Arquitectura de Software
(75.73/TB054) Curso Calonico

Trabajo Práctico 1

28 de octubre de 2024

Patricio Galvan
106166

Santiago Vaccarelli
106051

Matías Bacigalupo
103715

1. Introducción

El objetivo de éste trabajo práctico es comparar distintas estrategias y escenarios en un servicio HTTP, ver cómo impactan en los atributos de calidad y probar qué cambios se podrían hacer para mejorarlos.

Para ello se creó un servidor local con 4 servicios diferentes:

- **Ping** Un servicio para usar como healthcheck y como baseline para comparar con los demás servicios.
- **Dictionary** Un servicio que devuelve información de diccionario de una palabra en ingles.
- **Spaceflight News** Un servicio que devuelve los títulos de noticias sobre actividad espacial.
- **Useless Facts** Un servicio que devuelve un hecho al azar.

Estos servicios se sometieron a pruebas evaluando las siguientes tácticas:

1. **Caso base** Usado para tomar como referencia y verificar si hay mejoras con las tácticas aplicadas.
2. **Caché** Almacenar información recibida de la API y usarla al responder a un llamado.
3. **Replicación** Escalar el servicio a 3 copias.
4. **Rate Limiting** Limitar la frecuencia de consumo del servicio.

Para construir el servidor, simular tráfico, obtener datos del tráfico simulado en las pruebas, y graficar dichos datos, se usaron diversas tecnologías incluyendo Node.js, Docker, Docker Compose, Nginx, Redis, Artillery, cAdvisor, StatsD, Graphite y Grafana.

Para el análisis de las pruebas realizadas se tuvieron en cuenta varias métricas, incluyendo tiempos de respuesta para el cliente, tiempos de repuesta totales de los endpoints, estado de las solicitudes, uso de los recursos, y tiempos de respuesta de las APIs externas.

Más adelante se muestran los resultados de las pruebas de estrategias para cada servicio.

2. Componentes y Conectores

A continuación mostraremos diagramas de la estructura de componentes y conectores para cada estrategia.

