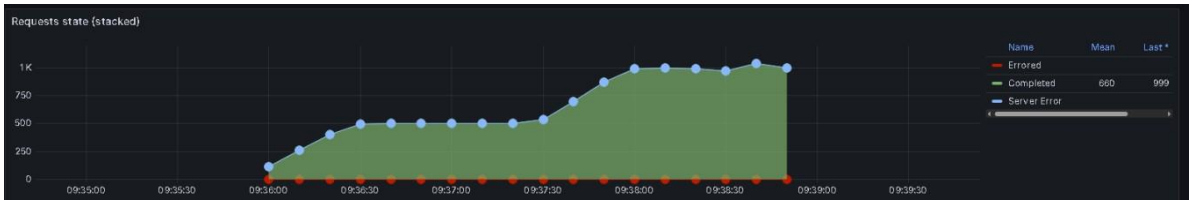
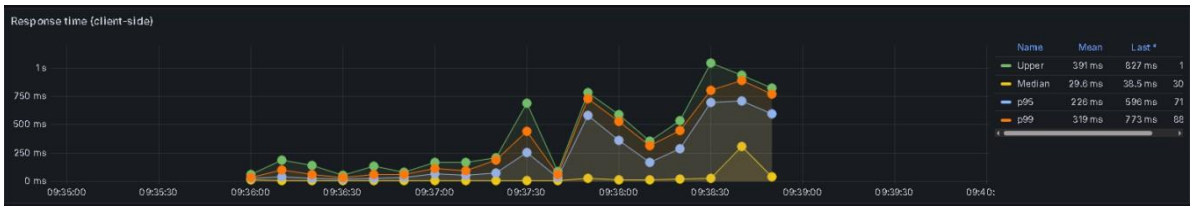
En este documento, se analizan las pruebas realizadas en Artillery, con los gráficos de Grafana. Se revisan las pruebas para cada endpoint, y para las versiones v1 (Original) y v1.1 (Version con mejoras)

Para los GET Accounts, GET Rates y PUT Rates, todas las solicitudes son procesadas correctamente, y no hay solicitudes con errores y todas siguen la misma forma en su gráfico. Ejemplo con GET Accounts:



GET_ACCOUNTS_V1

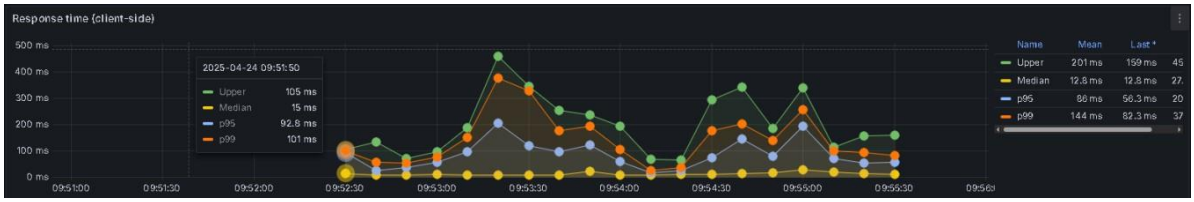
Podemos ver que casi todas las métricas siguen la misma tendencia: Primero tienen un pico, luego se “planchan”, y luego se repite. Los valores rondan entre los 500ms y 1s.



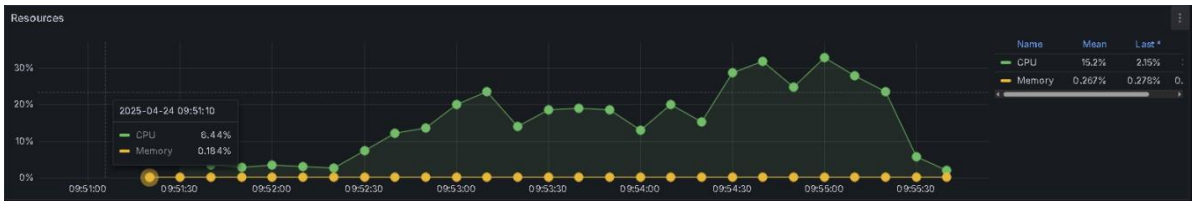
El uso de CPU no es significativo

GET_ACCOUNTS_V1.1

A diferencia del otro, en este gráfico vemos que se comienza con grandes picos, se van “planchando”, y luego los picos son mas pequeños, en general. Ademas si se ve bien, los valores no superan los 500ms, a diferencia del V1 donde los valores podían estar entre los 500ms y 1s

