Sistemas Operativos
Práctica 5
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:
   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:

```
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
  ```
  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:

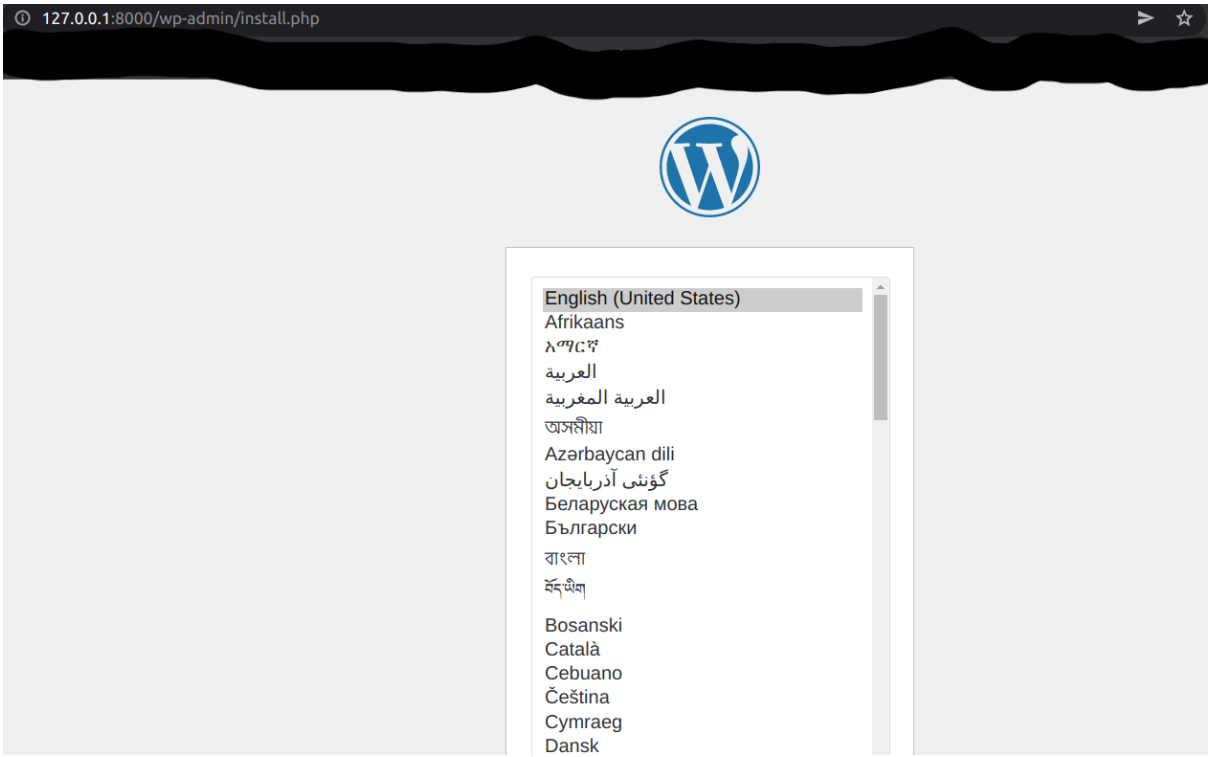
```
host:/docker compose-ej-1$ ls
docker-compose.yml  test
```

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.