# Sistemas Operativos
## 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.
2. ¿Qué es una imagen? ¿Y un contenedor? ¿Cuál es la principal diferencia entre ambos?
3. ¿Qué es Union Filesystem? ¿Cómo lo utiliza Docker?
4. ¿Qué rango de direcciones IP utilizan los contenedores cuando se crean? ¿De dónde la obtiene?
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?

**Taller:**

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

1. Instale Docker CE (Community Edition) en su sistema operativo. Ayuda: seguir las instrucciones de la página de Docker. La instalación más simple para distribuciones de GNU/Linux basadas en Debian es usando los repositorios.
2. Usando las herramientas (comandos) provistas por Docker realice las siguientes tareas:
    a. Obtener una imagen de la última versión de Ubuntu disponible. ¿Cuál es el tamaño en disco de la imagen obtenida? ¿Ya puede ser considerada un contenedor? ¿Qué significa lo siguiente: *Using default tag: latest*?
    b. De la imagen obtenida en el punto anterior iniciar un contenedor que simplemente ejecute el comando ls -l.
    c. ¿Qué sucede si ejecuta el comando docker [container] run ubuntu /bin/bash[1]? ¿Puede utilizar la shell Bash del contenedor?
        i. Modifique el comando utilizado para que el contenedor se inicie con una terminal interactiva y ejecutarlo. ¿Ahora puede utilizar la shell Bash del contenedor? ¿Por qué?
        ii. ¿Cuál es el PID del proceso bash en el contenedor? ¿Y fuera de éste?
        iii. Ejecutar el comando lsns. ¿Qué puede decir de los namespace?
        iv. Dentro del contenedor cree un archivo con nombre sistemas-operativos en el directorio raíz del filesystem y luego salga del contenedor (finalice la sesión de Bash utilizando las teclas Ctrl + D o el comando exit).
        v. Corrobore si el archivo creado existe en el directorio raíz del sistema operativo anfitrión (host). ¿Existe? ¿Por qué?
    d. Vuelva a iniciar el contenedor anterior utilizando el mismo comando (con una terminal interactiva). ¿Existe el archivo creado en el contenedor? ¿Por qué?

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

e. Obtenga el identificador del contenedor (container_id) donde se creó el archivo y utilícelo para iniciar con el comando *docker start -ia container_id* el contenedor en el cual se creó el archivo.
   i. ¿Cómo obtuvo el container_id para para este comando?
   ii. Chequee nuevamente si el archivo creado anteriormente existe. ¿Cuál es el resultado en este caso? ¿Puede encontrar el archivo creado?
f. ¿Cuántos contenedores están actualmente en ejecución? ¿En qué estado se encuentra cada uno de los que se han ejecutado hasta el momento?
g. Elimine todos los contenedores creados hasta el momento. Indique el o los comandos utilizados.

3. Creación de una imagen a partir de un contenedor. Siguiendo los pasos indicados a continuación genere una imagen de Docker a partir de un contenedor:
   a. Inicie un contenedor a partir de la imagen de Ubuntu descargada anteriormente ejecutando una consola interactiva de Bash.
   b. Instale el servidor web Nginx, https://nginx.org/en/, en el contenedor utilizando los siguientes comandos[2]:

```
export DEBIAN_FRONTEND=noninteractive
export TZ=America/Buenos_Aires
apt update -qq
apt install -y --no-install-recommends nginx
```

   c. Salga del contenedor y genere una imagen Docker a partir de éste. ¿Con qué nombre se genera si no se especifica uno?
   d. Cambie el nombre de la imagen creada de manera que en la columna Repository aparezca nginx-so y en la columna Tag aparezca v1.
   e. Ejecute un contenedor a partir de la imagen nginx-so:v1 que corra el servidor web nginx atendiendo conexiones en el puerto 8080 del host, y sirviendo una página web para corroborar su correcto funcionamiento. Para esto:
      I. En el Sistema Operativo anfitrión (host) sobre el cual se ejecuta Docker crear un directorio que se utilizará para este taller. Éste puede ser el directorio nginx-so dentro de su directorio personal o cualquier otro directorio - para los fines de este enunciado haremos referencia a éste como /home/so/nginx-so, por lo que en los lugares donde se mencione esta ruta usted deberá reemplazarla por la ruta absoluta al directorio que haya decidido crear en este paso.
      II. Dentro de ese directorio, cree un archivo llamado index.html que contenga el código HTML de este gist de GitHub: https://gist.github.com/ncuesta/5b959fce1c7d2ed4e5a06e84e5a7efc8.
      III. Cree un contenedor a partir de la imagen nginx-so:v1 montando el directorio del host (/home/so/nginx-so) sobre el directorio /var/www/html del contenedor, mapeando el puerto 80 del contenedor al puerto 8080 del host, y ejecutando el servidor nginx en primer plano[3]. Indique el comando utilizado.
   f. Verifique que el contenedor esté ejecutándose correctamente abriendo un navegador web y visitando la URL http://localhost:8080.

