

## **Proyecto - Sistemas Operativos**

### **Docker-Compose**

#### **Integrantes**

Marcelo Alejandro García Millán - 201941427

Kevin Alejandro Vélez Agudelo - 202123281

**Asignatura:** Sistemas Operativos

**Docente:** John Alexander Sanabria Ordóñez



Universidad Del Valle

Santiago de Cali

6 de febrero del 2023

## **Tabla de Contenido**

<b>Introducción y Metodología</b>	<b>3</b>
Ejecución	3
<b>Dificultades y Soluciones</b>	<b>5</b>
<b>Recursos</b>	<b>9</b>
Repositorio	9
Video de la aplicación en YouTube	9
<b>Referencias</b>	<b>10</b>

## Introducción y Metodología

En este proyecto, se tuvo como finalidad crear una aplicación conformada de dos partes: Una **Interfaz Web** y una **Base de Datos**, el objetivo era gestionar un contador por medio de la codificación de endpoints que manejan métodos de petición HTTP (GET, DELETE, POST Y PUT).

Para la solución de este proyecto se decidieron usar tres componentes clave: **Python** como el entorno de programación de la aplicación web, **Flask** para la creación de la aplicación web, y **PostgreSQL** como el sistema gestor de nuestra Base de Datos, adicionalmente, se usó **Psycopg** como el conector de la aplicación web con la Base de Datos gestionada por Postgres, este es de gran utilidad para hacerle peticiones y consultas a nuestra Base de Datos.

Finalmente, con **Docker** y **Docker Compose** podemos levantar la aplicación con la ayuda de contenedores que se comunicarán de forma conjunta.

### Ejecución

Para la ejecución de la aplicación, se requiere de tener como prerequisite los siguientes componentes en el sistema:

- Docker
- Docker Compose

Una vez con los componentes, con la terminal, nos adentramos en la carpeta del proyecto, y ejecutamos dos comandos:

1. El `docker-compose build`: nos servirá para construir la imagen que contendrá los servicios que especificamos en el archivo `docker-compose.yml`.
2. El `docker-compose up`: nos servirá para construir, iniciar y conectarnos a los contenedores de los servicios y así empezar la ejecución de la aplicación.

Ya con la aplicación en ejecución, podemos empezar a hacer peticiones HTTP, esto lo podemos hacer con el comando `curl -i -X <METHOD> <URL>`, explicado así:

- curl: Es una herramienta de comandos usada para la transferencia de datos hacia una URL especificada.
- -i: Esta bandera nos especifica que queremos que nos imprima en pantalla la salida de la petición HTTP que hemos hecho

- -x <METHOD>: Con esta bandera podemos especificar el método que queremos usar, en <METHOD> iría el método.
- <URL>: Ahí va la URL del sitio web al que queremos hacerle peticiones HTTP.

## Dificultades y Soluciones

En el desarrollo del proyecto se presentaron varias dificultades, unas más que otras pero hallamos alternativas o soluciones a ellas. La primera dificultad antes de empezar con el proyecto fue que el subsistema de Linux en el sistema operativo Windows (WSL) tenía muchas incompatibilidades con el programa Docker y uso de muchas librerías para el desarrollo de la aplicación web con Docker-Compose. La alternativa tomada fue instalar un sistema operativo Linux en una máquina virtual, mediante el programa Oracle VM virtualbox. Cada integrante usó una distribución de Linux diferente (Ubuntu y Mint) pero eso no fue impedimento para llevar a cabo el proyecto.



Imagen 1. Distribución de Linux: Ubuntu.



Imagen 2. Distribución de Linux: Mint.

Algo inusual es que para el compañero que usó la distribución de Mint, no tenía permisos de superusuario, se buscó en la web la solución, básicamente se debía crear un grupo con los permisos de superusuario y en él colocar el usuario actualmente logueado.

```
kvelez@kvelez-VirtualBox: ~/compose-project
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
CPU: 3.362s
CGroup: /system.slice/docker.service
└─964 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/conta

feb 04 16:09:55 kvelez-VirtualBox dockerd[964]: time="2023-02-04T16:09:55.913115
feb 04 16:09:55 kvelez-VirtualBox dockerd[964]: time="2023-02-04T16:09:55.91328
feb 04 16:09:56 kvelez-VirtualBox dockerd[964]: time="2023-02-04T16:09:56.35932
feb 04 16:09:56 kvelez-VirtualBox dockerd[964]: time="2023-02-04T16:09:56.42675
feb 04 16:09:58 kvelez-VirtualBox dockerd[964]: time="2023-02-04T16:09:58.39523
feb 04 16:09:58 kvelez-VirtualBox dockerd[964]: time="2023-02-04T16:09:58.77618
feb 04 16:09:59 kvelez-VirtualBox dockerd[964]: time="2023-02-04T16:09:59.20817
feb 04 16:09:59 kvelez-VirtualBox dockerd[964]: time="2023-02-04T16:09:59.21315
feb 04 16:09:59 kvelez-VirtualBox systemd[1]: Started Docker Application Contai
feb 04 16:09:59 kvelez-VirtualBox dockerd[964]: time="2023-02-04T16:09:59.43816
lines 1-22/22 (END)
^C
kvelez@kvelez-VirtualBox:~/compose-project$ sudo docker ps -a
[sudo] contraseña para kvelez:
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS   NAMES
kvelez@kvelez-VirtualBox:~/compose-project$ docker ps -a
Got permission denied while trying to connect to the Docker daemon socket at uni
x:///var/run/docker.sock: Get "http://%2Fvar%2Frun%2Fdocker.sock/v1.24/container
s/json?all=1": dial unix /var/run/docker.sock: connect: permission denied
kvelez@kvelez-VirtualBox:~/compose-project$
```

