

DESAFÍO

LOGGERS Y GZIP

ANÁLISIS COMPLETO DE

PERFORMANCE

CURSO BACKEND – COMISIÓN 32080 CLASE 31 Y 32

VALENTÍN ROMERO

Gzip

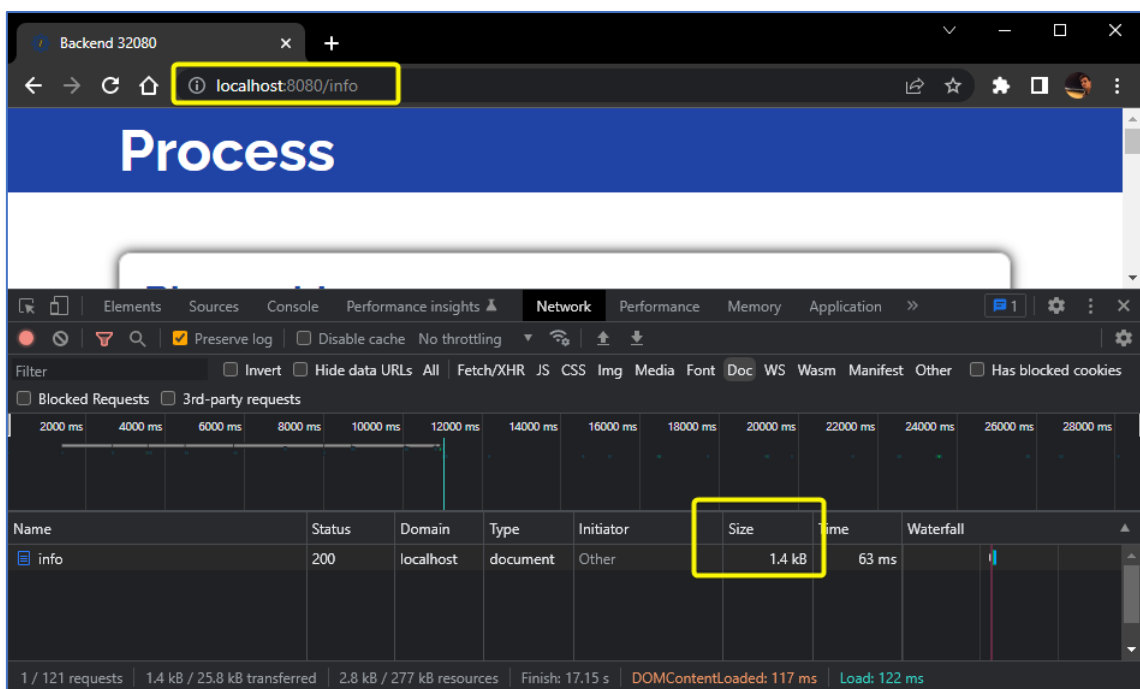
Verificar sobre la ruta /info con y sin compresión, la diferencia de cantidad de bytes devueltos en un caso y otro

- Prueba sobre la ruta /info con compresión:

Código con middleware de compression:

```
server.js x
desafio > Backend > server.js > ...
27 const compression = require('compression')
28 const { infoLogger, warnlogger, errorLogger } = require('./utils/winstonLogger')
29
30 const app = express()
31 const httpServer = http.createServer(app)
32 const io = new Server(httpServer, {})
33 //console.log('Modo: ', MODE.toUpperCase())
34 infoLogger.info(`MODULO: ${MODE.toUpperCase()}`)
35
36 app.use(compression())
37
38 app.use('/public', express.static(__dirname + '/public'))
39
```

Resultado sobre la ruta /info:



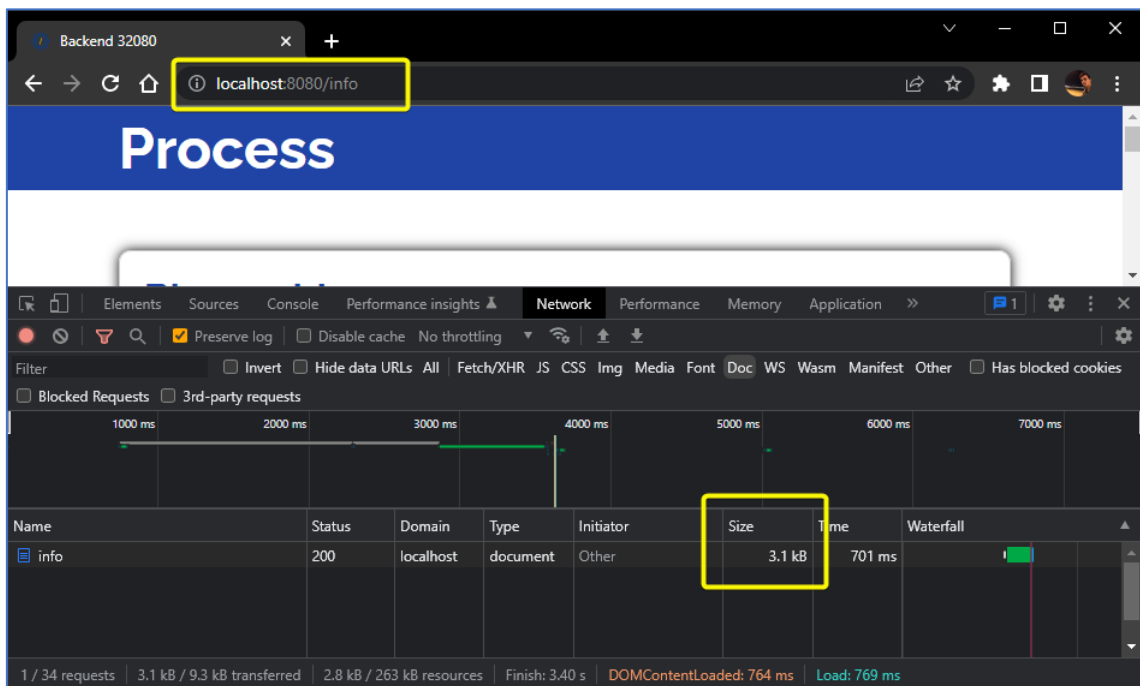
La cantidad de kb devueltos es de 1.4

b. Prueba sobre la ruta /info sin compresión:

Código con middleware de compression comentado

```
server.js M x
desafio > Backend > JS server.js > ...
27 const compression = require('compression')
28 const { infoLogger, warnlogger, errorLogger } = require('./utils/winstonLogger')
29
30 const app = express()
31 const httpServer = http.createServer(app)
32 const io = new Server(httpServer, {})
33 //console.log('Modo: ', MODE.toUpperCase())
34 infoLogger.info(`MODULO: ${MODE.toUpperCase()}`)
35
36 //app.use(compression())
37
38 app.use('/public', express.static(__dirname + '/public'))
39
```

Resultado sobre la ruta /info



La cantidad de kb devueltos en este caso es de 3.1

CONCLUSIÓN:

Para ambos escenarios donde se devuelve la misma información, cuando se utiliza compression, el peso de esta se reduce a más de la mitad.

Análisis de performance

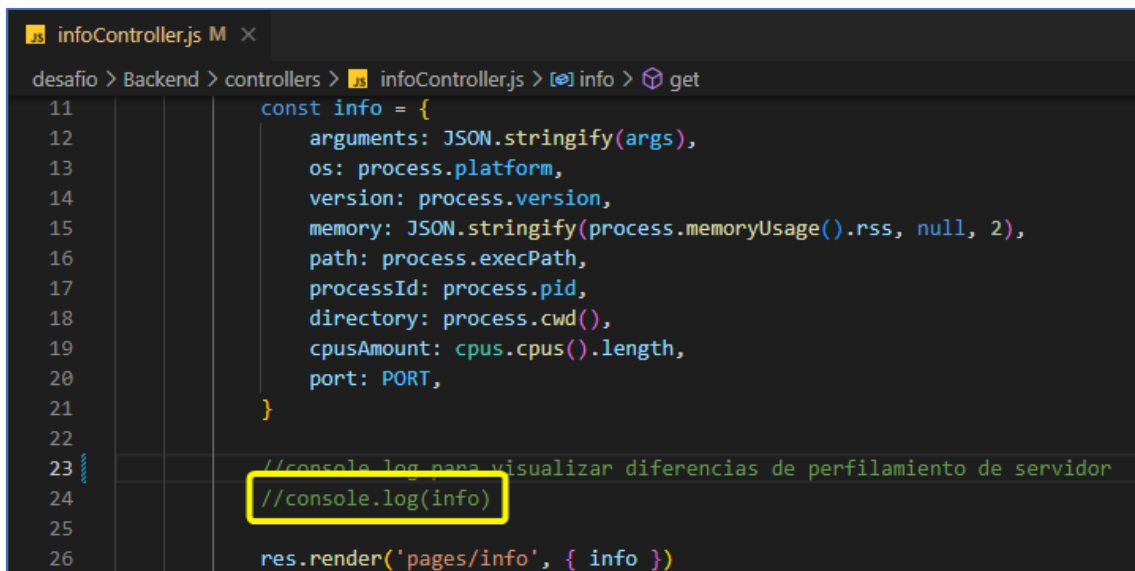
Vamos a trabajar sobre la ruta '/info', en modo fork, agregando o extrayendo un console.log de la información colectada antes de devolverla al cliente.

