Arquitectura de Software

75.73

Trabajo Práctico 1

Villordo, Micaela	103828
Palazon, Martín	102679
Covela, Maximiliano Gastón	102547

Octubre 2023

Índice

1. INTRODUCCIÓN

Se implementan cuatro servicios para los cuales se requiere analizar el impacto de dististas tácticas, al someterse a pruebas de carga y picos.

Se utilizaron las siguientes tecnologías:

- Node.js (+Express)
- Docker
- Docker Compose
- Nginx
- Redis
- Artillery + cAdvisor + StatsD + Graphite + Grafana (para tomar las mediciones correspondientes)

2. ESCENARIOS

Al diseñar las pruebas, se consideró pernitente personalizar los escenarios de prueba en base a la demanda esperada de cada servicio. Ergo se definió un escenario por servicio en el cual se intercalan las tacticas, para su correcta comparacion.

La idea general de los escenarios, para los distintos endpoints de la API, es someterlos a distintas fases de pruebas de carga (Load Test) y de pruebas de pico (Peak Test). Las pruebas de carga mantienen constante una cierta exigencia del servicio por un tiempo prolongado, por otro lado, las pruebas de pico aumentan la demanda del servicio durante un tiempo menor. Ambas fases se encuentran intercaladas en cada escenario.

2.1. Ping

Dado que este servicio no conecta con otra API, y solo devuelve un valor constante en todos sus llamados con código 200 OK. Se armó un escenario con mayor demanda de request que las demas para detectar el punto límite en donde empieza a fallar, empezando por fases más leves para confirmar su correcto funcionamiento en los casos triviales.

El escenario se definió:

```
config:
  environments:
    api:
      target: 'http://localhost:5555/api'
      plugins:
        statsd:
          host: localhost
          port: 8125
          prefix: "artillery-api"

  pool: 50 # All HTTP requests from all virtual users will be sent over the same
    connections

  phases:
    - name: Light Plain
      duration: 120
      arrivalRate: 1
    - name: Light Ramp
      duration: 60
      arrivalRate: 1
      rampTo: 20
    - name: Intermediate Plain
      duration: 120
      arrivalRate: 20
    - name: Heavy Ramp
      duration: 60
      arrivalRate: 20
      rampTo: 150
    - name: Heavy Plain
      duration: 120
      arrivalRate: 150

scenarios:
  - name: Ping Load Test (/)
    flow:
      - get:
          url: '/ping'
```

2.2. SpaceFlight_News

En este escenario se someterá al servicio SpaceFlight News a una serie de pruebas que puedan simular escenarios reales de baja, mediana y gran demanda. Finalmente aumentando más allá de lo esperable de un servicio de noticias, se busca encontrar el punto crítico del servicio.

En el siguiente bloque de código se puede ver cómo se armó el escenario:

```
config:
  environments:
    api:
      target: 'http://localhost:5555/api'
      plugins:
        statsd:
          host: localhost
          port: 8125
          prefix: "artillery-api"

  pool: 50 # All HTTP requests from all virtual users will be sent over the same
    connections

  phases:
    - name: Light Plain
      duration: 120
      arrivalRate: 1
    - name: Light Ramp
      duration: 60
      arrivalRate: 1
      rampTo: 15
    - name: Intermediate Plain
      duration: 120
      arrivalRate: 15
    - name: Heavy Ramp
      duration: 60
      arrivalRate: 15
      rampTo: 60
    - name: Heavy Plain
      duration: 120
      arrivalRate: 60
    - name: Extreme Ramp
      duration: 60
      arrivalRate: 60
      rampTo: 200
    - name: Extreme Plain
      duration: 120
      arrivalRate: 200

scenarios:
  - name: SpaceFlight Load Test (/)
    flow:
      - get:
          url: '/spaceflight_news'
```

2.3. Quote

El servicio Quote será puesto a prueba contra bajos volúmenes de tráfico base, se busca simular una demanda factible. Luego se aumentará hasta conseguir el límite de funcionalidad del servicio.

En el siguiente bloque de código se puede ver cómo se armó el escenario:

```
config:
  environments:
    api:
      target: 'http://localhost:5555/api'
      plugins:
        statsd:
          host: localhost
          port: 8125
          prefix: "artillery-api"

  pool: 50 # All HTTP requests from all virtual users will be sent over the same
    connections

  phases:
    - name: Light Plain
      duration: 120
      arrivalRate: 1
    - name: Light Ramp
      duration: 60
      arrivalRate: 1
      rampTo: 20
    - name: Intermediate Plain
      duration: 120
      arrivalRate: 20
    - name: Heavy Ramp
      duration: 60
      arrivalRate: 20
      rampTo: 150
    - name: Heavy Plain
      duration: 120
      arrivalRate: 150

scenarios:
  - name: Ping Load Test (/)
    flow:
      - get:
          url: '/ping'
```

2.4. Metar

En este escenario se someterá al servicio Metar a una serie de pruebas que puedan simular escenarios reales de baja, mediana y gran demanda. Al tratarse de información de aeropuertos, se espera un uso menos informal y abarcativo que los escenarios anteriores. Se busca encontrar el punto crítico del servicio.

En el siguiente bloque de código se puede ver cómo se armó el escenario:

```
config:
  environments:
    api:
      target: 'http://localhost:5555/api'
      payload:
        path: "stations.csv"
        order: sequence
        fields:
          - "station"
      plugins:
        statsd:
          host: localhost
          port: 8125
          prefix: "artillery-api"

  pool: 50 # All HTTP requests from all virtual users will be sent over the same
    connections

  phases:
    - name: Light Plain
      duration: 120
      arrivalRate: 10
    - name: Light Ramp
      duration: 60
      arrivalRate: 10
      rampTo: 60
    - name: Intermediate Plain
      duration: 120
      arrivalRate: 60
    - name: Heavy Ramp
      duration: 60
      arrivalRate: 60
      rampTo: 200
    - name: Heavy Plain
      duration: 120
      arrivalRate: 200
    - name: Extreme Ramp
      duration: 60
      arrivalRate: 200
      rampTo: 300
    - name: Extreme Plain
      duration: 120
      arrivalRate: 300

scenarios:
  - name: Metar Load Test (/)
    flow:
      - get:
          url: '/metar?station={{ station }}'
```

3. TÁCTICAS

Se procede a mostar los gráficos obtenidos realizando las pruebas con cada táctica en cada uno de los servicios, con el fin de poder realizar una comparación entre ellas. Los gráficos mostrarán información sobre:

- **Scenarios launched** - Cantidad de requests cada 10 segundos
- **Request state** - Estados de los requests, siendo *Completed* los exitosos con código 200, *Errored* los fallidos, por ejemplo, por timeout, *Pending* los pendientes y *Limited* los que dieron código 429 (Too many requests)
- **Response time (client-side)** - Tiempo de respuesta medido desde el lado del cliente
- **Response time (external API)** - Tiempo de respuesta del servicio externo
- **Response time (external API + own processing)** - Tiempo de respuesta del servicio externo más el tiempo de procesamiento propio de nuestro servicio

  Aclaración: Se ha observado que en todas las corridas de artillery, los gráficos **Response time (client-side)** y **Response time (external API + own processing)** muestran siempre la misma información, por lo que decidimos sólo mostrar las capturas del primer gráfico para no redundar.

3.1. CASO BASE

En esta sección se realizarán mediciones de los distintos servicios implementados, pasando por medio de nginx actuando como reverse proxy y al cual se comunicarán los clientes, sin utilizar tácticas adicionales tales como caché, replicación, etc.

