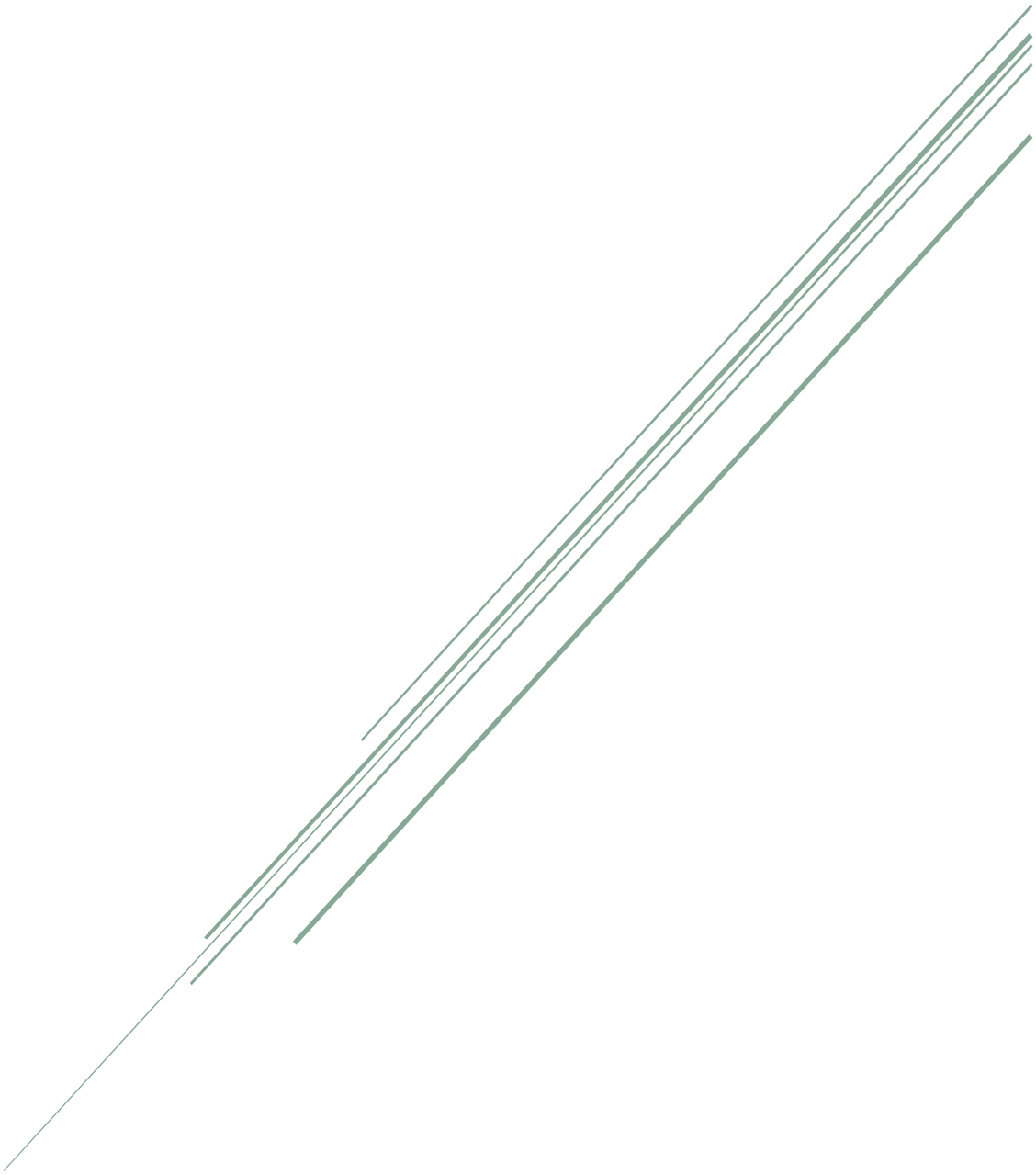
ANÁLISIS COMPLETO DEL PERFORMANCE DEL SERVIDOR



Nuñez Naiara

ANÁLISIS COMPLETO DEL PERFORMANCE DEL SERVIDOR

Introducción

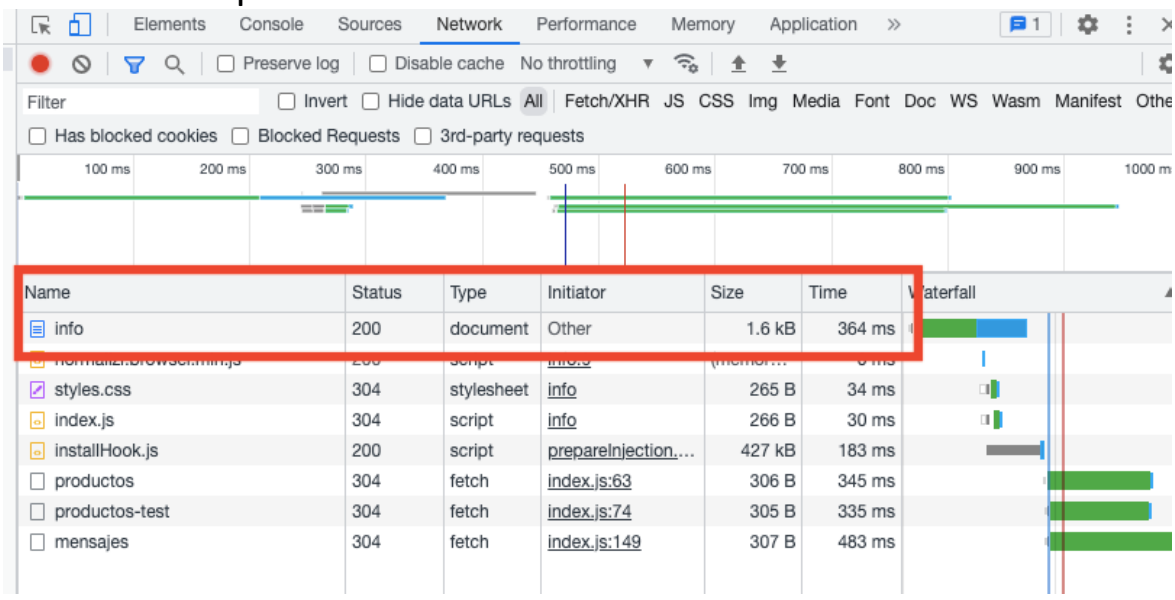
En éste informe realizo el análisis de los datos que obtuve al testear el servidor con artillery, profiling, autocannon, 0x y compresión gzip en la ruta '/info' empleando o no un console.log para mostrar los datos por consola antes de ser enviados al cliente.
Al final realizo una conclusión breve sobre los datos obtenidos.

