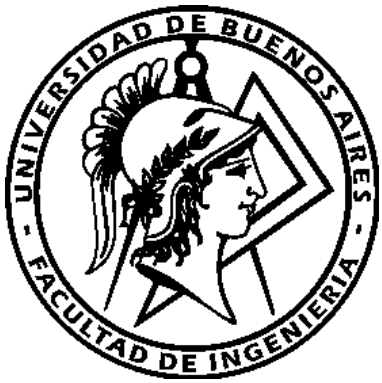
FACULTAD DE INGENIERÍA

UNIVERSIDAD DE BUENOS AIRES

75.73 - Arquitectura Del Software



TRABAJO PRÁCTICO
1er cuatrimestre 2018

Alumnos

Nahuel Martín Sosa	91289
Mauro Cortes	92436

Objetivo

La empresa en la que trabajan está planeando modernizar el producto que hacen. Este producto es un gran monolito (una única aplicación muy grande), que se viene haciendo hace muchos años y se volvió difícil de mantener.

La idea no es rehacerlo entero porque llevaría demasiado tiempo y sería muy riesgoso. En el corto plazo, lo que quieren hacer es separar unas pocas partes pequeñas de la gran aplicación a servicios externos, y hacer que la aplicación use esos servicios. Así, ven que cada servicio va a ser más fácil de mantener.

Tecnologías

Se pondrán a prueba dos tecnologías distintas, desplegadas de forma independiente a través de docker. Una será implementada mediante python, utilizando flask. Mientras que la segunda será bajo Node.js, mediante el framework Express. La principal diferencia entre las mismas, es que python responde de manera sincrónica, mientras que Node lo hace de forma asincrónica. Por lo que, a priori, esperamos ver mejores tiempos de respuesta en Node, aunque a un mayor costo de recursos.

Por último, en base a los resultados obtenidos, se elegirá una de las tecnologías para escalarla y analizar esta nueva distribución. Para poder realizar esto, se incorporará un load balancer mediante Nginx, el cual también se desplegará de forma independiente con docker.

Escenario de prueba

La estructura de los escenarios de prueba será la misma, independientemente de los tipos de endpoints a utilizar. La misma consistirá en un período de pedidos al servidor que será constante y de baja intensidad, seguido de un pico de solicitudes, para luego descender un poco y estabilizarse en otro período de forma constante pero de una intensidad más cercana al pico mencionado previamente.

Lo expuesto puede verse más claramente en el siguiente gráfico:

Se implementarán 3 tipos de endpoints distintos para evaluar los escenarios. Los mismos consistirán en lo siguiente:

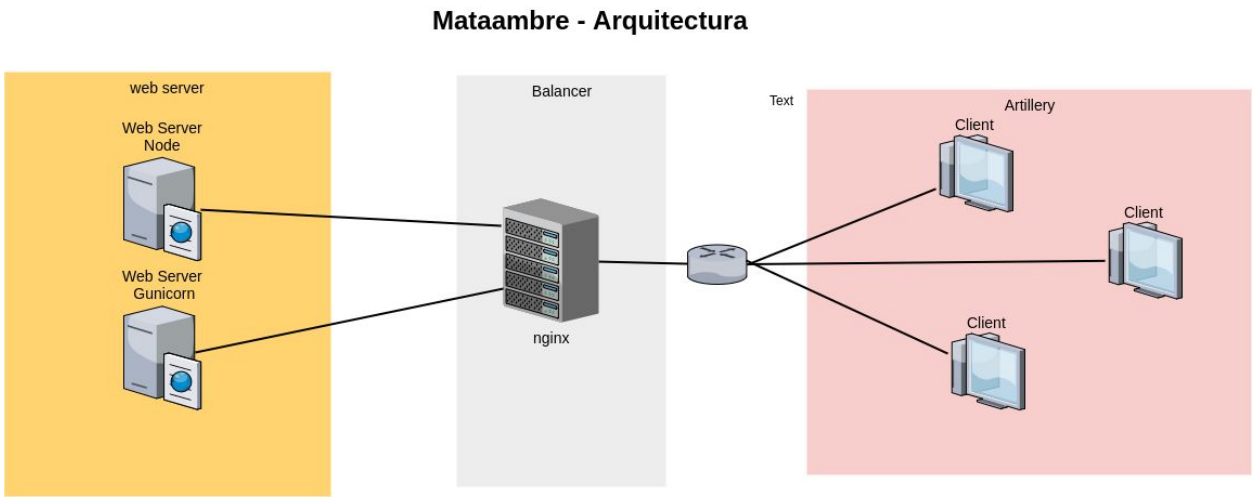
- Endpoint Rápido y Liviano:
    - Será un ping básico, que responderá de forma inmediata y sin consumir recursos. El mismo será utilizado para representar algunas de las funcionalidades básicas que tendrán los nuevos servicios.

- Endpoint Lento y Liviano:
    - Será implementado con un sleep, cual permitirá simular el impacto que tendría llamar a otro servicio externo, por lo que la respuesta final puede demorar pero consumen pocos recursos en ese nodo en procesarlo.

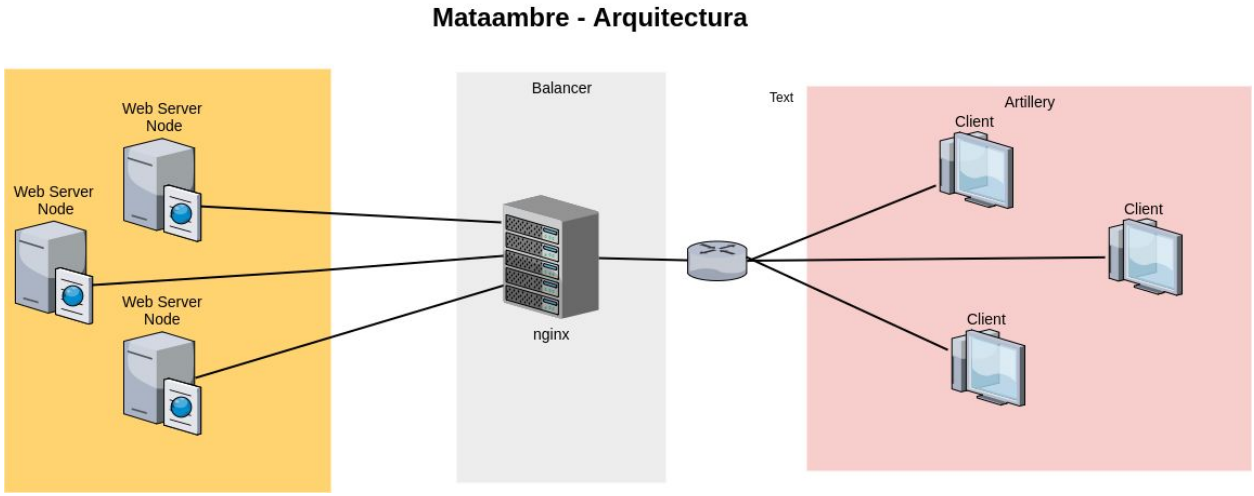
- Endpoint Lento y Pesado:
    - Esta implementación estará basada en un algoritmo para el cálculo de la serie de Fibonacci, en su versión recursiva, para que se genere un stack más pesado y por lo tanto, consuma más recursos.
      En particular, esto servirá para analizar si el uso de CPU y memoria puede marcar una diferencia notable.

