

Reporte Desafío 16

1. Incorporar al proyecto de servidor de trabajo la compresión gzip. Verificar sobre la ruta /info con y sin compresión, la diferencia de cantidad de bytes devueltos en un caso y otro.

Respuesta sin compresión

GET

http://localhost:8080/info

Send

Params

Authorization

Headers (7)

Body

Pre-request Script

Tests

Settings

Cookies

Headers 7 hidden

KEY	VALUE	DESCRIPTION		Bulk Edit	Presets
Key	Value	Description			

Body

Cookies

Headers (7)

Test Results

Status: 200 OK Time: 43 ms Size: 1.41 KB Save Response

KEY	VALUE
X-Powered-By	Express
Content-Type	text/html; charset=utf-8
Content-Length	1209
ETag	W/"4b9-943bdM6101R8gPICsp04qKFq35g"
Date	Sat, 11 Jun 2022 18:23:13 GMT
Connection	keep-alive
Keep-Alive	timeout=5

Respuesta utilizando compresión

Desafio 16 / New Request

Save

GET

http://localhost:8080/info

Send

Params

Authorization

Headers (7)

Body

Pre-request Script

Tests

Settings

Cookies

Headers 7 hidden

KEY	VALUE	DESCRIPTION		Bulk Edit	Presets
Key	Value	Description			

Body

Cookies

Headers (9)

Test Results

Status: 200 OK Time: 50 ms Size: 1.46 KB Save Response

KEY	VALUE
X-Powered-By	Express
Content-Type	text/html; charset=utf-8
ETag	W/"4b9-TtNfKBgD6G+VGRokBWXB5xzmj w"
Vary	Accept-Encoding
Content-Encoding	gzip
Date	Sat, 11 Jun 2022 18:16:59 GMT
Connection	keep-alive
Keep-Alive	timeout=5
Transfer-Encoding	chunked

Conclusión: Se observa una disminución de 0,05 KB en el tamaño de la respuesta del método info al utilizar compression.

2) El perfilamiento del servidor, realizando el test con --prof de node.js. Analizar los resultados obtenidos luego de procesarlos con --prof-process. Utilizaremos como test de carga Artillery en línea de comandos, emulando 50 conexiones concurrentes con 20 request por cada una. Extraer un reporte con los resultados en archivo de texto.

Resultado método info sin console log:

```
[Summary]:
```

ticks	total	nonlib	name
404	5.7%	6.4%	JavaScript
5897	83.4%	92.8%	C++
225	3.2%	3.5%	GC
716	10.1%		Shared libraries
54	0.8%		Unaccounted

Resultado método info con console log:

```
[Summary]:
```

ticks	total	nonlib	name
338	5.1%	5.8%	JavaScript
5421	81.2%	93.1%	C++
211	3.2%	3.6%	GC
848	12.7%		Shared libraries
66	1.0%		Unaccounted

Conclusión: Se observa que el método info que realiza el console.log tiene un mayor número de ticks que la versión que no realiza el console.log. Inicialmente se esperaba que se diera la situación contraria dado en la segunda configuración el código ejecuta una operación adicional.

3) 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)

Resultado método info sin console log:

```
> node ./src/util/benchmark.js
zsh: correct './src/util/benchmark.js' to './src/utls/benchmark.js' [nyae]? y
Running all tests
Running 20s test @ http://localhost:8080/info
100 connections
```

Stat	2.5%	50%	97.5%	99%	Avg	Stdev	Max
Latency	320 ms	388 ms	553 ms	558 ms	403.95 ms	59.75 ms	660 ms

```
process.stdout.write(Buffer.concat(bufs));
```

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	172	172	246	300	244.8	30.71	172
Bytes/Sec	248 kB	248 kB	355 kB	432 kB	353 kB	44.3 kB	248 kB

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

Resultado método info con console log:

```
> node ./src/utis/benchmark.js
Running all tests
Running 20s test @ http://localhost:8080/info
100 connections
```

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	172	172	246	388	244.8	38.71	172
Bytes/Sec	248 kB	248 kB	353 kB	432 kB	323 kB	40.1 kB	248 kB
Latency	353 ms	410 ms	580 ms	589 ms	432.45 ms	63.88 ms	685 ms

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

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	182	182	227	268	228.9	20.43	182
Bytes/Sec	262 kB	262 kB	327 kB	386 kB	330 kB	29.4 kB	262 kB

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

5k requests in 20.04s, 6.6 MB read

Conclusión: Se observa que el método info que tiene la operación adicional console.log puede atender menos requests por segundos y puede transmitir más información por segundo en línea con lo que se esperaba. Así también tiene una mayor latencia como se esperaba.

