Practica 4 – 2

1. Utilizando sus palabras, describa qué es Docker y enumere al menos dos beneficios que encuentre para el concepto de contenedores.

   Docker es un software que permite crear y ejecutar aplicaciones en contenedores aislados.

   - Beneficios:
     - Menos sobrecarga
       - Los contenedores requieren menos recursos del sistema que los entornos de máquinas virtuales tradicionales o de hardware porque no incluyen imágenes del sistema operativo.
     - Mayor portabilidad
       - Las aplicaciones que se ejecutan en contenedores se pueden poner en marcha fácilmente en sistemas operativos y plataformas de hardware diferentes.
     - Funcionamiento más constante
       - Los equipos de DevOps saben que las aplicaciones en contenedores van a ejecutarse igual, independientemente de dónde se pongan en marcha.
     - Mayor eficiencia
       - Los contenedores permiten poner en marcha, aplicar parches o escalar las aplicaciones con mayor rapidez.
     - Mejor desarrollo de aplicaciones
       - Los contenedores respaldan los esfuerzos ágiles y de DevOps para acelerar los ciclos de desarrollo, prueba y producción.

2. ¿Qué es una imagen? ¿Y un contenedor? ¿Cuál es la principal diferencia entre ambos?

   Imagen: es una plantilla de solo lectura y estática que contiene instrucciones para crear un contenedor. Sirve como base para crear contenedores.

Contenedor: es una instancia ejecutable y mutable de una imagen. Representa la aplicación en ejecución.

Al crear/eliminar un contenedor, se agrega/elimina una capa de lectura/escritura no presente en la imagen, llamada capa contenedora. A partir de una única imagen se pueden crear varios contenedores que comparten las capas de solo lectura de la imagen, pero que tienen su propia capa individual donde pueden realizar escrituras.

	Imagen de Docker	***Contenedor de Docker***
¿Qué es?	Un archivo que se puede volver a utilizar y se puede compartir, y que se usa para crear contenedores.	Una instancia de ejecución; un *software* autónomo.
Creado a partir de	un código de *software*, dependencias, bibliotecas y un Dockerfile.	Una imagen.
Composición	Capas de solo lectura.	Capas de solo lectura con una capa de lectura y escritura adicional en la parte superior.
Mutabilidad	Inmutable. Si hay cambios, debe crear un archivo nuevo.	Mutable; puede realizar cambios durante el tiempo de ejecución según sea necesario.
¿Cuándo utilizar?	Para almacenar los detalles de configuración	Para ejecutar la aplicación.

	de la aplicación como una plantilla.	

3. ¿Qué es Union Filesystem? ¿Cómo lo utiliza Docker?

   Es un mecanismo de montajes, no un nuevo file system. Permite que varios directorios sean montados en el mismo punto de montaje, apareciendo como un único file system. Permiten montar varias carpetas y crear una unión de sus contenidos, donde si un fichero se repite entre capas se usa el de la capa superior. Docker utiliza varias capas, siendo las capas inferiores read-only y la capa superior es la writable:

   - Las capas se montan una sobre otra (stacking)
   - Solo la última es R/W (la capa del contenedor). Las demás de solo lectura
   - Docker usa “storage drivers” para almacenar capas de una imagen y para almacenar datos en la capa escribible de un contenedor
   - Cada capa representa una instrucción en el archivo Dockerfile de la imagen
   - Cada capa es un conjunto de diferencias con la anterior

4. ¿Qué rango de direcciones IP utilizan los contenedores cuando se crean? ¿De dónde la obtiene?

   Docker usa la subred predeterminada 172.17. 0.0/16 para los contenedores.
   El demonio de Docker se encarga de crear subredes dinámicas y asignar direcciones IP a los contenedores. Además, cada red tiene una máscara de subred y un gateway predeterminado. Las direcciones IP se asignan mediante un servidor DHCP interno administrado por Docker.
   Los usuarios pueden crear sus propias redes personalizadas con rangos de IP específicos utilizando el comando docker network create.

5. ¿De qué manera puede lograrse que las datos sean persistentes en Docker? ¿Qué dos maneras hay de hacerlo? ¿Cuáles son las diferencias entre ellas?