2.1. Caso Base

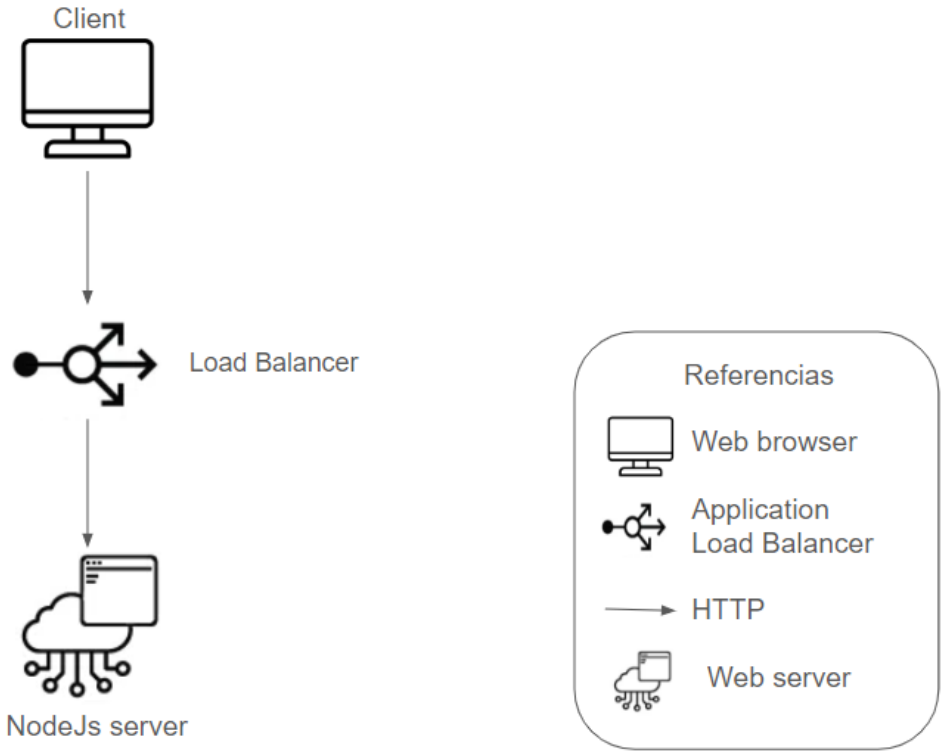

Figura 1: Estructura para Caso Base

2.2. Caché

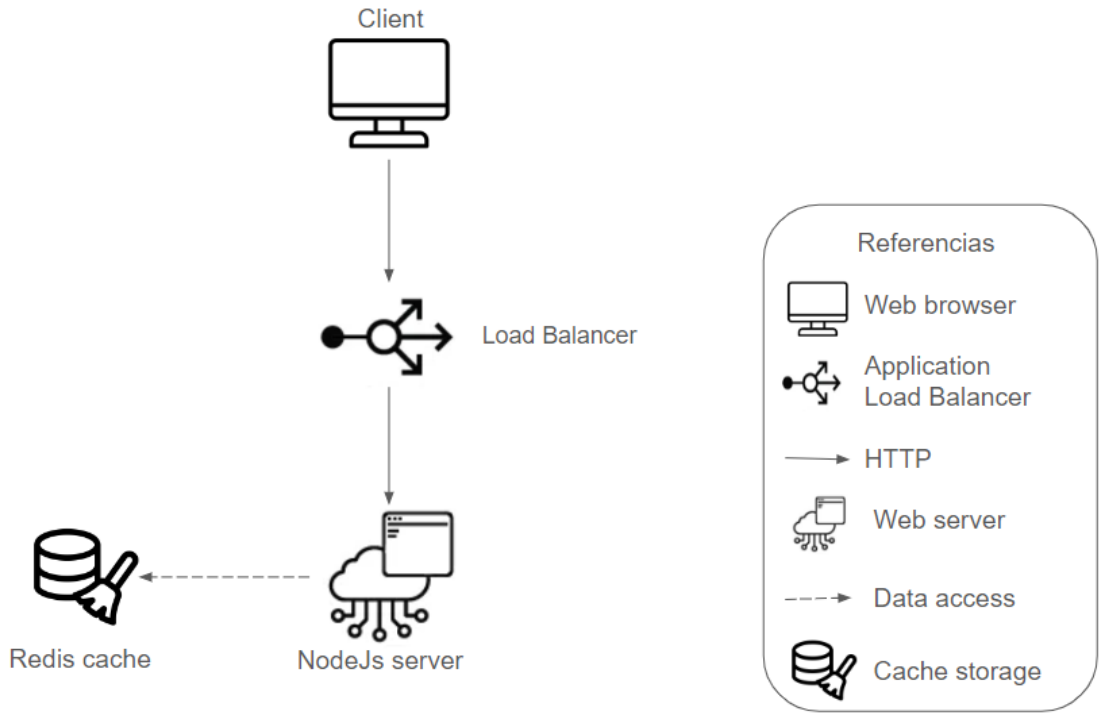

Figura 2: Estructura para Caché

2.3. Rate Limiting

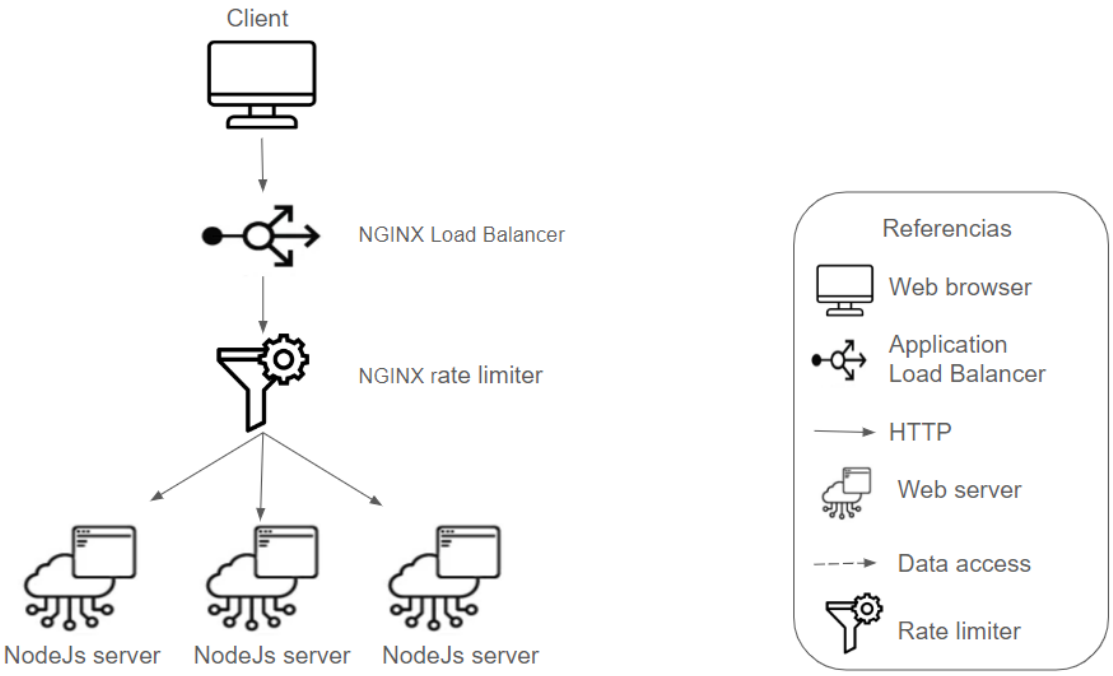

Figura 3: Estructura para Rate Limiting

2.4. Replication

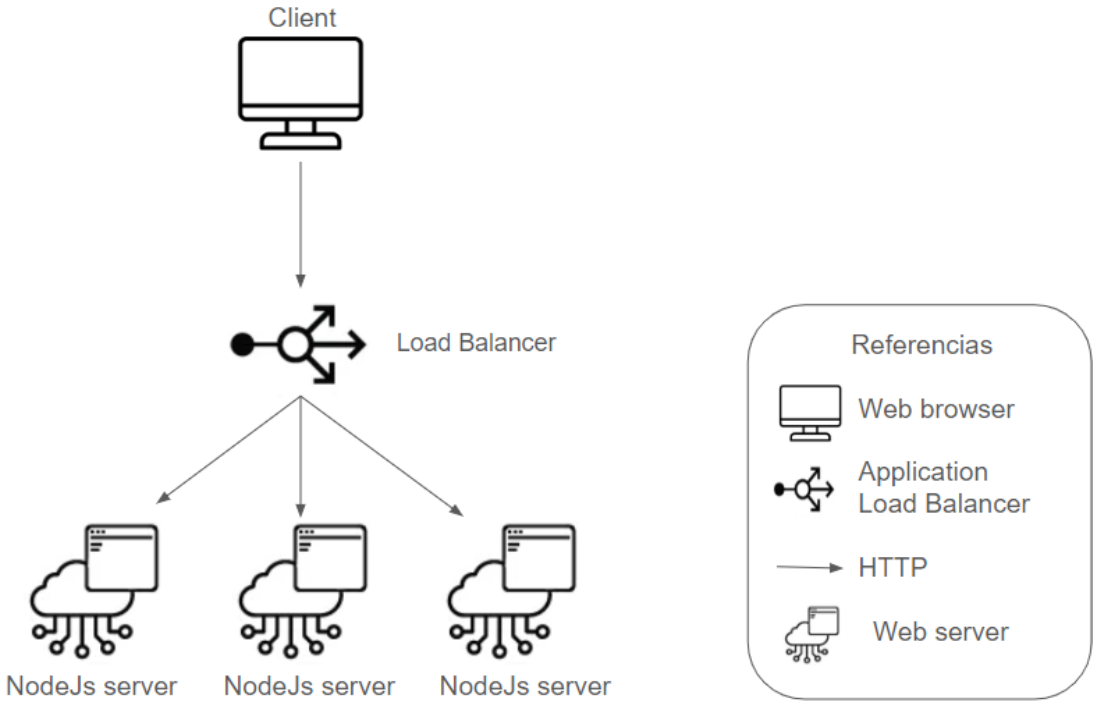

Figura 4: Estructura para Replication

3. Ping

Este servicio se usa como healthcheck y como baseline para comparar con los demás servicios.

