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 herramienta que te permite empacar una aplicación junto con todo lo que necesita para funcionar (como el sistema operativo, las librerías y dependencias) en un contenedor. De esta forma la aplicación está lista para funcionar en cualquier lugar, sin preocuparte por si el sistema de destino tiene todo configurado como lo necesita tu aplicación. **Docker te permite desarrollar, enviar y ejecutar aplicaciones de forma rápida y confiable sin importar el entorno en el que se ejecuten.**

Beneficios de los contenedores

- *Aislamiento ligero y eficiente:*
  - Cada contenedor tiene su propio entorno independiente gracias al uso de namespaces.
  - A diferencia de las máquinas virtuales, los contenedores no requieren un sistema operativo completo por cada instancia, lo que los hace más livianos y rápidos de iniciar.
- *Portabilidad y consistencia:*
  - Un contenedor corre igual en cualquier entorno: en tu computadora, en un servidor o en la nube, porque lleva todo consigo (app + dependencias).
  - Esto elimina el clásico "en mi máquina funciona".
- *Escalabilidad y facilidad de despliegue:*
  - Docker facilita la implementación masiva de servicios y su orquestación.
  - Es compatible con herramientas como Docker Compose para desplegar múltiples servicios en conjunto de forma sencilla.

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

Una Imagen es un template (molde) de solo lectura con todas las instrucciones para construir un contenedor. Puede basarse en otras imágenes y se compone de capas apiladas. Una forma fácil de comprender el concepto es pensar la Imagen como una receta de cocina.

Un Contenedor es una instancia de una imagen en ejecución. Tiene una capa adicional *escribible* donde se guardan los cambios realizados durante su ejecución. Una forma fácil de comprender el concepto es pensar la Contenedor como el plato servido en la mesa.

Diferencia principal

- *La imagen es el modelo inmutable*, mientras que *el contenedor es una instancia mutable y ejecutable* de esa imagen. Esto significa que desde una misma imagen podemos crear múltiples contenedores, cada uno con su propio entorno aislado y capacidad de ejecución.

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

Union Filesystem (UnionFS) es una técnica que permite montar múltiples sistemas de archivos en una única vista lógica. Es decir, **combina varias capas (read-only o writeable) en una sola estructura visible al usuario**.

Docker utiliza UnionFS como la base para **construir imágenes y contenedores en capas**:

- Cada *instrucción en un Dockerfile* (como `FROM`, `RUN`, `COPY`) crea una *nueva capa* en la imagen.
- Las capas inferiores son *de solo lectura*, mientras que la última capa, la del contenedor en ejecución, es *escribible*.
- Estas capas se *unen* mediante UnionFS para crear el sistema de archivos final del contenedor.
- Al eliminar un contenedor, se elimina solo su capa escribible; las demás capas no se modifican.
  Esto permite que las capas se **compartan entre imágenes** y se reutilicen, lo que reduce el uso de espacio y acelera las construcciones y despliegues.

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

Cuando Docker crea contenedores, estos suelen estar conectados por defecto a una red llamada `bridge`, que es una red virtual que Docker genera automáticamente al instalarse. **Los contenedores reciben direcciones IP privadas** del rango `172.17.0.0/16`, y por lo general, la IP del primer contenedor es `172.17.0.2`

¿De dónde obtiene Docker estas direcciones?

Docker crea internamente esta red usando un **bridge de Linux** (`docker0`) y configura automáticamente un **servidor DHCP interno** que asigna direcciones IP a los contenedores. Este DHCP forma parte del propio motor de red de Docker. Además:

- Docker puede crear **redes personalizadas** con otros rangos IP con `docker network create --subnet=...`.
- El rango predeterminado (`172.17.0.0/16`) puede modificarse en el archivo de configuración del daemon (`/etc/docker/daemon.json`).

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?

Por defecto, **los datos que se generan dentro de un contenedor se pierden al eliminarlo**, ya que se almacenan en su **capa escribible temporal**. Para evitar esto y lograr persistencia de datos (por ejemplo, para bases de datos, archivos de configuración, etc.), Docker permite **montar volúmenes externos**.

Formas de Persistencia

- *Volumes (volúmenes administrados por Docker)*
  - Se almacenan en una ruta interna del host (por defecto: `/var/lib/docker/volumes/`).
  - Docker los gestiona automáticamente.
  - Son independientes del layout del sistema operativo anfitrión.
  - Se declaran así: `docker run -v mi_volumen:/ruta/del/contenedor ...`
- *Bind Mounts (montajes enlazados)*
  - Permiten vincular una carpeta específica del host al contenedor.
  - Son más flexibles, pero dependen del sistema de archivos del host.
  - Se declaran así: `docker run -v /ruta/del/host:/ruta/del/contenedor ...`

Diferencias

Característica	Volumes	Bind Mounts
¿Quién lo administra?	Docker	El usuario (manual)
Ubicación	`/var/lib/docker/volumes/`	Cualquier carpeta del host
Portabilidad	Alta (independiente del host)	Baja (depende de ruta local)
Permisos	Docker controla el acceso	Más difícil de gestionar
Casos de uso	Bases de datos, persistencia confiable	Desarrollo local, debug, compartir código

Taller

> Versión del Kernel >
>
> No lo dice en ningún lado pero para todo esto hay que usar el kernel original que nos proveen, si intentamos hacerlo con el kernel compilado por nosotros Docker no funciona, yo estoy usando el `6.1.0-31-amd64`

Nota

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.

Para instalar Docker CE seguimos estos pasos que están especificados en la web de Docker

```
# para eliminar todos los paquetes conflictivos que podramos tener
for pkg in docker.io docker-doc docker-compose podman-docker containerd
runc; do sudo apt-get remove $pkg; done

# Add Docker's official GPG key:
sudo apt-get update
sudo apt-get install ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/debian/gpg -o
/etc/apt/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc

# Add the repository to Apt sources:
echo \
  "deb [arch=$(dpkg --print-architecture) signed-
by=/etc/apt/keyrings/docker.asc] https://download.docker.com/linux/debian \
  $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update

# instalamos la última versión con sus paquetes:
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-
plugin docker-compose-plugin

# verificamos la instalación:
so@so:/$ docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
```

```
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent
it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

2. Usando las herramientas (comandos) provistas por Docker realice las siguientes tareas:
    1. 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*?
    2. De la imagen obtenida en el punto anterior iniciar un contenedor que simplemente ejecute el comando `ls -l`.
    3. ¿Qué sucede si ejecuta el comando `docker [container] run ubuntu /bin/bash`[1]? ¿Puede utilizar la shell Bash del contenedor?
        1. 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?
        2. ¿Cuál es el PID del proceso bash en el contenedor? ¿Y fuera de éste?
        3. Ejecutar el comando `lsns`. ¿Qué puede decir de los namespace?
        4. 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).
        5. Corrobore si el archivo creado existe en el directorio raíz del sistema operativo anfitrión (host). ¿Existe? ¿Por qué?
    4. Vuelva a iniciar el contenedor anterior utilizando el mismo comando (con una terminal interactiva). ¿Existe el archivo creado en el contenedor? ¿Por qué?
    5. 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.
        1. ¿Cómo obtuvo el container_id para para este comando?

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

6. ¿Cuántos contenedores están actualmente en ejecución? ¿En qué estado se encuentra cada uno de los que se han ejecutado hasta el momento?
7. Elimine todos los contenedores creados hasta el momento. Indique el o los comandos utilizados.

> Info [1] >
>
> Los corchetes indican que el argumento `container` es opcional, pero no son parte del comando a ejecutar.

- 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*?

Obtención de la imagen y tamaño de la misma

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

No puede ser considerada un contenedor. Una imagen es un template (molde) de solo lectura con todas las instrucciones para construir un contenedor. Un contenedor es una instancia en ejecución de esa imagen.
Using default tag: latest significa que cuando hacemos `docker pull ubuntu` internamente se hace `docker pull ubuntu:latest`. `latest` es un *tag* por defecto que apunta a la versión más reciente estable de la imagen, pero **no siempre es la más nueva** ni la más recomendable para producción.

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

```
so@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 20 14:02 dev
drwxr-xr-x  32 root root 4096 May 20 14:02 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 177 root root    0 May 20 14:02 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 20 14:02 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
so@so:/$ docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED         STATUS
PORTS     NAMES
fd137fd205ba   ubuntu    "ls -l"   9 seconds ago   Exited (0) 6 seconds ago
agitated_galois
```

- ¿Qué sucede si ejecuta el comando `docker [container] run ubuntu /bin/bash`[1]? ¿Puede utilizar la shell Bash del contenedor?

```
so@so:/$ docker run ubuntu /bin/bash
so@so:/$
```

El contenedor arranca, ejecuta `/bin/bash`, pero no estamos en una shell interactiva, por lo tanto, el contenedor arranca y finaliza al instante (como pasó con `ls -l`).

- 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?

```
so@so:/$ docker run -it ubuntu /bin/bash
root@527c3f8d66b5:/#
```

Modificamos el comando para que use la opción `-it`:

- `-i`: Mantiene la entrada estándar abierta.
- `-t`: Asigna una terminal.
  De esta forma podemos usar Bash dentro del contenedor.
- ¿Cuál es el PID del proceso bash en el contenedor? ¿Y fuera de éste?

PID en el contenedor

```
root@527c3f8d66b5:/# ps aux
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  0.1  0.0   4588  3808 pts/0    Ss   14:05   0:00 /bin/bash
root           9 40.0  0.0   7888  3764 pts/0    R+   14:08   0:00 ps aux
```

PID en el Host

```
# docker inspect --format '{{.State.Pid}}' <container_id>
so@so:/$ docker inspect --format '{{.State.Pid}}' 527c3f8d66b5
0
```

Nos dan números distintos porque Docker usa namespaces.

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

`lsns` desde el Host

```
so@so:/$ lsns
        NS TYPE   NPROCS   PID USER COMMAND
4026531834 time       17   923 so   /lib/systemd/systemd --user
4026531835 cgroup     17   923 so   /lib/systemd/systemd --user
4026531836 pid        17   923 so   /lib/systemd/systemd --user
4026531837 user       17   923 so   /lib/systemd/systemd --user
4026531838 uts        17   923 so   /lib/systemd/systemd --user
4026531839 ipc        17   923 so   /lib/systemd/systemd --user
4026531840 net        17   923 so   /lib/systemd/systemd --user
4026531841 mnt        17   923 so   /lib/systemd/systemd --user
```

Listamos los namespaces activos. Cada contenedor corre en su propio conjunto de *namespaces*, por lo tanto, generamos un aislamiento entre el contenedor y el host.

- 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@527c3f8d66b5:/# touch /sistemas-operativos
root@527c3f8d66b5:/# exit
exit
```

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

```
so@so:/$ ls /sistemas-operativos
ls: no se puede acceder a '/sistemas-operativos': No existe el fichero o el
directorio
```

El sistema de archivos del contenedor está aislado del sistema anfitrión gracias a namespaces y layers. Cada contenedor tiene su propio filesystem virtual, por lo tanto, todo lo que se haga dentro del contenedor no afecta al host, salvo que montemos un volumen explícitamente para mantener los cambios.

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

```
so@so:/$ docker run -it ubuntu /bin/bash
root@4ae00b988247:/# ls /sistemas-operativos
ls: cannot access '/sistemas-operativos': No such file or directory
```

No, no existe el archivo. Cuando hacemos `docker run` de nuevo, creamos un contenedor completamente nuevo, con un sistema de archivos limpio basado en la imagen.

- 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.
  - ¿Cómo obtuvo el container_id para para este comando?
  - Chequee nuevamente si el archivo creado anteriormente existe. ¿Cuál es el resultado en este caso? ¿Puede encontrar el archivo creado?

