**Universidad Latina de Costa Rica**

**BISOF-18 – Sistemas Operativos II**

**Análisis de caso #1**

**Plataforma Web Institucional Escalable con Contenedores**

**Estudiante: Jose Alberto Murillo Sánchez**

1. Contexto actualizado

La Universidad Latina está desplegando una plataforma moderna para hospedar sitios institucionales en Java, Node.js, Drupal y WordPress, sobre contenedores Docker. El tráfico se distribuye con HAProxy y los despliegues se automatizan con CI/CD. El objetivo es lograr alta disponibilidad (HA), seguridad y escalabilidad horizontal, manteniendo costos y complejidad controlados.

2. Objetivo del caso

Diseñar e implementar una plataforma web basada en Docker con HAProxy y CI/CD, y simular el comportamiento de una aplicación web básica para comprender la integración entre sistemas operativos, redes y arquitectura distribuida.

**Referencia sugerida**: *HAProxy on Docker Swarm: Load Balancing and DNS Service Discovery*.  
**Herramienta sugerida para pruebas**: **Apache JMeter**.

3. Actividades y análisis técnicos

3.1 Simulación de un servicio en contenedor

**Objetivo:** containerizar un servidor Java “Hola mundo”, ejecutarlo localmente y analizar aislamiento.

#### **3.1.1 Código Java mínimo (HttpServer embebido)**

Crea HelloServer.java:

import com.sun.net.httpserver.HttpServer;  
import com.sun.net.httpserver.HttpHandler;  
import com.sun.net.httpserver.HttpExchange;  
import java.io.IOException;  
import java.io.OutputStream;  
import java.net.InetSocketAddress;  
  
public class HelloServer {  
 public static void main(String[] args) throws Exception {  
 int port = 8080;  
 HttpServer server = HttpServer.create(new InetSocketAddress(port), 0);  
 server.createContext("/", new RootHandler());  
 server.setExecutor(null);  
 System.out.println("Server running on http://0.0.0.0:" + port);  
 server.start();  
 }  
 static class RootHandler implements HttpHandler {  
 public void handle(HttpExchange exchange) throws IOException {  
 String response = "Hola mundo desde Java en Docker";  
 exchange.sendResponseHeaders(200, response.getBytes().length);  
 try (OutputStream os = exchange.getResponseBody()) { os.write(response.getBytes()); }  
 }  
 }  
}

#### **3.1.2 Dockerfile básico (multi-stage)**

Crea Dockerfile:

# Etapa de build  
FROM eclipse-temurin:17-jdk-alpine AS build  
WORKDIR /app  
COPY HelloServer.java .  
RUN javac HelloServer.java  
  
# Etapa de runtime (liviana)  
FROM eclipse-temurin:17-jre-alpine  
WORKDIR /srv  
COPY --from=build /app/HelloServer.class .  
EXPOSE 8080  
# Ejecutar como usuario no root por seguridad  
RUN adduser -D -H app && chown -R app:app /srv  
USER app  
CMD ["java","HelloServer"]

#### **3.1.3 Construcción y ejecución**

docker build -t hola-java:1.0 .  
docker network create webnet || true  
# Instancia 1  
docker run -d --name app1 --network webnet -p 8081:8080 hola-java:1.0  
# Prueba local  
curl -s http://localhost:8081

#### **3.1.4 Evaluación de aislamiento de procesos, red y persistencia**

* **Procesos/Namespaces:**
* docker exec app1 ps aux  
  docker exec app1 cat /proc/1/cgroup  
  docker inspect app1 --format '{{.State.Pid}}' # PID en host
* **Red:**
* docker inspect app1 --format '{{.NetworkSettings.Networks.webnet.IPAddress}}'  
  docker network inspect webnet | jq '.[0].Containers'
* **Persistencia (opcional):** montar volumen para logs o assets.
* docker run -d --name app1 --network webnet -p 8081:8080 \  
   -v $(pwd)/logs:/var/log/app hola-java:1.0