Arquitectura propuesta

Arquitectura: Node + Gunicorn + Balancer

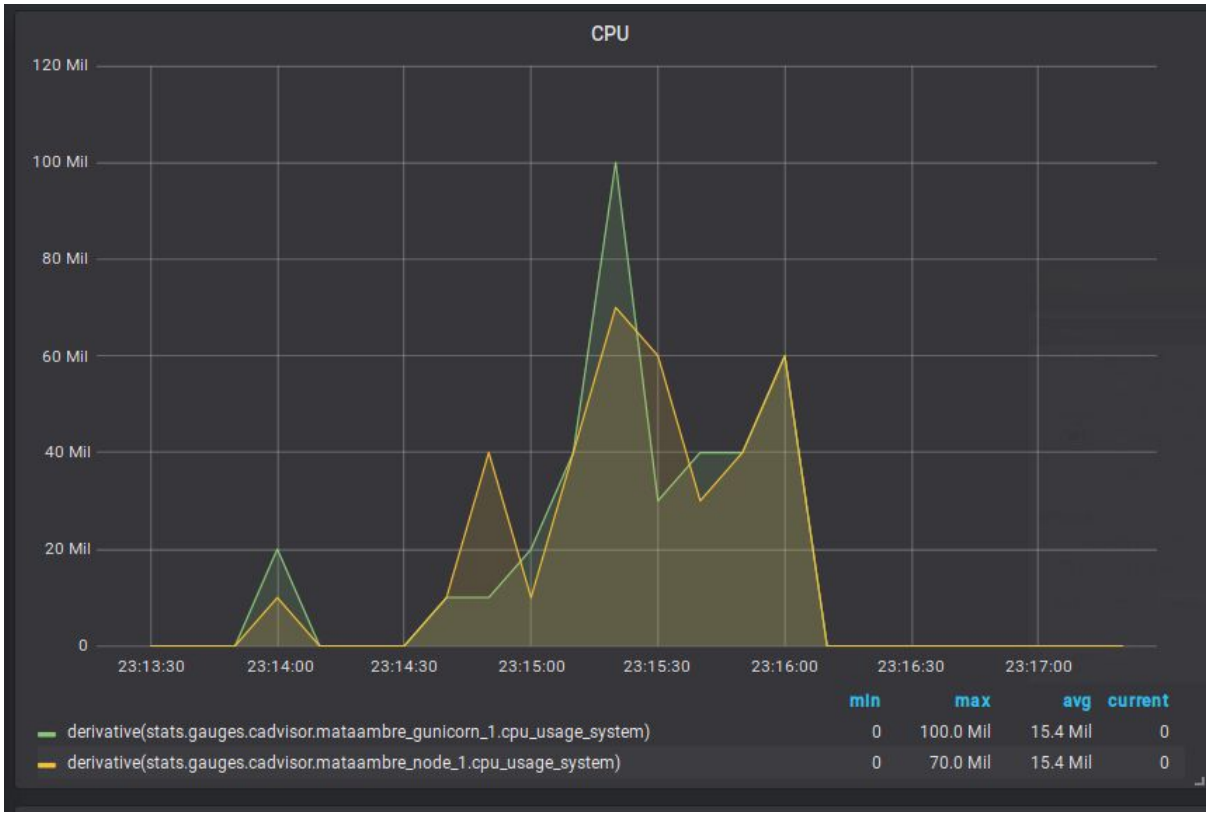


Arquitectura: Node replicado (3 nodos) + Balancer.

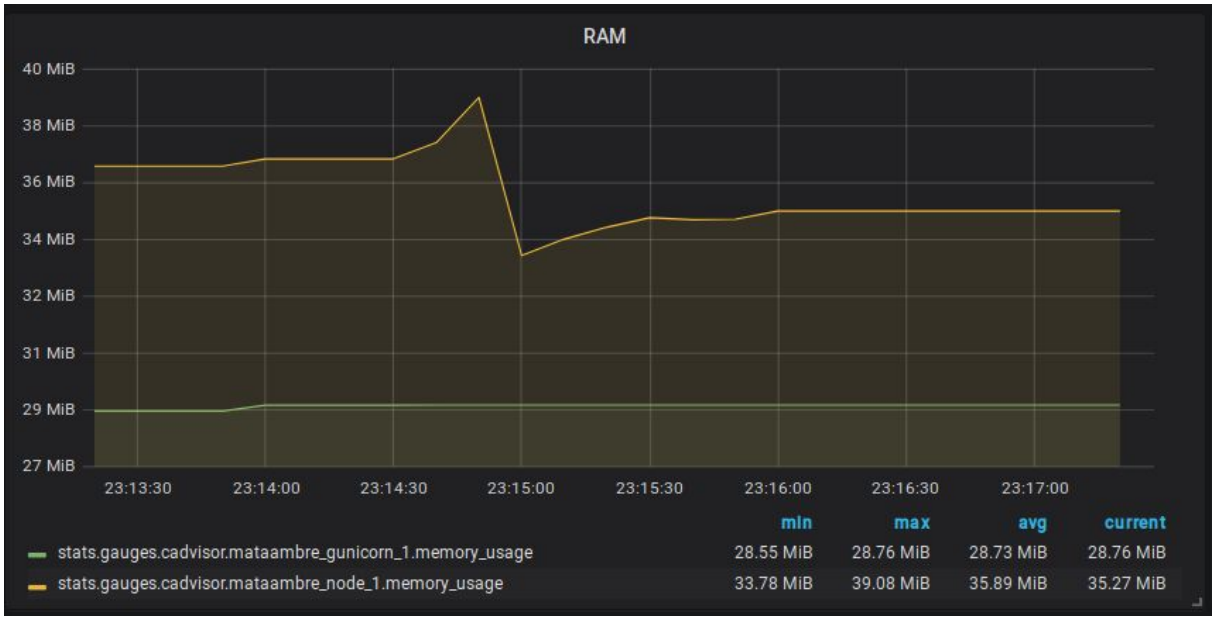


Resultados - Test Ping - Endpoint Rápido y Liviano

CPU - /ping Node (amarillo) Gunicorn (verde)