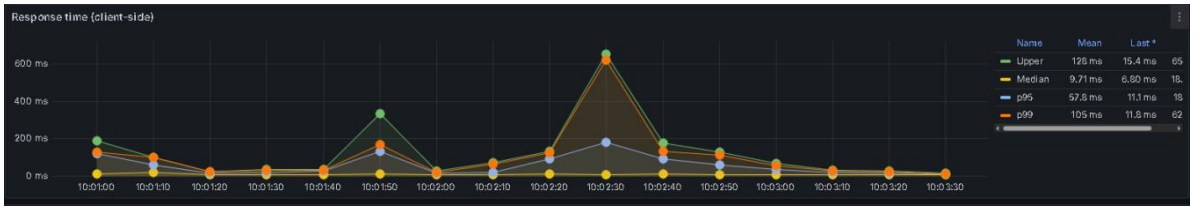


El uso de CPU es similar al de la versión v1.

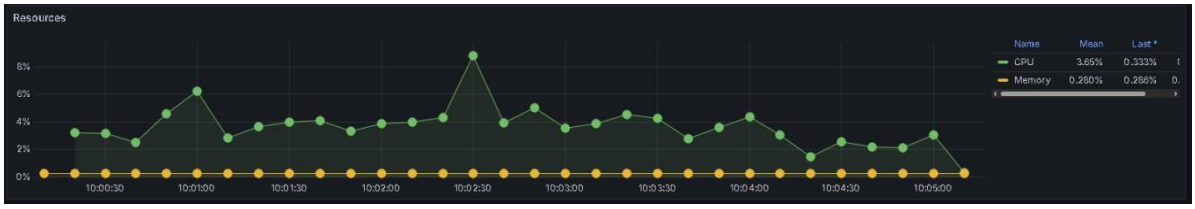


GET RATES V1

Acá se nota que pasa algo parecido que con el GET Accounts v1, con la diferencia de que los picos son esporádicos pero en general el gráfico está bastante plano y no se mueve mucho. Los valores como mucho llegan a casi 400ms y a 600ms.



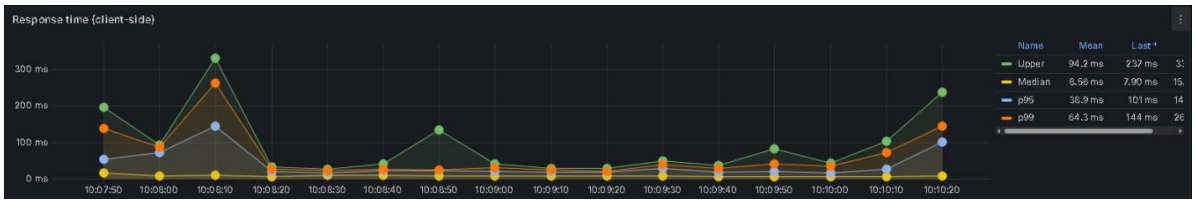
El uso de CPU es muy pequeño, con pocos picos.



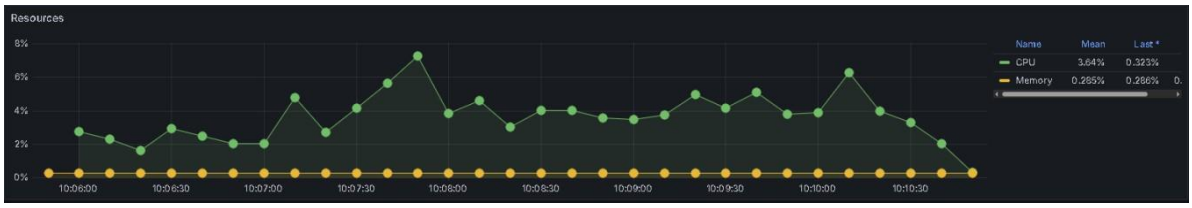
GET RATES V1.1

Vemos algo parecido a lo de GET Accounts v1.1, vemos que los picos decrecen en vez de crecer con el tiempo (En general), y en general el grafico está “planchado”. Además también notamos la

mejora en tiempos, pues el mayor pico no supera los 300ms, mientras que en el gráfico de GET Accounts v1, uno de los picos más pequeños era de 300ms

