Informe Técnico: Desafío Pruebas de rendimiento de extremo a extremo

1. Introducción

El proyecto se basa en la ejecución y análisis de los resultados de las pruebas de rendimiento utilizando Jmeter con integración con Jenkins, además de Prometheus y Grafana para observar los resultados de las pruebas.
La aplicación sometida a pruebas se basa en una aplicación node.js sencilla de e-commerce, la cúal consiste en 4 apis que tienen el objetivo de permitir a un usuario realizar el flujo de un pedido.
El objetivo de las pruebas es obtener métricas para evaluar el desempeño de las apis, y a través de estos resultados evaluar el cumplimiento de las SLA dadas.

2. Configuración del entorno

a. Herramientas utilizadas

Se utilizaron las versiones de las herramientas:

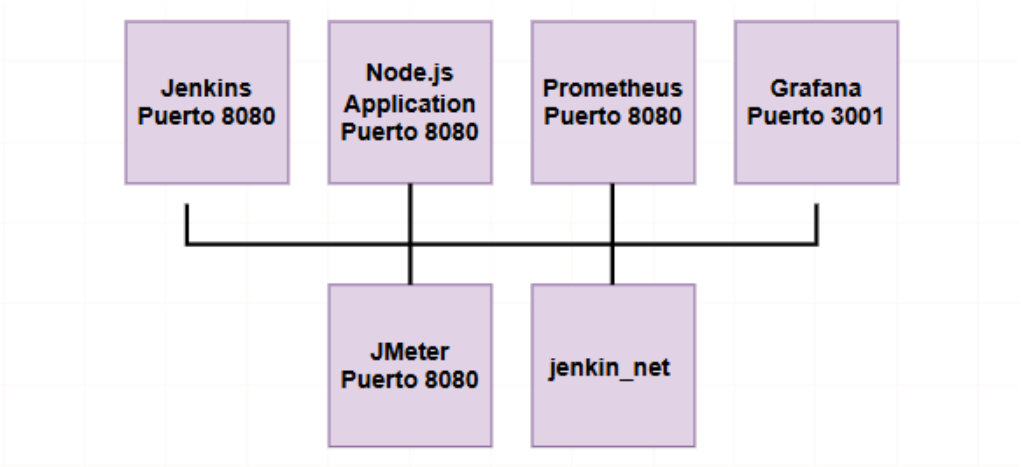
- Apache JMeter versión 5.6.3
- Prometheus versión 2.55.1
- Grafana versión 10.4.3
- Jenkins versión 2.516.3
- Docker versión 28.3.3

b. Arquitectura de Docker Compose

Se cuenta con los siguientes servicios desplegados en Docker para la ejecución de las pruebas:

- Grafana en el puerto 3001, imagen grafana/grafana:10.4.3
- Prometheus en el puerto 9090, imagen perfci-application
- Jenkins en el puerto 5000, imagen prom/prometheus:v2.55.1
- Application en el puerto 3000, imagen perfci-jenkins

c. Diagrama del sistema sometido a prueba



3. Diseño del plan de pruebas

a. Estructura del plan de pruebas JMeter

En el plan de pruebas diseñado se cuenta con un thread group de 10 usuarios, con un ramp-up 30 segundos y con 5 loops.
En este plan de pruebas se utilizan 4 muestreadores, uno por api a testear. Las endpoints dentro del alcance de las pruebas son los siguientes: POST /auth, GET /products, POST /cart, POST /checkout. Cada uno de estos muestreadores se colocó en un test fragment para que pueda ser reutilizado si se requieren realizar más pruebas.
Se utilizaron controladores de módulo para implementar los test fragments dentro del thread group. Por otro lado, se agregaron los listeners view result tree y un Backend Listener para obtener los datos de Prometheus y poder utilizarlos en Grafana.

En cuanto a la parametrización, se utilizaron dos variables: host y port, para poder realizar la ejecución utilizando Jerkins.
Para obtener los datos y realizar el login se utilizó el archivo csv llamado users.csv, el cúal contiene el email y password necesarios.
En este caso la aplicación no requiere que se utilice el token en los headers ni datos generados previamente por otra api, por lo que no es necesario la extracción de ningún dato. En el plan de pruebas se realiza de igual manera por si aporta valor.

4. Integración CI/CD

Para la ejecución de las pruebas y la obtención de las métricas se utilizó un pipeline de Jenkins. Las etapas de su ejecución fueron las siguientes:

- Realiza el checkout del código a través del repositorio de GitHub y construye la imagen de Docker de JMeter.
- Cuando la aplicación a testear está disponible iniciar un contenedor JMeter que se conecta a la aplicación para ejecutar los tests.
- Durante la ejecución, los resultados se almacenan y se genera el informe HTML. Esta información también puede verse desde Jerkins al terminar la ejecución de la pipeline.

5. Supervisión y observabilidad

Para obtener los resultados de las pruebas utilizamos Prometheus como data source y plasmamos los datos utilizando Grafana para que sean más entendibles.
Se utilizaron dos paneles de Grafana uno llamado “Performance Monitoring Dashboard” y otro llamado “Request and response data”. En estos paneles se muestran gráficos con el rendimiento de las apis, los distintos percentiles en cuanto al tiempo de respuesta, proporción de errores entre otras. Algunas de las consultas de utilizadas fueron las siguientes:

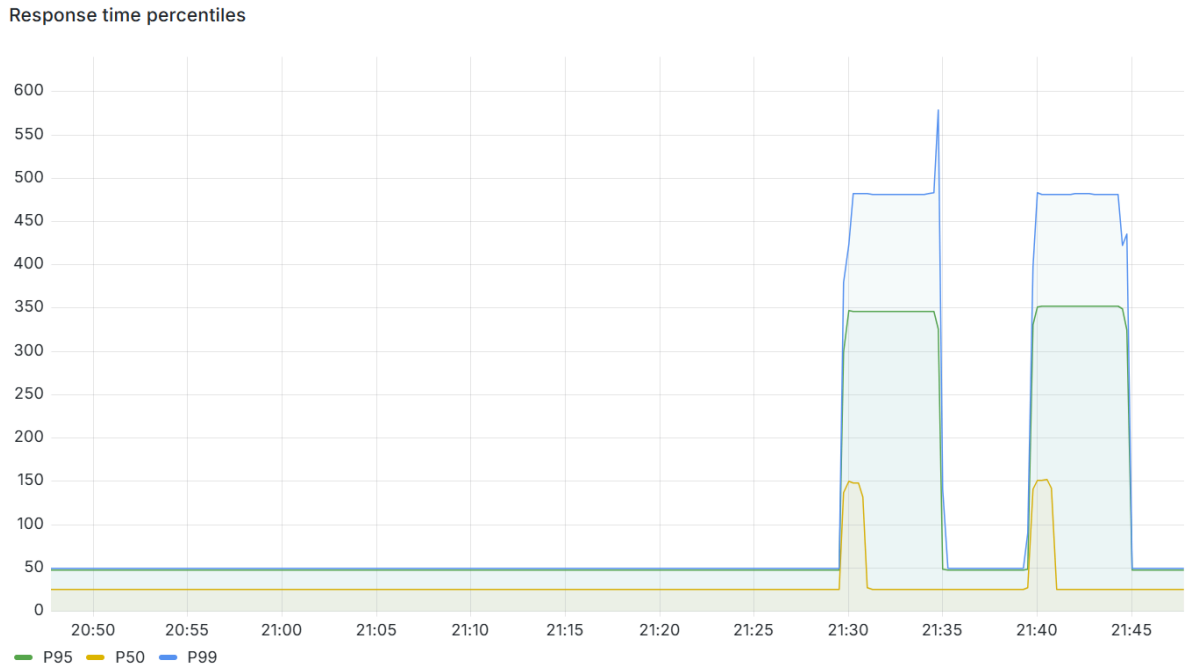
- rate(http_request_duration_ms_sum[$__rate_interval])
- sum(rate(http_request_duration_ms_count{route!~"/health|/metrics"}[1m]))
- histogram_quantile(0.95, rate(http_request_duration_ms_bucket[5m]))

6. Resultados

a. Métricas clave:

- Tiempos de respuesta

Escenario	P50(ms)	P95(ms)	P99(ms)
POST /checkout	271.50	409.80	413.00
GET /products	136.00	202.00	203.00
POST /auth	162.00	195.00	205.00
POST /cart	172.00	257.25	263.00
Total	172.00	374.65	411.96

