Coder - Backend Clase 16 Informe

Chevalier Esteban

Node Profiler con Artillery

Se levanta el servidor en node con el built-in profiler.

node –prof

Y a continuación ejecutamos las pruebas de desempeño con Artelly.

Guardamos las salidas en archivos de txt.

artillery quick -k --count 50 -n 20 https://localhost:8080/api/info > result_withOutLog.txt

artillery quick -k --count 50 -n 20 https://localhost:8080/api/info?verbose=1 > result_withLog.txt

Resultados

Podemos observar que el console.log genera una demora en la ejecución de cada query.

Sin console.log media 459.5 ms

```
All VUs finished. Total time: 12 seconds

--------------------------------
Summary report @ 15:57:16(-0300)
--------------------------------

http.codes.200: ..................................................... 1000
http.request_rate: .................................................. 95/sec
http.requests: ...................................................... 1000
http.response_time:
  min: .............................................................. 235
  max: .............................................................. 936
  median: ........................................................... 459.5
  p95: .............................................................. 671.9
  p99: .............................................................. 889.1
http.responses: ..................................................... 1000
vusers.completed: ................................................... 50
vusers.created: ..................................................... 50
vusers.created_by_name.0: ........................................... 50
vusers.failed: ...................................................... 0
vusers.session_length:
  min: .............................................................. 9772.3
  max: .............................................................. 10437.4
  median: ........................................................... 10201.2
  p95: .............................................................. 10407.3
  p99: .............................................................. 10407.3
```

Con console.log media 550.1ms

```
All VUs finished. Total time: 14 seconds

--------------------------------
Summary report @ 15:58:37(-0300)
--------------------------------

http.codes.200: ..................................................... 1000
http.request_rate: .................................................. 82/sec
http.requests: ...................................................... 1000
http.response_time:
  min: .............................................................. 226
  max: .............................................................. 1123
  median: ........................................................... 550.1
  p95: .............................................................. 757.6
  p99: .............................................................. 1022.7
http.responses: ..................................................... 1000
vusers.completed: ................................................... 50
vusers.created: ..................................................... 50
vusers.created_by_name.0: ........................................... 50
vusers.failed: ...................................................... 0
vusers.session_length:
  min: .............................................................. 11214.8
  max: .............................................................. 11969.4
  median: ........................................................... 11734.2
  p95: .............................................................. 11971.2
  p99: .............................................................. 11971.2
```

Realizamos el procesado del log generado por node –prof

Vemos que la ejecución del servidor con console.log necesita más ticks.

Sin console.log requiere 12612 ticks contra 11780 sin console.log

```
[Shared libraries]:
  ticks  total  nonlib   name
 12612   77.9%          /usr/local/lib/nodejs/bin/node
   513    3.2%          /usr/lib/x86_64-linux-gnu/libc.so.6
    36    0.2%          /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.30
     4    0.0%          [vdso]
```

```
[Shared libraries]:
  ticks  total  nonlib   name
 11780   75.8%          /usr/local/lib/nodejs/bin/node
   556    3.6%          /usr/lib/x86_64-linux-gnu/libc.so.6
    30    0.2%          /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.30
     3    0.0%          [vdso]
```

Autocannon

Se configura un benchmark para autocannon de 100 conexiones durante 20 segundos.

Podemos observar que cuando no se usa console.log la latencia es menor, lo que implica que el servidor responde a las consulta con mayor rapidez.

En la segunda tabla se observa la cantidad de solicitudes enviadas por segundo, y vemos que son console.log se consiguen enviar una mayor cantidad.

```
echevalier(main)@~/Devel/Coder/Backend/backend_clase16/desafio
$npm run test

> desafioclase16@1.0.0 test
> node benchmark.js

{ message: 'Running all benchmarks ...', level: 'info' }
Running 20s test @ https://localhost:8080/api/info
100 connections
```

Stat	2.5%	50%	97.5%	99%	Avg	Stdev	Max
Latency	1475 ms	1731 ms	3727 ms	3764 ms	1948.71 ms	624.09 ms	3804 ms

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	0	0	45	100	50	45.88	18
Bytes/Sec	0 B	0 B	124 kB	277 kB	138 kB	127 kB	49.8 kB

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

1k requests in 20.19s, 2.77 MB read
Running 20s test @ https://localhost:8080/api/info?verbose=1
100 connections
```

Stat	2.5%	50%	97.5%	99%	Avg	Stdev	Max
Latency	1489 ms	1736 ms	3858 ms	3880 ms	1957.6 ms	664.51 ms	4008 ms

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	0	0	50	100	49.85	42.07	2
Bytes/Sec	0 B	0 B	138 kB	277 kB	138 kB	116 kB	5.53 kB

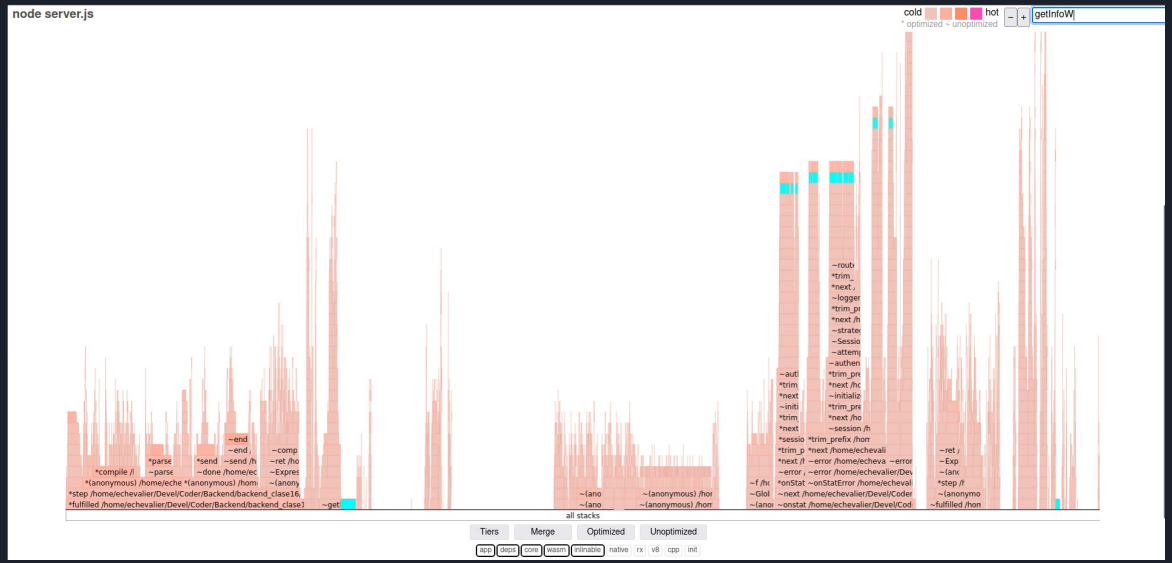
```
Req/Bytes counts sampled once per second.
# of samples: 20

1k requests in 20.21s, 2.76 MB read
```

0x junto con Autocannon

Generamos una gráfico de flama con la liberia 0x.

Podemos observar los picos donde el tamaño de la base son más grandes.

Estos son puntos que demoran la liberación del proceso de node.



0x gráfico de flama - Uso de console.log

Se analiza el gráfico buscando la función :

- “getInfoWithLog()” para el caso donde usa console.log()
- “getInfoWithoutLog()” en el caso donde no se usa console.log()

Con console.log() la base de los rectángulos es mayor.

Sin console.log() la base de los rectángulos es menor, lo que implica que demora menos el proceso en salir del stack.