UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

Cloud FaaS using ZeroMQ in NodeJS

SAD

M.U. en Ingeniería Informática

Martín Yerle

Ferran Pons

Miguel Feliu

31/01/2021

Contenido

Introducción

En este trabajo se va a exponer el proceso de desarrollo del proyecto de prácticas de la asignatura Servicios y aplicaciones distribuidas. Concretamente se trata de un FaaS con un servicio de colas, orientado al despliegue cloud.

Este trabajo se plantea como una oportunidad para realizar un sistema basado en microservicios, enfoque que está en pleno auge en la actualidad. Para ello haremos uso de los conocimientos aprendidos en la asignatura como son el uso de NodeJS con los módulos Express y ZeroMQ entre otros. Además, realizaremos un despliegue del sistema utilizando contenedores Docker mediante Docker-compose.

Como principales retos en este trabajo tenemos la coordinación entra las instancias de las colas (al menos 3), para que sean conscientes en todo momento de los workers que tiene cada una.

La funcionalidad básica de las peticiones del frontend que tienen que realizar los workers será la de “echo”, es decir devolver lo mismo que se ha enviado. En nuestro caso contaremos con una petición REST de tipo POST, todo lo que se le pase a esta en el body será devuelto al usuario, pasando por todo el sistema.

Contaremos también con un balanceador de carga de tipo “Round Robin” que permitirá distribuir las conexiones a las diferentes colas, tanto de clientes como de workers.

A continuación, se explicará la estructura de este sistema basado en microservicios de una manera gráfica, para ayudar a entender la implementación.

Seguidamente se expondrán los resultados de esta estructura con respecto a la propuesta inicial.

Finalmente, mostraremos los testeos realizados para comprobar las funcionalidades, mediante los cuales hemos asegurado el correcto funcionamiento del sistema.

Estructura

Dentro de este apartado haremos un análisis de forma general de la estructura escogida para llevar a cabo el proyecto y también comentaremos cada uno de los microservicios que conforman dicho proyecto de forma más concreta.

Como se mencionó anteriormente empezaremos por la estructura general del proyecto, así que a continuación tendremos la imagen que la representa:

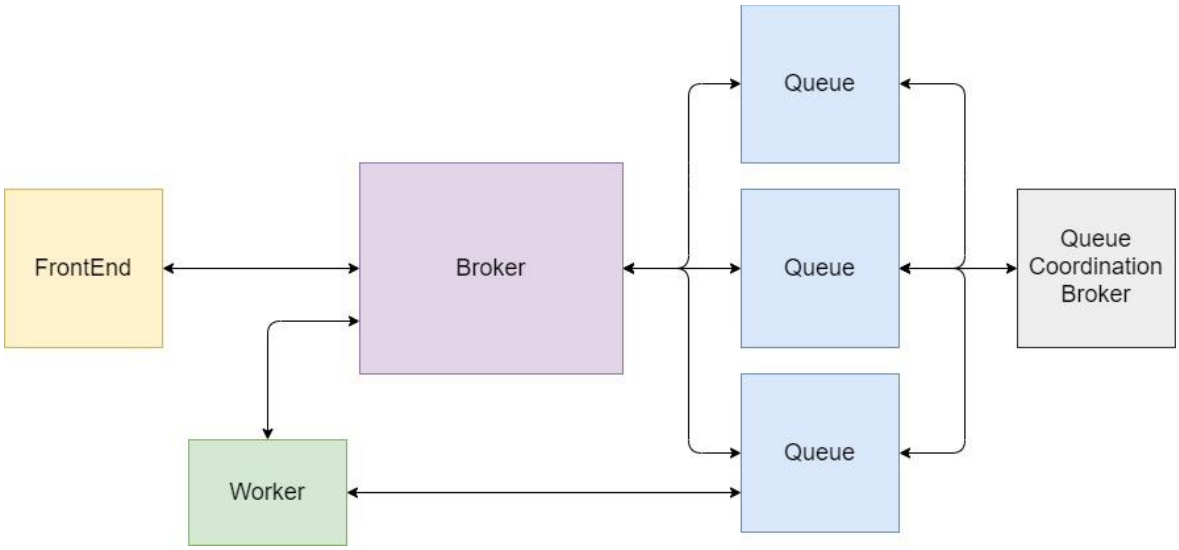

Ilustración 1 Arquitectura del Servicio

Como podemos observar en la imagen anterior, nuestra arquitectura está formada por 5 microservicios: FrontEnd, Broker, Worker, Queue, Queue Coordination Broker. Nuestro servicio empezará por una petición del FrontEnd, que llegará hasta el Broker. El Broker recibirá la petición y la enviará a una de las Queues que tenga a su disposición realizando un reparto balanceado, apoyándose en el sistema “Round Robin”.

Las Queues que previamente se habrán dado de alta en el Broker recibirán las peticiones que este último les enviará. Una vez que la Queue tenga en su poder la petición que inició el FrontEnd, se la pasará a una de los Workers que tenga asociado. En el caso de que no disponga de Workers asociados, la Queue, haciendo uso de Queue Coordination Broker sabrá si existen otras Queues con Workers disponibles que puedan realizar la tarea. Así que la Queue con la tarea asignada enviará dicha tarea a la otra Queue, para que ella y sus Workers se hagan cargo.

Los Workers, cuando se den de alta, deberán comunicarse con el Broker para solicitar la dirección de la Queue a la cualtienen que registrarse. Una vez recibida la dirección de la Queue, el Worker se comunicará con ella para asociarse y quedar a la espera de recibir las peticiones del cliente. Con lo que respecta a la resolución de las peticiones del cliente, el

Worker recibirá las solicitudes, realizará las operaciones necesarias y le devolverá a la Queue la petición resuelta.

La vuelta de la petición hasta el cliente es bastante trivial, una vez que la solicitud es resuelta por el Worker, este se lo pasa a la Queue asociada, esta ultima la devolverá al Broker para que este haga la entrega correspondiente al FrontEnd.

Una vez que ya hemos presentado y visto cómo funciona nuestra arquitectura en general, es momento de pasar al análisis del funcionamiento de cada microservicio en particular y como interactúan entre ellos.

FrontEnd – Broker:

Para el análisis de los microservicios empezaremos por el FrontEnd con el Broker:

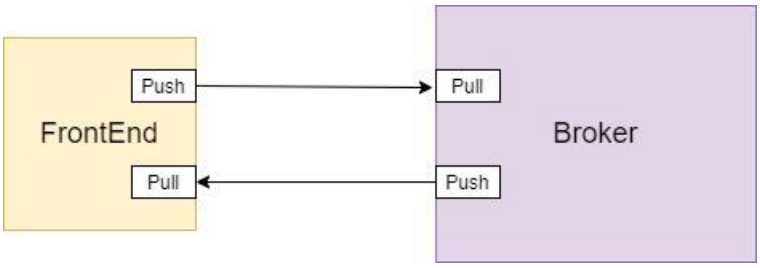

Ilustración 2 Arquitectura comunicación FrontEnd - Broker

Como podemos percibir gracias a la imagen anterior, el FrontEnd y el Broker se comunican haciendo uso del patrón Push/Pull. El FrontEnd hará uso de su socket push para enviar la petición al socket pull del Broker. Una vez que realiza la petición, el FrontEnd esperara recibir la respuesta de su solicitud en su socket pull proveniente del socket push del Broker.

