

Para ambas condiciones (con o sin console.log) en la ruta '/info'
OBTENER:

1) El perfilamiento del servidor, realizando el test con --prof de node.js.
Analizar los resultados obtenidos luego de procesarlos con
--prof-process.

CON CLG:

```
1 Statistical profiling result from profResultCLG-v8.log, (10074 ticks, 18 unaccounted, 0 excluded).
2
3 [Shared libraries]:
4 ticks total nonlib name
5 7578 75.2% /usr/bin/node
6 133 1.3% /usr/lib/x86_64-linux-gnu/libc-2.31.so
7 5 0.0% /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.28
8 4 0.0% [vdso]
```

```
[Summary]:
ticks total nonlib name
1008 10.0% 42.8% JavaScript
1328 13.2% 56.4% C++
356 3.5% 15.1% GC
7720 76.6% Shared libraries
18 0.2% Unaccounted
```

SIN CLG:

```
~/Desktop/Coder/clase24/v3/coderhouse-desafio16/p
Statistical profiling result from profResultNoCLG-v8.log, (11425 ticks, 15 unaccounted, 0 excluded).

[Shared libraries]:
ticks total nonlib name
8732 76.4% /usr/bin/node
170 1.5% /usr/lib/x86_64-linux-gnu/libc-2.31.so
13 0.1% /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.28
2 0.0% [vdso]
```

```
[Summary]:
ticks total nonlib name
1249 10.9% 49.8% JavaScript
1244 10.9% 49.6% C++
461 4.0% 18.4% GC
8917 78.0% Shared libraries
15 0.1% Unaccounted
```

Utilizaremos como test de carga Artillery en línea de comandos, emulando 50 conexiones concurrentes con 20 request por cada una. Extraer un reporte con los resultados en archivo de texto.

CON CLG:

[artilleryResultCLG](#)

SIN CLG:

[artilleryResultNoCLG](#)

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)

CON CLG:

```
> autocannon -c 100 -d 20 http://localhost:8080/info
Running 20s test @ http://localhost:8080/info
100 connections
```

Stat	2.5%	50%	97.5%	99%	Avg	Stdev	Max
Latency	670 ms	1039 ms	3407 ms	3928 ms	1251.19 ms	595.54 ms	4201 ms

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	1	1	90	123	78.75	32.34	1
Bytes/Sec	4.24 kB	4.24 kB	381 kB	521 kB	333 kB	137 kB	4.23 kB

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

2k requests in 20.04s, 6.67 MB read

SIN CLG:

```
> autocannon -c 100 -d 20 http://localhost:8080/info
Running 20s test @ http://localhost:8080/info
100 connections
```

Stat	2.5%	50%	97.5%	99%	Avg	Stdev	Max
Latency	725 ms	1036 ms	3179 ms	3642 ms	1224.8 ms	537.05 ms	3882 ms

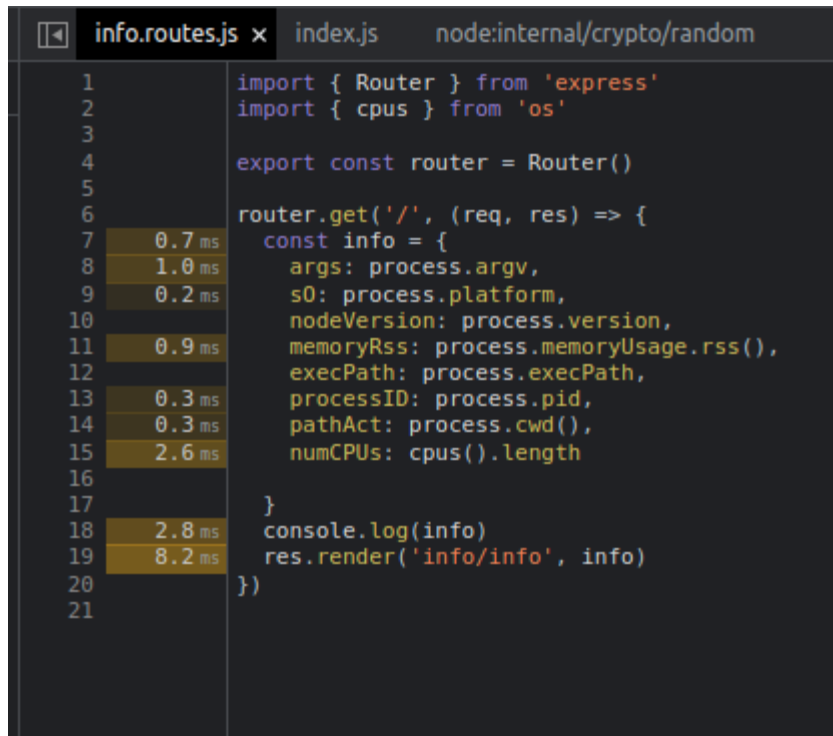
Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	1	1	82	128	79.8	33.07	1
Bytes/Sec	4.24 kB	4.24 kB	347 kB	542 kB	338 kB	140 kB	4.24 kB

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

2k requests in 20.04s, 6.76 MB read

2) El perfilamiento del servidor con el modo inspector de node.js --inspect. Revisar el tiempo de los procesos menos performantes sobre el archivo fuente de inspección.

CON CLG:

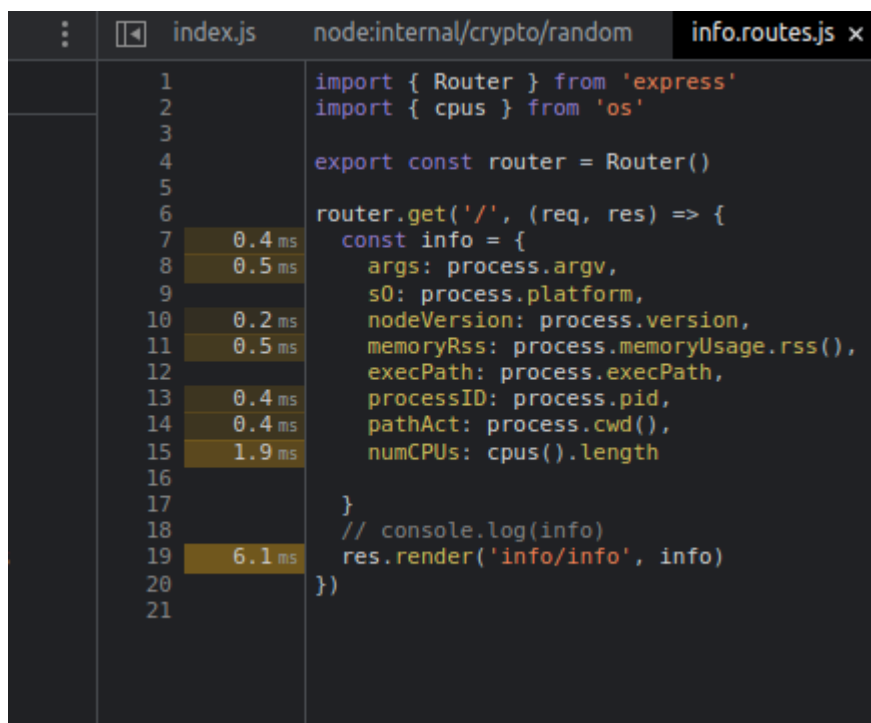


```
1 import { Router } from 'express'
2 import { cpus } from 'os'
3
4 export const router = Router()
5
6 router.get('/', (req, res) => {
7   const info = {
8     args: process.argv,
9     s0: process.platform,
10    nodeVersion: process.version,
11    memoryRss: process.memoryUsage.rss(),
12    execPath: process.execPath,
13    processID: process.pid,
14    pathAct: process.cwd(),
15    numCPUs: cpus().length
16  }
17  console.log(info)
18  res.render('info/info', info)
19 })
20
21
```

Performance profile for 'info.routes.js' (CLG):

Line	Time (ms)
7	0.7
8	1.0
9	0.2
11	0.9
13	0.3
14	0.3
15	2.6
18	2.8
19	8.2

SIN CLG:

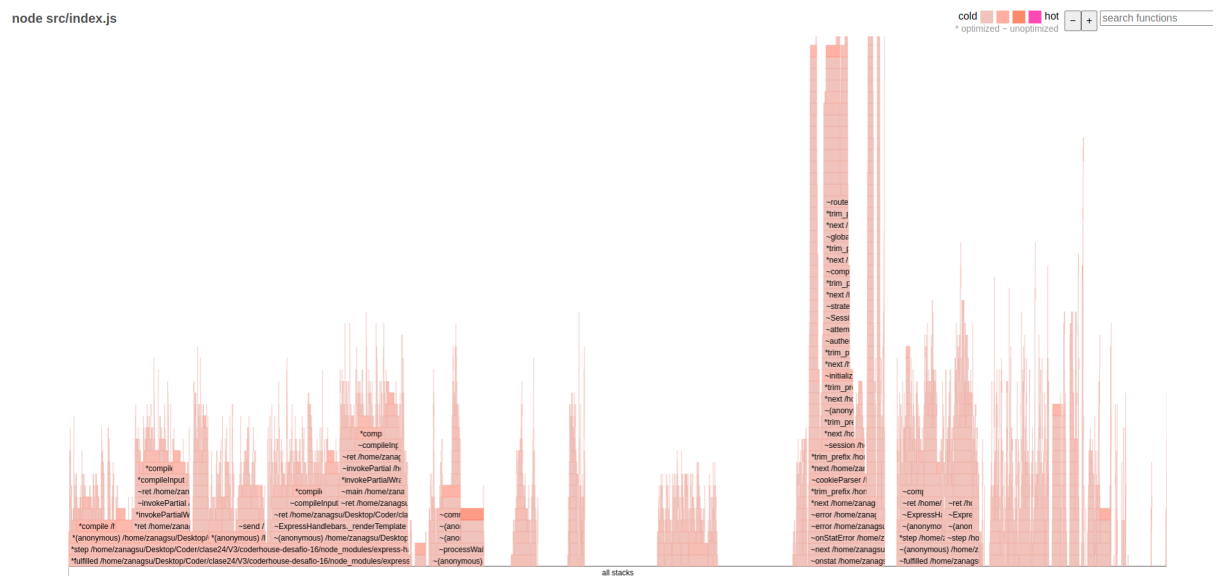


```
1 import { Router } from 'express'
2 import { cpus } from 'os'
3
4 export const router = Router()
5
6 router.get('/', (req, res) => {
7   const info = {
8     args: process.argv,
9     s0: process.platform,
10    nodeVersion: process.version,
11    memoryRss: process.memoryUsage.rss(),
12    execPath: process.execPath,
13    processID: process.pid,
14    pathAct: process.cwd(),
15    numCPUs: cpus().length
16  }
17  // console.log(info)
18  res.render('info/info', info)
19 })
20
21
```

Performance profile for 'info.routes.js' (SIN CLG):

Line	Time (ms)
7	0.4
8	0.5
10	0.2
11	0.5
13	0.4
14	0.4
15	1.9
19	6.1

CON CLG:



Mi conclusión es que con estas herramientas podemos evidenciar cómo afectan los procesos bloqueantes y no bloqueantes en nuestra aplicación, en este caso aplicando y removiendo un `console.log()` podemos notar cambios, se hace más visible en la captura del `-inspect` como demora al tener el `clg`.

Todos los archivos van a estar dentro de la carpeta `profiling` subidos al repo de github.