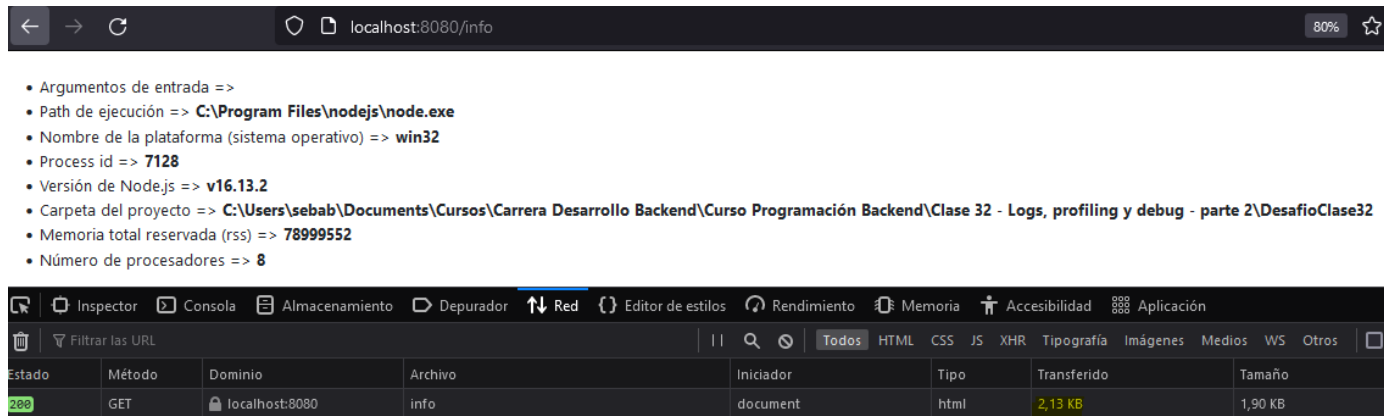


Desafío Clase 32

>> Consigna:

- 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.

Sin compresión: **/info**

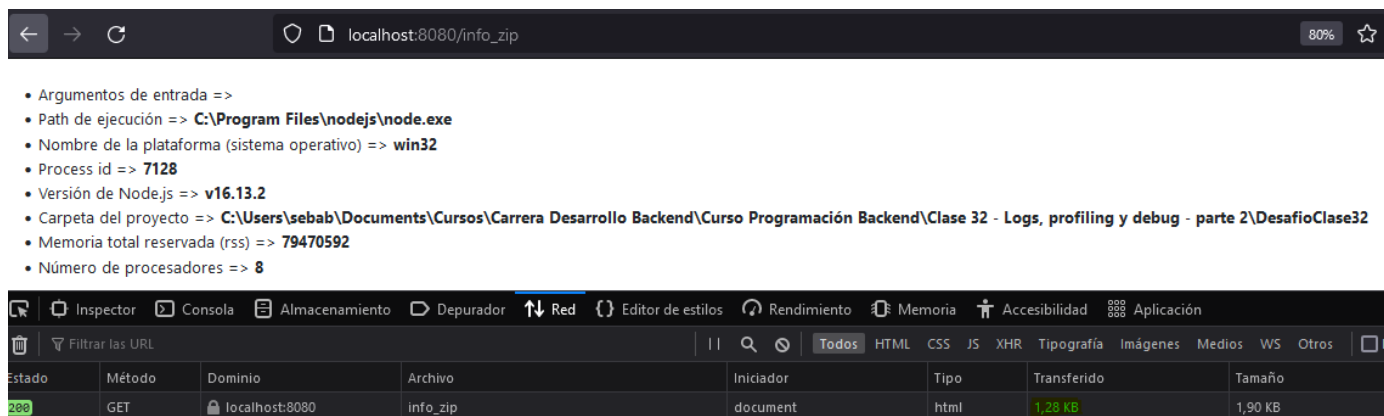


• Argumentos de entrada =>

- Path de ejecución => **C:\Program Files\nodejs\node.exe**
- Nombre de la plataforma (sistema operativo) => **win32**
- Process id => **7128**
- Versión de Node.js => **v16.13.2**
- Carpeta del proyecto => **C:\Users\sebab\Documents\Cursos\Carrera Desarrollo Backend\Curso Programación Backend\Clase 32 - Logs, profiling y debug - parte 2\DesafioClase32**
- Memoria total reservada (rss) => **78999552**
- Número de procesadores => **8**

