Proyecto: HardwareBulls

Test de perfilamiento: /api/products

HardwareBulls

PRELIMINARES - Pautas de Trabajo por interpretación:

Atento a lo indicado en la presentación de Power Point, correspondiente al Desafío de Pre-Entrega 3, se solicita:

"Realizar una prueba de performance en modo local, con y sin cluster, utilizando Artillery en el endpoint del listado de productos (con el usuario vez logueado). Verificar los resultados."

Según lo solicitado se entiende que se debe realizar un test de performance de carga.

- ✓ Que por "modo local", se apunta a un servidor local (no Heroku);
- ✓ Que por "con y sin cluster", se utilizará la código propio desarrollado de manera local por no indicarse el uso de manejadores en la producción de procesos (no PM2 - no Forever).
- ✓ Que la herramienta de uso debe ser Artillery, y por su ausencia no corresponde el uso de Autocanon o 0X.
- ✓ Que por "en el endpoint del listado de productos (con el usuario una vez logueado)", se interpreta que para el análisis se busca interpretar que sucede a la hora de la carga de los productos. Para lo que se desarrolla un endpoint test para tal tarea.

01

Test de perfilamiento con el comando --prof

HardwareBulls

Test Modo Fork

```
9   router.get("/test", (req, res)=>{
10    const data = await productsDao.listAll()
11    res.render("products", {products: data})
12  });
```

Se crea la ruta test, para analizar el desempeño puntual.

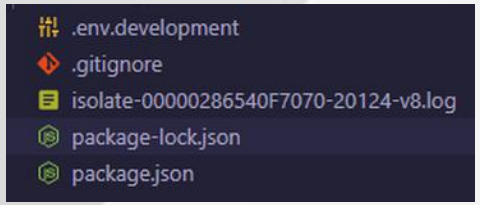
```
##MODO FORK O CLUSTER
SERVERMODE="FORK"
```

Por cuestiones de practicidad del desarrollador, se usa una variable de environment

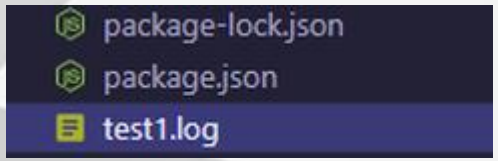
```
PROBLEMAS  SALIDA  TERMINAL  SQL CONSOLE  CONSOLA DE DEPURACIÓN

Ari@DESKTOP-PPRBKQD MINGW64 ~/Documents/repobackend/curso-ch-backend (main)
$ node --prof ./src/app.js
{"level":"info","message":"Server iniciado en modo:[FORK] con id de Proceso [20124] en puesto [8080]"}
{"level":"info","message":"Conectado a la Base de Datos de los Usuarios"}
```

Se ejecuta el comando node seguido de con el perfilador nativo de V8 en nodeJS

HardwareBulls



Se crea el archivo isolate. Que luego se dará en llamar “test1”



En otra terminal ejecutamos el comando de artillery para 20 request en 50 rafagas

```
Ari@DESKTOP-PPRBKQD MINGW64 ~/Documents/repobacken
d/curso-ch-backend (main)
$ artillery quick --count 20 -n 50 "http://localho
st:8080/test"
```

Ya se puede ver parte de los resultados en la consola

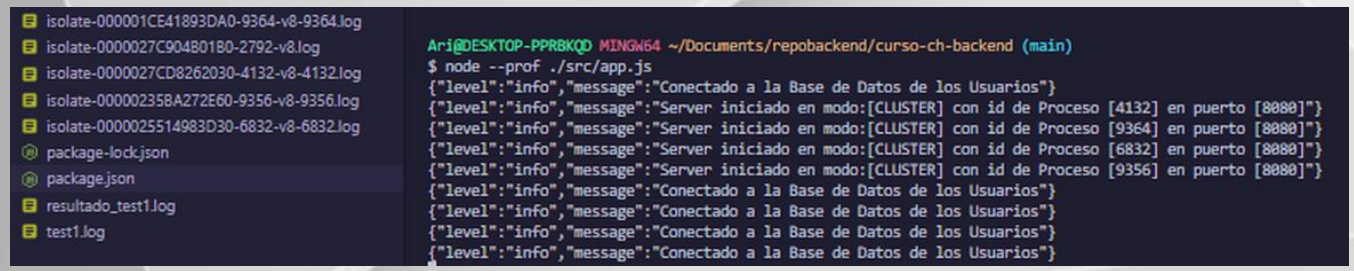
```
Phase completed: unnamed (index: 0, duration: 1s) 18:00:20(-030

-----------------------------------
Metrics for period to: 18:00:20(-0300) (width: 0.029s)
-----------------------------------

http.request_rate: ..... 1/sec
http.requests: ..... 1
vusers.created: ..... 1
vusers.created_by_name.0: ..... 1


-----------------------------------
Metrics for period to: 18:00:30(-0300) (width: 9.907s)
-----------------------------------

http.codes.200: ..... 317
http.request_rate: ..... 33/sec
http.requests: ..... 331
http.response_time:
  min: ..... 64
  max: ..... 1750
  median: ..... 301.9
  p95: ..... 804.5
  p99: ..... 1022.7
http.responses: ..... 317
vusers.created: ..... 19
vusers.created_by_name.0: ..... 19

All VUs finished. Total time: 35 seconds

-------------------------------
Summary report @ 18:00:52(-0300)
-------------------------------

http.codes.200: ..... 1000
http.request_rate: ..... 11/sec
http.requests: ..... 1000
http.response_time:
  min: ..... 64
  max: ..... 1750
  median: ..... 273.2
  p95: ..... 889.1
  p99: ..... 982.6
http.responses: ..... 1000
vusers.completed: ..... 20
vusers.created: ..... 20
vusers.created_by_name.0: ..... 20
vusers.failed: ..... 0
vusers.session_length:
  min: ..... 28534.3
  max: ..... 30679.5
  median: ..... 30040.3
  p95: ..... 30647.1
  p99: ..... 30647.1
```

HardwareBulls

Ya se puede ver parte de los resultados en la consola

Test Modo Cluster

Ejecutamos las mismas tareas con la diferemcia de que recibimos 5 archivos isolate por cada cluster del cpu:




```
Phase completed: unnamed (index: 0, duration: 1s) 18:41:04(-0300)

-------------------------------------
Metrics for period to: 18:41:10(-0300) (width: 6.657s)
-------------------------------------

http.codes.200: ..... 208
http.request_rate: ..... 34/sec
http.requests: ..... 224
http.response_time:
  min: ..... 85
  max: ..... 1379
  median: ..... 232.8
  p95: ..... 907
  p99: ..... 1326.4
http.responses: ..... 208
vusers.created: ..... 20
vusers.created_by_name.0: ..... 20


-------------------------------------
Metrics for period to: 18:41:20(-0300) (width: 9.338s)
-------------------------------------

http.codes.200: ..... 328
http.request_rate: ..... 35/sec
http.requests: ..... 328
http.response_time:
  min: ..... 142
  max: ..... 1815
  median: ..... 223.7
  p95: ..... 854.2
  p99: ..... 1043.3
http.responses: ..... 328


-------------------------------
Summary report @ 18:41:35(-0300)
-------------------------------

http.codes.200: ..... 1000
http.request_rate: ..... 17/sec
http.requests: ..... 1000
http.response_time:
  min: ..... 85
  max: ..... 1815
  median: ..... 219.2
  p95: ..... 889.1
  p99: ..... 1153.1
http.responses: ..... 1000
vusers.completed: ..... 20
vusers.created: ..... 20
vusers.created_by_name.0: ..... 20
vusers.failed: ..... 0
vusers.session_length:
  min: ..... 28282.6
  max: ..... 31262.8
  median: ..... 30040.3
  p95: ..... 30647.1
  p99: ..... 30647.1
```

HardwareBulls

VISTA COMPARATIVA

Summary en el podemos observar:
Ticks: la cantidad de datos aleatorios extraídos de la aplicación que desarrollé.
Total: el número total de muestras recolectadas.
Nonlib: el porcentaje de código que no es de una libraries.
Name: en que lenguaje se encuentra desarrollado.

El modo fork solo dispone de un Cpu y vuelca todo el trabajo en él.

```
[Summary]:
  ticks  total  nonlib   name
     61   0.3%  100.0%  JavaScript
      0   0.0%    0.0%  C++
     36   0.2%   59.0%  GC
  23799  99.7%          Shared libraries
```

Conclusión: **Lo que podemos inferir es que la aplicación, en la segunda prueba, incorporó una pieza de código que le permitió volcar el trabajo en diferentes cluster por el nro. de CPU. Lo que reporta luego en mejor desempeño a la hora de navergar la página.**

```
[Summary]:
  ticks  total  nonlib   name
      6   0.0%  100.0%  JavaScript
      0   0.0%    0.0%  C++
      7   0.0%  116.7%  GC
  18872  100.0%          Shared libraries
```

```
[Summary]:
  ticks  total  nonlib   name
     14   0.1%  100.0%  JavaScript
      0   0.0%    0.0%  C++
      7   0.0%   50.0%  GC
  18691  99.9%          Shared libraries
```

```
[Summary]:
  ticks  total  nonlib   name
     10   0.1%  100.0%  JavaScript
      0   0.0%    0.0%  C++
     10   0.1%  100.0%  GC
  18720  99.9%          Shared libraries
```