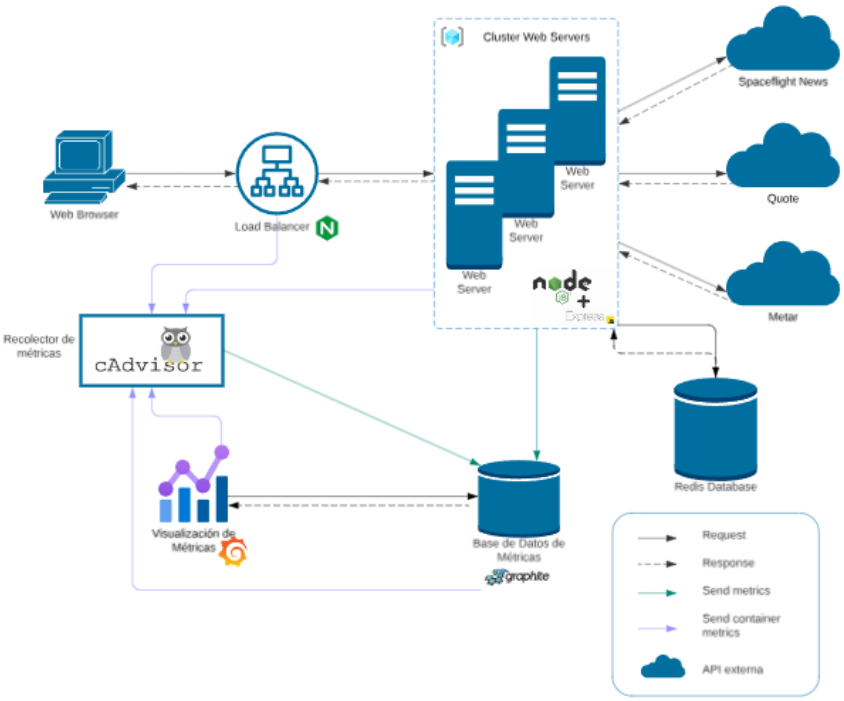

Figura 1: Vista Components & Connectors - Caso Base

3.1.1. Ping

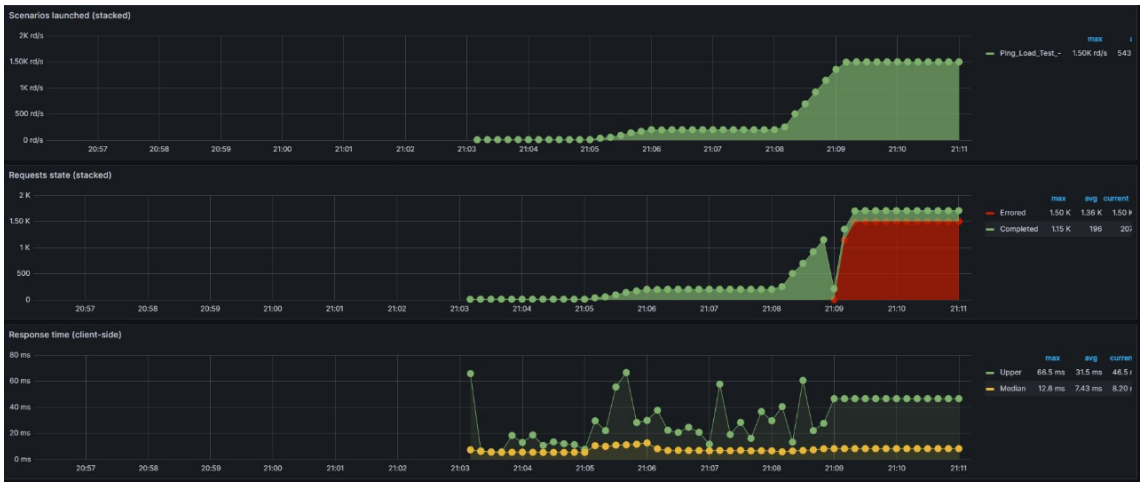

Figura 2: Ping Load Test - Caso Base

Para el Load Test se observa que el servicio responde exitosamente con código 200 por un largo período de tiempo, comenzado a aumentar y fluctuar el tiempo de respuesta visto por el cliente en el momento de comenzar la segunda fase *Light Ramp* en donde se aumenta progresivamente los usuarios de forma liviana durante 60 segundos.

Continúan las fluctuaciones del tiempo de respuesta visto por el cliente hasta que llega a un punto, en medio de la fase de *Heavy Ramp* en donde aumenta de forma más agresiva la cantidad de usuarios a lo largo de 60 segundos. Se cae totalmente el servicio y comienza a dar errores de Timeout en todos sus requests.

En este lapso se destaca que el response time visto por el cliente es constante, lo cual no significa que esté respondiendo lo esperado, sino que a todos les está dando el mismo error con el mismo tiempo de espera.

Dado que este servicio no consume ningúna API externa ni realiza ningún proceso complejo y viendo que la caída ocurrió también cuando se estaba consumiendo un porcentaje importante del procesador del servidor, se deduce que la abrupta caída se da debido a los límites de hardware del propio servidor, ya que al llegar al límite de consumo, falla todo lo que queda.

3.1.2. SpaceFlight_News

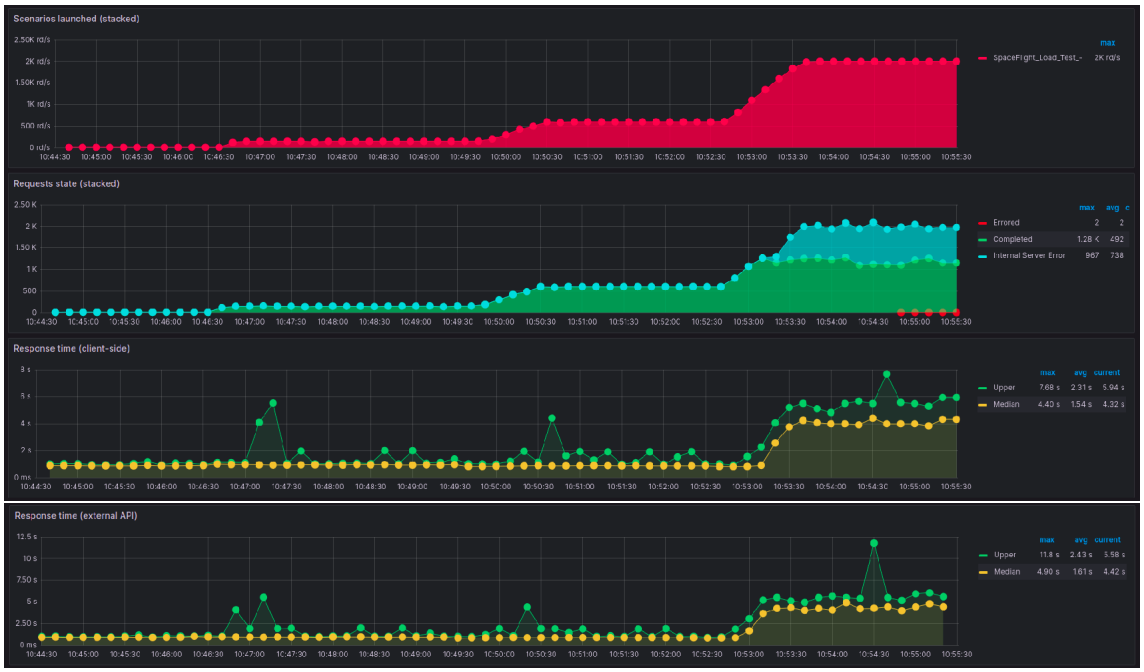

Figura 3: Spaceflight News Load Test - Caso Base

Para este servicio, se puede observar que los tiempos de respuesta son más elevados que en el caso del Ping, debido a que este endpoint debe consultar a la API externa de Spaceflight News para poder generar una respuesta. Con respecto a esto, también se observa que los tiempos de respuesta del endpoint se condicen con los de la API externa, lo que significa que el tiempo de procesamiento es despreciable en comparación al tiempo de consulta a la mencionada API.

Al principio, con una carga más baja (de 1 a 60 requests por segundo) la mediana de los tiempos de respuesta es de 1 segundo (como se mencionó, es un tiempo más elevado que el del Ping, por consultar una API externa), contando con algunos picos de más demora en responder (varios de 2 segundos y unos pocos de 5 segundos) como se condicen con picos de tiempo de respuesta en la API externa, se deduce que etsos pequeños picos pueden deberse a momentos en donde la API de Spaceflight News tuvo más tráfico en general por parte de otros usarios que también la consulten.

