LOGGERS, GZIP y ANÁLISIS DE PERFORMANCE

Retomemos nuestro trabajo para implementar compresión por Gzip, registros por loggueo, y analizar la performance de nuestro servidor.

Curso: Programación BackEnd
Comision: 31835
Alumno: Cristian Alberto Correa
Desafio_Clase32: LOGGERS, GZIP y ANÁLISIS DE PERFORMANCE
Repositorio: https://github.com/crisco85/Programacion_BackEnd/tree/Desafio_Clase32

Contenido

PROFILING DE SERVIDOR:

En el entregable se realizaron las siguientes 3 pruebas sobre la ruta “/info” del archivo “server.js” que se encuentra en el repositorio previamente indicado. 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 35,71% 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: .................................................. 30/sec
http.requests: ...................................................... 2000
http.response_time:
  min: .............................................................. 1
  max: .............................................................. 2025
  median: ........................................................... 133
  p95: .............................................................. 982.6
  p99: .............................................................. 1107.9
http.responses: ..................................................... 2000
```

- Sin Console Log:

```
http.codes.200: ..................................................... 1000
http.codes.302: ..................................................... 1000
http.request_rate: .................................................. 29/sec
http.requests: ...................................................... 2000
http.response_time:
  min: .............................................................. 1
  max: .............................................................. 1023
  median: ........................................................... 89.1
  p95: .............................................................. 963.1
  p99: .............................................................. 982.6
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:

```
crist@DESKTOP-ALNJ555 MINGW64 /f/Document/Cristian - Documents/CoderHouse/Programación BackEnd/Desafios/Desafio_Clase32 (Desafio_Clase32)
$ npx autocannon -c 100 -d 20 http://localhost:8080/info
npm WARN config global `--global`, `--local` are deprecated. Use `--location=global` instead.
Running 20s test @ http://localhost:8080/info
100 connections


| Stat | 2.5% | 50% | 97.5% | 99% | Avg | Stdev | Max |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Latency | 783 ms | 1021 ms | 1875 ms | 2037 ms | 1048.82 ms | 227.47 ms | 2944 ms |

| Stat | 1% | 2.5% | 50% | 97.5% | Avg | Stdev | Min |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Req/Sec | 35 | 35 | 95 | 97 | 91.75 | 13.1 | 35 |
| Bytes/Sec | 13.2 kB | 13.2 kB | 35.8 kB | 36.6 kB | 34.6 kB | 4.93 kB | 13.2 kB |

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

0 2xx responses, 1835 non 2xx responses
2k requests in 20.25s, 691 kB read
```

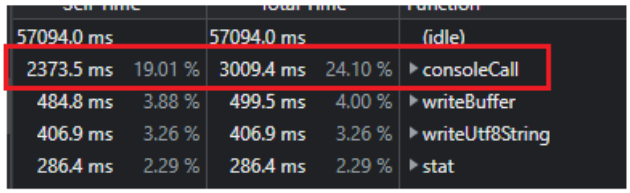
- Sin Console Log:

```
crist@DESKTOP-ALNJ555 MINGW64 /f/Document/Cristian - Documents/CoderHouse/Programación BackEnd/Desafios/Desafio_Clase32 (Desafio_Clase32)
$ npx autocannon -c 100 -d 20 http://localhost:8080/info
npm WARN config global `--global`, `--local` are deprecated. Use `--location=global` instead.
Running 20s test @ http://localhost:8080/info
100 connections


| Stat | 2.5% | 50% | 97.5% | 99% | Avg | Stdev | Max |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Latency | 279 ms | 997 ms | 1422 ms | 1993 ms | 990.75 ms | 287.82 ms | 3902 ms |

| Stat | 1% | 2.5% | 50% | 97.5% | Avg | Stdev | Min |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Req/Sec | 94 | 94 | 98 | 101 | 97.3 | 1.88 | 94 |
| Bytes/Sec | 35.4 kB | 35.4 kB | 36.9 kB | 38 kB | 36.6 kB | 703 B | 35.4 kB |

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

0 2xx responses, 1946 non 2xx responses
2k requests in 20.24s, 733 kB read
```

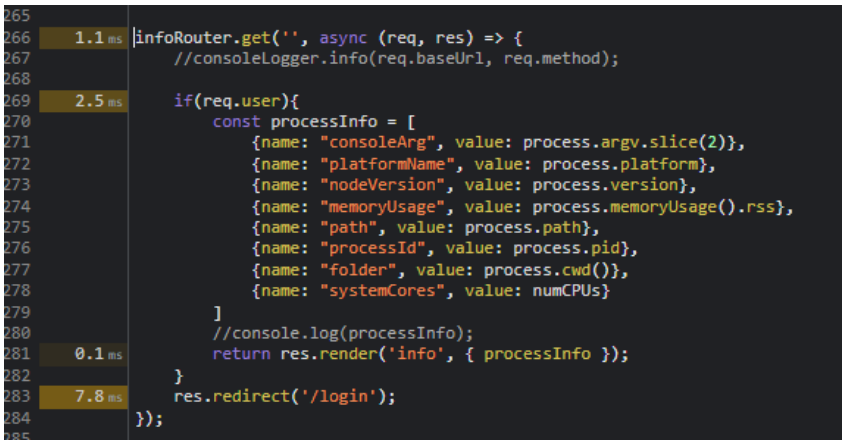
Se observa que sin Console Log se procesan levemente mas requests por segundo (97.3 vs 91.75 promedio). La latencia también es algo menor (990.75 vs 1048.82 promedio).

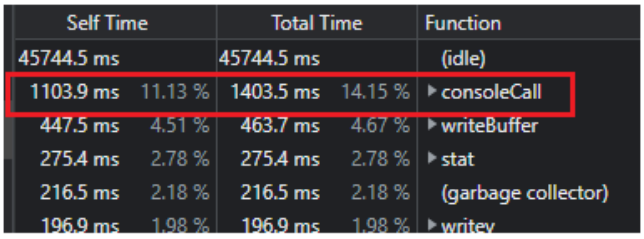
Perfilamiento del servidor con el modo inspector de node.js—inspect.

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



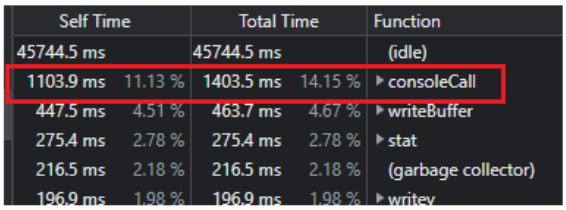
La siguiente imagen corresponde al perfilamiento del servidor (sin console log en la ruta /info) utilizando el modo inspect de Chrome:





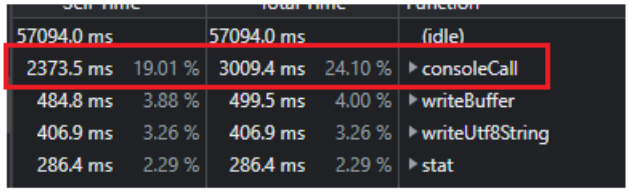
Haciendo una comparativa puede observar que el console log aporta una perdida de tiempo en el procesamiento de los requests de 2.9 ms.

- Sin console log:



Self Time		Total Time		Function
45744.5 ms		45744.5 ms		(idle)
1103.9 ms	11.13 %	1403.5 ms	14.15 %	consoleCall
447.5 ms	4.51 %	463.7 ms	4.67 %	writeBuffer
275.4 ms	2.78 %	275.4 ms	2.78 %	stat
216.5 ms	2.18 %	216.5 ms	2.18 %	(garbage collector)
196.9 ms	1.98 %	196.9 ms	1.98 %	writev

- Con console log:



57094.0 ms		57094.0 ms		(idle)
2373.5 ms	19.01 %	3009.4 ms	24.10 %	consoleCall
484.8 ms	3.88 %	499.5 ms	4.00 %	writeBuffer
406.9 ms	3.26 %	406.9 ms	3.26 %	writeUtf8String
286.4 ms	2.29 %	286.4 ms	2.29 %	stat

Y a nivel general, el proceso que mas tiempo toma es el de “consoleCall”. Dado que en 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 0x

No pude generar el diagrama de Flama con 0x, intenté hacer directamente como vimos en el curso

```
tick,0x7ff8fbf4feb4,114558285,0,0x0,8
tick,0x7ff8fbf4feb4,114573698,0,0x0,8
tick,0x7ff8fbf4feb4,114589035,0,0x0,8
tick,0x7ff8fbf4feb4,114604657,0,0x0,8
tick,0x7ff8fbf4feb4,114619565,0,0x0,8
tick,0x7ff8fbf4feb4,114634305,0,0x0,8
tick,0x7ff8fbf4feb4,114649701,0,0x0,8
tick,0x7ff8fbf4feb4,114665642,0,0x0,8
tick,0x7ff8fbf4feb4,114681418,0,0x0,8
tick,0x7ff8fbf4feb4,114697171,0,0x0,8
tick,0x7ff8fbf4feb4,114712837,0,0x0,8
Waiting for subprocess to exit...code-creation,LazyCompile,10,114718707,0xc4c24a369e,130,exit node:internal/process/p
18b00,~
code-source-info,0xc4c24a369e,30,4965,5449,C0O4979C5O4987C1704999C20O4987C2605026C3105043C3605057C4305074C4705089C52O
153C7305161C8605182C9405161C10005402C10505410C11505429C12305410C12905448,,
Process exited, generating flamegraphTypeError [ERR_INVALID_ARG_TYPE]:
The "path" argument must be of type string. Received undefined
    at new NodeError (node:internal/errors:372:5)
    at validateString (node:internal/validators:120:11)
    at join (node:path:429:7)
    at C:\Users\crist\AppData\Roaming\npm\node_modules\0x\index.js:89:27
    at processTicksAndRejections (node:internal/process/task_queues:96:5)
    at async zeroEks (C:\Users\crist\AppData\Roaming\npm\node_modules\0x\index.js:46:43)
    at async cmd (C:\Users\crist\AppData\Roaming\npm\node_modules\0x\cmd.js:98:21) {
  code: 'ERR_INVALID_ARG_TYPE'
}
```

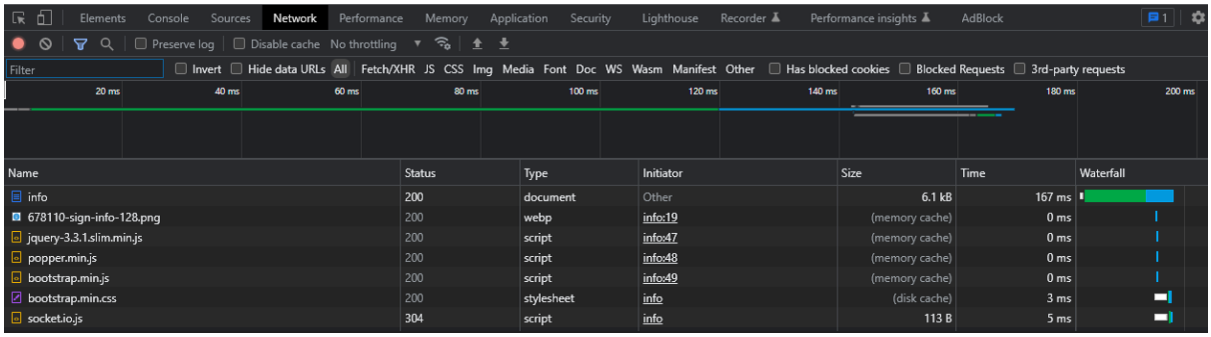
Después intente forzar la ejecución de 0x con autcannon;

```
crist@DESKTOP-ALNJ555 MINGW64 /f/Document/Cristian - Documents/CoderHouse/Progr
$ 0x -P 'autocannon -c 100 -d 20 http://localhost:8080/info' server.js
```

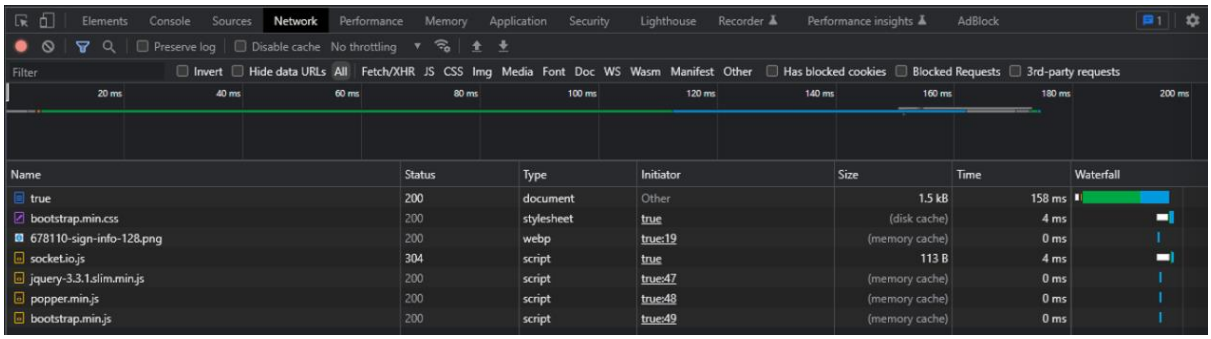
Lo desinstale y volví a instalar de manera global y no pude hacer que me genere los archivos correspondientes, siempre se lo mismo que en la figura previa.

Compresión con GZip

- Comprimido:



- Sin Comprimir:



Se puede observarse que al no comprimir el request a la ruta /info se transmiten 6.1 kB de información mientras que al comprimirse solo se transmiten 1.5 kB de información. Esto es un ahorro de un 75% en el tráfico. Cuando el volumen de información lo amerita, es recomendable el uso de 'compression'.