Práctica Docker Compose

1. Objetivo

Comprender los mecanismos para correr aplicaciones Docker multi-container

2. Herramientas a Utilizar

- Vagrant
- Docker Community Edition
- Docker Compose
- Vagrant Box Ubuntu

3. Desarrollo de la Practica

PARTE 1. Configuración de Vagrant

Esta práctica la desarrollaremos usando un Box de Ubuntu 22.04 en Vagrant. El Vagrantfile que usaremos es el siguiente (con el que venimos trabajando):

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

Vagrant.configure("2") do |config|


  if Vagrant.has_plugin? "vagrant-vbguest"
    config.vbguest.no_install  = true
    config.vbguest.auto_update = false
    config.vbguest.no_remote   = true
  end


  config.vm.define :clienteUbuntu do |clienteUbuntu|
    clienteUbuntu.vm.box = "bento/ubuntu-22.04"
    clienteUbuntu.vm.network :private_network, ip: "192.168.100.2"
    clienteUbuntu.vm.hostname = "clienteUbuntu"
  end

  config.vm.define :servidorUbuntu do |servidorUbuntu|
    servidorUbuntu.vm.box = "bento/ubuntu-22.04"
    servidorUbuntu.vm.network :private_network, ip: "192.168.100.3"
    servidorUbuntu.vm.hostname = "servidorUbuntu"
  end
end
```

PARTE 2. Instalación de Docker Compose en Ubuntu

```
$ sudo apt-get update
$ sudo apt-get install docker-compose-plugin
```

Verificar la versión de docker compose instalada:

```
$ docker compose version
Docker Compose version v2.12.2
```

PARTE 3. Crear archivo docker compose

Modificar (Crear) el archivo ~/.vimrc para trabajar con Yaml.

```
$ vim ~/.vimrc
```

Agregar la siguiente configuración, con el fin de trabajar adecuadamente con tabs y espacios en los archivos yaml.

```
" Configuracion para trabajar con archivos yaml
```

```
au! BufNewFile,BufReadPost *.{yaml,yml} set filetype=yaml foldmethod=indent
autocmd FileType yaml setlocal ts=2 sts=2 sw=2 expandtab
```

Una vez configurado vim se puede crear el archivo .yml. En esta parte se creara un archivo docker-compose.yml en algún lugar de su sistema.

```
vagrant@machine1:~$ mkdir dockerComposeTest
vagrant@machine1:~$ cd dockerComposeTest/
vagrant@machine1:~/dockerComposeTest$ touch docker-compose.yml
vagrant@machine1:~/dockerComposeTest$ ls
docker-compose.yml
vagrant@machine1:~/dockerComposeTest$ vim docker-compose.yml
```

Escribir (manualmente) lo siguiente en docker compose.yml

```
services:

  web:
    image: nginx

  database:
    image: redis
```

En este archivo incluimos dos contenedores desde imágenes existentes en docker hub, una imagen del servidor proxy de reversa nginx (https://hub.docker.com/_/nginx) y otra con el data store redis (https://hub.docker.com/_/redis).

Verificar la validez del archivo docker-compose.yml creado usando el comando docker compose config

```
$ docker compose config
name: dockercomposetest
services:
  database:
    image: redis
    networks:
      default: null
  web:
    image: nginx
    networks:
      default: null
networks:
  default:
    name: dockercomposetest_default
```

PARTE 4. Correr docker-compose

```
vagrant@machine1:~/dockerComposeTest$ sudo docker compose up -d

[+] Running 13/13
```

```
 ⠿ web Pulled
17.3s
   ⠿ 71689475aec2 Pull complete
12.7s
   ⠿ f88a23025338 Pull complete
12.8s
   ⠿ 0df440342e26 Pull complete
12.9s
   ⠿ eef26ceb3309 Pull complete
13.0s
   ⠿ 8e3ed6a9e43a Pull complete
13.1s
 ⠿ database Pulled
16.9s
   ⠿ e9995326b091 Pull complete
9.2s
   ⠿ f2cd78d6f24c Pull complete
9.4s
   ⠿ 8f3614d34c89 Pull complete
10.1s
   ⠿ 697fd51ec515 Pull complete
12.3s
   ⠿ a554cf50a327 Pull complete
13.0s
   ⠿ 66f93c02e79c Pull complete
13.2s
[+] Running 3/3
 ⠿ Network dockercomposetest_default       Created
0.3s
 ⠿ Container dockercomposetest-web-1       Started
2.7s
 ⠿ Container dockercomposetest-database-1  Started
2.7s
```

Verificar que estén corriendo los contenedores creados con Docker ps:

```
vagrant@machine1:~/dockerComposeTest$ sudo docker ps

CONTAINER ID   IMAGE     COMMAND                  CREATED              STATUS
PORTS      NAMES
4368860b0389   nginx     "/docker-entrypoint.…"   About a minute ago   Up About
a minute   80/tcp     dockercomposetest-web-1
1309edb4b855   redis     "docker-entrypoint.s…"   About a minute ago   Up About
a minute   6379/tcp   dockercomposetest-database-1
```

o con docker compose ps:

```
$ sudo docker compose ps

NAME                           COMMAND                  SERVICE             STATUS
PORTS
dockercomposetest-database-1   "docker-entrypoint.s…"   database            running
6379/tcp
dockercomposetest-web-1        "/docker-entrypoint.…"   web                 running
80/tcp
```

Con docker compose down se pueden detener los contenedores:

```
$ sudo docker compose down
[+] Running 3/2
 ⠿ Container dockercomposetest-database-1  Removed
0.8s
 ⠿ Container dockercomposetest-web-1       Removed
0.6s
 ⠿ Network dockercomposetest_default       Removed
0.0s
```

Con Docker compose ps se puede ver que todo está detenido

```
$ sudo docker compose ps
NAME                COMMAND             SERVICE             STATUS
PORTS
```

PARTE 5. Completar el docker-compose file

Ir a https://hub.docker.com/_/nginx y ver que otras opciones se pueden usar para completar el archivo .yml

En este caso agregaremos redirección de puertos desde el puerto 80 del contenedor al 9090 del box vagrant.

```
services:
  web:
    image: nginx
    ports:
    - 9090:80/tcp
  database:
    image: redis
```

Verificamos con Docker compose config

```
$ docker compose config
name: dockercomposetest
services:
  database:
    image: redis
    networks:
      default: null
  web:
    image: nginx
    networks:
      default: null
    ports:
    - mode: ingress
      target: 80
      published: "9090"
      protocol: tcp
networks:
  default:
```

```
    name: dockercomposetest_default
```

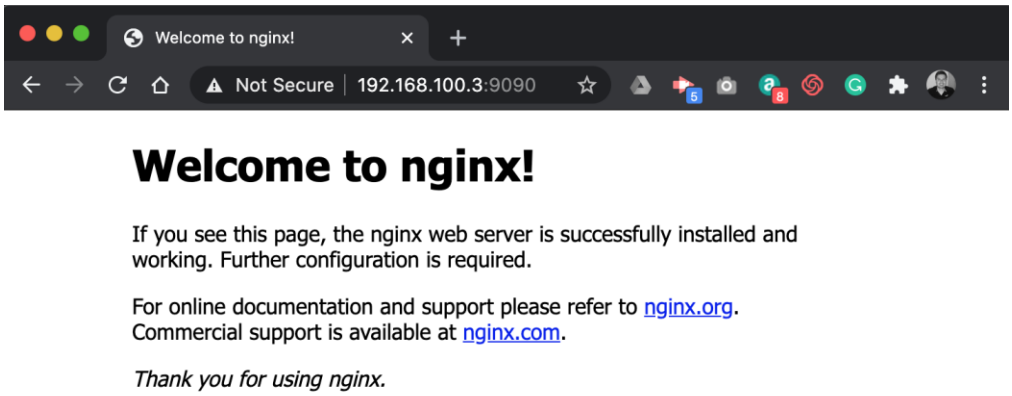
Ahora levantamos los contenedores