RAM - /ping Node (amarillo) Gunicorn (verde)



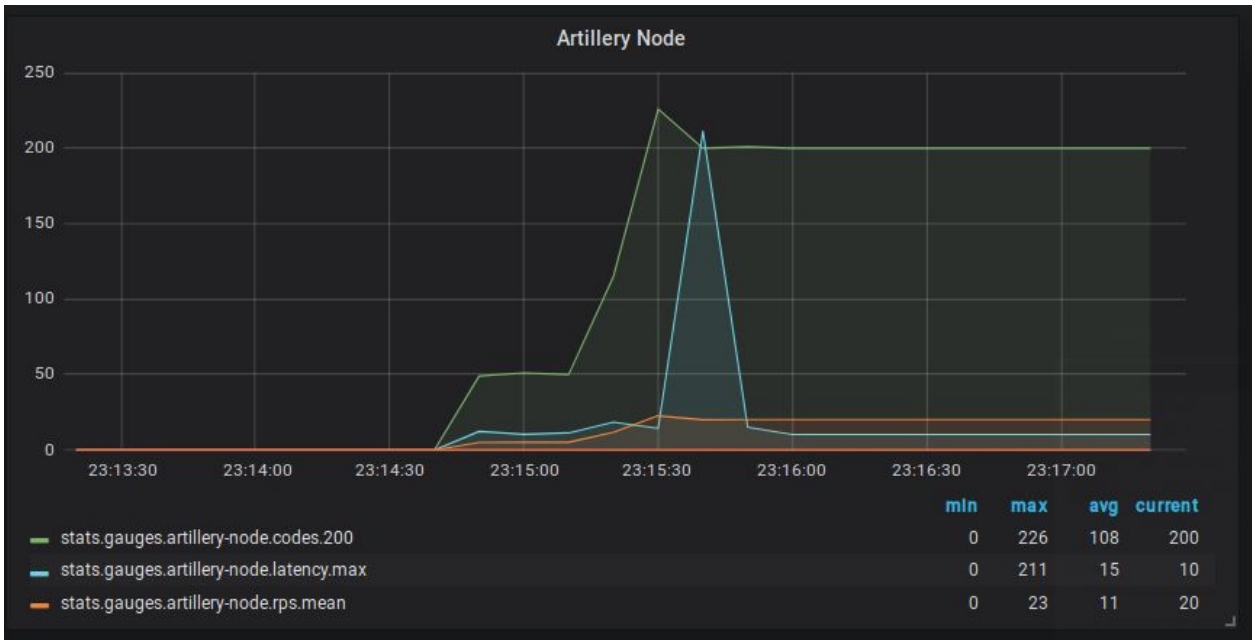
Artillery - /ping

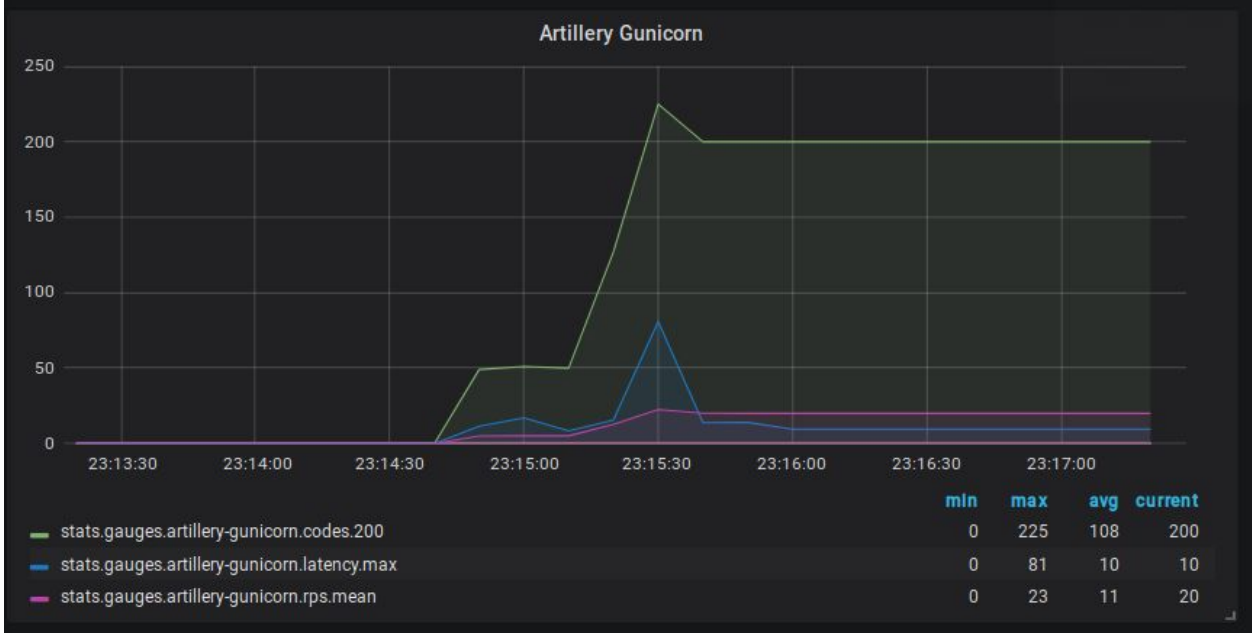
Node - Gráfico 1

Request Exitosos (verde) Latencia máxima (celeste)
Request por segundo (naranja) Request pendientes (rojo)

Gunicorn - Gráfico 2

Request exitosos (verde) Request defectuosos (rojo)
Latencia máxima (azul)
Request por segundo (lila) Request pendientes (violeta)



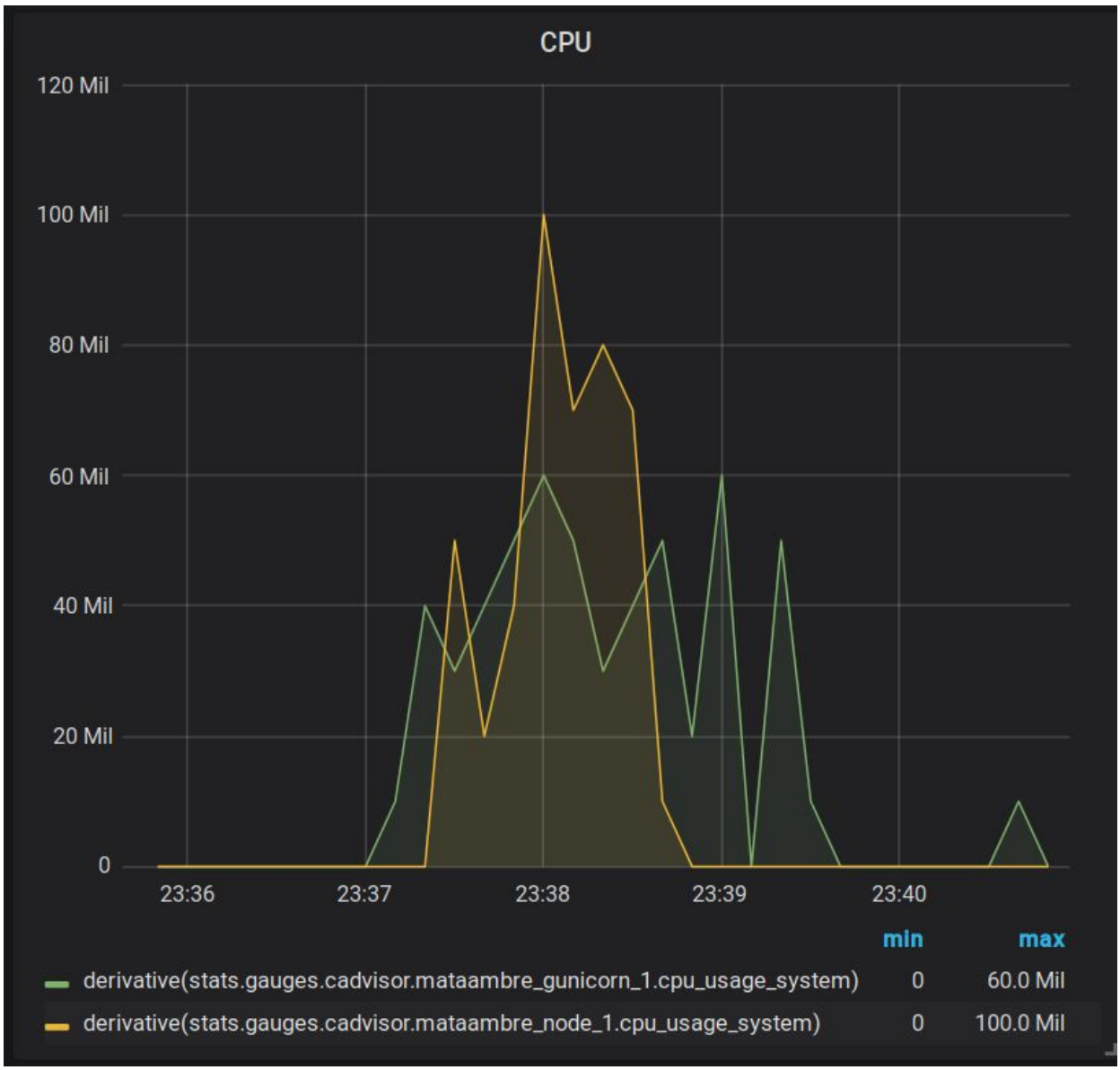


Observaciones

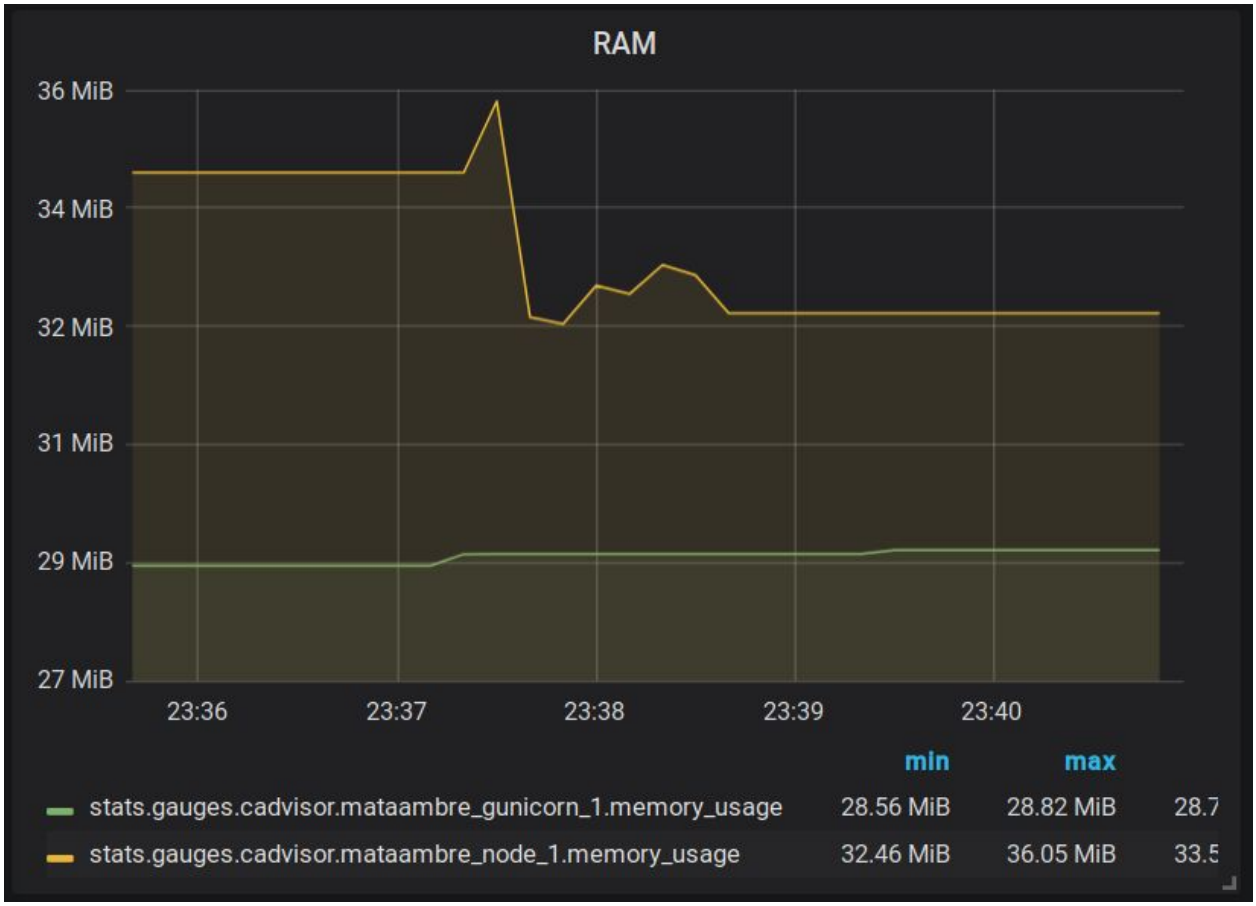
Se puede observar como node consume muchos más recursos, tanto CPU como memoria. Suponemos que se debe al hecho de mantener sus ejecuciones de forma asincrónica.

