

Test de compression GZIP:

Ruta `"/info"` **con** Compression:

Recursos: 448 B

Ruta `"/info"` **sin** Compression:

Recursos: 459 B

Tests de Artillery:

Comparación de `fork` vs `cluster`:

<pre>http.codes.200: 1000 http.request_rate: 561/sec http.requests: 1000 http.response_time: min: 4 max: 107 median: 49.9 p95: 87.4 p99: 76 http.responses: 1000 vusers.completed: 50 vusers.created: 50 vusers.created_by_name.0: 50 vusers.failed: 0 vusers.session_length: min: 707.7 max: 1143.5 median: 1002.4 p95: 1130.2 p99: 1130.2</pre>	<pre>http.codes.200: 1000 http.request_rate: 791/sec http.requests: 1000 http.response_time: min: 2 max: 46 median: 21.1 p95: 32.1 p99: 37 http.responses: 1000 vusers.completed: 50 vusers.created: 50 vusers.created_by_name.0: 50 vusers.failed: 0 vusers.session_length: min: 115.2 max: 556.7 median: 441.5 p95: 550.1 p99: 550.1</pre>
---	--

Fork

Cluster

Comparación modo `fork` incluyendo logeo de respuesta en terminal:

<pre>http.codes.200: 1000 http.request_rate: 842/sec http.requests: 1000 http.response_time: min: 2 max: 32 median: 13.9 p95: 22 p99: 25.8 http.responses: 1000 vusers.completed: 50 vusers.created: 50 vusers.created_by_name.0: 50 vusers.failed: 0 vusers.session_length: min: 172.9 max: 370.6 median: 301.9 p95: 361.5 p99: 368.8</pre>	<pre>http.codes.200: 895 http.request_rate: 312/sec http.requests: 923 http.response_time: min: 13 max: 242 median: 133 p95: 159.2 p99: 165.7 http.responses: 895 vusers.completed: 22 vusers.created: 50 vusers.created_by_name.0: 50 vusers.failed: 0 vusers.session_length: min: 1321.4 max: 2557.4 median: 2276.1 p95: 2515.5 p99: 2566.3</pre>
--	---

Sin Log

Con Log

Tests de Profile:

Comparación modo `fork` incluyendo logeo de respuesta en terminal:

<pre>[Shared libraries]: ticks total nonlib name 1673 90.6% C:\WINDOWS\SYSTEM32\ntdll.dll 166 9.0% C:\Program Files\nodejs\node.exe 2 0.1% C:\WINDOWS\System32\KERNEL32.DLL [JavaScript]: ticks total nonlib name 2 0.1% 40.0% JS: ^next C:\Users\Federico\Desktop\CODER-ENTREGAS 1 0.1% 20.0% JS: ^update node:internal/crypto/hash:99:40 1 0.1% 20.0% JS: ^configure C:\Users\Federico\Desktop\CODER-ENT 1 0.1% 20.0% JS: ^authenticate C:\Users\Federico\Desktop\CODER- [Summary]: ticks total nonlib name 5 0.3% 100.0% JavaScript 0 0.0% 0.0% C++ 6 0.3% 120.0% GC 1841 99.7% Shared libraries</pre>	<pre>[Shared libraries]: ticks total nonlib name 1951 89.2% C:\WINDOWS\SYSTEM32\ntdll.dll 233 10.6% C:\Program Files\nodejs\node.exe 1 0.0% C:\WINDOWS\System32\KERNELBASE.dll [JavaScript]: ticks total nonlib name 1 0.0% 33.3% JS: ^matchHeader node:http_outgoing:595:21 1 0.0% 33.3% JS: ^emit node:events:459:44 1 0.0% 33.3% JS: ^<anonymous> node:internal/fs/utils:359:35 [Summary]: ticks total nonlib name 3 0.1% 100.0% JavaScript 0 0.0% 0.0% C++ 6 0.3% 200.0% GC 2185 99.9% Shared libraries</pre>
---	---

