

## Activitat 2. Desplegament d'una aplicació php amb stack LAMP

### Index

Activitat 2. Desplegament d'una aplicació php amb stack LAMP.....	1
1. Introducció.....	1
2. Docker Compose.....	1
2.1 Fitxer docker-compose.yml.....	2
Definició del fitxer default.conf.....	6
Definició de la imatge. Fitxer Dockerfile.....	6
Definició del servei per a executar el servidor web al fitxer docker-compose.....	7
Definició del esquema de la base de dades de l'aplicació.....	8
3. Bibliografia / Webgrafia.....	14

### 1. Introducció

Com vam veure a la **pràctica anterior** per desplegar una **pàgina web** amb docker cal crear un fitxer **Dockerfile** on, a partir de la selecció d'una imatge base que conté el servidor web, **copiarem els fitxers de la nostra web** a la carpeta on tenim el **document root** del servidor.

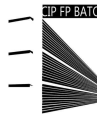
Una vegada tenim el Dockerfile, **creem una nova imatge** que conté la nostra pàgina web i **llençarem tants contenidors com necessitem** ([escalabilitat horitzontal](#)) de manera que cada contenidor representarà una instància de la nostra web.

Generalment, el **desplegament d'una aplicació web implica la interacció** de diferents serveis. En el cas d'una **aplicació php** tenim; *el servidor web, el servidor php-fpm i el servidor de base de dades*.

Amb la fi d'obtenir els màxim beneficis d'aquesta tecnologia i, seguint la seua filosofia, **cadascun d'aquests serveis** han de desplegar-se en **un contenidor diferent** i amb **configuracions pròpies**; *nom d'usuari, password, nom de la base de dades, direcció o ports d'escolta, opcions i directives del fitxers de configuració...* *Definir totes aquestes configuracions e interaccions mitjançant fitxers Dockerfile i desplegar-los tots junts amb un script de bash és una tasca engorrosa i amb difícil traçabilitat. Per això tenim la eina **docker-compose** que introduïrem al llarg de la següent pràctica.*

### 2. Docker Compose

Per a **instalar/actualitzar Docker Compose**, seguirem les passes indicades per al nostre sistema operatiu i que podem trobar a la [documentació oficial](#). En una versió ubuntu server es resumeix a executar el següent comandament:



```
sudo apt install docker-compose
```

Una vegada finalitza el procés consultarem la versió instal·lada per comprovar que funciona correctament.

```
profe@ddaw-batoi:~$ docker-compose -version
docker-compose version 1.29.2, build unknown
```

Aquesta ferramenta es basa en l'utilització d'un fitxer de text amb **format yaml** per indicar els recursos que han de desplegar-se per fer funcionar una web.

**YAML** és l'acrònim “**YAML Ain't Markup Language**”, que en valencià significa, “YAML no és un llenguatge de marques”. Pots obtindre més informació en <https://es.wikipedia.org/wiki/YAML>

L'ús de **YAML** dins de “Docker Compose” és senzill i fàcilment comprensible amb cadascun dels exemples. **Podeu** fer un **repàs** previ **als principals elements** de **YAML** revisant els exemples de la Wikipedia, disponibles en: <https://es.wikipedia.org/wiki/YAML#Ejemplos>

## 2.1 Fitxer docker-compose.yml

Per a definir un nou entorn de desplegament crearem en l'arrel del projecte un fitxer de text anomenat `docker-compose.yml`.

```
$ touch docker-compose.yml
```

La estructura de directoris que ens deu quedar a la nostra pràctica en aquest punt serà la següent

```
/usuari/home/practica6-2/
├─ docker-compose.yml
```

En primer lloc editarem l'arxiu i definirem la **versió** del llenguatge de la especificació de docker-compose que anem a utilitzar. (Esta directiva és opcional des de la versió 1.27.0)

```
version: "3.9"
```

Seguidament utilitzarem la paraula **services** per indicar, en forma **d'array associatiu**, el

**conjunt de contenidors** que necessita la aplicació per funcionar. A continuació definirem que la configuració de la nostra aplicació necessita 3 contenidors;

