Entrega Docker

Relizado por:

- David Massa Gallego (parte 1).
- Julio Martín García (parte 2).

Explicación de la parte 1

Enunciado

1. Crear un fichero docker-compose.yml con dos servicios: drupal + mysql.
2. Hacer que el servicio drupal utilice el puerto 81.
3. Hacer que ambos contenedores usen volúmenes para persistir información.
4. Comprobar que puede acceder a localhost:81 y mostrar la correcta configuración de Drupal.
5. Borrar los contenedores, explicar y demostrar si la información persiste

Introducción Práctica 3 Parte 1

En la primera parte de la práctica estamos creando un entorno donde se puede gestionar un sitio web basado en el servicio que nos proporciona Drupal, a raíz de una base de datos MySQL que es donde se guardará toda la información que la web necesite.

Para ello lo hacemos mediante un archivo llamado `docker-compose.yml`, el cual nos ahorra gran parte del trabajo porque en vez de instalar manualmente estas dos herramientas de las que hemos hablado anteriormente, lo automatizamos de tal manera que nos crea "**contenedores**". Dichos contenedores cuentan con estas herramientas de manera independiente.

Esto nos ahorra muchos quebraderos de cabeza, ya que configurando el archivo y sabiendo unos cuantos comandos en la terminal nos crea el entorno que queremos.

Vamos a proceder a explicar el `docker-compose` de la primera parte de la práctica 3. Voy a adjuntar el código entero del docker-compose y luego a desglosarlo parte por parte.

```
services:
  drupal:
    image: drupal:latest
    container_name: drupal-vs
    ports:
      - 81:80
    volumes:
      - volumenDrupal:/var/www/html/
    restart: always

  db:
    image: mysql:latest
    container_name: mysql-vs
    environment:
      MYSQL_ROOT_PASSWORD: example
      MYSQL_USER: vs
      MYSQL_PASSWORD: vs
      MYSQL_DATABASE: parte1
    volumes:
      - volumenSQL:/var/lib/mysql
    restart: always
volumes:
  volumenDrupal:
  volumenSQL:
```

- Como podemos observar el archivo se divide en dos partes: la parte de servicios que vamos a usar y la de volúmenes que hemos declarado.

Servicios (Parte 1)

Como se nos especifica en el enunciado vamos a usar en este caso dos servicios: `Drupal` y `MySQL`.

- Drupal:
    1. `image: drupal:latest` -> Indicamos que vamos a usar la última versión disponible de la imagen de Drupal en DH.
    2. `container_name: drupal-vs` -> Le damos nombre al contenedor que vamos a crear, en este caso: drupal-vs
    3. `ports:
    - 81:80` -> Mapea el puerto 81 del host al 80 del contenedor. Elegimos el 81 porque es el que se nos especifica en el enunciado.
    4. `volumes:
    - volumenDrupal:/var/www/html/` -> Crea un volumen llamado ***"volumenDrupal"*** en el host el cual vinculamos al directorio ***/var/ww/html*** dentro del contenedor, que es donde se almacena el código de Drupal. Esto lo hacemos para que la información persista dado a que, en el apartado 5 del enunciado se nos pide que se borre los contenedores para ver si la información se mantiene.

5. `restart: always` -> Esta última línea indica que el contenedor se reiniciará en el caso de que haya algún fallo, o bien se detenga. Nos proporciona alta disponibilidad.

- MySQL:
  1. `image: mysql:latest` -> Seleccionamos el uso de la última versión disponible de la imagen de MySQL en DH.
  2. `container_name: mysql-vs` -> Creamos un nombre para el contenedor en este caso será "mysql-vs".
  3. `environment:`
     `MYSQL_ROOT_PASSWORD: example`
     `MYSQL_USER: vs`
     `MYSQL_PASSWORD: vs`
     `MYSQL_DATABASE: parte1` -> En las variables del entorno declaramos creamos una contraseña root de MySQL la cual en este caso es: ***example***, un usuario y contraseña para dicho usuario en el que hemos asignado a ambas ***vs***, y por último el nombre que va tener la base de datos una vez se inicie el contenedor: ***parte1***.
  4. `volumes:
     - volumenSQL:/var/lib/mysql` -> Creamos un volumen llamado ***volumenSQL*** en el host, el cual está vinculado con el directorio de MySQL donde se almacenan los datos: ***/var/lib/mysql***, por la misma razón que explicamos antes en el apartado de **Drupal**.
  5. `restart: always` -> Mismo motivo que con el servicio de **Drupal**.

Volumes (Parte 1)

```
volumes:
  volumenDrupal:
  volumenSQL:
```

Aquí declaramos los volúmenes que hemos usado tanto en **Drupal** como en **MySQL**, para que toda la información persista por lo que si eliminamos o detenemos los contenedores creados los datos se mantendrán.

