

# INFORME DESAFIO LOGGERS Y PROFILING

## LAGORIO FRANCISCO

- 1- PUNTO DONDE PIDE ANALIZAR CON ARTILLERY AGREGANDO Y SACANDO UN `console.log` DEL ROUTER DE `/INFO`

```
server.js U  infoRouter.js U  package.json U  benchmark.js U
nginx-1.22.0 > desafioCoder > public > routes > infoRouter.js > ...
1  const { Router } = require("express");
2  const parseArgs = require("minimist");
3  const os = require("os");
4
5  const infoRouter = Router();
6
7  infoRouter.get("/info", (req, res) => {
8    try {
9      const args = parseArgs(process.argv.slice(2));
10     const info = {
11       argumentos: JSON.stringify(args),
12       directorioActual: process.cwd(),
13       idProceso: process.pid,
14       vNode: process.version,
15       rutaEjecutable: process.execPath,
16       sistemaOperativo: process.platform,
17       memoria: JSON.stringify(process.memoryUsage().rss, null, 2),
18       processNum: os.cpus().length,
19     };
20     console.log(info);
21     res.render("pages/info", info);
22   } catch (error) {
23     res.render(error.message);
24   }
25 });
```

result_prof-clg.txt U					result_prof-clg.txt U				
Statistical profiling result from sin-clg-v8.log, (11658 ticks, 5					Statistical profiling result from clg-v8.log, (18705 ticks, 0 unad				
[Shared libraries]:					[Shared libraries]:				
ticks	total	nonlib	name		ticks	total	nonlib	name	
8988	77.1%		C:\Windows\SYSTEM32\ntdll.dll		14847	79.4%		C:\Windows\SYSTEM32\ntdll.dll	
2528	21.7%		C:\Program Files\nodejs\node.exe		3714	19.9%		C:\Program Files\nodejs\node.exe	
7	0.1%		C:\Windows\System32\KERNEL32.DLL		15	0.1%		C:\Windows\System32\KERNELBASE.dll	
4	0.0%		C:\Windows\System32\KERNELBASE.dll		3	0.0%		C:\Windows\System32\KERNEL32.DLL	
[JavaScript]:					[JavaScript]:				
ticks	total	nonlib	name		ticks	total	nonlib	name	
48	0.4%	36.6%	LazyCompile: *resolve node:path:158:10		36	0.2%	28.6%	LazyCompile: *resolve node:path:158:10	
22	0.2%	16.8%	RegExp: [ \t]*<%_		11	0.1%	8.7%	RegExp: [ \t]*<%_	
3	0.0%	2.3%	LazyCompile: *scanLine C:\Users\FranLagor		6	0.0%	4.8%	LazyCompile: *toNamespacedPath node:path:	
3	0.0%	2.3%	LazyCompile: *normalizeString node:path:		3	0.0%	2.4%	LazyCompile: *next C:\Users\FranLagorio\U	
2	0.0%	1.5%	LazyCompile: *toNamespacedPath node:path:		3	0.0%	2.4%	Function: *realpathSync node:fs:2425:22	
2	0.0%	1.5%	LazyCompile: *nextPart node:fs:2401:31		2	0.0%	1.6%	RegExp: ^[\^_a-zA-Z\-\0-9!#\$%&'**.[~]+\$	
2	0.0%	1.5%	LazyCompile: *Module._findPath node:inter		2	0.0%	1.6%	RegExp: (?<=\\n	
2	0.0%	1.5%	Function: *validateString node:internal/v		2	0.0%	1.6%	LazyCompile: *readPackageScope node:inter	
2	0.0%	1.5%	Function: *realpathSync node:fs:2425:22		2	0.0%	1.6%	LazyCompile: *createError C:\Users\FranLa	
1	0.0%	0.8%	RegExp: _>[ \t]*		2	0.0%	1.6%	LazyCompile: *compileFunction node:vm:308	
1	0.0%	0.8%	RegExp: [^\t\x20-\x7e\x80-\xff]		2	0.0%	1.6%	LazyCompile: *Module._load node:internal	
1	0.0%	0.8%	LazyCompile: *syncExports node:internal/b		2	0.0%	1.6%	Function: *send C:\Users\FranLagorio\Desk	
1	0.0%	0.8%	LazyCompile: *readPackage node:internal/m		2	0.0%	1.6%	Function: *resOnFinish node:http_server	
1	0.0%	0.8%	LazyCompile: *readFileSync node:fs:450:22		2	0.0%	1.6%	Function: *alignPool node:buffer:159:19	
1	0.0%	0.8%	LazyCompile: *normalize node:path:304:12		2	0.0%	1.6%	Function: *_write node:internal/streams/v	
1	0.0%	0.8%	LazyCompile: *next C:\Users\FranLagorio\U		27	1.0%	0.8%	RegExp: ^([!#\$%&'**._\^~0-9A-Za-z-~]+)\$	
1	0.0%	0.8%	LazyCompile: *emit node:events:475:44		28	1.0%	0.8%	RegExp: ^((?:@[^\s\\\/\?[\^.\ \\%] /\\%	
1	0.0%	0.8%	LazyCompile: *dirname node:path:653:10		29	1.0%	0.8%	RegExp: [x00-x1f\x27\x5c\x7f-x9f][\u	
1	0.0%	0.8%	LazyCompile: *basename node:path:749:11		30	1.0%	0.8%	RegExp: [8<="	

Como se observa los ticks son mucho mayores en el caso de los `console.log`.

Según lo que encontré un tick es:

```
Statistical profiling result from isolate-0x10295c000-41373-v8.log, (761 ticks, 40 unaccounted, 0 excluded).
```

La primera línea dice que la aplicación ha usado 761 tics para ejecutar la aplicación. Un tic es como un ciclo de reloj utilizado por un proceso de nodo. Entonces, en teoría, la aplicación tardó 761 ciclos de reloj en ejecutarse. También puede encontrar una sección de resumen que desglosa el código JavaScript vs C++.

## Autocannon SIN console log

En este caso console.log esta comentado

```
1 const { Router } = require("express");
2 const parseArgs = require("minimist");
3 const os = require("os");
4
5 const infoRouter = Router();
6
7 infoRouter.get("/info", (req, res) => {
8   try {
9     const args = parseArgs(process.argv.slice(2));
10    const info = {
11      argumentos: JSON.stringify(args),
12      directorioActual: process.cwd(),
13      idProceso: process.pid,
14      vNode: process.version,
15      rutaEjecutable: process.execPath,
16      sistemaOperativo: process.platform,
17      memoria: JSON.stringify(process.memoryUsage().rss, null, 2),
18      processNum: os.cpus().length,
19    };
20    // console.log(info);
21    res.render("pages/info", info);
22  } catch (error) {
23    res.render(error.message);
24  }
25 });
26
27 module.exports = infoRouter;
```

## Autocannon CON console.log

```
1 const { Router } = require("express");
2 const parseArgs = require("minimist");
3 const os = require("os");
4
5 const infoRouter = Router();
6
7 infoRouter.get("/info", (req, res) => {
8   try {
9     const args = parseArgs(process.argv.slice(2));
10    const info = {
11      argumentos: JSON.stringify(args),
12      directorioActual: process.cwd(),
13      idProceso: process.pid,
14      vNode: process.version,
15      rutaEjecutable: process.execPath,
16      sistemaOperativo: process.platform,
17      memoria: JSON.stringify(process.memoryUsage().rss, null, 2),
18      processNum: os.cpus().length,
19    };
20    console.log(info);
21    res.render("pages/info", info);
22  } catch (error) {
23    res.render(error.message);
24  }
25 });
26
27 module.exports = infoRouter;
```

## Conclusión:

Como se observa el console.log hace que el render tarde 4 ms más.

## Autocannon SIN console log

```
C:\Windows\System32\cmd.exe
npm WARN config global '--global', '--local' are deprecated. Use '--location=global' instead.
> socket-starter-repo@1.0.0 test
> node benchmark.js

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



| Stat    | 2.5%    | 50%     | 97.5%   | 99%     | Avg        | Stdev     | Max     |
|---------|---------|---------|---------|---------|------------|-----------|---------|
| Latency | 1112 ms | 1251 ms | 2093 ms | 2220 ms | 1343.28 ms | 256.01 ms | 2449 ms |



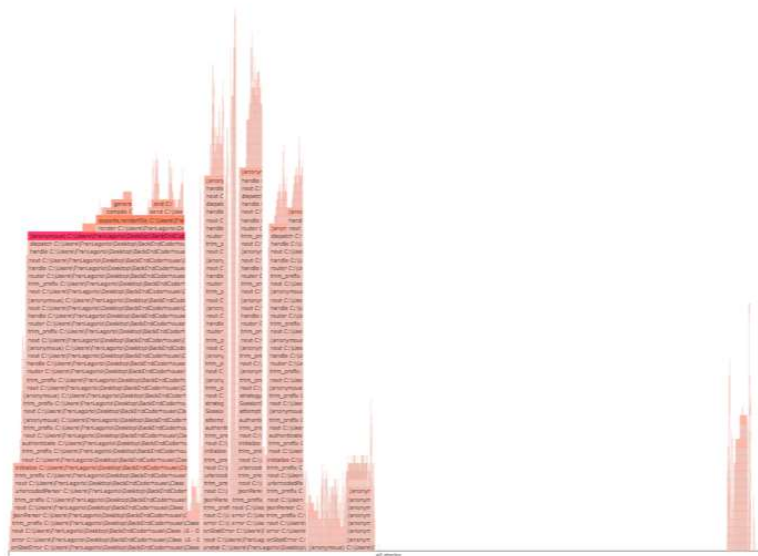
| Stat      | 1%  | 2.5% | 50%     | 97.5%   | Avg     | Stdev  | Min    |
|-----------|-----|------|---------|---------|---------|--------|--------|
| Req/Sec   | 0   | 0    | 990     | 1071    | 874.6   | 285.18 | 222    |
| Bytes/Sec | 0 B | 0 B  | 2.75 MB | 2.98 MB | 2.43 MB | 793 kB | 617 kB |



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

19k requests in 20.22s, 48.6 MB read
1k errors (0 timeouts)

C:\Users\FranLagorio\Desktop\BackEndCoderhouse\Clase 16 - Desafio Logs\nginx-1.22.0\desafioCoder\public>
```



## Autocannon CON console.log

```
C:\Windows\System32\cmd.exe
npm WARN config global '--global', '--local' are deprecated. Use '--location=global' instead.
> socket-starter-repo@1.0.0 test
> node benchmark.js

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



| Stat    | 2.5%    | 50%     | 97.5%   | 99%     | Avg        | Stdev     | Max     |
|---------|---------|---------|---------|---------|------------|-----------|---------|
| Latency | 1318 ms | 1438 ms | 2410 ms | 2488 ms | 1586.13 ms | 326.14 ms | 2820 ms |



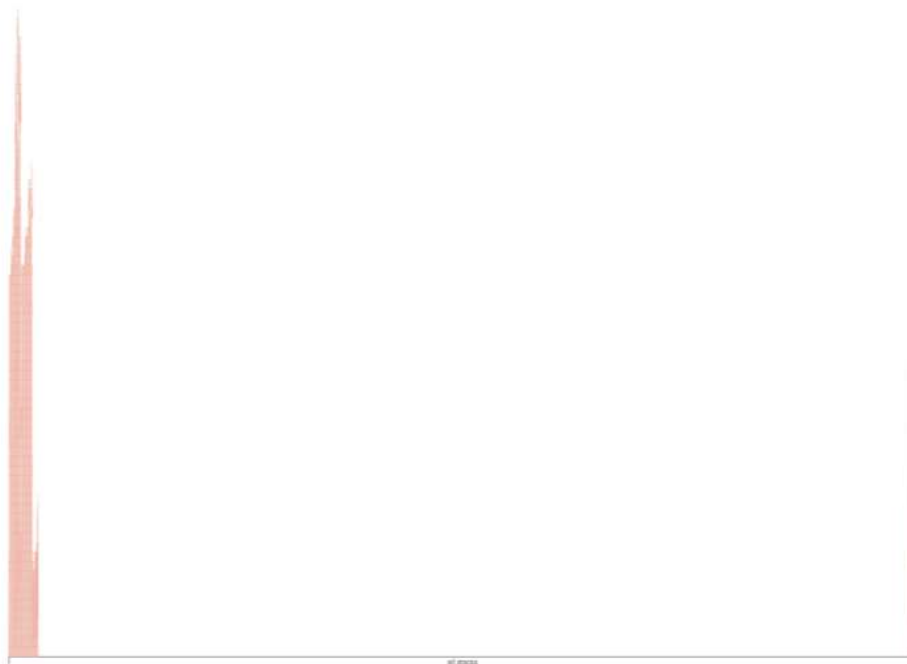
| Stat      | 1%  | 2.5% | 50%     | 97.5%   | Avg     | Stdev  | Min    |
|-----------|-----|------|---------|---------|---------|--------|--------|
| Req/Sec   | 0   | 0    | 874     | 988     | 743.45  | 295.78 | 350    |
| Bytes/Sec | 0 B | 0 B  | 2.43 MB | 2.75 MB | 2.07 MB | 822 kB | 973 kB |



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

16k requests in 20.39s, 41.3 MB read
1k errors (0 timeouts)

C:\Users\FranLagorio\Desktop\BackEndCoderhouse\Clase 16 - Desafio Logs\nginx-1.22.0\desafioCoder\public>
```



## Conclusión:

**El promedio de latencia es mayor con el console.log.**