Estado	Método	Dominio	Archivo	Iniciador	Tipo	Transferido	Tamaño
200	GET	localhost:8080	info	document	html	2.13 KB	1,90 KB

Con compresión: **/info_zip**



• Argumentos de entrada =>

- Path de ejecución => **C:\Program Files\nodejs\node.exe**
- Nombre de la plataforma (sistema operativo) => **win32**
- Process id => **7128**
- Versión de Node.js => **v16.13.2**
- Carpeta del proyecto => **C:\Users\sebab\Documents\Cursos\Carrera Desarrollo Backend\Curso Programación Backend\Clase 32 - Logs, profiling y debug - parte 2\DesafioClase32**
- Memoria total reservada (rss) => **79470592**
- Número de procesadores => **8**

Estado	Método	Dominio	Archivo	Iniciador	Tipo	Transferido	Tamaño
200	GET	localhost:8080	info_zip	document	html	1.28 KB	1,90 KB

- Luego implementar loggueo (con alguna librería vista en clase) que registre lo siguiente:
 - Ruta y método de todas las peticiones recibidas por el servidor (info)
 - Ruta y método de las peticiones a rutas inexistentes en el servidor (warning)
 - Errores lanzados por las apis de mensajes y productos, únicamente (error)

Considerar el siguiente criterio:

- Loggear todos los niveles a consola (info, warning y error)
- Registrar sólo los logs de warning a un archivo llamada warn.log
- Enviar sólo los logs de error a un archivo llamada error.log

(Resuelto en código con librería Winston...)

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

$ node index.js
Server is up and running on port 8080
New client connection! Id: xAlwZ9fpFrXep9ihAAAB
Client has left! Id: xAlwZ9fpFrXep9ihAAAB
2022-05-07T19:08:21.824Z [info] => [GET] => /
Inside deserializer
2022-05-07T19:08:25.985Z [info] => [GET] => /api/productos-test
Inside deserializer
New client connection! Id: ROpILWXP0f0IP7OQAAAD
Client has left! Id: ROpILWXP0f0IP7OQAAAD
2022-05-07T19:08:55.166Z [info] => [GET] => /api/productos-test2
Inside deserializer
2022-05-07T19:08:57.109Z [warn] => [GET] => /api/productos-test2
2022-05-07T19:09:02.803Z [info] => [GET] => /api/productos-testNOEXISTE
Inside deserializer
2022-05-07T19:09:04.743Z [warn] => [GET] => /api/productos-testNOEXISTE
2022-05-07T19:09:08.331Z [info] => [GET] => /api/productos-test
Inside deserializer
New client connection! Id: pGkuJyqFv-EvutrvAAAF
2022-05-07T19:09:23.451Z [error] => Hubo un error al guardar: Error generado con throw...

```

```

warn.log  X
DesafioClase32 > logger > warn.log
1  2022-05-07T19:08:57.110Z [warn] => [GET] => /api/productos-test2
2  2022-05-07T19:09:04.744Z [warn] => [GET] => /api/productos-testNOEXISTE
3

```

```

error.log  X
DesafioClase32 > logger > error.log
1  2022-05-07T19:09:23.451Z [error] => Hubo un error al guardar: Error generado con throw...
2

```

- Luego, realizar el análisis completo de performance del servidor con el que venimos trabajando.

Vamos a trabajar sobre la ruta '/info', en modo fork, agregando ó extrayendo un console.log de la información colectada antes de devolverla al cliente. Además desactivaremos el child_process de la ruta '/randoms'

Para ambas condiciones (con o sin console.log) en la ruta '/info' OBTENER:

1) 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.

Sin console.log

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
XebaX@XEBAX-PC MINGW64 ~/Documents/Cursos/Carrera Desarrollo Backend/Curso Programación Backend/Clase 32 - Logs, profiling y debug - parte 2/DesafioClase32
$ node --prof index.js
Server is up and running on port 8080