Luego, con una carga más elevada, en la fase de *Extreme Ramp*, puede verse que los request comienzan a fallar en gran medida (un tercio de los request enviados fallan) con error 500 (Internal Server Error) al mismo tiempo que tanto la API propia como la externa aumentan considerablemente sus tiempos de repsuesta (pasan de una mediana de 1 seg con picos de 2 y 5 seg, a una mediana de 5 seg con picos de 6 y 8 seg aprox.).

Esto implica que el limitante en este caso no fue el servidor que hostea la API propia, sino que fue la API externa, ya que hay muy pocos casos (2) donde el error es de Timeout (que se deben a la falta de recursos de hardware) y una gran cantidad de errores (en el orden de 1000) que se deben a la falla de la API externa.

3.1.3. Quote

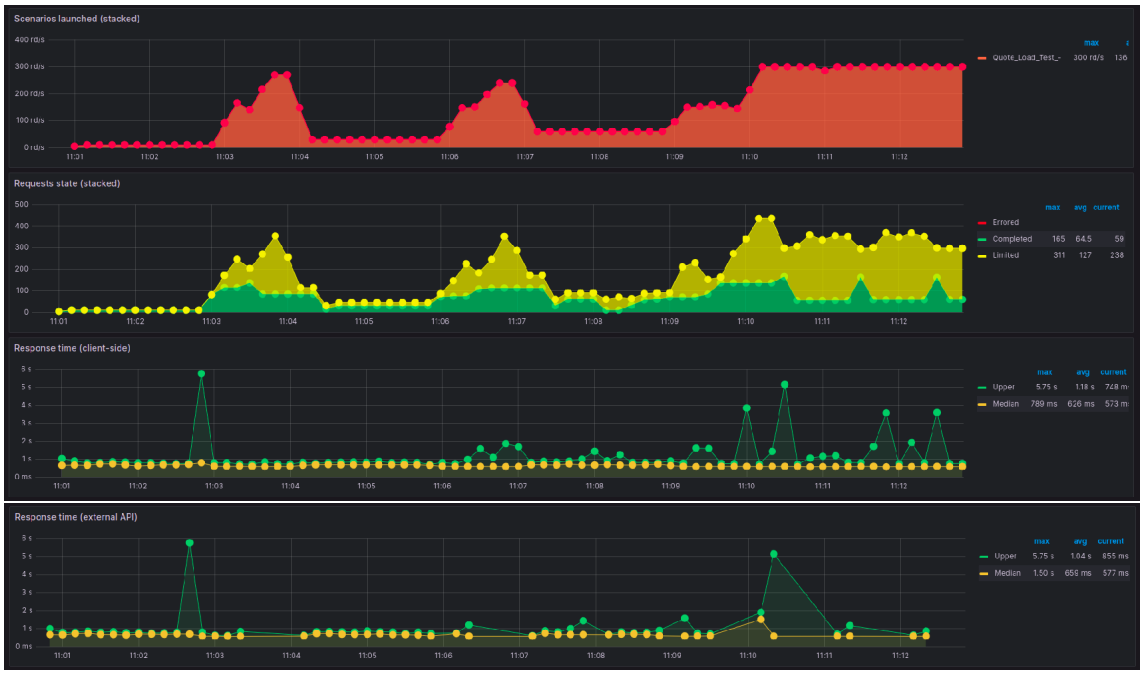

Figura 4: Quote Load Test - Caso Base

En el caso del endpoint de quote, la documentación de la API externa correspondiente indica lo siguiente: There is a rate limit of 180 requests per minute, per IP address. If you exceed the rate limit, the API will respond with a 429 error.

Por lo tanto, las fases del escenario de quote tienen una carga relativamente baja en comparación a las de spaceflight, ya que al llegar a las 180 requests por segundo, la API externa de quote se satura. Es por esto que en el gŕafico se ve que al ascender la carga a 3 requests por segundo en la fase *Light Ramp*, la API propia ya empieza a fallar devolviendo el error 429 (que viene de forwardear el error de la API externa), ya que 3 requests por segudno equivalen aproximadamente a 180 requests por minuto, rate limit en donde la API externa empieza a fallar. Luego puede verse que la API continúa respondiendo correctamente un cierto porcentaje de requests, pero la mayor parte devuelve error, debido a que la API externa que consume el endpoint de quote ya se encuentra saturada.

Puede observarse también que los tiempos de respuesta de la API propia y los de la API externa se condicen, lo que lleva a concluir que el tiempo de procesamiento en el endpoint de quote es mínimo/despreciable. Y otra cosa a destacar es que el tiempo de respuesta comienza a presentar mayor cantidad de picos en las últimas fases de *Extreme Ramp* y *Extreme Plain* que es cuando las APIs (tanto la propia como la externa) comienzan a devolver el error 429 (Reflejado como Limited en el dashboard).

3.1.4. Metar

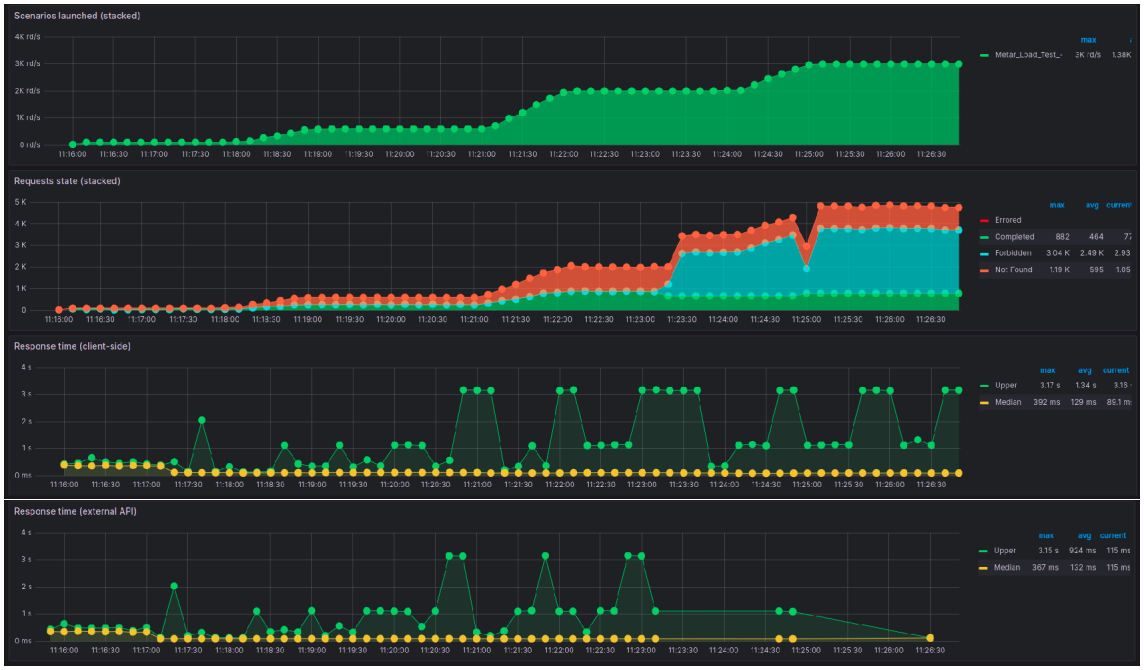

Figura 5: Metar Load Test - Caso Base

En el caso del endpoint de metar, puede observarse que a lo largo de la corrida ha devuelto el error 404, esto ocurre debido a que hay códigos de estaciones para los cuales la API externa no encuentra información para devolver en su respuesta a la API propia o devuelve información que no se puede parsear.

También se observa que al llegar a la fase de *Heavy Plain* la API ya comienza a devolver el error 403 (Forbidden), forwardeado de la API externa que devuelve ese error al fallar por haberse realizado demasiadas requests (en el caso de esta fase, 200 req por segundo).

Otro aspecto que cabe destacar es que los tiempos de respuesta de la API propia y de la API externa se condicen hasta llegar a la fase de *Heavy Plain* cuando los request a la API externa comienzan a devolver el error 403. Esto es así ya que a partir de ese momento el tiempo de respuesta de la API externa empieza a bajar debido a que deja de responder las request enviadas, pero el tiempo de respuesta de la API propia no baja debido a que debe seguir respondiendo las request enviadas, aunque sea devolviendo un error.

3.2. CACHÉ

En esta seccion se realizaran mediciones de los distintos servicios implementados, pasando por medio de nginx actuando como reverse proxy y agregando Redis como database de la informacion en cache.

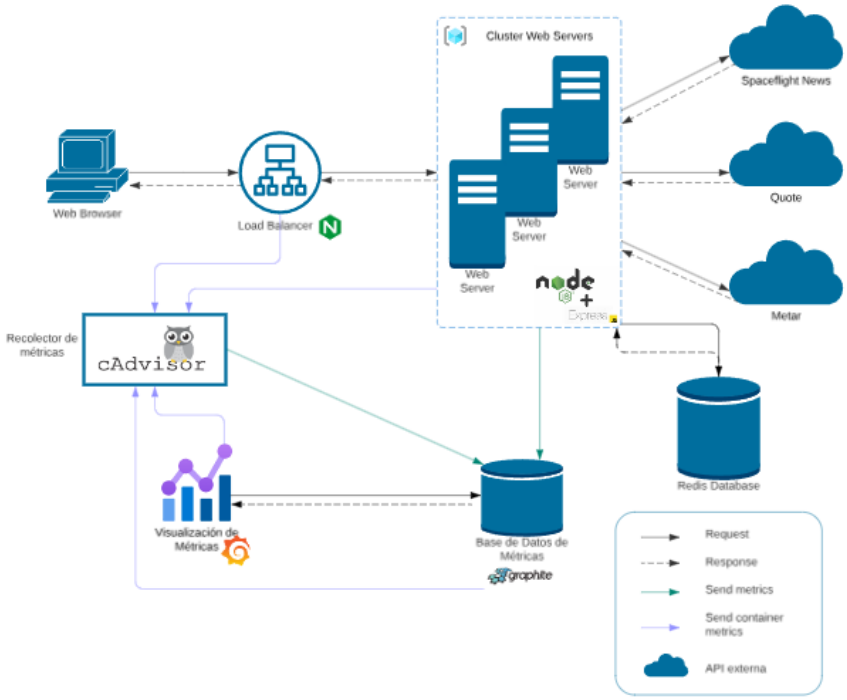

Figura 6: Vista Componentes - Caso cache

3.2.1. Ping

El servicio ping, al no necesitar una request de una API externa, no fue implementado ningun sistema de cache.

3.2.2. SpaceFlight_News

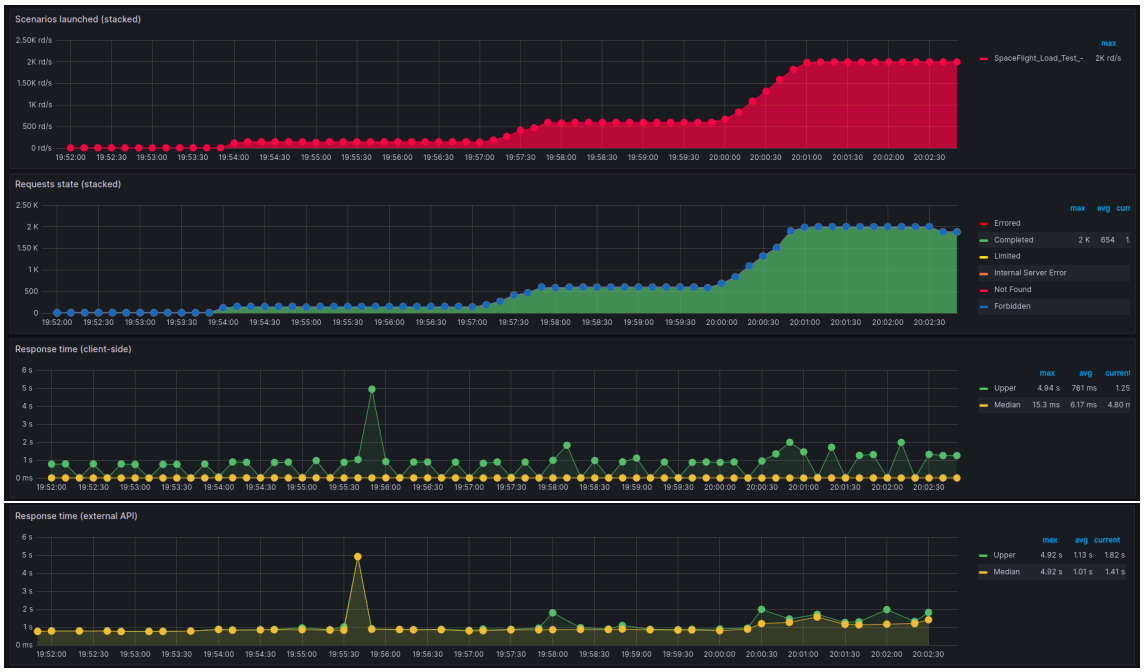

Figura 7: SpaceFlights Load Test - Caso cache

Este escenario fue construido haciendo caché de las respuestas anteriores, manteniendo la informacion generada 15 segundos.

Esto se puede notar en los picos a intervalos regulares del response time".

Se destaca una gran diferencia en la mediana, respecto del caso base (4 segundos vs 15 ms). Esto se explica con que la obtencion de la informacion, en la mayoria de los requests, al ser buscada en una base de datos, tiene una lectura casi inmediata.

Por otro lado se resalta la ausencia de errores, estos presentes en el caso de prueba en los casos mas extremos.

En conclusion, al cachear la informacion, se encuentran picos de mayor demanda a intervalos esperables por el provedor de servicio, y ademas se aumenta la disponibilidad. Se obtiene un servicio mas escalable.

3.2.3. Quote

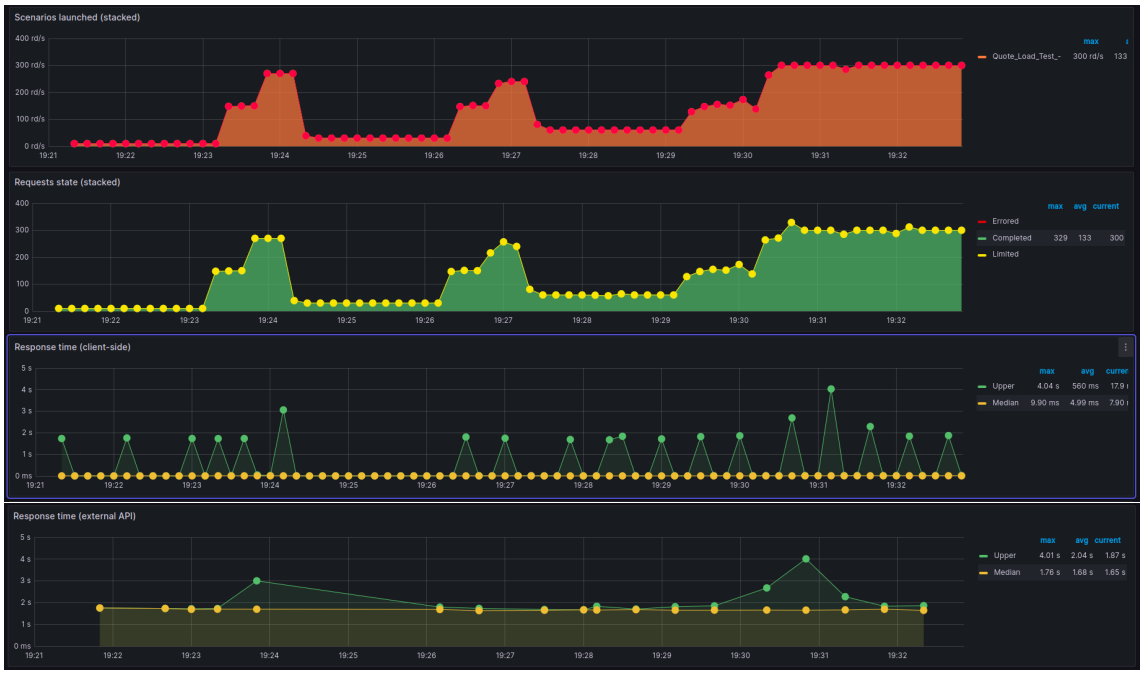

Figura 8: Quote Load Test - Caso cache

Este escenario fue construido guardando la respuesta de cincuenta requests cada vez que se consultaba con la API externa. Cada respuesta, al necesitarse que sea random, se eliminó una vez usada. Al encontrarse vacio el cache, se volvia a consultar con la API.

