

- A) Vamos a trabajar sobre la ruta '/info', en modo fork, agregando ó extrayendo un console.log de la información colectada antes de devolverla al cliente. Además desactivaremos el child_process de la ruta '/randoms'
- Para ambas condiciones (con o sin console.log) en la ruta '/info' OBTENER:

1) Resultados con Prof de Node

```
1 Statistical profiling result from prof-ConsoleLog.log, (1237 ticks, 0 unaccounted, 0
2
3 [Shared libraries]:
4   ticks total nonlib name
5   1111  89.8%      C:\WINDOWS\SYSTEM32\ntdll.dll
6     124  10.0%      C:\Program Files\nodejs\node.exe
7
8 [JavaScript]:
9   ticks total nonlib name
10    1    0.1%  50.0% Function: ^send D:\ProgramacionBackend\desafios\14\node_mod
11    1    0.1%  50.0% Function: ^isEmpty node:internal/fixe_queue:95:10
12
13 [C++]:
14   ticks total nonlib name
15
16 [Summary]:
17   ticks total nonlib name
18     2    0.2% 100.0% JavaScript
19     0    0.0%   0.0% C++
20     8    0.6% 400.0% GC
21   1235  99.8%      Shared libraries
22
23 [C++ entry points]:
24   ticks cpp total name
25
```

```
1 Statistical profiling result from prof-NoConsoleLog.log, (10122 ticks, 0 unaccount
2
3 [Shared libraries]:
4   ticks total nonlib name
5   9997  98.8%      C:\WINDOWS\SYSTEM32\ntdll.dll
6     121   1.2%      C:\Program Files\nodejs\node.exe
7
8 [JavaScript]:
9   ticks total nonlib name
10    2    0.0%  50.0% LazyCompile: *resolve node:path:158:10
11    1    0.0%  25.0% RegExp: ^.*[\\.\\/\\]
12    1    0.0%  25.0% Function: ^clearBuffer node:internal/streams/writable:532
13
14 [C++]:
15   ticks total nonlib name
16
17 [Summary]:
18   ticks total nonlib name
19     4    0.0% 100.0% JavaScript
20     0    0.0%   0.0% C++
21     7    0.1% 175.0% GC
22  10118 100.0%      Shared libraries
23
24 [C++ entry points]:
25   ticks cpp total name
```

2) Resultados de Artillery:

Con Console Log.

```
Phase started: unnamed (index: 0, duration: 1s) 11:11:38(-0300)

Phase completed: unnamed (index: 0, duration: 1s) 11:11:39(-0300)

-----
Metrics for period to: 11:11:40(-0300) (width: 1.172s)
-----

http.codes.200: ..... 743
http.request_rate: ..... 661/sec
http.requests: ..... 769
http.response_time:
  min: ..... 2
  max: ..... 91
  median: ..... 30.3
  p95: ..... 51.9
  p99: ..... 58.6
http.responses: ..... 743
vusers.completed: ..... 23
vusers.created: ..... 50
vusers.created_by_name.0: ..... 50
vusers.failed: ..... 0
vusers.session_length:
  min: ..... 194.6
  max: ..... 730.1
  median: ..... 507.8
  p95: ..... 713.5
  p99: ..... 713.5
```

Sin Console Log.

```
Phase started: unnamed (index: 0, duration: 1s) 11:09:54(-0300)

Phase completed: unnamed (index: 0, duration: 1s) 11:09:55(-0300)

-----
Metrics for period to: 11:10:00(-0300) (width: 1.019s)
-----

http.codes.200: ..... 1000
http.request_rate: ..... 990/sec
http.requests: ..... 1000
http.response_time:
  min: ..... 0
  max: ..... 31
  median: ..... 1
  p95: ..... 5
  p99: ..... 7.9
http.responses: ..... 1000
vusers.completed: ..... 50
vusers.created: ..... 50
vusers.created_by_name.0: ..... 50
vusers.failed: ..... 0
vusers.session_length:
  min: ..... 21.4
  max: ..... 143
  median: ..... 30.3
  p95: ..... 106.7
  p99: ..... 130.3
```

Luego utilizaremos Autocannon en línea de comandos, emulando 100 conexiones concurrentes realizadas en un tiempo de 20 segundos. Extraer un reporte con los resultados (puede ser un print screen de la consola)

1) Resumen Autocannon No Bloqueante

Stat	2.5%	50%	97.5%	99%	Avg	Stdev	Max
Latency	1 ms	44 ms	89 ms	95 ms	39.36 ms	27.22 ms	122 ms

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	1736	1736	2577	2689	2503.25	216.75	1736
Bytes/Sec	1.15 MB	1.15 MB	1.71 MB	1.78 MB	1.66 MB	144 kB	1.15 MB

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

2) Resumen Autocannon Bloqueante

Stat	2.5%	50%	97.5%	99%	Avg	Stdev	Max
Latency	6 ms	81 ms	106 ms	114 ms	74.38 ms	24.95 ms	163 ms

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	1061	1061	1361	1390	1333.25	77.52	1061
Bytes/Sec	701 kB	701 kB	900 kB	921 kB	882 kB	51.5 kB	701 kB

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

27k requests in 20.03s, 17.6 MB read

1) Inspect Bloqueante

DevTools is now available in Spanish! [Always match Chrome's language](#) [Switch DevTools to Spanish](#) [Don't show again](#)

Connection	Console	Profiler	Sources	Memory
Node	Filesystem	Snippets	server.js x	

```
// PLANTILLAS
78 app.set("view engine", "ejs");
79
80
81 const mensajesContenedor = new Contenedor("mensajes.txt");
82
83 app.use(loginRouter);
84 app.use(productsRouter);
85 app.use("/api", randomRouter);
86
87 app.get("/infoSinLog", (req, res) => {
88   res.send(`<h2>Argumentos de entrada: ${args}</h2>
89   <h2>Sistema Operativo: ${process.platform}</h2>
90   <h2>Version de Node: ${process.version}</h2>
91   <h2>Memoria total reservada: ${process.memoryUsage().heapUsed}</h2>
92   <h2>Path de Ejecucion: ${process.cwd()}</h2>
93   <h2>Process ID: ${process.pid}</h2>
94   <h2>Servidor escuchando en ${PORT} con Process id de ${
95     process.pid
96   } - ${new Date().toLocaleString()}</h2>
97   <h2>Process ID: ${cpus().length}</h2>
98   <h2>Carpeta del proyecto: ${process.cwd().split("\\").pop()}</h2>`);
99 });
100 // ----- INFO
101
102 0.7 ms app.get("/info", (req, res) => {
103   5.8 ms console.log(`<h2>Argumentos de entrada: ${args}</h2>
104   <h2>Sistema Operativo: ${process.platform}</h2>
105   <h2>Version de Node: ${process.version}</h2>
106   <h2>Memoria total reservada: ${process.memoryUsage().heapUsed}</h2>
107   <h2>Path de Ejecucion: ${process.cwd()}</h2>
108   <h2>Process ID: ${process.pid}</h2>
109   <h2>Servidor escuchando en ${PORT} con Process id de ${
110     process.pid
111   } - ${new Date().toLocaleString()}</h2>
112   <h2>Process ID: ${cpus().length}</h2>
113   <h2>Carpeta del proyecto: ${process.cwd().split("\\").pop()}</h2>`);
114
115   16.5 ms res.send(`<h2>Argumentos de entrada: ${args}</h2>
116   <h2>Sistema Operativo: ${process.platform}</h2>
117   <h2>Version de Node: ${process.version}</h2>
118   <h2>Memoria total reservada: ${process.memoryUsage().heapUsed}</h2>
119   <h2>Path de Ejecucion: ${process.cwd()}</h2>
120   <h2>Process ID: ${process.pid}</h2>
121   <h2>Servidor escuchando en ${PORT} con Process id de ${
122     process.pid
123   } - ${new Date().toLocaleString()}</h2>
124   <h2>Process ID: ${cpus().length}</h2>
125   <h2>Carpeta del proyecto: ${process.cwd().split("\\").pop()}</h2>`);
126   });
127 }
```

