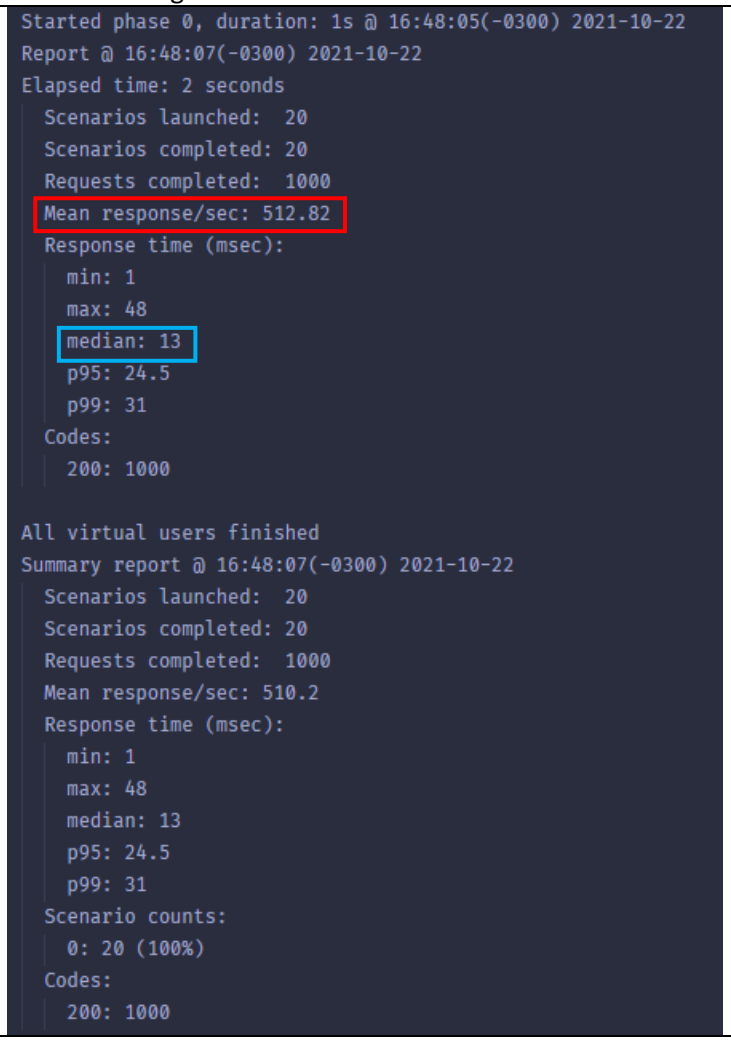
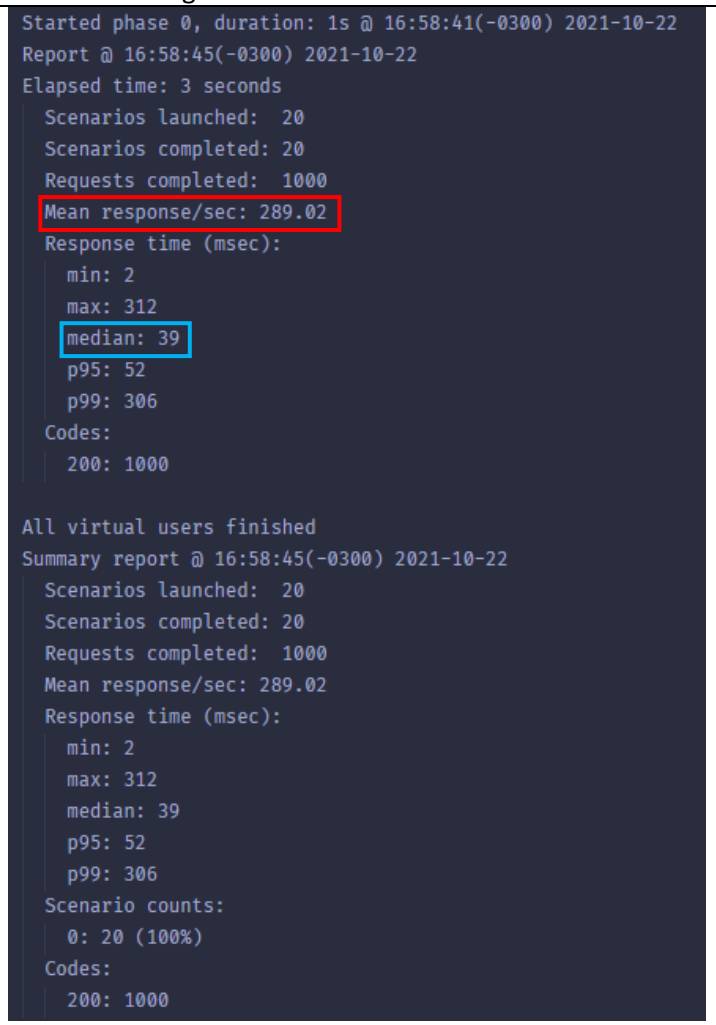
Informe Performance