Explicación de la parte 2

En esta sección se detallarán los aspectos importantes de la actividad, con el fin de aclarar lo que no se explica en el video.

Enunciado

1. Crear un fichero docker-compose.yml con dos servicios: wordpress + mariadb.
2. Hacer que el servicio wordpress utilice el puerto 82.
3. Hacer que ambos contenedores usen una red llamada redDocker.
4. Comprobar que puede acceder a localhost:82 y mostrar la correcta configuración de wordpress.
5. Borrar los contenedores, explicar y demostrar si la información persiste.

docker-compose.yml

En este apartado se desgranarán las diferentes secciones del `docker-compose`.

```
services:
  wordpress:
    image: wordpress
    container_name: wordpress_VS
    networks:
      - redDocker
    ports:
      - 82:80
    environment:
      WORDPRESS_DB_HOST: ${WORDPRESS_DB_HOST}
      WORDPRESS_DB_USER: ${WORDPRESS_DB_USER}
      WORDPRESS_DB_PASSWORD: ${WORDPRESS_DB_PASSWORD}
      WORDPRESS_DB_NAME: ${WORDPRESS_DB_NAME}

  database:
    image: mariadb
    container_name: mariadb_VS
    networks:
      - redDocker
    ports:
      - 3306:3306
    environment:
      MARIADB_ROOT_PASSWORD: ${MARIADB_ROOT_PASSWORD}
      MARIADB_USER: ${MARIADB_USER}
      MARIADB_PASSWORD: ${MARIADB_PASSWORD}
      MARIADB_DATABASE: ${MARIADB_DATABASE}

networks:
  redDocker:
```

`services`

El `docker-compose` nos presenta dos servicios:

- `wordpress`: un contenedor con la última versión disponible de la imagen de wordpress.
- `database`: un contenedor con una imagen `mariadb` como base de datos.
  Ambos servicios tienen asignado un nombre al contenedor para una mejor identificación de los mismos a la hora de manipularlos.

Dentro de estos servicios se han establecido las siguientes propiedades:

- `networks`.
- `ports`.
- `enviroment`.

`networks`

En ambos contenedores se asignará una misma red, requerida por el ejercicio, la cual se denomina `redDocker` y permite la correcta comunicación entre ambos contenedores.

`ports`

Se asignan los puertos oportunos a los contenedores. En el caso del `wordpress`, se le asigna el 82 del host, puesto que es un requisito del ejercicio, y el 80 al puerto del contenedor, pues es el 80 el predeterminado de `wordpress`.
En el contenedor con la base de datos, se les asigna, tanto al puerto de la máquina como al del contenedor el puerto 3306, ya que este es el predeterminado para bases de datos.

`environment`

Este apartado detalla las distintas variables de entorno necesarias para garantizar el correcto funcionamiento y sincronización de los contenedores. Como se observa, estas credenciales no se encuentran *hardcodeadas*, lo cual añade una capa adicional de seguridad a nuestra infraestructura, especialmente importante para futuras subidas a sistemas de control de versiones como GitHub.
Estas variables estarán guardadas en un archivo oculto con nombre `env`, cuyo contenido será el siguiente:

```
WORDPRESS_DB_HOST=database
WORDPRESS_DB_USER=vs
WORDPRESS_DB_PASSWORD=vs
WORDPRESS_DB_NAME=db_entregable
MARIADB_ROOT_PASSWORD=vs
MARIADB_USER=vs
MARIADB_PASSWORD=vs
MARIADB_DATABASE=db_entregable
```

Por último, añadir varias apuntes necesarios sobre las variables de entorno:

- Es necesario que tanto `WORDPRESS_DB_NAME` como `MARIADB_DATABASE` tengan el mismo valor, pues es lógico pensar que ambas variables se refieren a una misma base de datos y de no coincidir, encontraríamos errores como que no existe la base de datos, o no serían la misma.
- `WORDPRESS_DB_USER` y `WORDPRESS_DB_PASSWORD` han de ser igual que `MARIADB_USER` y `MARIADB_PASSWORD`. ya que hace referencia al mismo usuario.
- `WORDPRESS_DB_HOST` hace referencia al host de la base de datos, en el caso de una infraestructura dockerizada, se refiere el nombre del servicio, es decir, `database`.

`networks`

En esta última sección, es necesario definir las distintas redes que hemos creado para nuestra infraestructura. En nuestro caso, y como requisito de la práctica, hemos creado una red llamada `redDocker` la cual permite una correcta comunicación entre los diferentes contenedores.

Archivos de configuración de wordpress

Para acceder a estos archivos, necesitamos entrar en el contenedor de `wordpress`. Esto se logra con el comando `docker exec -it <nombre_contenedor> /bin/bash`, en nuestro caso quedaría de la siguiente manera:

- `docker exec -it wordpress_VS /bin/bash`
  Una vez dentro del contenedor, haciendo un `ls` podremos ver los diferentes archivos de configuración de `wordpress`. El que nos interesa ahora mismo es `wp-config.php`.

