

## Documentación - Tecnologías del lado del servidor.

### Cloud computing

#### 1. Introducción

En este documento se van a explicar las operaciones realizadas para configurar el entorno Cloud con Amazon Web Services (AWS), visto durante la asignatura y desde el que se ha podido distribuir la aplicación de API REST, desarrollada con Play Framework.

También se van a exponer los motivos por los que se han tomado ciertas decisiones a la hora de configurar los servicios Cloud, con tal de mejorar aspectos que afectan a nuestra aplicación, tan importantes como la disponibilidad, escalabilidad, tolerancia a errores, etc.

#### 2. Principales operaciones realizadas

##### 2.1. Despliegue de instancia EC2

En primer lugar, para la preparación del entorno que va a dar servicio a nuestra aplicación, vamos a repasar los pasos seguidos para su puesta en marcha de forma manual.

Empezamos creando una instancia utilizando el servicio EC2 que nos ofrece AWS. La configuración y prestaciones de la de la máquina son:

- Ubuntu Server 20.04 LTS 64bits
- Tipo de instancia: t2.micro. Dispone de 1 core y 1 gb de RAM
- 8gb de almacenamiento
- Con un *security group*: que nos permite configurar los permisos de conexión
- El paquete equivale al *ami*: ami-0767046d1677be5a0
- Al final de proceso, generamos un par de claves y lo descargamos en un fichero “.pem” que nos permitirá conectarnos más adelante por ssh.

Estas prestaciones básicas son las que nos ofrece el paquete gratuito de Amazon, motivo por el que lo hemos elegido. Sin embargo, como veremos a continuación, será posible replicar la misma configuración en diferentes instancias.

Step 7: Review Instance Launch

**AMI Details**

Free tier eligible **Ubuntu Server 20.04 LTS (HVM), SSD Volume Type - ami-0767046d1677be5a0**

Ubuntu Server 20.04 LTS (HVM), EBS General Purpose (SSD) Volume Type. Support available from Canonical (http://www.ubuntu.com/cloud/services). Root Device Type: ebs Visualization type: hvm

**Instance Type**

Instance Type	ECUs	vCPUs	Memory (GiB)	Instance Storage (GiB)	EBS-Optimized Available	Network Performance
t2.micro	-	1	1	EBS only	-	Low to Moderate

**Security Groups**

Security group name: launch-sg-1  
Description: launch-sg-1 created 2021-03-04T19:13:47.390+01:00

Type	Protocol	Port Range	Source	Description
SSH	TCP	22	0.0.0.0/0	

**Instance Details**

**Storage**

Volume Type	Device	Snapshot	Size (GiB)	Volume Type	IOPS	Throughput (MB/s)	Delete on Termination	Encrypted
Root	/dev/sda1	snap-07c3e64b523cd...	8	gp2	100 / 3000	N/A	Yes	Not Encrypted

**Tags**

Cancel Previous **Launch**

Una vez finalizado el proceso, podemos ver nuestra primera instancia creada en el listado:

**Instances (1/1)**

Filter instances

search: i-0969ec35f1a56a785 Clear filters

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IP
EC2-mimo-play-recetas	i-0969ec35f1a56a785	Stopped	t2.micro	-	No alarms	eu-central-1b	-	-

**Instance: i-0969ec35f1a56a785 (EC2-mimo-play-recetas)**

**Details** Security Networking Storage Status checks Monitoring Tags

**Instance summary**

Instance ID: i-0969ec35f1a56a785 (EC2-mimo-play-recetas)  
Instance state: **Stopped**  
Instance type: t2.micro

**Instance details**

Platform: Ubuntu (inferred)  
AMI ID: ami-0952440befd74cd8a  
AMI name: ami-0952440befd74cd8a

Public IPv4 address: -  
Public IPv4 DNS: -  
Elastic IP addresses: -  
IAM Role: -

Private IPv4 addresses: 172.31.35.140  
Private IPv4 DNS: ip-172-31-35-140.eu-central-1.compute.internal  
VPC ID: vpc-92f54bf8  
Subnet ID: subnet-bb269c7  
Monitoring: disabled  
Termination protection: -

Antes de iniciarla y conectarnos vamos a habilitar el puerto HTTP 80 donde va a funcionar nuestra aplicación (por defecto), para no tener problemas de conexión en las pruebas más adelante. En este caso, habilitamos el tráfico sin restricción de IP (0.0.0.0/0), aunque hay que tener en cuenta que en un entorno de producción no sería la configuración correcta por seguridad.

EC2 > Security Groups > sg-0e1015f7627a9abbb - mimo-security-group > Edit inbound rules

**Edit inbound rules**

Inbound rules control the incoming traffic that's allowed to reach the instance.

**Inbound rules**

Type	Protocol	Port range	Source	Description - optional
HTTP	TCP	80	Custom 0.0.0.0/0	
SSH	TCP	22	Custom 0.0.0.0/0	

Add rule

NOTE: Any edits made on existing rules will result in the edited rule being deleted and a new rule created with the new details. This will cause traffic that depends on that rule to be dropped for a very brief period of time until the new rule can be created.