3.1. Caso Base

Figura 5: Test de spike

3.2. Caché

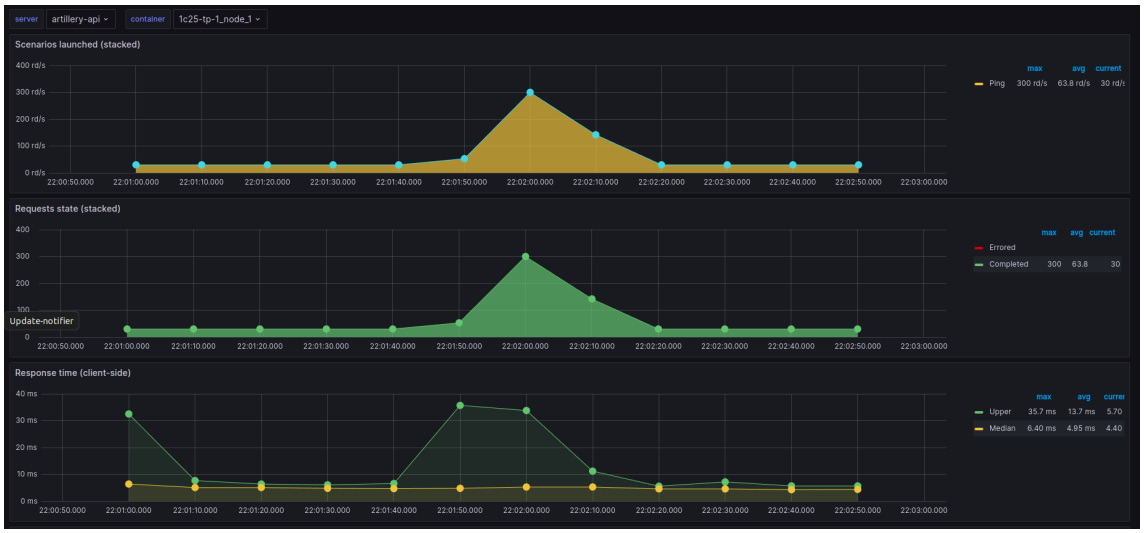

Figura 6: Test de spike con caché

3.3. Rate Limiting

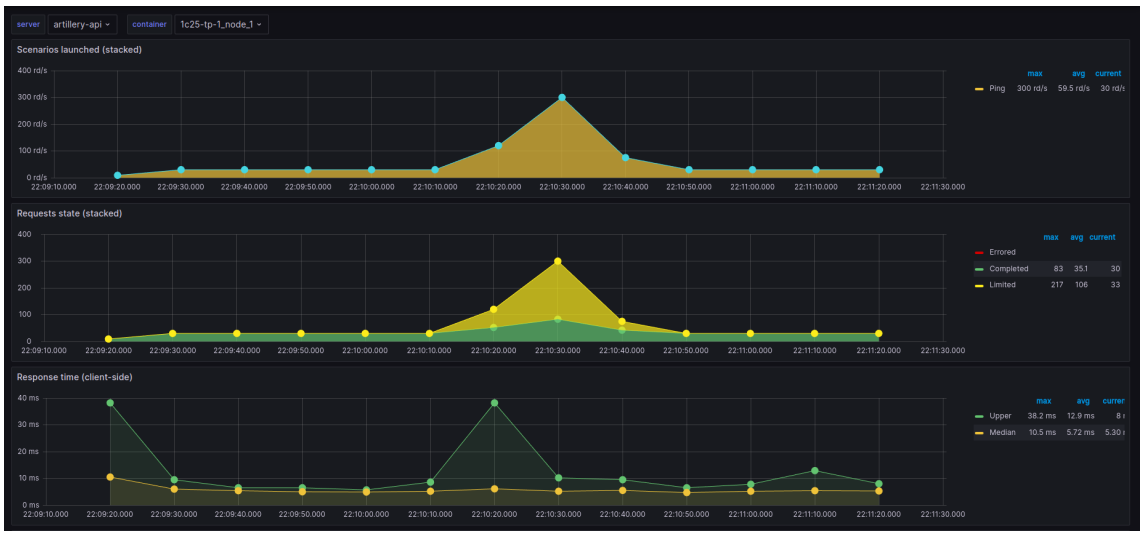

Figura 7: Test de spike con rate limiting

3.4. Análisis

Los gráficos de rendimiento obtenidos tras las pruebas de spike no muestran variaciones significativas entre las diferentes estrategias evaluadas respecto del caso base. Esto sugiere que el procesamiento del endpoint es lo suficientemente rápido y liviano como para no beneficiarse de optimizaciones adicionales.

Ya que el tiempo de respuesta base del endpoint es lo suficientemente bajo y la respuesta del endpoint no varía, no tiene sentido almacenar en memoria caché la respuesta constante.

Por otro lado, la implementación de rate limiting tampoco introduce una variación relevante en los resultados. Esto puede indicar que las pruebas de spike realizadas no alcanzaron un nivel de solicitudes tan alto como para que el rate limiting afectara el desempeño o la disponibilidad del servicio.

En conclusión, estas estrategias no aportan un beneficio tangible al servicio de Ping.

4. Dictionary

Este servicio devuelve información de diccionario de una palabra en ingles, consultando a la Free Dictionary API.

