Desarrollo

Para esta práctica comenzaremos creando un par de microservicios enfocados en realizar operaciones matemáticas básicas, en este caso definí un microservicio para sumar y otro para restar.

```python
from flask import Flask, request

app = Flask(__name__)

@app.route('/restar', methods=['POST'])
def restar()
    try
        data = request.get_json()
        num1 = data['num1']
        num2 = data['num2']
        resultado = num1 - num2
        return {'resultado' resultado}
    except Exception as e
        return {'error' str(e)}

if __name__ == '__main__'
    app.run(debug=True, host='0.0.0.0', port=5000)
```

Posterior a ello creé el archivo Dockerfile para cada microservicio, esto para poder sincronizarlos con una imagen y una red de Docker, de manera que permita su uso por medio de un balanceo de carga de trabajo.

Dockerfile

sumar > Dockerfile

```
1  FROM python:3.8
2
3  WORKDIR /app
4  COPY requirements.txt .
5  RUN pip install --no-cache-dir -r requirements.txt
6
7  COPY . .
8
9  CMD ["python", "sumar.py"]
```

Para este punto generaremos el archivo Docker-compose.yml que será el que nos ayudará a unir todos los microservicios junto con sus respectivas configuraciones para lograr conectarlas y ejecutarlas de manera simultanea en la red Docker.

Dockerfile | docker-compose.yml | nginx.conf

docker-compose.yml

```yaml
version: '3'

services:
  sumar:
    build: ./sumar
    expose:
      - "5000"
    networks:
      - calculadora-network

  restar:
    build: ./restar
    expose:
      - "5000"
    networks:
      - calculadora-network

  nginx:
    image: nginx
    volumes:
      - ./nginx.conf:/etc/nginx/nginx.conf
    ports:
      - "8080:80"
    networks:
      - calculadora-network

networks:
  calculadora-network:
    driver: bridge
```

Ejecutamos los comandos necesarios para crear las imágenes de Docker y dejamos que se realice las acciones necesarias.

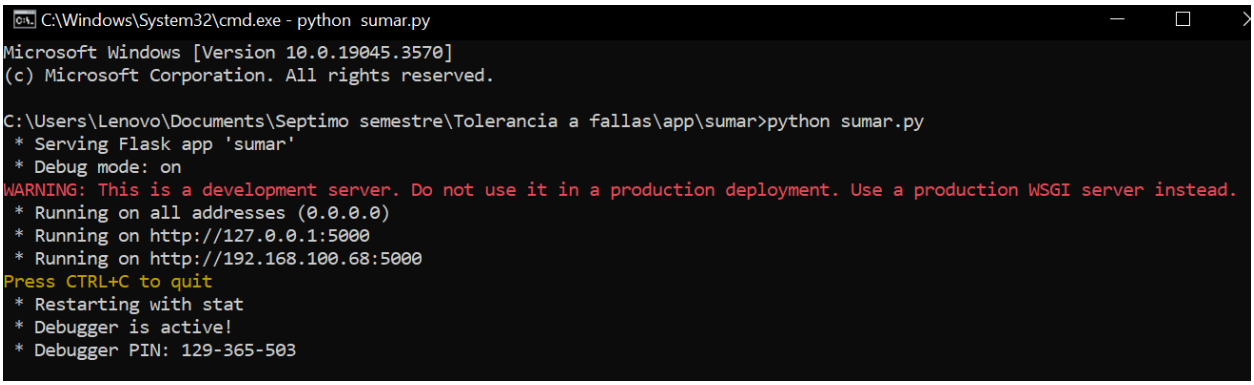
```
C:\Users\Lenovo\Documents\Septimo semestre\Tolerancia a fallas\app>docker-compose up --build
[+] Building 0.0s (0/0)                                                                                                  docker:default
[+] Building 8.3s (8/16)                                                                                                 docker:default
 => [restar internal] load build definition from Dockerfile                                                                       0.1s
 => => transferring dockerfile: 190B                                                                                              0.0s
 => [restar internal] load .dockerignore                                                                                          0.0s
 => => transferring context: 2B                                                                                                   0.0s
 => [sumar internal] load build definition from Dockerfile                                                                        0.1s
 => => transferring dockerfile: 189B                                                                                              0.0s
 => [sumar internal] load .dockerignore                                                                                           0.0s
 => => transferring context: 2B                                                                                                   0.0s
 => [sumar internal] load metadata for docker.io/library/python:3.8                                                               3.6s
 => [restar auth] library/python:pull token for registry-1.docker.io                                                              0.0s
 => [sumar 1/5] FROM docker.io/library/python:3.8@sha256:7a82536f5a2895b70416ccaffc49e6469d11ed8d9bf6bcfc52328faeae7c7710         4.7s
 => => resolve docker.io/library/python:3.8@sha256:7a82536f5a2895b70416ccaffc49e6469d11ed8d9bf6bcfc52328faeae7c7710               0.1s
 => => sha256:7a82536f5a2895b70416ccaffc49e6469d11ed8d9bf6bcfc52328faeae7c7710 1.86kB / 1.86kB                                     0.0s
 => => sha256:129534c722d189b3baf69f6e3289b799caf45f75da37035c854100852edcbd7d 2.01kB / 2.01kB                                     0.0s
 => => sha256:8457fd5474e70835e4482983a5662355d892d5f6f0f90a27a8e9f009997e8196 8.39MB / 49.58MB                                    4.6s
 => => sha256:795c73a8d985b6d1b7e5730dd2eece7f316ee2607544b0f91841d4c4142d9448 7.56kB / 7.56kB                                     0.0s
 => => sha256:13baa2029dde87a21b87127168a0fb50a007c07da6b5adc8864e1fe1376c86ff 10.49MB / 24.05MB                                   4.6s
 => => sha256:325c5bf4c2f26c11380501bec4b6eef8a3ea35b554aa1b222cbcd1e1fe11ae1d 5.24MB / 64.13MB                                    4.6s
 => [sumar internal] load build context                                                                                           0.1s
 => => transferring context: 712B                                                                                                 0.0s
 => [restar internal] load build context                                                                                          0.1s
 => => transferring context: 719B                                                                                                 0.0s
```

Ejecutamos el microservicio y podemos observar que está funcionando correctamente, en este caso mostramos el microservicio de restar.

```
C:\Windows\System32\cmd.exe - python  restar.py
Microsoft Windows [Version 10.0.19045.3570]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Lenovo\Documents\Septimo semestre\Tolerancia a fallas\app\restar>python restar.py
 * Serving Flask app 'restar'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://192.168.100.68:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 129-365-503
```

A continuación podemos observar el microservicio de sumar, por lo que vemos todo está funcionando correctamente y el siguiente paso sería ejecutarlo con balanceo de cargas con ayuda de kubernetes en la siguiente actividad de clase.



Conclusión

En conclusión, utilizar de microservicios en contenedores Docker ofrece ventajas clave para el desarrollo y despliegue de aplicaciones. La modularidad de los microservicios facilita la escalabilidad, el despliegue consistente y un desarrollo ágil. Docker, al proporcionar un entorno de ejecución consistente y gestionar eficientemente los contenedores, simplifica la administración y mejora la resiliencia de las aplicaciones distribuidas. Sin embargo, es importante tener en cuenta que esta arquitectura puede presentar desafíos iniciales, como la complejidad en la configuración y el monitoreo, así como un posible overhead de recursos. En última instancia, la elección de adoptar esta combinación dependerá de las necesidades específicas del proyecto y de la capacidad para abordar estos desafíos mediante una planificación cuidadosa y el uso de herramientas adecuadas.