Práctica 4B - Docker y Docker Compose

Docker

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

Docker es una plataforma que permite empaquetar aplicaciones con todas sus dependencias en contenedores. La idea es que sin importar el SO o las configuraciones locales una aplicaciones pueda andar en cualquier dispositivo.
Da portabilidad, porque podes ejecutar el contenedor en cualquier máquina
Da aislamiento

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

Una imagen es un molde de solo lectura con las instrucciones para construir un contenedor
Un contenedor es la instancia de una imagen en ejecución

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

Es el filesystem de los containers. Es un mecanismo de montajes, no un nuevo filesystem. Permite que varios directorios se monten en el mismo punto de montaje, apareciendo como un solo file system. La capa superior es de escritura, el resto solo de lectura.
Cuando se ejecuta el contenedor desde una imagen, se genera un union filesystem donde las capas se apilan una sobre otra y usando chroot se establece el filesystem como directorio raiz del contenedor.

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

172.17.0.0/16 → Es el rango por defecto en Docker para la red bridge.
También pueden usarse otros rangos privados como:
192.168.0.0/16
10.0.0.0/8
Las IPs son asignadas automáticamente por un componente de Docker llamado Docker Daemon, que actúa como un servidor DHCP interno para los contenedores conectados a la red por defecto (o redes personalizadas).

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

1. Volúmenes de Docker

Gestionados por Docker.

Se almacenan en /var/lib/docker/volumes/ (en Linux).

2. Bind mounts

Montan una carpeta del host (tu sistema operativo) en el contenedor.

Taller:
El siguiente taller le guiará paso a paso para la construcción de una imagen Docker utilizando dos mecanismos distintos para los cuales deberá investigar y documentar qué comandos y argumentos utiliza para cada caso.
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.

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@so:/home/so# docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
0622fac788ed: Pull complete
Digest: sha256:6015f66923d7afbc53558d7ccffd325d43b4e249f41a6e93eef074c9505d2233
Status: Downloaded newer image for ubuntu:latest
docker.io/library/ubuntu:latest
root@so:/home/so#
```

```
root@so:/home/so# docker images
REPOSITORY   TAG       IMAGE ID       CREATED       SIZE
ubuntu       latest    a0e45e2ce6e6   3 weeks ago   78.1MB
```

Nop, no es un contenedor. una imagen es como la plantilla
cuando no le aclaras usa latest (la última)
b. De la imagen obtenida en el punto anterior iniciar un contenedor que simplemente ejecute el comando ls -l.

```
root@so:/home/so# docker run ubuntu ls -l
total 48
lrwxrwxrwx   1 root root    7 Apr 22  2024 bin -> usr/bin
drwxr-xr-x   2 root root 4096 Apr 22  2024 boot
drwxr-xr-x   5 root root  340 May 25 23:16 dev
drwxr-xr-x   1 root root 4096 May 25 23:16 etc
drwxr-xr-x   3 root root 4096 Apr 15 14:11 home
lrwxrwxrwx   1 root root    7 Apr 22  2024 lib -> usr/lib
lrwxrwxrwx   1 root root    9 Apr 22  2024 lib64 -> usr/lib64
drwxr-xr-x   2 root root 4096 Apr 15 14:04 media
drwxr-xr-x   2 root root 4096 Apr 15 14:04 mnt
drwxr-xr-x   2 root root 4096 Apr 15 14:04 opt
dr-xr-xr-x 219 root root    0 May 25 23:16 proc
drwx------   2 root root 4096 Apr 15 14:11 root
drwxr-xr-x   4 root root 4096 Apr 15 14:11 run
lrwxrwxrwx   1 root root    8 Apr 22  2024 sbin -> usr/sbin
drwxr-xr-x   2 root root 4096 Apr 15 14:04 srv
dr-xr-xr-x  13 root root    0 May 25 23:16 sys
drwxrwxrwt   2 root root 4096 Apr 15 14:11 tmp
drwxr-xr-x  12 root root 4096 Apr 15 14:04 usr
drwxr-xr-x  11 root root 4096 Apr 15 14:11 var
```

crea un contenedor temporal y se destruye automáticamente

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

Crea un nuevo contenedor basado en la imagen ubuntu.
Ejecuta el programa /bin/bash dentro del contenedor.
Luego finaliza inmediatamente, porque no está en modo interactivo ni tiene una terminal asignada. No se puede usar la shell Bash interactivamente con este comando.
El contenedor se ejecuta y se detiene inmediatamente, sin darte acceso a la línea de comandos.

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@so:/home/so# docker run -it ubuntu /bin/bash
```

ahora chi pq -it Asocia la terminal del usuario (host) con la del contenedor.
Permite enviar comandos y ver sus resultados en tiempo real.
Le da a Bash el entorno que necesita para mantenerse ejecutándose.

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

arafue

```
root@so:/home/so# docker inspect --format '{{.State.Pid}}' 7d32cf07949d
2796
root@so:/home/so#
```

atroden

```
root@7d32cf07949d:/# ps aux
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  0.0  0.1   4588  3968 pts/0    Ss   23:20   0:00 /bin/bash
root           9  0.0  0.1   7888  3756 pts/0    R+   23:24   0:00 ps aux
root@7d32cf07949d:/#
```

Docker usa namespaces de PID, por lo que dentro del contenedor el proceso principal es siempre 1.

En el host, ese mismo proceso es solo uno más en la tabla global de procesos del sistema operativo.

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

```
root@7d32cf07949d:/# lsns
        NS TYPE   NPROCS PID USER COMMAND
4026531834 time        2   1 root /bin/bash
4026531837 user        2   1 root /bin/bash
4026532674 mnt         2   1 root /bin/bash
4026532675 uts         2   1 root /bin/bash
4026532795 ipc         2   1 root /bin/bash
4026532796 pid         2   1 root /bin/bash
4026532797 cgroup      2   1 root /bin/bash
4026532798 net         2   1 root /bin/bash
```

El comando lsns (list namespaces) muestra los espacios de nombres (namespaces) activos del sistema.
Permite ver cómo los procesos están aislados unos de otros en diferentes recursos del sistema, gracias a los namespaces.

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).

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

```
root@7d32cf07949d:/# touch /sistemas-operativos
root@7d32cf07949d:/# ls
bin   dev  home  lib64  mnt  proc  run   sistemas-operativos  sys  usr
boot  etc  lib   media  opt  root  sbin  srv                 tmp  var
```

```
root@so:/# ls
bin   etc         initrd.img.old  lost+found  opt   run   sys  var
boot  home        lib             media       proc  sbin  tmp  vmlinuz
dev   initrd.img  lib64           mnt         root  srv   usr  vmlinuz.old
```

no ta, porque se creó aislado en el file system del contenedor

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

si, el archivo está y va a estar hasta que se elimine el contenedor

si hubieramos hecho docker run si que no estaría porque haríamos un nuevo contenedor

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.

ah esto hice en el anterior xd

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

docker ps -a

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

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

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

docker container prune

o sino para borrar también los que se están ejecutando

docker rm -f $(docker ps -aq)

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.

```
root@936cca29d88e:/# docker run -it ubuntu:latest /bin/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

```
root@936cca29d88e:/# 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@so:/home/so# docker ps -a
CONTAINER ID   IMAGE          COMMAND       CREATED             STATUS                     PORTS     NAMES
7e007b6405ff   ubuntu:latest   "/bin/bash"   About a minute ago   Exited (0) 15 seconds ago             agitated_visvesvaraya
root@so:/home/so#
```

```
root@936cca29d88e:/# exit
exit
root@so:/home/so# docker ps -a
CONTAINER ID   IMAGE          COMMAND       CREATED          STATUS                       PORTS     NAMES
936cca29d88e   ubuntu:latest   "/bin/bash"   11 minutes ago   Exited (127) 35 seconds ago            intelligent_franklin
```

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@so:/home/so# docker tag f3ab79431bee nginx-so:v1
root@so:/home/so# docker images
REPOSITORY      TAG         IMAGE ID       CREATED          SIZE
nombre_imagen   etiqueta    75eebad54206   12 minutes ago   135MB
nginx-so        v1          f3ab79431bee   17 minutes ago   135MB
ubuntu          latest      a0e45e2ce6e6   3 weeks ago      78.1MB
hello-world     latest      74cc54e27dc4   4 months ago     10.1kB
```