Cancel Preview changes **Save rules**

Hecho esto, pasamos a iniciarla desde el botón Start de “Instance state”. Transcurrido un tiempo, el estado pasará a conectado.

En este momento obtendremos la dirección DNS pública para conectarnos y configurar la máquina. Esta dirección es dinámica, por lo que cada vez que re arranquemos la instancia, nos generará una diferente.

Antes de conectarnos, debemos establecer permisos al fichero.pem de claves, descargado anteriormente. Abrimos una consola e introducimos el comando:

```
chmod 400 FICHERO.pem
```

Hecho esto, para conectarnos por ssh, tecleamos el comando con el formato:

```
ssh -i FICHERO.pem ubuntu@DNS_PUBLICA
```

En este caso es ubuntu@, porque es el sistema operativo seleccionado anteriormente.

Con esto ya deberíamos estar conectados a la máquina.

```
javier@DESKTOP-DRR04AP-MINGW64 /d/Usuarios/Javier/Escritorio/keysAWS
$ ssh -i javier-mimo-2021.pem ubuntu@ec2-52-59-206-116.eu-central-1.compute.amazonaws.com
The authenticity of host 'ec2-52-59-206-116.eu-central-1.compute.amazonaws.com (52.59.206.116)' can't be established.
ECDSA key fingerprint is SHA256:18Cnpl4j5RV/eICyljePdICJ7LoUFjqvqFeopjcopcs.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'ec2-52-59-206-116.eu-central-1.compute.amazonaws.com,52.59.206.116' (ECDSA) to the list of known hosts.
Welcome to Ubuntu 20.04.2 LTS (GNU/Linux 5.4.0-1038-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

System information as of Thu Mar  4 19:00:13 UTC 2021

System load:  0.08          Processes:      107
Usage of /:   34.9% of 7.69GB Users logged in:  0
Memory usage: 73%          IPv4 address for eth0: 172.31.35.140
Swap usage:   0%

 * Introducing self-healing high availability clusters in MicroK8s.
   Simple, hardened, Kubernetes for production, from RaspberryPi to DC.
   https://microk8s.io/high-availability

25 updates can be installed immediately.
0 of these updates are security updates.
To see these additional updates run: apt list --upgradable

Last login: Thu Mar  4 16:37:51 2021 from 84.120.166.135
ubuntu@ip-172-31-35-140:~$
```

## 2.2. Configuración de aplicación y RDS

Pasamos entonces a instalar las dependencias necesarias para que nuestra aplicación pueda ser ejecutada. En nuestro caso es una aplicación que funciona con java, así que actualizamos los paquetes del Ubuntu e instalamos las dependencias Java mediante:

```
sudo apt-get update
```

```
sudo apt install -y default-jdk
```

Una vez finalizado subimos nuestra aplicación generada con versión de *Release*. Para generar la *Release* en Play Framework, revisamos la configuración del fichero application.conf y ejecutamos en la carpeta raíz del proyecto, el comando: `sbt dist`

En este caso nuestro fichero de configuración tiene las siguientes propiedades definidas:

```
* application.conf X
D: > Usuarios > Javier > Escritorio > * application.conf
1  # This is the main configuration file for the application.
2  # https://www.playframework.com/documentation/latest/ConfigFile
3
4  ## Models
5  ebean.default = ["models.*"]
6
7  ## Internationalisation
8  play.i18n.langs = [ "en", "es" ]
```

*Las configuraciones aquí definidas son aquellas que he considerado esenciales para el funcionamiento de la lógica de aplicación y cuyos valores no conviene parametrizar. Aquellos ajustes parametrizables los veremos más adelante.*

El zip resultante de la Release lo subimos a nuestra instancia con el comando:

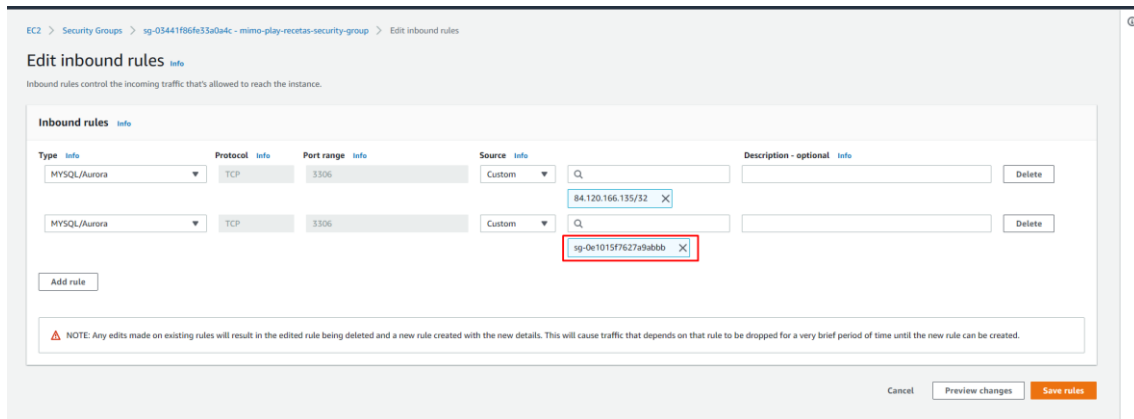
```
scp -i FICHERO.pem APLICACION.zip ubuntu@DNS_PUBLICA:/tmp
```

Volvemos a la consola con la conexión a la instancia y copiamos el fichero comprimido a la carpeta opt. Lo descomprimimos con: `unzip APLICACION.zip`, obteniendo la carpeta raíz con el contenido de la aplicación. En este caso hay que instalar previamente el gestor de descompresión: `sudo apt-get install unzip`

Antes de ponernos a configurar los parámetros de arranque, hay que crear la base de datos a la que vamos a conectar nuestra aplicación. Para ello, vamos al servicio RDS que nos ofrece AWS y pulsamos en Create Database. La configuración utilizada en este caso es:

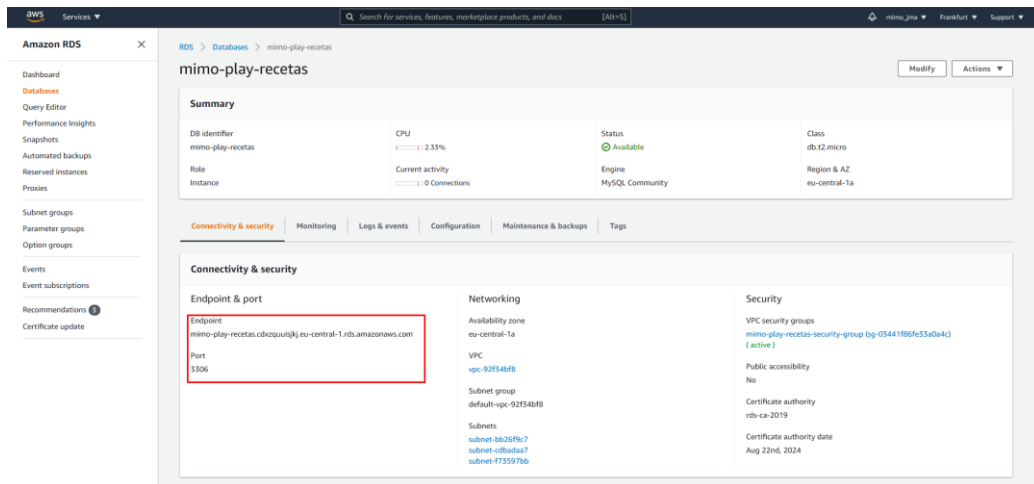
- Base de datos Mysql 8.0.20 Edición Community.
  - o Se ha seleccionado este motor de bases de datos por tener más soltura y experiencia que con el resto, aunque AWS nos ofrece otras opciones.
- Template: Free Tier
- Deshabilitar el autoescalado
- Crear y asociar un nuevo security group
- Deshabilitar el acceso público
- Ajustes de conexión (en este caso):
  - o DB instance identifier: *mimo\_play\_recetas*
  - o Username: *admin*
  - o Password: *admin123*

Una vez creada, vamos a modificar los permisos de su security group, con tal de habilitar la comunicación entre nuestra instancia y la base de datos:



Buscamos el identificador del security group de la instancia y la añadimos. En este caso, establecemos el puerto por defecto de MySQL, el 3306.

Hecho esto, ya podemos iniciar la base de datos y obtener el Endpoint y puerto de conexión para añadirla más adelante a nuestra configuración:



Pasamos por tanto a realizar a la configuración de arranque de la aplicación.

Accedemos de nuevo a la máquina, más concretamente a nuestra carpeta raíz de proyecto. Allí creamos el fichero que lanzará nuestra aplicación Play, con la configuración deseada. Se trata de un fichero .sh. Al editarlo, los parámetros de configuración aquí definidos han sido:

```
GNU nano 4.8 start.sh
# !/bin/sh
export JDBC_DATABASE_DRIVER=com.mysql.cj.jdbc.Driver
export JDBC_DATABASE_URL=jdbc:mysql://mimo-play-recetas.cdquuikj.eu-central-1.rds.amazonaws.com:3306/mimo_play_recetas
export JDBC_DATABASE_USERNAME=admin
export JDBC_DATABASE_PASSWORD=admin123

./bin/play-practica-final \
-Dhttp.port=80 \
-Dplay.http.secret.key='AmmvuVYIXRMLJ'<SIw7QrSwkCXyika`HC7ejgwAqplH>q?Bi;Qv0`8eosUca2D?^' \
-Dplay.evolutions.db.default.enabled=true \
-Dplay.evolutions.db.default.autoApply=true \
-Dplay.evolutions.db.default.autoApplyDowns=true \
-Dplay.filters.hosts.allowed.0='.' \
-Ddb.default.driver=${JDBC_DATABASE_DRIVER} \
-Ddb.default.url=${JDBC_DATABASE_URL} \
-Ddb.default.username=${JDBC_DATABASE_USERNAME} \
-Ddb.default.password=${JDBC_DATABASE_PASSWORD} \
-J-Xms128M \
-J-Xms256m \
-J-server \
```

Pasamos a describir que realizan estos comandos:

- El primer párrafo con los exports representa las variables de conexión a la base de datos, que hemos visto anteriormente y se utilizarán más adelante. Se ha decidido colocar aquí para centralizar los ajustes de conexión, de una forma parametrizable.
- En el siguiente párrafo se configura en el siguiente orden:
  - o El ejecutable de la aplicación
  - o El puerto en el que va a arrancar la aplicación
  - o El secret key asociado a la aplicación de Play, generado para funcionar en la Release.
  - o Las 3 siguientes líneas se refieren a la autoaplicación y configuración de los evolutions de Play, es decir, la creación de las tablas en base de datos, a partir de los modelos de datos definidos en la aplicación, con todos sus atributos.
  - o La siguiente línea hace referencia a los hosts habilitados para ejecutar la aplicación. En este caso el '.' representa que se pueda ejecutar en todos. Inicialmente se había pensado en dejar '.amazonaws.com', pero además de generar algunos warnings de conexión no permitida, limitaba la migración de la aplicación, si se cambiara a otro servidor cloud que no fuera de Amazon.
  - o Las siguientes 4 líneas son la asignación de las variables de conexión JDBC a la DB.
  - o Los últimos 3 son parámetros de configuración de memoria reservada y ejecución de la aplicación Java.

Si realizando estos pasos no hemos cometido errores, podemos lanzar este script de ejecución con la configuración necesaria y debe funcionar correctamente mostrando:

ubuntu@ip-172-31-35-140: /opt/recetas

```
ubuntu@ip-172-31-35-140:/opt/recetas$ sudo ./start.sh
[info] p.a.d.DefaultDBApi - Database [default] initialized
[info] p.a.d.HikariCPConnectionPool - Creating Pool for datasource 'default'
[info] play.api.Play - Application started (Prod) (no global state)
[info] p.c.s.AkkaHttpServer - Listening for HTTP on /0:0:0:0:0:0:0:0:80
```

Llegados a este punto, nos falta crear el fichero de sistema, para que nuestro script se ejecute como servicio en cuanto la instancia esté arrancada. Para ello vamos a la ruta de sistema:

/etc/systemd/system

En este caso lo vamos a crear como: recetas.service con el contenido:

```
ubuntu@ip-172-31-35-140: /etc/systemd/system
GNU nano 4.8 recetas.service
[Unit]
Description="Recetas Application"

[Service]
WorkingDirectory=/opt/recetas
ExecStart=/opt/recetas/start.sh
ExecStop=/bin/kill -TERM $MAINPID
Type=simple
Restart=always

[Install]
WantedBy=multi-user.target
```

Siendo en este caso “recetas” el directorio raíz de nuestra aplicación y “start.sh” el script creado anteriormente para lanzar la aplicación.

Solo nos queda ponerlo en funcionamiento con los comandos:

```
sudo systemctl daemon-reload
sudo systemctl enable recetas.service
sudo systemctl start recetas
```

Y para comprobar si funciona, hacemos un:

```
sudo systemctl status recetas
```

```
ubuntu@ip-172-31-35-140: /etc/systemd/system
ubuntu@ip-172-31-35-140: /etc/systemd/system$ sudo systemctl status recetas
* recetas.service - "Recetas Application"
   Loaded: loaded (/etc/systemd/system/recetas.service; enabled; vendor preset: enabled)
   Active: active (running) since Thu 2021-03-04 22:40:52 UTC; 28s ago
     Main PID: 8280 (start.sh)
       Tasks: 20 (limit: 1160)
      Memory: 260.4M
    CGroup: /system.slice/recetas.service
            └─8280 /bin/sh /opt/recetas/start.sh
              └─8281 java -Duser.dir=/opt/recetas -Dhttp.port=80 -Dplay.http.secret.key=AmmuVYIXRMLj<5Iw7Qr5wkCxyika'HC7ejgwAqplH>q7B1;Qv0'8eosuca207^ -Dplay.evolutions.c

Mar 04 22:40:52 ip-172-31-35-140 systemd[1]: Started "Recetas Application".
Mar 04 22:40:56 ip-172-31-35-140 start.sh[8281]: [info] p.a.d.DefaultDBApi - Database [default] initialized
Mar 04 22:40:56 ip-172-31-35-140 start.sh[8281]: [info] p.a.d.HikariCPConnectionPool - Creating Pool for datasource 'default'
Mar 04 22:41:01 ip-172-31-35-140 start.sh[8281]: [info] play.api.Play - Application started (Prod) (no global state)
Mar 04 22:41:02 ip-172-31-35-140 start.sh[8281]: [info] p.c.s.AkkaHttpServer - Listening for HTTP on /0:0:0:0:0:0:0:80
lines 1-15/15 (END)
```

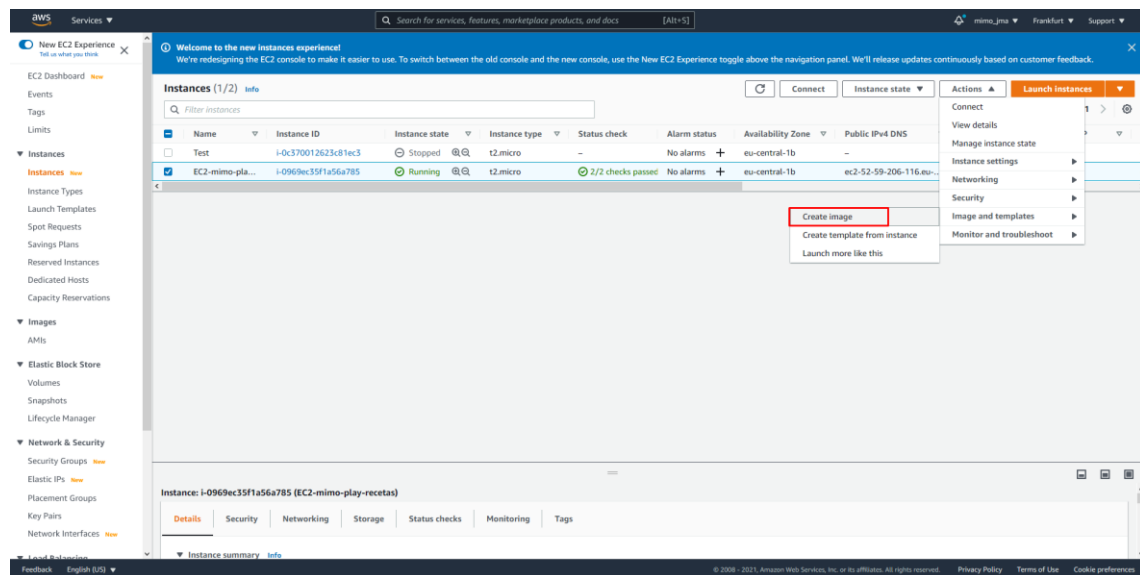
Con esta configuración como servicio y con opción de reiniciarse ante errores, aseguramos en la medida de lo posible que nuestra instancia mantenga en ejecución la aplicación desde que se inicia, siempre que esté disponible.

Por tanto, una vez realizado esto, ya tenemos nuestra aplicación funcionando correctamente y conectada a la base de datos.

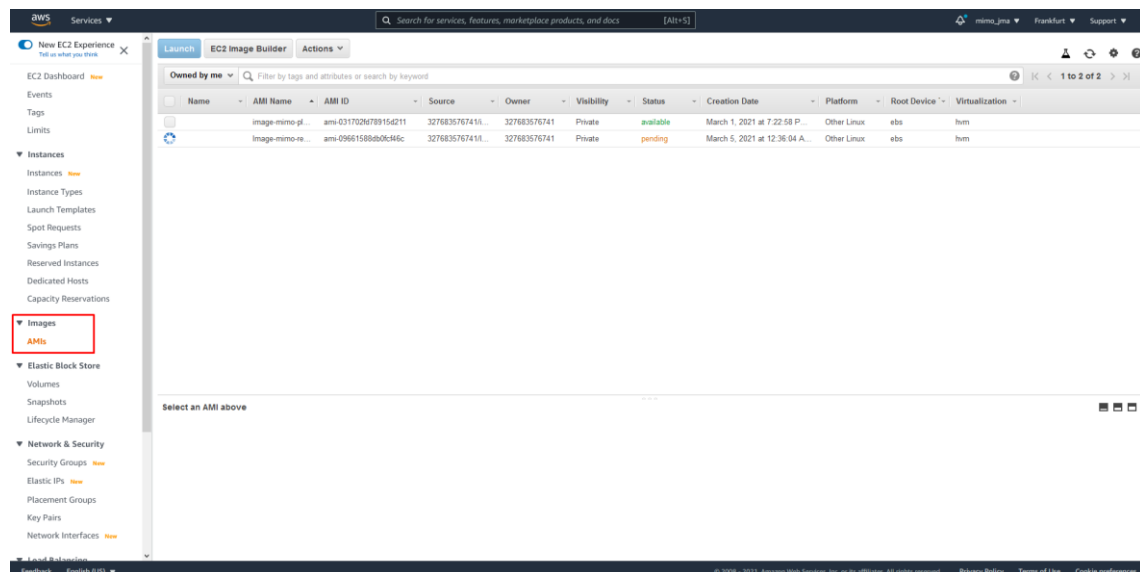
## 2.3. AMIs y LoadBalancers

Sin embargo, para facilitar la replicación y mejorar la escalabilidad horizontal de la aplicación, AWS nos ofrece algunos servicios muy sencillos de configurar y que nos aportan justo lo que necesitamos. Se trata de los AMIs y los Load Balancers, que comentaremos a continuación.

En primer lugar, vamos a ver los AMIs. Se trata de una foto o captura del estado de nuestra máquina/instancia creada. La foto incluye todas las configuraciones que hemos realizado anteriormente de forma manual, para replicarla tantas veces como queramos. Para ello, seleccionamos desde EC2 nuestra instancia creada anteriormente y vamos a la opción de create image desde el desplegable Actions:



Le indicamos un nombre y pulsamos en crear con los ajustes predeterminados. Hecho esto podemos consultarla desde el apartado Images > AMIs:

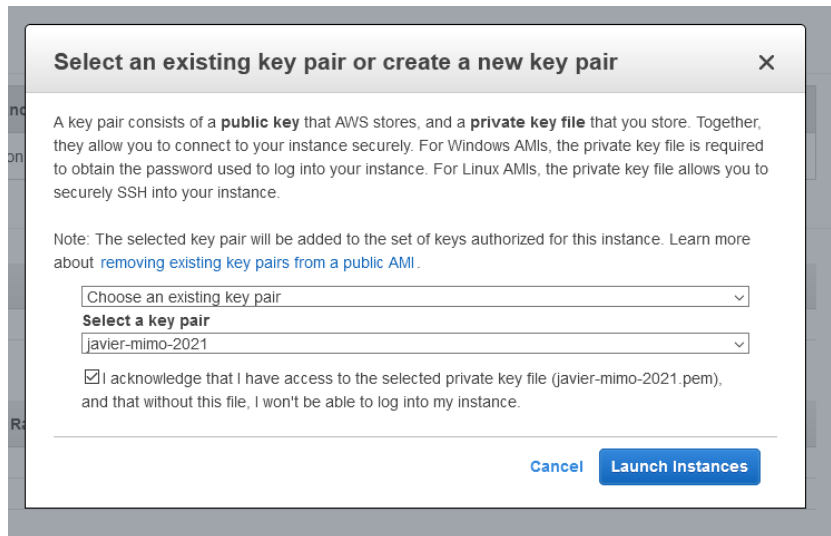


Cuando el estado pase a disponible, podremos crear instancias en base a esta imagen, es decir, nuevas instancias con la misma configuración que la original en el momento en el que se creó el AMI. Simplemente pulsamos en Launch y volveremos al mismo proceso de alta que hicimos al



comienzo de instancias. Aquí simplemente cabe recalcar que debemos asignarles a las nuevas, el mismo security group que a la primera, evitando así volver a configurar la conexión a la base de datos y abrir el puerto 80.

También es importante indicarle a AWS que ya tenemos un par de claves para esta conexión, no queremos crear otro:



**Select an existing key pair or create a new key pair** X

A key pair consists of a **public key** that AWS stores, and a **private key file** that you store. Together, they allow you to connect to your instance securely. For Windows AMIs, the private key file is required to obtain the password used to log into your instance. For Linux AMIs, the private key file allows you to securely SSH into your instance.

Note: The selected key pair will be added to the set of keys authorized for this instance. Learn more about [removing existing key pairs from a public AMI](#).

Choose an existing key pair

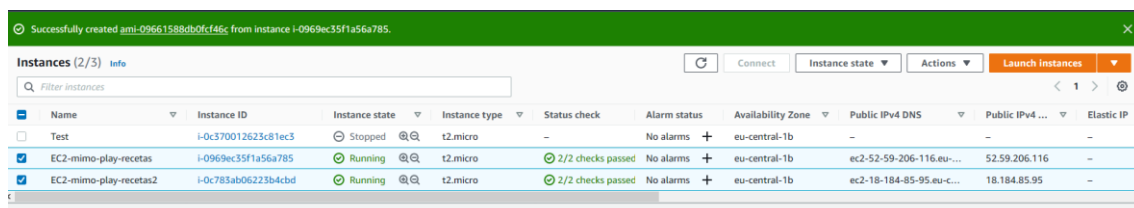
**Select a key pair**

javier-mimo-2021

☒ I acknowledge that I have access to the selected private key file (javier-mimo-2021.pem), and that without this file, I won't be able to log into my instance.

Cancel Launch Instances

Unos minutos más tarde, ya tendremos las dos instancias corriendo a la vez dando servicio a nuestra aplicación.



Successfully created ami-09661588db0cf46c from instance i-0969ec35f1a56a785.

Instances (2/3) Info

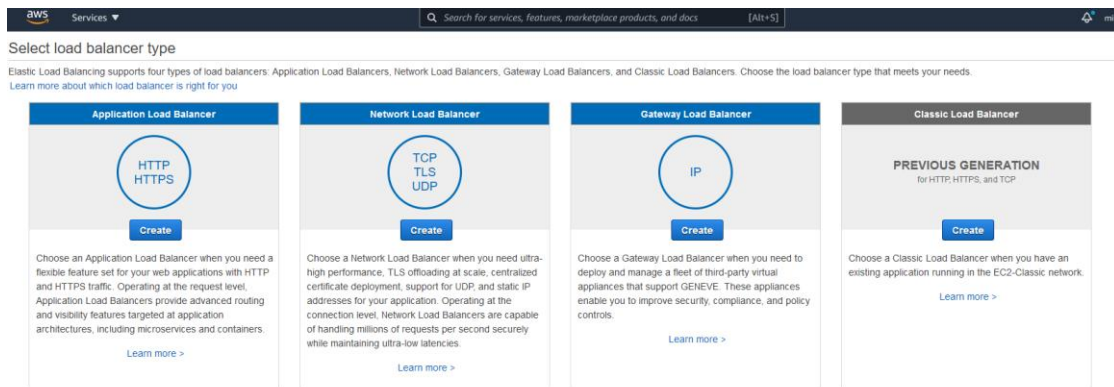
Filter instances

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 ...	Elastic IP
Test	i-0c370012623c81ec3	Stopped	t2.micro	–	No alarms	eu-central-1b	–	–	–
EC2-mimo-play-recetas	i-0969ec35f1a56a785	Running	t2.micro	2/2 checks passed	No alarms	eu-central-1b	ec2-52-59-206-116.eu-...	52.59.206.116	–
EC2-mimo-play-recetas2	i-0c783ab06223b4cbd	Running	t2.micro	2/2 checks passed	No alarms	eu-central-1b	ec2-18-184-85-95.eu-c...	18.184.85.95	–

Como vemos es una forma muy útil y rápida de escalar nuestras aplicaciones.

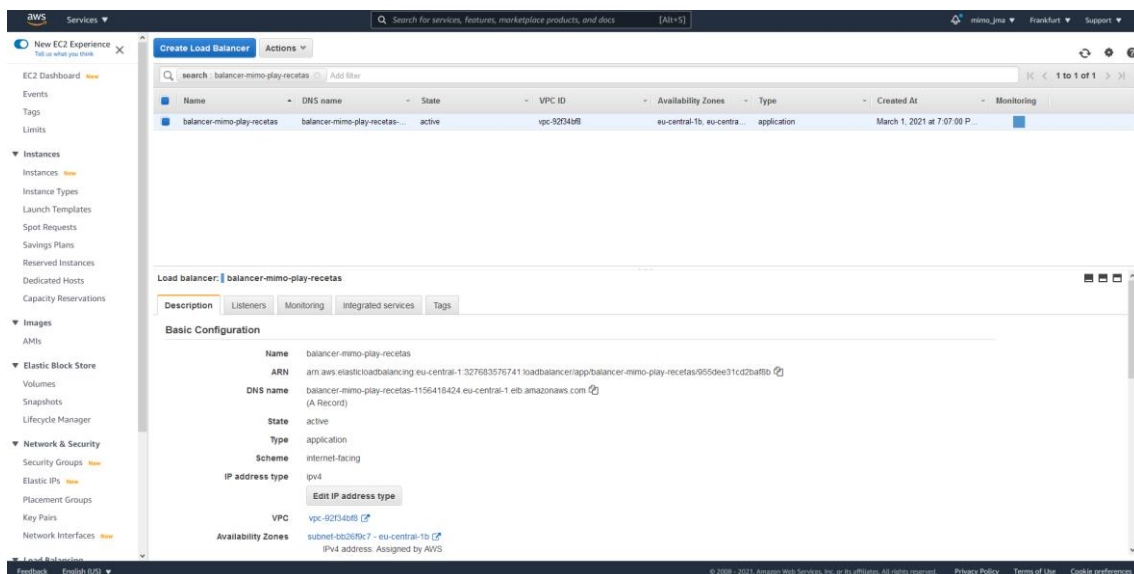
Sin embargo, cada una de ellas tiene una dirección DNS diferente (y dinámica), por lo que necesitamos algún servicio que nos centralice las peticiones en un único sitio, para no tener que ir cambiándolo manualmente. Aquí es donde entra en juego el Load Balancer, cuya función, además de centralizar las peticiones en una única URL estática, se encarga de derivar automáticamente las peticiones a la máquina cuya carga sea inferior en cada momento. Es por tanto, un servicio muy útil para el proceso de escalado y tolerancia a errores de la aplicación, como explicaremos más en detalle.

El proceso para crearlo se realiza desde el servicio de EC2: Load Balancing > Load Balancers > Create Load Balancer.

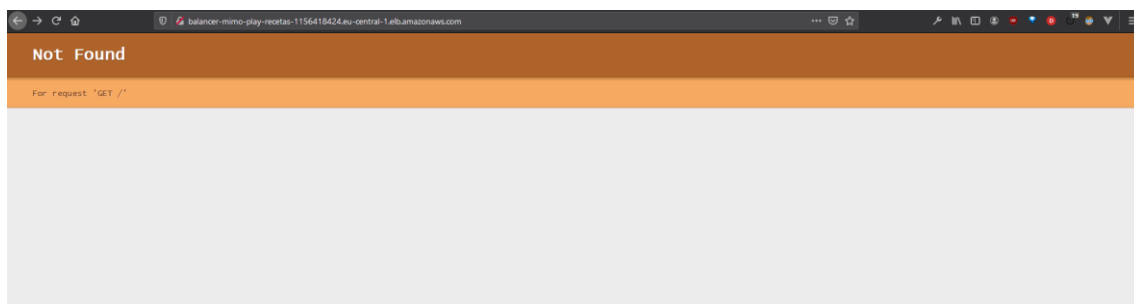


En nuestro caso, las peticiones a la aplicación se realizan a través del protocolo HTTP en el puerto 80, por lo que vamos a seleccionar a primera opción. La configuración de creación es:

- Indicamos el nombre del Load Balancer.
- Seleccionamos todas las zonas de disponibilidad que aparecen.
- Creamos un nuevo security group.
- Configuramos el routing en este caso indicando un nombre y la configuración por defecto.
- Finalmente seleccionamos el Target Group, con las instancias que va a incluir en el balanceador.




Si accedemos a la URL del balancer en el navegador, podemos ver que nos da una respuesta de nuestra aplicación:



Podemos consultarlo también desde el Target Group del Balancer:

Al igual que en el caso anterior, al probar la ruta /, las peticiones AWS lo detectan como status Unhealthy, por el motivo que ya comentado. Para solucionarlo, podemos cambiar el “Health check settings” de la configuración por defecto, que esperaba un 200.

 Services

Search for services, features, marketplaces

HTTP

Health check path

Use the default path of "/" to ping the root, or specify a custom path if preferred.

Up to 1024 characters allowed.

Advanced health check settings

Restore defaults

Port

The port the load balancer uses when performing health checks on targets. The default is the port on which each target receives traffic from the load balancer, but you can specify a different port.

☒ Traffic port

☐ Override

Healthy threshold

The number of consecutive health checks successes required before considering an unhealthy target healthy.

2-10

Unhealthy threshold

The number of consecutive health check failures required before considering a target unhealthy.

2-10

Timeout

The amount of time, in seconds, during which no response means a failed health check.

seconds

2-120

Interval

The approximate amount of time between health checks of an individual target

seconds

5-300

Success codes

The HTTP codes to use when checking for a successful response from a target. You can specify multiple values (for example, "200,202") or a range of values (for example, "200-299").

Vamos a cambiar la ruta a /user que sí está definida por la aplicación. La respuesta esperada en este caso es un 401, ya que se requiere pasar un Auth Token válido en la cabecera. Si tras hacer el cambio actualizamos el Health Status:

Group details

Targets

Monitoring

Tags

Registered targets (3)

2 matches

Status: healthy X

Clear filters

Deregister

Register targets

<input type="checkbox"/>	Instance ID	Name	Port	Zone	Status	Status details
<input type="checkbox"/>	i-03b97750e57c013de	EC2-mimo-play-recetas	80	eu-central-1b	healthy	
<input type="checkbox"/>	i-0c783ab06223b4cbd	EC2-mimo-play-recetas2	80	eu-central-1b	healthy	

La misma petición GET realizada desde un cliente HTTP como Postman y con el Token correctamente definido en la cabecera, nos devuelve un 200:

Por tanto, se están centralizando en una única URL la peticiones que realizamos. Por detrás el Load balancer se encarga de derivar la carga a las instancias disponibles. En caso de que se cayera alguna de las máquinas seguiríamos dando servicio con la otra. Por tanto, este servicio mejora la disponibilidad y la tolerancia a errores.

En principio hemos asociado dos instancias, ya que la carga no se espera que sea muy alta. Sin embargo, si por algún motivo fuera necesario añadir una instancia más, simplemente crearíamos otra instancia con el AMI anterior y lo añadiríamos desde el Target Group al Load Balancer. Está sería una forma de escalado horizontal.

Sería conveniente definir reglas para indicar el mínimo y máximo de instancias que deben funcionar en un balancer, para que AWS se encargara de gestionar y crear nuevas instancias si fuera el caso que las habituales estuvieran caídas, para llegar a los mínimos definidos. Esto puede realizarse desde el servicio de AUTO SCALING que nos ofrece EC2:

Auto Scaling groups (1)

↻

Edit

Delete

Create an Auto Scaling group

<input type="checkbox"/>	Name ▾	Launch template/configuration ↗	Instances ▾	Status ▾	Desired capacity ▾	Min ▾	Max ▾	Availability Zones ▾
<input type="checkbox"/>	tf-asg-2021030611322829200000f	terraform-2021030611322823450...	3	-	2	2	2	eu-central-1a, eu-central-1c, eu-central-1b

Esto sin duda, mejoraría muchísimo la disponibilidad y la tolerancia a errores, asegurando que siempre se va a dar servicio a nuestra aplicación, en base a las reglas definidas de mínimo y máximo.

Hasta aquí las herramientas vistas y su puesta en práctica a nuestra aplicación de API REST desarrollada. Cabe decir que el uso de herramientas de infraestructura como código (por ejemplo: packer y terraform), facilitan mucho el trabajo, ya que permiten la automatización de todas las operaciones que hemos realizado a lo largo de este proyecto. Además, facilita tener un control de versiones del propio generador de instancias y servicios, por si alguno de los requerimientos cambiara en algún momento. Sería la mejor forma de automatizar los despliegues de las diferentes versiones de nuestra aplicación en un futuro.

### 3. Conclusiones

Con esta práctica y la aplicación de todos los conceptos vistos sobre IAAS y sus diferencias notables frente a los PAAS, han sido de gran utilidad para tener una visión general del funcionamiento y despliegue de las aplicaciones Cloud. Sin duda es una tendencia de desarrollo al alza por todas las ventajas que supone respecto a los métodos más tradicionales.

Tanto por las facilidades que aporta como por la fiabilidad de que la aplicación va a estar dando servicio en todo momento.

Ha sido también una buena introducción a las herramientas principales de AWS así como al scripting en relación con el despliegue de los servicios. Será de gran utilidad a la hora de aplicarlo en un futuro y conocer más funcionalidades interesantes.