Esto se puede notar en los picos a intervalos irregulares del response time", estos aparecen mas cercanos en los momentos de mayor demanda.

Se destaca que la totalidad de los requests"terminó sin errores (vease seccion Requests State (stacked)), el caso base excedia los requests maximos de la API, concluyendo en un "Limited".

Por otro lado se resalta que la API externa fue consultada menos veces, por mas informacion por request que el caso base, y sin embargo, la mediana del response time (external API)"fue practicamente igual al caso base.

En conclusion, al cachear la informacion, se encuentran picos regulares de demanda a intervalos esperables por el provedor de servicio, y ademas se logra una menor dependecia del servicio de API externo.

3.2.4. Metar

Figura 9: METAR Load Test - Caso cache

Este escenario fue construido haciendo caché de las respuestas anteriores, manteniendo la informacion generada 15 segundos. Se consultó por un aeropuerto distinto cada vez, tomando de una muestra reducida que se repetia una vez finalizada.

Esto llevó a que su efectividad crezca al aumentar la cantidad de requests por segundo. Ya que se consultaba por informacion en cache.

Se destaca la menor cantidad de picos en el Response time (client-side)çontra el caso base, haciendo que los clientes perciban un servicio mas rapido.

Por otro lado se resalta la casi ausencia de errores, estos presentes en el caso de prueba durante toda la ejecucion. Con la misma cantidad de requests, el caso base obtiene la mayoria de respuestas en error, en cambio, con la informacion en cache se obtiene un error despreciable, obtenidos en los momentos de mayor demanda.

En conclusion, al cachear la informacion, se encuentran menos picos de mayor demanda a intervalos esperables por el provedor de servicio, y ademas se aumenta la cantidad de respuestas correctas. Se obtiene un servicio que se percibe como mas rapido y mas confiable por el usuario.

3.3. REPLICACIÓN

La táctica de replicación consiste en escalar el servicio a 3 copias, convirtiendo a nginx en un load balancer. En este caso, ya que los tres servidores presentan las miasmas características, el load balancer dividirá las request entre las réplicas utilizando el algoritmo por defecto, Round Robin, que distribuye las solicitudes de manera uniforme entre los servidores en secuencia.

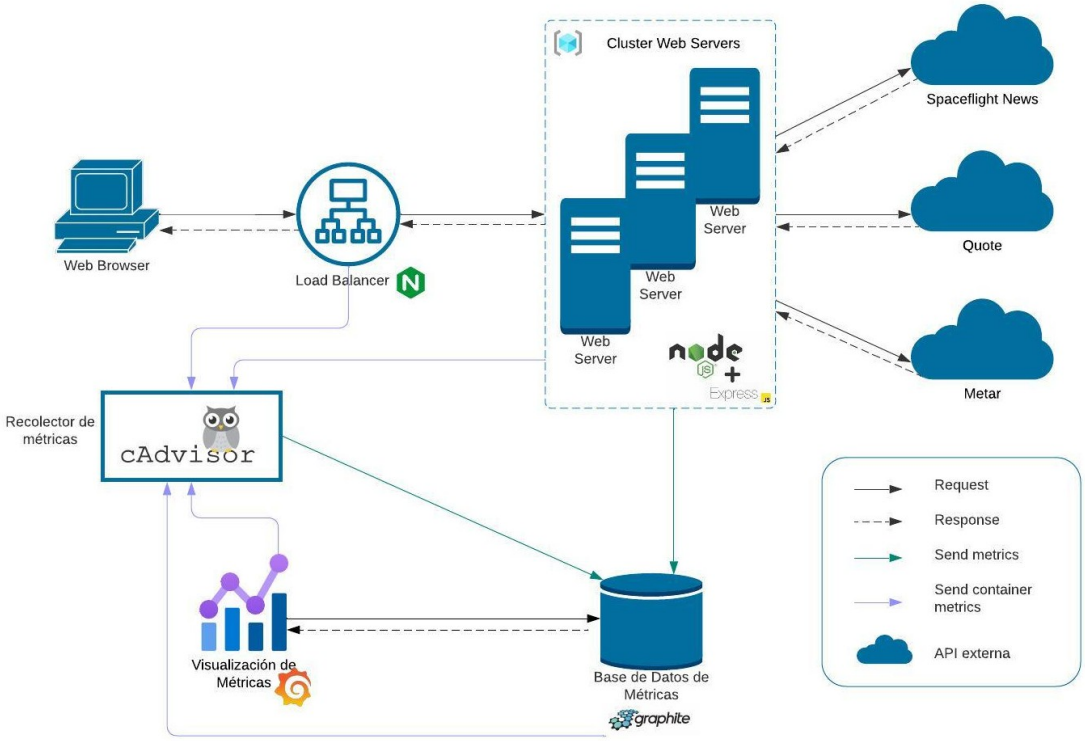

Figura 10: Vista Components & Connectors - Replicación

Para comprobar que las 3 réplicas se hayan levantado correctamente y que todas estén recibiendo tráfico, además de verificar el estado de los contenedores con el comando docker-compose ps, se pueden realizar pegadas al endpoint de ping y observar que el valor del header 'X-API-Id' va variando, lo que significa que las solicitudes están llegando a las distintas réplicas del servicio.

```
$ curl -vvv  localhost:5555/api/ping | grep 'X-API-Id'
< X-API-Id: eCxLN5U_hBWxB0lG-PSuM
< X-API-Id: I5iivQZ7P_XbdnpzzlyDJ
< X-API-Id: -KyrWk-UKsXOEgVOK8WGN
```

Figura 11: Comprobación de llegada de requests a las 3 réplicas

3.3.1. Ping

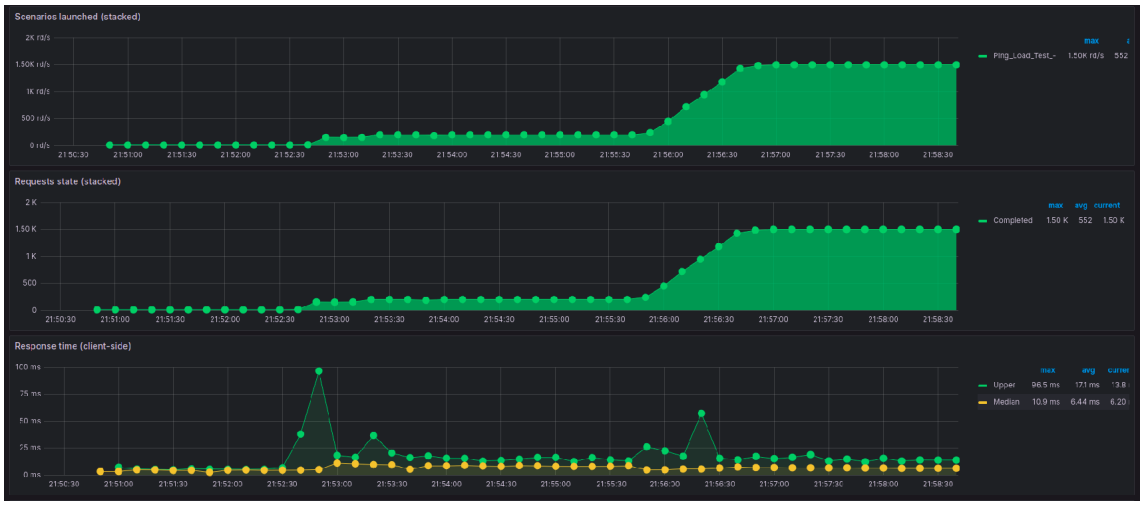

Figura 12: Ping Load Test - Caso Replicación

Puede observarse que al haber agregado 2 instancias más de la API propia (contando ahora con un cluster de servers), el servicio de Ping sometido a altas cargas ya no falla por Timeout y devuelve siempre código 200 con tiempos de respuesta cuya mediana ronda los 10 ms.