Docker tiene dos opciones para almacenar datos en el host y que los datos sean persistentes

Docker volume	***Bind mount***
Docker volume es el método recomendado para almacenar datos creados y utilizados por contenedores de Docker.	Bind mount existe desde las primeras versiones de Docker. Son menos útiles que los volúmenes
Se puede interactuar con ellos mediante CLI y API.	No se pueden acceder mediante comandos CLI. Sin embargo, se puede trabajar con ellos en la maquina host
Solo se necesita el nombre del volumen para montarlo.	Se debe proporcionar una ruta a la maquina host. Si no existe, Docker la crea automáticamente
Su contenido no debería ser modificado por otros procesos, debe mantenerse aislado	Pueden ser modificados por procesos que no sean de Docker

Docker volume	***Bind mount***
Se almacenan en /var/lib/docker/volumes	Se pueden almacenar en cualquier lado de la computadora host.
Administrado por Docker	Dependen del SO y estructura de directorios de la maquina host
Son compatibles con contenedores de Windows y Linux. Compartir volúmenes entre varios contenedores es más seguro.	Son más difíciles de transferir o realizar copias de seguridad que los volúmenes

Taller

1. Instale Docker CE (Community Edition) en su sistema operativo. Ayuda: seguir las instrucciones de la página de Docker. La instalación más simple para distribuciones de GNU/Linux basadas en Debian es usando los repositorios.

   Segui los pasos de https://docs.docker.com/engine/install/debian/#install-using-the-repository

2. Usando las herramientas (comandos) provistas por Docker realice las siguientes tareas:

a. Obtener una imagen de la última versión de Ubuntu disponible. ¿Cuál es el tamaño en disco de la imagen obtenida? ¿Ya puede ser considerada un contenedor? ¿Qué significa lo siguiente: Using default tag: latest?

```
root@so2022:~# docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
49b384cc7b4a: Pull complete
Digest: sha256:3f85b7caad41a95462cf5b787d8a04604c8262cdcdf9a472b8c52ef83375fe15
Status: Downloaded newer image for ubuntu:latest
docker.io/library/ubuntu:latest
root@so2022:~# docker images
REPOSITORY    TAG       IMAGE ID       CREATED         SIZE
ubuntu        latest    bf3dc08bfed0   4 weeks ago     76.2MB
hello-world   latest    d2c94e258dcb   13 months ago   13.3kB
```

El tamaño en disco es 76.2MB. No, no puede ser considerada un contenedor, puesto que solo descargue la imagen. Para crear un contenedor a partir de la imagen debería hacer *docker run*

Using default tag: latest significa que se está descargando la imagen sin especificar una etiqueta en particular y por defecto, Docker va a usar la etiqueta "latest", que hace referencia a la última versión disponible de la imagen en Docker Hub. Básicamente se está descargando la versión más reciente de la imagen.

b. De la imagen obtenida en el punto anterior iniciar un contenedor que simplemente ejecute el comando ls -l.

```
root@so2022:~# docker run ubuntu ls -l
total 48
lrwxrwxrwx   1 root root    7 Apr 22 13:08 bin -> usr/bin
drwxr-xr-x   2 root root 4096 Apr 22 13:08 boot
drwxr-xr-x   5 root root  340 May 27 16:44 dev
drwxr-xr-x   1 root root 4096 May 27 16:44 etc
drwxr-xr-x   3 root root 4096 Apr 29 14:05 home
lrwxrwxrwx   1 root root    7 Apr 22 13:08 lib -> usr/lib
lrwxrwxrwx   1 root root    9 Apr 22 13:08 lib64 -> usr/lib64
drwxr-xr-x   2 root root 4096 Apr 29 14:02 media
drwxr-xr-x   2 root root 4096 Apr 29 14:02 mnt
drwxr-xr-x   2 root root 4096 Apr 29 14:02 opt
dr-xr-xr-x 242 root root    0 May 27 16:44 proc
drwx------   2 root root 4096 Apr 29 14:05 root
drwxr-xr-x   4 root root 4096 Apr 29 14:05 run
lrwxrwxrwx   1 root root    8 Apr 22 13:08 sbin -> usr/sbin
drwxr-xr-x   2 root root 4096 Apr 29 14:02 srv
dr-xr-xr-x  13 root root    0 May 27 16:44 sys
drwxrwxrwt   2 root root 4096 Apr 29 14:05 tmp
drwxr-xr-x  12 root root 4096 Apr 29 14:02 usr
drwxr-xr-x  11 root root 4096 Apr 29 14:05 var
root@so2022:~#
```

c. ¿Qué sucede si ejecuta el comando docker [container] run ubuntu /bin/bash? ¿Puede utilizar la shell Bash del contenedor?

```
root@so2022:~# docker run ubuntu /bin/bash
root@so2022:~#
```

No, no puedo usar la shell Bash del contenedor

i. Modifique el comando utilizado para que el contenedor se inicie con una terminal interactiva y ejecutarlo. ¿Ahora puede utilizar la Shell Bash del contenedor? ¿Por qué?

```
root@so2022:~# docker run -it ubuntu /bin/bash
root@43ac53518bf5:/# ls
```

Si, ahora se puede. Esto se debe a que la opción -it indica a Docker que inicie el contenedor en modo interactivo con una terminal permitiendo que la entrada y salida estándar del contenedor estén conectadas a la terminal del host. Esto hace que se pueda interactuar con la Shell

Bash del contenedor como si estuviera en una sesión de terminal normal en el sistema local.

ii. ¿Cuál es el PID del proceso bash en el contenedor? ¿Y fuera de éste?

Dentro

```
root@43ac53518bf5:/# ps -ef | grep bash
root         1     0  0 16:49 pts/0    00:00:00 /bin/bash
root        11     1  0 16:53 pts/0    00:00:00 grep --color=auto bash
```

Fuera

```
root@so2022:~# docker container ls
CONTAINER ID   IMAGE     COMMAND       CREATED         STATUS         PORTS     NAMES
43ac53518bf5   ubuntu    "/bin/bash"   7 minutes ago   Up 7 minutes             romant
c_bell
root@so2022:~# docker top 43ac53518bf5
UID                 PID                 PPID                C                   STIME
            TTY                 TIME                CMD
root                10074               10052               0                   13:49
            pts/0               00:00:00            /bin/bash
```

iii. Ejecutar el comando lsns. ¿Qué puede decir de los namespace?

```
root@43ac53518bf5:/# lsns
        NS TYPE   NPROCS
4026531835 cgroup      2
4026531837 user        2
4026532230 mnt         2
4026532231 uts         2
4026532232 ipc         2
4026532233 pid         2
4026532235 net         2
so@so2022:~$ lsns
        NS TYPE   NPROCS   PID
4026531835 cgroup     74  1326
4026531836 pid        74  1326
4026531837 user       74  1326
4026531838 uts        74  1326
4026531839 ipc        74  1326
4026531840 mnt        74  1326
4026531992 net        74  1326
```

El SO host y el contenedor comparten los namespaces cgroup y user, el resto son diferentes.

iv. Dentro del contenedor cree un archivo con nombre sistemas-operativos en el directorio raíz del filesystem y luego salga del

contenedor (finalice la sesión de Bash utilizando las teclas Ctrl + D o el comando exit).

```
root@43ac53518bf5:/# touch sistemas-operativos
root@43ac53518bf5:/# ls
bin   dev  home  lib64  mnt  proc  run   sistemas-operativos  sys  usr
boot  etc  lib   media  opt  root  sbin  srv                 tmp  var
root@43ac53518bf5:/# exit
```

v. Corrobore si el archivo creado existe en el directorio raíz del sistema operativo anfitrión (host). ¿Existe? ¿Por qué?

No, no existe. Porque cuando se crea un archivo adentro del contenedor solo existe dentro del file system del mismo, no en el del SO host. Cada contenedor tiene su propio namespace. El directorio raíz del contenedor es independiente al del SO host.

d. Vuelva a iniciar el contenedor anterior utilizando el mismo comando (con una terminal interactiva). ¿Existe el archivo creado en el contenedor? ¿Por qué?

No, no existe.

```
root@so2022:~# docker container ls
CONTAINER ID   IMAGE     COMMAND       CREATED              STATUS              PORTS
   NAMES
6090f7341b1e   ubuntu    "/bin/bash"   About a minute ago   Up About a minute
   vibrant_keldysh
```

Esto sucede porque se crea una nueva instancia. Se puede ver que se tiene otro container ID.

e. Obtenga el identificador del contenedor (container_id) donde se creó el archivo y utilícelo para iniciar con el comando docker start -ia container_id el contenedor en el cual se creó el archivo.

i. ¿Cómo obtuvo el container_id para para este comando?

```
root@so2022:~# docker ps --all
CONTAINER ID   IMAGE         COMMAND       CREATED          STATUS
 PORTS     NAMES
6090f7341b1e   ubuntu        "/bin/bash"   3 minutes ago    Up 3 minutes
          vibrant_keldysh
43ac53518bf5   ubuntu        "/bin/bash"   25 minutes ago   Exited (0) 10 minutes ago
          romantic_bell
ef363cfd3d1b   ubuntu        "/bin/bash"   27 minutes ago   Exited (0) 27 minutes ago
          relaxed_gagarin
722a0b3e7bfa   ubuntu        "ls -l"       29 minutes ago   Exited (0) 29 minutes ago
          zen_bartik
c82a5d59cffc   hello-world   "/hello"      40 minutes ago   Exited (0) 40 minutes ago
          dreamy_pare
```

ii. Chequee nuevamente si el archivo creado anteriormente existe. ¿Cuál es el resultado en este caso? ¿Puede encontrar el archivo creado?

```
root@so2022:/# docker start -ia 43ac53518bf5
root@43ac53518bf5:/# ls
bin   dev  home  lib64  mnt  proc  run   sistemas-operativos  sys  usr
boot  etc  lib   media  opt  root  sbin  srv                  tmp  var
```

f. ¿Cuántos contenedores están actualmente en ejecución? ¿En qué estado se encuentra cada uno de los que se han ejecutado hasta el momento?

```
root@so2022:~# docker ps
CONTAINER ID   IMAGE     COMMAND       CREATED          STATUS          PORTS     NAMES
43ac53518bf5   ubuntu    "/bin/bash"   27 minutes ago   Up 48 seconds             roman
tic_bell
```

Solo 1 y está en estado Up.

```
CONTAINER ID   IMAGE         COMMAND       CREATED          STATUS
 PORTS     NAMES
6090f7341b1e   ubuntu        "/bin/bash"   7 minutes ago    Exited (0) 3 minutes ago
          vibrant_keldysh
43ac53518bf5   ubuntu        "/bin/bash"   29 minutes ago   Up 2 minutes
          romantic_bell
ef363cfd3d1b   ubuntu        "/bin/bash"   31 minutes ago   Exited (0) 31 minutes ago
          relaxed_gagarin
722a0b3e7bfa   ubuntu        "ls -l"       33 minutes ago   Exited (0) 33 minutes ago
          zen_bartik
c82a5d59cffc   hello-world   "/hello"      44 minutes ago   Exited (0) 44 minutes ago
          dreamy_pare
```

El resto se encuentra en estado Exited con código de salida 0 (finalizo sin errores)

g. Elimine todos los contenedores creados hasta el momento. Indique el o los comandos utilizados.

```
root@so2022:~# docker stop $(docker ps -aq)
6090f7341b1e
43ac53518bf5
ef363cfd3d1b
722a0b3e7bfa
c82a5d59cffc
root@so2022:~# docker rm $(docker ps -aq)
6090f7341b1e
43ac53518bf5
ef363cfd3d1b
722a0b3e7bfa
c82a5d59cffc
```

3. Creación de una imagen a partir de un contenedor. Siguiendo los pasos indicados a continuación genere una imagen de Docker a partir de un contenedor:

   a. Inicie un contenedor a partir de la imagen de Ubuntu descargada anteriormente ejecutando una consola interactiva de Bash.

   b. Instale el servidor web Nginx, https://nginx.org/en/, en el contenedor utilizando los siguientes comandos:
   export DEBIAN_FRONTEND=noninteractive
   export TZ=America/Buenos_Aires
   apt update -qq
   apt install -y --no-install-recommends nginx

   c. Salga del contenedor y genere una imagen Docker a partir de éste. ¿Con qué nombre se genera si no se especifica uno?

```
root@so2022:~# docker ps -a
CONTAINER ID   IMAGE     COMMAND       CREATED              STATUS
       PORTS     NAMES
12f27fea9cee   ubuntu    "/bin/bash"   About a minute ago   Exited (0) 7 seconds
 ago              focused_feynman
root@so2022:~# docker commit 12f27fea9cee
sha256:29af2f353a38555cf20a9070b8cb9ffe8e4c5479e9cb54dc6135275f99a2a5e8
root@so2022:~# docker images
REPOSITORY    TAG       IMAGE ID       CREATED         SIZE
<none>        <none>    29af2f353a38   6 seconds ago   122MB
ubuntu        latest    bf3dc08bfed0   4 weeks ago     76.2MB
hello-world   latest    d2c94e258dcb   13 months ago   13.3kB
```