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 sirviendo una página web para corroborar su correcto funcionamiento. Para esto:

```
root@so:/home/so# docker run -d -p 8080:80 --name nginx-so-server nginx-so:v1 /usr/sbin/nginx -g "daemon off;"
55cdd6ec0651dd89b1934b17af6485a55594e093edda44f80e3941cddda54089
root@so:/home/so# curl http://localhost:8080
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
```

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.

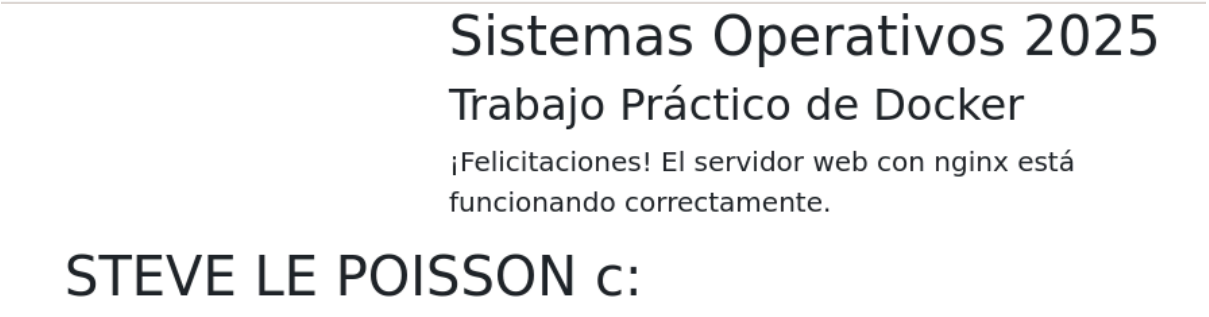
III. Cree un contenedor a partir de la imagen nginx-so:v1 montando el directorio del host (/home/so/nginx-so) sobre el directorio /var/www/html del contenedor, mapeando el puerto 80 del contenedor al puerto 8080 del host, y ejecutando el servidor nginx en primer plano. Indique el comando utilizado.

root@so:/home/so/nginx-so# docker run -d -p 8080:80 --name nginx-so-server -v /home/so/nginx-so:/var/www/html nginx-so:v1 /usr/sbin/nginx -g "daemon off;"

- `-d` → corre el contenedor en modo detached (en background).
- `-p 8080:80` → expone el puerto 80 del contenedor al puerto 8080 de tu host.
- `--name nginx-so-server` → le asigna ese nombre al contenedor.
- `nginx-so:v1` → imagen a usar.
- `/usr/sbin/nginx -g "daemon off;"` → inicia Nginx en primer plano (para que el contenedor no se detenga).

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

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?



nop, no es necesario reiniciar el container

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

Si nginx se ejecuta en primer plano, Docker detecta que el proceso sigue vivo y el contenedor permanece activo.

Si nginx se ejecuta como daemon (proceso en segundo plano, background), el proceso padre termina rápidamente y Docker piensa que el contenedor ya terminó, por lo que lo detiene.

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 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.

```
FROM ubuntu:latest
EXPOSE 80
RUN apt-get update && apt-get install -y nginx && apt-get clean
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@so:/home/so/nginx-so# docker build -t nginx-so:v2 /home/so/nginx-so
+] Building 10.7s (5/7)                                          docker:default
```

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

```
root@so:/home/so/nginx-so# docker run -d -p 8090:80 --name nginx-so-v2-container
 nginx-so:v2
fe8f267e027ac37d5276604aefee792123e52d3fca411169133fc56e3dd5b8de
```

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, los cambios no se reflejan en el navegador.
Porque en esta imagen (nginx-so:v2) el archivo index.html fue copiado al construir la imagen, usando COPY en el Dockerfile. Esto significa que el archivo se encuentra dentro del sistema de archivos del contenedor, y no está vinculado al archivo del host.