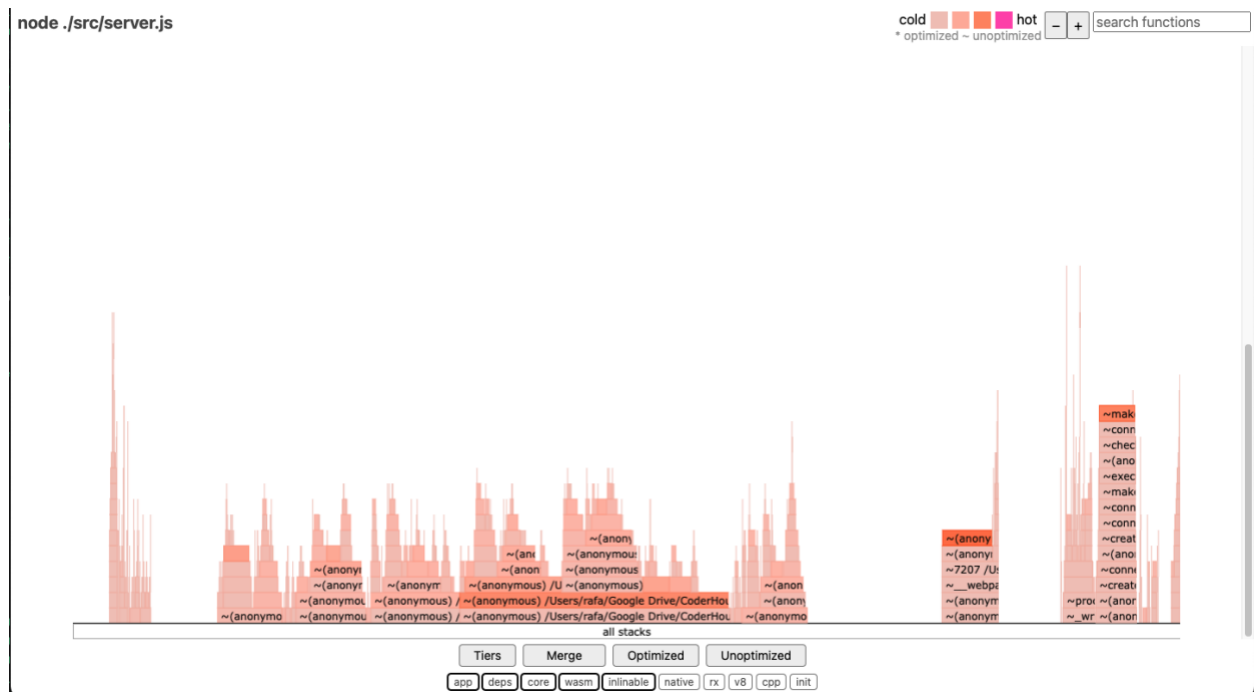
4) El perfilamiento del servidor con el modo inspector de node.js --inspect. Revisar el tiempo de los procesos menos performantes sobre el archivo fuente de inspección.

Resultado método info sin console log:

Self Time	Total Time	Function
12972.2 ms	12972.2 ms	(idle)
2265.4 ms 12.17 %	2603.8 ms 13.99 %	createFunctionContext javascript-compiler.js:223
2265.4 ms 12.17 %	2603.8 ms 13.99 %	compile javascript-compiler.js:64
996.4 ms 5.36 %	996.4 ms 5.36 %	getCPUs
877.2 ms 4.71 %	877.2 ms 4.71 %	readdir
865.6 ms 4.65 %	865.6 ms 4.65 %	open
559.9 ms 3.01 %	559.9 ms 3.01 %	(garbage collector)
535.1 ms 2.88 %	535.1 ms 2.88 %	(program)
515.8 ms 2.77 %	515.8 ms 2.77 %	memoryUsage
351.4 ms 1.89 %	513.6 ms 2.76 %	next
254.9 ms 1.37 %	254.9 ms 1.37 %	stat
252.9 ms 1.36 %	1150.3 ms 6.18 %	SourceNode_walk
250.4 ms 1.35 %	865.9 ms 4.65 %	parse
190.7 ms 1.03 %	255.8 ms 1.37 %	SourceNode_add
188.9 ms 1.02 %	7022.2 ms 37.74 %	initialize
187.7 ms 1.01 %	187.7 ms 1.01 %	close
180.4 ms 0.97 %	180.4 ms 0.97 %	registerDestroyHook
165.2 ms 0.89 %	396.1 ms 2.13 %	YargsInstance
161.2 ms 0.87 %	161.2 ms 0.87 %	writeUtf8String
157.7 ms 0.85 %	731.6 ms 3.93 %	(anonymous)
155.9 ms 0.84 %	301.6 ms 1.62 %	parse
151.5 ms 0.81 %	656.3 ms 3.53 %	wrap
134.6 ms 0.72 %	134.6 ms 0.72 %	writv
133.5 ms 0.72 %	204.1 ms 1.10 %	init
130.4 ms 0.70 %	202.0 ms 1.09 %	FSReqCallback
125.9 ms 0.68 %	10399.4 ms 55.89 %	step
109.8 ms 0.59 %	109.8 ms 0.59 %	read
106.8 ms 0.57 %	2085.5 ms 11.21 %	(anonymous)
106.2 ms 0.57 %	5467.8 ms 29.39 %	(anonymous)
106.0 ms 0.57 %	106.0 ms 0.57 %	fstat
99.8 ms 0.54 %	152.1 ms 0.82 %	resolve
96.4 ms 0.52 %	116.5 ms 0.63 %	(anonymous)
95.1 ms 0.51 %	6279.7 ms 33.75 %	getInfo
92.0 ms 0.49 %	92.0 ms 0.49 %	Hash
84.3 ms 0.45 %	140.8 ms 0.76 %	nextTick
82.6 ms 0.44 %	82.6 ms 0.44 %	quotedString
81.2 ms 0.44 %	394.9 ms 2.12 %	replaceStack
80.3 ms 0.43 %	80.3 ms 0.43 %	extend
79.2 ms 0.43 %	45203.9 ms 242.94 %	handle
76.5 ms 0.41 %	119.4 ms 0.64 %	getInternalMethods