Obtención del container_id

```
# es el 527c3f8d66b5
so@so:/$ docker ps -a
CONTAINER ID   IMAGE     COMMAND       CREATED          STATUS
PORTS     NAMES
4ae00b988247   ubuntu    "/bin/bash"   4 minutes ago    Up 4 minutes
quizzical_nash
```

```
527c3f8d66b5   ubuntu    "/bin/bash"   29 minutes ago   Exited (0) 11
minutes ago              eager_borg
a2ce86e58f15   ubuntu    "/bin/bash"   31 minutes ago   Exited (0) 31
minutes ago              happy_cannon
fd137fd205ba   ubuntu    "ls -l"       33 minutes ago   Exited (0) 33
minutes ago              agitated_galois
```

Chequeo del archivo

```
# -a: asjunta la entrada/salida del contenedor
so@so:/$ docker start -ia 527c3f8d66b5
root@527c3f8d66b5:/# ls /sistemas-operativos
/sistemas-operativos
```

Podemos encontrar el archivo porque estamos usando el mismo contenedor en el que creamos el archivo. Docker preserva el sistema de archivos de cada contenedor mientras no sea eliminado.

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

```
so@so:/$ docker ps -a
CONTAINER ID   IMAGE     COMMAND       CREATED          STATUS
PORTS     NAMES
4ae00b988247   ubuntu    "/bin/bash"   12 minutes ago   Up 11 minutes
quizzical_nash
527c3f8d66b5   ubuntu    "/bin/bash"   37 minutes ago   Exited (127) 5
seconds ago              eager_borg
a2ce86e58f15   ubuntu    "/bin/bash"   38 minutes ago   Exited (0) 38
minutes ago              happy_cannon
fd137fd205ba   ubuntu    "ls -l"       40 minutes ago   Exited (0) 40
minutes ago              agitated_galois
```

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

```
# docker rm $(docker ps -aq) elimina todos los contenedores (no las
imágenes) que no estén en ejecución
so@so:/$ docker rm $(docker ps -aq)
4ae00b988247
527c3f8d66b5
a2ce86e58f15
fd137fd205ba
```

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:
    1. Inicie un contenedor a partir de la imagen de Ubuntu descargada anteriormente ejecutando una consola interactiva de Bash.
    2. Instale el servidor web Nginx, https://nginx.org/en/, en el contenedor utilizando los siguientes comandos[2]:
    ```
    export DEBIAN_FRONTEND=noninteractive
    export TZ=America/Buenos_Aires
    apt update -qq
    apt install -y --no-install-recommends nginx
    ```
    3. Salga del contenedor y genere una imagen Docker a partir de éste. ¿Con qué nombre se genera si no se especifica uno?
    4. Cambie el nombre de la imagen creada de manera que en la columna Repository aparezca nginx-so y en la columna Tag aparezca v1.
    5. 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:
        1. 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.
        2. 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.
        3. Cree un contenedor a partir de la imagen nginx-so:v1 montando el directorio 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[3]. Indique el comando utilizado.
    6. Verifique que el contenedor esté ejecutándose correctamente abriendo un navegador web y visitando la URL http://localhost:8080.
    7. Modifique el archivo index.html agregándole un párrafo con su nombre y número de alumno. ¿Es necesario reiniciar el contenedor para ver los cambios?
    8. Analice: ¿por qué es necesario que el proceso nginx se ejecute en primer plano? ¿Qué ocurre si lo ejecuta sin -g 'daemon off;'?

Info [2] >

Los dos primeros comandos exportan dos variables de ambiente para que la instalación de una de las dependencias de nginx (el paquete `tzdata`) no requiera que interactivamente se respondan preguntas sobre la ubicación geográfica a utilizar

Info [3] >

Para iniciar el servidor nginx en primer plano utilice el comando `nginx -g 'daemon off;'`

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

```
so@so:/$ docker run -it ubuntu /bin/bash
root@f201f2ac9483:/#
```

- Instale el servidor web Nginx, https://nginx.org/en/, en el contenedor utilizando los siguientes comandos[2]:
  `export DEBIAN_FRONTEND=noninteractive`
  `export TZ=America/Buenos_Aires`
  `apt update -qq`
  `apt install -y --no-install-recommends nginx`

```
root@f201f2ac9483:/# export DEBIAN_FRONTEND=noninteractive
root@f201f2ac9483:/# export TZ=America/Buenos_Aires
root@f201f2ac9483:/# apt update -qq
All packages are up to date.
root@f201f2ac9483:/# apt install -y --no-install-recommends nginx
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  iproute2 libbpf1 libcap2-bin libelf1t64 libgssapi-krb5-2 libk5crypto3
  libkeyutils1 libkrb5-3 libkrb5support0 libmnl0 libtirpc-common
  libtirpc3t64 libxtables12 nginx-common
Suggested packages:
  iproute2-doc python3:any krb5-doc krb5-user fcgiwrap nginx-doc
  ssl-cert
Recommended packages:
```

```
  libatm1 libpam-cap krb5-locales
The following NEW packages will be installed:
  iproute2 libbpf1 libcap2-bin libelf1t64 libgssapi-krb5-2 libk5crypto3
  libkeyutils1 libkrb5-3 libkrb5support0 libmnl0 libtirpc-common
  libtirpc3t64 libxtables12 nginx nginx-common
0 upgraded, 15 newly installed, 0 to remove and 0 not upgraded.
Need to get 2684 kB of archives.
After this operation, 7795 kB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu noble-updates/main amd64 libelf1t64
amd64 0.190-1.1ubuntu0.1 [57.8 kB]
Get:2 http://archive.ubuntu.com/ubuntu noble/main amd64 libbpf1 amd64
1:1.3.0-2build2 [166 kB]
Get:3 http://archive.ubuntu.com/ubuntu noble/main amd64 libmnl0 amd64 1.0.5-
2build1 [12.3 kB]
Get:4 http://archive.ubuntu.com/ubuntu noble-updates/main amd64
libkrb5support0 amd64 1.20.1-6ubuntu2.5 [34.1 kB]
Get:5 http://archive.ubuntu.com/ubuntu noble-updates/main amd64 libk5crypto3
amd64 1.20.1-6ubuntu2.5 [82.0 kB]
Get:6 http://archive.ubuntu.com/ubuntu noble/main amd64 libkeyutils1 amd64
1.6.3-3build1 [9490 B]
Get:7 http://archive.ubuntu.com/ubuntu noble-updates/main amd64 libkrb5-3
amd64 1.20.1-6ubuntu2.5 [347 kB]
Get:8 http://archive.ubuntu.com/ubuntu noble-updates/main amd64 libgssapi-
krb5-2 amd64 1.20.1-6ubuntu2.5 [143 kB]
Get:9 http://archive.ubuntu.com/ubuntu noble/main amd64 libtirpc-common all
1.3.4+ds-1.1build1 [8094 B]
Get:10 http://archive.ubuntu.com/ubuntu noble/main amd64 libtirpc3t64 amd64
1.3.4+ds-1.1build1 [82.6 kB]
Get:11 http://archive.ubuntu.com/ubuntu noble/main amd64 libxtables12 amd64
1.8.10-3ubuntu2 [35.7 kB]
Get:12 http://archive.ubuntu.com/ubuntu noble-updates/main amd64 libcap2-bin
amd64 1:2.66-5ubuntu2.2 [34.2 kB]
Get:13 http://archive.ubuntu.com/ubuntu noble/main amd64 iproute2 amd64
6.1.0-1ubuntu6 [1120 kB]
Get:14 http://archive.ubuntu.com/ubuntu noble-updates/main amd64 nginx-
common all 1.24.0-2ubuntu7.3 [31.2 kB]
Get:15 http://archive.ubuntu.com/ubuntu noble-updates/main amd64 nginx amd64
1.24.0-2ubuntu7.3 [520 kB]
Fetched 2684 kB in 4s (686 kB/s)
debconf: delaying package configuration, since apt-utils is not installed
Selecting previously unselected package libelf1t64:amd64.
(Reading database ... 4381 files and directories currently installed.)
Preparing to unpack .../00-libelf1t64_0.190-1.1ubuntu0.1_amd64.deb ...
Unpacking libelf1t64:amd64 (0.190-1.1ubuntu0.1) ...
Selecting previously unselected package libbpf1:amd64.
Preparing to unpack .../01-libbpf1_1%3a1.3.0-2build2_amd64.deb ...
```

```
Unpacking libbpf1:amd64 (1:1.3.0-2build2) ...
Selecting previously unselected package libmnl0:amd64.
Preparing to unpack .../02-libmnl0_1.0.5-2build1_amd64.deb ...
Unpacking libmnl0:amd64 (1.0.5-2build1) ...
Selecting previously unselected package libkrb5support0:amd64.
Preparing to unpack .../03-libkrb5support0_1.20.1-6ubuntu2.5_amd64.deb ...
Unpacking libkrb5support0:amd64 (1.20.1-6ubuntu2.5) ...
Selecting previously unselected package libk5crypto3:amd64.
Preparing to unpack .../04-libk5crypto3_1.20.1-6ubuntu2.5_amd64.deb ...
Unpacking libk5crypto3:amd64 (1.20.1-6ubuntu2.5) ...
Selecting previously unselected package libkeyutils1:amd64.
Preparing to unpack .../05-libkeyutils1_1.6.3-3build1_amd64.deb ...
Unpacking libkeyutils1:amd64 (1.6.3-3build1) ...
Selecting previously unselected package libkrb5-3:amd64.
Preparing to unpack .../06-libkrb5-3_1.20.1-6ubuntu2.5_amd64.deb ...
Unpacking libkrb5-3:amd64 (1.20.1-6ubuntu2.5) ...
Selecting previously unselected package libgssapi-krb5-2:amd64.
Preparing to unpack .../07-libgssapi-krb5-2_1.20.1-6ubuntu2.5_amd64.deb ...
Unpacking libgssapi-krb5-2:amd64 (1.20.1-6ubuntu2.5) ...
Selecting previously unselected package libtirpc-common.
Preparing to unpack .../08-libtirpc-common_1.3.4+ds-1.1build1_all.deb ...
Unpacking libtirpc-common (1.3.4+ds-1.1build1) ...
Selecting previously unselected package libtirpc3t64:amd64.
Preparing to unpack .../09-libtirpc3t64_1.3.4+ds-1.1build1_amd64.deb ...
Adding 'diversion of /lib/x86_64-linux-gnu/libtirpc.so.3 to /lib/x86_64-
linux-gnu/libtirpc.so.3.usr-is-merged by libtirpc3t64'
Adding 'diversion of /lib/x86_64-linux-gnu/libtirpc.so.3.0.0 to /lib/x86_64-
linux-gnu/libtirpc.so.3.0.0.usr-is-merged by libtirpc3t64'
Unpacking libtirpc3t64:amd64 (1.3.4+ds-1.1build1) ...
Selecting previously unselected package libxtables12:amd64.
Preparing to unpack .../10-libxtables12_1.8.10-3ubuntu2_amd64.deb ...
Unpacking libxtables12:amd64 (1.8.10-3ubuntu2) ...
Selecting previously unselected package libcap2-bin.
Preparing to unpack .../11-libcap2-bin_1%3a2.66-5ubuntu2.2_amd64.deb ...
Unpacking libcap2-bin (1:2.66-5ubuntu2.2) ...
Selecting previously unselected package iproute2.
Preparing to unpack .../12-iproute2_6.1.0-1ubuntu6_amd64.deb ...
Unpacking iproute2 (6.1.0-1ubuntu6) ...
Selecting previously unselected package nginx-common.
Preparing to unpack .../13-nginx-common_1.24.0-2ubuntu7.3_all.deb ...
Unpacking nginx-common (1.24.0-2ubuntu7.3) ...
Selecting previously unselected package nginx.
Preparing to unpack .../14-nginx_1.24.0-2ubuntu7.3_amd64.deb ...
Unpacking nginx (1.24.0-2ubuntu7.3) ...
Setting up libkeyutils1:amd64 (1.6.3-3build1) ...
Setting up libtirpc-common (1.3.4+ds-1.1build1) ...
```

```
Setting up libelf1t64:amd64 (0.190-1.1ubuntu0.1) ...
Setting up libkrb5support0:amd64 (1.20.1-6ubuntu2.5) ...
Setting up libcap2-bin (1:2.66-5ubuntu2.2) ...
Setting up libmnl0:amd64 (1.0.5-2build1) ...
Setting up libk5crypto3:amd64 (1.20.1-6ubuntu2.5) ...
Setting up libxtables12:amd64 (1.8.10-3ubuntu2) ...
Setting up libkrb5-3:amd64 (1.20.1-6ubuntu2.5) ...
Setting up libbpf1:amd64 (1:1.3.0-2build2) ...
Setting up libgssapi-krb5-2:amd64 (1.20.1-6ubuntu2.5) ...
Setting up libtirpc3t64:amd64 (1.3.4+ds-1.1build1) ...
Setting up iproute2 (6.1.0-1ubuntu6) ...
Setting up nginx (1.24.0-2ubuntu7.3) ...
invoke-rc.d: unknown initscript, /etc/init.d/nginx not found.
invoke-rc.d: could not determine current runlevel
Setting up nginx-common (1.24.0-2ubuntu7.3) ...
Processing triggers for libc-bin (2.39-0ubuntu8.4) ...
root@f201f2ac9483:/# nginx -v
nginx version: nginx/1.24.0 (Ubuntu)
```