{ } Line 102, Column 20

Console

2) Inspect No Bloqueante

DevTools is now available in Spanish! [Always match Chrome's language](#) [Switch DevTools to Spanish](#) [Don't show again](#)

Connection	Console	Profiler	Sources	Memory
Node	Filesystem	Snippets	server.js x	

```
//
78 // PLANTILLAS
79 app.set("view engine", "ejs");
80
81 const mensajesContenedor = new Contenedor("mensajes.txt");
82
83 app.use(loginRouter);
84 app.use(productsRouter);
85 app.use("/api", randomRouter);
86
87 0.1 ms app.get("/infoSinLog", (req, res) => {
88   7.7 ms res.send(`<h2>Argumentos de entrada: ${args}</h2>
89   <h2>Sistema Operativo: ${process.platform}</h2>
90   <h2>Version de Node: ${process.version}</h2>
91   <h2>Memoria total reservada: ${process.memoryUsage().heapUsed}</h2>
92   <h2>Path de Ejecucion: ${process.cwd()}</h2>
93   <h2>Process ID: ${process.pid}</h2>
94   <h2>Servidor escuchando en ${PORT} con Process id de ${
95     process.pid
96   } - ${new Date().toLocaleString()}</h2>
97   <h2>Process ID: ${cpus().length}</h2>
98   <h2>Carpeta del proyecto: ${process.cwd().split("\\").pop()}</h2>`);
99 });
100 // ----- INFO
101
102 app.get("/info", (req, res) => {
103   console.log(`<h2>Argumentos de entrada: ${args}</h2>
104   <h2>Sistema Operativo: ${process.platform}</h2>
105   <h2>Version de Node: ${process.version}</h2>
106   <h2>Memoria total reservada: ${process.memoryUsage().heapUsed}</h2>
107   <h2>Path de Ejecucion: ${process.cwd()}</h2>
108   <h2>Process ID: ${process.pid}</h2>
109   <h2>Servidor escuchando en ${PORT} con Process id de ${
110     process.pid
111   } - ${new Date().toLocaleString()}</h2>
112   <h2>Process ID: ${cpus().length}</h2>
113   <h2>Carpeta del proyecto: ${process.cwd().split("\\").pop()}</h2>`);
114
115   res.send(`<h2>Argumentos de entrada: ${args}</h2>
116   <h2>Sistema Operativo: ${process.platform}</h2>
117   <h2>Version de Node: ${process.version}</h2>
118   <h2>Memoria total reservada: ${process.memoryUsage().heapUsed}</h2>
119   <h2>Path de Ejecucion: ${process.cwd()}</h2>
120   <h2>Process ID: ${process.pid}</h2>
121   <h2>Servidor escuchando en ${PORT} con Process id de ${
122     process.pid
123   } - ${new Date().toLocaleString()}</h2>
124   <h2>Process ID: ${cpus().length}</h2>
125   <h2>Carpeta del proyecto: ${process.cwd().split("\\").pop()}</h2>`);
126 });
127 }
```


1) Grafico de flama con procesos no bloqueantes



Tiers Merge Optimized Unoptimized

app deps core wasm inlinable native rx v8 cpp init

2) Grafico de Flama con procesos bloqueantes



Tiers Merge Optimized Unoptimized

app deps core wasm inlinable native rx v8 cpp init

Conclusion:

En todos los aspectos el console log fue bloqueante con un request rate de 661/seg contra 990/seg, y en especial y más importante con el console log activado fallaron 50 request a diferencia de la prueba sin console log que no tuvo failed request.

Y en relación a los Ticks, hubo 10 veces más con el console log que sin.

Analizando los resultados de Autocannon se ve claramente que con procesos bloqueantes existieron 1061 request por segundo, contra 1700 (casi un 70% más) en procesos no bloqueantes, lo que permite mas request en menos tiempo.

Utilizando inspect, vemos en el código como el endpoint /Info (bloqueante) tomo 0.7ms, contra los 0.1ms del endpoint no bloqueante /infoSinLog

En comparación de ambos gráficos de flama, se ve que los procesos bloqueantes son más anchos en relación a la prueba de no bloqueante.