

- Artillery en línea de comandos, emulando 50 conexiones concurrentes con 20 request por cada una:

Artillery sin console.log

```
ntrega16 > artillery_sin_console.txt
1 Phase started: unnamed (index: 0, duration: 1s) 13:51:25(-0600)
2
3 Phase completed: unnamed (index: 0, duration: 1s) 13:51:26(-0600)
4
5 -----
6 Metrics for period to: 13:51:30(-0600) (width: 0.997s)
7 -----
8
9 http.codes.200: ..... 1000
10 http.request_rate: ..... 1000/sec
11 http.requests: ..... 1000
12 http.response_time:
13 | min: ..... 0
14 | max: ..... 16
15 | median: ..... 1
16 | p95: ..... 2
17 | p99: ..... 3
18 http.responses: ..... 1000
19 vusers.completed: ..... 50
20 vusers.created: ..... 50
21 vusers.created_by_name.0: ..... 50
22 vusers.failed: ..... 0
23 vusers.session_length:
24 | min: ..... 17.4
25 | max: ..... 85.5
26 | median: ..... 21.5
27 | p95: ..... 47
28 | p99: ..... 68.7
29
```

El profiling:

```
ntrega16 > result_prof_sin.txt
17
18 [Summary]:
19 | ticks total nonlib name
20 | 7 0.4% 100.0% JavaScript
21 | 0 0.0% 0.0% C++
22 | 12 0.7% 171.4% GC
23 | 1608 99.6% Shared libraries
24
```

Artillery con console.log

```
triga16 > ≡ artillery_con_console.txt
1 Phase started: unnamed (index: 0, duration: 1s) 13:52:21(-0600)
2
3 Phase completed: unnamed (index: 0, duration: 1s) 13:52:22(-0600)
4
5 -----
6 Metrics for period to: 13:52:30(-0600) (width: 1.673s)
7 -----
8
9 http.codes.200: ..... 1000
10 http.request_rate: ..... 605/sec
11 http.requests: ..... 1000
12 http.response_time:
13 | min: ..... 2
14 | max: ..... 93
15 | median: ..... 40
16 | p95: ..... 55.2
17 | p99: ..... 63.4
18 http.responses: ..... 1000
19 vusers.completed: ..... 50
20 vusers.created: ..... 50
21 vusers.created_by_name.0: ..... 50
22 vusers.failed: ..... 0
23 vusers.session_length:
24 | min: ..... 288.2
25 | max: ..... 949.4
26 | median: ..... 788.5
27 | p95: ..... 944
28 | p99: ..... 944
29
```

El profiling:

```
6 > ≡ result_prof_con.txt
| ticks total nonlib name
|
[Summary]:
| ticks total nonlib name
| 7 0.4% 100.0% JavaScript
| 0 0.0% 0.0% C++
| 12 0.7% 171.4% GC
| 1714 99.6% Shared libraries
```

- Autocannon en línea de comandos, emulando 100 conexiones concurrentes realizadas en un tiempo de 20 segundos.

Autocannon sin console.log

```
C:\Users\Pablo\Desktop\Courses\CoderBackend\entrega16>autocannon -c 100 -d 20 "http://localhost:8080/info-json"
Running 20s test @ http://localhost:8080/info-json
100 connections
```

Stat	2.5%	50%	97.5%	99%	Avg	Stdev	Max
Latency	27 ms	31 ms	43 ms	50 ms	30.63 ms	4.23 ms	76 ms

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	2321	2321	3285	3401	3226.6	219.06	2321
Bytes/Sec	1.24 MB	1.24 MB	1.75 MB	1.81 MB	1.72 MB	117 kB	1.24 MB

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

```
65k requests in 20.11s, 34.4 MB read
```

Autocannon con console.log

```
C:\Users\Pablo\Desktop\Courses\CoderBackend\entrega16>autocannon -c 100 -d 20 "http://localhost:8080/info-json"
Running 20s test @ http://localhost:8080/info-json
100 connections
```

Stat	2.5%	50%	97.5%	99%	Avg	Stdev	Max
Latency	29 ms	32 ms	50 ms	55 ms	32.61 ms	5.54 ms	106 ms








Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	1827	1827	3099	3209	3026.1	303.7	1827
Bytes/Sec	972 kB	972 kB	1.65 MB	1.71 MB	1.61 MB	162 kB	972 kB

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


```
61k requests in 20.07s, 32.2 MB read
```

- 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.

Inspect sin console log

Connection		Console		Profiler		Sources		Memory	
 				Heavy (Bottom Up) ▾				 	
Profiles				Self Time		Total Time		Function	
				16019.9 ms		16019.9 ms		(idle)	
CPU PROFILES				72.2 ms 9.04 %		72.2 ms 9.04 %		▶ writeUtf8String	
				51.1 ms 6.40 %		51.1 ms 6.40 %		(garbage collector)	
 Profile 1 Save				37.2 ms 4.66 %		37.2 ms 4.66 %		(program)	
				27.6 ms 3.46 %		79.5 ms 9.95 %		▶ initialize	
				20.3 ms 2.54 %		20.3 ms 2.54 %		▶ writev	
				16.9 ms 2.12 %		29.3 ms 3.66 %		▶ nextTick	
				14.6 ms 1.83 %		42.9 ms 5.37 %		▶ hash	
				11.1 ms 1.33 %		11.1 ms 1.33 %		▶ ...	

Inspect con console log

		Heavy (Bottom Up) ▾			
Profiles	Self Time		Total Time		Function
CPU PROFILES	19609.5 ms		19609.5 ms		(idle)
 Profile 1 <u>Save</u>	1870.2 ms	39.77 %	1870.2 ms	39.77 %	▶ writeUtf8String
	1406.9 ms	29.92 %	3372.4 ms	71.72 %	▶ consoleCall
	100.6 ms	2.14 %	100.6 ms	2.14 %	(garbage collector)
	70.0 ms	1.49 %	70.0 ms	1.49 %	▶ getColorDepth
	54.9 ms	1.17 %	54.9 ms	1.17 %	(program)
	44.2 ms	0.94 %	44.2 ms	0.94 %	▶ writev

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



Conclusiones:

- Agregar el gzip disminuye la cantidad de información que se envía al comprimir los datos.
- Cuando se añaden procesos bloqueantes, como console.log, el tiempo de respuesta de las peticiones se incrementa como se puede observar con las pruebas de artillery, autocannon y 0x.