ANÁLISIS COMPLETO DE PERFORMANCE DEL SERVIDOR

El primer test que le realice al servidor es a través del --prof de Nodejs, en dicho test (como también en los posteriores) lo que hice fue analizar la ruta /info y la /info2, en donde se compara el rendimiento entre ambas cuya única diferencia es la implementación, en el caso de /info2, de un console.log que me muestra por consola los datos antes de entregarlos al render de la vista.

PROF INTERNO

Este test arrojo los siguiente datos:

Con console.log

Statistical profiling result from logInfo.log, (14000 ticks, 3 unaccounted, 0 excluded).

[Summary]:

ticks	total	nonlib	name
109	0.8%	97.3%	JavaScript
0	0.0%	0.0%	C++
61	0.4%	54.5%	GC
13888	99.2%		Shared libraries
3	0.0%		Unaccounted

Sin console.log

Statistical profiling result from nologInfo.log, (3864 ticks, 0 unaccounted, 0 excluded).

[Summary]:

ticks	total	nonlib	name
4	0.1%	100.0%	JavaScript
0	0.0%	0.0%	C++

29	0.8%	725.0%	GC
3860	99.9%		Shared libraries

Aqui podemos ver que la diferencia de latencia entre las dos rutas es de casi 4 veces.

ARTILLERY

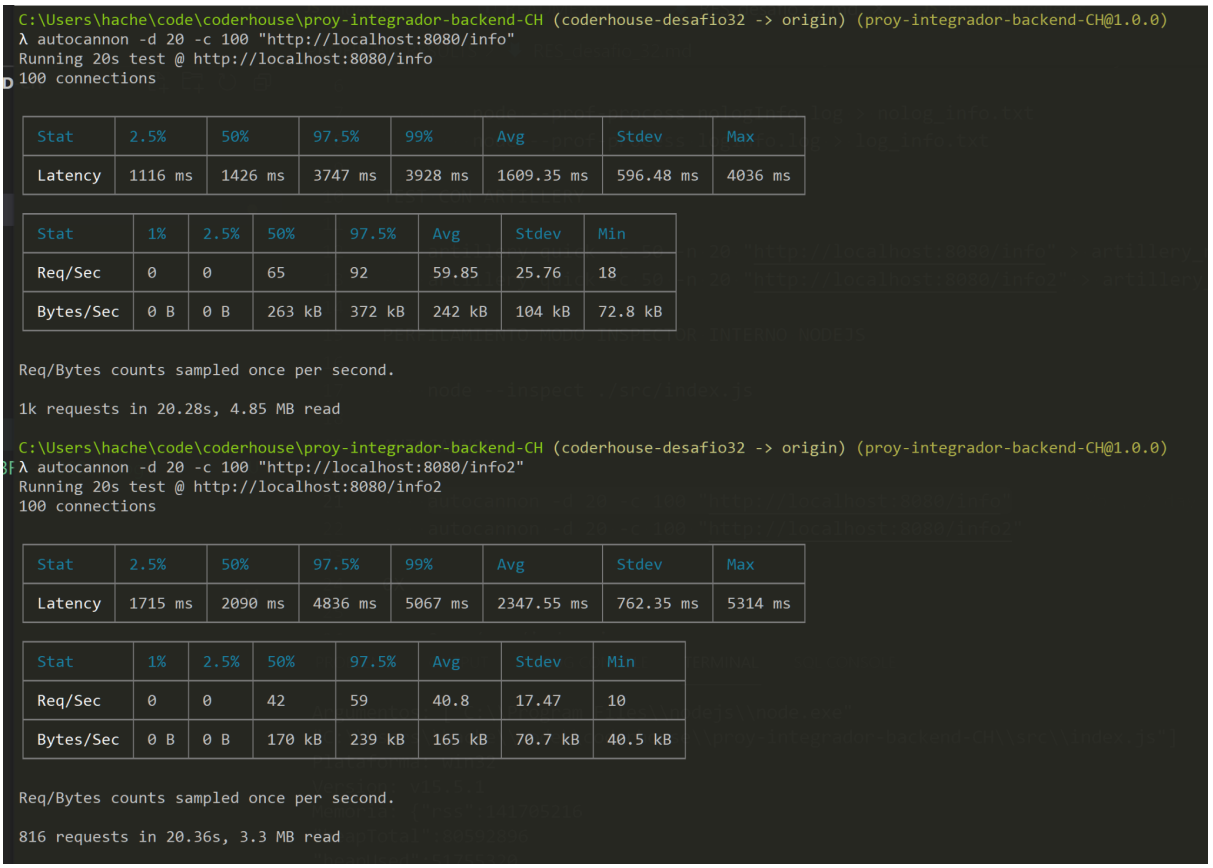
Luego, hice el test de carga con Artillery el cual me dio los siguientes resultados:

Con console.log	Sin console.log
All virtual users finished Summary report @ 12:46:16(-0300) 2021-05-18 Scenarios launched: 50 Scenarios completed: 50 Requests completed: 1000 Mean response/sec: 41.05 Response time (msec): min: 352 max: 1626 median: 879 p95: 1359 p99: 1455.5 Scenario counts: 0: 50 (100%) Codes: 200: 1000	All virtual users finished Summary report @ 12:47:50(-0300) 2021-05-18 Scenarios launched: 50 Scenarios completed: 50 Requests completed: 1000 Mean response/sec: 47.98 Response time (msec): min: 204 max: 933 median: 594 p95: 812 p99: 861 Scenario counts: 0: 50 (100%) Codes: 200: 1000

En este caso vemos que la diferencia de respuesta entre las dos rutas no es tan marcada como en el test anterior, ya que no llega siquiera a ser del doble.

AUTOCANNON

Se realizo este test emulando 100 conexiones concurrentes en un tiempo de 20 segundos.
Los resultados obtenidos se muestran en la siguiente comparativa, el primer caso es el correspondiente a la ruta sin console.log y el segundo a la que si tiene console.log:

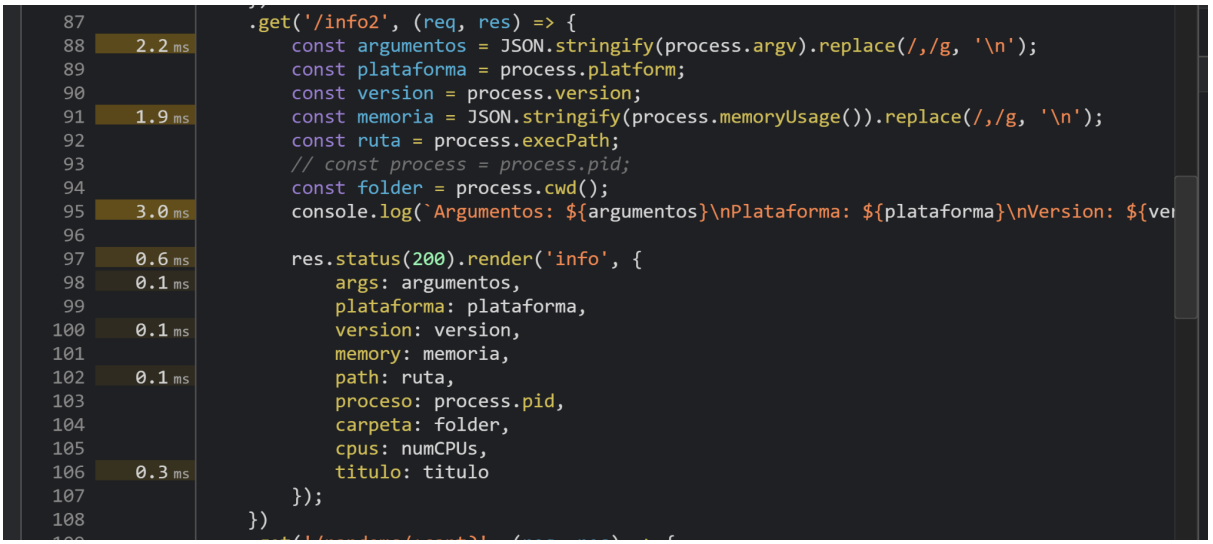


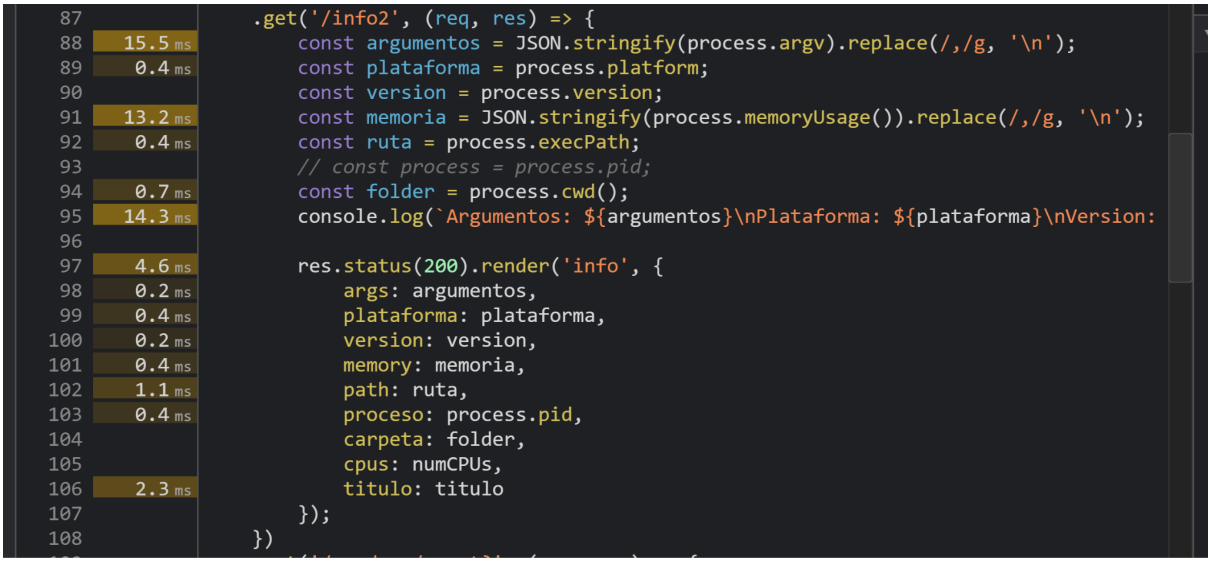
```
C:\Users\hache\code\coderhouse\proy-integrador-backend-CH (coderhouse-desafio32 -> origin) (proy-integrador-backend-CH@1.0.0)
λ autocannon -d 20 -c 100 "http://localhost:8080/info"
Running 20s test @ http://localhost:8080/info
100 connections

| Stat    | 2.5%    | 50%     | 97.5%   | 99%     | Avg        | Stdev     | Max     |
| Latency | 1116 ms | 1426 ms | 3747 ms | 3928 ms | 1609.35 ms | 596.48 ms | 4036 ms |

| Stat      | 1%  | 2.5% | 50%    | 97.5%  | Avg    | Stdev    | Min     |
| Req/Sec   | 0   | 0    | 65     | 92     | 59.85  | 25.76    | 18      |
| Bytes/Sec | 0 B | 0 B  | 263 kB | 372 kB | 242 kB | 104 kB   | 72.8 kB |

Req/Bytes counts sampled once per second.

1k requests in 20.28s, 4.85 MB read

C:\Users\hache\code\coderhouse\proy-integrador-backend-CH (coderhouse-desafio32 -> origin) (proy-integrador-backend-CH@1.0.0)
λ autocannon -d 20 -c 100 "http://localhost:8080/info2"
Running 20s test @ http://localhost:8080/info2
100 connections

| Stat    | 2.5%    | 50%     | 97.5%   | 99%     | Avg        | Stdev     | Max     |
| Latency | 1715 ms | 2090 ms | 4836 ms | 5067 ms | 2347.55 ms | 762.35 ms | 5314 ms |

| Stat      | 1%  | 2.5% | 50%    | 97.5%  | Avg    | Stdev    | Min     |
| Req/Sec   | 0   | 0    | 42     | 59     | 40.8   | 17.47    | 10      |
| Bytes/Sec | 0 B | 0 B  | 170 kB | 239 kB | 165 kB | 70.7 kB  | 40.5 kB |

Req/Bytes counts sampled once per second.

816 requests in 20.36s, 3.3 MB read
```

Nuevamente se observa, que si bien hay un incremento en la latencia del segundo con respecto al primero, de ninguna manera llega a duplicar ese valor, como asi tampoco la cantidad de respuestas por segundos, que se acerca a un 50% por encima uno del otro en promedio.

MODO INSPECT

En este caso, he realizado los test en repetidas oportunidades (aqui solo consigno dos de cada ruta) dado que no recibia los mismos valores al repetir el mismo test, sin embargo he podido constatar que los procesos que mas tiempo insumian seguian siendo los mismos. Y dentro de esos procesos estaba por supuesto el console.log
Como curiosidad, en el caso de la ruta con console.log, este proceso paso de ser el que mas tiempo consumia a ser el segundo, desplazado por el stringify con replace del process.argv.
Los dos primeros print de pantalla corresponden a la ruta con console.log y los dos segundos a la que no lo tiene:





```
74          .get('/info', (req, res) => {
75   1.5 ms     res.status(200).render('info', {
76   4.0 ms         args: JSON.stringify(process.argv).replace(/,/g, '\n'),
77              plataforma: process.platform,
78              version: process.version,
79   2.3 ms     memory: JSON.stringify(process.memoryUsage()).replace(/,/g, '\n'),
80   0.1 ms     path: process.execPath,
81              proceso: process.pid,
82   0.1 ms     carpeta: process.cwd(),
83              cpus: numCPUs,
84   0.4 ms     titulo: titulo
85          });
86      })
```

```
73      })
74      .get('/info', (req, res) => {
75   2.3 ms     res.status(200).render('info', {
76   3.6 ms         args: JSON.stringify(process.argv).replace(/,/g, '\n'),
77              plataforma: process.platform,
78   0.2 ms     version: process.version,
79   3.3 ms     memory: JSON.stringify(process.memoryUsage()).replace(/,/g, '\n'),
80              path: process.execPath,
81              proceso: process.pid,
82              carpeta: process.cwd(),
83              cpus: numCPUs,
84   0.7 ms     titulo: titulo
85          });
86      })
```

DIAGRAMA DE FLAMA CON 0x

Finalmente, tenemos el test con 0x, aqui hicimos la simulación de carga con Autocannon con una salvedad, tuve que bajar la carga (los request y la cantidad de conecciones) porque sino se rompia la aplicacion.
En este caso corri primero la ruta sin console.log y luego la otra.
Aqui tambien se evidencia, pero de forma mucho mas grafica, como se va cargando el stack de respuestas.

CONCLUSIÓN

A modo de resumen sobre lo trabajado en este desafio, creo poder entrever una posible progresión en el uso de las diferentes herramientas.
Esta progresión comenzaría por las herramientas de diagnostico general, como el --prof, artillery o autocannon, que me indican, a grandes rasgos, que estoy teniendo mayor demora o no en la resolución de solicitudes.
Luego podríamos pasar por 0x con su diagrama de flama, que si bien me da la posibilidad de comenzar a entrever cuales son los procesos que estan demorando al resto, no tiene una interface del todo amigable lo que implica que el trabajo de hurgar en lo que esta ocurriendo en nuestro codigo es bastante tedioso.
Y asi, finalmente podríamos usar la herramienta, que a mi forma de ver, mas en detalle nos deja ver como se esta comportando nuestro codigo, me refiero al comando --inspect de node en conjunto con el navegador Chrome, ya que este método nos permite ver linea por linea cuanto se esta demorando cada sentencia.