# Reporte del desafío nº15 del curso de Backend

Alejandro Portaluppi

Debes asegurarte de estar posicionado en la carpeta servidorConBalanceDeCarga.

Considera ejecutar “taskkill /F /IM node.exe” entre un ejercicio y otro para eliminar todos los procesos de node creados anteriormente, pero antes asegúrate de no estar comprometiendo un proyecto aparte.

Algunos comandos que podrían serte de utilidad:

- **task:**

  tasklist /fi "imagename eq node.exe" | Trae una lista con información de todas las instancias que se generan en node.exe

  taskkill /pid X -f | Elimina el proceso con id X

  taskkill /F /IM node.exe | Elimina todos los procesos de NodeJs

- **pm2:**

  pm2 list | Para ver la lista de aplicativos de pm2 que se están ejecutando

  pm2 start ecosystem.config.cjs | Para iniciar pm2 utilizando el ecosystem

  pm2 monit | Para monitorear los aplicativos

  pm2 delete all | Para eliminar las instancias de pm2

  pm2 delete X | Para eliminar el proceso con pid=X

  pm2 flush | Elimina los logs

  pm2 restart all | Reinicia todos las instancias

- **nginx:**

  nginx | Inicializar nginx

  nginx -s reload | Toma un servidor de nginx que esté escuchando y lo reinicia

1) a) Ejecuta “node src/app.js --mode CLUSTER” en la terminal para ejecutar el servidor en “modo cluster”.

   Este modo creará un cluster de servidores escuchando en el puerto 8080. El cluster será creado desde node utilizando el módulo nativo “cluster”.

   La cantidad de servidores depende de cuántos soporte la computadora que trata de abrirlos.

b) Ejecuta “node src/app.js --mode FORK” en la terminal para ejecutar el servidor en “modo fork”. Este modo es el que viene por defecto al ejecutar “node src/app.js”

El propósito de este modo es verificar que nodemon crea un proceso adicional, ya que forkea a nuestro servidor.

Luego de ejecutar el comando mencionado, ejecuta el comando ‘tasklist /fi "imagename eq node.exe"’ en otra terminal, para mostrar la lista de procesos de Node.js que se están ejecutando en el sistema, en mi caso la lista es la siguiente:

```
PS C:\nginx-1.23.3\curso-backend\servidorConBalanceDeCarga> tasklist /fi "imagename eq node.exe"

Nombre de imagen              PID Nombre de sesión Núm. de ses Uso de memor
========================= ======== ================ =========== ============
node.exe                    10756 Console                    1    30.484 KB
PS C:\nginx-1.23.3\curso-backend\servidorConBalanceDeCarga>
```

Observo que actualmente hay un proceso ejecutándose.

Acto seguido debes detener el proceso con CTRL+C en la terminal original y volver a solicitar la lista para verificar que el proceso ya no está:

```
PS C:\nginx-1.23.3\curso-backend\servidorConBalanceDeCarga> tasklist /fi "imagename eq node.exe"
INFORMACIÓN: no hay tareas ejecutándose que coincidan con los
criterios especificados.
PS C:\nginx-1.23.3\curso-backend\servidorConBalanceDeCarga>
```

b) Ejecuta “nodemon src/app.js --mode FORK” en la terminal para ejecutar el servidor utilizando nodemon y solicita la lista tal como se mencionó anteriormente. Actualmente a mí me muestra la siguiente lista:

```
PS C:\nginx-1.23.3\curso-backend\servidorConBalanceDeCarga> tasklist /fi "imagename eq node.exe"

Nombre de imagen              PID Nombre de sesión Núm. de ses Uso de memor
========================= ======== ================ =========== ============
node.exe                    10816 Console                    1    33.628 KB
node.exe                     6120 Console                    1    36.024 KB
PS C:\nginx-1.23.3\curso-backend\servidorConBalanceDeCarga>
```

Verificamos que nodemon crea un proceso adicional.

2) Ahora vamos a trabajar con el módulo pm2. Para lograr eso se instaló el módulo globalmente (npm i pm2 -g) y se creó un archivo “ecosystem.config.cjs” para trabajar más cómodamente con él.

Nuestro modulo tiene seteada la opción "type": "module" en el package.json para indicarle a NodeJs que el proyecto está escrito en sintaxis de módulo ECMAScript, pero el inconveniente con pm2 es que sus configuraciones no soportan esta opción, por lo tanto debemos modificar levemente el código que viene por defecto (observar que ecosystem.config.cjs tiene la extensión cjs por esta razón).