4.1. Caso Base

Figura 8: Test de stress

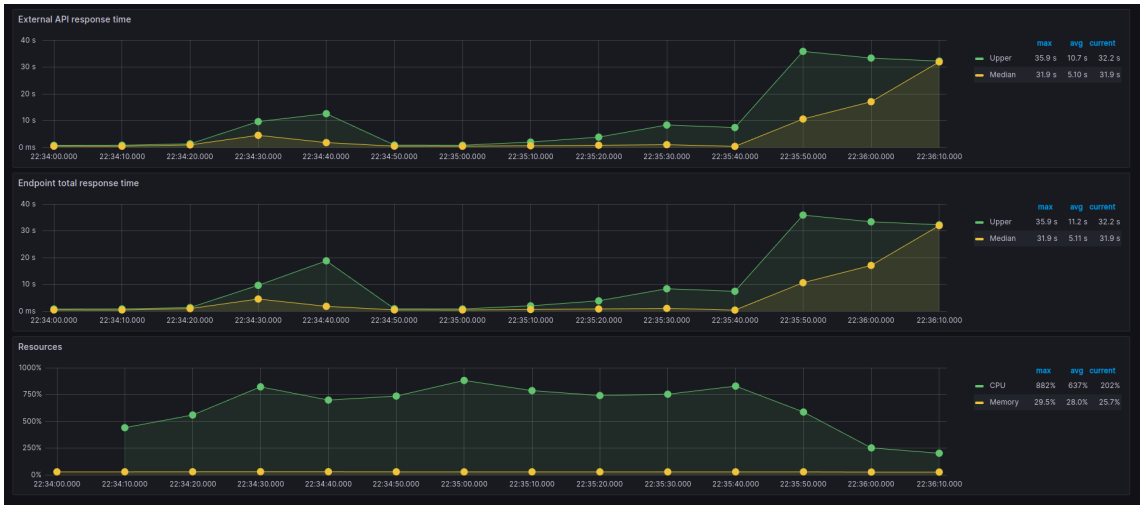

Figura 9: Test de stress

4.2. Caché

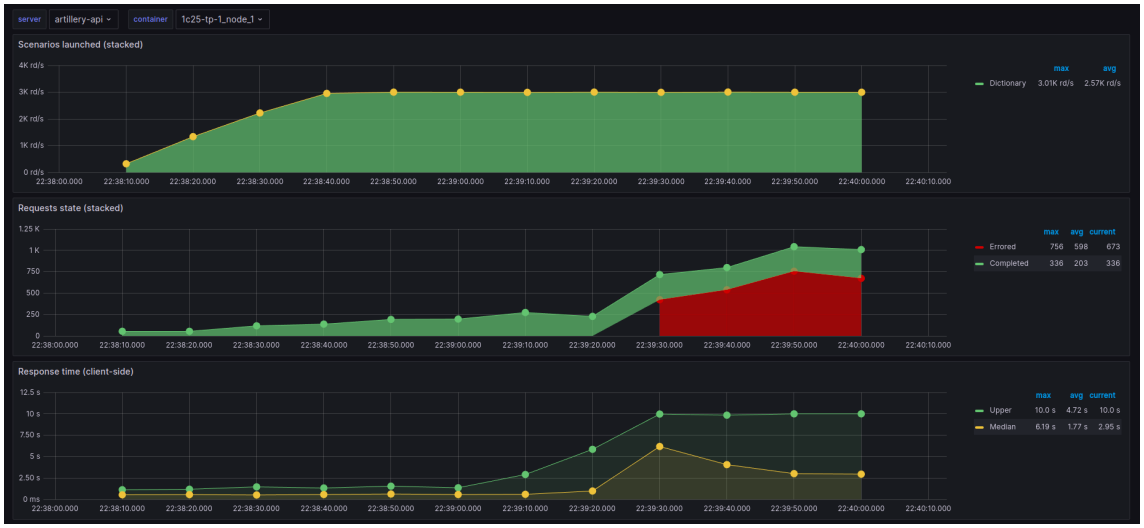

Figura 10: Test de stress con caché

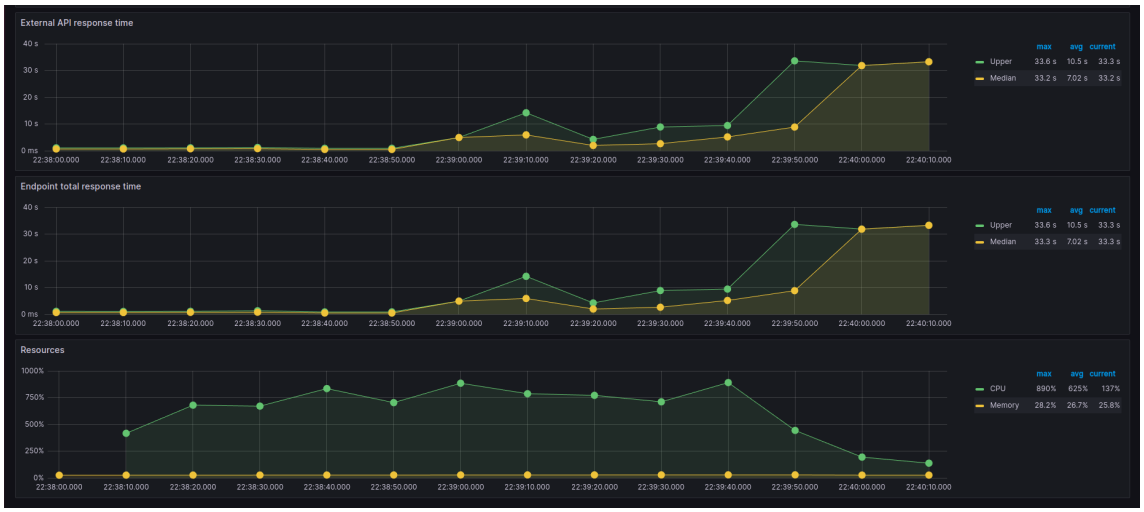

Figura 11: Test de stress con caché

4.3. Rate Limiting

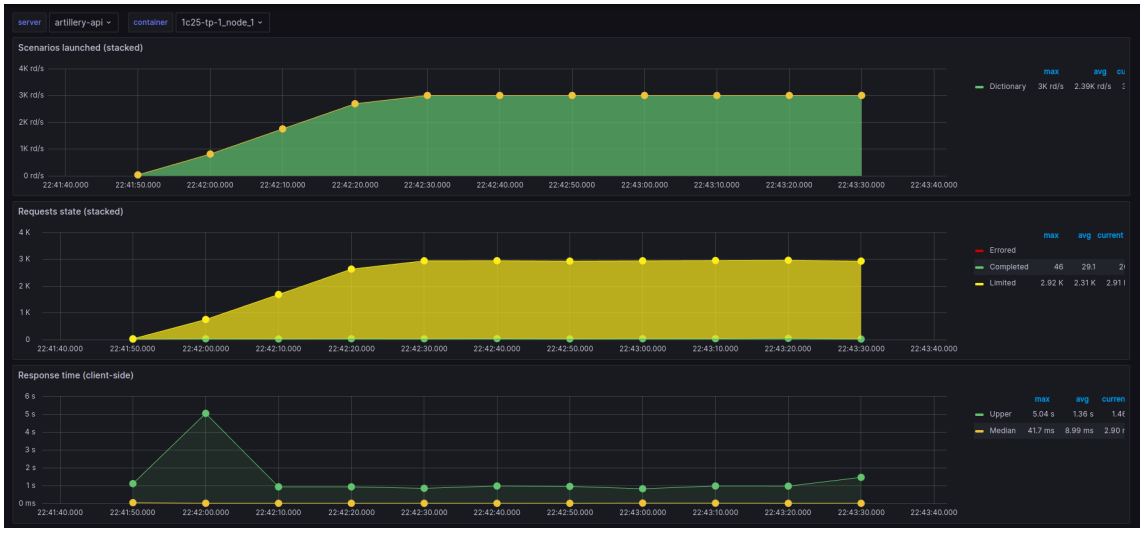

Figura 12: Test de stress con rate limiting

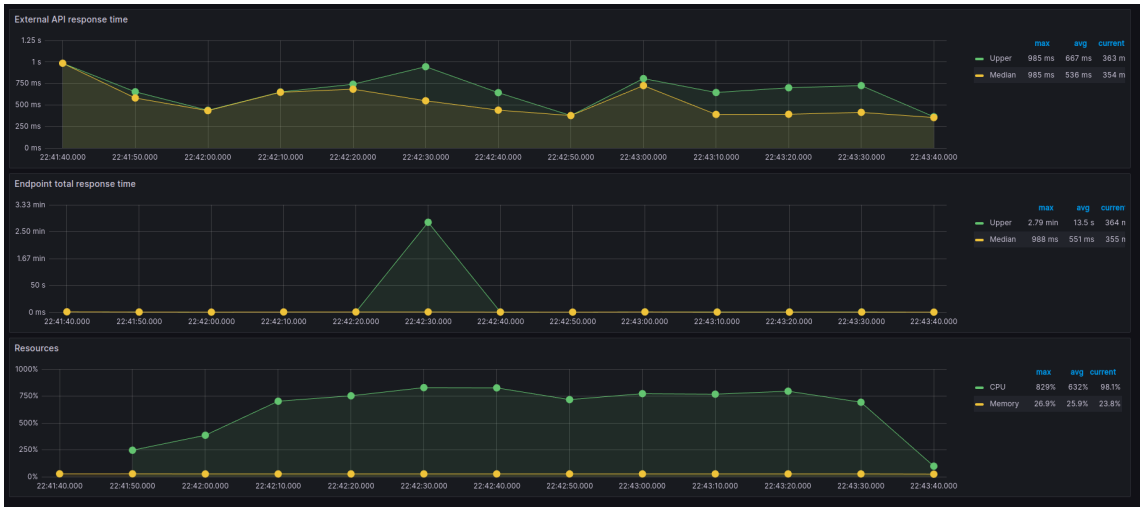

Figura 13: Test de stress con rate limiting

4.4. Análisis

El gráfico de caso base indica que el servidor solo puede responder en promedio a 50 requests por segundo. Al someterlo a 3000 requests por segundo la gran mayoría son rechazadas.

Al introducir una capa de cache que almacena la respuesta para las palabras ya consultadas, se observa una leve mejora en la tasa de éxito de las solicitudes y en la latencia del sistema. Con esta estrategia, una vez que el significado de una palabra ha sido almacenado en la cache, las consultas posteriores pueden responder rápidamente sin necesidad de hacer nuevas solicitudes a la API externa. Esto reduce la carga en el servicio externo y mejora la capacidad de respuesta del sistema, pero solo es efectiva cuando el conjunto