1) --prof

a) Artillery /info con console.log

En la terminal ingresamos:
node --prof index.js

Artillery 50 conexiones, 20 request cada una, registrar en archivo result_clog.txt
En otra terminal ingresamos:
artillery quick --count 20 -n 50 "http://localhost:8080/info" > result_clog.txt

-Renombrar archivo isolate como clog-v8.log, luego ejecutar:
node --prof-process clog-v8.log > result_prof-clog.txt

**Resultado en archivo result_prof-clog.txt (carpeta raíz del proyecto):*

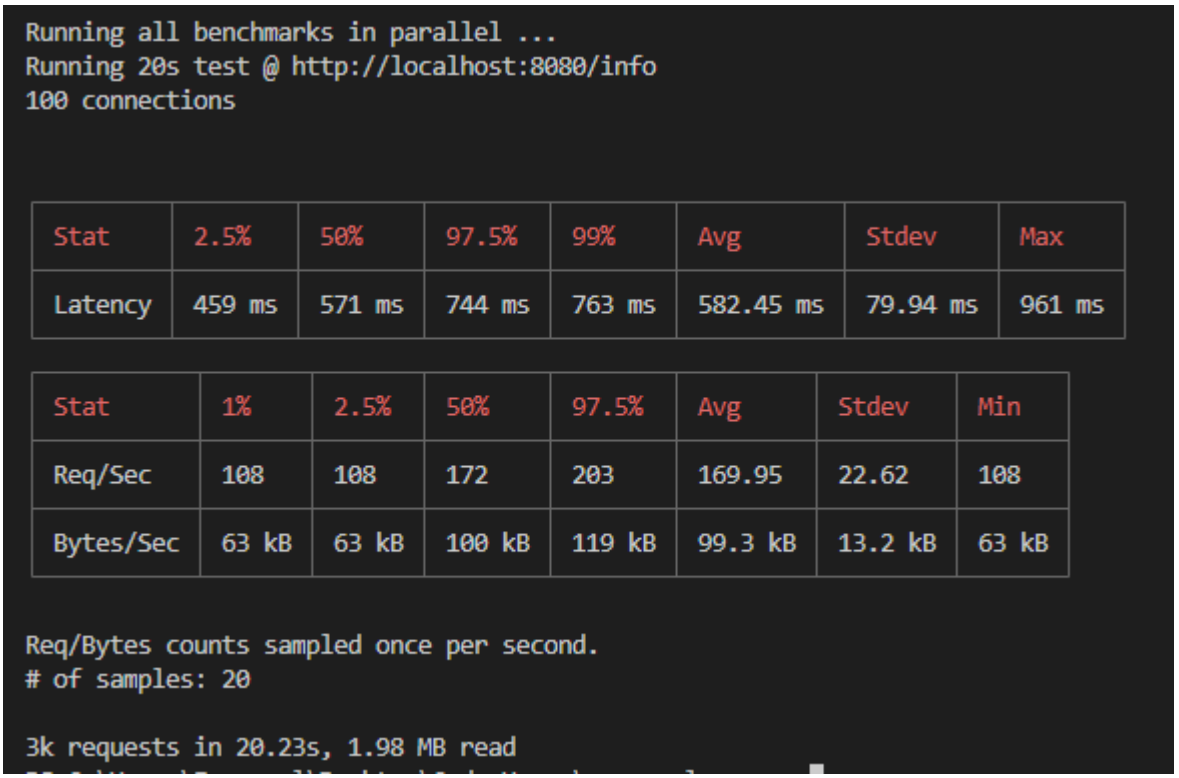
```
33  --------------------------------
34  Summary report @ 22:46:59(-0400)
35  --------------------------------
36
37  http.codes.200: ..................................................................... 1000
38  http.request_rate: .................................................................. 72/sec
39  http.requests: ...................................................................... 1000
40  http.response_time:
41    min: .............................................................................. 18
42    max: .............................................................................. 285
43    median: ........................................................................... 117.9
44    p95: .............................................................................. 190.6
45    p99: .............................................................................. 273.2
46  http.responses: ..................................................................... 1000
47  vusers.completed: ................................................................... 20
48  vusers.created: ..................................................................... 20
49  vusers.created_by_name.0: ........................................................... 20
50  vusers.failed: ...................................................................... 0
51  vusers.session_length:
52    min: .............................................................................. 5985.4
53    max: .............................................................................. 6333.1
54    median: ........................................................................... 6187.2
55    p95: .............................................................................. 6312.2
56    p99: .............................................................................. 6312.2
57
```

b) Autocannon /info con console.log

En la terminal ingresamos:
node --prof index.js

Autocannon 100 conexiones 20 segundos, en otra terminal ingresamos:
npm test
//esto ejecuta el archivo benchmark.js con la configuración de autocannon