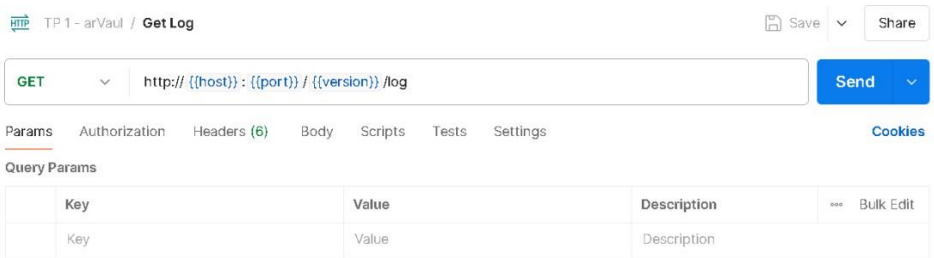


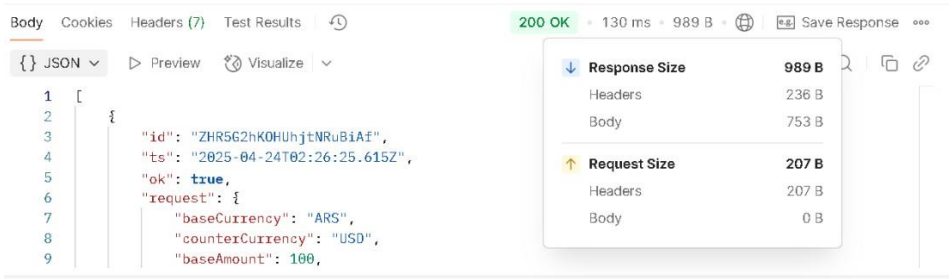
El uso de CPU tiene picos un poco mas pequeños a comparación del v1.



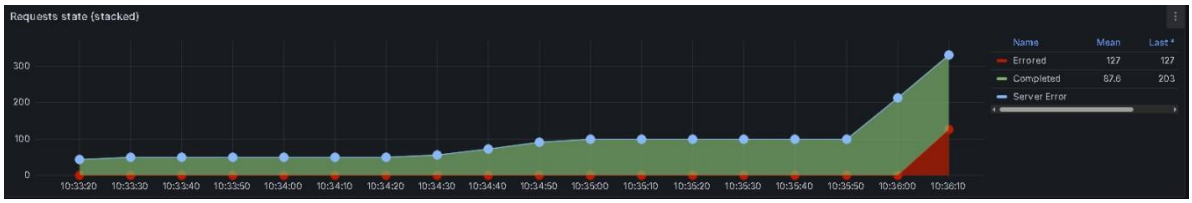
GET LOG V1

Con un log de este tamaño (989 bytes)

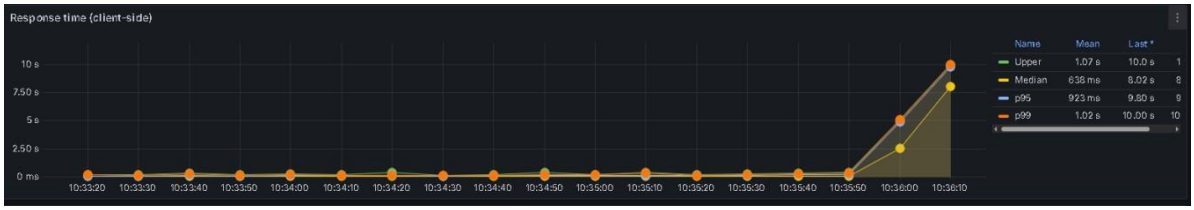




Vemos que empiezan a surgir errores en el tramo final de las pruebas. No es como en los demás GET que se procesa todo correctamente.



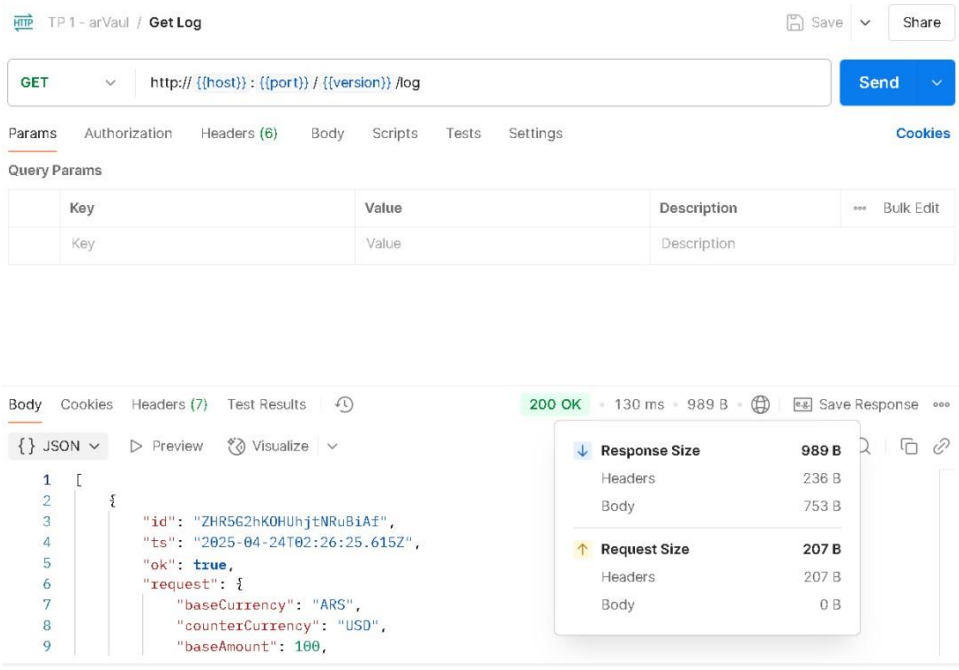
Los response time están “planchados” pero aumentan mucho en el tramo final también.



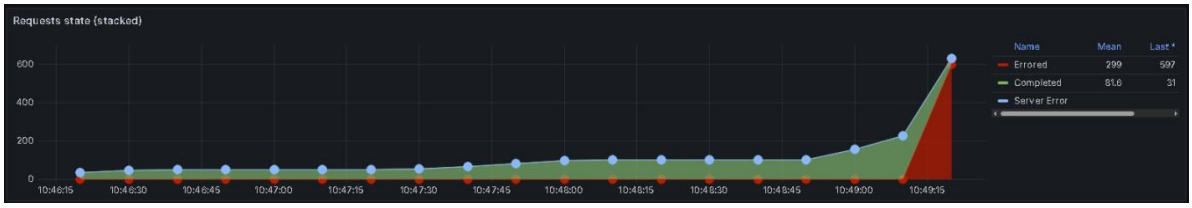
Y el uso de CPU también se hace muy significativo al final

GET LOG V1.1

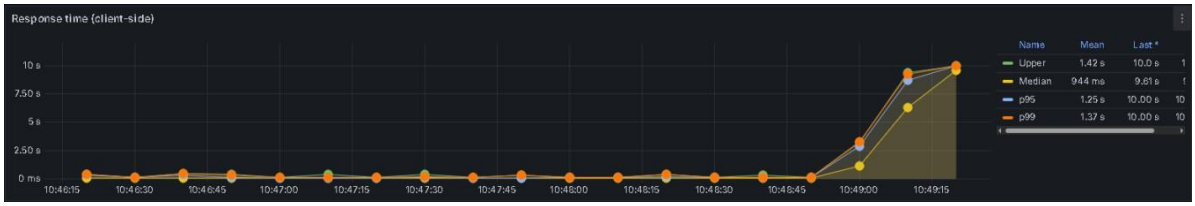
Con un log de este tamaño



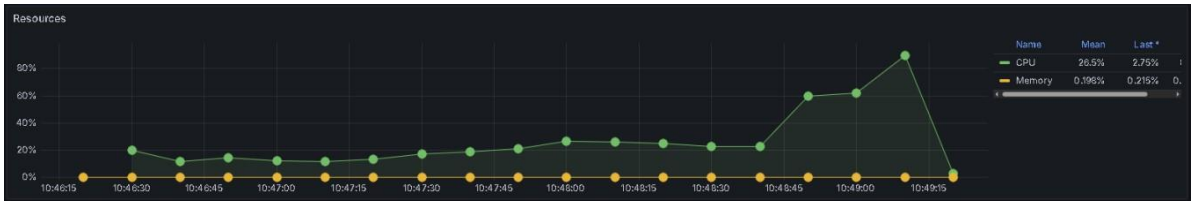
En este gráfico se ve que la tasa de errores es mas alta en esta version



