

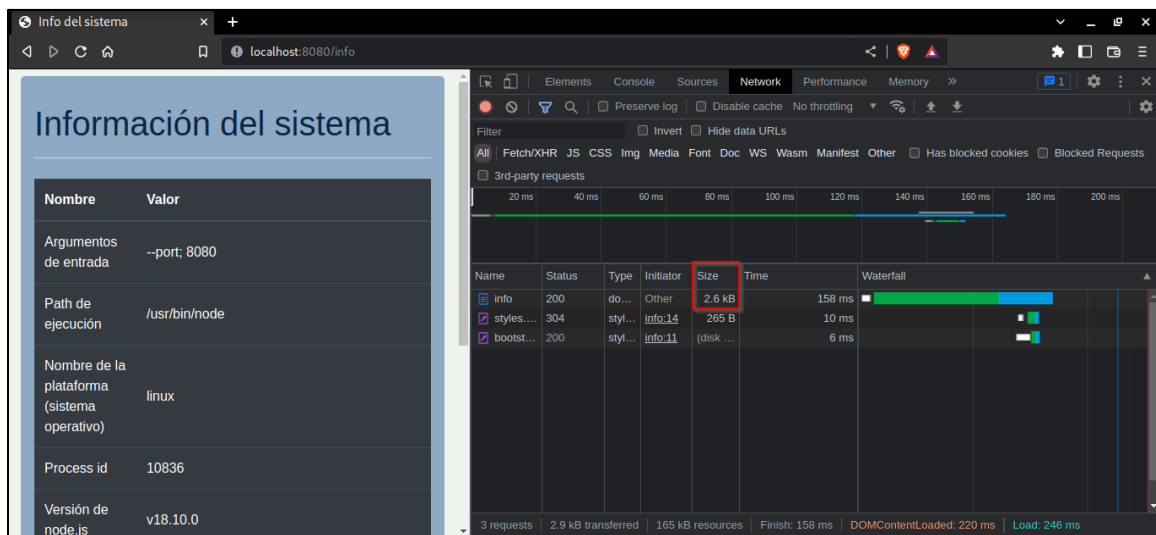


Informe de análisis completo de performance del servidor

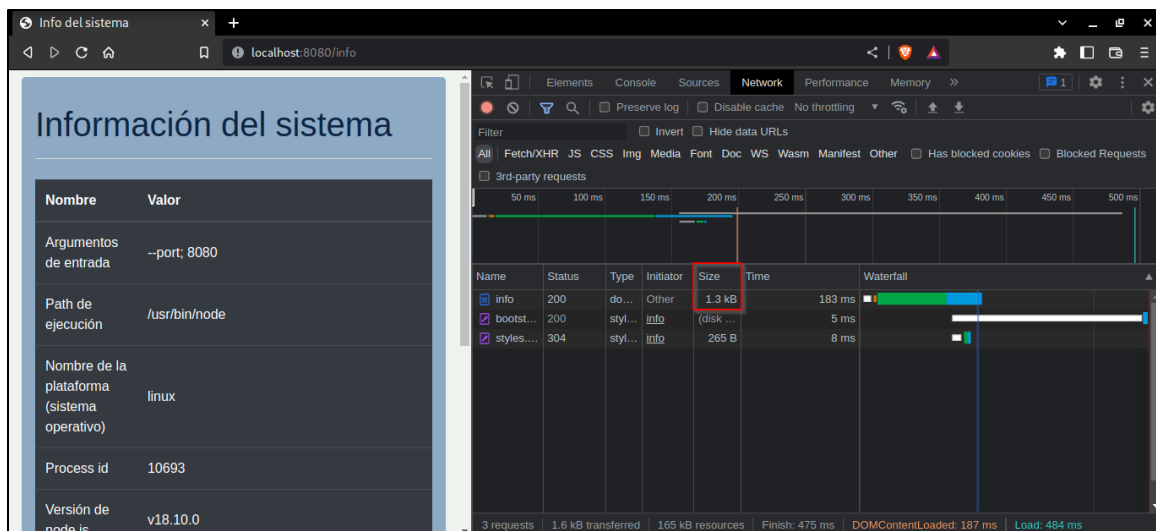
Curso: Programación Backend

Año 2022

Sin compresión con GZIP:



Con compresión con GZIP:



Para el análisis del servidor se van a ejecutar diferentes tipos de perfilamientos sobre dos variantes posibles, el primer caso será sin ninguna operación bloqueante sobre el fichero de la ruta info, y en el segundo caso se le agregarán una líneas con la sentencia console.log para generar operaciones bloqueantes.

Las diferencias se pueden apreciar a continuación:

Sin operaciones bloqueantes:

```
info.routes.js M X  randoms.routes.js M
src > routes > info.routes.js > [0] default
1  import { Router } from 'express';
2  import compression from "compression";
3  import { cpus } from 'os';
4  const router = Router();
5
6  router.get('/', compression(), (req, res) => {
7    const args = process.argv.slice(2).join(' ');
8    const info = {
9      args, //Argumentos de entrada
10     path: process.execPath, //Path de ejecución
11     os: process.platform, //Nombre de la plataforma (sistema operativo)
12     pid: process.pid, //Process id
13     nodeVersion: process.version, //Versión de node.js
14     dirPath: process.cwd(), //Carpeta del proyecto
15     memoryUsage: process.memoryUsage.rss() / 2 ** 20, //Memoria total reservada (rss) en MiB
16     numCPUs: cpus().length,
17   };
18   /* ----- Evaluación de rendimiento ----- */
19   // console.log(info.args);
20   // console.log(info.path);
21   // console.log(info.os);
22   // console.log(info.pid);
23   // console.log(info.nodeVersion);
24   // console.log(info.dirPath);
25   // console.log(info.memoryUsage);
26   // console.log(info.numCPUs);
27   /* ----- */
28   res.status(200).render('partials/viewInfo', {
29     info,
30   });
31 });
32
33 export default router;
```

Con operaciones bloqueantes (líneas descomentadas):

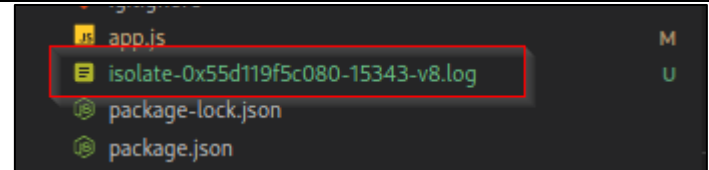
```
};
/* ----- Evaluación de rendimiento ----- */
console.log(info.args);
console.log(info.path);
console.log(info.os);
console.log(info.pid);
console.log(info.nodeVersion);
console.log(info.dirPath);
console.log(info.memoryUsage);
console.log(info.numCPUs);
/* ----- */
res.status(200).render('partials/viewInfo', {
  info
```

1) Perfilamiento con `--prof` nativo de NodeJS:

Para el primer caso se ejecutará el server con el siguiente comando por terminal:

```
(alexisvp97@kali)-[~/Escritorio/CursoBackend/Desafios/Desafio_14]
$ node --prof server.js --port 8080 FORK
```

Al ejecutar lo mostrado anteriormente se creará de manera automática un archivo cifrado que se verá de la siguiente manera:

	Cabe destacar que, al finalizar la ejecución del server, este fichero será renombrado con el siguiente nombre: sin_console.log y movido a una carpeta llamada analisiis .
---	---

Antes de finalizar el server se abrirá otra consola en el directorio del proyecto y se ejecutará la herramienta **artillery** realizando 50 conexiones concurrentes con 20 peticiones por cada una y el resultado de esto será almacenado en la carpeta **analisiis** en el fichero **info_sin_console.log.txt**

El comando para realizar esta operación es el siguiente:

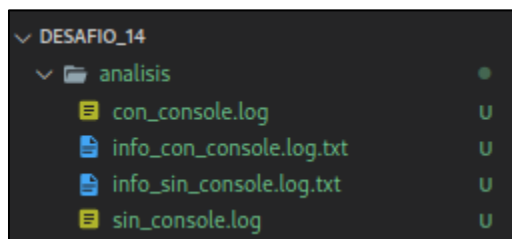
```
(alexisvp97@kali)-[~/Escritorio/CursoBackend/Desafios/Desafio_14]
$ artillery quick --count 20 -n 50 "http://localhost:8080/info" > analisis/info_sin_console.log.txt
```

Cuando finalice la ejecución de **artillery** ya se puede terminar el proceso del server y proceder a replicar los pasos con la otra variante.