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`.

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.

- a. Prueba con `console.log(info)` comentado desde el controlador `infoController.js`, que trabaja para la vista correspondiente a la ruta `/info`



```
11 const info = {
12   arguments: JSON.stringify(args),
13   os: process.platform,
14   version: process.version,
15   memory: JSON.stringify(process.memoryUsage().rss, null, 2),
16   path: process.execPath,
17   processId: process.pid,
18   directory: process.cwd(),
19   cpusAmount: cpus.cpus().length,
20   port: PORT,
21 }
22
23 //console.log para visualizar diferencias de perfilamiento de servidor
24 //console.log(info)
25
26 res.render('pages/info', { info })
```

Resultado del perfilamiento del servidor realizado con --prof

```
result_sin_clg.txt M X
desafio > Backend > artillery > result_sin_clg.txt
1 Running scenarios...
2 Phase started: unnamed (index: 0, duration: 1s) 19:56:58(-0300)
3
4 Phase completed: unnamed (index: 0, duration: 1s) 19:56:59(-0300)
5
6 All VUs finished. Total time: 10 seconds
7
8 -----
9 Summary report @ 19:57:05(-0300)
10 -----
11
12 http.codes.200: ..... 1000
13 http.request_rate: ..... 100/sec
14 http.requests: ..... 1000
15 http.response_time:
16   min: ..... 6
17   max: ..... 194
18   median: ..... 74.4
19   p95: ..... 159.2
20   p99: ..... 183.1
```

Resultados obtenidos luego de procesarlos con --prof-process

```
result_v8_sin.clg.txt U X
desafio > Backend > artillery > result_v8_sin.clg.txt
62 [Bottom up (heavy) profile]:
63 Note: percentage shows a share of a particular caller in the total
64 amount of its parent calls.
65 Callers occupying less than 1.0% are not shown.
66
67 ticks parent name
68 2381 77.3% C:\Windows\SYSTEM32\ntdll.dll
69 56 2.4% D:\Program Files\nodejs\node.exe
70 3 5.4% D:\Program Files\nodejs\node.exe
```

b. Prueba con console.log(info) descomentado

```

infoController.js M x
desafio > Backend > controllers > infoController.js > info > get
11 const info = {
12   arguments: JSON.stringify(args),
13   os: process.platform,
14   version: process.version,
15   memory: JSON.stringify(process.memoryUsage().rss, null, 2),
16   path: process.execPath,
17   processId: process.pid,
18   directory: process.cwd(),
19   cpusAmount: cpus.cpus().length,
20   port: PORT,
21 }
22
23 //console.log para visualizar diferencias de perfilamiento de servidor
24 console.log(info)
25
26 res.render('pages/info', { info })
    
```

Resultado del perfilamiento del servidor realizado con --prof

```

result_con_clg.txt M x
desafio > Backend > artillery > result_con_clg.txt
1 Running scenarios...
2 Phase started: unnamed (index: 0, duration: 1s) 19:55:47(-0300)
3
4 Phase completed: unnamed (index: 0, duration: 1s) 19:55:48(-0300)
5
6 All VUs finished. Total time: 15 seconds
7
8 -----
9 Summary report @ 19:55:59(-0300)
10 -----
11
12 http.codes.200: ..... 1000
13 http.request_rate: ..... 68/sec
14 http.requests: ..... 1000
15 http.response_time:
16   min: ..... 6
17   max: ..... 258
18   median: ..... 125.2
19   p95: ..... 198.4
20   p99: ..... 247.2
    
```

