Containers - Ejercicios

En esta práctica veremos cómo utilizar containers para correr servicios de manera rápida.

1. Setup: Windows como Plataforma

- Pueden usar WSL
- Pueden instalar Docker usando el siguiente link
  https://docker-doc.readthedocs.io/zh_CN/stable/installation/windows.html

2. Práctica Guiada 1: Introducción a Docker

Para manejar **Docker** como un *non-root user*

El *daemon de Docker* se conecta a un *socket* de Unix en lugar de un puerto TCP. Por defecto, el *socket de Unix* tiene como propietario al usuario **root** y otros usuarios pueden solamente accederlo usando `sudo`. El *daemon de Docker* siempre corre como el usuario `root`.

Si uno no quiere escribir el comando `docker` antecediendo el comando `sudo`, debemos crear un grupo Unix llamado **docker** y agregar usuarios al mismo. Cuando el *daemon de Docker* comienza, crea un *socket de Unix* accesible por miembos del grupo de **Docker**.

- **Paso 1:** Crear un grupo de Docker y agregar su usuario:

```
$ sudo groupadd docker

$ sudo usermod -aG docker $USER
```

- **Paso 2:** Deslogguearse y volverse a loggear para que el grupo sea reevaluado.

  Si se realiza el *testing* en una máquina virtual, puede que sea necesario reiniciar la máquina virtual para que se hagan los cambios.

  En un entorno de escritorio Linux como X Windows, cierre la sesión por completo y luego vuelva a iniciarla.

  En Linux, también puede ejecutar el siguiente comando para activar los cambios en los grupos:

```
$ newgrp docker
```

- **Paso 3:** Comando **`run`**

  Lo primero que vamos a hacer es bajar una imagen:

```
$ docker pull hello-world
```

  Ahora vamos a correr la imagen:

```
$ docker run hello-world
```

- **Paso 4:** Comando **`images`** Para ver las imágenes que tenemos bajadas podemos usar el siguiente comando:

```
$ docker images
```

```
REPOSITORY    TAG       IMAGE ID          CREATED          SIZE
hello-world   latest    4ab4c602aa5e      3 months ago     1.84kB
```

- **Paso 5:** Corriendo un web server Lo siguiente que vamos hacer es correr un servidor NGINX y mostrar un mensaje:

```
$ mkdir /tmp/web
$ echo "Hola Mundo" > /tmp/web/index.html
$ docker run -name nginx-test -v /tmp/web:/usr/share/nginx/html:ro -p 8080:80 -d nginx
```

Luego de esto vamos a poder entrar a http://<IP del host>:8080/ y ver el mensaje "`Hola Mundo`".

El último comando fue un tanto complejo así que veremos que hace en detalle:

- ***-name nginx-test***
  - *Define el nombre del nuevo container como nginx-test. Cuando queramos interactuar desde el container con este host podremos usar este nombre.*
- ***-v /tmp/web:/usr/share/nginx/html:ro***
  - *Este comando indica que queremos montar un directorio del host dentro del container. En este caso la web minimalista que creamos con los primeros dos comandos.*
- ***-p 8080:80***
  - *Esta opción sirve para forwardear el puerto 8080 del host al 80 del container. Esto es lo que nos permite acceder al NGINX.*
- ***-d***
  - *Indica que el container se correrá en modo background, liberando la terminal actual.*
- ***nginx:alpine***
  - *Indica a docker la imagen que queremos usar para correr el container.*

Para ver los containers que tenemos levantados vamos a usar el comando **`ps`**:

```
$ docker ps
```

```
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS               NAMES
b5fb72ccfd5f        nginx:alpine        "nginx -g 'daemon of..."   10 seconds ago      Up 8 seconds        80/tcp              nginx-test
```

Finalmente detenemos y borramos el container:

```
$ docker rm -f nginx-test
```

Práctica Guiada 2: Orquestación con Docker-compose

Como se vio en la parte anterior correr un solo container resulta sencillo. Sin embargo correr manualmente varios a la vez puede resultar tedioso y propenso a errores.

