

COMPRESSION

Ruta /info sin compression

Se quita el middleware 'compression()'

Se ejecuta el servidor

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

Click derecho / Inspeccionar / Network / Se obtiene data

SIZE: 655 B

Ruta /info con compression

Se coloca el middleware 'compression()'

Se ejecuta el servidor

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

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 <http://localhost:8080/info> 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.codes.200: ..... 353
http.request_rate: ..... 378/sec
http.requests: ..... 378
http.response_time:
  min: ..... 3
  max: ..... 80
  median: ..... 29.1
  p95: ..... 57.4
  p99: ..... 68.7
http.responses: ..... 353
vusers.completed: ..... 8
vusers.created: ..... 33
vusers.created_by_name.0: ..... 33
vusers.failed: ..... 0
vusers.session_length:
  min: ..... 273.3
  max: ..... 665.8
  median: ..... 399.5
  p95: ..... 645.6
  p99: ..... 645.6
```

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 <http://localhost:8080/info> para generar una petición

7. Realizar un test de carga con Artillery por línea 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.codes.200: ..... 1000
http.request_rate: ..... 364/sec
http.requests: ..... 1000
http.response_time:
  min: ..... 3
  max: ..... 188
  median: ..... 85.6
  p95: ..... 135.7
  p99: ..... 169
http.responses: ..... 1000
vusers.completed: ..... 50
vusers.created: ..... 50
vusers.created_by_name.0: ..... 50
vusers.failed: ..... 0
vusers.session_length:
  min: ..... 752.7
  max: ..... 1996.6
  median: ..... 1863.5
  p95: ..... 1978.7
  p99: ..... 1978.7
```

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**

```
Statistical profiling result from bloq-v8.log, (3794 ticks, 0 unaccounted, 0 excluded).

[Shared libraries]:
  ticks total nonlib   name
  3458  91.1%         C:\Windows\SYSTEM32\ntdll.dll
   327   8.6%         C:\Program Files\nodejs\node.exe
     1   0.0%         C:\Windows\System32\KERNELBASE.dll
```

Archivo **result_prof-no_bloq.txt**

Statistical profiling result from no_bloq-v8.log, (10073 ticks, 0 unaccounted, 0 excluded).

```
[Shared libraries]:
```

ticks	total	nonlib	name
9690	96.2%		C:\Windows\SYSTEM32\ntdll.dll
372	3.7%		C:\Program Files\nodejs\node.exe

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%	97.5%	99%	Avg	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 kB	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		
109	1.0 ms	router.get('/info', (req, res) => {
110	1.7 ms	const { url, method } = req;
111	7.0 ms	logger.info(`Petición recibida por el servidor. Ruta \${method} - \${url}`);
112	3.4 ms	const Argumentos = process.argv.slice(2);
113	0.1 ms	const Plataforma = process.platform;
114		const Version = process.version;
115	1.8 ms	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	Plataforma: `Sistema operativo (SO) - \${Plataforma}`,
124		Version: `Version de Node.js utilizada - \${Version}`,
125	2.4 ms	Memoria: `Memoria total reservada (RSS) - \${Memoria}`,
126	0.4 ms	Path: `Path de ejecución - \${Path}`,
127	1.0 ms	CPUs: `Cantidad de procesadores presentes en el servidor - \${numCPUs}`,
128	1.5 ms	Id: `Process ID - \${Id}`,
129		Carpeta: `Carpeta del proyecto - \${Carpeta}`,
130		};
131		
132	27.3 ms	console.log('Aquí van los datos');
133	21.1 ms	console.log(datos);
134		
135	50.7 ms	res.json(datos);
136	0.1 ms	});
137		

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

Gráfico flama

108		
109	4.5 ms	router.get('/info', (req, res) => {
110	4.9 ms	const { url, method } = req;
111	8.1 ms	logger.info(`Petición recibida por el servidor. Ruta \${method} - \${url}`);
112	7.5 ms	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();
119	5.5 ms	const numCPUs = os.cpus().length;
120		
121	0.2 ms	const datos = {
122	0.5 ms	Argumentos: Argumentos,
123	4.8 ms	Plataforma: `Sistema operativo (SO) - \${Plataforma}`,
124	0.8 ms	Version: `Version de Node.js utilizada - \${Version}`,
125	2.9 ms	Memoria: `Memoria total reservada (RSS) - \${Memoria}`,
126	0.6 ms	Path: `Path de ejecución - \${Path}`,
127	1.0 ms	CPUs: `Cantidad de procesadores presentes en el servidor - \${numCPUs}`,
128	5.4 ms	Id: `Process ID - \${Id}`,
129	0.3 ms	Carpeta: `Carpeta del proyecto - \${Carpeta}`,
130		};
131	93.5 ms	res.json(datos);
132	0.2 ms	});
133		

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