El contenido de este archivo es el siguiente:

```
<?php
/**
 * The base configuration for WordPress
 *
 * The wp-config.php creation script uses this file during the installation.
 * You don't have to use the website, you can copy this file to "wp-config.php"
 * and fill in the values.
 *
 * This file contains the following configurations:
 *
 * * Database settings
 * * Secret keys
 * * Database table prefix
 * * ABSPATH
 *
 * This has been slightly modified (to read environment variables) for use in
Docker.
 *
 * @link https://developer.wordpress.org/advanced-administration/wordpress/wp-
config/
 *
 * @package WordPress
 */

// IMPORTANT: this file needs to stay in-sync with
https://github.com/WordPress/WordPress/blob/master/wp-config-sample.php
// (it gets parsed by the upstream wizard in
https://github.com/WordPress/WordPress/blob/f27cb65e1ef25d11b535695a660e7282b98eb
742/wp-admin/setup-config.php#L356-L392)

// a helper function to lookup "env_FILE", "env", then fallback
if (!function_exists('getenv_docker')) {
    // https://github.com/docker-library/wordpress/issues/588 (WP-CLI will load
this file 2x)
    function getenv_docker($env, $default) {
        if ($fileEnv = getenv($env . '_FILE')) {
            return rtrim(file_get_contents($fileEnv), "\r\n");
        }
        else if (($val = getenv($env)) !== false) {
            return $val;
        }
        else {
            return $default;
```

```
        }
    }
}

// ** Database settings - You can get this info from your web host ** //
/** The name of the database for WordPress */
define( 'DB_NAME', getenv_docker('WORDPRESS_DB_NAME', 'wordpress') );

/** Database username */
define( 'DB_USER', getenv_docker('WORDPRESS_DB_USER', 'example username') );

/** Database password */
define( 'DB_PASSWORD', getenv_docker('WORDPRESS_DB_PASSWORD', 'example password')
);

/**
 * Docker image fallback values above are sourced from the official WordPress
installation wizard:
 *
https://github.com/WordPress/WordPress/blob/1356f6537220ffdc32b9dad2a6cdbe2d010b7
a88/wp-admin/setup-config.php#L224-L238
 * (However, using "example username" and "example password" in your database is
strongly discouraged.  Please use strong, random credentials!)
 */

/** Database hostname */
define( 'DB_HOST', getenv_docker('WORDPRESS_DB_HOST', 'mysql') );

/** Database charset to use in creating database tables. */
define( 'DB_CHARSET', getenv_docker('WORDPRESS_DB_CHARSET', 'utf8') );

/** The database collate type. Don't change this if in doubt. */
define( 'DB_COLLATE', getenv_docker('WORDPRESS_DB_COLLATE', '') );

/**#@+
 * Authentication unique keys and salts.
 *
 * Change these to different unique phrases! You can generate these using
 * the {@link https://api.wordpress.org/secret-key/1.1/salt/ WordPress.org
secret-key service}.
 *
 * You can change these at any point in time to invalidate all existing cookies.
 * This will force all users to have to log in again.
 *
 * @since 2.6.0
 */
define( 'AUTH_KEY',         getenv_docker('WORDPRESS_AUTH_KEY',
'083200f9e72fae5b4ead31e752654b1460da9bbc') );
define( 'SECURE_AUTH_KEY',  getenv_docker('WORDPRESS_SECURE_AUTH_KEY',
 '37ddad51bfd6a13fb696bffb31ea83c176eb3207') );
define( 'LOGGED_IN_KEY',    getenv_docker('WORDPRESS_LOGGED_IN_KEY',
 'e2fb3c6e607b73fa3a543f9428931bf6aedfcf53') );
define( 'NONCE_KEY',        getenv_docker('WORDPRESS_NONCE_KEY',
 'db18ab97dd33a21664072be606d2dba32a17ad0d') );
define( 'AUTH_SALT',        getenv_docker('WORDPRESS_AUTH_SALT',
 'fdd9e92a55fe3eab811a5ca05b5af1b9b57fa36c') );
```

```
define( 'SECURE_AUTH_SALT', getenv_docker('WORDPRESS_SECURE_AUTH_SALT',
'59312efccd70dcbd685aabbfe3965991a5213d6e') );
define( 'LOGGED_IN_SALT',   getenv_docker('WORDPRESS_LOGGED_IN_SALT',
'f5dd5be7026f4e577ba5d9c06523a2125831be8f') );
define( 'NONCE_SALT',       getenv_docker('WORDPRESS_NONCE_SALT',
'47303070dbebd603c6954782bbf6b7064c2718a1') );
// (See also https://wordpress.stackexchange.com/a/152905/199287)

/**#@-*/

/**
 * WordPress database table prefix.
 *
 * You can have multiple installations in one database if you give each
 * a unique prefix. Only numbers, letters, and underscores please!
 *
 * At the installation time, database tables are created with the specified
prefix.
 * Changing this value after WordPress is installed will make your site think
 * it has not been installed.
 *
 * @link https://developer.wordpress.org/advanced-administration/wordpress/wp-
config/#table-prefix
 */
$table_prefix = getenv_docker('WORDPRESS_TABLE_PREFIX', 'wp_');

/**
 * For developers: WordPress debugging mode.
 *
 * Change this to true to enable the display of notices during development.
 * It is strongly recommended that plugin and theme developers use WP_DEBUG
 * in their development environments.
 *
 * For information on other constants that can be used for debugging,
 * visit the documentation.
 *
 * @link https://developer.wordpress.org/advanced-administration/debug/debug-
wordpress/
 */
define( 'WP_DEBUG', !!getenv_docker('WORDPRESS_DEBUG', '') );

/* Add any custom values between this line and the "stop editing" line. */

// If we're behind a proxy server and using HTTPS, we need to alert WordPress of
that fact
// see also https://wordpress.org/support/article/administration-over-ssl/#using-
a-reverse-proxy
if (isset($_SERVER['HTTP_X_FORWARDED_PROTO']) &&
strpos($_SERVER['HTTP_X_FORWARDED_PROTO'], 'https') !== false) {
    $_SERVER['HTTPS'] = 'on';
}
// (we include this by default because reverse proxying is extremely common in
container environments)

if ($configExtra = getenv_docker('WORDPRESS_CONFIG_EXTRA', '')) {
    eval($configExtra);
```

```
}

/* That's all, stop editing! Happy publishing. */

/** Absolute path to the WordPress directory. */
if ( ! defined( 'ABSPATH' ) ) {
    define( 'ABSPATH', __DIR__ . '/' );
}

/** Sets up WordPress vars and included files. */
require_once ABSPATH . 'wp-settings.php';
```

