  
Facultad de Ingeniería en Software

Sistemas Operativos II

Despliegue Automatizado de un Servidor Web Seguro y Escalable en Entornos Linux Usando Docker como Infraestructura como Código:

Revisión e Implementación Aplicada

Estudiante:

Farid Alexander Moreno Fontana

Profesor:

Carlos Andrés Méndez Rodriguez

Fecha: 18 Abril 2025

Tabla de Contenidos

[Definición del Tema de Investigación 2](#_Toc195888008)

[Objetivo General 3](#_Toc195888009)

[Objetivos Específicos 3](#_Toc195888010)

[Metodología de Trabajo 3](#_Toc195888011)

[Herramientas y Recursos Necesarios 4](#_Toc195888012)

[Marco Teórico 4](#_Toc195888013)

[Implementación de experimentos o simulaciones basadas en el diseño metodológico 5](#_Toc195888014)

[Pasos para la implementación técnica 5](#_Toc195888015)

[1. Generación de certificados SSL auto-firmados: 5](#_Toc195888016)

[2. Estructura del proyecto: 5](#_Toc195888017)

[3. Archivo docker-compose.yml: 6](#_Toc195888018)

[4. Archivo deploy.sh (automatización del entorno): 7](#_Toc195888019)

[5. Configuraciones de Nginx (nginx.conf): 7](#_Toc195888020)

[6. Configuración de HAProxy (haproxy.cfg): 7](#_Toc195888021)

[Resultados preliminares 8](#_Toc195888022)

[Análisis de resultados y ajustes necesarios 8](#_Toc195888023)

[Referencias 9](#_Toc195888024)

# Definición del Tema de Investigación

Este proyecto de investigación busca explorar el uso de tecnologías de contenedores, particularmente Docker, como herramienta de Infrastructure as Code para el despliegue automatizado de servidores web seguros y escalables en sistemas Linux. La propuesta se enfocará en dos ejes principales: (1) una revisión bibliográfica de investigaciones actuales y buenas prácticas publicadas por instituciones académicas o desarrolladores en torno a este tipo de soluciones, y (2) el desarrollo de un proyecto aplicado de pequeña escala que permita evaluar aspectos técnicos y de rendimiento. El entorno desarrollado integrará servicios como servidores web (por ejemplo, Nginx o Apache), configuraciones de seguridad (certificados SSL, firewall), y balanceo de carga mediante herramientas como HAProxy. Con esta combinación se busca aportar conocimiento sobre cómo las soluciones basadas en contenedores pueden optimizar la implementación de servicios en entornos Linux, haciendo énfasis en escalabilidad, automatización y seguridad.

# Objetivo General

Investigar y analizar soluciones actuales de despliegue automatizado de servidores web seguros y escalables usando Docker como herramienta de *Infrastructure as Code*, complementando el estudio con el desarrollo de un prototipo aplicado que permita evaluar sus beneficios en entornos Linux.

# Objetivos Específicos

1. Revisar literatura científica y técnica sobre el uso de Docker y herramientas de automatización en la implementación de servicios web.
2. Identificar buenas prácticas de seguridad, escalabilidad y rendimiento aplicables a arquitecturas basadas en contenedores.
3. Diseñar y desarrollar un entorno automatizado de despliegue de servidor web con Docker, integrando elementos como firewall, certificados SSL y balanceo de carga.
4. Evaluar el desempeño del entorno mediante métricas como tiempo de respuesta, consumo de recursos y tolerancia a fallos.
5. Comparar los resultados obtenidos con estudios previos y proponer recomendaciones o mejoras potenciales.

# Metodología de Trabajo

Este proyecto seguirá una metodología de investigación aplicada, combinando una revisión bibliográfica con el desarrollo de un prototipo funcional. El trabajo se dividirá en dos fases principales:

1. Fase de investigación documental:  
   Se recopilarán y analizarán artículos académicos, publicaciones técnicas, blogs especializados y documentación oficial sobre el uso de Docker, balanceadores de carga, buenas prácticas de despliegue web y herramientas de Infrastructure as Code. El objetivo es construir un marco teórico sólido que respalde las decisiones de diseño e implementación.
2. Fase práctica de desarrollo:  
   Se diseñará y desarrollará un entorno automatizado de despliegue de un servidor web en Linux utilizando Docker y herramientas complementarias. Esta fase incluirá la configuración de servicios, scripts de automatización, medidas de seguridad y evaluación del rendimiento mediante pruebas controladas. Los resultados serán documentados y comparados con la literatura revisada.