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

```
root@f201f2ac9483:/# exit
exit
so@so:/$ docker ps -a
CONTAINER ID   IMAGE     COMMAND       CREATED          STATUS
PORTS     NAMES
f201f2ac9483   ubuntu    "/bin/bash"   12 minutes ago   Exited (0) 34
seconds ago             adoring_feynman
# usamos commit para crear una imagen y le proveemos el container_id
so@so:/$ docker commit f201f2ac9483
sha256:acfffb5948177359e4e7fba39812b4ff1478a839ed01e962b64b28ec8f8733a2
# al no especificar ni nombre ni tags, se asigna <none>
so@so:/$ docker images
REPOSITORY    TAG       IMAGE ID       CREATED          SIZE
<none>        <none>    acfffb594817   16 seconds ago   135MB
ubuntu        latest    a0e45e2ce6e6   3 weeks ago      78.1MB
hello-world   latest    74cc54e27dc4   3 months ago     10.1kB
```

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

```
# a docker tag le mandamos el id de la imagen
so@so:/$ docker tag acfffb594817 nginx-so:v1
```

```
so@so:/$ docker images
REPOSITORY    TAG       IMAGE ID       CREATED         SIZE
nginx-so      v1        acfffb594817   2 minutes ago   135MB
ubuntu        latest    a0e45e2ce6e6   3 weeks ago     78.1MB
hello-world   latest    74cc54e27dc4   3 months ago    10.1kB
```

- 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:
  - 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.

```
mkdir -p /home/so/nginx-so
```

- 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.

```
so@so:/$ curl -o /home/so/nginx-so/index.html
https://gist.githubusercontent.com/ncuesta/5b959fce1c7d2ed4e5a06e84e5a7efc8/
raw/
  % Total    % Received % Xferd  Average Speed   Time    Time     Time
Current
                                 Dload  Upload   Total   Spent    Left
Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:-- 100
869  100   869    0     0    892      0 --:--:-- --:--:-- --:--:-- 100   869
100   869    0     0    892      0 --:--:-- --:--:-- --:--:--   945
so@so:/$ cat /home/so/nginx-so/index.html
<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.cs
s" rel="stylesheet" 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>

          <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>
```

- Cree un contenedor a partir de la imagen nginx-so:v1 montando el directorio 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[3]. Indique el comando utilizado.

```
so@so:/$ docker run -d -p 8080:80 -v /home/so/nginx-so:/var/www/html nginx-
so:v1 nginx -g 'daemon off;'
86f20791a84e0abf2d452e66a2c0f9b3398cf2d678a379ad2bd1347a9428b0c9
```

El comando hace lo siguiente:

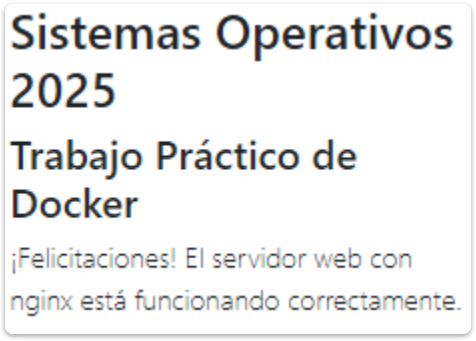
- `-d`: Ejecuta el contenedor en segundo plano.
- `-p 8080:80`: Mapea el puerto 8080 del host al puerto 80 del contenedor.
- `-v /home/so/nginx-so:/var/www/html`: Monta el directorio del host en `/var/www/html` del contenedor.
- `nginx-so:v1`: Es la imagen que creamos.
- `nginx -g 'daemon off;'`: Inicia Nginx en primer plano, es necesario para que el contenedor no se detenga.
- Verifique que el contenedor esté ejecutándose correctamente abriendo un navegador web y visitando la URL http://localhost:8080.

Haciendo curl desde la terminal

```
so@so:/$ 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.cs
s" rel="stylesheet" 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>

          <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>
```



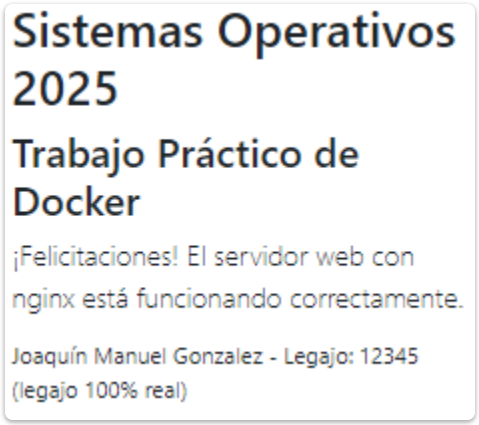
- Modifique el archivo index.html agregándole un párrafo con su nombre y número de alumno. ¿Es necesario reiniciar el contenedor para ver los cambios?

Haciendo curl desde la terminal

```
so@so:/$ 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.cs
s" rel="stylesheet" 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>

          <p class="lead">¡Felicitaciones! El servidor web con nginx está
funcionando correctamente.</p>
          <p>Joaquín Manuel Gonzalez - Legajo: 12345 (legajo 100% real)<p>
        </div>
      </div>
    </div>

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



No es necesario reiniciarlo. Como montamos el archivo con `-v /home/so/nginx-so:/var/www/html`, estamos usando un volumen (bind mount), es decir, el contenedor está viendo el archivo en tiempo real desde el host.

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

Esto es fundamental en Docker:

- Docker espera que el proceso principal del contenedor (el que se especifica en el comando de docker run) siga activo.
- Si ese proceso termina, Docker da por finalizado el contenedor.

Si ejecutamos sin `-g 'daemon off;'` Nginx intenta ejecutarse en **modo daemon (en segundo plano)**, por lo tanto el proceso principal finaliza y **el contenedor se detiene inmediatamente**.

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.
    1. En el directorio del host creado en el punto anterior (/home/so/nginx-so), cree un archivo Dockerfile que realice los siguientes pasos:
        1. Comenzar en base a la imagen oficial de Ubuntu.
        2. Exponer el puerto 80 del contenedor.
        3. Instalar el servidor web nginx.
        4. Copiar el archivo index.html del mismo directorio del host al directorio /var/www/html de la imagen.
        5. 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.*
    2. Utilizando el Dockerfile que generó en el punto anterior construya una nueva imagen Docker guardándola localmente con el nombre nginx-so:v2.
    3. 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.
    4. 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?
    5. 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?

6. 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?

> Info [4] >
>
> La documentación oficial de Docker describe las tres formas posibles para indicar el comando principal de una imagen: https://docs.docker.com/engine/reference/builder/#cmd.

- 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.
  - En el directorio del host creado en el punto anterior (/home/so/nginx-so), cree un archivo Dockerfile que realice los siguientes pasos:
    - Comenzar en base a la imagen oficial de Ubuntu.
    - Exponer el puerto 80 del contenedor.
    - Instalar el servidor web nginx.
    - Copiar el archivo index.html del mismo directorio del host al directorio /var/www/html de la imagen.
    - 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.*

Contenido del Dockerfile

```
# 1. Imagen base
FROM ubuntu:latest

# 2. Exponer el puerto 80
EXPOSE 80

# 3. Instalar nginx (sin interacción)
ENV DEBIAN_FRONTEND=noninteractive
ENV TZ=America/Buenos_Aires

RUN apt update -qq && \
```

```
    apt install -y --no-install-recommends nginx && \
    apt clean && \
    rm -rf /var/lib/apt/lists/*

# 4. Copiar archivo index.html al contenedor
COPY index.html /var/www/html/

# 5. Comando para iniciar nginx en primer plano (forma exec)
CMD ["nginx", "-g", "daemon off;"]
```

Explicación de las instrucciones

- `FROM`: Usa Ubuntu como base.
- `EXPOSE`: Indica que la imagen espera tráfico en el puerto 80 (no lo publica).
- `ENV`: Define variables de entorno (en este caso para que `tzdata` no sea interactivo).
- `RUN`: Actualiza paquetes e instala `nginx`.
- `COPY`: Copia el archivo `index.html` desde el host al contenedor.
- `CMD`: Define el comando por defecto al ejecutar un contenedor desde esta imagen. La *forma exec* garantiza que las señales (ej. SIGTERM) se manejen correctamente.
- Utilizando el Dockerfile que generó en el punto anterior construya una nueva imagen Docker guardándola localmente con el nombre nginx-so:v2.

```
so@so:~/nginx-so$ docker build -t nginx-so:v2 .
[+] Building 49.8s (8/8) FINISHED
docker:default
 => [internal] load build definition from Dockerfile
0.2s
 => => transferring dockerfile: 519B
0.0s
 => [internal] load metadata for docker.io/library/ubuntu:latest
0.0s
 => [internal] load .dockerignore
0.1s
 => => transferring context: 2B
0.0s
 => [1/3] FROM docker.io/library/ubuntu:latest
0.1s
 => [internal] load build context
0.3s
 => => transferring context: 977B
0.0s
 => [2/3] RUN apt update -qq &&     apt install -y --no-install-recommends
nginx &&       42.3s
 => [3/3] COPY index.html /var/www/html/
```

```
0.3s
 ⇒ exporting to image
6.1s
 ⇒ ⇒ exporting layers
5.9s
 ⇒ ⇒ writing image
sha256:52e1bcaa5d6ae6b404345ee1ae340568c14234cd78120ee3d070e2852df42  0.0s
 ⇒ ⇒ naming to docker.io/library/nginx-so:v2
0.0s
so@so:~/nginx-so$ docker images
REPOSITORY    TAG       IMAGE ID       CREATED          SIZE
nginx-so      v2        52e1bcaa5d6a   14 seconds ago   86.6MB
nginx-so      v1        acfffb594817   24 hours ago     135MB
ubuntu        latest    a0e45e2ce6e6   3 weeks ago      78.1MB
hello-world   latest    74cc54e27dc4   3 months ago     10.1kB
```

Usamos el comando `docker build`

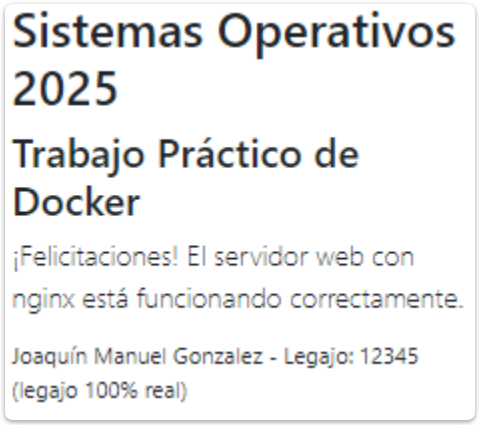
- `docker build` se utiliza para construir una Imagen de Docker a partir de un Dockerfile y un contexto (generalmente un directorio que contiene los archivos necesarios).
- `-t nginx-so:v2`: `-t` (o `--tag`) asigna un nombre y una etiqueta (`tag`) a la imagen. En este caso, la imagen se llamará `nginx-so` y tendrá la versión `v2`.
- `.`: Es el *contexto de construcción*, que generalmente es el directorio actual donde se encuentra el `Dockerfile`. Docker copia los archivos de este directorio para usarlos durante la construcción.
- 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.

