UNIVERSIDAD LATINA DE COSTA RICA

Proyecto

Estudiante:

Javier Chinchilla Lugo

Curso:

Sistemas Operativos II

Profesor:

CARLOS ANDRES MENDEZ RODRIGUEZ

Diciembre, 2024

Sede San Pedro

Proyecto:

Este proyecto consistió en configurar un entorno que aloje una aplicación web simple utilizando Docker y permita el balanceo de carga con múltiples contenedores. Se implementaron los pasos necesarios para levantar los servicios, ejecutar pruebas de rendimiento y proponer mejoras para el entorno configurado.

Configurar un entorno con Docker que aloje una aplicación web simple y permita el uso de múltiples contenedores en Ubuntu

Actualizar paquetes del sistema:

```
sudo apt update

sudo apt upgrade
```

Instalar Docker:

```
sudo apt install docker.io -y
```

Habilitar y verificar el servicio Docker:

```
sudo systemctl start docker

sudo systemctl enable docker

sudo systemctl status docker
```

Instalar Docker Compose:

```
sudo apt install docker-compose -y
```

Verificar que Docker y Docker Compose están instalados:

```
docker --version

docker-compose --version
```

Agregar permisos a tu usuario para usar Docker sin sudo:

```
sudo usermod -aG docker $USER
```

Reinicia sesión para que los cambios surtan efecto:

```
Exit
```

Inicio sesión nuevamente.

Preparar los archivos del proyecto

Sigue los pasos para crear los archivos directamente en la terminal de Ubuntu.

Crear el directorio del proyecto:

```
mkdir docker-loadbalancer

cd docker-loadbalancer
```

Crear los archivos requeridos:

Crear el archivo app.py:

```
nano app.py
```

Codigo:

```
from flask import Flask

import os


app = Flask(__name__)


@app.route("/")

def hello():

    return f"Hello from container {os.getenv('HOSTNAME')}!"
```

```
if __name__ == "__main__":
    app.run(host="0.0.0.0", port=5000)
```

Crear el archivo Dockerfile:
nano Dockerfile

Codigo:

```
FROM python:3.9-slim

WORKDIR /app

COPY app.py /app

RUN pip install flask

CMD ["python", "app.py"]
```

Crear el archivo docker-compose.yml:
nano docker-compose.yml

Codigo:

```
version: '3.8'

services:

app:

        image: myapp

        build:

                context: .

        deploy:

                replicas: 3

        environment:
```

```
            - HOSTNAME=${HOSTNAME}
    haproxy:
            image: haproxy:latest
            ports:
                    - "80:80"
            volumes:
                    - ./haproxy.cfg:/usr/local/etc/haproxy/haproxy.cfg
```

Crear el archivo de configuración de HAProxy:

```
nano haproxy.cfg
```

Codigo:

```
global
    log stdout format raw local0
defaults
    log global
    mode http
    timeout connect 5000ms
    timeout client 50000ms
    timeout server 50000ms
frontend http_front
    bind *:80
    default_backend http_back
backend http_back
    balance roundrobin
    server app1 app:5000 check
```

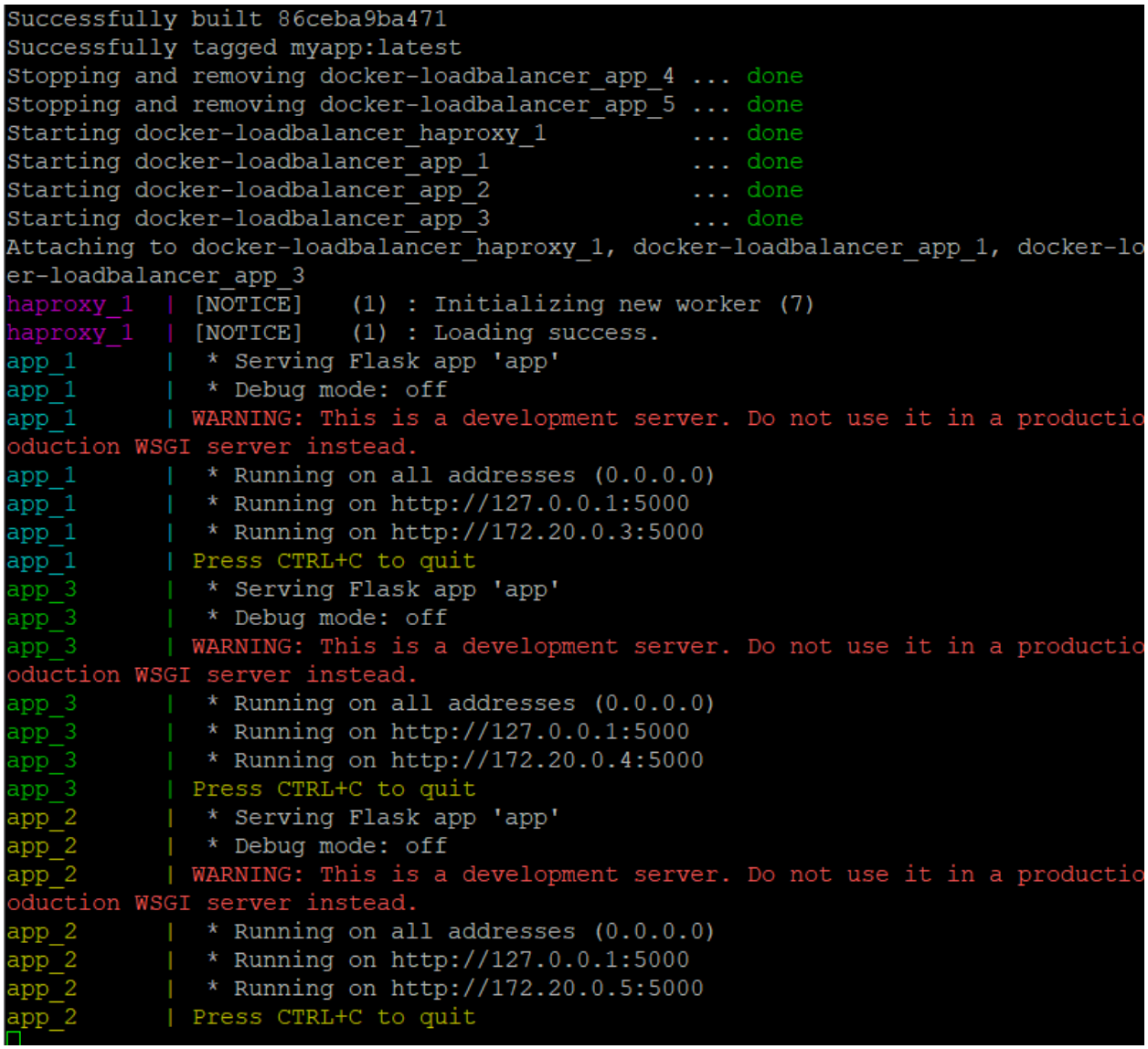
```
server app2 app:5000 check

server app3 app:5000 check
```

Ejecutar los contenedores en Ubuntu

Construir y ejecutar con Docker Compose:

Levantar todos los servicios:

```
docker-compose up --build
```



Verifica que todo está funcionando: curl http://localhost

```
jchinchilla@jchinchillaserver:~/docker-loadbalancer$ curl http://localhost
Hello from container !jchinchilla@jchinchillaserver:~/docker-loadbalancer$
```

Escalar réplicas manualmente:

docker-compose up --scale app=5

---

4. Realizar pruebas de rendimiento

Instalar Apache Benchmark o Siege:

Instalar Siege
sudo apt install siege

Ejecutar una prueba:
siege -c10 -r10 http://localhost/

Simula 100 solicitudes con 10 solicitudes concurrentes.

```
New configuration template added to /home/jchinchilla/.siege
Run siege -C to view the current settings in that file

{       "transactions":                        100,
        "availability":                      100.00,
        "elapsed_time":                        0.34,
        "data_transferred":                    0.00,
        "response_time":                       0.03,
        "transaction_rate":                  294.12,
        "throughput":                          0.01,
        "concurrency":                         9.44,
        "successful_transactions":             100,
        "failed_transactions":                   0,
        "longest_transaction":                 0.20,
        "shortest_transaction":                0.00
}
jchinchilla@jchinchillaserver:~/docker-loadbalancer$
```

Resultados obtenidos:

- Transacciones: 100
- Disponibilidad: 100%
- Tiempo de respuesta promedio: 0.03 segundos
- Transacciones fallidas: 0

Propuestas de Mejora

1. **Configurar un Servidor WSGI para Producción:** El servidor Flask utilizado es para desarrollo. Implementar un servidor WSGI como Gunicorn o uWSGI.
2. **Automatizar Escalado:** Configurar un sistema de orquestación como Kubernetes o Docker Swarm para gestionar automáticamente el número de réplicas según la carga.
3. **Mejorar la Seguridad de HAProxy:**
    - Configurar HTTPS utilizando certificados SSL.
    - Restringir accesos no autorizados mediante reglas ACL en HAProxy.
4. **Monitoreo y Logging:**
    - Implementar herramientas como Prometheus y Grafana para monitorear el rendimiento.
    - Configurar logs detallados en HAProxy para analizar tráfico y detectar posibles problemas.
5. **Pruebas de Carga Adicionales:**
    - Realizar pruebas más extensivas con herramientas como Apache JMeter.
    - Analizar el rendimiento bajo diferentes niveles de concurrencia.

Con esta configuración, el entorno está preparado para simular un sistema con balanceo de carga utilizando Docker y HAProxy. Las mejoras propuestas permitirán llevar el proyecto hacia un entorno más robusto y profesional.