

# Perfilamiento del Servidor con prof process y Artillery

Se observa que la ruta no bloqueante tiene la mitad en mediana de respuesta que la bloqueante

```
node > result_consolelog.txt
30
31 All VUs finished. Total time: 10 seconds
32
33 Summary report @ 08:18:00(-0300)
34
35
36 http_codes.200: ..... 1000
37 http.request_rate: ..... 63/sec
38 http.requests: ..... 1000
39
40 http_response_time:
41 min: ..... 8
42 max: ..... 357
43 median: ..... 247.2
44 p95: ..... 327.1
45 p99: ..... 347.3
46
47 http.responses: ..... 1000
48
49 vusers.completed: ..... 50
50 vusers.created: ..... 50
51 vusers.created_by_name.0: ..... 50
52 vusers.failed: ..... 0
53 vusers.session_length:
54 min: ..... 6343
55 max: ..... 7478
56 median: ..... 7407.5
57 p95: ..... 7407.5
58 p99: ..... 7407.5

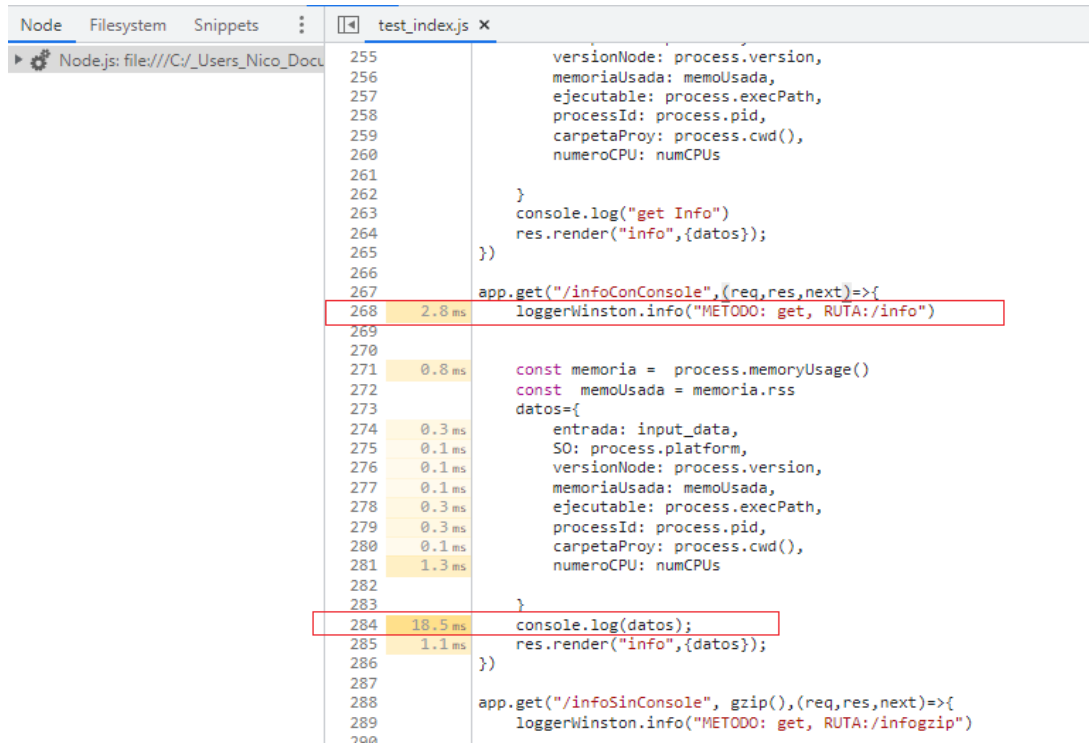
node > result_Sinconsolelog.txt
49 All VUs finished. Total time: 6 seconds
50
51 Summary report @ 08:18:55(-0300)
52
53
54 http_codes.200: ..... 1000
55 http.request_rate: ..... 58/sec
56 http.requests: ..... 1000
57
58 http_response_time:
59 min: ..... 5
60 max: ..... 249
61 median: ..... 144
62 p95: ..... 198.4
63 p99: ..... 228.2
64
65 http.responses: ..... 1000
66
67 vusers.completed: ..... 50
68 vusers.created: ..... 50
69 vusers.created_by_name.0: ..... 50
70 vusers.failed: ..... 0
71 vusers.session_length:
72 min: ..... 2524
73 max: ..... 3796.1
74 median: ..... 3678.4
75 p95: ..... 3752.7
76 p99: ..... 3828.5
```

```
node > result_bloq_isolog
25 1 0.0% 4.8% LazyCompile: *deserializeObject C:\Users\Wico\Document
26 1 0.0% 4.8% LazyCompile: *command C:\Users\Wico\Documents\Curso Fu
27 1 0.0% 4.8% LazyCompile: *anonymous internal/per_context/primord
28
29 [C++]:
30 ticks total nonlib name
31
32 [Summary]:
33 ticks total nonlib name
34 20 0.4% 95.2% JavaScript
35 0 0.0% 0.0% C++
36 17 0.3% 81.0% GC
37 5213 99.6% Shared libraries
38 1 0.0% Unaccounted
39
40 [C++ entry points]:
41 ticks cpp total name
42
43 [Bottom up (heavy) profile]:
44 Note: percentage shows a share of a particular caller in the total
45 amount of its parent calls.
46 Callers occupying less than 1.0% are not shown.
47
48 ticks parent name
49 4742 90.6% C:\Windows\SYSTEM32\ntdll.dll
50 92 1.9% LazyCompile: *writeOrBuffer internal/streams/writable.js:334
51 89 96.7% LazyComile: *Writable.write internal/streams/writable.is:

node > result_noBloq_isolog
22 1 0.0% 6.3% LazyCompile: *isPathSeparator path.js:38:25
23 1 0.0% 6.3% LazyCompile: *extname path.js:796:10
24 1 0.0% 6.3% LazyCompile: *dirname path.js:618:10
25 1 0.0% 6.3% LazyCompile: *deserializeObject C:\Users\Wico\Document
26 1 0.0% 6.3% LazyCompile: *append internal/linkedList.js:29:16
27
28 [C++]:
29 ticks total nonlib name
30
31 [Summary]:
32 ticks total nonlib name
33 16 0.6% 100.0% JavaScript
34 0 0.0% 0.0% C++
35 11 0.4% 68.8% GC
36 2702 99.4% Shared libraries
37
38 [C++ entry points]:
39 ticks cpp total name
40
41 [Bottom up (heavy) profile]:
42 Note: percentage shows a share of a particular caller in the total
43 amount of its parent calls.
44 Callers occupying less than 1.0% are not shown.
45
46 ticks parent name
47 2314 85.1% C:\Windows\SYSTEM32\ntdll.dll
48
```

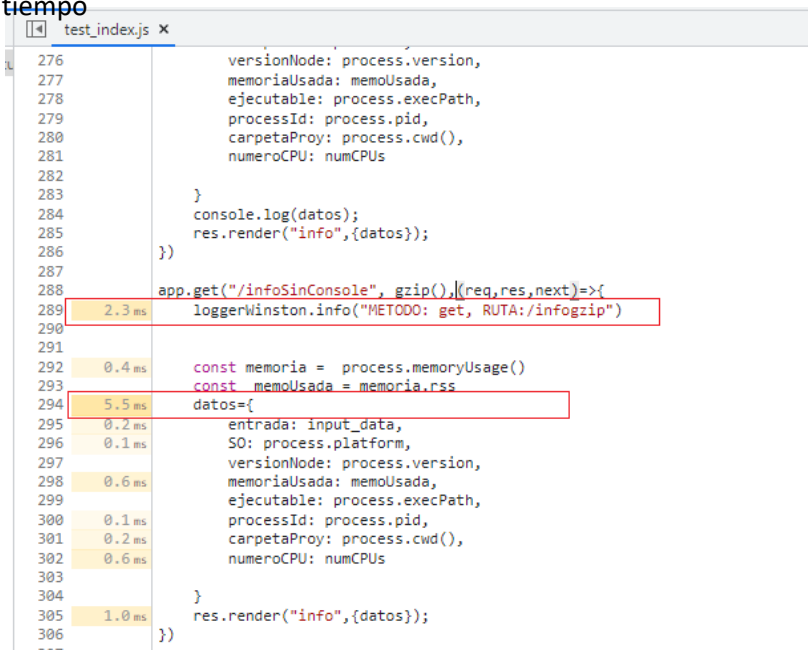
# Perfilamiento con el inspetor de node

**Bloqueante:** se observa que console.log es el que mas tiempo consume



```
Node.js: file:///C:/_Users_Nico_Docu... test_index.js x
255     versionNode: process.version,
256     memoriaUsada: memoUsada,
257     ejecutable: process.execPath,
258     processId: process.pid,
259     carpetaProy: process.cwd(),
260     numeroCPU: numCPUs
261
262   }
263   console.log("get Info")
264   res.render("info",{datos});
265 })
266
267 app.get("/infoConConsole",(req,res,next)=>{
268   loggerWinston.info("METODO: get, RUTA:/info")
269
270   const memoria = process.memoryUsage()
271   const memoUsada = memoria.rss
272   datos={
273     entrada: input_data,
274     SO: process.platform,
275     versionNode: process.version,
276     memoriaUsada: memoUsada,
277     ejecutable: process.execPath,
278     processId: process.pid,
279     carpetaProy: process.cwd(),
280     numeroCPU: numCPUs
281   }
282   console.log(datos);
283   res.render("info",{datos});
284 })
285
286 app.get("/infoSinConsole", gzip(),(req,res,next)=>{
287   loggerWinston.info("METODO: get, RUTA:/infogzip")
288
289   const memoria = process.memoryUsage()
290   const memoUsada = memoria.rss
291   datos={
292     entrada: input_data,
293     SO: process.platform,
294     versionNode: process.version,
295     memoriaUsada: memoUsada,
296     ejecutable: process.execPath,
297     processId: process.pid,
298     carpetaProy: process.cwd(),
299     numeroCPU: numCPUs
300   }
301   res.render("info",{datos});
302 })
```

**No bloqueante:** se observa que console.log no está pero el armado del objeto es el que consume mas tiempo



```
test_index.js x
276     versionNode: process.version,
277     memoriaUsada: memoUsada,
278     ejecutable: process.execPath,
279     processId: process.pid,
280     carpetaProy: process.cwd(),
281     numeroCPU: numCPUs
282
283   }
284   console.log(datos);
285   res.render("info",{datos});
286 })
287
288 app.get("/infoSinConsole", gzip(),(req,res,next)=>{
289   loggerWinston.info("METODO: get, RUTA:/infogzip")
290
291   const memoria = process.memoryUsage()
292   const memoUsada = memoria.rss
293   datos={
294     entrada: input_data,
295     SO: process.platform,
296     versionNode: process.version,
297     memoriaUsada: memoUsada,
298     ejecutable: process.execPath,
299     processId: process.pid,
300     carpetaProy: process.cwd(),
301     numeroCPU: numCPUs
302   }
303   res.render("info",{datos});
304 })
```

## Grafico de Flama con autocannon