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 se ven pq el archivo index se copió al crear la imagen y al volver a crear el contenedor usando la misma imagen no actualizamos el index

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?
ahora chi, porque volvimos a copiarlo uwu

Docker Compose

1. Utilizando sus palabras describa, ¿qué es docker compose?
Es una herramienta para correr aplicaciones que necesitan muchos contenedores
En lugar de ejecutar muchos comandos docker run uno por uno, podés definir todos los servicios, redes y volúmenes en un único archivo y levantarlos con un solo comando (docker-compose up).
se definen: Qué imágenes o Dockerfiles usar, qué puertos mapear, qué volúmenes montar, cómo se conectan entre sí los servicios y otros parámetros como variables de entorno o dependencias.

2. ¿Qué es el archivo compose y cual es su función? ¿Cuál es el “lenguaje” del archivo?
compose.yaml facilita la creación de servicios, almacenamiento y red
El archivo compose.yaml (o docker-compose.yml) es un archivo de configuración en formato YAML que se utiliza para definir y ejecutar aplicaciones Docker multi-contenedor.En él se pueden definir:
-Servicios (por ejemplo: un backend, una base de datos, un servidor web, etc.)
-Volúmenes (almacenamiento persistente)
-Redes personalizadas
-Dependencias entre servicios
-Variables de entorno
-Montaje de archivos o directorios del host
El archivo está escrito en YAML (YAML Ain't Markup Language), un lenguaje de serialización de datos legible por humanos, usado comúnmente para configuración

3. ¿Cuáles son las versiones existentes del archivo docker-compose.yaml existentes y qué características aporta cada una? ¿Son compatibles entre sí?¿Por qué?
Versión 1: Muy básica, sin version: explícito. Obsoleta.
Versión 2.x: Agrega soporte para redes, volúmenes, dependencias entre servicios. Más flexible.

Versión 3.x: Pensada para Docker Swarm, incluye opciones de despliegue (deploy, replicas).
Versión Compose-spec (sin número): Es la versión actual recomendada. Usa un estándar unificado, sin necesidad de especificar version:.

4. Investigue y describa la estructura de un archivo compose.
Desarrolle al menos sobre los siguientes bloques indicando para qué se usan:
services: Define los contenedores que correrán en la app.
build: Construye una imagen Docker desde un Dockerfile local.
image: Usa una imagen Docker ya creada (local o remota).
volumes: Monta carpetas o volúmenes entre host y contenedor para persistencia o compartir archivos.
restart: Política para reiniciar el contenedor si se detiene o falla.
depends_on: Establece el orden de arranque entre servicios.
environment: Define variables de entorno dentro del contenedor.
ports: Mapea puertos del contenedor a puertos del host para acceso externo.
expose: Expone puertos solo para comunicación interna entre contenedores, no al host.
networks: Configura redes personalizadas para conectar servicios entre sí.
healthcheck: se utiliza dentro de la definición de un servicio en un archivo docker-compose.yml para verificar el estado de salud de un contenedor. Docker ejecuta pruebas periódicas para determinar si el servicio está funcionando correctamente

5. Conceptualmente: ¿Cómo se podrían usar los bloques “healthcheck” y “depends_on” para ejecutar una aplicación Web dónde el backend debería ejecutarse si y sólo si la base de datos ya está ejecutándose y lista?
`healthcheck`: Sirve para que Docker verifique si un contenedor está *realmente listo* y funcionando correctamente, no sólo iniciado. Por ejemplo, para la base de datos, se puede definir un comando que pruebe si responde a consultas.

`depends_on`: Indica que un servicio (ej: backend) debe esperar a que otro (ej: base de datos) se *inicie* primero. Pero por defecto sólo verifica que el contenedor esté "up", no que esté listo.

6. Indique qué hacen y cuáles son las diferencias entre los siguientes comandos:
a. docker compose create y docker compose up
create: Crea los contenedores pero no los inicia.
up: Crea y además inicia los contenedores (y crea redes y volúmenes si no existen).
b. docker compose stop y docker compose down
stop: Detiene los contenedores pero los deja existentes (pueden reiniciarse luego).
down: Detiene y elimina los contenedores, redes y por defecto los volúmenes no anclados.
c. docker compose run y docker compose exec
run: Ejecuta un comando en un nuevo contenedor temporal basado en el servicio (no usa contenedor ya corriendo).
exec: Ejecuta un comando dentro de un contenedor ya en ejecución del servicio
d. docker compose ps
Lista los contenedores gestionados por Compose con su estado actual.
e. docker compose logs

Muestra los logs (salida estándar) de los contenedores gestionados por Compose.
7. ¿Qué tipo de volúmenes puede utilizar con docker compose? ¿Cómo se declara cada tipo en el archivo compose?
Named Volumes (Persistentes, administrados por Docker)
Uso: Volúmenes administrados por Docker, almacenados en /var/lib/docker/volumes/.Ideales para persistencia de datos (ej: bases de datos).

Bind Mounts (Vinculan directorios del host al contenedor):
Uso:Vinculan directorios o archivos específicos del host al contenedor.Útiles para desarrollo (cambios en el host se reflejan al instante).
Volúmenes Temporales (tmpfs Mounts)
Uso:Almacenan datos en memoria RAM (no persisten después de reiniciar el contenedor).Ideales para datos temporales sensibles (ej: archivos de sesión).
8. ¿Qué sucede si en lugar de usar el comando “docker compose down” utilizo “docker compose down -v/--volumes”?
✅ docker compose down → Elimina contenedores y redes, pero NO borra volúmenes (los datos persisten).
🔴 docker compose down -v → Elimina todo (contenedores, redes y volúmenes). ¡Los datos se pierden!

Instalación de docker compose
En la práctica anterior se instaló el entorno de ejecución Docker-CE. Es requisito para esta práctica tener dicho entorno instalado y funcionando en el dispositivo donde se pretenda realizar la misma.
En el sitio https://docs.docker.com/compose/install/ se puede encontrar la guía para instalar docker-compose en distintos SO.
Docker-compose es simplemente un binario, por lo que lo único que se necesita es descargar el binario, ubicarlo en algún lugar que el PATH de nuestro dispositivo pueda encontrarlo y que tenga los permisos necesarios para ser ejecutado.
En la actualidad existen 2 versiones del binario docker-compose. Vamos a utilizar la versión 2. Para instalar la versión 2.18.1, vamos a descargarla y ubicarla en el directorio /usr/local/bin/docker-compose, para que de esta manera quede accesible mediante el PATH de nuestra CLI:

```
~$ sudo curl -SL
https://github.com/docker/compose/releases/download/v2.18.1/docker-compo
se-linux-x86_64 -o /usr/local/bin/docker-compose
```

Una vez descargado, le damos permiso de ejecución:

```
~$ sudo chmod +x /usr/local/bin/docker-compose
```

De esta manera ya tendremos docker-compose disponible. Para asegurarnos que esté instalado correctamente, verificamos la versión instalada corriendo desde la consola:

```
~$ docker compose --version
Docker Compose version v2.18.1
```

Ejercicio guiado - Instanciando un Wordpress y una Base de Datos.
Dado el siguiente código de archivo compose:

```
version: "3.9"
services:
db:
image: mysql:5.7
networks:
- wordpress
volumes:
- db_data:/var/lib/mysql
restart: always
environment:
MYSQL_ROOT_PASSWORD: somewordpress
MYSQL_DATABASE: wordpress
MYSQL_USER: wordpress
MYSQL_PASSWORD: wordpress
wordpress:
depends_on:
- db
image: wordpress:latest
networks:
- wordpress
volumes:
- ${PWD}:/data
- wordpress_data:/var/www/html
ports:
- "127.0.0.1:8000:80"
restart: always
environment:
WORDPRESS_DB_HOST: db
WORDPRESS_DB_USER: wordpress
WORDPRESS_DB_PASSWORD: wordpress
WORDPRESS_DB_NAME: wordpress
volumes:
db_data: {}
wordpress_data: {}
networks:
wordpress:
```

`Preguntas`
Intente analizar el código ANTES de correrlo y responda:
● ¿Cuántos contenedores se instancian?
2: wordpress y db
● ¿Por qué no se necesitan Dockerfiles?
porque usa imagenes ya creadas
wordpress:latest

image: mysql:5.7

● ¿Por qué el servicio identificado como “wordpress” tiene la siguiente línea?
depends_on:
- db
El servicio posee la linea “-db” para indicar que requiere levantar primero el container con la base de datos antes de ejecutar su imagen
● ¿Qué volúmenes y de qué tipo tendrá asociado cada contenedor?
- db_data:/var/lib/mysql: nombrado
- ${PWD}:/data: blind mount
- wordpress_data:/var/www/html: nombrado

● ¿Por que uso el volumen nombrado
volumes:
- db_data:/var/lib/mysql
para el servicio db en lugar de dejar que se instancie un volumen anónimo con el contenedor?
Para tener persistencia de la db. SI ni lo tuvieramos, se eliminan los datos al borrar el contenedor

● ¿Qué genera la línea
volumes:
- ${PWD}:/data
en la definición de wordpress?
Vincula el directorio actual de tu computadora (donde ejecutás docker-compose) con la carpeta /data del contenedor WordPress.
Son variables de entorno que se pasan al contenedor al inicializarlo.
● ¿Qué representa la información que estoy definiendo en el bloque environment de cada servicio?¿Cómo se “mapean” al instanciar los contenedores?

Cuando se ejecuta el contenedor, Docker traduce cada línea del bloque environment a una variable de entorno del sistema dentro del contenedor.

● ¿Qué sucede si cambio los valores de alguna de las variables definidas en bloque “environment” en solo uno de los contenedores y hago que sean diferentes? (Por ej: cambio SOLO en la definición de wordpress la variable WORDPRESS_DB_NAME)
🔧 Contexto básico
Ambos contenedores (db y wordpress) deben estar de acuerdo sobre:

el nombre de la base de datos (MYSQL_DATABASE vs WORDPRESS_DB_NAME)

el usuario (MYSQL_USER vs WORDPRESS_DB_USER)

la contraseña (MYSQL_PASSWORD vs WORDPRESS_DB_PASSWORD)

y la dirección (WORDPRESS_DB_HOST)

WordPress necesita conectarse a la base de datos con ciertos datos que le pasás como variables de entorno, y esos datos deben coincidir con los que se usaron para configurar MySQL.

● ¿Cómo sabe comunicarse el contenedor “wordpress” con el contenedor “db” si nunca doy información de direccionamiento?

Docker Compose crea automáticamente una red interna (wordpress) donde los contenedores pueden comunicarse por nombre de servicio.
Entonces wordpress puede usar db como hostname gracias a esta red interna y Docker se encarga del "resolving" (como si fuera DNS).

● ¿Qué puertos expone cada contenedor según su Dockerfile? (pista: navegue el sitio https://hub.docker.com/_/wordpress y https://hub.docker.com/_/mysql para acceder a los Dockerfiles que generaron esas imágenes y responder esta pregunta.)

MySQL: 3306 (solo accesible dentro de la red Docker wordpress).
WordPress: 80 (mapeado a 127.0.0.1:8000 en el host).

● ¿Qué servicio se “publica” para ser accedido desde el exterior y en qué puerto?¿Es necesario publicar el otro servicio? ¿Por qué?

WordPress está publicado en el puerto 127.0.0.1:8000 (mapeando el puerto 80 del contenedor al puerto 8000 del host, pero solo accesible localmente).
WordPress se comunica con MySQL internamente a través de la red Docker wordpress usando el nombre del servicio db como host y el puerto 3306

`Instanciando`

Cree un directorio llamada docker-compose-ej-1 donde prefiera, ubíquese dentro de éste y cree un archivo denominado docker-compose.yml pegando dentro el código anterior. La herramienta docker-compose, por defecto, espera encontrar en el directorio desde donde se la invoca un archivo docker-compose.yml (por eso lo creamos con ese nombre). Si existe, lee este archivo compose y realiza el despliegue de los recursos allí definidos.

Ahora, desde ese directorio ejecute el comando “docker compose up”, lo que resulta en el comienzo del despliegue de nuestros servicios. Como es la primera vez que lo corremos y si no tenemos las imágenes en la caché local de nuestro dispositivo, se descargan las imágenes de los dos servicios que estamos iniciando (recordar lo visto en la práctica anterior).

```
version: "3.9"

services:
  db:
        image: mysql:5.7
        networks:
        - wordpress
        volumes:
        - db_data:/var/lib/mysql
        restart: always
        environment:
        MYSQL_ROOT_PASSWORD: somewordpress
```

```
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpress
      MYSQL_PASSWORD: wordpress

  wordpress:
      depends_on:
      - db
      image: wordpress:latest
      networks:
      - wordpress
      volumes:
      - ${PWD}:/data
      - wordpress_data:/var/www/html
      ports:
      - "127.0.0.1:8000:80"
      restart: always
      environment:
      WORDPRESS_DB_HOST: db
      WORDPRESS_DB_USER: wordpress
      WORDPRESS_DB_PASSWORD: wordpress
      WORDPRESS_DB_NAME: wordpress

volumes:
  db_data: {}
  wordpress_data: {}

networks:
  wordpress:


~$ docker compose up
[+] Running 34/34
✔ wordpress 21 layers [⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿] 0B/0B Pulled
121.3s
✔ f03b40093957 Pull complete
8.2s
✔ 662d8f2fcdb9 Pull complete
9.4s
✔ 78fe0ef5ed77 Pull complete
27.5s
……..
108.8s
✔ db 11 layers [⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿] 0B/0B Pulled
104.9s
✔ e83e8f2e82cc Pull complete
31.6s
✔ 0f23deb01b84 Pull complete
38.2s
```

……
[+] Building 0.0s (0/0)
[+] Running 5/5
✔ Network so_wordpress Created
2.4s
✔ Volume "so_wordpress_data" Created
1.1s
✔ Volume "so_db_data" Created
1.1s
✔ Container so-db-1 Created
8.2s
✔ Container so-wordpress-1 Created
3.0s
Attaching to so-db-1, so-wordpress-1
so-db-1 | 2023-06-05 20:10:12+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 5.7.42-1.el7 started.
so-db-1 | 2023-06-05 20:10:12+00:00 [Note] [Entrypoint]: Switching to dedicated user 'mysql'
so-db-1 | 2023-06-05 20:10:12+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 5.7.42-1.el7 started
……..

En este punto, quedará la consola conectada a los servicios y estaremos viendo los logs exportados de los servicios. Si cerramos la consola o detenemos el proceso con ctrl+c, los servicios se darán de baja porque iniciamos los servicios en modo foreground. Para no quedar“pegados” a la consola podemos iniciar los servicios en modo “detached” de modo que queden corriendo en segundo plano (background), igual que como se hace con el comando “docker run -d IMAGE”:

~$ docker compose up -d

De esta manera veremos sólo información de que los servicios se inician y su nombre, pero la consola quedará “libre”.

Si quisiéramos conectarnos a alguno de los contenedores que docker-compose inició, por ejemplo el contenedor de wordpress, podemos hacerlo de la manera tradicional que se vio en la práctica de Docker (“docker exec [OPTIONS] CONTAINER COMMAND [ARG...]”) utilizando el identificador apropiado para el contenedor, o mediante el comando que docker-compose también brinda para hacerlo y usar su nombre de servicio (“wordpress” en este caso):

~$ docker compose exec wordpress /bin/bash
root@4dd0bcce2cb1:/var/www/html#

Aquí puedo enviar el comando /bin/bash porque el contenedor lo soporta; si eso no funcionase, la mayoría soportan al menos /bin/sh.

