# Análisis de Caso: Plataforma Web Institucional Escalable con Contenedores

## 1. Contexto y Justificación

La Universidad Latina de Costa Rica está actualizando su infraestructura tecnológica para alojar sus sitios web institucionales utilizando plataformas como Java, NodeJS, Drupal y WordPress. El propósito es asegurar alta disponibilidad, robustez y escalabilidad a través de contenedores Docker, balanceo de carga mediante HAProxy y procesos de automatización CI/CD. Esta nueva arquitectura permite un uso más eficiente de los recursos, incrementa la resiliencia del sistema y facilita la gestión de picos de tráfico de manera efectiva.2. Objetivo del Proyecto

Desarrollar y simular una plataforma web institucional que garantice escalabilidad, alta disponibilidad y seguridad, utilizando contenerización de servicios, balanceo de carga, simulación de fallos, configuración de HTTPS y evaluación del rendimiento.3. Desarrollo Técnico y Análisis

### 3.1. Simulación de un Servicio en Contenedor

Se construye un Dockerfile basado en la imagen openjdk:17. La aplicación en formato .jar se copia al contenedor y se expone el puerto 8080. Posteriormente, se ejecuta localmente para analizar el aislamiento de procesos, red y persistencia dentro del contenedor.Ejemplo de Dockerfile:  
FROM openjdk:17

COPY app.jar /app.jar

EXPOSE 8080

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

### 3.2. Implementación de Balanceo de Carga con HAProxy

Se despliegan dos instancias del contenedor Java y se configura HAProxy para distribuir el tráfico entre ellas, utilizando un esquema round robin. Se verifica el correcto balanceo de solicitudes mediante un navegador o herramientas como curl.e levantan dos instancias del contenedor Java y se configura HAProxy para distribuir el tráfico. Se utiliza balanceo round robin y se verifica con navegador o curl.

### 3.3. Pruebas de Alta Disponibilidad

Se simula la caída de una instancia para verificar que HAProxy mantenga el servicio activo. Opcionalmente se utiliza Docker Swarm para replicación automática.

### 3.4. Seguridad de la plataforma

Se implementa HTTPS mediante un certificado auto-firmado o Let's Encrypt. Se analizan los riesgos de usar HTTP plano y se aplican medidas complementarias como firewalls y control de acceso por roles para proteger la aplicación.

### 3.5. Escalabilidad y Mantenimiento

Se comparan las características del entorno simulado con sitios reales. Se plantea la escalabilidad horizontal utilizando Docker Swarm y se emplean volúmenes NFS para garantizar la persistencia de datos.

## 4. Resultados Esperados

* Plataforma capaz de manejar tráfico distribuido.
* Recuperación automática ante fallos.
* Conexiones seguras mediante HTTPS.
* Preparada para escalar horizontalmente de manera eficiente.

## 5. Objetivos de Aprendizaje Alcanzados

• Comprender la gestión de puertos en entornos de contenedores.

• Implementar balanceadores de carga en arquitecturas distribuidas.

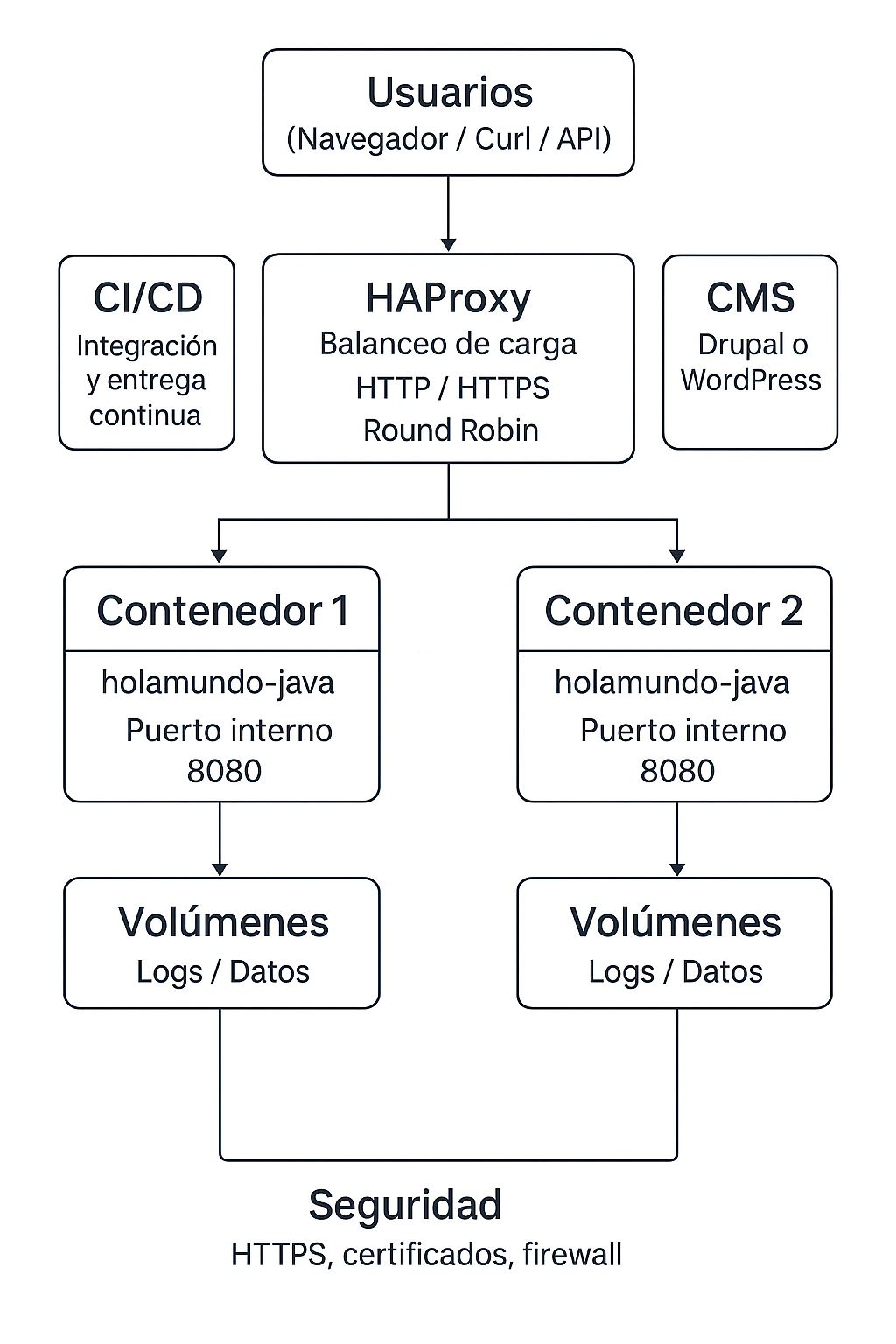
• Aplicar medidas de seguridad en aplicaciones web modernas.

• Realizar monitoreo y pruebas de rendimiento de la plataforma.

## 6. Entregables

* Informe técnico detallando configuraciones y resultados.
* Capturas de pruebas de funcionamiento.
* Diagramas de arquitectura de la plataforma.
* Archivos de configuración: Dockerfile, haproxy.cfg, docker-compose.yml.

## 7. Diagrama de Arquitectura



## 8. Archivos de Configuración

### 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 para la App

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