Reporte de los resultados (Print Screen de la consola):



```
Running all benchmarks in parallel ...
Running 20s test @ http://localhost:8080/info
100 connections
```

Stat	2.5%	50%	97.5%	99%	Avg	Stdev	Max
Latency	459 ms	571 ms	744 ms	763 ms	582.45 ms	79.94 ms	961 ms

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	108	108	172	203	169.95	22.62	108
Bytes/Sec	63 kB	63 kB	100 kB	119 kB	99.3 kB	13.2 kB	63 kB

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

3k requests in 20.23s, 1.98 MB read
```

1) --prof

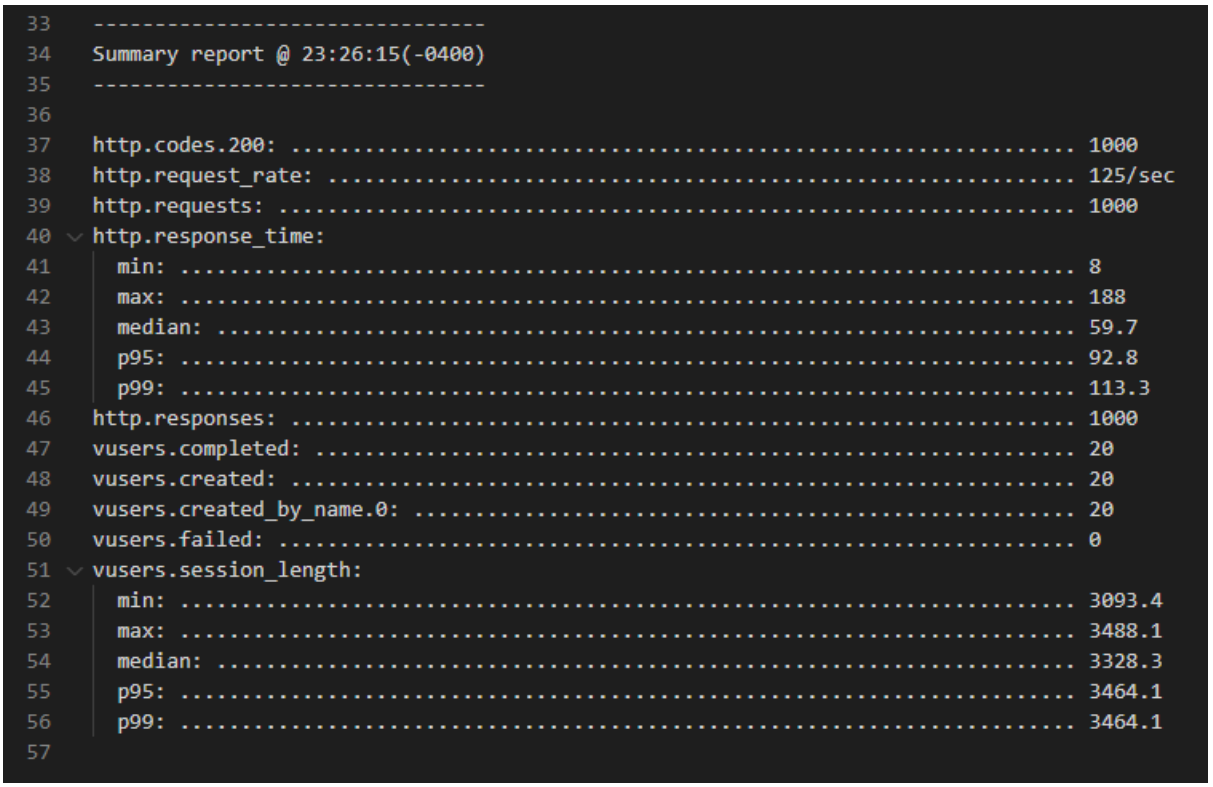
c) Artillery /info sin console.log

En la terminal ingresamos:
node --prof index.js

Artillery 50 conexiones, 20 request cada una, registrar en archivo result_noclog.txt
En otra terminal ingresamos:
artillery quick --count 20 -n 50 “http://localhost:8080/info” > result_noclog.txt

-Renombrar archivo isolate como noclog-v8.log, luego ejecutar:
node --prof-process noclog-v8.log > result_prof-noclog.txt

**Resultado en archivo result_prof-noclog.txt (carpeta raíz del proyecto):*



d) Autocannon /info sin console.log

En la terminal ingresamos:
node --prof index.js

Autocannon 100 conexiones 20 segundos, en otra terminal ingresamos:
npm test
//esto ejecuta el archivo benchmark.js con la configuración de autocannon

Reporte de los resultados (Print Screen de la consola):

