DESPLIEGUE DE APLICACIONES WEB

DESPLIEGUE DE UNA APLICACIÓN PHP CON NGINX Y MYSQL USANDO DOCKER Y DOCKER-COMPOSE

AUTOR: Alberto Martínez Pérez

CURSO: 2º CFGS Desarrollo de Aplicaciones Web (DAW)

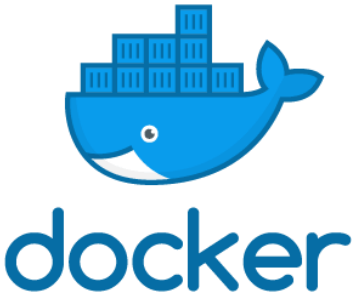
MÓDULO: Despliegue de Aplicaciones Web

FECHA DE ENTREGA: 10 de marzo de 2024

ÍNDICE

INTRODUCCIÓN

A. ¿QUÉ ES DOCKER?



Docker es una plataforma de código abierto diseñada para facilitar la creación, implementación y ejecución de aplicaciones en contenedores. Los contenedores son entornos ligeros y portátiles que encapsulan una aplicación junto con sus dependencias y configuraciones, permitiendo que se ejecute de manera consistente en diferentes entornos.

Utiliza características de aislamiento de recursos del kernel de Linux como por ejemplo cgroups o los namespaces para permitir que los contenedores se ejecuten dentro de una única instancia de Linux y así evitar la sobrecarga de iniciar y mantener múltiples máquinas virtuales.

B. ¿QUÉ ES DOCKER COMPOSE?

Para ejecutar una aplicación Docker necesitamos de un fichero llamado Dockerfile que sirve como configuración del contenedor siendo por tanto variable en función del contenedor.

Este proceso de crear múltipls ficheros de configuración y ejecutarlos puede resultar tedioso, aquí es donde entraría docker composer el cual podemos definir como una herramienta que nos permite definir y ejecutar múltiples contenedores.

Estos contenedores se definirán en un único fichero de configuración llamado docker-compose y que tiene la extensión .yml. Por tanto, con un único comando se generarán e inciarán todos los servicios.

Usar Compose es básicamente un proceso de tres pasos:

a) Definir el entorno de nuestra aplicación con un Dockerfile para que pueda reproducirse en cualquier lugar.
b) Definir los servicios que componen la aplicación docker-compose.yml para que puedan ejecutarse juntos en un entorno aislado.
c) Ejecutar docker-compose up y Compose inicia y ejecuta toda su aplicación.

Este proceso se denomina orquestación de contenedores.

CONFIGURACIONES PREVIAS

Para la ejecución de esta práctica se requiere de una máquina virtual con un sistema operativo Linux, en nuestro caso se utilizará un Ubuntu 22.04.

Además, esta máquina cuenta con Docker y docker-compose ya instalado.

DOCKERIZACIONES

Lo primero que debemos crear es un directorio común para todos los contenedores que vamos a crear.

Lo vamos a crear en nuestro /home con el nombre "practicaDocker":

mkdir /home/usuario/practicaDocker

```
albertom-cliente@docker:~$ mkdir /home/albertom-cliente/practicaDocker
albertom-cliente@docker:~$ ls -la
total 112
drwxr-x--- 15 albertom-cliente albertom-cliente 4096 feb 27 17:01 .
drwxr-xr-x  3 root             root             4096 nov  1 12:38 ..
-rw-------  1 albertom-cliente albertom-cliente 1286 feb 27 16:51 .bash_history
-rw-r--r--  1 albertom-cliente albertom-cliente  220 nov  1 12:38 .bash_logout
-rw-r--r--  1 albertom-cliente albertom-cliente 3771 nov  1 12:38 .bashrc
drwx------ 11 albertom-cliente albertom-cliente 4096 nov  1 12:57 .cache
drwx------ 11 albertom-cliente albertom-cliente 4096 nov  1 12:52 .config
drwxr-xr-x  2 albertom-cliente albertom-cliente 4096 nov  1 12:50 Descargas
drwxr-xr-x  2 albertom-cliente albertom-cliente 4096 nov  1 12:50 Documentos
drwxr-xr-x  2 albertom-cliente albertom-cliente 4096 nov  1 12:50 Escritorio
drwxr-xr-x  2 albertom-cliente albertom-cliente 4096 nov  1 12:50 Imágenes
drwx------  3 albertom-cliente albertom-cliente 4096 nov  1 12:50 .local
drwxr-xr-x  2 albertom-cliente albertom-cliente 4096 nov  1 12:50 Música
drwxr-xr-x  2 albertom-cliente albertom-cliente 4096 nov  1 12:50 Plantillas
drwxrwxr-x  2 albertom-cliente albertom-cliente 4096 feb 27 17:01 practicaDocker
-rw-r--r--  1 albertom-cliente albertom-cliente  807 nov  1 12:38 .profile
drwxr-xr-x  2 albertom-cliente albertom-cliente 4096 nov  1 12:50 Público
```

A) <u>CREACIÓN DE UN CONTENEDOR NGINX</u>

Lo primero que necesitamos es un contenedor que tenga NGINX para alojar nuestra aplicación en PHP.

