[75.73] Arquitectura de Software

Trabajo Práctico N°1

1er Cuatrimestre de 2023

Integrantes:

- 99535 - Agustín Leguizamon
- 99529 - Franco Scaccheri
- 105774 - Nicolas Zulaica
- 105836 - Elián Foppiano

Índice

Proyecto:

Servicio HTTP que representa una API que consume otras APIs para dar información a sus usuarios. Se implementará utilizando Express.

La API será sometida a diversas intensidades/escenarios de carga en algunas configuraciones de deployment. Se tomarán mediciones y analizarán los resultados.

Tácticas implementadas

- Caso base: servirá como baseline para comparar el resto de las tácticas.
- Rate limiting: se configurará nginx para limitar la tasa de consumo de la API.
- Caché: se implementará lazy y active population en los endpoints que correspondan
- Replicación: se replicará el servicio a 3 copias, convirtiendo nginx en un load balancer

Objetivo:

Comparar algunas tecnologías, ver cómo diversos aspectos impactan en los atributos de calidad y probar qué cambios se podrían hacer para mejorarlos.

Aprender a usar una variedad de tecnologías útiles y muy usadas hoy en día, incluyendo: NodeJs, Docker, Nginx, Redis, Artillery, Grafana, Graphite, entre otras.

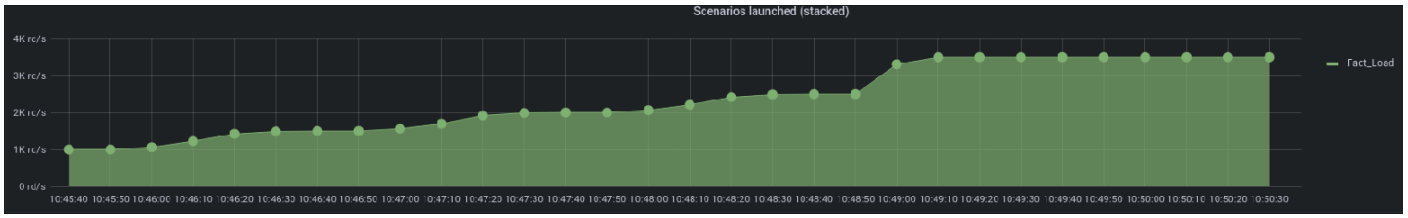
Pruebas de performance

Detalles

Para cada táctica se utilizó una variante personalizada de Load Testing (Load Testing + Stress Testing), la cual consta de varias etapas compuestas por un incremento constante de la cantidad de request por segundo hasta llegar a una meseta en donde el valor se mantiene durante un periodo de tiempo, para luego volver a incrementar la cantidad de request hasta llegar a una nueva meseta, y así sucesivamente hasta llevar al servicio a un punto de carga muy alto y por último terminar con un Load Spike que implica una gran carga de request en un intervalo corto de tiempo.

Este tipo de escenario lo corremos en cada una de las variantes (Caché, Réplica, Rate Limiting) y el Caso Base (sin ninguna táctica) de manera de poder obtener métricas de cada uno de ellos para poder hacer una comparación.

Como ejemplo mostramos uno de los gráficos en donde se puede apreciar el modelo de Load Testing que usamos.



Cantidad de requests cada 10 segundos

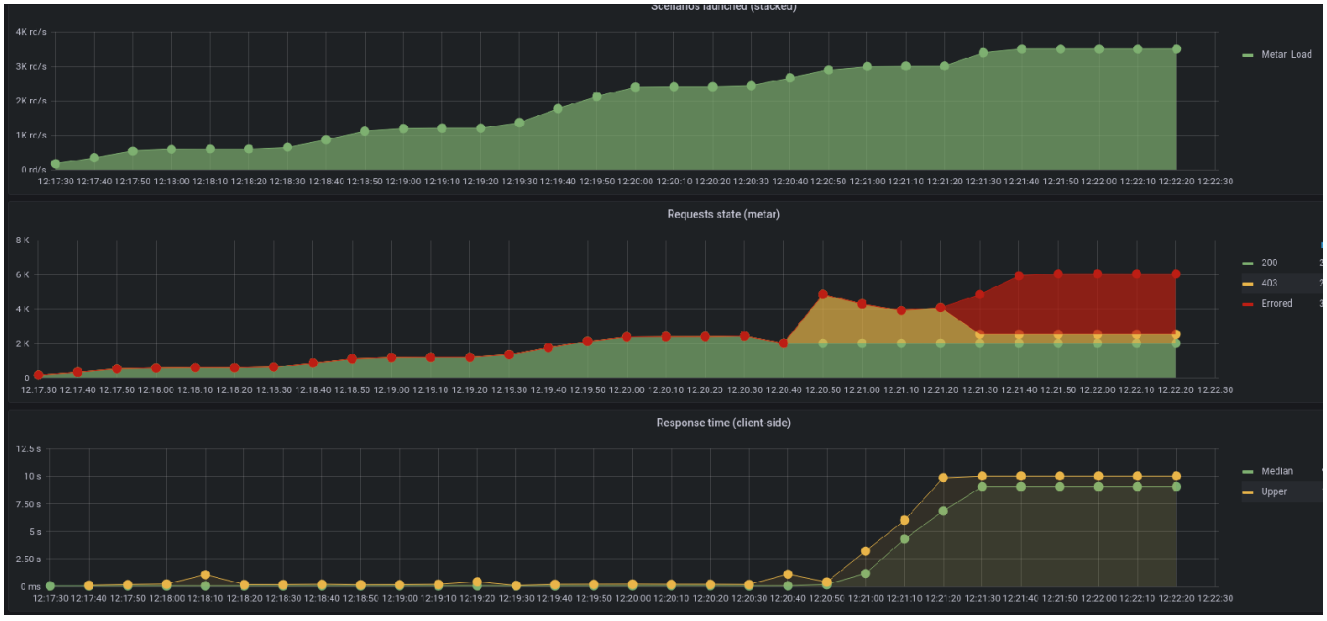
Descripción de los gráficos

Para entender mejor los gráficos que se mostrarán en cuanto a los resultados de cada una de las Pruebas de Performance, se dará una breve explicación de qué información da cada uno de los paneles de Grafana.

Tomamos como ejemplo el gráfico de Metar ante un Load Testing sin ninguna táctica aplicada (Caso Base).

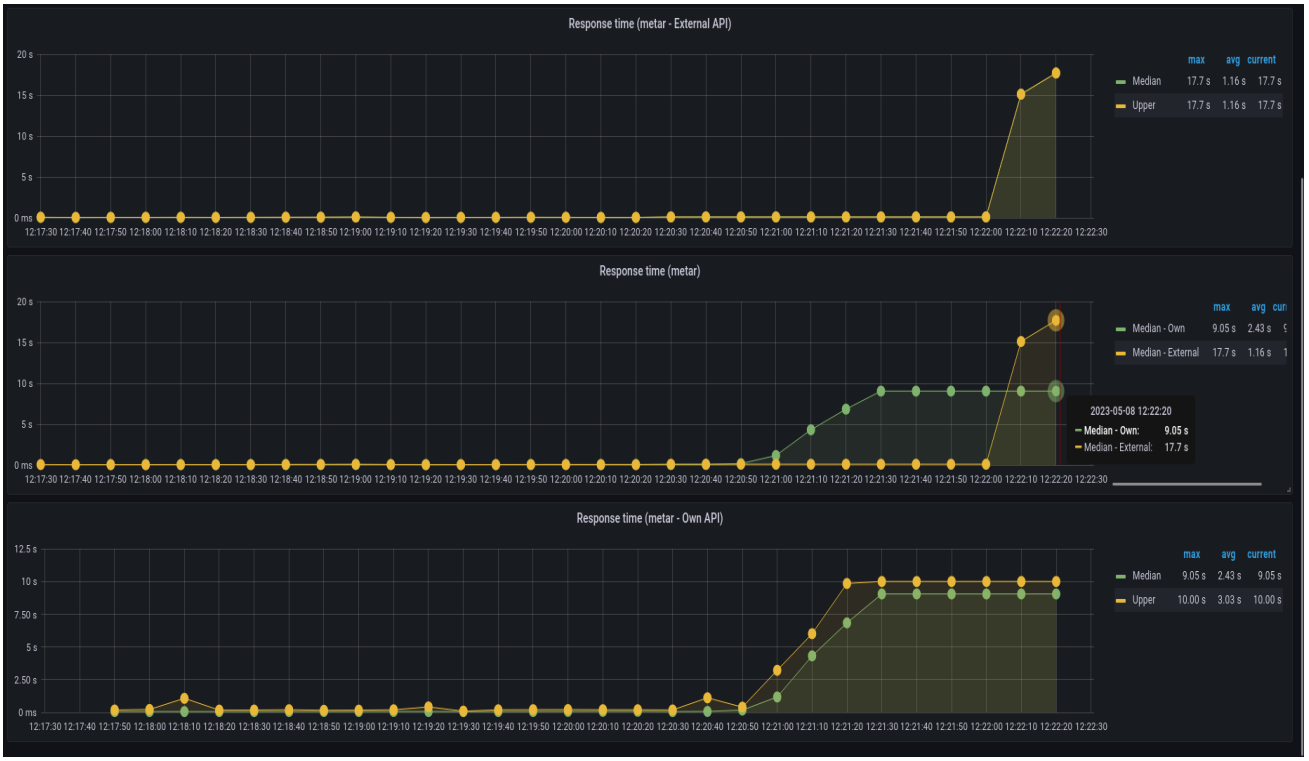
Cada prueba viene acompañada de 6 paneles, 3 de ellos pertenecen al enunciado y son:

- Scenarios launched: cantidad de requests en 10 segundos.
- Request State: status code de la respuesta del servicio, o “Errored” en caso de producirse timeout.
- Response time: tiempo de respuesta que observa el cliente.



Se agregaron 3 paneles más, estos son:

- Response time - External API: demora del endpoint externo.
- Response Time - Own API: demora endpoint propio.
- Response time: muestra la mediana del Response Time - External API y la mediana de mediana de Response Time - Own API, ambas con respecto a los datos de los últimos 10 segundos.



Caso Base

Arquitectura

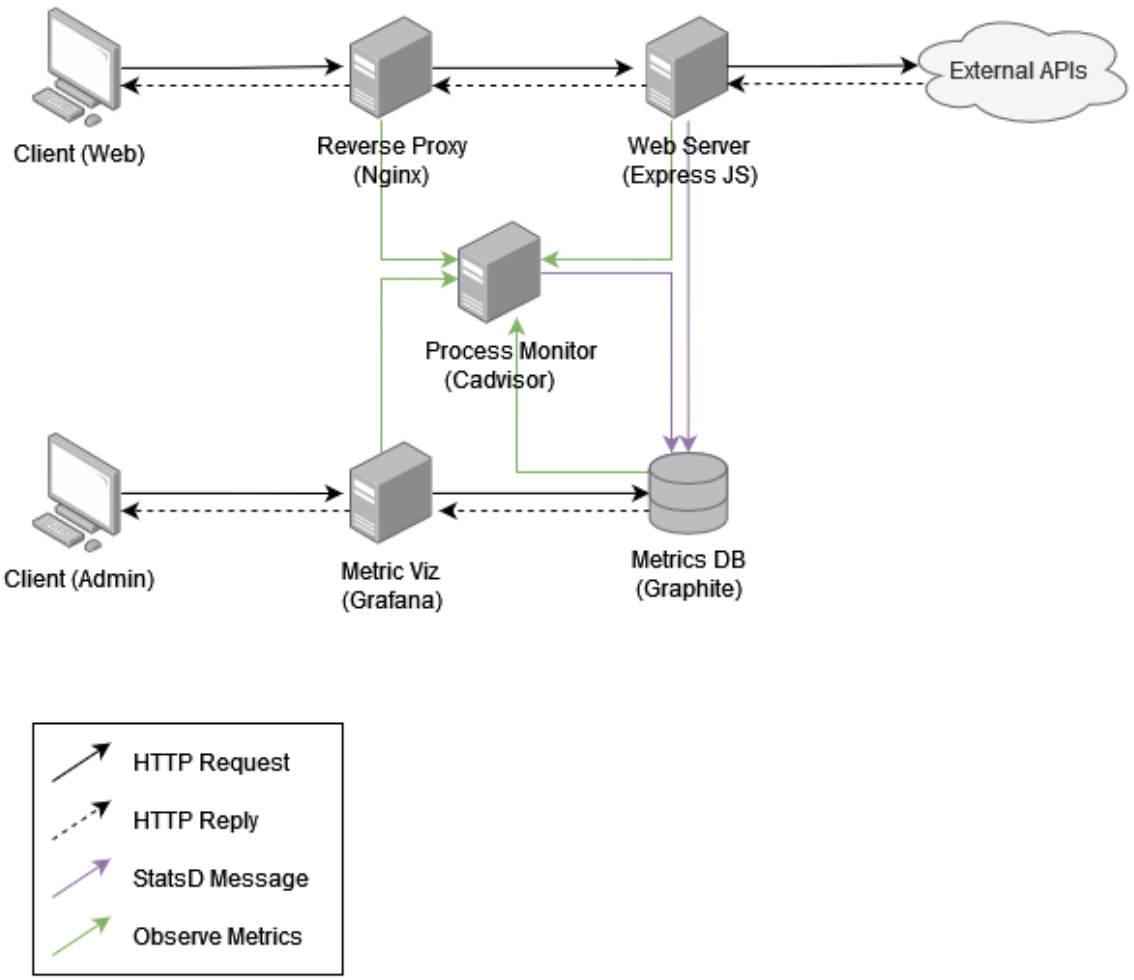


Fig 1.1. Vista de Componentes y Conectores del Caso Base

La arquitectura base del sistema cuenta de los siguientes componentes:

- Reverse proxy (nginx): punto de entrada de los clientes a la API, redirige las peticiones al servidor.
- Web server: Servidor que procesa las peticiones, se conecta con APIs externas que proveen los servicios de sus distintas rutas.
- Process Monitor (cadvisor): Se encarga de monitorear el estado y recursos de los containers del sistema.
- Metrics DB (Graphite): Funciona como repositorio de las métricas del sistema.
- Metrics Viz (Grafana): Provee una interfaz para que los administradores puedan visualizar métricas pertinentes del sistema.

Pruebas de performance

Useless facts

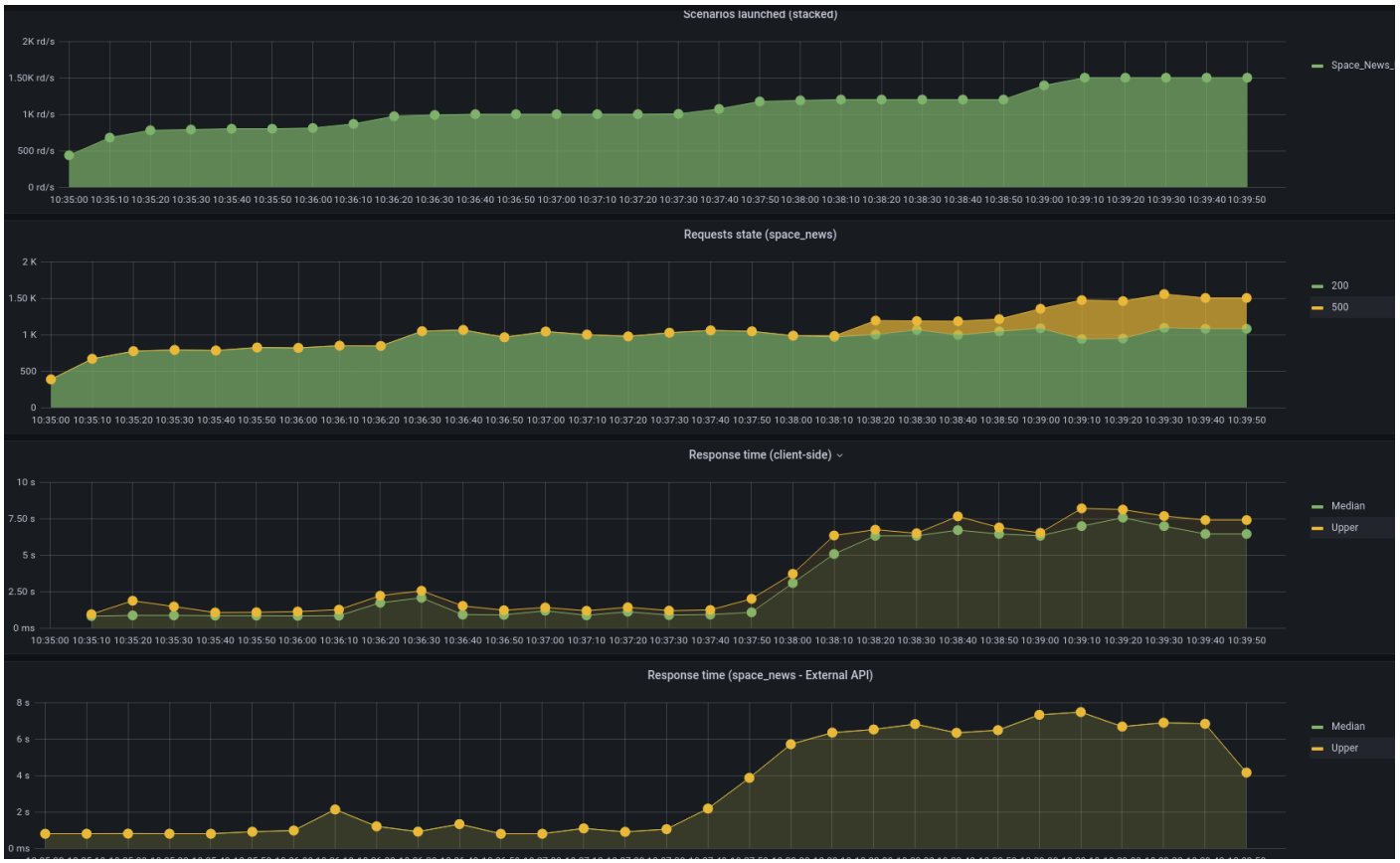
Observaciones:

El servicio responde las request sin problemas hasta llegar a la fase Plain Load 4 (250 request/sec) en donde se aprecia un incremento tanto en el Response time de External API como en el Response Time. Cuando se produce el Load Spike (350 request/sec) el incremento en el Response Time client-side es mucho mayor y directamente se empiezan a producir ETIMEDOUT hasta que ninguna request devuelve un 200. Esto significa que la herramienta de testing (artillery) no está recibiendo los paquetes HTTP Response correspondientes.

En el tercer gráfico se puede ver que el tiempo de respuesta de la API externa se mantiene bajo de manera constante, excepto por la meseta que se produce al iniciar los timeouts. Esto probablemente se deba a un error de medición propio. Como nuestro servidor está tardando en responder, la ejecución del código que marca los timestamps de inicio y fin también se ve afectada.

Concluimos, entonces, que en este caso el principal factor limitante es nuestro servidor. Es esperable que la técnica de replicación sea beneficiosa para la disponibilidad del sistema.

Space News



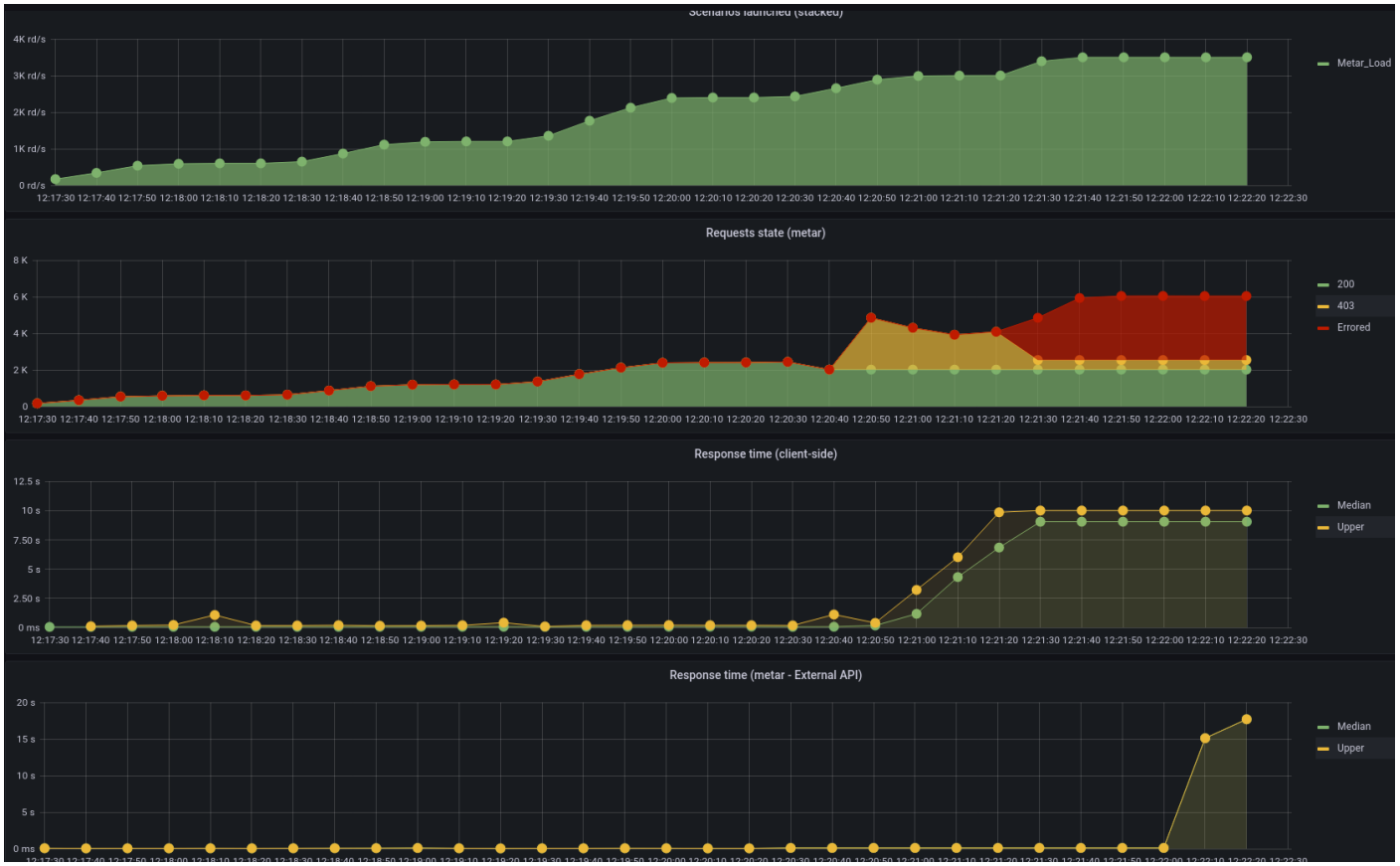
Observaciones:

El endpoint responde sin problemas hasta la fase Ramp up load 3, que finaliza con 120 requests por segundo. Cuando se mantiene esta cantidad a lo largo de 60 segundos, en la fase Plain load 3, algunas requests comienzan a fallar, devolviendo error 500 (Internal Server Error). También se observa un gran incremento en los tiempos de respuesta, tanto de la API externa como la propia: antes de estas fases, el valor de ambos era de 1 segundo, aproximadamente. Luego, la media aumenta hasta 7.5 segundos.

Es interesante notar que, a diferencia del endpoint anterior, no hubo ningún timeout.

Con estos datos, concluimos que el factor limitante en este endpoint es la API externa. Esto nos permite pronosticar que la táctica de replicación no dará mejores resultados que el baseline. Un caché sería una mejor opción.

Metar



Observaciones:

El endpoint funciona correctamente hasta la fase Ramp up load 4, momento en el que empieza a devolver error 403 (Forbidden). Este es un mecanismo de la API externa para indicarnos que se están realizando demasiadas peticiones (en este caso, alrededor de 300 requests/s). Si seguimos aumentando la carga, nuestro servidor se satura, y no es capaz de contestar las peticiones a tiempo, por lo que artillery da timeout.

También vemos que, pese a devolver error 403, la API externa mantiene un muy buen tiempo de respuesta, alrededor de los 60 milisegundos. Al final de la prueba aumenta, pero seguramente se trata del mismo problema que el caso anterior (error en la medición).

De acuerdo con este análisis, podemos predecir que la replicación no sería muy útil en este caso. Nuestro servidor será capaz de responder más requests por segundo sin dar timeout, pero la API externa devolverá error 403 a todas esas peticiones. Al igual que el endpoint previo, un caché podría ser mejor.

Táctica: Rate Limiting

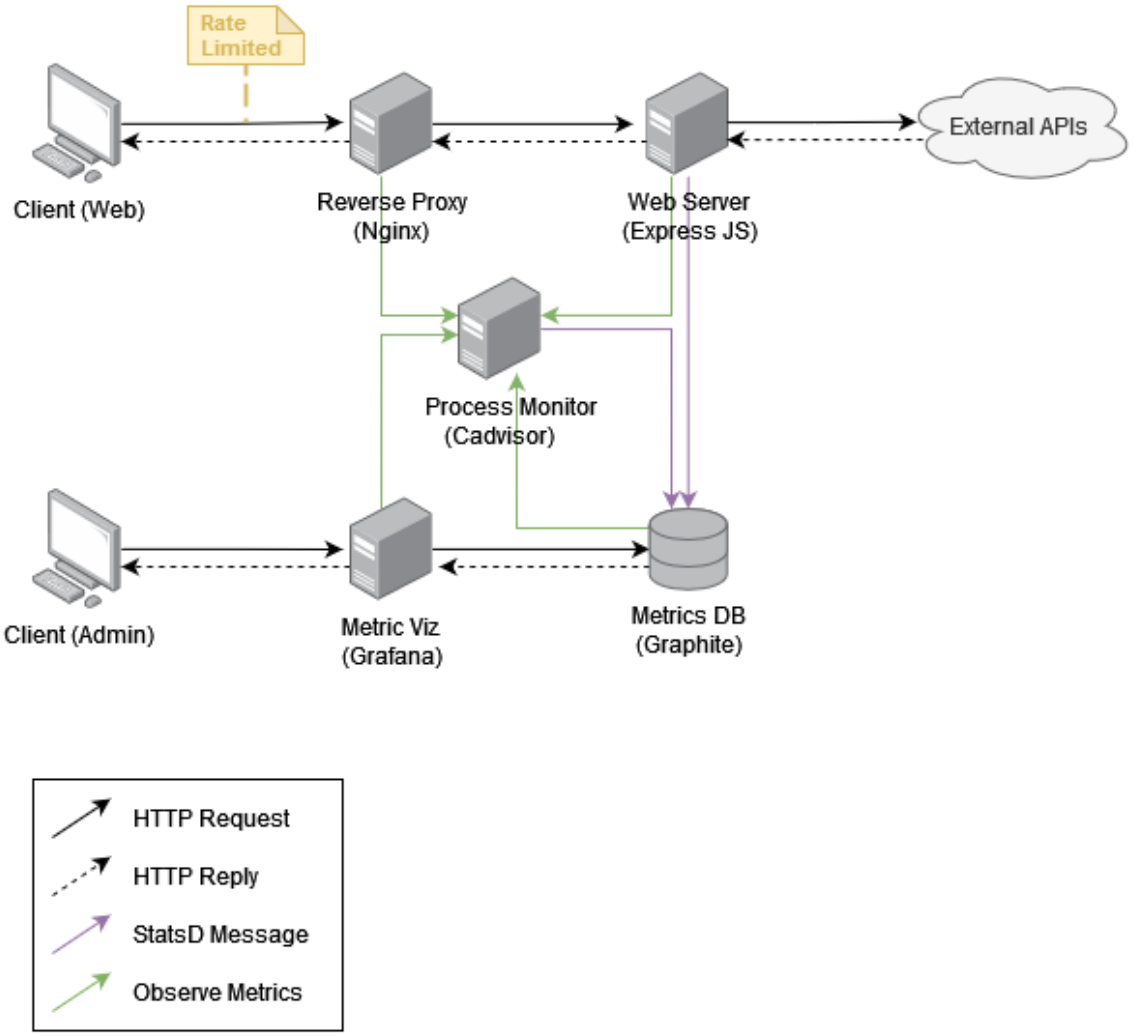


Fig 2.1. Vista de Componentes y Conectores utilizando Rate Limiting

Se establece un límite de la cantidad de pedidos por periodo de tiempo que acepta el “reverse proxy” (nginx) a modo de limitar la carga sobre el servidor. Se eligió una tasa de 500 r/s basándose en la capacidad del servidor de atender pedidos sin fallar. Los pedidos rechazados reciben una respuesta con estado 429 (Error: Too Many Requests)

Pruebas de performance

Useless facts

Observaciones:

Se observa claramente que el tiempo de respuesta de la API externa se mantiene prácticamente constante durante toda la ejecución de la prueba, excepto en un pico casi al final.

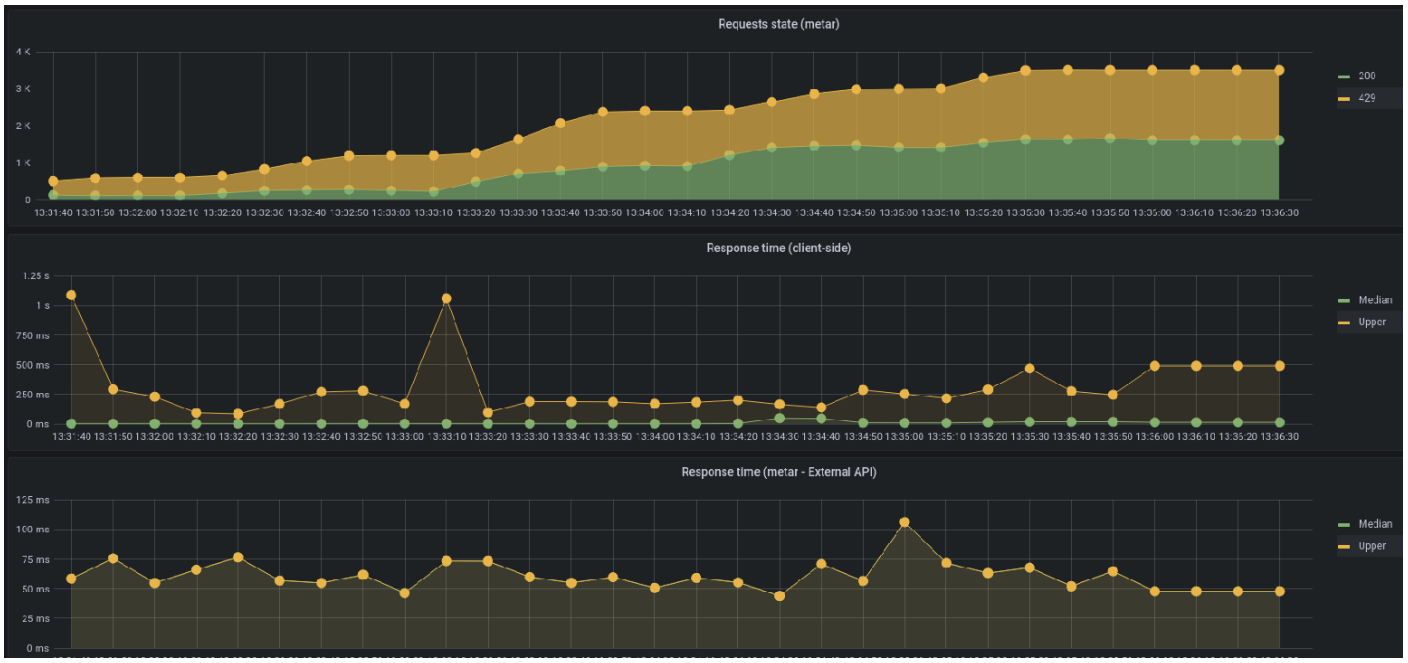
La mediana del tiempo de respuesta client-side es, en la mayoría de las fases, inferior a los 20 milisegundos. Esto es porque un gran porcentaje de las requests son interceptadas directamente por nginx, que devuelve inmediatamente error 429 (Too Many Requests).

Space News

Observaciones:

Este caso es prácticamente análogo al endpoint anterior: el response time de la API externa se mantiene constante, y la mediana del tiempo de respuesta client-side es muy baja gracias a la acción de nginx.

Metar



Observaciones:

Parecido a los casos anteriores, el Response Time de la API externa se mantiene en un rango de valores entre 50ms y 75ms. Lo mismo sucede con el Response Time Client Side donde el tiempo de respuesta no aumenta dado que el aumento en la cantidad de requests es muy poco producto del Rate Limiting. Lo que sí se ve afectado es la cantidad de usuarios a los que el servicio es capaz de responder dado que se limita en gran medida.

