#### **COMPRESSION**

## Ruta /info sin compression

Se quita el middleware 'compression()'

Se ejecuta el servidor

Se ingresa a la ruta <a href="http://localhost:8080/info">http://localhost:8080/info</a>

Click derecho / Inspeccionar / Network / Se obtiene data

**SIZE: 655 B** 

## Ruta /info con compression

Se coloca el middleware 'compression()'

Se ejecta el servidor

Se ingresa a la ruta <a href="http://localhost:8080/info">http://localhost:8080/info</a>

Click derecho / Inspeccionar / Network / Se obtiene data

**SIZE: 677 B** 

# **ANÁLISIS DE PERFORMANCE**

1. Iniciamos el servidor en modo profiler (sin console.log en la ruta /info)

```
> node --prof server.js
```

- 2. Ingresamos con un navegador a <a href="http://localhost:8080/info">http://localhost:8080/info</a> para generar una petición
- 3. Realizar un test de carga con Artillery por linea de comando.

> artillery quick -c 50 -n 20 "http://localhost:8080/info" > artillery NObloq.txt

```
Phase started: unnamed (index: 0, duration: 1s) 02:19:59(-0300)
Phase completed: unnamed (index: 0, duration: 1s) 02:20:00(-0300)
Metrics for period to: 02:20:00(-0300) (width: 0.834s)
http.response_time:
p95: ..... 57.4
vusers.session length:
p99: .....
```

- 4. Cerramos la terminal y renombramos el archivo Isolate generado como "no bloq-v8.log"
- 5. Iniciamos el servidor en modo profiler (CON console.log en la ruta /info)

```
> node --prof server.js
```

- 6. Ingresamos con un navegador a <a href="http://localhost:8080/info">http://localhost:8080/info</a> para generar una petición
- 7. Realizar un test de carga con Artillery por linea de comando.

```
> artillery quick -c 50 -n 20 "http://localhost:8080/info" > artillery_bloq.txt
```

```
Phase started: unnamed (index: 0, duration: 1s) 02:27:06(-0300)
Phase completed: unnamed (index: 0, duration: 1s) 02:27:07(-0300)
Metrics for period to: 02:27:10(-0300) (width: 2.787s)
http.request_rate: ...... 364/sec
http.response_time:
vusers.failed: ...... 0
vusers.session length:
```

- 8. Cerramos la terminal y renombramos el archivo Isolate generado como "bloq-v8.log"
- 9. Decodificamos los dos archivos .log que se crearon.

```
> node.exe --prof-process bloq-v8.log > result_prof-bloq.txt
> node.exe --prof-process no_bloq-v8.log > result_prof-no_bloq.txt
```

#### Archivo result prof-bloq.txt

## Archivo result prof-no bloq.txt

## **AUTOCANNON**

Emular 100 conexiones en un periodo de tiempo de 20 segundos.

1. Configuramos el archivo *package.json* agregando los siguientes scripts:

```
"scripts": {
   "test": "node benchmark.js",
   "start": "0x server.js"
},
```

2. Abrimos una consola y ejecutamos

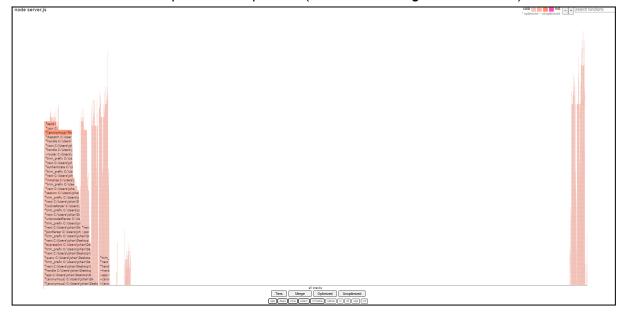
```
> npm start
```

3. Abrimos otra consola y ejecutamos

```
> npm test
```

4. Al finalizar el test cerramos la primera consola donde tiramos el start para poder terminar el proceso y generar la carpeta donde se encontrará el gráfico de flama.

Gráfico flama realizado con un proceso bloqueante (CON console.log en la ruta /info).



#### Resumen por consola de Autocannon

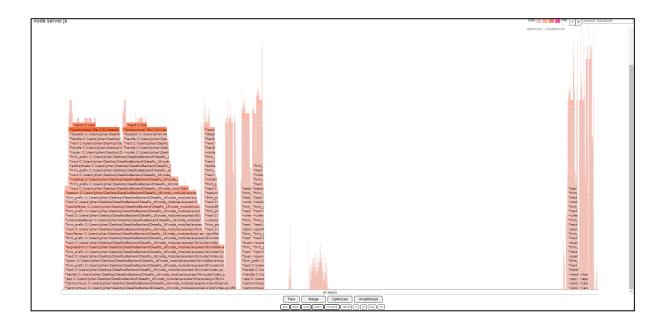
100 connections											
Stat	2.5%	50	3 <b>%</b> 9		7.5%		99%		/g	Stdev	Max
Latency	52 ms	115 ms		279 ms		369 ms		127.79 ms		59.63 ms	544 ms
Stat	1%		2.5%		50%		97.5%		Avg	Stdev	Min
Req/Sec	407		407		821		937		776.6	137.83	407
Bytes/Sec	267 I	кB	267 kB		538 kB		614 kB		509 kB	90.3 kB	267 kB
Req/Bytes counts sampled once per second. # of samples: 20											
16k requests in 20.07s, 10.2 MB read											