Para solucionar este problema existe la herramienta **`docker-compose`**. La misma se encarga de orquestar múltiples containers que aportarían distintos servicios que actúan como de la misma aplicación.

- **Paso 1:** Setup

  Creamos el directorio del proyecto

  ```
  $ mkdir /tmp/composetest
  $ cd /tmp/composetest
  ```

  Creamos la app: Para esto creamos un archivo llamado `/tmp/composetest/app.py` con el siguiente contenido.

  ```
  import time

  from flask import Flask
  import redis

  app = Flask(__name__)
  cache = redis.Redis(host='redis', port=6379)
  ```

```
def get_hit_count():
    retries = 5
    while True:
        try:
            return cache.incr('hits')
        except redis.exceptions.ConnectionError as exc:
            if retries == 0:
                raise exc
            retries -= 1
            time.sleep(0.5)


@app.route('/')
def hello():
    count = get_hit_count()
    return 'Hello World! I have been seen {} times.\n'.format(count)

if __name__ == "__main__":
    app.run(host="0.0.0.0", debug=True)
```

Esta es una mini webapp que usa un servidor Redis para llevar a cabo el conteo de los usuarios que visitan la pagina. Notar que el host y puerto de este servicio es `redis:6379`.

Para esto creamos un archivo llamado `/tmp/composetest/requirements.txt` con el siguiente contenido:

```
flask
redis
```

Este archivo contiene las dependencias de la app.

- **Paso 2:** Crear el Dockerfile

  Para esto creamos un archivo llamado `/tmp/composetest/Dockerfile` con el siguiente contenido:

```
FROM python:3.4-alpine
ADD . /code
WORKDIR /code
RUN pip install -r requirements.txt
CMD ["python", "app.py"]
```

  Este archivo será usado para crear la imagen que usaremos para correr la app. En la 5ta línea se instalan las dependencias de la app y en la 6ta se define el comando que se correrá al iniciar la imagen.

- **Paso 3:** Crear el archivo de orquestación

  Creamos un archivo llamado `/tmp/composetest/docker-compose.yml` con el siguiente contenido:

```
version: '3'
services:
  web:
    build: .
    ports:
     - "8080:5000"
  redis:
    image: "redis:alpine"
```

  Este archivo define dos servicios:

  - **web**
    - Usa una imagen construida a partir del Dockerfile en el directorio actual
    - Expone el puerto 80 del container a al 8080 del host.

- **redis**
  - Este servicio usa una imagen que se bajará automáticamente del registro de imágenes de Docker Hub.

- **Paso 4:** Construir y correr la app Primero levantamos el servicio:

  ```
  $ docker-compose up
  ```

  En este paso se bajarán y construirán las imágenes que sean necesarias para correr todos los servicios definidos.

  Hecho esto podremos acceder a la ip del host a través de un browser cualquiera para probar el servicio.

Práctica independiente: Crear una imagen para correr un comando

Como mencionamos al comienzo es posible usar el comando docker run para correr containers directamente desde la shell.

Para practicar esto crearemos una imagen que al ser corrida tomará dos cadenas de caracteres e imprimirá por pantalla la concatenación de ambas.

Será necesario definir los siguientes archivos en un directorio cualquiera (por ej: `/tmp/parte3/`):

- El archivo Dockerfile. Usar como guia el del punto anterior.
- El código python a correr (ej: `/tmp/parte3/code.py`).

El código Python puede ser algo como esto:

```
import argparse

def main(a, b):
    print(a+b)

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description="Ejercicio clase Docker.")
    parser.add_argument('-a')
    parser.add_argument('-b')

    args = parser.parse_args()
    a = args.a
    b = args.b
    main(a, b)
```

La salida esperada sería la siguiente:

```
$ python code.py -a hola -b mundo
holamundo
```

Se espera obtener el mismo resultado cuando se corra el container.

3. Bonus

Modificar el código anterior y anterior para que en lugar de concatenar caracteres el programa recibe el nombre de un archivo CSV con una lista de números, lea su contenido e imprima por pantalla la suma del mismo.

Para esto va a ser necesario pasar el argumento `-v` a docker run de modo que el script tenga acceso al archivo del host que se desea leer.