de palabras está limitado y la repetición de términos es probable. En este caso se trata de un diccionario de 4000 palabras, lo que explica que solo se vea una mejora leve al someter al endpoint a 3000 requests por segundo.

Por otro lado, al imponer un límite en el número de solicitudes que pueden ser procesadas por segundo con la estrategia de rate limiting, se mitiga el impacto de la sobrecarga y protege tanto el endpoint local como la API externa de un exceso de solicitudes. Si bien esto significa que la mayoría de las solicitudes serán rechazadas o ralentizadas bajo carga extrema, la estabilidad general del servicio mejora en comparación con el caso base.

5. Spaceflight News

Este servicio devuelve los títulos de las 5 últimas noticias sobre actividad espacial, obtenidas desde la Spaceflight News API.

5.1. Caso Base

Figura 14: Test de carga

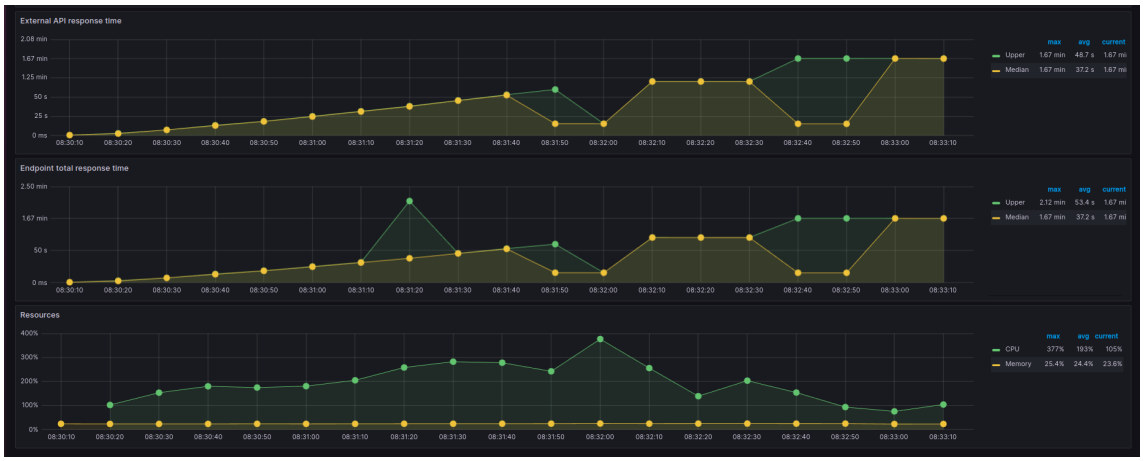

Figura 15: Test de carga

5.2. Caché

Como la respuesta es la misma para todos los usuarios mientras no haya nuevas noticias, se decidió almacenar la respuesta en el momento en que el primer cliente hace un request (lazy population) y usarla para todos los usuarios siguientes por 1 hora.