En este archivo se detalla la configuración general de `wordpress`, con algunas lineas a tener en cuenta como la siguiente:

- `define( 'DB_NAME', getenv_docker('WORDPRESS_DB_NAME', 'wordpress') );`
  En esta linea, se establece el nombre de la base de datos, la cual será, o bien la que se le pase como variable de entorno a través de docker, o en caso de que no se haya pasado, será `wordpress`.

¿Cómo podemos saber si se han establecido bien las variables de entorno?
Para saber que las variables de entornos que le hemos pasado se han establecido correctamente, podemos usar el comando `env` dentro de la consola del contenedor, el cual nos sacará la siguiente salida, donde podremos ver todas las variables de entorno del contenedor:

```
HOSTNAME=b38162da449a
PHP_VERSION=8.2.25
APACHE_CONFDIR=/etc/apache2
PHP_INI_DIR=/usr/local/etc/php
GPG_KEYS=39B641343D8C104B2B146DC3F9C39DC0B9698544
E60913E4DF209907D8E30D96659A97C9CF2A795A 1198C0117593497A5EC5C199286AF1F9897469DC
PHP_LDFLAGS=-Wl,-O1 -pie
PWD=/var/www/html
HOME=/root
PHP_SHA256=330b54876ea1d05ade12ee9726167332058bccd58dffa1d4e12117f6b4f616b9
WORDPRESS_DB_HOST=database
PHPIZE_DEPS=autoconf        dpkg-dev        file        g++         gcc
libc-dev        make        pkg-config      re2c
TERM=xterm
PHP_URL=https://www.php.net/distributions/php-8.2.25.tar.xz
SHLVL=1
PHP_CFLAGS=-fstack-protector-strong -fpic -fpie -O2 -D_LARGEFILE_SOURCE -
D_FILE_OFFSET_BITS=64
WORDPRESS_DB_PASSWORD=vs
APACHE_ENVVARS=/etc/apache2/envvars
WORDPRESS_DB_USER=vs
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
WORDPRESS_DB_NAME=db_entregable
PHP_ASC_URL=https://www.php.net/distributions/php-8.2.25.tar.xz.asc
PHP_CPPFLAGS=-fstack-protector-strong -fpic -fpie -O2 -D_LARGEFILE_SOURCE -
D_FILE_OFFSET_BITS=64
_=/usr/bin/env
```

Persistencia de datos

Nuestra infraestructura, al no tener volúmenes montados, no tendrá persistencia de datos, esto es, una vez apaguemos los contenedores, la información de estos se perderá y, aunque los levantemos de nuevo, volveremos a empezar desde 0, como si fuese la primera vez que levantamos los contenedores.