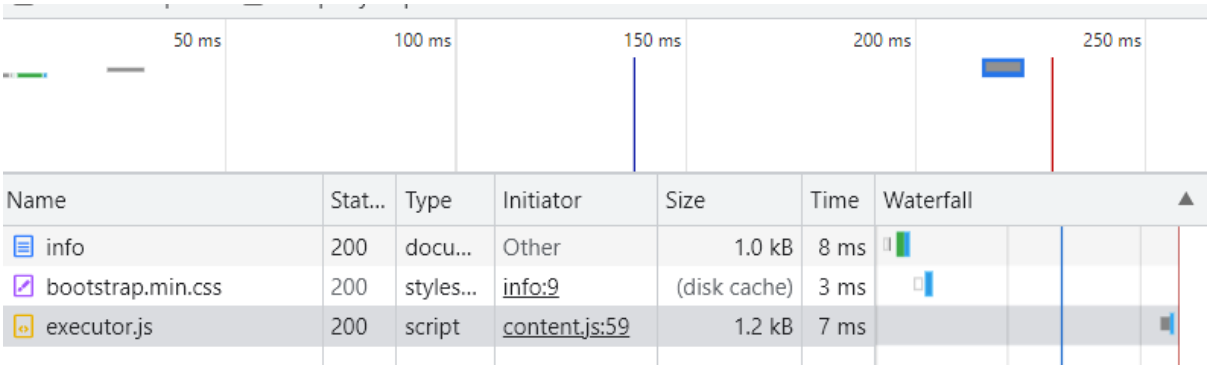
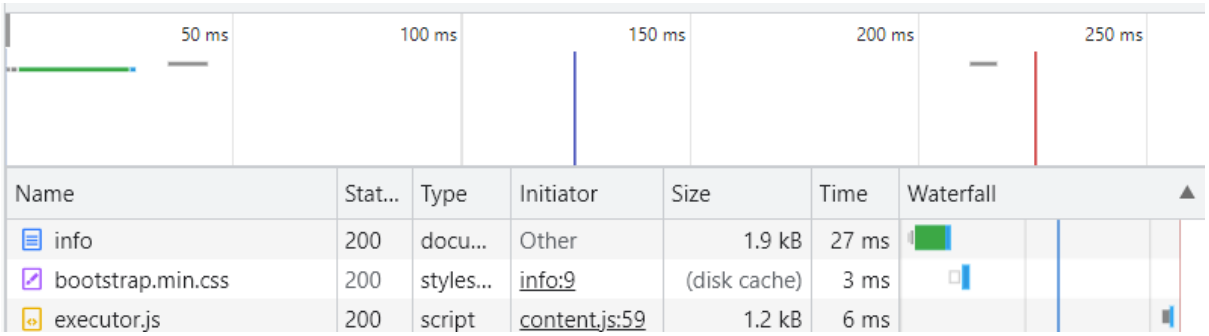
Desafío Nro. 14:

Uso de Compression (Gzip): Incorporamos a nuestro proyecto el uso de compresión y a continuación vemos como mejoro el rendimiento:

- *Utilizando Compression:* Size 1.0kb / Time 8ms



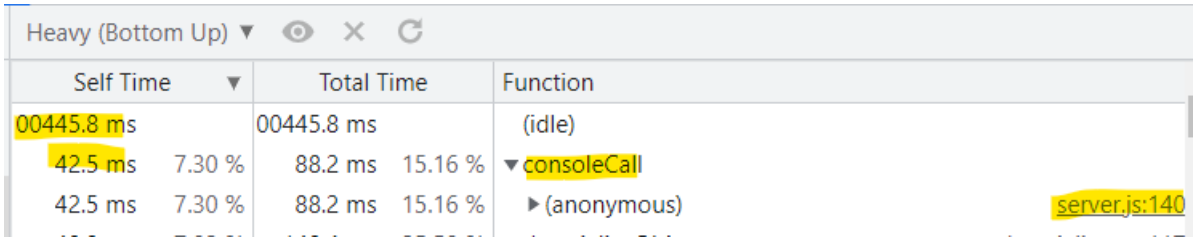
- *Sin utilizar Compression:* Size 1.9kb / Time 27ms



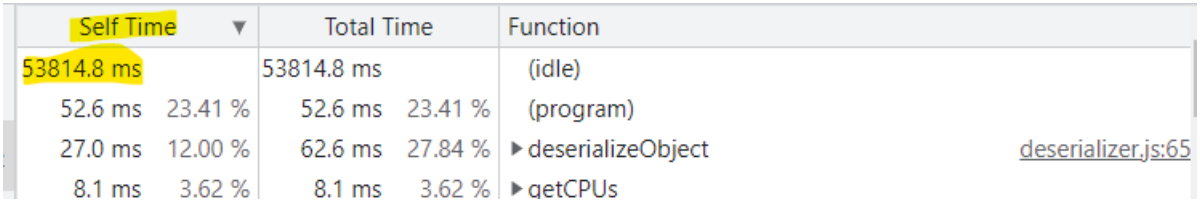
Conclusión: Mejora el rendimiento en un 47,37% y el tiempo de respuesta en un 70,4%.