   Se genera sin nombre, con <none>

d. Cambie el nombre de la imagen creada de manera que en la columna Repository aparezca nginx-so y en la columna Tag aparezca v1.

```
root@so2022:~# docker tag 29af2f353a38 nginx-so:v1
root@so2022:~# docker images
REPOSITORY    TAG       IMAGE ID       CREATED              SIZE
nginx-so      v1        29af2f353a38   About a minute ago   122MB
ubuntu        latest    bf3dc08bfed0   4 weeks ago          76.2MB
hello-world   latest    d2c94e258dcb   13 months ago        13.3kB
```

e. Ejecute un contenedor a partir de la imagen nginx-so:v1 que corra el servidor web nginx atendiendo conexiones en el puerto 8080 del host, y si página web para corroborar su correcto funcionamiento. Para esto:

   i. En el Sistema Operativo anfitrión (host) sobre el cual se ejecuta Docker crear un directorio que se utilizará para este taller. Éste puede ser el directorio nginx-so dentro de su directorio personal o cualquier otro directorio - para los fines de este enunciado haremos referencia a éste como /home/so/nginx-so, por lo que en los lugares donde se mencione esta ruta usted deberá reemplazarla por la ruta absoluta al directorio que haya decidido crear en este paso.

   ii. Dentro de ese directorio, cree un archivo llamado index.html que contenga el código HTML de este gist de GitHub: https://gist.github.com/ncuesta/5b959fce1c7d2ed4e5a06e84e5a7efc8.

   iii. Cree un contenedor a partir de la imagen nginx-so:v1 montando el directorio del host (/home/so/nginx-so) sobre el directorio /var/www/html del contenedor, mapeando el puerto 80 del contenedor al puerto 8080 del host, y ejecutando el servidor nginx en primer plano. Indique el comando utilizado

```
root@so2022:~# docker run -d -p 8080:80 -v /home/so/nginx-so:/var/www/html --name mi_contenedor nginx-so:v1 ngin
x -g 'daemon off;'
4cc30b418eb46d57c71e4188371c02f0c40afc0d75e20e71fadc4bcf98470eb3
```

-d: indica que el contenedor se ejecute en segundo plano (en modo "detached").

-p 8080:80: mapeo el puerto 80 del contenedor al puerto 8080 del host.

-v /home/so/nginx-so:/var/www/html: se monta el directorio /home/so/nginx-so del host en el directorio /var/www/html del contenedor.

--name mi_contenedor: asigna el nombre mi_contenedor al contenedor.

nginx-so:v1: nombre y etiqueta de la imagen desde la cual se crea el contenedor.

nginx -g 'daemon off;': ejecuta el servidor nginx en primer plano

f. Verifique que el contenedor esté ejecutándose correctamente abriendo un navegador web y visitando la URL http://localhost:8080.

```
root@so2022:~# curl http://localhost:8080
<html lang="es">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>nginx en un container!</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0/dist/css/bootstrap.min.css" rel="stylesheet" integr
ity="sha384-wEmeIV1mKuiNpC+IOBjI7aAzPcEZeedi5yW5f2yOq55WWLwNGmvvx4Um1vskeMj0" crossorigin="anonymous">
  </head>
  <body>
    <div class="container">
      <div class="row justify-content-center">
        <div class="col-6">
          <h1>Sistemas Operativos <span id="y"></span></h1>
          <h2>Trabajo Práctico de Docker</h2>

          <p class="lead">¡Felicitaciones! El servidor web con nginx está funcionando correctamente.</p>
        </div>
      </div>
    </div>

    <script>document.getElementById('y').innerHTML = new Date().getFullYear()</script>
  </body>
</html>
```

g. Modifique el archivo index.html agregándole un párrafo con su nombre y número de alumno. ¿Es necesario reiniciar el contenedor para ver los cambios?

No, no es necesario ya que se usa un bind mount.

h. Analice: ¿por qué es necesario que el proceso nginx se ejecute en primer plano? ¿Qué ocurre si lo ejecuta sin -g 'daemon off;'?

```
root@so2022:~# docker run -d -p 8080:80 -v /home/so/nginx-so:/var/www/html --name mi_contenedor nginx-so:v1 ngin
x
83f829942adc7e10bfd9ccb34f04878eb283daa98315c821915794d7b908c670
root@so2022:~# docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
```

El proceso se ejecutaría en segundo plano como un demonio, lo que haría que el contenedor se detenga inmediatamente después de que el proceso nginx se inicie ya que Docker espera que el proceso principal del contenedor esté en ejecución para mantenerlo activo.

