EXAMEN CHRISTIAN MILLÁN SORIA

1.a. ¿Es Docker lo mismo que una máquina virtual? Explica tu respuesta.

Una máquina virtual es un sistema que crea un entorno aislado en el que se ejecuta un sistema operativo completo y sus aplicaciones, como si estuvieran en un ordenador separado. Esto permite ejecutar aplicaciones en diferentes sistemas operativos y entornos de manera aislada y segura.

Por otro lado, Docker utiliza los llamados "contenedores". En lugar de crear un sistema operativo completo y aislado, los contenedores de Docker aislan solo la aplicación y sus dependencias. Esto hace que los contenedores sean más ligeros y más rápidos que las máquinas virtuales, a costa de un menor nivel de aislamiento.

1.b. ¿Qué es una imagen Docker?

Una imagen Docker es un archivo que contiene todo lo necesario para ejecutar una aplicación en un contenedor Docker. Incluye el código de la aplicación, las bibliotecas, los archivos de configuración y todo lo necesario para que la aplicación funcione de manera independiente en un contenedor Docker.

Las imágenes Docker son el núcleo de Docker y se utilizan para crear y ejecutar contenedores. Al crear un contenedor, Docker descarga la imagen y la usa como base para crear una instancia única y aislada del contenedor.

2.a. ¿Qué es Docker Hub?

Docker Hub es un servicio en la nube de Docker que permite a los desarrolladores compartir y descargar imágenes Docker. Es un repositorio público de imágenes Docker que se pueden utilizar para crear contenedores en cualquier máquina que tenga Docker instalado.

Docker Hub permite a los desarrolladores publicar y compartir sus imágenes con la comunidad, lo que facilita la colaboración y la reutilización de código.

En resumen, Docker Hub es una plataforma centralizada para la gestión y el intercambio de imágenes Docker, que permite a los desarrolladores colaborar y compartir imágenes de manera sencilla y eficiente.

2.b. ¿Qué son los contenedores?

Los contenedores de Docker son una forma de empaquetar y distribuir aplicaciones y sus dependencias.

Los contenedores de Docker son muy útiles porque permiten a los desarrolladores empaquetar y distribuir aplicaciones de manera consistente y segura. Esto significa que los desarrolladores pueden estar seguros de que la aplicación funcionará de la misma manera en cualquier máquina que tenga Docker instalado, como hemos podido ver en clase.

3. Describe qué hacen los siguientes comandos:

a.

```
docker image pull debian:bullseye
```

Este comando descarga una imagen del sistema operativo "Debian" de Docker Hub. Además especifica que la versión a descargar de la imagen es la "bullseye".

b.

```
docker image ls | grep Ubuntu
```

Este comando muestra todas las imágenes relacionadas con Ubuntu (las filtra con el "grep"), es decir, todas las imágenes cuyo nombre o versión correspondan a Ubuntu.

c.

```
docker run --name Debian -d -it debian:bullseye
```

Este comando crea un nuevo contenedor Docker con una serie de parámetros:

- "--name Debian": establece un nombre para el contenedor, en este caso el nombre es "Debian"
- "-d": Devuelve el terminal al usuario, es decir, el contenedor se ejecuta en segundo plano y no se queda a la espera de mostrar logs ni nada por el estilo, sino que permite que el usuario pueda seguir utilizando el mismo terminal
- "-it": establece que lo que quiere el usuario es ejecutar el contenedor de forma interactiva, es decir, poder utilizar el terminal de comandos del contenedor una vez este se inicie

Por último, el "debian:bullseye" establece la imagen Docker que se va a utilizar para crear dicho contenedor.

d.

```
docker exec -it Debian bash
```

Este comando permite conectar/dejar utilizar el contenedor al usuario, es decir, con el "exec" declara esto mismo, con el "-it" establece que será de forma interactiva (para poder utilizar su termina de comandos), "Debian" es el nombre del contenedor y el "bash" es el shell con el que se quiere utilizar este contenedor.

e.

```
docker stop ac ab fg && docker rm ac ab fg
```

Esta línea ejecuta dos comandos diferentes:

- El primero para los contenedores cuyos tokens/IDs empiezan por "ac", "ab" y "fg"
- El segundo, separado del primero por "&&", elimina los contenedores cuyos tokens/IDs empiezan por "ac", "ab" y "fg", es decir, los contenedores que ya han sido parados por el comando anterior, ya que para

eliminar un contenedor es necesario pararlo primero

f.

```
docker build –t mi_propia_imagen:v01 .
```

El comando "docker build -t mi_propia_imagen:v01 ." crea una nueva imagen de Docker a partir de un archivo Dockerfile en el directorio actual:

- "docker build" es el comando para construir una nueva imagen de Docker
- "-t mi_propia_imagen:v01" le da a la imagen el nombre y la etiqueta "mi_propia_imagen:v01"
- El punto "." al final del comando indica el contexto de construcción, es decir, el directorio en el que se encuentra el archivo Dockerfile
- El archivo Dockerfile contiene las instrucciones para construir la imagen de Docker. La imagen resultante incluirá todos los recursos y configuraciones especificadas en el Dockerfile

g.

```
docker login -u2smr && docker search mariadb
```

El comando "docker login -u2smr && docker search mariadb" inicia sesión en un registro de Docker y busca una imagen en el registro.

- "docker login -u2smr" inicia sesión en un registro de Docker con el nombre de usuario "2smr"
- "docker search mariadb" busca una imagen en el registro de Docker que tenga "mariadb" en su nombre o etiqueta
- El operador "&&" indica que los comandos "docker login" y "docker search" deben ejecutarse uno después del otro y que el segundo solo se ejecutará si el primero se completa correctamente. Esto significa que primero se iniciará sesión en el registro de Docker y luego se buscará la imagen

h.

```
docker tag mi_propia_imagen:v01 2smr/miweb:v01
```

El comando "docker tag mi_propia_imagen:v01 2smr/miweb:v01" etiqueta una imagen de Docker con un nuevo nombre y etiqueta.

- *"docker tag" es el comando para etiquetar una imagen de Docker**
- "mi_propia_imagen:v01" es el nombre y la etiqueta actual de la imagen
- "2smr/miweb:v01" es el nuevo nombre y la etiqueta para la imagen. El nombre incluye el prefijo de usuario "2smr" y la etiqueta "v01"

Esta operación de etiquetado permite referenciar la imagen con un nombre y etiqueta diferentes y puede ser útil, por ejemplo, para compartir la imagen con otros usuarios o publicarla en un registro de Docker

i.

```
docker push 2smr/miweb:v01
```

El comando "docker push 2smr/miweb:v01" envía una imagen de Docker a un registro de Docker.

- "docker push" es el comando para enviar una imagen de Docker a un registro de Docker
- "2smr/miweb:v01" es el nombre y la etiqueta de la imagen que se va a enviar. Incluye el prefijo de usuario "2smr" y la etiqueta "v01"

---

4. Usa el contenido de la carpeta "~/miweb" para montar un contenedor servidor web (puerto 80). Condición: si se cambia el código del fichero "index.html", estos cambios se reflejarán automáticamente al refrescar el navegador.

Lo primero es descargar la imagen de Nginx para poder crear el contenedor servidor web:

```
docker pull nginx
```

```
ubuntu@ip-172-31-58-100:~/miweb$ docker pull nginx
Using default tag: latest
latest: Pulling from library/nginx
8740c948ffd4: Pull complete
d2c0556a17c5: Pull complete
c8b9881f2c6a: Pull complete
693c3ffa8f43: Pull complete
8316c5e80e6d: Pull complete
b2fe3577faa4: Pull complete
Digest: sha256:b8f2383a95879e1ae064940d9a200f67a6c79e710ed82ac42263397367e7cc4e
Status: Downloaded newer image for nginx:latest
docker.io/library/nginx:latest
```

Lo siguiente es crear el contenedor y establecer la carpeta con el contenido en el directorio predeterminado de Nginx para páginas web.

```
docker run -d -p 80:80 -v ~/miweb:/usr/share/nginx/html nginx
```

Y ahora se puede visitar en un navegador para ver la página web.

De forma predeterminada, en el "index.html", hay un "h1" que dice "Mi Primera pagina web".

```
ubuntu@ip-172-31-58-100:~/miweb$ cat index.html
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html><head><meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
<title>Mi primera pagina web </title>
<style type="text/css">
<!--
.Estilo1 {color: #0000FF}
-->
</style>
</head>
<body>
<h1 align="center"><font color="#0000FF">Mi Primera pagina web </font></h1>
<hr>
<p>Esto tan sencillo es una verdadera pagina web, aunque le falta el contenido, pero todo llegar.</p>
<p>Vamos a modificar esta pagina para ver si funciona. Esta modificacion se vera reflejada a la voz de ya</p>

</body></html>
```

Para comprobar que si se hace un cambio en el archivo "index.html" se cambia el contenido de la página, utilizo el comando "nano" y edito el mismo. Así es como se ve la página después de mi cambio:

```
nano index.html
```

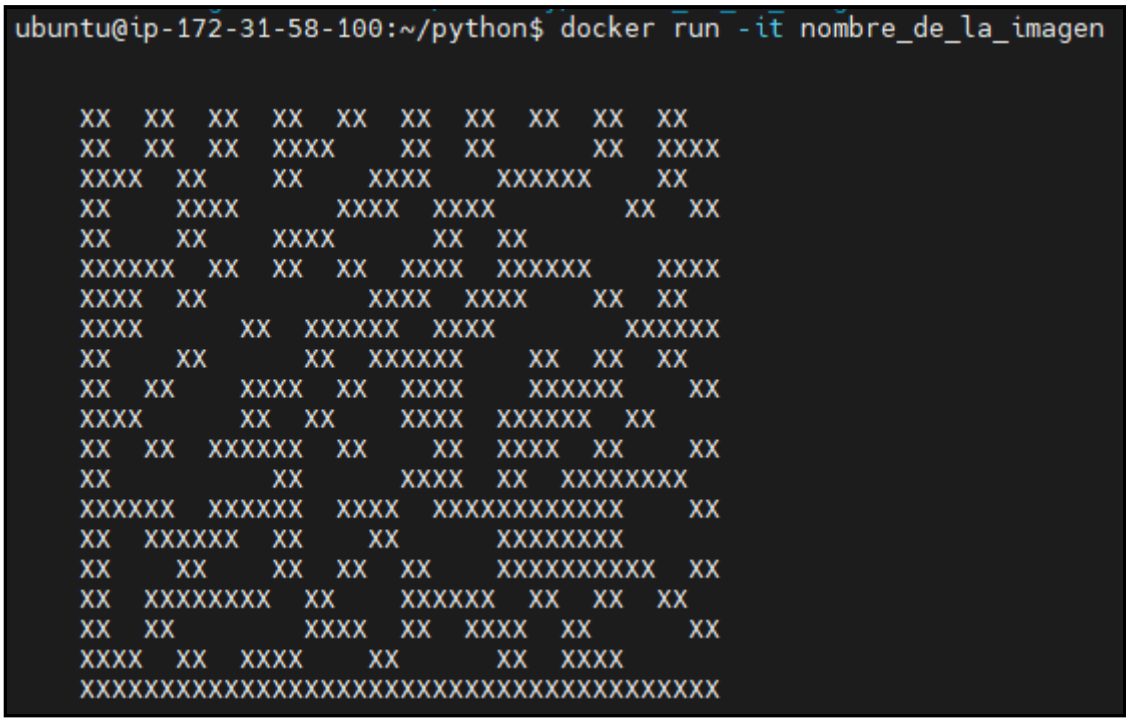
5.a. Crea una imagen usando el fichero Dockerfile del directorio "python" y ejecuta dicha imagen devolviendo el control en la consola de comandos.

Me sitúo en el directorio "python" y compruebo que el contenido del Dockerfile es correcto. Una vez he hecho esto ejecuto el siguiente comando:

```
docker build -t nombre_de_la_imagen .
```

Y por último creo un contenedor nuevo utilizando esta imagen recién generada. El contenedor muestra por pantalla un código QR generado por el archivo "codigo.py".

```
docker run -it nombre_de_la_imagen
```



5.b. Crea una base de datos basada en mariadb:latest.

Primero creo un directorio en el home para trabajar en él:

```
mkdir mariadb-data
```

Desde fuera de él, ejecuto el siguiente comando de Docker para crear el contenedor con todas las variables de entorno necesarias para cumplir los apartados del examen:

```
docker run --name bbdd -p 3336:3306 -e MARIADB_ROOT_PASSWORD=root -e
MARIADB_DATABASE=prueba -e MARIADB_USER=invitado -e MARIADB_PASSWORD=invitado -v
$PWD/mariadb-data:/var/lib/mysql -d mariadb:latest
```

Me conecto al contenedor y dentro de él ejecuto el comando "show databases;". Esto se puede hacer todo en una misma línea de comando con el argumento "-e":

```
docker exec -it bbdd mysql -uroot -proot -e "show databases;"
```

```
ubuntu@ip-172-31-58-100:~$ docker exec -it bbdd mysql -uroot -proot -e "show databases;"
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| prueba             |
| sys                |
+--------------------+
```