Los tiempos desde el lado del cliente son parecido a los de la v1.

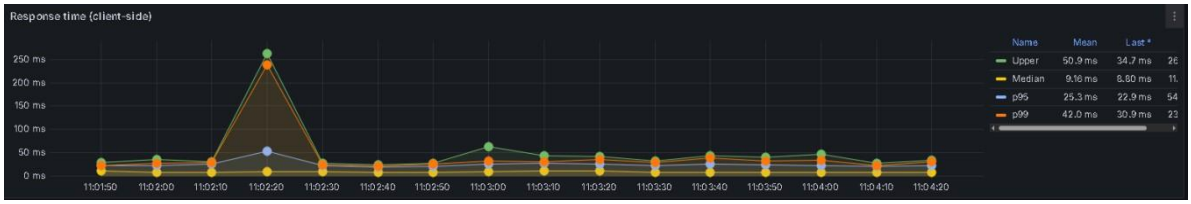


Y el uso de CPU es parecido que en el v1.

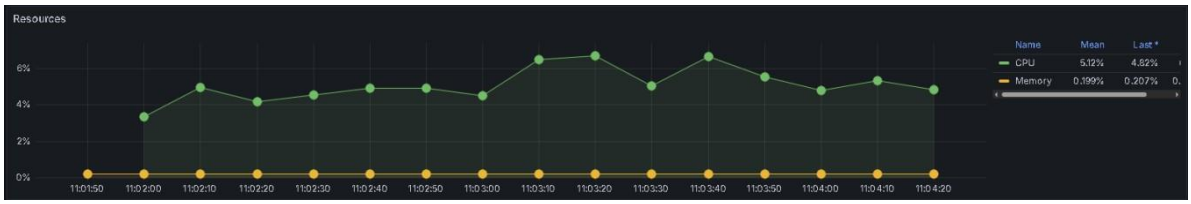


PUT RATES V1

Hay un pico que alcanza aproximadamente los 250ms pero en general el gráfico está mas “planchado”

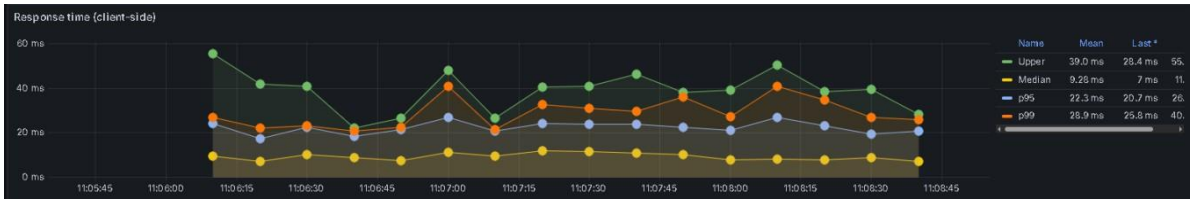


Y el uso de CPU es menor.

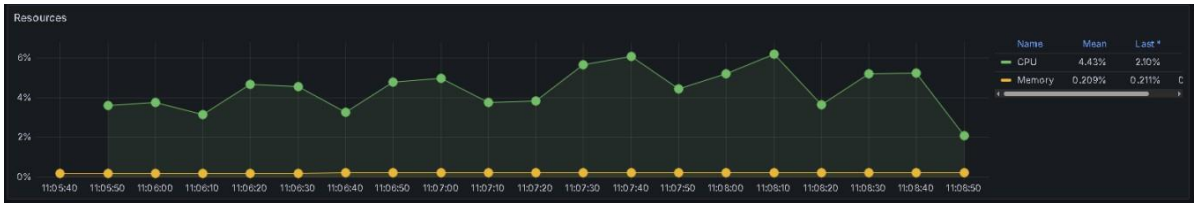


PUT RATES V1.1

A diferencia del v1, este gráfico tiene mas movimiento, pero los tiempos son mucho mejores, el pico está entre los 60ms.

