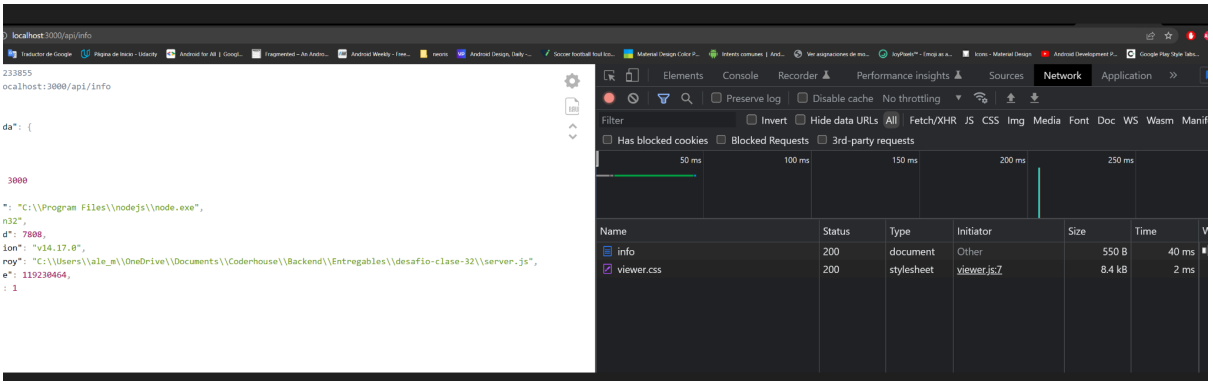
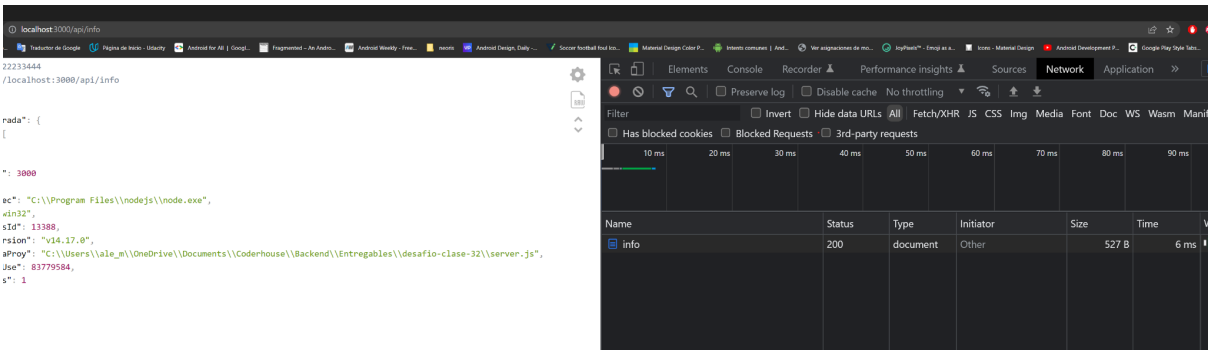
Conclusión a partir de los datos obtenidos

Respecto a usar o no la compresión de gzip:

Ruta /api/info con gzip:



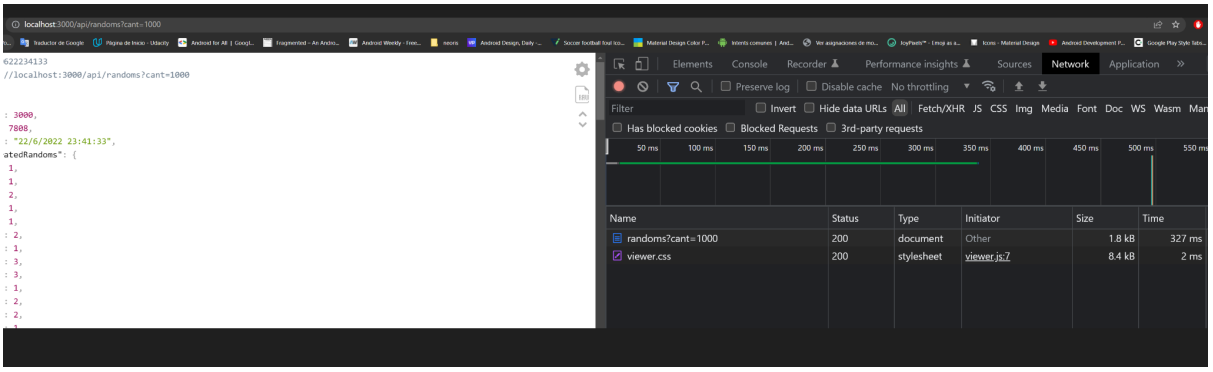
Ruta /api/info sin gzip:



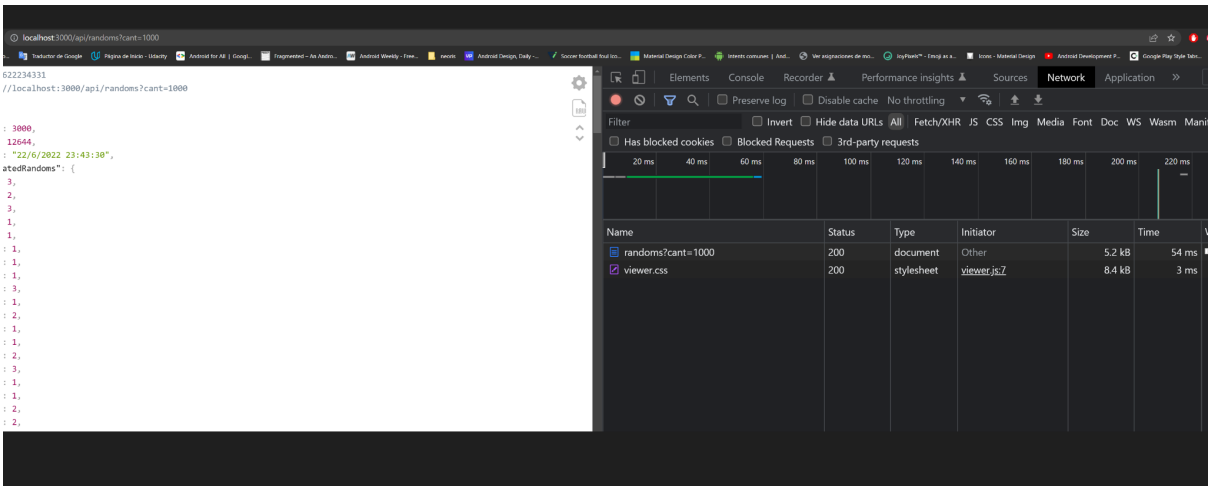
Notar que sin usar gzip el request fue más rápido(6 ms) y comprimió más (527 b).
En cambio usando gzip tardó 40 ms y el response ocupó 550b.
Contrario a lo que hubiéramos esperado al usar gzip el response fue más pesado.

Si en vez de usar /api/info usamos /api/randoms el resultado es distinto:

Ruta /api/randoms con gzip:



Ruta /api/randoms sin gzip:



En este caso sin gzip el response es de 5.2kb y con gzip es de 1.8kb como se hubiera esperado. Obviamente cuando se usa gzip el response tarda más (327ms) que si no se lo utiliza (54ms).

De esto se puede concluir que gzip es más efectivo a medida que el tamaño de los datos es mayor (solo a nivel compresión, a nivel tiempo es lo contrario).

Perfilamiento del servidor usando --prof-process y Artillery:

Se ejecuta el servidor en modo fork y usando - -prof de node.js.
Además se utiliza Artillery como test de carga simulando 50 conexiones diferentes con 20 requests cada una al endpoint /api/info.
Como proceso bloqueante se agrega un console.log al endpoint y luego se quita ese console.log para la prueba no bloqueante.

Se obtuvieron los siguientes resultados:

result_prof-bloq.txt

```
[Summary]:
  ticks  total  nonlib   name
      9   0.2%   90.0%  JavaScript
      0   0.0%    0.0%  C++
     23   0.6%  230.0%  GC
   4055  99.8%          Shared libraries
      1   0.0%          Unaccounted

[C++ entry points]:
  ticks    cpp   total   name

[Bottom up (heavy) profile]:
  Note: percentage shows a share of a particular caller in the total
```

result_prof-nobloq.txt

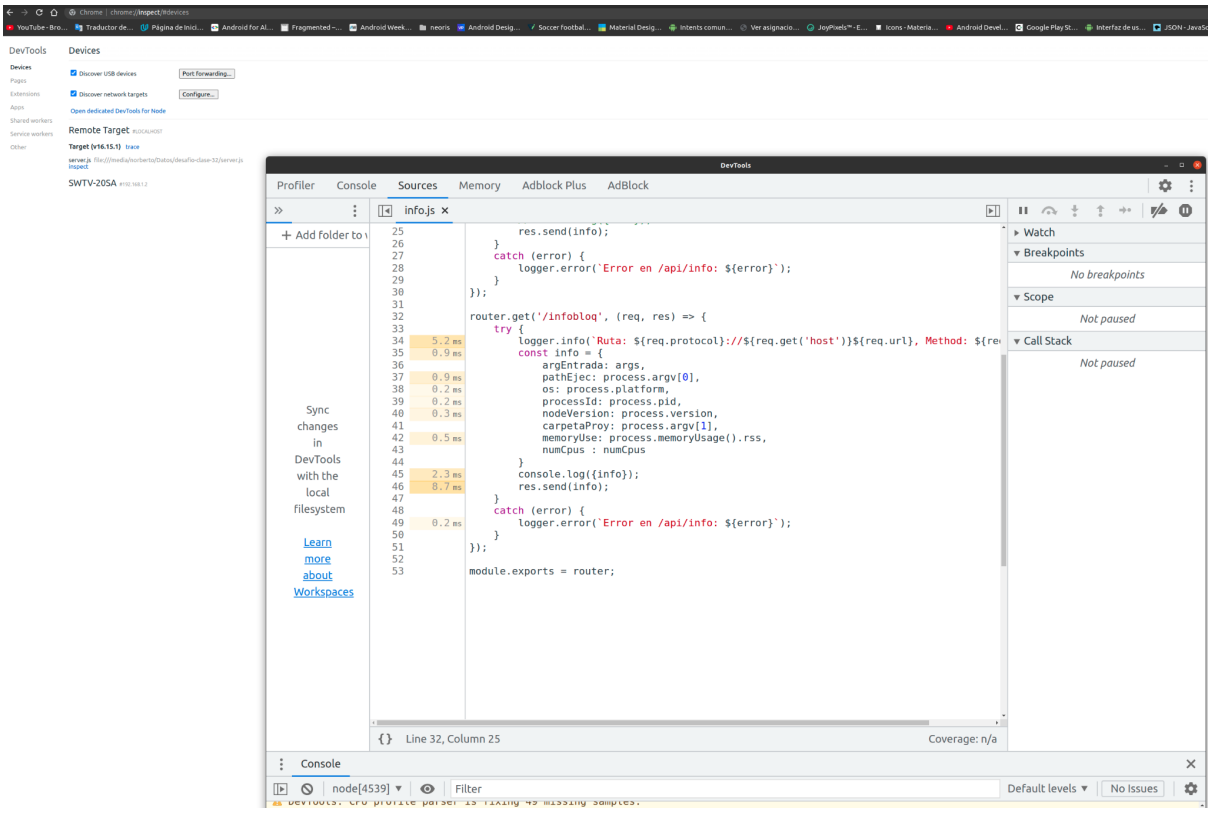
```
[C++]:
  ticks  total  nonlib   name

[Summary]:
  ticks  total  nonlib   name
     16   0.8%   94.1%  JavaScript
      0   0.0%    0.0%  C++
     15   0.7%   88.2%  GC
   2042  99.2%          Shared libraries
      1   0.0%          Unaccounted

[C++ entry points]:
  ticks    cpp   total   name
```

Estos resultados se pueden ver en la carpeta /profiling (archivos result_prof-bloq.txt y result_prof-nobloq.txt) del proyecto.
Como se puede ver en la captura y comparando ambos Summaries, el proceso no bloqueante tiene casi la mitad de ticks (Shared libraries) que el proceso bloqueante como era esperado.

Perfilamiento usando el modo inspect y Chrome:

Con el modo inspector de node.js --inspect y el navegador Chrome se hizo una prueba con Autocannon generando 50 conexiones con 20 requests cada una:

artillery quick --count 20 -n 50 "http://localhost:8080/api/infobloq" > result_bloq_inspect_chrome.txt



Como se puede observar el proceso de logueo y console.log retrasan bastante (5.2ms + 2.3ms) todo el proceso. Al realizar estas dos tareas se tarda casi el doble en consumir este endpoint.

Perfilamiento obtenido de utilizar Autocannon y 0x:

Se utilizan las rutas /api/info (no bloqueante) y /api/infobloq (bloqueante, se utiliza un console.log que muestra lo que se va a enviar como response).

Luego de correr npm test con la ruta no bloqueante obtuve este resultado:

```
ale_m@DESKTOP-L355ML9 MINGW64 ~/OneDrive/Documents/Coderhouse/Backend/Entregables/desafio-clase-32
$ npm test

> desafio-clase-24@1.0.0 test C:\Users\ale_m\OneDrive\Documents\Coderhouse\Backend\Entregables\desafio-clase-32
> node benchmark.js

Running benchmark...
Running 20s test @ http://localhost:8080/api/info
100 connections


| Stat | 2.5% | 50% | 97.5% | 99% | Avg | Stdev | Max |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Latency | 56 ms | 65 ms | 94 ms | 107 ms | 67.29 ms | 9.91 ms | 136 ms |

| Stat | 1% | 2.5% | 50% | 97.5% | Avg | Stdev | Min |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Req/Sec | 1077 | 1077 | 1512 | 1571 | 1477.05 | 115.87 | 1077 |
| Bytes/Sec | 555 kB | 555 kB | 779 kB | 809 kB | 761 kB | 59.7 kB | 555 kB |

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

30k requests in 20.11s, 15.2 MB read
```