Táctica: Replicación

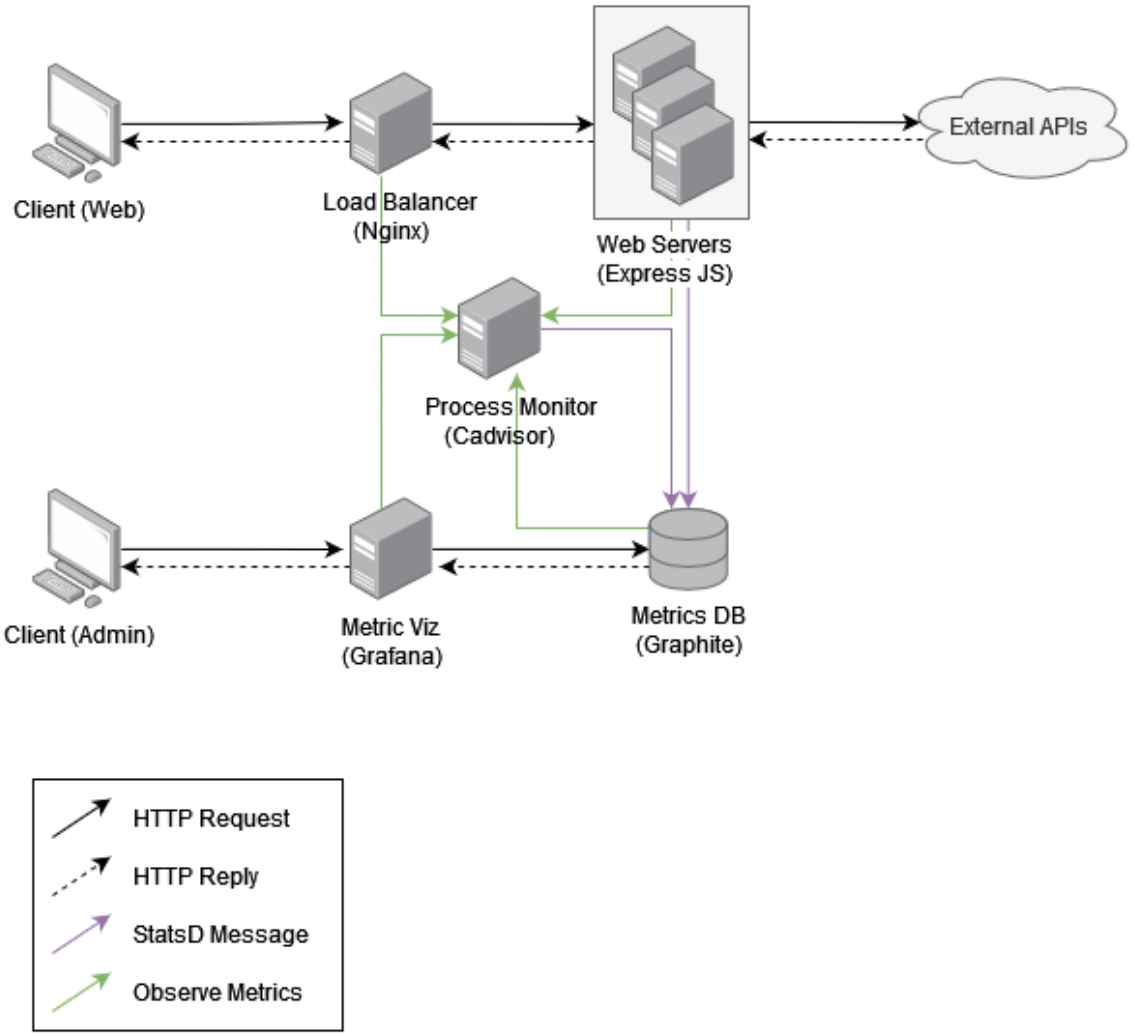


Fig 3.1. Vista de Componentes y Conectores utilizando replicación

Se replica el servidor web, generando tres copias identificadas que proveen el mismo servicio y se adapta el “reverse proxy” (nginx) para funcionar como un balanceador de carga que distribuye los pedidos a los distintos servidores.

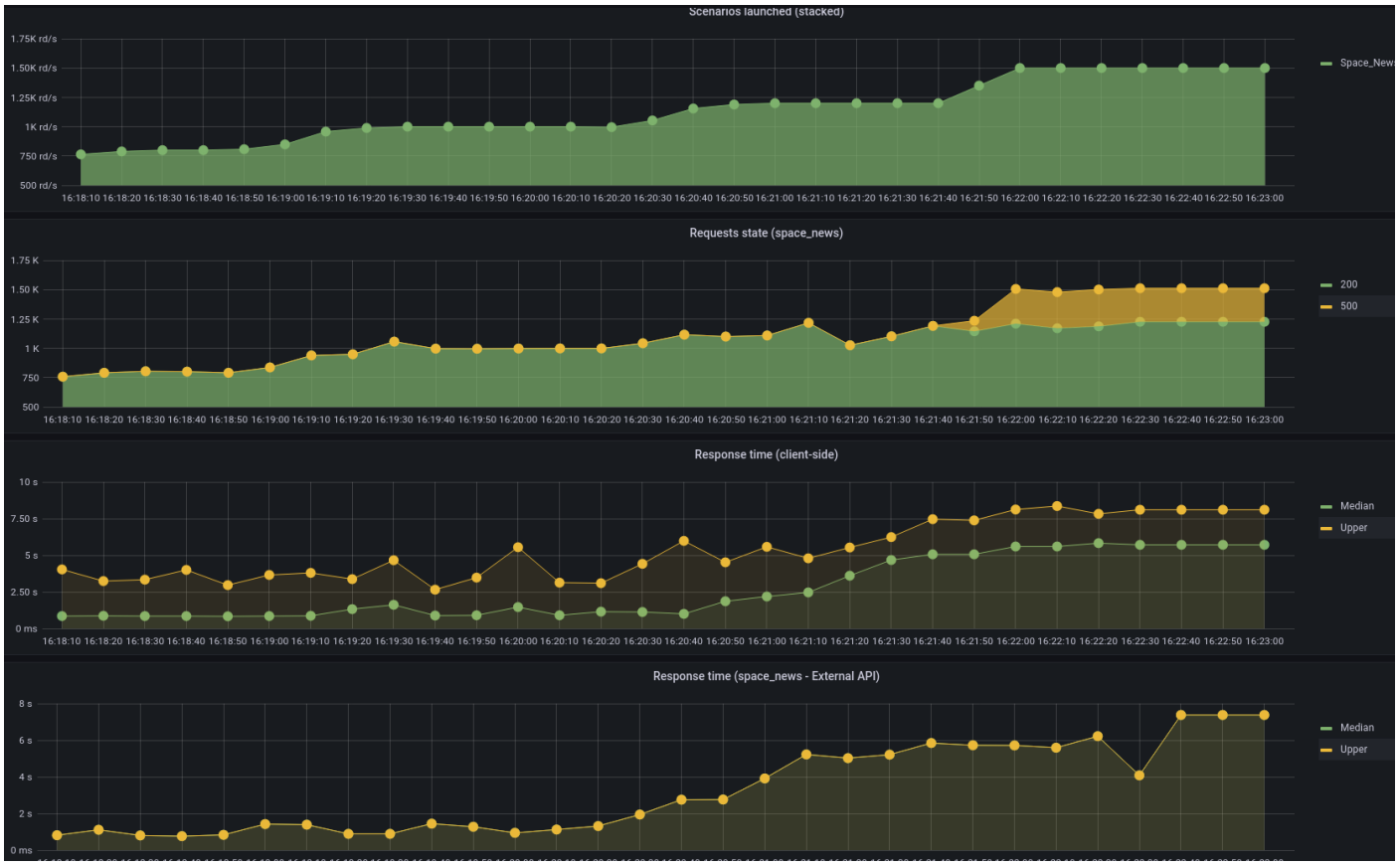
Pruebas de performance

Useless facts

Observaciones:

Tal y como habíamos predicho, la replicación benefició a la disponibilidad del endpoint en gran medida. Tan solo 6 requests dieron timeout, mientras que el resto fueron respondidas con un tiempo de respuesta que media los 980 milisegundos.

