1. Primero sin clúster

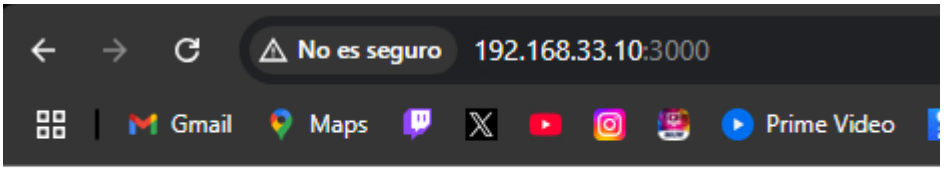
1-Instalamos node y revisamos su versión:

```
vagrant@bookworm:~$ node -v
v22.14.0
vagrant@bookworm:~$ npm -v
10.9.2
vagrant@bookworm:~$
```

2- Creamos la app

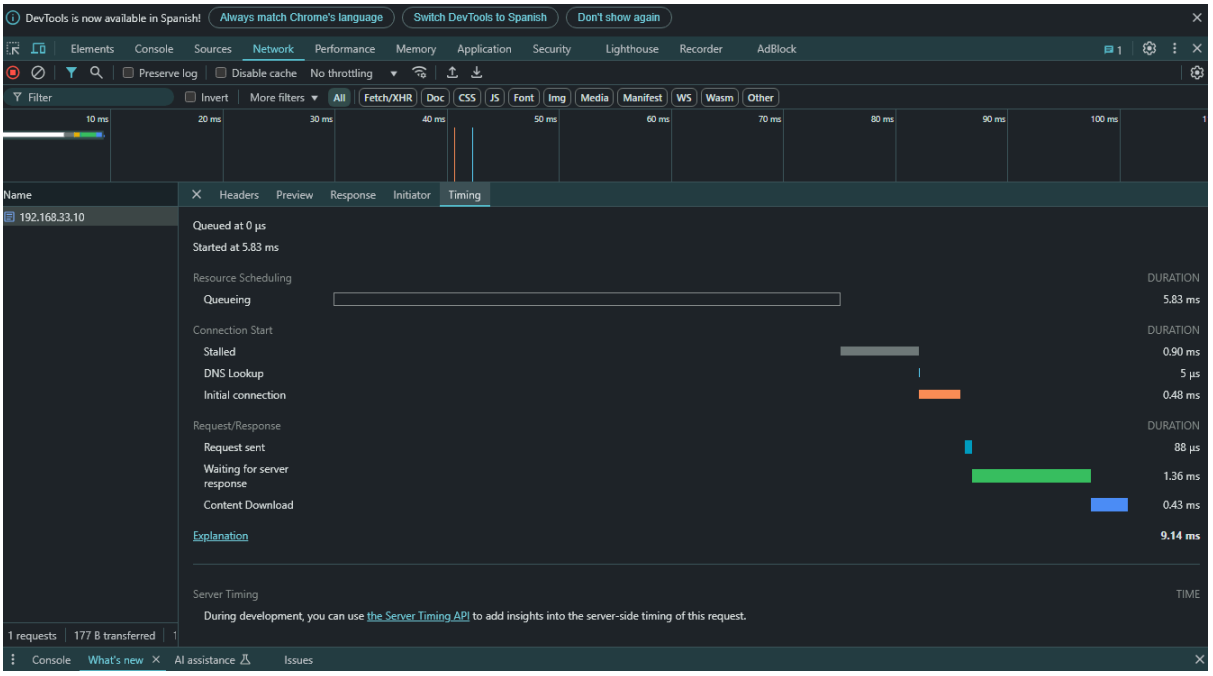
```
vagrant@bookworm:~$ mkdir app
vagrant@bookworm:~$ cd app
vagrant@bookworm:~/app$ npm init -y
Wrote to /home/vagrant/app/package.json:

{
  "name": "app",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": ""
}



vagrant@bookworm:~/app$ npm install express

added 69 packages, and audited 70 packages in 5s

14 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
vagrant@bookworm:~/app$
```

