Desafío 16 - Loggers, GZip y Análisis de Performance - Joaquín Carré

Consigna 1

Al verificar la ruta /info sin el *console.log()* con y sin compresión la diferencia de bytes devueltos se observa que *con compresión devuelve 91975680 bytes*, en cambio *sin compresión devuelve 92196864 bytes* siendo **levemente menor con compresión**. Al verificar con el *console.log()* se observa que *con compresión devuelve 90087424 bytes* y *sin compresión 90722304 bytes* siendo de nuevo **levemente menor con compresión**.

Consigna 2

Node built-in profiler

Se instala previamente y de forma global el módulo Artillery con el comando:

npm i -g artillery

Luego, asegurándonos que en la ruta /info no se aplique el console.log(), se inicia la consigna abriendo una terminal y escribiendo el comando:

node --prof app.js

generando un archivo Isolate, para luego abrir otra terminal y utilizar el comando:

artillery quick --count 50 -n 20 'http://localhost:8080/info' > result_no-console.txt

para generar un archivo *result_no-console.txt* para la ruta /info y sin uso del console.log(). Finalmente, se cierra el proceso en la primer terminal con el comando Ctrl+C y se renombra el archivo *Isolate[...].log* por *info_no-console.log* y se escribe el siguiente comando:

node --prof-process info_no-console.log > result_prof_no-console.txt

creando el archivo de decodificación *result_prof_no-console.txt* que trae el siguiente resumen de la inspección:

```
[Summary]:
  ticks  total  nonlib   name
    10    0.5%  100.0%  JavaScript
     0    0.0%    0.0%  C++
    15    0.7%  150.0%  GC
  2115   99.5%          Shared libraries
```

Por otro lado se utiliza el comando:

artillery quick --count 50 -n 20 'http://localhost:8080/info' > result_console.txt

para generar otro archivo *result_console.txt* para la ruta /info y con el uso del console.log(). Finalmente, se cierra el proceso en la primer terminal con el comando Ctrl+C y se renombra el archivo *Isolate[...].log* por *info_console.log* y se escribe el siguiente comando:

node --prof-process info_console.log > result_prof_console.txt

creando el archivo de decodificación *result_prof_console.txt* que trae el siguiente resumen de la inspección:

```
[Summary]:
  ticks  total  nonlib   name
    14    0.8%  100.0%  JavaScript
     0    0.0%    0.0%  C++
     7    0.4%   50.0%  GC
  1644   99.2%          Shared libraries
```

De este estudio se pudo concluir que **en Shared Libraries del apartado Summary el proceso que no tenía console.log() se lleva más ticks por lo que este proceso es el que más consume recursos.**

Node inspect

Lo mismo se hace con el *Inspect* de node encendiendo el servidor con el comando:

node --inspect app.js

para luego entrar al navegador y en la barra de navegación escribir:

chrome://inspect

ingresando a la DevTools donde comienza a grabar un perfil mientras se ejecuta en otra terminal el siguiente comando:

artillery quick --count 50 -n 20 'http://localhost:8080/info' > result_no-console-inspect.txt

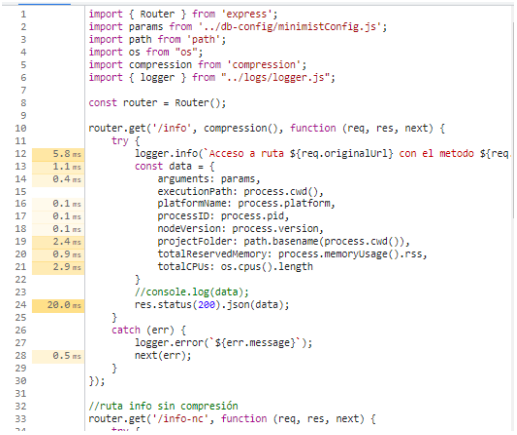
para el caso sin console.log().

Repitiendo la anterior secuencia hasta comenzar a grabar un perfil se ejecuta un nuevo comando:

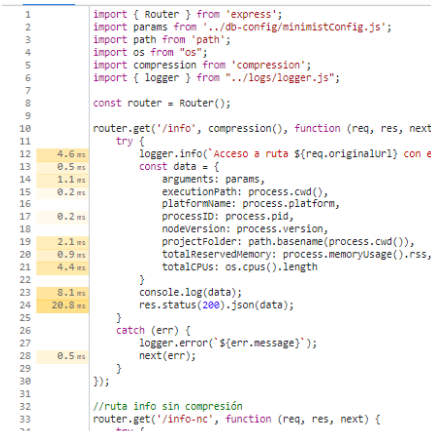
artillery quick --count 50 -n 20 'http://localhost:8080/info' > result_console-inspect.txt

para el caso con console.log().

Luego se observa la información obtenida en las siguientes imágenes:



GET /Info sin console.log()



GET /Info con console.log()

Se observa una menor performance cuando se tiene el console.log(), con una diferencia de más de 9ms respecto al que no lo tiene. Algo contrario a lo visto anteriormente.

Consigna 3

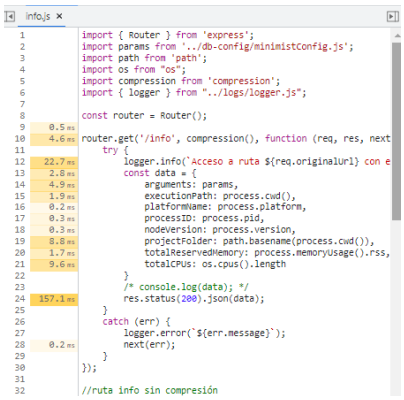
Se realiza el perfilamiento del servidor con el modo Inspect escribiendo el comando:

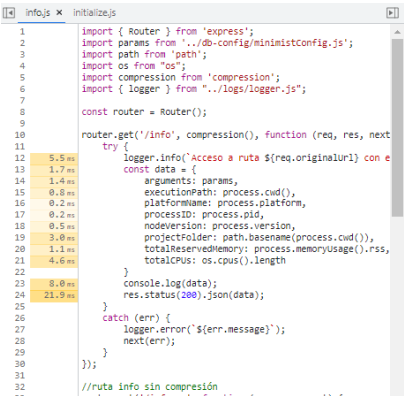
node --inspect app.js

Luego en otra terminal se escribe el comando:

npm run test

y se graba todo el proceso hasta finalizar y finalizar el proceso inspect de la primer terminal, obteniendo los siguientes resultados:





De esto se puede concluir, y en base a los resultados anteriores y a los diagramas de flama realizados con 0x (se encuentran en la carpeta archivos_desafio/autocannon), que **al agregar el console.log() mejora la performance del servidor**. Por simple lógica debería ocurrir lo contrario, pero es un problema que no se pudo encontrar el porqué, pero las inspecciones arrojan este resultado final.

NOTA: para iniciar el perfilamiento del servidor con 0x para generar el diagrama de flama se escribe el comando:

npm start

y en otra terminal se repite el comando:

npm run test