- **db:** per al sistema gestor de base de dades.
- **web:** Per al servidor web
- **php:** per al servidor d'aplicacions

```

docker-compose.yml

version: "3.9"

services:

  db:

    # Paràmetres per llençar el contenidor de bd

  app:

    # Paràmetres per llençar el contenidor que contendrà el servidor web junt
    al codi font de la nostra aplicació.

  php:

    # Paràmetres per llençar el contenidor que executarà el codi php de la nostra
    aplicació

```

## 2.2 Definició d'un contenidor amb mysql.

Tornarem a editar el fitxer `docker-compose.yml` e introduïrem les següents directives per a desplegar el servidor de bases de dades.

```

docker-compose.yml

version: "3.9"

services:
  db:
    image: mysql:8.3.0
    volumes:
      - db_data:/var/lib/mysql

    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: 1234
      MYSQL_DATABASE: todo-list-db
      MYSQL_USER: todo-user
      MYSQL_PASSWORD: 1234

```

A continuació, analitzarem cadascuna de les directives que hem utilitzat:

- **imatge:** Defineix la imatge que utilitzarà el contenidor, en el nostre cas hem especificat que faça ús de la imatge **oficial de mysql** amb el tag **8.3.0**. Aquesta imatge la podem encontrar al [repositori](#) de docker-hub.

```
image: mysql:8.3.0
```

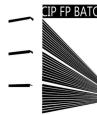
- **volumes:** Defineix quin volum utilitzarà el servei per a persistir la informació. **Un volum és una entitat independent en el sistema** que **podem muntar sobre diferents contenidors**. En el nostre cas, **hem definit que el directori `/var/lib/mysql`** del contenidor (directori en el que es guarden les dades de totes les base de dades en format binary) **estarà ubicat en el volum `db_data`**. Aquest volum el definirem més endavant.

```
volumes:
  - db_data:/var/lib/mysql
```

- **restart:** Aquesta opció indica el comportament que volem que tinguin els contenidors quan hi haja una fallada del sistema o quan parem el contenidor . En el nostre cas **hem el·legit** la opció **`always`** perquè volem que el **contenidor es re-llance automàticament** (útil per a contenidors que puguen caure per una fallada, però siguen necessaris perquè l'aplicació funcione). Pots veure totes les opcions de definició en la [documentació oficial](#).

```
restart: always
```

- **environment:** Aquesta opció conté amb un vector associatiu, una serie de variables de entorn (en forma de parell variable/valor) de cada una de les opcions que volem configurar al contenidor. A efectes pràctics, estem definint:
  - El **password** de l'usuari root de MySQL
  - El **nom** de la **base de dades** que crearà per defecte "todo-list-db".
  - Un **usuari** amb permisos de root per a la **base de dades anterior** anomenat "todo-user" (necessari per a connexions remotes) i el seu **password** com "1234".



❗ Les **variables** que podem **configurar** per a un contenidor han sigut definides en el **moment** en el que es va **crear la imatge**. En el nostre cas, ho va fer la empresa de oracle i les podem consultar en la informació del [repositori oficial](#).

## Activitat 1

Consulta la documentació oficial i contesta breument i **amb les teues paraules** a les següents **qüestions**:

- 1.1.- Quines variables d'entorn poden configurar-se en la imatge de mysql? Enumera-les i descriu per a que serveix cadascuna.
- 1.2.- Quines altres opcions podem indicar per a la directiva **restart**? Explica el comportament del contenidor amb cada opció davant una parada manual i davant una fallada del mateix.

### 2.3 Definició d'un contenidor per a desplegar l'aplicació

Fins ara, hem declarat que necessitem un contenidor que gestione mysql amb una configuració concreta. Ara definirem el **contenidor amb nginx** que desplegarà la nostra aplicació i que farà ús de l'anterior.