4. Creación de una imagen Docker a partir de un archivo Dockerfile. Siguiendo los pasos indicados a continuación, genere una nueva imagen a partir de los pasos descritos en un Dockerfile.

   a. En el directorio del host creado en el punto anterior (/home/so/nginx-so), cree un archivo Dockerfile que realice los siguientes pasos:
      i. Comenzar en base a la imagen oficial de Ubuntu.

      ii. Exponer el puerto 80 del contenedor.

      iii. Instalar el servidor web nginx.

      iv. Copiar el archivo index.html del mismo directorio del host al directorio /var/www/html de la imagen.

      v. Indicar el comando que se utilizará cuando se inicie un contenedor a partir de esta imagen para ejecutar el servidor nginx en primer plano: nginx -g 'daemon off;'. Use la forma exec 4 para definir el comando, de manera que todas las señales que reciba el contenedor sean enviadas directamente al proceso de nginx.
      Ayuda: las instrucciones necesarias para definir los pasos en el Dockerfile son FROM, EXPOSE, RUN, COPY y CMD.

```
 GNU nano 3.2                          Dockerfile

FROM ubuntu

EXPOSE 80

RUN apt-get update && apt install -y --no-install-recommends nginx

COPY index.html /var/www/html/

CMD ["nginx", "-g", "daemon off;"]
```

b. Utilizando el Dockerfile que generó en el punto anterior construya una nueva imagen Docker guardándola localmente con el nombre nginx-so:v2.

```
root@so2022:~# docker build -t nginx-so:v2 /home/so/nginx-so/
[+] Building 102.5s (8/8) FINISHED                              docker:default
 => [internal] load build definition from Dockerfile                      0.1s
 => => transferring dockerfile: 198B                                      0.0s
 => [internal] load metadata for docker.io/library/ubuntu:latest          0.0s
 => [internal] load .dockerignore                                         0.0s
 => => transferring context: 2B                                           0.0s
 => CACHED [1/3] FROM docker.io/library/ubuntu:latest                     0.0s
 => [internal] load build context                                         0.0s
 => => transferring context: 32B                                          0.0s
 => [2/3] RUN apt-get update && apt install -y --no-install-recommends  101.9s
 => [3/3] COPY index.html /var/www/html/                                  0.1s
 => exporting to image                                                    0.3s
 => => exporting layers                                                   0.2s
 => => writing image sha256:00f41e79376126c5fd2ee19b8d18de6dbab399378f873  0.0s
 => => naming to docker.io/library/nginx-so:v2                            0.0s
root@so2022:~# docker images
REPOSITORY    TAG       IMAGE ID       CREATED          SIZE
nginx-so      v2        00f41e793761   5 seconds ago    122MB
nginx-so      v1        29af2f353a38   46 minutes ago   122MB
ubuntu        latest    bf3dc08bfed0   4 weeks ago      76.2MB
hello-world   latest    d2c94e258dcb   13 months ago    13.3kB
```

c. Ejecute un contenedor a partir de la nueva imagen creada con las opciones adecuadas para que pueda acceder desde su navegador web a la página a través del puerto 8090 del host. Verifique que puede visualizar correctamente la página accediendo a http://localhost:8090.

```
root@so2022:~# docker run -d -p 8090:80 --name mi_contenedor_2 nginx-so:v2
75cf7ef3f18034ff979eb5b0861d704917806a6ade5e814912175c71a29f3277
root@so2022:~# docker ps
CONTAINER ID   IMAGE         COMMAND                  CREATED         STATUS         PORTS
           NAMES
75cf7ef3f180   nginx-so:v2   "nginx -g 'daemon of…"   5 seconds ago   Up 3 seconds   0.0.0.0:8090->80/tcp, :::80
90->80/tcp   mi_contenedor_2
```

```
root@so2022:~# curl ttp://localhost:8090
curl: (1) Protocol "ttp" not supported or disabled in libcurl
root@so2022:~# curl http://localhost:8090
<html lang="es">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>nginx en un container!</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0/dist/css/bootstrap.min.css" rel="stylesheet" integr
ity="sha384-wEmeIV1mKuiNpC+IOBjI7aAzPcEZeedi5yW5f2yOq55WWLwNGmvvx4Um1vskeMj0" crossorigin="anonymous">
  </head>
  <body>
    <div class="container">
      <div class="row justify-content-center">
        <div class="col-6">
          <h1>Sistemas Operativos <span id="y"></span></h1>
          <h2>Trabajo Práctico de Docker</h2>
          <h3> Agustina Rojas <h3>

          <p class="lead">¡Felicitaciones! El servidor web con nginx está funcionando correctamente.</p>
        </div>
      </div>
    </div>

    <script>document.getElementById('y').innerHTML = new Date().getFullYear()</script>
  </body>
</html>
```