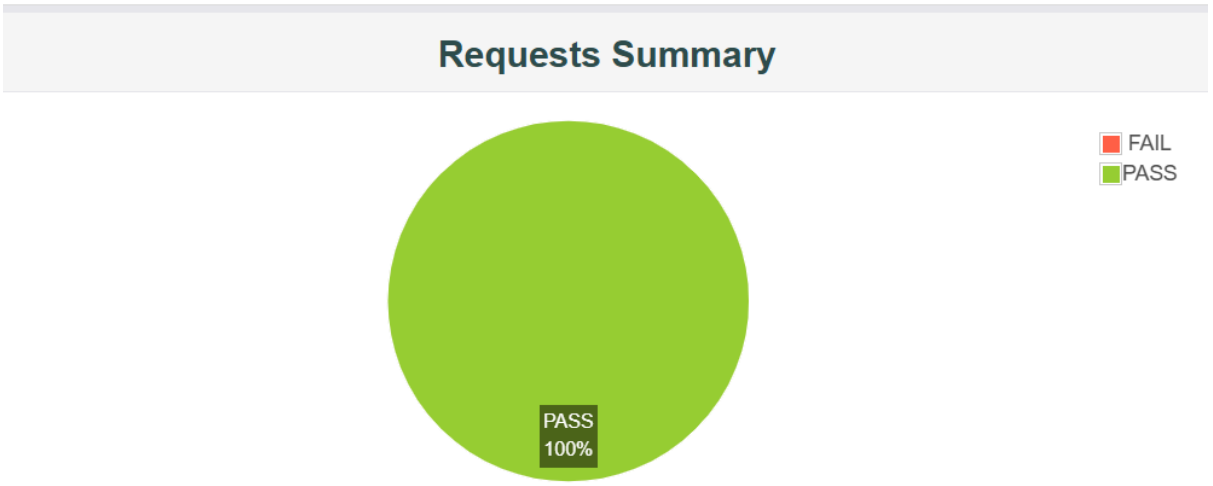


- Rendimiento

Escenario	Transacciones/segundo
POST /checkout	1.66
GET /products	1.67
POST /auth	1.66
POST /cart	1.66
Total	6.73

- Tasa de error

Se registró una tasa de error del 0%. Donde de 200 samples ejecutados ninguna prueba presentó errores.



7. Estimación de la capacidad

La concurrencia realizada en las pruebas en la aplicación es de 6.730(rendimiento) * 0.374(tiempo de respuesta p95) = 2,517 usuarios simultáneos.
La cantidad de usuarios máximos sin influir el tiempo de respuesta ni rendimiento serían:6.730(rendimiento objetivo) * 1.000(tiempo de respuesta p95 máximo esperado) * 0.7(factor de seguridad) = 4,711.

En resumen, la concurrencia observada en la prueba fue de aproximadamente 2,5 usuarios simultáneos según Ley de Little (basado en TPS y P95). Si consideramos un tiempo de respuesta máximo aceptable de 1 s y un factor de seguridad del 70%, la aplicación podría soportar teóricamente hasta 4.700 usuarios concurrentes.

8. Retos y soluciones

Entre los desafíos que me encontré realizando esta práctica fue el desconocimiento del funcionamiento de Jenkins lo que me llevó a que se presentaran bastantes errores a la hora de iniciar el pipeline que tuve que ir solucionando. Entre ellos, tuve problemas con la branch de git de donde tomaba los datos, por lo que tuve que setearla en el jenkinsfile. También había métodos que no eran soportados, por ejemplo el Math.random() el cuál tuve que reemplazar en el código.
Otra dificultad que se me presentó fue la implementación del listener de Prometheus, el cúal no disponibiliza las métricas de Jmeter en Prometheus. Lo resolví cambiando el listener utilizado.

9. Conclusión

Se observa que la aplicación cumple con los objetivos de p50 < 500ms y p95 < 1000ms, y que su tasa de error es 0% lo que no pone el riesgo la permanencia de los usuarios en la aplicación. En cuanto a la capacidad soportada, como posible mejora se podría evaluar incrementar los recursos de infraestructura para que una mayor cantidad de usuarios

puedan utilizar los servicios a la vez, pero se remarca que con el estado actual del sistema no se detectaron problemas de rendimiento que afectan la experiencia del usuario.

Para próximos proyectos, se tendrá en cuenta lo aprendido de cómo implementar Grapana y Prometheus para visualizar los datos obtenidos. Esto debido a que permite mostrar el resultado de las pruebas para que sea más entendible y poder respaldar a través de gráficas las conclusiones de rendimiento y capacidad que se determinan.