Universidad del País Vasco Euskal Herriko Unibertsitatea
BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

Ingeniaritza Goi Eskola Teknikoa
Escuela Técnica Superior de Ingeniería
Bilbao

ADMINISTRACIÓN DE SISTEMAS

NGINX & EXPRESS APP

9 de diciembre de 2021

Kerman Sanjuan Malaxechevarria

Índice

Capítulo 1

Introducción

1.1. Introducción

Este proyecto corresponde al trabajo individual de administración de sistemas. El objetivo será implementar una pequeña aplicación relacionada con NGINX. En nuestro caso hemos desarrollado un *endpoint* de una *API-REST* en Typescript, utilizando la librería express.

1.2. NGINX

Primero es necesario hablar sobre **NGINX**, que es la herramienta asignada a nuestro trabajo. NGINX es un servidor web, contando con funcionalidades como *load-balancer* y *reverse-proxy*, el cual explicaremos detalladamente más adelante.

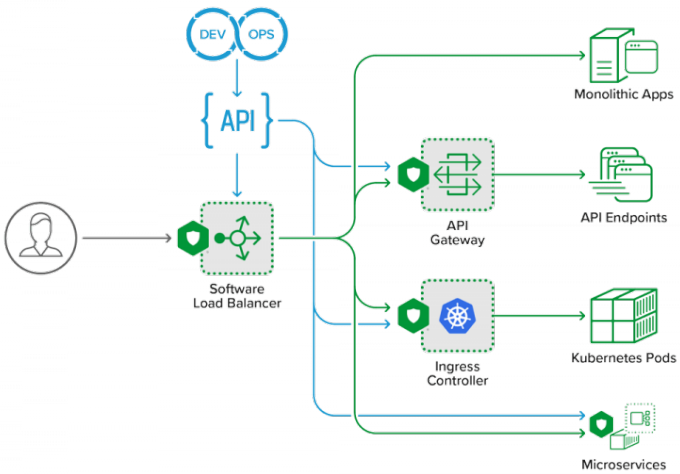

Figura 1.1: *Ejemplo de uso con NGINX*

Nginx está diseñado para ofrecer un bajo uso de memoria y alta concurrencia. Lo que hace diferente a Nginx es su arquitectura a la hora de manejar procesos, ya que otros servidores

web como Apache crean un hilo por cada solicitud. En lugar de crear nuevos procesos para cada solicitud web, Nginx usa un enfoque asincrónico basado en eventos donde las solicitudes se manejan en un solo hilo. Es decir, un mismo hilo hace "turnos"para poder prestar atención a todas las solicitudes. Con Nginx, un proceso maestro puede controlar múltiples procesos de trabajo, este proceso *master* mantiene los procesos de trabajo, y son estos lo que hacen el procesamiento real.

Ese hilo o proceso incluye varios microprocesos o llamadas de trabajo. Esto se traduce en que los hilos parecidos se gestionan bajo un proceso de trabajo, teniendo cada proceso de trabajo unidades de menor tamaño que llevan el nombre de conexiones de trabajo. Estas unidades son las que se dedican a manejar el hilo de las solicitudes. Las conexiones de trabajo (que puede llegar a atender más de 1000 solicitudes similares), entregan las solicitudes a un proceso de trabajo, que, a su vez, lo enviará a un proceso maestro. Al final, ese proceso maestro suministrará el resultado de las solicitudes correspondientes.[1]

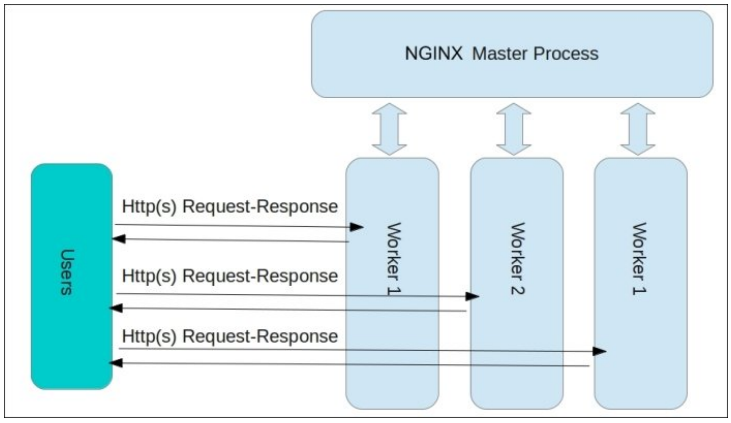

Figura 1.2: *Multithreading con NGINX*

La ventaja más notable que tiene este servidor web frente a todos sus competidores (Apache, Tomcat, Microsoft IIS... etc) es el rendimiento que ofrece, ya que es capaz de soportar enormes cantidades de conexiones de forma simultanea.

En conclusion, NGINX es un servidor web que también actúa como proxy de correo electrónico, proxy inverso y equilibrador de carga. La estructura del software es asincrónica y controlada por eventos; esto permite que se procesen múltiples solicitudes al mismo tiempo. NGINX también es altamente escalable, lo que significa que sus servicios crecerán con el crecimiento del tráfico de clientes. Siendo así que NGINX y Apache son los dos mejores servidores web del mercado.

Capítulo 2

Documentación

2.1. Uso y finalidades de la aplicación

Como hemos mencionado en la introducción, el objetivo de la aplicación es unir utilizar NGINX como un reverse proxy, es decir, las conexiones dirigidas al puerto 80 se redireccionaran al puerto de la API-REST habilitando de ese modo la conexión. Nuestra API-REST se encuentra en el *endpoint* http://localhost**/geolocate**.

Estas son las funcionalidades implementadas:

Protocolo	**Parametros**	**Devuelve**
GET	Ninguno	Lista todos los usuarios, sus localizaciones y identificadores
GET	?postal_code=X	Lista los usuarios con el código postal X
DELETE	?postal_code=X	Elimina todas las entradas que tenían el código postal definido.

Tabla 2.1: Funcionamiento de la aplicación.

2.2. Docker

Una vez desarrollado el código y para asegurar el funcionamiento de la aplicación en cualquier maquina, se nos ha pedido la *Dockerización* de la aplicación desarrollada.

Para ello hemos creado dos imágenes en docker, las cuales se ejecutaran dentro del mismo contenedor, como ya veremos mas adelante. La primera imagen es la encargada de NGINX y de configurar el *reverse-proxy*, la segunda es la encargada de ejecutar todo el *set-up* de la aplicación ExpressJS (Instalación de paquetes y ejecución de la aplicación).

2.2.1. Imágenes y contenedores

Para entrar en detalle de lo previamente mencionado, vamos a explicar los pasos que ha seguido cada imagen por encima.

- **NGINX:** Se ha creado un Dockerfile basandose en la imagen oficial de NGINX, el cual se ha expuesto el puerto 80 y se ejecuta un comando de arranque, el cual crea los ficheros de configuración para hacer funcionar el *reverse-proxy*. Está ha sido la parte más importante, debido a que la configuración tenía que ser establecida de esta forma,ya que las carpetas de configuración de NGINX se escribían una vez iniciada la imagen.
- **ExpressJS:** Una aplicación REST-API programada en Typescript, la cual se conecta y crea una base de datos utilizando una ORM. Esta aplicación escucha en el puerto 3000, que es donde el *reverse-proxy* redirige las peticiones desde el puerto 80.

2.2.1.1. Detalles de los Dockerfile

Se ha utilizado una imagen *Node17* de base para crear el Dockerfile que gestiona el *backend* de la aplicación.

2.2.2. Docker-Compose

Para poder ejecutar las dos imágenes simultáneamente y conectarlas entre ellas se ha utilizado la funcionalidad *Docker-compose*. Para ello, se cargan las imágenes que previamente hemos subido a Dockerhub para ahorrar tiempos de *buildeo*. Esto se ha hecho de esta forma para ahorrar tiempo, ya que normalmente la aplicación tardaba varios minutos en construirse (debido a las dependencias del *backend*).

2.2.3. Bind-Mount

Se ha realizado un *bind-mount* de la base de datos, de ese modo podemos mantener los cambios de la base de datos una vez cerrado el contenedor. Por temas de seguridad y reusabilidad se ha almacenado una copia de la base de datos en la carpeta /nodejs/data/, para así poder restablecer la base de datos con la versión original.

2.2.3.1. Detalles del Docker-Compose

Se han utilizado dos funcionalidades para poder hacer que la aplicación funcione de manera correcta. Por un lado tenemos la opción **tty** , lo cual hace que NGINX se ejecute

constantemente, sin que pare en ningún momento abriendo una *shell* interactiva. Por otro lado la opción **link**, la cual permite enlazar dos redes de forma interna, para así poder vincular dos puertos. Es decir, los enlaces permiten definir alias adicionales mediante los cuales se puede acceder a un servicio desde otro servicio.

2.3. Kubernetes

El segundo punto a desarrollar era la implementación de nuestra aplicación para que funcionara en Kubernetes. Para esto se han creado varios ficheros YML, los cuales son ejecutados mediante el comando *kubctl appy -f nombreDirectorio*.

Fichero	Tipo	Funcionalidad
api-deployment	Deployment	Lanza la API-REST
api-service	Service	Lista los usuarios con el código postal X
nginx-deployment	Deployment	Elimina todas las entradas que tenían el código postal definido.
nginx-service	Service	Lanza NGINX
load-balancer	LoadBalancer	Permite la conexión a NGINX desde una dirección externa*

Tabla 2.2: Funcionamiento en Kubernetes.

* Como su nombre indica, también se encarga de balancear la carga.

2.3.1. Detalles de Kubernetes

Para realizar la implementación y el testeo de Kubernetes se ha utilizado Google Cloud Platform y para ejecutar en local la herramienta **minikube**.

Capítulo 3

Final

3.1. Conclusiones

Ha sido un proyecto muy interesante, que ha permitido aplicar todo lo aprendido en clase estos últimos temas. Además de todo eso, en mi caso personal me ha permitido trabajar con NGINX,JS,Docker y Kubernetes, las cuales son tecnologías muy actuales y de gran relevancia en el sector. De ese modo he podido ampliar mi visión respecto al mundo del desarrollo,la ingeniería del software y DevOps.

Bibliografía

[1] que-es-nginx-como-funciona, 2021. [Online; accessed 3-December-2021].