```
so@so:~/nginx-so$ docker run -d -p 8090:80 --name nginx_v2_container nginx-
so:v2
d3fe735c04e74b373a69b68cadf7231ee82fb094019f0fa6baae68b7855dd32b
so@so:~/nginx-so$ 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.cs
s" rel="stylesheet" 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>

          <p class="lead">¡Felicitaciones! El servidor web con nginx está
funcionando correctamente.</p>
          <p>Joaquín Manuel Gonzalez - Legajo: 12345 (legajo 100% real)<p>
        </div>
      </div>
    </div>

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



- 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?

Nuevo Contenido de `index.html`

```
<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.cs
s" rel="stylesheet" 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>
          <p class="lead">¡Felicitaciones! El servidor web con nginx está
funcionando correctamente.</p>
    <p>Joaquín Manuel Gonzalez - Legajo: 12345 (legajo 100% real)<p>
    <p>Fecha actual: 21 de Mayo de 2025<p>
        </div>
      </div>
    </div>
    <script>document.getElementById('y').innerHTML = new
Date().getFullYear()</script>
  </body>
</html>
```

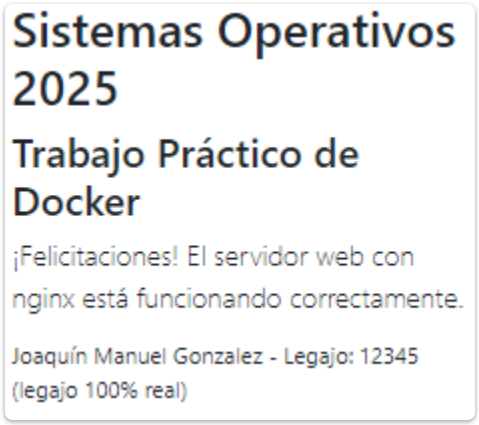
Consulta con `curl`

```
so@so:~/nginx-so$ 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.cs
s" rel="stylesheet" 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>

          <p class="lead">¡Felicitaciones! El servidor web con nginx está
funcionando correctamente.</p>
          <p>Joaquín Manuel Gonzalez - Legajo: 12345 (legajo 100% real)<p>
        </div>
```

```
      </div>
    </div>

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



No, no se verá reflejado el cambio, porque el archivo `index.html` fue copiado dentro de la imagen durante la construcción. No hay conexión con el archivo `index.html` del host en este contenedor.

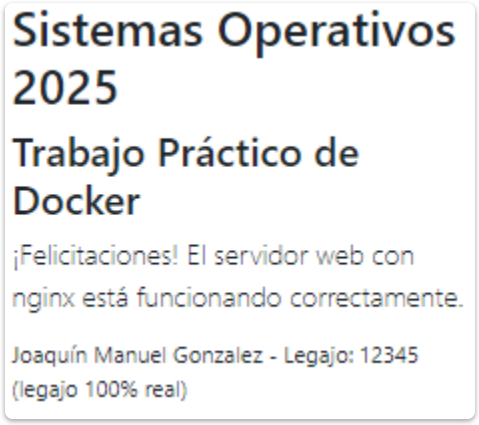
- Termine el contenedor iniciado antes y cree uno nuevo utilizando el mismo comando. Recargue la página en su navegador web. ¿Se ven ahora reflejados los cambios realizados en el archivo HTML? ¿Por qué?

```
so@so:~/nginx-so$ docker stop nginx_v2_container
nginx_v2_container
so@so:~/nginx-so$ docker rm nginx_v2_container
nginx_v2_container
so@so:~/nginx-so$ docker run -d -p 8090:80 --name nginx_v2_container nginx-
so:v2
063fef96934e2bad9f2c10443aa46b099cc7e39d9769fa9c5277af8158396cf6
so@so:~/nginx-so$ 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.cs
s" rel="stylesheet" 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>

          <p class="lead">¡Felicitaciones! El servidor web con nginx está
funcionando correctamente.</p>
          <p>Joaquín Manuel Gonzalez - Legajo: 12345 (legajo 100% real)<p>
        </div>
      </div>
    </div>

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



No, sigue sin verse. Porque el archivo HTML fue copiado al contenedor en la construcción de la imagen. La imagen no se vuelve a construir sola ni ve los cambios hechos en el host.

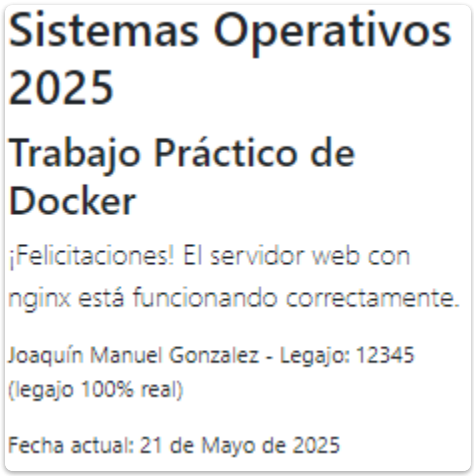
- Vuelva a construir una imagen Docker a partir del Dockerfile creado anteriormente, pero esta vez dándole el nombre nginx-so:v3. Cree un contenedor a partir de ésta y acceda a la página en su navegador web. ¿Se ven reflejados los cambios realizados en el archivo HTML? ¿Por qué?

```
so@so:~/nginx-so$ docker build -t nginx-so:v3 .
[+] Building 4.1s (8/8) FINISHED
docker:default
 ⇒ [internal] load build definition from Dockerfile
0.1s
 ⇒ ⇒ transferring dockerfile: 519B
```

```
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
0.1s
 ⇒ ⇒ transferring context: 1.02kB
0.0s
 ⇒ CACHED [2/3] RUN apt update -qq &&     apt install -y --no-install-
recommends nginx &  0.0s
 ⇒ [3/3] COPY index.html /var/www/html/
0.1s
 ⇒ exporting to image
3.3s
 ⇒ ⇒ exporting layers
3.2s
 ⇒ ⇒ writing image
sha256:cdbb16f6f3c7b273e9f36c4688107bef3de79cbd24afb87944aa0fa109686  0.0s
 ⇒ ⇒ naming to docker.io/library/nginx-so:v3
0.0s
so@so:~/nginx-so$ docker run -d -p 8090:80 --name nginx_v3_container nginx-
so:v3
08905c4afd20137c34f4ce6e8062502ff13142bd2042a655447fcdc958173920
so@so:~/nginx-so$ 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.cs
s" rel="stylesheet" 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>
```

```
          <p class="lead">¡Felicitaciones! El servidor web con nginx está
funcionando correctamente.</p>
          <p>Joaquín Manuel Gonzalez - Legajo: 12345 (legajo 100% real)<p>
          <p>Fecha actual: 21 de Mayo de 2025<p>
        </div>
      </div>
    </div>

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



Sí, ahora sí se ve el cambio. Porque el `index.html` actualizado fue copiado nuevamente a la imagen durante esta nueva construcción (`v3`).

Docker Compose

1. Utilizando sus palabras describa, ¿qué es docker compose?

Docker Compose es una herramienta oficial de Docker que permite definir, configurar y ejecutar aplicaciones que requieren múltiples contenedores de forma sencilla y declarativa. Se usa principalmente cuando una aplicación no está compuesta por un solo contenedor, sino por varios servicios que deben funcionar en conjunto.

2. ¿Qué es el archivo compose y cual es su función? ¿Cuál es el “lenguaje” del archivo?

El archivo compose es un archivo de configuración que se utiliza con Docker Compose para definir los servicios que componen una aplicación multi-contenedor. Su función principal es: Describir qué servicios (contenedores) necesita nuestra aplicación, cómo deben ejecutarse, qué variables de entorno usar, qué redes y volúmenes compartir, etc. Es el que orquesta el

despliegue de múltiples contenedores de forma declarativa y automática. **Este archivo puede contener:**

- Imágenes a usar o Dockerfiles a construir.
- Comandos a ejecutar al iniciar cada contenedor.
- Variables de entorno.
- Volúmenes montados entre host y contenedor.
- Puertos mapeados (host:contenedor).
- Redes personalizadas.
- Políticas de reinicio.
- Dependencias entre servicios.

El archivo compose está escrito en **YAML** (Yet Another Markup Language), que es un **lenguaje de marcado legible por humanos** muy utilizado para archivos de configuración. **Características de YAML:**

- Usa indentación para definir jerarquías (no se usan llaves ni corchetes como en JSON).
- Es declarativo, no ejecutable.
- Muy usado en DevOps (Kubernetes también usa YAML, por ejemplo).

3. ¿Cuáles son las versiones existentes del archivo docker-compose.yaml existentes y qué características aporta cada una? ¿Son compatibles entre sí? ¿Por qué?

Versiones del archivo `docker-compose.yml`:

- *Versión 1 (Obsoleta):*
  - Era el formato original, muy limitado.
  - No soportaba muchas características modernas como redes o volúmenes nombrados.
  - Ya no se recomienda usarla.
- *Versión 2.x (2006-2020 aprox.):*
  - Introdujo la posibilidad de definir volúmenes, redes, y políticas de reinicio.
  - Permitió más opciones para orquestación básica.
  - Comenzó a acercarse al formato de Docker Swarm.
- *Versión 3.x (actual recomendada):*
  - Diseñada para ser compatible con Docker Swarm Mode, el orquestador nativo de Docker.
  - Introdujo conceptos como:
    - `deploy` (para escalado, replicación, políticas de actualización, etc.)
    - `configs` y `secrets`
    - Mejores redes

- Última versión estable y recomendada: 3.9 (desde junio de 2020).

Compatibilidad entre sí

Compatibilidad	¿Qué significa?	¿Es posible?
Hacia atrás	Usar una versión nueva del binario Docker Compose con archivos viejos	Generalmente sí, pero puede haber advertencias o limitaciones.
Hacia adelante	Usar un archivo moderno (ej. v3.9) con una versión vieja del binario	Puede fallar si el binario no entiende algunas claves nuevas.

Es importante ver que aunque la estructura general entre `2.x` y `3.x` es similar, **algunas claves cambian o dejan de estar disponibles** dependiendo de si estás usando Docker Compose tradicional o Docker Swarm.

4. Investigue y describa la estructura de un archivo compose. Desarrolle al menos sobre los siguientes bloques indicando para qué se usan:
    1. services
    2. build
    3. image
    4. volumes
    5. restart
    6. depends_on
    7. environment
    8. ports
    9. expose
    10. networks

Estructura general de un archivo `docker-compose.yml`

```yaml
version: "3.9"
services:
  nombre_servicio:
    image: ...
    build: ...
    ports: ...
    environment: ...
    volumes: ...
    depends_on: ...
    restart: ...
    expose: ...
```

```
    networks: ...
volumes:
  ...
networks:
  ...
```

`services`

- Define los contenedores (servicios) que componen la aplicación. Cada clave dentro de `services` representa un contenedor distinto. Cada `service` se traduce en un contenedor Docker al ejecutarlo con `docker compose up`.

```
services:
  web:
    image: nginx
  db:
    image: mysql
```

`build`

- Se usa para construir una imagen personalizada a partir de un `Dockerfile`.

```
services:
  web:
    build: ./web

# o más detallado

build:
  context: ./web
  dockerfile: Dockerfile.dev
```

`image`

- Indica la imagen a utilizar para el contenedor. Puede ser:
  - Una imagen pública (de Docker Hub)
  - Una imagen local ya construida
  - Un nombre para etiquetar si se usa junto con build

```
image: nginx:latest
```

`volumes`

- Permite montar volúmenes entre el host y el contenedor, o usar volúmenes gestionados por Docker para persistencia de datos.