Dins de la carpeta `/usuari/home/practica6-2` crearem una **carpeta nginx** i dins 2 fitxers de text buits:

- El primer contindrà el fitxer `Dockerfile` per a crear la imatge amb el **servidor nginx** i el codi font de la **nostra aplicació**.
- El segon fitxer amb nom `default.conf` contindrà la configuració del vhost que volem i que copiarem en el moment de construir la imatge per a que el copie al contenidor.

Per finalitzar, **clonarem** el [següent repositori](#) que conté el projecte de l'aplicació que volem desplegar **al directori arrel**.

```
git clone https://gitlab.com/alecogi-edu/ddaw-ud6-a2.git
```

La estructura de directoris que ha de quedar en aquest punt és la següent:

```
/usuari/home/practica6-2/
```

```
├── docker-compose.yml
├── nginx
│   ├── default.conf
│   └── Dockerfile
└── ddaw-ud6-a2
    ├── config
    ├── db_backup
    └── ....
```

### Definició del fitxer `default.conf`

A continuació, editarem el fitxer `default.conf` e inclourem les següents directives.

```
default.conf
```

```
server {
    listen 80;
    root /usr/share/nginx/html;
    index index.php;
    location / {
        try_files $uri $uri/ /index.php?$args;
    }
    location ~ [^/]\.php(/|$) {
        fastcgi_split_path_info ^(.+?\.php)(/.*)$;
        include fastcgi_params;
        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
        fastcgi_param PATH_INFO $fastcgi_path_info;
        fastcgi_param PATH_TRANSLATED $document_root$fastcgi_path_info;
        fastcgi_pass php:9000;
    }
}
```

Donat aquest punt del curs, l'única directiva que cal explicar en aquesta configuració és `fastcgi_pass php:9000`. com hauràs observat, **no estem especificant la ip** del contenidor que tindrà el servici php-fpm, sinó que **estem especificant el nom "php"** que es com hem anomenat al servici que s'encarregarà de desplegar el servidor d'aplicacions. *Açó ho podem fer perquè **docker té un servidor de noms intern (DNS) que realitza les conversions entre el nom que li hem ficat a cada servei i la ip real que tindrà el contenidor quan haja sigut desplegat.***

### Definició de la imatge. Fitxer `Dockerfile`

A continuació editarem el fitxer `Dockerfile` per crear la imatge amb el servidor nginx



### Dockerfile

```
FROM nginx:latest
COPY ./default.conf /etc/nginx/conf.d/default.conf
```

## Activitat 2

Tenint en compte el que hem après a la pràctica anterior:

- 2.1.- Explica el que fa cada una de les línies que hem definit al fitxer Dockerfile.
- 2.2.- Crea una imatge anomenada `app-example:v1` que quede guardada a la teua màquina.  
*Pega una captura de pantalla on demostres que aquesta imatge està creada i present al teu sistema operatiu.*

### Definició del servei per a executar el servidor web al fitxer `docker-compose`

A continuació, tornarem al fitxer `docker-compose.yml` y definirem el nostre contenidor que **executarà el servidor web amb al codi font de l'aplicació**.

### docker-compose.yml

```
version: "3.9"
services:
  ...
  app:
    image: app-example:v1
    depends_on:
      - db
      - php
    ports:
      - "8008:80"
    volumes:
      - ./ddaw-ud6-a2/:/usr/share/nginx/html
    restart: always
```

A continuació, analitzarem cada una de directives que hem utilitzat.

- **image: app-example:v1**: s'utilitza com a imatge base, la creada en el punt anterior que conté el servidor nginx i els fitxers font de la nostra aplicació.
- **depends\_on**: Aquest **contenidor necessita** del **servicis** anomenats com a **db** i **php** per funcionar.

```
depends_on:
  - db
  - php
```

- **ports**: El port 8008 de la màquina anfitriona es redirigirà al port 80 d'aquest contenidor.

```
ports:
  - "8008:80"
```

- **volumes**: Hem definit un **volum de tipus "bind mount"**. La diferència respecte al volum utilitzat en el punt 1, es que **ací simplement estem enllaçant un directori o fitxer local amb un directori o fitxer de la màquina** de forma que qualssevol modificació que fem en el fitxer o directori de l'anfitrió es veurà reflectida en la màquina client. En aquest cas l'hem utilitzat per ficar el codi font de l'aplicació en el document root de nginx.

```
volumes:
  - ./ddaw-ud6-a2/:/usr/share/nginx/html
```