3-Nos conectamos para comprobar que todo va bien

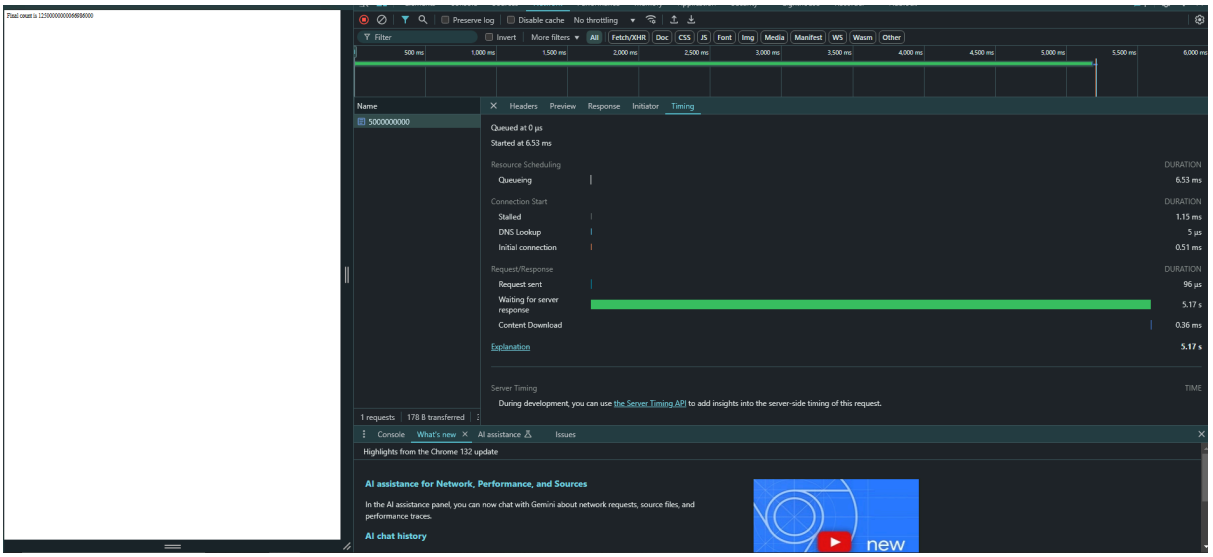


Hello World!

4-Hacemos una petición con un valor pequeño para ver el tiempo de respuesta



5- Aumentamos el valor y volvemos ha hacer una petición para comparar los tiempos de respuesta



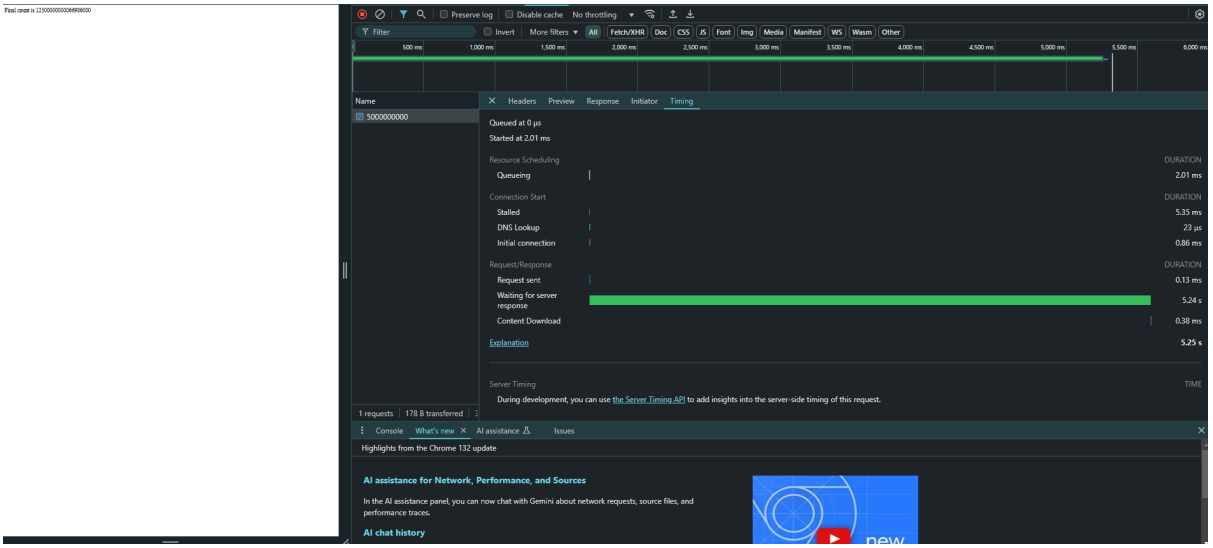
Como vemos el tiempo a aumentado consideradamente, unos 5 segundos

