Entregable Clase 32:

De la primera consigna podemos ver la diferencia del peso de la página mediante compression:

info	200	do...	Other	371 B
info	200	do...	Other	11.2 kB

El salto es abismal pasamos de 11.2kb a 371b en la misma página, nota adjunto, siempre usar compression.

Consignas:

1- Profiling con fork y cluster:

```
Statistical profiling result from fork-v8.log, (1553 ticks, 0 unaccounted, 0 excluded).

[Shared libraries]:
  ticks  total  nonlib   name
   1465   94.3%          C:\WINDOWS\SYSTEM32\ntdll.dll
     84    5.4%          C:\Program Files\nodejs\node.exe

[JavaScript]:
  ticks  total  nonlib   name
      1    0.1%   25.0%  RegExp: ^[\^_`a-zA-Z\-0-9!#$%&'*+.|~]+$
      1    0.1%   25.0%  RegExp: [^\t\x20-\x7e\x80-\xff]
      1    0.1%   25.0%  LazyCompile: *resolve node:path:158:10
      1    0.1%   25.0%  Function: ^Module._nodeModulePaths node:internal/modules/cjs/loader:669:37

[C++]:
  ticks  total  nonlib   name

[Summary]:
  ticks  total  nonlib   name
      4    0.3%  100.0%  JavaScript
      0    0.0%    0.0%  C++
      4    0.3%  100.0%  GC
   1549   99.7%          Shared libraries
```

```
Statistical profiling result from cluster-v8.log, (2158 ticks, 0 unaccounted, 0 excluded).

[Shared libraries]:
  ticks  total  nonlib   name
   2085   96.6%          C:\WINDOWS\SYSTEM32\ntdll.dll
     73    3.4%          C:\Program Files\nodejs\node.exe

[JavaScript]:
  ticks  total  nonlib   name

[C++]:
  ticks  total  nonlib   name

[Summary]:
  ticks  total  nonlib   name
      0    0.0%    NaN%  JavaScript
      0    0.0%    NaN%  C++
      2    0.1%  Infinity%  GC
   2158  100.0%          Shared libraries

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

La diferencia básica que encontré, obviamente porque el server no tiene mucho procesamiento, es que en modo Cluster consume más ticks que serían la cantidad de procesos que corre para iniciar pero en cuanto al uso de cada proceso es mucho más veloz en modo cluster que en fork.

2- Tiempos de procesos

```
57
58   1.0 ms      app.get ('/sumar', (req, res) => {
59                   const {a,b}=req.query;
60   1.4 ms          if (Number(a)&&Number(b)){
61  13.2 ms          logger.info(`La suma es ${parseInt(a) +  parseInt(b)}`);
62                   res.send(`La suma es ${parseInt(a) +  parseInt(b)}`);
63                   } else {
64                   error.error("Los numeros no son validos");
65                   res.send("Los numeros no son validos");
66                   }
67                   }
68               )
69
70               app.get('/randoms', (req, res) => {
71                   res.send(`Servidor express en Puerto: ${PUERTO}, Workers: ${numProcesadores}, PID: ${process.pid} - ${(new Date()).toLocaleString()}`);
72               });
73
74               app.get('/datos', (req, res) => {
75  58.8 ms          res.send(`Servidor express en Puerto: ${PUERTO}, Workers: ${numProcesadores}, PID: ${process.pid} - ${(new Date()).toLocaleString()}`);
76               });
77
78               app.listen(PUERTO, () => {
79               console.log(`Servidor escuchando en el puerto: ${PUERTO}`);
80               });
```

En cuanto a los tiempo de procesos probé los 2 endpoints el de SUMAR y el de DATOS, me sorprendió que en general el de DATOS fuera más lento.

```
--------------------------------
Summary report @ 18:20:32(-0300)
--------------------------------