```

```

Administrador: Windows PowerShell
PS C:\Users\sebab\Documents> artillery quick -c 50 -n 20 "http://localhost:8080/info" > result_sin_consoleLog.txt
PS C:\Users\sebab\Documents>

```

```

Administrador: Windows PowerShell
PS C:\Users\sebab\Documents\Cursos\Carrera Desarrollo Backend\Curso Programación Backend\Clase 32 - Logs, profiling y debug - parte 2/DesafioClase32> node --prof-process prof_sin_consoleLog-v8.log > result_profiler_sin_consoleLog.txt

```

result_profiler_sin_consoleLog.txt

DesafioClase32 > result_profiler_sin_consoleLog.txt

36				
37	[Summary]:			
38	ticks	total	nonlib	name
39	30	0.7%	100.0%	JavaScript
40	0	0.0%	0.0%	C++
41	53	1.3%	176.7%	GC
42	4072	99.3%		Shared libraries

Con console.log

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
XebaX@XEBAX-PC MINGW64 ~/Documents/Cursos/Carrera Desarrollo Backend/Curso Programación Backend/Clase 32 - Logs, profiling y debug - parte 2/DesafioClase32
$ node --prof index.js
Server is up and running on port 8080

```

```
Administrador: Windows PowerShell
PS C:\Users\sebab\Documents> artillery quick -c 50 -n 20 "http://localhost:8080/info" > result_con_consoleLog.txt
PS C:\Users\sebab\Documents>
```

```
Administrador: Windows PowerShell
PS C:\Users\sebab\Documents\Cursos\Carrera Desarrollo Backend\Curso Programación Backend\Clase 32 - Logs, profiling y debug - parte 2\DesafioClase32> node --prof-process prof_con_consoleLog-v8.log > result_profiler_con_consoleLog.txt
```

```
result_profiler_con_consoleLog.txt X
DesafioClase32 > result_profiler_con_consoleLog.txt
67 [Summary]:
68 ticks total nonlib name
69 66 0.6% 97.1% JavaScript
70 0 0.0% 0.0% C++
71 74 0.6% 108.8% GC
72 11838 99.4% Shared libraries
73 2 0.0% Unaccounted
74
```

- 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)

2) 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.

Sin console.log

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
XebaX@XEBAX-PC MINGW64 ~/Documents/Cursos/Carrera Desarrollo Backend/Curso Programación Backend/Clase 32 - Logs, profiling y debug - parte 2/DesafioClase32
$ node --inspect index.js
Debugger listening on ws://127.0.0.1:9229/3b286318-9a2a-4a5d-8a44-af52ccf43a60
For help, see: https://nodejs.org/en/docs/inspector
Server is up and running on port 8080
[]
```

Administrador: Windows PowerShell

```
PS C:\Users\sebab\Documents> 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	404 ms	445 ms	737 ms	798 ms	471.18 ms	78.71 ms	843 ms

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	100	100	200	300	210	45.22	100
Bytes/Sec	218 kB	218 kB	436 kB	654 kB	458 kB	98.6 kB	218 kB

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

4k requests in 20.26s, 9.16 MB read
PS C:\Users\sebab\Documents>

DevTools - Node.js			
Connection Console Sources Memory Profiler			
Heavy (Bottom Up)			
Profiles	Self Time	Total Time	Function
CPU PROFILES	52007.2 ms	52007.2 ms	(idle)
Profile 1	1305.0 ms 6.87 %	1305.0 ms 6.87 %	(garbage collector)
	1151.6 ms 6.06 %	1151.6 ms 6.06 %	writeUtf8String
	766.7 ms 4.03 %	766.7 ms 4.03 %	(program)
	753.0 ms 3.96 %	779.5 ms 4.10 %	next
	726.9 ms 3.82 %	726.9 ms 3.82 %	stat
	555.8 ms 2.92 %	2760.5 ms 14.53 %	SourceNode_walk
	497.8 ms 2.62 %	626.8 ms 3.30 %	SourceNode_add
	483.5 ms 2.54 %	483.5 ms 2.54 %	writew
	385.1 ms 2.03 %	385.1 ms 2.03 %	quotedString
	320.4 ms 1.69 %	1645.1 ms 8.66 %	wrap
	297.0 ms 1.56 %	1222.5 ms 6.43 %	parse
	291.5 ms 1.53 %	987.8 ms 5.20 %	createFunctionContext
	281.3 ms 1.48 %	784.2 ms 4.13 %	setupHelperArgs
	256.6 ms 1.35 %	2374.3 ms 12.49 %	authenticate
	254.0 ms 1.34 %	307.8 ms 1.62 %	resolve
	248.6 ms 1.31 %	248.6 ms 1.31 %	registerDestroyHook

Con console.log

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Xebax@XEBAX-PC MINGW64 ~/Documents/Cursos/Carrera Desarrollo Backend/Curso Programación Backend/Clase 32 - Logs, profiling y debug - parte 2/DesafioClase32
$ node --inspect index.js
Debugger listening on ws://127.0.0.1:9229/b721f32b-ed4d-484b-87ba-9f58ecc15ab5
For help, see: https://nodejs.org/en/docs/inspector
Server is up and running on port 8080
```

