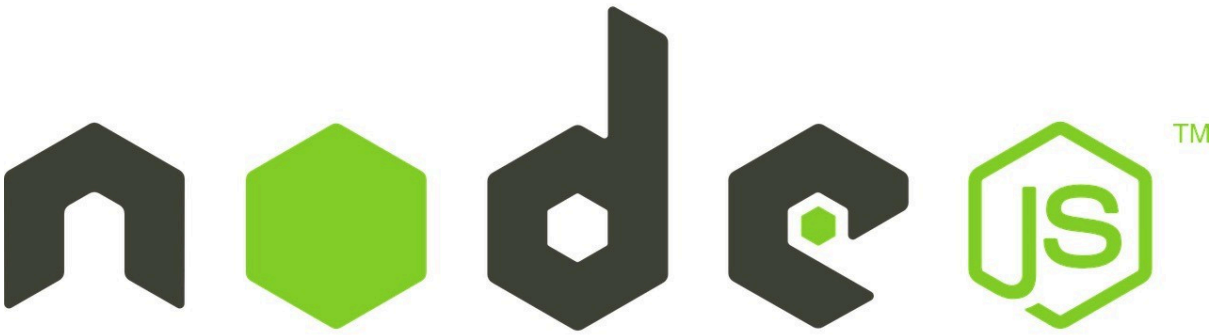
# Despliegue cluster con Node Express.js




**Made By**: Lucas Garcia Diaz.

# 1.-Prueba sin el cluster.

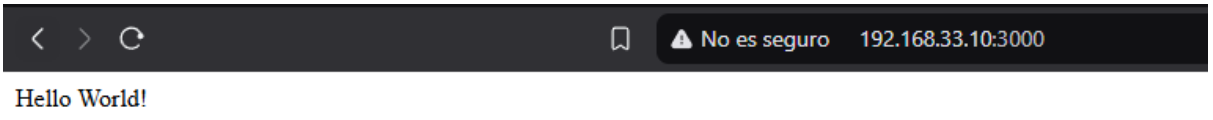
Instalamos las dependencias del cluster.

```
    sudo apt-get update -y
    sudo apt-get install -y nodejs npm
```

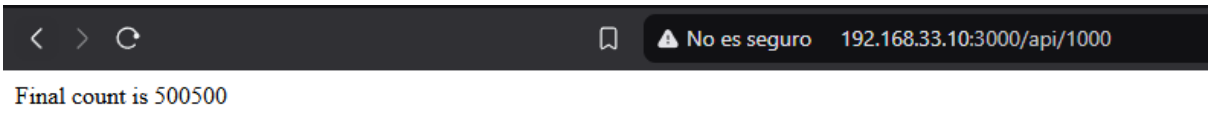
Creamos una carpeta donde irá nuestro proyecto:

```
  mkdir sincluster
  cd sincluster
  npm init -y
  npm install express
  cp -v /vagrant/config/sincluster/index.js /home/vagrant/sincluster/
  node index.js
```

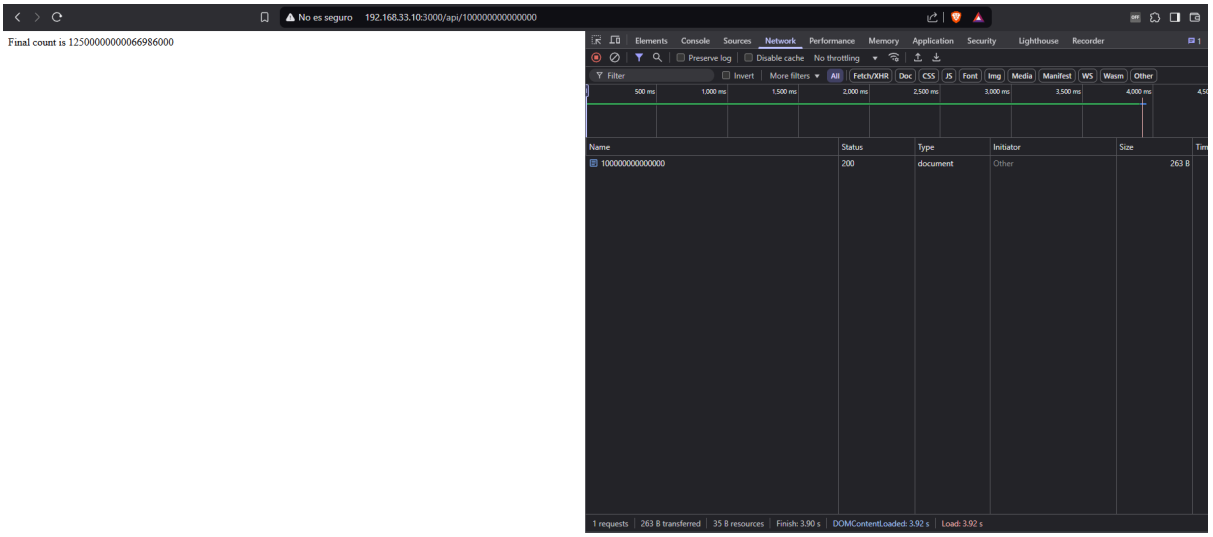
Para ***'/'***:




Para ***'/api/n:'***:




Tiempos:



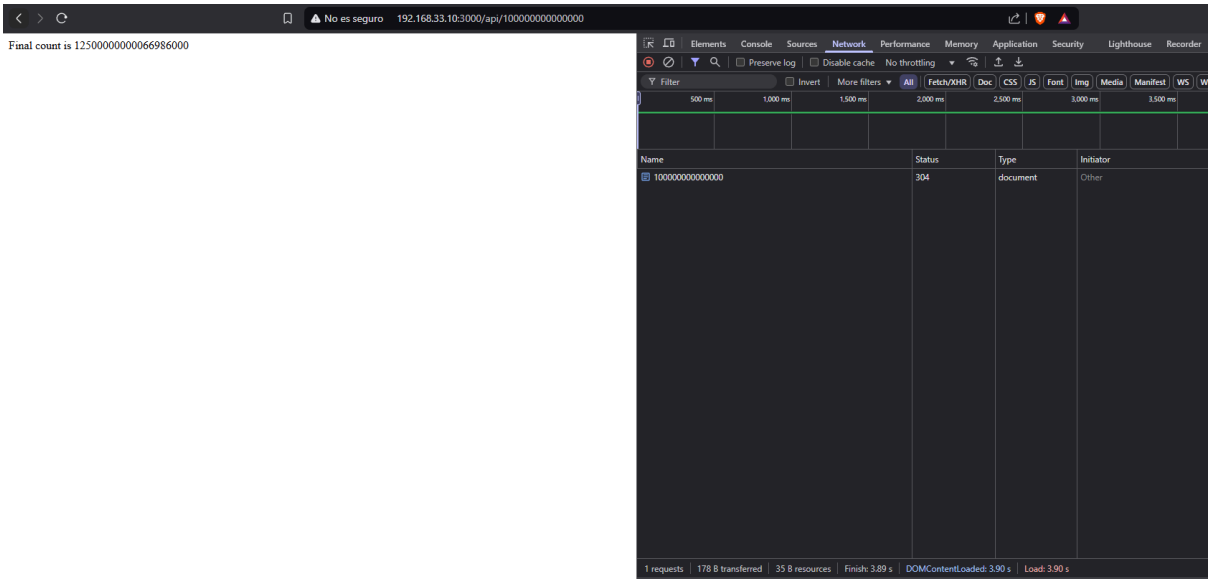

**3.92 segundos** con una cifra de 100000000000000.

# 2.-Prueba con  clusters.

Creamos una carpeta donde irá nuestro proyecto:

```
mkdir concluster
cd concluster
npm init -y
npm install express
cp -v /vagrant/config/concluster/index.js /home/vagrant/concluster/
node index.js
```

Tiempos:

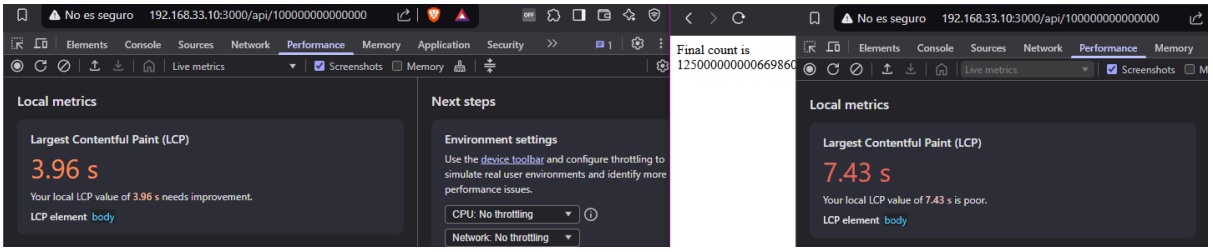



**3.90 segundos** con una cifra de 100000000000000.

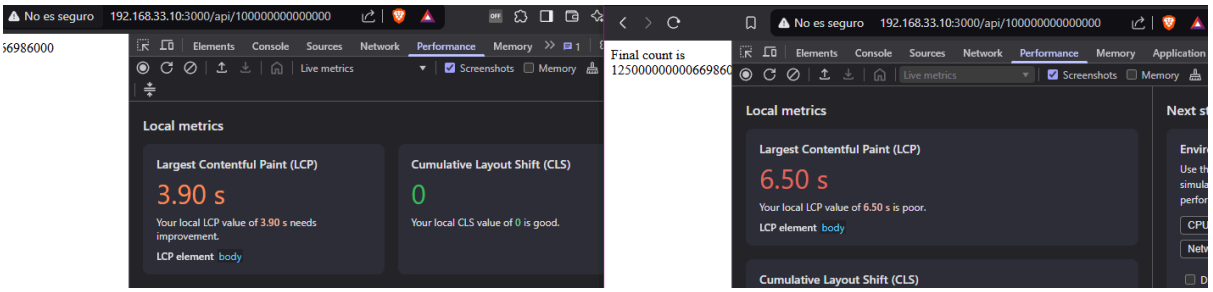
Igual.

Pero si los comprobamos a 2 clientes a la vez:

Prueba sin cluster:




Prueba con cluster:




Mejora un poco.

# 3.-Métricas de rendimiento.

Instalamos globalmente loadtest.

```
npm install -g loadtest
```

Ejecutamos la aplicación con el cluster y con otra terminal.

```
node index.js
loadtest http://localhost:3000/api/500000 -n 1000 -c 100
```

```
vagrant@bullseye:~$ loadtest http://localhost:3000/api/500000 -n 1000 -c 100
(node:1532) [DEP0060] DeprecationWarning: The `util._extend` API is deprecated. Please use Object.assign() instead.
(Use `node --trace-deprecation ...` to show where the warning was created)

Target URL:          http://localhost:3000/api/500000
Max requests:        1000
Concurrent clients:  100
Agent:               none

Completed requests:  1000
Total errors:        0
Total time:          0.741 s
Mean latency:        69.6 ms
Effective rps:       1350

Percentage of requests served within a certain time
  50%      70 ms
  90%      81 ms
  95%      84 ms
  99%      96 ms
 100%      104 ms (longest request)
```

Vamos a comprobar con cluster y sin cluster con 500000000 peticiones.

## Sin cluster:

```
Target URL:          http://localhost:3000/api/500000000
Max requests:        1000
Concurrent clients:  100
Agent:               none

Completed requests:  1000
Total errors:        0
Total time:          339.718 s
Mean latency:        32280.8 ms
Effective rps:       3

Percentage of requests served within a certain time
  50%      33869 ms
  90%      34377 ms
  95%      34521 ms
  99%      34671 ms
 100%      34711 ms (longest request)
```

Ha tardado **339.718s**

Con cluster:

```
Target URL:          http://localhost:3000/api/500000000
Max requests:        1000
Concurrent clients:  100
Agent:               none

Completed requests:  1000
Total errors:        0
Total time:          173.411 s
Mean latency:        16503.1 ms
Effective rps:       6

Percentage of requests served within a certain time
  50%      17241 ms
  90%      17634 ms
  95%      17751 ms
  99%      17930 ms
 100%      17970 ms (longest request)
```

Ha tardado **173.411s.**

El servidor con cluster ha sido **1.95x** más rápido que el que tenia no tenia cluster.