http.codes.200: ..................................................... 1000
http.request_rate: .................................................. 244/sec
http.requests: ...................................................... 1000
http.response_time:
  min: .............................................................. 3
  max: .............................................................. 231
  median: ........................................................... 108.9
  p95: .............................................................. 153
  p99: .............................................................. 179.5
http.responses: ..................................................... 1000
vusers.completed: ................................................... 50
vusers.created: ..................................................... 50
vusers.created_by_name.0: ........................................... 50
vusers.failed: ...................................................... 0
vusers.session_length:
  min: .............................................................. 697.3
  max: .............................................................. 2313.2
  median: ........................................................... 2143.5
  p95: .............................................................. 2276.1
  p99: .............................................................. 2276.1
```

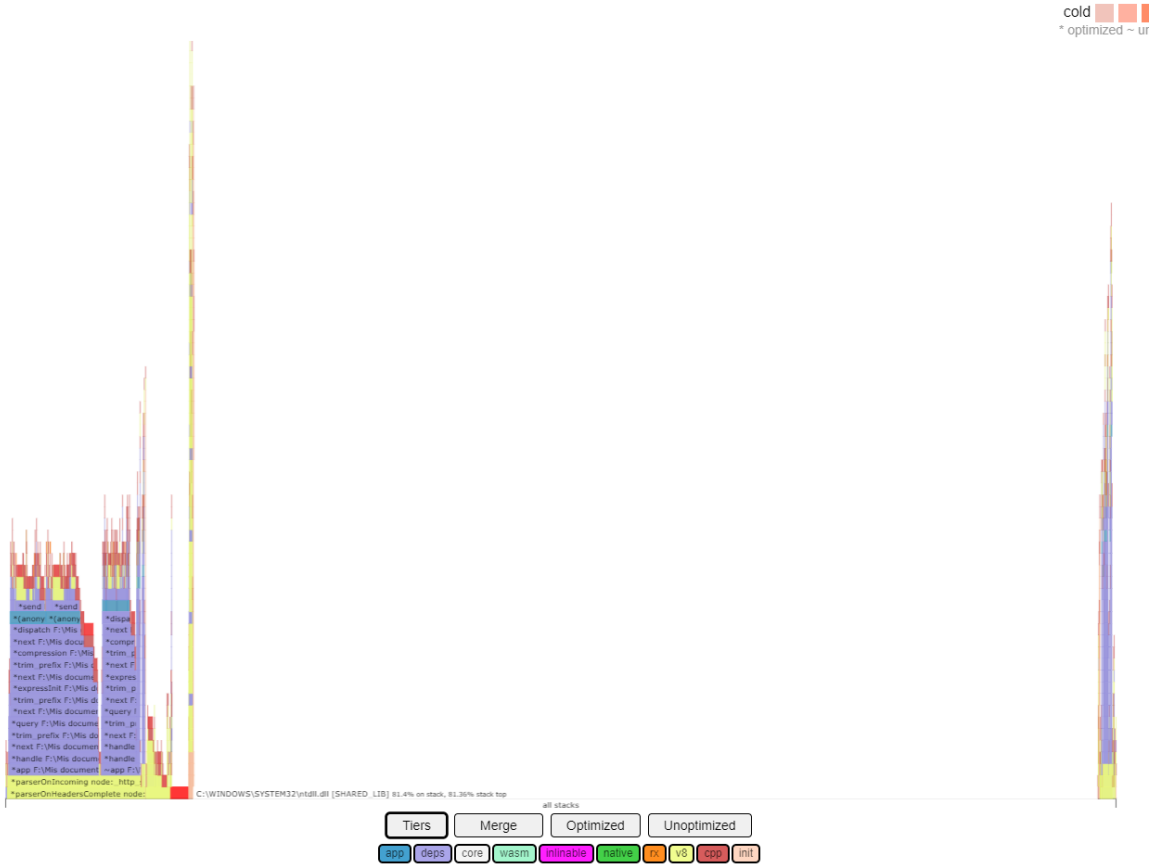
```
--------------------------------
Summary report @ 18:20:57(-0300)
--------------------------------

http.codes.200: ..................................................... 1000
http.request_rate: .................................................. 1000/sec
http.requests: ...................................................... 1000
http.response_time:
  min: .............................................................. 0
  max: .............................................................. 41
  median: ........................................................... 4
  p95: .............................................................. 8.9
  p99: .............................................................. 13.1
http.responses: ..................................................... 1000
vusers.completed: ................................................... 50
vusers.created: ..................................................... 50
vusers.created_by_name.0: ........................................... 50
vusers.failed: ...................................................... 0
vusers.session_length:
  min: .............................................................. 19.4
  max: .............................................................. 185.3
  median: ........................................................... 133
  p95: .............................................................. 162.4
  p99: .............................................................. 172.5
```

Endpoint DATOS

Endpoint SUMAR

Ya sea por el inspect de Chrome o por artillery en ambos casos la respuesta fue la misma.

3- Diagrama de Flama con 0x y Autocannon



Según mi grafico de flama no tengo aplicaciones que estén bloqueando el servidor, porque claramente la idea fue otra, pero aprendí varias cosas en el proceso. Tanto –inspect como 0x no soportan servidores del tipo CLUSTER, en realidad 0x, porque inspect lo pude utilizar pero no pude encontrar mis procesos.

Los procesos que más demoraron fueron los encargados de que el server levante ya node como express.

Diego Fernando Ramirez

CONCLUSION:

Fue muy enriquecedor poder probar cada endpoint y cada método que tenemos para hacer pruebas de stress a nuestro server. Me encanto poder ver los procesos y graficarlo para ver qué es lo que está haciendo que nuestro server sea más lento y me encantaría probar cada server para ver su performance y como poder mejorarla.

En general los servers en modo cluster demorar en su arranque, pero sus procesos son más rápidos.

No todas las formas de stressar el server son aptas para todas las configuraciones. Por lo que cada prueba tiene tendencia a usar sus propios métodos.

Para el análisis general me gustó mucho más usar autocannon porque configuro todo en un archivo individual y solo me genera el resultado, quizá estaría mejor que me generara un archivo consecuencia de la prueba. Artillery me fue más útil para probar individualmente cada endpoint del server y me permitio forzarlo tanto para fork como cluster sin problemas y me genero un archivo para cada prueba, quizá lo único malo sea que es por comandos. El inspect que nos proporcionan Chrome y Node fue interesante porque de manera muy sutil nos muestra los segundo que demoro en cada función individual, me gustaría realizar más pruebas en futuros servidores.

En cuando al grafico de Flama es interesante de ver pero me resulta muy difícil desmenuzar cada parte por lo que imagino que el grafico es más útil cuando utilizamos servidores más grandes con mucho volumen de datos.