|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **DEPARTAMENTO:** | **Ciencias de la Computación** | **CARRERA:** | **Ing. de Software** | | |
| **ASIGNATURA:** | Pruebas de Software | **NIVEL:** | Séptimo | **FECHA:** | 08/02/2025 |
| **DOCENTE:** | Ing. Luis Castillo | **PRÁCTICA N°:** | 1 | **CALIFICACIÓN:** |  |

**Pruebas de carga y rendimiento**

**Allan Vinicio Panchi Pillajo**

**RESUMEN**

El presente laboratorio tiene como finalidad evaluar el rendimiento de sistemas web mediante pruebas de carga realizadas con la herramienta k6. Se abordan dos escenarios: una API REST básica y un backend completo con autenticación JWT y conexión a una base de datos MongoDB. La finalidad es observar el comportamiento del sistema bajo diferentes niveles de carga, midiendo parámetros como latencia, tasa de errores y capacidad de respuesta.

Palabras Claves: k6, pruebas de rendimiento, API REST, backend, JWT, MongoDB, carga concurrente, latencia, rendimiento web, Node.js.

1. **INTRODUCCIÓN:**

En el contexto del desarrollo de aplicaciones web modernas, es fundamental asegurar no solo la funcionalidad de un sistema, sino también su capacidad de responder eficientemente ante múltiples usuarios concurrentes. Este laboratorio se enfoca en aplicar pruebas de rendimiento a diferentes niveles de complejidad utilizando la herramienta k6, reconocida por su eficiencia y facilidad de integración con entornos de desarrollo. Se pondrá a prueba una API básica y un backend con autenticación basada en JWT y persistencia en MongoDB, evaluando métricas clave para garantizar la robustez y escalabilidad del sistema.

1. **OBJETIVO(S):**

2.1. Evaluar el rendimiento de una API REST simple bajo escenarios de carga controlada.

2.2. Aplicar pruebas de carga a un backend con autenticación JWT y conexión a MongoDB utilizando k6.

2.3. Medir y analizar métricas clave como latencia, tasa de errores y tiempo de respuesta para determinar la capacidad del sistema bajo estrés.

1. **MARCO TEÓRICO:**

**K6**

k6 es una herramienta de código abierto desarrollada por Grafana Labs para realizar pruebas de carga y rendimiento. Está diseñada para ser eficiente, sencilla de usar y permite simular múltiples usuarios concurrentes. Los scripts se escriben en JavaScript, lo que la hace muy accesible para desarrolladores web. Entre sus métricas destacan: tiempo de respuesta, latencia, errores por segundo y throughput.

**Pruebas de rendimiento**

Las pruebas de rendimiento son un tipo de prueba no funcional que permite identificar cómo se comporta un sistema bajo diferentes condiciones de carga. Evalúan aspectos como el tiempo de respuesta, el consumo de recursos, la escalabilidad y la tolerancia a fallos. Son clave para garantizar que el sistema pueda soportar el uso en producción.

**Node.js**

Node.js es un entorno de ejecución para JavaScript construido sobre el motor V8 de Chrome. Permite desarrollar aplicaciones del lado del servidor de manera eficiente y escalable, ideal para sistemas que requieren alta concurrencia y bajo consumo de recursos.

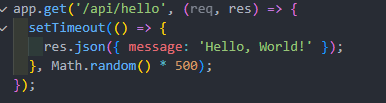
**MongoDB**

MongoDB es una base de datos NoSQL orientada a documentos. Utiliza un formato BSON para almacenar información, lo que permite alta flexibilidad y escalabilidad. Es muy utilizada en entornos de desarrollo ágil y se integra fácilmente con Node.js.

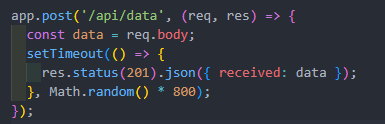
1. **DESCRIPCIÓN DEL PROCEDIMIENTO:**

**Creación del servidor:**

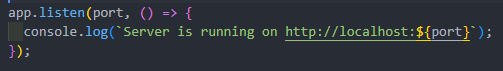
Se crea un servidor web básico utilizando el framework Express en Node.js, que expone dos rutas API: una de tipo GET en /api/hello y otra POST en /api/data. La primera responde con un mensaje "Hello, World!"