El Modo CLuster tiene cuatro cpus para volcar el trabajo y llevarlo de un modo un poco más distribuido

```
[Summary]:
  ticks  total  nonlib   name
     53   0.3%   98.1%  JavaScript
      0   0.0%    0.0%  C++
     25   0.1%   46.3%  GC
  18762  99.7%          Shared libraries
      1   0.0%          Unaccounted
```

```
[Summary]:
  ticks  total  nonlib   name
     31   0.2%  100.0%  JavaScript
      0   0.0%    0.0%  C++
     23   0.1%   74.2%  GC
  18773  99.8%          Shared libraries
```

02

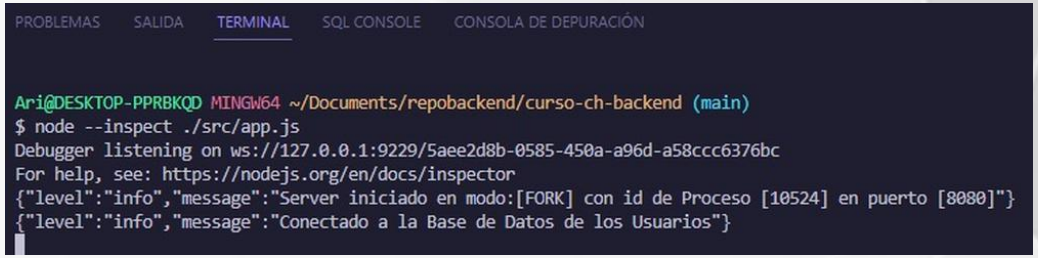
Test de perfilamiento con el comando --inspect

HardwareBulls

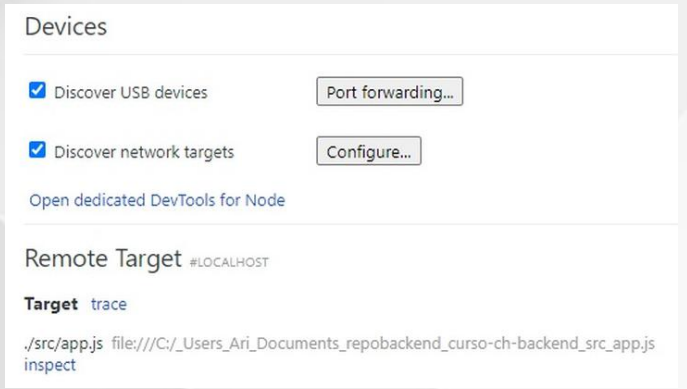
Test Modo Fork y Cluster

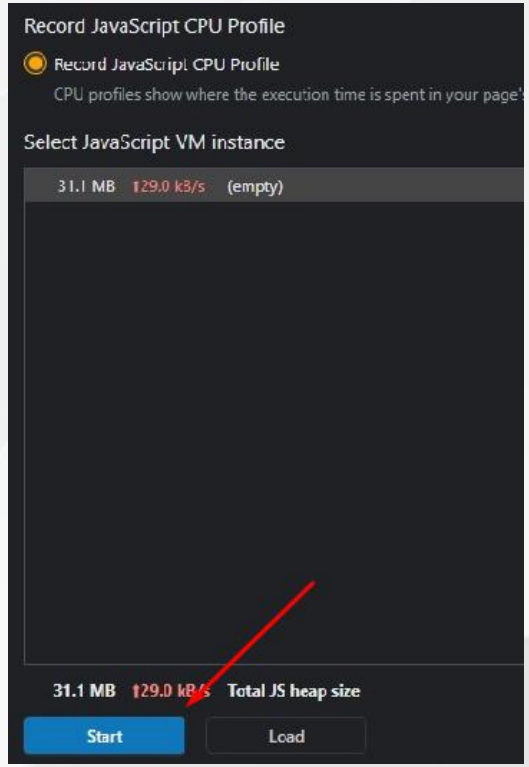
El proceso para practicar el test es prácticamente el mismo lo que cambia es la variable environment.

Se ejecuta el comando node con el doble flag inspect



Abrimos el browser de chrome. Vamos a las DevTools y abrimos el target inspect de ./src/app.js y allí hacemos click en start



