Sistemas de Gestión de Seguridad de la Información 2021/2022

Laboratorio 0 – Docker

Este laboratorio no entra en el examen, pero es necesario para la entrega 1.

Objetivos:

1. Obtener un conocimiento básico de los elementos más importantes de Docker y su funcionamiento: imágenes, containers, repositorios, volúmenes, etc.
2. Hacer un despliegue simple de varios servicios mediante docker-compose.

Recursos necesarios:

- Ubuntu (Usuario/password: lsi/lsi).
- Archivos Docker disponibles en eGela.
- Proyecto básico docker-compose disponible en GitHub.

Índice:

1.- Gestionar imágenes

Docker tiene un repositorio local que contiene las imágenes que vamos a usar en nuestro ordenador.

- Puedes ver la imágenes disponibles en tu repositorio local mediante `$ docker images`

El repositorio remoto más común se encuentra en Docker Hub[1], que es el repositorio configurado por defecto al instalar Docker[2].

- Explora las imágenes que se pueden encontrar en Docker Hub.

Vamos a descargar una imagen del repositorio remoto al repositorio local:

- Pincha en el enlace de la imagen **hello-world**.
- Como se indica en la página, descarga la imagen al repositorio local: `$ docker pull hello-world`

¿Con qué comando se suben imágenes a Docker Hub desde nuestro repositorio local?

1 http://hub.docker.com
2 En muchas empresas configuran sus propios repositorios Docker para facilitar el despliegue de aplicaciones en sus clientes.

2.- Ejecutar containers

En Docker, los containers se ejecutan a partir de una imagen.

- Ejecuta un container a partir de la imagen **hello-world**: `$ docker run hello-world`

¿Qué output nos da la ejecución del container?

- Ejecuta `$ docker run -it ubuntu bash`

¿De dónde sale la imagen **ubuntu**?

¿Qué diferencia hay entre **docker run** y **docker run -it**?

¿Por qué ha cambiado el prompt de la terminal?

¿Si hacemos un listado mediante **ls**, a qué máquina pertenecen los directorios?

¿Qué output nos da el comando **docker ps -a**?

Para parar los containers, necesitamos su nombre o id:

- Ejecuta `$ docker kill nombre|id` (Después ejecuta `$ docker ps -a`)

Aunque los containers no están funcionando, hay que eliminarlos:

- Ejecuta `$ docker rm nombre|id` (Después ejecuta `$ docker ps -a` y `$ docker images`)

¿Qué diferencia hay entre parar y borrar un container?

¿Cómo afecta a la imagen de la que ha surgido el container?

¿Cómo se borra una imagen?

Vuelve a ejecutar un container desde la imagen **ubuntu**:

- `$ docker run -it ubuntu bash`
- En otra terminal, ejecuta `$ docker exec nombre_container ls`

¿Qué diferencia hay entre **run** y **exec**?

3.- Construir imágenes

Para construir una imagen Docker necesitamos un Dockerfile. Un Dockerfile es un archivo de texto plano que le dice a Docker cómo tiene que construir la imagen. Por ejemplo[3]:

- **FROM**: la imagen base a usar.
- **ADD**: añade archivos locales a la imagen.
- **RUN**: ejecuta comandos.

```
FROM ubuntu
ADD msg /opt/
RUN apt-get update && apt-get install less
RUN date >> /opt/msg
CMD ["cat","/opt/msg"]
```

3 El proceso de construcción de una imagen es más complejo de lo que aquí se indica (Cache, containers intermedios, otras instrucciones, etc.).

- **CMD**: el comando que se ejecutará al arrancar el container a partir de la imagen descrita en el Dockerfile.

Cuando ejecutemos un container a partir de esta imagen, ¿Que output vamos a obtener? ¿Por qué?

Vamos a construir una imagen a partir del Dockerfile que se encuentra en eGela:

- Baja el Dockerfile de eGela, junto al archivo **msg** que contiene un mensaje.
- Ejecuta `$ docker build -t="nombre" .` en el mismo directorio (El nombre puede ser cualquiera).
- Comprueba que la imagen ha sido construida y añadida al repositorio local mediante `$ docker images`
- Ejecuta un container de la imagen que acabamos de construir mediante `$ docker run nombre`

¿Qué output nos da al ejecutar el container? ¿Cómo cambiarías el mensaje que se obtiene?

Docker nos permite montar directorios que son compartidos por el host y el container[4], es decir que ambos pueden leer y escribir en esos directorios. Para probarlo:

- Crea un directorio llamado **dir-msg** que contenga un archivo **msg2** con la cadena “iep”: `$ mkdir dir-msg && echo "iep" > dir-msg/msg2`
- Ejecuta un container a partir de la imagen ubuntu, de manera interactiva, montando el directorio **dir-msg** dentro del container en el directorio **/app**: `$ docker run -it -v "$(pwd)"/dir-msg:/app ubuntu bash`
- Una vez dentro del container, asegúrate de que se ha montado correctamente mediante `$ cat app/msg2`
- En otra terminal, cambia el contenido de **dir-msg/msg2** y vuelve a ejecutar `$ cat app/msg2` dentro del container.

¿Ha cambiado el contenido del archivo? ¿Por qué?

Los containers son, por definición, efímeros y de una existencia muy corta. Por lo tanto los volúmenes en Docker son muy importantes ya que nos permiten persistir datos que de otra manera se perderían al borrar el container.

A la hora de desarrollar aplicaciones que se van a desplegar mediante Docker es muy común trabajar de la siguiente manera:

1. Montar directorio de desarrollo con la aplicación y el directorio con los datos en el container.
2. Desarrollar y hacer pruebas.
3. Cuando obtengamos una versión estable de la aplicación, añadirla al Dockerfile.

4 Hay varios métodos para montar volumenes. Aquí nos referimos al bind-mount.

4.- Ejecutar servicios

Mediante **docker-compose** podemos definir un grupo de servicios que se ejecuten a la vez de manera coordinada, basándose cada servicio en una imagen Docker. Teniendo en cuenta que hoy en día muchas aplicaciones se basan en combinaciones de servicios (Base de datos, servidor web, otros servicios, etc.) ésta es una característica muy importante.

Vamos a ver como funciona **docker-compose** explorando el proyecto que tenéis que usar como base para la entrega 1. El proyecto consta de tres servicios que conforman una aplicación web muy sencilla: un servidor Web con una aplicación PHP (La aplicación Web propiamente dicha) que accede a una Base de Datos Maria DB, la Base de Datos Maria DB, y un servidor Web con la aplicación PHPMyAdmin para gestionar la Base de Datos Maria DB.

- Clona el repositorio de GitHub que contiene el proyecto (O descárgalo): `$ git clone https://github.com/mikel-egana-aranguren/docker-lamp.git`

El archivo **docker-lamp/docker-compose.yml** contiene la definición de los servicios. En este caso hay tres servicios (El nombre del servicio y de la imagen Docker en el que se basa pueden ser diferentes):

```
web:
  image: web
  environment:
    - ALLOW_OVERRIDE=true
  ports:
    - "81:80"
  links:
    - db
  volumes:
    - ./app:/var/www/html/

db:
  image: mariadb
  restart: always
  volumes:
    - ./mysql:/var/lib/mysql
  environment:
    MYSQL_ROOT_PASSWORD: root
    MYSQL_USER: admin
    MYSQL_PASSWORD: test
    MYSQL_DATABASE: database
  ports:
    - "8889:3306"

phpmyadmin:
  image: phpmyadmin/phpmyadmin:latest
  links:
    - db
  ports:
    - 8890:80
  environment:
    MYSQL_USER: admin
    MYSQL_PASSWORD: test
    MYSQL_DATABASE: database
```

- **web**: este servicio se basa en la imagen “web” construida a partir del Dockerfile, que contiene un servidor web Apache y una aplicacion PHP definida en /app (Esta imagen es simplemente una extensión de la imagen oficial de PHP[5]). Se enlaza al servicio **db** y redirige el puerto 81 del host al puerto 80 del container (Donde se ejecuta Apache).
- **db**: la imagen mariadb es la imagen oficial que provee la base de datos Maria DB. En este caso el servicio se ejecuta obteniendo los datos del volumen ./mysql (Es decir que si volvemos a ejecutar un container, los datos se cargarán de ese directorio y no se perderán, aunque el container haya desaparecido), con la configuración de la sección “environment” y redirigiendo el puerto del host 8889 al puerto del container 3306.
- **phpmyadmin**: este servicio se basa en la imagen oficial de PHPMyAdmin, que se conecta al servicio **db** y se usa para administrar la base de datos, redirigiendo el puerto del host 8890 al puerto del container 80.

Para desplegar el proyecto:

- Sitúa la terminal dentro del repositorio /docker-lamp.
- Construye la imagen **web**: `$ docker build -t="web" .`
- Despliega los servicios mediante `$ docker-compose up`
- Visita la web en http://localhost:81.
- Para añadir los datos necesarios, visita http://localhost:8890/ (Tal y como lo hemos definido en docker-compose.yml, usuario “admin”, password “test”). Haz click en “database” y luego en “import”, desde donde elegimos el archivo docker-lamp/database.sql.

5 Uno de los puntos fuertes de Docker es poder extender imágenes ya existentes en nuestro Dockerfile.

- Vuelve a http://localhost:81, debería tener más información.
- Para parar los servicios, en otra terminal, `$ docker-compose down`