## Definició del esquema de la base de dades de l'aplicació

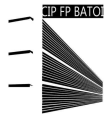
Per a que l'aplicació funcione, **hem de definir l'esquema** de la **base de dades** la primera vegada que s'inicie el contenidor. Per fer-ho, la documentació oficial de la imatge de mysql ens diu:

### Initializing a fresh instance

When a container is started for the first time, a new database with the specified name will be created and initialized with the provided configuration variables. Furthermore, it will execute files with extensions `.sh`, `.sql` and `.sql.gz` that are found in `/docker-entrypoint-initdb.d`. Files will be executed in alphabetical order. You can easily populate your `mysql` services by [mounting a SQL dump into that directory](#) and provide [custom images](#) with contributed data. SQL files will be imported by default to the database specified by the `MYSQL_DATABASE` variable.

Es a dir, haurem de ficar el script de creació de la base de dades que ens proporciona l'aplicació i que encontrem en la ruta **ddaw-ud6-a2/db\_backup** del repositori que hem clonat a la carpeta **/docker-entrypoint-initdb.d** del contenidor. D'aquesta forma s'executarà el script i es crearà el esquema inicial de la base de dades.





Per fer-ho utilitzarem un **volum de tipus "bind mount"** que, com hem explicat al punt anterior, **estem enllaçant un directori o fitxer local amb un directori o fitxer del contenidor** de forma que qualsevol modificació que fem en el fitxer o directori de l'anfitrió es veurà reflectida en la màquina client.

Editarem el fitxer `docker-compose.yml` e incluirem el següent "bind mount" en el contenidor de mysql

```

docker-composer.yml

version: "3.9"
services:
  db:
    image: mysql:8.3.0
    volumes:
      - db_data:/var/lib/mysql
      - ./ddaw-ud6-a2/db_backup/crm_db.sql:/docker-entrypoint-initdb.d/crm_db.sql
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: 1234
      MYSQL_DATABASE: todo-list-db
      MYSQL_USER: todo-user
      MYSQL_PASSWORD: 1234

```

## Activitat 3

Explica amb les teues paraules la **diferència** entre el volum utilitzat en la **primera part** (directori on es guarden les dades de les bases de dades i el **bind mount** utilitzat en aquest punt. Pots fer servir el [següent recurs](#) que tens a aules.

### 2.4 Definició d'un contenidor per executar el servidor php-fpm

Com hem estudiat a les unitats anterior, **nginx** és un servidor web que **necessita** d'un **servidor d'aplicacions** per a executar el **codi** amb extensió **.php**. En aquesta part, definirem el servei que ens permetrà fer aquesta tasca.

Primer, modificarem la imatge PHP i instal·larem l'extensió de PHP `pdo_mysql`, que ens permet connectar-nos des de la nostra aplicació a la BBDD MySQL. Per fer-ho crearem un **directori php** i dins un nou fitxer **dockerfile**.

```
/usuari/home/practica6-2/
├── php
│   └── Dockerfile
```

Dins del dockerfile ficarem les següents directives

#### Dockerfile.yml

```
FROM php:8.2-fpm

RUN apt-get update && apt-get install -y && docker-php-ext-install pdo_mysql
```

Una vegada més, hem d'editar el fitxer **docker-compose.yml** amb l'objectiu de definir el servici que instanciarà el contenidor amb aquesta imatge.

#### docker-composer.yml

```
version: "3.9"
services:
  ...
  php:
    build: ./php/
    volumes:
      - ./ddaw-ud6-a2/:/usr/share/nginx/html
    expose:
      - 9000
    environment:
      DB_HOST: db:3306
      DB_USER: todo-user
      DB_PASSWORD: 1234
      DB_NAME: todo-list-db
```

Passem a analitzar cada una de les directives utilitzades:

- **build: ./php/**: Especifiquem que la **imatge base** per a **aquest contenidor** ha d'obtindre-la d'un **fitxer Dockerfile** que ha de buscar en la carpeta **php**.
- **volumes**: Hem crear un "bind mount", per a que el servidor php-fpm puga accedir als fitxers font del web a desplegar.

- **expose:** Obrim el port 9000 per a que el servidor web pugui connectar-se amb el servidor php-fpm.
- **environment:** Hem desenvolupat l'**aplicació** per a que el fitxer de configuració **config/database.params.php** llija el valor de les variables d'entorn indicades de manera que **conega les credencials** i la **direcció del sistema gestor de base de dades** i pugui connectar amb ella.

## 2.5 Definició dels volums de dades utilitzats

Per finalitzar **definim una nova etiqueta** al nivell de **services** amb el volum que hem creat al punt 1 per guardar les dades del servidor de mysql. Editarem el fitxer **docker-compose.yml** i afegirem les següents directives

```

docker-compose.yml

version: "3.9"
services:
  db:
    ....
  app:
    ...
  php:
    ...
volumes:
  db_data:

```

## 2.5 Llançant el nostre exemple d'aplicació "docker-compose.yml"

Una vegada creat i entès el nostre fitxer "docker-compose.yml", podem posar en marxa el nostre servei situant-nos en el directori on està este fitxer i escrivint:

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

Amb l'opció **"up"** indiquem que s'interprete la plantilla definida en "docker-compose.yml" i amb **"-d"** indiquem que l'execució siga en segon pla. Amb l'execució del comandament veurem el següent:

```

→ practica6-2 docker-compose up -d
[+] Running 11/1
  ✓ db 10 layers [□□□□□□□□□□] 0B/0B Pulled
[+] Building 115.6s (6/6) FINISHED
=> [php internal] load build definition from Dockerfile
=> => transferring dockerfile: 130B
=> [php internal] load .dockerignore
=> => transferring context: 2B
=> [php internal] load metadata for docker.io/library/php:8.2-fpm
=> [php 1/2] FROM docker.io/library/php:8.2-fpm@sha256:52e13cbfa178342e477bbb8c24edde931f6a834dda3f06555947266651c425e
=> => resolve docker.io/library/php:8.2-fpm@sha256:52e13cbfa178342e477bbb8c24edde931f6a834dda3f06555947266651c425e
=> => sha256:e22afa33f327082e95c674a5bdf3b6d25760aca5abce3b032de2c444ba4864ed 104.35MB / 104.35MB