## 4.-Uso de PM2 para administrar un clúster de Node.js.

Instalamos PM2:

```
npm install pm2 -g
```

Lo ejecutamos de la siguiente manera:

```
pm2 start index.js -i 0
```


Para detenerlo.

```
pm2 stop index.js
```

Crearemos el archivo ecosystem donde podemos poner configuraciones especificas.

```
cp -v /vagrant/config/concluster/ecosystem.config.js
/home/vagrant/concluster
```

Ejecutamos con el archivo:

```
vagrant@bullseye:~/concluster$ sudo pm2 start ecosystem.config.js
[PM2][WARN] Applications Cluster not running, starting...
[PM2] App [Cluster] launched (2 instances)
```

| id | name | namespace | version | mode | pid | uptime | ↺ | status | cpu | mem | user | watching |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | Cluster | default | 1.0.0 | cluster | 249430 | 0s | 0 | online | 0% | 39.3mb | root | disabled |
| 1 | Cluster | default | 1.0.0 | cluster | 249437 | 0s | 0 | online | 0% | 33.9mb | root | disabled |

# 5.-Tarea:

Averiguar qué hace

- 1. pm2 ls .
- 2. pm2 logs.
- 3. pm2 monit.

1.-pm2 ls

```
vagrant@bullseye:~/concluster$ pm2 ls
```

| id | name | namespace | version | mode | pid | uptime | ↺ | status | cpu | mem | user | watching |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | index | default | 1.0.0 | fork | 0 | 0 | 0 | stopped | 0% | 0b | vagrant | disabled |

Lista los scripts o aplicaiones que se están ejecutando.

