Verificar sobre la ruta /info con y sin compresión, la diferencia de cantidad de bytes devueltos en un caso y otro.

Sin compresión:

Name	Type	Size
info	document	4.5 kB

```
router.get('/info', (req,res)=>{
    const info = {
        inputArguments: JSON.stringify(args),
        cpuNumber:      os.cpus().length,
        platformName:   process.platform,
        versionNode:    process.version,
        rss:            process.memoryUsage().rss,
        path:           process.argv[0],
        processId:      process.pid,
        projectFolder:  `${process.cwd()}`
    }
    res.render('index', {info})
});
```

Con compresión:

Name	Type	Size
info	document	1.7 kB

```
router.get('/info', compression(),(req,res)=>{
    const info = {
        inputArguments: JSON.stringify(args),
        cpuNumber:      os.cpus().length,
        platformName:   process.platform,
        versionNode:    process.version,
        rss:            process.memoryUsage().rss,
        path:           process.argv[0],
        processId:      process.pid,
        projectFolder:  `${process.cwd()}`
    }
    res.render('index', {info})
});
```

1) --prof

node_info.txt

```
--------------------------------
Summary report @ 17:33:38(-0300)
--------------------------------

http.codes.200: ..................................................................... 1000
http.request_rate: .................................................................. 116/sec
http.requests: ...................................................................... 1000
http.response_time:
  min: .............................................................................. 15
  max: .............................................................................. 279
  median: ........................................................................... 190.6
  p95: .............................................................................. 237.5
  p99: .............................................................................. 257.3
http.responses: ..................................................................... 1000
vusers.completed: ................................................................... 50
vusers.created: ..................................................................... 50
vusers.created_by_name.0: ........................................................... 50
vusers.failed: ...................................................................... 0
vusers.session_length:
  min: .............................................................................. 2753.2
  max: .............................................................................. 3779.1
  median: ........................................................................... 3605.5
  p95: .............................................................................. 3752.7
  p99: .............................................................................. 3752.7
```

node_prof.txt

```
[Summary]:
   ticks  total  nonlib   name
     276    1.7%   99.3%  JavaScript
       0    0.0%    0.0%  C++
     237    1.5%   85.3%  GC
   15893   98.3%          Shared libraries
       2    0.0%          Unaccounted
```

Autocannon

2) --inspect

Node .\benchmark.js

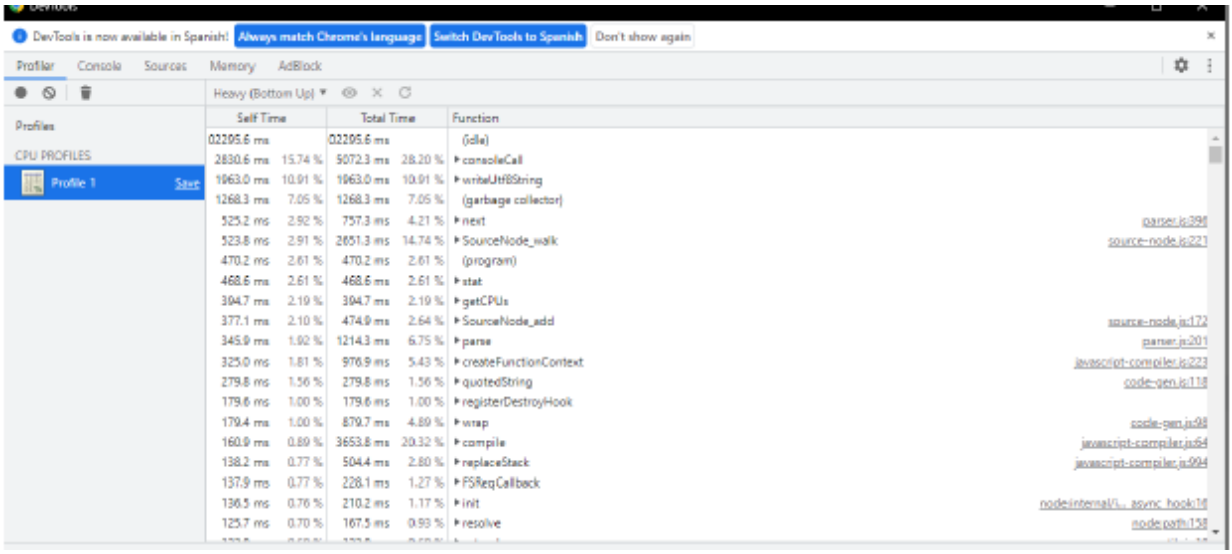
```
Running benchmarks in Parallel
Running 20s test @ http://localhost:8000/info
100 connections


  Stat      2.5%     50%      97.5%    99%      Avg         Stdev       Max
  Latency   463 ms   522 ms   704 ms   742 ms   543.48 ms   67.84 ms    721 ms

  Stat        1%       2.5%     50%      97.5%    Avg      Stdev    Min
  Req/Sec     100      100      108      214      101.9    27.44    100
  Bytes/Sec   457 kB   457 kB   859 kB   978 kB   831 kB   125 kB   457 kB

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

4k requests in 20.05s, 16.6 MB read
```

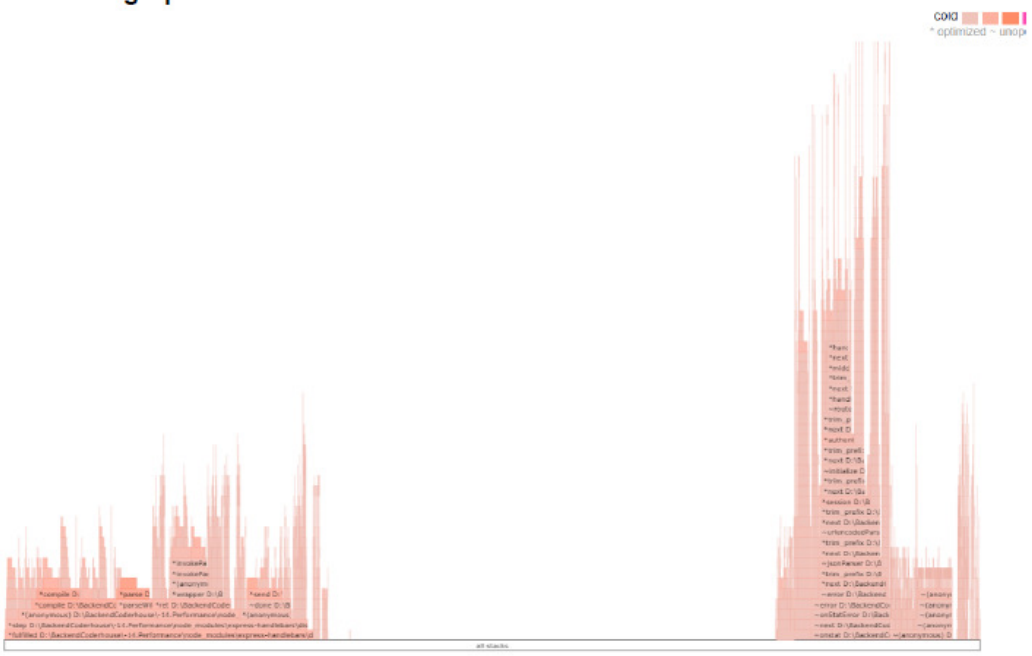


```
0.1 ms  router.get('/info', compression(),(req,res)=>{
0.4 ms      const info = {
16.2 ms         inputArguments: JSON.stringify(args),
2.0 ms          cpuNumber:      os.cpus().length,
2.6 ms          platformName:   process.platform,
0.1 ms          versionNode:    process.version,
1.5 ms          rss:            process.memoryUsage().rss,
1.3 ms          path:           process.argv[0],
0.2 ms          processId:      process.pid,
0.4 ms          projectFolder:  `${process.cwd()}`
        }
9.0 ms      console.log(info)

14.6 ms     res.render('index', {info})
    });
    router.get('*', (req, res)=>{
        const router = req.url;
        const method = req.method;
        warnLogger.warn(`Route: ${router}. Method: ${method}`);
        res.send('no bueno. Mal ahí: 404', 404);
      });

    module.exports = router
```

3) 0x
/2232.0x/flamegraph.html



Se observan que los logs en consola afectan al performance y tambien la funcion Next() de Logger que incorporé en el middleware.

Ya que este se ejecuta en cada peticion a la pagina.

Tambien hay mucho procesamiento en los RES.JSON.

Tambien puede ser que consuma mucho los proccess para conseguir info(cpus, platform)