



## 1- Incorporar al proyecto de servidor de trabajo la compresión gzip.

```
const compression = require('compression')
router.use(compression())
```

## 2- Verificar sobre la ruta /info con y sin compresión, la diferencia de cantidad de bytes devueltos en un caso y otro.

- Sin compresión: 857B

Name	Status	Type	Initiator	Size	Time	Waterfall
 info	200	document	Other	857 B	55 ms	

- Con compresión: 856B

Name	Status	Type	Initiator	Size	Time	Waterfall
 info	200	document	Other	856 B	10 ms	

## 3- Luego implementar logggeo (con alguna librería vista en clase) que registre lo siguiente:

Realizado con winston

```
const buildProdLogger = () => {
  const prodLogger = winston.createLogger({
    transports: [
      new winston.transports.File({ filename: "warn.log", level: "warn" }),
      new winston.transports.File({ filename: "error.log", level: "error" }),
    ],
  });

  return prodLogger;
};

const buildDevLogger = () => {
  const devLogger = winston.createLogger({
    transports: [new winston.transports.Console({ level: "info" })],
  });

  return devLogger;
};
```

4- **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.

Prender el servidor en modo prof

```
PS C:\Users\Kevin\Documents\Full Stack\04. Backend\clase 32\clase-32> node --prof server.js -- 8081
Base Firebase Conectada
```

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.

Archivo artillery

```
PS C:\Users\Kevin\Documents\Full Stack\04. Backend\clase 32\clase-32> artillery quick --count 50 -n 20 http://localhost:8081/info > result_fork.txt
PS C:\Users\Kevin\Documents\Full Stack\04. Backend\clase 32\clase-32> node --prof-process prof_info.log > result_prof.txt
```

--prof-process

```
PS C:\Users\Kevin\Documents\Full Stack\04. Backend\clase 32\clase-32> node --prof-process prof_info.log > result_prof.txt
(node:16300) ExperimentalWarning: VM Modules is an experimental feature. This feature could change at any time
(Use 'node --trace-warnings ...' to show where the warning was created)
Could not find function 0x22d23be90e8
Could not find function 0x22d23be90e8
Could not find function 0x22d23be9560
Could not find function 0x2a1b5fda8e0
Could not find function 0x22d23be95b0
Could not find function 0x22d23be9560
Could not find function 0x22d23be90e8
Could not find function 0x22d23be9560
Could not find function 0x2a1b5fda8e0
Could not find function 0x22d23be90e8
Could not find function 0x22d23be9560
Could not find function 0x2a1b5fda8e0
Could not find function 0x22d23be95b0
Could not find function 0x237e789f7d8
Could not find function 0x237e78a5378
Could not find function 0x2247a30fca0
Could not find function 0x22d23be9850
Could not find function 0x22d23be9560
Could not find function 0x22d23be90e8
Could not find function 0x22d23be9560
Could not find function 0x2a1b5fda8e0
Could not find function 0x237e789f648
Could not find function 0x22d23be9850
Could not find function 0x22d23be9560
Could not find function 0x237e789faa8
Could not find function 0x237e789f968
Could not find function 0x237e789f8c8
Could not find function 0x237e789f918
Could not find function 0x22d23be9850
Could not find function 0x22d23be9560
Could not find function 0x22d23bfc880
Could not find function 0x237e789a040
Could not find function 0x237e78a04a0
Could not find function 0x2a1b5fd8098
Could not find function 0x237e789f6e8
Could not find function 0x237e789f698
Could not find function 0x237e789f9b8
```

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)

Pantallazo de parte del reporte:

```

Path de ejecución: 'C:\\Program Files\\nodejs\\node.exe'
Process id: 9972
Carpeta del proyecto: C:\\Users\\Kevin\\Documents\\Full Stack\\04. Backend\\clase 32\\clase-32
Procesadores presentes: 9972
Titulo del proceso: Administrador: C:\\WINDOWS\\system32\\cmd.exe
Sistema operativo: win32
Version de Node: v16.13.0
Memoria total reservada: {
  rss: 273453056,
  heapTotal: 234332160,
  heapUsed: 202086528,
  external: 25789288,
  arrayBuffers: 18548620
}

Path de ejecución: 'C:\\Program Files\\nodejs\\node.exe'
Process id: 9972
Carpeta del proyecto: C:\\Users\\Kevin\\Documents\\Full Stack\\04. Backend\\clase 32\\clase-32
Procesadores presentes: 9972
Titulo del proceso: Administrador: C:\\WINDOWS\\system32\\cmd.exe
Sistema operativo: win32
Version de Node: v16.13.0
Memoria total reservada: {
  rss: 273653760,
  heapTotal: 234332160,
  heapUsed: 202146432,
  external: 25789312,
  arrayBuffers: 18548644
}

Path de ejecución: 'C:\\Program Files\\nodejs\\node.exe'
Process id: 9972
Carpeta del proyecto: C:\\Users\\Kevin\\Documents\\Full Stack\\04. Backend\\clase 32\\clase-32
Procesadores presentes: 9972
Titulo del proceso: Administrador: C:\\WINDOWS\\system32\\cmd.exe
Sistema operativo: win32
Version de Node: v16.13.0
Memoria total reservada: {
  rss: 272838656,
  heapTotal: 234332160,
  heapUsed: 202206240,
  external: 25789336,
  arrayBuffers: 18548668
}

