Clúster Web con Balanceo de Carga

Implementación y evaluación práctica de un clúster Apache usando mod_proxy_balancer.

Juan Sebastian Ballesteros Sierra (2200405)

Juan Esteban Salazar Toro (2221681)

Jean Paul Ordoñez Ibarguen (2221275)

Contenido

01 · Sobre el proyecto

02 · mod_proxy_balancer

03 · Configuración

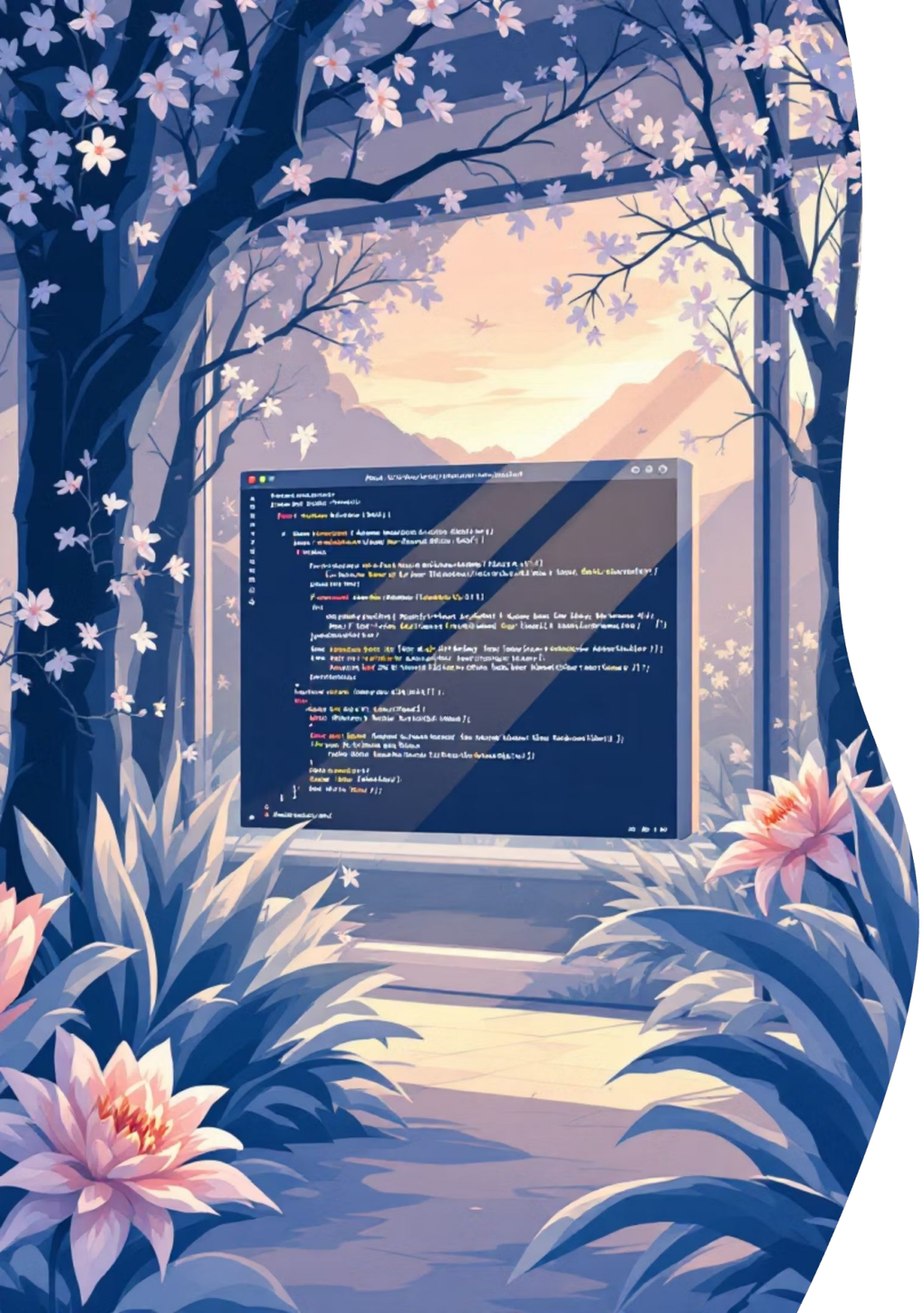
04 · Pruebas

05 · Resultados

06 · Conclusiones

Resumen del Proyecto

Despliegue de un servidor web con balanceo de carga usando Apache mod_proxy_balancer. Entorno virtualizado con Vagrant + VirtualBox: un balanceador (frontend) y dos backends. Objetivo: analizar desempeño bajo distintas cargas con Artillery.



¿Qué es mod_proxy_balancer?

Es un módulo de Apache que actúa como balanceador inverso: recibe solicitudes cliente y las distribuye entre varios servidores backend para optimizar recursos, reducir latencia y evitar sobrecargas.

Balanceador inverso

Centraliza tráfico y decide a qué backend enviar cada petición.

Optimización

Mejora uso de recursos y reduce latencia.

Disponibilidad

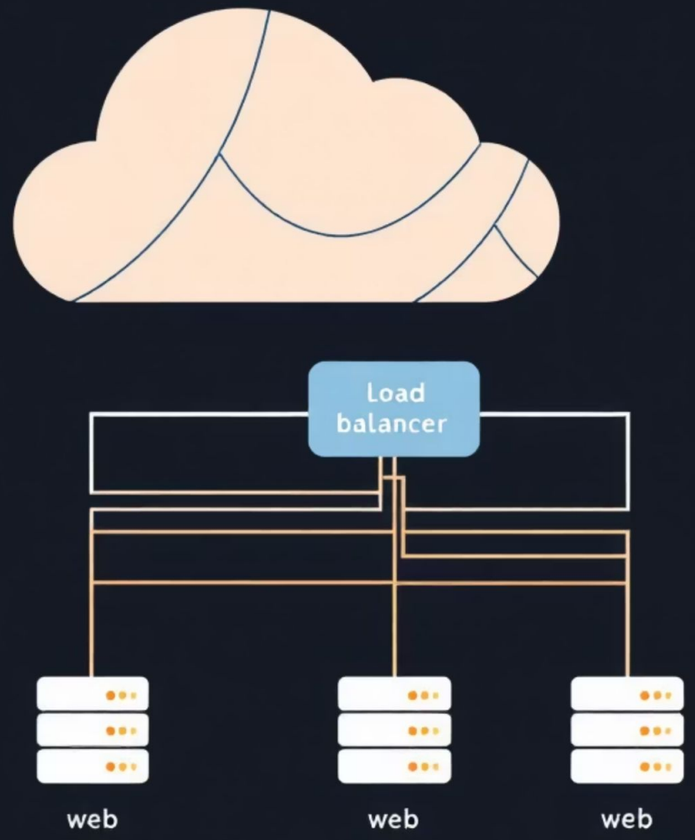
Evita fallos por sobrecarga en un único nodo.

Topología y Direcciones

Arquitectura: tres VMs Ubuntu teniendo 3 nodos.

Frontend (balanceador) en 192.168.50.30. Algoritmo de balanceo: byrequests.

Backends en 192.168.50.10 y 192.168.50.20. Paneles: balancer-manager y server-status habilitados.



Configuración Automatizada

Toda la infraestructura se creó con un Vagrantfile que instala y configura Apache en cada VM para garantizar reproducibilidad y despliegue rápido. Vagrantfile disponible en el repositorio del proyecto.

Repositorio: https://github.com/EGSans/ProyectoFinall_ServiciosTelematicos

Metodología de Pruebas

Pruebas de carga con Artillery: escenarios variados simulando entre 25 y 150 usuarios concurrentes. Objetivo: medir tasa de éxito, latencia, errores y comportamiento bajo picos y pruebas prolongadas.

1

heavy-api.yml

Peticiones rápidas a endpoints API.

2

heavy-db.yml

Consultas simuladas a base de datos.

3

soak-test.yml

Prueba prolongada de resistencia.

4

surge-test.yml

Picos repentinos de carga.

5

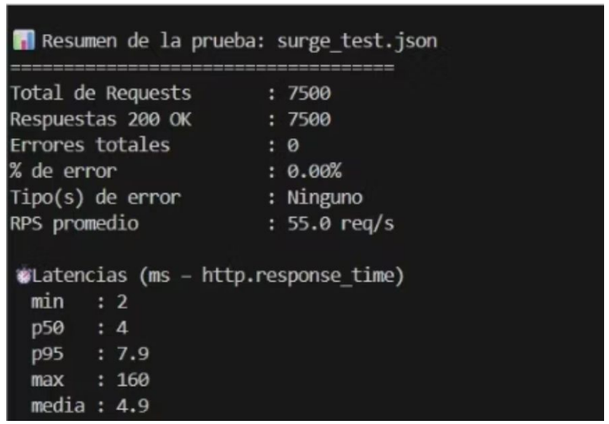
heavy-medio.yml

Carga moderada y constante.

Procesamiento de Resultados

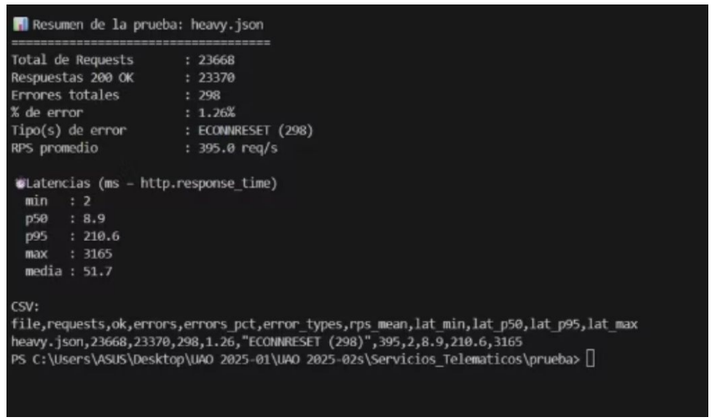
Artillery generó JSON con métricas por escenario. Un script en Node.js procesó esos archivos para resumir: número de peticiones, tasa de éxito, latencia y tipos de error, facilitando la interpretación.

Descripción grafica de resultados



Resultado 1

Surge_test.json

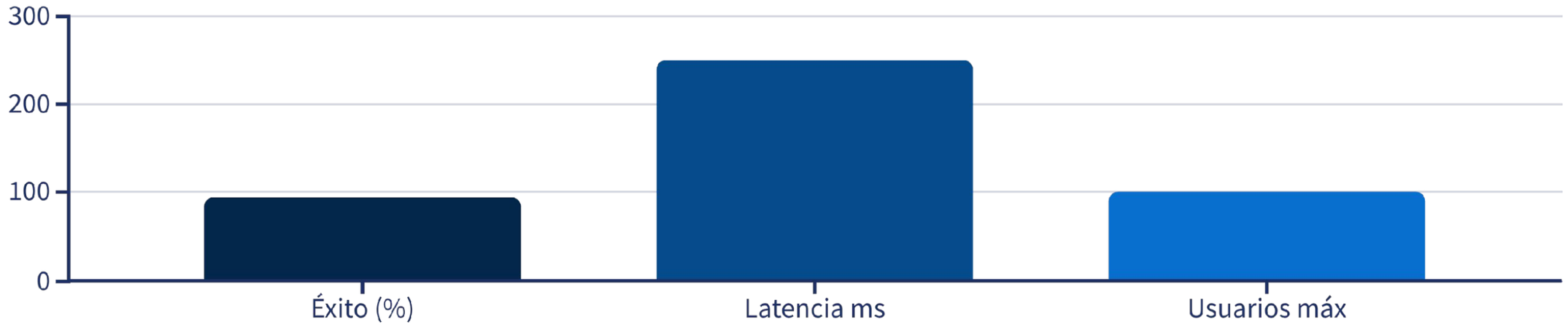


Resultado 2

Heavy.json

Resultados Clave

- Estabilidad hasta ~100 usuarios concurrentes.
- Tasa de éxito promedio >95% en cargas medias.
- Latencia promedio 180–320 ms en la mayoría de escenarios.
- En surge y soak tests la latencia aumentó por limitación de memoria (1 GB/VM).
- Byrequests demostró repartir peticiones de forma equitativa.



Conclusiones y Aprendizajes

Validamos que Apache mod_proxy_balancer, combinado con virtualización y automatización, mejora disponibilidad y distribución de peticiones. La configuración byrequests fue eficaz hasta 100 usuarios; la limitación principal fue memoria por VM. La práctica fortaleció competencias en infraestructura, configuración web y análisis de rendimiento.

Práctico

Demostración real del impacto del balanceo de carga.

Técnico

Automatización reproducible con Vagrant.

Medible

Análisis cuantitativo con Artillery y Node.js.

Muchas gracias!