```

```

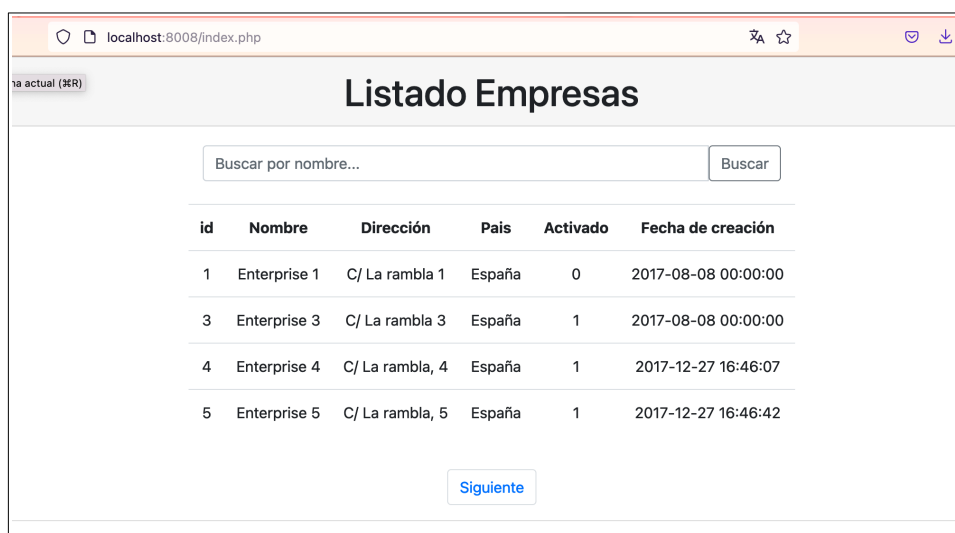
=> => sha256:52e13cbfa178342e477bbb8c24edde931f6a834dda3f06555947266651c425e 1.86kB / 1.86kB 0.0s
=> => sha256:8daa41d8cf31ab18ad0fd3defdd3912a1e5aed91c9c821e531cf2c2bae6680 2.41kB / 2.41kB 0.0s
=> => sha256:3a95dcec60350663284915f66e96b2d216179ebe405663a818c81de0f22f45e1 222B / 222B 1.6s
=> => sha256:dd2f9cce09e72e2b92c2f91a3c6a4e153e6130c3d5e215a7c098b86670329db1 11.95kB / 11.95kB 0.0s
=> => sha256:1aa61ea11ee8a864fd6b368bd09be18de3692f7ada24f03df32e585107432160 270B / 270B 0.3s
=> => extracting sha256:3a95dcec60350663284915f66e96b2d216179ebe405663a818c81de0f22f45e1 0.0s
=> => extracting sha256:d0dd4d907a5a0466309979118a069acdb11f40b438ed7bb7e548cb33a558ea56 60.0s
=> [php 2/2] RUN apt-get update && apt-get install -y && docker-php-ext-install pdo_mysql 0.2s
=> [php] exporting to image 0.2s
=> => exporting layers 0.0s
=> => writing image sha256:0ccb00f0a9ed8a253f77b8cf2c5b9b03fe6363e85b4aa46113764a4cc6e1ad9d 0.0s
=> => naming to docker.io/library/practica6-2-php 0.0s
[+] Running 5/5
✔ Network practica6-2_default Created 0.2s
✔ Volume "practica6-2_db_data" Created 0.0s
✔ Container php-container Started 0.7s
✔ Container practica6-2-db-1 Started 3.3s
✔ Container practica6-2-app-1 Started

```



Tens un resum del **principals comandaments** de docker-compose al [apartat de recursos](#) del aula virtual.

Si accedim ara amb una navegador a la direcció <http://127.0.0.1:8008> veurem la nostra aplicació en marxa.



id	Nombre	Dirección	Pais	Activado	Fecha de creación
1	Enterprise 1	C/ La rambla 1	España	0	2017-08-08 00:00:00
3	Enterprise 3	C/ La rambla 3	España	1	2017-08-08 00:00:00
4	Enterprise 4	C/ La rambla, 4	España	1	2017-12-27 16:46:07
5	Enterprise 5	C/ La rambla, 5	España	1	2017-12-27 16:46:42

## 2.6 Gestió de volums

Una vegada que hem ficat en marxa l'aplicació podem veure que s'han creat nos volums al sistema de docker amb la següent ordre.

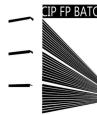
```
$ docker volume ls
```

```
→ ~ docker volume ls
```

```
DRIVER    VOLUME NAME
local     practica6-2_db_data
```

Per a poder **esborrar un volumen**, primer **hem de parar el contenidor o contenidors** que estan utilitzant-los.

Amb **docker-compose down** parem y eliminen els contenidors creats i amb l'ordre **docker**



`volume rm {ID_DEL_CONTENIDOR}` borrem el volumen.

```
→ practica6-2 docker-compose down
[+] Running 4/4
✓ Container practica6-2-app-1 Removed
✓ Container practica6-2-db-1 Removed
✓ Container practica6-2-php-1 Removed
✓ Network practica6-2_default Removed
→ practica6-2 docker volume rm practica6-2_db_data
practica6-2_db_data
→ practica6-2 █
```



Aquestes últimes ordres **són importants** ja que si volem que torne a executar-se el **script de creació de la base de dades**, o volem canviar el nom o usuari que s'ha creat anteriorment, haurem de esborrar el volum abans d'executar l'ordre `docker-compose up`