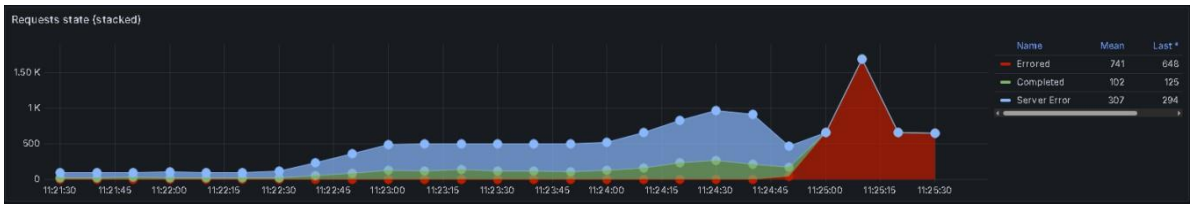


Y el uso de CPU es parecido al de la versión v1.

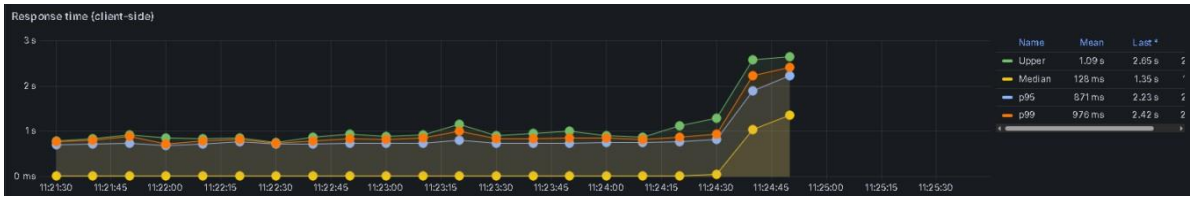


POST EXCHANGE V1

En estas pruebas, la tasa de error al final es muy recurrente, y también en el medio hay muchos errores de servidor.



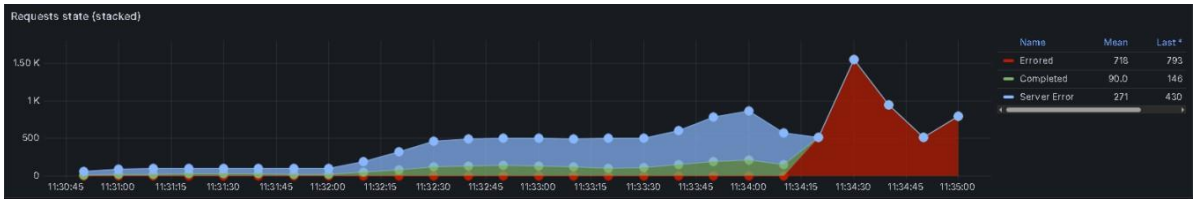
Los tiempos se mantienen en el segundo, pero en el tramo final aumentan de forma importante.



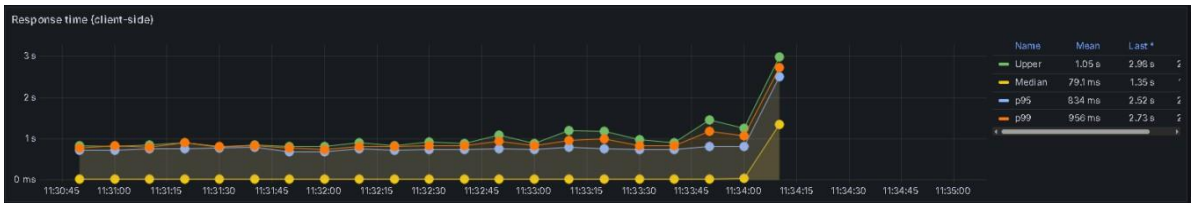
Y el uso de CPU es alto pero se mantiene en niveles excepto por algún pico.

