Centro de Procesamiento de Datos

Práctica 2. Contenedores Docker (II)

Objetivo:
Ampliar conocimiento sobre contenedores Docker.

Entrega:
Presentar un documento pdf en SWAD→Actividades→ Práctica 2 con la siguiente información:

- Crear una imagen personalizada de Docker con Apache de forma que cuando se acceda a http://localhost:8888 aparezca vuestro nombre (editando el index.html) (según apartado I).
- A partir de los pasos explicados en el apartado II subir la imagen a hub.docker.com e indicar el nombre de la imagen creada.
- Contenido del fichero Dockerfile personalizado del apartado II y ficheros utilizados.
- Según el apartado III, una vez desplegado el servidor Wordpress, editar la página principal para que aparezca el nombre del usuario y realizar una captura de pantalla.

Opcional:
- Siguiendo los pasos del apartado IV, ejecute el test que permita evaluar el tiempo de ejecución de un benchmark, y comprobar cómo podemos aumentar o reducir la CPU dedicada y por tanto el tiempo de ejecución. Prepare un contenedor con dicho experimento, súbalo a hub.docker.com e indique en el documento los pasos que realiza para el experimento y los tiempos de ejecución obtenidos.

Rúbrica:

Capturas de pantalla	30%
Descripción del trabajo desarrollado	20%
Entrega de apartados opcionales	20%
Entrega en plazo	30%

Desarrollo:
Esta práctica estudiamos:

- Cómo automatizar la creación de contenedores
- Utilizar el repositorio hub.docker.com
- El uso de múltiples contenedores con Docker-compose para crear escenarios más complejos basados en varias aplicaciones
- El control de recursos en contenedores

I) Creación de contenedores basada en Dockerfile

Los ficheros Dockerfile permiten automatizar el proceso de creación de contenedores. El manual de referencia de Dockerfile lo podemos encontrar en:
https://docs.docker.com/engine/reference/builder/

Creamos un directorio y dentro creamos el fichero Dockerfile

```
FROM debian
MAINTAINER Usuario CPD "usuario@ugr.es"
RUN apt-get update && apt-get install -y apache2 && apt-get clean && rm -rf /var/lib/apt/lists/*

ENV APACHE_RUN_USER www-data
ENV APACHE_RUN_GROUP www-data
ENV APACHE_LOG_DIR /var/log/apache2
EXPOSE 80
ADD ["index.html","/var/www/html/"]
ENTRYPOINT ["/usr/sbin/apache2ctl", "-D", "FOREGROUND"]
```

También creamos nuestro fichero *index.html* personalizado.

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>Prueba CPD</title>
</head>
<body>
   <h1>Prueba inicial CPD</h1>
</body>
</html>
```

Posibles entradas del fichero:

- **MAINTAINER:** Identifica datos del autor, nombre y dirección de correo electrónico
- **ENV**: Configura las variables de entorno, clave, valor.
- **ADD**: Esta instrucción se encarga de copiar los ficheros y directorios desde una ubicación especificada y los agrega al sistema de ficheros del contenedor. Si se trata de añadir un fichero comprimido, al ejecutarse el guión lo descomprimirá de manera automática.
- **COPY**: Permite copiar ficheros.
- **EXPOSE**: Indica los puertos en los que va a escuchar el contenedor. Hay que combinarlo con la opción ***-p*** de `docker run`
- **VOLUME**: Permite utilizar en el contenedor una ubicación de nuestro *host,* y poder almacenar datos de manera permanente. Los volúmenes de los contenedores siempre son accesibles en el *host* anfitrión, en la ubicación: `/var/lib/docker/volumes/`
- **WORKDIR**: El directorio por defecto donde se ejecutan las órdenes.
- **USER**: Por defecto, todas las acciones son realizadas por el usuario *root*. Se puede indicar un usuario diferente.
- **SHELL**: En los contenedores, el punto de entrada es el comando `/bins/sh -c` para ejecutar los comandos específicos en CMD, o los comandos especificados en línea de comandos para la acción *run*.
- **ARG**: Se pueden añadir parámetros a nuestro Dockerfile durante la construcción del contenedor.

Recomendaciones para crear Dockerfiles:
https://docs.docker.com/develop/develop-images/dockerfile_best-practices/

Para construir

```
docker build -t pruebacpd .
```

Para ejecutar:

```
docker run --name x1 -p8080:80  -d pruebacpd
```

Si queremos acceder de forma interactiva dentro del contenedor:

```
docker exec -it pruebacpd /bin/sh
```

II) Publicar nuestra imagen en Docker hub

Docker hub es un repositorio que puede almacenar contenedores de forma pública y privada.
Creamos una cuenta (gratis) en https://hub.docker.com

Podemos acceder a nuestros repositorios mediante el navegador y desde nuestra consola podemos asociarnos con (utilizamos nuestro nombre de usuario y clave):

```
docker login
```

Vamos a crear una nueva imagen basada desde una imagen alpine que requiere poco recursos.
Creamos un directorio nuevo y un fichero Dockerfile:

```
FROM alpine:3.20

RUN apk update
RUN apk add curl
RUN apk add vim
```

Creamos el contenedor

```
docker build -t usuariocpd/cpd1:1.0 .
```

El . final indica el directorio donde se encuentra el Dockerfile

Subimos la imagen al repositorio con:

```
docker push usuariocpd/cpd1:1.0
```

Donde **usuariocpd** es el nombre de la cuenta que hemos creado en Docker hub.

Podemos realizar modificaciones, ej: añadiendo una línea a nuestro Dockerfile

```
RUN apk add git
```

Reconstruimos la imagen y podemos asignar un nuevo tag (también podríamos utilizar el mismo tag para actualizar nuestra imagen anterior):

```
docker build -t usuariocpd/cpd1:1.1 .
docker pull
```

Docker crea imágenes cache intermedias para acelerar el proceso de creación en cada etapa, por eso conviene “reducir” el número de líneas combinándolas en nuestro Dockerfile:

```
FROM alpine:3.20
RUN apk update && \
    apk add curl && \
    apk add vim && \
    apk add git
```

O bien:

```
FROM alpine:3.20

RUN apk update && apk add \
    curl \
    git \
    vim
```

También podemos utilizar la opción --no-cache en docker build

III) Múltiples contenedores con docker-compose

Podemos combinar múltiples contenedores para crear soluciones más complejas que requieren múltiples aplicaciones.
Como ejemplo vamos a crear un entorno de Wordpress basado en dos elementos principales: una base de datos Mysql y el propio Wordpress.

Creamos un directorio y el fichero *docker-compose.yml*

```
version: '3.3'

services:
   db:
     image: mysql:5.7
     volumes:
       - db_data:/var/lib/mysql
     restart: always
     environment:
       MYSQL_ROOT_PASSWORD: somewordpress
       MYSQL_DATABASE: wordpress
       MYSQL_USER: wordpress
       MYSQL_PASSWORD: cpdwordpress

   wordpress:
     depends_on:
       - db
     image: wordpress:latest
     ports:
```

```
      - "8000:80"
    restart: always
    environment:
      WORDPRESS_DB_HOST: db:3306
      WORDPRESS_DB_USER: wordpress
      WORDPRESS_DB_PASSWORD: cpdwordpress
volumes:
   db_data: {}
```

Construimos

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

Podemos comprobar que está funcionando:

```
docker-compose ps
```

Accedemos a nuestro nuevo servidor Wordpress mediante http://localhost:8000
Y creamos nuestra primera página que personalizamos:

Si queremos parar:

```
docker-compose stop
```

Reiniciar:

```
docker-compose start
```

Parar y borrar contenedores:

```
docker-compose down
```

Para ver los volúmenes:

```
docker volume ls
docker volume inspect cpd24_25_db_data
```

Si queremos parar y borrar contenedores y la base de datos (que está almacenada como volumen):

```
docker-compose down --volumes
```

IV) Limitando el uso de CPU de los contenedores

Podemos limitar el uso de CPU y de memoria de un contenedor cuando lanzamos la ejecución con docker run.
Ej:

```
docker run -it --cpus=".5" ubuntu /bin/bash
```

En https://docs.docker.com/config/containers/resource_constraints/#configure-the-default-cfs-scheduler
podemos encontrar la información sobre distintas formas de controlar dichos recursos.

Instale el programas sysbench (con apt) y ejecute

```
sysbench --test=cpu --cpu-max-prime=20000 run
```

Comprobar cómo podemos aumentar o reducir la CPU dedicada y por tanto el tiempo de ejecución.

Prepare un contenedor con dicho experimento, súbalo a hub.docker.com e indique en el documento los pasos que realiza para el experimento y los tiempos de ejecución obtenidos.