Esto se debe a que el limitante en el caso base era el hardware del propio servidor, por lo que al haberlo escalado horizontalmente agregando 2 instancias más, este limitante ya no es un problema y la API cuenta con un hardware lo suficientemente sólido para poder responder a las pruebas de carga sin fallar,

En resumen, esta tática mejora la disponibilidad del sistema ya que aumenta el tiempo que el sistema se encuentra disponible para que el usuario lo consulte (en el caso base al llegar a la carga del *Heavy Ramp* el sistema dejaba de estar disponible para los usuarios, devolviendo un error 500; ahora con esta táctica el sistema soporta el mal funcionamiento de uno de sus componentes y si un servidor se encuentra saturado, cuenta con otros dos que puedan responder correctamente al usuario que se encuentre enviando requests a la API).

También mejora la performance, ya que en el caso base a medida que el servidor se saturaba por consumo máximo del procesador, la API demoraba más en responder a los usuarios, ahora con la redundancia agregada esto ya no ocurre y la API hasta en los escenarios de más carga responde a los usuarios en un tiempo menor que en el caso base.

Por último, la replicación también mejoró la fiabilidad, ya que en el caso base sólo se tenía un servidor y si este fallaba, fallaba la API (punto único de falla). Ahora al contar con 3 servidores, si uno falla, hay otros dos que pueden seguir funcionando para atender las requests.

3.3.2. SpaceFlight_News

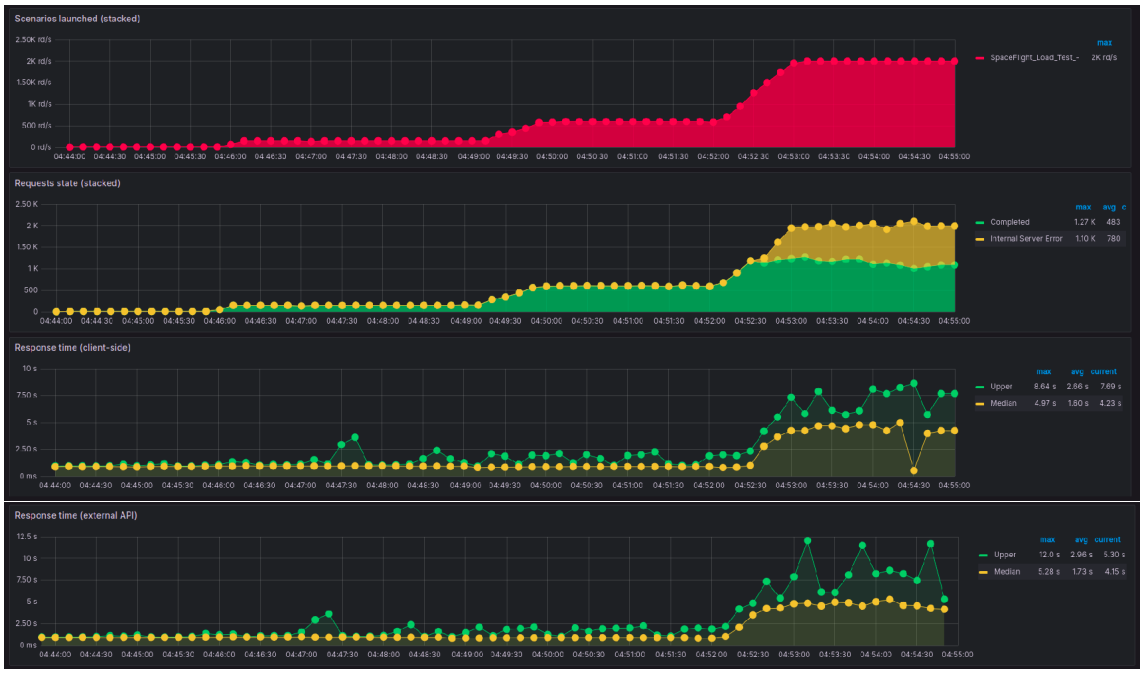

Figura 13: SpaceFlight News Load Test - Caso Replicación

Puede observarse que los 4 gráficos son muy similares a los del caso base de Spaceflight News. Esta táctica no fue beneficiosa, ya que el limitante en el caso base era la API externa, por lo que haber agregado más servidores no cambia el hecho de que la API externa comienza a fallar cuando la carga comienza a subir a 200 requests por segundo en la fase de *Extreme Ramp*.

Lo que sí cabe destacar es que ya no se dan erroes por Timeout (si bien eran muy pocos en el caso base, existían algunos de estos errores), esto se debe a la redundancia agregada ya que antes el procesador se encontraba sobrecargado al final de la fase de *Extreme Ramp* y comenzaba a dar error de Timeout para algunas pocas request, ahora que el sistema escaló a tres servidores, esto ya no ocurre.

3.3.3. Quote

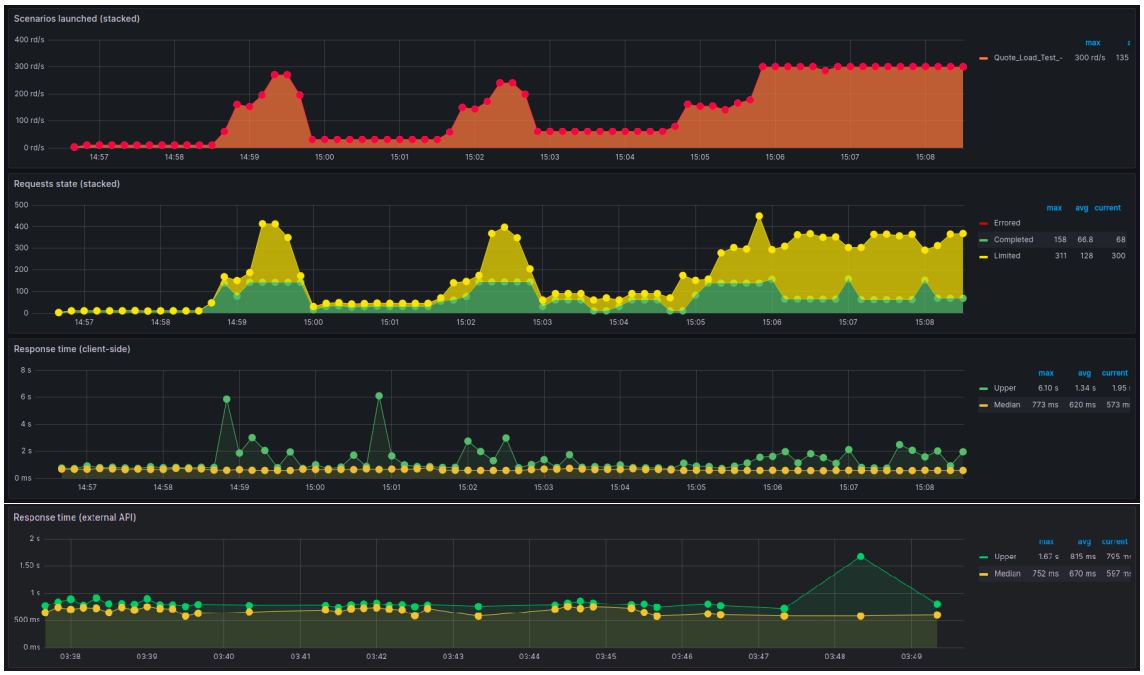

Figura 14: Quote Load Test - Caso Replicación

Así como en el endpoint de Spaceflight News, el limitante en Quote también es la API externa. Esta comienza a fallar en la fase de *Light Ramp* con una carga muy baja de 3 requests por segundo, así como lo hacía en el caso base.