Resultados - Test Sleep - Endpoint Lento y Liviano

CPU - /sleep Node (amarillo) Gunicorn (verde)



RAM - /sleep Node (amarillo) Gunicorn (verde)



Artillery - /sleep

<u>Node - Gráfico 1</u>

Request Exitosos (verde) Latencia máxima (celeste)
Request por segundo (naranja) Request pendientes (rojo)

<u>Gunicorn - Gráfico 2</u>

Request exitosos (verde) Request defectuosos (rojo)
Latencia máxima (azul)
Request por segundo (lila) Request pendientes (violeta)

Observaciones

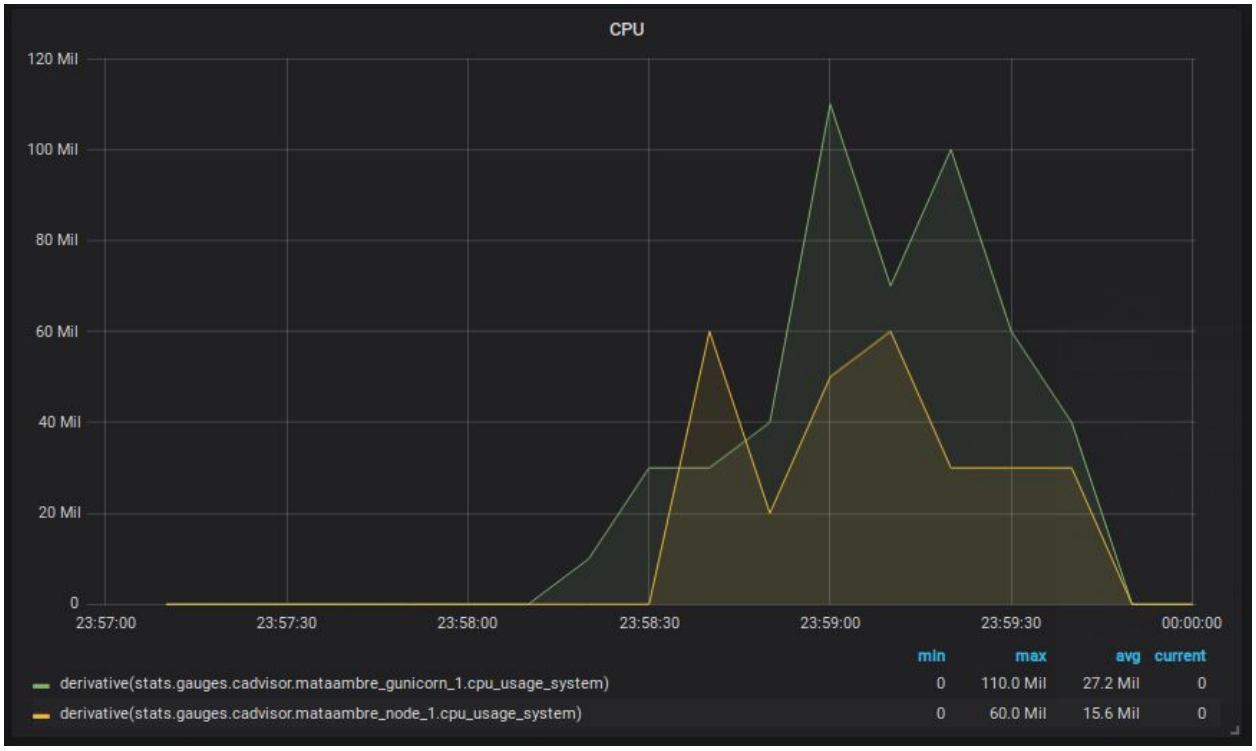
Más allá de los resultados característicos de las pruebas, para incorporar la latencia máxima al gráfico de Artillery-Gunicorn, fue necesario escalarla para que poder apreciar el resto de los valores. Esto es así, ya que la unidad de la misma es milisegundos, y la latencia máxima alcanzó los 80 segundos, debido a que algunos request se perdieron, o han llegado al timeout correspondiente.

Por otro lado, se puede observar del gráfico de CPU, que Node al ser asincrónico, termina de ejecutar los request mucho antes que Gunicorn, logrando a su vez, que todas las respuestas sean exitosas.

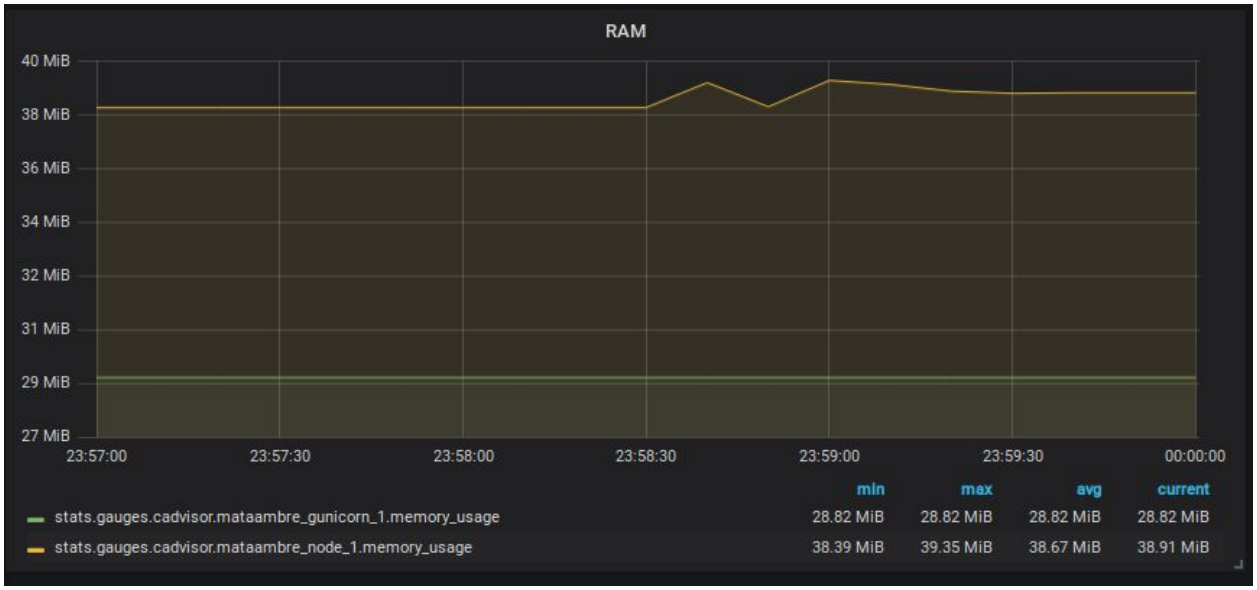
Respecto a la memoria, se puede observar un escenario muy parecido al endpoint del /ping.

