



**Clase 30.** Programación Backend

# ***PROXY & NGINX***



## ***OBJETIVOS DE LA CLASE***

- Conocer el concepto de proxy y comprender las diferencias entre proxy directo e inverso.
- Conocer Nginx con sus configuraciones y usos.

# ***CRONOGRAMA DEL CURSO***

Clase 29



**Clusters y Escalabilidad**

Clase 30



**PROXY & NGINX**

Clase 31



**Logs, profiling & debug**

***REPASANDO...***

***CODER HOUSE***

- Cuando hablamos de Cluster nos referimos al **uso de subprocesos que permite aprovechar la capacidad del procesador del servidor donde se ejecute la aplicación.**
- 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.

***CLUSTER***

Una herramienta **CLI** simple para **garantizar** que un script determinado se ejecute de forma continua (es decir, para siempre).

***FOREVER***

- 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**.

***PM2***



***¿Alguna pregunta hasta  
ahora?***



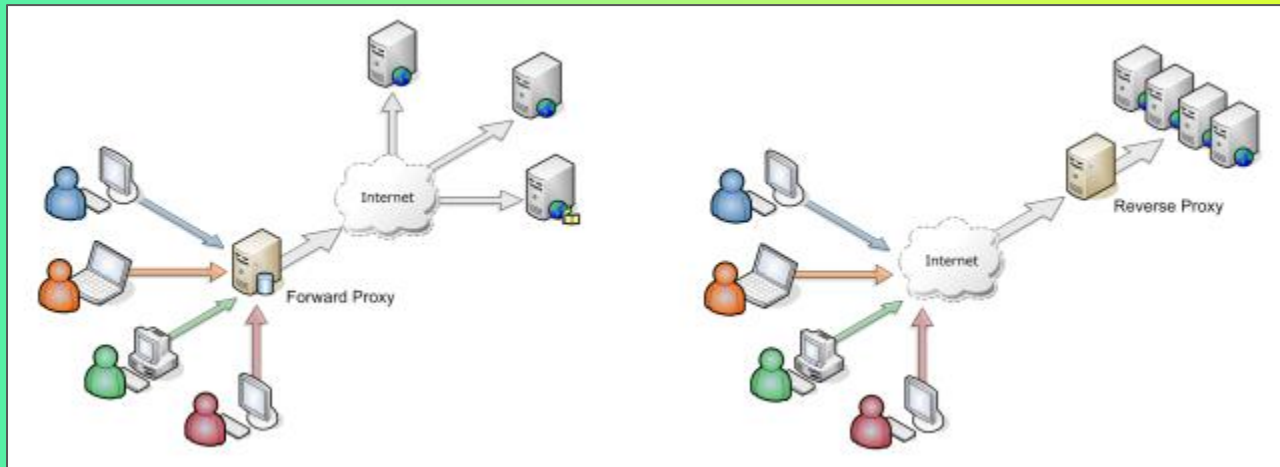
# ***SERVIDOR PROXY***

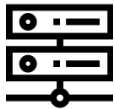


# ***¿Qué es?***

- Es un servidor que hace de intermediario entre las conexiones de un cliente y un servidor de destino, filtrando todos los paquetes entre ambos.
- Sin el proxy, la conexión entre cliente y servidor de origen a través de la web es directa.
- Se utiliza para navegar por internet de forma más anónima ya que oculta las IP, sea del cliente o del servidor de origen.
- Por ser intermediario, ofrece funcionalidades como control de acceso, registro del tráfico, mejora de rendimiento, entre otras.

# ***FORWARD PROXY vs. REVERSE PROXY***





# ***Proxy directo (forward)***

- Es un servidor proxy que se coloca entre el cliente y la web.
- Recibe la petición del cliente para acceder a un sitio web, y la transmite al servidor del sitio, para que este no se entere de qué cliente está haciendo la petición.
- Lo utiliza un cliente cuando quiere anonimizar su IP.
- Es útil para mejorar la privacidad, y para evitar restricciones de contenido geográfico (contenido bloqueado en cierta región).



# ***Proxy inverso (reverse)***

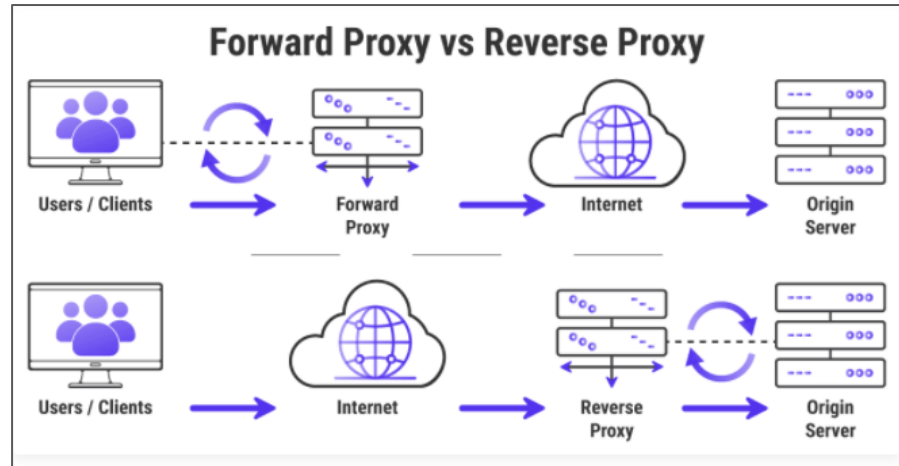


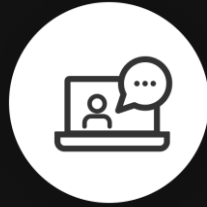
- En este caso, el servidor proxy se coloca entre la web y el servidor de origen.
- Entonces, el que se mantiene en el anonimato es el servidor de origen. Garantiza que ningún cliente se conecte directo con él y por ende mejore su seguridad.
- En general el cifrado SSL de un sitio web seguro se crea en el proxy (y no directamente en el servidor).
- Además, es útil para distribuir la carga entre varios servidores web.

# ***Similitudes y diferencias***



- Ambos pueden trabajar juntos, ya que no se superponen en su funcionamiento.
- Los clientes/usuarios pueden utilizar un proxy directo y los servidores de origen un proxy inverso.





***¿Alguna pregunta hasta  
ahora?***

# ***REVERSE PROXY EN BACKEND***



# *Utilizar proxy inverso en backend*



Existen varios beneficios por lo que, al crear un sitio web, conviene utilizar un proxy inverso:

- **Balancear la carga:** Un solo servidor de origen, en una página con millones de visitantes diarios, no puede manejar todo el tráfico entrante. El proxy inverso puede recibir el tráfico entrante antes de que llegue al servidor de origen. Si este está sobrecargado o cae completamente, puede distribuir el tráfico a otros servidores sin afectar la funcionalidad del sitio. Es el principal uso de los servidores proxy inverso.

# *Utilizar proxy inverso en backend*



- **Seguridad mejorada:** Al ocultar el proxy inverso la IP del servidor de origen de un sitio web, se puede mantener el anonimato del mismo, aumentando considerablemente su seguridad. Al tener al proxy como intermediario, cualquier atacante que llegue va a tener una traba más para llegar al servidor de origen.
- **Potente caching:** Podemos utilizar un proxy inverso para propósitos de aceleración de la web, almacenando en caché tanto el contenido estático como el dinámico. Esto puede reducir la carga en el servidor de origen, resultando en un sitio web más rápido.

# *Utilizar proxy inverso en backend*



- **Compresión superior:** Un proxy inverso es ideal para comprimir las respuestas del servidor. Esto se utiliza mucho ya que las respuestas del servidor ocupan mucho ancho de banda, por lo que es una buena práctica comprimirlas antes de enviarlas al cliente.
- **Cifrado SSL optimizado:** Cifrar y descifrar las solicitudes SSL/TLS para cada cliente puede ser muy difícil para el servidor de origen. Un proxy inverso puede hacer esta tarea para liberar los recursos del servidor de origen para otras tareas importantes, como servir contenido.

# *Utilizar proxy inverso en backend*



- **Monitoreo y registro del tráfico:** Un proxy inverso captura cualquier petición que pase por él. Por lo tanto, podemos usarlos como un centro de control para monitorear y registrar el tráfico. Incluso si utilizamos varios servidores web para alojar todos los componentes de nuestro sitio web, el uso de un proxy inverso facilitará la supervisión de todos los datos entrantes y salientes del sitio.



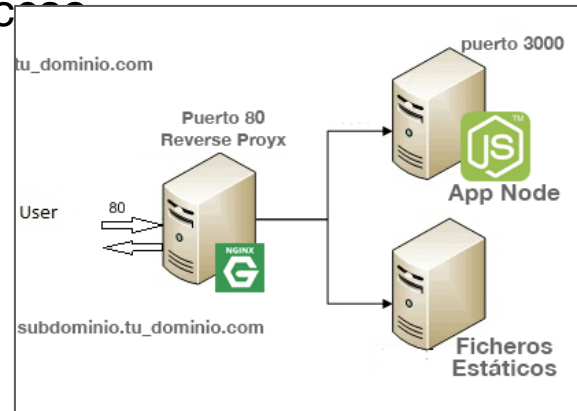
***¿Alguna pregunta hasta  
ahora?***

***NGINX***

# ¿Qué es?



- Nginx es un servidor web, orientado a eventos (como Node) que **actúa como un proxy** lo que nos permite redireccionar el tráfico entrante en función del dominio de dónde vienen, hacia el proceso y puerto que nos interese.
- Se usa para solucionar el problema que se genera al correr nuestra app Node en el puerto 80, para que sea accesible desde una IP o dominio, y queremos utilizar el mismo puerto con otro proceso



# ***Configurar Nginx para Windows***



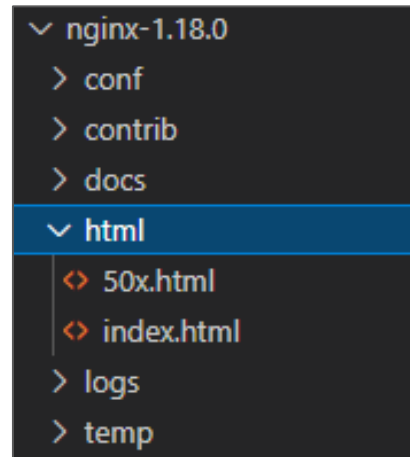
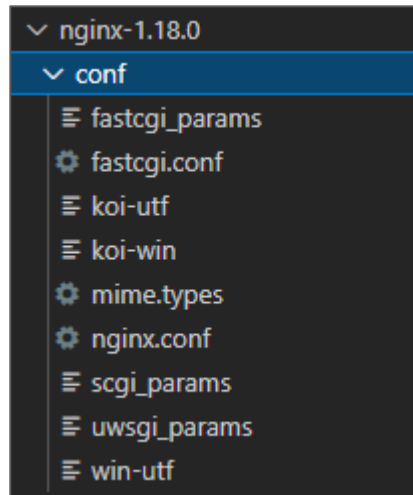
- Para configurar Nginx para Windows, primero debemos descargarlo del link: <http://nginx.org/en/download.html>
- Descargar la última versión *mainline*.
- Para empezar a configurarlo, en consola (por ejemplo, en el disco C: directamente) descomprimos el archivo descargado, e inicializamos el Nginx abriendo el ejecutable: `nginx.exe`



# ***Configurar Nginx para Windows***



- Observamos la estructura de carpetas del Nginx.
- La configuración se encuentra en la carpeta llamada *conf*.
- El espacio público está en la carpeta llamada *html*.



# Configurar Nginx para Windows



- Luego, para listar los procesos de Nginx, podemos usar el comando tasklist (aunque también podemos usar el Administrador de Tareas).

```
C:\nginx-1.19.10>tasklist /fi "imagename eq nginx.exe"

Image Name           PID Session Name        Session#    Mem Usage
=====
nginx.exe            652 Console              0           2 780 K
nginx.exe            1332 Console             0           3 112 K
```

- Uno de los procesos es el *master* y el otro es *worker*.
- Si Nginx no inicia, buscar la razón del error en la carpeta *logs/error.log*.
- Si no existe el archivo, entonces la razón la encontramos en el *Windows Event Log*.

# ***Configurar Nginx para Windows***



- Nginx en Windows se ejecuta como una aplicación estándar, pero también se puede operar mediante los siguientes comandos por consola:

`./nginx.exe -s stop` para un apagado rápido.

`./nginx.exe -s quit` para un cierre más elegante.

`./nginx.exe -s reload` para reiniciar el servidor al cambiar la configuración, iniciar nuevos procesos de trabajo con una nueva configuración, cierre elegante de los procesos de trabajo antiguos.

`./nginx.exe -s reopen` para reabrir logs de archivos.



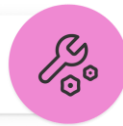
# ***EJECUTAR SERVIDOR NGINX***

*Tiempo: 8 minutos*

# ***Ejecutar servidor Nginx***

*Tiempo: 8 minutos*

Desafío  
generico



Descargar el servidor Nginx y ponerlo en funcionamiento.

Verificar que se encuentre corriendo como proceso del sistema operativo.

Hacer un request a su ruta raíz y verificar que esté ofreciendo el index.html que se encuentra en carpeta html. Realizar un cambio en dicha html y comprobar que se refleje en el navegador.

Integrar un css al index.html que modifique algún estilo del sitio de prueba (Ej. el color de un título). Luego añadir un archivo Javascript que saque un mensaje 'Hola Nginx!!!' por consola. Verificar que estos cambios se vean al requerir la página.



***BREAK***

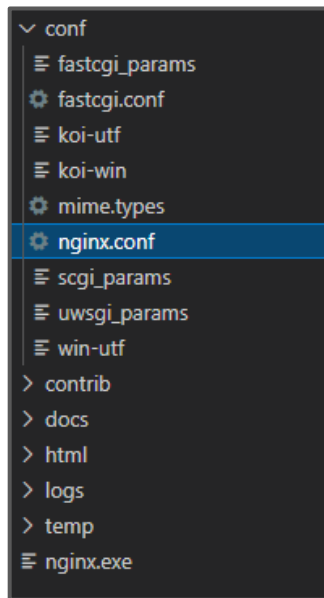
**¡5/10 MINUTOS Y VOLVEMOS!**

# ***NGINX CON PROXY INVERSO***

# *Configurar Nginx con proxy inverso*



- Vamos a configurar un servidor Nginx para utilizarlo con proxy inverso.
- Para eso, primero, cambiamos el archivo nginx.conf de la carpeta conf del Nginx por el siguiente código.



```
events {  
}  
  
http {  
    include      mime.types;  
    default_type application/octet-stream;  
  
    upstream node_app {  
        server 127.0.0.1:8081;  
        server 127.0.0.1:8082 weight=3;  
    }  
  
    server {  
        listen      80;  
        server_name mginx_node;  
        root        ../NginxNode/public;  
  
        location /datos/ {  
            proxy_pass http://node_app;  
        }  
    }  
}
```



# Configurar Nginx con proxy inverso



```
events {  
}  
  
http {  
    include      mime.types;  
    default_type application/octet-stream;  
  
    upstream node_app {  
        server 127.0.0.1:8081;  
        server 127.0.0.1:8082 weight=3;  
    }  
  
    server {  
        listen      80;  
        server_name mginx_node;  
        root        ../NginxNode/public;  
  
        location /datos/ {  
            proxy_pass http://node_app;  
        }  
    }  
}
```

- Podemos ver que están definidos los dos servidores de Node.
- El segundo se está usando como balanceador de carga (por eso se pone `weight=3`). Si no estuviera el peso, la carga se distribuye mitad para cada uno.
- Luego, configuramos el puerto, el nombre del servidor de Nginx y la ruta hacia el espacio público del proyecto en Node. En este caso, un directorio más arriba, dentro del proyecto de servidor node.

# *Proyecto en Node para el Nginx*



- Luego, creamos un proyecto de Node, donde el server.js lo configuramos de la siguiente forma.
- Debemos tener instalado el módulo PM2 para que todo funcione.

```
const express = require('express')

const app = express()

//app.use(express.static('public'))

const PORT = parseInt(process.argv[2]) || 8080

app.get('/datos', (req,res) => {
  console.log(`port: ${PORT} -> Fyh: ${Date.now()}`)
  res.send(`Servidor express <span style="color:blueviolet;">(Nginx)</span> en ${PORT} -
  <b>PID ${process.pid}</b> - ${new Date().toLocaleString()}`)
})

app.listen(PORT, err => {
  if(!err) console.log(`Servidor express escuchando en el puerto ${PORT} - PID WORKER ${process.pid}`)
})
```

# ***Proyecto en Node para el Nginx***



- En el código anterior, vemos que está comentada la línea de código del recurso del espacio público `app.use(express.static('public'))`.
- Está comentado, ya que si nuestro servidor Node se encargará de recursos, perdería rendimiento.
- Es por eso que Nginx se encarga de ofrecer los recursos estáticos.

# ***Ejecutando los servers de Node***



- Con esto hecho, ya podemos iniciar el proyecto de Node, tanto en ***modo Fork*** como en ***modo Cluster***.
- Prender el servidor en modo *Fork*:

```
$ pm2 start server.js --name="Server1" --watch -- 8081
```

- Nos debe aparecer en consola algo parecido a lo siguiente.

id	name	namespace	version	mode	pid	uptime	⓪	status	cpu	mem	user	watching
0	Server1	default	1.0.0	fork	17676	0s	0	online	0%	35.2mb	tam...	enabled

# Ejecutando los servers de Node



- Prender el servidor en modo *Cluster*:

```
$ pm2 start server.js --name="Server2" --watch -i max -- 8082
```

- Nos debe aparecer en consola algo parecido a lo siguiente.

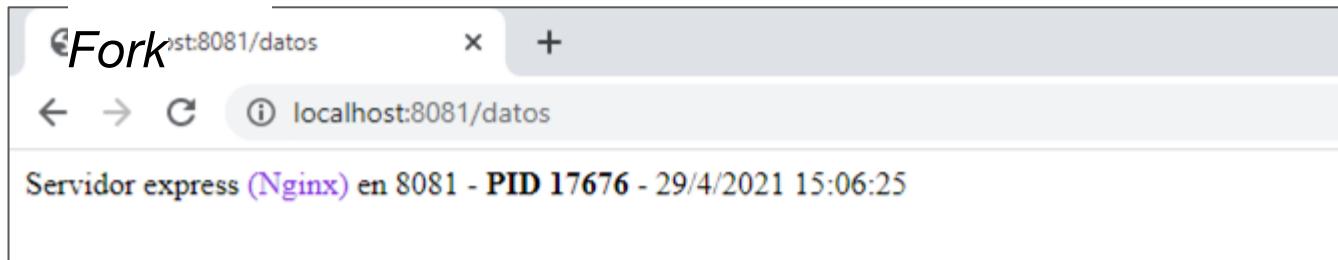
id	name	namespace	version	mode	pid	uptime	🔄	status	cpu	mem	user	watching
0	Server1	default	1.0.0	fork	17676	10s	0	online	0%	30.4mb	tam..	enabled
1	Server2	default	1.0.0	cluster	18004	3s	0	online	0%	36.9mb	tam..	enabled
2	Server2	default	1.0.0	cluster	18820	2s	0	online	0%	37.1mb	tam..	enabled
3	Server2	default	1.0.0	cluster	18972	2s	0	online	0%	36.9mb	tam..	enabled
4	Server2	default	1.0.0	cluster	3968	2s	0	online	0%	37.4mb	tam..	enabled
5	Server2	default	1.0.0	cluster	19876	2s	0	online	0%	37.0mb	tam..	enabled
6	Server2	default	1.0.0	cluster	19244	2s	0	online	4.7%	37.1mb	tam..	enabled
7	Server2	default	1.0.0	cluster	8896	1s	0	online	32.8%	36.9mb	tam..	enabled
8	Server2	default	1.0.0	cluster	14640	1s	0	online	50%	33.0mb	tam..	enabled

# ***Ejecutando los servers de Node***



- Podemos entonces chequear el funcionamiento de ambos servidores en el navegador:

Servidor



Servidor



# *Ejecutando el server de Nginx*



- Ahora ya podemos iniciar el Nginx. Para eso, primero lanzamos el ejecutable nginx.exe
- Luego, con el siguiente comando chequeamos que esté funcionando correctamente. Se debe mostrar en consola los procesos master y worker, como vimos anteriormente `$ tasklist /fi "imagename eq nginx.exe"`
- Con esto, ya podemos acceder al espacio público desde el puerto 80 (el predeterminado del navegador, el configurado en Nginx).  
*Esto es posible, por la siguiente línea del código del archivo nginx.conf mostrado anteriormente:*

```
root                ../NginxNode/public;
```

**NOTA:** El proyecto de Node se llama *NginxNode* y está en la misma carpeta que el Nginx.

# *Ejecutando el server de Nginx*



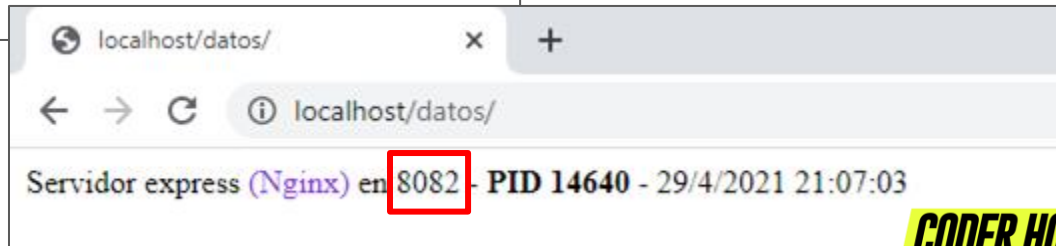
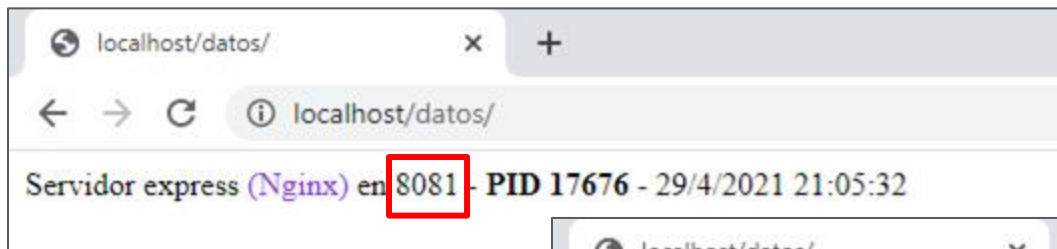
- Si modificamos la configuración, debemos reiniciar el servidor de Nginx para que la misma se ejecute, como lo indicamos en las diapositivas anteriores, a través de la consola, con el comando `‘./nginx.exe -s reload’`
- Se puede cambiar la configuración del peso en el balanceador de carga. Si sacamos este, al entrar a localhost/datos, e ir recargando, de forma pareja se ejecuta el puerto 8081, luego el 8082, luego de nuevo el 8081 y así sucesivamente.



# ***Ejecutando el server de Nginx***



Con el peso de 3 en el balanceador (puerto 8082), ejecuta primero el puerto 8081 una vez, luego 3 veces el 8082 y luego vuelve una vez al 8081 y así sucesivamente. Conviene hacerlo así ya que el servidor 8081 está en modo Fork, por lo que tiene menos capacidad de proceso que el 8082 que está en modo Cluster.



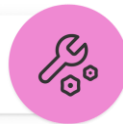


# ***PROBAR BALANCEADOR DE CARGA***

*Tiempo: 10 minutos*

# ***Probar balanceador de carga***

Desafío  
generico



*Tiempo: 10 minutos*

Realizar los siguientes cambios en el servidor Nginx que venimos utilizando:

- Volver a configurar el servidor Nginx con el puerto 80 como puerto de escucha.
- Derivar la información entrante en su ruta '/datos' a tres instancias Node.js escuchando en distintos puertos: ej: 8081 y 8082.
- Las instancias de node responderán en esa ruta: el número de puerto de escucha, su pid y la fecha y hora actual, ej: 'Server en PORT(puerto) - PID(pid) - FYH(fyh)'
- En principio el balanceo de carga será equitativo para las dos instancias.

# ***Probar balanceador de carga***

Desafío  
generico



*Tiempo: 10 minutos*

- La primera instancias (8081) correrá en modo fork (por código), la otra (8082) en modo cluster (por código) utilizando PM2 modo fork (sin -i max).
- Probar que con cada request cambie el servidor que responde en forma equitativa.
- Luego cambiar la configuración del servidor para que la carga se distribuya 1-3 entre las dos instancias de Node, es decir, la instancia 8082 atenderá el triple de paquetes que la instancias 8081. Verificar esta operación.

**NOTA:** la instancia de servidor node deberá recibir como primer parámetro el puerto de escucha y como segundo el modo de trabajo: FORK ó CLUSTER.



# ***SITIOS OFRECIDOS POR NODE O NGINX***

*Tiempo: 5 minutos*

# ***Sitios ofrecidos por Node o Nginx***

Desafío  
generico



*Tiempo: 5 minutos*

Realizar un sitio web sencillo (con HTML, CSS y Javascript) que sea ofrecido en principio por el servidor de node.js del desafío anterior, utilizando el servicio de archivos estáticos de express.

Las instancias de Node.js continuarán corriendo con PM2 en modo watch para incorporar nuevos cambios.

Comprobar que el sitio web se vea correctamente desde las dos instancias de Node.js en su ruta raíz.

Luego cambiar la configuración de Nginx para que ofrezca el sitio realizado en su ruta raíz, desactivando en las instancias de Node.js el servicio estático de archivos.

Comprobar que ahora Nginx sea el que ofrezca el sitio web y no Node.js



## ***SERVIDOR CON BALANCE DE CARGA***

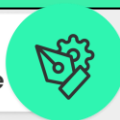
Retomemos nuestro trabajo para poder ejecutar el servidor en modo fork o cluster, ajustando el balance de carga a través de Nginx.

# EJECUTAR SERVIDORES NODE

**Formato:** link a un repositorio en Github con el proyecto cargado.

**Sugerencia:** no incluir los node\_modules

Desafío  
entregable



## >> Consigna:

Tomando con base el proyecto que vamos realizando, agregar un parámetro más en la ruta de comando que permita ejecutar al servidor en modo fork o cluster. Dicho parámetro será 'FORK' en el primer caso y 'CLUSTER' en el segundo, y de no pasarlo, el servidor iniciará en modo fork.

- Agregar en la vista info, el número de procesadores presentes en el servidor.
- Ejecutar el servidor (modos FORK y CLUSTER) con nodemon verificando el número de procesos tomados por node.
- Ejecutar el servidor (con los parámetros adecuados) utilizando Forever, verificando su correcta operación. Listar los procesos por Forever y por sistema operativo.



# EJECUTAR SERVIDORES NODE

**Formato:** link a un repositorio en Github con el proyecto cargado.

**Sugerencia:** no incluir los node\_modules

Desafío  
entregable



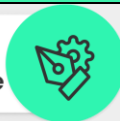
- Ejecutar el servidor (con los parámetros adecuados: modo FORK) utilizando PM2 en sus modos modo fork y cluster. Listar los procesos por PM2 y por sistema operativo.
- Tanto en Forever como en PM2 permitir el modo escucha, para que la actualización del código del servidor se vea reflejado inmediatamente en todos los procesos.
- Hacer pruebas de finalización de procesos fork y cluster en los casos que corresponda.

# ***SERVIDOR NGINX***

**Formato:** link a un repositorio en Github con el proyecto cargado.

**Sugerencia:** no incluir los node\_modules

Desafío  
entregable



## **>> Consigna:**

Configurar Nginx para balancear cargas de nuestro servidor de la siguiente manera:

Redirigir todas las consultas a */api/randoms* a un cluster de servidores escuchando en el puerto 8081. El cluster será creado desde node utilizando el módulo nativo *cluster*.

El resto de las consultas, redirigirlas a un servidor individual escuchando en el puerto 8080.

Verificar que todo funcione correctamente.

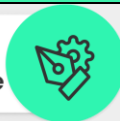
Luego, modificar la configuración para que todas las consultas a */api/randoms* sean redirigidas a un cluster de servidores gestionado desde nginx, repartiéndolas equitativamente entre 4 instancias escuchando en los puertos 8082, 8083, 8084 y 8085 respectivamente.

# ***SERVIDOR NGINX***

**Formato:** link a un repositorio en Github con el proyecto cargado.

**Sugerencia:** no incluir los node\_modules

Desafío  
entregable



## **>> Aspectos a incluir en el entregable:**

Incluir el archivo de configuración de nginx junto con el proyecto.

Incluir también un pequeño documento en donde se detallen los comandos que deben ejecutarse por línea de comandos y los argumentos que deben enviarse para levantar todas las instancias de servidores de modo que soporten la configuración detallada en los puntos anteriores.

Ejemplo:

- pm2 start ./miservidor.js -- --port=8080 --modo=fork
- pm2 start ./miservidor.js -- --port=8081 --modo=cluster
- pm2 start ./miservidor.js -- --port=8082 --modo=fork
- ...

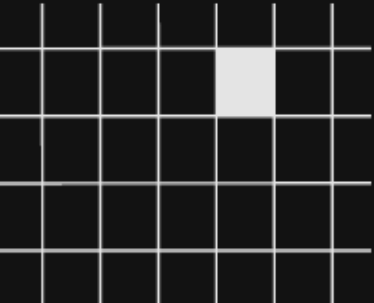
***¿PREGUNTAS?***





# ***¡MUCHAS GRACIAS!***

Resumen de lo visto en clase hoy:

- Módulo Cluster.
  - Forever.
  - PM2.
  - Proxy directo e inverso.
  - Nginx, configuraciones y uso.
- 



***OPINA Y VALORA ESTA CLASE***

***#DEMOCRATIZANDO LA EDUCACIÓN***