



Desafío: Loggers, gzip y análisis de performance

Verificar sobre la ruta /info con y sin compresión, la diferencia de cantidad de bytes devueltos en un caso y otro.

Con **compresión**:

Size
715 B

Sin **compresión**:

Size	▼
644 B	

ANÁLISIS DE PERFORMANCE

1. Iniciamos el servidor en modo profiler (sin console.log en la ruta /info)

```
node --prof server.js
```

Ingresar a <http://localhost:8081/info>

Realizar carga con Artillery

```
artillery quick -c 50 -n 20 "http://localhost:8081/info" > NObloq.txt
```

Resultado:

```
Phase started: unnamed (index: 0, duration: 1s) 02:05:24(-0300)

Phase completed: unnamed (index: 0, duration: 1s) 02:05:25(-0300)

-----
Metrics for period to: 02:05:30(-0300) (width: 2.238s)
-----

http.codes.200: ..... 1000
http.request_rate: ..... 452/sec
http.requests: ..... 1000
http.response_time:
  min: ..... 1
  max: ..... 120
  median: ..... 34.8
  p95: ..... 66
  p99: ..... 87.4
http.responses: ..... 1000
vusers.completed: ..... 50
vusers.created: ..... 50
vusers.created_by_name.0: ..... 50
vusers.failed: ..... 0
```

2. Iniciamos el servidor en modo profiler (con console.log en la ruta /info)

```
node --prof server.js
```

Ingresar a <http://localhost:8081/info>

Realizar carga con Artillery

```
artillery quick -c 50 -n 20 "http://localhost:8081/info" > bloq.txt
```

```
Phase started: unnamed (index: 0, duration: 1s) 02:13:25(-0300)

Phase completed: unnamed (index: 0, duration: 1s) 02:13:26(-0300)

-----
Metrics for period to: 02:13:30(-0300) (width: 2.79s)
-----

http.codes.200: ..... 1000
http.request_rate: ..... 362/sec
http.requests: ..... 1000
http.response_time:
| min: ..... 4
| max: ..... 210
| median: ..... 80.6
| p95: ..... 169
| p99: ..... 206.5
http.responses: ..... 1000
vusers.completed: ..... 50
vusers.created: ..... 50
vusers.created_by_name.0: ..... 50
vusers.failed: ..... 0
vusers.session_length:
```

```
node --prof-process no-bloq-v8.log > result_no-bloq.txt
node --prof-process bloq-v8.log > result_bloq.txt
```

result_bloq.txt

```
Statistical profiling result from bloq-v8.log, (15432 ticks, 0 unaccounted, 0 excluded).

[Shared libraries]:
  ticks  total  nonlib   name
  14871  96.4%    C:\WINDOWS\SYSTEM32\ntdll.dll
     535   3.5%    C:\Program Files\nodejs\node.exe
       4   0.0%    C:\WINDOWS\System32\KERNELBASE.dll
       1   0.0%    C:\WINDOWS\System32\KERNEL32.DLL

[JavaScript]:
```

result_no-bloq.txt

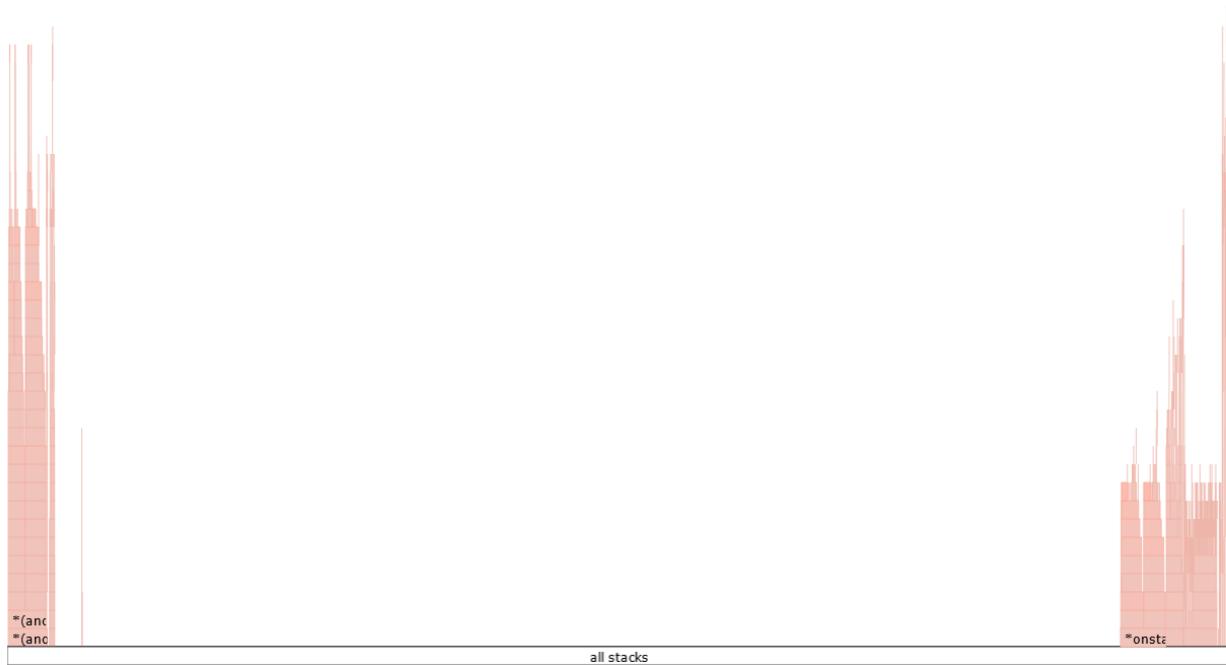
```
Statistical profiling result from no-bloq-v8.log, (25882 ticks, 0 unaccounted, 0 excluded).

[Shared libraries]:
  ticks  total  nonlib   name
  25344   97.9%      C:\WINDOWS\SYSTEM32\ntdll.dll
    531    2.1%      C:\Program Files\nodejs\node.exe
      1    0.0%      C:\WINDOWS\System32\KERNELBASE.dll

[JavaScript]:
```

Autocannon

Gráfico flama realizado con un proceso bloqueante



```
Running 20s test @ http://localhost:8081/info
100 connections
```

Stat	2.5%	50%	97.5%	99%	Avg	Stdev	Max
Latency	44 ms	58 ms	124 ms	146 ms	64.21 ms	19.03 ms	164 ms

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	962	962	1581	1800	1548.35	191.55	962
Bytes/Sec	691 kB	691 kB	1.14 MB	1.29 MB	1.11 MB	138 kB	691 kB

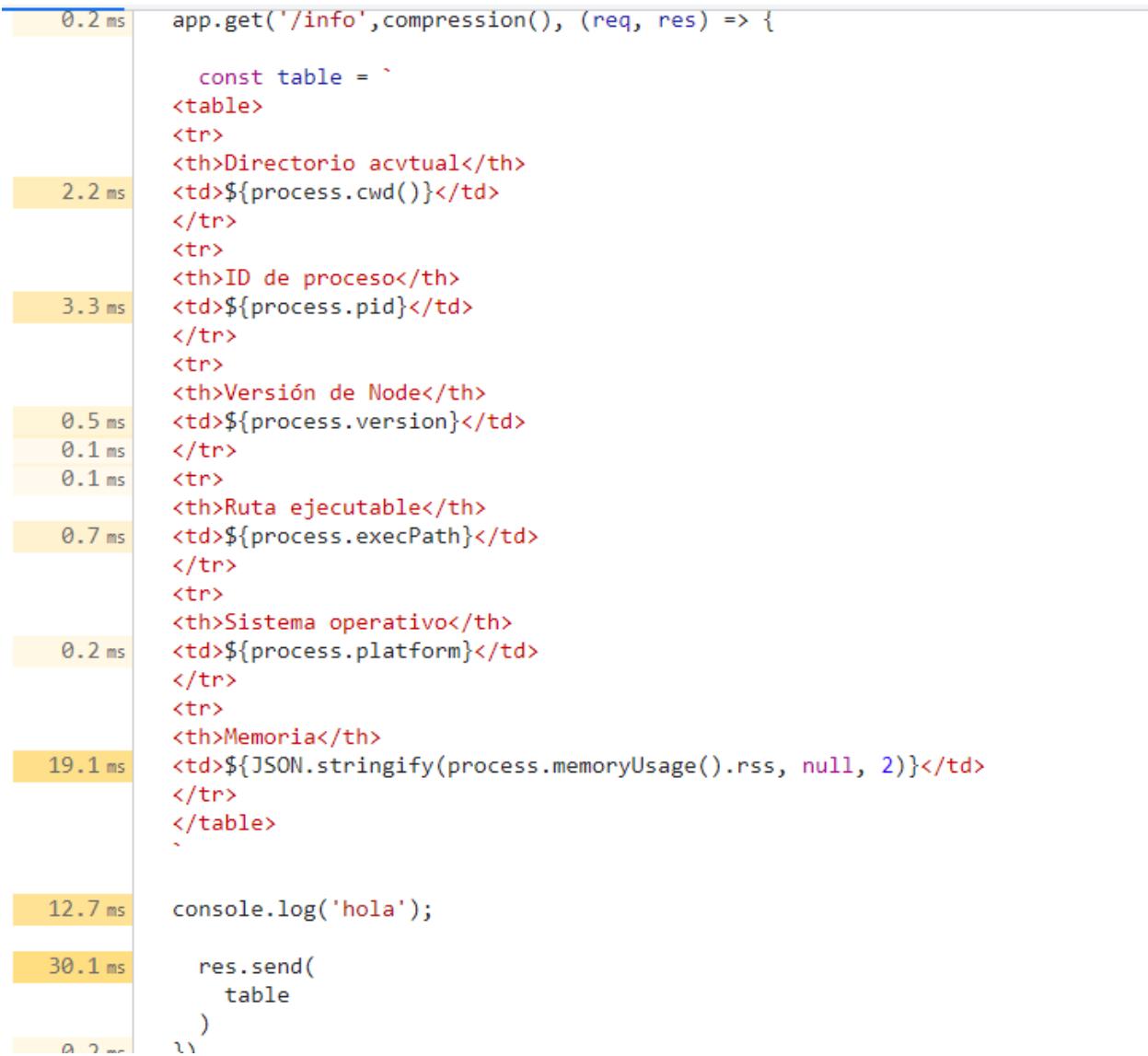
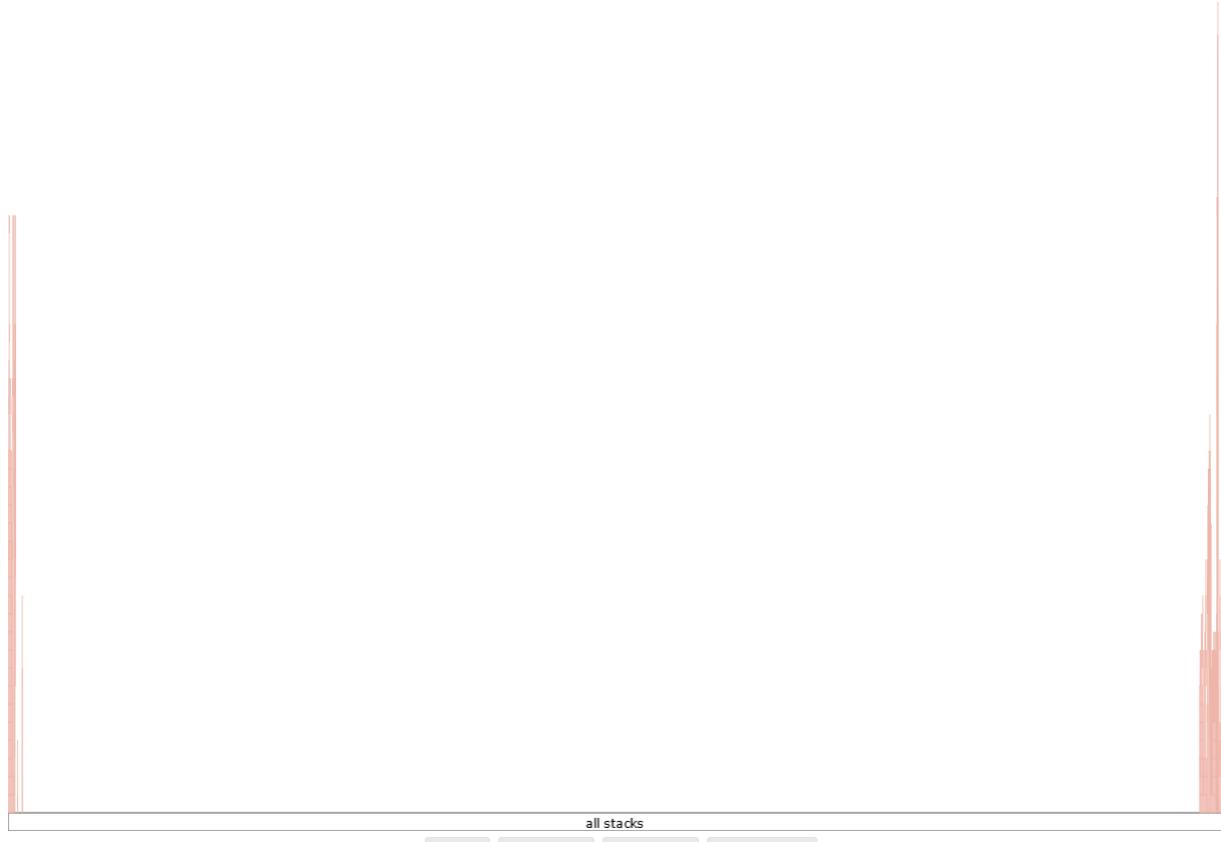


Gráfico flama realizado con un proceso **no** bloqueante



```
Running 20s test @ http://localhost:8081/info
100 connections
```

Stat	2.5%	50%	97.5%	99%	Avg	Stdev	Max
Latency	48 ms	66 ms	167 ms	179 ms	75.16 ms	28.4 ms	222 ms

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	700	700	1387	1653	1323.65	273.52	700
Bytes/Sec	502 kB	502 kB	995 kB	1.19 MB	949 kB	196 kB	502 kB

```

149
150     app.get('/info', compression(), (req, res) => {
151         const table = `
152         <table>
153             <tr>
154                 <th>Directorio acvtual</th>
155                 <td>${process.cwd()}</td>
156             </tr>
157             <tr>
158                 <th>ID de proceso</th>
159                 <td>${process.pid}</td>
160             </tr>
161             <tr>
162                 <th>Versión de Node</th>
163                 <td>${process.version}</td>
164             </tr>
165             <tr>
166                 <th>Ruta ejecutable</th>
167                 <td>${process.execPath}</td>
168             </tr>
169             <tr>
170                 <th>Sistema operativo</th>
171                 <td>${process.platform}</td>
172             </tr>
173             <tr>
174                 <th>Memoria</th>
175                 <td>${JSON.stringify(process.memoryUsage().rss, null, 2)}</td>
176             </tr>
177         </table>
178         `
179         0.2 ms
180         res.send(
181             table
182         )
```

```

## Conclusión

Como se puede observar en los ejemplos previos, resulta altamente aconsejable evitar la utilización de funciones que bloqueen el proceso, ya que estas ralentizan significativamente el rendimiento del servidor. Es importante recordar que en entornos de alta concurrencia, como aquellos que involucran una gran cantidad de solicitudes por segundo, cualquier obstáculo que detenga el flujo de la ejecución puede tener un impacto adverso en la capacidad de respuesta del sistema.

Una alternativa efectiva es hacer uso de funciones no bloqueantes, también conocidas como funciones asíncronas, que permiten continuar con la ejecución mientras se espera una respuesta. Esto evita que el proceso se detenga innecesariamente, mejorando así la eficiencia y la capacidad de respuesta del servidor.