Para ello crearemos un directorio dentro de /practicaDocker con el nombre "nginx":

```
albertom-cliente@docker:~/practicaDocker$ mkdir nginx
albertom-cliente@docker:~/practicaDocker$ ls -la
total 12
drwxrwxr-x  3 albertom-cliente albertom-cliente 4096 feb 27 17:03 .
drwxr-x--- 15 albertom-cliente albertom-cliente 4096 feb 27 17:01 ..
drwxrwxr-x  2 albertom-cliente albertom-cliente 4096 feb 27 17:03 nginx
```

Dentro del directorio /practicaDocker dbemos crear un fichero con extensión .yml que se llamará docker-compose.yml que es el fichero de configuración general de docker-compose tal y como se explicó en la introducción de la práctica.

Dentro de este fichero añadiremos estas líneas:

```
  GNU nano 6.2
nginx:
  image: nginx:latest
  container_name: nginx-container
  ports:
    - 80:80
```

Esto significa que debe descargarse la última versión de nginx desde los repositorios de Docker y que se guardará en un contenedor con nombre nginx-container. Además, se escuchará en el puerto 80 del contenedor que se corresponderá con el puerto 80 de nuestra máquina virtual.

Para que esto ocurra debemos lanzar el siguiente comando:

docker-compose up -d

La opción -d (o Daemon) indica que el contenedor se ha de ejecutar en segundo plano.

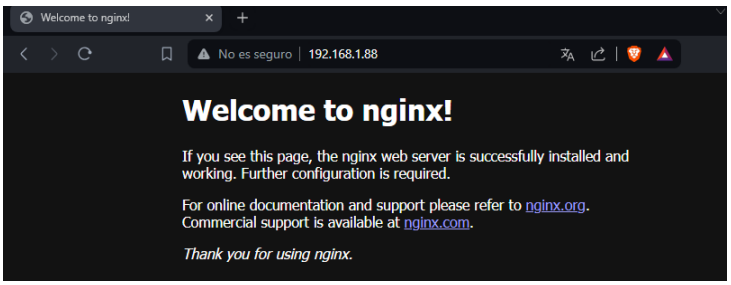
```
albertom-cliente@docker:~/practicaDocker$ docker-compose up -d
Pulling nginx (nginx:latest)...
latest: Pulling from library/nginx
e1caac4eb9d2: Pull complete
88f6f236f401: Pull complete
c3ea3344e711: Pull complete
cc1bb4345a3a: Pull complete
da8fa4352481: Pull complete
c7f80e9cdab2: Pull complete
18a869624cb6: Pull complete
Digest: sha256:c26ae7472d624ba1fafd296e73cecc4f93f853088e6a9c13c0d52f6ca5865107
Status: Downloaded newer image for nginx:latest
Creating nginx-container ...
```

Para comprobar que el contendor se ejecuta usamos el comando:

docker ps

```
albertom-cliente@docker:~/practicaDocker$ docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS          PORTS                                   NAMES
8e898148cf77   nginx:latest   "/docker-entrypoint.…"   47 seconds ago   Up 43 seconds   0.0.0.0:80->80/tcp, :::80->80/tcp   nginx-container
```

Además, si abrimos el navegador de nuestro anfitrión y accedemos a la URL http://ip_maquina_virtual:80 accederemos a la web por defecto de NGINX. Para esto es necesario que la red de la máquina permita la comunicación anfitrión-huesped, en nuestro caso utilizamos una conexión de tipo adaptador puente.



B) CREACIÓN DE UN CONTENEDOR PHP

Ahora vamos a crear el directorio y los documentos necesarios para este contenedor.