Resultados - Test Fibo - Endpoint Lento y Pesado

CPU - /fibo Node (amarillo) Gunicorn (verde)



RAM - /fibo Node (amarillo) Gunicorn (verde)



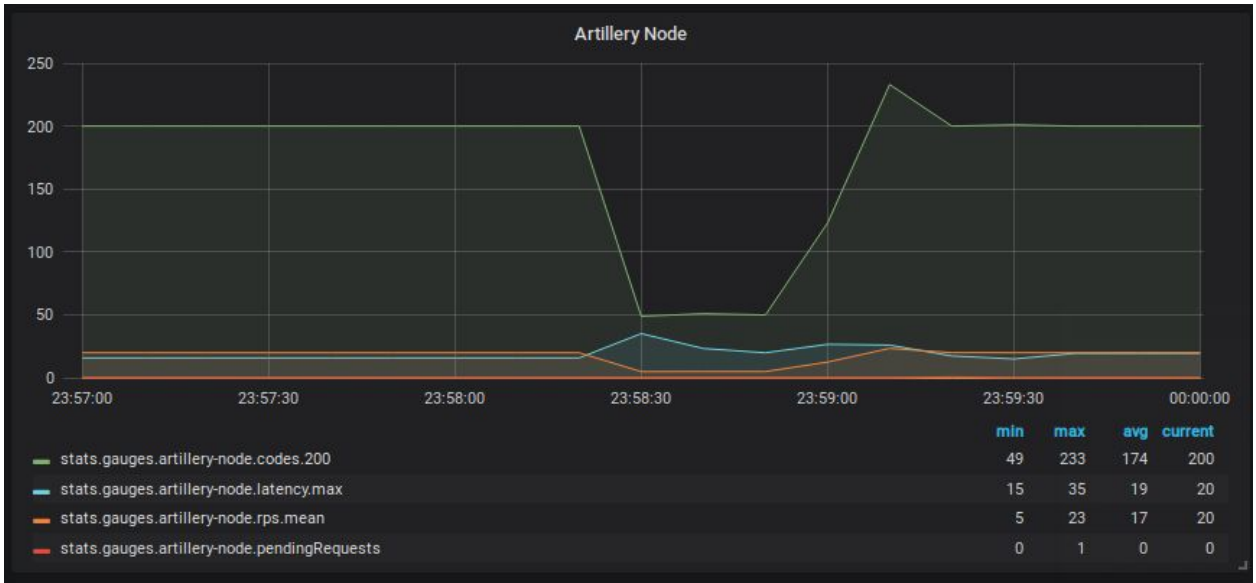
Artillery - /fibo

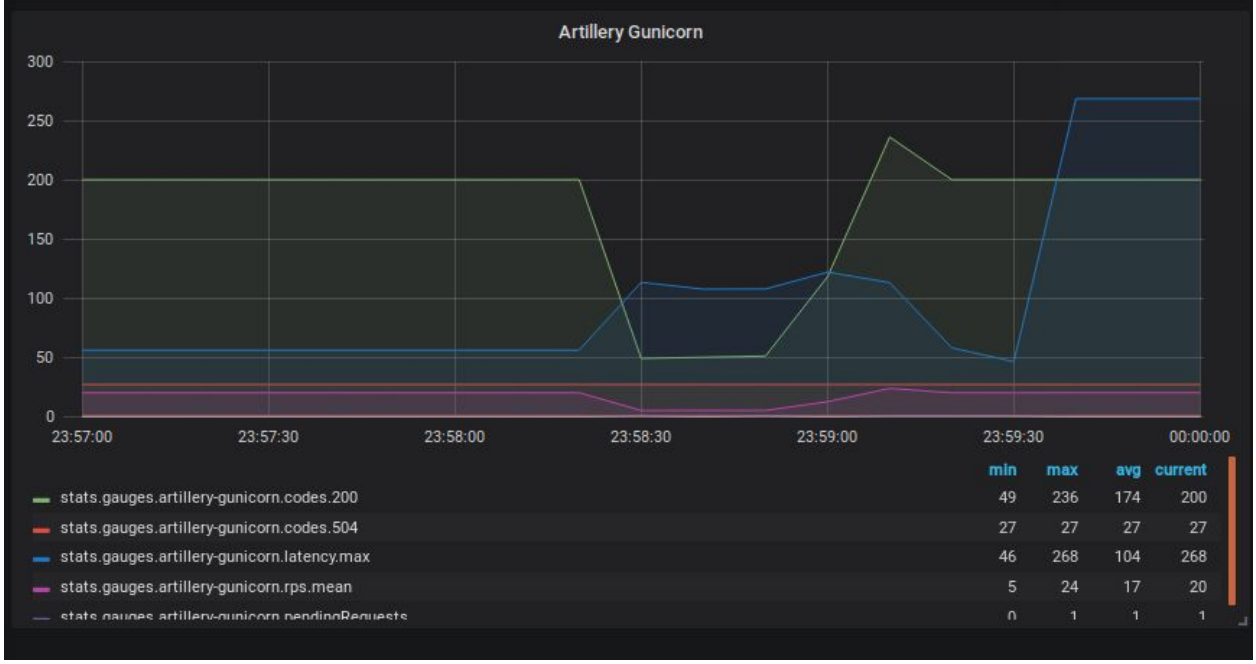
Node - Gráfico 1

Request Exitosos (verde) Latencia máxima (celeste)
Request por segundo (naranja) Request pendientes (rojo)

Gunicorn - Gráfico 2

Request exitosos (verde) Request defectuosos (rojo)
Latencia máxima (azul)
Request por segundo (lila) Request pendientes (violeta)





