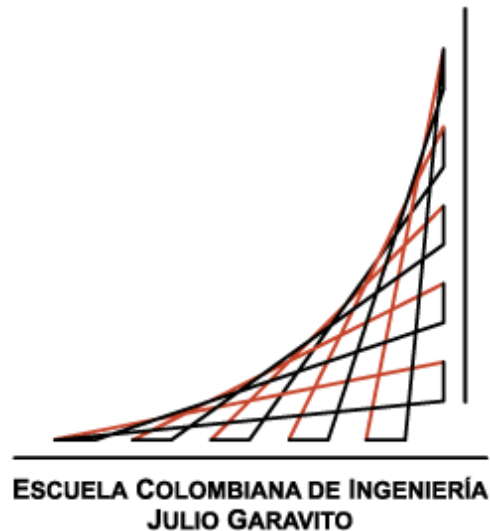


Taller 5 Arem

Fernando Barrera Barrera

Luis Daniel Benavides Navarro

Arquitecturas Empresariales



Índice

1. Introduccion	2
2. Conceptos Basicos	2
3. Diseño	2
4. Pre-Requisitos	3
5. Montaje en Intancia EC2 AWS	3
6. Ejecucion y Pruebas	4
7. Conclusion	6
8. Bibliografía	7

1. Introduccion

Este taller fue hecho para comprender la gran utilidad de los dockers ,ademas entender el funcionamiento de los balanceadores round robin y como estos distribuyen equitativamente las cargas de trabajo,como tambien entender como se puede montar una arquitectura de servicios contenidos en dokers y desplegarlos en un ambiente cloud en este caso Aws.

2. Conceptos Basicos

- **Maven:** Herramienta de software dedicada a la estructutracion y construccion de proyectos java. [4]
- **Git:** software de control de versiones de proyectos [2]
- **Java :** es un lenguaje de programacion orientado a objetos que se desarrollo en los años 90
- **Docker:** Es un contenedor que permite empaquetar todos los recursos necesarios para el despliegue de una aplicacion en cualquier entorno [3]
- **EC2 AWS:** Es una maquina virtual que o recurso de infraestructura virtualizada que ofrece AWS [1]
- **Mongo DB :** Una base de datos no relacional que usa documentos JSON para el almacenmiento datos [5]

3. Diseño

En este taller cuenta con tres log services,cada uno almacenados en dockers independientes en los puertos 8001,8002y 8003 y estos log services se encargan de insertar los nuevos logs y consulta los logs registrados en la base de datos que se encuentra en otro docker en el puerto 27017.

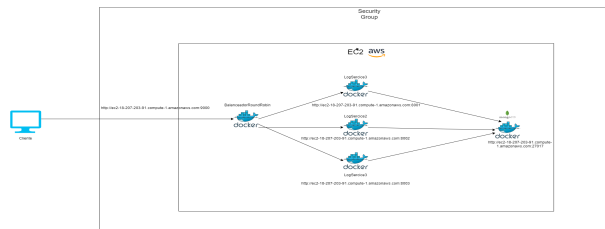


Figura 1:

Los log services se encargan de seleccionar los ultimos 10 logs insertados y se los retorna al docker que contiene el balancedador round robin en el puertuerto 9000 y este balancedador se encarga de balancear las cargas de trabajo de los log services equitativamente , ya que despues de cada operacion ya sea de consulta o de insercion cambia a otro log service y asi sucesivamente va enviadoles peticiones a los tres log services y por ultimo se llevaron los 5 dockers a una instancia ec2 aws donde se evidencia su correcto funcionamiento y se verifica que los puertos donde se despliega cada docker este abierto en el security group de la instancia ec2.