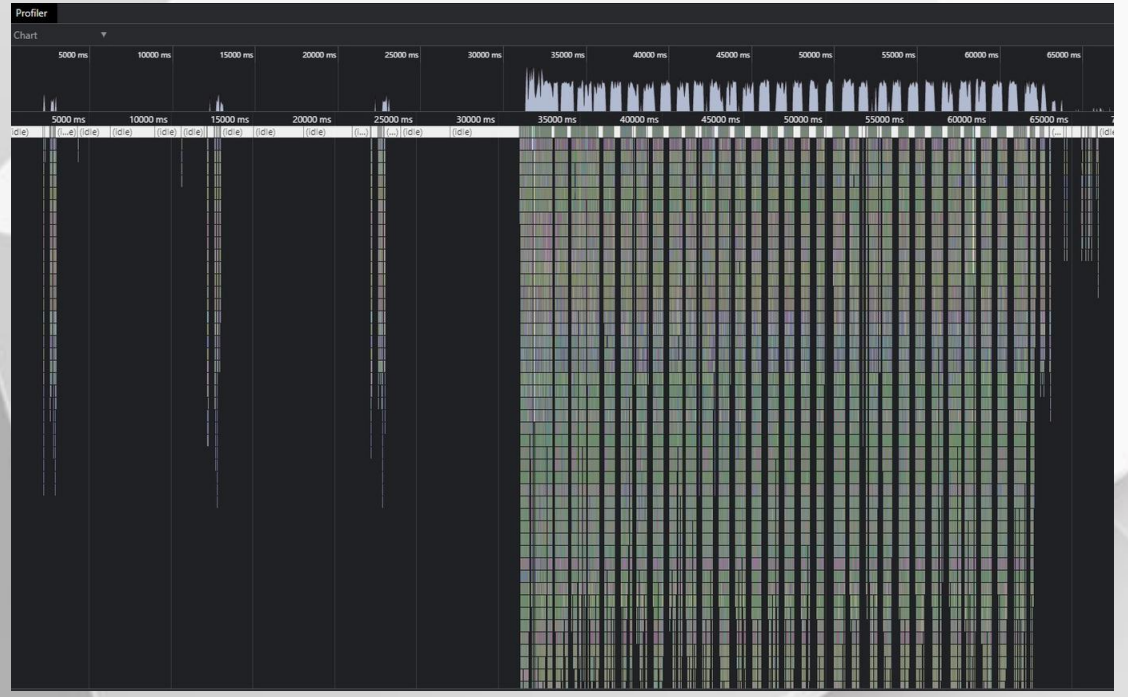


Y ejecutamos el mismo comando de Artillery, con la misma cantidad de consultas por rafaga de consultas

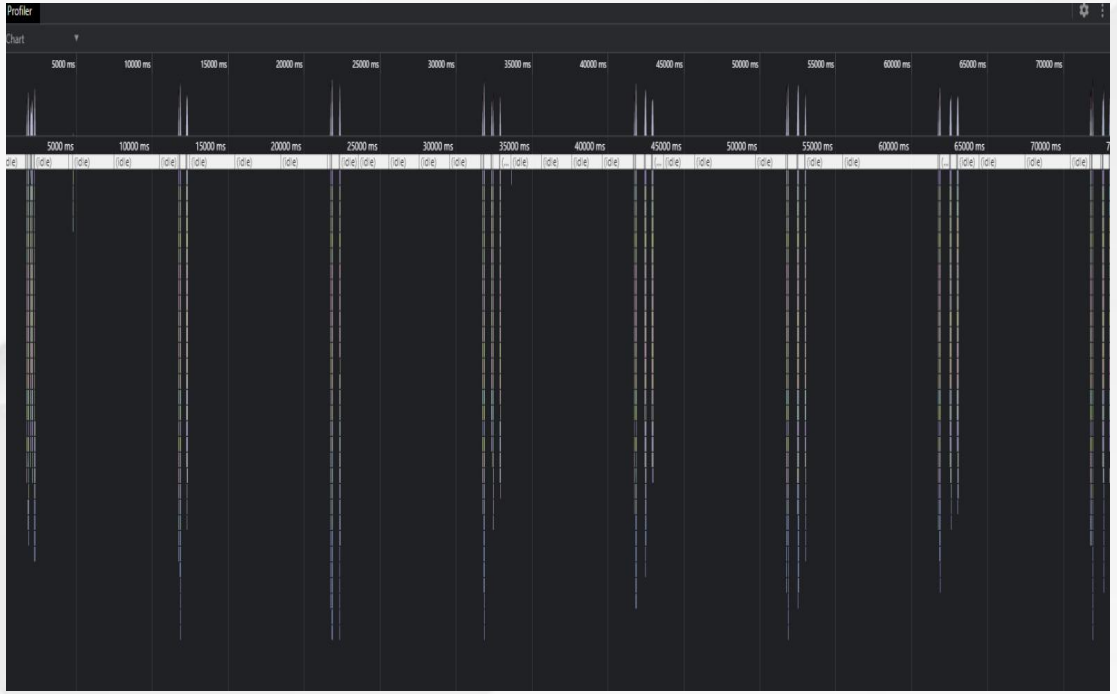
HardwareBulls

VISTA COMPARATIVA

Vista del Chart en Modo Fork



Vista del Chart en Modo Cluster



Podemos ver que la pila de funciones desplegadas en un stack son más profuncas y son más extensas en su desarrollo temporal, por esa misma cuestión. La imagen hace conicidir todo con la conclusión a la que se genera en el primer test realizado