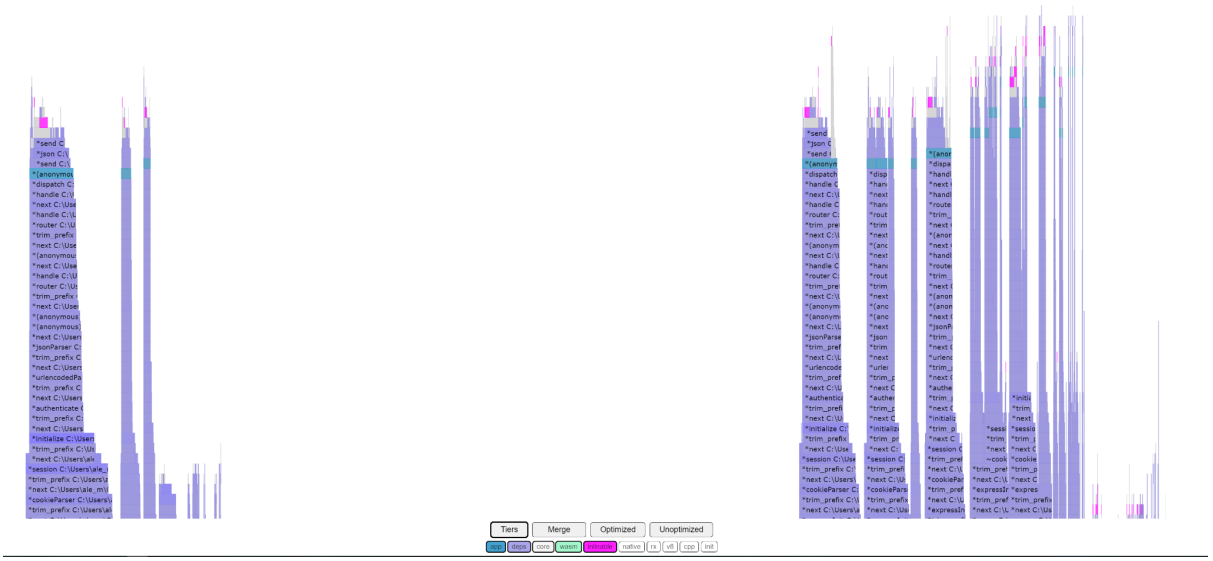
con la ruta bloqueante se obtuvo:

```
ale_m@DESKTOP-L355ML9 MINGW64 ~/OneDrive/Documents/Coderhouse/Backend/Entregables/desafio-clase-32
$ npm test

> desafio-clase-24@1.0.0 test C:\Users\ale_m\OneDrive\Documents\Coderhouse\Backend\Entregables\desafio-clase-32
> node benchmark.js

Running benchmark...
Running 20s test @ http://localhost:8080/api/infobloq
100 connections


| Stat | 2.5% | 50% | 97.5% | 99% | Avg | Stdev | Max |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Latency | 65 ms | 77 ms | 149 ms | 175 ms | 85.19 ms | 24.69 ms | 308 ms |

| Stat | 1% | 2.5% | 50% | 97.5% | Avg | Stdev | Min |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Req/Sec | 680 | 680 | 1234 | 1399 | 1166.8 | 195.28 | 680 |
| Bytes/Sec | 350 kB | 350 kB | 636 kB | 721 kB | 601 kB | 101 kB | 350 kB |

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

23k requests in 20.08s, 12 MB read
```

En ambas pruebas el tiempo y la cantidad de conexiones fue el mismo.
Como se hubiera esperado con el endpoint no bloqueante se pudieron hacer más requests (30k vs 23k). El tiempo de respuesta promedio fue menor en el caso no bloqueante (67.29ms vs 85.19ms).
Siguiendo de la misma manera con el resto de los parámetros medidos, se puede observar que la ruta no bloqueante es mucho más performante que la bloqueante.

