ARQUITECTURA EMPRESARIAL
TALLER 3
CLIENTES Y SERVICIOS

Andres Ricardo Martinez Diaz

Septiembre 2020

1. Introducción

La situación consiste en crear una aplicación web que podrá insertar mensajes en una base de datos y consultar los últimos 10. La arquitectura de esta aplicación funciona de la siguiente manera: Un servicio web y solo uno será el encargado de recibir las peticiones del cliente y distribuirlas a otros servicios web usando un balanceo de carga Round robin. Round robin es un método para seleccionar todos los abstractos en un grupo de manera equitativa y en un orden racional, normalmente comenzando por el primer elemento de la lista hasta llegar al último y empezando de nuevo desde el primer elemento. Los otros servicios web son quienes se conectarán una base de datos NoSQL en este caso se usará mongoDB ya que es una base de datos de documentos que ofrece una gran escalabilidad y flexibilidad, y un modelo de consultas e indexación avanzado. En total serán 5 servicios, el servicio que realizara el balanceo de carga, 3 servicios web que se conectarán directamente a la base de datos y la base de datos misma. Cada uno de estos servicios funcionara en contenedores Docker. Al final se desplegará este servicio en Docker en una instancia EC2, una máquina virtual en la nube de AWS, y se probará su funcionamiento. Estos contenedores en Docker solo podrán comunicarse entre sí y el único que se comunicará fuera del entorno de Docker será el servicio de balanceo de carga para mantener la seguridad de la base de datos. Docker", el software de TI, es una tecnología de creación de contenedores que permite la creación y el uso de contenedores de Linux®. Con docker, puede usar los contenedores como máquinas virtuales extremadamente livianas y modulares.

2. Diseño

Para construir los servicios web se usará el framework de java spark, y habran dos clases principales para estos servicios SparkWebServer y AppRoundRobin.

2.1. MongoServices

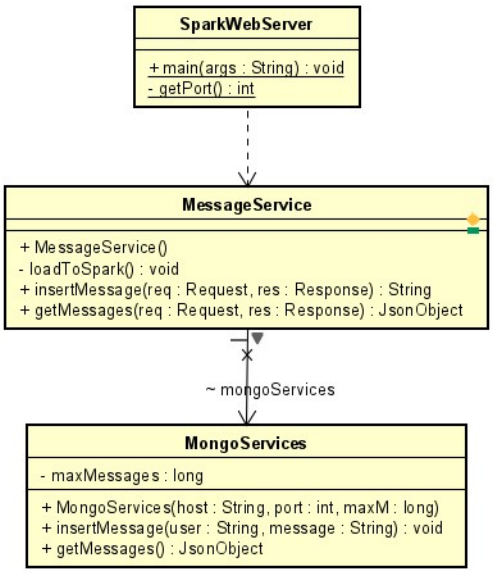
Esta clase será quien se encargue de realizar la conexión directa con la base de datos, en su constructor creara el cliente con la dirección y el puerto que le entran como parámetro, además tendrá un atributo de máximo de los últimos mensajes que consultara.Los métodos serán los siguientes: InsertMessage: Este método recibirá el usuario y contenido del mensaje, y creara un documento con esta información más la fecha del momento en que se esta insertando el mensaje. Luego insertará el documento en la base de datos. getMessages: Este método se encargará de realizar la consulta de los ultimos 10 documentos en la base de datos, los transformafa en un objeto Json y retornara este objeto a quien lo haya llamado

2.2. MessageService

Esta clase se encargará de crear una instancia de MongoServices en y, usando Spark, asociar métodos con rutas en el api. Los métodos serán los siguientes. InsertMessage: Este método extraerá la información del mensaje de un objeto Json y se lo pasará a MongoServices para que se encargue de agregar los mensajes. getMessages: Este método se encargará de pedir los mensajes de MongoServices e indicar en encabezado de la respuesta que es un contenido de tipo json.

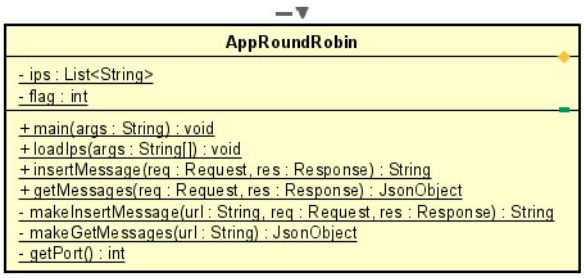
2.3. SparkWebServer

Esta clase es la clase principal del servicio web que se conecta a la base de datos y que será replicado tres veces. Esta clase creara una instancia de MessageService e iniciara spark por el puerto que se haya indicado en el sistema.