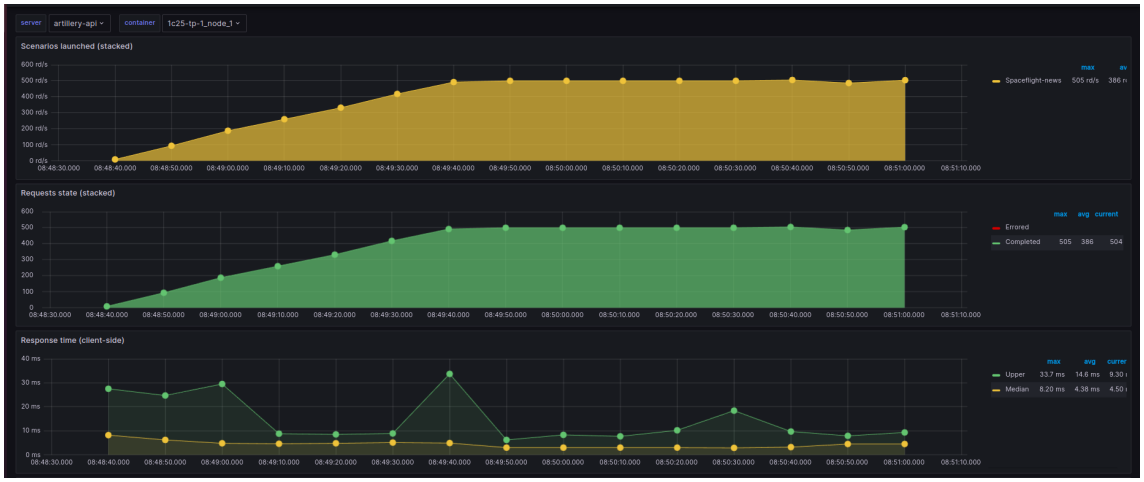

Figura 16: Test de carga con caché

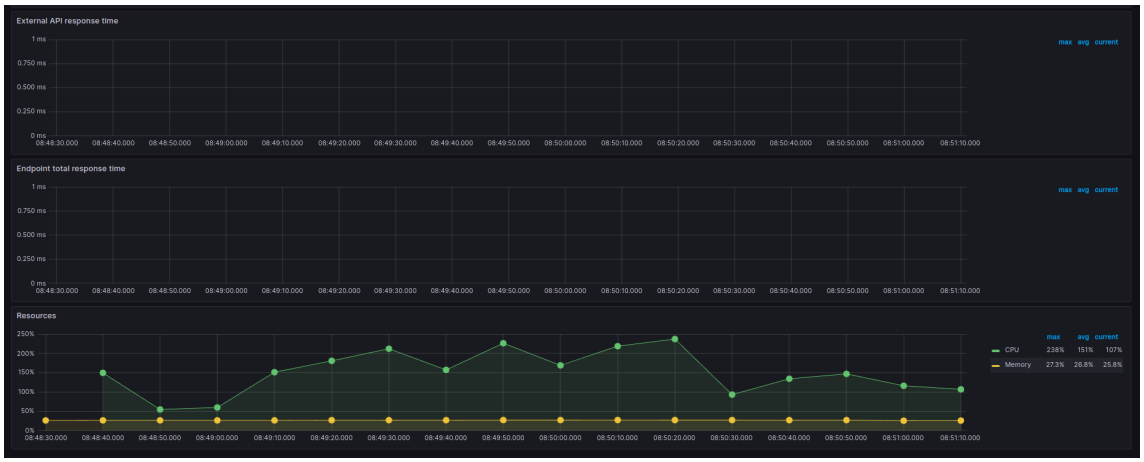

Figura 17: Test de carga con caché

5.3. Rate Limiting

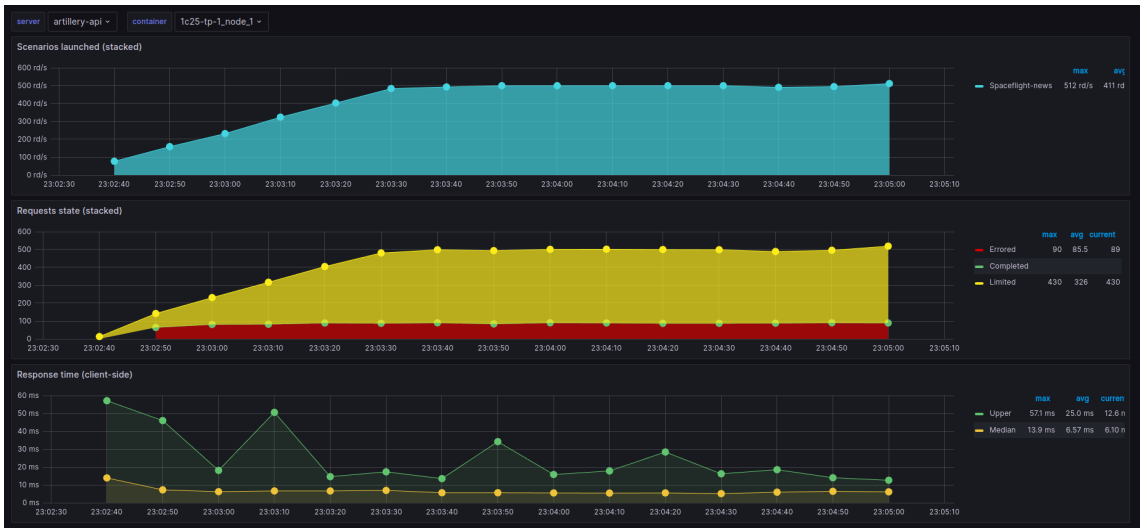

Figura 18: Test de carga con rate limiting

Figura 19: Test de carga con rate limiting

5.4. Análisis

Bajo una carga alta de solicitudes, la API externa rápidamente se ve saturada, lo que lleva a un aumento en los tiempos de respuesta y a una tasa significativa de fallos en las solicitudes.

Al implementar una cache que almacena los títulos de las noticias consultadas, el sistema reduce la necesidad de realizar múltiples solicitudes idénticas a la API externa para las mismas noticias. Dado que las noticias no se actualizan constantemente, la cache permite responder rápidamente a las solicitudes repetidas sin depender de la disponibilidad de la API externa. Esto resulta en una mayor tasa de éxito y en una latencia reducida para las solicitudes. La estrategia de cache es especialmente efectiva en este endpoint.

La implementación de rate limiting ayuda a evitar una saturación tanto del sistema local como de la API externa, ya que restringe el flujo de solicitudes y permite que el sistema mantenga

un nivel de desempeño más estable. Aunque algunas solicitudes pueden ser rechazadas debido al límite impuesto, esta estrategia contribuye a una tasa de éxito más estable para el tráfico que sí se permite, reduciendo los fallos totales y manteniendo el servicio disponible sin interrupciones.

6. Random Useless Facts

Este servicio devuelve un hecho irrelevante al azar, tomado de Useless Facts.