2. Ahora con cluster

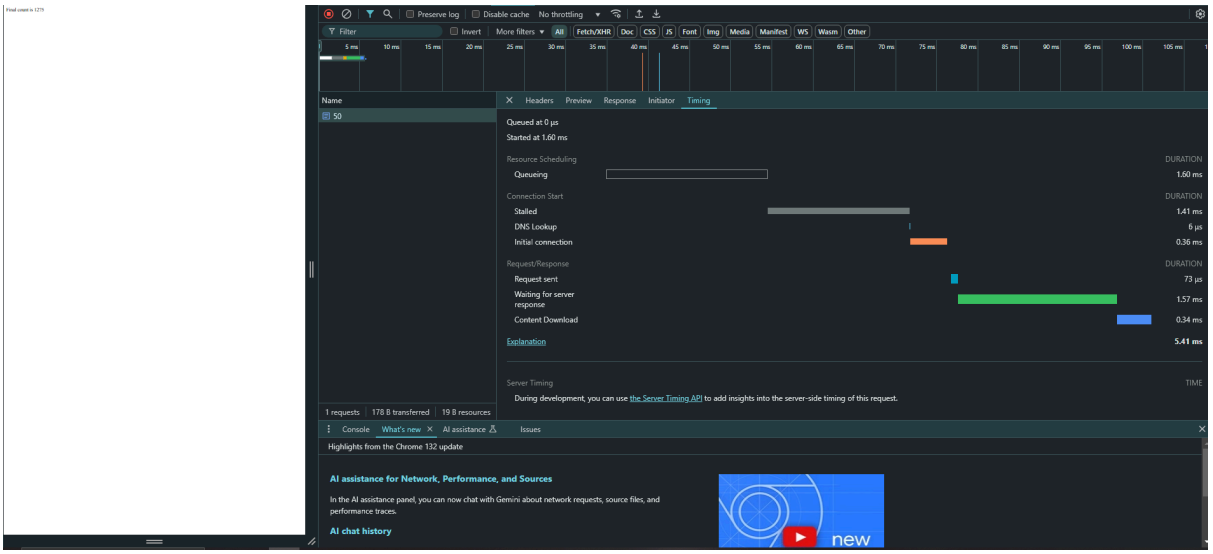
1- Volvemos a crear una aplicacion con cluster

```
vagrant@bookworm:~/app$ node app-cluster.js
Master 25974 is running
Worker 25981 started on port 3000
Worker 25982 started on port 3000
```

2-Hacemos una petición grande



Y comparamos con una petición pequeña:



La segunda solicitud espera a que termine la primera para ofrecer una respuesta

3. Métricas de rendimiento

Realizaremos una prueba de carga en nuestras dos aplicaciones para ver cómo cada una maneja una gran cantidad de conexiones entrantes. Usaremos el paquete loadtest para esto