Imagen 3. Comandos en terminal (BASH) sin permisos de superusuario.

Para crear el grupo con permisos administrativos se ejecuta en la terminal el siguiente comando: `$ sudo groupadd docker`, después, para añadir el usuario se coloca el comando `$ sudo usermod -aG docker $USER`, luego se loguea en el nuevo grupo creado con: `$ newgrp docker`, por último se digita un comando de docker para saber si los cambios se hicieron exitosamente, un ejemplo sencillo puede ser consultar las imágenes docker mediante: `docker ps -a`.

Mientras se iba avanzando en el proyecto, se consultó sobre la API de la base de datos Postgres, se encontró que su puerto era 5432 y que por ende se debía usar otro puerto diferente al que se iba usar para las consultas, usamos el puerto 5555 del anfitrión para la conexión con la base de datos.

```
volumes:
  - ./sql/1schema.sql:/docker-entrypoint-initdb.d/1schema.sql
  - ./sql/2data.sql:/docker-entrypoint-initdb.d/2data.sql
ports:
  - "5555:5432"
```

Imagen 4. Puertos usados para la conexión con API Postgres..

Otra dificultad fue la de lograr la exitosa conexión con la base de datos, se creía que con el puerto era suficiente, investigando nos dimos cuenta que se necesitaba un adaptador de PostgreSQL para la aplicación de Python, este nos permitió hacer las peticiones a la base de datos respectivamente. Se agrega este adaptador en el documento requires.txt.

```
1 flask
2 psycopg2-binary
```

Imagen 5. Documento requires.txt.

Mediante el desarrollo del servidor no tuvimos ninguna complicación, siguiendo las indicaciones del docente y un tutorial alojado en la plataforma de Youtube pudimos completar el servidor con docker-compose. La última dificultad que se presentó fue la ejecución del servidor mediante el comando `docker-compose up`, la primera vez que lo ejecutamos nos salió un error de que el servidor no existía o que la base de

datos estaba lista para reiniciarse, esto era debido a que primero se creaba el servidor antes que la base de datos.

```
flask api db Success. You can now start the database server using:
pg_ctl -D /var/lib/postgresql/data -l logfile start

waiting for server to start....2023-02-06 19:33:02.362 UTC [47] LOG: starting PostgreSQL 15.1 (Debian 15.1-1.pgdb118+1) on x86_64-pc-linux-gnu, compiled by gcc (Debian 10.2.1-6) 10.2-1.20210110, 64-bit
2023-02-06 19:33:02.366 UTC [47] LOG: listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"
2023-02-06 19:33:02.375 UTC [50] LOG: database system was shut down at 2023-02-06 19:33:01 UTC
2023-02-06 19:33:02.389 UTC [47] LOG: database system is ready to accept connections
done
server started
CREATE DATABASE
/usr/local/bin/docker-entrypoint.sh: running /docker-entrypoint-initdb.d/schema.sql
CREATE TABLE
/usr/local/bin/docker-entrypoint.sh: running /docker-entrypoint-initdb.d/2data.sql
INSERT 0 1
2023-02-06 19:33:02.682 UTC [47] LOG: received fast shutdown request
waiting for server to shut down....2023-02-06 19:33:02.688 UTC [47] LOG: aborting any active transactions
2023-02-06 19:33:02.691 UTC [47] LOG: background worker "logical replication launcher" (PID 53) exited with exit code 1
2023-02-06 19:33:02.691 UTC [48] LOG: shutting down
2023-02-06 19:33:02.696 UTC [48] LOG: checkpoint starting: shutdown immediate
2023-02-06 19:33:02.983 UTC [48] LOG: checkpoint complete: wrote 921 buffers (5.6%); 0 WAL file(s) added, 0 removed, 0 recycled; write=0.084 s, sync=0.189 s, total=0.292 s; sync files=252, longest=0.005 s, average=0.001 s; distance=4225 kB, estimate=4225 kB
2023-02-06 19:33:02.987 UTC [47] LOG: database system is shut down
done
server stopped
PostgreSQL init process complete; ready for start up.
2023-02-06 19:33:03.109 UTC [1] LOG: starting PostgreSQL 15.1 (Debian 15.1-1.pgdb118+1) on x86_64-pc-linux-gnu, compiled by gcc (Debian 10.2.1-6) 10.2.1.20210110, 64-bit
2023-02-06 19:33:03.111 UTC [1] LOG: listening on IPv4 address "0.0.0.0", port 5432
2023-02-06 19:33:03.111 UTC [1] LOG: listening on IPv6 address ":::", port 5432
2023-02-06 19:33:03.123 UTC [1] LOG: listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"
2023-02-06 19:33:03.131 UTC [67] LOG: database system was shut down at 2023-02-06 19:33:02 UTC
2023-02-06 19:33:03.136 UTC [1] LOG: database system is ready to accept connections
```

Imagen 6. Foto de terminal Bash dónde se ejecutó el comando: docker-compose up por primera vez.