Se concluye que esta táctica no presenta ninguna mejora evidenciable en los dashboards.

3.3.4. Metar

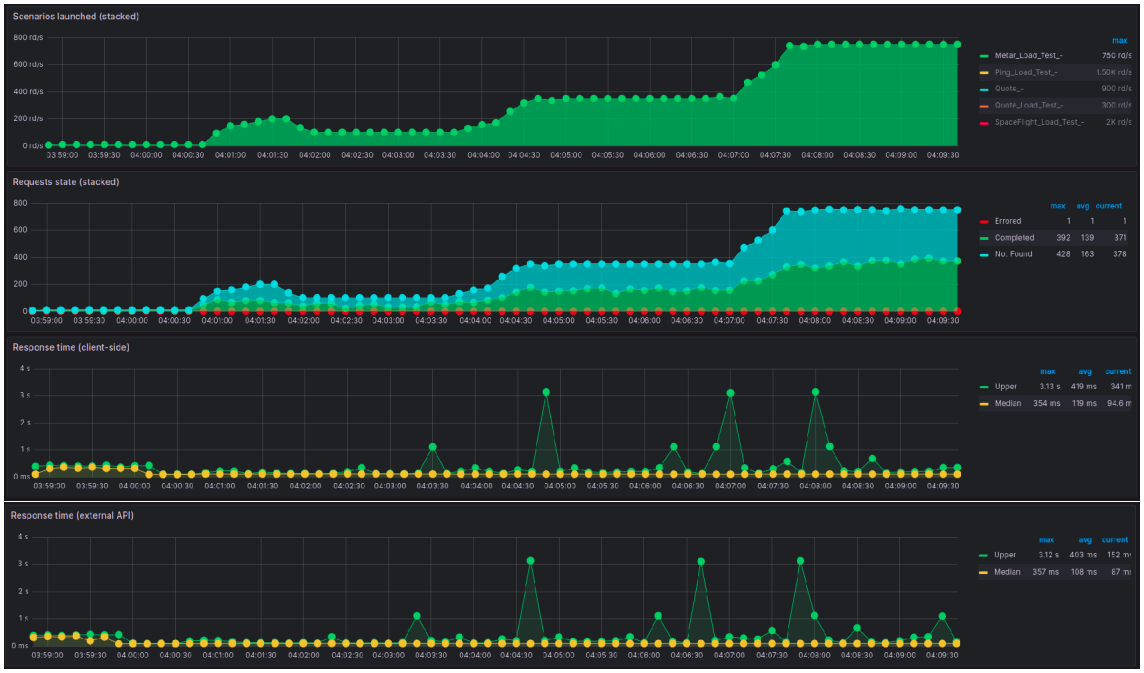

Figura 15: Metar Load Test - Caso Replicación

Así como ocurrió con los endpoints Spaceflight News y Quote, esta tática no presenta ninguna mejora con respecto al caso base, ya que el limitante es también la API externa y agregar más servidores no genera ninguna mejora sobre la mencionada API. Los errores Not Found y Forbidden presentan una cantidad muy similar a la hallada en el caso base y los tiempos de respuesta también se asemejan. En conclusión esta táctica no trajo beneficios.

3.4. RATE LIMITING

En esta táctica le agregaremos al caso base una limitación de requests por segundo para cada IP mediante nginx como reverse proxy.

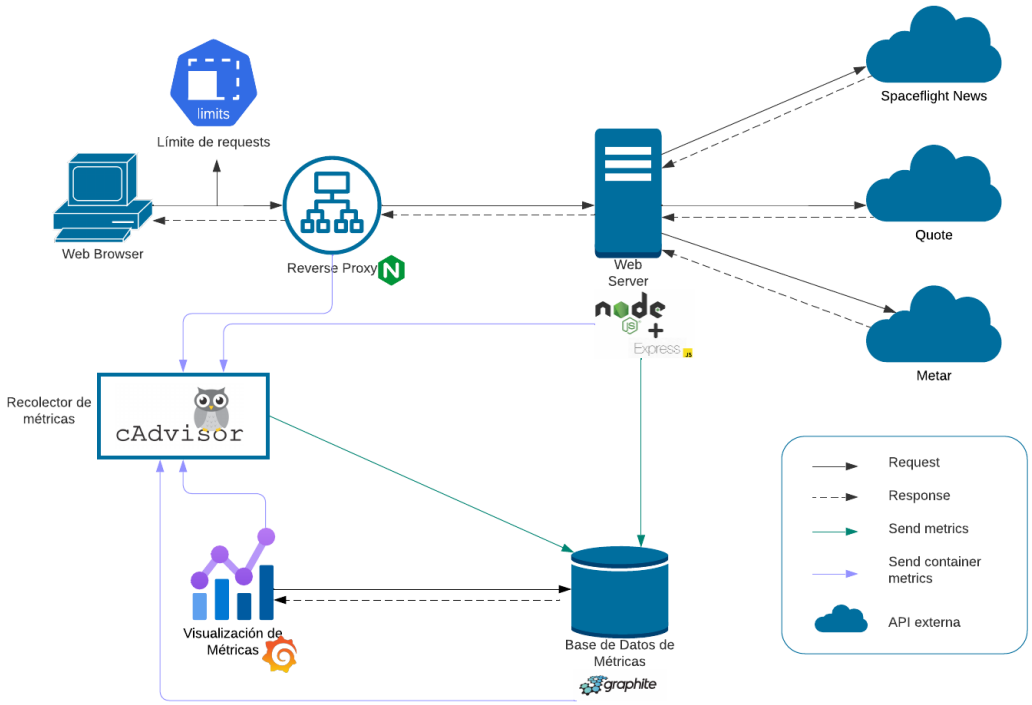

Figura 16: Vista Components & Connectors - Rate Limiting

3.4.1. Ping

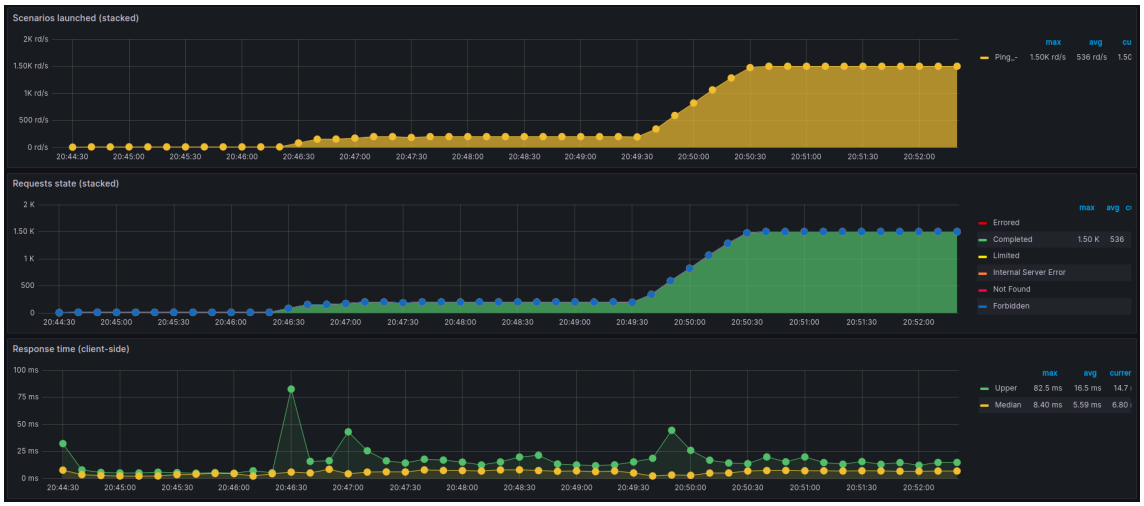

Figura 17: Ping Load Test - Caso Rate Limiting

En este caso, comparando con la prueba del Ping en el caso base, podemos ver que ya no falla en ningún momento del escenario dado que se limitan los requests que van a llegar y no genera un pico en el consumo de recursos del servidor, por lo tanto el servicio se mantiene vivo y respondiendo exitosamente durante todas las fases.