	Sin console log	Con console log
Artillery (1)		

Noide Profiler (2)

```
Statistical profiling result from prof_noConsole.log
(3471 ticks, 2 unaccounted, 0 excluded).

 [Shared libraries]:···

 [JavaScript]:···

 [C++]:···

 [Summary]:
   ticks  total  nonlib   name
      80   2.3%    6.8%  JavaScript
    1101  31.7%   93.1%  C++
     379  10.9%   32.0%  GC
    2288  65.9%          Shared libraries
       2   0.1%          Unaccounted
```

```
Statistical profiling result from prof_console.log,
(4358 ticks, 5 unaccounted, 0 excluded).

 [Shared libraries]:···

 [JavaScript]:···

 [C++]:···

 [Summary]:
   ticks  total  nonlib   name
     162   3.7%   13.8%  JavaScript
    1004  23.0%   85.7%  C++
     351   8.1%   30.0%  GC
    3187  73.1%          Shared libraries
       5   0.1%          Unaccounted
```

Autocannon (3)

```
➜ back git:(desafio-31) ✗ autocannon -c 100 -d 20 "http://localhost:8080/api/inf
Running 20s test @ http://localhost:8080/api/info
100 connections
```

Stat	2.5%	50%	97.5%	99%	Avg	Stdev	Max
Latency	119 ms	206 ms	409 ms	419 ms	228.79 ms	89.86 ms	422 ms

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	238	238	370	782	431.75	170.29	238
Bytes/Sec	132 kB	132 kB	206 kB	433 kB	240 kB	94.3 kB	132 kB

```
Req/Bytes counts sampled once per second.

9k requests in 20.03s, 4.8 MB read
```

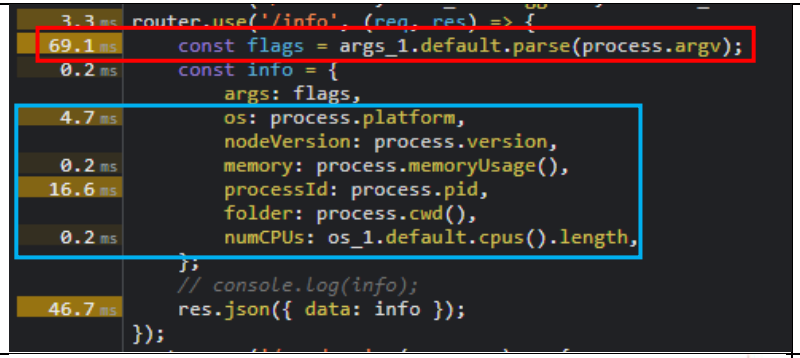
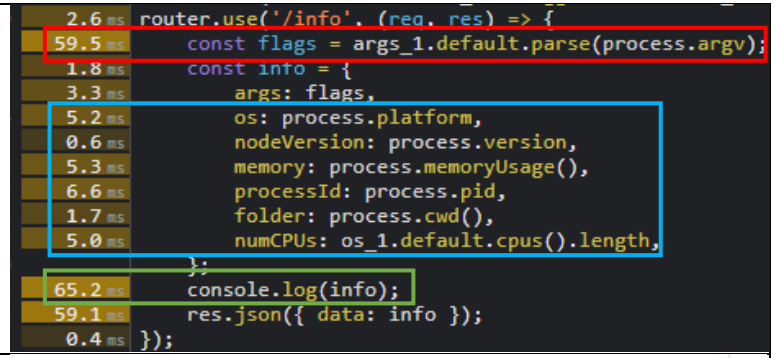
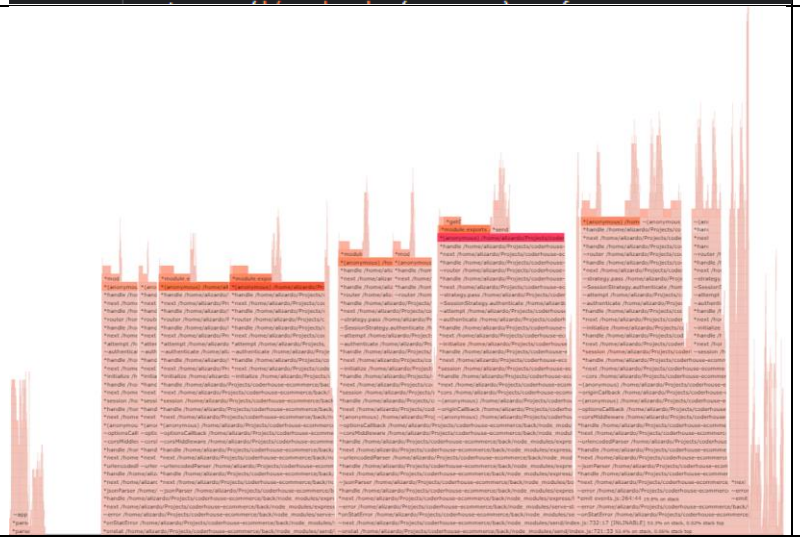
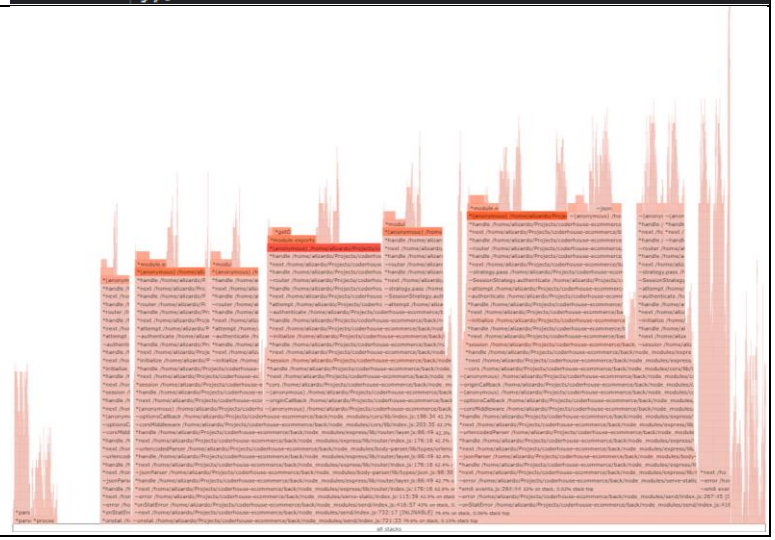
```
➜ back git:(desafio-31) ✗ autocannon -c 100 -d 20 "http://localhost:8080/api/i
Running 20s test @ http://localhost:8080/api/info
100 connections
```

Stat	2.5%	50%	97.5%	99%	Avg	Stdev	Max
Latency	321 ms	361 ms	454 ms	458 ms	370.47 ms	46.52 ms	464 ms

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	221	221	271	302	266.55	26.13	221
Bytes/Sec	123 kB	123 kB	151 kB	167 kB	148 kB	14.3 kB	123 kB

```
Req/Bytes counts sampled once per second.

5k requests in 20.03s, 2.96 MB read
```

Inspector de node (4)		
Flamegraph (5)		

Análisis:

(1) En ambos casos se emularon 50 conexiones concurrentes con 20 requests cada una, Se observa que en el caso sin console.log, la cantidad de respuestas promedio por segundo fue mayor en aproximadamente 40% que en el caso con console.log (recuadros rojos, 512.82 vs. 289.02), también el tiempo promedio de respuesta (recuadros azules) fue aproximadamente 30% menor en el caso sin

console.log (13ms), que en el caso con console.log (39ms), **siendo entonces evidente que el código que no incluye al console.log tiene mucho mejor performance**

(2) Al realizar la misma prueba mencionada anteriormente con Artillery, se observa que el proceso sin console.log genera aproximadamente 20-30% menos ticks (recuadros rojos) que el proceso con console.log, **siendo igualmente el proceso sin console.log más eficiente**

(3) Se emularon 100 conexiones concurrentes en un tiempo de 20 segundos. En el proceso sin console.log, se observa una latencia promedio (recuadros rojos) de 228.9ms, mientras que en el proceso con console.log esta es de 370.47ms, siendo la latencia aproximadamente 40% menor en el proceso sin console.log. Igualmente la cantidad promedio de requests por segundos (recuadros azules) fue mayor por aproximadamente 40% en el proceso sin console.log (431.75) que en el proceso con console.log (266.55), por último se puede observar que la cantidad total de requests (recuadros verdes) en el proceso sin console.log fue mucho mayor (9k) que en el proceso con console.log (5k), una diferencia de aproximadamente 55%, **teniendo entonces el proceso sin console.log mucho mejor performance.**

(4) En ambas pruebas se observa una latencia importante en la línea en la cual se obtienen los argumentos de la línea de comandos (recuadros rojos), esto podría mejorarse moviendo esta variable fuera de la requests para que de esta forma solo se obtengan una vez, y no cada vez que se haga una request a ese endpoint, igualmente en ambos casos se observa latencia en las líneas en donde se obtienen valores a partir de la variable global 'process' (recuadros azules), esto igualmente podría solucionarse de la misma manera mencionada. Por último, se observa una latencia importante en la request que incluye el console.log en la línea en la que se utiliza el mismo (recuadro verde), lo que disminuye el rendimiento del proceso. **En este caso igualmente sigue teniendo mejor performance el proceso que no incluye el console.log**

(5) En ambos gráficos se observan zonas de meseta y picos de actividad relativamente parecidos, en las zonas rojas, ambos gráficos señalan un problema en el archivo ‘src/routes/index.js’ línea 20, esta línea corresponde a la sección en la que se obtienen los argumentos de la línea de comandos, se podría solucionar como se mencionó anteriormente en ambos casos. **En este caso no se ve que ninguno de los dos procesos tenga un performance significativamente mayor al otro**

Conclusión:

De acuerdo con las pruebas mencionadas y análisis realizado el proceso que no incluye el console.log tiene un performance mucho mejor que el proceso que lo incluye