Durante todo el proyecto se mantendrá un enfoque iterativo, ajustando el diseño y las herramientas conforme se obtenga nueva información o se detecten oportunidades de mejora.

# Herramientas y Recursos Necesarios

Para llevar a cabo el proyecto se utilizarán herramientas de software libre y recursos técnicos accesibles, entre ellos:

* Docker: Plataforma principal para la creación, despliegue y gestión de contenedores.
* Docker Compose: Para definir y orquestar múltiples servicios de forma automatizada.
* HAProxy: Como balanceador de carga de código abierto, para pruebas de escalabilidad y alta disponibilidad.
* Nginx o Apache: Servidor web para ser configurado dentro del contenedor.
* iptables / ufw: Herramientas de firewall para configurar medidas de seguridad a nivel de red.
* Linux (Ubuntu/Debian): Sistema operativo base para pruebas y desarrollo.
* Bash scripting: Para automatización de tareas y configuración del entorno.
* Visual Studio Code o cualquier editor compatible con YAML, Dockerfile y bash.
* Computadora personal con entorno Linux (nativo o virtualizado).

Adicionalmente, se utilizarán plataformas académicas y motores de búsqueda como IEEE Xplore, Google Scholar y ScienceDirect para la obtención de fuentes confiables.

# Marco Teórico

El uso de tecnologías de contenedores como Docker ha transformado la forma en que se despliegan y escalan aplicaciones en entornos Linux. Según Merkel (2014), Docker permite empaquetar aplicaciones con todas sus dependencias en contenedores ligeros, aislados y portables, lo que mejora la consistencia y la eficiencia del entorno de ejecución.

En cuanto a la seguridad y aislamiento de los contenedores, Soltesz et al. (2007) señalan que los contenedores ofrecen una separación efectiva de procesos, reduciendo los riesgos de interferencia entre servicios. No obstante, Shiri et al. (2019) destacan que se deben aplicar medidas adicionales como firewalls, control de tráfico y configuraciones seguras para mitigar vulnerabilidades propias del sistema operativo anfitrión.

Respecto al despliegue de servidores web, Apache y Nginx son ampliamente utilizados por su rendimiento y compatibilidad con múltiples entornos. De acuerdo con Netcraft (2023), ambos servidores siguen siendo los más utilizados en internet, y su integración con Docker permite una configuración ágil y reproducible.

HAProxy, por su parte, es una herramienta clave en esquemas de balanceo de carga y alta disponibilidad. Según su documentación oficial (HAProxy Technologies, 2021), permite distribuir el tráfico entre múltiples servidores con base en diferentes algoritmos como round-robin y menor número de conexiones, además de incluir verificaciones de estado (health checks) y protección contra sobrecargas.

Finalmente, la automatización del despliegue mediante herramientas de Infrastructure as Code (IaC), como Docker Compose, facilita la gestión reproducible de entornos complejos. Hashimoto (2013) y Turnbull (2014) coinciden en que IaC permite una mayor agilidad en la implementación y mantenimiento de servicios, reduciendo errores humanos y mejorando la trazabilidad de cambios

# **Implementación de experimentos o simulaciones basadas en el diseño metodológico**

Como parte del segundo eje de esta investigación, se procedió con la construcción y validación de un entorno automatizado que despliega un clúster de servicios utilizando tecnologías de contenedores. Este entorno fue diseñado para simular una arquitectura web básica pero funcional, que incluye:

* Un **servidor Nginx** como proxy inverso y servidor estático.
* Un backend simulado o página de prueba servida por **Apache**.
* Un **balanceador de carga con HAProxy** configurado para distribuir el tráfico entrante hacia múltiples instancias de backend.
* Implementación de **certificados SSL auto-firmados** para habilitar HTTPS.
* Aplicación de **reglas de firewall** simples utilizando iptables dentro del contenedor o redireccionamientos seguros a través de puertos específicos.
* Automatización del despliegue completo mediante un **script Bash** y archivo docker-compose.yml.