```
$ sudo docker compose up -d
[+] Running 3/3
 ⠿ Network dockercomposetest_default       Created
0.1s
 ⠿ Container dockercomposetest-database-1  Started
1.1s
 ⠿ Container dockercomposetest-web-1       Started
1.1s
```

Verificamos:

```
$ sudo docker compose ps
NAME                          COMMAND                  SERVICE
STATUS              PORTS
dockercomposetest-database-1   "docker-entrypoint.s…"   database
running             6379/tcp
dockercomposetest-web-1        "/docker-entrypoint.…"   web
running             0.0.0.0:9090->80/tcp, :::9090->80/tcp
```

En el browser:



Detener la aplicación usando docker compose down

```
$ sudo docker compose down
[+] Running 3/2
 ⠿ Container dockercomposetest-database-1  Removed
0.7s
```

```
 ⠿ Container dockercomposetest-web-1       Removed
10.5s
 ⠿ Network dockercomposetest_default       Removed
0.1s
```

NOTA:

Si se requiere cambiar uno de los servicios se puede usar el comando:

```
sudo docker compose up -d --no-deps --build <nombre del servicio>
```

Por ejemplo:

```
sudo docker compose up -d --no-deps --build web
```

PARTE 5. Escalar la aplicación

En esta parte escalaremos el contenedor database para instanciarse 4 veces

```
$ sudo docker compose up -d --scale database=4
[+] Running 5/5
 ⠿ Container dockercomposetest-web-1       Running
0.0s
 ⠿ Container dockercomposetest-database-4  Started
2.8s
 ⠿ Container dockercomposetest-database-1  Started
2.6s
 ⠿ Container dockercomposetest-database-2  Started
2.7s
 ⠿ Container dockercomposetest-database-3  Started
2.0s
```

Verificar:

```
$ sudo docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED              STATUS
PORTS                                      NAMES
247f7a96f603   redis     "docker-entrypoint.s…"   About a minute ago   Up About
a minute   6379/tcp                             dockercomposetest-database-2
2bfd98f9aacf   redis     "docker-entrypoint.s…"   About a minute ago   Up About
a minute   6379/tcp                             dockercomposetest-database-1
48c896c96216   redis     "docker-entrypoint.s…"   About a minute ago   Up About
a minute   6379/tcp                             dockercomposetest-database-4
fe80364c7de9   redis     "docker-entrypoint.s…"   About a minute ago   Up About
a minute   6379/tcp                             dockercomposetest-database-3
786c094a0634   nginx     "/docker-entrypoint.…"   5 minutes ago        Up 5
minutes        0.0.0.0:9090->80/tcp, :::9090->80/tcp   dockercomposetest-web-1
```

En este caso podemos ver que el contenedor redis se instanció 4 veces mientras que el nginx una sola vez.

Detener los contenedores

```
$ sudo docker compose down
```

PARTE 5. Docker compose + Flask + MySQL

Para esta parte trabajaremos con el repositorio https://github.com/stavshamir/docker-tutorial. Clonar así:

```
$ git clone https://github.com/stavshamir/docker-tutorial
```

```
$ cd docker-tutorial
```

Una vez clonado deberá tener la siguiente estructura de archivos en el directorio clonado:

```
$ tree
.
├── app
│   ├── app.py
│   ├── Dockerfile
│   └── requirements.txt
├── db
│   └── init.sql
├── docker-compose.yml
├── GitLab-CI.md
└── README.md
```

Dockerfile

```
# Use an official Python runtime as an image
FROM python:3.6

# The EXPOSE instruction indicates the ports on which a container
# will listen for connections
# Since Flask apps listen to port 5000  by default, we expose it
EXPOSE 5000

# Sets the working directory for following COPY and CMD instructions
# Notice we haven't created a directory by this name - this instruction
```

```
# creates a directory with this name if it doesn’t exist
WORKDIR /app

# Install any needed packages specified in requirements.txt
COPY requirements.txt /app
RUN pip install -r requirements.txt

# Run app.py when the container launches
COPY app.py /app
CMD python app.py
```

En este archivo se especifica lo siguiente:

- Nuestra imagen estará basada en una imagen de Python 3.6
- Exponga el puerto 5000 (para Flask)
- Cree un directorio de trabajo en el que se copiarán requirements.txt y app.py
- Instale los paquetes necesarios y ejecute la aplicación

Necesitamos que nuestras dependencias (Flask y mysql-connector) se instalen y se entreguen con la imagen, por lo que debemos crear el archivo requirements.txt mencionado anteriormente:

requirements.txt:

```
Flask
mysql-connector
```

Ahora podemos crear una imagen Docker para nuestra aplicación, pero aún no podemos usarla, ya que depende de MySQL, que, como práctica recomendada, residirá en un contenedor diferente. Utilizaremos Docker compose para facilitar la orquestación de los dos contenedores independientes en una aplicación.

Archivo docker-compose.yml:

```
version: "2"
services:
  app:
    build: ./app
    links:
      - db
    ports:
      - "5000:5000"

  db:
    image: mysql:5.7
    ports:
      - "32000:3306"
    environment:
      MYSQL_ROOT_PASSWORD: root
    volumes:
      - ./db:/docker-entrypoint-initdb.d/:ro
```

Estamos utilizando dos servicios, uno es un contenedor que expone la API REST (app) y otro contiene la base de datos (db).

build: especifica el directorio que contiene el Dockerfile que contiene las instrucciones para construir este servicio

links: vincula este servicio a otro contenedor. Esto también nos permitirá usar el nombre del servicio en lugar de tener que buscar la dirección IP del contenedor de la base de datos y expresar una dependencia que determinará el orden de inicio del contenedor.

puertos: mapeo de puertos <Host>:<Container>.

image: estamos utilizando una imagen existente de un repositorio. Es importante especificar la versión; si su cliente mysql instalado no es de la misma versión, pueden producirse problemas.

environment: agrega variables de entorno. La variable especificada es necesaria para esta imagen y, como su nombre lo indica, configura la contraseña para el usuario raíz de MySQL en este contenedor. Aquí se especifican más variables.

ports: dado que es probable que tengamos una instancia de mysql en ejecución en el host usando este puerto, la estamos asignando a una diferente. Tenga en cuenta que la asignación es solo del host al contenedor, por lo que nuestro contenedor de servicios de aplicaciones seguirá utilizando el puerto 3306 para conectarse a la base de datos.

volumes: dado que queremos que el contenedor se inicialice con nuestro esquema, conectamos el directorio que contiene nuestro script init.sql al punto de entrada para este contenedor, que según la especificación de la imagen ejecuta todos los scripts .sql en el directorio dado. Note el :ro al final de `./db:/docker-entrypoint-initdb.d/:ro` esto significa el que volumen es read only.

app.py

```
from typing import List, Dict
from flask import Flask
import mysql.connector
import json

app = Flask(__name__)


def favorite_colors() -> List[Dict]:
    config = {
        'user': 'root',
        'password': 'root',
        'host': 'db',
        'port': '3306',
        'database': 'knights'
    }
    connection = mysql.connector.connect(**config)
    cursor = connection.cursor()
    cursor.execute('SELECT * FROM favorite_colors')
```

```
    results = [{name: color} for (name, color) in cursor]
    cursor.close()
    connection.close()

    return results


@app.route('/')
def index() -> str:
    return json.dumps({'favorite_colors': favorite_colors()})


if __name__ == '__main__':
    app.run(host='0.0.0.0')
```

Nos estamos conectando como root con la contraseña configurada en el archivo docker-compose.yml. Tenga en cuenta que definimos explícitamente el host (que es localhost de forma predeterminada) ya que el servicio SQL está en un contenedor diferente al que ejecuta este código. Podemos (y debemos) usar el nombre "db" ya que este es el nombre del servicio que definimos y vinculamos anteriormente, y el puerto es 3306 y no 32000 ya que este código no se está ejecutando en el host.

init.sql:

Se crea la base de datos knights y se insertan algunos registros.

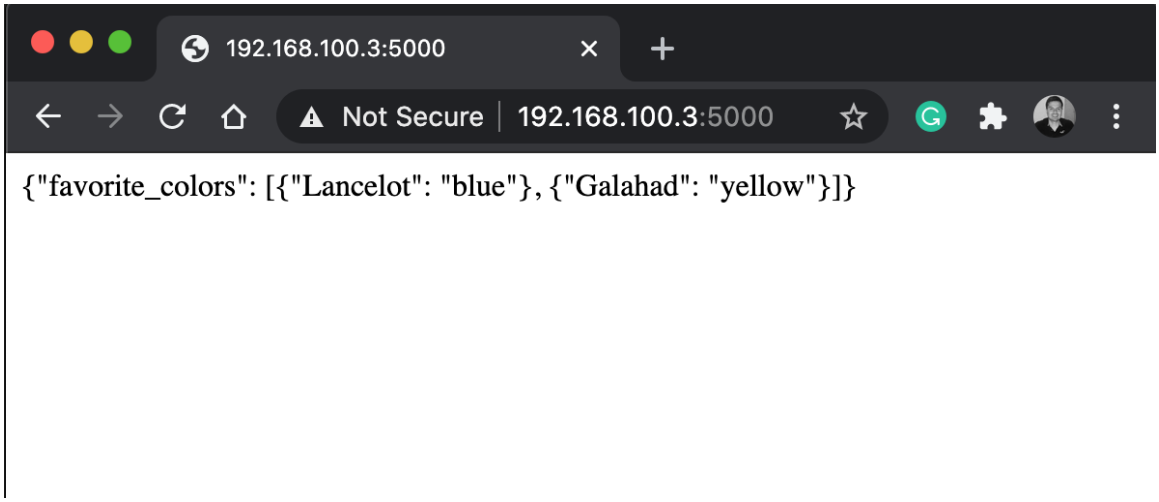
```
CREATE DATABASE knights;
use knights;

CREATE TABLE favorite_colors (
  name VARCHAR(20),
  color VARCHAR(10)
);

INSERT INTO favorite_colors
  (name, color)
VALUES
  ('Lancelot', 'blue'),
  ('Galahad', 'yellow');
```

Correr la aplicación

Desde el directorio docker-tutorial ejecutar:

```
$ sudo docker compose up -d
```

Verificar en el browser



Una vez finalice las pruebas puede parar la aplicación usando CTRL+C y destruir los contenedores docker usando docker compose down.

Adicionalmente, puede acceder a la base de datos directamente desde el box de vagrant utilizando el mysql client y el siguiente comando (asegúrese de que su cliente tenga la misma versión de MySQL especificada en docker-compose.yml):

Instalar mysql-client

```
sudo apt-get install mysql-client
```

Conectarse a la base de datos (password root)

```
$ mysql --host=127.0.0.1 --port=32000 -u root -p
```

Es obligatorio especificar la IP del host, ya que cuando se usa el "localhost" predeterminado, MySQL ignora el parámetro del puerto. Recuerde que password es root de acuerdo a la configuración en app.py.

4. Ejercicio

1. ¿Como Docker compose facilita el trabajo de implementación con respecto a Docker básico?
2. Configure un servidor FTP usando docker compose

   Ver: https://www.youtube.com/watch?v=y9sPUF7vRC0&t=181s

5. Desafío [Hasta 0.5 puntos en notas de practicas]

1. Demuestre el uso de docker compose para una aplicación de su interés. Puede tomar uno de los ejemplos descritos aquí: https://docs.docker.com/compose/samples-for-compose/ o usar otras fuentes.

6. Bibliografía

- Sitio oficial Docker. https://www.docker.com/
- Tutorial Docker Compose. https://www.youtube.com/watch?v=HUpIoF_conA
- Configuracion de vim para trabajar con Yaml. https://lornajane.net/posts/2018/vim-settings-for-working-with-yaml
- Docker Compose Releases. https://github.com/docker/compose/releases
- Contenerdor docker nginx. https://hub.docker.com/_/nginx
- Dockerizing a Flask-MySQL app with docker-compose. https://stavshamir.github.io/python/dockerizing-a-flask-mysql-app-with-docker-compose/