[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

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

g. Modifique el archivo index.html agregándole un párrafo con su nombre y número de alumno. ¿Es necesario reiniciar el contenedor para ver los cambios?
h. Analice: ¿por qué es necesario que el proceso nginx se ejecute en primer plano? ¿Qué ocurre si lo ejecuta sin -g 'daemon off;'?

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

a. En el directorio del host creado en el punto anterior (/home/so/nginx-so), cree un archivo Dockerfile que realice los siguientes pasos:
   i. Comenzar en base a la imagen oficial de Ubuntu.
   ii. Exponer el puerto 80 del contenedor.
   iii. Instalar el servidor web nginx.
   iv. Copiar el archivo index.html del mismo directorio del host al directorio /var/www/html de la imagen.
   v. Indicar el comando que se utilizará cuando se inicie un contenedor a partir de esta imagen para ejecutar el servidor nginx en primer plano: nginx -g 'daemon off;'. Use la forma exec[4] para definir el comando, de manera que todas las señales que reciba el contenedor sean enviadas directamente al proceso de nginx.
   Ayuda: las instrucciones necesarias para definir los pasos en el Dockerfile son FROM, EXPOSE, RUN, COPY y CMD.
b. Utilizando el Dockerfile que generó en el punto anterior construya una nueva imagen Docker guardándola localmente con el nombre nginx-so:v2.
c. Ejecute un contenedor a partir de la nueva imagen creada con las opciones adecuadas para que pueda acceder desde su navegador web ala página a través del puerto 8090 del host. Verifique que puede visualizar correctamente la página accediendo a http://localhost:8090.
d. Modifique el archivo index.html del host agregando un párrafo con la fecha actual y recargue la página en su navegador web. ¿Se ven reflejados los cambios que hizo en el archivo? ¿Por qué?
e. Termine el contenedor iniciado antes y cree uno nuevo utilizando el mismo comando. Recargue la página en su navegador web. ¿Se ven ahora reflejados los cambios realizados en el archivo HTML? ¿Por qué?
f. Vuelva a construir una imagen Docker a partir del Dockerfile creado anteriormente, pero esta vez dándole el nombre nginx-so:v3. Cree un contenedor a partir de ésta y acceda a la página en su navegador web. ¿Se ven reflejados los cambios realizados en el archivo HTML? ¿Por qué?

# Docker Compose

1. Utilizando sus palabras describa, ¿qué es docker compose?
2. ¿Qué es el archivo compose y cual es su función? ¿Cuál es el “lenguaje” del archivo?
3. ¿Cuáles son las versiones existentes del archivo docker-compose.yaml existentes y qué características aporta cada una? ¿Son compatibles entre sí?¿Por qué?
4. Investigue y describa la estructura de un archivo compose.
   Desarrolle al menos sobre los siguientes bloques indicando para qué se usan:

[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.

   a. `services`
   b. `build`
   c. `image`
   d. `volumes`
   e. `restart`
   f. `depends_on`
   g. `environment`
   h. `ports`
   i. `expose`
   j. `networks`
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?
6. Indique qué hacen y cuáles son las diferencias entre los siguientes comandos:
   a. `docker compose create` y `docker compose up`
   b. `docker compose stop` y `docker compose down`
   c. `docker compose run` y `docker compose exec`
   d. `docker compose ps`
   e. `docker compose logs`
7. ¿Qué tipo de volúmenes puede utilizar con docker compose? ¿Cómo se declara cada tipo en el archivo compose?
8. ¿Qué sucede si en lugar de usar el comando “`docker compose down`” utilizo “`docker compose down -v/--volumes`”?

## Instalación de docker compose

En la práctica anterior se instaló el entorno de ejecución Docker-CE. Es requisito para esta práctica tener dicho entorno instalado y funcionando en el dispositivo donde se pretenda realizar la misma.

En el sitio https://docs.docker.com/compose/install/ se puede encontrar la guía para instalar docker-compose en distintos SO.

