

Curso: PROGRAMACION BACKEND

Comision: 30975

Alumno: Matias Herreros

Entregable #: 15 - logs, profiling & debug

Repositorio: <https://github.com/MHerreros/BackendCoderhouse/tree/entregable15>

Contenido

PROFILING DE SERVIDOR:	2
Profiling con ' —prof ' de node.js	3
Profiling con ' —prof ' de node.js utilizando Artillery	4
Profiling con ' —prof ' de node.js utilizando Autocannon	5
Perfilamiento del servidor con el modo inspector de node.js —inspect.....	6
Diagrama de flama con 0x	7
Compresión con GZip.....	8
Anexo:	9

PROFILING DE SERVIDOR:

Según lo solicitado en la consigna del entregable, se realizaron las siguientes 3 pruebas sobre la ruta “/info” del archivo “server.js” que se encuentra en el repositorio del entregable ([link to repo](#)). Sobre dicha ruta se incorporó/elimino un “console log” para ver la diferencia en el análisis.

- 1) Perfilamiento del servidor con—**prof de node.js**
 - a. Utilizar como test de carga Artillery en línea de comandos, emulando 50 conexiones concurrentes con 20 request por cada una.
 - b. Luego utilizar Autocannon en línea de comandos, emulando 100 conexiones concurrentes realizadas en un tiempo de 20 segundos.
- 2) Perfilamiento del servidor con el **modo inspector de node.js**—inspect.
- 3) Diagrama de flama con 0x, emulando la carga con Autocannon con los mismos parámetros anteriores.
- 4) Compresión con GZIP.

Profiling con '—prof' de node.js

- Con Console Log:

```
[Summary]:
  ticks  total  nonlib   name
    42    0.8%   97.7%  JavaScript
     0    0.0%    0.0%    C++
     9    0.2%   20.9%    GC
  5141   99.2%           Shared libraries
     1    0.0%           Unaccounted
```

- Sin Console Log:

```
[Summary]:
  ticks  total  nonlib   name
    27    0.1%   93.1%  JavaScript
     0    0.0%    0.0%    C++
    14    0.1%   48.3%    GC
 24834   99.9%           Shared libraries
     2    0.0%           Unaccounted
```

Sin el 'Console Log' en la ruta '/info' se tiene aproximadamente un ahorro del 50% de ticks en eventos relacionados a Java Script.

Profiling con '—prof' de node.js utilizando Artillery

- Con Console Log:

```
http.codes.200: ..... 1000
http.codes.302: ..... 1000
http.request_rate: ..... 27/sec
http.requests: ..... 2000
http.response_time:
  min: ..... 1
  max: ..... 2849
  median: ..... 788.5
  p95: ..... 982.6
  p99: ..... 1130.2
http.responses: ..... 2000
```

- Sin Console Log:

```
http.codes.200: ..... 1000
http.codes.302: ..... 1000
http.request_rate: ..... 30/sec
http.requests: ..... 2000
http.response_time:
  min: ..... 1
  max: ..... 2947
  median: ..... 772.9
  p95: ..... 963.1
  p99: ..... 1022.7
http.responses: ..... 2000
```

Según los resultados de Artillery, el tiempo medio de procesamiento de solicitudes a '/info' fue levemente menor cuando no se utilizó el 'console log'.

Profiling con ‘—prof’ de node.js utilizando Autocannon

- Con Console Log:

```
Carlos Tuma@DESKTOP-85GD7BR MINGW64 ~/Desktop/Backend/BackendCoderhouse/Entregables/HerrerosMatiasEntregable15 (entregable15)
$ npx autocannon -c 100 -d 20 http://localhost:8080/info/true
Running 20s test @ http://localhost:8080/info/true
100 connections
```

Stat	2.5%	50%	97.5%	99%	Avg	Stdev	Max
Latency	717 ms	1001 ms	1838 ms	2032 ms	1040.17 ms	224.14 ms	2986 ms

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	41	41	95	127	93.65	14.03	41
Bytes/Sec	15.2 kB	15.2 kB	35.2 kB	47.1 kB	34.7 kB	5.19 kB	15.2 kB

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

0 2xx responses, 1873 non 2xx responses
2k requests in 20.28s, 694 kB read

- Sin Console Log:

```
Carlos Tuma@DESKTOP-85GD7BR MINGW64 ~/Desktop/Backend/BackendCoderhouse/Entregables/HerrerosMatiasEntregable15 (entregable15)
$ npx autocannon -c 100 -d 20 http://localhost:8080/info
Running 20s test @ http://localhost:8080/info
100 connections
```

Stat	2.5%	50%	97.5%	99%	Avg	Stdev	Max
Latency	284 ms	994 ms	1148 ms	1969 ms	983.9 ms	287.13 ms	3906 ms

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	96	96	98	101	97.85	0.91	96
Bytes/Sec	35.6 kB	35.6 kB	36.3 kB	37.5 kB	36.3 kB	340 B	35.6 kB

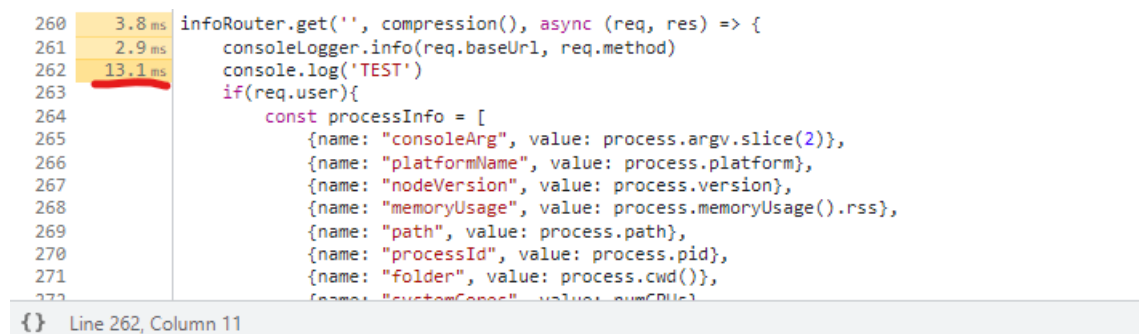
Req/Bytes counts sampled once per second.
of samples: 20

0 2xx responses, 1957 non 2xx responses
2k requests in 20.27s, 725 kB read

Se observa que sin Console Log se procesan levemente mas requests por segundo (97.85 vs 93.65 promedio). La latencia también es algo menor (983.9 vs 1040.2 promedio).

Perfilamiento del servidor con el modo inspector de node.js—inspect.

La siguiente imagen corresponde al perfilamiento del servidor (con console log en la ruta /info) utilizando el modo inspect de Chrome:



```

260 3.8 ms infoRouter.get('', compression(), async (req, res) => {
261 2.9 ms   console.log(req.baseUrl, req.method)
262 13.1 ms   console.log('TEST')
263         if(req.user){
264             const processInfo = [
265                 {name: "consoleArg", value: process.argv.slice(2)},
266                 {name: "platformName", value: process.platform},
267                 {name: "nodeVersion", value: process.version},
268                 {name: "memoryUsage", value: process.memoryUsage().rss},
269                 {name: "path", value: process.path},
270                 {name: "processId", value: process.pid},
271                 {name: "folder", value: process.cwd()},
272                 {name: "systemCpu", value: process.cpuUsage()}

```

Se puede observar que el console log aporta una perdida de tiempo en el procesamiento de los requests de 13.1 ms. Esta perdida se da ante una prueba hecha con autocannon con una concurrencia de 100 durante 20 seg.

- Sin console log:

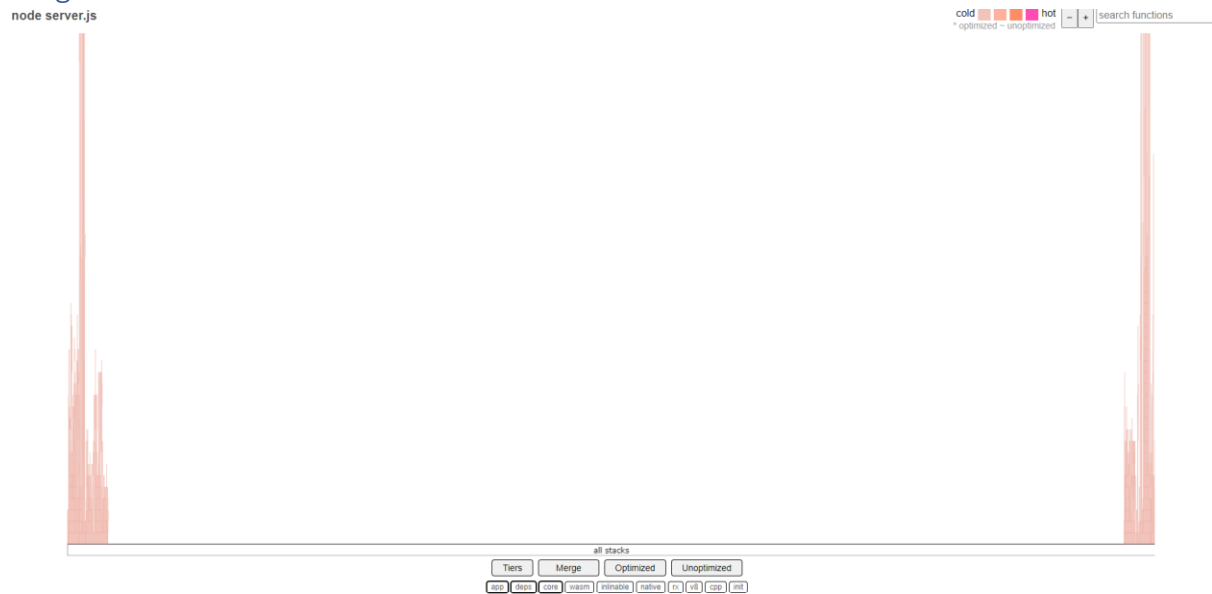
Self Time	Total Time	Function
27099.2 ms 69.83 %	27099.2 ms 69.83 %	(program)
5888.3 ms 15.17 %	6643.5 ms 17.12 %	▼ consoleCall
5888.3 ms 15.17 %	6643.5 ms 17.12 %	▶ (anonymous)
578.2 ms 1.49 %	578.2 ms 1.49 %	▶ writeUtf8String
250.7 ms 0.65 %	250.7 ms 0.65 %	▶ writev
241.7 ms 0.62 %	241.7 ms 0.62 %	▶ writeBuffer
115.9 ms 0.30 %	115.9 ms 0.30 %	(garbage collector)
103.6 ms 0.27 %	437.9 ms 1.13 %	▶ deserializeObject

- Con console log:

Self Time	Total Time	Function
05704.0 ms 85.83 %	05704.0 ms 85.83 %	(program)
10628.6 ms 8.63 %	12079.6 ms 9.81 %	▼ consoleCall
5509.2 ms 4.47 %	6266.3 ms 5.09 %	▶ (anonymous)
5119.4 ms 4.16 %	5813.3 ms 4.72 %	▶ (anonymous)
1106.8 ms 0.90 %	1106.8 ms 0.90 %	▶ writeUtf8String
264.5 ms 0.21 %	264.5 ms 0.21 %	▶ writeBuffer
244.0 ms 0.20 %	244.0 ms 0.20 %	▶ writev
143.4 ms 0.12 %	556.5 ms 0.45 %	▶ deserializeObject

Puede observarse que, a nivel general, el proceso que mas tiempo toma es el de “consoleCall”. Revisando mas a detalle este proceso se observa que aquí están incluidos los console logs realizados por el servidor (los realizados por log4js y los console log nativos propios del script). De la comparación de las tablas de arriba, puede notarse que el tiempo de procesamiento asociado a los console logs casi se duplica al añadir el console log en la ruta /info.

