



ESCUELA COLOMBIANA DE INGENIERIA JULIO GARATIVO

**TALLER DE INTRODUCCIÓN A VIRTUALIZACIÓN Y PROGRAMACION
DISTRIBUIDA**

2020

Ian Sebastián Martínez Rey

Arquitectura y gobernabilidad Tecnológica

Maestría en informática

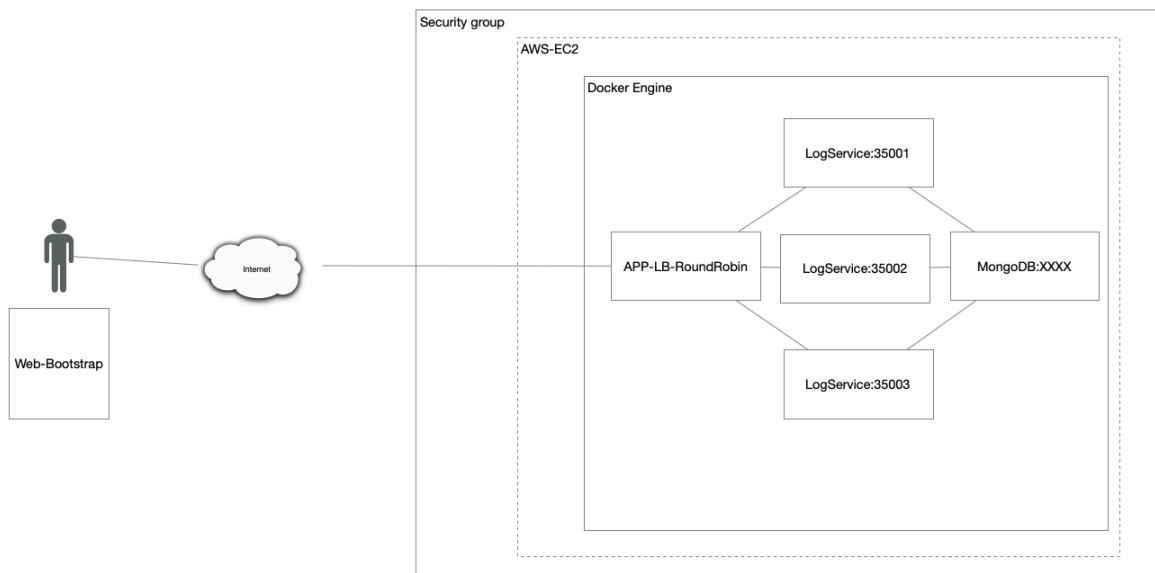
INDICE

- 1. Objetivo**
- 2. Desarrollo actividad**
 - a. Creación aplicación Backend**
 - b. Creación contenedores Web y Base de Datos**
 - c. Creación contenedor balanceador de carga**
 - d. Cargue de imágenes a Docker Hub**
 - e. Creación instancia en AWS**
 - f. Despliegues contenedores en AWS**
- 3. Conclusiones**

OBJETIVO

Crear una aplicación web pequeña usando el micro-framework de Spark java (<http://sparkjava.com/>). Una vez creada esta aplicación construir los contenedores de Docker requeridos para realizar el despliegue en un ambiente local y en una máquina virtual de AWS

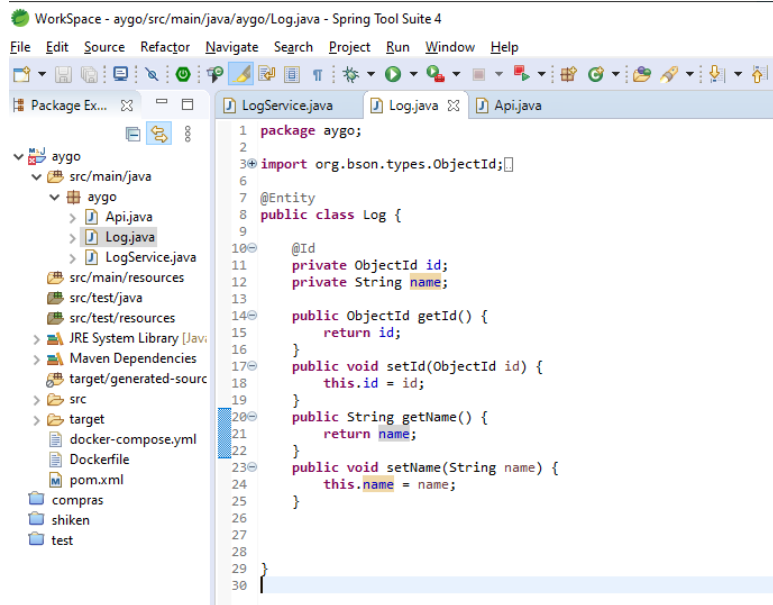
Arquitectura de la aplicación



DESARROLLO ACTIVIDAD

Creación aplicación Backend

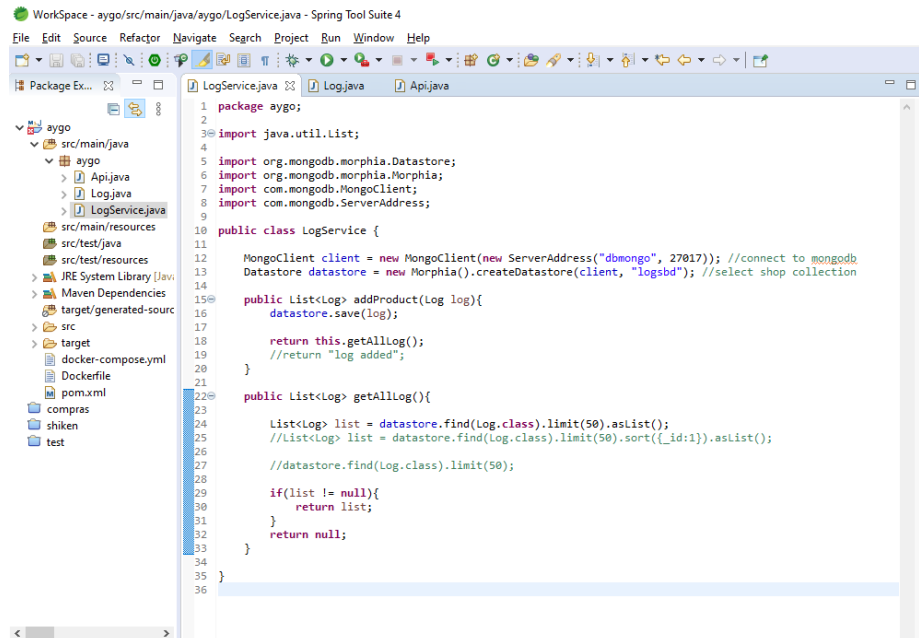
1. Creamos la entidad que define la entidad Log



The screenshot shows the Spring Tool Suite 4 IDE with the 'Log.java' file open. The file is located in the package 'aygo' under the path 'src/main/java/aygo'. The code defines an entity class 'Log' with an '@Id' annotation and a 'name' attribute. The class has methods for getting and setting the ID and name.

```
1 package aygo;
2
3 import org.bson.types.ObjectId;
4
5
6
7 @Entity
8 public class Log {
9
10     @Id
11     private ObjectId id;
12     private String name;
13
14     public ObjectId getId() {
15         return id;
16     }
17     public void setId(ObjectId id) {
18         this.id = id;
19     }
20     public String getName() {
21         return name;
22     }
23     public void setName(String name) {
24         this.name = name;
25     }
26
27
28
29 }
30
```

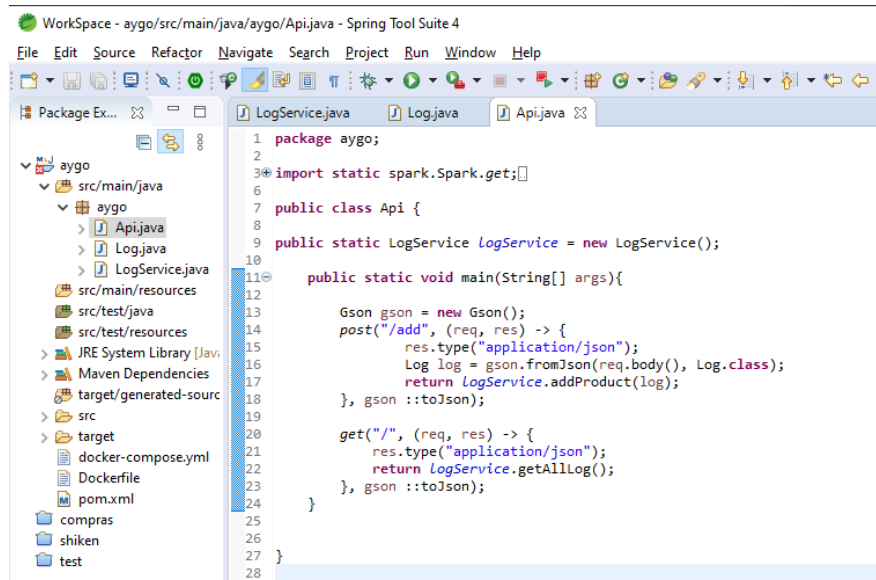
2. Se crean los servicios que se van a asociar a la entidad



The screenshot shows the Spring Tool Suite 4 IDE with the 'LogService.java' file open. The file is located in the package 'aygo' under the path 'src/main/java/aygo'. The code defines a service class 'LogService' that interacts with a MongoDB database. It includes imports for 'java.util.List', 'org.mongodb.morphia.Datastore', 'org.mongodb.morphia.Morphia', 'com.mongodb.MongoClient', and 'com.mongodb.ServerAddress'. The class has methods for adding a product and getting all logs.

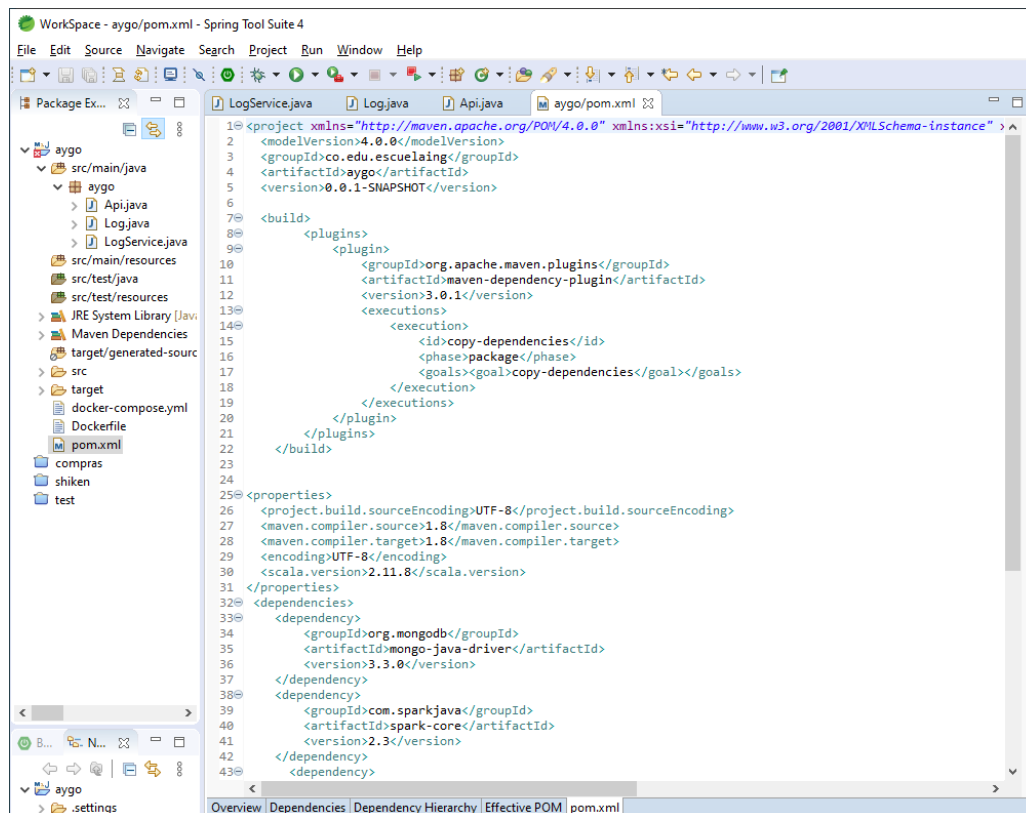
```
1 package aygo;
2
3 import java.util.List;
4
5 import org.mongodb.morphia.Datastore;
6 import org.mongodb.morphia.Morphia;
7 import com.mongodb.MongoClient;
8 import com.mongodb.ServerAddress;
9
10 public class LogService {
11
12     MongoClient client = new MongoClient(new ServerAddress("dbmongo", 27817)); //connect to mongodb
13     Datastore datastore = new Morphia().createDatastore(client, "logsbd"); //select shop collection
14
15     public List<Log> addProduct(Log log){
16         datastore.save(log);
17         return this.getAllLog();
18         //return "log added";
19     }
20
21
22     public List<Log> getAllLog(){
23
24         List<Log> list = datastore.find(Log.class).limit(50).asList();
25         //List<Log> list = datastore.find(Log.class).limit(50).sort(new org.mongodb.morphia.Sort("_id", true)).asList();
26         //datastore.find(Log.class).limit(50);
27
28         if(list != null){
29             return list;
30         }
31         return null;
32     }
33
34
35 }
36
```

3. Se definen los servicios Rest con los que se va a trabajar por medio de la clase Api



```
1 package aygo;
2
3 import static spark.Spark.get;
4
5
6
7 public class Api {
8
9     public static LogService logService = new LogService();
10
11     public static void main(String[] args){
12
13         Gson gson = new Gson();
14         post("/add", (req, res) -> {
15             res.type("application/json");
16             Log log = gson.fromJson(req.body(), Log.class);
17             return logService.addProduct(log);
18         }, gson :: toJson);
19
20         get("/", (req, res) -> {
21             res.type("application/json");
22             return logService.getAllLog();
23         }, gson :: toJson);
24     }
25
26
27
28 }
```

4. En el archivo pom.xml especificamos que todas las dependencias deben copiarse a la carpeta target.



```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" >
3     <modelVersion>4.0.0</modelVersion>
4     <groupId>co.edu.escuelaing</groupId>
5     <artifactId>aygo</artifactId>
6     <version>0.0.1-SNAPSHOT</version>
7
8     <build>
9         <plugins>
10             <plugin>
11                 <groupId>org.apache.maven.plugins</groupId>
12                 <artifactId>maven-dependency-plugin</artifactId>
13                 <version>3.0.1</version>
14                 <executions>
15                     <execution>
16                         <id>copy-dependencies</id>
17                         <phase>package</phase>
18                         <goals><goal>copy-dependencies</goal></goals>
19                     </execution>
20                 </executions>
21             </plugin>
22         </plugins>
23     </build>
24
25     <properties>
26         <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
27         <maven.compiler.source>1.8</maven.compiler.source>
28         <maven.compiler.target>1.8</maven.compiler.target>
29         <encoding>UTF-8</encoding>
30         <scala.version>2.11.8</scala.version>
31     </properties>
32
33     <dependencies>
34         <dependency>
35             <groupId>org.mongodb</groupId>
36             <artifactId>mongo-java-driver</artifactId>
37             <version>3.3.0</version>
38         </dependency>
39         <dependency>
40             <groupId>com.sparkjava</groupId>
41             <artifactId>spark-core</artifactId>
42             <version>2.3</version>
43         </dependency>
44     </dependencies>
45 </project>
```

5. Verificamos el funcionamiento

GET

← → ↻ ⓘ localhost:8080

```
[{"id":{"timestamp":1601499254,"machineIdentifier":812076,"processIdentifier":6816,"counter":10144036},"name":"test 3"}]
```

POST

Untitled Request Comments 0

POST http://localhost:8080/add Send Save

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings Cookies Code

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL Text

```
1 {
2   "name": "test 8"
3 }
4
```

Body Cookies Headers (6) Test Results Status: 200 OK Time: 63 ms Size: 1011 B Save Response

Pretty Raw Preview Visualize JSON ≡ 🔍

```
1
2 {
3   "id": {
4     "timestamp": 1601499254,
5     "machineIdentifier": 812076,
6     "processIdentifier": 6816,
7     "counter": 10144036
8   },
9   "name": "test 3"
10 },
11 {
12   "id": {
13     "timestamp": 1601506993,
14     "machineIdentifier": 1141188,
15     "processIdentifier": 1,
16     "counter": 1120724
17   },
18   "name": "test 4"
19 },
```

Creación contenedores Web y Base de Datos

1. Una vez verificado procedemos con la creación de los archivos docker-compose.yml y dockerfile

docker-compose.yml

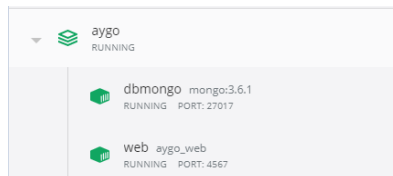
```
1  version: '2'
2
3  services:
4    web:
5      build:
6        context: .
7        dockerfile: Dockerfile
8      container_name: web
9      ports:
10       - "4567:4567"
11
12    dbmongo:
13      image: mongo:3.6.1
14      container_name: dbmongo
15      volumes:
16       - mongodb:/data/db
17       - mongodb_config:/data/configdb
18      ports:
19       - 27017:27017
20      command: mongod
21
22  volumes:
23    mongodb:
24    mongodb_config:
25
26  networks:
27    default:
28      external:
29        name: backend
```

dockerfile

```
1  FROM openjdk:8
2
3  WORKDIR /usrapp/bin
4
5  ENV PORT 6000
6
7  COPY /target/classes /usrapp/bin/classes
8  COPY /target/dependency /usrapp/bin/dependency
9
10 CMD ["java", "-cp", "./classes:./dependency/*", "aygo.Api"]
```

2. Finalizado el proceso se procede con la creación de los contenedores con la instrucción

docker-compose up -d



3. Luego de esto creamos 3 nuevos contenedores basados en la imagen
 - `docker run -d -p 34000:4567 --name firstdockercontainer --expose 34000 --network backend --link dbmongo:dbmongo aygo_web`
 - `docker run -d -p 34001:4567 --name firstdockercontainer2 --expose 34001 --network backend --link dbmongo:dbmongo aygo_web`
 - `docker run -d -p 34002:4567 --name firstdockercontainer3 --expose 34002 --network backend --link dbmongo:dbmongo aygo_web`

Creación contenedor del balanceador de carga

El siguiente paso es crear un contenedor que realizara el balanceo de carga entre los contenedores que acabamos de crear.

1. Creamos el archivo de configuración de nginx

```
1 user nginx;
2 worker_processes auto;
3 error_log /var/log/nginx/error.log warn;
4 pid /var/run/nginx.pid;
5
6 events {
7     worker_connections 1024;
8 }
9
10 http {
11
12     upstream backend {
13         server firstdockercontainer:34000;
14         server firstdockercontainer3:34002;
15         server firstdockercontainer2:34001;
16     }
17
18     server {
19         listen 8080;
20         server_name localhost 127.0.0.1;
21
22         location / {
23             proxy_pass http://backend/;
24             add_header Access-Control-Allow-Origin *;
25             proxy_set_header X-Forwarded-For $remote_addr;
26         }
27     }
28 }
```

2. Luego creamos el archivo dockerfile

```
1 FROM nginx
2
3 COPY conf/nginx.conf /etc/nginx/nginx.conf
4
5 EXPOSE 8080
```

3. Por último, creamos la imagen del balanceador de carga con el siguiente comando

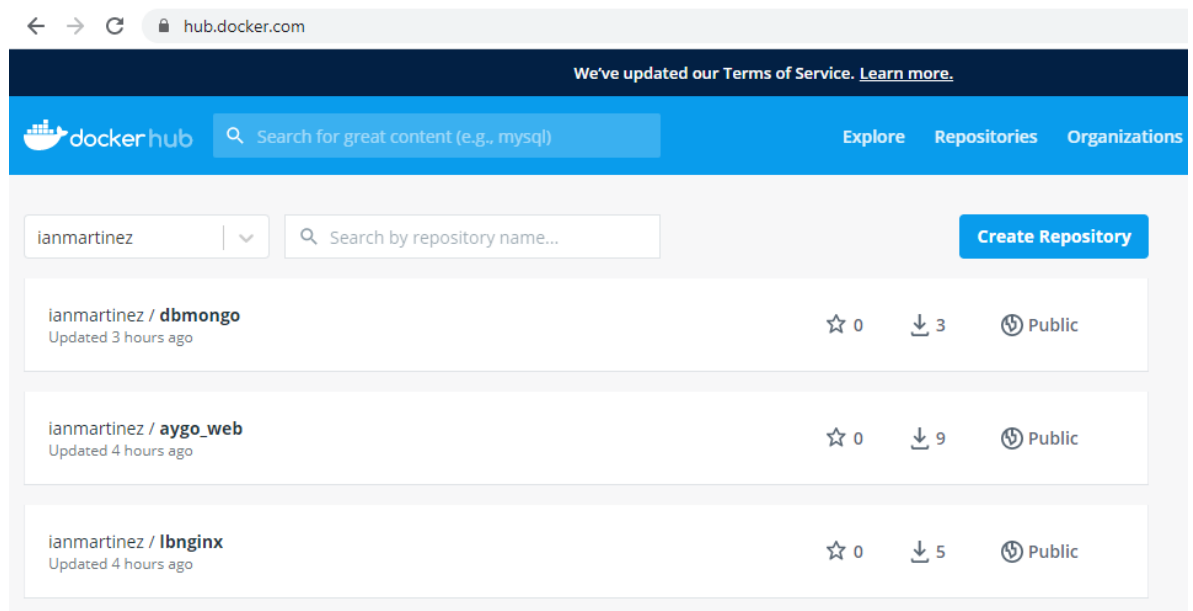
`docker run -d -p 8080:8080 --name lbnginx --expose 8080 --network backend lbnginx`

4. Con el comando docker ps, vemos los 5 contenedores activos

```
D:\Java Projects\Workspace\aygo>docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS                               NAMES
cb6f4a239fa74  apnginx       "/docker-entrypoint..." 3 hours ago   Up 3 hours   0.0.0.0:80->80/tcp                 apnginx
f90c906232aa  lbnginx       "/docker-entrypoint..." 4 hours ago   Up 4 hours   80/tcp, 0.0.0.0:8080->8080/tcp     lbnginx
4aadfd325249  aygo_web     "java -cp ./classes:..." 4 hours ago   Up 4 hours   34002/tcp, 0.0.0.0:34002->4567/tcp  firstdockercontainer3
df000eaacc23  aygo_web     "java -cp ./classes:..." 4 hours ago   Up 4 hours   34001/tcp, 0.0.0.0:34001->4567/tcp  firstdockercontainer2
ab00e2d25c14  aygo_web     "java -cp ./classes:..." 5 hours ago   Up 5 hours   34000/tcp, 0.0.0.0:34000->4567/tcp  firstdockercontainer
1a1dccc33dc   mongo:3.6.1  "docker-entrypoint.s..." 23 hours ago   Up 23 hours   0.0.0.0:27017->27017/tcp          dbmongo
1095147bbe21  aygo_web     "java -cp ./classes:..." 23 hours ago   Up 23 hours   0.0.0.0:4567->4567/tcp            web
```


Cargue de imágenes a Docker Hub

1. Realizamos la creación de cada imagen con el comando docker build
 - `docker build -t aygo_web .`
 - `docker build -t dbmongo .`
 - `docker build -t lbnginx .`
2. Colocamos un tag a cada imagen para poderlas identificar en el repositorio de docker hub
 - `docker tag aygo_web ianmartinez/aygo_web`
 - `docker tag aygo_web ianmartinez/dbmongo`
 - `docker tag aygo_web ianmartinez/lbnginx`
3. Con el comando `docker login` ingresamos a nuestra cuenta para cargar las imágenes
4. Con la instrucción `docker push` cargamos cada imagen al repositorio
 - `docker push ianmartinez/aygo_web:latest`
 - `docker push ianmartinez/dbmongo:latest`
 - `docker push ianmartinez/lbnginx:latest`
5. Una vez completado el cargue podemos visualizar los repositorios en la página de Docker hub



Creación instancia en AWS

1. En el módulo de EC2 de AWS, creamos una nueva instancia con sistema operativo Amazon Linux

Step 1: Choose an Amazon Machine Image (AMI)

[Cancel and Exit](#)

An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. You can select an AMI provided by AWS, our use community, or the AWS Marketplace; or you can select one of your own AMIs.

[Search by Systems Manager parameter](#)

Quick Start

My AMIs

AWS Marketplace

Community AMIs

Amazon Linux

Free tier eligible

Amazon Linux 2 AMI (HVM), SSD Volume Type - ami-03657b56516ab7912 (64-bit x86) / ami-023b120e01f4779c1 (64-bit Arm)

Amazon Linux 2 comes with five years support. It provides Linux kernel 4.14 tuned for optimal performance on Amazon EC2, systemd 219, GCC 7.3, Glibc 2.26, Binutils 2.29.1, and the latest software packages through extras.

Root device type: ebs Virtualization type: hvm ENA Enabled: Yes

Select

☒ 64-bit (x86)

☐ 64-bit (Arm)

2. Seleccionamos la instancia t2.micro que se encuentra dentro del paquete gratuito

Step 2: Choose an Instance Type

Amazon EC2 provides a wide selection of instance types optimized to fit different use cases. Instances are virtual servers that can run applications. They have varying combinations of CPU, memory, storage, and networking capacity, and give you the flexibility to choose the appropriate mix of resources for your applications. [Learn more](#) about instance types and how they can meet your computing needs.

Filter by: All instance types Current generation Show/Hide Columns

Currently selected: t2.micro (Variable ECUs, 1 vCPUs, 2.5 GHz, Intel Xeon Family, 1 GiB memory, EBS only)

	Family	Type	vCPUs	Memory (GiB)	Instance Storage (GB)	EBS-Optimized Available	Network Performance	IPv6 Support
<input type="checkbox"/>	General purpose	t2.nano	1	0.5	EBS only	-	Low to Moderate	Yes
<input checked="" type="checkbox"/>	General purpose	t2.micro Free tier eligible	1	1	EBS only	-	Low to Moderate	Yes

3. Una vez creada la instancia procedemos a abrir los puertos que se van a exponer

EC2 > Grupos de seguridad > sg-007de8d6067eae036 - launch-wizard-1

sg-007de8d6067eae036 - launch-wizard-1

Eliminar grupo de seguridad

Copiar al nuevo grupo de seguridad

Detalles

Nombre del grupo de seguridad launch-wizard-1	ID del grupo de seguridad sg-007de8d6067eae036	Descripción launch-wizard-1 created 2020-10-01T15:59:16.575-05:00	ID de la VPC vpc-c807f3a3
Propietario 531697871639	Número de reglas de entrada 7 Entradas de permisos	Número de reglas de salida 1 Entrada de permiso	

Reglas de entrada

Reglas de salida

Etiquetas

Reglas de entrada

Editar reglas de entrada

Tipo	Protocolo	Intervalo de puertos	Origen	Descripción: opcional
TCP personalizado	TCP	8080	0.0.0.0/0	Puerto Balanceador Nginx
SSH	TCP	22	0.0.0.0/0	-

Despliegues contenedores en AWS

1. Ingresamos a la instancia con las llaves creadas

```
ec2-user@ip-172-31-17-120:~  
D:\AWS>ssh -i "ismr.pem" ec2-user@ec2-3-19-227-132.us-east-2.compute.amazonaws.com  
Last login: Thu Oct 1 21:17:55 2020 from dynamic-186-154-122-155.dynamic.etb.net.co  
  
  _ |  _ |  _ )  
 _ | ( _ | /  
 _ | \ _ | _ |   Amazon Linux 2 AMI  
  
https://aws.amazon.com/amazon-linux-2/  
  
  _ |  _ |  _ )  
 _ | ( _ | /  
 _ | \ _ | _ |   Amazon Linux 2 AMI  
  
https://aws.amazon.com/amazon-linux-2/  
[ec2-user@ip-172-31-17-120 ~]$
```

- ## 2. Instalamos docker por medio de la instrucción

- yum install Docker

- ### 3. Creamos la red donde se colocarán los contenedores

- `docker network create --driver bridge backend`

```
[ec2-user@ip-172-31-17-120 ~]$ docker network create --driver bridge backend
9f3d9b44687501a8b6453707efb8c5c91e957e401d199d091f27560fb83872ac
[ec2-user@ip-172-31-17-120 ~]$
```

- #### 4. Contenedor mongo

```
docker run -d -p 27017:27017 --name dbmongo --network backend mongo:3.6.1
```

```
[ec2-user@ip-172-31-17-120 ~]$ docker run -d -p 27017:27017 --name dbmongo mongo:3.6.1
Unable to find image 'mongo:3.6.1' locally
3.6.1: Pulling from library/mongo
c4bb02b17bb4: Pull complete
3f58e3bb3be4: Pull complete
a229fb575a6e: Pull complete
8f5ddc533743: Pull complete
5e9d2af6e206: Pull complete
3b6c28c0235b: Pull complete
fd6b165aa317: Pull complete
772467f0b4cd: Pull complete
a94d919fbb86: Pull complete
b0cad17917cd: Pull complete
Digest: sha256:c78f6debfb5b10fe2ed390105a729123f3365a33e5aada6f5539922d1d7c75dc
Status: Downloaded newer image for mongo:3.6.1
f34598a4cc97eb2d6c70a489a8c7add6ef8941278da76dc9c982dea7f5bb60e7
[ec2-user@ip-172-31-17-120 ~]$
```

- ## 5. Contenedor web

```
docker run -d -p 4567:4567 --name web --network backend ianmartinez/aygo_web
```

```
[ec2-user@ip-172-31-17-120 ~]$ docker run -d -p 4567:4567 --name web --network backend ianmartinez/aygo_web
Unable to find image 'ianmartinez/aygo_web:latest' locally
latest: Pulling from ianmartinez/aygo_web
57df1a1f1ad8: Pull complete
71e126169501: Pull complete
1af28a55c3f3: Pull complete
03f1c9932170: Pull complete
881ad7aafb13: Pull complete
db6a1d457bc3: Pull complete
b7fe4ef08350: Pull complete
d5936be1adf2: Pull complete
78ae984fa100: Pull complete
b3b750a867ab: Pull complete
Digest: sha256:dccfc53f908fbc593b60ac554533cddbb345b39e139cdf8932c102bf6fa5ec2f
Status: Downloaded newer image for ianmartinez/aygo_web:latest
e2214c277648954f2a76b73d4592060cbea26cef5d56ba80bab77155ffa84205
[ec2-user@ip-172-31-17-120 ~]$
```

6. Despliegue contenedores basados en la imagen aygo_web
 - a. `docker run -d -p 34000:4567 --name firstdockercontainer --expose 34000 --network backend ianmartinez/aygo_web`
 - b. `docker run -d -p 34001:4567 --name firstdockercontainer2 --expose 34001 --network backend ianmartinez/aygo_web`
 - c. `docker run -d -p 34002:4567 --name firstdockercontainer3 --expose 34002 --network backend ianmartinez/aygo_web`

```
[ec2-user@ip-172-31-17-120 ~]$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
9581e15fda2d	ianmartinez/aygo_web	"java -cp ./classes:..."	9 seconds ago	Up 8 seconds	34002/tcp, 0.0.0.0:34002->4567/tcp	firstdockercontainer3
6337f521fd3	ianmartinez/aygo_web	"java -cp ./classes:..."	21 seconds ago	Up 20 seconds	34001/tcp, 0.0.0.0:34001->4567/tcp	firstdockercontainer2
837efbe828e6	ianmartinez/aygo_web	"java -cp ./classes:..."	About a minute ago	Up About a minute	34000/tcp, 0.0.0.0:34000->4567/tcp	firstdockercontainer
e2214c277648	ianmartinez/aygo_web	"java -cp ./classes:..."	2 minutes ago	Up 2 minutes	0.0.0.0:4567->4567/tcp	web
3250dd362c1b	mongo:3.6.1	"docker-entrypoint.s..."	3 minutes ago	Up 3 minutes	0.0.0.0:27017->27017/tcp	dbmongo

```
[ec2-user@ip-172-31-17-120 ~]$
```

7. Se realiza el despliegue del balanceador de carga nginx

`docker run -d -p 8080:8080 --name lbnginx --expose 8080 --network backend ianmartinez/lbnginx`

```
[ec2-user@ip-172-31-17-120 ~]$ docker run -d -p 8080:8080 --name lbnginx --expose 8080 --network backend ianmartinez/lbnginx
Unable to find image 'ianmartinez/lbnginx:latest' locally
latest: Pulling from ianmartinez/lbnginx
d121f8d1c412: Pull complete
ebd81fc8c071: Pull complete
655316c160af: Pull complete
d15953c0e0f8: Pull complete
2ee525c3c3cc: Pull complete
f95e88d68097: Pull complete
Digest: sha256:7aa37136906e6bd661ea9926835bdf06127dbb92ff91f23ab4c33a961e0a343e
Status: Downloaded newer image for ianmartinez/lbnginx:latest
9e470aacf218bf1b04aadd4dc58123724ae53f95cfbce44c266f890ecf01ce0d
[ec2-user@ip-172-31-17-120 ~]$
```

8. Finalmente observamos todos los contenedores desplegados completando así el despliegue de la arquitectura

```
ec2-user@ip-172-31-17-120:~$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
9e470aacf218	ianmartinez/lbnginx	"docker-entrypoint..."	27 seconds ago	Up 26 seconds	80/tcp, 0.0.0.0:8080->8080/tcp	lbnginx
9581e15fda2d	ianmartinez/aygo_web	"java -cp ./classes:..."	About a minute ago	Up About a minute	34002/tcp, 0.0.0.0:34002->4567/tcp	firstdockercontainer3
6337f521fd3	ianmartinez/aygo_web	"java -cp ./classes:..."	About a minute ago	Up About a minute	34001/tcp, 0.0.0.0:34001->4567/tcp	firstdockercontainer2
837efbe828e6	ianmartinez/aygo_web	"java -cp ./classes:..."	2 minutes ago	Up 2 minutes	34000/tcp, 0.0.0.0:34000->4567/tcp	firstdockercontainer
e2214c277648	ianmartinez/aygo_web	"java -cp ./classes:..."	3 minutes ago	Up 3 minutes	0.0.0.0:4567->4567/tcp	web
3250dd362c1b	mongo:3.6.1	"docker-entrypoint.s..."	4 minutes ago	Up 4 minutes	0.0.0.0:27017->27017/tcp	dbmongo

```
[ec2-user@ip-172-31-17-120 ~]$
```

9. Visualizamos la aplicación desde el navegador

← → × ⓘ ec2-3-19-227-132.us-east-2.compute.amazonaws.com:8080

```
[{"id":{"timestamp":1601499254,"machineIdentifier":812076,"processIdentifier":6816,"counter":10144036},"name":"test 3"}, {"id":{"timestamp":1601506993,"machineIdentifier":1141188,"processIdentifier":1,"counter":1120724},"name":"test 4"}, {"id":{"timestamp":1601507692,"machineIdentifier":1141188,"processIdentifier":1,"counter":11160268},"name":"test 5"}, {"id":{"timestamp":1601511861,"machineIdentifier":7135301,"processIdentifier":1,"counter":2121948},"name":"test 6"}, {"id":{"timestamp":1601583621,"machineIdentifier":7135301,"processIdentifier":1,"counter":2121949},"name":"test 6"}, {"id":{"timestamp":1601584109,"machineIdentifier":7135301,"processIdentifier":1,"counter":2121950},"name":"test 7"}]
```

10. Se realiza la prueba del servicio desde un cliente web, donde se envía la cadena test 8 para lo cual el servidor devuelve una lista con los registros agregados incluyendo el enviado

<http://190.144.21.226:8000/>

← → ↻ No es seguro | 190.144.21.226:8000

NUEVAS TENDENCIAS EN ARQUITECTURA EMPRESARIAL

Taller de Introducción a Virtualización y prog. distribuida

Descripcion Actividad

El taller consiste en crear una aplicación web pequeña usando el micro-framework de Spark java (<http://sparkjava.com/>). Una vez tengamos esta aplicación procederemos a construir un container para docker para la aplicación y los desplegaremos y configuraremos en nuestra máquina local. Luego, cerremos un repositorio en DockerHub y subiremos la imagen al repositorio. Finalmente, crearemos una máquina virtual de en AWS, instalaremos Docker , y desplegaremos el contenedor que acabamos de crear.

Ejecucion Actividad

Formulario

String

test 8

Enviar

Repuesta Servidor

```
[{"timestamp":1601506993,"machineIdentifier":1141188,"processIdentifier":1,"counter":1120724,"name":"test 4"}, {"timestamp":1601507692,"machineIdentifier":1141188,"processIdentifier":1,"counter":11160268,"name":"test 5"}, {"timestamp":1601511861,"machineIdentifier":7135301,"processIdentifier":1,"counter":2121948,"name":"test 6"}, {"timestamp":1601583621,"machineIdentifier":7135301,"processIdentifier":1,"counter":2121949,"name":"test 6"}, {"timestamp":1601584109,"machineIdentifier":7135301,"processIdentifier":1,"counter":2121950,"name":"test 7"}, {"timestamp":1601597619,"machineIdentifier":7135301,"processIdentifier":1,"counter":2121951,"name":"test 8"}]
```

CONCLUSIONES

El taller desarrollado es una clara representación del modelo de escalamiento horizontal actual utilizado por las diferentes compañías, el cual aprovecha de la mejor manera despliegue y funcionalidades provistas por Docker.

Podemos otra parte podemos evidenciar que por medio del despliegue de varios contenedores con el mismo código y un balanceador de carga, se logra atender un mayor número de peticiones, aumentar la seguridad e incrementar o reducir la infraestructura de la aplicación por demanda.