3.2 Balanceo de carga con HAProxy

**Objetivo:** exponer 2 instancias de la app Java y distribuir tráfico.

#### **3.2.1 Levantar segunda instancia**

docker run -d --name app2 --network webnet -p 8082:8080 hola-java:1.0

#### **3.2.2 Configuración mínima de HAProxy**

Crea haproxy.cfg:

global  
 log stdout format raw daemon  
 maxconn 2048  
  
defaults  
 log global  
 mode http  
 option httplog  
 option dontlognull  
 timeout connect 5s  
 timeout client 30s  
 timeout server 30s  
  
frontend http\_in  
 bind \*:80  
 # Redirección a HTTPS si se configura TLS (ver sección 3.4)  
 # http-request redirect scheme https unless { ssl\_fc }  
 default\_backend apps  
  
backend apps  
 balance roundrobin  
 option httpchk GET /  
 server app1 app1:8080 check  
 server app2 app2:8080 check  
  
listen stats  
 bind \*:8404  
 stats enable  
 stats uri /  
 stats refresh 5s

#### **3.2.3 HAProxy en contenedor**

docker run -d --name hap --network webnet -p 80:80 -p 8404:8404 \  
 -v $(pwd)/haproxy.cfg:/usr/local/etc/haproxy/haproxy.cfg:ro \  
 --add-host app1:$(docker inspect -f '{{range.NetworkSettings.Networks}}{{.IPAddress}}{{end}}' app1) \  
 --add-host app2:$(docker inspect -f '{{range.NetworkSettings.Networks}}{{.IPAddress}}{{end}}' app2) \  
 haproxy:2.9

Nota: En entornos Docker nativos, el nombre de contenedor suele resolverse por DNS dentro de la misma red. El --add-host es una alternativa si se usa host.docker.internal u otros escenarios.

#### **3.2.4 Verificación**

# Alternar respuestas entre app1 y app2  
for i in {1..6}; do curl -s http://localhost/; echo; done  
# Panel de estadísticas  
xdg-open http://localhost:8404 # o abra en navegador

#### **3.2.5 Análisis de tipos de balanceo**

* **roundrobin (por defecto):** reparte solicitudes de forma equitativa.
* **leastconn:** envía tráfico a la instancia con menos conexiones activas (útil para cargas desiguales).
* **source/ip-hash:** fija clientes a una instancia (sesiones pegajosas).  
  Configuración de ejemplo:

backend apps  
 balance leastconn  
 cookie SRV insert indirect nocache  
 server app1 app1:8080 check cookie A  
 server app2 app2:8080 check cookie B

3.3 Alta disponibilidad (HA)

**Objetivo:** tolerar fallos sin perder el servicio.

#### **3.3.1 Simular falla**

docker stop app2  
curl -s http://localhost/

**Esperado:** HAProxy mantiene respuestas sirviendo desde app1. En stats el backend app2 aparece DOWN.

#### **3.3.2 Replicación con Docker Swarm (opcional)**

# Inicializar Swarm (en laboratorio local)  
docker swarm init  
# Crear red overlay para servicios  
docker network create -d overlay webmesh  
# Publicar servicio con réplicas  
docker service create --name hola-svc --replicas 3 --network webmesh -p 8080:8080 hola-java:1.0  
# Escalar  
docker service scale hola-svc=5

En Swarm, el DNS interno resuelve tasks.<service> para descubrimiento; HAProxy puede apuntar al nombre del servicio o a un resolver DNS para auto-descubrir endpoints.

3.4 Seguridad

**Objetivo:** exponer el sitio por HTTPS, minimizar superficie de ataque y aplicar controles.

#### **3.4.1 TLS con certificado autofirmado**

mkdir -p certs && cd certs  
openssl req -x509 -nodes -days 365 -newkey rsa:2048 \  
 -keyout haproxy.key -out haproxy.crt -subj "/CN=example.local"  
cat haproxy.crt haproxy.key > haproxy.pem

haproxy.cfg (fragmento):

frontend https\_in  
 bind \*:443 ssl crt /usr/local/etc/haproxy/certs/haproxy.pem  
 default\_backend apps  
  
frontend http\_in  
 bind \*:80  
 http-request redirect scheme https unless { ssl\_fc }  
 default\_backend apps

Ejecutar:

docker run -d --name hap --network webnet -p 80:80 -p 443:443 \  
 -v $(pwd)/haproxy.cfg:/usr/local/etc/haproxy/haproxy.cfg:ro \  
 -v $(pwd)/certs:/usr/local/etc/haproxy/certs:ro \  
 haproxy:2.9

#### **3.4.2 Riesgos de HTTP plano**

* Robo de credenciales (sniffing), manipulación MITM, pérdida de integridad.
* Solución: **TLS**, **HSTS**, **redirección 80→443**, **TLS modernos**.

