Comparativa “prof”

Console

```
cleStatistical profiling result from console-v8.log, (826 ticks, 1 unaccounted, 0 excluded).

[Shared libraries]:
  ticks  total  nonlib   name
    609   73.7%          C:\Windows\SYSTEM32\ntdll.dll
    206   24.9%          C:\Program Files\nodejs\node.exe

[JavaScript]:
  ticks  total  nonlib   name
      2    0.2%   18.2%  LazyCompile: *normalize node:path:304:12
      1    0.1%    9.1%  RegExp: (<%%|%%>|<%=|<%-|<%_|<%#|<%|%>|-%>|_%>)
      1    0.1%    9.1%  LazyCompile: *nextTick node:internal/process/task_queues:104:18
      1    0.1%    9.1%  LazyCompile: *formatProperty node:internal/util/inspect:1810:24
      1    0.1%    9.1%  Function: ^resolve node:path:158:10
      1    0.1%    9.1%  Function: ^hasObserver node:internal/perf/observe:441:21
      1    0.1%    9.1%  Function: ^compile D:\Lucas\Proyectos\Cursos Programacion\BACKEND CODER\DESAFIOS ENTREGABLES\DES
      1    0.1%    9.1%  Function: ^Negotiator D:\Lucas\Proyectos\Cursos Programacion\BACKEND CODER\DESAFIOS ENTREGABLES\
      1    0.1%    9.1%  Function: ^<anonymous> node:internal/validators:73:3

[C++]:
  ticks  total  nonlib   name

[Summary]:
  ticks  total  nonlib   name
     10    1.2%   90.9%  JavaScript
      0    0.0%    0.0%  C++
      3    0.4%   27.3%  GC
    815   98.7%          Shared libraries
      1    0.1%          Unaccounted

[C++ entry points]:
  ticks    cpp   total   name

[Bottom up (heavy) profile]:
 Note: percentage shows a share of a particular caller in the total
 amount of its parent calls.
 Callers occupying less than 1.0% are not shown.
```

No Console

```
Statistical profiling result from noConsole-v8.log, (1439 ticks, 0 unaccounted, 0 excluded).

 [Shared libraries]:
   ticks  total  nonlib   name
    1316   91.5%          C:\Windows\SYSTEM32\ntdll.dll
     116    8.1%          C:\Program Files\nodejs\node.exe
       1    0.1%          C:\Windows\System32\KERNELBASE.dll

 [JavaScript]:
   ticks  total  nonlib   name
       2    0.1%   33.3%  RegExp: [ \t]*<%_
       2    0.1%   33.3%  LazyCompile: *resolve node:path:158:10
       1    0.1%   16.7%  Function: ^strategy.pass D:\Lucas\Proyectos\Cursos Programacion\BACKEND CODER\DESAFIOS ENTREGABLES\D
       1    0.1%   16.7%  Function: ^alignPool node:buffer:159:19

 [C++]:
   ticks  total  nonlib   name

 [Summary]:
   ticks  total  nonlib   name
       6    0.4%  100.0%  JavaScript
       0    0.0%    0.0%  C++
       4    0.3%   66.7%  GC
    1433   99.6%          Shared libraries

 [C++ entry points]:
   ticks    cpp   total   name

 [Bottom up (heavy) profile]:
  Note: percentage shows a share of a particular caller in the total
  amount of its parent calls.
  Callers occupying less than 1.0% are not shown.

   ticks parent  name
   1316   91.5%  C:\Windows\SYSTEM32\ntdll.dll
```

Comparativa “artillery”

Console

```
Running scenarios...
Phase started: unnamed (index: 0, duration: 1s) 20:20:23(-0300)

Phase completed: unnamed (index: 0, duration: 1s) 20:20:24(-0300)

All VUs finished. Total time: 5 seconds

--------------------------------
Summary report @ 20:20:27(-0300)
--------------------------------

http.codes.200: ..................................................................... 1000
http.request_rate: .................................................................. 189/sec
http.requests: ...................................................................... 1000
http.response_time:
  min: .............................................................................. 3
  max: .............................................................................. 133
  median: ........................................................................... 37
  p95: .............................................................................. 59.7
  p99: .............................................................................. 100.5
http.responses: ..................................................................... 1000
vusers.completed: ................................................................... 20
vusers.created: ..................................................................... 20
vusers.created_by_name.0: ........................................................... 20
vusers.failed: ...................................................................... 0
vusers.session_length:
  min: .............................................................................. 1477.6
  max: .............................................................................. 2049.2
  median: ........................................................................... 1901.1
  p95: .............................................................................. 2018.7
  p99: .............................................................................. 2018.7
```

No Console

```
Running scenarios...
Phase started: unnamed (index: 0, duration: 1s) 20:20:59(-0300)

Phase completed: unnamed (index: 0, duration: 1s) 20:21:00(-0300)

All VUs finished. Total time: 5 seconds

--------------------------------
Summary report @ 20:21:03(-0300)
--------------------------------

http.codes.200: ..................................................................... 1000
http.request_rate: .................................................................. 250/sec
http.requests: ...................................................................... 1000
http.response_time:
  min: .............................................................................. 0
  max: .............................................................................. 23
  median: ........................................................................... 2
  p95: .............................................................................. 4
  p99: .............................................................................. 10.1
http.responses: ..................................................................... 1000
vusers.completed: ................................................................... 20
vusers.created: ..................................................................... 20
vusers.created_by_name.0: ........................................................... 20
vusers.failed: ...................................................................... 0
vusers.session_length:
  min: .............................................................................. 79.3
  max: .............................................................................. 196.2
  median: ........................................................................... 144
  p95: .............................................................................. 183.1
  p99: .............................................................................. 183.1
```

Comparativa “autocannon”

Console

```
Lucas@DESKTOP-KVVS175 MINGW64 /d/Lucas/Proyectos/Cursos Programacion/BACKEND CODER/DESAFIOS ENTREGABLES/DESAFIO 13
$ npm test

> clase-10@1.0.0 test
> node benchmark.js

Running all benchmarks in parallel ...
Running 20s test @ http://localhost:8080/info
100 connections
```

Stat	2.5%	50%	97.5%	99%	Avg	Stdev	Max
Latency	166 ms	178 ms	239 ms	264 ms	185.86 ms	21.54 ms	321 ms

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	391	391	542	586	534.96	43.4	391
Bytes/Sec	1.07 MB	1.07 MB	1.48 MB	1.6 MB	1.46 MB	118 kB	1.07 MB

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

11k requests in 20.05s, 29.2 MB read
```

No Console

```
Lucas@DESKTOP-KVVS175 MINGW64 /d/Lucas/Proyectos/Cursos Programacion/BACKEND CODER/DESAFIOS ENTREGABLES/DESAFIO 13
$ npm test

> clase-10@1.0.0 test
> node benchmark.js

Running all benchmarks in parallel ...
Running 20s test @ http://localhost:8080/info
100 connections
```

Stat	2.5%	50%	97.5%	99%	Avg	Stdev	Max
Latency	50 ms	53 ms	80 ms	95 ms	55.72 ms	8.85 ms	139 ms

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	1222	1222	1863	1920	1777.15	195.7	1222
Bytes/Sec	3.33 MB	3.33 MB	5.08 MB	5.24 MB	4.85 MB	534 kB	3.33 MB

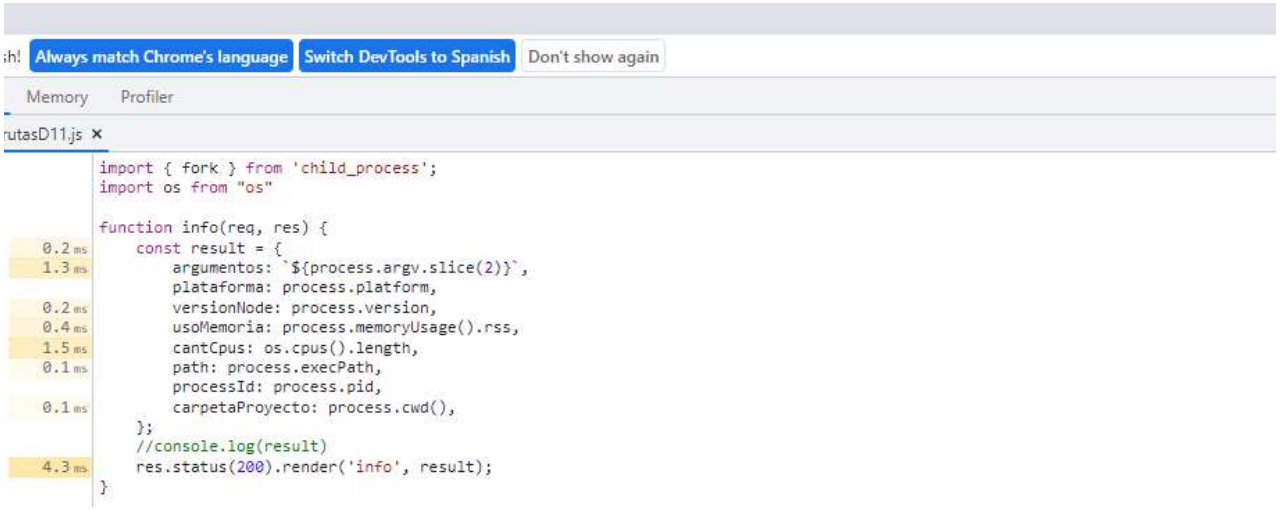
```
Req/Bytes counts sampled once per second.
# of samples: 20

36k requests in 20.04s, 96.9 MB read
```

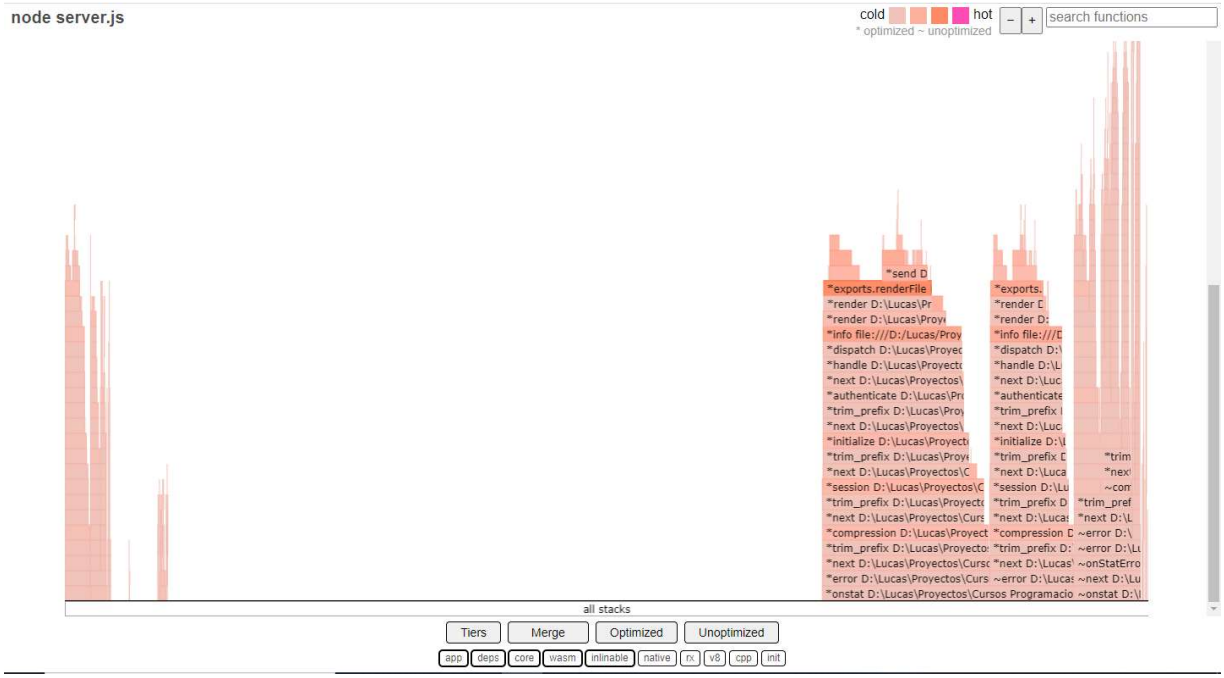
Comparativa “inspect”

Console

No Console



Flamegraph



Conclusión

Como era de esperarse al sacar el console.log del código el servidor funciona de manera más eficiente y puede más request por segundo como podemos ver claramente tanto con “Autocannon” como con “Artillery”. A los archivos hechos con “prof-process” no le veo mucho sentido pero entiendo que sirven para un análisis más profundo que no estamos realizando.
Lo que si encuentro interesante es lo realizado con “inspect” donde vemos información precisa de en cuánto tiempo se ejecuta cada línea de código.
Por otro lado el “flamegraph” es una buena herramienta para ver dónde enfocar los recursos a la hora de querer mejorar el rendimiento de nuestro servidor.
Una última conclusión que saqué de este ejercicio es que tanto “Artillery” como “Autocannon” son buenas herramientas para realizar pruebas de procesos, pero creo que están hechas para usarse en diferentes momentos. “Autocannon” está hecha para pruebas rápidas y repetitivas donde no es importante guardar un registro pero sí tener la información rápida y entendible con solo un vistazo. Y “Artillery” por el otro lado es para hacer pruebas donde si se requiere guardar un registro donde la información esté más detallada.