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

```
Statistical profiling result from prof-ConsoleLog.log, (1237 ticks, 0 unaccounted, 

 [Shared libraries]:
   ticks  total  nonlib   name
    1111   89.8%          C:\WINDOWS\SYSTEM32\ntdll.dll
     124   10.0%          C:\Program Files\nodejs\node.exe

 [JavaScript]:
   ticks  total  nonlib   name
       1    0.1%   50.0%  Function: ^send D:\ProgramacionBackend\desafios\14\node_mo
       1    0.1%   50.0%  Function: ^isEmpty node:internal/fixed_queue:95:10

 [C++]:
   ticks  total  nonlib   name

 [Summary]:
   ticks  total  nonlib   name
       2    0.2%  100.0%  JavaScript
       0    0.0%    0.0%  C++
       8    0.6%  400.0%  GC
    1235   99.8%          Shared libraries

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

```
Statistical profiling result from prof-NoConsoleLog.log, (10122 ticks, 0 unaccount

 [Shared libraries]:
   ticks  total  nonlib   name
    9997   98.8%          C:\WINDOWS\SYSTEM32\ntdll.dll
     121    1.2%          C:\Program Files\nodejs\node.exe

 [JavaScript]:
   ticks  total  nonlib   name
       2    0.0%   50.0%  LazyCompile: *resolve node:path:158:10
       1    0.0%   25.0%  RegExp: ^.*[\.\/\\]
       1    0.0%   25.0%  Function: ^clearBuffer node:internal/streams/writable:532

 [C++]:
   ticks  total  nonlib   name

 [Summary]:
   ticks  total  nonlib   name
       4    0.0%  100.0%  JavaScript
       0    0.0%    0.0%  C++
       7    0.1%  175.0%  GC
   10118  100.0%          Shared libraries

 [C++ entry points]:
   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

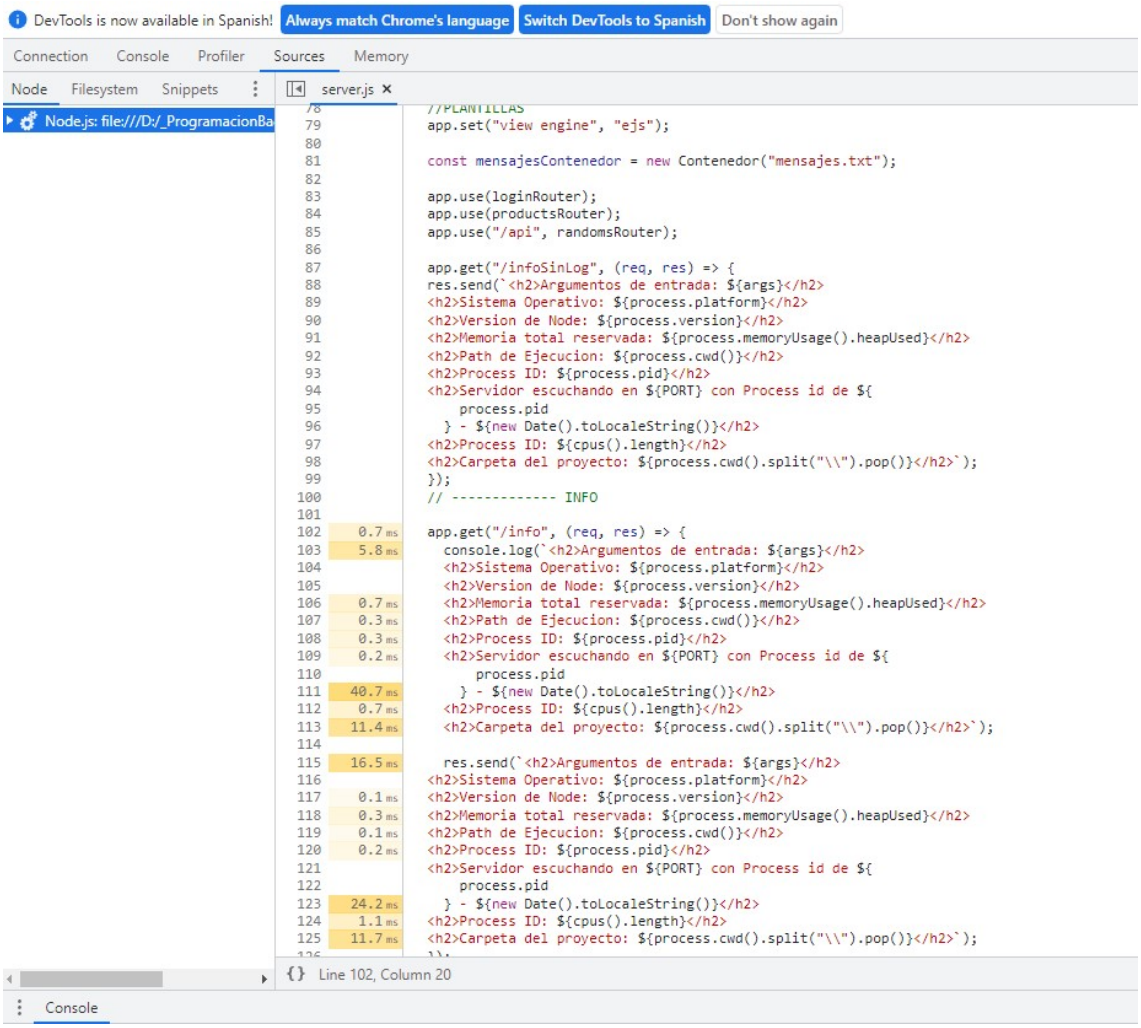
Stat	2.5%	50%	97.5%	99%	Avg	Stdev	Max
Latency	6 ms	81 ms	106 ms	114 ms	74.38 ms	24.95 ms	163 ms

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

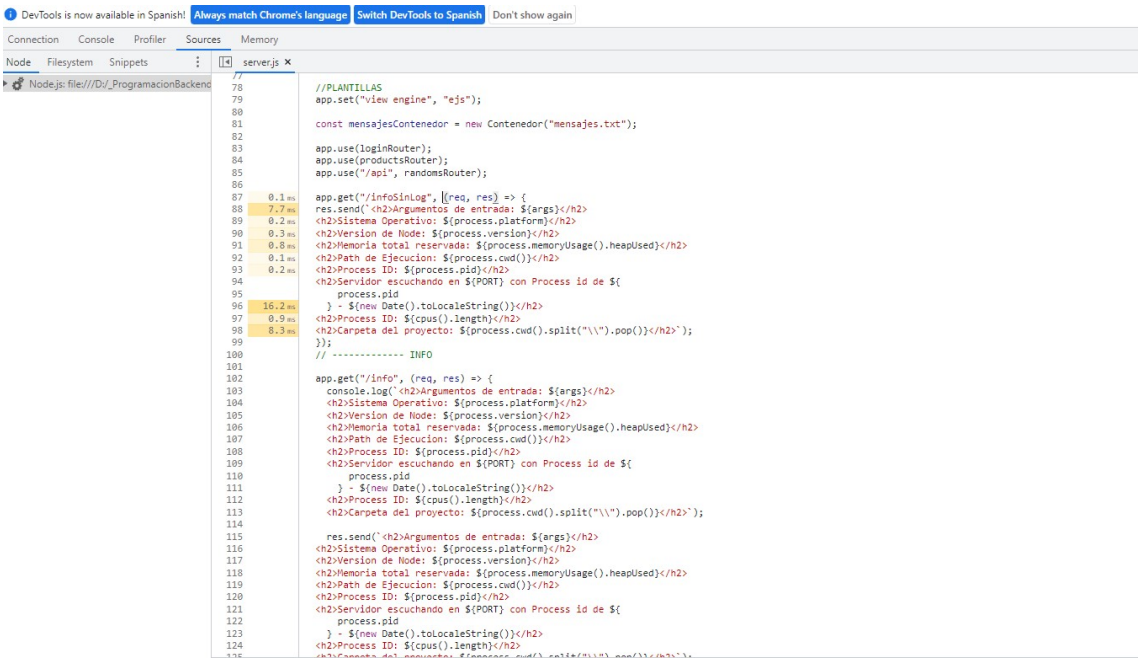
Req/Bytes counts sampled once per second.
# of samples: 20

27k requests in 20.03s, 17.6 MB read

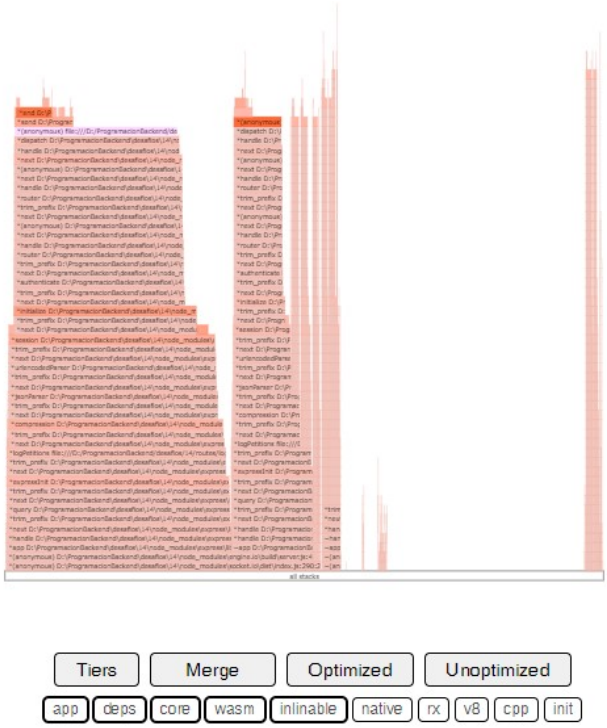
1) Inspect Bloqueante



2) Inspect No Bloqueante



1) Grafico de flama con procesos no bloqueantes



2) Grafico de Flama con procesos bloqueantes

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 bloquente /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.