**Diego Fernando Rodríguez Torres.**

**TALLER DE INTRODUCCIÓN A VIRTUALIZACIÓN Y PROG. DISTRIBUIDA.**

***Resumen:***

En este proyecto se crea una aplicación en Java que permite recibir del usuario una cadena de caracteres a través de un formulario web. Usando un balanceo “RoundRobin”, esta petición es enviada a alguna de las tres instancias de un servicio REST que se encarga de almacenar la cadena de caracteres en una base de datos Mongo y a su vez devuelve las últimas 10 cadenas guardadas en formato JSON.

Para el desarrollo de este proyecto se crearon tres imágenes:

**1) dockerlogfacade:** está el cliente web que por medio de un form (html) toma la cadena de caracteres y después realiza un balanceo RoundRobin con el objetivo de enviar la cadena a una instancia del servicio “logservice”, el cual se encargará de almacenar la cadena en la base de datos. En su construcción se utilizó el framework Spark java.

- Para la construcción de esta imagen se usó Docker en Windows y se creó a partir del siguiente dockerfile:

FROM openjdk:8

WORKDIR /usrapp/bin

ENV PORT 4000

COPY /target/classes /usrapp/bin/classes

COPY /target/dependency /usrapp/bin/dependency

CMD [“java”,”-cp”,”./classes:./dependency/\*”,”co.edu.escuelaing.logfacaderoundrobin.LogFacadeService”]

- Comando Docker para construir la imagen:

tros\Maestria Informatica\Arquitectura y Gobernabilidad\javatutorial\logfacaderoundrobin> docker build --tag dockerlogfacade .

[+] Building 3.1s (10/10) FINISHED

=> [internal] load build definition from Dockerfile 0.0s

=> => transferring dockerfile: 291B 0.0s

=> [internal] load .dockerignore 0.0s

=> => transferring context: 2B 0.0s

=> [internal] load metadata for docker.io/library/openjdk:8 2.2s

=> [auth] library/openjdk:pull token for registry-1.docker.io 0.0s

=> [internal] load build context 0.4s

=> => transferring context: 3.11MB 0.4s

=> [1/4] FROM docker.io/library/openjdk:8@sha256:9d233cf187b7022a8f285ed7c162b0a87624de1dbe09dcc499bfab83c22a66d5 0.0s

=> CACHED [2/4] WORKDIR /usrapp/bin 0.0s

=> [3/4] COPY /target/classes /usrapp/bin/classes 0.1s

=> [4/4] COPY /target/dependency /usrapp/bin/dependency 0.1s

=> exporting to image 0.1s

=> => exporting layers 0.1s

=> => writing image sha256:89e0f8f08406b315b72611fdb319db0218a9d35e6a031a36ac5300dc4ff106f0 0.0s

=> => naming to docker.io/library/dockerlogfacade

**2) dockersparklogservice:** este es el servicio “logservice” que se encarga de recibir la cadena de caracteres del “logfacade” y la almacena en la base de datos mongo. De igual forma se encarga de obtener los 10 últimos registros de la base de datos y los muestra en formato JSON. Se crea la base de datos “tallerdb” y la colección “mensajesusuario” donde es almacenada la cadena de caracteres. En su construcción se utilizó mongo-java-driver y Spark java.

Para la construcción de esta imagen se usó Docker en Windows y se creó a partir del siguiente dockerfile:

FROM openjdk:8

WORKDIR /usrapp/bin

ENV PORT 4569

COPY /target/classes /usrapp/bin/classes

COPY /target/dependency /usrapp/bin/dependency

CMD [“java”,”-cp”,”./classes:./dependency/\*”,”co.edu.escuelaing.log\_service.LogService”]

- Comando para construir la imagen:

docker build --tag dockersparklogservice .

**Mongo:3.6.1:** la base de datos Mongodb que se ejecuta en el Puerto 27017, allí se almacena la cadena de caracteres y la fecha en que se envió. Se basa en el archivo docker-compose.yml del taller.

Después de verificar el funcionamiento de toda la aplicación en el equipo local se procedió a cargar las imágenes a DockerHUB.

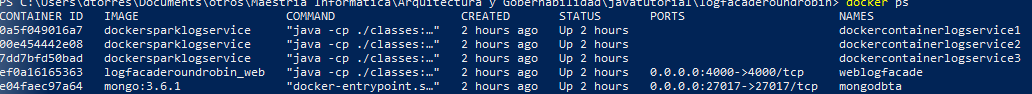


Figure 1 Contenedores ejecutándose en equipo local.

Graphical user interface, text, application, email

Description automatically generated

Figure 2 Mensaje en cliente web.

Graphical user interface, text, application, email

Description automatically generated

Figure 3 Respuesta mensaje almacenado.

Imágenes en DockerHUB:

Dockerlogfacade 🡪 TAG Latest.

Dockersparklogservice🡪TAG v1

Mongo:3.6.1🡪 TAG v2

A screenshot of a computer

Description automatically generated

Figure 4 Imágenes en DockerHub

***Proceso de despliegue en AWS***

Primero se crea una máquina virtual Linux en AWS:

Graphical user interface, text, application

Description automatically generated

Figure 5 Instancia AWS.

Se realiza una actualización, se instala Docker y se inicia este servicio:

[ec2-user@ip-172-31-82-210 ~]$ sudo yum update -y

Loaded plugins: extras\_suggestions, langpacks, priorities, update-motd

Resolving Dependencies

--> Running transaction check

---> Package amazon-linux-extras.noarch 0:2.0.0-1.amzn2 will be updated

---> Package amazon-linux-extras.noarch 0:2.0.1-1.amzn2 will be an update…..

[ec2-user@ip-172-31-82-210 ~]$ sudo yum install docker

Loaded plugins: extras\_suggestions, langpacks, priorities, update-motd

Resolving Dependencies

--> Running transaction check

---> Package docker.x86\_64 0:20.10.7-3.amzn2 will be installed……

Antes de crear las instancias de los contenedores Docker se debe crear una red para que estos se puedan comunicar entre sí, se crea la red “redtaller”

[ec2-user@ip-172-31-82-210 ~]$ docker network create redtaller

421e8d8e776aa456ac49a5820a7f7112c665deec610b94977d144baa4f6b410b

[ec2-user@ip-172-31-82-210 ~]$

[ec2-user@ip-172-31-82-210 ~]$ docker network ls

NETWORK ID NAME DRIVER SCOPE

b96e5b535f3d bridge bridge local

34b11b1f4b7d host host local

88d333a8fb98 none null local

421e8d8e776a redtaller bridge local

[ec2-user@ip-172-31-82-210 ~]$

A partir de las imágenes en Dockerhub se crean las instancias de los contenedores Docker y se asocian a la red anteriormente creada. Es importante que se mantengan los nombres de los hosts ya que en el código de la aplicación están configurados con ese nombre. De igual forma se debe configurar una variable de entorno llamada “PORT” con el valor “4569” con el fin que el servicio “logservice” se inicie en ese puerto.

export PORT="4569"

Graphical user interface

Description automatically generated

Figure 6 Instancias contenedores Docker en VM AWS.

Imágenes de prueba del servicio:

Graphical user interface, text, application

Description automatically generated

Figure 7 Cliente Web en la VM AWS

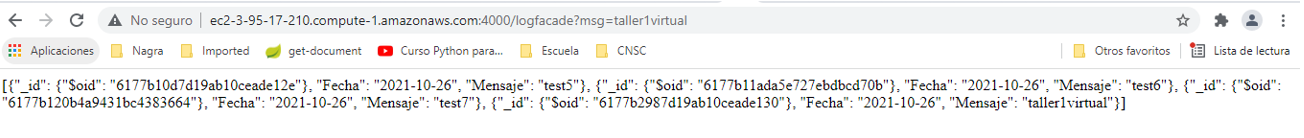


Figure 8 Respuesta listado de mensajes en BD.

A picture containing timeline

Description automatically generated

Figure 9 Logs servicio “logfacade” se observa el balanceo Roundrobin.

Text

Description automatically generated with low confidence

Figure 10 Logs servicios “logservice3” y “logservice2”

A picture containing text

Description automatically generated

Figure 11 logs servicio “logservice1”.

Text

Description automatically generated with medium confidence

Figure 12 Mensajes guardados en la base de Datos.