POST EXCHANGE V1.1

La tasa de error no es muy distinta a la de v1



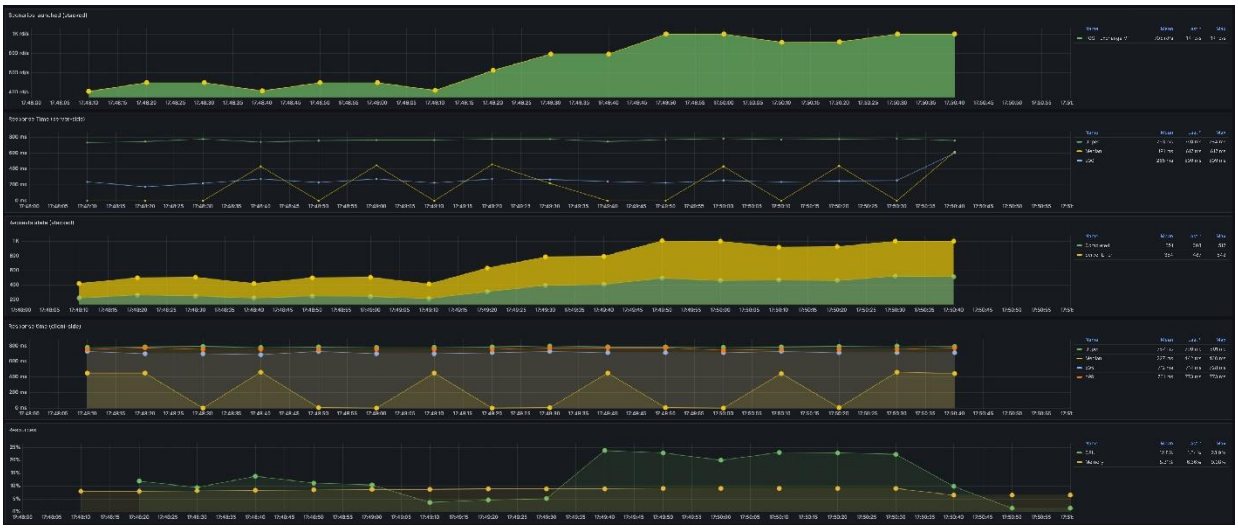
No hay mucha diferencia con el v1 en tiempos



Y el uso de recursos es similar al de v1

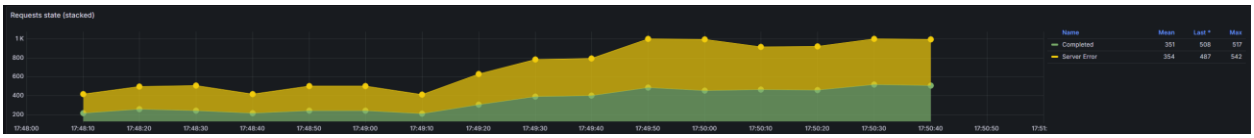
POST_EXCHANGE_V1 (pool 500)

Analizamos la mejora en escalabilidad al aumentar el pool de las pruebas de artillery de 50 a 500 par llevar al limite la escalabilidad del sistema.

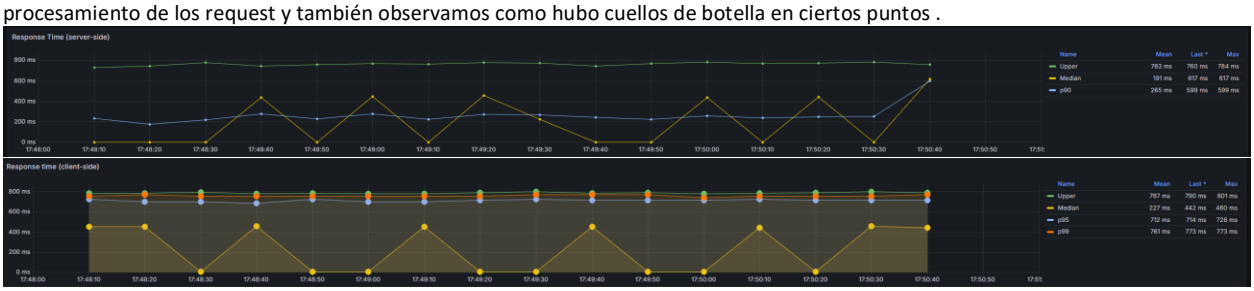


El test alcanza los 1000 request por segundo

Ante este escenario de carga intensa, el sistema no fue capaz de procesar en su totalidad los request enviados y comenzó a mandar Server Error



