Análisis de Caso 1 - Javier Guzmán Tiffer

1. Contexto y Justificación

La Universidad Latina de Costa Rica está en proceso de implementar una plataforma moderna para alojar sus sitios web institucionales, empleando tecnologías como Java, NodeJS, Drupal y WordPress. La solución propuesta se basa en contenedores Docker, balanceadores de carga HAProxy y un flujo de integración y entrega continua (CI/CD). Este enfoque busca garantizar alta disponibilidad, escalabilidad y seguridad, maximizando el aprovechamiento de recursos, aumentando la resiliencia y asegurando que los servicios respondan de manera eficiente a variaciones en la demanda.

2. Objetivo del Proyecto

Diseñar, implementar y simular una plataforma web institucional que incorpore principios de contenedorización, balanceo de carga, tolerancia a fallos, uso de HTTPS y buenas prácticas de seguridad, con el fin de obtener un entorno robusto, seguro y escalable.

3. Desarrollo Técnico y Análisis

3.1 Simulación de un Servicio en Contenedor

Se elaboró un Dockerfile partiendo de la imagen base 'openjdk:17', copiando el archivo .jar de la aplicación y exponiendo el puerto 8080. Posteriormente, se ejecutó el contenedor de manera local para evaluar aspectos clave como el aislamiento de procesos, el manejo de red y la persistencia de datos.

Ejemplo de Dockerfile:
FROM openjdk:17
COPY app.jar /app.jar
EXPOSE 8080
CMD ["java", "-jar", "/app.jar"]

3.2 Balanceo de Carga con HAProxy

Se desplegaron dos instancias del contenedor Java y se configuró HAProxy para distribuir las solicitudes entrantes utilizando el método de balanceo round robin. La verificación del comportamiento se realizó mediante un navegador y la herramienta curl.

3.3 Alta Disponibilidad

Se simuló la caída de una de las instancias del servicio para confirmar que HAProxy mantenía la disponibilidad de la aplicación. De forma opcional, se empleó Docker Swarm para replicar automáticamente las instancias y garantizar la continuidad del servicio.

3.4 Seguridad

Se configuró HTTPS utilizando certificados auto-firmados o gestionados por Let's Encrypt. Se evaluaron los riesgos asociados al uso de HTTP sin cifrar y se implementaron medidas adicionales como el control de acceso por roles y el uso de firewalls.

3.5 Escalabilidad y Mantenimiento

Se realizó una comparación con sitios institucionales reales desarrollados en Drupal y WordPress, identificando similitudes y diferencias. Se propuso un enfoque de escalado horizontal mediante Docker Swarm y el uso de volúmenes NFS para asegurar la persistencia de los datos.

4. Resultados Esperados

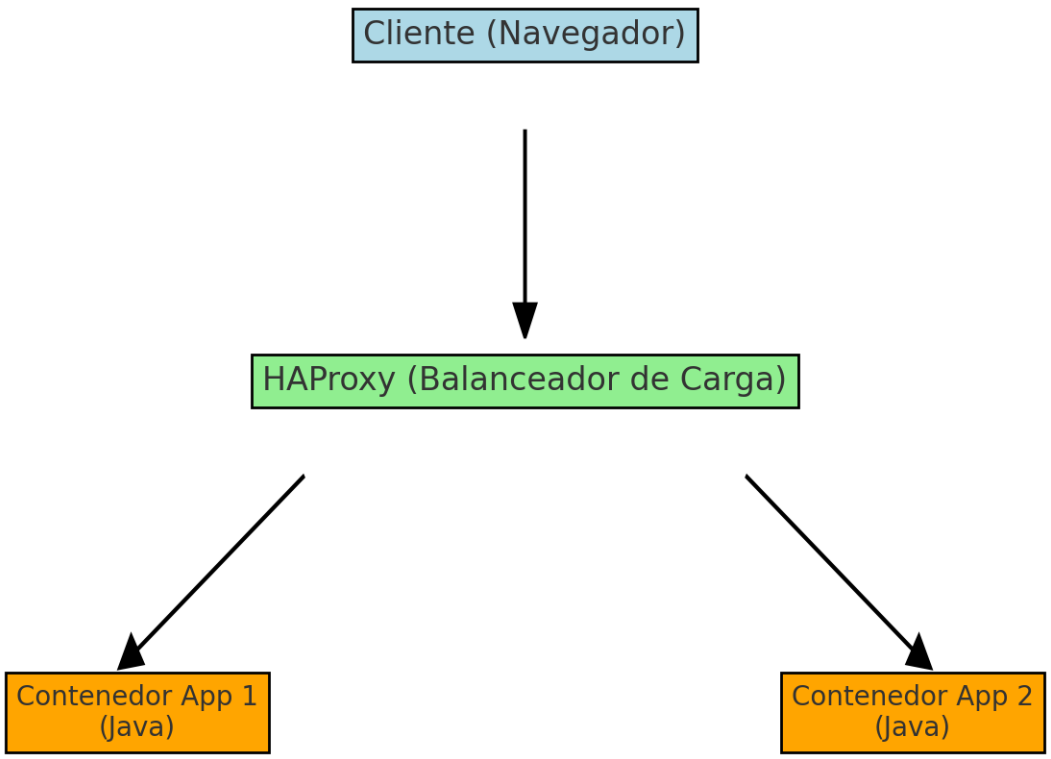
La plataforma resultante deberá ser capaz de distribuir el tráfico entre varias instancias, recuperarse automáticamente ante fallos, garantizar conexiones seguras mediante HTTPS y adaptarse a incrementos de demanda a través de escalado horizontal.

5. Objetivos de Aprendizaje Alcanzados

- Entender la asignación y gestión de puertos en entornos con contenedores.
- Implementar balanceadores de carga en arquitecturas distribuidas.
- Aplicar medidas de seguridad modernas en aplicaciones web.
- Ejecutar pruebas de rendimiento y monitoreo en sistemas distribuidos.

6. Entregables

- Diagrama



- **Informe Tecnico**

1. Introducción

Este informe describe la implementación de una arquitectura de plataforma web institucional

utilizando contenedores Docker, balanceador de carga HAProxy, y prácticas de seguridad y alta disponibilidad.

2. Arquitectura Implementada

- Cliente accede a la plataforma mediante un navegador web (HTTP/HTTPS).

- Peticiones son recibidas por un balanceador de carga HAProxy.

- HAProxy distribuye el tráfico entre múltiples instancias de la aplicación Java (contenedores Docker).

- Comunicación interna se realiza en una red Docker privada.

- Certificados SSL implementados para cifrado HTTPS.

3. Configuraciones Clave

- Dockerfile para la aplicación Java:

```
FROM openjdk:17

COPY app.jar /app.jar

EXPOSE 8080

CMD ["java", "-jar", "/app.jar"]
```

- Configuración de HAProxy (haproxy.cfg):

```
balance roundrobin

server app1 app1:8080 check

server app2 app2:8080 check
```

- docker-compose.yml define servicios para HAProxy y las dos instancias de aplicación.

4. Resultados Obtenidos

- Balanceo de carga efectivo entre contenedores.

- Alta disponibilidad: si una instancia falla, el servicio sigue operativo.

- Conexiones seguras mediante HTTPS.

- Arquitectura modular que facilita escalabilidad horizontal.

5. Conclusiones

La solución implementada permite un servicio web robusto, escalable y seguro,

adecuado para entornos de alta demanda.

- **Archivos de configuración utilizados (Dockerfile, haproxy.cfg, docker-compose.yml).**

docker-compose.yml

```
version: "3.8"
services:
 haproxy:
  image: haproxy:2.7
  container_name: haproxy
  ports:
   - "80:80"
   - "443:443"
  volumes:
   - ./haproxy.cfg:/usr/local/etc/haproxy/haproxy.cfg:ro
   - ./certs:/etc/haproxy/certs:ro
  depends_on:
   - app1
   - app2

 app1:
  build: ./app
  container_name: app1
  ports:
   - "8081:8080"

 app2:
  build: ./app
  container_name: app2
  ports:
   - "8082:8080"

networks:
 default:
  driver: bridge
```

haproxy.cfg

```
global
   log stdout format raw local0

defaults
   log     global
   option  httplog
   option  dontlognull
   timeout connect 5000
   timeout client  50000
   timeout server  50000

frontend http_front
   bind *:80
   default_backend http_back

frontend https_front
   bind *:443 ssl crt /etc/haproxy/certs/cert.pem
   default_backend http_back

backend http_back
   balance roundrobin
   server app1 app1:8080 check
   server app2 app2:8080 check
```

Dockerfile de la Aplicación

```
FROM openjdk:17
COPY app.jar /app.jar
EXPOSE 8080
CMD ["java", "-jar", "/app.jar"]
```