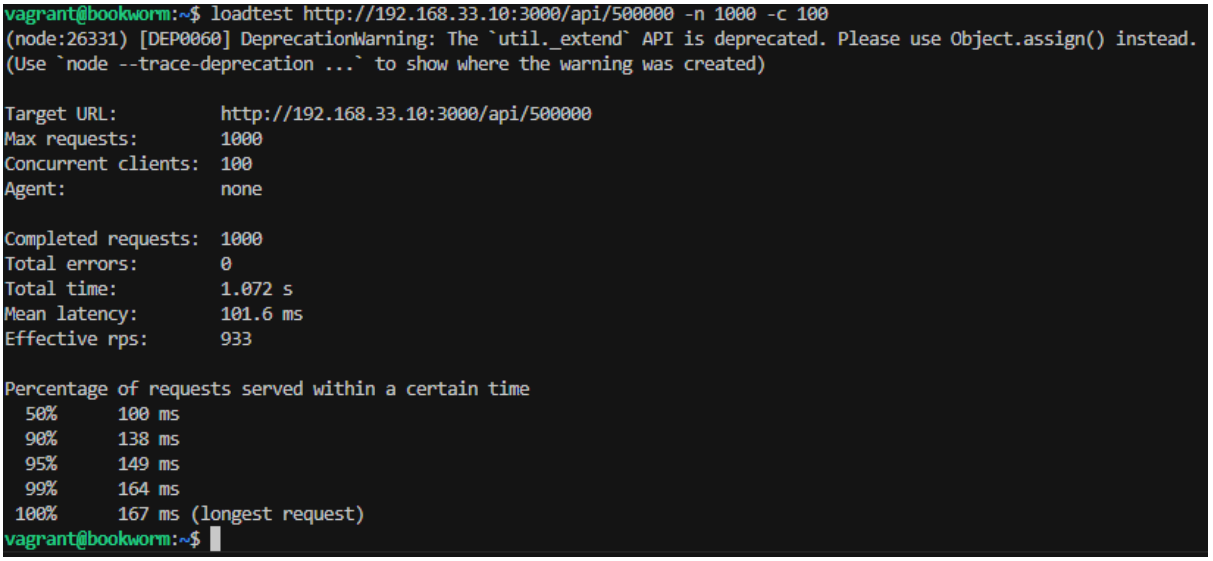
Mientras ejecutamos la aplicación, en otro terminal realizamos la siguiente prueba de carga:

loadtest http://localhost:3000/api/500000 -n 1000 -c 100

El comando anterior enviará 1000 solicitudes a la URL dada, de las cuales 100 son concurrentes.

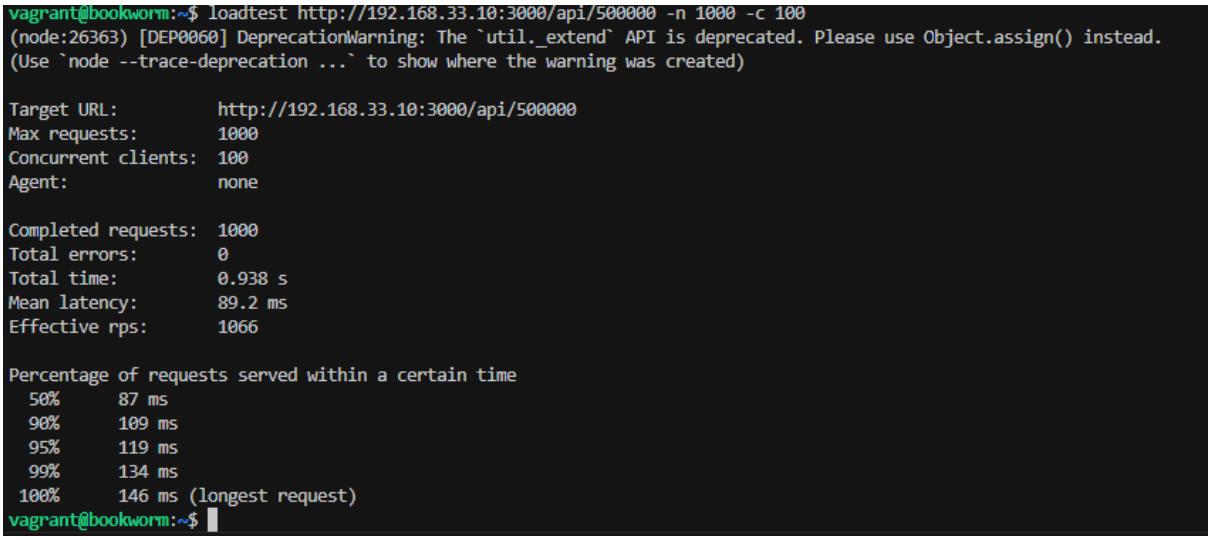
1. Tiempo total de prueba: 1. 072 s

2. Latencia media, el tiempo promedio que tarda en completar una solicitud: 101. 6 ms
3. RPS (Request per second): 933



Esta petición es a la aplicación sin cluster, ahora probamos con la de cluster:

1. Tiempo total de prueba: 0.938 s
2. Latencia media: 89. 2 ms
3. RPS (Request per second):1066



Como vemos los tiempos de latencia son menores en la aplicación con cluster

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

Vamos a utilizarlo con nuestra primera aplicación, la que no tenia cluester. Para ello ejecutaremos el siguiente comando:

pm2 start app.js -i 0

```
vagrant@bookworm:~/app$ pm2 start app.js -i 0

                        -------------

__/\\\\\\\\\\\\\____/\\\\____________/\\\\____/\\\\\\\\\_____
 _\/\\\/////////\\\_\/\\\\\\________/\\\\\\__/\\\///////\\\___
  _\/\\\_______\/\\\_\/\\\//\\\____/\\\//\\\_\///______\//\\\__
   _\/\\\\\\\\\\\\\/__\/\\\\///\\\/\\\/_\/\\\___________/\\\/___
    _\/\\\/////////____\/\\\__\///\\\/___\/\\\________/\\\//_____
     _\/\\\_____________\/\\\____\///_____\/\\\_____/\\\//________
      _\/\\\_____________\/\\\_____________\/\\\___/\\\/___________
       _\/\\\_____________\/\\\_____________\/\\\__/\\\\\\\\\\\\\\\_
        _\///______________\///______________\///__\///////////////__


                          Runtime Edition

        PM2 is a Production Process Manager for Node.js applications
                     with a built-in Load Balancer.

                Start and Daemonize any application:
                $ pm2 start app.js

                Load Balance 4 instances of api.js:
                $ pm2 start api.js -i 4

                Monitor in production:
                $ pm2 monitor

                Make pm2 auto-boot at server restart:
                $ pm2 startup

                To go further checkout:
                http://pm2.io/


                        -------------

[PM2] Spawning PM2 daemon with pm2_home=/home/vagrant/.pm2
[PM2] PM2 Successfully daemonized
[PM2] Starting /home/vagrant/app/app.js in cluster_mode (0 instance)
[PM2] Done.
```

id	name	namespace	version	mode	pid	uptime	↺	status	cpu	mem	user	watching
0	app	default	1.0.0	cluster	26409	0s	0	online	0%	55.1mb	vagrant	disabled
1	app	default	1.0.0	cluster	26416	0s	0	online	0%	51.9mb	vagrant	disabled

```
vagrant@bookworm:~/app$
```

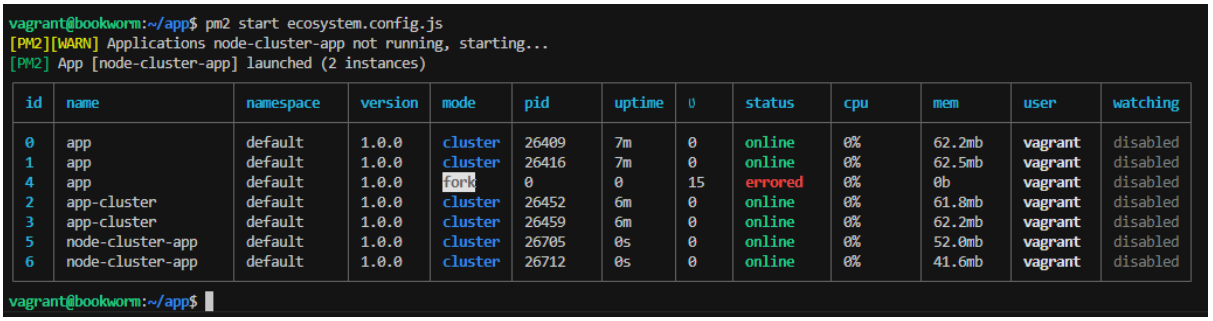
Y para detener la aplicación: pm2 stop app.js

```
vagrant@bookworm:~/app$ pm2 stop app.js
[PM2] Applying action stopProcessId on app [app.js](ids: [ 0, 1, 4, 5, 6 ])
[PM2] [app](0) ✓
[PM2] [app](1) ✓
[PM2] [app.js](4) ✓
[PM2] [node-cluster-app](5) ✓
[PM2] [node-cluster-app](6) ✓
```

id	name	namespace	version	mode	pid	uptime	↺	status	cpu	mem	user	watching
0	app	default	1.0.0	cluster	0	0	0	stopped	0%	0b	vagrant	disabled
1	app	default	1.0.0	cluster	0	0	0	stopped	0%	0b	vagrant	disabled
4	app	default	1.0.0	fork	0	0	15	stopped	0%	0b	vagrant	disabled
2	app-cluster	default	1.0.0	cluster	26452	2h	0	online	0%	63.7mb	vagrant	disabled
3	app-cluster	default	1.0.0	cluster	26459	2h	0	online	0%	64.3mb	vagrant	disabled
5	node-cluster-app	default	1.0.0	cluster	0	0	0	stopped	0%	0b	vagrant	disabled
6	node-cluster-app	default	1.0.0	cluster	0	0	0	stopped	0%	0b	vagrant	disabled

```
vagrant@bookworm:~/app$
```

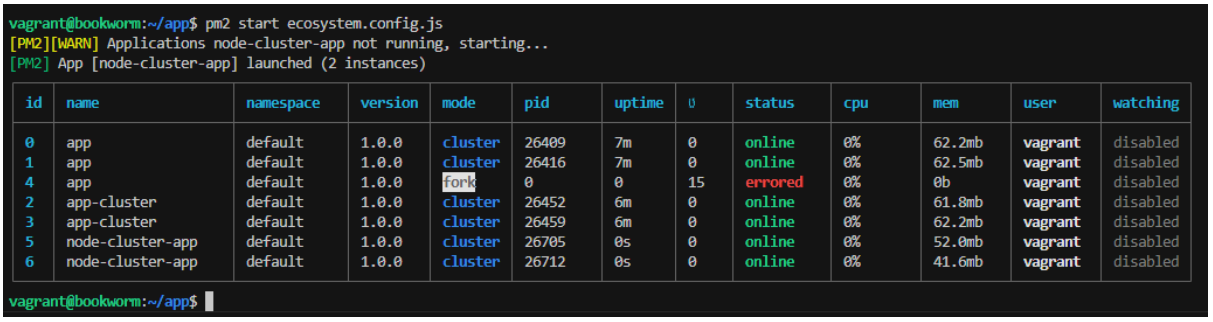
Crearemos el archivo Ecosystem con el siguiente comando:
pm2 ecosystem



Que generará un archivo llamado ecosystem.config.js. Para el caso concreto de nuestra aplicación,
necesitamos modificarlo como se muestra a continuación:

```
module.exports = {
    apps: [{
        name: "node-cluster-app",
        script: "app.js",
        instances: 0,
        exec_mode: "cluster",
    }],
};
```

Al configurar exec_mode con el valor cluster, le indica a PM2 que balancee la carga entre cada instancia. instances está configurado a 0 como antes, lo que generará tantos workers como núcleos de CPU



Tarea
Investiga los siguientes comandos y explica que salida por terminal nos ofrecen y para qué se utilizan:

1. pm2 ls

```
vagrant@bookworm:~/app$ pm2 ls
```

id	name	namespace	version	mode	pid	uptime	↺	status	cpu	mem	user	watching
0	app	default	1.0.0	cluster	26409	9m	0	online	0%	62.5mb	vagrant	disabled
1	app	default	1.0.0	cluster	26416	9m	0	online	0%	62.5mb	vagrant	disabled
4	app	default	1.0.0	fork	0	0	15	errored	0%	0b	vagrant	disabled
2	app-cluster	default	1.0.0	cluster	26452	8m	0	online	0%	62.8mb	vagrant	disabled
3	app-cluster	default	1.0.0	cluster	26459	8m	0	online	0%	62.8mb	vagrant	disabled
5	node-cluster-app	default	1.0.0	cluster	26705	88s	0	online	0%	62.0mb	vagrant	disabled
6	node-cluster-app	default	1.0.0	cluster	26712	88s	0	online	0%	61.5mb	vagrant	disabled

```
vagrant@bookworm:~/app$
```

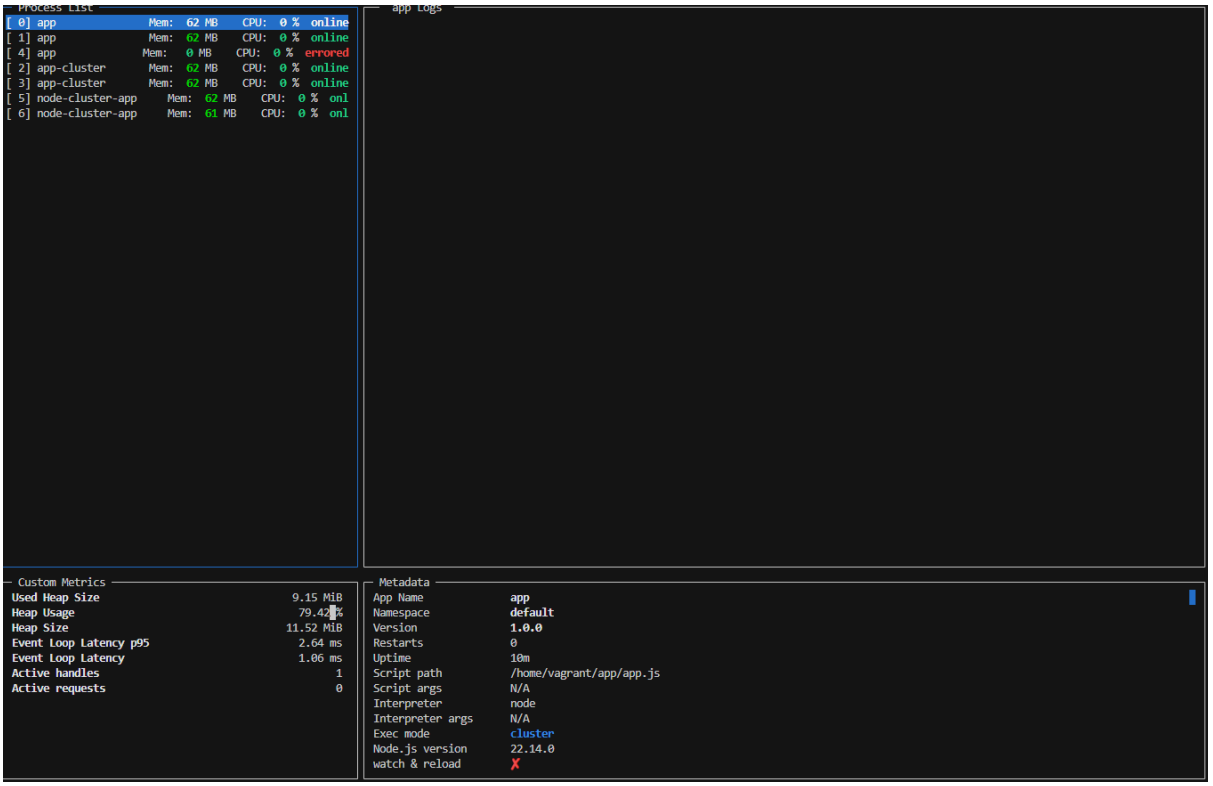
Lista los procesos en ejecución

2. pm2 logs

```
vagrant@bookworm:~/app$ 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        | 2025-02-15T12:14:33: PM2 log: App [app:4] exited with code [1] via signal [SIGINT]
PM2        | 2025-02-15T12:14:33: PM2 log: App [app:4] starting in -fork mode-
PM2        | 2025-02-15T12:14:33: PM2 log: App [app:4] online
PM2        | 2025-02-15T12:14:33: PM2 log: App [app:4] exited with code [1] via signal [SIGINT]
PM2        | 2025-02-15T12:14:33: PM2 log: App [app:4] starting in -fork mode-
PM2        | 2025-02-15T12:14:33: PM2 log: App [app:4] online
PM2        | 2025-02-15T12:14:33: PM2 log: App [app:4] exited with code [1] via signal [SIGINT]
PM2        | 2025-02-15T12:14:33: PM2 log: App [app:4] starting in -fork mode-
PM2        | 2025-02-15T12:14:33: PM2 log: App [app:4] online
PM2        | 2025-02-15T12:14:33: PM2 log: App [app:4] exited with code [1] via signal [SIGINT]
PM2        | 2025-02-15T12:14:33: PM2 log: Script /home/vagrant/app/app.js had too many unstable restarts (16). Stopped. "errored"
PM2        | 2025-02-15T12:19:29: PM2 log: App [node-cluster-app:5] starting in -cluster mode-
PM2        | 2025-02-15T12:19:29: PM2 log: App [node-cluster-app:5] online
PM2        | 2025-02-15T12:19:29: PM2 log: App [node-cluster-app:6] starting in -cluster mode-
PM2        | 2025-02-15T12:19:29: PM2 log: App [node-cluster-app:6] online

/home/vagrant/.pm2/logs/app-out.log last 15 lines:
0|app      | App listening on port 3000
0|app      | App listening on port 3000

/home/vagrant/.pm2/logs/app-cluster-error.log last 15 lines:
/home/vagrant/.pm2/logs/app-cluster-out.log last 15 lines:
2|app-clus | Worker 26452 started on port 3000
2|app-clus | Worker 26459 started on port 3000

/home/vagrant/.pm2/logs/app-error.log last 15 lines:
0|app      |     at listenInCluster (node:net:1994:12)
0|app      |     at Server.listen (node:net:2099:7)
0|app      |     at Function.listen (/home/vagrant/app/node_modules/express/lib/application.js:635:24)
0|app      |     at Object.<anonymous> (/home/vagrant/app/app.js:23:5)
0|app      |     at Module._compile (node:internal/modules/cjs/loader:1554:14)
0|app      |     at Object..js (node:internal/modules/cjs/loader:1706:10)
0|app      |     at Module.load (node:internal/modules/cjs/loader:1289:32)
0|app      |     at Function._load (node:internal/modules/cjs/loader:1108:12)
0|app      |     at Object.<anonymous> (/home/vagrant/.nvm/versions/node/v22.14.0/lib/node_modules/pm2/lib/ProcessContainerFork.js:33:23) {
0|app      |   code: 'EADDRINUSE',
0|app      |   errno: -98,
0|app      |   syscall: 'listen',
0|app      |   address: '::',
0|app      |   port: 3000
0|app      | }

/home/vagrant/.pm2/logs/node-cluster-app-error-5.log last 15 lines:
/home/vagrant/.pm2/logs/node-cluster-app-error-6.log last 15 lines:
/home/vagrant/.pm2/logs/node-cluster-app-out-5.log last 15 lines:
5|node-clu | App listening on port 3000

/home/vagrant/.pm2/logs/node-cluster-app-out-6.log last 15 lines:
6|node-clu | App listening on port 3000
```

Muestra los registros de la aplicación.

3. pm2 monit



Proporciona una vista en tiempo real de uso de CPU y memoria.

5. Cuestiones

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

La agrupación en clústeres mejora el rendimiento en **cargas altas** y solicitudes concurrentes, pero en algunos casos puede generar sobrecarga adicional. Si la carga es baja o moderada, una única instancia de Node.js puede ser **más eficiente** que un clúster con múltiples procesos.