Desarrollo

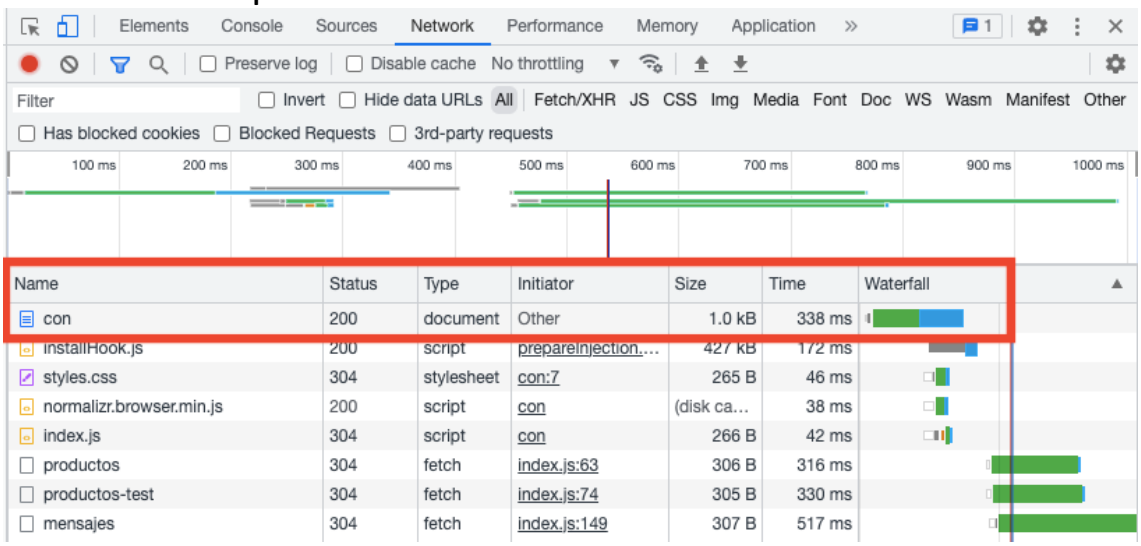
Compresión gzip

A modo de prueba configuré una ruta de 'info/con' para poder hacer la comparación utilizando la compresión gzip. Desde Google con mi cuenta:

- Sin compresión:

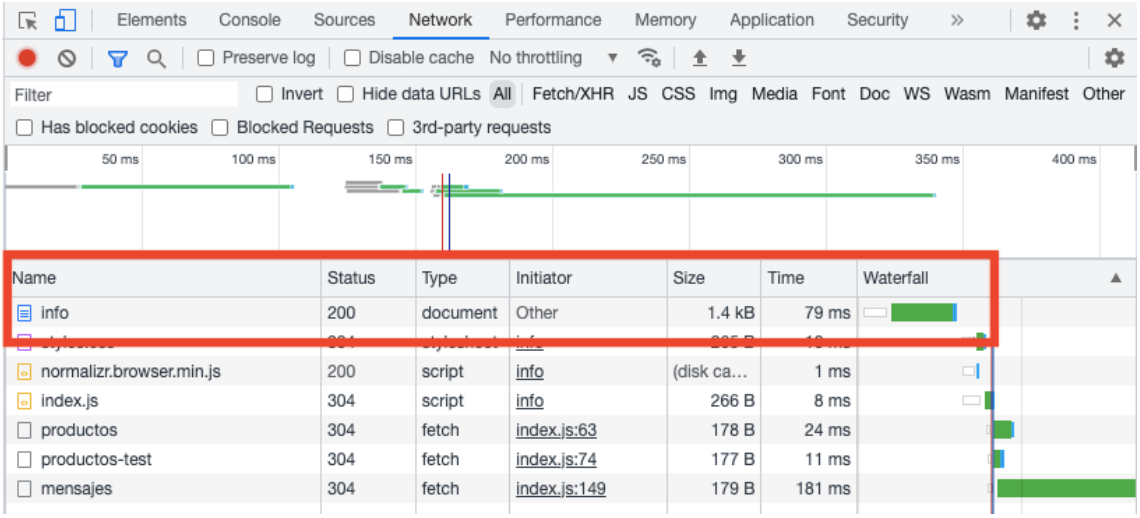


- Con compresión:

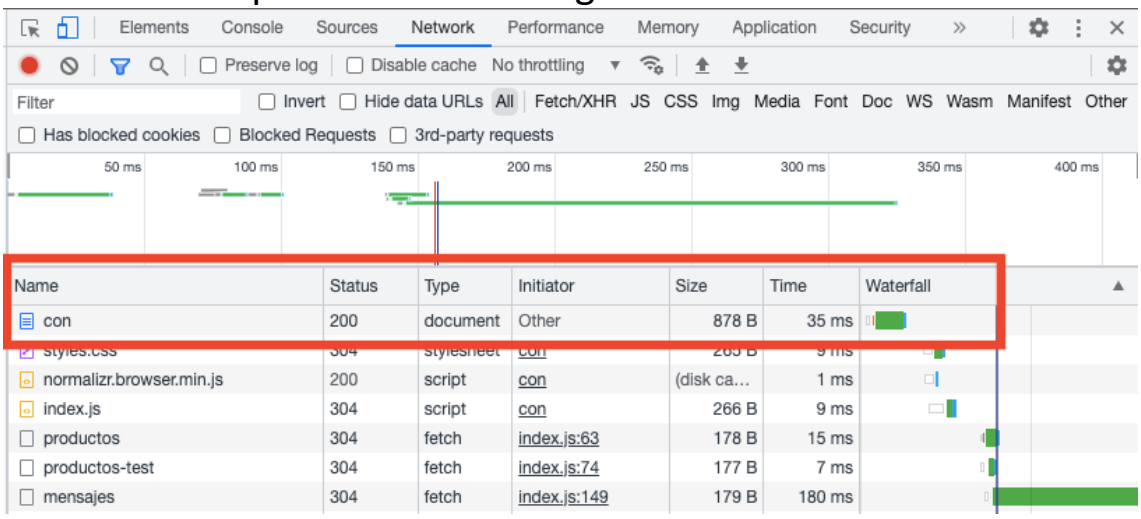


A simple vista sólo hay una diferencia de 0.6kb, esto es debido al caché almacenado en Google según tengo entendido. La diferencia es más obvia cuando se ingresa a la misma ruta en modo incógnito.

- Sin compresión modo incógnito:



- Con compresión modo incógnito:



Acá se aprecia mejor la diferencia de tamaño pasando de 1.4kb sin compresión a 878b con compresión.

Perfilamiento con profiling

Pasando a analizar la ruta '/info' con y sin console.log, estos fueron los resultados obtenidos:

- Con console.log:

```
[Summary]:
  ticks  total  nonlib   name
     31   1.0%    1.0%  JavaScript
   2956  90.7%   98.7%  C++
     67   2.1%    2.2%  GC
    264   8.1%          Shared libraries
      9   0.3%          Unaccounted
```

- Sin console.log:

```
[Summary]:
  ticks   total  nonlib   name
     34    1.1%    1.1%  JavaScript
   2916   91.0%   98.3%  C++
     59    1.8%    2.0%  GC
    240    7.5%          Shared libraries
     16    0.5%          Unaccounted
```

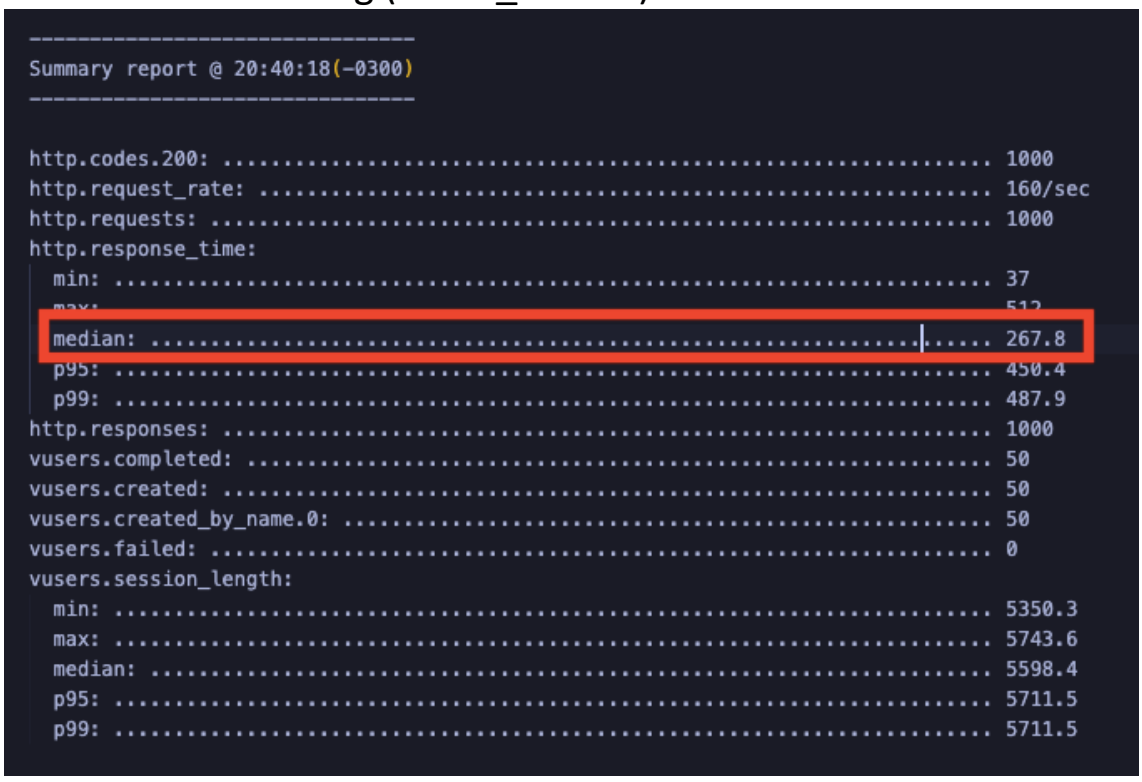
Fijándonos en shared libraries podemos observar una leve diferencia en los ticks. Con console.log figuran 264 ticks mientras que sin mostrar los datos en consola figuran 240 ticks. Son 24 ticks de diferencia.

Para obtener estos archivos emplee el comando “node –prof server.js” y después usé “curl -X GET http://localhost:8080/info”.

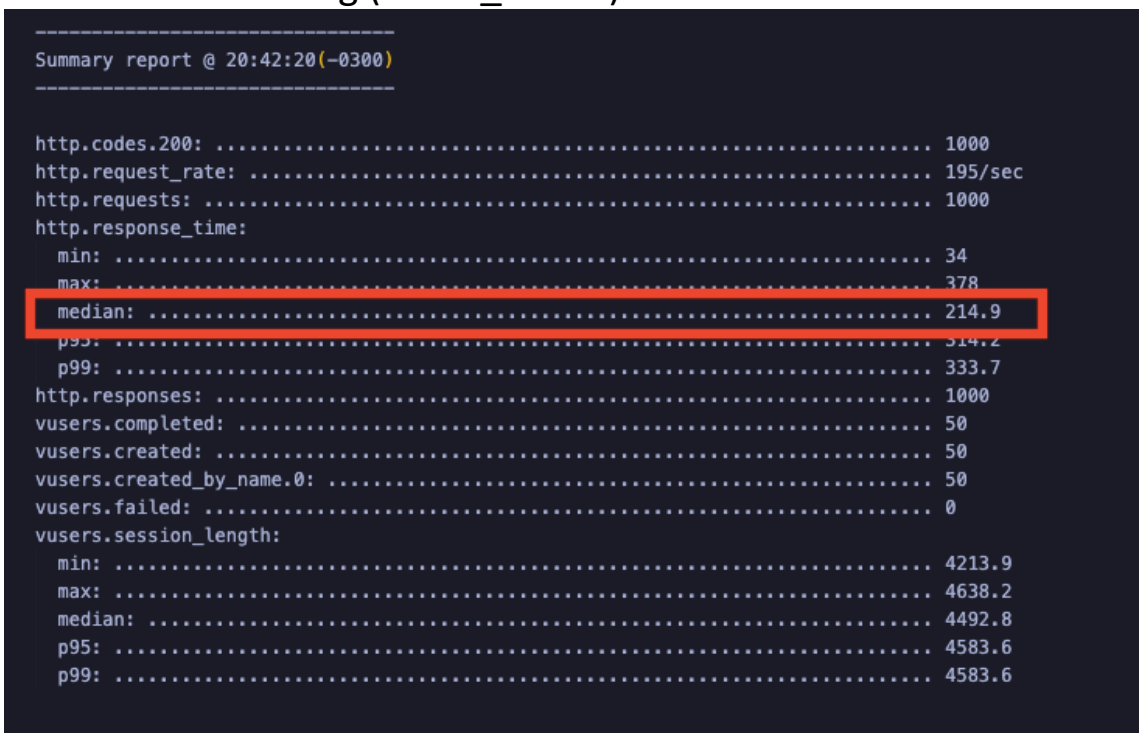
Artillery

Con artillery nos pedían que hiciéramos un testeo de 50 conexiones con 20 peticiones. Basándome en los archivos de result_con.txt y result_sin.txt pude observar lo siguiente:

- Con console.log (result_con.txt):



- Sin console.log (result_sin.txt):



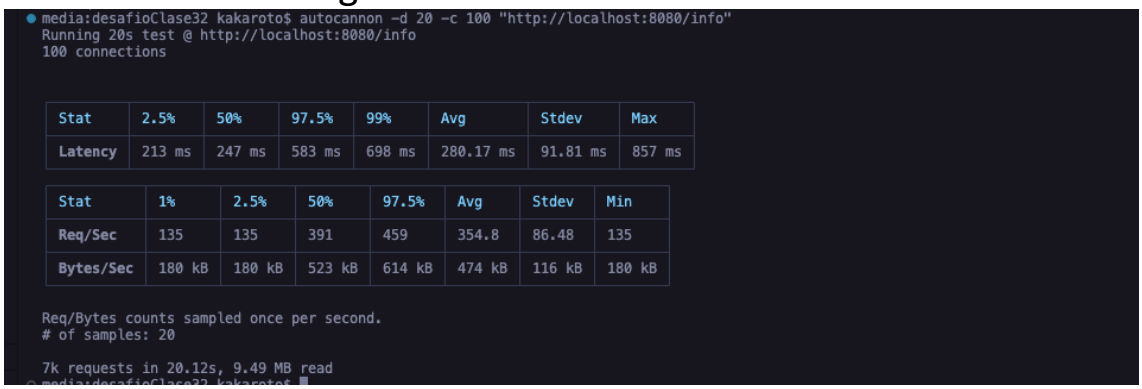
```
-------------------------------
Summary report @ 20:42:20(-0300)
-------------------------------

http.codes.200: ..................................................... 1000
http.request_rate: .................................................. 195/sec
http.requests: ...................................................... 1000
http.response_time:
  min: .............................................................. 34
  max: .............................................................. 378
  median: ........................................................... 214.9
  p95: .............................................................. 314.2
  p99: .............................................................. 333.7
http.responses: ..................................................... 1000
vusers.completed: ................................................... 50
vusers.created: ..................................................... 50
vusers.created_by_name.0: ........................................... 50
vusers.failed: ...................................................... 0
vusers.session_length:
  min: .............................................................. 4213.9
  max: .............................................................. 4638.2
  median: ........................................................... 4492.8
  p95: .............................................................. 4583.6
  p99: .............................................................. 4583.6
```

Lo que podemos observar es que entre 267.8 de respuesta de tiempo de la primera imagen y 214.9 de la segunda, hay una diferencia de casi 53 segundos.

Autocannon

Con autocannon teníamos que establecer 100 conexiones en 20 segundos. Esto es lo que se mostró por consola en cada caso:

- Sin console.log:



```
media:desafioClase32 kakaroto$ autocannon -d 20 -c 100 "http://localhost:8080/info"
Running 20s test @ http://localhost:8080/info
100 connections
```

Stat	2.5%	50%	97.5%	99%	Avg	Stdev	Max
Latency	213 ms	247 ms	583 ms	698 ms	280.17 ms	91.81 ms	857 ms

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	135	135	391	459	354.8	86.48	135
Bytes/Sec	180 kB	180 kB	523 kB	614 kB	474 kB	116 kB	180 kB

```
Req/Bytes counts sampled once per second.
# of samples: 20

7k requests in 20.12s, 9.49 MB read
```

- Con console.log:

```
media:desafioClase32 kakaroto$ autocannon -d 20 -c 100 "http://localhost:8080/info"
Running 20s test @ http://localhost:8080/info
100 connections
```

Stat	2.5%	50%	97.5%	99%	Avg	Stdev	Max
Latency	376 ms	621 ms	1758 ms	1863 ms	717.78 ms	317.27 ms	1890 ms

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	50	50	150	229	138.35	52.33	50
Bytes/Sec	66.9 kB	66.9 kB	201 kB	306 kB	185 kB	69.9 kB	66.8 kB

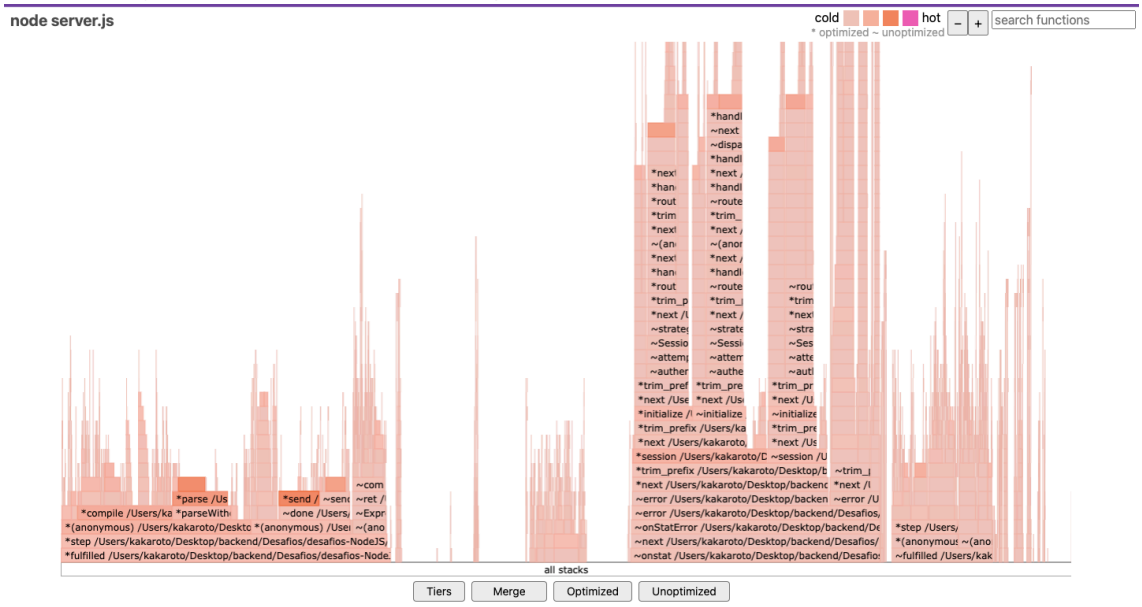
```
Req/Bytes counts sampled once per second.
# of samples: 20

3k requests in 20.15s, 3.7 MB read
```

Como podemos observar en la última línea figura que al mostrar los datos de la ruta '/info' por consola genera 3mil peticiones en 20 segundos. En cambio, si no mostramos los datos de la ruta por consola obtenemos 7 mil peticiones en 20 segundos. Acá se puede notar con más detalle lo pesado que puede resultar el console.log en nuestra aplicación.

Diagrama de flama 0x

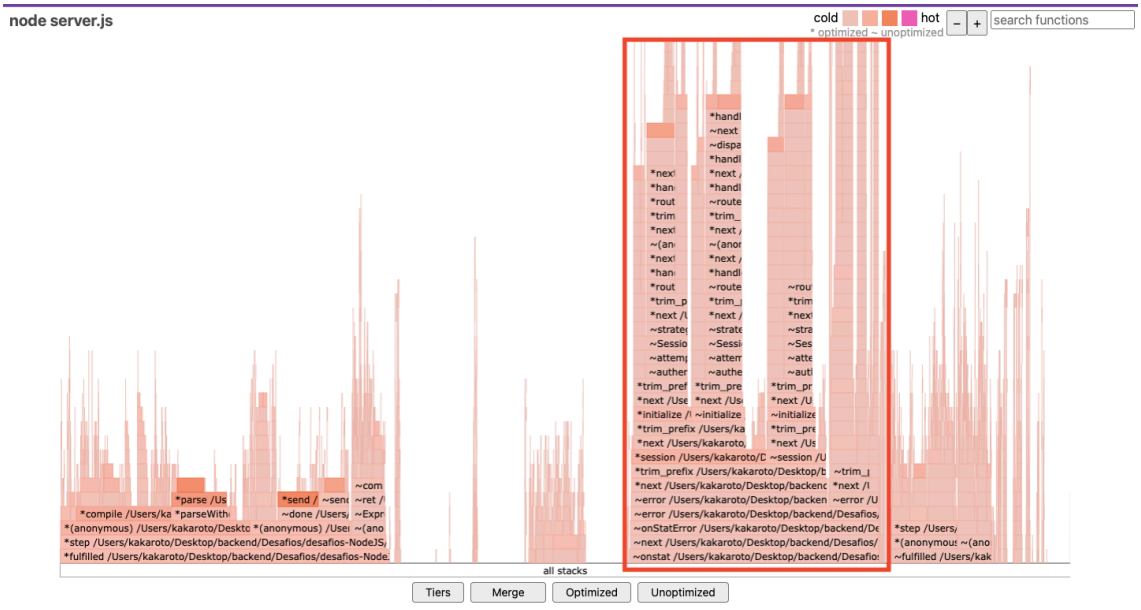
Acá es donde se volvió confuso identificar los datos del diagrama. Hice dos pruebas diferentes, una renderizando los datos con handlebars y otra con res.json para ver si el diagrama era más fácil de analizar.

- El primer caso dio el siguiente resultado:



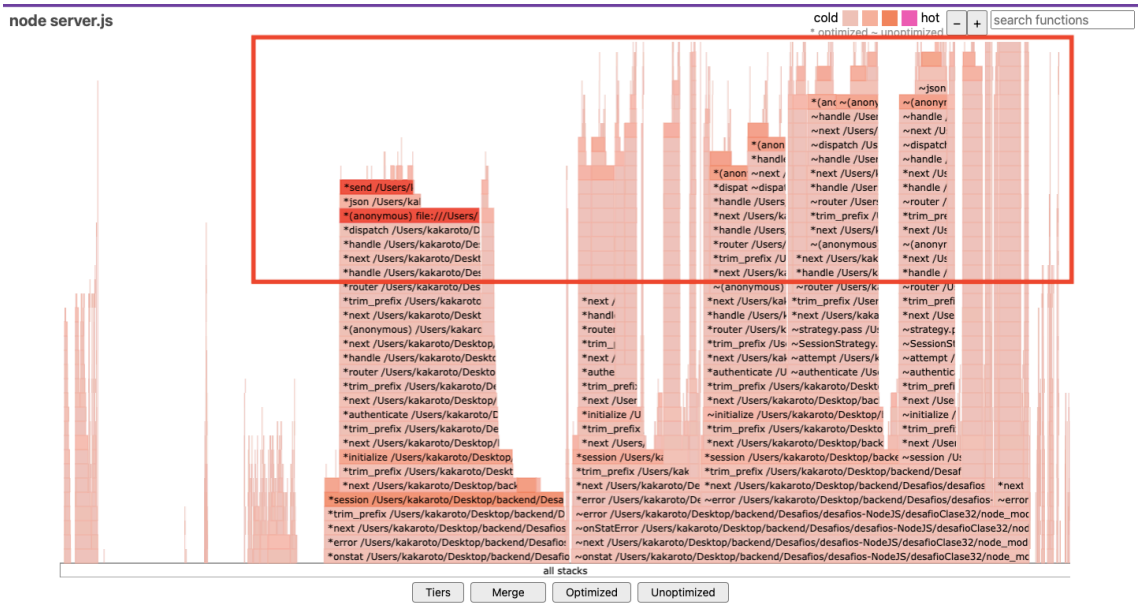
- El segundo caso dio el siguiente resultado:

En ambas situaciones utilicé artillery primero en la función de la ruta de /info con el console.log y después sin el console.log. Originalmente pensé que, en la primera foto, el lado izquierdo representaba la primera función y el lado derecho la segunda. Pero al ver la información me di cuenta que la mayoría de cosas son las funciones de node_modules. Únicamente encontré las funciones del console.log en esta sección:



Son los cuadrados naranja más oscuro.

En el segundo intento fue más notorio lo que ralentizó el console.log al funcionamiento de la app.



Inspect

Intenté hacer el análisis con Google, pero no me aparecieron los segundos que duró cada función. Lo que sí apareció fue el console.log arriba de todas las funciones como lo que más tardó en cargar.

Self Time		Total Time		Function	
22636.8 ms		22636.8 ms		(idle)	
1438.2 ms	15.47 %	1586.9 ms	17.07 %	consoleCall	
1438.2 ms	15.47 %	1586.9 ms	17.07 %	(anonymous)	infoRouter.js:7
558.5 ms	6.01 %	558.5 ms	6.01 %	getCPUs	

Conclusión

A nivel general, lo que se pudo observar es que el console.log ralentiza más la aplicación por lo que provoca que se resuelvan menos peticiones de los usuarios. Esto provoca un bajo rendimiento en la aplicación y no es un caso deseable si se llevara a producción.

Esperaba que la diferencia en algunos casos fuera más marcada que lo que dio el profiling, pero a gran escala se notó el cambio.