Resultado método info con console log:

Self Time	Total Time	Function
10150.9 ms	10150.9 ms	(idle)
2223.8 ms 11.97 %	2560.5 ms 13.79 %	createFunctionContext javascript-compiler.js:223
978.3 ms 5.27 %	978.3 ms 5.27 %	getCPUs
905.2 ms 4.87 %	905.2 ms 4.87 %	open
885.7 ms 4.77 %	885.7 ms 4.77 %	readdir
575.3 ms 3.10 %	575.3 ms 3.10 %	(garbage collector)
548.6 ms 2.95 %	548.6 ms 2.95 %	(program)
462.0 ms 2.49 %	462.0 ms 2.49 %	memoryUsage
313.6 ms 1.89 %	489.3 ms 2.63 %	next
264.6 ms 1.42 %	264.6 ms 1.42 %	stat
255.8 ms 1.38 %	1160.9 ms 6.25 %	SourceNode_walk
245.0 ms 1.32 %	833.4 ms 4.49 %	parse
204.7 ms 1.10 %	204.7 ms 1.10 %	close
190.0 ms 1.02 %	190.0 ms 1.02 %	registerDestroyHook
178.1 ms 0.96 %	240.1 ms 1.29 %	SourceNode_add
176.9 ms 0.95 %	763.9 ms 4.11 %	(anonymous)
172.3 ms 0.93 %	6994.9 ms 37.67 %	initialize
166.0 ms 0.89 %	400.1 ms 2.15 %	YargsInstance
154.6 ms 0.83 %	632.1 ms 3.40 %	wrap
153.6 ms 0.83 %	153.6 ms 0.83 %	writeUtf8String
148.6 ms 0.80 %	305.3 ms 1.64 %	parse
133.5 ms 0.72 %	207.3 ms 1.12 %	FSReqCallback
132.1 ms 0.71 %	132.1 ms 0.71 %	writv
127.0 ms 0.68 %	127.0 ms 0.68 %	read
127.0 ms 0.68 %	202.3 ms 1.09 %	init
123.0 ms 0.66 %	10305.5 ms 55.49 %	step
118.0 ms 0.64 %	118.0 ms 0.64 %	fstat
101.2 ms 0.55 %	122.5 ms 0.66 %	(anonymous)
99.1 ms 0.53 %	2087.7 ms 11.24 %	(anonymous)
97.9 ms 0.53 %	5394.5 ms 29.05 %	(anonymous)
96.8 ms 0.52 %	149.6 ms 0.81 %	resolve
96.6 ms 0.52 %	6261.7 ms 33.72 %	getInfo
95.5 ms 0.51 %	95.5 ms 0.51 %	extend
88.9 ms 0.48 %	388.4 ms 2.09 %	replaceStack
88.8 ms 0.48 %	88.8 ms 0.48 %	Hash
83.1 ms 0.45 %	137.5 ms 0.74 %	nextTick
80.6 ms 0.43 %	120.7 ms 0.65 %	getInternalMethods
78.9 ms 0.42 %	45275.7 ms 243.81 %	handle
78.7 ms 0.42 %	78.7 ms 0.42 %	quotedString
77.7 ms 0.42 %	3402.8 ms 18.32 %	compile



Resultado método info con console log:



Conclusión: Luego de una inspección visual no se observan mayores diferencias entre el código que imprime en consola el resultado del método info con el que no lo hace. Se hubiese esperado ver alguna diferencia en la cantidad de peaks o la duración del stack total.