

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

a. CON COMPRESIÓN

Name	Status	Type	Initiator	Size
info	200	document	Other	1.5 kB

```
app.get('/info', compression(), (req, res) => {  
  const info = {  
    inputArguments: JSON.stringify(args.MODE),  
    cpuNum: 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 })  
  console.log(info)  
});
```

b. SIN COMPRESION

Name	Status	Type	Initiator	Size
info	200	document	Other	3.6 kB

```
app.get('/info', (req, res) => {  
  const info = {  
    inputArguments: JSON.stringify(args.MODE),  
    cpuNum: 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 })  
  console.log(info)  
});
```

2) análisis completo de performance del servidor con el que venimos trabajando.

a. CONSOLE

a.1. -- prof

```
All VUs finished. Total time: 6 seconds

-----
Summary report @ 20:55:56(-0300)
-----

http.codes.200: ..... 1000
http.request_rate: ..... 104/sec
http.requests: ..... 1000
http.response_time:
  min: ..... 26
  max: ..... 301
  median: ..... 214.9
  p95: ..... 278.7
  p99: ..... 295.9
http.responses: ..... 1000
vusers.completed: ..... 50
vusers.created: ..... 50
vusers.created_by_name.0: ..... 50
vusers.failed: ..... 0
vusers.session_length:
  min: ..... 3897.6
  max: ..... 4300.6
  median: ..... 4147.4
  p95: ..... 4231.1
  p99: ..... 4316.6
```

a.2. -- prof -- process

```
[Summary]:
  ticks  total  nonlib   name
    29    0.3%  100.0%  JavaScript
     0    0.0%   0.0%   C++
    14    0.2%  48.3%    GC
  8372   99.7%           Shared libraries
```

b. SIN CONSOLE

b.1 -- prof

```
All VUs finished. Total time: 6 seconds

-----
Summary report @ 21:01:42(-0300)
-----

http.codes.200: ..... 1000
http.request_rate: ..... 105/sec
http.requests: ..... 1000
http.response_time:
  min: ..... 27
  max: ..... 292
  median: ..... 219.2
  p95: ..... 267.8
  p99: ..... 284.3
http.responses: ..... 1000
vusers.completed: ..... 50
vusers.created: ..... 50
vusers.created_by_name.0: ..... 50
vusers.failed: ..... 0
vusers.session_length:
  min: ..... 3892.5
  max: ..... 4254.3
  median: ..... 4147.4
  p95: ..... 4231.1
  p99: ..... 4231.1
```

b.2 -- prof -- process

```
[Summary]:
  ticks  total  nonlib   name
    41     1.3%  100.0%  JavaScript
     0     0.0%   0.0%   C++
    19     0.6%   46.3%    GC
  3038    98.7%           Shared libraries
```

2) 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)

a. - -INSPECT

	56.9 ms	0.30 %	6225.0 ms	32.83 %	▼ (anonymous)
140					app.get('/info', (req, res) => {
141	1.6 ms				const info = {
142	4.5 ms				inputArguments: JSON.stringify(args.MODE),
143	4.1 ms				cpuNum: os.cpus().length,
144	4.8 ms				platformName: process.platform,
145	1.3 ms				versionNode: process.version,
146	2.3 ms				rss: process.memoryUsage().rss,
147	2.0 ms				path: process.argv[0],
148	0.4 ms				processId: process.pid,
149	0.9 ms				projectFolder: `\${process.cwd()}`
150					}
151	21.1 ms				res.render('index', { info })
152	13.6 ms				console.log(info)
153					
154					});
155					
156					app.get('*', (req, res) => {
157					const router = req.url;
158					const method = req.method;
159					warnLogger.warn(`Route: \${router}. Method: \${method}`);
160	0.1 ms				res.send('Mal: 404', 404);
161					});
162					

b. 0x (26096.0x /flamegraph.html)



CONCLUSION:

Se observan que los logs en consola afectan al performance y también la función Next() de Logger que incorporé en el middleware. Ya que este se ejecuta en cada petición a la pagina.

Tambien puede ser que los proccess consuman mucho para conseguir información, ej los argumentos, cpus, plataforma.