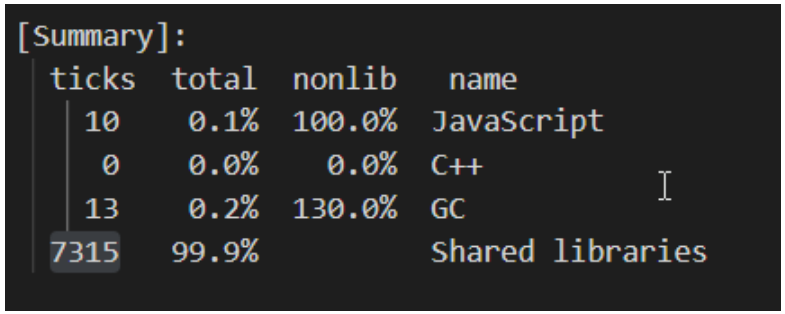
Análisis de Performance

Análisis sobre la ruta '**/info**' en modo fork, agregando o extrayendo un console.log

1)

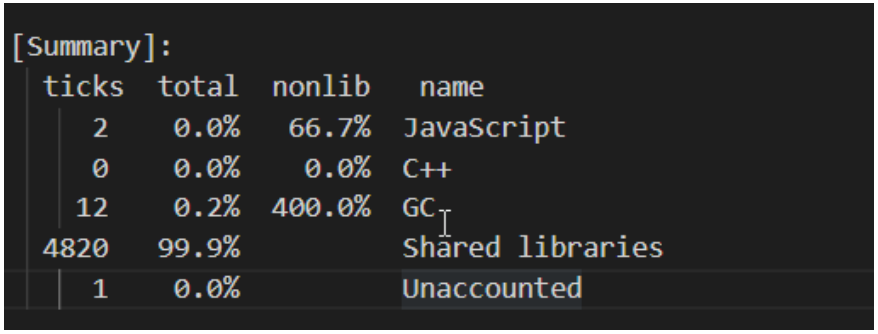
- Perfilamiento del servidor con ***Node built.in profiler y Artillery*** (como test de carga)

Modo Bloqueante



```
[Summary]:
  ticks  total  nonlib   name
    10    0.1%  100.0%  JavaScript
     0    0.0%    0.0%  C++
    13    0.2%  130.0%  GC
  7315   99.9%          Shared libraries
```

Modo No Bloqueante



```
[Summary]:
  ticks  total  nonlib   name
     2    0.0%   66.7%  JavaScript
     0    0.0%    0.0%  C++
    12    0.2%  400.0%  GC
  4820   99.9%          Shared libraries
     1    0.0%          Unaccounted
```

- Perfilamiento del servidor con ***Autocannon*** (como test de carga)

Modo Bloqueante

```
Running 20s test @ http://localhost:8080/info
100 connections
```

Stat	2.5%	50%	97.5%	99%	Avg	Stdev	Max
Latency	43 ms	58 ms	106 ms	120 ms	61.77 ms	15.06 ms	138 ms

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	902	902	1693	1803	1605.45	238.85	902
Bytes/Sec	411 kB	411 kB	771 kB	821 kB	730 kB	109 kB	410 kB

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

32k requests in 20.06s, 14.6 MB read
```

Modo No Bloqueante

```
Running 20s test @ http://localhost:8080/info
100 connections
```

Stat	2.5%	50%	97.5%	99%	Avg	Stdev	Max
Latency	31 ms	40 ms	114 ms	130 ms	47.67 ms	20.54 ms	178 ms

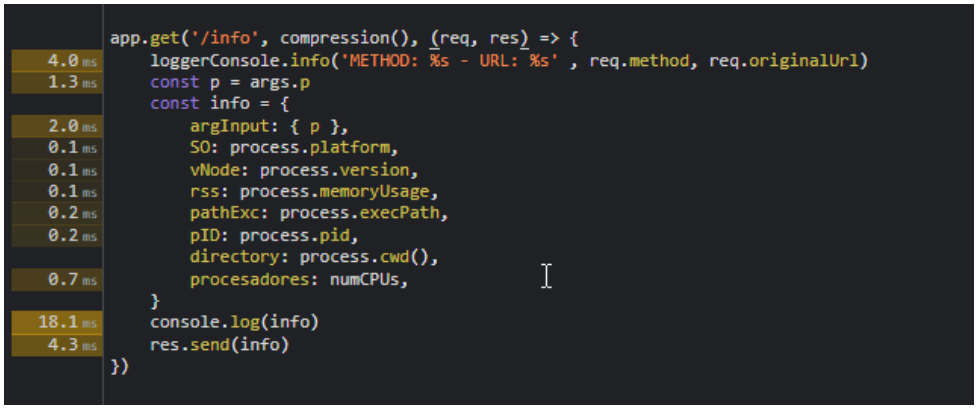
Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	804	804	2119	2541	2072.06	513.71	804
Bytes/Sec	366 kB	366 kB	965 kB	1.16 MB	943 kB	234 kB	366 kB

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

42k requests in 20.07s, 18.9 MB read
```

2) Perfilamiento del servidor con **Node Inspect**

Modo Bloqueante



```
app.get('/info', compression(), (req, res) => {
    loggerConsole.info('METHOD: %s - URL: %s' , req.method, req.originalUrl)
    const p = args.p
    const info = {
        argInput: { p },
        SO: process.platform,
        vNode: process.version,
        rss: process.memoryUsage,
        pathExc: process.execPath,
        pID: process.pid,
        directory: process.cwd(),
        procesadores: numCPUs,
    }
    console.log(info)
    res.send(info)
})
```

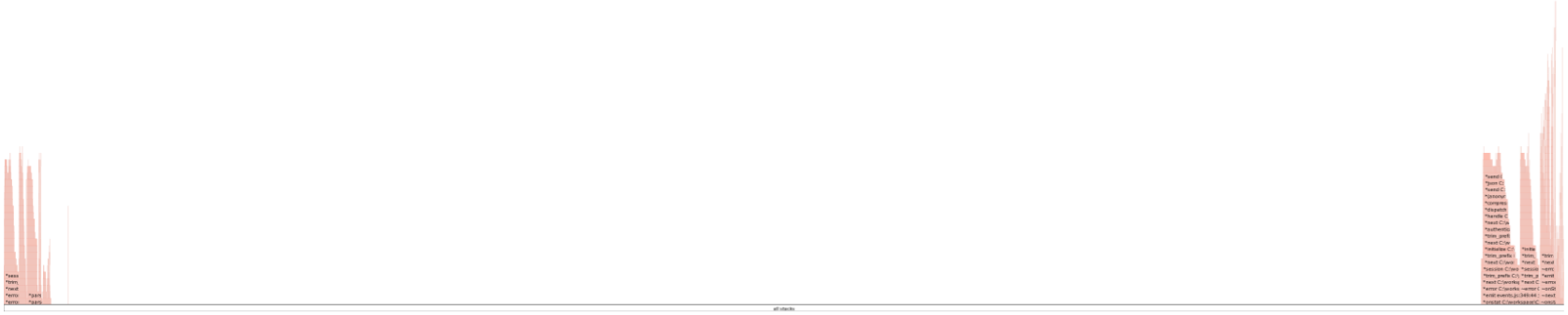
Modo No Bloqueante

```
app.get('/info', compression(), (req, res) => {
    loggerConsole.info('METHOD: %s - URL: %s' , req.method, req.originalUrl)
    const p = args.p
    const info = {
        argInput: { p },
        SO: process.platform,
        vNode: process.version,
        rss: process.memoryUsage,
        pathExc: process.execPath,
        pID: process.pid,
        directory: process.cwd(),
        procesadores: numCPUs,
    }
    // console.log(info)
    res.send(info)
})
```

3) Diagrama de flama con **0x** , utilizando **Autocannon** para test de carga

Modo Bloqueante

Modo No Bloqueante



Conclusión

Realizando los test con las herramientas de carga Autocannon y Artillery, lo que se demuestra es la alta lentitud o latencia en los tiempos de respuesta en los casos de funciones síncronas, mientras que los casos de no uso de estas, los tiempos bajan la mitad, la utilización de los recursos es más eficiente y las respuestas a las diferentes solicitudes son más rápidas.

En el caso de servidores con mejores recursos, en cuanto a procesadores, se puede sacar mejor provecho de esto, utilizando los que se necesitan (para no sobrecargar a los procesos) y así brindar una mejor respuesta a los usuarios de la app y también mejorar la interacción entre las piezas de software.