

# Informe de desafio de logs , y profiling

Tomas Juarez

sobre la ruta '/info', en modo fork, agregando ó extrayendo un console.log se hace un perfilamiento del servidor con --prof de node.js y procesando los datos con --prof-process

```
286 }
287 })
288
289 app.get('/info',(req,res)=>{
290   const data = {
291     os: os ,
292     nodeVersion: nodeV,
293     path: paTh,
294     processId: processId,
295     folderPath: folderPath,
296     maxRSS: maxRSS,
297     procesos:numOfProcess,
298     puerto:PORT
299   }
300 }
301 console.log(data)
302 res.send(data)
303 })
```

er Ir Ejecutar Terminal Ayuda result\_prof-no-clg.txt - desafio de sockets - Visual Studio Code

JS server.js • result\_prof-clg.txt result\_prof-no-clg.txt X JS benchmark.js

desafioCoder > public > result\_prof-no-clg.txt

1 Statistical profiling result from no-clg-v8.log, (76913 ticks, 6 unaccounted, 0 excluded).

2

3 [Shared libraries]:

4 ticks total nonlib name

5 73538 95.6% C:\Windows\SYSTEM32\ntdll.dll

6 3123 4.1% C:\Program Files\nodejs\node.exe

7 9 0.0% C:\Windows\System32\KERNELBASE.dll

8 4 0.0% C:\Windows\system32\mswsock.dll

9 2 0.0% C:\Windows\System32\WS2\_32.dll

10 2 0.0% C:\Windows\System32\KERNEL32.DLL

11

12 [JavaScript]:

13 ticks total nonlib name

14 21 0.0% 8.9% LazyCompile: \*resolve node:path:158:10

15 19 0.0% 8.1% LazyCompile: \*serializeInto C:\Users\Usuario\Desktop\Back-end-test\desafio de sockets\desafioCoder

16 8 0.0% 3.4% LazyCompile: \*emit node:events:340:44

17 5 0.0% 2.1% LazyCompile: \*nextTick node:internal/process/task\_queues:104:18

18 4 0.0% 1.7% LazyCompile: \*next C:\Users\Usuario\Desktop\Back-end-test\desafio de sockets\desafioCoder\public\n

19 4 0.0% 1.7% LazyCompile: \*deserializeObject C:\Users\Usuario\Desktop\Back-end-test\desafio de sockets\desafioC

20 3 0.0% 1.3% LazyCompile: \*get node:internal/streams/duplex:94:8

21 3 0.0% 1.3% LazyCompile: \*Readable.read node:internal/streams/readable:394:35

22 3 0.0% 1.3% Function: \*write C:\Users\Usuario\Desktop\Back-end-test\desafio de sockets\desafioCoder\public\nod

23 2 0.0% 0.9% RegExp: ^((?:[0-9a-fA-F]{1,4}){7})(?:[0-9a-fA-F]{1,4})|((?:[0-9a-fA-F]{1,4}){6})(?:((

24 2 0.0% 0.9% RegExp: [^\t\x20-\x7e\x80-\xff]

25 2 0.0% 0.9% LazyCompile: \*writeOrBuffer node:internal/streams/writable:365:23

26 2 0.0% 0.9% LazyCompile: \*serializeObject C:\Users\Usuario\Desktop\Back-end-test\desafio de sockets\desafioCod

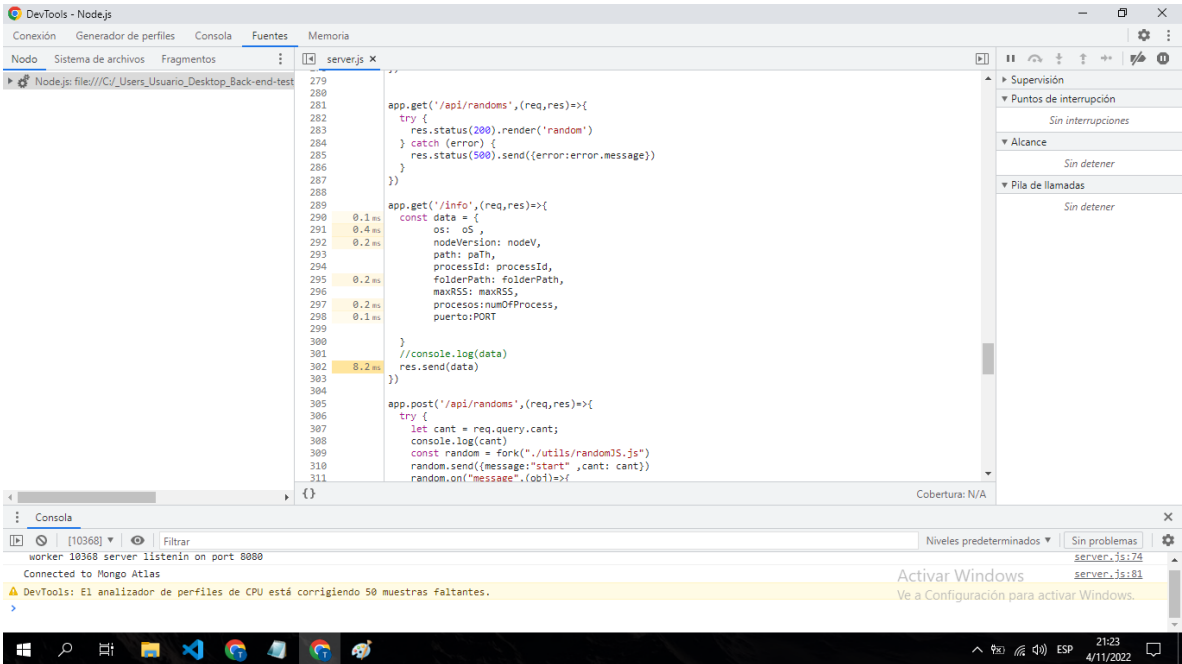
27 2 0.0% 0.9% LazyCompile: \*isUint8Array node:internal/util/types:13:22

28 2 0.0% 0.9% LazyCompile: \*getEncodingOps node:buffer:678:24

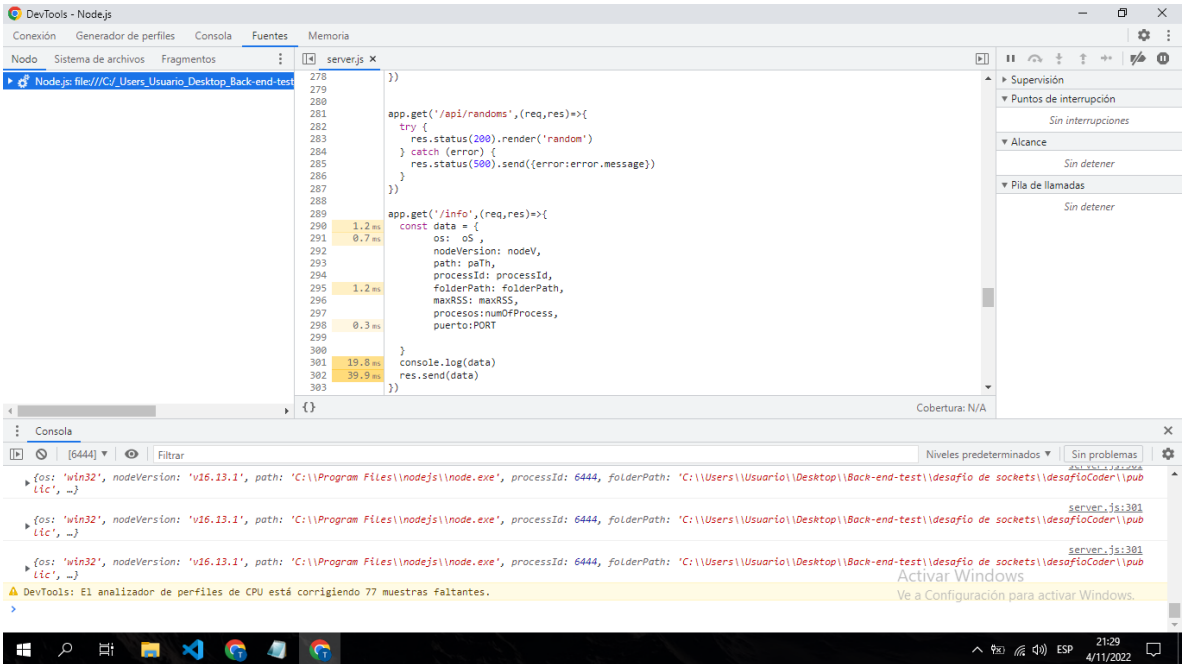
29 2 0.0% 0.9% LazyCompile: \*Module.findPath node:internal/modules/cjs/loader:404:28

Como se puede ver en los resultados de los test (con console.log() en la imagen izquierda y sin console.log en la imagen de la derecha ), en el test sin console.log() la cantidad de ticks disminuyo considerablemente .

## Inspect y Autocannon con console.log()



## Inspect y autocannon sin console.log()



## Conclusión:

En el caso de inspect y autocannon vemos también que en el caso en el que el console.log() esta comentado se reduce el tiempo de renderizado unos 31,7 ms , en conclusión el uso de console.log() genera un perdida considerable de performance ya que consume demasiados recursos.