2.-pm2 logs

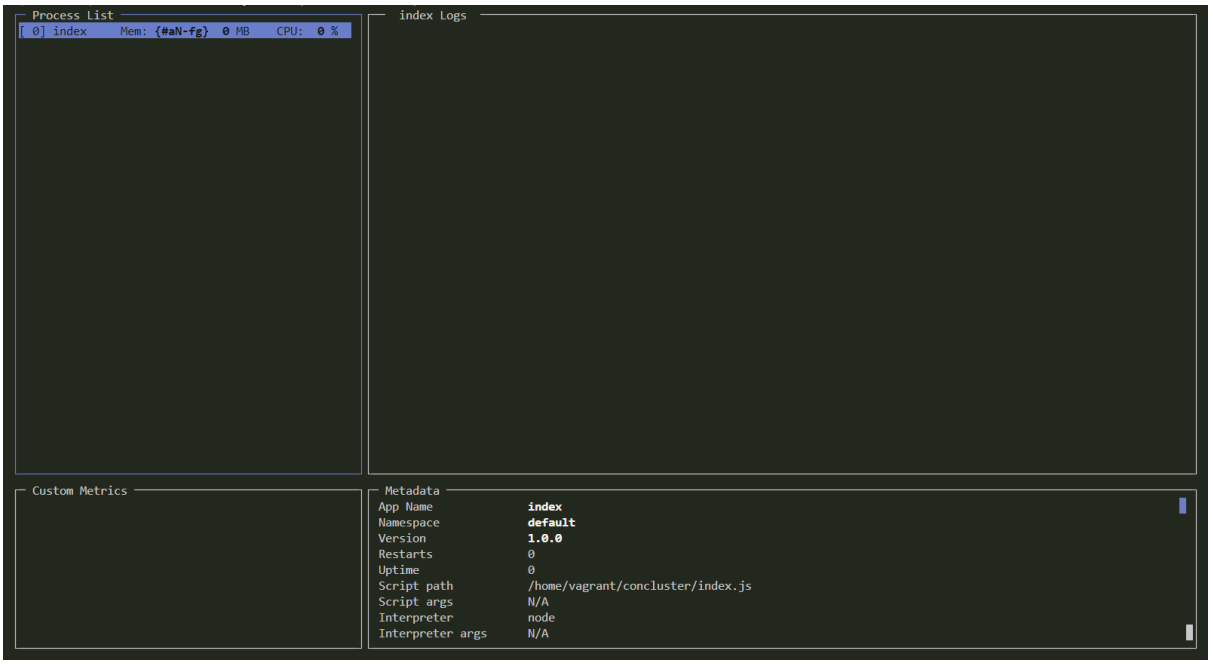
```
vagrant@bullseye:~/concluster$ pm2 logs
[TAILING] Tailing last 15 lines for [all] processes (change the value with --lines option)
/home/vagrant/.pm2/pm2.log last 15 lines:
PM2        | 2025-02-08T23:15:16: PM2 log: PM2 home              : /home/vagrant/.pm2
PM2        | 2025-02-08T23:15:16: PM2 log: PM2 PID file          : /home/vagrant/.pm2/pm2.pid
PM2        | 2025-02-08T23:15:16: PM2 log: RPC socket file       : /home/vagrant/.pm2/rpc.sock
PM2        | 2025-02-08T23:15:16: PM2 log: BUS socket file       : /home/vagrant/.pm2/pub.sock
PM2        | 2025-02-08T23:15:16: PM2 log: Application log path : /home/vagrant/.pm2/logs
PM2        | 2025-02-08T23:15:16: PM2 log: Worker Interval       : 30000
PM2        | 2025-02-08T23:15:16: PM2 log: Process dump file     : /home/vagrant/.pm2/dump.pm2
PM2        | 2025-02-08T23:15:16: PM2 log: Concurrent actions    : 2
PM2        | 2025-02-08T23:15:16: PM2 log: SIGTERM timeout       : 1600
PM2        | 2025-02-08T23:15:16: PM2 log: ================================================================================
PM2        | 2025-02-08T23:15:16: PM2 log: App [index:0] starting in -fork mode-
PM2        | 2025-02-08T23:15:16: PM2 log: App [index:0] online
PM2        | 2025-02-08T23:46:29: PM2 log: Stopping app:index id:0
PM2        | 2025-02-08T23:46:29: PM2 log: App [index:0] exited with code [0] via signal [SIGINT]
PM2        | 2025-02-08T23:46:29: PM2 log: pid=1743 msg=process killed

/home/vagrant/.pm2/logs/index-error.log last 15 lines:
0|index    | }
0|index    | Error: bind EADDRINUSE null:3000
0|index    |     at listenOnPrimaryHandle (node:net:2018:18)
0|index    |     at rr (node:internal/cluster/child:163:12)
0|index    |     at Worker.<anonymous> (node:internal/cluster/child:113:7)
0|index    |     at process.onInternalMessage (node:internal/cluster/utils:49:5)
0|index    |     at process.emit (node:events:536:35)
0|index    |     at emit (node:internal/child_process:949:14)
0|index    |     at process.processTicksAndRejections (node:internal/process/task_queues:91:21) {
0|index    |   errno: -98,
0|index    |   code: 'EADDRINUSE',
0|index    |   syscall: 'bind',
0|index    |   address: null,
0|index    |   port: 3000
0|index    | }

/home/vagrant/.pm2/logs/index-out.log last 15 lines:
0|index    | Worker 249229 started
0|index    | worker 249229 died
0|index    | Let's fork another worker!
```

