UNIVERSIDAD AUSTRAL

INGENIERÍA

Sistemas Distribuidos

Trabajo Práctico 1

Profesor

- Fernando Rodríguez

Alumnos

- Pedro Aráoz
- Katia Cammisa
- Rocío Ferreiro Rico
- Alejo Ramírez Gismondi

Fecha de entrega

- 08/04/2021

1) ¿El servicio es escalable ? ¿ Qué ocurre con la escalabilidad en tamaño, geográfica y administrativa ? ¿ Qué límites tiene ?

El servicio es escalable en cantidad de requests realizadas, ya que puede balancear la carga entre sus múltiples stubs para poder procesar mayor cantidad de requests. Sin embargo, en este punto del TP estos stubs están hosteados en la misma máquina con lo cual el uso que pueden hacer de memoria y CPU está limitado a los recursos de esa máquina. Un punto a favor es que al estar construido sobre un container de Docker, es muy sencillo mover el sistema a un equipo con más recursos si se necesita, pero esto aún no está automatizado, como lo podría estar en Amazon.

Respecto a la escalabilidad geográfica esto todavía no está implementado, ya que el servicio está hosteado localmente sin contemplar ningún tipo de sistema de distribución. Esto limita la homogeneidad de la latencia de las requests a la ubicación geográfica desde donde se llamen.

La escalabilidad Administrativa, como se mencionó anteriormente, tampoco es ideal. La implementación actual contempla que no se tenga que administrar manualmente el balanceo de carga entre stubs ni tampoco el caso en el que alguno de sus stubs se encuentre unhealthy y tenga que cambiar al otro. Sin embargo, si se quiere crear más stubs o si se necesita aumentar o reducir la memoria o la CPU este proceso no está automatizado ni es fácilmente escalable, ya que necesita intervención manual.

2) ¿Qué técnicas utilizó para mejorar la escalabilidad?

En este TP se utilizaron varias técnicas para mejorar la escalabilidad de la aplicación. En primer lugar, se implementaron los servicios utilizando gRPC (Google Remote Procedure Calls), el cual utiliza protocol buffers como lenguaje de descripción de interfaz.

Lo que nos permite gRPC es contar con herramientas como load balancing, tracing y health checking, lo cual utilizamos para manejar la escalabilidad en términos de la carga de requests. Además, Protocol Buffer ofrece transmisión de mensajes entre microservicios en formato binario, con lo cual la comunicación de los servicios con los clientes y entre sí es mucho más rápido que si se utilizara un protocolo como json.

Esta forma de implementar la arquitectura tiene además la ventaja de que puede implementarse en cualquier lenguaje de programación. Es decir, que clientes y servicios escritos en distintos lenguajes pueden comunicarse a través de este protocolo disfrutando de los beneficios que este trae.

Paralelamente se encapsuló el servicio en un container de Docker, para garantizar que pueda correr en cualquier máquina sin estar atado a un sistema operativo en particular ni necesitar descargar dependencias directamente en la máquina donde corre. Esto provee un salto en el potencial de escalabilidad, ya que permite que se migre la aplicación a un nuevo equipo o a un servidor de Amazon sin problemas.

3) Explicar si se logra o no los siguientes tipos de transparencia (Ubicación, Migración) y cómo se logra.

Transparencia de Ubicación no se logra en esta iteración del TP ya que solo nos podemos conectar al servicio a través de un puerto local. Se podria hostear el container en un servidor para poder conectarse remotamente. Esto afortunadamente es sencillo porque está en un contenedor de Docker. Sin embargo, el problema que tiene es que al no tener una red de distribución de contenido, la latencia que tendrá para requests realizadas desde distintos puntos geográficos será muy diferente, lo cual viola la transparencia de Ubicación.

Respecto a la transparencia de Migración nuestra aplicación lo logra con bastante eficacia. Al estar en un container de Docker es muy sencillo levantar el servicio en una máquina distinta, con lo cual se puede migrar fácilmente a un servidor en Amazon, a una PC local o a cualquier plataforma que pueda correr contenedores de Docker.

4) Si una instancia del servicio falla, ¿funciona el fail-over?, ¿Cómo?

En caso de una falla en uno de los stubs del servicio, el proceso de las requests se podrá redirigir a los demás stubs que se encuentren en estado healthy. Esto servirá como fail-over para evitar el fallo de la aplicación.

Cuando un stub devuelve una request con estado Failure, este se elimina de la lista de healthy stubs y se continúa procesando con los stubs que estén disponibles. La política de load balancing que elegimos es el “pickfirst” donde se envía la request al primer healthy stub disponible. En caso de que este falle se reemplaza por el segundo.

Esta solución no contempla el caso en el que todos los stubs detengan su ejecución. En ese caso se deberá tener una intervención manual.

5) Con respecto a los principios las arquitecturas SOA. ¿Cómo se representa el contrato del servicio? ¿Qué puede decir con respecto al encapsulamiento?

El contrato que se establece entre los servicios es el medio por el cual se comunican. El GeoService de proto cuenta con una especificación de los métodos que luego se van a implementar en la clase de Scala (o cualquier lenguaje) para formar el servicio. Los métodos aceptan y retornan mensajes definidos en el mismo archivo .proto, los cuales poseen una serie de características identificadas con tipo de dato, nombre y número identificador el cual debe ser único dentro de cada mensaje.

El encapsulamiento se encuentra bien desarrollado ya que no deja ver las conexiones entre las interfaces y las API’s. El usuario puede obviar la implementación de los métodos y propiedades para concentrarse sólo en cómo utilizarlos.