Después de un retardo aleatorio de hasta 500 milisegundos, simulando una latencia variable. La segunda recibe datos en formato JSON desde el cuerpo de la solicitud y los devuelve en la respuesta con un estado 201 (creado).

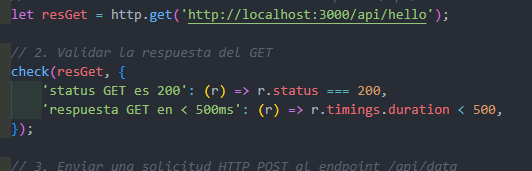


También después de una demora aleatoria de hasta 800 milisegundos. Además, el servidor está configurado para escuchar en el puerto 3000 y utiliza un middleware para procesar cuerpos JSON.



Se realiza un script para la prueba automatizada creado con la herramienta k6 para evaluar el rendimiento de una API local desarrollada con Express. Simula el comportamiento de usuarios virtuales en tres etapas: un calentamiento con 10 usuarios, una carga sostenida con 50 y un enfriamiento gradual.

Durante cada iteración, los usuarios envían una solicitud GET al endpoint /api/hello y una solicitud POST a /api/data, incluyendo un objeto JSON con información de la iteración

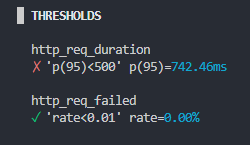




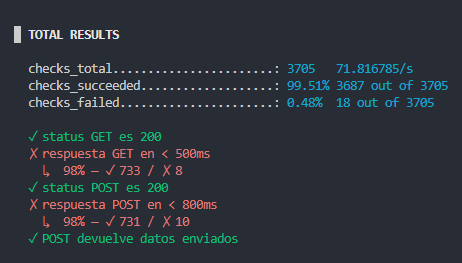
Luego, se validan aspectos como el código de respuesta, el tiempo de respuesta (latencia) y la integridad de los datos recibidos. Además, se establecen umbrales que determinan si el sistema cumple con los niveles de rendimiento deseados, como mantener el 95% de las respuestas por debajo de 500 ms y una tasa de error menor al 1%.



Una vez definido nuestras pruebas, y nuestro backend básico, corremos el comando k6 run carga\_y\_rendimiento,js y se procede a analizar las métricas

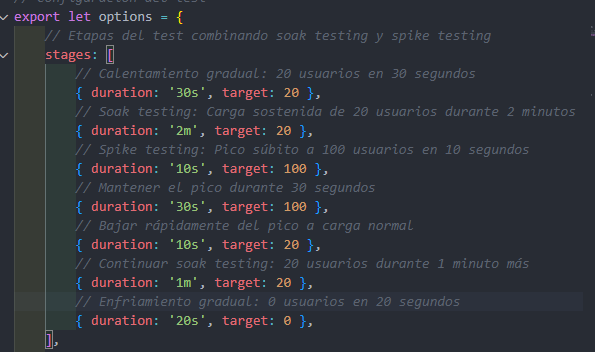


El servidor demostró una alta disponibilidad con un 0% de peticiones fallidas, cumpliendo exitosamente el umbral establecido de menos del 1% de fallos. Sin embargo, el rendimiento en tiempo de respuesta no fue óptimo, ya que el 95% de las peticiones tardaron hasta 742.46ms en procesarse, superando significativamente el umbral establecido de 500ms.



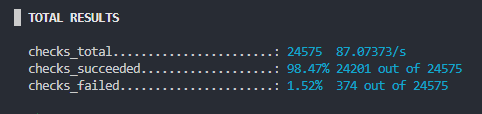
En cuanto a las validaciones funcionales, el sistema mostró una excelente confiabilidad con un 99.51% de checks exitosos, confirmando que tanto los endpoints GET como POST están funcionando correctamente y devolviendo los datos esperados. Los únicos fallos registrados estuvieron relacionados principalmente con los tiempos de respuesta que excedieron los límites establecidos para algunas peticiones, no con errores funcionales del sistema.