Resultados obtenidos luego de procesarlos con --prof-process

```

result_v8_con_clg.txt U x
desafio > Backend > artillery > result_v8_con_clg.txt
61 note: percentage shows a share of a particular caller in the total
62 amount of its parent calls.
63 Callers occupying less than 1.0% are not shown.
64
65 ticks parent name
66 5818 86.6% C:\Windows\SYSTEM32\ntdll.dll
67
68 838 12.5% D:\Program Files\nodejs\node.exe
69 584 69.7% D:\Program Files\nodejs\node.exe
70 133 22.8% Function: ^handleWriteReq node:internal/stream_base_commons:45:24
    
```

ANÁLISIS Y CONCLUSIÓN:

El software se nota notoriamente (a nivel de análisis no de apariencia al usuario) menos performante cuando involucra procesos bloqueantes, comprado con los casos en los que esto no sucede:

- Para el caso en el cual la información trabajada por el controlador se envía por consola (generando un proceso bloqueante) se generan más del doble de ticks
- A su vez en este escenario, el tiempo de respuesta es poco más del doble, comparado al proceso en el cual no se ejecuta el "console.log(info)"
- También se observa que en este caso se realizan 68 solicitudes HTTP por segundo, mientras que en el caso contrario se alcanza a 100 solicitudes por segundo.

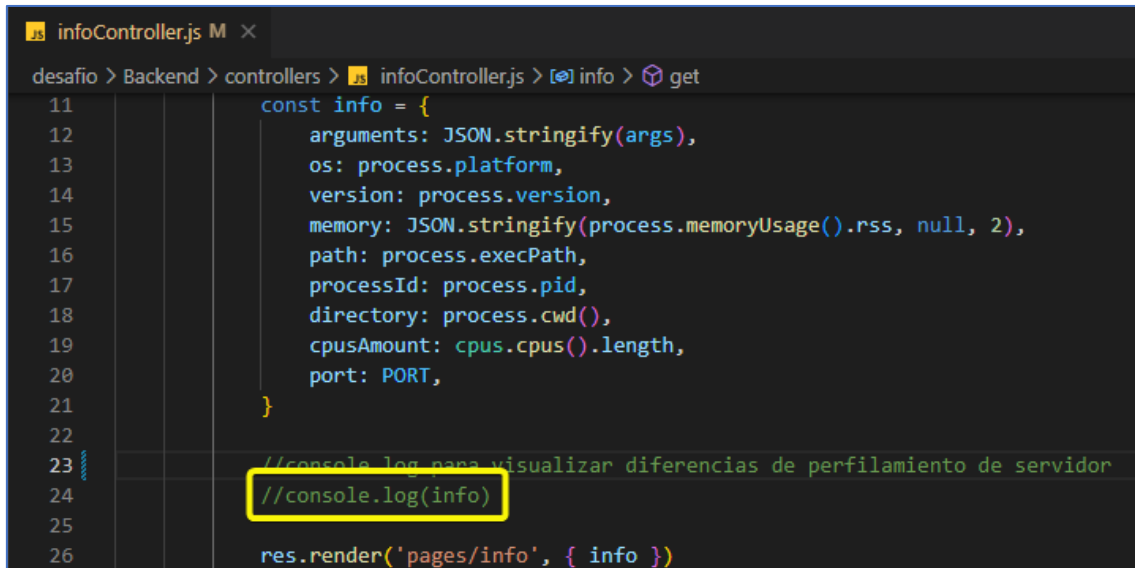
Autocannon y 0x

Luego utilizaremos Autocannon en línea de comandos, emulando 100 conexiones concurrentes realizadas en un tiempo de 20 segundos.

Archivo benchmark.js:

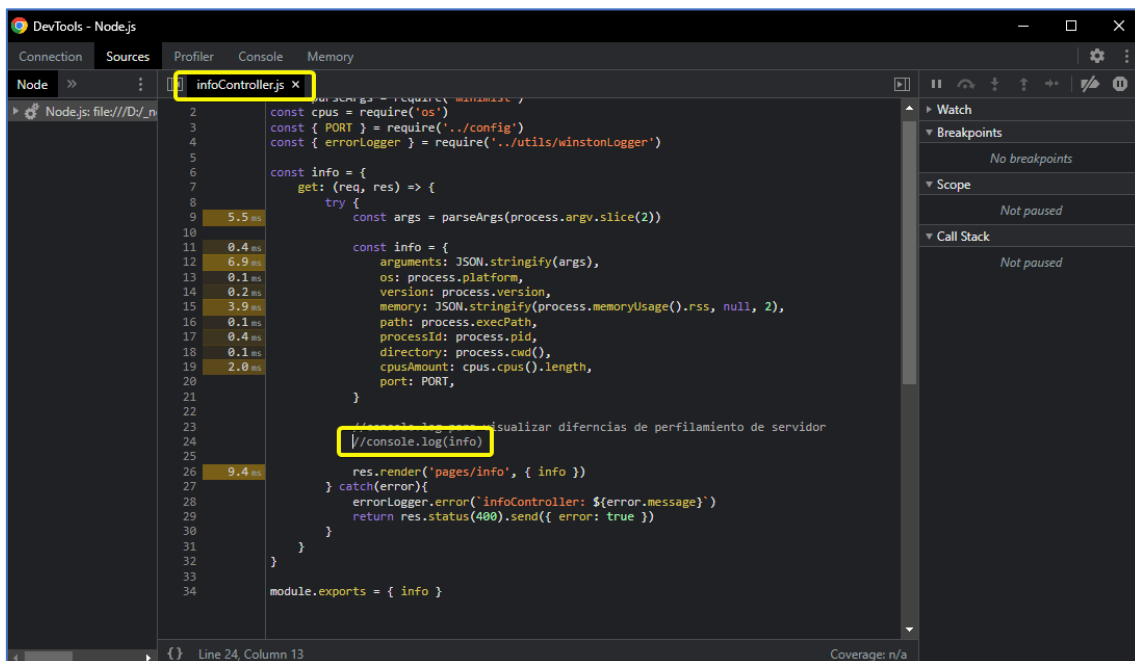
```
JS benchmark.js U x
desafio > Backend > JS benchmark.js > ...
1  const autocannon = require('autocannon')
2  const { PassThrough } = require('stream')
3
4  function run(url) {
5      const buf = []
6      const outputStream = new PassThrough();
7      const inst = autocannon({
8          url,
9          connections: 100,
10         duration: 20,
11     })
12     autocannon.track(inst, { outputStream });
13     outputStream.on('data', (data) => buf.push(data))
14     inst.on('done', function () {
15         process.stdout.write(Buffer.concat(buf))
16     })
17 }
18
19 console.log('Running all benchmarks in parallel ...')
20
21 run('http://localhost:8080/info')
```

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.
3. El diagrama de flama con 0x, emulando la carga con Autocannon con los mismos parámetros anteriores.
 - a. Prueba con console.log(info) comentado desde el controlador infoController.js:



```
desafío > Backend > controllers > infoController.js > info > get
11 const info = {
12   arguments: JSON.stringify(args),
13   os: process.platform,
14   version: process.version,
15   memory: JSON.stringify(process.memoryUsage().rss, null, 2),
16   path: process.execPath,
17   processId: process.pid,
18   directory: process.cwd(),
19   cpusAmount: cpus.cpus().length,
20   port: PORT,
21 }
22
23 //console.log para visualizar diferencias de perfilamiento de servidor
24 //console.log(info)
25
26 res.render('pages/info', { info })
```

Resultado desde la herramienta Node inspect del navegador:



```
Node.js: file:///D:/n...
2 const cpus = require('os')
3 const { PORT } = require('../config')
4 const { errorLogger } = require('../utils/winstonLogger')
5
6 const info = {
7   get: (req, res) => {
8     try {
9       const args = parseArgs(process.argv.slice(2))
10
11       const info = {
12         arguments: JSON.stringify(args),
13         os: process.platform,
14         version: process.version,
15         memory: JSON.stringify(process.memoryUsage().rss, null, 2),
16         path: process.execPath,
17         processId: process.pid,
18         directory: process.cwd(),
19         cpusAmount: cpus.cpus().length,
20         port: PORT,
21       }
22
23       //console.log para visualizar diferencias de perfilamiento de servidor
24       //console.log(info)
25
26       res.render('pages/info', { info })
27     } catch (error) {
28       errorLogger.error('infoController: ${error.message}')
29       return res.status(400).send({ error: true })
30     }
31   }
32 }
33
34 module.exports = { info }
```


Ejecutando desde consola externa, el archivo benchmark.js (npm test):

```
C:\Windows\system32\cmd.exe
D:\nginx-1.23.2\desafio\Backend>npm test
> desafio@1.0.0 test
> node benchmark.js

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

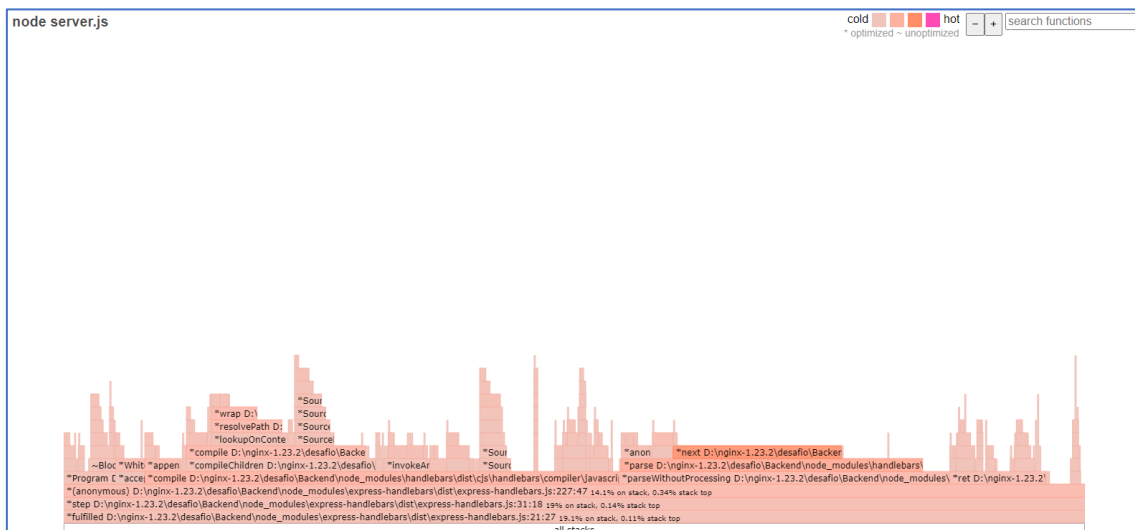
Stat	2.5%	50%	97.5%	99%	Avg	Stdev	Max
Latency	244 ms	282 ms	429 ms	615 ms	294.94 ms	56.22 ms	674 ms

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	162	162	355	393	338.55	56.27	162
Bytes/Sec	488 kB	488 kB	1.07 MB	1.18 MB	1.02 MB	169 kB	488 kB

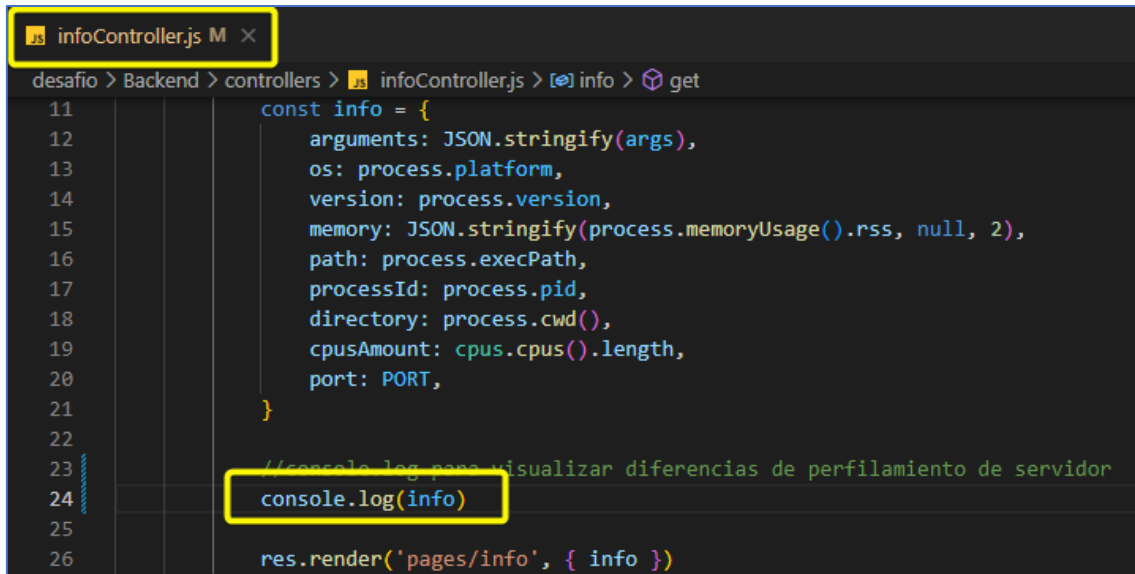
```
Req/Bytes counts sampled once per second.
# of samples: 20

7k requests in 20.17s, 20.4 MB read
D:\nginx-1.23.2\desafio\Backend>
```

Diagrama de flama:

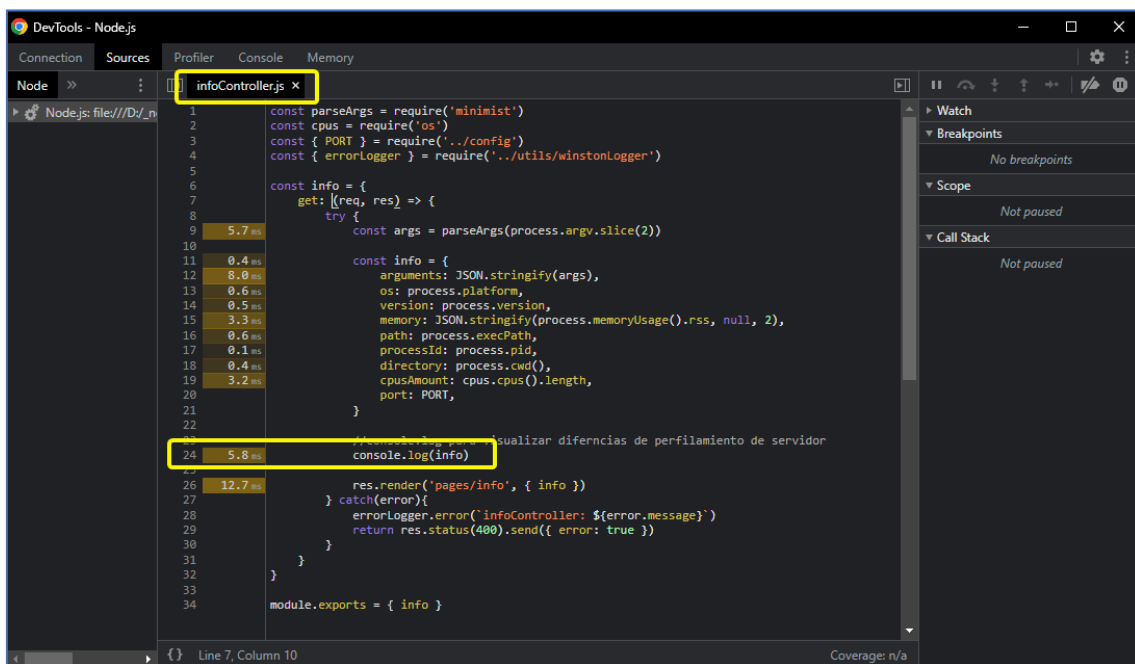


b. Prueba con console.log(info) descomentado:



```
11 const info = {
12   arguments: JSON.stringify(args),
13   os: process.platform,
14   version: process.version,
15   memory: JSON.stringify(process.memoryUsage().rss, null, 2),
16   path: process.execPath,
17   processId: process.pid,
18   directory: process.cwd(),
19   cpusAmount: cpus.cpus().length,
20   port: PORT,
21 }
22
23 //console.log para visualizar diferencias de perfilamiento de servidor
24 console.log(info)
25
26 res.render('pages/info', { info })
```

Resultado desde la herramienta Node inspect del navegador:



```
1 const parseArgs = require('minimist')
2 const cpus = require('os')
3 const { PORT } = require('../config')
4 const { errorLogger } = require('../utils/winstonLogger')
5
6 const info = {
7   get: (req, res) => {
8     try {
9       const args = parseArgs(process.argv.slice(2))
10
11       const info = {
12         arguments: JSON.stringify(args),
13         os: process.platform,
14         version: process.version,
15         memory: JSON.stringify(process.memoryUsage().rss, null, 2),
16         path: process.execPath,
17         processId: process.pid,
18         directory: process.cwd(),
19         cpusAmount: cpus.cpus().length,
20         port: PORT,
21       }
22
23       //console.log para visualizar diferencias de perfilamiento de servidor
24       console.log(info)
25
26       res.render('pages/info', { info })
27     } catch (error) {
28       errorLogger.error(`infoController: ${error.message}`)
29       return res.status(400).send({ error: true })
30     }
31   }
32 }
33
34 module.exports = { info }
```

Ejecutando desde consola externa, el archivo benchmark.js (npm test):

```
C:\Windows\system32\cmd.exe
D:\nginx-1.23.2\desafio\Backend>npm test
> desafio@1.0.0 test
> node benchmark.js

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

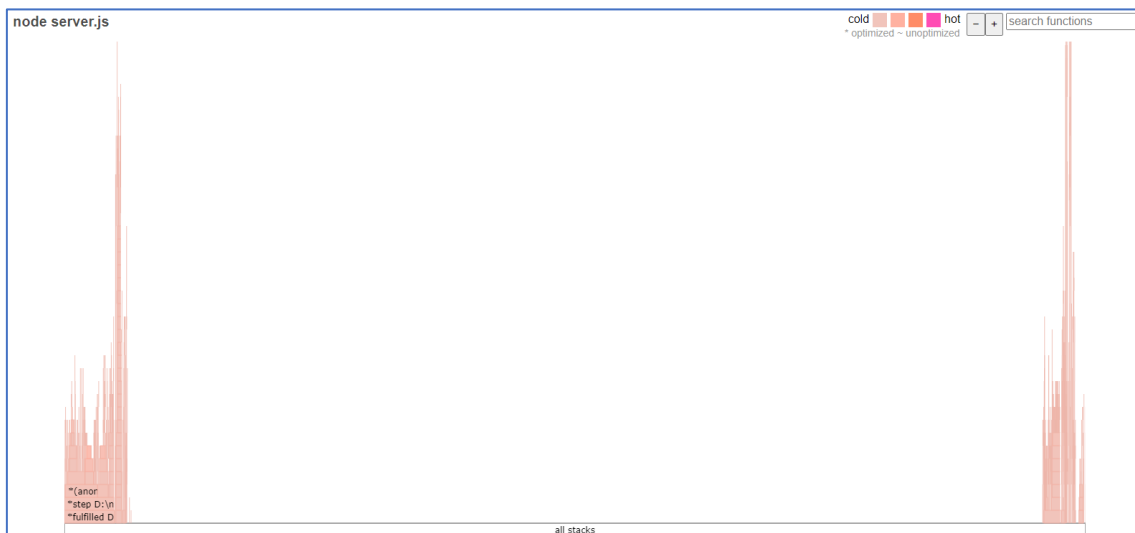
Stat    2.5%    50%    97.5%    99%    Avg    Stdev    Max
Latency 348 ms   394 ms  602 ms  833 ms  416.14 ms  78.57 ms  853 ms

Stat    1%     2.5%    50%    97.5%    Avg    Stdev    Min
Req/Sec 100     100     250    288     239.3   43.58    100
Bytes/Sec 301 kB  301 kB  753 kB  867 kB  720 kB  131 kB   301 kB

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

5k requests in 20.2s, 14.4 MB read
D:\nginx-1.23.2\desafio\Backend>
```

Diagrama de flama:



CONCLUSIÓN:

- El proceso bloqueante del “console.log(info)” demanda 5.8 ms, por lo que perjudica la performance del programa.
- La latencia está relacionada a los console.log y demás procesos bloqueantes que ejecute el programa.