```
Running all benchmarks in parallel ...
Running 20s test @ http://localhost:8080/info
100 connections
```

Stat	2.5%	50%	97.5%	99%	Avg	Stdev	Max
Latency	177 ms	205 ms	334 ms	375 ms	221.79 ms	46.42 ms	519 ms

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	287	287	472	525	449.4	63.55	287
Bytes/Sec	167 kB	167 kB	275 kB	307 kB	262 kB	37.3 kB	167 kB

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

9k requests in 20.19s, 5.25 MB read
```

//

2) --inspect (con Artillery)

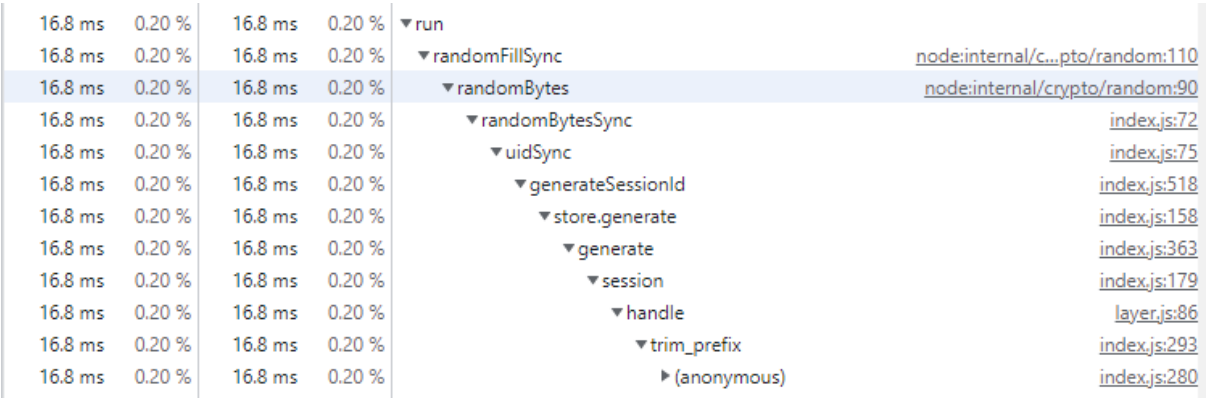
a) /info con console.log

En la terminal ingresamos:
node --inspect index.js

En el navegador ingresamos a:
chrome://inspect

Artillery 50 conexiones, 20 request cada una.
En otra terminal ingresamos:
artillery quick --count 20 -n 50 “http://localhost:8080/info”

**Revisar el tiempo de los procesos menos performantes*
Reporte de los resultados (Print Screen):



16.8 ms	0.20 %	16.8 ms	0.20 %	▼run	
16.8 ms	0.20 %	16.8 ms	0.20 %	▼randomFillSync	node:internal/c...pto/random:110
16.8 ms	0.20 %	16.8 ms	0.20 %	▼randomBytes	node:internal/crypto/random:90
16.8 ms	0.20 %	16.8 ms	0.20 %	▼randomBytesSync	index.js:72
16.8 ms	0.20 %	16.8 ms	0.20 %	▼uidSync	index.js:75
16.8 ms	0.20 %	16.8 ms	0.20 %	▼generateSessionId	index.js:518
16.8 ms	0.20 %	16.8 ms	0.20 %	▼store.generate	index.js:158
16.8 ms	0.20 %	16.8 ms	0.20 %	▼generate	index.js:363
16.8 ms	0.20 %	16.8 ms	0.20 %	▼session	index.js:179
16.8 ms	0.20 %	16.8 ms	0.20 %	▼handle	layer.js:86
16.8 ms	0.20 %	16.8 ms	0.20 %	▼trim_prefix	index.js:293
16.8 ms	0.20 %	16.8 ms	0.20 %	▶(anonymous)	index.js:280

```
267
268             // store route for dispatch on change
269   0.3 ms    if (route) {
270  19.0 ms      req.route = route;
271             }
272
273             // Capture one-time layer values
274  16.9 ms    req.params = self.mergeParams
275               ? mergeParams(layer.params, parentParams)
276               : layer.params;
277             var layerPath = layer.path;
278
279             // this should be done for the layer
280   2.5 ms    self.process_params(layer, paramcalled, req, res, function (err) {
281   0.2 ms      if (err) {
282                 next(layerError || err)
283   3.4 ms      } else if (route) {
284   3.1 ms        layer.handle_request(req, res, next)
285             } else {
286  11.5 ms        trim_prefix(layer, layerError, layerPath, path)
287             }
288
289   0.5 ms      sync = 0
290           });
291         }
292
```

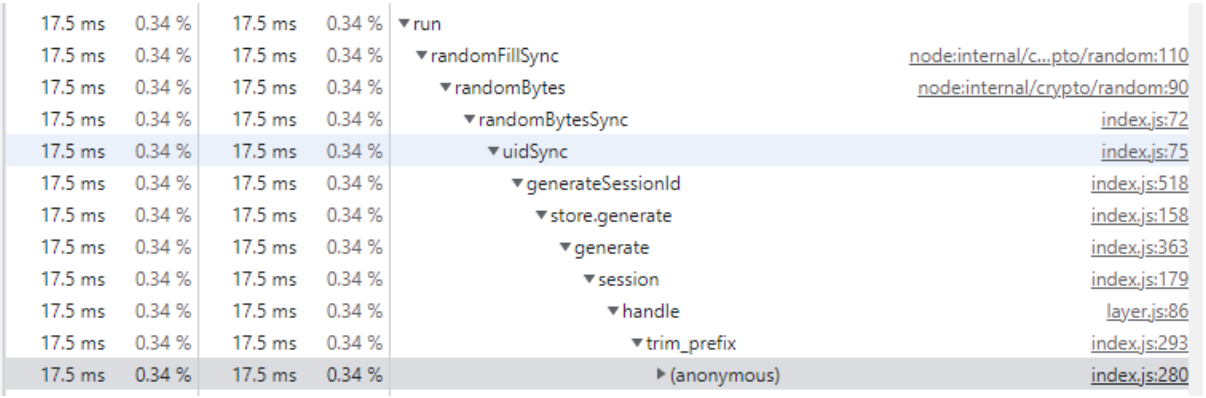
b) /info sin console.log

En la terminal ingresamos:
node --inspect index.js

En el navegador ingresamos a:
chrome://inspect

Artillery 50 conexiones, 20 request cada una.
En otra terminal ingresamos:
artillery quick --count 20 -n 50 “http://localhost:8080/info”

**Revisar el tiempo de los procesos menos performantes*
Reporte de los resultados (Print Screen):



17.5 ms	0.34 %	17.5 ms	0.34 %	▼run	
17.5 ms	0.34 %	17.5 ms	0.34 %	▼randomFillSync	node:internal/c...pto/random:110
17.5 ms	0.34 %	17.5 ms	0.34 %	▼randomBytes	node:internal/crypto/random:90
17.5 ms	0.34 %	17.5 ms	0.34 %	▼randomBytesSync	index.js:72
17.5 ms	0.34 %	17.5 ms	0.34 %	▼uidSync	index.js:75
17.5 ms	0.34 %	17.5 ms	0.34 %	▼generateSessionId	index.js:518
17.5 ms	0.34 %	17.5 ms	0.34 %	▼store.generate	index.js:158
17.5 ms	0.34 %	17.5 ms	0.34 %	▼generate	index.js:363
17.5 ms	0.34 %	17.5 ms	0.34 %	▼session	index.js:179
17.5 ms	0.34 %	17.5 ms	0.34 %	▼handle	layer.js:86
17.5 ms	0.34 %	17.5 ms	0.34 %	▼trim_prefix	index.js:293
17.5 ms	0.34 %	17.5 ms	0.34 %	▶(anonymous)	index.js:280

```
268          // store route for dispatch on change
269          if (route) {
270   16.1 ms  req.route = route;
271          }
272
273          // Capture one-time layer values
274   17.4 ms  req.params = self.mergeParams
275    0.1 ms    ? mergeParams(layer.params, parentParams)
276              : layer.params;
277    0.3 ms  var layerPath = layer.path;
278
279          // this should be done for the layer
280    0.7 ms  self.process_params(layer, paramcalled, req, res, function (err) {
281            if (err) {
282              next(layerError || err)
283    2.5 ms  } else if (route) {
284    1.0 ms    layer.handle_request(req, res, next)
285            } else {
286    9.1 ms    trim_prefix(layer, layerError, layerPath, path)
287            }
288
```

//

3) 0x - con Autocannon.

a) /info con console.log

En la terminal ingresamos:
0x index.js

Autocannon 100 conexiones 20 segundos, en otra terminal ingresamos:
npm test
//esto ejecuta el archivo benchmark.js con la configuración de autocannon

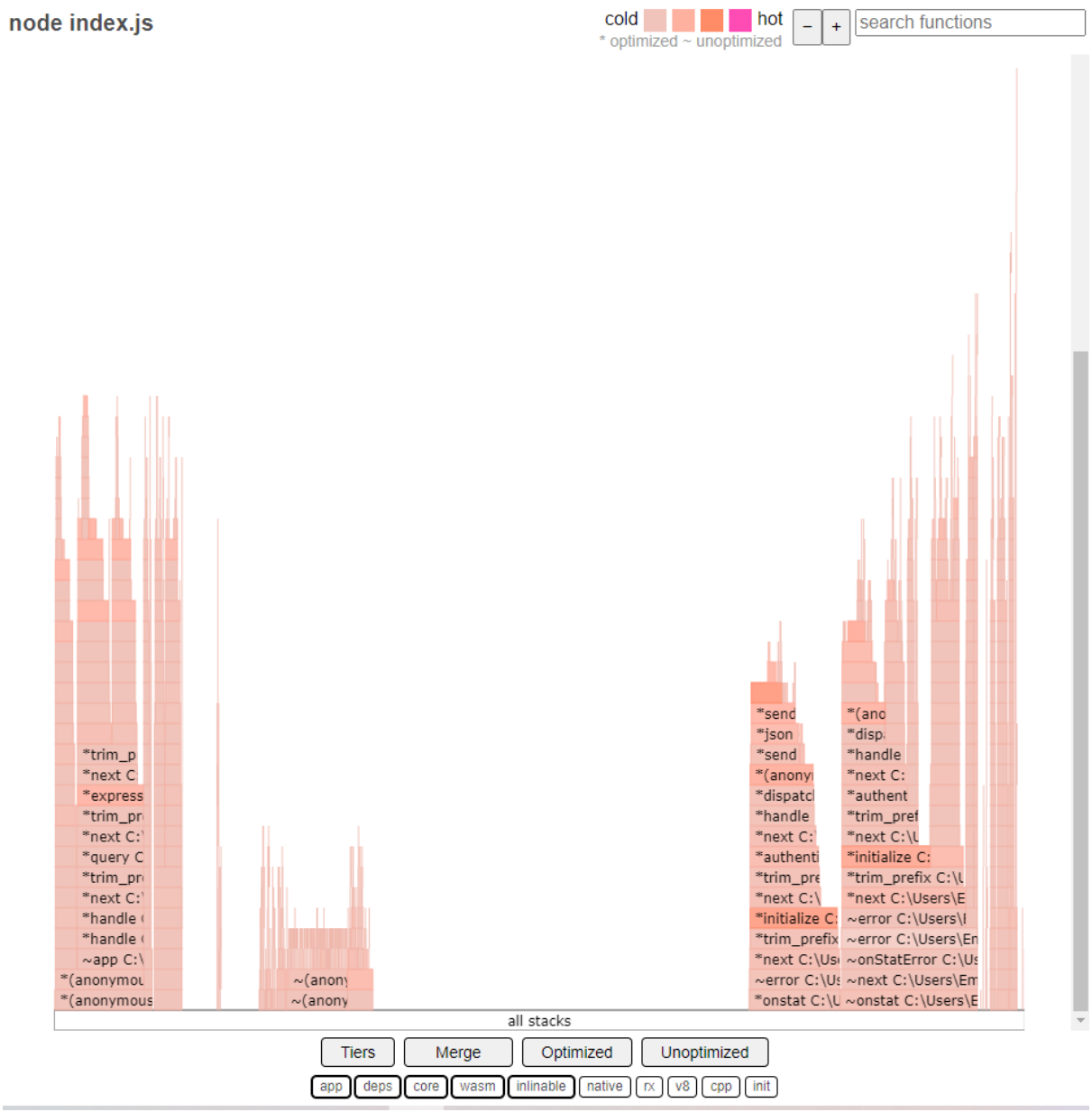
Reporte de los resultados (Print Screen de la consola):

b) /info sin console.log

En la terminal ingresamos:
0x index.js

Autocannon 100 conexiones 20 segundos, en otra terminal ingresamos:
npm test
//esto ejecuta el archivo benchmark.js con la configuración de autocannon

Reporte de los resultados (Print Screen de la consola):



//

CONCLUSIÓN

Luego de inspeccionar los diferentes resultados arrojados por las pruebas, según las instrucciones del desafío, podemos concluir que todas ellas son congruentes en señalar que la ruta /info con el console.log, consume una mayor cantidad de recursos, a diferencia de la ruta /info sin el console.log, la cual se ejecuta con mayor rapidez.

//