Este directorio tendrá el nombre "www" y dentro de él tendrá un nuevo directorio de nombre "html" que almacenará un fichero index.php.

```
albertom-cliente@docker:~/practicaDocker$ mkdir www
albertom-cliente@docker:~/practicaDocker$ mkdir ./www/html
albertom-cliente@docker:~/practicaDocker$ touch ./www/html/index.php
```

Dentro del fichero .php tendremos el siguiente contenido:

```
  GNU nano 6.2                          ./www/html/index.php *
<!DOCTYPE html>
<head>
  <title>¡Hola mundo!</title>
</head>

<body>
  <h1>¡Hola mundo!</h1>
  <p><?php echo 'Estamos corriendo PHP, version: ' . phpversion(); ?></p>
</body>
```

Ahora dentro del directorio /nginx debemos crear el archivo de configuración por defecto (default.conf) para que NGINX pueda ejecutar la aplicación PHP que acabamos de crear.

Para ello dentro de este fichero que crearemos con touch o con nano, debemos incluir el siguiente contenido:

```
  GNU nano 6.2                          ./nginx/default.conf *
server {

     listen 80 default_server;
     root /var/www/html;
     index index.html index.php;

     charset utf-8;

     location / {
      try_files $uri $uri/ /index.php?$query_string;
     }

     location = /favicon.ico { access_log off; log_not_found off; }
     location = /robots.txt { access_log off; log_not_found off; }

     access_log off;
     error_log /var/log/nginx/error.log error;

     sendfile off;

     client_max_body_size 100m;

     location ~ .php$ {
      fastcgi_split_path_info ^(.+.php)(/.+)$;
      fastcgi_pass php:9000;
      fastcgi_index index.php;
      include fastcgi_params;
      fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
      fastcgi_intercept_errors off;
      fastcgi_buffer_size 16k;
      fastcgi_buffers 4 16k;
    }

     location ~ /.ht {
      deny all;
     }
    }
```

Ahora vamos a crear el primer Dockerfile, el cual debemos crear dentro del directorio /nginx con el siguiente contenido:

```
 GNU nano 6.2                                ./nginx/Dockerfile *
FROM nginx:latest
COPY ./default.conf /etc/nginx/conf.d/default.conf
```

Por último, editamos el fichero docker-compose.yml quedando de la siguiente forma:

```
 GNU nano 6.2
services:
  nginx:
    build: ./nginx/
    container_name: nginx-container
    ports:
      - 80:80
    links:
      - php
    volumes:
      - ./www/html/:/var/www/html/

  php:
    image: php:7.0-fpm
    container_name: php-container
    expose:
      - 9000
    volumes:
      - ./www/html/:/var/www/html/
```

Por tanto, ahora se creará un nuevo contenedor de PHP en la versión 7.0 y con el nombre php-container. Se lanzará en el puerto 9000 y enlazará el contenedor de nginx con el contenedor de php.

Además, se creará un volumen el cual se montará en /var/www/html.

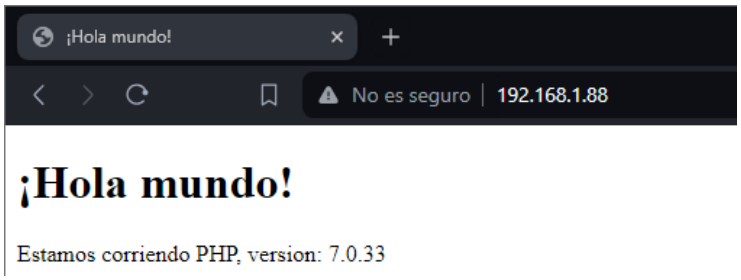
Para que esto ocurra debemos ejecutar de nuevo el comando docker-compose up -d en nuestro directorio raíz.

```
albertom-cliente@docker:~/practicaDocker$ docker-compose up -d
Creating network "practicadocker_default" with the default driver
Pulling php (php:7.0-fpm)...
7.0-fpm: Pulling from library/php
177e7ef0df69: Pull complete
9bf89f2eda24: Pull complete
350207dcf1b7: Pull complete
a8a33d96b4e7: Pull complete
8069cdba34b5: Pull complete
25ae788babab: Pull complete
1df95586682b: Pull complete
4c66f054f322: Pull complete
b7221dc910cf: Pull complete
71b3dbdbc6f4: Pull complete
Digest: sha256:8e8644c0fd61424939fff072ca617469550bbb659158301a2dcadf334746a1c2
Status: Downloaded newer image for php:7.0-fpm
Building nginx
[+] Building 1.4s (7/7) FINISHED
```

Si a continuación ejecutamos un docker ps veremos que tenemos dos contenedores:

```
albertom-cliente@docker:~/practicaDocker$ docker ps
CONTAINER ID   IMAGE                 COMMAND                  CREATED              STATUS              PORTS                                   NAMES
0d9377ee8fde   practicadocker_nginx  "/docker-entrypoint.…"   About a minute ago   Up About a minute   0.0.0.0:80->80/tcp, :::80->80/tcp   nginx-container
56c6cda97152   php:7.0-fpm           "docker-php-entrypoi…"   About a minute ago   Up About a minute   9000/tcp                            php-container
```