## **Pasos para la implementación técnica**

A continuación se detallan los pasos realizados para desplegar de forma automatizada el entorno complet

### **Generación de certificados SSL auto-firmados**:

El objetivo de este paso fue permitir conexiones HTTPS seguras hacia el servidor Nginx. Aunque en entornos de producción se utilizan certificados emitidos por autoridades confiables como Let's Encrypt, en este proyecto se optó por certificados auto-firmados para simular un entorno seguro sin depender de terceros.

El siguiente comando genera una clave privada (server.key) y un certificado público (server.crt) válido por un año:

*mkdir certs*

*openssl req -x509 -nodes -days 365 -newkey rsa:2048 \*

*-keyout certs/server.key -out certs/server.crt \*

*-subj "/C=CR/ST=SanJose/L=SanJose/O=ProyectoDocker/CN=localhost"*

### **Estructura del proyecto**:

Se organizó el entorno en carpetas independientes para cada componente (Apache, Nginx, HAProxy), facilitando su mantenimiento y reutilización. La estructura general fue la siguiente:

📁 /  
├── **apache/**: contiene el Dockerfile que define la imagen del backend.  
├── **haproxy/**: contiene el archivo haproxy.cfg con la configuración del balanceador y su Dockerfile.  
├── **nginx/**: contiene el nginx.conf y Dockerfile para configurar el proxy inverso.  
├── **certs/**: almacena los certificados SSL generados manualmente.  
├── **docker-compose.yml**: orquesta todos los servicios.  
└── **deploy.sh**: script que automatiza el despliegue completo del sistema.

Esta separación en carpetas ayuda a mantener el proyecto limpio y modular, siguiendo buenas prácticas de infraestructura como código.

### **Archivo docker-compose.yml**:

Este archivo define todos los servicios que serán desplegados en contenedores. Se configuraron cuatro servicios:

* web: corre Nginx con HTTPS habilitado.
* apache1 y apache2: dos instancias de backend para distribuir carga.
* haproxy: distribuye las solicitudes entrantes entre los Apache.

Configuración del servicio web:

*version: '3.8'*

*services:*

*web:*

*build: ./nginx*

*ports:*

*- "80:80"*

*- "443:443"*

*volumes:*

*- ./certs:/etc/nginx/certs*

*networks:*

*- webnet*

*apache1:*

*build: ./apache*

*networks:*

*- webnet*

*apache2:*

*build: ./apache*

*networks:*

*- webnet*

*haproxy:*

*build: ./haproxy*

*ports:*

*- "8080:80"*

*networks:*

*- webnet*

### **Archivo deploy.sh** (automatización del entorno):

Este script Bash simplifica la ejecución del entorno. Primero verifica que los puertos necesarios no estén en uso. Si están libres, construye las imágenes y levanta los servicios en segundo plano.

*#!/bin/bash*

*echo "[+] Verificando puertos..."*

*for port in 80 443 8080; do*

*if lsof -i:$port; then*

*echo "[!] Puerto $port en uso. Abortando."*

*exit 1*

*fi*

*done*

*echo "[+] Construyendo contenedores..."*

*docker-compose up --build -d*

*echo "[+] Servicios desplegados. Verifique en* [*https://localhost*](https://localhost)*"*

Esto garantiza que el entorno pueda desplegarse desde cero sin intervención manual, alineado con los principios de automatización y reproducibilidad.

### **Configuraciones de Nginx (nginx.conf)**:

El servidor Nginx está configurado como ”proxy inverso”. Su función es recibir solicitudes externas y redirigirlas internamente hacia el balanceador HAProxy.

*server {*

*listen 443 ssl;*

*server\_name localhost;*

*ssl\_certificate /etc/nginx/certs/server.crt;*

*ssl\_certificate\_key /etc/nginx/certs/server.key;*

*location / {*

*proxy\_pass http://haproxy:80;*

*}*

*}*

### **Configuración de HAProxy (haproxy.cfg)**:

HAProxy se configuró con un esquema de balanceo de carga tipo round-robin, que distribuye equitativamente el tráfico entre las instancias apache1 y apache2. También se configuraron health checks para detectar si alguna instancia falla.

global

maxconn 2048

defaults

mode http

timeout connect 5000ms

timeout client 50000ms

timeout server 50000ms

frontend http-in

bind \*:80

default\_backend apache-backend

backend apache-backend

balance roundrobin

server apache1 apache1:80 check

server apache2 apache2:80 check

## Despliegue Automatizado del Entorno

Una vez definido el entorno, servicios involucrados, configuraciones de seguridad, estructura del proyecto y requisitos técnicos, se consolidó todo en un único script automatizado con el objetivo de garantizar portabilidad, reproducibilidad y facilidad de uso.

El script realiza las siguientes acciones clave:

* Verifica e instala Docker y Docker Compose si no están presentes.
* Crea la estructura de carpetas necesarias (certs, nginx, apache, haproxy).
* Genera archivos de configuración y Dockerfiles si no existen.
* Crea certificados SSL auto-firmados para habilitar conexiones seguras.
* Verifica que los puertos requeridos estén disponibles.
* Despliega todo el entorno usando docker compose up.

Este enfoque reduce los errores manuales, acelera el despliegue y permite que el sistema pueda replicarse fácilmente en nuevos entornos.

## 

A continuación, se presenta el script utilizado:

*#!/bin/bash*

*set -e*

*echo "Verificando e instalando dependencias..."*

*# Instalar Docker si no está*

*if ! command -v docker &> /dev/null; then*

*echo " Instalando Docker..."*

*sudo apt update*

*sudo apt install -y docker.io*

*sudo systemctl enable docker*

*sudo systemctl start docker*

*sudo usermod -aG docker $USER*

*else*

*echo "Docker ya instalado."*

*fi*

*# Instalar Docker Compose si no está*

*if ! command -v docker-compose &> /dev/null; then*

*echo "Instalando Docker Compose..."*

*sudo apt install -y docker-compose*

*else*

*echo "Docker Compose ya instalado."*

*fi*

*echo "Verificando estructura del proyecto..."*

*# Crear carpetas necesarias*

*mkdir -p certs nginx apache haproxy*

*# Crear docker-compose.yml si no existe*

*if [ ! -f docker-compose.yml ]; then*

*cat > docker-compose.yml <<EOF*

*version: '3.8'*

*services:*

*web:*

*build: ./nginx*

*ports:*

*- "80:80"*

*- "443:443"*

*volumes:*

*- ./certs:/etc/nginx/certs*

*networks:*

*- webnet*

*apache1:*

*build: ./apache*

*networks:*

*- webnet*

*apache2:*

*build: ./apache*

*networks:*

*- webnet*

*haproxy:*

*build: ./haproxy*

*ports:*

*- "8080:80"*

*networks:*

*- webnet*

*networks:*

*webnet:*

*EOF*

*echo "docker-compose.yml creado."*

*fi*

*# Crear certificados si no existen*

*if [ ! -f certs/server.crt ] || [ ! -f certs/server.key ]; then*

*echo "Generando certificados SSL auto-firmados..."*

*openssl req -x509 -nodes -days 365 -newkey rsa:2048 \*

*-keyout certs/server.key -out certs/server.crt \*

*-subj "/C=CR/ST=SanJose/L=SanJose/O=ProyectoDocker/CN=localhost"*

*fi*

*# Crear archivos de Nginx*

*if [ ! -f nginx/nginx.conf ]; then*

*cat > nginx/nginx.conf <<EOF*

*server {*

*listen 443 ssl;*

*server\_name localhost;*

*ssl\_certificate /etc/nginx/certs/server.crt;*

*ssl\_certificate\_key /etc/nginx/certs/server.key;*

*location / {*

*proxy\_pass http://haproxy:80;*

*}*

*}*

*EOF*

*fi*

*if [ ! -f nginx/Dockerfile ]; then*

*echo -e "FROM nginx:latest\nCOPY nginx.conf /etc/nginx/nginx.conf" > nginx/Dockerfile*

*fi*

*# Crear archivos de Apache*

*if [ ! -f apache/Dockerfile ]; then*

*echo -e "FROM httpd:latest\nRUN echo '<h1>Hola desde Apache</h1>' > /usr/local/apache2/htdocs/index.html" > apache/Dockerfile*

*fi*

*# Crear archivos de HAProxy*

*if [ ! -f haproxy/haproxy.cfg ]; then*

*cat > haproxy/haproxy.cfg <<EOF*

*global*

*maxconn 2048*

*defaults*

*mode http*

*timeout connect 5000ms*

*timeout client  50000ms*

*timeout server  50000ms*

*frontend http-in*

*bind \*:80*

*default\_backend apache-backend*

*backend apache-backend*

*balance roundrobin*

*server apache1 apache1:80 check*

*server apache2 apache2:80 check*

*EOF*

*fi*

*if [ ! -f haproxy/Dockerfile ]; then*

*echo -e "FROM haproxy:latest\nCOPY haproxy.cfg /usr/local/etc/haproxy/haproxy.cfg" > haproxy/Dockerfile*

*fi*

*# Verificar puertos*

*echo "Verificando puertos..."*

*for port in 80 443 8080; do*

*if lsof -i:$port &> /dev/null; then*

*echo "El puerto $port está en uso. Abortando."*

*exit 1*

*fi*

*done*

*# Desplegar contenedores*

*echo "Desplegando servicios..."*

*docker compose up -d --build*

*echo "Entorno desplegado correctamente."*

*echo "Accede desde https://localhost o la IP de tu VM (aceptá el certificado auto-firmado)."*

## **Análisis de resultados y ajustes necesarios**

Durante el desarrollo, integración y prueba del script de despliegue, se identificaron varias necesidades de ajuste para garantizar un funcionamiento más robusto, compatible y libre de errores.

Los principales cambios implementados fueron:

1. Verificación previa de Docker y Docker Compose:   
   Se incorporó una verificación automática para instalar estos componentes solo si no existen, evitando errores en sistemas ya configurados.
2. Creación condicional de archivos y carpetas:  
   Se añadieron validaciones para no sobrescribir archivos existentes (como docker-compose.yml, certificados SSL o configuraciones de Nginx/HAProxy).
3. Generación automática de certificados SSL:  
   Se automatizó la creación de un certificado auto-firmado si no existía, eliminando dependencias manuales.
4. Verificación de puertos en uso:   
   Se introdujo un chequeo previo de puertos (80, 443 y 8080) para detectar conflictos antes del despliegue y evitar fallos en docker-compose.
5. Corrección en despliegue final:  
   Se ajustó la instrucción de despliegue a docker compose up -d --build (con espacio, sin guion medio), compatible con versiones modernas de Docker.
6. Manejo de errores:  
   Se agregó set -e al inicio del script para que se detenga automáticamente ante cualquier error crítico, aumentando la confiabilidad.

Como resultado de estos ajustes, se da una reducción significativa de fallos durante el despliegue en entornos limpios, asegurando que la experiencia sea más estable desde el primer uso.

Además, se mejoró la experiencia del usuario gracias a la incorporación de mensajes más claros y una ejecución más fluida del script.

Estas mejoras también contribuyeron a aumentar la portabilidad y compatibilidad del entorno automatizado entre distintas versiones de Ubuntu Server, facilitando su implementación en diversos escenarios.

# Referencias

HAProxy Technologies. (2021). \*HAProxy Documentation\*. <https://www.haproxy.com/documentation/haproxy/>

Hashimoto, M. (2013). \*Vagrant: Up and Running\*. O'Reilly Media.

Merkel, D. (2014). Docker: Lightweight Linux Containers for Consistent Development and Deployment. \*Linux Journal\*, (239), 2.

Netcraft. (2023). Web Server Survey. <https://news.netcraft.com/archives/category/web-server-survey/>

Shiri, S., Bakhtiyari, K., & Dehghantanha, A. (2019). Container Security: Issues, Challenges, and the Road Ahead. \*IEEE Access\*, 7, 52976–52996. <https://doi.org/10.1109/ACCESS.2019.2912896>

Soltesz, S., Pötzl, H., Fiuczynski, M. E., Bavier, A., & Peterson, L. (2007). Container-based Operating System Virtualization: A Scalable, High-performance Alternative to Hypervisors. In \*Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007\* (pp. 275–287). ACM.

Turnbull, J. (2014). \*The Docker Book: Containerization is the new virtualization\*. James Turnbull.