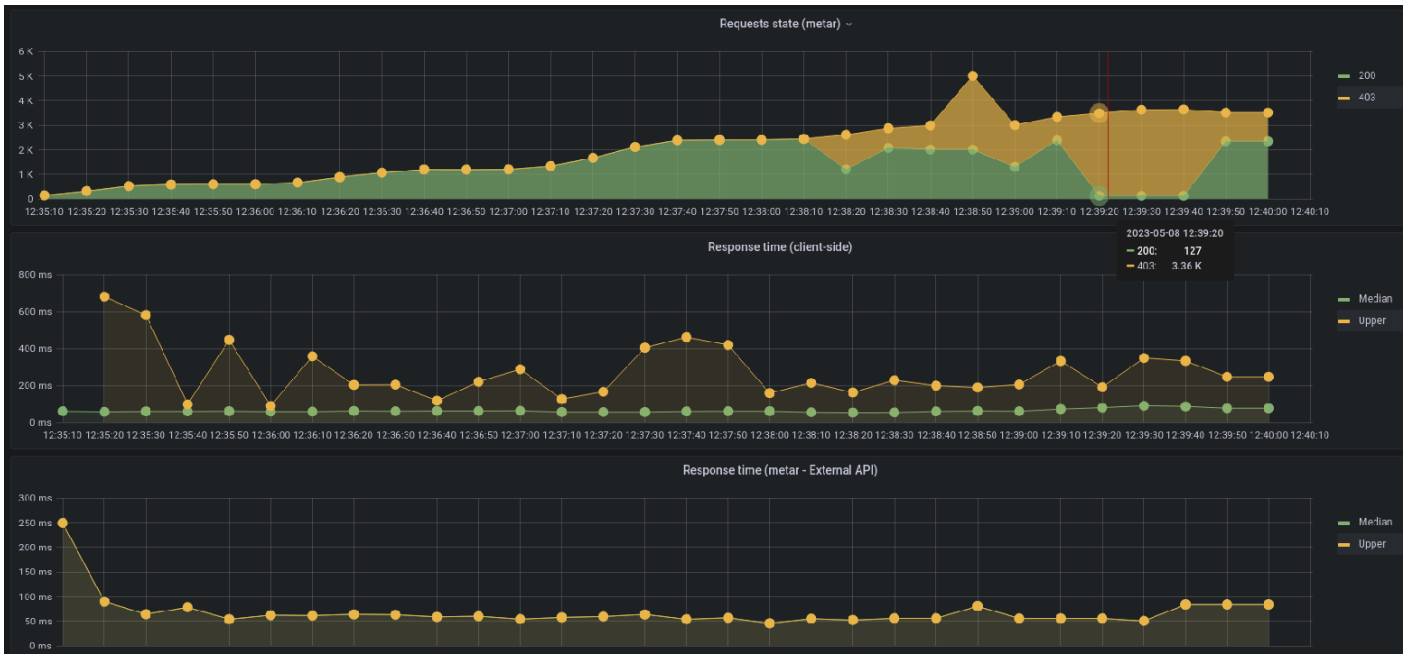
Space News



Observaciones:

La táctica no fue muy útil, porque sabíamos que el factor limitante era la API externa. Los 3 gráficos son muy similares a los del Caso Base lo que indica que no recibimos ningún beneficio de aplicar esta táctica.

Metar



Observaciones:

La diferencia con el Caso Base es que no se produjo ningún ETIMEDOUT pero fue reemplazado por errores tipo 403 que se incrementaron hasta el punto de ser el único tipo de respuesta que se recibía de la API externa. Como habíamos predicho en el análisis del baseline, esta táctica no fue útil.

Táctica: Caché

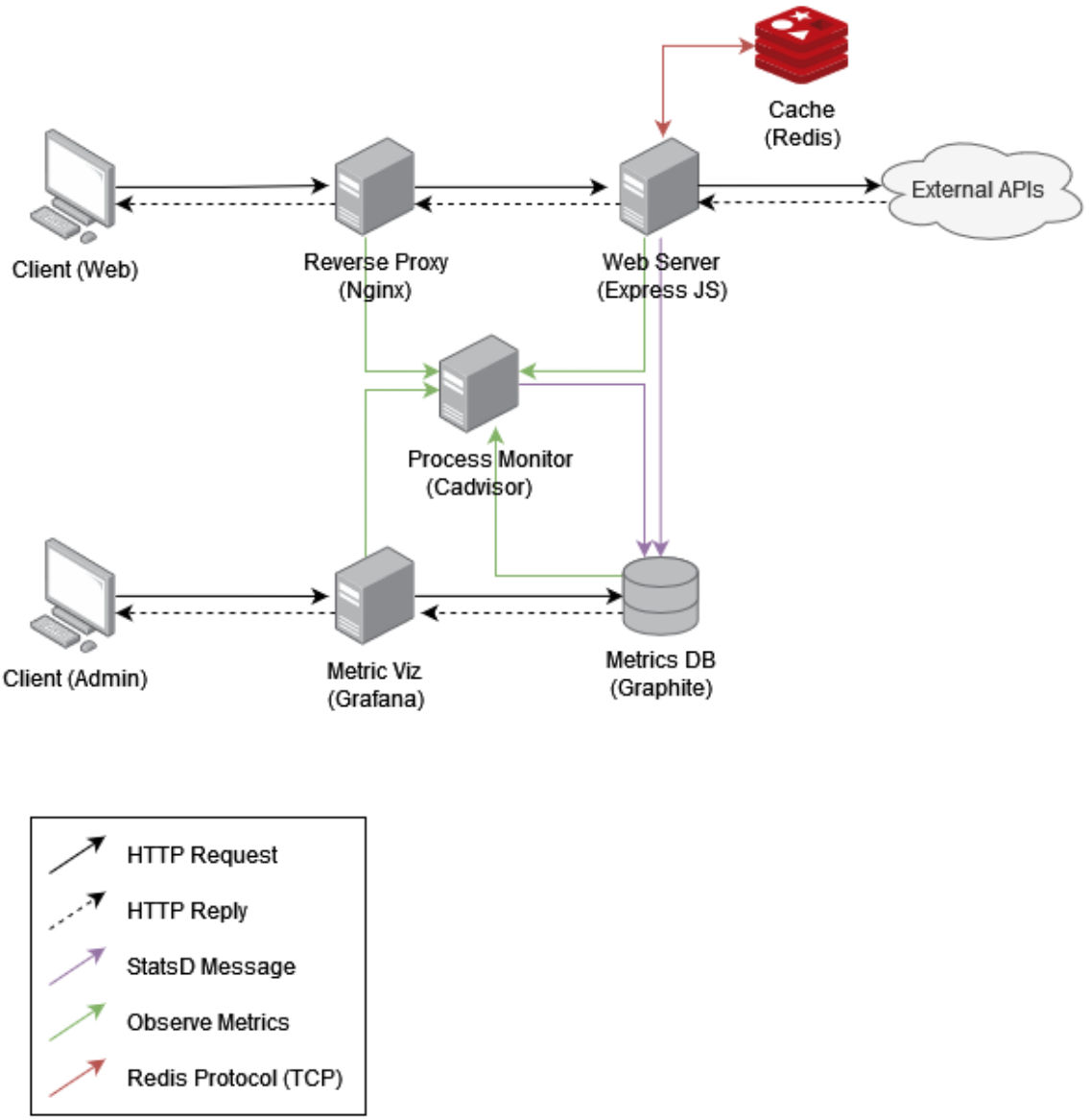


Fig 4.1. Vista de Componentes y Conectores utilizando caché

Se implementó esta táctica usando como caché a Redis, que es un data store en memoria, dentro de la aplicación Node. Para cada servicio de la aplicación (Useless Facts, Metar y Space News) se usaron distintas estrategias de caché, teniendo en cuenta varios criterios como tamaño, llenado, tiempo de vida y vaciado. Consideramos que toda la información de los servicios es cacheable.

