# Desafío Clase 32: Loggers, Gzip y Análisis de performance

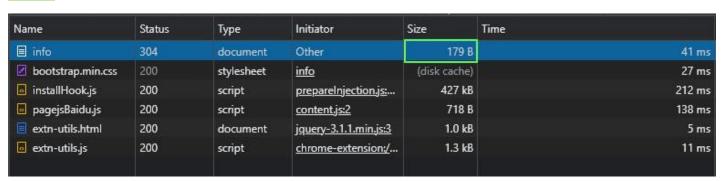
### **Gzip**

Diferencia de cantidad de bytes:

Con compresión:

Status	Туре	Initiator	Size	Time
304	document	Other	179 B	38 ms
200	stylesheet	<u>info</u>	(disk cache)	62 ms
200	script	prepareInjection.js:525	427 kB	281 ms
200	script	content.js:2	718 B	205 ms
200	document	jquery-3.1.1.min.js:3	1.0 kB	4 ms
200	script	chrome-extension://e	1.3 kB	5 ms
	304 200 200 200 200	304 document 200 stylesheet 200 script 200 script 200 document	304         document         Other           200         stylesheet         info           200         script         prepareInjection.js:525           200         script         content.js:2           200         document         jquery-3.1.1.min.js:3	304         document         Other         179 B           200         stylesheet         info         (disk cache)           200         script         prepareInjection.js:525         427 kB           200         script         content.js:2         718 B           200         document         jquery-3.1.1.min.js:3         1.0 kB

Sin compresión:



**Conclusiones:** no hay diferencia en cantidad de bytes, pero si en el tiempo de respuesta que es menor cuando se usa la ruta con compresión.

### Análisis de performance

#### Procesamiento de resultados de test --prof con --prof-process

Ruta '/info' sin console.log:

```
[Summary]:
ticks total nonlib name
13 0.0% 100.0% JavaScript
0 0.0% 0.0% C++
16 0.1% 114.3% GC
30320 100.0% Shared libraries
```

- Ruta '/info' con console.log:

```
[Summary]:
ticks total nonlib name
14 0.0% 92.9% JavaScript
0 0.0% 0.0% C++
20 0.1% 142.9% GC
38866 100.0% Shared libraries
```

**Conclusiones:** si bien la diferencia no es tan grande, se ve en el nro de ticks en la prueba de la ruta con y sin console.

### Resultados de test de carga con Artillery:

- Ruta '/info' sin console.log:

#### - Ruta '/info' con console.log:

**Conclusiones:** la diferencia en el tiempo de respuesta en la prueba de la ruta con y sin console es notable, siendo mayor en el segundo caso, tanto el <u>tiempo</u> mínimo, máximo como el promedio.

### Resultados de test de carga con Autocannon:

#### - Ruta '/info' sin console.log:

Running 20s test @ http://localhost:8080/info 100 connections Stat 2.5% 50% 97.5% 99% Stdev Avg Max Latency 22 ms 1026 ms 1670 ms 1837 ms 987.68 ms 396.02 ms 2061 ms 2.5% 50% 97.5% Stat 1% Avg Stdev Min Req/Sec 46 46 94 158 99.05 30.35 46 111 kB 111 kB 228 kB 73.5 kB 111 kB Bytes/Sec 383 kB 240 kB

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

2k requests in 20.13s, 4.8 MB read

### - Ruta '/info' con console.log:

2k requests in 20.13s, 4.8 MB read Running 20s test @ http://localhost:8080/info-console 100 connections

Stat	2.5%	50%	97.5%	99%	Avg	Stdev	Max
Latency	630 ms	1135 ms	3551 ms	3775 ms	1231.36 ms	598.12 ms	3867 ms

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	0	0	64	155	77.85	38.83	13
Bytes/Sec	0 B	0 B	154 kB	372 kB	187 kB	93.2 kB	31.2 kB

Req/Bytes counts sampled once per second.

# of samples: 20

2k requests in 20.5s, 3.74 MB read

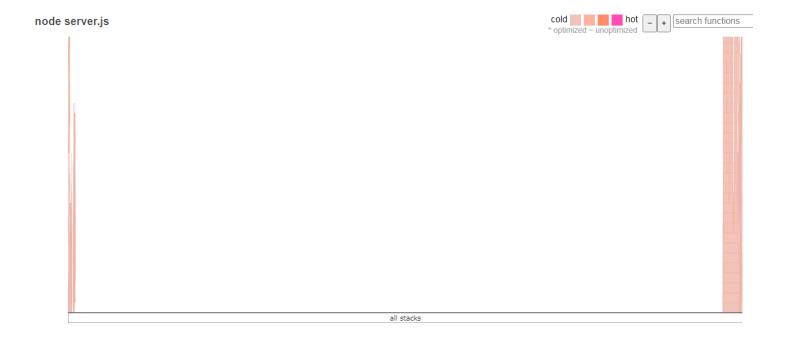
**Conclusiones:** el tiempo de latencia es mayor en la ruta con console.log, la respuesta es más rápida en el caso de la ruta sin console.log.

#### Perfilamiento del servidor con el modo inspector de node.js

```
const memoria = process.memoryUsage();
13
14
              const pathExe = process.execPath;
15
              const processId = process.pid;
              const carpeta = process.cwd();
16
17
              const procesadores = cpu.length;
18
              infoRouter.get("/info-console", (req, res) => {
19
                console.log(`Argumentos: ${argumentos}
20
      12.0 ms
       0.1 ms
                             Plataforma: ${plataforma}
21
                             Version: ${version}
22
23
                             Memoria: ${memoria}
       0.8 ms
                             PathExe: ${pathExe}
24
25
       0.3 ms
                             IdProcesador: ${processId}
                             Carpeta: ${carpeta}
26
                             Procesadores: ${procesadores}`);
27
       2.2 ms
28
       1.3 ms
                res.render(path.join(process.cwd(), "/views/pages/info.ejs"), {
29
                  argumentos: argumentos,
30
                  plataforma: plataforma,
                  version: version,
31
       0.1 ms
       0.3 ms
                  memoria: memoria.rss,
32
                  pathExe: pathExe,
33
34
                  processId: processId,
35
                  carpeta: carpeta,
                  procesadores: procesadores,
36
       0.4 ms
37
                });
38
             });
39
              infoRouter.get("/info", compression(), (req, res) => {
40
                res.render(path.join(process.cwd(), "/views/pages/info.ejs"), {
       6.9 ms
41
42
                  argumentos: argumentos,
43
                  plataforma: plataforma,
                  version: version,
44
       0.8 ms
                  memoria: memoria.rss,
45
46
       0.1 ms
                  pathExe: pathExe,
                  processId: processId,
47
48
       0.1 ms
                  carpeta: carpeta,
                  procesadores: procesadores,
49
       0.4 ms
50
                });
51
              });
52
              export default infoRouter;
53
Lines 40 columns 40
```

**Conclusiones:** En el caso de la ruta info con console.log, podemos ver que tiene el proceso menos perfomante con tiempo de carga de 12ms; en el caso de la ruta sin console.log, cuenta con un proceso máximo de 6.9ms, prácticamente la mitad que el anterior.

## Diagrama de Flama:



**Conclusiones:** El lado izquierdo representa la ruta info sin console.log mientras en el derecho se realiza mayor procesamiento (por la ruta info con el console.log con los datos) y por lo tanto el gráfico es más ancho.

#### **Conclusiones Finales**

Los distintos test demuestran que la ruta info con console.log consume mayor tiempo de procesamiento que cuando se analiza la ruta sin console.log, de todos modos la diferencia es mínima al tratarse de test a nivel local y que el servidor se ejecuta en los todos los casos en modo fork.