#### **3.4.3 Controles de acceso y firewalls**

* **Cortafuegos host (UFW/iptables):** permitir solo 80/443 (y 22 para administración).
* **Seguridad en contenedores:**
  + Ejecutar como **usuario no root** (ya configurado en Dockerfile).
  + Reducir **capabilities** (--cap-drop ALL y añadir solo las necesarias).
  + Políticas seccomp/apparmor, **read-only rootfs** cuando sea posible.
  + Variables de entorno/secretos gestionados desde el orquestador.

#### **3.4.4 Integración CI/CD (ejemplo con GitHub Actions)**

Archivo .github/workflows/build.yml (resumen):

name: build-and-push  
on: { push: { branches: [ "main" ] } }  
jobs:  
 docker:  
 runs-on: ubuntu-latest  
 steps:  
 - uses: actions/checkout@v4  
 - uses: docker/setup-buildx-action@v3  
 - uses: docker/login-action@v3  
 with:  
 registry: ghcr.io  
 username: ${{ github.actor }}  
 password: ${{ secrets.GITHUB\_TOKEN }}  
 - uses: docker/build-push-action@v6  
 with:  
 push: true  
 tags: ghcr.io/<org>/hola-java:latest

Despliegue: un job adicional puede conectarse por SSH a un nodo y ejecutar docker service update --image ghcr.io/<org>/hola-java:latest hola-svc.

3.5 Escalabilidad y mantenimiento

* **Comparativa con Drupal/WordPress:**
  + Separar PHP-FPM y servidor web (Nginx/Apache) en contenedores.
  + Almacenar wp-content/sites/default/files en volúmenes NFS/Gluster o S3 compatibles.
  + Base de datos gestionada externamente (MariaDB/Managed DB), con backups y replicación.
* **Crecimiento horizontal:** aumentar réplicas; usar HAProxy/Swarm/K8s con healthchecks y readiness.
* **Mantenimiento:** imágenes pequeñas, escaneo de vulnerabilidades, rotación de logs, monitorización (Prometheus/Grafana/HAProxy stats).

#### **3.5.1 docker-compose.yml (opcional, laboratorio local)**

version: "3.9"  
services:  
 app:  
 build: .  
 image: hola-java:1.0  
 networks: [ webnet ]  
 deploy:  
 replicas: 2  
 haproxy:  
 image: haproxy:2.9  
 ports:  
 - "80:80"  
 - "8404:8404"  
 volumes:  
 - ./haproxy.cfg:/usr/local/etc/haproxy/haproxy.cfg:ro  
 networks: [ webnet ]  
networks:  
 webnet: {}

4. Objetivos de aprendizaje ampliados

|  |  |  |
| --- | --- | --- |
| Categoría | Antes | Ahora |
| Sockets/HTTP | Conceptual | Servicio Java corriendo en contenedor, expuesto vía HAProxy/HTTPS |
| Puertos | Teoría | Mapeos reales Docker (p.ej., -p 8081:8080), reglas de firewall |
| Concurrencia | Threads | Balanceo entre múltiples instancias, controles de conexión |
| OS y procesos | Local | Namespaces, cgroups, PIDs aislados en contenedores |
| Seguridad | Conceptual | TLS, HSTS, roles, CI/CD seguro, backups y gestión de secretos |
| Disponibilidad | Un solo punto de fallo | HA real con HAProxy (y Keepalived/Swarm opcional) |
| Escalabilidad | No aplicable | Escalado horizontal con Swarm/orquestador |
| Persistencia | Inexistente | Volúmenes (NFS/Gluster), políticas de backup y restauración |

5. Entregables

1. **Informe técnico** con secciones de:
   * **Rendimiento:** resultados de JMeter (RPS, latencias P50/P90/P99, tasa de errores), incluyendo metodología.
   * **Escalabilidad:** pruebas al subir réplicas y análisis del impacto en throughput/latencias.
   * **Seguridad:** evidencia de HTTPS, redirección 80→443, usuario no root, firewall y controles en contenedores.
   * **Alta disponibilidad:** escenarios de fallo y comportamiento observado en HAProxy (stats, logs).
2. **Evidencias gráficas y configuraciones**:
   * Capturas de curl, navegador, panel http://localhost:8404, logs de HAProxy.
   * Fragmentos de Dockerfile, haproxy.cfg, docker-compose.yml, workflow de CI/CD.
   * Topologías/diagramas simples usados.
3. **Repositorio (GitHub/privado)** con:
   * Código fuente (HelloServer.java).
   * Archivos de infraestructura (Docker/HAProxy/CI-CD).
   * README con pasos de reproducción.