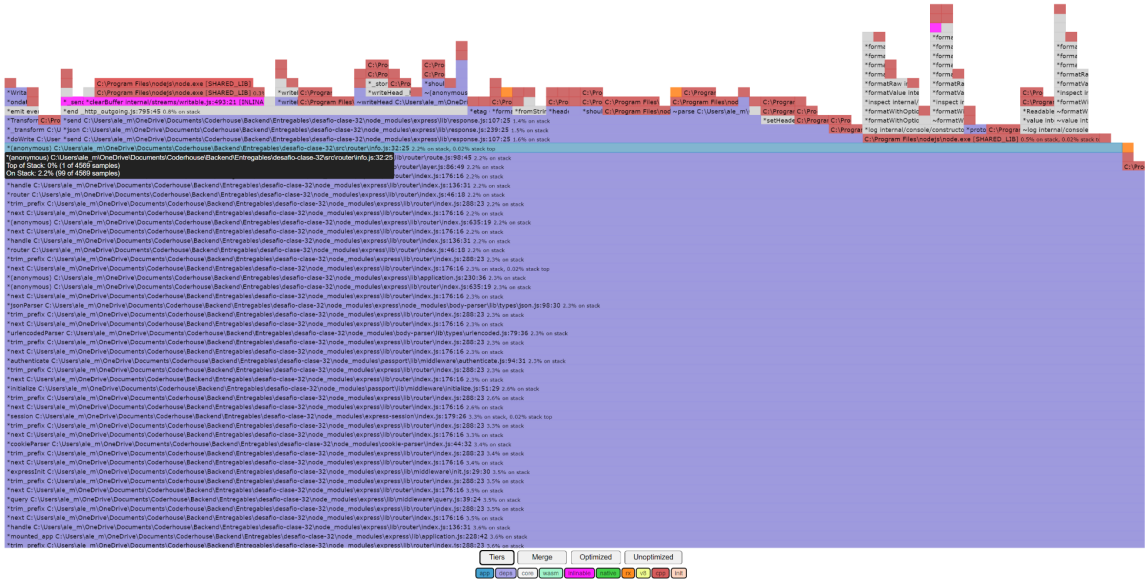
Diagrama de flama:

En la carpeta /diagrama.0x se puede encontrar el archivo flamegraph.html que el diagrama de flama obtenido de hacer estas pruebas con Autocannon.

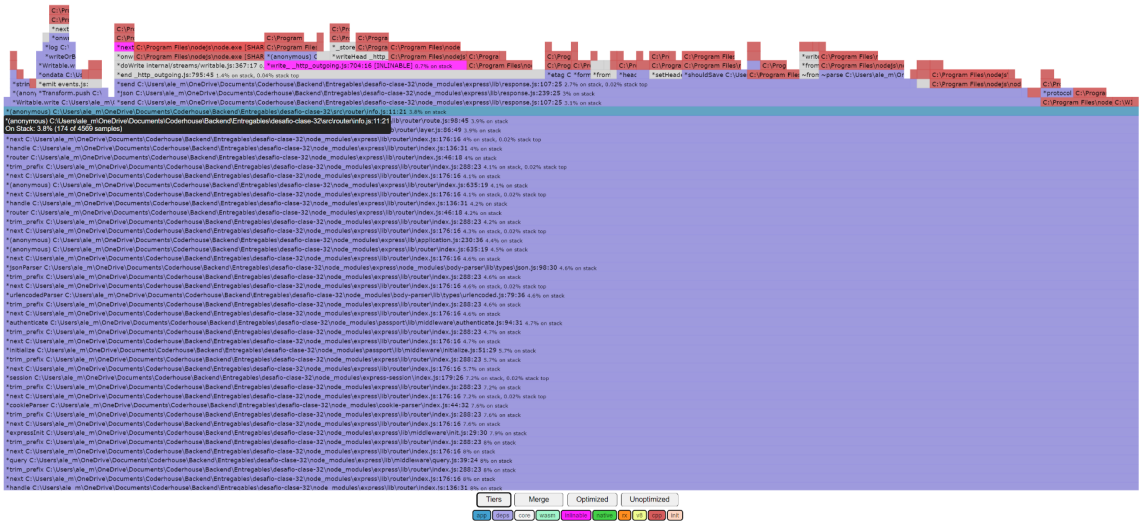


Primero se ejecutó la ruta bloqueante y luego la no bloqueante.
Haciendo una inspección rápida del gráfico, se puede observar que en la parte bloqueante (parte izquierda) los procesos de la aplicación (celeste) son más anchos (y por lo tanto están bloqueando por más tiempo a las otras tareas) y son menos que en la parte no bloqueante.
La parte no bloqueante tiene más picos que además son muy finos (como se esperaba).

zoom en uno de los picos no bloq:



zoom en uno de los picos bloq:



Como conclusión final, podemos decir que en lo posible siempre se deben evitar los procesos bloqueantes.