Sin Log

Con Log

Captura de la terminal test de Autocannon:

```
> node benchmark.js

Corriendo el benchmark...
Running 20s test @ http://localhost:8080/info
100 connections

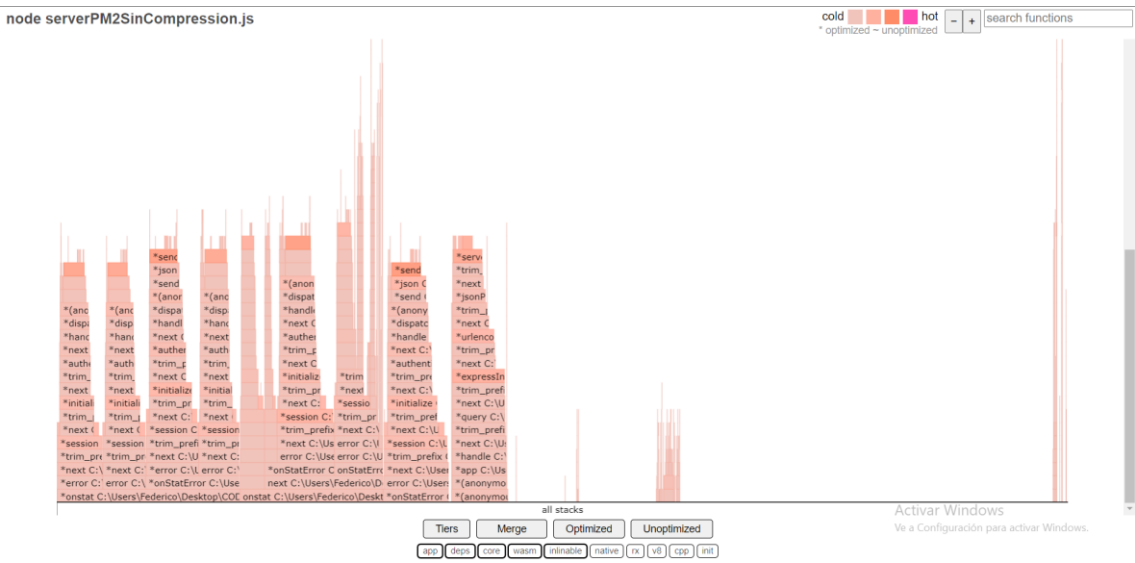
| Stat | 2.5% | 50% | 97.5% | 99% | Avg | St |
| dev | Max |      |      |      |      |    | |
|---|---|---|---|---|---|---|---|
| Latency | 37 ms | 42 ms | 69 ms | 76 ms | 45.49 ms | 8.64 ms | 122 ms |
|-----|-----|-----|-----|-----|-----|

| Stat | 1% | 2.5% | 50% | 97.5% | Avg |
| Stdev | Min |      |      |      |      | | |
|---|---|---|---|---|---|---|---|
| Req/Sec | 1295 | 1295 | 2211 | 2391 | 2175.1 | 235.94 | 1295 |
| Bytes/Sec | 889 kB | 889 kB | 1.52 MB | 1.64 MB | 1.5 MB | 163 kB | 888 kB |

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

44k requests in 20.06s, 29.9 MB read
```

Captura del diagrama flama de 0x:



Test de inspect con Chrome:

Dentro de “run” en el inspect se destaca la siguiente operación sincrónica:

```
74
75   for (var i = 0; i < generateAttempts; i++) {
76     try {
77       return crypto.randomBytes(size)
78     } catch (e) {
79       err = e
80     }
81   }
82
```

Conclusión:

De los test comparativos se puede apreciar como una operación síncrona eleva notablemente los recursos consumidos y el tiempo de respuesta en operaciones tan simple como las de la ruta “/info”.

También se puede ver como mejora la performance cuando se ejecuta en cluster en comparación con un fork.

Los resultados del testeo de compresión no fueron muy contundentes, probablemente porque el mensaje que se evaluó era bastante corto y plano, pero si se ve una leve mejora.