La solución fue crear una flag (es una condición para el documento docker-compose) dentro del archivo docker-compose.yml.

```

1 version: "3.9"
2 services:
3   web:
4     build: .
5     restart: on-failure
6     ports:
7       - "7000:5000"
8     volumes:
9       - ./code
10    environment:
11      FLASK_DEBUG: "true"
12  postgres:
13    image: "postgres:latest"
14    container_name: flask_api_db
15    restart: on-failure
16    environment:
17      - DATABASE_HOST=postgres
18      - POSTGRES_DB=flask_api
19      - POSTGRES_USER=postgres
20      - POSTGRES_PASSWORD=admin
21    volumes:
22      - ./sql/1schema.sql:/docker-entrypoint-initdb.d/1schema.sql
23      - ./sql/2data.sql:/docker-entrypoint-initdb.d/2data.sql
24    ports:
25      - "5555:5432"

```

Imagen 8. Documento docker-compose.yml, en la línea 5 se especifica el flag colocado.

Este flag nos permite básicamente repetir el proceso de construcción del servidor si sale un error, con esto nos ahorramos volver a ejecutar el comando: `docker-compose up`.

```
kvelez@kvelez-VirtualBox:~/compose-project$ docker-compose up
Recreating flask_api_db ... done
Recreating compose-project_web_1 ... done
Attaching to compose-project_web_1, flask_api_db
flask_api_db | PostgreSQL Database directory appears to contain a database; Skipping initialization
flask_api_db |
flask_api_db | 2023-02-06 20:06:23.831 UTC [1] LOG: starting PostgreSQL 15.1 (Debian 15.1-1.pgdg110+1) on x86_64-pc-linux-gnu, compiled by gcc (Debian 10.2.1-6) 10.2.1 20210110, 64-bit
flask_api_db | 2023-02-06 20:06:23.832 UTC [1] LOG: listening on IPv4 address "0.0.0.0", port 5432
flask_api_db | 2023-02-06 20:06:23.832 UTC [1] LOG: listening on IPv6 address ":::", port 5432
flask_api_db | 2023-02-06 20:06:23.838 UTC [1] LOG: listening on Unix socket "/var/run/postgresql/.s.PGSQL_5432"
flask_api_db | 2023-02-06 20:06:23.846 UTC [27] LOG: database system was shut down at 2023-02-06 19:58:44 UTC
flask_api_db | 2023-02-06 20:06:23.863 UTC [1] LOG: database system is ready to accept connections
web_1 | * Serving Flask app 'app.py'
web_1 | * Debug mode: on
web_1 | WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
web_1 | * Running on all addresses (0.0.0.0)
web_1 | * Running on http://172.0.0.1:5000
web_1 | * Running on http://172.18.0.2:5000
web_1 | Press CTRL+C to quit
web_1 | * Restarting with stat
web_1 | * Debugger is active!
web_1 | * Debugger PIN: 355-285-594
```

Imagen 9. Docker-compose ejecutándose exitosamente al primer comando de docker-compose up.



## Recursos

### Repositorio

- <https://github.com/MarceloUnivalle/compose-project>



### Video de la aplicación en YouTube

- <https://www.youtube.com/watch?v=5nn-2NprW7w>



## Referencias

- *The psycpg2 module content — Psycpg 2.9.5 documentation.* (n.d.). Psycpg. Retrieved February 5, 2023, from <https://www.psycpg.org/docs/module.html>
- *How to fix docker: Got permission denied issue.* (2018, February 23). Stack Overflow. Retrieved February 5, 2023, from <https://stackoverflow.com/questions/48957195/how-to-fix-docker-got-permission-denied-issue>
- UskoKruM2010. (2022, 23 de abril). *REST API con Python, Flask y PostgreSQL | Crea y Prueba tu REST API (GET, POST, PUT, DELETE).* [Video]. YouTube. From <https://www.youtube.com/watch?v=rjiit-xcqYI>
- *Compose specification.* (n.d.). Docker Documentation. Retrieved February 6, 2023, from <https://docs.docker.com/compose/compose-file/>
- *Documentation: 15: PostgreSQL 15.1 Documentation.* (n.d.). PostgreSQL. Retrieved February 6, 2023, from <https://www.postgresql.org/docs/current/>