La primera vegada que llancem l'aplicació, es crearà la **imatge temporal** per al **contenedor del php-fpm** que hem definit mitjançant el fitxer `/php/Dockerfile` si volem tornar-la a crear perquè hem modificat el fitxer Dockerfile haurem de desplegar l'aplicació utilitzant la opció `--build`

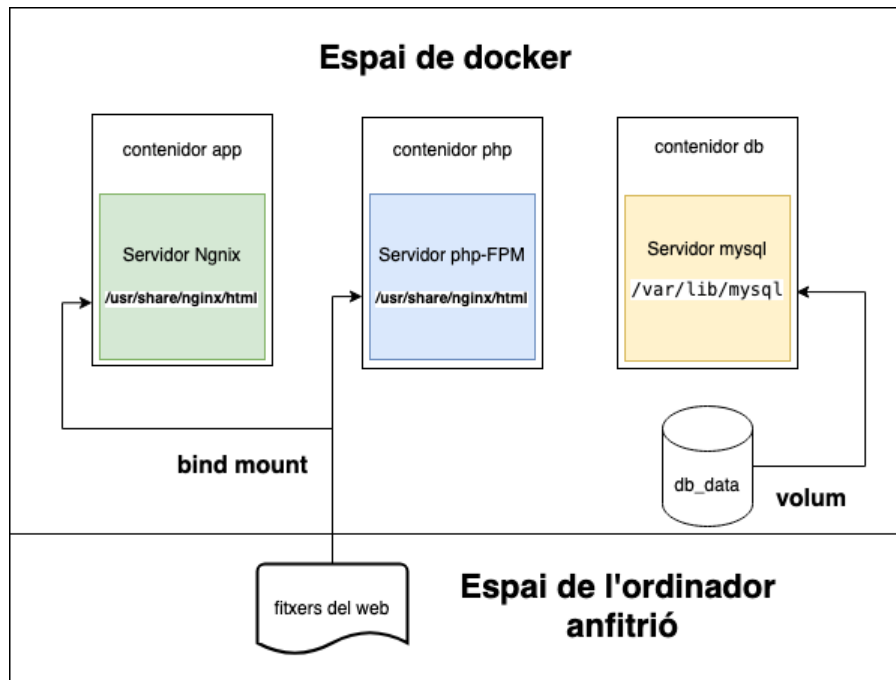
```
docker-compose up --build
```



Recorda que pots connectar-te a un contenidor per veure el seu interior amb el següent comandament

```
docker exec -it {CONTAINER_ID/CONTAINER_NAME} /bin/bash
```

La següent figura mostra un resum de l'arquitectura que hem desplegat amb l'eina `docker-compose` per ficar en marxa l'aplicació d'exemple.



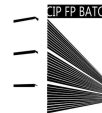
## Activitat 4.- Desplegament d'una aplicació Web

Du a terme el desplegament del web indicat als punt anteriors però tenint en compte les següents peculiaritats:

- El servidor de base de dades s'anomenarà "sgbd" i la contrasenya del usuari de la base de dades serà "ud6a02password".
- La base de dades s'anomenarà 'crm\_db', i el usuari de connexió 'db-user'
- El volum per a emmagatzemar les dades de les bases de dades rebrà el nom de "mysql-data".
- El virtualhost sols atendrà el domini "grupoXX.lan"
- El document root del servidor nginx serà "/var/www/html"

Una vegada desplegada l'aplicació mostra captures de pantalla de:

- L'aplicació funcionant accedint des del navegador.
- El logs d'accés tant del contenidor que desplega el servidor web com del contenidor que desplega el servidor d'aplicacions.
- Els volums que s'han creat
- El fitxer docker-compose.yml que has definit
- El fitxer de configuració del server root



### 3. Bibliografia / Webgrafía

- Documentació de la imatge docker php. [https://hub.docker.com/\\_/php](https://hub.docker.com/_/php)
- Docker cli reference. <https://docs.docker.com/engine/reference/run/>
- Documentació oficial de docker compose. <https://docs.docker.com/compose/>
- Curs docker Sergi Garcia. <https://sergarb1.github.io/CursoIntroduccionADocker/>