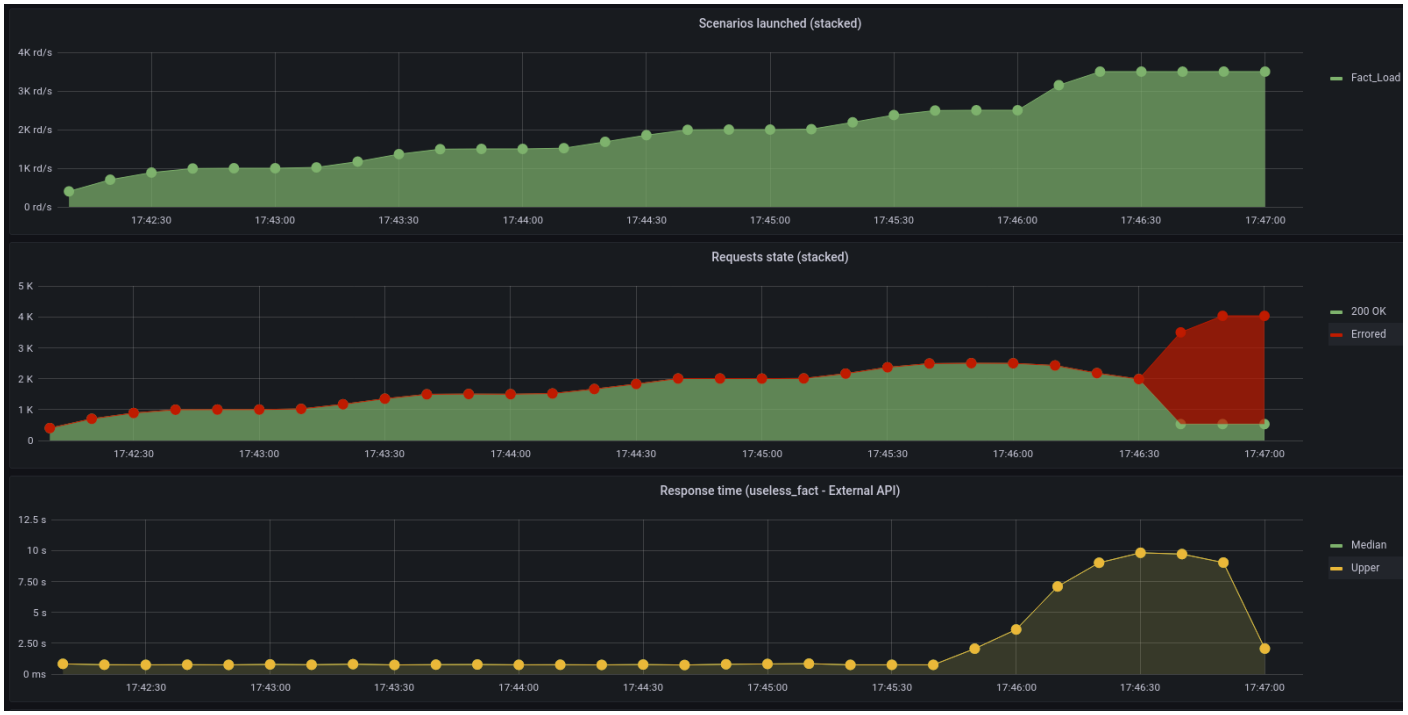
Para el caso de Space News, se usó lazy population, con un tamaño de 5 artículos cacheados, con un tiempo de vida de 5 segundos y sin vaciado, de forma que los artículos “expiran” cuando termina su tiempo de vida.

El servicio de Metar tiene una caché similar a Space News, con lazy population, sin vaciado y TTL de 5 segundos, con la diferencia de que en vez de cachear solamente el último Metar obtenido, se cachea el último Metar de cada estación pedida, almacenando un ítem por cada estación (en los últimos 5 segundos).

Por último, para el caso de Useless Facts no se usó una caché en Redis, sino que se implementó una caché local en memoria con active population, 100 ítems y con vaciado tal que se elimina un ítem una vez que es consumido. De esta forma, siempre habrá algún fact disponible para el cliente cada vez que pida uno, sin repetir la misma respuesta. En este caso, no hay tiempo de vida para los ítems, quedan almacenados indefinidamente hasta ser consumidos.

Pruebas de performance

Useless facts



Observaciones:

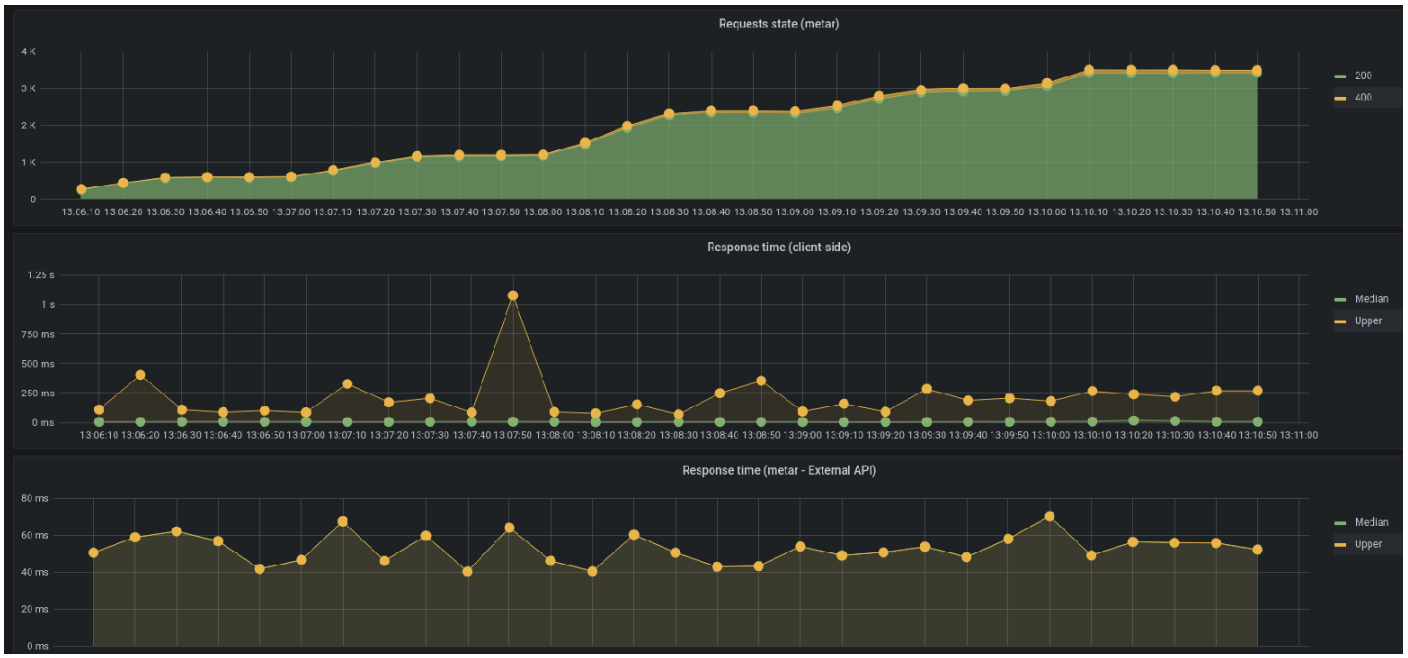
Se observa un comportamiento similar al caso base, donde el servidor se vuelve incapaz de atender una gran cantidad de respuestas. Se nota una diferencia, que se genera un piso de respuesta en vez de fallar todos los pedidos.

Space News

Observaciones:

Como era de esperar, la táctica mejora mucho el tiempo de respuesta, notamos una tendencia constante tanto en la mediana como en la máxima (que se debe a la invalidación de caché). También, notamos como ahora se resuelven algunas peticiones que antes estaban limitadas por la API externa.

Metar



Observaciones:

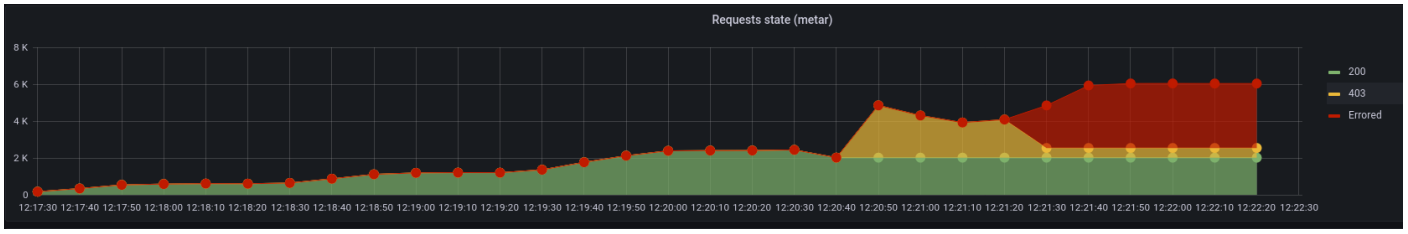
Nuevamente notamos una gran mejora en la latencia de las respuestas, con una tendencia constante. Además en este caso las peticiones se están resolviendo de manera efectiva incluso con alta carga, a diferencia del caso base que era limitado y/o fallaba.

Comparación

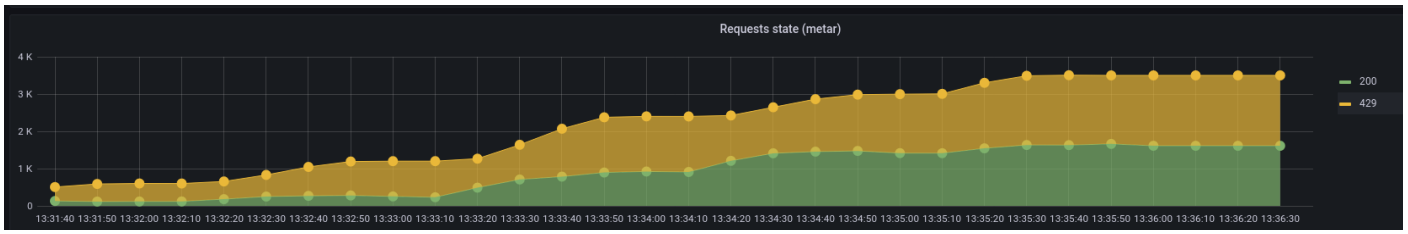
Viendo que en general el comportamiento de los distintos endpoints es análogo entre diferentes tácticas, tomaremos las métricas de “Metar” para compararlas, siendo estas las más expresivas.

Estado de Respuestas

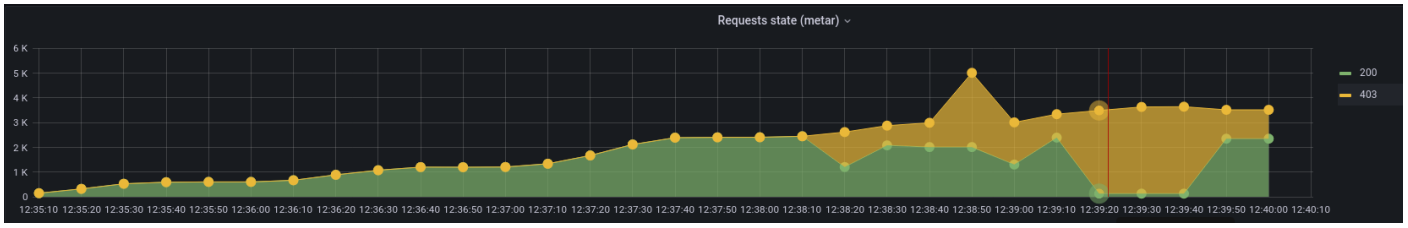
Caso Base



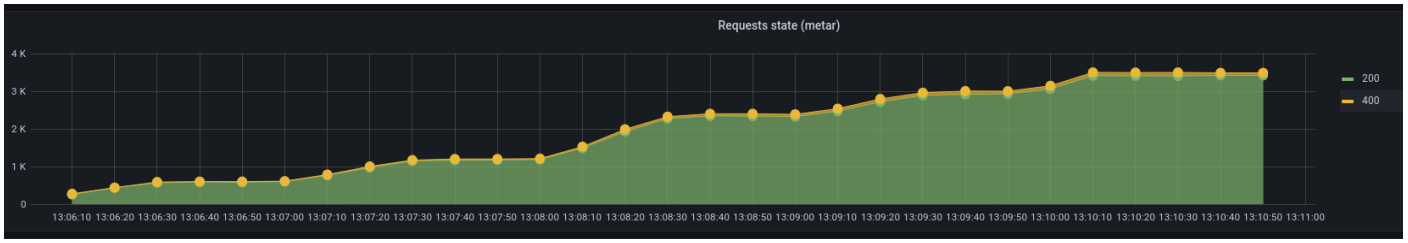
Táctica: Rate Limiting



Táctica: Replicación



Táctica: Caché

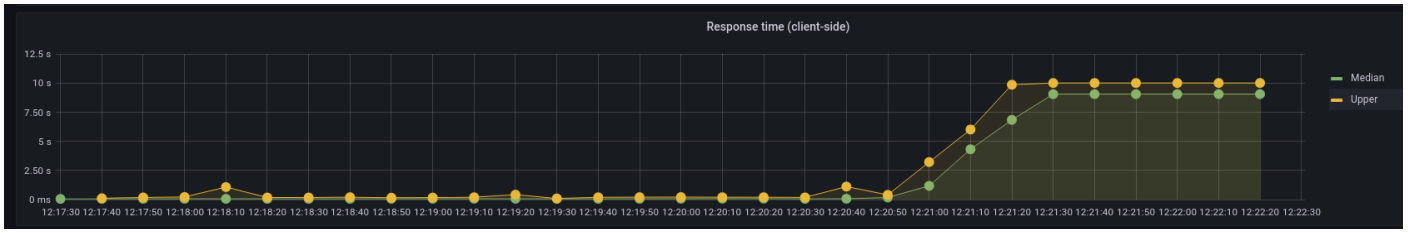


Observaciones

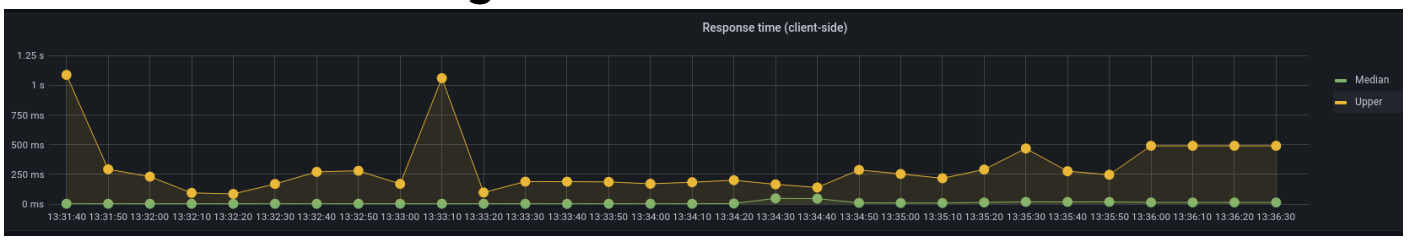
Como era de esperar, observamos una gran mejora por sobre el caso base, especialmente al utilizar caché, donde todas las requests se resuelven correctamente. Para el caso de rate limitado, apreciamos que ningún mensaje falla, pero varios son limitados, obteniendo menor throughput con baja carga a cambio de sostener el funcionamiento a mayor carga. Por otro lado, al replicar el servidor multiplicamos el piso de respuesta bajo alta carga, pero aun así se ve limitado y resulta en errores.

Latencia

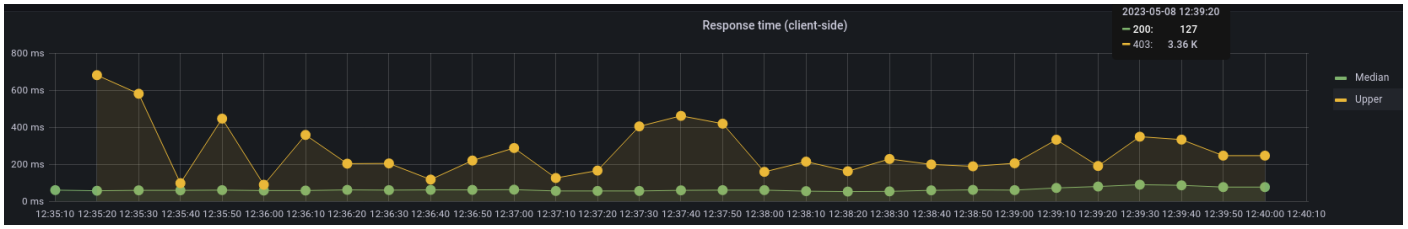
Caso Base



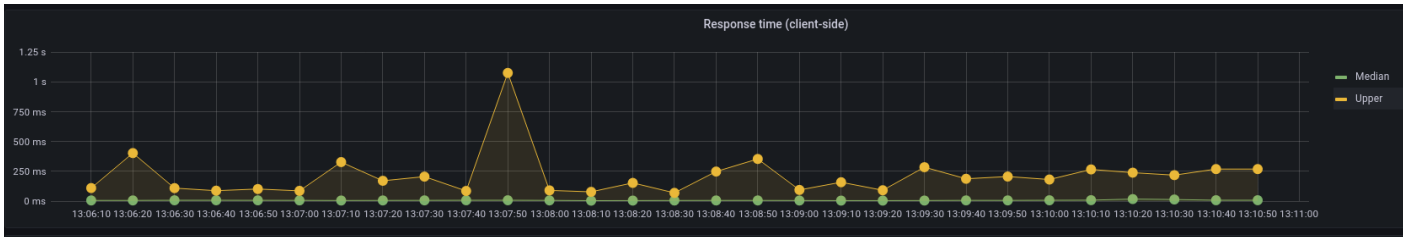
Táctica: Rate Limiting



Táctica: Replicación



Táctica: Caché



Observaciones

Nuevamente obtenemos buenos resultados comparando al caso base. La técnica de replicación da un resultado interesante, ya que su latencia es baja hasta cierto límite de carga. Al utilizar caché vemos un rápido tiempo de respuesta salvo algunos pequeños picos, que esperaríamos al invalidar la caché; esta táctica es muy efectiva, ya que consigue mantener el nivel de servicio y atender una gran cantidad de respuestas. Por último, la táctica de rate limiting da los mejores resultados de latencia, pero hay que tener en cuenta que esto se debe a que mantiene fija la cantidad de usuarios que atiende.