Si accedemos ahora desde el navegador de nuestro equipo anfitrión veremos la web Hola mundo que hemos creado en nuestro fichero index.php:



Una vez completado este paso deberíamos tener un directorio de ficheros como el siguiente:

```
albertom-cliente@docker:~/practicaDocker$ tree
.
├── docker-compose.yml
├── nginx
│   ├── default.conf
│   └── Dockerfile
└── www
    └── html
        └── index.php

3 directories, 4 files
```

C) CREACIÓN DE UN CONTENEDOR PARA DATOS

Ahora vamos a crear un contenedor independiente que se va a encargar de guardar los datos y que enlazaremos con el resto de los contenedores.

Este no requerirá de un directorio específico para él, sino que simplemente requerirá de una actualización del fichero YAML de Docker-Compose.

```
 GNU nano 6.2
services:
  nginx:
    build: ./nginx/
    container_name: nginx-container
    ports:
      - 80:80
    links:
      - php
    volumes_from:
      - app-data

  php:
    image: php:7.0-fpm
    container_name: php-container
    expose:
      - 9000
    volumes_from:
      - app-data

  app-data:
    image: php:7.0-fpm
    container_name: app-data-container
    volumes:
      - ./www/html/:/var/www/html/
    command: "true"
```

Y a continuación ejecutamos los comandos para descargar los contenedores necesarios y para comprobar el listado de contenedores como en pasos anteriores:

```
alberto-cliente@docker:~/practicaDocker$ docker-compose up -d
Recreating app-data-container ... done
Recreating php-container      ... done
Recreating nginx-container    ... done
alberto-cliente@docker:~/practicaDocker$ docker ps
CONTAINER ID   IMAGE                 COMMAND                  CREATED         STATUS         PORTS                                   NAMES
b3d805930884   practicadocker_nginx  "/docker-entrypoint.…"   7 seconds ago   Up 5 seconds   0.0.0.0:80->80/tcp, :::80->80/tcp   nginx-container
35c3e493c874   php:7.0-fpm           "docker-php-entrypoi…"   8 seconds ago   Up 6 seconds   9000/tcp                            php-container
```

D) CREACIÓN DE UN CONTENEDOR MySQL

En este apartado vamos a crear un contenedor para una base de datos MySQL y lo enlazaremos con el resto de conendores.

Lo primero que debemos hacer es instalar la extensión PHP para MySQL, para esto necesitaremos crear un nuevo directorio de nombre "php" y dentro de él un fichero Dockerfile con el siguiente contenido:

```
  GNU nano 6.2
FROM php:7.0-fpm
RUN docker-php-ext-install pdo_mysql
```

Modificamos de nuevo el fichero docker-compose.yml para crear un contenedor para MySQL, para ello tendremos el siguiente código dentro del fichero:

```
  GNU nano 6.2
services:
  nginx:
    build: ./nginx/
    container_name: nginx-container
    ports:
      - 80:80
    links:
      - php
    volumes_from:
      - app-data
  php:
    build: ./php/
    container_name: php-container
    expose:
      - 9000
    links:
      - mysql
    volumes_from:
      - app-data

  app-data:
    image: php:7.0-fpm
    container_name: app-data-container
    volumes:
      - ./www/html/:/var/www/html/
    command: "true"

  mysql:
    image: mysql:5.7
    container_name: mysql-container
    volumes_from:
      - mysql-data
    environment:
      MYSQL_ROOT_PASSWORD: secret
      MYSQL_DATABASE: mydb
      MYSQL_USER: myuser
      MYSQL_PASSWORD: password

  mysql-data:
    image: mysql:5.7
    container_name: mysql-data-container
    volumes:
      - /var/lib/mysql
    command: "true"
```

Una vez hecho debemos modificar el fichero index.php para comprobar una conexión a la base de datos:

```
 GNU nano 6.2                                                                ./www/html/index.php *
<!DOCTYPE html>
     <head>
      <title>¡Hola mundo!</title>
     </head>

     <body>
      <h1>¡Hola mundo!</h1>
      <p><?php echo 'Estamos corriendo PHP, version: ' . phpversion(); ?></p>
      <?
       $database ="mydb";
       $user = "myuser";
       $password = "password";
       $host = "mysql";

       $connection = new PDO("mysql:host={$host};dbname={$database};charset=utf8", $user, $password);
       $query = $connection->query("SELECT TABLE_NAME FROM information_schema.TABLES WHERE TABLE_TYPE='BASE TABLE'");
       $tables = $query->fetchAll(PDO::FETCH_COLUMN);

        if (empty($tables)) {
          echo "<p>No hay tablas en la base de datos \"{$database}\".</p>";
        } else {
          echo "<p>La base de datos \"{$database}\" tiene las siguientes tablas:</p>";
          echo "<ul>";
            foreach ($tables as $table) {
              echo "<li>{$table}</li>";
            }
          echo "</ul>";
        }
        ?>
    </body>
</html>
```