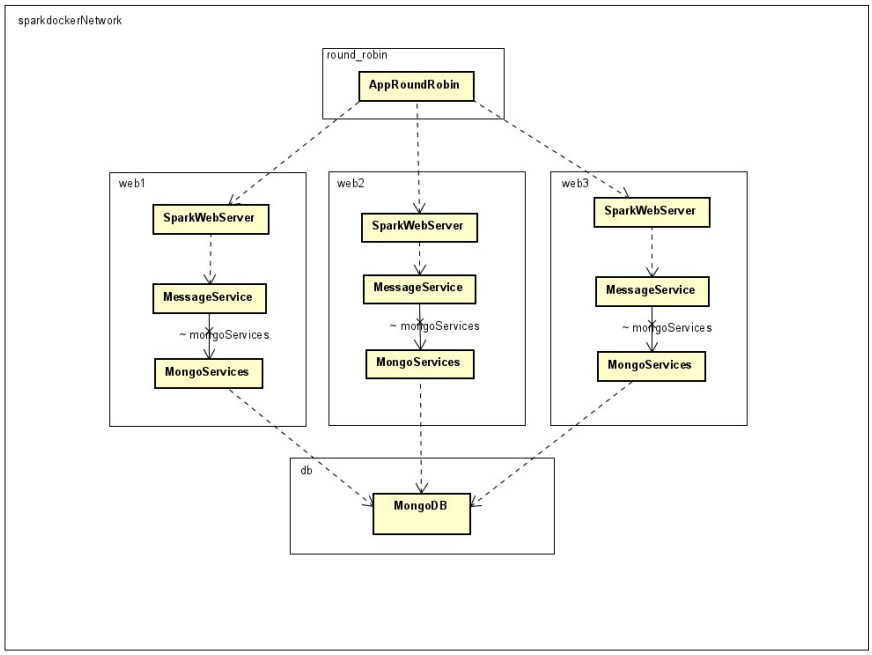
2.4. AppRoundRobin

Esta clase es la clase principal del servicio web que se conecta a otros servicios, su trabajo será recibir como argumento las direcciones de los servicios web, de esta manera podrá ejecutarse con múltiples servicios web SparkWebServer y no se verá limitado en este aspecto y al ejecutarse iniciará spark por el puerto que se haya indicado en el sistema. Su método loadIps recibirá las direcciones ip y las agregara a una lista que tiene la clase como atributo. Sus métodos insertMessage y getMessages serán los que estarán asociados al API. estos métodos escogerán una direccion ip, mediante un atributo, usando el flag actual después definirán un nuevo flag, es decir el servicio web al que le va a pasar la solicitud, luego makeInsertMessage y makeGetMessages usando librerías de red de java enviaran la petición al servicio web escogido y devolverán lo que sea que este haya respondido.



3. Arquitectura en Docker

Dentro de docker los servicios se comunicarán de la siguiente manera:



4. Evaluación en AWS

```
[ec2-user@ip-172-31-40-91 AREP-Lab05-SparkDocker]$ docker ps
CONTAINER ID        IMAGE                 COMMAND                  CREATED             STATUS              PORTS                     NAMES
a5c2cc6f5227        sparkdocker           "java -cp ./classes:…"   4 minutes ago       Up 3 minutes                                  web2
6c8d81ca2041        mongo:3.6.1           "docker-entrypoint.s…"   4 minutes ago       Up 3 minutes        27017/tcp                 db
235d742cce0d        sparkdocker_round_robin   "java -cp ./classes:…"   4 minutes ago       Up 3 minutes        0.0.0.0:35004->6000/tcp   round_robin
ba41c4465296        sparkdocker           "java -cp ./classes:…"   4 minutes ago       Up 4 minutes                                  web1
2f2e9af8003f        sparkdocker           "java -cp ./classes:…"   4 minutes ago       Up 4 minutes                                  web3
[ec2-user@ip-172-31-40-91 AREP-Lab05-SparkDocker]$
```

ec2-3-85-42-154.compute-1.amaz

ec2-3-85-42-154.compute-1.amazonaws.com:35004

Mensajes

Usuario: AWS 1

Mensaje:

Primer mensaje aws.

Ver Mensajes | Borrar información

Enviar

ec2-3-85-42-154.compute-1.amaz

ec2-3-85-42-154.compute-1.amazonaws.com:35004/api/v1/getMessages?

JSON | Datos sin procesar | Cabeceras

Guardar | Copiar | Contraer todo | Expandir todo | Filtrar JSON

```
Messages:
  0:
    Id:       "5f691000fdb752332628ed8e"
    User:     "AWS 1"
    Message:  "Primer mensaje aws."
    Date:     "Mon Sep 21 20:41:36 UTC 2020"
```


```
"Containers": {
    "235d742cce0dab903d398e103c3aaf31c99b436d56c3907325c064aa02906adf": {
        "Name": "round_robin",
        "EndpointID": "3aef8a737633d4826740b591c71e83cd6887a4f420cf543ee5ae812a65c28578",
        "MacAddress": "02:42:66:18:00:06",
        "IPv4Address": "102.24.0.6/16",
        "IPv6Address": ""
    },
    "2f2e9af8003fdf70ddc6d9af9e57dac65f7a310d20ecd7e1362e49f69385d2a7": {
        "Name": "web3",
        "EndpointID": "d3d3dcafaf82c621656bba33f911a9994b196948a0b69cdfd97c37f880da76a8",
        "MacAddress": "02:42:66:18:00:04",
        "IPv4Address": "102.24.0.4/16",
        "IPv6Address": ""
    },
    "6c8d81ca2041ec0bcddcd6960fe1239da259b9f9daa4f71866ab8a5f526ed53a": {
        "Name": "db",
        "EndpointID": "85e355053fff7eed02d52e9c78317ccd8c468cdcf542258c19698abc8ed52508",
        "MacAddress": "02:42:66:18:00:05",
        "IPv4Address": "102.24.0.5/16",
        "IPv6Address": ""
    },
    "a5c2cc6f52278e6daf94b784be37ae5687a4aead0b8090cae476d14f754829c3": {
        "Name": "web2",
        "EndpointID": "182feb82d64f1a8b045856e04b9985e3dad70f4b24060e703d0e482e46d9be54",
        "MacAddress": "02:42:66:18:00:03",
        "IPv4Address": "102.24.0.3/16",
        "IPv6Address": ""
    },
    "ba41c44652969f41650f9e117b29df6fa9fc092650eb4c32074c9d80e5d2f4dd": {
        "Name": "web1",
        "EndpointID": "a132046a4708bfabe46b0fb07807b4e62ff4f40cc1e6fd4ea96745d93fbdc3a2",
        "MacAddress": "02:42:66:18:00:02",
        "IPv4Address": "102.24.0.2/16",
        "IPv6Address": ""
    }
},
```


```
[ec2-user@ip-172-31-40-91 AREP-Lab05-SparkDocker]$ docker logs round_robin
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
http://102.24.0.2:6000/api/v1/setMessage
http://102.24.0.3:6000/api/v1/getMessages
GET Response Code :: 200
http://102.24.0.4:6000/api/v1/getMessages
GET Response Code :: 200
http://102.24.0.2:6000/api/v1/getMessages
GET Response Code :: 200
http://102.24.0.3:6000/api/v1/getMessages
GET Response Code :: 200
http://102.24.0.4:6000/api/v1/getMessages
GET Response Code :: 200
http://102.24.0.2:6000/api/v1/getMessages
GET Response Code :: 200
[ec2-user@ip-172-31-40-91 AREP-Lab05-SparkDocker]$
```

5. Conclusiones

1. Lo más desafiante fue aprender acerca de docker, una vez se entendía que tanto se podía hacer con docker el resto del desarrollo fue bastante sencillo. Cuando los contenedores en Docker se cargan el entorno les asigna una dirección ip por defecto, cuando note esto busqué como asignarles direcciones ip especificas en el docker-compose de esta manera me aseguro de que sin importar en donde se creen los contenedores siempre les serán asignadas las mismas ip y estarán en la misma red.

2. Otro reto fue aprende a realizar consultas e inserciones en MongoDB y a aprender a hacerlas en Java, esto me pareció muy interesante ya nunca había realizado consultas en bases de datos NoSQL.

6. Bibliografía

- Red Hat. (s. f.). ¿Qué es DOCKER? Recuperado 19 de septiembre de 2020, de https://www.redhat.com/es/topics/containers/what-is-docker