Administrador: Windows PowerShell

```
PS C:\Users\sebab\Documents> 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	901 ms	971 ms	1206 ms	1279 ms	988.76 ms	78.46 ms	1298 ms

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	0	0	100	151	100	26.98	90
Bytes/Sec	0 B	0 B	218 kB	329 kB	218 kB	58.8 kB	196 kB

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

2k requests in 20.27s, 4.36 MB read
PS C:\Users\sebab\Documents>

DevTools Profiler - Heavy (Bottom Up)

Profile	Self Time	Total Time	Function	
46495.6 ms	46495.6 ms	(idle)	(idle)	
4077.5 ms	21.69 %	6075.3 ms	32.31 %	consoleCall
4077.5 ms	21.69 %	6075.3 ms	32.31 %	(anonymous)
2522.4 ms	13.42 %	2522.4 ms	13.42 %	writeUtf8String
1124.4 ms	5.98 %	1124.4 ms	5.98 %	(garbage collector)
537.3 ms	2.86 %	537.3 ms	2.86 %	stat
504.1 ms	2.68 %	504.1 ms	2.68 %	(program)
419.5 ms	2.23 %	435.3 ms	2.32 %	next
300.7 ms	1.60 %	1491.3 ms	7.93 %	SourceNode_walk
277.0 ms	1.47 %	354.8 ms	1.89 %	SourceNode_add
238.3 ms	1.27 %	238.3 ms	1.27 %	writev
204.8 ms	1.09 %	8103.3 ms	43.10 %	authenticate

DevTools Sources - index.js

```
169
170
171 app.get('/login-error', (req, res) => {
172   return res.render('error', { textoError: 'USER ERROR LOGIN' });
173 });
174
175 app.get('/register-error', (req, res) => {
176   return res.render('error', { textoError: 'USER ERROR SIGNUP' });
177 });
178
179 //Info
180 app.get('/info', async (req, res) => {
181   console.log(`
182     * Argumentos de entrada => ${process.argv.slice(2)}
183     * Path de ejecución => ${process.execPath}
184     * Nombre de la plataforma (sistema operativo) => ${process.platform}
185     * Process id => ${process.pid}
186     * Versión de Node.js => ${process.version}
187     * Carpeta del proyecto => ${process.cwd()}
188     * Memoria total reservada (rss) => ${process.memoryUsage().rss}
189     * Número de procesadores => ${CPU_NUMBERS}
190   `);
191 });
```

3) El diagrama de flama con 0x, emulando la carga con Autocannon con los mismos parámetros anteriores.

Sin console.log

Administrador: Windows PowerShell

```
PS C:\Users\sebab\Documents\DesafioClase32> 0x .\index.js
ProfilingServer is up and running on port 8080
```

Administrador: Windows PowerShell

```
PS C:\Users\sebab\Documents> 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	204 ms	251 ms	531 ms	660 ms	276.18 ms	88.23 ms	796 ms

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	100	100	378	487	361.75	88.39	100
Bytes/Sec	207 kB	207 kB	783 kB	1.01 MB	749 kB	183 kB	207 kB