Y una vez dentro del contenedor, puedo navegar sus directorios normalmente. Si nos dirigimos al directorio /data, veremos dentro el contenido de nuestro directorio “docker-compose-ej-1” (solo tenemos el archivo docker-compose.yml) ya que montamos ese directorio como un volumen:

root@4dd0bcce2cb1:/var/www/html# cd /data/
root@4dd0bcce2cb1:/data# ls
docker-compose.yml

```
root@3755a7f57311:/var/www/html# cd /data/
root@3755a7f57311:/data# ls
docker-compose.yml
root@3755a7f57311:/data#
```

Y si creamos algún archivo dentro de este directorio, lo vemos también reflejado afuera del contenedor (el volumen montado como rw funciona en ambas direcciones).
Dentro del contenedor:
root@4dd0bcce2cb1:/data# touch test
root@4dd0bcce2cb1:/data# ls
docker-compose.yml test
En el host:
host:/docker compose-ej-1$ ls
docker-compose.yml test

```
root@3755a7f57311:/data# touch test
root@3755a7f57311:/data# ls
docker-compose.yml  test
root@3755a7f57311:/data#
```

```
root@so:/home/so# cd docker-compose-ej-1/
root@so:/home/so/docker-compose-ej-1# ls
docker-compose.yml  test
root@so:/home/so/docker-compose-ej-1#
```

Ahora que tenemos todo instanciado y funcionando, vamos a listar los servicios que iniciamos. Para esto vamos a correr:
~$ docker compose ps
Name Command State Ports
---
docker-compose- docker-entrypoint.sh Up 3306/tcp, 33060/tcp
ej-1_db_1 mysqld
docker-compose- docker-entrypoint.sh Up 127.0.0.1:8000->80/tcp
ej-1_wordpress_1 apach ...

```
NAME                                IMAGE              COMMAND
  SERVICE     CREATED         STATUS         PORTS
docker-compose-ej-1-db-1            mysql:5.7          "docker-entrypoint.s…"
  db          6 minutes ago   Up 4 minutes   3306/tcp, 33060/tcp
docker-compose-ej-1-wordpress-1     wordpress:latest   "docker-entrypoint.s…"
  wordpress   6 minutes ago   Up 4 minutes   127.0.0.1:8000->80/tcp
root@so:/home/so/docker-compose-ej-1#
```

Como se puede observar, el servicio denominado “docker-compose-ej-1_wordpress_1” está exponiendo el puerto 80 del contenedor en la dirección 127.0.0.1 puerto 8000 de nuestro dispositivo “host”. Esto quiere decir que tenemos en nuestro dispositivo un puerto 8000 “abierto” aceptando conexiones y si ingresamos

desde un navegador a la direccion “127.0.0.1:8000” veremos la página de inicio de la aplicación
Wordpress:

De este modo, hemos realizado el despliegue de una aplicación wordpress y de su base de datos mediante el uso de contenedores y la herramienta docker-compose. Desde este punto, solo queda continuar con la instalación de wordpress desde el browser.

Si queremos detener los servicios podemos ejecutar el comando:

```
~$ docker-compose stop
Stopping docker-compose-ej-1_wordpress_1 ... done
Stopping docker-compose-ej-1_db_1 ... done
```

y para eliminarlos:

```
~$ docker compose down
Removing docker-compose-ej-1_wordpress_1 ... done
Removing docker-compose-ej-1_db_1 ... done
Removing network docker-compose-ej-1_wordpress
```

Pero atención, esto elimina los contenedores pero no SUS VOLÚMENES DE DATOS, por lo que si volvemos a levantar los servicios por más que hayamos eliminado los contenedores, veremos que todas las modificaciones que hayamos realizado en la instalación de wordpress y datos agregados a la base de datos aún están presentes. Si queremos eliminar todo rastro de un despliegue previo, tendremos que eliminar los contenedores y también los volúmenes asociados utilizando el flag -v en docker-compose down:

```
~$ docker-compose down -v
Stopping docker-compose-ej-1_wordpress_1 ... done
Stopping docker-compose-ej-1_db_1 ... done
Removing docker-compose-ej-1_wordpress_1 ... done
Removing docker-compose-ej-1_db_1 ... done
Removing network docker-compose-ej-1_wordpress
Removing volume docker-compose-ej-1_db_data
Removing volume docker-compose-ej-1_wordpress_data
```

De esta manera, hemos eliminado todo lo instanciado por el docker compose. Solo quedan las imágenes Docker descargadas en la caché local del dispositivo, las cuales deben eliminar por su cuenta.