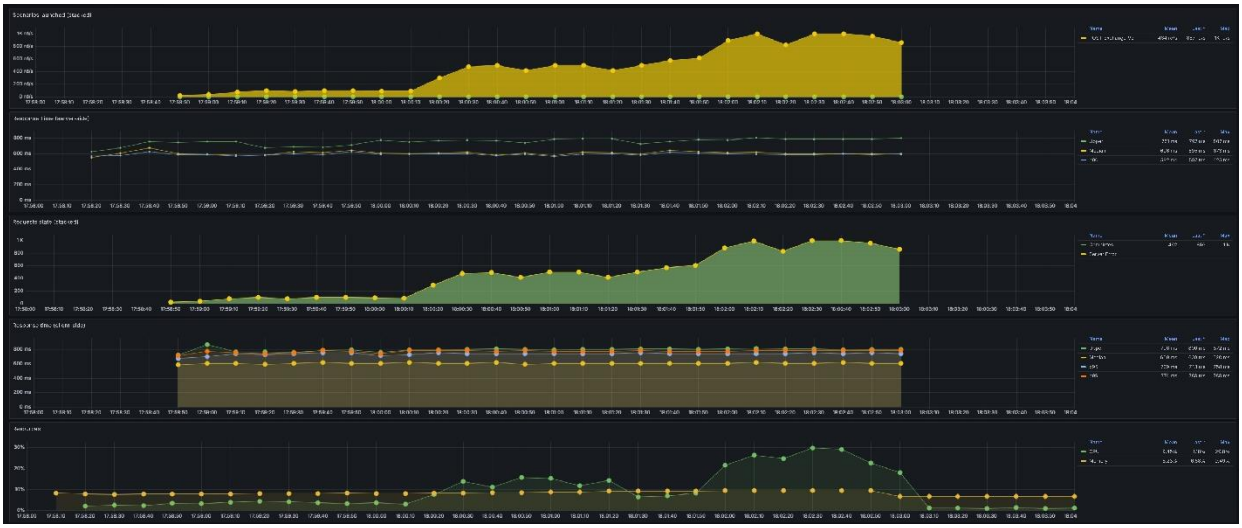
Observando los response time vemos un leve aumento en el tiempo de respuesta entre el lado servidor y el cliente debido al procesamiento de los request y también observamos como hubo cuellos de botella en ciertos puntos .



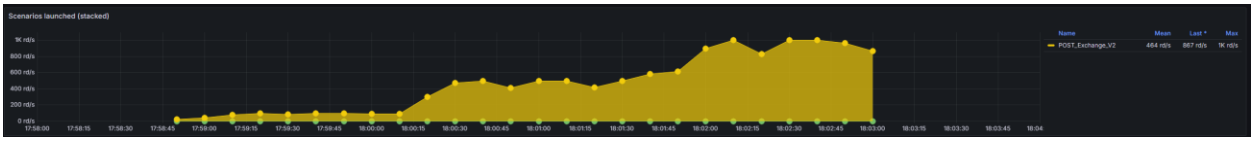
El uso de recursos no llevó a la PC a su límite por lo que el entorno era capaz de procesar todo correctamente.

POST_EXCHANGE_V2 (pool 500)

Analizamos el caso del código con la implementación de Redis



Vemos como también en esta prueba se alcanzan los 1000 request por segundo



Pero en este escenario de carga intensa, la mejora de Redis permitió aumentar la escalabilidad , entonces el sistema ahora procesa correctamente todos los casos. Lo cual representa una mejora sustancial en la estabilidad del sistema y la capacidad de responder eficientemente en situaciones de alta concurrencia.

Los tiempos de respuesta son más estables , sin picos lo cual representa una mayor estabilidad para el cliente y una sensación más consistente del funcionamiento del sistema.

El consumo de recursos fue ligeramente mayor en algunos momentos pero tampoco llevó al limite a la PC y el incremento no muestra un costo significativo de recursos.