d. Modifique el archivo index.html del host agregando un párrafo con la fecha actual y recargue la página en su navegador web. ¿Se ven reflejados los cambios que hizo en el archivo? ¿Por qué?

   No, no se ven reflejados porque el contenedor está usando la versión del index.html que esta en la imagen creada.

e. Termine el contenedor iniciado antes y cree uno nuevo utilizando el mismo comando. Recargue la página en su navegador web. ¿Se ven ahora reflejados los cambios realizados en el archivo HTML? ¿Por qué?

   No, no se ven reflejados los cambios por lo mismo que explique arriba.

f. Vuelva a construir una imagen Docker a partir del Dockerfile creado anteriormente, pero esta vez dándole el nombre nginx-so:v3. Cree un contenedor a partir de ésta y acceda a la página en su navegador web. ¿Se ven reflejados los cambios realizados en el archivo HTML? ¿Por qué?

```
root@so2022:~# docker build -t nginx-so:v3 /home/so/nginx-so/
[+] Building 0.3s (8/8) FINISHED                                  docker:default
 => [internal] load build definition from Dockerfile                      0.0s
 => => transferring dockerfile: 198B                                      0.0s
 => [internal] load metadata for docker.io/library/ubuntu:latest          0.0s
 => [internal] load .dockerignore                                         0.0s
 => => transferring context: 2B                                           0.0s
 => [1/3] FROM docker.io/library/ubuntu:latest                            0.0s
 => [internal] load build context                                         0.0s
 => => transferring context: 957B                                         0.0s
 => CACHED [2/3] RUN apt-get update && apt install -y --no-install-recomm  0.0s
 => [3/3] COPY index.html /var/www/html/                                  0.1s
 => exporting to image                                                    0.0s
 => => exporting layers                                                   0.0s
 => => writing image sha256:8843389f42a552b03be8ef26dfd7fce02ea05f5f3d64d  0.0s
 => => naming to docker.io/library/nginx-so:v3                            0.0s
```

```
root@so2022:~# docker run -d -p 8090:80 --name mi_contenedor_2 nginx-so:v3
859f1e288b59aaf4865d9f50f9d6fa54e9d7970a4737f7cad8aa90668425d2d4
```

```
root@so2022:~# curl http://localhost:8090
<html lang="es">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>nginx en un container!</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0/dist/css/bootstrap.min.css" rel="s
tylesheet" integrity="sha384-wEmeIV1mKuiNpC+IOBjI7aAzPcEZeedi5yW5f2yOq55WWLwNGmvvx4Um1vskeMj0"
crossorigin="anonymous">
  </head>
  <body>
    <div class="container">
      <div class="row justify-content-center">
        <div class="col-6">
          <h1>Sistemas Operativos <span id="y"></span></h1>
          <h2>Trabajo Práctico de Docker</h2>
          <h3> Agustina Rojas <h3>
          <p> 27/05/2024 <p>
          <p class="lead">¡Felicitaciones! El servidor web con nginx está funcionando correctam
ente.</p>
        </div>
      </div>
    </div>
```

Ahora si se ven reflejados los cambios porque la imagen creada tiene la versión del index.html actualizado con la última modificación.

Fuentes

https://www.netapp.com/es/devops-solutions/what-are-containers/

https://aws.amazon.com/es/compare/the-difference-between-docker-images-and-containers/

https://www.freecodecamp.org/espanol/news/como-obtener-la-direccion-ip-de-un-contenedor-docker-explicado-con-ejemplos/

https://docs.docker.com/network/#:~:text=By%20default%2C%20the%20container%20gets,default%20subnet%20mask%20and%20gateway.

https://www.geeksforgeeks.org/docker-volume-vs-bind-mount/