6.1. Caso Base

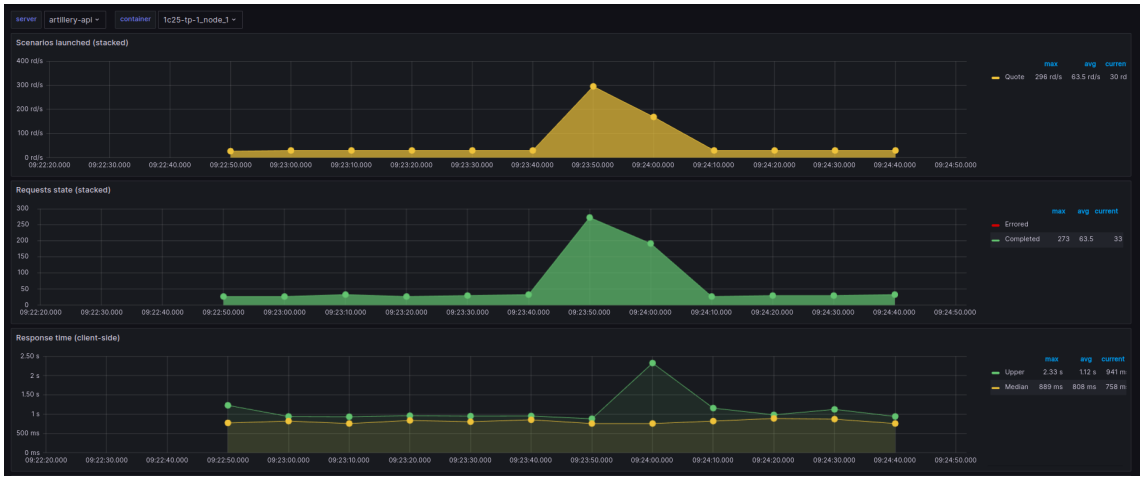

Figura 20: Test de spike

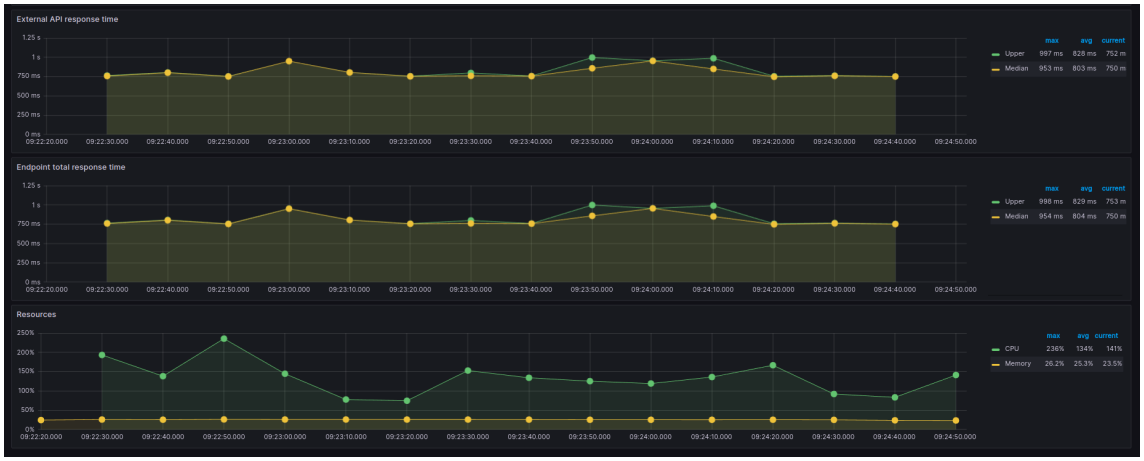

Figura 21: Test de spike

6.2. Caché

No tiene sentido cachear una respuesta aleatoria que cambia con cada solicitud.

6.3. Replicación

Para la replicación se usaron 3 copias del servicio en simultáneo.

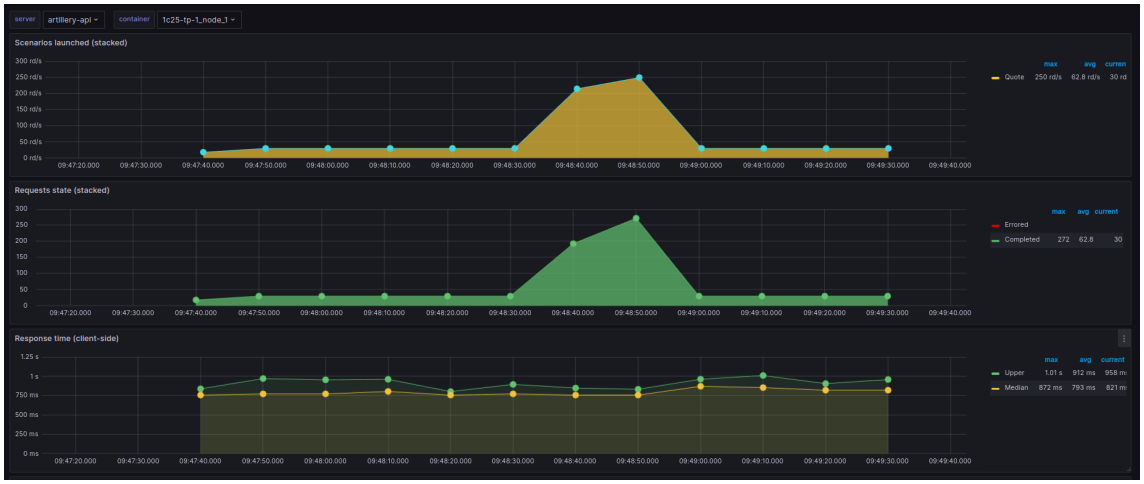

Figura 22: Test de spike con balanceo

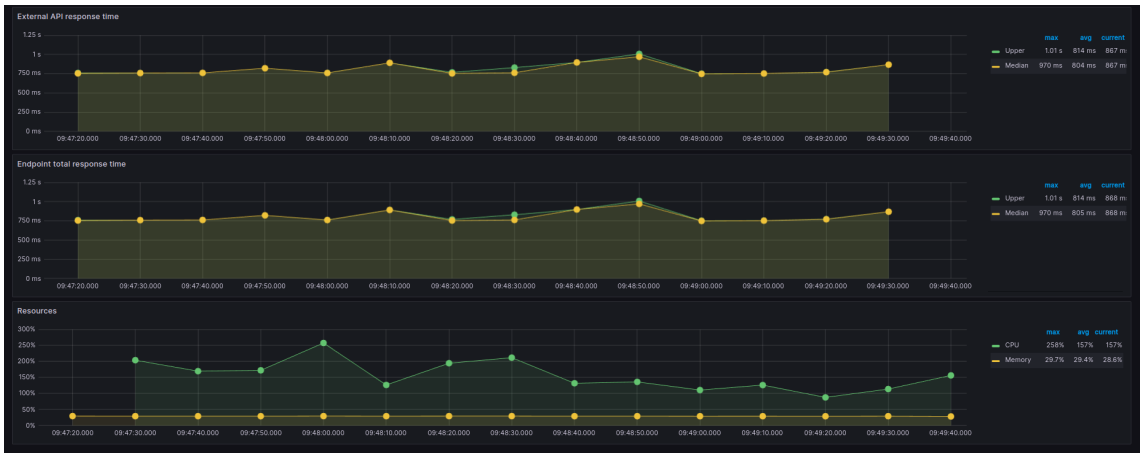

Figura 23: Test de spike con balanceo

6.4. Rate Limiting

Se limitó el rate de respuestas aceptadas a 3 por segundo.

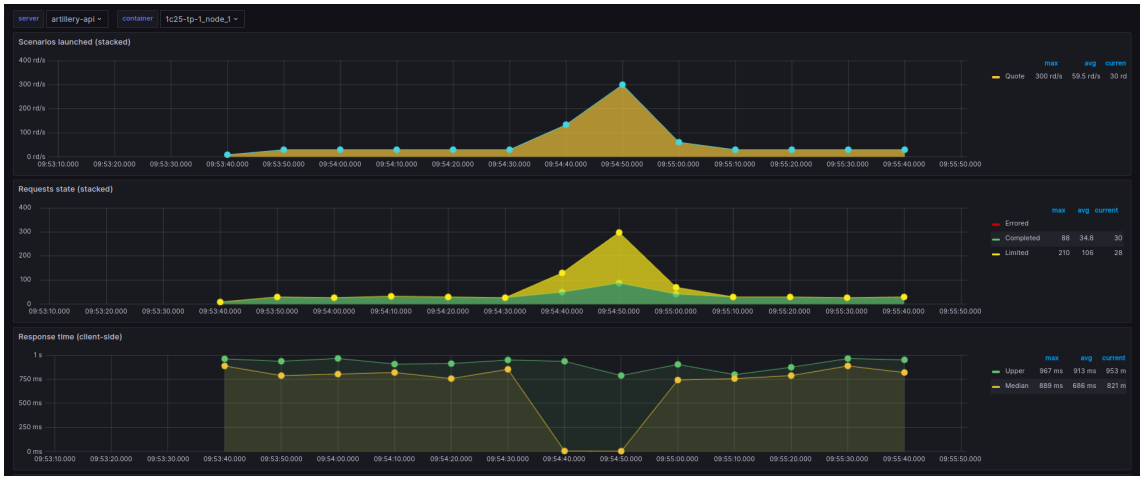

Figura 24: Test de spike con rate limiting y balanceo

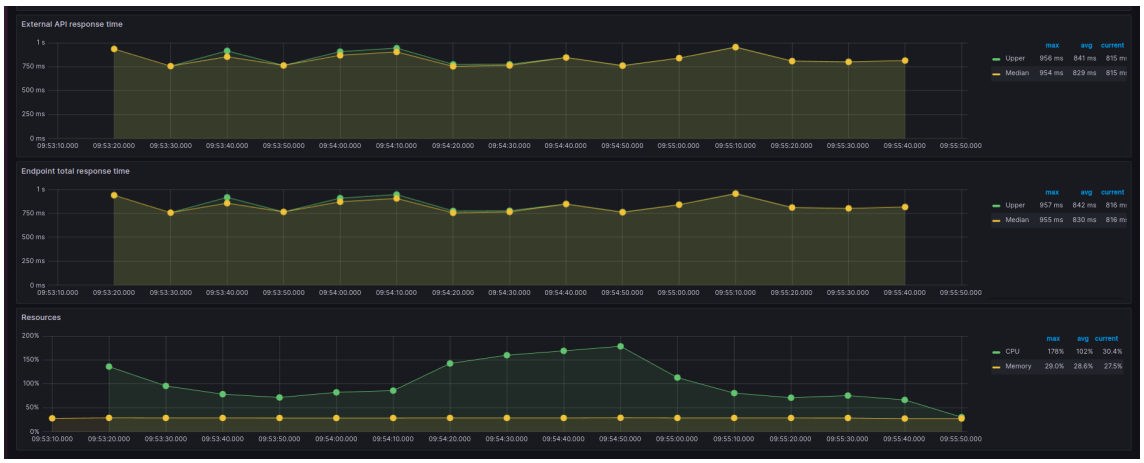

Figura 25: Test de spike con rate limiting y balanceo

6.5. Análisis

Cada solicitud al endpoint se traduce en una consulta directa a la API externa para obtener un nuevo dato aleatorio. Bajo una alta carga de solicitudes, esta configuración genera rápidamente una saturación en el sistema debido a la gran cantidad de consultas a la API externa, que probablemente no está diseñada para soportar un tráfico tan elevado. Como resultado, el endpoint experimenta tiempos de respuesta elevados y una alta tasa de fallos.

Dado que el servicio devuelve datos aleatorios, el uso de una cache no aportaría ningún beneficio.

La replicación, al distribuir el tráfico de solicitudes entre tres servidores, mejora la capacidad del sistema para manejar la carga, permitiendo una mayor cantidad de respuestas exitosas. En este caso, la replicación distribuye las solicitudes de manera uniforme, disminuyendo la carga en cada instancia de la API y mejorando la disponibilidad del sistema. Con esta estrategia se reducen los tiempos de espera y la tasa de fallos.

La estrategia de rate limiting protege tanto la API externa como el sistema en general de sobrecargas. Aunque esta estrategia resulta en la denegación de algunas solicitudes cuando la carga es muy alta, ayuda a mantener la estabilidad del sistema y a reducir la tasa de fallos al restringir el

tráfico a un nivel manejable. En combinación con la estrategia de replicación, esto asegura que el sistema responde consistentemente a un volumen manejable de solicitudes sin saturar los recursos.

7. Conclusiones

Analizando los gráficos de cada servicio se concluye lo siguiente sobre las estrategias testeadas:

La estrategia de caché reduce la dependencia de la API externa y aumenta la performance cuando se trata de una respuesta que no cambia muy frecuentemente, por lo que es útil para el servicio de Spaceflight News. No vale la pena aplicarla cuando los datos varían mucho en cada solicitud.

La estrategia de replicación disminuye el tiempo de respuesta y el consumo de recursos (por cada servidor), y aumenta la disponibilidad del sistema. Además, resultó muy exitoso usarla en conjunto con rate limiting.

La estrategia de rate limiting mejoró el tiempo de respuesta y estabilidad en los casos testeados, protegiendo los recursos del sistema, pero se debería usar con criterio ya que puede empeorar la experiencia de usuario cuando el límite es más bajo que la capacidad de la API.

8. Modificaciones de la re-entrega

Se agregaron los diagramas Componentes y Conectores para cada estrategia

Se agregaron test de estrés para cada endpoint.

Se incremento el tiempo de tests para tener mas puntos de datos en las métricas.

Se agregaron métricas propias (tiempo de respuesta de endpoint y de la API externa)