Worker – Broker:

El siguiente análisis será sobre funcionamiento del Worker y de su interacción con el Broker:

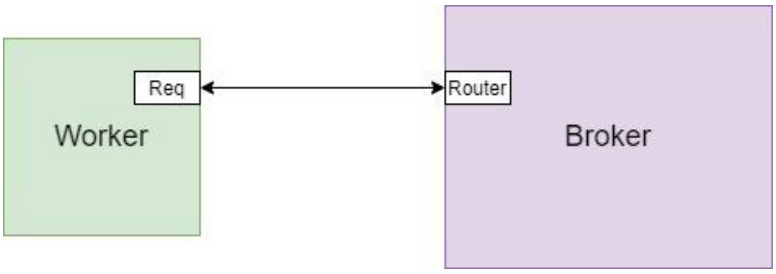

Ilustración 3 Arquitectura comunicación Worker – Broker

Mediante la imagen anterior vemos que el Worker se comunica con el Broker mediante un socket req, el cual es utilizado para hacer la solicitud de la dirección de alguna Queue disponible. Además, podemos observar que el Broker le devolverá la dirección de una de las Queue que tenga registrada como disponible mediante un Router.

Broker – Queue:

Dentro de este apartado veremos como una Queue interactúa con el Broker y mediante que socket lo hace:

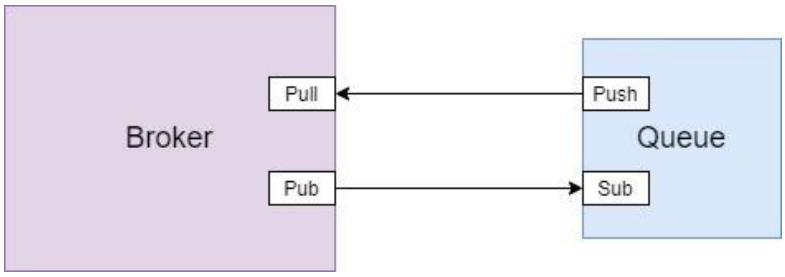

Ilustración 4 Arquitectura comunicación Broker – Queue

Observando la imagen anterior, vemos que se usan dos patrones, Pub/Sub y Push/Pull. La Queue hace uso del socket Push para dos acciones, una es para registrarse en el Broker realizando el envío de la dirección a la cual deberán de darse de alta los Worker. La segunda acción para la cual hace uso dicho socket es para enviar al Broker la petición que resolvió el Worker.

El otro patrón que vincula el Broker con la Queue es el Pub/Sub, este es utilizado para el envío de las peticiones del FrontEnd por parte del Broker, la elección de dicho patrón se debe a que cada Queue estará subscrita a un topic diferente, esto facilitará la realización de una distribución "Round Robin" por parte del Broker quien es el encargado de efectuar la publicación de cada petición.

Worker – Queue:

A continuación, veremos cómo trabaja la relación entre Workers y Queues para los envíos y resoluciones de peticiones:

Ilustración 5 Arquitectura comunicación Queue – Worker

La imagen anterior nos muestra que la comunicación mediante los dos microservicios (Worker y Queue) se ha realizado con un Req/Router, el Worker hará uso del socket Req para asociarse a una Queue especifica. Esa Queue será la que obtuvo por parte del Broker como vimos anteriormente. Una vez que se comunica y se asocia a la Queue pertinente, el Worker utilizará dicho socket para esperar la asignación de peticiones por parte de la Queue y devolverle las respuestas. Tanto las peticiones que tratará el Worker como las respuestas que enviará serán tratadas mediante el socket Router que tienen instalado las Queues.

Queue – Queue Coordination Broker:

En este apartado veremos cómo hemos gestionado la coordinación entre las diferentes Queues, analizando la arquitectura empleada, los sockets elegidos y como con esos sockets logramos gestionar la información necesaria para llevar a cabo dicha funcionalidad.

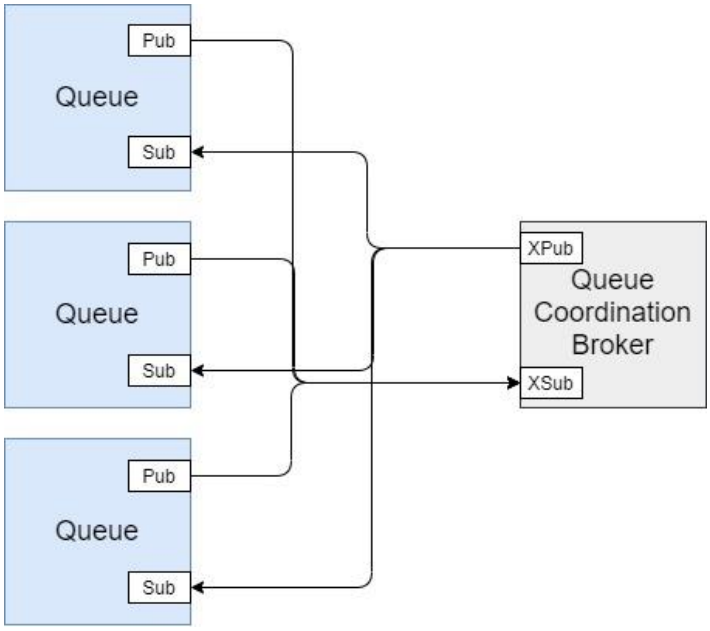

Ilustración 6 Arquitectura comunicación Coordinación entre Queues.

Como se puede apreciar en la figura anterior, la relación entre los sockets de ambos servicios es de Pub/XSub y Sub/XPub. Lo primero que debemos comentar es el motivo por el cual nos hemos decantado por implementar el patrón XPub/XSub, esto se debe a que mediante este patrón se logra crear un proxy. Con dicha implementación logramos hacer que un socket se logre subscribir a varios sockets publicadores.

Gracias a las posibilidades que nos brinda el patrón implementado (Proxy XSub/XPub), hemos creado un único topic, donde todos los miembros (Queues y Queue Coordination Broker) realizaran publicaciones y subscripciones. El mensaje que se enviará usando el topic general contendrá un identificador "type" que indicará la finalidad del mensaje:

- Estado de la Queue: el cual indicara el número de Workers disponible que tiene en ese momento la Queue.
- Trabajo: dicho mensaje contendrá el identificador de una Queue con Workers disponibles, la cual deberá de hacerse cargo de la peticion que lleva asociada. El mensaje vendrá de una Queue sin Workers disponibles, a la que se le asigno una petición y no puede satisfacerla.

Mediante estas acciones logramos satisfacer nuestros objetivos:

- Informar a cada microservicio cuando se produce un cambio en el estado de la Queue (llegada de un Worker nuevo).
- Cada uno sabrá el número de Workers disponibles que posee cada Queue.

- Realizar el envío de una petición a responder desde un Queue sin Worker disponibles a una Queue con Workers disponibles.
- Satisfacer todas las peticiones del cliente, por más que el broker asigne una petición a una Queue sin Workers disponibles.

Broker (Balanceador de carga):

En este apartado comentaremos como hemos aplicado el balanceador de carga en el Broker. En nuestro proyecto hemos realizado un balanceo tanto para la asignación de peticiones a Queues como para la asignación de Workers a Queues. Para poder realizar dicha tarea el Broker tendrá dos listas (Queues para Workers y Queues para peticiones), las que ira asignado y barajando para poder hacer uso de un sistema "Round Robin".

Despliegue con Docker-compose

Por último, el despliegue de todo este sistema se ha hecho mediante Docker-compose.

En primer lugar, se han creado las imágenes de cada servicio mediante un Dockerfile en cada carpeta. Seguidamente se ha construido el archivo docker-compose.yml que contiene la especificación de cada microservicio que compone el sistema y sus dependencias con otros microservicios.

Los parámetros necesarios, como pueden ser las direcciones de los sockets de los distintos servicios, se han resuelto mediante variables de entorno.

De todos los servicios se puede realizar un escalado automático mediante Docker-compose. En el caso de las colas este debe ser manual, ya que cada cola debe llevar un nombre distinto para operar correctamente en nuestro sistema. También se ha expuesto el puerto "3000" donde el servicio REST escuchará peticiones, mediante la ip local localhost:3000.

Para realizar el despliegue es tan sencillo como situarse en la carpeta raíz del proyecto y ejecutar el siguiente comando:

```
docker-compose up
```

Si se quieren desplegar más instancias de workers, se puede hacer mediante el comando

```
docker-compose up –scale worker=[NUM_WORKERS]
```

Una vez realizado el despliegue, podemos comprobar el funcionamiento del sistema de forma básica realizando una petición POST con cualquier JSON como *body* a la ip del service REST "localhost:3000/echo". Se debería recibir una respuesta con el mismo JSON que incluía el *body* de nuestra petición.

Repositorio Git

Todo el desarrollo del proyecto se ha llevado a cabo dentro de un repositorio Git hospedado en la plataforma GitHub, donde se han ido subiendo los cambios realizados por cada uno de los miembros mediante commits.

El enlace a dicho repositorio es el siguiente: https://github.com/miguelfeliu/sadlab/

Para clonar el repositorio simplemente debemos ejecutar el siguiente comando:

```
git clone https://github.com/miguelfeliu/sadlab/
```

Resultados

Llegados a este apartado del documento nos centraremos en comentar el resultado final obtenido con nuestro servicio y cuáles son las funciones que logramos satisfacer de la propuesta inicial. Partimos que para nuestro proyecto haremos un despliegue utilizando Docker, además consta de cinco microservicios:

- **FrontEnd**: que será capaz de recibir peticiones del cliente REST de tipo POST.
- **Broker**: se encargará de dar de alta las Queues que se vayan creando en nuestro servicio, también tendrá que distribuir las peticiones que lleguen del FrontEnd a las diferentes Queues empleando un sistema "Round Robin". Además, será el que asigne a cada Worker nuevo una Queue, proceso que también será realizado bajo una asignación "Round Robin". Por último, tendrá que llevar a cabo la devolución al FrontEnd de la petición resulta.
- **Worker:** este componente de nuestro proyecto será el que se encargue de resolver las peticiones del cliente. Serán Workers del tipo "echo", que estarán asignados a una Queue especifica.
- **Queue:** este microservicio será el encargado de recibir las peticiones por parte del Broker y asignárselas a un Worker. Para ello, tendrá constancia de la cantidad de Workers que tenga las otras Queues. Esto se podrá realizar gracias a que cada instancia de este componente será capaz de informar al resto de Queues de la llegada de un nuevo Worker en sus filas. También implementamos mediante Pings un control de seguimiento de que Workers siguen vivos en la Queue y cuales fallaron o se cerraron. Esto le permitirá poder enviar peticiones a otras Queues cunado no disponga de Workers para asignarles una petición.
- **Queue Coordinator Broker:** este componente fue creado para facilitar la realización de la coordinación entre Queues. Este microservicio estará conectado con cada Queue haciendo el papel de un Proxy para mantener a todas las Queues informadas del número de Workers que tiene cada una.

Con la combinación de todos los microservicios anteriores logramos implementar un sistema capaz de recibir peticiones REST del tipo POST de un cliente que le devolverá lo mismo que nos había enviado, es decir una funcionalidad de tipo "echo". Además, hemos dotado a nuestro servicio de un balanceador de carga y de un seguimiento de Workers, permitiendo saber en todo momento si ocurre un fallo en alguno de ellos. También cabe destacar que si se realiza una petición cuando no hay ningún Worker disponible en ninguna Queue, dicha petición permanecerá en esa Queue hasta que se conecte un Worker en esa misma colay pueda resolver la petición en cuestión.

Testeo

Dentro de este apartado expondremos la batería de tests que se realizaron para comprobar el correcto funcionamiento de nuestro proyecto. Los scripts realizados se encuentran alojados en una carpeta denominada test. Para la ejecución de los tests es necesario disponer de un gnome terminal, todos los tests están numerados y a continuación explicaremos cada uno de ellos:

Test 1:

En el primer test se ejecuta:

- Broker
- 3 colas (queueA, queueB, queueB)
- 3 workers
- FrontEnd
- Queue Coordinación Broker

Una vez que todos los microservicios están ejecutándose se lanza una petición *curl* y obtendremos la respuesta en ese mismo terminal. Con dicha prueba comprobamos como nuestro servicio es capaz de responder una petición proveniente del cliente.

Test 2:

El siguiente test buscara demostrar que una petición que es enviada a un Queue, sin tener Workers disponibles, se quedará esperando en la Queue hasta que se conecte un Worker. Una vez el Worker se conecta, resuelve la peticion y le llega al cliente la respuesta. Para realizar la siguiente prueba fue necesario:

- Broker
- FrontEnd
- 3 Queues (queueA, queueB, queueB)
- Queue Coordinación Broker

Una vez que estan esos tres microservicios se lanza la peticion curl por parte del cliente. Ahora con la peticion lanzada se lanzará a ejecución el microservicio:

- Worker

Para poder ver la respuesta tendremos que esperar unos segundos para que el worker se conecte a la Queue y resuelva peticion.

Test 3:

El test número tres lo busca comprobar es que, si enviamos una petición a una Queue que no posee Workers disponibles, pero alguna de las otras Queue si, la Queue que recibió la petición la delegará a otra Queue con Workers disponibles, los cuales serán encargados de resolver la petición inicial y mandarle la respuesta al cliente. Para eso ejecutamos:

- FrontEnd
- Broker
- 3 Queues (queueA, queueB, queueB)
- 3 Workers
- Queue Coordinación Broker

Una vez que están todos los microservicios en marcha, hacemos fallar el Worker ubicado en la QueueA y enviamos la petición curl. Una vez enviada la petición podremos observar en el terminal que se realizó dicha petición como obtenemos la respuesta que provendrá que otra Queue.

Test 4:

El test número 4 comprueba que el balanceador de carga de las peticiones y los Workers se basa en un “Round Robin”. Para comprobar eso hemos ejecutado:

- FrontEnd
- Broker
- 3 Queues (queueA, queueB, queueB)
- Queue Coordinación Broker

Cuando tenemos todos los microservicios anteriores ejecutando, haremos uso de un bucle para ir creando Workers y así podremos ir viendo en el terminal de Queue Coordination Broker como cambia el estado de las Queues.

Extras

Seguimiento de los Workers:

En el proyecto realizado hemos implementado el extra que se relacionaba con el seguimiento de los Workers por parte de las Queues. Las Queues siempre tienen conocimiento de los que pasa con los Workers, si llega uno nuevo a una Queue así como si se elimina o cae uno de ellos.

Para poder realizar dicho control, nos hemos asegurado de que cada vez que un Worker se conecte a una Queue por primera vez, la Queue en cuestión realizara un Broadcast del número de Workes disponible que tiene. Este le llegará al resto de Queues mediante el Queue Coordination Broker. Con lo que respecta a la caída o eliminación de un Worker, lo hemos controlado realizando Pings a los Workers. Cuando alguno de los Pings falle, la Queue que sufra el fallo avisará al resto de incidente sufrido y actualizarán todos los números de Workers disponibles en dicha Queue.

Conclusiones

Para finalizar el documento podemos asegurar de que estamos orgullosos del trabajo realizado, ya que hemos sido capaces de implementar el despliegue de un FaaS basado en la interacción de diferentes microservicios que logra satisfacer en gran medida la propuesta realizada inicialmente.

Por otra parte, también somos conscientes de que hay margen de mejora en algunos aspectos, como por ejemplo la técnica utilizada con respecto a la cantidad de veces que se envía un mensaje para ser procesado, que en nuestro caso es “At most once”. Además, se podría dotar a nuestro proyecto con la posibilidad de tener Workers con una especialización diferente.