Docker-compose es simplemente un binario, por lo que lo único que se necesita es descargar el binario, ubicarlo en algún lugar que el PATH de nuestro dispositivo pueda encontrarlo y que tenga los permisos necesarios para ser ejecutado.

En la actualidad existen 2 versiones del binario docker-compose. **Vamos a utilizar la versión 2**. Para instalar la versión 2.18.1, vamos a descargarla y ubicarla en el directorio /usr/local/bin/docker-compose, para que de esta manera quede accesible mediante el PATH de nuestra CLI:

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

Una vez descargado, le damos permiso de ejecución:

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

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

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

## Ejercicio guiado - Instanciando un Wordpress y una Base de Datos.

Dado el siguiente código de archivo compose:

```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?

```yaml
depends_on:
  - db
```

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

- ¿Por que uso el volumen nombrado
  ```
  volumes:
        - db_data:/var/lib/mysql
  ```
  para el servicio `db` en lugar de dejar que se instancie un volumen anónimo con el contenedor?
- ¿Qué genera la línea
  ```
  volumes:
    - ${PWD}:/data
  ```
  en la definición de `wordpress`?
- ¿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?

### 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 directorio `/data`, veremos dentro el contenido de nuestro directorio "`docker-compose-ej-1`" (solo tenemos el archivo `docker-compose.yml`) ya que montamos ese directorio como un volumen:

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

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
```

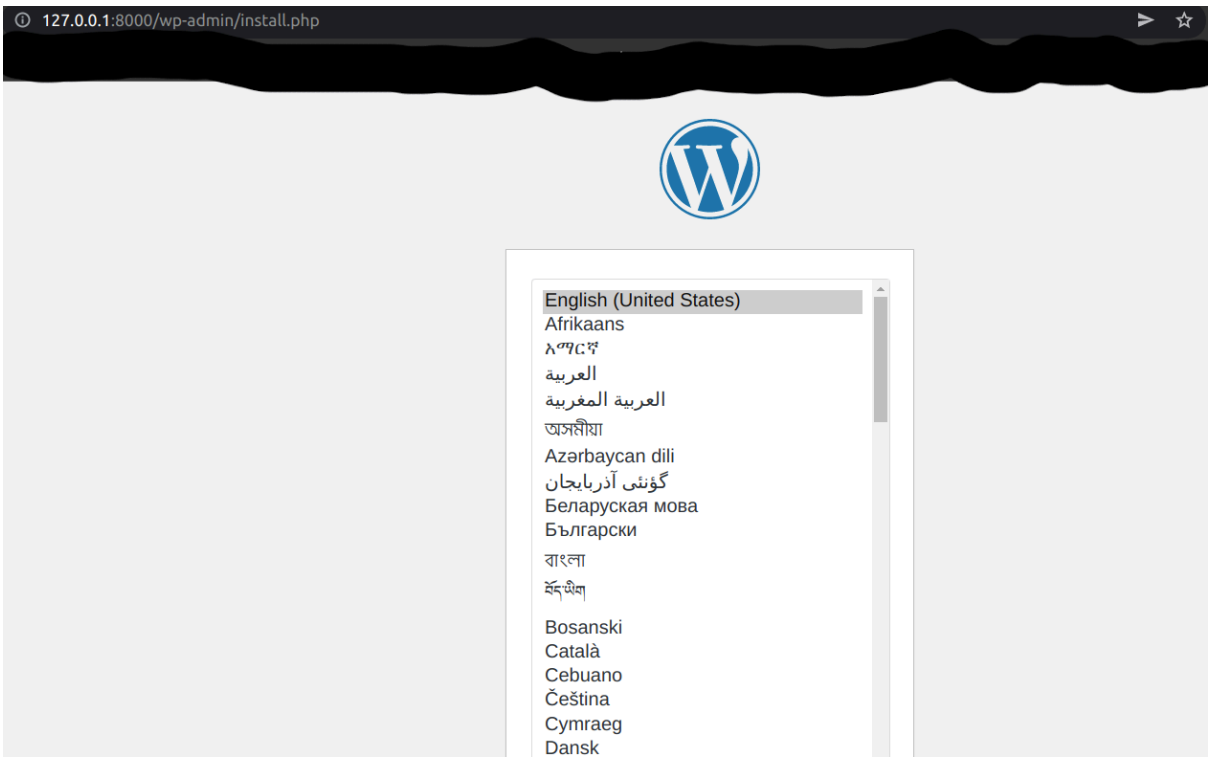
En el host:

```
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.