Observaciones

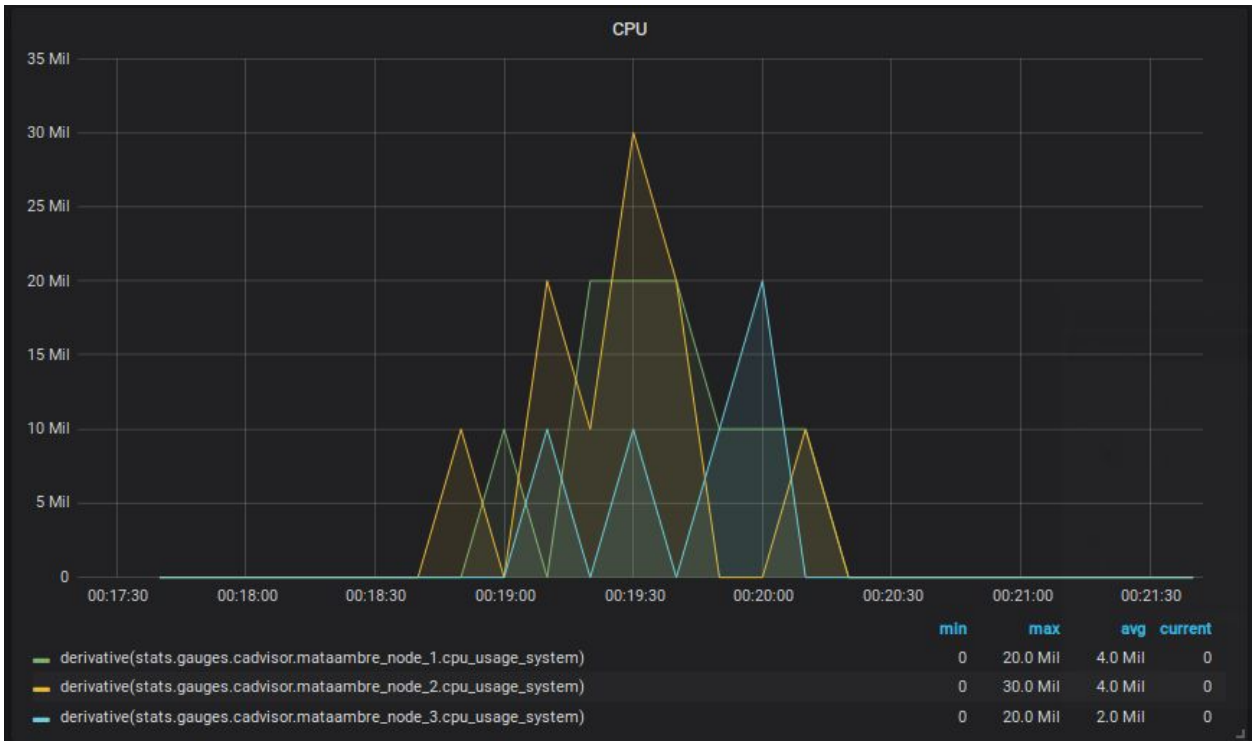
Al igual que en el caso anterior del Endpoint /Sleep, lo más relevante ha sido la imposibilidad por parte de Gunicorn de completar los request de forma exitosa, respecto de la efectividad de Node.

Respecto de los recursos, se puede observar como Node consume más memoria, pero conlleva menos procesamiento, mientras que Gunicorn lo contrario, consume más CPU y menos memoria.

Resultados - Test Ping - Node Replicated - Endpoint Rápido y Liviano

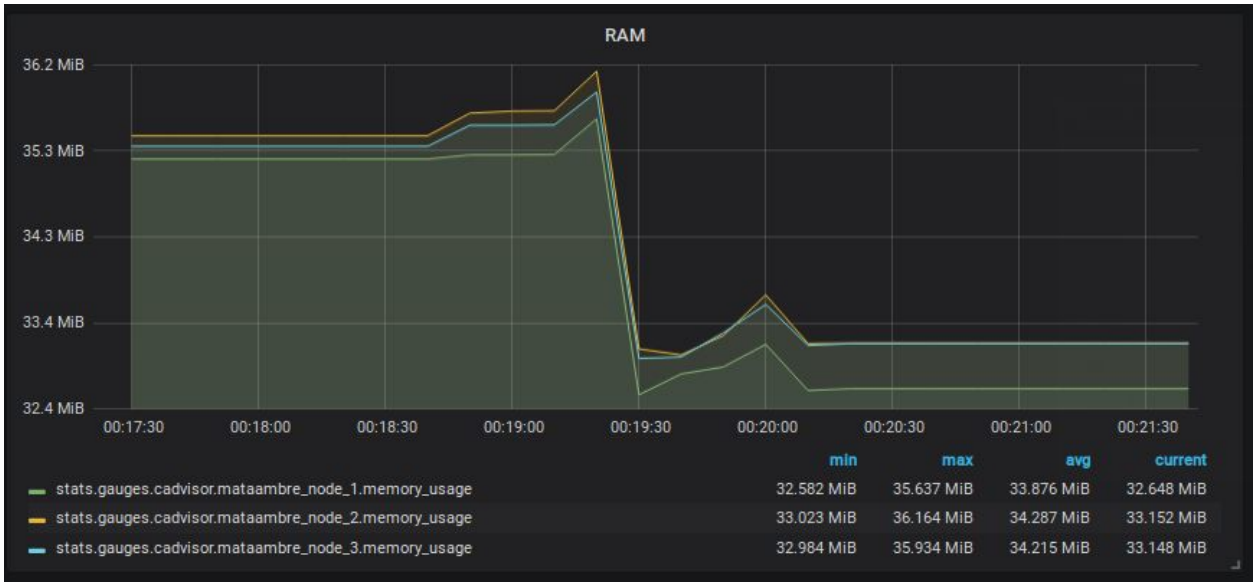
CPU - /ping

Node (3 nodos) + Load Balancer



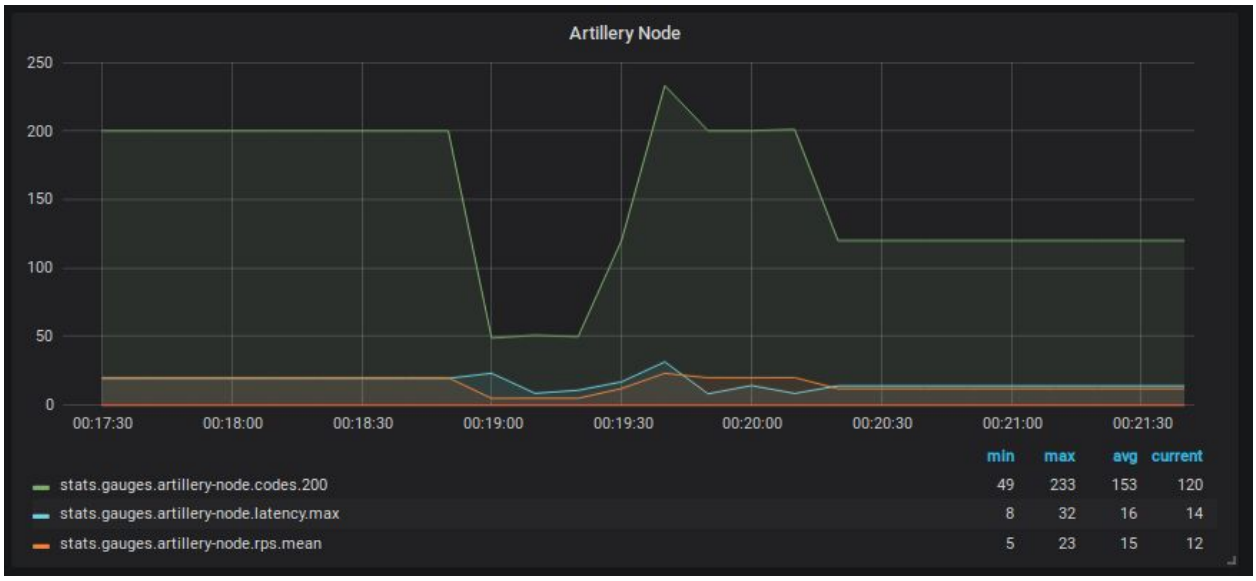
RAM - /ping

Node (3 nodos) + Load Balancer



Artillery- /ping

Node (3 nodos) + Load Balancer



Observaciones

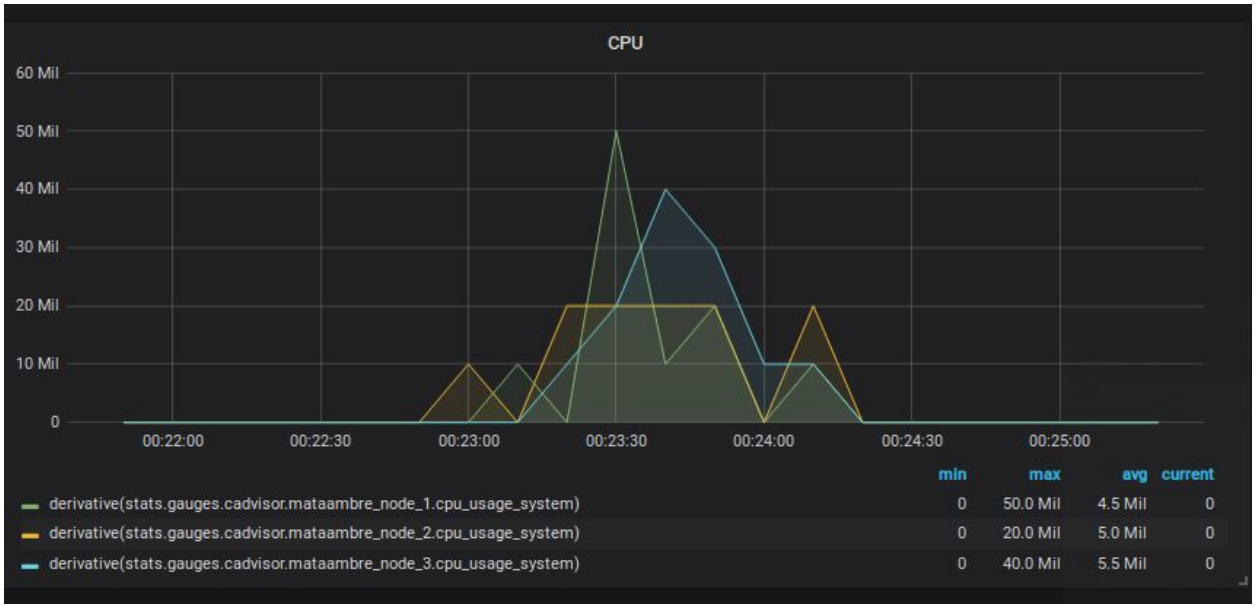
Lo más relevante de este caso, respecto del mismo nodo pero sin replicar, es la mejora en las latencias de respuestas, las cuales rondan los 32 milisegundos máximo, mientras que en la arquitectura de nodo único, llegan hasta los 200 milisegundos.

Se nota como los procesamientos de requests se distribuyen entre los distintos nodos al apreciar los gráficos de CPU y memoria.

Resultados - Test Fibo - Node Replicated - Endpoint Lento y Pesado

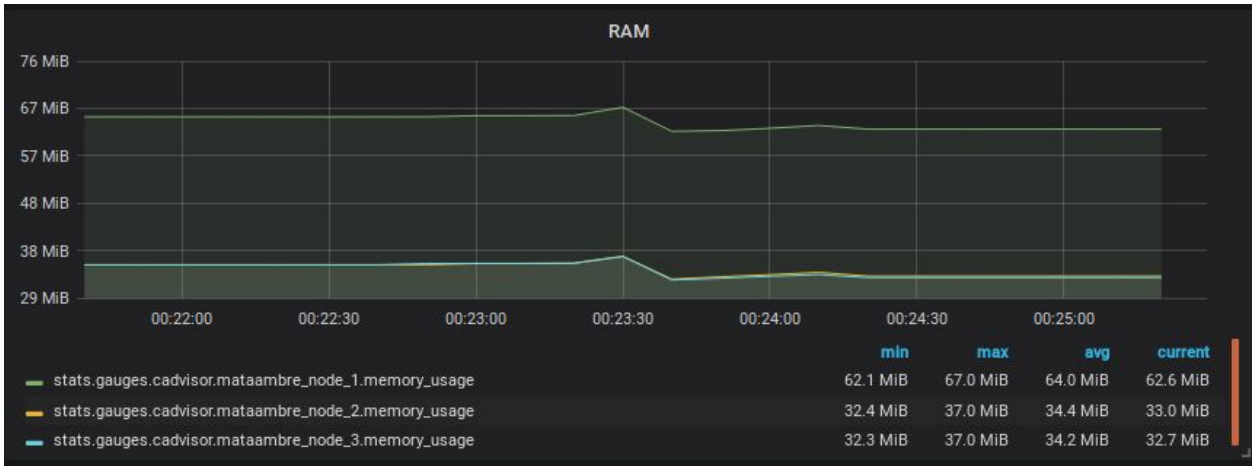
CPU - /fibo

Node (3 nodos) + Load Balancer



RAM - /fibo

Node (3 nodos) + Load Balancer



Artillery- /fibo

Node (3 nodos) + Load Balancer

Observaciones

A este caso le corresponden las mismas aclaraciones que al anterior de /ping, salvo por la latencia máxima de respuesta, donde no se aprecia el salto observado para el endpoint mencionado. Tal vez si se eligiera un número más alto para realizar los cálculos de fibonacci se lograra apreciar una mejora considerable.

Conclusiones:

¿Qué tecnologías conviene usar?

- Recursos disponibles (hardware)
  ¿Qué tecnologías conviene usar?
  Se puede observar como node consume muchos más recursos, tanto CPU como memoria pero todas las respuestas sean exitosas en los distintos escenarios de prueba.

Por el momento gana Node, nos queda medir los siguientes escenarios

1. Límite de carga de peticiones
2. Consulta de recursos de gran estructura.
3. Consumo de red.
4. Software limitado

Continuara...