Implementando una arquitectura en docker y aws

Jeisson Geovanny Sanchez Ramos

Septiembre 2020

1 Introduccion

En el presente documento miraremos una manera en que podemos implementar un balanceador de carga y una serie de esclavos (Nodos hijos), para servir una aplicación web de alta disponibilidad, a partir de una máquina virtual AWS.

2 Vistaso general

La arquitectura solicitado consiste básicamente de un nodo master, él requiere exclusivamente definir a que nodo se va a dirigir el mensaje enviado, para esto la manera, para lograr dicho objetivo debemos seleccionar un conjunto de puertos mediante los cuales nuestros nodos van a servir información, y nosotros vamos a dirigir las solicitudes, de igual manera necesitamos inspeccionar el contenido recibido para poder reenviarlo (Content-Type, Body,parameters,Method).

Para los puertos requerimos trabajar con un entero atómico que nos dé el puerto al que debemos redirigir la petición, por motivos de memoria se decidió realizar su implementación, ya que la opción más sencilla era incrementando y perder el mod del número, dado a que este incrementa, va a llegar un tope en que el servidor no tenga la capacidad para responder a las solicitudes.

3 Arquitectura de la aplicación en los nodos

Para la implementación de la aplicación en los nodos, se decidió utilizar spark, con funciones lamda y servidor de contenido estático, también utilizamos el driver de mongodb para la comunicación con la base de datos de manera directa sin ningún otro framework como intermediario, la aplicación la dividimos en las capas tradicionales del desarrollo web, servicios, modelo, persistencia y nuestra clase App que nos sirve de controlador.

La app consiste en un grupo de registros asociados a una fecha, esta fecha se decidió tomarla directamente del servidor mediante el backend.

También definimos un front end que nos permite consumir los datos mediante una interfaz gráfica bastante sencilla.

Para la capa de datos por facilidad se decidió implementar un mini ORM, que nos devuelva objetos propios de nuestro modelo ya que estos nos facilitan realizar algunas validaciones sobre los distintos datos que nos son enviados a la aplicación.

En el controlador para devolver la respuesta se decidió utilizar la librería GSon para transformar nuestras entidades a registros que claramente el browser pueda interpretar como lo es Json.

4 Alta disponibilidad

Nuestra aplicación web tiene como ventaja que posee 3 nodos diferentes atendiendo peticiones, esto nos permite responder a un mayor número de peticiones sin que se lleguen a generar algún tipo de error.

Pero hay consideraciones para tener en cuenta, esto es ventajoso en términos de procesos, pero tienen la limitante de que comparte recursos físicos con la máquina, así cuando llegamos al máximo de memoria en nuestro equipo vamos a tener problema de indisponibilidad, evento que no es tan notorio en distintas maquinas, ya que estas al tener memorias independientes una va a dejar de funcionar mientras las otras podrán continuar respondiendo.

Por el tipo de implementación del balanceador de carga, asumimos que todos los nodos están funcionando, en caso de que un nodo falle, vamos a rechazar el 33,3

5 Nivelar Carga

En términos de responsabilidades se decidió que nuestro balanceador de carga iba a consumir los recursos expuestos de nuestra maquina por los puertos 90,91 y 92, nuestro balanceador de carga va exclusivamente a consumir los recursos, ya sean html, css,javascript, json, etc directamente de cada nodo, el nodo maestro mira del request y response el tipo de solicitud para que la respuesta pueda sentirse como si hubiera sido dada directamente por el servidor.

Para nivelar la carga al ser necesario hacer peticiones a los nodos hijos, nos fue necesario crear una clase que nos permitiera ejecutar la petición, y nos devolviera el body y el Content-Type dado por la aplicación, para así evitarnos la necesidad de realizar algún tipo de cálculo, y que nuestro javascript conociera su tipo y fuera capaz de construir su respuesta, para evitarnos el paso innecesario de pasar un string a JSon que jQuery hace en base al content-type de la

respuesta.

Resulta importante considerar que estas validaciones del Content-Type son importantes a tener en cuenta ya que un balanceador de carga es una abstracción muy genérica y el objetivo de su uso se encuentra en que sea un intermediario invisible para el usuario, que nos de la misma respuesta del servidor, y nos haga sentir que estamos realizando nuestras peticiones a una única máquina, aunque en este caso sean 3 nodos.

Por motivos prácticos de este trabajo se definió que la url de nuestra maquina estuviera quemada, aunque se hubiera podido pasar como una variable de entorno al programa para crear una solución mas genérica, y de igual manera se hubiera podido realizar con los puertos, ya que estos podían haber sido dados como un rango, también por medio de las variables de entorno.

Para la arquitectura propuesta únicamente requeríamos un conjunto de peticiones get y un único post, el balanceador de carga fue diseñado para interceptar todas las peticiones get y post, independientemente de su url de origen.

La clase construida para realizar los request a los nodos, por la composición de requerimientos podría realizar las operaciones de delete y put, vale a clarar que este balanceador de carga al no contemplar todos los headers, no admitiría operaciones que requieran un token de autenticación, o otros contenidos que se suelen pasar mediante las cabeceras de la petición.

6 Conclusion

Como pudimos ver en el presente documento Docker es una herramienta que nos ayuda a compartir nuestras aplicaciones, y garantizar que estas sirven en diferentes equipos, pese a no tener instalado algún requerimiento particular de la imagen, pero para la solución es una propuesta interesante y bastante más ligera que utilizar un conjunto de máquinas virtuales, de igual manera tiene un conjunto de limitaciones, como lo es que todas las imágenes se encuentran ligadas a una única pieza de hardware.

Como pudimos ver en las consideraciones de un balanceador de carga, logramos el objetivo de hacer que fuera invisible para el usuario, aunque no completamente ya que debimos haber incluido otros headers de la petición, pero en términos de lo que solicitaba el ejercicio llegamos allí.