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

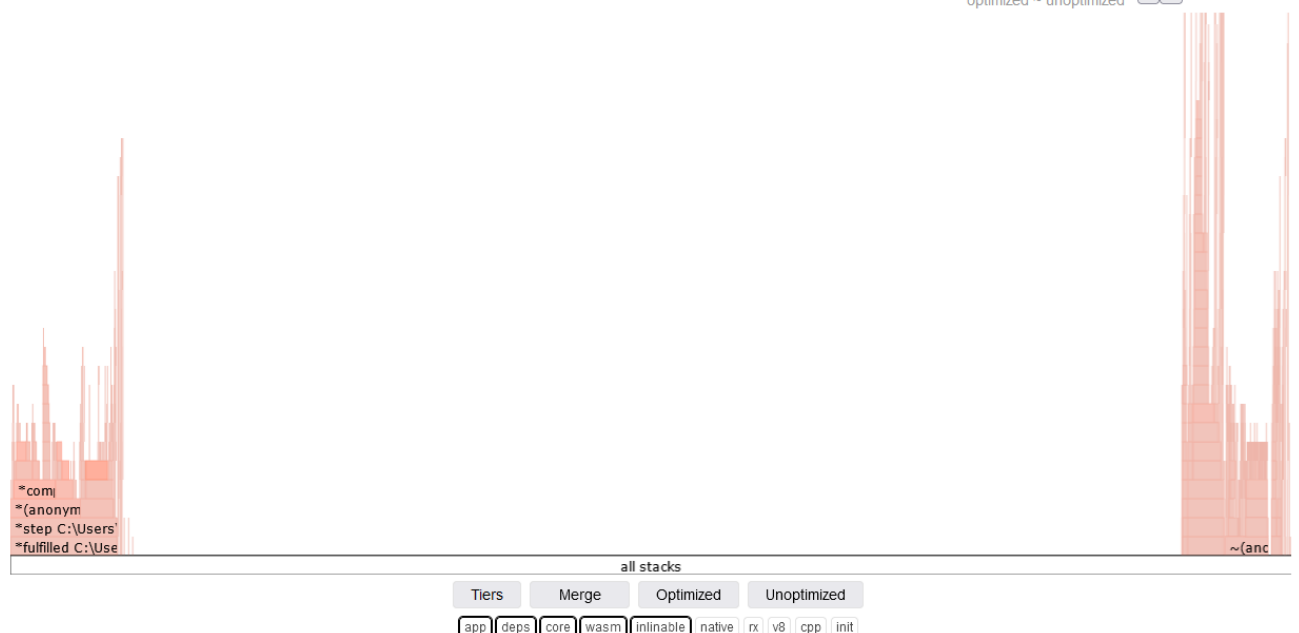
7k requests in 20.18s, 15 MB read
PS C:\Users\sebab\Documents>

Administrador: Windows PowerShell

```
Flamegraph generated in
file:///C:/Users/sebab/Documents/DesafioClase32/6272.0x/flamegraph.html
```

node .index.js

cold hot search functions
* optimized ~ unoptimized



Con console.log

Administrador: Windows PowerShell

```
PS C:\Users\sebab\Documents\DesafioClase32> 0x .\index.js
ProfilingServer is up and running on port 8080
```

