**ACTIVITAT AVALUABLE AC2**

**Mòdul:** MP08- Desplegament d’aplicacions web

**UF:** UF3 – Aplicacions web

**Professor:** Albert Guardiola

**Data límit d’entrega:** 30/03/25

**Mètode d’entrega:** Per mitjà del Clickedu de l’assignatura. Les activitats entregades més enllà de la data límit només podran obtenir una nota de 5.

**Instruccions:** S’ha d’entregar un únic document amb el nom:

***MP08-UF3-AC2-Nom\_Alumne.doc (o pdf)***

Es valorará la presentació.

**Resultats de l’aprenentatge:**

RA 1. Verifica l’execució d’aplicacions web comprovant els paràmetres de configuració de serveis de xarxa.

**En esta práctica aprenderemos los principios básicos de despliegue de aplicaciones en contenedores Docker.**

**1ª parte - Despliegue de un contenedor Docker**

**Tarea 0. Instala Docker Desktop:**

***Instalación de Docker Desktop:***

* **Para Windows:** [**https://docs.docker.com/desktop/windows/install/**](https://docs.docker.com/desktop/windows/install/)

**Recuerda que en Windows, la instalación te pedirá que instales, previamente, el kernel de Linux (WSL2):** [**https://docs.microsoft.com/es-es/windows/wsl/install-manual#step-4---download-the-linux-kernel-update-package**](https://docs.microsoft.com/es-es/windows/wsl/install-manual#step-4---download-the-linux-kernel-update-package)

* **Para Mac:** [**https://docs.docker.com/desktop/mac/install/**](https://docs.docker.com/desktop/mac/install/)

**Tarea 1 (2 puntos). a)Prepara una carpeta *holaMundo* en tu sistema de ficheros local que contenga el siguiente contenido:**

**-Una carpeta *src* con un simple *index.php* en su interior como el siguiente:**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Hola Mundo</title>

</head>

<body>

    <?php

        echo "Hola Mundo desde un contenedor Docker";

    ?>

</body>

</html>

**-Un fichero *Dockerfile* (sin extensión), con el siguiente contenido:**

FROM php:7.1-apache

COPY src/ /var/www/html

EXPOSE 80

**b)Desde el terminal de línea de comandos, ubícate en la raíz del proyecto y ejecuta el siguiente comando para compilar una nueva imagen**:

***docker build -t holamundo .***

**Observa cómo Docker descarga la imagen base (capa *php:7.1-apache*), y compila a partir de ella una nueva imagen con las instrucciones dispuestas en el *Dockerfile*. La imagen se llamará *holamundo*.**

**c)Arranca un nuevo contenedor Docker que ejecute la imagen que acabas de crear:**

***docker run -dp 3000:80 holamundo***

**d)Visualiza la salida de tu aplicación en**

<http://localhost:3000/>

**e)Analizemos lo que acaba de ocurrir. Para ello, explica:**

**-El contenido del Dockerfile.**

**-Las opciones del comando de arranque del contenedor.**

**f)Observa que puedes acceder a la terminal de la sesión de Linux que acabas de desplegar a través del panel de control de Docker Desktop.**

**Tarea 2 (1 punto) a) Utiliza el comando *docker ps* para listar los contenedores activos. Observa que a cada contenedor se le asigna un ID y un nombre aleatorio (un nombre curioso IMHO).**

**b)Para el contenedor y bórralo de memoria:**

***docker stop <nombre-contenedor>***

***docker rm <nombre-contenedor>***

**c)Verás que desaparece del panel de control de Docker.**

**b) Actualiza el código de tu aplicación: por ejemplo, cambia el valor del *echo*.**

**c)Recompila la imagen y vuelve a arrancar el contenedor. Observa los cambios en el navegador.**

**Tarea 3 (2 puntos) – Ahora vamos a montar el código de la aplicación como un volumen externo al contenedor (*bind mount*), para que no haya que recompilar cada vez los cambios que se realizan en el código.**

**a)Borra el contenedor que acabas de crear.**

**b)Comenta la línea *COPY...* del fichero Dockerfile para que no se copie el código de la aplicación a la imagen compilada.**

**c)Recompila la imagen.**

**d)Ahora arranca el contenedor con la opcion -v, para que el código se monte como un volumen externo a la imagen:**

***-En Windows:***

***docker run -dp 3000:80 -v "%cd%"/src:/var/www/html holamundo* (CMD)**

***docker run -dp 3000:80 -v ${PWD}/src:/var/www/html holamundo* (PowerShell)**

***-En Mac:***

***docker run -dp 3000:80 -v "$(pwd)"/src:/var/www/html holamundo* (bash)**

**Es necesario incicarle al comando de Docker que la ruta de la carpeta /src es relativa al lugar desde el que se ejecuta el comando. Para ello, en función del tipo de terminal que usemos, hay que usar un comando de *shell* u otro.**

**e)Comprueba que ahora los cambios en el código se refrescan automáticamente en el *front-end*.**

**Tarea 4 (2 puntos) – A continuación, montaremos un CRUD completo, con base de datos incluida, conectando dos contenedores en red.**

