

Proyecto integrador - Testing de Performance

Pío Garetto

Introducción

Este trabajo implementa un pipeline end to end de pruebas de rendimiento sobre una API de e-commerce (autenticación, productos, carrito y checkout) con la siguiente solución:

- Infraestructura con Docker Compose: application (Node), jenkins, prometheus y grafana
- Plan de pruebas en Apache JMeter 5.6.3 ejecutado de forma headless dentro de un contenedor dedicado. El plan está parametrizado (threads, ramp, loops, host, port, SLA_MS) y realiza el flujo E2E usando CSV de usuarios y correlación (extracción de token).
- Aserciones: códigos HTTP 200 en cada request y SLA por assertion verificando responseTime menor o igual a SLA_MS (por defecto 1000 ms).
- CI/CD con Jenkins (pipeline declarativo): build de la imagen JMeter, health-check de la AUT, ejecución de pruebas, generación y publicación de reportes (JMeter HTML + resumen), archivo de artefactos y Gate (SLA + errores).
- Prometheus analiza /metrics (contadores e histogramas por ruta); Grafana muestra paneles de throughput, p95 y error rate.
- Ejecuciones: baseline con 10 usuarios y corrida comparativa planificada con 20.
- Entregables: reportes HTML/JTL, dashboard en Grafana y documentos (ejecutivo/técnico).

Pipeline perf-pipeline

This build requires parameters:

threads

Usuarios concurrentes

10

ramp

Ramp-up (s)

30

loops

Loops por usuario

5

SLA_MS

SLA por request (ms)

1000

▶ Build

Cancel

En la captura de pantalla se puede observar la pantalla de Jenkins “Build with Parameters” para el pipeline de performance, con opciones de concurrencia, ramp-up, loops y SLA configurables antes de ejecutar la prueba.

Informe ejecutivo

Resumen ejecutivo

Se probó el flujo completo de compra (login, productos, carrito, checkout) para validar que, bajo carga realista la aplicación cumpla con SLA p50 menor a 500ms, p95 menor a 1000ms y errores menor o igual a 1%. Con la finalidad de reducir riesgos operativos antes de escalar usuarios.

Conclusiones principales

Al correr las pruebas con 20 usuarios, se obtuvo un 0% de errores, el rendimiento fue de 15.58 peticiones por segundo en promedio.

- Latencias (total): P50 = 136ms, P95 = 345ms, P99 = 394ms

Al correr la prueba con 50 usuarios, se obtuvo un 0% de errores, el rendimiento fue de 194.14 peticiones por segundo en promedio.

- Latencias (total): P50 = 138ms, P95 = 344ms, P99 = 402ms

En ambos casos se cumplen los SLA.

Repercusión en el negocio

- Experiencia: las latencias son bajas y no hay errores.
- Confiabilidad: el flujo de compras funciona de punta a punta sin fallas
- Riesgo principal: el paso de “checkout” es el más lento. Sería el primero en degradarse.
- Capacidad: el sistema responde bien y con margen.

Recomendaciones

- Afinar checkout si se acerca al límite al escalar.
- Escalar gradualmente si se espera subida de tráfico en la aplicación.

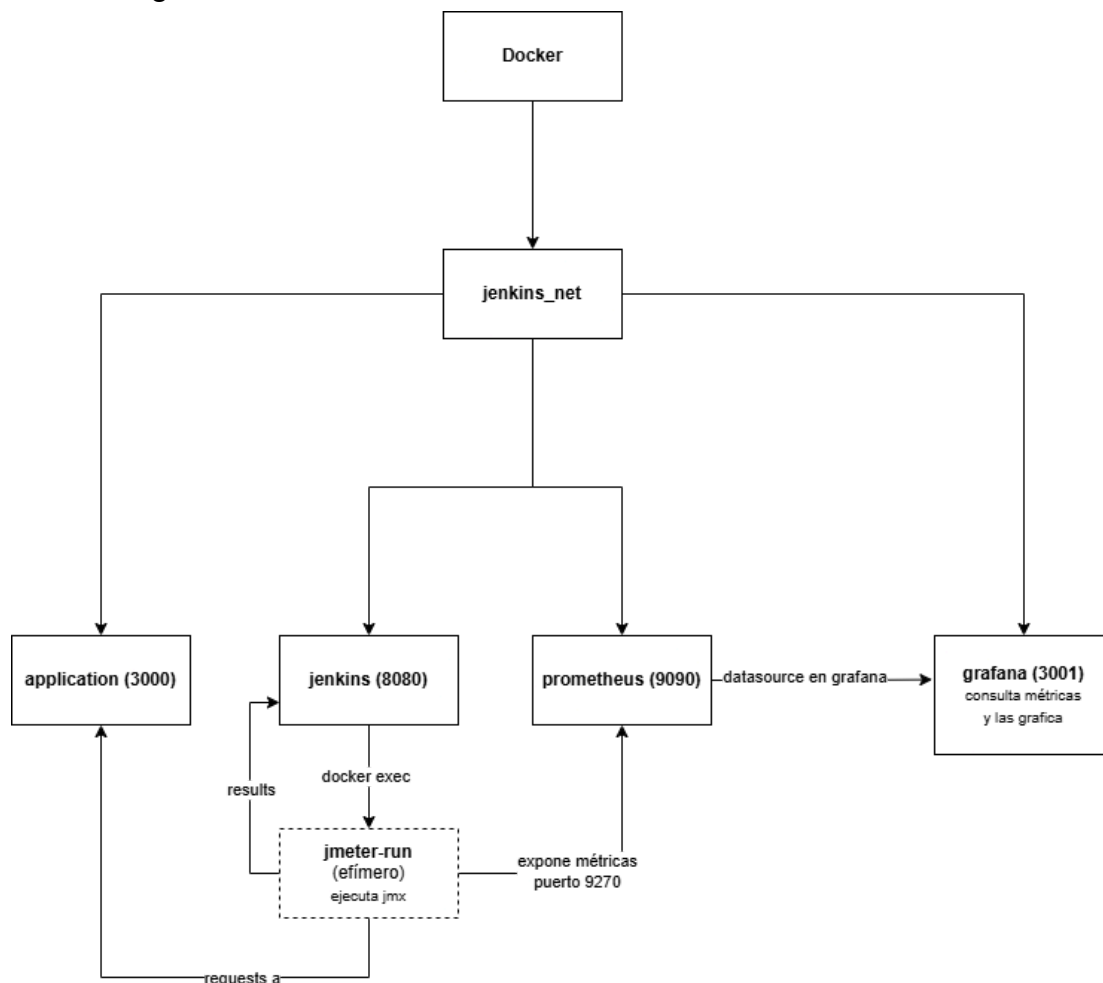
Conclusiones finales

La aplicación cumple con los objetivos de rendimiento y estabilidad. El sistema está apto para avanzar con un incremento moderado de la concurrencia, pero manteniendo vigilancia sobre el paso de checkout y activando mejoras puntuales sobre el mismo en caso de necesitarlo.

Informe técnico

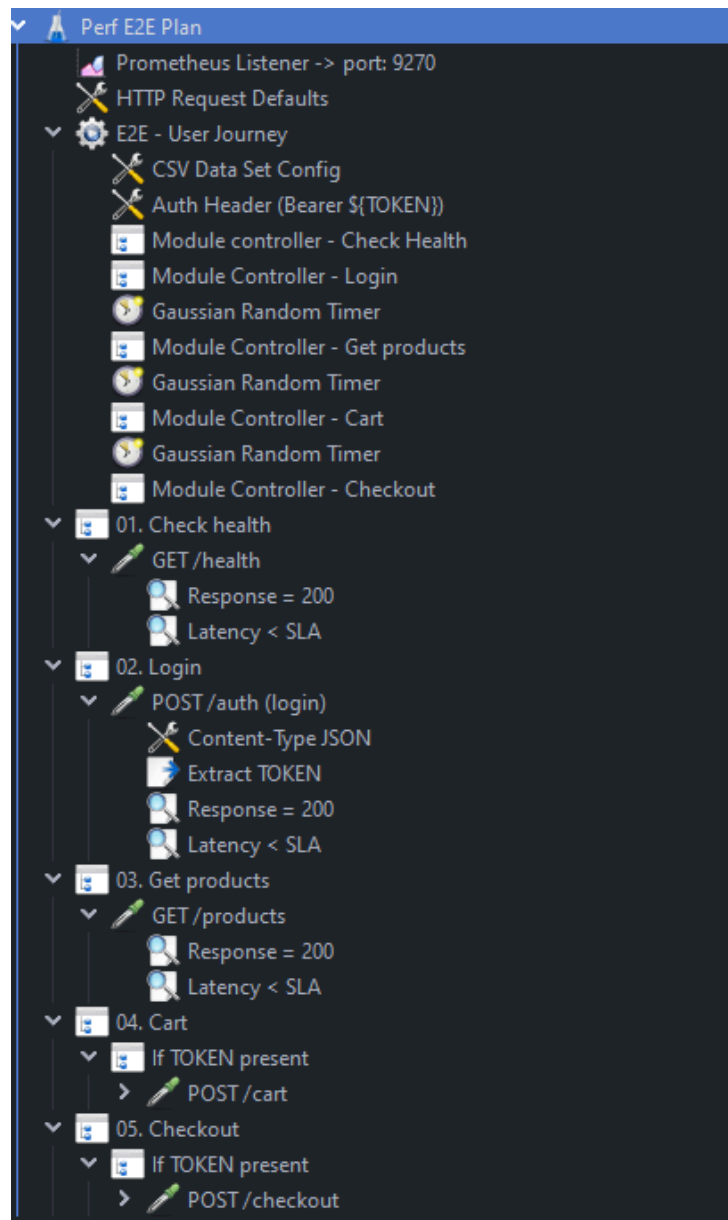
Configuración del entorno

- Herramientas, versiones:
 - Docker, Docker Compose.
 - Jenkins LTS
 - Apache JMeter 5.6.3 con Prometheus Listener 0.60
 - Prometheus 2.55.1
 - Grafana 10.4.3
- Arquitectura de docker compose
 - application - 3000
 - jenkins - 8080
 - prometheus - 9090
 - grafana - 3001
- Diagrama:



Diseño de plan de pruebas

- Estructura: Thread Group con el siguiente recorrido.
 - GET /health -> POST /auth -> GET /products -> POST /cart -> POST /checkout
- Parametrización:
 - host: application
 - port: 3000
 - threads: por propiedades
 - ramp: por propiedades
 - loops: por propiedades
 - SLA_P95_MS: por propiedades
- Afirmaciones y correlación:
 - Assertions por código HTTP 200 y response time < SLA_P95_MS.
 - Extracción de token de login con JSON Extractor (usado en las otras requests).
 - Verificación de existencia de token en requests que lo necesitan mediante If controller.
- Métricas Prometheus Listener
 - **jmeter_request_duration_ms (histogram)** - ResponseTime
 - **jmeter_requests_total (counter)** - SuccessTotal
 - **jmeter_requests_errors_total (counter)** - FailureTotal
- Estructura completa del Plan de Pruebas



- Se puede observar uso de Prometheus Listener, fragmentación en módulos, intervalos aleatorios entre los módulos (lo que demoraría un usuario real yendo de, por ejemplo, carrito hacia checkout). También hay aserciones tanto de tiempo como de código HTTP de respuesta. Además de eso, se implementó uso de controlador If para solamente realizar algunos requests si se tiene token de usuario.

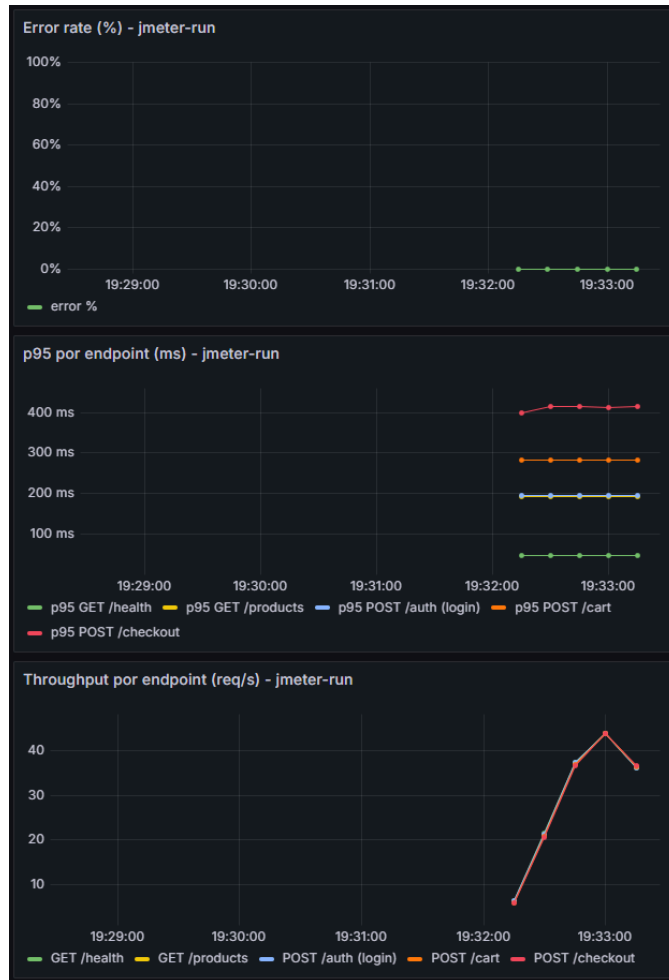
Integración CI/CD

- Pipeline Jenkins
 - Checkout SCM (<https://github.com/piogaretto/performance-final-challenge>)
 - Build imagen jmeter-prom.
 - Espera AUT en el endpoint /health

- Ejecuta jmeter-run (copia el .jmx y .csv)
- Publica resultados y expone métricas
- Gate de calidad: error %, p95 y assertions. Pasa o falla.

Supervisión y observabilidad

- Targets de Prometheus:
 - jmeter-run:9270/metrics -> RPS, errores, tiempos de respuesta.
- Paneles de Grafana.



En el gráfico se puede observar las gráficas de las estadísticas al correr el pipeline con 50 usuarios simultáneos.


- Query error rate: $100 * \frac{\text{sum}(\text{rate}(\text{jmeter_requests_errors_total}[1m]))}{\text{clamp_min}(\text{sum}(\text{rate}(\text{jmeter_requests_total}[1m])), 1e-9)}$
- Query p95 por endpoint: $\text{histogram_quantile}(0.95, \text{sum by (le, label)} (\text{rate}(\text{jmeter_request_duration_ms_bucket}[1m])))$
- Query Throughput: $\text{sum by (label)} (\text{rate}(\text{jmeter_requests_total}[1m]))$

Resultados

- Volumen total: 12500 requests en 81 segundos.
- Tiempos de respuesta
 - P50: 138.00ms
 - P95: 344.00ms
 - P99: 402.00ms
- Panel HTML

Statistics												
Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)
Label ^	#Samples ↕	FAIL ↕	Error % ↕	Average ↕	Min ↕	Max ↕	Median ↕	90th pct ↕	95th pct ↕	99th pct ↕	Transactions/s ↕	Received ↕ Sent ↕
Total	12500	0	0.00%	139.46	0	420	138.00	273.00	344.00	402.00	194.14	51.51 67.69
GET /health	2500	0	0.00%	0.59	0	29	1.00	1.00	1.00	2.00	39.31	8.33 11.55
GET /products	2500	0	0.00%	126.61	50	201	127.00	186.00	193.00	199.00	39.63	10.87 11.84
POST /auth (login)	2500	0	0.00%	141.12	80	202	142.00	189.00	194.00	200.00	39.56	14.68 15.54
POST /cart	2500	0	0.00%	157.98	60	261	157.00	238.00	250.95	258.00	39.57	8.99 14.91
POST /checkout	2500	0	0.00%	271.01	120	420	273.00	389.00	402.00	417.00	39.50	9.56 15.04


Resultados ejecutando pipeline con 50 usuarios simultáneos y 50 ciclos. Se pueden observar los percentiles por endpoint, el % de error y RPS.





Performance Test Report


Build: #54 | **Branch:** origin/main | **Date:** Thu Oct 16 22:18:20 UTC 2025

Test Environment: Docker Containerized | **Application:** E-commerce API

**Test Volume**
12500 Total Requests

**Success Rate**
100.0% (12500/12500)

**Response Times**
139 ms Avg · 344 ms p95
0–420 ms Range



Additional Reports

- [Detailed JMeter HTML Report](#)
- [Raw Test Results \(JTL\)](#)

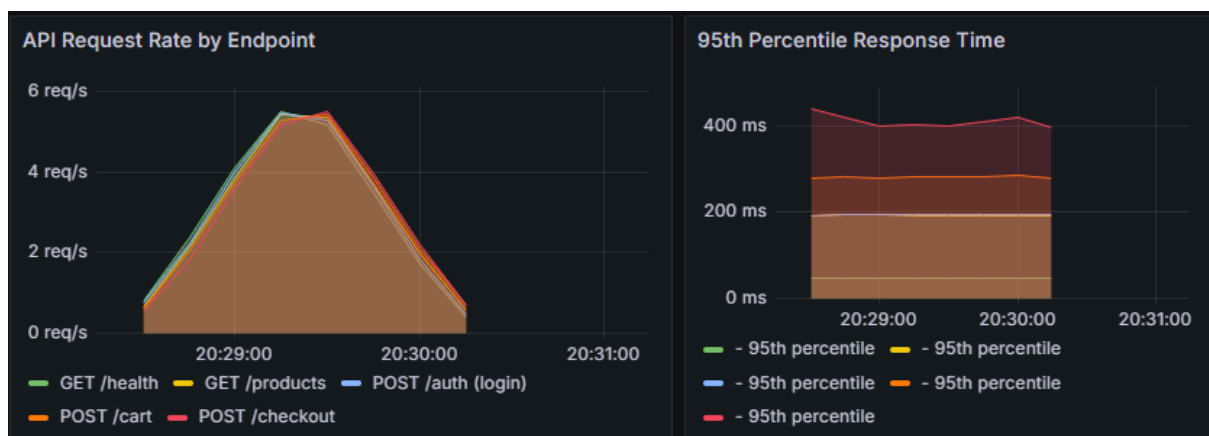
En la imagen podemos observar el contenido del apartado Performance Summary, el cual tiene estadísticas y links a los datos crudos y al reporte HTML.

Estimación y capacidad

- Teniendo en cuenta el p95 de 402ms en checkout, el sistema tiene un margen de más del doble de capacidad antes de rozar el SLA de 1000ms.
- Ley de Little:
 - $\lambda = 194 \text{ req/s}$
 - $W = 0.139 \text{ s}$
 - $L = 194 \times 0.139 \approx 26.966$
 - **27 requests concurrentes.**

Tendencias de rendimiento

En los siguientes gráficos de latencia y RPS muestran estabilidad a lo largo de la ejecución de las pruebas.



Se puede observar que en el pico de peticiones por segundo la latencia no aumenta.

Retos y soluciones

- Problemas con el archivo Jenkins en local.
 - Solución: uso de opción **"Pipeline from SCM"**
- Exposición de métricas JMeter a través de Prometheus Listener (la configuración de host por defecto era 127.00.0.1)
 - Solución: parámetro **prometheus.ip=0.0.0.0** disponible en la documentación del plugin.
- Publicación de reportes y enlaces con métricas NaN y enlaces que no funcionaban.
 - Solución: cálculo de métricas con awk/wc antes del hereDoc, verificación de rutas (publishHTML apuntando a

`${OUT_DIR}/jmeter-report/index.html` y
`${REPORTS_DIR}/generated/performance_summary.html`)

- Reportes HTML vacíos dentro de Jenkins:
 - Solución: colocar el siguiente comando en la Script Console de Jenkins: `System.setProperty("hudson.model.DirectoryBrowserSupport.CSP", "default-src 'self'; img-src * data: blob:; style-src 'self' 'unsafe-inline'; script-src 'self' 'unsafe-inline' 'unsafe-eval'; sandbox allow-scripts allow-same-origin;")`
- Panel de error % de Grafana mostrando “no data” cuando la tasa de error era 0%.
 - Solución: ajustar consulta para mostrar **0%** cuando no hay errores.

Conclusiones

Realicé pruebas de performance mediante un pipeline reproducible (Jenkins + JMeter en contenedor efímero + Prometheus + Grafana) y con reportes automatizados. La aplicación cumplió el SLA (p50 menor a 500ms, p95 menor a 1000 ms y error menor a 1%). Todo esto con 50 usuarios simultáneos realizando 50 ciclos completos del “camino completo”.

- p95 global: ~344 ms
- tasa de error: 0%
- throughput total: ~194 req/s
- p95 por endpoint: /health ~2 ms, /products ~193 ms, /auth ~199 ms, /cart ~251 ms, /checkout ~402 ms

Aun así, identifiqué **/checkout** como el tramo más sensible. Mi recomendación es optimizarlo puntualmente y repetir pruebas para ver el comportamiento del mismo. Con la parametrización del plan y las métricas de JMeter expuestas mediante Prometheus en Grafana, el resultado es una base sólida que se puede reproducir rápidamente y comparar resultados en futuras mejoras de la aplicación.