Administrador: Windows PowerShell

```
PS C:\Users\sebab\Documents> 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	212 ms	233 ms	428 ms	704 ms	253.48 ms	70.79 ms	812 ms

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	100	100	418	475	393.85	84.64	100
Bytes/Sec	207 kB	207 kB	867 kB	985 kB	816 kB	175 kB	207 kB

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

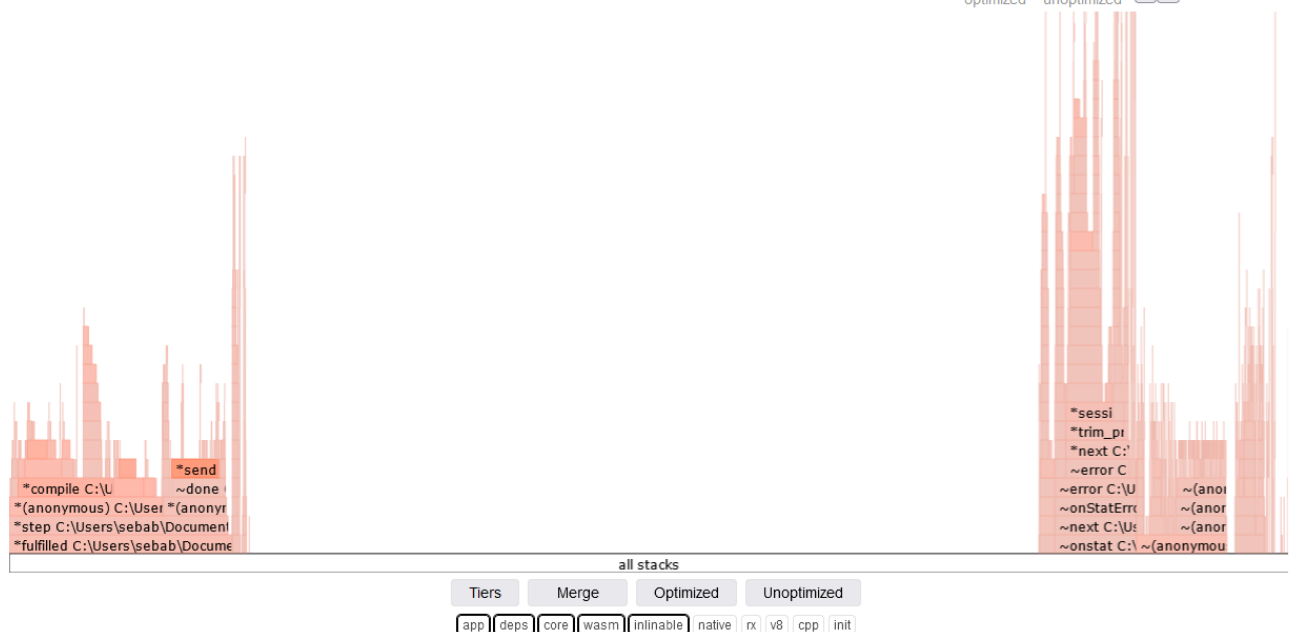
8k requests in 20.17s, 16.3 MB read
PS C:\Users\sebab\Documents>

Administrador: Windows PowerShell

```
Flamegraph generated in
file://C:\Users\sebab\Documents\DesafioClase32\11632.0x\flamegraph.html
```

node .index.js

cold hot
* optimized ~ unoptimized



Conclusión análisis de performance:

En todos los casos analizados se visualiza una mejor performance ejecutando el servidor **sin** console.log(). Esto era de esperarse debido a que el console.log() es un proceso bloqueante para el servidor, lo que hace que se produzcan demoras al ejecutarse como un proceso síncrono.

A continuación, se describen los casos analizados:

✓ **Análisis profiler nativo Node.js (--prof) + Artillery**

Analizando los resultados del profiler, vemos una mejoría en la app cuando **no** se imprime el console.log(). Verificando los ticks totales, se visualizan:

- Más del doble de ticks asociados a proceso de Javascript en la versión con console.log()
- Casi el triple de ticks asociados a "Shared libraries" en la versión con console.log()

✓ **Análisis inspect Chrome + Autocannon**

Nuevamente se visualiza una mejor performance ejecutando el servidor sin console.log(). Verificando el resumen provisto por "autocannon", vemos un promedio de 100 request por segundo al ejecutar el server con console.log(). Mientras que sin console.log() vemos un promedio de 210 request por segundo. Además se visualiza un promedio de latencia reducida casi a la mitad al ejecutarse sin console.log().

Por otro lado, vemos en el profiler de Chrome un aumento en los tiempos de respuesta al ejecutarse con console.log(). Se registró un 32% del tiempo total de la prueba procesando salidas por consola.

✓ **Análisis 0x + Autocannon**

Por último, el gráfico de flama nos permite visualizar que existen más procesos bloqueantes al ejecutar el servidor **con** console.log() que ejecutándolo sin console.log().