tornar a crear la imatge, tot i modificant i afegint alguns arxius. Aquesta configuració habilita la depuració de php dins el contenidor: Instal·la xdebug, l'habilita al php

Primer **creem** l'arxiu **xdebugconfig** amb aquest contingut:

```
[xdebug]
xdebug.mode=develop,debug
#Troba el client a partir dels http headers de la petició
xdebug.discover_client_host=1
xdebug.client_port = 9003
xdebug.start_with_request=yes
xdebug.idekey='key-xdebug'
```

Afegim a la configuració del Dockerfile :

```
RUN apt-get update \
&& pecl install xdebug-3.1.5\
&& docker-php-ext-enable xdebug
COPY conf/xdebugconfig /tmp
RUN cat /tmp/xdebugconfig >> /usr/local/etc/php/conf.d/docker-php-ext-xdebug.ini
```

El que fem amb aquest afegit al Dockerfile es actualitzar la llista de repositoris i instal·lar el mòdul **xdebug**.

Tal com recomanen a la documentació del Docker Hub, és millor especificar la versió del mòdul, doncs la comanda **pecl** no comprova incompatibilitats, i instal·la la última versió.

Després el Dockerfile executa l'script personalitzat del creador de la imatge, el **docker-php-ext-enable**. La configuració de php, en aquesta imatge la tenim a **/usr/local/etc/php/** en comptes de **/etc/php**.

Finalment en el Dockerfile copiem l'arxiu **xdebugconfig** a la carpeta **tmp** del contenidor i **afegim** la nostra configuració a l'arxiu **.ini** que ja hi ha creat.

El Dockerfile quedarà així.:

```
FROM php:7.4-apache
COPY conf/myapp.conf /etc/apache2/sites-available/myapp.conf
WORKDIR /var/www/html
COPY src/ ./
RUN a2enmod rewrite \
&& a2disssite 000-default \
&& a2ensite myapp
```

```

RUN apt-get update \
&& pecl install xdebug-3.1.5\
&& docker-php-ext-enable xdebug

COPY conf/xdebugconfig /tmp
RUN cat /tmp/xdebugconfig >> /usr/local/etc/php/conf.d/docker-php-ext-xdebug.ini

```

Esborrem el contenidor i creem altra vegada la imatge:

```

docker container stop myappserver1&& \
docker container rm myappserver1 && \
docker build -t myapp .&& \
docker run -d -p 8008:80 --name myappserver1 myapp

```

Comprovem que el servidor esta altra vegada engegat, i que ha funcionat l'habilitació del xdebug.

En aquesta línia, hauríem de veure que s'ha processat el .ini del **xdebug**.

PHP Version 7.4.30

System	Linux 8ce51b947875 5.4.0-124-generic #140-Ubuntu SMP Thu Aug 4 02:23:37 UTC 2022 x86_64
Build Date	Aug 23 2022 15:23:16
Configure Command	'./configure' '--build=x86_64-linux-gnu' '--with-config-file-path=/usr/local/etc/php' '--with-config-file-scan-dir=/usr/local/etc/php/conf.d' '--enable-option-checking=fatal' '--with-mhash' '--with-pic' '--enable-ftp' '--enable-mbstring' '--enable-mysqlnd' '--with-password-argon2' '--with-sodium=shared' '--with-pdo-sqlite=/usr' '--with-sqlite3=/usr' '--with-curl' '--with-iconv' '--with-openssl' '--with-readline' '--with-zlib' '--disable-phpdbg' '--with-pear' '--with-libdir=lib/x86_64-linux-gnu' '--disable-cgi' '--with-apxs2' 'build_alias=x86_64-linux-gnu'
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/usr/local/etc/php
Loaded Configuration File	(none)
Scan this dir for additional .ini files	/usr/local/etc/php/conf.d
Additional .ini files parsed	/usr/local/etc/php/conf.d/docker-php-ext-sodium.ini, /usr/local/etc/php/conf.d/docker-php-ext-xdebug.ini
PHP API	20190902
PHP Extension	20190902
Zend Extension	320190902
Zend Extension Build	API320190902.NTS
PHP Extension Build	API20190902.NTS

I que el **xdebug** està habilitat:

xdebug

Version 3.1.5

[Support Xdebug on Patreon, GitHub, or as a business](#)

Enabled Features (through 'xdebug.mode' setting)		
Feature	Enabled/Disabled	Docs
Development Helpers	✓ enabled	<input type="checkbox"/>
Coverage	✗ disabled	<input type="checkbox"/>
GC Stats	✗ disabled	<input type="checkbox"/>
Profiler	✗ disabled	<input type="checkbox"/>
Step Debugger	✓ enabled	<input type="checkbox"/>
Tracing	✗ disabled	<input type="checkbox"/>

Optional Features	
Compressed File Support	no
Clock Source	clock_gettime

Debugger	enabled
IDE Key	key-xdebug

Directive	Local Value	Master Value
xdebug.auto_trace	(setting renamed in Xdebug 3)	(setting renamed in Xdebug 3)
xdebug.cli_color	0	0
xdebug.client_discovery_header	no value	no value
xdebug.client_host	localhost	localhost
xdebug.client_port	9003	9003
xdebug.cloud_id	no value	no value
xdebug.collect_assignments	Off	Off
xdebug.collect_includes	(setting removed in Xdebug 3)	(setting removed in Xdebug 3)
xdebug.collect_params	(setting removed in Xdebug 3)	(setting removed in Xdebug 3)

Docker compose.

Introducció

Docker compose és el següent pas en els contenidors. Es tracta de crear una col·lecció de contenidors que treballen entre ells, sovint en una xarxa virtual privada entre ells.

Utilitzant un arxiu amb sintaxi [yaml](#) que es digui docker-compose.yml, podem definir quins contenidors utilitzar, amb quines imatges, com es diran, i si existeix alguna dependència entre ells.

A part de mapejar ports i muntar volums, Docker també té el concepte de xarxa virtual. Docker compose crea automàticament una xarxa virtual entre tots els contenidors definits.

Instal·lació

Primer caldrà instal·lar Docker-compose:

```
apt install docker-compose
```

Confirmem amb un

```
root@srv100dfs: [/opt/dockerbuild/mycompose] > docker-compose -v
docker-compose version 1.25.0, build unknown
```

Fitxer docker-compose.yml

Un cop instal·lat, creem el primer arxiu docker-compose.yml: i l'analitzem:

```
version: "3"

services:
  web:
    container_name: web
    image: "php:7.4-apache"
    depends on:
      - mariadb
    restart: unless-stopped
    ports:
      - '8888:80'
    volumes:
      - "/opt/dockervolumes/apache/html:/var/www/html"
    links:
      - mariadb:db
  mariadb:
    container_name: mariadb
    image: "mariadb:10.1"
    restart: unless-stopped
    volumes:
      - "/opt/dockervolumes/mysql/data:/var/lib/mysql/"
      - "/opt/dockervolumes/mysql/conf:/etc/mysql/"
    environment:
      TZ: "Europe/Madrid"
      MYSQL_ALLOW_EMPTY_PASSWORD: "no"
      MYSQL_ROOT_PASSWORD: "rootpwd"
      MYSQL_USER: "testuser"
      MYSQL_PASSWORD: "testpassword"
      MYSQL_DATABASE: "testdb"
```

Version: "3": versió del docker-compose.

Services: llista dels contenidors que es crearan, en aquest exemple 2. un anomenat web i l'altre mariadb.

Per cada servei:

container_name: nom del contenidor que es crearà

image: nom de la imatge base que s'utilitzarà. (també es pot especificar una imatge ja creada que sigui preconstruïda, o donar instruccions per construir-ne una (fer el build))

depends_on: i el nom del contenidor que es necessita. Això determinarà l'ordre d'engegada i aturada.

restart: política de reinici. Si s'atura el contenidor, es tornarà a engegar

ports: mapeig de ports per al servei

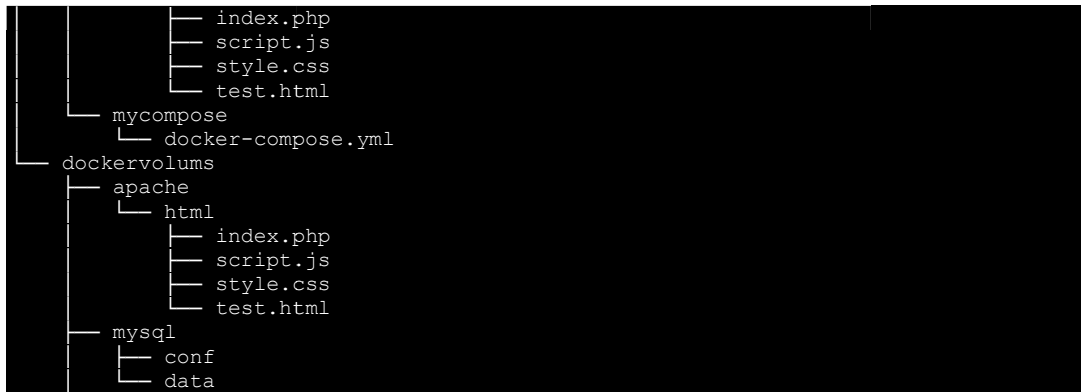
volumes: mapeig de carpetes per al servei

links: ens permet especificar que els contenidors es veuran entre ells, i alhora també un alias per al servei. Un contenidor podrà comunicar-se amb l'altre a partir del nom del servei o del seu àlies. No és obligatori, els contenidors ja es poden comunicar entre ells amb el nom del servei. En aquest exemple podem utilitzar la base de dades des de php **especificant mariadb o db com a hostname**.

environment: Important, aquí hi ha una llista de variables d'entorn que pot necessitar el contenidor per funcionar. En aquest cas, el servidor **mariadb** necessita aquests paràmetres per configurar un password de root, i un usuari i password addicional. Cal consultar la documentació de la imatge, en aquesta cas mariadb, per saber quines variables d'entorn necessitem.

La nostra estructura de carpetes, pel moment és la següent:

```
/opt/
├── dockerbuild
│   ├── myapp
│   │   ├── conf
│   │   │   ├── myapp.conf
│   │   │   └── xdebugconfig
│   │   ├── Dockerfile
│   │   └── src
```

La carpeta **dockerbuild** és on tenim els arxius de configuració per docker build i docker compose.

En el capítol anterior de creació d'imatges personalitzada, hem creat la carpeta **myapp** amb el Dockerfile, src i conf.

En aquest capítol, hem creat la carpeta **mycompose** amb l'arxiu docker-compose.yml

La carpeta **dockervolumes** conté les carpetes on es muntaran les pàgines web (apache/html) i base de dades de mariadb (mysql/data).

Comandes docker-compose

Ordres docker-compose Totes executades **en la carpeta** on estigui el docker-compose.yml

Crear i executar els contenidors (-d per a que funcionin en 2n pla, sinó en fer ctrl+c s'aturen els processos).

```
docker-compose up -d
```

Aturar i engegar els contenidors

```
docker-compose stop
docker-compose start
```

Esborrar els contenidors

```
docker-compose rm
```


Veure els logs: (recomenable mapejar els volums dels contenidors on els seveis deixen el log, per tal de poder-los veure en temps real i analitzar-los, fora del contenidor)

```
docker-compose logs
```

Després de fer un docker-compose up -d comprovem que funciona:



Pagina Index web container des de docker-compose

PHP Version 7.4.30	
	
System	Linux c2bcf520bdc4 5.4.0-124-generic #140-Ubuntu SMP Thu Aug 4 02:23:37 UTC 2022 x86_64
Build Date	Aug 23 2022 15:23:16
Configure Command	./configure '--build=x86_64-linux-gnu' '--with-config-file-path=/usr/local/etc/php' '--with-config-file-scan-dir=/usr/local/etc/php/conf.d' '--enable-option-checking=fatal' '--with-mhash' '--with-pic' '--enable-ftp' '--enable-mbstring' '--enable-mysqlnd' '--with-password-argon2' '--with-sodium=shared' '--with-pdo-sqlite=/usr' '--with-sqlite3=/usr' '--with-curl' '--with-iconv' '--with-openssl' '--with-readline' '--with-zlib' '--disable-phpdbg' '--with-pear' '--with-libdir=lib/x86_64-linux-gnu' '--disable-cgi' '--with-apxs2' 'build_alias=x86_64-linux-gnu'
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled

Amb aquest exemple, les dades del servidor mariadb sempre estan en la carpeta:

`/opt/dockervolumes/mysql/data`

i les pàgines web en

/opt/dockervolumes/apache/html

A l'arxiu docker-compose.yml, hem comentat una línia, (amb hashtag). Correspon a la opció de muntar un segon volum amb la configuració personalitzada de mariadb. Si hi deixem els arxius de configuració de mariadb, els utilitzarà en engegar el servei. El mateix es podria fer per al servidor apache amb php.

Imatge personalitzada en docker-compose

Com potser s'ha observat en el phpinfo, la imatge que hem triat ve únicament amb el driver pdo per a SQLite. Caldria construir una imatge prèviament amb php amb els mòduls ja instal·lats, i utilitzar-la en comptes de la estàndart. Aquesta opció, però requereix construir la imatge prèviament, de manera manual, i tenir-la instal·lada en el nostre sistema.

Hi ha una altra opció i es que el docker-compose faci el build de la imatge si no la troba, especificant el dockerfile corresponent, tal i com hem fet al capítol anterior.

Una recomanació és que utilitzem carpetes separades per cada cosa. Podem utilitzar la carpeta **myapp** per a que el docker-compose contrueixi la imatge del capítol anterior i la utilitzi. Substituïm l'apartat image per:

```
# image: "php:7.4-apache"
build:
context: ../myapp/
dockerfile:Dockerfile
```

Dockeritzar una aplicacio node.js

(en desenvolupament)

Veure comanda CMD en Dockerfile.

Referències:

<https://ataarea.es/tutorial/docker/>

<https://apuntes.de/docker-certificacion-dca/#gsc.tab=0>

<https://docs.docker.com/compose/compose-file/compose-file-v3/>

<https://github.com/jamj2000/Docker>

Carpeta projecte lamp dins de activitats transversals.

Tutorial docker: `docker run -d -p 80:80 docker/getting-started`

Tutorial node.js amb docker: <https://nodejs.org/en/docs/guides/nodejs-docker-webapp/>

<https://stackify.com/docker-build-a-beginners-guide-to-building-docker-images/>