**-Operaciones previas:**

**a)Primero, actualiza el contenido de la carpeta *src* de tu proyecto con el contenido de la carpeta *phpcrud* adjunta.**

**b)Borra los contenedores creados hasta ahora.**

**c)Crea una nueva red en Docker. La utilizaremos para conectar los dos contenedores entre sí:**

***docker network create crud*.**

**(Puedes listar las redes existentes con *docker network ls*).**

**-Arranque del contenedor de MySQL:**

**d)Ahora arrancaremos un contenedor con MySQL a partir de una imagen oficial del** [**Docker Hub**](https://hub.docker.com/search?q=) **(marcada en azul):**

***docker run -d \***

***--network crud --network-alias mysql \***

***-v crud-mysql-data:/var/lib/mysql \***

***-e MYSQL\_ROOT\_PASSWORD=1234 -e MYSQL\_DATABASE=crud \***

***mysql:5.7***

**Atención, se trata de un único comando. Solo se divide en líneas para mejorar la legibilidad.**

**-Observa que:**

**-En este caso no hace falta compilar previamente la imagen: la usamos tal como está (*mysql:5.7)*, descargada directamente desde el repositorio de imágenes oficiales, Docker Hub.**

**-Lanzamos el contenedor asociado a una red (*crud*) y con un alias en esa red (*mysql*), para poder ser identificado fácilmente.**

**-Montamos en el contenedor un volumen externo. El punto de montaje en el sistema de ficheros linux es */var/lib/mysql*, y el nombre del volumen externo es *crud-mysql-data*. En este caso el volumen externo no es ningún directorio de nuestro equipo amfitrión, sino un volumen que Docker se encarga de gestionar. Puedes inspeccionarlo en el panel de control de Docker Desktop.**

**e)Des de Docker Desktop,** [**(o desde el propio VSCode, si tienes instalada la extensión Docker de Microsoft),**](https://learn.microsoft.com/en-us/visualstudio/docker/tutorials/docker-tutorial) **abre el terminal del contenedor con MySQL. Entra en el cliente de *mysql* con el usuario *root*. Comprueba que la base de datos *crud* está creada. Selecciónala y ejecuta el contenido del *script* *crud.sql* que se incluye en el proyecto. Puedes copiar y pegar el contenido directamente al terminal, una vez has entrado en el cliente *mysql*.**

**f) Usa *docker volume ls* para listar los volúmenes. Observa que Docker ha creado un volumen para el uso del contenedor de MySQL, con el nombre que le hemos asignado en el comando de arranque del contenedor. Esto hace que sus datos sean persistentes. Anota el nombre del volumen. También puedes inspeccionarlo en Docker Desktop.**

**-Arranque del contenedor de aplicación:**

**g)Actualiza el Dokerfile para que se instalen en la imagen los conectores de PHP a MySQL. Observa como completamos la imagen a partir de la capa base *php:7.1-apache*:**

FROM php:7.1-apache

# COPY src/ /var/www/html

RUN apt update -y && apt upgrade -y

RUN docker-php-ext-install mysqli && docker-php-ext-enable mysqli

EXPOSE 80

**g)Comprueba que en el archivo *conexion.php* figuran los parámetros de conexión adecuados (host, usuario, contraseña, etc). El *host* debe ser el alias del contenedor de MySQL en la red Docker. Recuerda que no te estarás conectando a tu servidor local de MySQL (si lo tuvieres), sino al contenedor Docker que acabamos de arrancar con una imagen de MySQL.**

**h)Recompila la imagen de tu proyecto con el nombre *crud*.**

**i)Arranca un contenedor con la imagen que acabas de compilar (en azul), ahora indicando que queremos incluir el contenedor en la red *crud*:**

***docker run -dp 3000:80 \***

***-v "%cd%"/src:/var/www/html \***

***--network crud \***

***crud***

**j)Comprueba que la aplicación se presenta adecuadamente en** <http://localhost:3000/>**. Si no es así, deberás depurar los pasos anteriores.**

**2ª parte - Despliegue de un contenedor Docker Compose**

***La aplicación DOCKER COMPOSE permite componer aplicaciones (mono o multi-contenedor) de manera más cómoda. Una APLICACIÓN encapsula distintos SERVICIOS, cada uno de ellos corriendo en un contenedor separado. Generar una aplicación con Docker Compose facilita enormemente el proceso de configuración, compilación y despliegue de una imagen, especialmente en los casos de múltiples contenedores.***

**Tarea 00 – Instalación de Docker Compose:**

**Para ello, sigue las instrucciones en** [**https://docs.docker.com/compose/install/**](https://docs.docker.com/compose/install/)**.**

**Si utilizas la extensión Docker de VSCode, no es necesario instalar separadamente Docker Compose. La extensión implementa los comandos de composición que utilizaremos.**

**Tarea 4 (2 puntos) – Una aplicación generable con Docker Compose se describe en un fichero *docker-compose.yml*. Este fichero sigue el formato** [**YAML**](https://es.wikipedia.org/wiki/YAML) **(que se utiliza también en los ficheros de configuración de Hadoop).**

