

## Análisis de performance

Se realizó un análisis de la aplicación trabajando sobre la ruta GET /info agregando y extrayendo el logueo de la información del sistema operativo antes de devolverla al cliente para observar los cambios que se producen en su performance.

- 1) Perfilamiento del servidor usando --prof y --prof-process para procesar los resultados:
  - a) usando Artillery para el test de carga emulando 50 conexiones concurrentes con 20 request por cada una

Al realizar las pruebas con Artillery podemos observar que la diferencia en velocidad de respuesta es significativa: cuando se loguean los datos antes de devolverlos, la aplicación procesa unos 114 requests por segundo con una latencia promedio de 67 milisegundos aproximadamente. Por el contrario, cuando no se loguea, la aplicación llega a procesar casi el doble de requests por segundo. (Ver `result.fork.nolog` y `result.fork.withlog`).

```
http.codes.200: ..... 1000
http.request_rate: ..... 114/sec
http.requests: ..... 1000
http.response_time:
  min: ..... 8
  max: ..... 133
  median: ..... 67.4
  p95: ..... 115.6
  p99: ..... 127.8
http.responses: ..... 1000
```

```
http.codes.200: ..... 1000
http.request_rate: ..... 216/sec
http.requests: ..... 1000
http.response_time:
  min: ..... 3
  max: ..... 78
  median: ..... 32.1
  p95: ..... 46.1
  p99: ..... 53
```

- b) usando Autocannon para el test de carga emulando 100 conexiones concurrentes realizadas en un tiempo de 20 segundos

Al realizar las pruebas con Autocannon también se observan importantes diferencias en la performance: la aplicación llega a responder 14 mil requests en el tiempo de la prueba cuando no loguea y 10 mil cuando sí lo hace (una diferencia del 40%). Estos

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	400	400	730	807	710.4	87.43	400
Bytes/Sec	645 kB	645 kB	1.18 MB	1.3 MB	1.15 MB	141 kB	645 kB

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

14k requests in 20.11s, 22.9 MB read

resultados tienen coherencia con lo observado en el punto a). (Ver resultado-autocannon-con-log.png y resultado-autocannon-sin-log.png)

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	300	300	518	597	517.5	56.77	300
Bytes/Sec	483 kB	483 kB	835 kB	962 kB	834 kB	91.5 kB	483 kB

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

10k requests in 20.15s, 16.7 MB read

Al revisar el resultado de la prueba de rendimiento utilizando `--prof-process` podemos ver que el proceso de Javascript con logueo se lleva más ticks que en el caso que no se loguea, aunque aumentan los ticks de las Shared Libraries (podría deberse a que durante la prueba en el caso sin logueo la aplicación procesó más requests en el mismo tiempo). (Ver `prof-process_no-log.txt` y `prof-process_with-log.txt`)

prof-process\_with-log.txt: Bloc de notas

Archivo Edición Formato Ver Ayuda

[Summary]:

ticks	total	nonlib	name
30	0.4%	100.0%	JavaScript
0	0.0%	0.0%	C++
18	0.3%	60.0%	GC
6998	99.6%		Shared libraries

prof-process\_no-log.txt: Bloc de notas

Archivo Edición Formato Ver Ayuda

[Summary]:

ticks	total	nonlib	name
22	0.1%	100.0%	JavaScript
0	0.0%	0.0%	C++
15	0.1%	68.2%	GC
23053	99.9%		Shared libraries

- 2) Perfilamiento del servidor usando `--inspect`, revisando el tiempo de los procesos menos performantes sobre el archivo fuente de inspeccion (uso artillery para el test de carga emulando 50 conexiones concurrentes con 20 request por cada una).

Al revisar los resultados con `--inspect` se puede observar, sobre todo para el caso con logueo, que la aplicación dedica bastante tiempo a armar strings con formato y realizando llamados a consola.

no-log-CPU-20221006T205037.cpubprofile			with-log-CPU-20221006T205203.cpubprofile		
C: > Users > TT > Desktop > reporte > with-log-CPU-20221006T205203.cpubprofile					
Filter functions or files, or start a query()					
Self Time	Total Time	File	Self Time	Total Time	File
25.048,45ms	25.048,45ms	(idle)	25.048,45ms	25.048,45ms	(idle)
1200,18ms	1200,18ms	> writeUtf8String	1200,18ms	1200,18ms	> writeUtf8String
1096,54ms	2503,15ms	> consoleCall	1096,54ms	2503,15ms	> consoleCall

- 3) Diagrama de flama con 0x usando Autocannon para el test de carga emulando 100 conexiones concurrentes realizadas en un tiempo de 20 segundos.

A simple vista los diagramas de flama se observan bastante similares. Aunque en el caso que no se realiza el logueo de datos tal vez se puedan identificar más picos y menos mesetas, la aplicación se comporta como no bloqueante en ambos escenarios (no se observan zonas del gráfico coloreadas como "hot"). (Ver `flamegraph-con-log.html` y `flamegraph-sing-log.html`).

Como conclusión final podemos mencionar la importancia de la utilización de herramientas y estrategias adecuadas para el logueo en nuestras aplicaciones según el entorno: no sería buena idea llevar a producción esta aplicación con el método `/info` realizando un log en consola para cada llamado ya que repercute notablemente en la performance del servidor.