

# Taller de modularización Docker y a AWS

Calderón Ortega Andrés Mateo

Arquitecturas Empresariales

Ingeniería de sistemas, Escuela Colombiana de Ingeniería Julio Garavito,  
Bogotá, Colombia

12 de marzo de 2021

**Resumen:** En este documento se realizará la explicación de la arquitectura utilizada para el desarrollo de un aplicativo web el cual fue construido usando Java, Docker y Maven, además se explicara la forma en que se decidió construir este aplicativo y los componentes que este posee, además del desarrollo guiado por una programación orientada a objetos, y la forma en como este fue desplegado tanto de manera local y como fue desplegado en AWS.

## 1. Introducción

El problema tratado en este proyecto es la implementación de un aplicativo web que sea fácilmente desplegado tanto de manera local como en un servidor (AWS) este debe ser trabajado en Docker (contenedor), además de esto debe utilizar un algoritmo para balanceo de carga como lo es el algoritmo RoundRobin, debido a que deben existir 3 contenedores iguales en funcionamiento que deben ser capaces de realizar inserciones a una base de datos como lo es la base de datos MongoDB y debe retornar como respuesta la consulta de los últimos 10 registros que se tengan de esta base de datos, y un último contenedor que deberá ser capaz de atender solicitudes web y realizar as peticiones correspondientes a los otros 3 contenedores haciendo uso del algoritmo de balanceo de carga, para este último contenedor debe existir una interfaz amigable para el usuario en la cual él ingrese un mensaje y en una tabla se muestren los registros de los últimos 10 con la fecha en la cual se registró.

Para este proyecto se utilizó la versión 8 de Java, adicionalmente fue construido haciendo uso del ambiente de desarrollo IntelliJ, para el manejo de dependencias se utilizó Maven y por último se utilizó Docker para realizar la construcción de los contenedores de software; es de vital importancia que este se encuentre instalado en la máquina en la cual se va a utilizar este proyecto.

## 2. Arquitectura

Para la solución del problema creamos 5 contenedores, uno para almacenar la base de datos y su conexión a MongoDB, otros 3 que serán los encargados de conectarse a la base de datos y retornar la respuesta de los ultimos 10 registros con fecha, y el último el cual implementa el algoritmo de balanceo de cargas y dará soporte web para los usuarios.

En el proyecto base de los 3 contenedores, utilizamos el framework el cual tendrá una única función de aceptar peticiones post al path '/insertar' y desde ahí se comunicara con la clase MongoDBConnection la cual se encargara de realizar la conexión con la base de datos y manejar las peticiones de 'insertMensaje' y 'consultarMensajes', una para insertar el mensaje enviado por el body de la petición post y la otra para consultar los últimos 10 mensajes respectivamente, los datos que este recibe por medio del body de la petición es un objeto de tipo JSON y a sí mismo el resultado de la petición es del mismo tipo.

Para el proyecto base del contenedor único encargado del balanceo, igualmente se utiliza el framework spark, el cual contendrá un método post

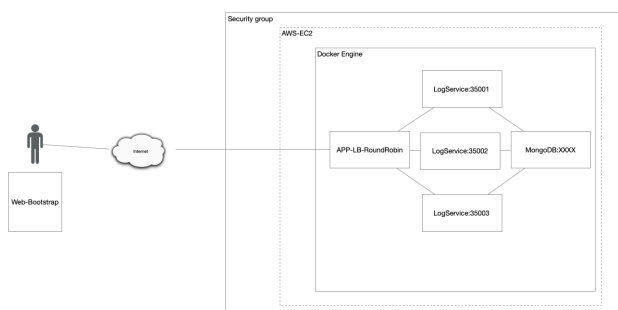


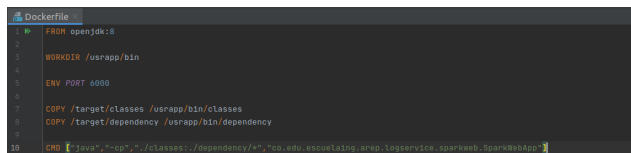
Figura 1: Arquitectura Del Problema

por el path 'insertarMensajes' en el cual mandara el body de la petición a la clase LogServiceAPI, en la cual a través del método 'roundRobinGetPort' obtendrá el puerto del servidor al cual va a enviar la solicitud, y posteriormente hará la solicitud post al contenedor correspondiente, para la solución de cuál red utilizar para realizar la petición se utiliza la red '172.17.0.1' que es la red por defecto que se instala al instalar Docker.

### 3. Construcción y Despliegue

#### 3.1. Construcción en Docker

Para la construcción de docker cada uno de los proyectos debe de tener un archivo de tipo Dockerfile, el cual contendrá una configuración básica de en qué lenguaje va a correr (JAVA), porque puerto, que debe copiar de donde se está ejecutando, el comando que se va a ejecutar finalmente cuando este se inicie.



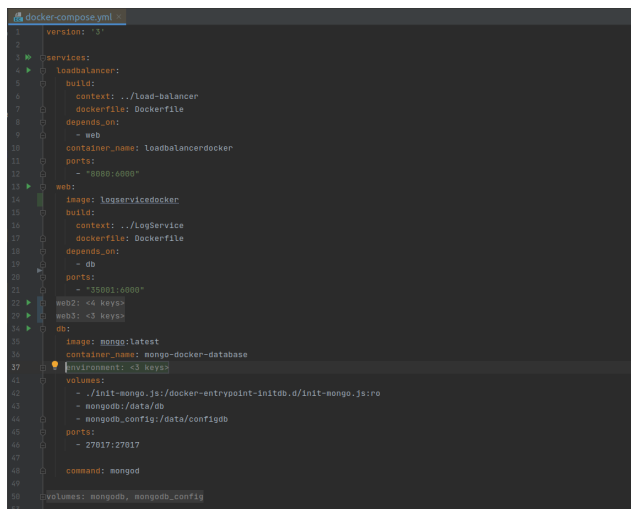
```

FROM openjdk:8
WORKDIR /usrapp/bin
ENV PORT 6000
COPY /target/classes /usrapp/bin/classes
COPY /target/dependency /usrapp/bin/dependency
CMD java -cp "/usrapp/bin/classes:/usrapp/bin/dependency/*" com.example.demo.LogServiceAPI

```

Figura 2: Archivo Dockerfile

Para realizar una automatización de la construcción de estos contenedores utilizamos un archivo llamado 'docker-compose.yml' el cual tendrá la configuración para cada uno de los contenedores indicando el archivo Dockerfile de cada uno y los nuevos puertos indicados para los mismos.



```

version: '3'
services:
  loadbalancer:
    build:
      context: ../load-balancer
      dockerfile: Dockerfile
    depends_on:
      - web
    container_name: loadbalancerdocker
    ports:
      - "8080:8080"
  web:
    image: logservicedocker
    build:
      context: ../LogService
      dockerfile: Dockerfile
    depends_on:
      - db
    ports:
      - "35001:8080"
  mongo:
    image: mongo:latest
    container_name: mongo-docker-database
    environment:
      <3 keys
    volumes:
      - ./init-mongo.js:/docker-entrypoint-initdb.d/init-mongo.js:ro
      - mongodb:/data/db
      - mongodb_config:/data/configdb
    ports:
      - 27017:27017
    command: mongod
    volumes:
      - mongodb, mongodb_config

```

Figura 3: Archivo docker-compose.yml

#### 3.2. Despliegue en AWS

Para el despliegue con AWS se utilizó una máquina virtual EC2, en la cual se instaló Docker y se realizaron las configuraciones correspondientes para que al momento de apagar e iniciar nuevamente la máquina esta inicie el servicio de Docker e inicie los contenedores de manera automática, además de configurar las políticas de seguridad para que permita el flujo por el puerto designado para el contenedor que brindara el soporte al usuario, las imágenes Docker fueron subidas a un repositorio en DockerHub para poderlas descargar en la máquina virtual de AWS para su posterior puesta en funcionamiento.

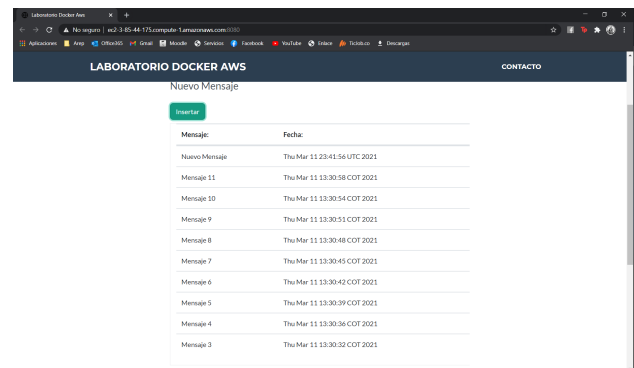


Figura 4: Despliegue en AWS

### 4. Conclusiones

Se adquirieron conocimientos en la utilización de Docker y en su forma de operar, adicionalmente se pudo visualizar e interiorizar el gran nivel de importancia que llegan a tener algoritmos de balanceo de cargas debido a las grandes cantidades de datos que se pueden llegar a manejar en minutos, además podemos ver la gran utilidad que se tiene al utilizar AWS debido a la facilidad en la creación de máquinas virtuales y su velocidad de procesamiento en las mismas, además de la facilidad de integración entre repositorios Docker como DockerHub y nuestras máquinas locales y las máquinas en AWS, lo que permite un ambiente de desarrollo fácil de manejar y acoplable. Finalmente se vio la utilidad de frameworks como Spark al momento de desarrollar debido a que permite despliegues rápidos y fácil de usar, con código limpio y entendible.

## Referencias

- [1] Algoritmo de Planificación Round Robin. (2021). Retrieved 12 March 2021, from <https://blogcitchia.wordpress.com/2017/04/05/algoritmo-de-planificacion-round-robin/>
- [2] Overview of Docker Compose. (2021). Retrieved 12 March 2021, from <https://docs.docker.com/compose/>
- [3] Do a Simple HTTP Request in Java — Baeldung. (2021). Retrieved 12 March 2021, from <https://www.baeldung.com/java-http-request>
- [4] Daniel Benavides. 2021. Clase de laboratorio. [26 Febrero 2021].