**En el que se muestra a continuación, se describe una aplicación consistente dos imágenes (que correrán en dos contenedores separados): una imagen customizada (*crud,* que se describe en el *Dockerfile* que ya hemos ido viendo); y una imagen oficial de *mysql*. Nótese que ambas partes de la aplicación se describen como servicios, cada uno de los cuáles lleva asociadas una serie de opciones de configuración (puertos, volúmenes externos, redes a las que está conectado, etc.):**

services:

  crud:

    build:

      context: .

      dockerfile: ./Dockerfile

    ports:

      - 3000:80

    volumes:

      - ./src:/var/www/html

    links:

      - mysql

    networks:

      - default

  mysql:

    image: mysql:5.7

    volumes:

      - crud-mysql-data:/var/lib/mysql

    environment:

      MYSQL\_ROOT\_PASSWORD: root

      MYSQL\_DATABASE: crud

    networks:

      - default

volumes:

  crud-mysql-data:

    external: true

    name: crud-mysql-data

**Observa la correlación entre los comandos que usaste para desplegar la aplicación en la actividad anterior y las directivas de configuración del servicio en el *docker-compose.yml.***

**a)¿Dónde se describe la imagen que corresponde al servicio *crud*?**

**b)¿Por qué crees que se utiliza la directiva *image* para describir la imagen del servicio *mysql*, mientras que en el servicio *app* se utiliza la directiva *build*?**

**Observa que utilizamos como volumen del contenedor *mysql* un volumen externo. Asegúrate de que utilizas el volumen que inspeccionaste en la Tarea 4.f. Esto permitirá que la aplicación trabaje con una base de datos ya inicializada.**

**c)El fichero *docker-compose.yml* debe incluirse en el directorio de proyecto, junto con el fichero *Dockerfile*. Una vez descrita la aplicación en el YAML, ésta puede desplegarse solamente en un paso, con el comando**

***docker-compose up -d***

**d)Muestra la aplicación funcionando en** [**http://localhost:3000**](http://localhost:3000)

**e)¿Para que se utiliza el la opción *-d* en el comando?**

**Tarea 5 (1 punto) – a)Puedes replegar la aplicación (esto es, parar su ejecución y eliminarla de memoria), com el comando**

***docker-compose down***

**b)Vuelve a desplegarla mediante el comando de la tarea anterior, y comprueba cómo la aplicación aparece de nuevo en el panel de control de Docker Desktop.**

**c)¿Se guarda, entre sucesivas ejecuciones de la aplicación, la información introducida en la base de datos? ¿Por qué?**

***ATENCIÓN: Los contenedores son procesos que consumen memoria. Hay que acostumbrarse a pararlos y eliminarlos cuando ya no se utilicen.***

***Las imágenes son ficheros que ocupan mucho espacio en disco. Hay que acostumbrarse a borralos cuando no se utilicen. No hay que sufrir por volverlos a necesitar. En los ficheros de configuración (Dockerfile, docker-compose.yml) está toda la información necessaria para volver a levantar los contenedores.***

***Otra cosa pasa con los volúmenes externos. Estos sólo deben ser eliminados si no queremos persistir la información entre sucesivas ejecuciones de la aplicación.***

***Puedes gestionar las tres cosas cómodamente desde el panel de control de Docker Desktop.***

**ANEXO – Otro ejemplo: *docker-compose.yml* para levantar una pila LAMP (Linux + Apache + MySQL + PHP) que sirve Wordpress.**

services:

  db:

    container\_name: assetsdb

    image: mysql

    volumes:

      - ./dump:/docker-entrypoint-initdb.d

      - persistent:/var/lib/mysql

    restart: always

    ports:

      - '3309:3306'

    environment:

      MYSQL\_ROOT\_PASSWORD: root

      MYSQL\_DATABASE: asset\_db

      MYSQL\_USER: asset\_user

      MYSQL\_PASSWORD: asset\_pwd

  wordpress:

    # depends\_on:

    #  - db

    links:

      - db

    image: wordpress:latest

    volumes:

      - ./wp-content:/var/www/html/wp-content

      - ./apache-logs:/var/log/apache2

      - ./.docker/php-apache.conf:/etc/apache2/sites-enabled/000-default.conf

      - ./.docker/uploads.ini:/usr/local/etc/php/conf.d/uploads.ini

    ports:

      - "80:80"

    restart: always

    environment:

      WORDPRESS\_DB\_HOST: assetsdb

      WORDPRESS\_DB\_USER: asset\_user

      WORDPRESS\_DB\_PASSWORD: asset\_pwd

      WORDPRESS\_DB\_NAME: assetsdb

      WORDPRESS\_TABLE\_PREFIX: wp\_

      WORDPRESS\_DEBUG: 0

      DISABLED\_PLUGINS:

  phpmyadmin:

    image: phpmyadmin

    restart: always

    ports:

      - 8089:80

    links:

      - db:db

    environment:

      MYSQL\_USER: asset\_user

      MYSQL\_PASSWORD: asset\_pwd

      MYSQL\_ROOT\_PASSWORD: root

volumes:

  persistent: