



# 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

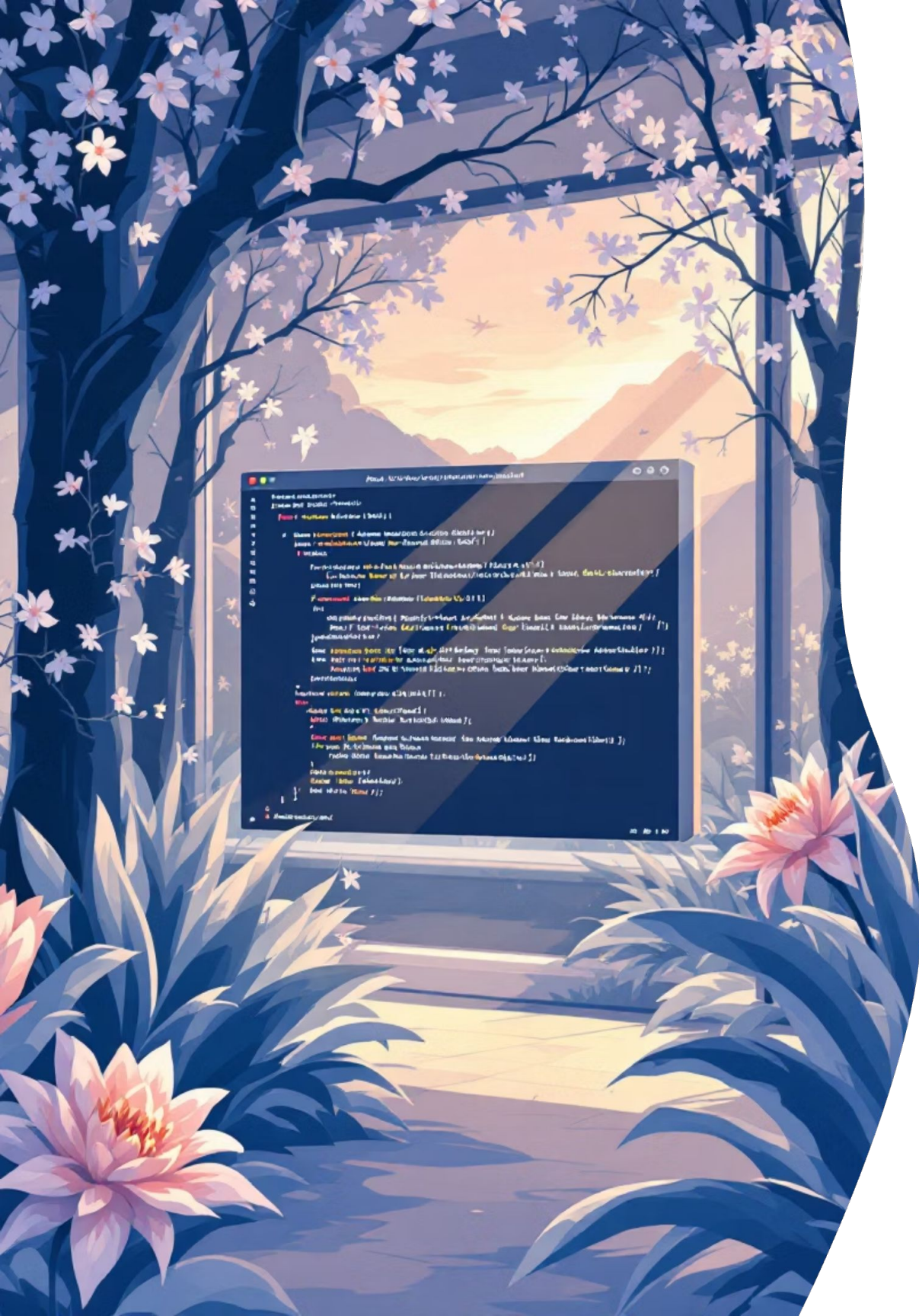
06 • Conclusiones



# 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

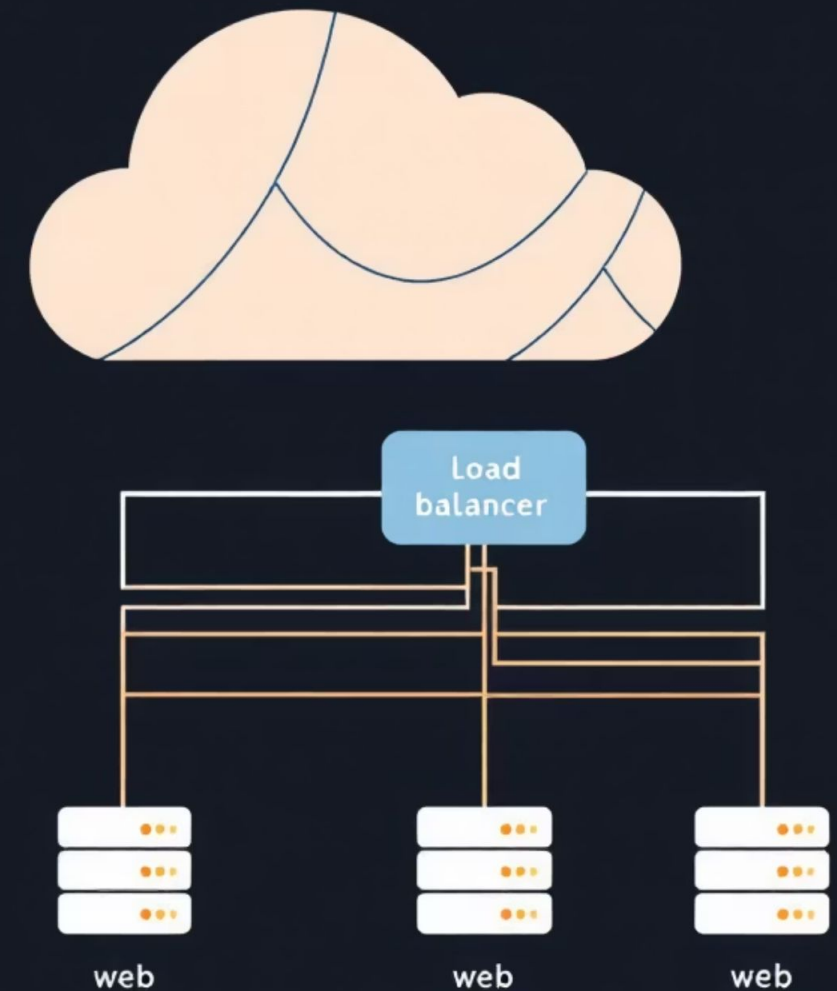
Evita fallos por sobrecarga en un único nodo.

# Topología y Direcciones

Arquitectura: tres VMs Ubuntu teniendo 3 nodos.

**VM3 (balanceador) en 192.168.50.30.** Balanceador de carga con Apache y el módulo mod\_proxy\_balancer activo

**VM1 y VM2 en 192.168.50.10 y 192.168.50.20.** Servidores web Apache con páginas HTML y funcionalidades REST.







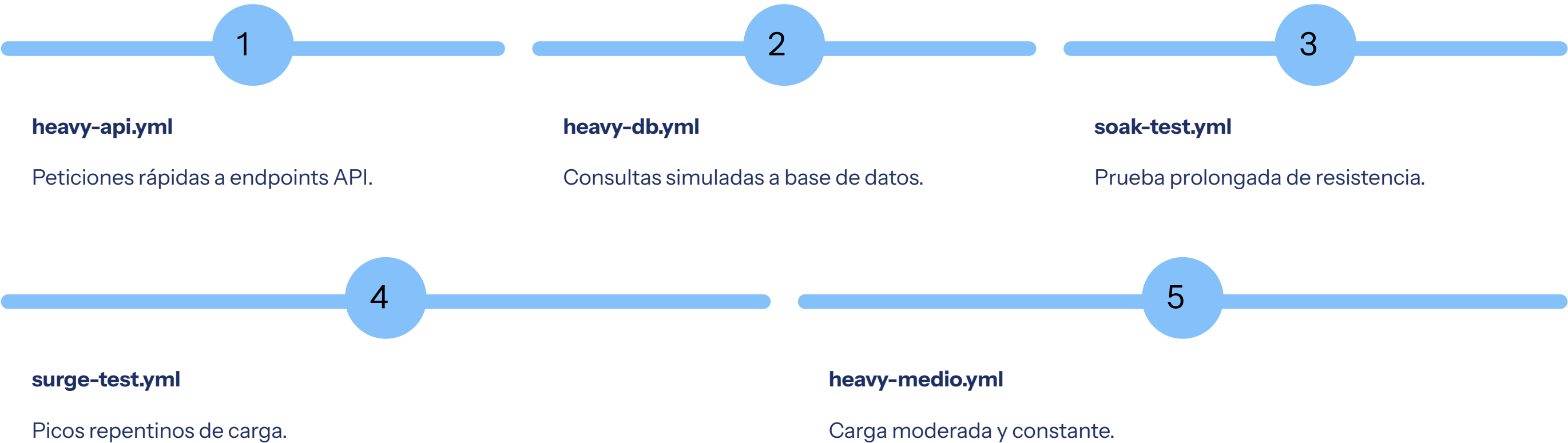
# Configuración Automatizada

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/ProyectoFinal\\_ServiciosTelematicos](https://github.com/EGSans/ProyectoFinal_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.





# 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

```
Resumen de la prueba: surge_test.json
=====
Total de Requests      : 7500
Respuestas 200 OK      : 7500
Errores totales        : 0
% de error              : 0.00%
Tipo(s) de error       : Ninguno
RPS promedio           : 55.0 req/s

⚡Latencias (ms - http.response_time)
min   : 2
p50   : 4
p95   : 7.9
max   : 160
media : 4.9
```

Resultado 1

Surge\_test.json

```
Resumen de la prueba: heavy.json
=====
Total de Requests      : 23668
Respuestas 200 OK      : 23370
Errores totales        : 298
% de error              : 1.26%
Tipo(s) de error       : ECONNRESET (298)
RPS promedio           : 395.0 req/s

⚡Latencias (ms - http.response_time)
min   : 2
p50   : 8.9
p95   : 210.6
max   : 3165
media : 51.7

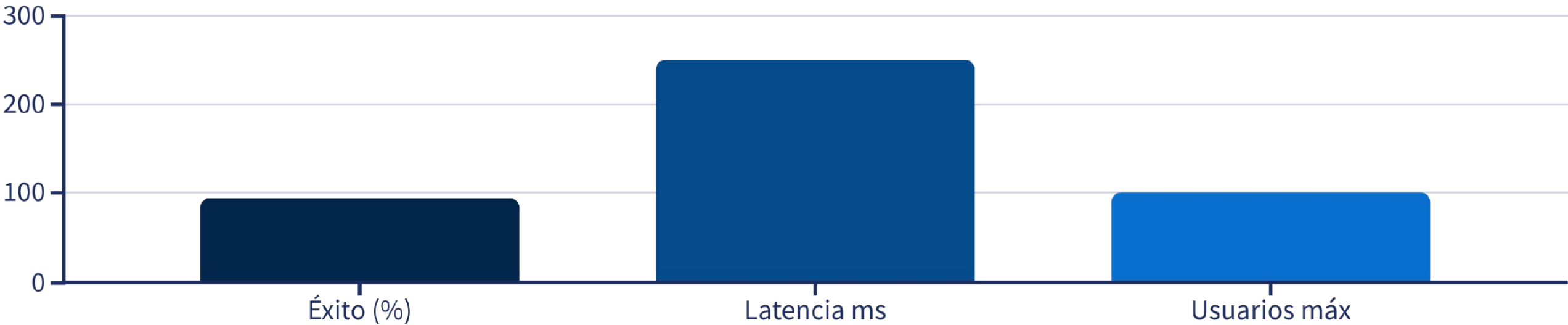
CSV:
file,requests,ok,errors,errors_pct,error_types,rps_mean,lat_min,lat_p50,lat_p95,lat_max
heavy.json,23668,23370,298,1.26,"ECONNRESET (298)",395,2,8.9,210.6,3165
PS C:\Users\ASUS\Desktop\UAO 2025-01\UAO 2025-02s\Servicios_Telematicos\prueba>
```

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!