Uso de Node Inspect Realizamos un perfilamiento del servidor por medio del node inspect en Google chrome, testeando la ruta /info y obteniendo los siguientes resultados:

- *Con tareas bloqueantes:*



- *Sin tareas bloqueantes:*



Conclusión: Vemos que al ejecutar el console.log se agrega una tarea bloqueante que consume 42.5ms.

Uso de Profiler (+ Artillery): Realizamos un perfilamiento del servidor, testeando la ruta /info y obteniendo los siguientes resultados:

- *Con tareas bloqueantes:* Incorporando un console.log utilizamos los siguientes comandos:
    1. node --prof server.js 8082 FORK
    2. artillery quick --count 20 -n 50 "http://localhost:8082/info" > result_bloq.txt
    3. node --prof-process bloq-v8.log > result_prof-bloq.txt

Vemos los siguientes resultados de rendimiento:

```
Statistical profiling result from bloq-v8.log, (22863 ticks, 0 unaccounted, 0 excluded).

 [Shared libraries]:
   ticks  total  nonlib   name
  22255   97.3%          C:\WINDOWS\SYSTEM32\ntdll.dll
    585    2.6%          C:\Program Files\nodejs\node.exe
      5    0.0%          C:\WINDOWS\System32\KERNELBASE.dll
      1    0.0%          C:\WINDOWS\System32\KERNEL32.DLL
```

```
http.response_time:
  min: ......................................................................... 20
  max: ......................................................................... 94
  median: ...................................................................... 24.8
  p95: ......................................................................... 58.6
  p99: ......................................................................... 58.6
http.responses: ................................................................ 20
vusers.created: ................................................................ 20
vusers.created_by_name.0: ...................................................... 20
```

- *Con tareas NO bloqueantes:* utilizamos los siguientes comandos:
    4. node --prof server.js 8082 FORK
    5. artillery quick --count 20 -n 50 "http://localhost:8082/info" > result_nobloq.txt
    6. node --prof-process nobloq-v8.log > result_prof-nobloq.txt

Vemos los siguientes resultados de rendimiento:

```
Statistical profiling result from nobloq-v8.log, (6397 ticks, 0 unaccounted, 0 excluded).

 [Shared libraries]:
   ticks  total  nonlib   name
   5933   92.7%          C:\WINDOWS\SYSTEM32\ntdll.dll
    453    7.1%          C:\Program Files\nodejs\node.exe
      1    0.0%          C:\WINDOWS\System32\KERNELBASE.dll
      1    0.0%          C:\WINDOWS\System32\KERNEL32.DLL
```

```
http.response_time:
  min: ......................................................................... 9
  max: ......................................................................... 253
  median: ...................................................................... 10.9
  p95: ......................................................................... 228.2
  p99: ......................................................................... 228.2
http.responses: ................................................................ 20
vusers.created: ................................................................ 20
```

Conclusión: Baja la cantidad de ticks en un 72% y la mediana de tiempo de espera en un 56%

Uso de Autocannon y 0x Realizamos un análisis del servidor con autocannon y 0x, testeando la ruta /info y obteniendo los siguientes resultados:

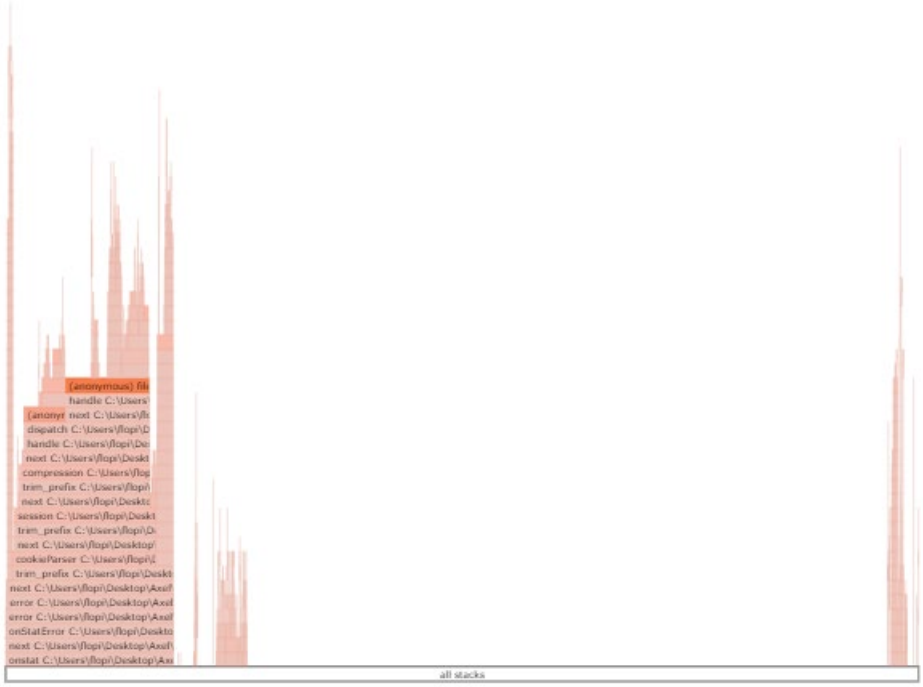
- *Con tareas bloqueantes:*

Stat	2.5%	50%	97.5%	99%	Avg	Stdev	Max
Latency	233 ms	314 ms	511 ms	534 ms	330.63 ms	74.59 ms	583 ms

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	202	202	301	391	299.95	55.13	202
Bytes/Sec	381 kB	381 kB	568 kB	738 kB	566 kB	104 kB	381 kB

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

6k requests in 20.11s, 11.3 MB read
```



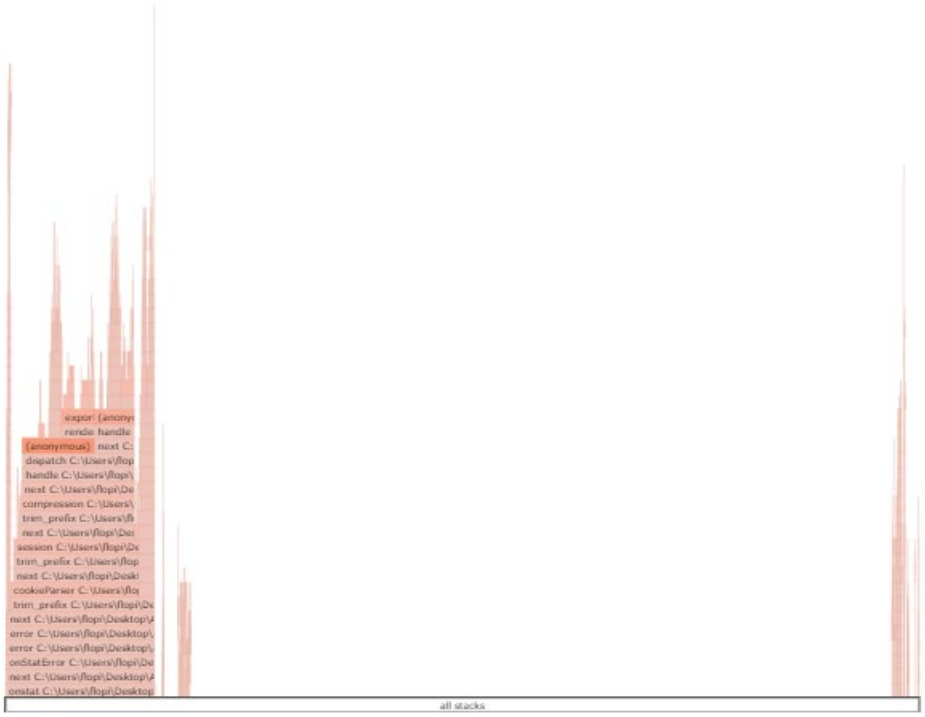
- *Sin tareas bloqueantes:*

Stat	2.5%	50%	97.5%	99%	Avg	Stdev	Max
Latency	148 ms	316 ms	425 ms	434 ms	314.01 ms	63.08 ms	442 ms

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	248	248	313	471	316.05	51.01	248
Bytes/Sec	468 kB	468 kB	591 kB	889 kB	596 kB	96.2 kB	468 kB

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

6k requests in 20.12s, 11.9 MB read
```



Conclusión: En promedio el tiempo de ejecución disminuye un 5% y se leen un 5,3% kb/s más.

Conclusión final: El rendimiento y desempeño del servidor se ve sumamente mejorado si aplicamos:

- Compresión.
- Trabajamos en modo Clúster.
- Evitamos tareas bloqueantes.