



Clase 29. Programación Backend

Clusters y Escalabilidad



OBJETIVOS DE LA CLASE

- Conocer e implementar el módulo *cluster*.
- Aprender acerca del módulo *Forever* y su diferencia con *Nodemon*.
- Conocer e implementar el módulo *PM2* y su ventaja sobre *Forever*.

CRONOGRAMA DEL CURSO

Clase 28



Global & Child process

Clase 29



Clusters y Escalabilidad

Clase 30



PROXY & NGINX

CLUSTER EN NODEJS



¿Qué es Cluster?

- Cuando hablamos de Cluster nos referimos al **uso de subprocesos que permite aprovechar la capacidad del procesador del servidor donde se ejecute la aplicación.**
- Como vimos la clase pasada, Node se ejecuta en un solo proceso (single thread), y entonces no aprovechamos la máxima capacidad que nos puede brindar un procesador multicore (múltiples núcleos).
- Al usar el cluster, lo que hacemos es, en el caso de estar ejecutando sobre un servidor multicore, hacer uso de todos los núcleos del mismo, aprovechando al máximo su capacidad.



¿Cómo funciona?

- Node nos provee un **módulo llamado cluster** para hacer uso de esto. El mismo, permite crear fácilmente procesos hijo.
- Lo que hace es clonar el worker maestro y delegarle la carga de trabajo a cada uno de ellos, de esa manera se evita la sobrecarga a un solo núcleo del procesador.
- Con un método similar al que vimos de *Fork*, se crea una copia del proceso actual. En ese momento, el primer proceso se convierte en maestro o *master*, y la copia en un trabajador o *worker*.

MÓDULO CLUSTER

Usar el módulo *Cluster*



- Primero requerimos el módulo *cluster* y el *http* para crear el servidor.
- En la constante `numCPUs` lo que hacemos es crear tantos workers como CPUs tengamos en el sistema.

```
const cluster = require('cluster');  
const http = require('http');  
const numCPUs = require('os').cpus().length;
```


Usar el módulo Cluster



```
if (cluster.isMaster) {  
  console.log(`Master ${process.pid} is running`);  
  
  // Fork workers.  
  for (let i = 0; i < numCPUs; i++) {  
    cluster.fork();  
  }  
  
  cluster.on('exit', (worker, code, signal) => {  
    console.log(`worker ${worker.process.pid} died`);  
  });  
} else {  
  // Workers can share any TCP connection  
  // In this case it is an HTTP server  
  http.createServer((req, res) => {  
    res.writeHead(200);  
    res.end('hello world\n');  
  }).listen(8000);  
  
  console.log(`Worker ${process.pid} started`);  
}
```

- Es habitual hacer que el proceso master se dedique únicamente a gestionar a los workers, y que sean los workers los que hagan el “trabajo sucio”.
- Entonces, si entra al *if* crea *workers*, y si va al *e/se* abre el servidor. Como vemos en el código.

Usar el módulo *Cluster*



```
if (cluster.isMaster) {
  console.log(`Master ${process.pid} is running`);

  // Fork workers.
  for (let i = 0; i < numCPUs; i++) {
    cluster.fork();
  }

  cluster.on('exit', (worker, code, signal) => {
    console.log(`worker ${worker.process.pid} died`);
  });
} else {
  // Workers can share any TCP connection
  // In this case it is an HTTP server
  http.createServer((req, res) => {
    res.writeHead(200);
    res.end('hello world\n');
  }).listen(8000);

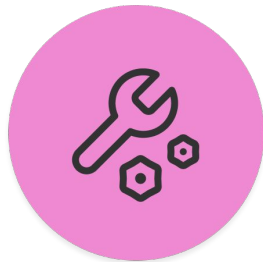
  console.log(`Worker ${process.pid} started`);
}
```

- Dentro del *for*, en el proceso *master*, creamos un *worker* por cada CPU.
- Con `cluster.on` y el comando “*exit*” controlamos la salida de estos workers.
- Como mencionamos antes, en los *workers*, es decir, cuando `cluster.isMaster` es falso, creamos un servidor HTTP.
- Recientemente, se ha migrado el uso de `isMaster` a `isPrimary` para evitar la alusión a la esclavitud.

Algunos comandos útiles



- Si usamos Powershell:
 - `tasklist /fi "imagename eq node.exe"`
 - lista todos los procesos de node.js activos
 - `taskkill /pid <PID> /f`
 - mata un proceso por su número de PID
- Si usamos Bash:
 - `fuser <PORT>/tcp [-k]`
 - encuentra [y mata] al proceso ocupando el puerto PORT



SERVIDOR NODE CON FORK

Tiempo: 5 minutos

Servidor Node con Nodemon + fork

Desafío
generico



Tiempo: 5 minutos

Realizar un servidor Node.js basado en express al que se le pase como parámetro el número de puerto de escucha. De no recibir este dato, el servidor iniciará en el puerto 8080.

Al ponerse en línea el servidor representará por consola el puerto de conexión y su número de proceso (pid).

En el endpoint raíz '/' deberá devolver un mensaje con el siguiente formato:
'Servidor express en (PORT) - PID (pid) - (fecha y hora actual){'



Poner en marcha el servidor con node (sin nodemon) y verificar en el sistema operativo el proceso de node y su pid. Hacerlo con nodemon y ver la diferencia (constatar que nodemon crea un proceso padre forkeando a nuestro server). En ambos casos el puerto de escucha será el 8081.

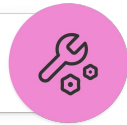


SERVIDOR NODE CON CLUSTER

Tiempo: 10 minutos

Servidor Node con Cluster

Desafío
generico



Tiempo: 10 minutos

Tomando como base el ejercicio anterior, agregar la lógica que permita construir un cluster de servidores, poniendo un evento de escucha en cada servidor para reiniciarlo si se cae.

- Representar por consola el número de procesadores detectados, el momento en el que un servidor se cae, el pid de los procesos worker y el del master.
- Poner en marcha el servidor con node y nodemon en el puerto 8081, verificando en cada caso, la respuesta en su ruta raíz y el número de procesos activos de node en el sistema operativo, relacionándolos con el número de procesadores.
- Finalizar en el sistema operativo un proceso worker comprobando que se reinicia en un nuevo pid.
- Como último identificar el pid del master y finalizar su proceso de node, analizando qué ocurre en el caso de haberlo ejecutado con node y con nodemon.

MÓDULO FOREVER



¿Qué es?

- Cuando ejecutamos un proyecto de Node en un servidor en el que lo tengamos desplegado, dejamos la consola “ocupada” con esa aplicación. Si queremos seguir haciendo cosas o arrancar otro proyecto no podemos, ya que tendríamos que detener la aplicación pulsando *Ctrl+C* quedando la consola libre nuevamente.
- Por otro lado, si el servidor se parara por un fallo, nuestra aplicación no se arrancaría de nuevo.
- Ambos problemas se pueden resolver con el *módulo* **Forever** de Node.



Comparación con Nodemon

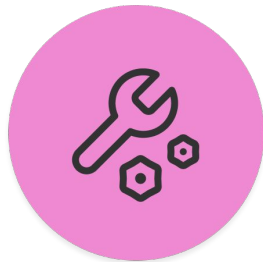
- Como ya vimos, cada vez que hacemos cambios en alguno de los archivos del programa, debemos parar e iniciar nuevamente el servidor.
- El **módulo Nodemon** de Node, evita esto y se reinicia de forma automática ante cualquier cambio en los archivos del programa en ejecución.
- Sin embargo, *Nodemon* solo nos sirve en desarrollo. Cuando estamos en producción, no se puede hacer uso de este módulo
- Esta es la **ventaja de Forever**, ya que este puede utilizarse en producción. Además, nos sirve también para reiniciar el servidor ante un fallo del mismo.



Usando 'forever' por línea de comando

- `forever start <filename> [args]`: inicia un nuevo proceso
- `forever list`: lista todos los procesos activos
- `forever stop <PID>`: detiene un proceso según su id de proceso
- `forever stopall`: detiene todos los procesos activos
- `forever --help`: muestra la ayuda completa

Para esto es recomendable haber instalado el módulo en forma global: `npm i -g forever`

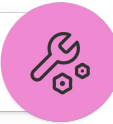


SERVIDOR NODE CON FOREVER

Tiempo: 10 minutos

Servidor Node con Forever

Desafío
generico



Tiempo: 10 minutos

- Poner en marcha tres servidores (con el formato del primer ejercicio: sin cluster) utilizando Forever. Dichos procesos escucharán en los puertos 8081, 8082 y 8083 respectivamente.
- Generar un request a sus rutas raíz comprobando que respondan adecuadamente.
- Verificar en el sistema operativo la cantidad de procesos levantados de node y analizar el porqué.
- Finalizar por sistema operativo el proceso de cada uno de estos servidores, comprobando que Forever los ponga en marcha nuevamente (tendrían que iniciar con un nuevo pid).
- Listar todos los servidores activos y comprobar la finalización de un proceso y de todos por parte de Forever, comprobando en el sistema operativo los procesos de node levantados.



BREAK

¡5/10 MINUTOS Y VOLVEMOS!

MÓDULO PM2



¿Qué es?

- Es un gestor de procesos (Process Manager), es decir, un programa que controla la ejecución de otro proceso.
- Permite chequear si el proceso se está ejecutando, reiniciar el servidor si este se detiene por alguna razón, gestionar los *logs*, *etc.*
- Lo más importante es que **PM2** simplifica las aplicaciones de Node para ejecutarlas como **cluster**.
- Es decir, que podemos escribir nuestra aplicación sin pensar en el *cluster*, y luego *PM2* se encarga de crear un número dado de *worker processes* para ejecutar la aplicación.



- Es capaz de aguantar cantidades enormes de tráfico con un consumo de recursos realmente reducido y con herramientas que permiten realizar la monitorización de las aplicaciones de manera remota.
- La ventaja principal sobre el módulo *forever* es el tema del *cluster* embebido en este módulo, como mencionamos antes.



Empezar a usarlo

1. Para empezar a utilizarlo, primero debemos instalarlo

```
$ npm install pm2 -g
```

2. Luego, podemos iniciar la ejecución de forma genérica con el comando

```
$ pm2 start app.js
```

3. Con el mismo, la aplicación queda monitoreada y en ejecución siempre.



Empezar a usarlo

- Se puede iniciar la ejecución en modo fork o en modo cluster. Los comandos que utilizamos son:

```
----- MODO FORK -----  
pm2 start app.js --name="ServerX" --watch -- PORT  
pm2 start app.js --name="Server1" --watch -- 8081  
pm2 start app.js --name="Server2" --watch -- 8082  
  
----- MODO CLUSTER -----  
pm2 start app.js --name="ServerX" --watch -i max -- PORT  
pm2 start app.js --name="Server3" --watch -i max -- 8083
```



Usar PM2

- Podemos listar todas las aplicaciones que se están ejecutando con

```
$ pm2 list
```

- El listado puede resultar muy similar a este

```
unitech@t450: ~  
>>> pm2 ls
```

App name	id	mode	pid	status	restart	uptime	cpu	mem	watching
api	0	cluster	27287	online	0	2m	0%	35.5 MB	disabled
api	5	cluster	27308	online	0	2m	0%	32.4 MB	disabled
api	6	cluster	27348	online	0	2m	0%	33.7 MB	disabled
front	3	fork	27303	online	0	2m	0%	26.5 MB	enabled
healthcheck	2	fork	0	stopped	0	0	0%	0 B	disabled
mailer	4	fork	27309	online	0	2m	0%	26.1 MB	disabled
worker	1	fork	27292	online	0	2m	0%	24.9 MB	disabled

Use 'pm2 show <id/name>' to get more details about an app



Usar PM2

- Para detener, reiniciar o eliminar una de las aplicaciones de la lista, podemos ejecutar alguno de los siguientes comandos

```
$ pm2 stop      <app_name|namespace|id|'all'|json_conf>  
$ pm2 restart   <app_name|namespace|id|'all'|json_conf>  
$ pm2 delete    <app_name|namespace|id|'all'|json_conf>
```

- Para obtener detalle de una aplicación:

```
$ pm2 describe <id|app_name>
```



Usar PM2

- Para monitorear sus logs, métricas e información:

```
$ pm2 monit
```

- Para consultar logs:

```
$ pm2 logs
```

- Para hacer flush de logs:

```
$ pm2 flush
```

- Para ver las opciones de comandos disponibles:

```
$ pm2 --help
```



SERVIDOR NODE CON PM2

Tiempo: 15 minutos

Servidor Node con PM2

Tiempo: 15 minutos

Desafío
generico



Poner en marcha dos servidores (con el formato del primer ejercicio: sin cluster) utilizando PM2.

Uno de los servidores escuchará en el puerto 8081 y se ejecutará en modo 'fork'. El otro lo hará en el puerto 8082 y se ejecutará en modo 'cluster'.

- Generar un request a cada uno de ellos comprobando que respondan adecuadamente.
- Verificar en el sistema operativo la cantidad de procesos levantados y analizar el porqué.
- Finalizar por sistema operativo el proceso de cada uno de estos servidores (fork y cluster), comprobando que PM2 los ponga en marcha nuevamente (tendrían que iniciar con un nuevo pid).
- Con PM2 listar todos los servidores activos y e ir finalizando los procesos (por id y por name), verificando en el sistema operativo, para cada operación, los procesos activos de node.

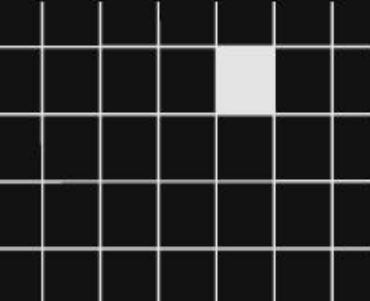
¿PREGUNTAS?





¡MUCHAS GRACIAS!

Resumen de lo visto en clase hoy:

- Módulo Cluster.
 - Forever.
 - PM2.
- 



OPINA Y VALORA ESTA CLASE

#DEMOCRATIZANDO LA EDUCACIÓN

CODER HOUSE