```
# para un servicio
volumes:
  - ./html:/usr/share/nginx/html
  - data:/var/lib/mysql

# en la raíz
volumes:
  data:  # volumen nombrado
```

`restart`

- Define la política de reinicio automático del contenedor en caso de falla o reinicio del sistema. Valores posibles:
  - `no` (default): no reinicia
  - `always`: siempre reinicia
  - `on-failure`: solo si falla
  - `unless-stopped`: reinicia excepto si fue detenido manualmente

```
restart: always
```

`depends_on`

- Indica que un servicio depende de otro y debe arrancar después.

```
services:
  web:
    depends_on:
      - db
```

> **Nota**
>
> Esto no garantiza que `db` esté completamente lista antes de iniciar `web`. Para eso, se recomienda agregar una lógica de espera (e.g., script `wait-for-it.sh` o `healthcheck`)

`enviroment`

- Define variables de entorno que se pasan al contenedor. Puede cargarse desde un archivo `.env`.

```
environment:
  MYSQL_ROOT_PASSWORD: mypass
  MYSQL_DATABASE: mydb
```

`ports`

- Mapea puertos del host a puertos del contenedor.

```
ports:
  - "8080:80"
```

`expose`

- Expone puertos solo para otros contenedores en la misma red, no al host. Es útil para que los contenedores se comuniquen internamente, sin exponer el puerto fuera del entorno Compose.

```
expose:
  - "3306"
```

`networks`

- Permite definir o unir servicios a redes personalizadas, en lugar de usar la red default.

```
# al final del archivo
networks:
  backend:
    driver: bridge

# en cada servicio
services:
  web:
    networks:
      - backend
```

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?

Lo que pasa como problema principal es que el `depends_on` solo garantiza el orden de arranque de los contenedores, no espera a que estén listos. Es decir, el backend de la aplicación Web podría empezar mientras la base de daos todavía se está inicializando.

La solución correcta para estos casos es usar `healthcheck` + `depends_on` + una lógica de espera

- *Usar `healthcheck` en la base de datos:*
  - Esto define una condición de "salud" para que Docker Compose sepa si el servicio está verdaderamente listo.
  - Ejemplo de como se podría aplicar (Docker marcará el contenedor como "healthy" solo si el comando `pg_isready` confirma que la base está lista):

```
services:
  db:
    image: postgres:15
    environment:
      POSTGRES_PASSWORD: example
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U postgres"]
      interval: 10s
      timeout: 5s
      retries: 5
```

- *Agregar lógica de espera en el backend:*
  - Como `depends_on` no espera healthcheck, necesitamos que el backend espere activamente que `db` esté "healthy" antes de iniciarse.
  - Podemos usar para esto un script como `wait-for.it.sh` o `dockerize`.
  - Ejemplo de un Dockerfile de backend (Esperamos a que el puerto 5432 esté disponible antes de correr la aplicación):

```
COPY wait-for-it.sh /wait-for-it.sh
ENTRYPOINT ["/wait-for-it.sh", "db:5432", "--", "npm", "start"]
```

- *Opcionalmente con `Compose v3.9` podemos usar `condition: service_healthy` (Solo en Swarm).*
  - Ejemplo que no es soportado en Docker Compose tradicional:

```
depends_on:
  db:
    condition: service_healthy
```

6. Indique qué hacen y cuáles son las diferencias entre los siguientes comandos:
    1. `docker compose create` y `docker compose up`
    2. `docker compose stop` y `docker compose down`
    3. `docker compose run` y `docker compose exec`
    4. `docker compose ps`
    5. `docker compose logs`

`docker compose create` y `docker compose up`

- `docker compose create`
    - Crea los contenedores definidos en el archivo `compose.yml`, pero no los inicia. Es útil para hacer revisiones antes de ejecutarlos.
    - Luego de ejecutarlo no vamos a ver que los servicios funcionen todavía.
- `docker compose up`
    - Crea e inicia los contenedores automáticamente.
    - Si los contenedores ya existen, simplemente los arranca.
    - Si se usa `-d`, lo hace en segundo plano.
- *Diferencia clave:*
    - `create` sólo prepara, `up` prepara y ejecuta.

`docker compose stop` y `docker compose down`

- `docker compose stop`
    - Detiene los contenedores, pero los mantiene creados.
    - Podemos reiniciarlos después con `docker compose start`.
- `docker compose down`
    - Detiene y elimina los contenedores, redes, volúmenes *anónimos*, etc.
    - Deja el sistema limpio.
- *Diferencia clave:*
    - `stop` conserva los recursos, `down` los elimina

`docker compose run` y `docker compose exec`

- `docker compose run`
    - Crea y ejecuta un contenedor nuevo temporal basado en un servicio.
    - Usado para ejecutar comandos *ad-hoc* (tests, scripts).
    - Ejemplo: `docker compose run backend ls /app`
- `docker compose exec`
    - Ejecuta un comando dentro de un contenedor ya en ejecución.

    - Ideal para debugging o entrar a un servicio.
    - Ejemplo: `docker compose exec backend bash`
- *Diferencia clave:*
    - `run` crea un nuevo contenedor, `exec` usa uno ya activo.

`docker compose ps`

- Muestra la lista de contenedores activos definidos en el `compose.yml`.
- Similar a `docker ps`, pero filtrado para el proyecto Compose.

`docker compose logs`

- Muestra los logs (salida estándar) de los servicios.
- Por defecto muestra los de todos, pero podemos filtrar por servicio.
- Ejemplo: `docker compose logs`, para un servicio `docker compose logs db`, en modo seguimiento `docker compose -f web`.

7. ¿Qué tipo de volúmenes puede utilizar con docker compose? ¿Cómo se declara cada tipo en el archivo compose?

Tipos de volúmenes en Docker Compose

- *Volúmenes nombrados (named volumes)*
    - Gestionados completamente por Docker. Sirven para persistir datos sin importar en qué máquina se corra el contenedor, mientras se tenga acceso al volumen.
    - *Ventajas:*
        - Docker se encarga de crearlos y ubicarlos.
        - Persisten incluso después de eliminar contenedores.
        - No dependen de rutas locales del host.
    - Declaración en el `compose.yml` (en el ejemplo `dbdata` es el nombre del volumen y `/var/lib/mysql` es la ruta dentro del contenedor donde se montará):

```yaml
services:
  db:
    image: mysql
    volumes:
      - dbdata:/var/lib/mysql

volumes:
  dbdata:
```

- *Volúmenes de tipo bind (bind mounts)*

- Montan un directorio o archivo directamente desde el sistema de archivos del host. Sirven para desarrollar localmente y ver los cambios en tiempo real dentro del contenedor.
- *Ventajas:*
  - Muy útil en desarrollo (cambios en vivo).
  - Permite compartir archivos con el contenedor.
- *Desventajas:*
  - No tan portables (depende de rutas del host).
  - No recomendables en producción sin control.
- Declaración en el `compose.yml` (en el ejemplo `./html` es un directorio en el host (ruta relativa al `docker-compose.yml`) y `/usr/share/nginx/html` es la ruta en el contenedor)

```
services:
  web:
    image: nginx
    volumes:
      - ./html:/usr/share/nginx/html
```

8. ¿Qué sucede si en lugar de usar el comando "docker compose down" utilizo "docker compose down -v/--volumes"?

Si en ve de ejecutar `docker compose down` hacemos `docker compose down -v` se va a hacer lo mismo que `docker compose down` pero adicionalmente, se eliminan también todos los volúmenes nombrados creados por ese proyecto Compose. Es importante la diferencia porque normalmente usamos los volúmenes nombrados para persistir datos importantes si usamos `-v` perdemos los datos.

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-compose-
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
```

Instalación

```
root@so:~# curl -SL
https://github.com/docker/compose/releases/download/v2.18.1/docker-compose-
linux-x86_64 -o /usr/local/bin/docker-compose
  % Total    % Received % Xferd  Average Speed   Time    Time     Time
Current
                                 Dload  Upload   Total   Spent    Left
Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--
0
100 52.0M  100 52.0M    0     0  5377k      0  0:00:09  0:00:09 --:--:--
7045k
root@so:~# chmod +x /usr/local/bin/docker-compose
root@so:~# docker compose version
Docker Compose version v2.35.1
```

Ejercicio guiado - Instanciando un Wordpress y una Base de Datos

Dado el siguiente código de archivo compose

```yaml
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?
- ¿Por qué no se necesitan Dockerfiles?
- ¿Por qué el servicio identificado como “wordpress” tiene la siguiente línea?

```
depends_on:
    - db
```

- ¿Qué volúmenes y de qué tipo tendrá asociado cada contenedor?
- ¿Por que uso el volumen nombrado (VER ABAJO) para el servicio `db` en lugar de dejar que se instancie un volumen anónimo con el contenedor?

```
volumes:
    - db_data:/var/lib/mysql
```

- ¿Qué genera la línea (VER ABAJO) en la definición de `wordpress`?

```
volumes:
    - ${PWD}:/data
```

- ¿Qué representa la información que estoy definiendo en el bloque `environment` de cada servicio? ¿Cómo se “mapean” al instanciar los contenedores?
- ¿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`)
- ¿Cómo sabe comunicarse el contenedor “wordpress” con el contenedor “db” si nunca doy información de direccionamiento?
- ¿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.)
- ¿Qué servicio se “publica” para ser accedido desde el exterior y en qué puerto? ¿Es necesario publicar el otro servicio? ¿Por qué?
- **¿Cuántos contenedores se instancian?**

Se instancian *2 contenedores* `db` y `wordpress`.

- **¿Por qué no se necesitan Dockerfiles?**

No se necesitan porque los dos servicios que usamos *tienen imágenes oficiales ya preparadas y listas para usar* (`mysql:5.7` y `wordpress:latest`). Usamos los Dockerfiles cuando *queremos construir una imagen personalizada*.

- ¿Por qué el servicio identificado como “wordpress” tiene la siguiente línea?

```
depends_on:
    - db
```

El servicio `wordpress` tiene esa línea porque necesita conectarse a la base de datos al arrancar, para eso tiene sus variables `WORDPRESS_DB_HOST: db` y `WORDPRESS_DB_USER: wordpress`. Si el contenedor `wordpress` se ejecutara antes que `db`, podría fallar la conexión inicial al MySQL. Por eso `depends_on` asegura que al menos `db` se inicie antes que `wordpress`, no garantiza que `db` esté listo para cuando `wordpress` se quiera conectar.

- ¿Qué volúmenes y de qué tipo tendrá asociado cada contenedor?

El contenedor `db` usa un *named volume* `- db_data:/var/lib/mysql`. El contenedor `wordpress` usa un *bind mount* `- ${PWD}:/data` y un *named volume* `- wordpress_data:/var/www/html`.

- ¿Por que uso el volumen nombrado (VER ABAJO) para el servicio `db` en lugar de dejar que se instancie un volumen anónimo con el contenedor?

```
volumes:
    - db_data:/var/lib/mysql
```

Usamos el volumen nombrado porque de esa forma *creamos un volumen persistente y reutilizable con nombre (db_data) que Docker puede identificar y conservar, incluso si eliminamos o recreamos el contenedor*. Si no lo nombramos, Docker crea un *volumen anónimo* como `/var/lib/docker/volumes/2fcd91c7b7b87a03a2.../_data` que no tiene nombre, se pierde fácilmente y es difícil de reutilizar o gestionar. *Ventajas de nombrarlos:*

Ventaja	Explicación
Persistencia controlada	El volumen `db_data` persiste aunque el contenedor `db` sea eliminado.
Limpieza segura	Podemos eliminarlo manualmente con `docker volume rm db_data` si lo deseamos.
Inspección y visibilidad	Es más fácil de listar (`docker volume ls`) e inspeccionar (`inspect`).
Gestión vía Compose	Podemos declararlo y versionarlo en nuestro `compose.yml`.
Reutilización entre contenedores	Podríamos compartir `db_data` con otro contenedor si lo necesitamos.

- ¿Qué genera la línea (VER ABAJO) en la definición de `wordpress`?

```
volumes:
    - ${PWD}:/data
```

Esa línea monta un volumen *bind mount* que toma el directorio actual del host (donde ejecutamos `docker compose up`) y lo monta dentro del contenedor Wordpress en la ruta `/data`. *El efecto que tiene en el contendor es:*

- El directorio del host se hace accesible dentro del contenedor en la ruta `/data`.
- *Sirve para:*
  - Compartir archivos con el contenedor.
  - Guardar archivos generados por Wordpress en el host.
- ¿Qué representa la información que estoy definiendo en el bloque `environment` de cada servicio? ¿Cómo se "mapean" al instanciar los contenedores?

En el bloque `environment` de cada servicio definimos las *variables de entorno* que van a ser inyectadas dentro del contenedor cuando se cree e instancie.

En el contenedor `db`

- `MYSQL_ROOT_PASSWORD`, `MYSQL_DATABASE`, `MYSQL_USER`, `MYSQL_PASSWORD` configuran la base de datos MySQL inicial:
  - La contraseña del usuario root.
  - El nombre de la base de datos que se crea automáticamente.
  - El usuario y su contraseña que podrá acceder a esa base de datos.

  En el contenedor `wordpress`
- `WORDPRESS_DB_HOST`, `WORDPRESS_DB_USER`, `WORDPRESS_DB_PASSWORD`, `WORDPRESS_DB_NAME` configuran la conexión que el contenedor WordPress hará a la base de datos MySQL:
  - Host (el nombre del servicio `db` dentro de la red Docker).
  - Usuario, contraseña y base de datos para conectarse.

¿Cómo se mapean?

- Cuando Docker Compose crea el contenedor para cada servicio:
  - Toma las variables definidas en `environment`.
  - Las inyecta dentro del contenedor como variables de entorno del sistema operativo (por ejemplo, accesibles con `echo $MYSQL_ROOT_PASSWORD` dentro del contenedor).
  - El software dentro del contenedor, que está programado para leer esas variables, las utiliza para su configuración inicial.

- ¿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`)

Si hacemos ese cambio vamos a lograr que `wordpress` no se pueda conectar a la base de datos ya que el nombre no es el mismo. Los contenedores si bien están conectados en la misma red, no sincronizan el contenido de sus variables de entorno. *Nosotros somos los responsables de que las configuraciones coincidan cuando dependen entre sí.*

- ¿Cómo sabe comunicarse el contenedor "wordpress" con el contenedor "db" si nunca doy información de direccionamiento?

`wordpress` sabe comunicarse con `db` porque están conectados en la misma red definida como `wordpress` en el `compose.yml`. Gracias a eso:

- Docker Compose crea una red virtual interna para que los contenedores se comuniquen.
- Cada contenedor puede ser accedido por su nombre de servicio como si fuera un hostname dentro de esa red.

Los accesos se resuelven automáticamente por el DNS interno de Docker:

- Docker levanta un servicio de resolución de DNS interno.
- Cualquier contenedor puede hacer una petición a `db` y Docker sabe que se refiere al contenedor del servicio `db`.
- Esto no requiere que se defina una IP fija ni que se configure archivos `hosts`.
- ¿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.)

La imagen oficial de MySQL expone el puerto 3306/TCP, que es el puerto estándar para conexiones al servidor de bases de datos MySQL. La imagen oficial de WordPress basada en Apache expone el puerto 80/TCP, que es el puerto estándar para tráfico HTTP.

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

El servicio que se "publica" es `wordpress` viendo que en la sección `ports` define `- "127.0.0.1:8000:80"`. Esto significa que el puerto 80 del contenedor (donde corre Apache y WordPress) se mapea al puerto 8000 del host. Sólo accesible desde localhost (`127.0.0.1`).

No es necesario "publicar" `db` ya que no necesita ser accedido desde fuera del entorno de Docker, solo por WordPress, que está dentro de la misma red interna (`wordpress`). Esto es más seguro, ya que evita abrir puertos innecesarios que podrían ser atacados desde el exterior.

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).

```
~$ 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 /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
```

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
```

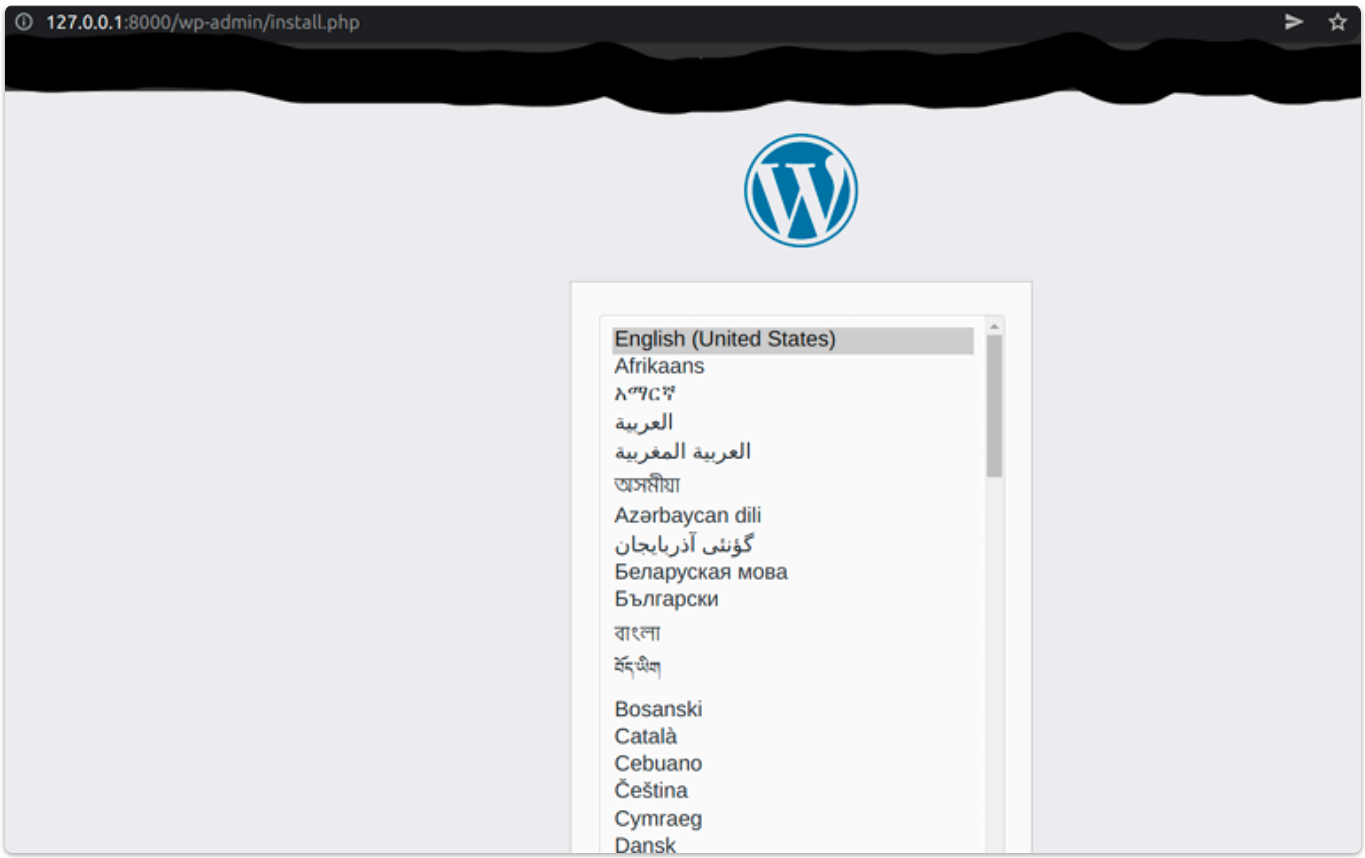
En el host:

```
host:/docker compose-ej-1$ ls
docker-compose.yml test
```

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 ...
```

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.

Proceso de Instanciación

```
so@so:~$ mkdir docker-compose-ej-1
so@so:~$ cd docker-compose-ej-1/
so@so:~/docker-compose-ej-1$ vim docker-compose.yml
so@so:~/docker-compose-ej-1$ docker compose up
WARN[0000] /home/so/docker-compose-ej-1/docker-compose.yml: the attribute
`version` is obsolete, it will be ignored, please remove it to avoid
potential confusion
[+] Running 35/35
 ✓ wordpress Pulled
151.8s
 ✓ db Pulled
151.2s
[+] Running 5/5
 ✓ Network docker-compose-ej-1_wordpress        Created
0.8s
 ✓ Volume "docker-compose-ej-1_wordpress_data"  Created
0.0s
 ✓ Volume "docker-compose-ej-1_db_data"         Created
0.0s
 ✓ Container docker-compose-ej-1-db-1           Cr...
0.5s
 ✓ Container docker-compose-ej-1-wordpress-1    Created
0.4s
Attaching to db-1, wordpress-1
db-1         | 2025-05-22 00:33:48+00:00 [Note] [Entrypoint]: Entrypoint
script for MySQL Server 5.7.44-1.el7 started.
db-1         | 2025-05-22 00:33:50+00:00 [Note] [Entrypoint]: Switching to
```

```
dedicated user 'mysql'
db-1         | 2025-05-22 00:33:50+00:00 [Note] [Entrypoint]: Entrypoint
script for MySQL Server 5.7.44-1.el7 started.
wordpress-1  | WordPress not found in /var/www/html - copying now...
db-1         | 2025-05-22 00:33:52+00:00 [Note] [Entrypoint]: Initializing
database files
db-1         | 2025-05-22T00:33:52.610639Z 0 [Warning] TIMESTAMP with
implicit DEFAULT value is deprecated. Please use --
explicit_defaults_for_timestamp server option (see documentation for more
details).
db-1         | 2025-05-22T00:33:56.303354Z 0 [Warning] InnoDB: New log files
created, LSN=45790
db-1         | 2025-05-22T00:33:56.816671Z 0 [Warning] InnoDB: Creating
foreign key constraint system tables.
db-1         | 2025-05-22T00:33:57.031433Z 0 [Warning] No existing UUID has
been found, so we assume that this is the first time that this server has
been started. Generating a new UUID: 72660c21-36a4-11f0-84ff-7ede76753eba.
db-1         | 2025-05-22T00:33:57.077651Z 0 [Warning] Gtid table is not
ready to be used. Table 'mysql.gtid_executed' cannot be opened.
db-1         | 2025-05-22T00:33:57.781291Z 0 [Warning] A deprecated TLS
version TLSv1 is enabled. Please use TLSv1.2 or higher.
db-1         | 2025-05-22T00:33:57.781492Z 0 [Warning] A deprecated TLS
version TLSv1.1 is enabled. Please use TLSv1.2 or higher.
db-1         | 2025-05-22T00:33:57.782425Z 0 [Warning] CA certificate ca.pem
is self signed.
db-1         | 2025-05-22T00:33:58.253151Z 1 [Warning] root@localhost is
created with an empty password ! Please consider switching off the --
initialize-insecure option.
wordpress-1  | Complete! WordPress has been successfully copied to
/var/www/html
wordpress-1  | No 'wp-config.php' found in /var/www/html, but
'WORDPRESS_...' variables supplied; copying 'wp-config-docker.php'
(WORDPRESS_DB_HOST WORDPRESS_DB_NAME WORDPRESS_DB_PASSWORD
WORDPRESS_DB_USER)
wordpress-1  | AH00558: apache2: Could not reliably determine the server's
fully qualified domain name, using 172.18.0.3. Set the 'ServerName'
directive globally to suppress this message
wordpress-1  | AH00558: apache2: Could not reliably determine the server's
fully qualified domain name, using 172.18.0.3. Set the 'ServerName'
directive globally to suppress this message
wordpress-1  | [Thu May 22 00:34:11.009011 2025] [mpm_prefork:notice] [pid
1:tid 1] AH00163: Apache/2.4.62 (Debian) PHP/8.2.28 configured -- resuming
normal operations
wordpress-1  | [Thu May 22 00:34:11.014885 2025] [core:notice] [pid 1:tid 1]
AH00094: Command line: 'apache2 -D FOREGROUND'
db-1         | 2025-05-22 00:34:14+00:00 [Note] [Entrypoint]: Database files
```

```
initialized
db-1         | 2025-05-22 00:34:14+00:00 [Note] [Entrypoint]: Starting
temporary server
db-1         | 2025-05-22 00:34:14+00:00 [Note] [Entrypoint]: Waiting for
server startup
db-1         | 2025-05-22T00:34:15.958657Z 0 [Warning] TIMESTAMP with
implicit DEFAULT value is deprecated. Please use --
explicit_defaults_for_timestamp server option (see documentation for more
details).
db-1         | 2025-05-22T00:34:15.986159Z 0 [Note] mysqld (mysqld 5.7.44)
starting as process 125 ...
db-1         | 2025-05-22T00:34:16.115473Z 0 [Note] InnoDB: PUNCH HOLE
support available
db-1         | 2025-05-22T00:34:16.115793Z 0 [Note] InnoDB: Mutexes and
rw_locks use GCC atomic builtins
db-1         | 2025-05-22T00:34:16.115835Z 0 [Note] InnoDB: Uses event
mutexes
db-1         | 2025-05-22T00:34:16.115863Z 0 [Note] InnoDB: GCC builtin
__atomic_thread_fence() is used for memory barrier
db-1         | 2025-05-22T00:34:16.115889Z 0 [Note] InnoDB: Compressed
tables use zlib 1.2.13
db-1         | 2025-05-22T00:34:16.115914Z 0 [Note] InnoDB: Using Linux
native AIO
db-1         | 2025-05-22T00:34:16.124202Z 0 [Note] InnoDB: Number of pools:
1
db-1         | 2025-05-22T00:34:16.131307Z 0 [Note] InnoDB: Using CPU crc32
instructions
db-1         | 2025-05-22T00:34:16.243452Z 0 [Note] InnoDB: Initializing
buffer pool, total size = 128M, instances = 1, chunk size = 128M
db-1         | 2025-05-22T00:34:16.385123Z 0 [Note] InnoDB: Completed
initialization of buffer pool
db-1         | 2025-05-22T00:34:16.437327Z 0 [Note] InnoDB: If the mysqld
execution user is authorized, page cleaner thread priority can be changed.
See the man page of setpriority().
db-1         | 2025-05-22T00:34:16.512590Z 0 [Note] InnoDB: Highest
supported file format is Barracuda.
db-1         | 2025-05-22T00:34:16.574385Z 0 [Note] InnoDB: Creating shared
tablespace for temporary tables
db-1         | 2025-05-22T00:34:16.574603Z 0 [Note] InnoDB: Setting file
'./ibtmp1' size to 12 MB. Physically writing the file full; Please wait ...
db-1         | 2025-05-22T00:34:16.821696Z 0 [Note] InnoDB: File './ibtmp1'
size is now 12 MB.
db-1         | 2025-05-22T00:34:16.932222Z 0 [Note] InnoDB: 96 redo rollback
segment(s) found. 96 redo rollback segment(s) are active.
db-1         | 2025-05-22T00:34:16.933246Z 0 [Note] InnoDB: 32 non-redo
rollback segment(s) are active.
```

```
db-1         | 2025-05-22T00:34:16.962334Z 0 [Note] InnoDB: Waiting for
purge to start
db-1         | 2025-05-22T00:34:17.021037Z 0 [Note] InnoDB: 5.7.44 started;
log sequence number 2768943
db-1         | 2025-05-22T00:34:17.063731Z 0 [Note] InnoDB: Loading buffer
pool(s) from /var/lib/mysql/ib_buffer_pool
db-1         | 2025-05-22T00:34:17.066167Z 0 [Note] Plugin 'FEDERATED' is
disabled.
db-1         | 2025-05-22T00:34:17.137010Z 0 [Note] InnoDB: Buffer pool(s)
load completed at 250522  0:34:17
db-1         | 2025-05-22T00:34:17.248881Z 0 [Note] Found ca.pem, server-
cert.pem and server-key.pem in data directory. Trying to enable SSL support
using them.
db-1         | 2025-05-22T00:34:17.249031Z 0 [Note] Skipping generation of
SSL certificates as certificate files are present in data directory.
db-1         | 2025-05-22T00:34:17.249043Z 0 [Warning] A deprecated TLS
version TLSv1 is enabled. Please use TLSv1.2 or higher.
db-1         | 2025-05-22T00:34:17.249046Z 0 [Warning] A deprecated TLS
version TLSv1.1 is enabled. Please use TLSv1.2 or higher.
db-1         | 2025-05-22T00:34:17.250733Z 0 [Warning] CA certificate ca.pem
is self signed.
db-1         | 2025-05-22T00:34:17.250923Z 0 [Note] Skipping generation of
RSA key pair as key files are present in data directory.
db-1         | 2025-05-22T00:34:17.281263Z 0 [Warning] Insecure
configuration for --pid-file: Location '/var/run/mysqld' in the path is
accessible to all OS users. Consider choosing a different directory.
db-1         | 2025-05-22T00:34:17.358454Z 0 [Note] Event Scheduler: Loaded
0 events
db-1         | 2025-05-22T00:34:17.361899Z 0 [Note] mysqld: ready for
connections.
db-1         | Version: '5.7.44'  socket: '/var/run/mysqld/mysqld.sock'
port: 0  MySQL Community Server (GPL)
db-1         | 2025-05-22 00:34:17+00:00 [Note] [Entrypoint]: Temporary
server started.
db-1         | '/var/lib/mysql/mysql.sock' -> '/var/run/mysqld/mysqld.sock'
db-1         | 2025-05-22T00:34:17.906403Z 3 [Note] InnoDB: Stopping purge
db-1         | 2025-05-22T00:34:17.976363Z 3 [Note] InnoDB: Resuming purge
db-1         | 2025-05-22T00:34:17.998858Z 3 [Note] InnoDB: Stopping purge
db-1         | 2025-05-22T00:34:18.030609Z 3 [Note] InnoDB: Resuming purge
db-1         | 2025-05-22T00:34:18.049391Z 3 [Note] InnoDB: Stopping purge
db-1         | 2025-05-22T00:34:18.093619Z 3 [Note] InnoDB: Resuming purge
db-1         | 2025-05-22T00:34:18.114356Z 3 [Note] InnoDB: Stopping purge
db-1         | 2025-05-22T00:34:18.159859Z 3 [Note] InnoDB: Resuming purge
db-1         | Warning: Unable to load '/usr/share/zoneinfo/iso3166.tab' as
time zone. Skipping it.
db-1         | Warning: Unable to load '/usr/share/zoneinfo/leapseconds' as
```

```
time zone. Skipping it.
db-1         | Warning: Unable to load '/usr/share/zoneinfo/tzdata.zi' as
time zone. Skipping it.
db-1         | Warning: Unable to load '/usr/share/zoneinfo/zone.tab' as
time zone. Skipping it.
db-1         | Warning: Unable to load '/usr/share/zoneinfo/zone1970.tab' as
time zone. Skipping it.
db-1         | 2025-05-22 00:34:25+00:00 [Note] [Entrypoint]: Creating
database wordpress
db-1         | 2025-05-22 00:34:25+00:00 [Note] [Entrypoint]: Creating user
wordpress
db-1         | 2025-05-22 00:34:25+00:00 [Note] [Entrypoint]: Giving user
wordpress access to schema wordpress
db-1         |
db-1         | 2025-05-22 00:34:25+00:00 [Note] [Entrypoint]: Stopping
temporary server
db-1         | 2025-05-22T00:34:26.043695Z 0 [Note] Giving 0 client threads
a chance to die gracefully
db-1         | 2025-05-22T00:34:26.043932Z 0 [Note] Shutting down slave
threads
db-1         | 2025-05-22T00:34:26.043944Z 0 [Note] Forcefully disconnecting
0 remaining clients
db-1         | 2025-05-22T00:34:26.043953Z 0 [Note] Event Scheduler: Purging
the queue. 0 events
db-1         | 2025-05-22T00:34:26.048903Z 0 [Note] Binlog end
db-1         | 2025-05-22T00:34:26.049646Z 0 [Note] Shutting down plugin
'ngram'
db-1         | 2025-05-22T00:34:26.049661Z 0 [Note] Shutting down plugin
'partition'
db-1         | 2025-05-22T00:34:26.049664Z 0 [Note] Shutting down plugin
'BLACKHOLE'
db-1         | 2025-05-22T00:34:26.049668Z 0 [Note] Shutting down plugin
'ARCHIVE'
db-1         | 2025-05-22T00:34:26.049670Z 0 [Note] Shutting down plugin
'PERFORMANCE_SCHEMA'
db-1         | 2025-05-22T00:34:26.049699Z 0 [Note] Shutting down plugin
'MRG_MYISAM'
db-1         | 2025-05-22T00:34:26.049702Z 0 [Note] Shutting down plugin
'MyISAM'
db-1         | 2025-05-22T00:34:26.049712Z 0 [Note] Shutting down plugin
'INNODB_SYS_VIRTUAL'
db-1         | 2025-05-22T00:34:26.049715Z 0 [Note] Shutting down plugin
'INNODB_SYS_DATAFILES'
db-1         | 2025-05-22T00:34:26.049717Z 0 [Note] Shutting down plugin
'INNODB_SYS_TABLESPACES'
db-1         | 2025-05-22T00:34:26.049719Z 0 [Note] Shutting down plugin
```

```
'INNODB_SYS_FOREIGN_COLS'
db-1         | 2025-05-22T00:34:26.049721Z 0 [Note] Shutting down plugin
'INNODB_SYS_FOREIGN'
db-1         | 2025-05-22T00:34:26.049723Z 0 [Note] Shutting down plugin
'INNODB_SYS_FIELDS'
db-1         | 2025-05-22T00:34:26.049725Z 0 [Note] Shutting down plugin
'INNODB_SYS_COLUMNS'
db-1         | 2025-05-22T00:34:26.049727Z 0 [Note] Shutting down plugin
'INNODB_SYS_INDEXES'
db-1         | 2025-05-22T00:34:26.049729Z 0 [Note] Shutting down plugin
'INNODB_SYS_TABLESTATS'
db-1         | 2025-05-22T00:34:26.049731Z 0 [Note] Shutting down plugin
'INNODB_SYS_TABLES'
db-1         | 2025-05-22T00:34:26.049733Z 0 [Note] Shutting down plugin
'INNODB_FT_INDEX_TABLE'
db-1         | 2025-05-22T00:34:26.049735Z 0 [Note] Shutting down plugin
'INNODB_FT_INDEX_CACHE'
db-1         | 2025-05-22T00:34:26.049737Z 0 [Note] Shutting down plugin
'INNODB_FT_CONFIG'
db-1         | 2025-05-22T00:34:26.049739Z 0 [Note] Shutting down plugin
'INNODB_FT_BEING_DELETED'
db-1         | 2025-05-22T00:34:26.049741Z 0 [Note] Shutting down plugin
'INNODB_FT_DELETED'
db-1         | 2025-05-22T00:34:26.049743Z 0 [Note] Shutting down plugin
'INNODB_FT_DEFAULT_STOPWORD'
db-1         | 2025-05-22T00:34:26.049745Z 0 [Note] Shutting down plugin
'INNODB_METRICS'
db-1         | 2025-05-22T00:34:26.049747Z 0 [Note] Shutting down plugin
'INNODB_TEMP_TABLE_INFO'
db-1         | 2025-05-22T00:34:26.049749Z 0 [Note] Shutting down plugin
'INNODB_BUFFER_POOL_STATS'
db-1         | 2025-05-22T00:34:26.049753Z 0 [Note] Shutting down plugin
'INNODB_BUFFER_PAGE_LRU'
db-1         | 2025-05-22T00:34:26.049756Z 0 [Note] Shutting down plugin
'INNODB_BUFFER_PAGE'
db-1         | 2025-05-22T00:34:26.049758Z 0 [Note] Shutting down plugin
'INNODB_CMP_PER_INDEX_RESET'
db-1         | 2025-05-22T00:34:26.049760Z 0 [Note] Shutting down plugin
'INNODB_CMP_PER_INDEX'
db-1         | 2025-05-22T00:34:26.049762Z 0 [Note] Shutting down plugin
'INNODB_CMPMEM_RESET'
db-1         | 2025-05-22T00:34:26.049764Z 0 [Note] Shutting down plugin
'INNODB_CMPMEM'
db-1         | 2025-05-22T00:34:26.049767Z 0 [Note] Shutting down plugin
'INNODB_CMP_RESET'
db-1         | 2025-05-22T00:34:26.049769Z 0 [Note] Shutting down plugin
```

```
'INNODB_CMP'
db-1         | 2025-05-22T00:34:26.049772Z 0 [Note] Shutting down plugin
'INNODB_LOCK_WAITS'
db-1         | 2025-05-22T00:34:26.049774Z 0 [Note] Shutting down plugin
'INNODB_LOCKS'
db-1         | 2025-05-22T00:34:26.049777Z 0 [Note] Shutting down plugin
'INNODB_TRX'
db-1         | 2025-05-22T00:34:26.049779Z 0 [Note] Shutting down plugin
'InnoDB'
db-1         | 2025-05-22T00:34:26.050586Z 0 [Note] InnoDB: FTS optimize
thread exiting.
db-1         | 2025-05-22T00:34:26.052063Z 0 [Note] InnoDB: Starting
shutdown...
db-1         | 2025-05-22T00:34:26.153752Z 0 [Note] InnoDB: Dumping buffer
pool(s) to /var/lib/mysql/ib_buffer_pool
db-1         | 2025-05-22T00:34:26.155501Z 0 [Note] InnoDB: Buffer pool(s)
dump completed at 250522  0:34:26
db-1         | 2025-05-22T00:34:28.022577Z 0 [Note] InnoDB: Shutdown
completed; log sequence number 12220055
db-1         | 2025-05-22T00:34:28.025758Z 0 [Note] InnoDB: Removed
temporary tablespace data file: "ibtmp1"
db-1         | 2025-05-22T00:34:28.027514Z 0 [Note] Shutting down plugin
'MEMORY'
db-1         | 2025-05-22T00:34:28.027574Z 0 [Note] Shutting down plugin
'CSV'
db-1         | 2025-05-22T00:34:28.027595Z 0 [Note] Shutting down plugin
'sha256_password'
db-1         | 2025-05-22T00:34:28.027610Z 0 [Note] Shutting down plugin
'mysql_native_password'
db-1         | 2025-05-22T00:34:28.028074Z 0 [Note] Shutting down plugin
'binlog'
db-1         | 2025-05-22T00:34:28.034131Z 0 [Note] mysqld: Shutdown
complete
db-1         |
db-1         | 2025-05-22 00:34:28+00:00 [Note] [Entrypoint]: Temporary
server stopped
db-1         |
db-1         | 2025-05-22 00:34:28+00:00 [Note] [Entrypoint]: MySQL init
process done. Ready for start up.
db-1         |
db-1         | 2025-05-22T00:34:28.433951Z 0 [Warning] TIMESTAMP with
implicit DEFAULT value is deprecated. Please use --
explicit_defaults_for_timestamp server option (see documentation for more
details).
db-1         | 2025-05-22T00:34:28.435065Z 0 [Note] mysqld (mysqld 5.7.44)
starting as process 1 ...
```

```
db-1         | 2025-05-22T00:34:28.453935Z 0 [Note] InnoDB: PUNCH HOLE
support available
db-1         | 2025-05-22T00:34:28.454135Z 0 [Note] InnoDB: Mutexes and
rw_locks use GCC atomic builtins
db-1         | 2025-05-22T00:34:28.454149Z 0 [Note] InnoDB: Uses event
mutexes
db-1         | 2025-05-22T00:34:28.454153Z 0 [Note] InnoDB: GCC builtin
__atomic_thread_fence() is used for memory barrier
db-1         | 2025-05-22T00:34:28.454157Z 0 [Note] InnoDB: Compressed
tables use zlib 1.2.13
db-1         | 2025-05-22T00:34:28.454160Z 0 [Note] InnoDB: Using Linux
native AIO
db-1         | 2025-05-22T00:34:28.455938Z 0 [Note] InnoDB: Number of pools:
1
db-1         | 2025-05-22T00:34:28.459809Z 0 [Note] InnoDB: Using CPU crc32
instructions
db-1         | 2025-05-22T00:34:28.482899Z 0 [Note] InnoDB: Initializing
buffer pool, total size = 128M, instances = 1, chunk size = 128M
db-1         | 2025-05-22T00:34:28.503257Z 0 [Note] InnoDB: Completed
initialization of buffer pool
db-1         | 2025-05-22T00:34:28.522658Z 0 [Note] InnoDB: If the mysqld
execution user is authorized, page cleaner thread priority can be changed.
See the man page of setpriority().
db-1         | 2025-05-22T00:34:28.543608Z 0 [Note] InnoDB: Highest
supported file format is Barracuda.
db-1         | 2025-05-22T00:34:28.615922Z 0 [Note] InnoDB: Creating shared
tablespace for temporary tables
db-1         | 2025-05-22T00:34:28.620415Z 0 [Note] InnoDB: Setting file
'./ibtmp1' size to 12 MB. Physically writing the file full; Please wait ...
db-1         | 2025-05-22T00:34:28.744721Z 0 [Note] InnoDB: File './ibtmp1'
size is now 12 MB.
db-1         | 2025-05-22T00:34:28.745934Z 0 [Note] InnoDB: 96 redo rollback
segment(s) found. 96 redo rollback segment(s) are active.
db-1         | 2025-05-22T00:34:28.745950Z 0 [Note] InnoDB: 32 non-redo
rollback segment(s) are active.
db-1         | 2025-05-22T00:34:28.746970Z 0 [Note] InnoDB: Waiting for
purge to start
db-1         | 2025-05-22T00:34:28.826401Z 0 [Note] InnoDB: 5.7.44 started;
log sequence number 12220055
db-1         | 2025-05-22T00:34:28.829109Z 0 [Note] InnoDB: Loading buffer
pool(s) from /var/lib/mysql/ib_buffer_pool
db-1         | 2025-05-22T00:34:28.832750Z 0 [Note] InnoDB: Buffer pool(s)
load completed at 250522  0:34:28
db-1         | 2025-05-22T00:34:28.834524Z 0 [Note] Plugin 'FEDERATED' is
disabled.
db-1         | 2025-05-22T00:34:28.845430Z 0 [Note] Found ca.pem, server-
```

```
cert.pem and server-key.pem in data directory. Trying to enable SSL support
using them.
db-1         | 2025-05-22T00:34:28.850038Z 0 [Note] Skipping generation of
SSL certificates as certificate files are present in data directory.
db-1         | 2025-05-22T00:34:28.850187Z 0 [Warning] A deprecated TLS
version TLSv1 is enabled. Please use TLSv1.2 or higher.
db-1         | 2025-05-22T00:34:28.850221Z 0 [Warning] A deprecated TLS
version TLSv1.1 is enabled. Please use TLSv1.2 or higher.
db-1         | 2025-05-22T00:34:28.851565Z 0 [Warning] CA certificate ca.pem
is self signed.
db-1         | 2025-05-22T00:34:28.852098Z 0 [Note] Skipping generation of
RSA key pair as key files are present in data directory.
db-1         | 2025-05-22T00:34:28.866425Z 0 [Note] Server hostname (bind-
address): '*'; port: 3306
db-1         | 2025-05-22T00:34:28.866543Z 0 [Note] IPv6 is available.
db-1         | 2025-05-22T00:34:28.866566Z 0 [Note]   - '::' resolves to
'::';
db-1         | 2025-05-22T00:34:28.866593Z 0 [Note] Server socket created on
IP: '::'.
db-1         | 2025-05-22T00:34:29.028507Z 0 [Warning] Insecure
configuration for --pid-file: Location '/var/run/mysqld' in the path is
accessible to all OS users. Consider choosing a different directory.
db-1         | 2025-05-22T00:34:29.049914Z 0 [Note] Event Scheduler: Loaded
0 events
db-1         | 2025-05-22T00:34:29.050995Z 0 [Note] mysqld: ready for
connections.
db-1         | Version: '5.7.44'  socket: '/var/run/mysqld/mysqld.sock'
port: 3306  MySQL Community Server (GPL)
Gracefully stopping... (press Ctrl+C again to force)
[+] Stopping 2/2
 ✓ Container docker-compose-ej-1-wordpress-1  Stopped
1.3s
 ✓ Container docker-compose-ej-1-db-1         Stop...
0.2s

so@so:~/docker-compose-ej-1$ docker compose up -d
WARN[0000] /home/so/docker-compose-ej-1/docker-compose.yml: the attribute
`version` is obsolete, it will be ignored, please remove it to avoid
potential confusion
[+] Running 2/2
 ✓ Container docker-compose-ej-1-db-1         Star...
2.0s
 ✓ Container docker-compose-ej-1-wordpress-1  Started
4.2s
so@so:~/docker-compose-ej-1$ docker compose exec wordpress /bin/bash
WARN[0000] /home/so/docker-compose-ej-1/docker-compose.yml: the attribute
```

```
`version` is obsolete, it will be ignored, please remove it to avoid
potential confusion
root@7a745aa44e3c:/var/www/html# cd /data/
root@7a745aa44e3c:/data# ls
docker-compose.yml
root@7a745aa44e3c:/data# touch test
root@7a745aa44e3c:/data# ls
docker-compose.yml  test
root@7a745aa44e3c:/data# exit
exit
so@so:~/docker-compose-ej-1$ docker compose ps
WARN[0000] /home/so/docker-compose-ej-1/docker-compose.yml: the attribute
`version` is obsolete, it will be ignored, please remove it to avoid
potential confusion
NAME                                IMAGE              COMMAND
SERVICE     CREATED         STATUS         PORTS
docker-compose-ej-1-db-1            mysql:5.7          "docker-entrypoint.s…"
db          4 minutes ago   Up 2 minutes   3306/tcp, 33060/tcp
docker-compose-ej-1-wordpress-1     wordpress:latest   "docker-entrypoint.s…"
wordpress   4 minutes ago   Up 2 minutes   127.0.0.1:8000→80/tcp
so@so:~/docker-compose-ej-1$ docker compose stop
WARN[0000] /home/so/docker-compose-ej-1/docker-compose.yml: the attribute
`version` is obsolete, it will be ignored, please remove it to avoid
potential confusion
[+] Stopping 2/2
 ✓ Container docker-compose-ej-1-wordpress-1  Stopped
1.8s
 ✓ Container docker-compose-ej-1-db-1         Stopped
2.5s
so@so:~/docker-compose-ej-1$ docker compose down
WARN[0000] /home/so/docker-compose-ej-1/docker-compose.yml: the attribute
`version` is obsolete, it will be ignored, please remove it to avoid
potential confusion
[+] Running 3/3
 ✓ Container docker-compose-ej-1-wordpress-1  Removed
0.0s
 ✓ Container docker-compose-ej-1-db-1         Removed
0.0s
 ✓ Network docker-compose-ej-1_wordpress      Removed
0.4s
 so@so:~/docker-compose-ej-1$ docker compose down -v
WARN[0000] /home/so/docker-compose-ej-1/docker-compose.yml: the attribute
`version` is obsolete, it will be ignored, please remove it to avoid
potential confusion
[+] Running 2/2
 ✓ Volume docker-compose-ej-1_wordpress_data  Removed
```

```
0.1s
 ✓ Volume docker-compose-ej-1_db_data         Removed
0.2s
```

En la terminal del Host

```
so@so:/$ ls /home/so/docker-compose-ej-1/
docker-compose.yml  test
```

Vista en el navegador