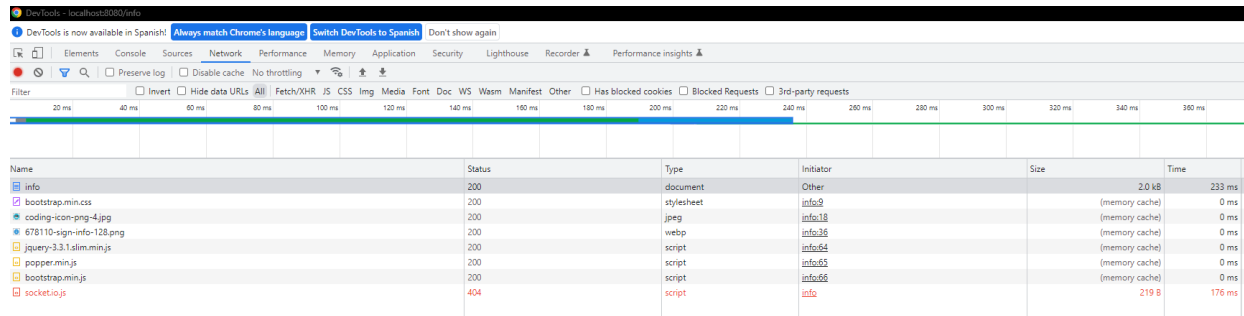
Diagrama de flama con Ox



En la imagen de arriba pueden observarse 2 picos de actividad. El primero se corresponde a una prueba con 'autocannon' hacia la ruta '/info' sin 'console log' y el segundo corresponde a la misma prueba, pero hacia la ruta '/info' con 'console log'. Si bien es difícil de apreciar en esta captura, la altura de la línea con 'console log' es mayor que la prueba donde no se tiene dicho 'console'. En ambos casos la consulta a '/info' y el 'console log' (dependiendo del caso) se encuentran bien arriba en el gráfico de flama, es decir que bloquean momentáneamente los procesos de abajo. Sin embargo, el bloqueo es por muy poco tiempo por lo que no se ve un gran impacto en el gráfico (mesetas).

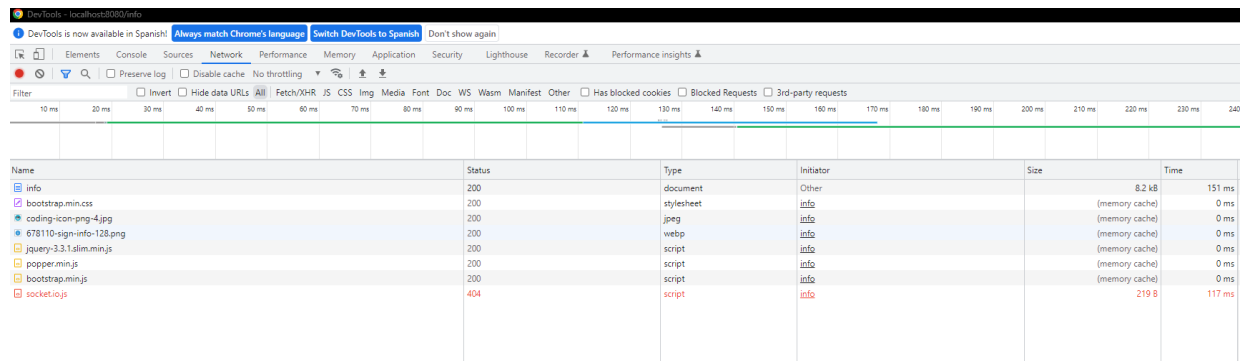
Compresión con GZip

- Comprimido:



Name	Status	Type	Initiator	Size	Time
info	200	document	Other	2.0 kB	233 ms
bootstrap.min.css	200	stylesheet	info:9	(memory cache)	0 ms
coding-icon-png-4.jpg	200	jpeg	info:18	(memory cache)	0 ms
678110-sign-info-128.png	200	webp	info:36	(memory cache)	0 ms
jquery-3.3.1.slim.min.js	200	script	info:64	(memory cache)	0 ms
popper.min.js	200	script	info:65	(memory cache)	0 ms
bootstrap.min.js	200	script	info:66	(memory cache)	0 ms
socket.io.js	404	script	info	219 B	176 ms

- Sin Comprimir:



Name	Status	Type	Initiator	Size	Time
info	200	document	Other	8.2 kB	151 ms
bootstrap.min.css	200	stylesheet	info	(memory cache)	0 ms
coding-icon-png-4.jpg	200	jpeg	info	(memory cache)	0 ms
678110-sign-info-128.png	200	webp	info	(memory cache)	0 ms
jquery-3.3.1.slim.min.js	200	script	info	(memory cache)	0 ms
popper.min.js	200	script	info	(memory cache)	0 ms
bootstrap.min.js	200	script	info	(memory cache)	0 ms
socket.io.js	404	script	info	219 B	117 ms

Puede observarse que, al no comprimir el request a la ruta /info se transmiten 8.2 kB de información mientras que al comprimirse solo se transmiten 2.0 kB de información. Esto es un ahorro de un 75% en el trafico de información. En definitiva, cuando el volumen de información lo amerita, es recomendable el uso de ‘compression’.

Anexo:

Toda la documentación respaldatoria se encuentra cargada en el repositorio público. El link a dicho repositorio se encuentra en la primera página del informe.