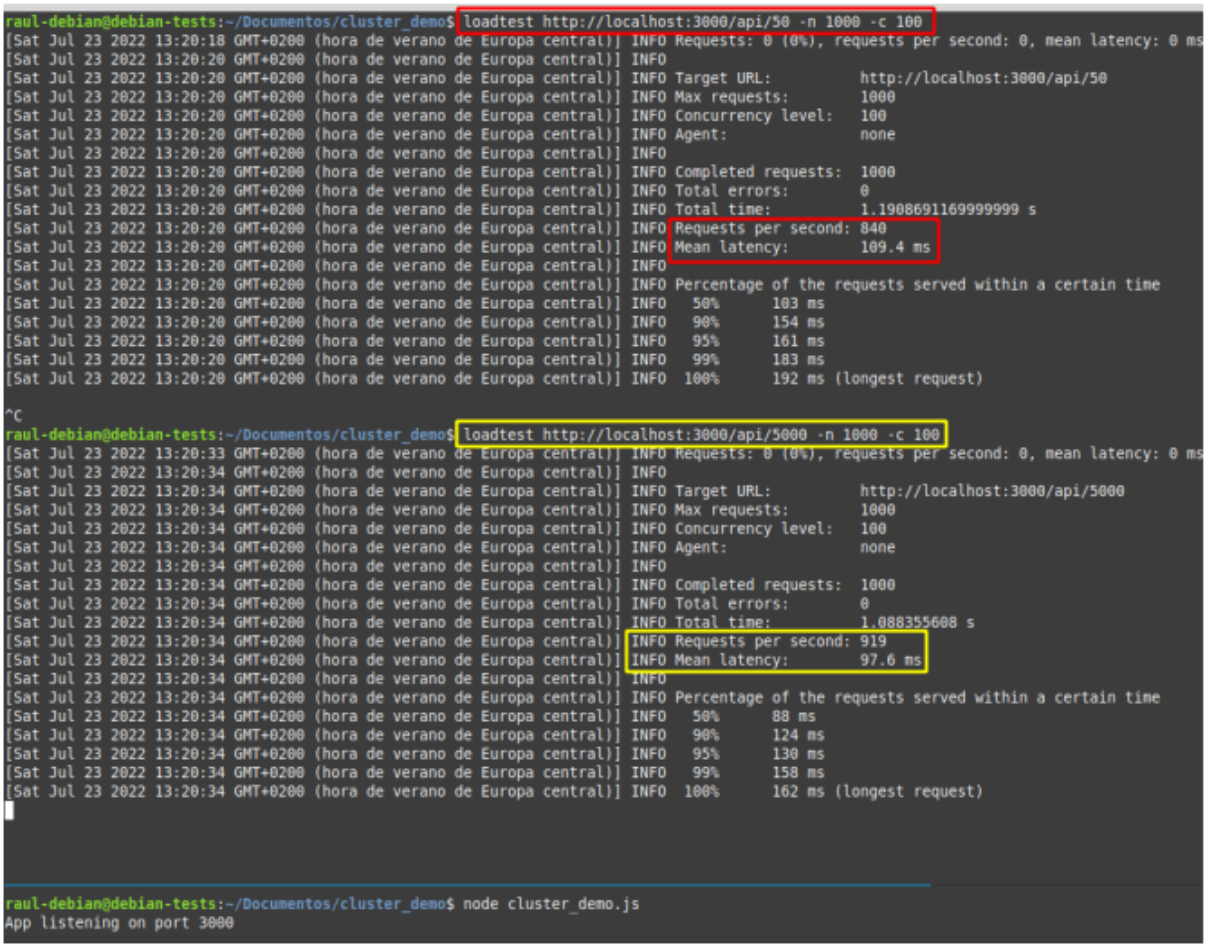
Muestra los logs.

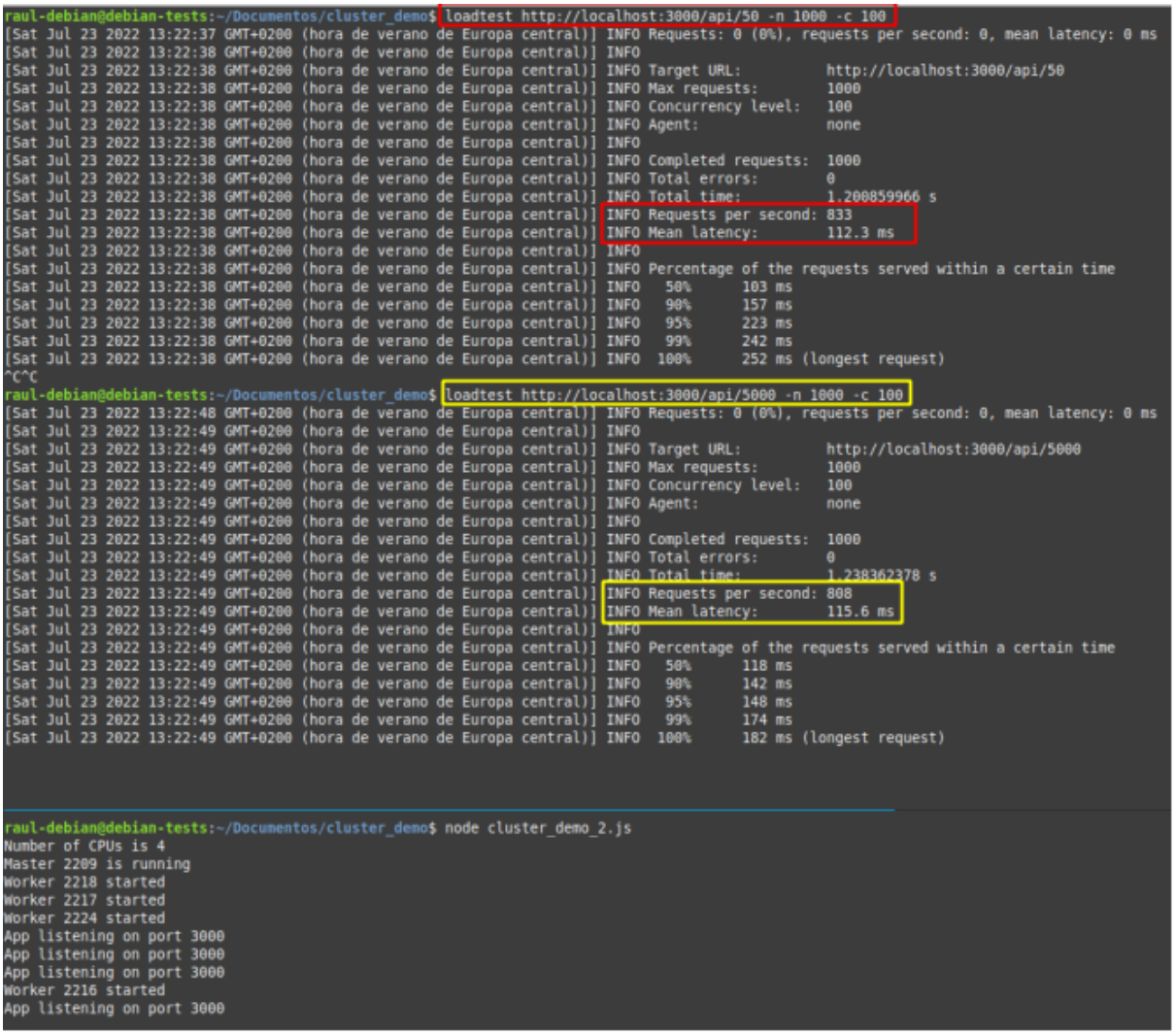
3.-pm2 monit.




Te sale un monitor a tiempo real de los detalles de las aplicaciones o script que han sido iniciadas.

# 6.-Cuestiones:






La primera imagen ilustra los resultados de unas pruebas de carga sobre la aplicación sin clúster y la segunda sobre la aplicación clusterizada.

¿Sabrías decir por qué en algunos casos concretos, como este, la aplicación sin clusterizar tiene mejores resultados?

Podría deberse a que el proceso de clusterización introduce una sobrecarga adicional en forma de comunicación entre procesos, coordinación y gestión de recursos compartidos. Además, una

aplicación sin clusterizar puede aprovechar de forma más directa la caché de la CPU y evitar los costes asociados a los cambios de contexto y a la sincronización entre procesos, lo que se traduce en mejores resultados en ciertas pruebas de carga.