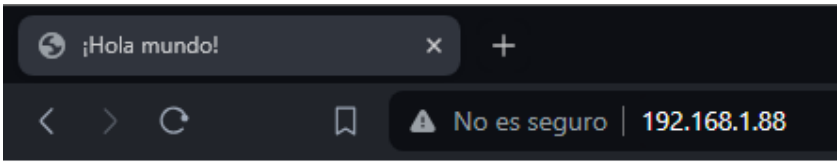
Completado esto lanzamos los contenedores de nuevo:

```
albertom-cliente@docker:~/practicaDocker$ docker-compose up -d
Pulling mysql-data (mysql:5.7)...
5.7: Pulling from library/mysql
20e4dcae4c69: Download complete
1c56c3d4ce74: Download complete
e9f03a1c24ce: Download complete
68c3898c2015: Download complete
6b95a940e7b6: Download complete
90986bb8de6e: Download complete
ae71319cb779: Downloading [================>                                  ]  8.661MB/25.53MB
ffc89e9dfd88: Download complete
43d05e938198: Waiting
064b2d298fba: Waiting
df9a4d85569b: Waiting
```

Y comprobamos que están ejecutándose, para ello utilizamos la opción -a del comando docker ps:

```
albertom-cliente@docker:~/practicaDocker$ docker ps -a
CONTAINER ID   IMAGE                COMMAND                  CREATED          STATUS                      PORTS                                  NAMES
111f7d5bda52   practicadocker_nginx "/docker-entrypoint.…"   6 seconds ago    Up 4 seconds                0.0.0.0:80->80/tcp, :::80->80/tcp      nginx-container
683f0b1bd232   practicadocker_php   "docker-php-entrypoi…"   11 seconds ago   Up 9 seconds                9000/tcp                               php-container
3692fe084cd6   mysql:5.7            "docker-entrypoint.s…"   14 seconds ago   Up 11 seconds               3306/tcp, 33060/tcp                    mysql-container
7d1ec5c74327   mysql:5.7            "docker-entrypoint.s…"   16 seconds ago   Exited (0) 12 seconds ago                                          mysql-data-container
7e5f795575f8   php:7.0-fpm          "docker-php-entrypoi…"   9 minutes ago    Exited (0) 13 seconds ago                                          app-data-container
```

Si ahora accedemos desde el navegador web de la máquina anfitriona veremos lo siguiente:



¡Hola mundo!

Estamos corriendo PHP, version: 7.0.33

No hay tablas en la base de datos "mydb".

Una vez finalizado este paso debemos tener una estructura de ficheros similar a esta:

```
albertom-cliente@docker:~/practicaDocker$ tree
.
├── docker-compose.yml
├── nginx
│   ├── default.conf
│   └── Dockerfile
├── php
│   └── Dockerfile
└── www
    └── html
        └── index.php

4 directories, 5 files
```

E) VERIFICACIÓN DE CONEXIÓN A LA BASE DE DATOS

Como vemos al final del apartado anterior se nos muestra el mensaje por defecto "No hay tablas en la base de datos "mydb"." Pero esto no es cierto ya que sí tenemos tablas, pero no son accesibles para un usuario normal.

Para que sean visibles debemos modificar el fichero index.php y cambiar los valores anteriores de $user por root y $password por secret:

```
 GNU nano 6.2
<!DOCTYPE html>
    <head>
     <title>¡Hola mundo!</title>
    </head>

    <body>
     <h1>¡Hola mundo!</h1>
     <p><?php echo 'Estamos corriendo PHP, version: ' . phpversion(); ?></p>
     <?
      $database ="mydb";
      $user = "root";
      $password = "secret";
      $host = "mysql";

      $connection = new PDO("mysql:host={$host};dbname={$database};charset=utf8
      $query = $connection->query("SELECT TABLE_NAME FROM information_schema.TA
      $tables = $query->fetchAll(PDO::FETCH_COLUMN);
```

Si ahora refrescamos la página en el navegador veremos las tablas: