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
                                                   document
                                                           Other
                                                                           1.5 kB
   app.get('/info', compression(), (req, res) => {
       const info = {
            inputArguments: JSON.stringify(args.MODE),
            cpuNum: os.cpus().length,
            rss: process.memoryUsage().rss,
            path: process.argv[0],
            processId: process.pid,
            projectFolder: `${process.cwd()}`
       res.render('index', { info })
       console.log(info)
   });
```

b. SIN COMPRESION

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.response_time:
vusers.completed: ..... 50
vusers.failed: ...... 0
vusers.session_length:
```

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.response time:
vusers.failed: ...... 0
vusers.session_length:
```

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
```

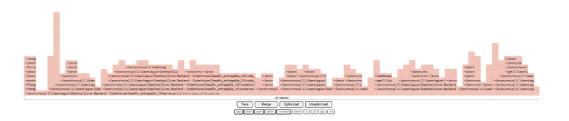
- 2) Luego utilizaremos Autocannon en línea de comandos, emulando 100 conexiones concurrentes realizadas en un tiempo de 20 segundos. Extraer un reporte con los resultados (puede ser un print screen de la consola)
 - a. -INSPECT

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

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

node server.js





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 peticion a la pagina.

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