3.4.2. SpaceFlight_News

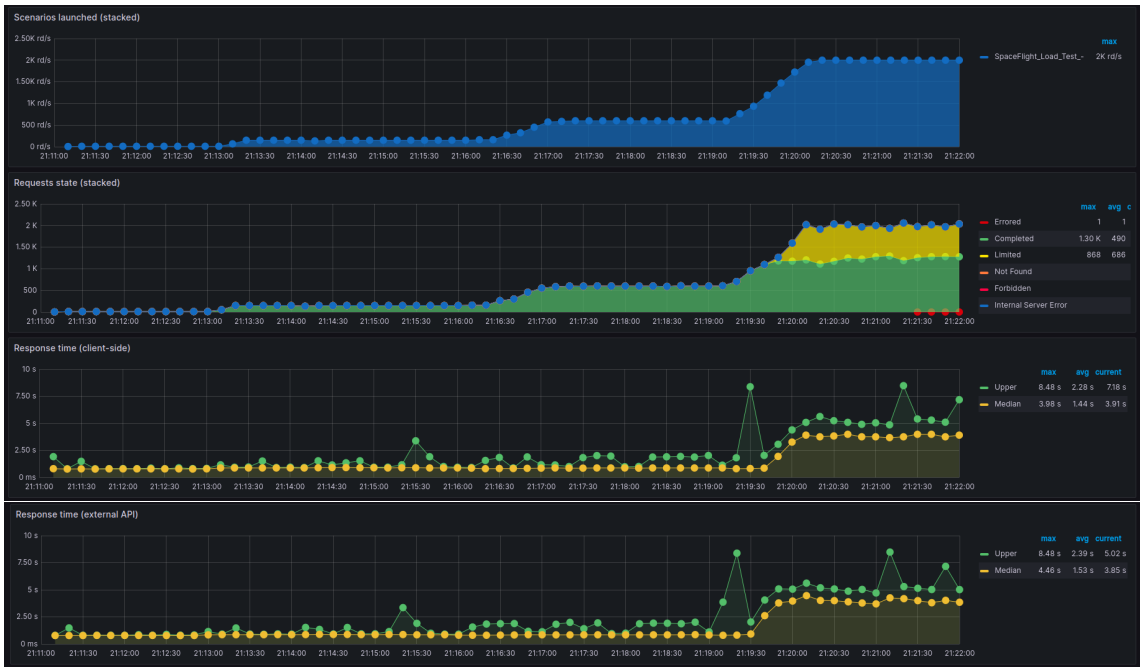

Figura 18: Spaceflight News Load Test - Caso Rate Limiting

Vemos que en comparación con el caso base, en este caso obtenemos más respuestas con código 429 (Limited) ya que estamos limitando los requests que pueden llegarnos. De esta forma evitamos saturar el servicio externo y podemos deshacernos de los códigos 500. No hubo una diferencia significativa en los tiempos de respuesta.

3.4.3. Quote

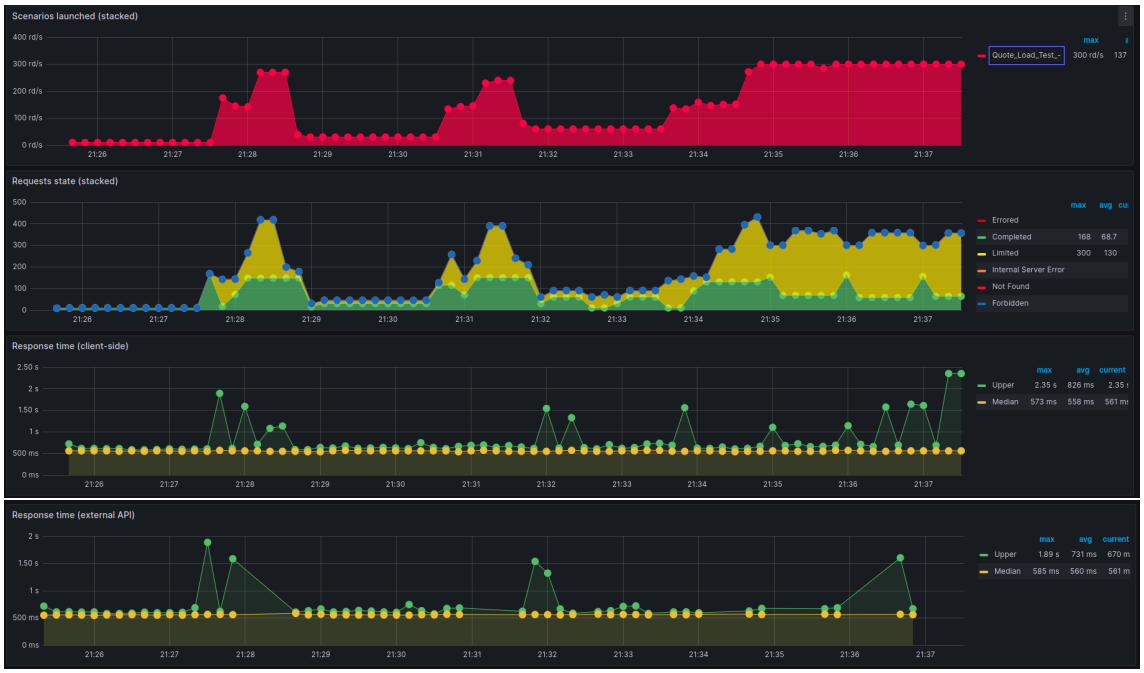

Figura 19: Spaceflight News Load Test - Caso Rate Limiting

Para este servicio, con el rate limiting pudimos eliminar las respuestas erróneas de la API externa, dado que lo evitamos devolviendo el error apropiado desde el propio servidor, en este caso también 429 (Too Many Requests). La única diferencia es en quién responde esto.

3.4.4. Metar

Figura 20: Spaceflight News Load Test - Caso Rate Limiting

Al igual que con los anteriores, obtenemos más errores con código 429 cuando empieza a limitar los requests gracias a la limitación por parte del reverse proxy. Vemos que los códigos que anteriormente eran 403 Forbidden ahora son 429. Si hubiera casos que no encuentra información, seguiría devolviendo los 404 dado que eso depende del resultado del servicio externo.

4. CONCLUSIONES

De las 3 tácticas puestas en práctica, la que mejores resultados dio fue la de utilizar un caché, ya que permitió que los endpoints Spaceflight News y Quote ya no fallaran y que el endpoint Metar reduzca considerablemente la cantidad de errores devueltos. Sin embargo, al ser una táctica que sólo implica agregar un caché no generó cambios sobre el comportamiento del Ping (el cual es tan sólo un healthcheck de la API)

Por otro lado, la táctica de rate limiting permitió que ninguno de los enpoints analizados fallara por errores de Forbidden (403) o 500 (Internal Server Error), sino que al limitar la cantidad de requests que llegan a la API, llevó a que el único error devuelto por dicha API fuera 429. Esto indica que se estaban realizando demasiadas requests (Too Many Requests), es decir, por encima del límite seteado.

Por último, la táctica de replicación sólo mejoró el comportamiento del Ping, por lo que fue la táctica menos útil dentro de los análisis realizados.