Reglas de entrada				
Tipo	Protocolo	Intervalo de puertos	Origen	Descripción: opcional
TCP personalizado	TCP	9000	0.0.0.0/0	Balanceador
SSH	TCP	22	0.0.0.0/0	-
TCP personalizado	TCP	27017	0.0.0.0/0	Mongo
TCP personalizado	TCP	8001 - 8003	0.0.0.0/0	LogService

Figura 2:

4. Pre-Requisitos

- Git
- Java
- Maven
- Docker

5. Montaje en Instancia EC2 AWS

Para realizar el montaje de nuestra arquitectura docker en aws primero accedemos a nuestra instancia ec2 por medio de ssh que ya debe tener el puerto 8001,8002,8003,9000 y 27017 abiertos y primero descargamos la imagen de mongo para crear el docker

```
C:\Users\jfm_14\Downloads\ssh -i "AREP-FERNANDO.pem" ec2-user@ec2-18-207-203-91.compute-1.amazonaws.com
Last login: Tue Sep 22 02:22:13 2020 from 186.119.234.170

Amazon Linux AMI

https://aws.amazon.com/amazon-linux-ami/2018.03-release-notes/
[ec2-user@ip-172-31-82-37 ~]$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS
[ec2-user@ip-172-31-82-37 ~]$ docker pull mongo:3.6.1
3.6.1: Pulling from library/mongo
c4bb02b17bb4: Pull complete
af5a0bb3b0a: Pull complete
a229fb575afe: Pull complete
b9d22af6e206: Pull complete
f86d162a317: Pull complete
72a67f0b4d: Pull complete
0a6a10f10008: Pull complete
b9cad17017cd: Pull complete
Digest: sha256:c7846db0b310fe2ed300105a720123f3365a33e5aade6f5539922dd7c75dc
Status: Downloaded newer image for mongo:3.6.1
[ec2-user@ip-172-31-82-37 ~]$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
mongo                3.6.1              1208574c8af9       2 years ago        360MB
[ec2-user@ip-172-31-82-37 ~]$
```

Figura 3:

Luego procedemos a crear la docker con una instancia de la base de datos mongoDB y el docker se llamara db y se ejecutara por el puerto 27017

```
[ec2-user@ip-172-31-82-37 ~]$ docker run -d -p 27017:27017 --name db mongo
Unable to find image 'mongo:latest' locally
latest: Pulling from library/mongo
5d821c54847: Pull complete
a510eae5d8c: Pull complete
a40eb0b9f40: Pull complete
1242b40c4ff: Pull complete
6a0a740f6cf: Pull complete
c6d02aab224: Pull complete
8d0c2ff7f392: Pull complete
a20b09c404c: Pull complete
8d0e7c1260e: Pull complete
77f31d60b382: Pull complete
e328c4807a2: Pull complete
1b05c57744e: Pull complete
Digest: sha256:ebcd042054d9974c3c310d761b0db3955115448242da15101c1246db0af
Status: Downloaded newer image for mongo:latest
9828153dc0d1841390af1ebfacc2ff07a886abff3f6593ce330cab70e27aca3
[ec2-user@ip-172-31-82-37 ~]$
```

Figura 4:

Una vez el docker con la instancia de la base de datos mongoDB este ejecutando entramos al cliente mongo para crear la base de datos arem y la coleccion logs que sera donde los log services realizaran consultas e inserciones

```
[ec2-user@ip-172-31-82-37 ~]$ docker exec -it db mongo
MongoDB shell version v4.4.1
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session session ( "id" : UUID("a517baf-3546-4908-950c-d6d071b30d7") )
MongoDB server version: 4.4.1
Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
https://docs.mongodb.com/
Questions? Try the MongoDB Developer Community Forums
https://community.mongodb.com
The server generated these startup warnings when booting:
2020-09-22T05:17:42.170+00:00: Using the XFS filesystem is strongly recommended with the WiredTiger storage engine. See http://dochub.mongodb.org/core/products/filesystem
2020-09-22T05:17:42.007+00:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
2020-09-22T05:17:42.007+00:00: Soft rlimits too low
2020-09-22T05:17:42.007+00:00: currentvalue: 1024
2020-09-22T05:17:42.007+00:00: recommendedvalue: 64000
...
Enable MongoDB's free cloud-based monitoring service, which will then receive and display
metrics about your deployment (DB, OS, operation statistics, etc).
The monitoring data will be available on a MongoDB website with a unique URL accessible to you
and anyone you share the URL with. MongoDB may use this information to make product
improvements and to suggest MongoDB products and deployment options to you.
```

Figura 5:

Ahora vamos a desplegar el Docker del balanceador haciendo uso de la imagen fernando15/balanceadorroundrobin y lo pondremos a ejecutar en el puerto 9000 con el comando docker run

```

ec2-user@ip-172-31-82-37 ~]$ docker run -d -p 9000:6000 --name balaceador fernando15/balaceadorroundrobin
Unable to find image 'fernando15/balaceadorroundrobin:latest' locally
latest: Pulling from fernando15/balaceadorroundrobin
57d9f1af1ad8: Pull complete
71e126169501: Pull complete
1af28a55c3f3: Pull complete
83f1c932170: Pull complete
881a7aaf13: Pull complete
2b6a1d457bc3: Pull complete
b7fe4ef88350: Pull complete
4eb70f125864: Pull complete
2b6fcdaf664: Pull complete
4c29af04aa31: Pull complete
Digest: sha256:d49f3256a8c45d1ca7933ed145761a67fc86c65b9f89e2d48c20def5ee1e2c
Status: Downloaded newer image for fernando15/balaceadorroundrobin:latest
f0aa8ccc384e54d3d4d3959362e17201751b0ccab1ebb3ac4d5fbc4f8fcd1
ec2-user@ip-172-31-82-37 ~]$

```

Figura 6:

Ahora vamos a desplegar el docker con el primer log service haciendo uso de la imagen fernando15/logservice y lo llamaremos logservice1 y lo pondremos a ejecutar en el puerto 8001 con el comando docker run

```

ec2-user@ip-172-31-82-37 ~]$ docker run -d -p 8001:6000 --name logservice1 fernando15/logservice
Unable to find image 'fernando15/logservice:latest' locally
latest: Pulling from fernando15/logservice
57d9f1af1ad8: Already exists
71e126169501: Already exists
1af28a55c3f3: Already exists
83f1c932170: Already exists
881a7aaf13: Already exists
2b6a1d457bc3: Already exists
b7fe4ef88350: Already exists
ae1b93365604: Pull complete
7c6a072b1050: Pull complete
32d70077289: Pull complete
Digest: sha256:9921e13758652a28b9e7ac541cfe49465732787a3cbb8a2bds67e8dde68385
Status: Downloaded newer image for fernando15/logservice:latest
e08570280152a6f093db627b685eaf05d5e8d44efee81ab57c1c526ed10ad
ec2-user@ip-172-31-82-37 ~]$

```

Figura 7:

Para los dockers del log service 3 y 2 se hara tambien ncon el comando docker run en los puertos 8002 y 8003 respectivamente, con la imagen fernando15/logservice que se descargo para la creacion del docker del log service 1

```

ec2-user@ip-172-31-82-37 ~]$ docker run -d -p 8002:6000 --name logservice2 fernando15/logservice
2b335ace01ba52954524056d903ea701c945b67c994e2bd95b1c48d70722672d
ec2-user@ip-172-31-82-37 ~]$ docker run -d -p 8003:6000 --name logservice3 fernando15/logservice
066d8d337886d65c5ba215428946fcd2faa18e6e0a0d2304f14785878b5fd42
ec2-user@ip-172-31-82-37 ~]$

```

Figura 8:

Ahora con el comando docker ps podemos ver que ya tenemos nuestros 5 dockers con sus respectivos servicios corriendo en la instancia ec2 de aws

```

ec2-user@ip-172-31-82-37 ~]$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED              STATUS              PORTS
b66dd4337886       fernando15/logservice  "java -cp ./classes:..." 31 seconds ago      Up 38 seconds      0.0.0.0:8003->6000/tcp
e08570280152       fernando15/logservice  "java -cp ./classes:..." 45 seconds ago      Up 44 seconds      0.0.0.0:8002->6000/tcp
logservice2        fernando15/logservice  "java -cp ./classes:..." About a minute ago   Up About a minute   0.0.0.0:8001->6000/tcp
logservice3        fernando15/logservice  "java -cp ./classes:..." 2 minutes ago       Up 2 minutes       0.0.0.0:9000->6000/tcp
f0aa8ccc384         fernando15/balaceadorroundrobin "java -cp ./classes:..." 3 minutes ago       Up 3 minutes       0.0.0.0:27017->27017/tcp
mongo              mongo                 "docker-entrypoint.sh..." 3 minutes ago       Up 3 minutes       0.0.0.0:27017->27017/tcp

```

Figura 9:

6. Ejecucion y Pruebas

Para ejecutar probar el funcionamiento de los servicios contenidos en dockers en la instancia ec2 simplemente accedemos con la direccion ip de la instancia ec2 y el puerto 9000 que corresponde al servicio de balanceador que recibe los datos y balancea la cargas de trabajo



Figura 10:

Al entrar podemos ver que ya hay tres logs en la base de datos asi que vamos a insertar mas hasta llegar a la insercion del log numero 11 asi se vera la tabla de logs



Figura 11:

Ahora al insertar el log 11 se vera que este sera agregado al final de la tabla y primer log desaparecera ya que los log service estan programados solo para retornar los 10 ultimos logs registrados en la base de datos como se vera a continuacion



Figura 12:

Podemos tambien acceder a un log service con la ip de la instancia ec2 y el puerto 8001 que corresponde al puerto del log service 1 y vemos que este lo que le retorna al balanceador es un archivo json con string separados por comas que corresponden a los mensajes y fechas de los 10 ultimos log registrados en la base de datos



Figura 13:

Tambien podemos ver todos los logs registrados en la base de datos mongoDB accediendo al cliente y verificando los datos insertados en la coleccion logs de la base de datos arem

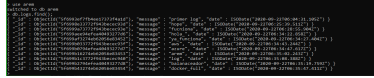


Figura 14:

Para probar el correcto funcionamiento del balanceador en la pagina de entrada hay una linea de texto que nos dice en que log service estamos actualmente es decir ellog service que realiza la ultima operacion en este caso de consulta vemos que estamos en log service 3 es decir ellogservice 3 fue el que hizo la consulta a la base datos y se lo retorno al balanceador



Figura 15:

Ahora vamos a insertar un nuevo mensaje y dado que el logservice3 fue el que hizo la ultima operacion ,el logservice 1 se encargara de adicionar el nuevo registro a la base de datos y el logservice2 se encargara de realizar otra consulta para actualizar los datos de la tabla con el nuevo registro por esa razon despues de la insercion, el balanceador nos indica que nos encontramos en el logservice2 que fue el que realizo la ultima operacion



Figura 16:

7. Conclusion

Este taller fue util para comprender la gran utilidad de los dockers y como podemos montar una arquitectura basica de servicios usando dockers como tambien comprender la importancia del balanceador de cargas para distribuir las cargas de trabajo de forma mas equitativa y se mejore la optimizacion del flujo de operaciones del flujo de trabajo de la arquitectura de servicios.

8. Bibliografía

- [1] AWS. *EC2*. URL: <https://aws.amazon.com/es/ec2/instance-types/>. (entered: 2020).
- [2] Código Facilito. *Git*. URL: <https://codigofacilito.com/articulos/que-es-git>. (entered: 16-08-2015).
- [3] techtarget. *Docker*. URL: <https://searchdatacenter.techtarget.com/es/definicion/Docker>. (entered: 2020).
- [4] Wikipedia. *Maven*. URL: <https://es.wikipedia.org/wiki/Maven>. (entered: 31-03-2020).
- [5] wikipedia. *MongoDB*. URL: <https://es.wikipedia.org/wiki/MongoDB>. (entered: 27-08-2020).