Lo que se debe hacer es ir al archivo ProcessContainer.js ubicado en pm2/lib. La carpeta pm2 puede estar en distintas ubicaciones según dónde esté ubicado tu npm, en mi caso está en C: / Users / [usuario] / AppData / Roaming / npm / node_modules/

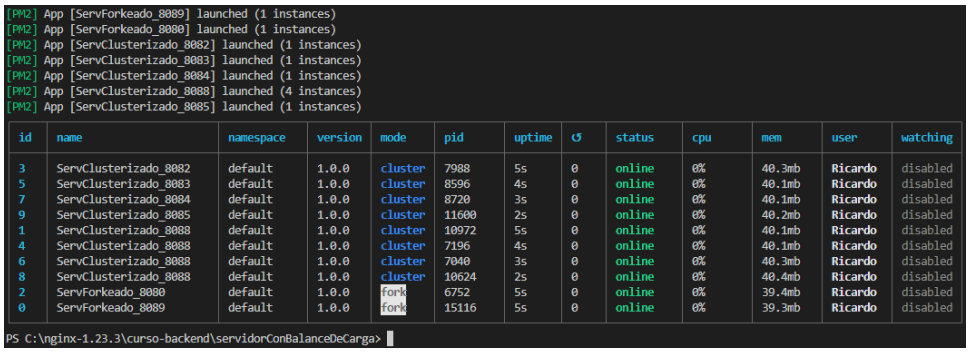
Una vez que llegaste al archivo reemplaza una línea de código:

```
300     if (ProcessUtils.isESModule(script) === true)
301       // import(process.env.pm_exec_path); //*Código original
302       import(require("url").pathToFileURL(process.env.pm_exec_path)); //*Código modificado
303     else
304       require('module')._load(script, null, true);
```

Una vez hecho esto ya podemos usar pm2 en nuestro proyecto.

a) Queremos ejecutar el servidor utilizando pm2 en sus modos fork y cluster (observar que son distintos a los fork y closter de “1)”, ya que serán creados por pm2), y luego en ambos casos mostrar una lista de los aplicativos iniciados. Para esto, ejecuta “pm2 start ecosystem.config.cjs” en la terminal. Previamente desearas eliminar todos los procesos gestionados por pm2 en este proyecto en caso de que no sea tu primera vez ejecutando este código. Si no tienes problema con eliminar todos los procesos gestionados por pm2 en la computadora puedes ejecutar “pm2 delete all” para mayor rapidez.

Te debe aparecer en la terminal una lista similar a esta:




Por ahora sólo nos vamos a concentrar en el “ServForkeado_8089” y “ServClusterizado_8088”, de los demás hablaremos en el ítem 3b).

Observar que ambos fueron creados gracias a los dos primeros objetos del archivo ecosystem.config.cjs. Uno con el modo de ejecución fork (por defecto) y otro con cluster. Observar que “ServClusterizado_8088” se inició con cuatro instancias, las máximas permitidas por mi computadora.

Tambien podemos listar los procesos gracias al comando ya mencionado ‘tasklist /fi "imagename eq node.exe"’:

```
PS C:\nginx-1.23.3\curso-backend\servidorConBalanceDeCarga> tasklist /fi "imagename eq node.exe"

Nombre de imagen              PID Nombre de sesión Núm. de ses Uso de memor
========================= ======== ================ =========== ============
node.exe                     3560 Console                    2    35.396 KB
node.exe                    15116 Console                    2    34.872 KB
node.exe                    10972 Console                    2    35.464 KB
node.exe                     6752 Console                    2    34.888 KB
node.exe                     7988 Console                    2    35.556 KB
node.exe                     7196 Console                    2    35.416 KB
node.exe                     8596 Console                    2    35.220 KB
node.exe                     7040 Console                    2    35.460 KB
node.exe                     8720 Console                    2    35.344 KB
node.exe                    10624 Console                    2    36.064 KB
node.exe                    11600 Console                    2    35.388 KB
PS C:\nginx-1.23.3\curso-backend\servidorConBalanceDeCarga>
```

Observar que esta lista tiene una fila más gracias a un proceso extra que crea node (aunque posiblemente puedas ver más procesos extra si tu maquina tiene otros programas node ejecutándose)

b) A continuación vamos a hacer pruebas de finalización. La idea es eliminar los procesos creados para corroborar que se levantan nuevamente gracias a la configuración interna de pm2.

Con el comando “taskkill /pid 7040 /f” eliminamos el proceso con pid 7040, y gracias al comando “pm2 list” vemos la lista de procesos de pm2, corroborando que el proceso eliminado fue reemplazado por uno nuevo con pid 5272:

```
PS C:\nginx-1.23.3\curso-backend\servidorConBalanceDeCarga> taskkill /pid 7040 /f
Correcto: se terminó el proceso con PID 7040.
PS C:\nginx-1.23.3\curso-backend\servidorConBalanceDeCarga> pm2 list
```

| id | name | namespace | version | mode | pid | uptime | ↺ | status | cpu | mem | user | watching |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 3 | ServClusterizado_8082 | default | 1.0.0 | cluster | 7988 | 83s | 0 | online | 0% | 35.0mb | Ricardo | disabled |
| 5 | ServClusterizado_8083 | default | 1.0.0 | cluster | 8596 | 82s | 0 | online | 0% | 34.9mb | Ricardo | disabled |
| 7 | ServClusterizado_8084 | default | 1.0.0 | cluster | 8720 | 81s | 0 | online | 0% | 34.8mb | Ricardo | disabled |
| 9 | ServClusterizado_8085 | default | 1.0.0 | cluster | 11600 | 80s | 0 | online | 0% | 34.8mb | Ricardo | disabled |
| 1 | ServClusterizado_8088 | default | 1.0.0 | cluster | 10972 | 83s | 0 | online | 0% | 34.8mb | Ricardo | disabled |
| 4 | ServClusterizado_8088 | default | 1.0.0 | cluster | 7196 | 82s | 0 | online | 0% | 34.8mb | Ricardo | disabled |
| 6 | ServClusterizado_8088 | default | 1.0.0 | cluster | 5272 | 6s | 1 | online | 0% | 40.3mb | Ricardo | disabled |
| 8 | ServClusterizado_8088 | default | 1.0.0 | cluster | 10624 | 80s | 0 | online | 0% | 35.5mb | Ricardo | disabled |
| 2 | ServForkeado_8080 | default | 1.0.0 | fork | 6752 | 83s | 0 | online | 0% | 34.4mb | Ricardo | disabled |
| 0 | ServForkeado_8089 | default | 1.0.0 | fork | 15116 | 83s | 0 | online | 0% | 34.4mb | Ricardo | disabled |

```
PS C:\nginx-1.23.3\curso-backend\servidorConBalanceDeCarga>
```

Un razonamiento análogo podemos hacer para el servidor en modo fork:

```
PS C:\nginx-1.23.3\curso-backend\servidorConBalanceDeCarga> taskkill /pid 15116 /f
Correcto: se terminó el proceso con PID 15116.
PS C:\nginx-1.23.3\curso-backend\servidorConBalanceDeCarga> pm2 list
```

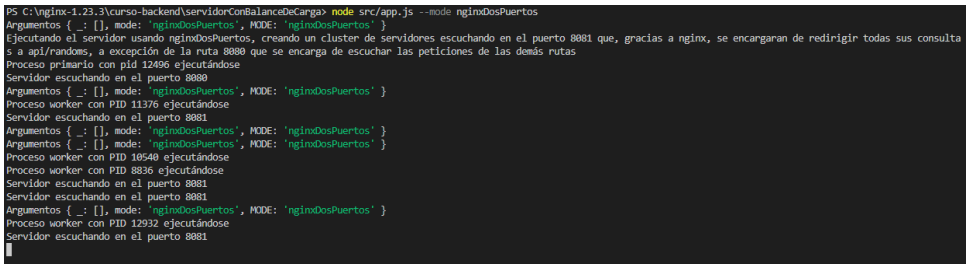
| id | name | namespace | version | mode | pid | uptime | ↺ | status | cpu | mem | user | watching |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 3 | ServClusterizado_8082 | default | 1.0.0 | cluster | 7988 | 4m | 0 | online | 0% | 35.5mb | Ricardo | disabled |
| 5 | ServClusterizado_8083 | default | 1.0.0 | cluster | 8596 | 4m | 0 | online | 0% | 34.9mb | Ricardo | disabled |
| 7 | ServClusterizado_8084 | default | 1.0.0 | cluster | 8720 | 4m | 0 | online | 0% | 34.9mb | Ricardo | disabled |
| 9 | ServClusterizado_8085 | default | 1.0.0 | cluster | 11600 | 4m | 0 | online | 0% | 35.4mb | Ricardo | disabled |
| 1 | ServClusterizado_8088 | default | 1.0.0 | cluster | 10972 | 4m | 0 | online | 0% | 35.4mb | Ricardo | disabled |
| 4 | ServClusterizado_8088 | default | 1.0.0 | cluster | 7196 | 4m | 0 | online | 0% | 34.8mb | Ricardo | disabled |
| 6 | ServClusterizado_8088 | default | 1.0.0 | cluster | 5272 | 3m | 1 | online | 0% | 34.8mb | Ricardo | disabled |
| 8 | ServClusterizado_8088 | default | 1.0.0 | cluster | 10624 | 4m | 0 | online | 0% | 35.8mb | Ricardo | disabled |
| 2 | ServForkeado_8080 | default | 1.0.0 | fork | 6752 | 4m | 0 | online | 0% | 34.8mb | Ricardo | disabled |
| 0 | ServForkeado_8089 | default | 1.0.0 | fork | 8140 | 3s | 1 | online | 0% | 39.8mb | Ricardo | disabled |

```
PS C:\nginx-1.23.3\curso-backend\servidorConBalanceDeCarga>
```

3) Finalmente vamos a utilizar y configurar nginx para balancear las cargas de nuestro servidor de distintas maneras. Antes de eso debemos llevar nuestra carpeta de trabajo (“servidorConBalanceDeCarga” en mi caso) dentro de la carpeta donde tienes instalado el nginx.

a) Ejecuta “node src/app.js --mode nginxDosPuertos” en la terminal para crear un cluster de servidores escuchando en el puerto 8081, con la excepción de uno que escuchara en el 8080 (asegúrate de haber terminado los procesos del ejercicio anterior para no tener problemas en el puerto 8080). Este cluster de servidores se crea con el módulo nativo cluster.

Dejo una captura de pantalla para observar los logs, ya que nos servirán más adelante:

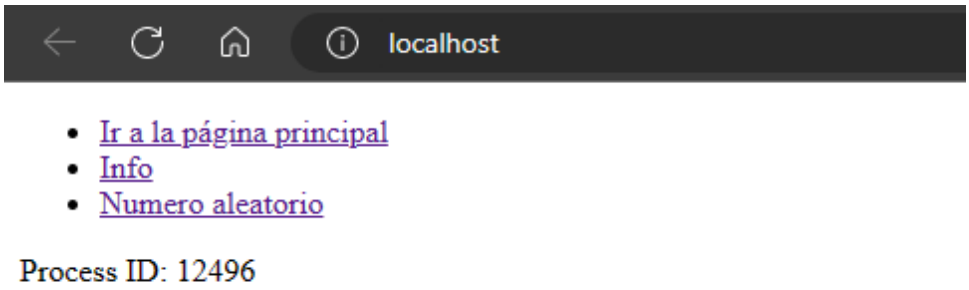



Nuestro objetivo actual es utilizar nginx para redirigir todos los servidores con puerto 8081 a la ruta api/randoms, ya que en esa ruta se realiza un cálculo que requiere mucho esfuerzo computacional y por lo tanto requiere un tiempo de ejecución relativamente elevado. Por otro lado, el puerto 8080 se encargará de escuchar las peticiones de todas las demás rutas.

Para que esta configuración sea posible primero debemos modificar el archivo “nginx.conf”. Lo debes encontrar en la carpeta “conf”, que a su vez se ubica en la carpeta donde está alojado tu nginx (yo la tengo nombrada como “nginx-1.23.3”). Te dejo una copia de este archivo ya modificado en la carpeta de este proyecto para que puedas mirarlo. Si deseas entenderlo mejor puedes ver los comentarios que deje ahí.

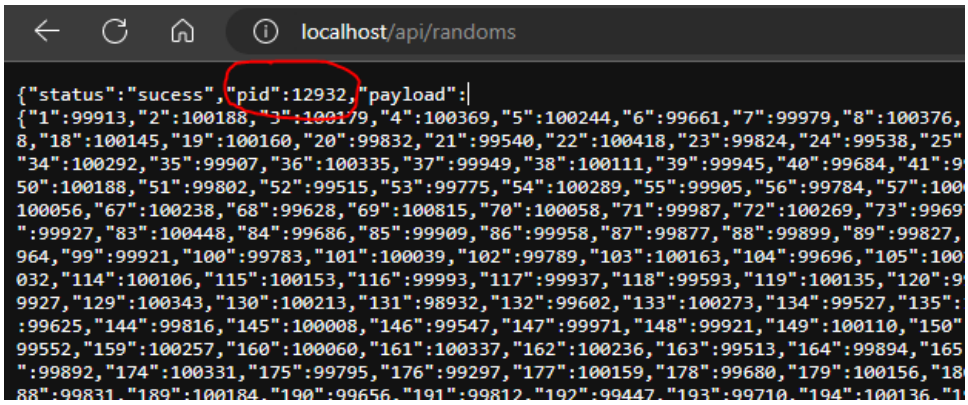
Abre la terminal de windows, dirígete a la carpeta de nginx y escribe “nginx” para inicializarlo. Lo que hará el programa será leer la configuración ya mencionada para redirigir la ruta “api/randoms” al puerto 8081 y todas las demás al 8080, tal como habíamos planeado.

Finalmente debemos corroborar que todo funciona correctamente. Para eso nos dirigimos a la url “localhost” y corroboramos que el processID recibido coincide con el que se asoció al puerto 8080 según los mensajes en la consola (ver imagen anterior), en mi caso es 12496:




Lo mismo podríamos hacer en la ruta /info (te lo dejo a ti). Si no te funcionó es posible que hayas iniciado nginx previamente y no se haya cerrado. Puedes “finalizar la tarea” desde el administrador de tareas y volver a intentarlo.

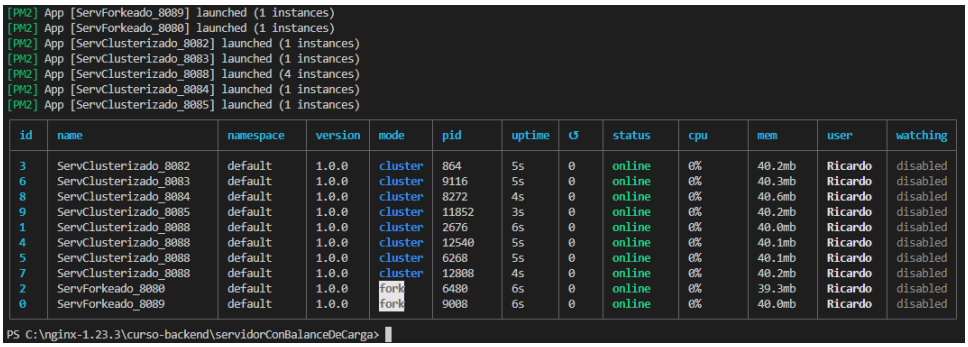
Ahora vamos a la ruta “api/randoms” y verificamos que el pid ya no corresponde al asociado al 8080, sino alguno de los asociados al 8081, en mi caso es el 12932:




b) Ahora queremos modificar la configuración para que todas las consultas a /api/randoms sean redirigidas a cuatro clusters de una instancia cada una, escuchando en los puertos 8082, 8083, 8084 y 8085 respectivamente, donde nginx se encargará de organizar estos puertos y hacer un balance de peso entre ellos.

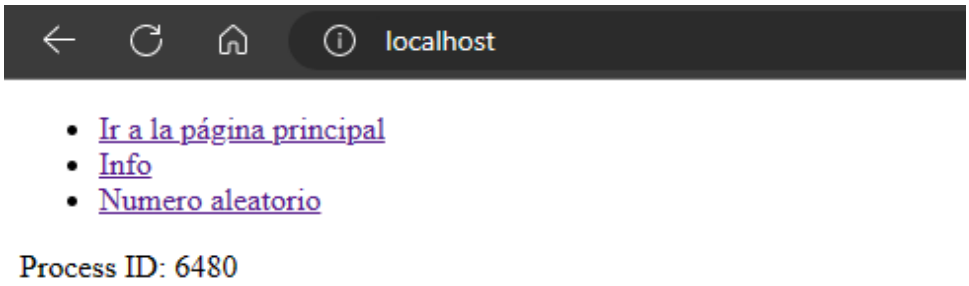
Al igual que en el ítem a) esto ya lo dejé configurado. Puedes chequear los comentarios de “nginx.conf” si necesitas más información. Sin embargo, a diferencia de antes, en este ítem uso pm2.

Ejecuta “pm2 start ecosystem.config.cjs” en la terminal para levantar los puertos deseados. En mi caso la terminal dice lo siguiente:




Recordemos que en el ejercicio 2a) nos concentramos en “ServForkeado_8089” y “ServClusterizado_8088”, ahora debemos concentrarnos en los demás ya que son los servidores que corresponden a este ejercicio. Para mayor claridad, en el nombre coloco el puerto asignado.

Para corroborar que funciona debes ir a la ruta base y a /info tal como antes y verificar que el process id coincide con el asignado al puerto 8080, que en mi caso (según la tabla de la terminal) es 6480. Dejo una captura de lo que se ve en mi ruta base:




Ahora vamos a la ruta api/randoms y verificamos que el process id se corresponde con alguno de los puertos asignados entre el 8082 al 8085, tal como esperábamos.

La novedad con respecto al ítem a) es observar el balanceo configurado: recarga la página api/randoms y deberás ver que el process id cambia, indicándonos que cada vez que solicitamos entrar a dicha ruta los puertos indicados cambian para balancear el esfuerzo entre ellos. Dejo tres capturas de pantalla mostrando este hecho: