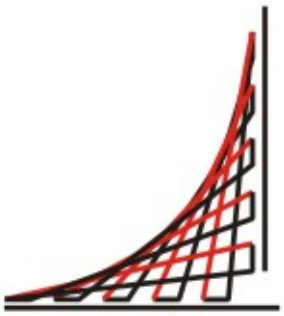

ESCUELA COLOMBIANA DE INGENIERÍA JULIO GARAVITO

AREP

Arquitectura Empresarial

Documentación Arquitectura: modularización con virtualización e Introducción a Docker y a AWS

Authors:
Andrés Felipe Marcelo Rubiano

Escuela Colombiana de Ingeniería Julio Garavito
Septiembre 2020

Índice

1. Introduction

La transformación digital en el mundo ha impulsado el crecimiento del uso de servicios cloud en las organizaciones, esto convierte las necesidades de computación sobre la nube en una oportunidad vital para publicar los servicios que se necesiten de manera rápida, práctica y segura. El objetivo de este informe es el de presentar resultados acerca de la implementación en contenedores y despliegue en AWS de una arquitectura empresarial que contenga un balanceador de carga, tres servidores web que reciban peticiones y un servidor de base de datos NoSQL Mongo.

2. Arquitectura

La arquitectura desarrollada se presenta a cotinuación:

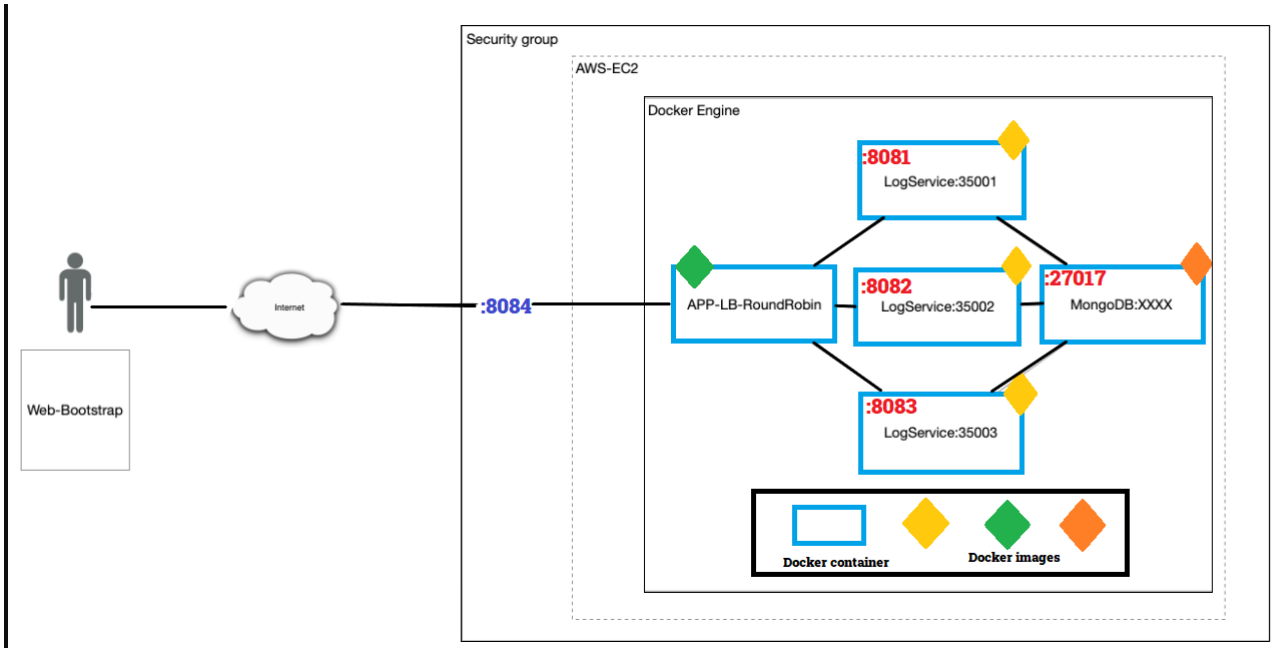

Figura 1: Diseño y Arquitectura

Para entender el diagrama es necesario explicar cuales son las funciones de cada componente y que servicios se ofrecen y se consumen.

- **Security Group Box:** Este componente se encarga de permitir el tráfico entrante y saliente de la EC2.Para esta practica el unico abierto para entrada y salida es el **8084** y a través de este puerto se haran las consultas al balanceador de carga.
- **AWS-EC2:** Máquina virtual almacenada en AWS. En este caso usamos una instancia con sistema operativo Linux.

- **Docker Engine:** Este componente describe la conexion y comportamiento de los contenedores e imagenes almacenados en la máquina. Aqui se almacenan los contenedores(recuadro azul) e imagenes necesarios para el funcionamiento del despliegue.Las imagenes que aqui residen (señaladas con una figura de rombo de colores amarillo verde y anaranjado) son las siguientes:
  - **Verde:** Imagen del servidor balanceador de carga (Implementación de Round Robin con spark Framework).
  - **Amarillo:** Imagen del servidor que recibirá las peticiones con spark Framework. En el diagrama se encuentran 3 contenedores con el mismo color de rombo, esto es debido a que se crean 3 instancias distintas con la misma imagen.
  - **Anaranjado:** Imagen del motor de base de datos NoSQL MongoDB

  Las imagenes verde y amarilla fueron subidas a DockerHub con la implementación respectiva en maven y la imagen de MongoDB fue tomada directamente de la plataforma. Cada imagen tiene sus respectivos contenedores corriendo por los puertos descritos en el diagrama(8084 para el contenedor de balanceador de carga, 8081-8083 para los contenedores de los servidores que antienden las peticiones y 27017 para el contenedor del mtor de base de datos).

3. Implementación

Se realizó la implementación de el balanceador de carga y el servidor que atiende las peticiones en 2 proyectos en java y maven separados, a cada uno de estos proyectos se les agrego su respectivo Dockerfile, se crearon las imagenes y se subieron a **dockerhub** (Link servidor Web / Link Balanceador de carga).
Una vez subidos a DockerHub, en la máquina EC2 de AWS se realizó la respetiva instalación de docker y se ejecutó el docker-compose (docker-compose up -d –scale web=3) para crear los contenedores de las imagenes mencionadas anteriormente y crear y ejecutar la imagen y contenedor del Motor de base de datos Mongo DB.

Figura 2: Ejecución de docker-compose para crear los contenedores

Los contenedores subieron exitosamente y se pueden encontrar ejecutando el comando docker ps, a continuación se presentan los contenedores corriendo por los puertos especificados en el diagrama de arquitectura.

```
[ec2-user@ip-172-31-46-36 LAB5-AREP]$ docker ps
CONTAINER ID        IMAGE                          COMMAND                  CREATED             STATUS              PORTS                     NAMES
d5edcffd5eff        andresmarcelo7/arep5web        "java -cp ./classes:…"   19 hours ago        Up 52 seconds       0.0.0.0:8083->6000/tcp    lab5-
arep_web_3
7a8bb3a3b8d9        andresmarcelo7/arep5web        "java -cp ./classes:…"   19 hours ago        Up 53 seconds       0.0.0.0:8082->6000/tcp    lab5-
arep_web_2
b711d1a90554        andresmarcelo7/arep5web        "java -cp ./classes:…"   19 hours ago        Up 53 seconds       0.0.0.0:8081->6000/tcp    lab5-
arep_web_1
babb0bd591d4        andresmarcelo7/arep5balancer   "java -cp ./classes:…"   19 hours ago        Up 55 seconds       0.0.0.0:8084->6000/tcp    lab5-
arep_balancer_1
668bfed1f1fa        mongo:latest                   "docker-entrypoint.s…"   19 hours ago        Up 55 seconds       0.0.0.0:27017->27017/tcp  db
[ec2-user@ip-172-31-46-36 LAB5-AREP]$
```

Figura 3: Información de los contenedores en ejecución

El grupo de seguridad de la EC2 se configuró especificamente para que sólo el puerto 8084(que pertenece al balanceador de carga) recibiera el trafico de entrada y de salida de peticiones como se muestra a continuación.

Figura 4: Grupo de seguridad de EC2

4. Pruebas

A continuación se presenta evidencia del funcionamiento de la arquitectura. desplegada en AWS:

4.1. Peticion POST al endpoint de base de datos

Como se mencionó anteriormente las peticiones son distribuidas a uno de los 3 servidores disponibles por medio de un balanceador de carga con algoritmo de Round Robin. Para evidenciar en qué servidor se procesó la peticion se decidió agregar un número(que indica el numero del servidor) al texto que el usuario envió.

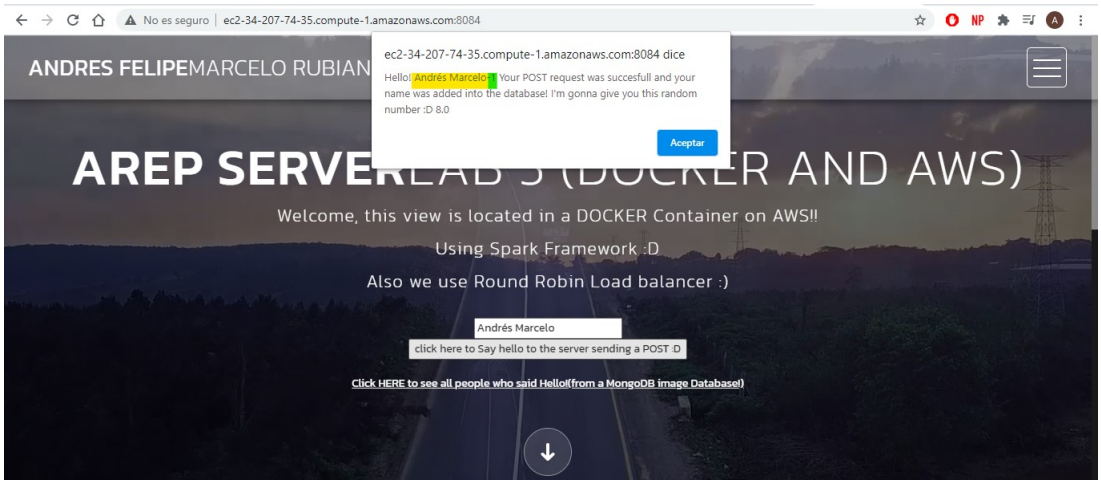

Figura 5: POST request al endpoint /testPost del servidor indicado por el balanceador de carga (Servidor 1)

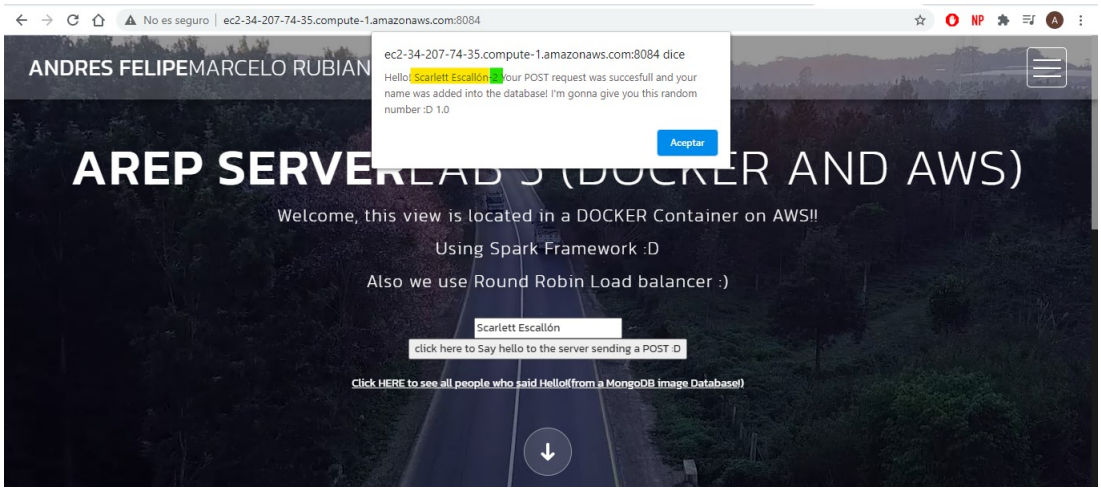

Figura 6: POST request al endpoint /testPost del servidor indicado por el balanceador de carga (Servidor 2)

4.2. Petición GET al endpoint de base de datos

finalmente haremos un GET request a la base de datos MongoDB para mostrar los datos almacenados anteriormente con los nombres ingresados,la fecha de ingreso y el servidor que procesó la request. Cabe resaltar también este tipo de consultas a la base de datos **tambien pasan por el balanceador de carga** pero no se evidencian al usuario.

Figura 7: GET request al endpoint /testDB (Comunicación con Base de datos) a uno de los servidores

5. Conclusión

Esta actividad permitió conocer una arquitectura implementada con docker y desplegada en una maquina virtual de AWS, se entendió el funcionamiento, comportamiento y metodología de las imágenes y contenedores con Docker y tambien se fortalecieron los conocimientos de despliegue y configuración de máquinas virtuales en la nube.