Para pruebas soak y spike, se configuraron las etapas del código a las siguientes

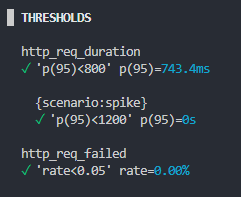


Con esos cambios se volvió a correr el comando

La prueba combinada de soak testing y spike testing se ejecutó exitosamente durante 4 minutos y 42 segundos, completando 4,915 iteraciones con un pico máximo de 100 usuarios virtuales concurrentes. El sistema procesó un total de 9,830 peticiones HTTP a una tasa promedio de 34.8 peticiones por segundo,

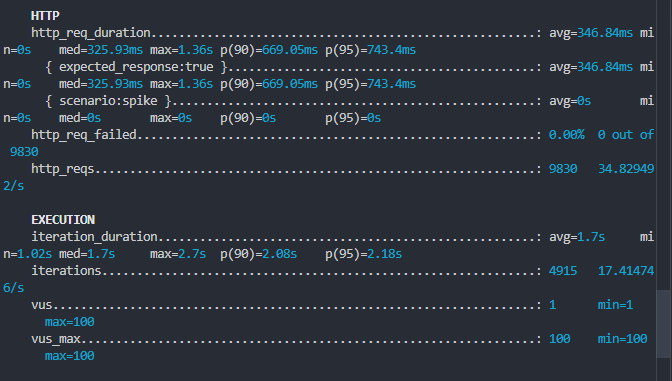


El servidor mostró una excelente estabilidad durante las fases de carga sostenida de larga duración, manteniendo un comportamiento consistente con 0% de peticiones fallidas y cumpliendo todos los umbrales establecidos. El tiempo promedio de respuesta fue de 346.84ms con un percentil 95 de 743.4ms, manteniéndose dentro del umbral establecido de 800ms.



Durante el pico súbito de 100 usuarios concurrentes, el sistema demostró una notable resiliencia, manejando efectivamente la carga extrema sin presentar fallos. Aunque se registraron algunas degradaciones en los tiempos de respuesta (máximo registrado de 1.36 segundos), el sistema se mantuvo operativo y funcional.

Se ejecutaron 24,575 validaciones con un 98.47% de éxito (24,201 exitosas), indicando que tanto los endpoints GET como POST mantuvieron su funcionalidad correcta bajo condiciones de estrés prolongado. Los fallos registrados (374 checks fallidos, 1.52%) estuvieron relacionados principalmente con tiempos de respuesta que excedieron los límites durante momentos de alta carga, pero no con errores funcionales del sistema.



1. **CONCLUSIONES:**

El uso de k6 permite realizar pruebas de rendimiento efectivas, adaptadas tanto a servicios simples como a sistemas más complejos con autenticación y base de datos.

Las pruebas revelan cuellos de botella y puntos críticos en el comportamiento del sistema, lo cual es clave para mejorar su estabilidad y escalabilidad.

1. **RECOMENDACIONES:**

Integrar pruebas de rendimiento en el ciclo de desarrollo continuo para detectar problemas antes del despliegue a producción.

Monitorear constantemente las métricas obtenidas y ajustar parámetros de infraestructura o código para mejorar el rendimiento sostenido del sistema.

1. **BIBLIOGRAFÍA:**

[1] Grafana Labs, “k6 - Performance testing tool for developers,” [Online]. Available: https://k6.io

[2] M. Fowler, “Non-Functional Requirements,” martinfowler.com, [Online]. Available: https://martinfowler.com/articles/nonFunctionalRequirements.html

[3] Node.js Foundation, “Node.js,” [Online]. Available: https://nodejs.org

[4] MongoDB Inc., “MongoDB Documentation,” [Online]. Available: https://www.mongodb.com/docs