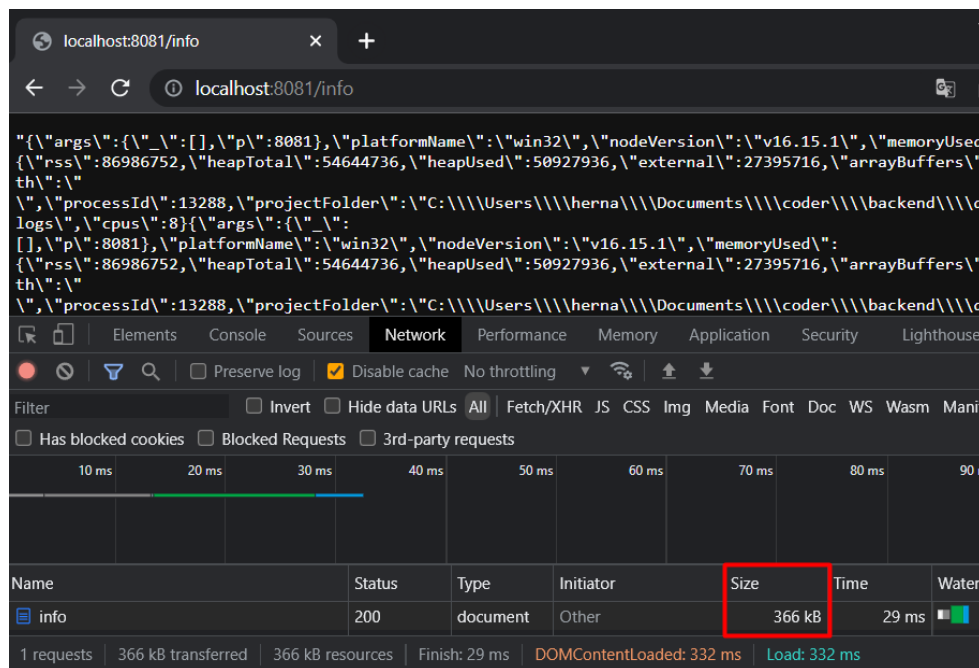
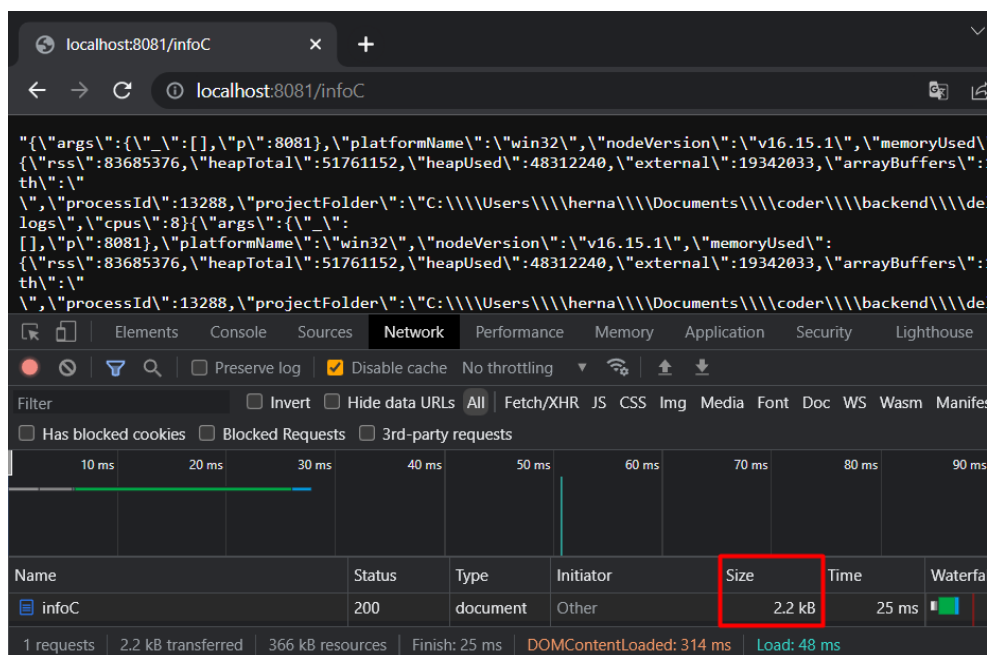


## Análisis de compresión:

- Uso de la ruta /info, que no utiliza o compression, y agregándole repeat():



- Uso de la ruta /infoC, que sí utiliza compression, y agregándole el mismo repeat():



- Conclusiones:

Por más que la compresión pueda resultar útil, es preciso analizar previamente el size de la respuesta original, ya que en casos donde la respuesta es poco pesada, la respuesta comprimida puede llegar a pesar más.

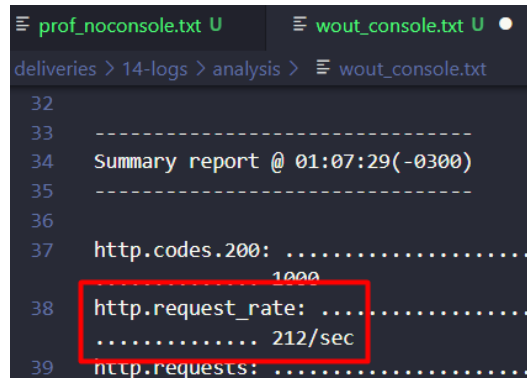
Los resultados interesantes y relevantes se ven cuando el peso de la respuesta original es significativo.

## Análisis de performance:

### Perfilamiento del servidor con Artillery:

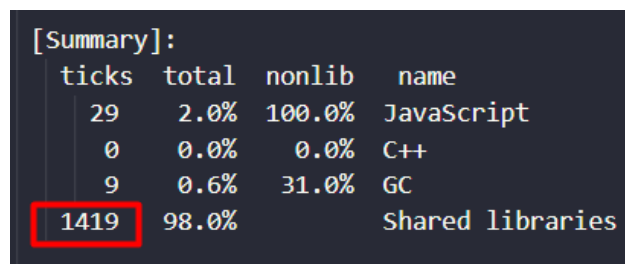
- Uso de la ruta /info sin console.log():

En vista del archivo generado por artillery, las requests por segundo llegan a 50/sec:



```
deliveries > 14-logs > analysis > wout_console.txt
32
33
34 Summary report @ 01:07:29(-0300)
35
36
37 http.codes.200: .....
38 http.request_rate: ..... 212/sec
39 http.requests: .....
```

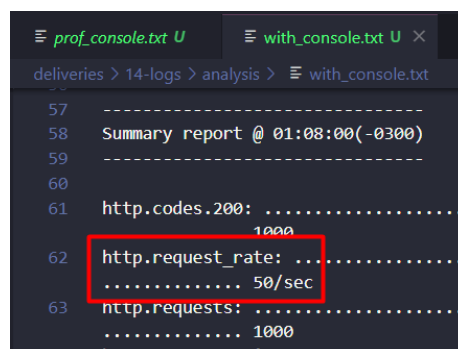
Y analizando el archivo generado por el built in profiler de node, los ticks llegan a 2124:



```
[Summary]:
  ticks total nonlib  name
    29   2.0% 100.0% JavaScript
     0   0.0%  0.0% C++
     9   0.6% 31.0% GC
  1419  98.0%      Shared libraries
```

- Uso de la ruta /info con console.log():

En vista del archivo generado por artillery, las requests por segundo llegan a 50/sec:



```
deliveries > 14-logs > analysis > with_console.txt
57
58 Summary report @ 01:08:00(-0300)
59
60
61 http.codes.200: .....
62 http.request_rate: ..... 50/sec
63 http.requests: .....
64 http.response_time: .....
```

Y analizando el archivo generado por el built in profiler de node, los ticks llegan a 2124:

| [Summary]: |       |        |                  |
|------------|-------|--------|------------------|
| ticks      | total | nonlib | name             |
| 28         | 1.3%  | 100.0% | JavaScript       |
| 0          | 0.0%  | 0.0%   | C++              |
| 8          | 0.4%  | 28.6%  | GC               |
| 2124       | 98.7% |        | Shared libraries |

### Perfilamiento del servidor en modo inspector:

- Uso de la ruta /info sin console.log():

The screenshot shows the Node.js Inspector interface. The 'Sources' panel is active, displaying the file 'server.js'. The code for the `app.get('/info', (req, res) => { ... })` endpoint is visible. The performance profiler is running, and the timeline shows the execution of the `res.json()` call, which took 19.8 ms. The timeline also shows the execution of the `console.log()` call, which took 0.4 ms. The profiler is showing the execution of the `res.json()` call, which took 19.8 ms. The profiler is showing the execution of the `res.json()` call, which took 19.8 ms.

- Uso de la ruta /info con console.log():

```

Node >> server.js x
207 })
208
209 app.get('/info', (req, res) => {
210   // logInfo(`URL: ${req.url} & METHOD: ${req.method}`)
211   console.log({
212     args: args,
213     platformName: process.platform,
214     nodeVersion: process.version,
215     memoryUsed: process.memoryUsage(),
216     execPath: process.title,
217     processId: process.pid,
218     projectFolder: process.cwd(),
219     cpus: os.cpus().length
220   })
221   res.json({
222     args: args,
223     platformName: process.platform,
224     nodeVersion: process.version,
225     memoryUsed: process.memoryUsage(),
226     execPath: process.title,
227     processId: process.pid,
228     projectFolder: process.cwd(),
229     cpus: os.cpus().length
230   })

```

Vemos cómo el console log agrega aproximadamente un 20% más del que le tomaría a la ruta enviar solamente la respuesta.

**Perfilamiento del servidor con 0x y Autocannon:**

- Uso de la ruta /info sin console.log():

```

Running all benchmarks in parallel ...
Running 20s test @ http://127.0.0.1:8080/info
100 connections

```

| Stat    | 2.5%   | 50%    | 97.5%  | 99%    | Avg       | Stdev    | Max    |
|---------|--------|--------|--------|--------|-----------|----------|--------|
| Latency | 209 ms | 241 ms | 416 ms | 432 ms | 266.48 ms | 56.74 ms | 490 ms |

| Stat      | 1%     | 2.5%   | 50%    | 97.5%  | Avg    | Stdev   | Min    |
|-----------|--------|--------|--------|--------|--------|---------|--------|
| Req/Sec   | 200    | 200    | 400    | 484    | 374.2  | 79.44   | 200    |
| Bytes/Sec | 114 kB | 114 kB | 229 kB | 278 kB | 215 kB | 45.7 kB | 114 kB |

Puede verse como el avg de latencia y requests por segundo son 266,48 ms y 374,2 respectivamente

- Uso de la ruta /info con console.log():

8k requests in 20.12s, 4.29 MB read  
Running 20s test @ http://127.0.0.1:8080/infoConsole  
100 connections

| Stat    | 2.5%   | 50%    | 97.5%  | 99%    | Avg       | Stdev    | Max    |
|---------|--------|--------|--------|--------|-----------|----------|--------|
| Latency | 209 ms | 240 ms | 410 ms | 523 ms | 267.27 ms | 61.87 ms | 605 ms |

| Stat      | 1%     | 2.5%   | 50%    | 97.5%  | Avg    | Stdev   | Min    |
|-----------|--------|--------|--------|--------|--------|---------|--------|
| Req/Sec   | 200    | 200    | 400    | 454    | 370    | 76.12   | 200    |
| Bytes/Sec | 114 kB | 114 kB | 230 kB | 261 kB | 212 kB | 43.8 kB | 114 kB |

Puede verse como el avg de latencia y requests por segundo son 267,27 ms y 370 respectivamente, ambos valores menos “veloces” que sin el console.log()

### **Conclusiones::**

Los resultados resultan lógicos y esperables.

El test de carga con el console log tiene más ticks (latencia) y menos respuestas por segundo a causa del proceso bloqueante que está agregando.

Esto puede verse también en el uso del modo inspector, donde se ve la carga relevante que agrega tener un console.log() respecto al envío de una respuesta sin este.

Mismo con el diagrama de flama y las stats generadas por 0x, el proceso bloqueante muestra partes más largas y duraderas que interrumpen el resto de la performance y no permiten al servidor trabajar más rápido.