## Inspect

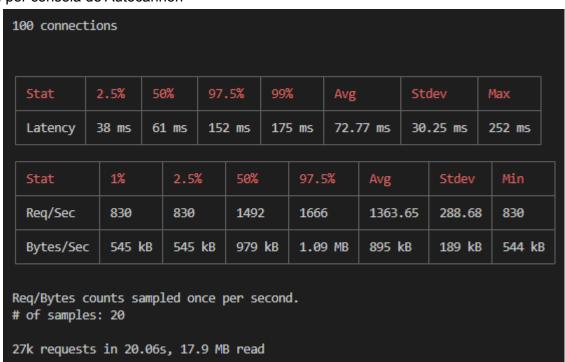
```
108
     1.0 ms router.get('/info', (req, res) => {
109
             const { url, method } = req;
110
     1.7 ms
      7.0 ms
               logger.info(`Petición recibida por el servidor. Ruta ${method} - ${url}`);
111
112
     3.4 ms
               const Argumentos = process.argv.slice(2);
               const Plataforma = process.platform;
113 0.1 ms
114
               const Version = process.version;
    1.8 ms
115
               const Memoria = process.memoryUsage().rss;
116 0.1 ms
               const Path = process.execPath;
117
               const Id = process.pid;
118 0.1 ms
               const Carpeta = process.cwd();
119 3.1 ms
               const numCPUs = os.cpus().length;
120
121 0.6 ms
               const datos = {
122
     0.1 ms
                 Argumentos: Argumentos,
123 0.9 ms
                   Pltataforma: `Sistema operativo ( SO ) - ${Plataforma}`,
                   Version: `Version de Node.js utilizada - ${Version}`,
124
125
     2.4 ms
                   Memoria: `Memoria total reservada ( RSS ) - ${Memoria}`,
126
                   Path: `Path de ejecución - ${Path}`,
     0.4 ms
                   CPUs: `Cantidad de procesadores presentes en el servidor - ${numCPUs}`,
127
     1.0 ms
128 1.5 ms
                    Id: `Process ID - ${Id}`,
129
                    Carpeta: `Carpeta del proyecto - ${Carpeta}`,
130
                };
131
132
     27.3 ms
              console.log('Aqui van los datos');
133
    21.1 ms
                console.log(datos);
134
    50.7 ms
135
               res.json(datos);
    0.1 ms });
136
137
```

Repetimos el proceso pero esta vez sin tener un proceso bloqueante (sin console.log en la ruta /info).

#### Gráfico flama



# Resumen por consola de Autocannon



Inspect

```
108
109
       4.5 ms router.get('/info', (req, res) => {
110
       4.9 ms
                 const { url, method } = req;
                 logger.info(`Petición recibida por el servidor. Ruta ${method} - ${url}`);
111
      8.1 ms
       7.5 ms
112
                 const Argumentos = process.argv.slice(2);
113
       0.5 ms
                 const Plataforma = process.platform;
114
       0.2 ms
                 const Version = process.version;
115
       3.1 ms
                 const Memoria = process.memoryUsage().rss;
116
       0.1 ms
                 const Path = process.execPath;
117
       0.1 ms
                 const Id = process.pid;
118
       0.2 ms
                 const Carpeta = process.cwd();
                 const numCPUs = os.cpus().length;
119
       5.5 ms
120
121 0.2 ms
                 const datos = {
122
      0.5 ms
                     Argumentos: Argumentos,
                      Pltataforma: `Sistema operativo ( SO ) - ${Plataforma}`,
123
       4.8 ms
124
      0.8 ms
                      Version: `Version de Node.js utilizada - ${Version}`,
125
       2.9 ms
                     Memoria: `Memoria total reservada ( RSS ) - ${Memoria}`,
                     Path: `Path de ejecución - ${Path}`,
CPUs: `Cantidad de procesadores presentes en el servidor - ${numCPUs}`,
126
       0.6 ms
127
       1.0 ms
                      Id: `Process ID - ${Id}`,
128
       5.4 ms
                      Carpeta: `Carpeta del proyecto - ${Carpeta}`,
129
      0.3 ms
130
                  };
131
      93.5 ms
                  res.json(datos);
132 0.2 ms });
```

#### **CONCLUSIONES:**

En *Artillery* podemos apreciar claramente una media en el tiempo de respuesta de 29.1 para el proceso no bloqueante frente a una media de 85.6 para el proceso bloqueante.

Con *Autocannon* se aprecia claramente el aumento de latencia y disminución en la speticiones por segundos. Siento 127.79ms de media en latencia para cuando hay procesos bloqueantes frente a 72.77 de media de latencias para procesos no bloqueantes.

La cantidad de request atentidas fue de 407 y 830 respectivamente. Dejando en evidencia que se atiende a poco más del doble de solicitudes cuando no hay proceso bloqueante.