Para el segundo caso, luego de haber descomentado las líneas que habilitan los procesos bloqueantes (`console.log`) cómo se vio en las imágenes presentadas anteriormente, será necesario realizar los mismos pasos que antes. La única diferencia es que el comando de **artillery** se mandará a otro archivo con un nombre diferente (**info_con_console.log.txt**):

```
(alexisvp97@kali)-[~/Escritorio/CursoBackend/Desafios/Desafio_14]
$ artillery quick --count 20 -n 50 "http://localhost:8080/info" > analisis/info_con_console.log.txt
```

Después de esto, los archivos generados por el análisis se verán de la siguiente manera:



Por último, es necesario procesar los ficheros .log ya que como se mencionó antes estos están encriptados. Esto se puede lograr con los siguientes comandos:

```
(alexisvp97@kali) ~/Escritorio/CursoBackend/Desafios/Desafio_14
$ node --prof-process analisis/sin console.log > analisis/result_prof-sin console.txt
(node:16374) 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)

(alexisvp97@kali) ~/Escritorio/CursoBackend/Desafios/Desafio_14
$ node --prof-process analisis/con console.log > analisis/result_prof-con console.txt
(node:16551) 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)
```

Los resultados del perfilamiento generando una carga al servidor con **artillery** son los siguientes:

Sin console.log				
result_prof-sin_console.txt U X				
analysis >	result_prof-sin_console.txt			
2				
3	[Shared libraries]:			
4	ticks	total	nonlib	name
5	9834	62.7%		/usr/lib/x86_64-linux-gnu/libnode.so.108
6	445	2.8%		/usr/lib/x86_64-linux-gnu/libc.so.6
7	58	0.4%		/usr/lib/x86_64-linux-gnu/libcrypto.so.3
8	23	0.1%		/usr/lib/x86_64-linux-gnu/libssl.so.3
9	20	0.1%		/usr/lib/x86_64-linux-gnu/libuv.so.1.0.0
10	15	0.1%		/usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.30
11	5	0.0%		[vdso]
12	2	0.0%		/usr/lib/x86_64-linux-gnu/libicuuc.so.71.1
13				

Con console.log				
result_prof-con_console.txt U X				
analysis >	result_prof-con_console.txt			
2				
3	[Shared libraries]:			
4	ticks	total	nonlib	name
5	10998	61.7%		/usr/lib/x86_64-linux-gnu/libnode.so.108
6	476	2.7%		/usr/lib/x86_64-linux-gnu/libc.so.6
7	66	0.4%		/usr/lib/x86_64-linux-gnu/libcrypto.so.3
8	57	0.3%		/usr/lib/x86_64-linux-gnu/libssl.so.3
9	19	0.1%		/usr/lib/x86_64-linux-gnu/libuv.so.1.0.0
10	12	0.1%		/usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.30
11	9	0.1%		[vdso]
12	3	0.0%		/usr/lib/x86_64-linux-gnu/libicuuc.so.71.1
13				

Luego se realizará un proceso similar, pero con la herramienta **autocannon**.

Para esto se debe iniciar el server de la misma manera que se hizo anteriormente (comentando y descomentando las líneas de los console.log).

Sin console.log

```
(alexisvp97@kali) - [~/Escritorio/CursoBackend/Desafios/Desafio_14]
$ autocannon -c 100 -d 20 http://localhost:8080/info
Running 20s test @ http://localhost:8080/info
100 connections
```

Stat	2.5%	50%	97.5%	99%	Avg	Stdev	Max
Latency	812 ms	1024 ms	1994 ms	2223 ms	1106.41 ms	269.97 ms	3030 ms

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	7	7	95	128	89.4	25.71	7
Bytes/Sec	18.5 kB	18.5 kB	251 kB	339 kB	237 kB	68.1 kB	18.5 kB

Req/Bytes counts sampled once per second.
 # of samples: 20
 2k requests in 20.08s, 4.73 MB read

Con console.log

```
(alexisvp97@kali) - [~/Escritorio/CursoBackend/Desafios/Desafio_14]
$ autocannon -c 100 -d 20 http://localhost:8080/info
Running 20s test @ http://localhost:8080/info
100 connections
```

Stat	2.5%	50%	97.5%	99%	Avg	Stdev	Max
Latency	820 ms	1015 ms	2059 ms	2377 ms	1083.54 ms	292.85 ms	3796 ms

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	3	3	96	134	90.9	26.43	3
Bytes/Sec	7.95 kB	7.95 kB	254 kB	355 kB	241 kB	70 kB	7.94 kB

Req/Bytes counts sampled once per second.
 # of samples: 20
 2k requests in 20.08s, 4.81 MB read

Como se puede apreciar en las imágenes anteriores, se ingresa el mismo comando para ejecutar las dos cargas al servidor.

2) Perfilamiento con -inspect :

Sin console.log

```

1 import { Router } from 'express';
2 import compression from "compression";
3 import { cpus } from 'os';
4 const router = Router();
5
6 router.get('/', compression(), (req, res) => {
7   const args = process.argv.slice(2).join(' ');
8   const info = {
9     args, //Argumentos de entrada
10    path: process.execPath, //Path de ejecución
11    os: process.platform, //Nombre de la plataforma (sistema operativo)
12    pid: process.pid, //Process id
13    nodeVersion: process.version, //Versión de node.js
14    dirPath: process.cwd(), //Carpeta del proyecto
15    memoryUsage: process.memoryUsage.rss() / 2 ** 20, //Memoria total reservada (rss) en MiB
16    numCPUs: cpus().length,
17  };
18  /* ----- Evaluación de rendimiento ----- */
19  // console.log(info.args);
20  // console.log(info.path);
21  // console.log(info.os);
22  // console.log(info.pid);
23  // console.log(info.nodeVersion);
24  // console.log(info.dirPath);
25  // console.log(info.memoryUsage);
26  // console.log(info.numCPUs);
27  /* ----- */
28  res.status(200).render('partials/viewInfo', {
29    info,
30  });
31 });
32
33 export default router;
  
```

Con console.log

```

1 import { Router } from 'express';
2 import compression from "compression";
3 import { cpus } from 'os';
4 const router = Router();
5
6 router.get('/', compression(), (req, res) => {
7   const args = process.argv.slice(2).join(' ');
8   const info = {
9     args, //Argumentos de entrada
10    path: process.execPath, //Path de ejecución
11    os: process.platform, //Nombre de la plataforma (sistema operativo)
12    pid: process.pid, //Process id
13    nodeVersion: process.version, //Versión de node.js
14    dirPath: process.cwd(), //Carpeta del proyecto
15    memoryUsage: process.memoryUsage.rss() / 2 ** 20, //Memoria total reservada (rss) en MiB
16    numCPUs: cpus().length,
17  };
18  /* ----- Evaluación de rendimiento ----- */
19  console.log(info.args);
20  console.log(info.path);
21  console.log(info.os);
22  console.log(info.pid);
23  console.log(info.nodeVersion);
24  console.log(info.dirPath);
25  console.log(info.memoryUsage);
26  console.log(info.numCPUs);
27  /* ----- */
28  res.status(200).render('partials/viewInfo', {
29    info,
30  });
31 });
32
33 export default router;
  
```

Para este caso se utilizó la consola con las devs tools de Brave y se generó una carga al servidor con autocannon.



4) Conclusión:

Con base en los análisis de los diferentes tipos de perfilamiento realizados, generando carga al servidor de dos maneras diferentes (artillery y autocannon) y comparando los resultados obtenidos, se puede concluir que la ejecución de tareas y/o procesos bloqueantes adiciona una carga extra al proyecto ralentizando el acceso a los recursos.