

## Análisis completo de performance

### 1) Artillery:

artillery quick --count 50 --num 20 http://localhost:8080/info --output info.txt

#### Sin console.log(data):

```
11 routers > . info.js > ...
12 router.get("/", /* compression(), */ async (req,res)=>{
13   try{
14     const data = {
15       argumentosDeEntrada: process.argv.slice(2),
16       nombreDeLaPlataforma: process.platform,
17       versionDeNode: process.version,
18       memoriaTotalReservada: process.memoryUsage(),
19       pathDeEjecucion: process.execPath,
20       processId: process.pid,
21       carpetaDelProyecto: process.cwd(),
22       cantidadDeProcesadores: cpu.length
23     }
24     // console.log(data)
25     res.send(data)
26   } catch(err) {
27     res.status(404).send(err)
28   }
29 }
30
31 export { router }
```

#### Output:

```
1 infoSinConsoleLog.txt
2 {
3   "aggregate": {
4     "counters": {
5       "vusers.created_by_name.0": 50,
6       "vusers.created": 50,
7       "http.requests": 1000,
8       "http.codes.200": 1000,
9       "http.responses": 1000,
10      "vusers.failed": 0,
11      "vusers.completed": 50
12    },
13    "rates": {
14      "http.request_rate": 381
15    },
16    "firstCounterAt": 1672184643623,
17    "firstHistogramAt": 1672184643642,
18    "lastCounterAt": 1672184646287,
19    "lastHistogramAt": 1672184646287,
20    "firstMetricAt": 1672184643623,
21    "lastMetricAt": 1672184646287,
22    "period": 1672184640000,
23    "summaries": {
24      "http.response_time": {
25        "min": 1,
26        "max": 246,
27        "count": 1000,
28        "p50": 80.6,
29        "median": 80.6,
30        "p75": 106.7,
31        "p90": 127.8,
32        "p95": 144,
33        "p99": 159.2,
34        "p999": 223.7
35      }
36    }
37  }
38 }
```

#### Info con console.log(data):

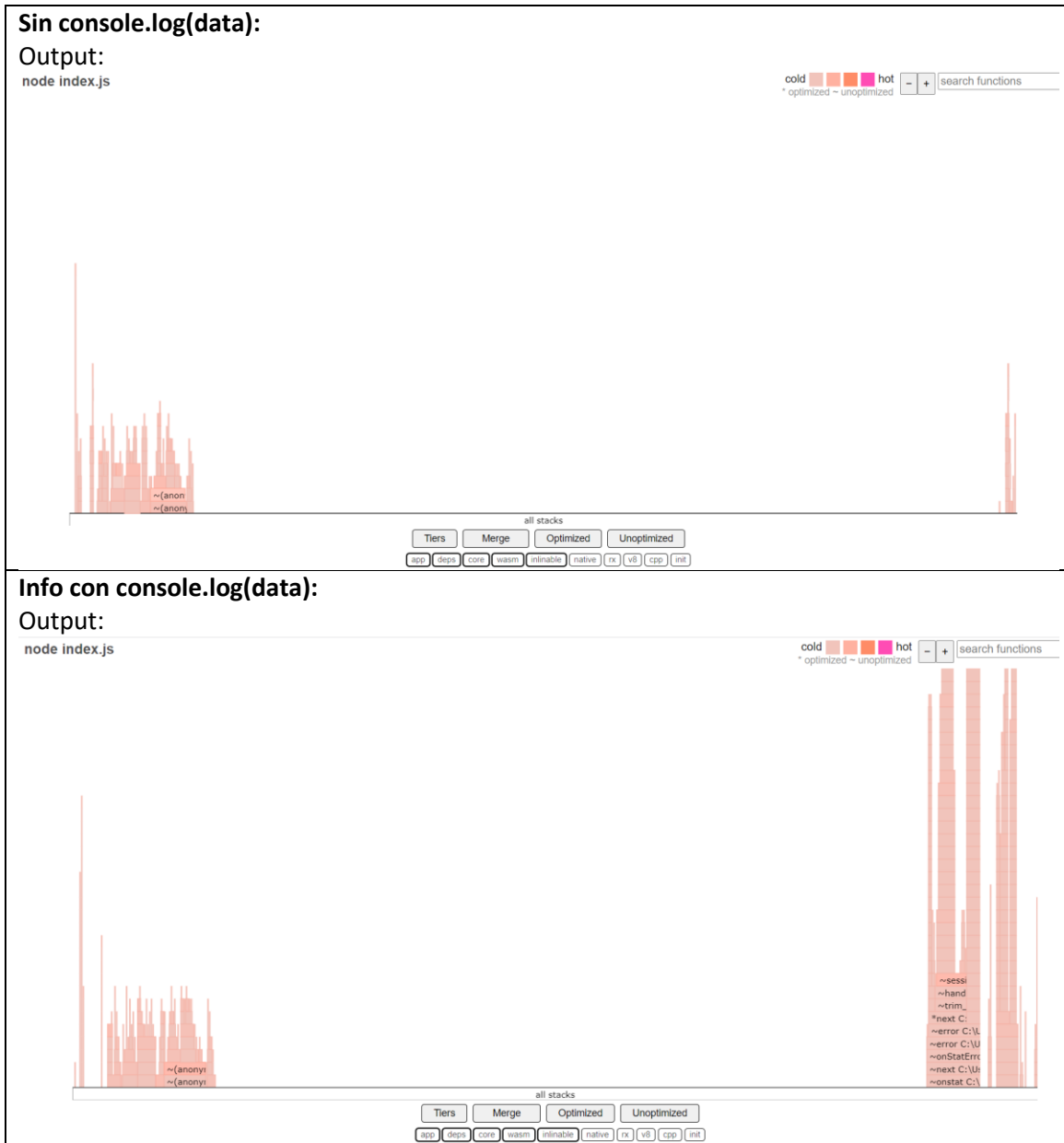
```
11 routers > . info.js > ...
12 router.get("/", /* compression(), */ async (req,res)=>{
13   try{
14     const data = {
15       argumentosDeEntrada: process.argv.slice(2),
16       nombreDeLaPlataforma: process.platform,
17       versionDeNode: process.version,
18       memoriaTotalReservada: process.memoryUsage(),
19       pathDeEjecucion: process.execPath,
20       processId: process.pid,
21       carpetaDelProyecto: process.cwd(),
22       cantidadDeProcesadores: cpu.length
23     }
24     console.log(data)
25     res.send(data)
26   } catch(err) {
27     res.status(404).send(err)
28   }
29 }
30
31 export { router }
```

#### Output:

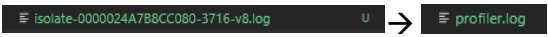
```
1 infoConConsoleLog.txt
2 {
3   "aggregate": {
4     "counters": {
5       "http.requests": 1000,
6       "http.codes.200": 1000,
7       "http.responses": 1000,
8       "vusers.failed": 0,
9       "vusers.completed": 50,
10      "vusers.created_by_name.0": 50,
11      "vusers.created": 50
12    },
13    "rates": {
14      "http.request_rate": 259
15    },
16    "firstCounterAt": 1672183587444,
17    "firstHistogramAt": 1672183587468,
18    "lastCounterAt": 1672183590662,
19    "lastHistogramAt": 1672183590662,
20    "firstMetricAt": 1672183587444,
21    "lastMetricAt": 1672183590662,
22    "period": 1672183590000,
23    "summaries": {
24      "http.response_time": {
25        "min": 1,
26        "max": 236,
27        "count": 1000,
28        "p50": 111.1,
29        "median": 111.1,
30        "p75": 147,
31        "p90": 172.5,
32        "p95": 183.1,
33        "p99": 190.6,
34        "p999": 232.8
35      }
36    }
37  }
38 }
```

### 2) 0x + Artillery

- 0x -o index.js
- artillery quick --count 50 --num 20 http://localhost:8080/info
- matamos el servidor



### 3) node profiler + Artillery

- node --prof index.js
- artillery quick \--count 50 \--num 20 \http://localhost:8080/info
- renombrar el file generado a "profiler.log"  

- node --prof-process profiler.log > result\_profiler.txt

## Sin console.log(data):

Output:

```
# result_profiler.txt
1 Statistical profiling result from profiler.log, (1796 ticks, 0 unaccounted, 0 excluded).
2
3 [Shared libraries]:
4 ticks total nonlib name
5 1598 89.0% C:\WINDOWS\SYSTEM32\ntdll.dll
6 192 10.7% C:\Program Files\nodejs\node.exe
7 1 0.1% C:\WINDOWS\System32\KERNELBASE.dll
8
9 [JavaScript]:
10 ticks total nonlib name
11 2 0.1% 40.0% LazyCompile: *resolve node:path:158:10
12 1 0.1% 20.0% LazyCompile: *next C:\Users\leand\Desktop\Proyectos_program\backend32105_glitch\node_modules\express\lib\router\index.js:177:16
13 1 0.1% 20.0% Function: ^handle C:\Users\leand\Desktop\Proyectos_program\backend32105_glitch\node_modules\express\lib\application.js:165:29
14 1 0.1% 20.0% Function: ^Socket.resume node:net:630:35
15
16 [C++]:
17 ticks total nonlib name
18
19 [Summary]:
20 ticks total nonlib name
21 5 0.3% 100.0% JavaScript
22 0 0.0% 0.0% C++
23 14 0.8% 280.0% GC
24 1791 99.7% Shared libraries
25
26 [C++ entry points]:
27 ticks cpp total name
28
29 [Bottom up (heavy) profile]:
30 Note: percentage shows a share of a particular caller in the total
31 amount of its parent calls.
32 Callers occupying less than 1.0% are not shown.
33
34 ticks parent name
35 1598 89.0% C:\WINDOWS\SYSTEM32\ntdll.dll
36
37 192 10.7% C:\Program Files\nodejs\node.exe
38 1 0.1% C:\WINDOWS\System32\KERNELBASE.dll
```

## Info con console.log(data):

Output:

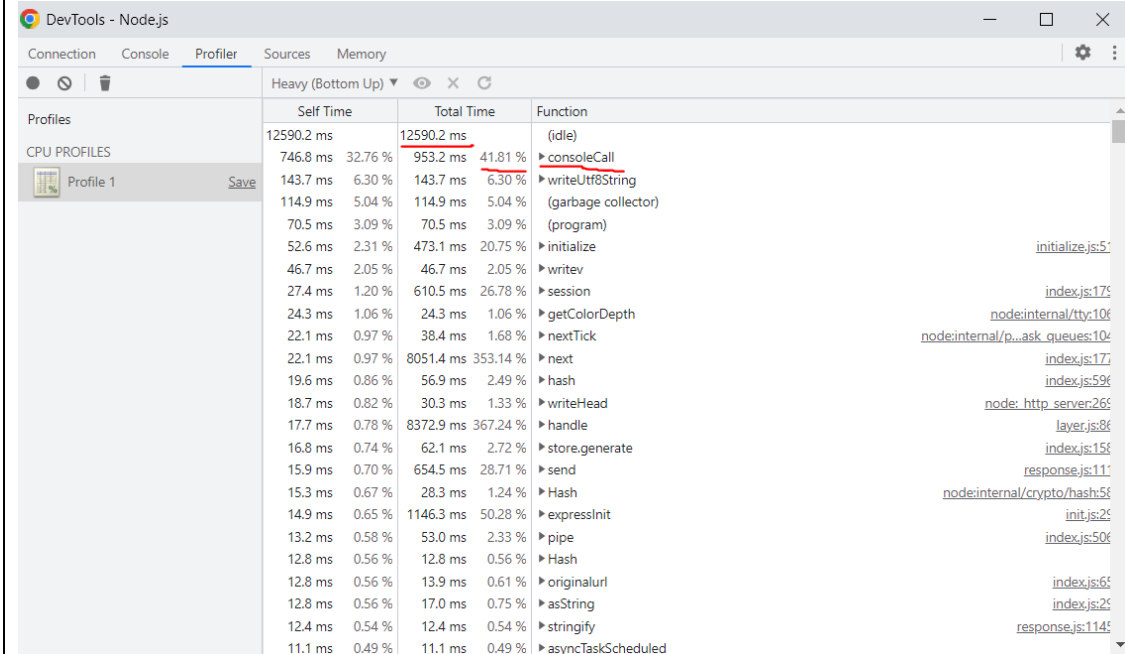
```
# result_profiler_consolelog.txt
1 Statistical profiling result from profiler_consolelog.log, (1077 ticks, 0 unaccounted, 0 excluded).
2
3 [Shared libraries]:
4 ticks total nonlib name
5 815 75.7% C:\WINDOWS\SYSTEM32\ntdll.dll
6 252 23.4% C:\Program Files\nodejs\node.exe
7 1 0.1% C:\WINDOWS\System32\KERNELBASE.dll
8
9 [JavaScript]:
10 ticks total nonlib name
11 1 0.1% 11.1% LazyCompile: *toNamespacedPath node:path:618:19
12 1 0.1% 11.1% LazyCompile: *resolve node:path:158:10
13 1 0.1% 11.1% LazyCompile: *nextPart node:fs:2384:31
14 1 0.1% 11.1% LazyCompile: *emit node:events:340:44
15 1 0.1% 11.1% Function: ^secondaryOK C:\Users\leand\Desktop\Proyectos_program\backend32105_glitch\node_modules\mongodb\lib\read_preference.js:164:16
16 1 0.1% 11.1% Function: ^realpathSync node:fs:2408:22
17 1 0.1% 11.1% Function: ^originalurl C:\Users\leand\Desktop\Proyectos_program\backend32105_glitch\node_modules\parseurl\index.js:65:22
18 1 0.1% 11.1% Function: ^nativeModuleRequire node:internal/bootstrap/loaders:332:29
19 1 0.1% 11.1% Function: ^asString C:\Users\leand\Desktop\Proyectos_program\backend32105_glitch\node_modules\date-format\lib\index.js:29:18
20
21 [C++]:
22 ticks total nonlib name
23
24 [Summary]:
25 ticks total nonlib name
26 9 0.8% 100.0% JavaScript
27 0 0.0% 0.0% C++
28 9 0.8% 100.0% GC
29 1068 99.2% Shared libraries
30
31 [C++ entry points]:
32 ticks cpp total name
33
34 [Bottom up (heavy) profile]:
35 Note: percentage shows a share of a particular caller in the total
36 amount of its parent calls.
```

## 4) Inspector del explorador chromium:

- node --inspect index.js
- chrome://inspect
- profiler / start
- artillery quick \--count 50 \--num 20 \http://localhost:8080/info
- stop

## Sin console.log(data):

### Output:

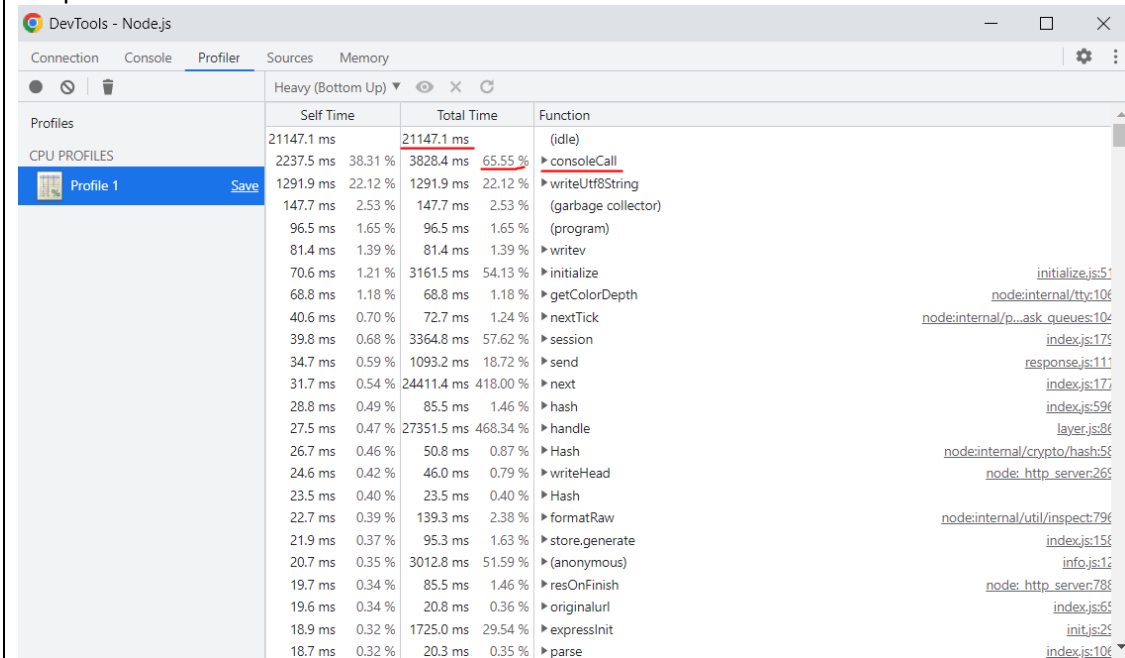


The screenshot shows the DevTools Profiler with the 'Profile 1' selected. The table displays the following data:

| Self Time        | Total Time         | Function            |
|------------------|--------------------|---------------------|
| 12590.2 ms       | 12590.2 ms         | (idle)              |
| 746.8 ms 32.76 % | 953.2 ms 41.81 %   | consoleCall         |
| 143.7 ms 6.30 %  | 143.7 ms 6.30 %    | writeUtf8String     |
| 114.9 ms 5.04 %  | 114.9 ms 5.04 %    | (garbage collector) |
| 70.5 ms 3.09 %   | 70.5 ms 3.09 %     | (program)           |
| 52.6 ms 2.31 %   | 473.1 ms 20.75 %   | initialize          |
| 46.7 ms 2.05 %   | 46.7 ms 2.05 %     | writev              |
| 27.4 ms 1.20 %   | 610.5 ms 26.78 %   | session             |
| 24.3 ms 1.06 %   | 24.3 ms 1.06 %     | getColorDepth       |
| 22.1 ms 0.97 %   | 38.4 ms 1.68 %     | nextTick            |
| 22.1 ms 0.97 %   | 8051.4 ms 353.14 % | next                |
| 19.6 ms 0.86 %   | 56.9 ms 2.49 %     | hash                |
| 18.7 ms 0.82 %   | 30.3 ms 1.33 %     | writeHead           |
| 17.7 ms 0.78 %   | 8372.9 ms 367.24 % | handle              |
| 16.8 ms 0.74 %   | 62.1 ms 2.72 %     | store.generate      |
| 15.9 ms 0.70 %   | 654.5 ms 28.71 %   | send                |
| 15.3 ms 0.67 %   | 28.3 ms 1.24 %     | Hash                |
| 14.9 ms 0.65 %   | 1146.3 ms 50.28 %  | expressInit         |
| 13.2 ms 0.58 %   | 53.0 ms 2.33 %     | pipe                |
| 12.8 ms 0.56 %   | 12.8 ms 0.56 %     | Hash                |
| 12.8 ms 0.56 %   | 13.9 ms 0.61 %     | originalurl         |
| 12.8 ms 0.56 %   | 17.0 ms 0.75 %     | asString            |
| 12.4 ms 0.54 %   | 12.4 ms 0.54 %     | stringify           |
| 11.1 ms 0.49 %   | 11.1 ms 0.49 %     | asyncTaskScheduled  |

## Info con console.log(data):

### Output:



The screenshot shows the DevTools Profiler with the 'Profile 1' selected. The table displays the following data:

| Self Time         | Total Time          | Function            |
|-------------------|---------------------|---------------------|
| 21147.1 ms        | 21147.1 ms          | (idle)              |
| 2237.5 ms 38.31 % | 3828.4 ms 65.55 %   | consoleCall         |
| 1291.9 ms 22.12 % | 1291.9 ms 22.12 %   | writeUtf8String     |
| 147.7 ms 2.53 %   | 147.7 ms 2.53 %     | (garbage collector) |
| 96.5 ms 1.65 %    | 96.5 ms 1.65 %      | (program)           |
| 81.4 ms 1.39 %    | 81.4 ms 1.39 %      | writev              |
| 70.6 ms 1.21 %    | 3161.5 ms 54.13 %   | initialize          |
| 68.8 ms 1.18 %    | 68.8 ms 1.18 %      | getColorDepth       |
| 40.6 ms 0.70 %    | 72.7 ms 1.24 %      | nextTick            |
| 39.8 ms 0.68 %    | 3364.8 ms 57.62 %   | session             |
| 34.7 ms 0.59 %    | 1093.2 ms 18.72 %   | send                |
| 31.7 ms 0.54 %    | 24411.4 ms 418.00 % | next                |
| 28.8 ms 0.49 %    | 85.5 ms 1.46 %      | hash                |
| 27.5 ms 0.47 %    | 27351.5 ms 468.34 % | handle              |
| 26.7 ms 0.46 %    | 50.8 ms 0.87 %      | Hash                |
| 24.6 ms 0.42 %    | 46.0 ms 0.79 %      | writeHead           |
| 23.5 ms 0.40 %    | 23.5 ms 0.40 %      | Hash                |
| 22.7 ms 0.39 %    | 139.3 ms 2.38 %     | formatRaw           |
| 21.9 ms 0.37 %    | 95.3 ms 1.63 %      | store.generate      |
| 20.7 ms 0.35 %    | 3012.8 ms 51.59 %   | (anonymous)         |
| 19.7 ms 0.34 %    | 85.5 ms 1.46 %      | resOnFinish         |
| 19.6 ms 0.34 %    | 20.8 ms 0.36 %      | originalurl         |
| 18.9 ms 0.32 %    | 1725.0 ms 29.54 %   | expressInit         |
| 18.7 ms 0.32 %    | 20.3 ms 0.35 %      | parse               |

**Conclusión:** El tiempo de respuesta suele ser menor en el caso sin console.log, pero también existe una varianza significativa entre 2 tests corridos bajo las mismas condiciones.