```

```

PS C:\\Users\\Kevin\\Documents\\Full Stack\\04. Backend\\clase 32\\clase-32> npm test
> tarea-8@1.0.0 test
> node benchmark.js

```

```

Running all benchmarks in parallel ...
Running 20s test @ http://localhost:8081/info
100 connections

```

Stat	2.5%	50%	97.5%	99%	Avg	Stdev	Max
Latency	94 ms	177 ms	429 ms	497 ms	198.55 ms	80.19 ms	657 ms

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	275	275	510	676	501.1	101.86	275
Bytes/Sec	236 kB	236 kB	437 kB	580 kB	429 kB	87.4 kB	235 kB

```

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

```

```

10k requests in 20.12s, 8.58 MB read

```

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.

## Prender el servidor

```
PS C:\Users\Kevin\Documents\Full Stack\04. Backend\clase 32\clase-32> node --inspect server.js -- 8081 fork
Debugger listening on ws://127.0.0.1:9229/668f430e-6e01-4aec-919c-18a11524f09a
For help, see: https://nodejs.org/en/docs/inspector
Base Firebase Conectada
Debugger attached.
```

## Pantallazo de Chrome://inspect

Self Time	Total Time	Function
00509.6 ms	00509.6 ms	(idle)
31.3 ms	21.43 %	78.3 ms 53.57 % ▶ deserializeObject <a href="#">deserializers:117</a>
21.0 ms	14.37 %	21.0 ms 14.37 % (program)
5.9 ms	4.05 %	11.0 ms 7.54 % ▶ serialize <a href="#">bson.ts:137</a>
5.4 ms	3.73 %	6.0 ms 4.13 % ▶ writeBuffer
4.4 ms	3.02 %	4.4 ms 3.02 % ▶ Long <a href="#">long.ts:136</a>
3.5 ms	2.38 %	5.1 ms 3.49 % ▶ nextTick <a href="#">node:internal/p...ask queues:104</a>
3.2 ms	2.22 %	88.1 ms 60.32 % callbackTrampoline <a href="#">node:internal/async_hooks:118</a>
3.2 ms	2.22 %	4.3 ms 2.94 % emitHook <a href="#">node:internal/async_hooks:228</a>
2.6 ms	1.75 %	2.6 ms 1.75 % ▶ utf8Slice
2.3 ms	1.59 %	4.9 ms 3.33 % ▶ slice <a href="#">node:buffer:593</a>
1.9 ms	1.27 %	3.2 ms 2.22 % ▶ emitInitScript <a href="#">node:internal/async_hooks:485</a>
1.7 ms	1.19 %	83.4 ms 57.06 % ▶ onStreamRead <a href="#">node:internal/s...se commons:171</a>
1.6 ms	1.11 %	1.6 ms 1.11 % (garbage collector)
1.6 ms	1.11 %	76.0 ms 51.98 % ▶ processIncomingData <a href="#">message_stream.ts:135</a>
1.6 ms	1.11 %	68.4 ms 46.83 % ▶ (anonymous) <a href="#">connection.ts:256</a>
1.6 ms	1.11 %	90.1 ms 61.67 % ▶ addChunk <a href="#">node:internal/s...s/readable:304</a>
1.5 ms	1.03 %	6.0 ms 4.13 % ▶ serializeInto <a href="#">serializer.ts:750</a>
1.4 ms	0.95 %	1.7 ms 1.19 % ▶ slice <a href="#">node:buffer:1115</a>
1.3 ms	0.87 %	51.6 ms 35.32 % ▼ parse <a href="#">commands.ts:643</a>
1.3 ms	0.87 %	51.6 ms 35.32 % ▶ onMessage <a href="#">connection.ts:381</a>
1.3 ms	0.87 %	5.2 ms 3.57 % ▶ Long.comp <a href="#">long.ts:420</a>
1.0 ms	0.71 %	1.0 ms 0.71 % ▶ FastBuffer <a href="#">node:internal/buffer:958</a>
1.0 ms	0.71 %	171.2 ms 117.14 % ▶ emit <a href="#">node:events:340</a>
1.0 ms	0.71 %	66.8 ms 45.71 % ▶ onMessage <a href="#">connection.ts:381</a>
1.0 ms	0.71 %	6.3 ms 4.29 % ▶ serverUpdateHandler <a href="#">topology.ts:608</a>
1.0 ms	0.71 %	3.8 ms 2.62 % ▶ Long.compare <a href="#">long.ts:403</a>
0.9 ms	0.63 %	5.1 ms 3.49 % processTicksAndRejections <a href="#">node:internal/p...task queues:68</a>
0.9 ms	0.63 %	3.4 ms 2.30 % ▶ onwrite <a href="#">node:internal/s...s/writable:425</a>

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

```
PS C:\Users\Kevin\Documents\Full Stack\04. Backend\clase 32\clase-32> npm start

> tarea-8@1.0.0 start
> 0x server.js

ProfilingBase Firebase Conectada
Flamegraph generated in
```

node server.js

cold   hot  
\* optimized ~ unoptimized   search functions



Conclusión: todos los procesos se encuentran en una escala de dos colores y se compone por líneas finas, por lo que no bloquean los subsiguientes elementos de manera significativa y lo que lo hace eficiente a su vez.