

## DESPLIEGUE DE UNA APLICACIÓN WEB MULTICAPA ALTAMENTE DISPONIBLE Y SEGURA MEDIANTE AWS ELASTIC BEANSTALK

AWS Elastic Beanstalk es un servicio que permite a los desarrolladores desplegar y administrar rápidamente aplicaciones en la nube de AWS sin necesidad de gestionar la infraestructura en la que se ejecutan las aplicaciones, reduciendo con ello la complejidad y sobrecarga administrativa que conlleva el aprovisionamiento, balanceo de carga, escalado y monitorización de la aplicación.

AWS Elastic Beanstalk permite desplegar aplicaciones desarrolladas en Java, PHP, .NET, Ruby, Python, Go y Node.js en instancias de Amazon EC2.

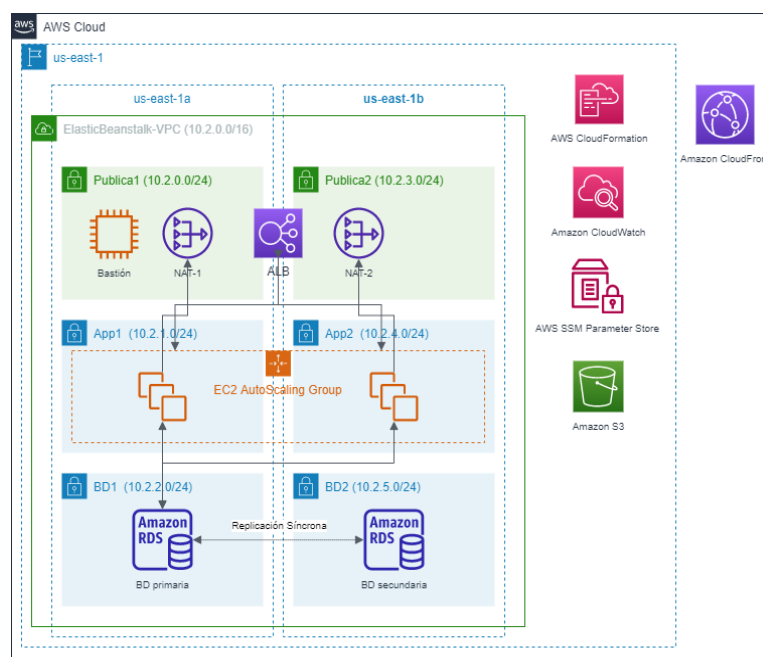
Es posible interactuar con AWS Elastic Beanstalk de la misma forma que con el resto de los servicios; es decir, mediante la Consola de Administración de AWS, mediante la AWS CLI y programáticamente desde los SDKs. Adicionalmente, AWS pone a nuestra disposición una interfaz de línea de comandos de más alto nivel especialmente diseñada para AWS Elastic Beanstalk desde la que podemos desplegar con facilidad y rapidez las aplicaciones.

El objetivo de esta práctica es, precisamente, dar a conocer las posibilidades de esta herramienta desplegando una sencilla aplicación web multicapa sin estado desarrollada en PHP utilizando AWS Elastic Beanstalk, junto con otros servicios para diseñar una infraestructura altamente disponible, escalable y segura.

### Requerimientos:

- Disponer de acceso a los recursos de AWS a través de un *sandbox* de AWS Academy
- Disponer de un entorno Linux con *Git* instalado, así como de la AWS CLI configurada con las credenciales del AWS Academy Learner Lab

### Arquitectura propuesta:



## Realización:

### CREACIÓN DE LA INFRAESTRUCTURA DE RED

En primer lugar, debemos definir el entorno de red donde se lanzarán los recursos computacionales de *AWS Elastic Beanstalk*. Para ello, debe crearse una VPC (*Virtual Private Cloud*) que, siguiendo el esquema planteado tendrá asignado el bloque CIDR 10.2.0.0/16 y abarcará dos zonas de disponibilidad. Se crearán en dicha VPC las siguientes subredes:

- *Publica1* (10.2.0.0/24) y *Publica2* (10.2.3.0/24), subredes públicas donde se desplegará el ALB (*Application Load Balancer*)
- *App1* (10.2.1.0/24) y *App2* (10.2.4.0/24), donde se desplegarán las instancias EC2 del entorno de *AWS Elastic Beanstalk*
- *BD1* (10.2.2.0/24) y *BD2* (10.2.5.0/24), donde se ubicarán las instancias RDS en *multi-AZ*, tanto la primaria como la secundaria, respectivamente

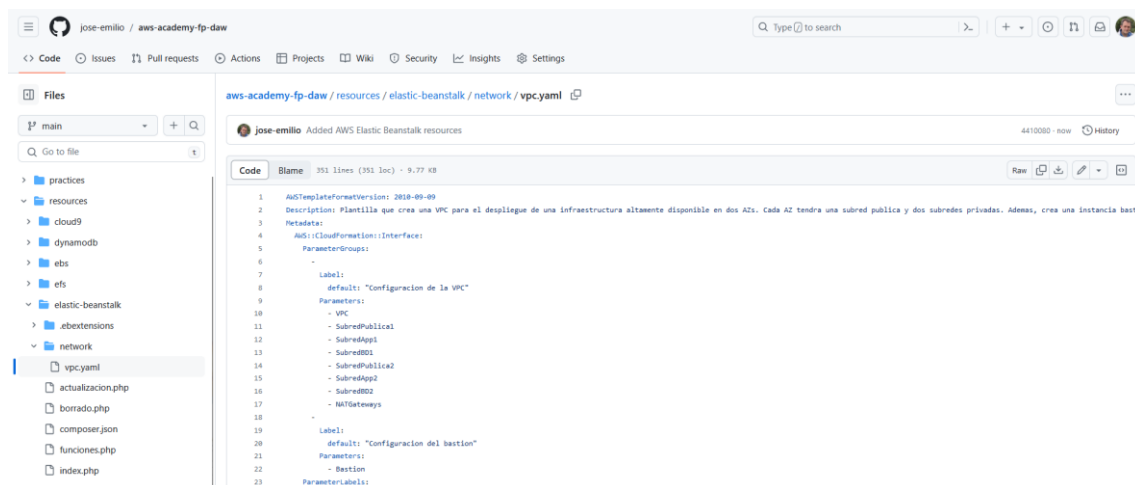
Esta arquitectura nos permitirá desplegar nuestra aplicación de tres capas, asegurando los recursos del *backend*, mientras exponemos el balanceador de carga públicamente a través de Internet.

Como el objetivo de esta práctica no es la creación de la infraestructura de red, ésta se facilita mediante una plantilla de *AWS CloudFormation* llamada *vpc.yaml* incluida en el directorio *network* del repositorio.

Además, aparte de la infraestructura de red, esta plantilla crea:

- Una instancia EC2 que utilizaremos como bastión para conectarnos a la BD y poder configurarla.
  - Dos dispositivos Gateway NAT, uno en cada zona de disponibilidad, para permitir que los recursos en subredes privadas puedan iniciar conexiones hacia Internet.
- 1) Para poder desplegar la arquitectura propuesta es necesario crear la infraestructura de red anterior, compuesta por una VPC con dos subredes privadas y dos subredes públicas en cada zona de disponibilidad. Debido a que el despliegue de dicha infraestructura no es el objetivo de esta práctica, se proporciona una plantilla de *AWS CloudFormation* para tal fin. Dicha plantilla, llamada *vpc.yaml*, la podemos descargar desde:

<https://raw.githubusercontent.com/jose-emilio/aws-academy-fp-daw/main/resources/elastic-beanstalk/network/vpc.yaml>



The screenshot shows a GitHub repository for 'jose-emilio / aws-academy-fp-daw'. The file 'vpc.yaml' is selected in the 'network' directory. The file content is as follows:

```
1  AWSTemplateFormatVersion: 2010-09-09
2  Description: Plantilla que crea una VPC para el despliegue de una infraestructura altamente disponible en dos AZs. Cada AZ tendrá una subred pública y dos subredes privadas. Además, crea una instancia bast
3  Metadata:
4    AWS::CloudFormation::Interface:
5      ParameterGroups:
6        -
7          Label:
8            default: "Configuración de la VPC"
9          Parameters:
10            - VPC
11            - SubredPublica1
12            - SubredApp1
13            - SubredBD1
14            - SubredPublica2
15            - SubredApp2
16            - SubredBD2
17            - NATGateways
18        -
19          Label:
20            default: "Configuración del bastión"
21          Parameters:
22            - Bastion
23          ParameterLabels:
```

- 2) Para desplegar la pila de recursos, basta con ejecutar la siguiente orden, sustituyendo los valores resaltados por los adecuados (el nombre de la pila puede ser cualquiera):

```
$ aws cloudformation deploy --template-file <ruta-vpc.yaml> --stack-name <nombre-pila> --region us-east-1
```

Transcurridos unos pocos minutos, la infraestructura de red ya estará preparada.

## CREACIÓN DE LA BASE DE DATOS

El siguiente paso es lanzar una instancia de Amazon RDS con compatibilidad con MariaDB en un despliegue en Multi-AZ que garantice la conmutación por error automática en caso de fallo en una de las zonas de disponibilidad donde se desplegará la infraestructura.

Para ello, seguiremos los pasos siguientes:

- 3) En primer lugar, se creará un **grupo de subredes** en Amazon RDS, compuesto por las subredes BD1 y BD2. Para ello necesitaremos conocer los IDs de ambas subredes, por lo que las almacenaremos en diferentes variables de entorno; BD1 y BD2. Ejecutamos las siguientes órdenes, que extraen dicha información de la pila de AWS CloudFormation desplegada:

```
$ BD1=$(aws cloudformation describe-stacks --stack-name vpc-stack --region us-east-1 --query 'Stacks[0].Outputs[?OutputKey==`BD1`].OutputValue' --output text)

$ BD2=$(aws cloudformation describe-stacks --stack-name vpc-stack --region us-east-1 --query 'Stacks[0].Outputs[?OutputKey==`BD2`].OutputValue' --output text)
```

- 4) A continuación, se crea el grupo de subredes:

```
$ aws rds create-db-subnet-group --db-subnet-group-name workshop-subnet-group --db-subnet-group-description "Grupo de subredes RDS para workshop" --subnet-ids $BD1 $BD2 --region us-east-1
```

El resultado será un documento JSON con un contenido similar al siguiente, en el que puede visualizarse la información del grupo de subredes de RDS creado:

```
{
  "DBSubnetGroup": {
    "DBSubnetGroupName": "workshop-subnet-group",
    "DBSubnetGroupDescription": "Grupo de subredes RDS para workshop",
    "VpcId": "vpc-xxxxxxxxxxxxxxxx",
    "SubnetGroupStatus": "Complete",
    "Subnets": [
      {
        "SubnetIdentifier": "subnet-xxxxxxxxxxxxxxxx",
        "SubnetAvailabilityZone": {
          "Name": "us-east-1b"
        },
        "SubnetOutpost": {},
        "SubnetStatus": "Active"
      },
      {
        "SubnetIdentifier": "subnet-xxxxxxxxxxxxxxxx",
        "SubnetAvailabilityZone": {
          "Name": "us-east-1a"
        },
        "SubnetOutpost": {},
        "SubnetStatus": "Active"
      }
    ]
  }
}
```

```

    "DBSubnetGroupArn": "arn:aws:rds:us-east-1:123456789012:subgrp:workshop-subnet-
group",
    "SupportedNetworkTypes": [
        "IPv4"
    ]
}
}

```

- 5) El siguiente paso es crear un grupo de seguridad para la instancia RDS que permita el tráfico por el puerto TCP 3306 (MySQL/MariaDB). Para ello ejecutamos las siguientes instrucciones:

```

$ VPC=$(aws cloudformation describe-stacks --stack-name vpc-stack --region us-east-1 --
query 'Stacks[*].Outputs[?OutputKey==`VPC`].OutputValue' --output text)

$ RDS_SG=$(aws ec2 create-security-group --description "Grupo de seguridad RDS workshop"
--group-name workshop-rds-sg --vpc-id $VPC --region us-east-1 --output text)

$ aws ec2 authorize-security-group-ingress --group-id $RDS_SG --protocol tcp --port 3306
--cidr 0.0.0.0/0 --region us-east-1

```

- 6) Por último, se crea la instancia RDS con la siguiente orden, sustituyendo los campos resaltados por el nombre de usuario privilegiado y su contraseña, que posteriormente se empleará para la conexión a la base de datos:

```

$ aws rds create-db-instance --db-name workshop --db-instance-identifier workshop-rds --
engine mariadb --db-instance-class db.t4g.small --master-username <usuario-BD> --master-
user-password <contraseña-BD> --vpc-security-group-ids $RDS_SG --db-subnet-group-name
workshop-subnet-group --allocated-storage 20 --multi-az --tags Key=Name,Value=workshop-
RDS --region us-east-1

```

Con la instrucción anterior, se creará una instancia de RDS en un despliegue multi-AZ, utilizando las subredes indicadas en el grupo de subredes, para garantizar la alta disponibilidad de la BD. En caso de un error en la base de datos principal, el servicio *Amazon RDS* conmutará por error a la base de datos en espera. El resultado de la ejecución anterior debería mostrar un documento JSON como el siguiente:

```

{
  "DBInstance": {
    "DBInstanceIdentifier": "workshop-rds", //Identificador de la instancia
    "DBInstanceClass": "db.t4g.small", //Tipo de la instancia
    "Engine": "mariadb", //Motor de la BD
    "DBInstanceStatus": "creating",
    "MasterUsername": "admin", //Usuario administrador de la BD
    "DBName": "workshop", //Nombre de la BD inicial
    "AllocatedStorage": 20,
    "PreferredBackupWindow": "04:51-05:21",
    "BackupRetentionPeriod": 1,
    "DBSecurityGroups": [],
    "VpcSecurityGroups": [
      {
        "VpcSecurityGroupId": "sg-0a0b7e5ce39942c93", //Grupo de seguridad
        "Status": "active"
      }
    ],
    "DBParameterGroups": [
      {
        "DBParameterGroupName": "default.mariadb10.6",
        "ParameterApplyStatus": "in-sync"
      }
    ],
    "DBSubnetGroup": {
      "DBSubnetGroupName": "workshop-subnet-group", //Grupo de subredes
      "DBSubnetGroupDescription": "Grupo de subredes RDS para workshop",

```

```

    "VpcId": "vpc-0ddc78dc5a32bb64a",
    "SubnetGroupStatus": "Complete",
    "Subnets": [
      {
        "SubnetIdentifier": "subnet-0a27aa5230bb3bca8",
        "SubnetAvailabilityZone": {
          "Name": "us-east-1b"
        },
        "SubnetOutpost": {},
        "SubnetStatus": "Active"
      },
      {
        "SubnetIdentifier": "subnet-0124605009aba6ae5",
        "SubnetAvailabilityZone": {
          "Name": "us-east-1a"
        },
        "SubnetOutpost": {},
        "SubnetStatus": "Active"
      }
    ]
  },
  "PreferredMaintenanceWindow": "sun:09:45-sun:10:15",
  "PendingModifiedValues": {
    "MasterUserPassword": "*****"
  },
  "MultiAZ": true, //Despliegue en Multi-AZ
  "EngineVersion": "10.6.10",
  "AutoMinorVersionUpgrade": true,
  "ReadReplicaDBInstanceIdentifiers": [],
  "LicenseModel": "general-public-license",
  "OptionGroupMemberships": [
    {
      "OptionGroupName": "default:mariadb-10-6",
      "Status": "in-sync"
    }
  ],
  "PubliclyAccessible": false,
  "StorageType": "gp2",
  "DbInstancePort": 0,
  "StorageEncrypted": false,
  "DbiResourceId": "db-AS5KJ6247MD4QB24AVKW3CXIYE",
  "CACertificateIdentifier": "rds-ca-2019",
  "DomainMemberships": [],
  "CopyTagsToSnapshot": false,
  "MonitoringInterval": 0,
  "DBInstanceArn": "arn:aws:rds:us-east-1:123456789012:db:workshop-rds",
  "IAMDatabaseAuthenticationEnabled": false,
  "PerformanceInsightsEnabled": false,
  "DeletionProtection": false,
  "AssociatedRoles": [],
  "TagList": [
    {
      "Key": "Name",
      "Value": "workshop-RDS"
    }
  ],
  "CustomerOwnedIpEnabled": false,
  "BackupTarget": "region",
  "NetworkType": "IPV4"
}
}

```

- 7) Tras esperar unos minutos a que la instancia RDS se aprovisiona, se procederá a crear el esquema de la base de datos que necesita la aplicación. Para ello, nos conectaremos de forma segura mediante la consola de Amazon EC2 (utilizando *Session Manager*) a la instancia Bastión (que se creó cuando se desplegó la infraestructura de la red). La instancia EC2 Bastión viene preinstalada con el cliente de MySQL por lo que ejecutamos la siguiente orden, sustituyendo el valor resaltado por el nombre de usuario establecido anteriormente:

```
$ DATABASE=$(aws rds describe-db-instances --region us-east-1 --query  
'DBInstances[?DBInstanceIdentifier==`workshop-rds`].Endpoint.Address' --output text)  
  
$ mysql -u <usuario-BD> -p -h $DATABASE
```

- 8) Una vez establecida la conexión contra la base de datos, se crea una sencilla tabla llamada CONTACTOS con la siguiente orden en SQL:

```
create table workshop.contactos(id varchar(10),nombre varchar(100),email  
varchar(60),telefono varchar(12),cargo varchar(30),fechaCreacion datetime, primary key  
(ID));
```

Con esto, se habría terminado de configurar la base de datos y terminamos la sesión en la instancia EC2 Bastión.

- 9) Sin embargo, por motivos de seguridad, no se introducirán las credenciales (contraseña) de la base de datos dentro del código de la aplicación, sino que se permitirá que el servicio de *AWS Systems Manager Parameter Store* custodie el secreto. Posteriormente, cuando se despliegue la infraestructura de la aplicación se otorgarán a las instancias EC2 los permisos temporales necesarios para que puedan acceder al valor del parámetro que almacena la contraseña. Para ello, vamos a crear un secreto en AWS Systems Manager Parameter Store con la siguiente orden, sustituyendo los valores resaltados por el nombre del secreto almacenado (puede ser cualquiera) y por la contraseña del usuario privilegiado de la base de datos RDS:

```
$ aws ssm put-parameter --name <nombre-secreto> --value <contraseña-BD> --type  
SecureString --region us-east-1
```

La sentencia anterior crea un parámetro secreto cifrado mediante la clave predeterminada del servicio *AWS Key Management Service* (KMS). El nombre de este parámetro se le pasará a la aplicación como una variable de entorno a la que accederá en tiempo de ejecución. Si todo ha funcionado según lo esperado, recibiremos un documento JSON con un contenido similar al siguiente:

```
{  
  "Version": 1,  
  "Tier": "Standard"  
}
```

## CREACIÓN DE UN ENTORNO PARA LA APLICACIÓN MEDIANTE AWS ELASTIC BEANSTALK

La aplicación que se desplegará se encuentra dentro del repositorio. Se trata de una sencilla aplicación en PHP que realiza operaciones CRUD sobre una tabla de contactos. Para garantizar la privacidad y seguridad de la conexión a la base de datos, todos los parámetros de configuración se pasarán mediante las siguientes variables de entorno:

- **AWS\_REGION.** Región donde se encuentra almacenado el secreto de *AWS Systems Manager Parameter Store* que contiene la contraseña de la BD
- **RDS\_HOSTNAME.** Contendrá el punto de enlace de la instancia RDS que se ha creado
- **RDS\_USER\_NAME.** Contendrá el nombre de usuario creado para la conexión a la BD
- **RDS\_DB\_SECRET.** Contendrá el nombre del secreto de *AWS Systems Manager Parameter Store* donde se custodia la contraseña de conexión a la BD
- **RDS\_DB\_NAME.** Nombre de la base de datos creada en la instancia RDS, en el caso de esta práctica es *workshop*, pero podría ser cualquier otro

En el siguiente fragmento de código del archivo *functions.php* puede verse cómo se utilizan estas variables de entorno para parametrizar la conexión contra la instancia RDS:

```
<?php

require './vendor/autoload.php';
use Aws\Ssm\SsmClient;
use Aws\Exception\AwsException;

function conectar_mariadb() {
    try {
        //Se obtiene la contraseña de la BD desde AWS SSM Parameter Store
        $cliente = new SsmClient(['version' => 'latest', 'region' => $_SERVER['AWS_REGION']]);
        $resultado = $cliente->getParameter(['Name' => $_SERVER['RDS_DB_SECRET'], 'WithDecryption' => true]);
        $password=$resultado['Parameter']['Value'];
    }
    catch (SsmException $e) {
        echo $e->getMessage() . "\n";
    }
    $DATABASE_HOST = $_SERVER['RDS_HOSTNAME'];
    $DATABASE_USER = $_SERVER['RDS_USER_NAME'];
    $DATABASE_PASS = $password;
    $DATABASE_NAME = $_SERVER['RDS_DB_NAME'];
    try {
        return new PDO('mysql:host=' . $DATABASE_HOST . ';dbname=' . $DATABASE_NAME . ';charset=utf8', $DATABASE_USER, $DATABASE_PASS);
    } catch (PDOException $exception) {
        // Si hubiera un error con la conexión, se para el script y se visualiza el error
        exit('Error al conectar con la base de datos ' . $exception);
    }
}
```

*AWS Elastic Beanstalk* es un servicio PaaS (*Platform as a Service*) que permite a los desarrolladores desplegar y administrar aplicaciones en la nube de AWS de una forma ágil y sencilla. Los desarrolladores sólo tienen que cargar el código de la aplicación y dejar que el servicio se encargue de forma automática de los detalles del despliegue, como el aprovisionamiento de la infraestructura subyacente y su capacidad, el balanceo de carga, el escalado automático y la monitorización del estado de la aplicación.

*AWS Elastic Beanstalk* es un servicio compatible con aplicaciones desarrolladas en Go, Java, PHP, .NET, Node.js, Python, Docker e incluso resulta posible utilizar plataformas personalizadas.

*AWS Elastic Beanstalk* dispone de una CLI personalizada (EB CLI), que se va a utilizar durante esta práctica. Para ello debe descargarse e instalarse en el entorno, siguiendo las instrucciones del siguiente enlace:

<https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/eb-cli3-install.html#eb-cli3-install.scripts>

Para comprender cómo funciona *AWS Elastic Beanstalk*, debemos conocer los siguientes conceptos:

- **Aplicación.** Es una colección lógica de componentes de *AWS Elastic Beanstalk*, incluyendo entornos, versiones y configuraciones de entornos. Se podría decir que, conceptualmente, es similar a una carpeta.
- **Versión de Aplicación.** Se refiere a una iteración específica y etiquetada de código desplegable para una aplicación web. Una versión de la aplicación apunta a un objeto de un bucket de *Amazon S3* que contiene el código desplegable (por ejemplo, un archivo WAR en Java). Las aplicaciones pueden tener múltiples versiones y cada versión es única. Un entorno puede tener desplegada una única versión de la aplicación en un momento determinado.
- **Entorno.** Es una colección de recursos de AWS que ejecutan una versión de una aplicación determinada. Una versión de una aplicación puede ejecutarse a la vez en diferentes entornos. Cuando se crea un entorno, *AWS Elastic Beanstalk* aprovisiona los recursos e infraestructura necesaria para ejecutar la versión de la aplicación elegida.
- **Configuración del entorno.** Es una colección de parámetros y configuraciones que definen cómo se comporta el entorno y sus recursos asociados. Cuando se actualiza la configuración

del entorno, el servicio *AWS Elastic Beanstalk* aplica automáticamente los cambios sobre los recursos existentes o elimina y despliega nuevos recursos.

- 10) En primer lugar, pasaremos por crear nuestra aplicación. Para ello, clonamos previamente el repositorio siguiente:

```
$ git clone https://github.com/jose-emilio/aws-academy-fp-daw.git
```

A continuación, creamos un nuevo directorio, copiamos los contenidos de la aplicación en el nuevo directorio y lo inicializamos como un repositorio de git y hacemos un *commit* inicial:

```
$ mkdir repo-elastic-beanstalk
$ cp -R aws-academy-fp-daw/resources/elastic-beanstalk/* ./repo-elastic-beanstalk
$ cd repo-elastic-beanstalk
$ git init
$ git add *
$ git commit -m "Primer commit "
```

El siguiente paso es inicializar nuestra aplicación mediante la orden:

```
$ eb init --region us-east-1 -v
```

Tras esto, se iniciará un asistente que nos preguntará sobre el nombre de la aplicación:

```
Enter Application Name
(default is "aws-academy-elasticbeanstalk"): workshop
```

La CLI detecta que nuestra aplicación está escrita en PHP, ante lo que confirmamos y seleccionamos la última versión disponible del *runtime*:

```
It appears you are using PHP. Is this correct?
(Y/n): Y

Select a platform branch.
1) PHP 8.1 running on 64bit Amazon Linux 2
2) PHP 8.0 running on 64bit Amazon Linux 2
3) PHP 7.4 running on 64bit Amazon Linux 2 (Deprecated)
(default is 1): 1
```

Continuamos respondiendo al asistente tal y como se muestra en la siguiente imagen:

```
Do you wish to continue with CodeCommit? (Y/n): n

Do you want to set up SSH for your instances?
(Y/n): Y

Select a keypair.
1) vockey
2) [ Create new KeyPair ]
(default is 1): 1
```

Una vez concluido el asistente, ya habremos creado nuestra aplicación llamada *workshop* y necesitaremos crear un entorno para desplegarla.

El entorno de AWS Elastic Beanstalk que crearemos estará compuesto por:

- Un **grupo de autoescalado de instancias EC2**, con un mínimo de 2 instancias, pudiendo escalar hasta 6 instancias, distribuidas entre las dos zonas de disponibilidad habilitadas en



nuestra infraestructura de red. Estas instancias EC2 alojarán las versiones de nuestra aplicación y se desplegarán dentro de las subredes *App1* y *App2*

- Un **balanceador de carga de aplicación**, desplegado dentro de las subredes *Publica1* y *Publica2*, y que se encargará de distribuir el tráfico HTTP entrante entre las diferentes instancias EC2 de nuestra capa de aplicación

- 11) Antes de seguir adelante, vamos a necesitar los IDs de las subredes públicas donde operará nuestro balanceador de carga, y las subredes privadas donde se desplegarán nuestras instancias EC2; almacenaremos esta información en diferentes variables de entorno:

```
$ PUB1=$(aws cloudformation describe-stacks --stack-name vpc-stack --region us-east-1 --query 'Stacks[0].Outputs[?OutputKey==`Publica1`].OutputValue' --output text)

$ PUB2=$(aws cloudformation describe-stacks --stack-name vpc-stack --region us-east-1 --query 'Stacks[0].Outputs[?OutputKey==`Publica2`].OutputValue' --output text)

$ APP1=$(aws cloudformation describe-stacks --stack-name vpc-stack --region us-east-1 --query 'Stacks[0].Outputs[?OutputKey==`App1`].OutputValue' --output text)

$ APP2=$(aws cloudformation describe-stacks --stack-name vpc-stack --region us-east-1 --query 'Stacks[0].Outputs[?OutputKey==`App2`].OutputValue' --output text)
```

- 12) Para crear el entorno de nuestra aplicación deberemos utilizar la siguiente orden, que creará el entorno, cargará en un bucket de S3 la primera versión de nuestra aplicación y la desplegará en la infraestructura computacional subyacente.

```
$ DATABASE=$(aws rds describe-db-instances --region us-east-1 --query 'DBInstances[?DBInstanceIdentifier==`workshop-rds`].Endpoint.Address' --output text)

$ eb create <entorno> -c <CNAME-entorno> -sr LabRole -ip LabInstanceProfile --min-instances 2 --max-instances 6 -i t4g.micro -k vockey --envvars AWS_REGION=us-east-1,RDS_HOSTNAME=$DATABASE,RDS_USER_NAME=admin,RDS_DB_NAME=workshop,RDS_DB_SECRET=<nombre-secreto> --vpc.ec2subnets $APP1,$APP2 --elb-type application --vpc.elbpublic --vpc.id $VPC --vpc.elbsubnets $PUB1,$PUB2
```

En la instrucción anterior debe sustituirse el valor de los campos resaltados:

- *<entorno>*. Debe indicarse el nombre de un entorno de la aplicación, el que se prefiera.
- *<CNAME-entorno>*. Debe indicarse un subdominio que no exista dentro del dominio *elasticbeanstalk.com*
- *<nombre-secreto>*. Debe indicarse el nombre del secreto de *AWS Systems Manager Parameter Store* que se creó anteriormente

Además, la instrucción toma diferentes parámetros de configuración:

- *-sr LabRole*. Es el rol de servicio que necesita *AWS Elastic Beanstalk* para poder, en nuestro nombre, crear los recursos que componen el entorno. Al estar el uso del servicio *AWS IAM* muy limitado en los *AWS Academy Learner Labs*, no tenemos más remedio que asignar el rol de IAM pregenerado llamado *LabRole* que otorga permisos sobre TODAS las acciones que se nos permite realizar (es sin duda, una muy mala práctica...)
- *-ip LabInstanceProfile*. Es el rol que asumirán las instancias EC2 del entorno para interactuar con el resto de los servicios de AWS. En el caso de este taller, nuestras instancias EC2 necesitan permisos para acceder al secreto almacenado en *AWS Systems Manager Parameter Store*. De nuevo, no estamos autorizados para crear roles de IAM y nos vemos abocados a utilizar el perfil de instancia *LabInstanceProfile* (mala práctica)
- *--min-instances*. Número mínimo de instancias EC2 que dispondrá el entorno
- *--max-instances*. Número máximo de instancias EC2 hasta las que podrá escalar el entorno

- `-i t4g.micro`. Tipo de instancia EC2. Se utilizar una `t4g.micro` como prueba; en un entorno real de producción quizás no fuese demasiado apropiada y habría que buscar una instancia de mayores prestaciones
- `-k vockey`. Es el par de claves con las que se podrá acceder vía SSH a las instancias EC2 del entorno. El par de claves `vockey` ya viene pregenerada y podemos descargar la clave privada fácilmente desde la consola en los AWS Academy Learner Labs
- `--envvars`. Se declaran las variables de entorno. Estas variables se leerán desde la aplicación para acceder de forma segura a la base de datos.
- `--vpc.id $VPC`. Indica la VPC donde se desplegarán los recursos del entorno
- `--elb-type application`. Se define el tipo de balanceador de carga, en este caso será un ALB (*Application Load Balancer*) ya que recibirá solicitudes de capa 7 (HTTP) de los clientes
- `--vpc.elbpublic`. Inicialá el ALB en subredes públicas de la VPC
- `--vpc.elbsubnets $PUB1,$PUB2`. Se asignan las subredes públicas donde actuará el ALB
- `--vpc.ec2subnets $APP1,$APP2`. Se asignan las subredes privadas donde se desplegarán las instancias EC2 del entorno

Tras la ejecución de la instrucción anterior, podremos ir comprobando todos los eventos que ocurren y cómo se generan los recursos del entorno:

```

Creating application version archive "app-8d40-221122_104424861404".
Uploading workshop/app-8d40-221122_104424861404.zip to S3. This may take a while.
Upload Complete.
Environment details for: development
  Application name: workshop
  Region: us-east-1
  Deployed Version: app-8d40-221122_104424861404
  Environment ID: e-h7y7kxjexq
  Platform: arn:aws:elasticbeanstalk:us-east-1::platform/PHP 8.1 running on 64bit Amazon
Linux 2/3.5.1
  Tier: WebServer-Standard-1.0
  CNAME: workshop-devel.us-east-1.elasticbeanstalk.com
  Updated: 2022-11-22 10:44:29.644000+00:00
Printing Status:
2022-11-22 10:44:28    INFO    createEnvironment is starting.
2022-11-22 10:44:29    INFO    Using elasticbeanstalk-us-east-1-004677159793 as Amazon
S3 storage bucket for environment data.
2022-11-22 10:44:50    INFO    Created target group named:
arn:aws:elasticloadbalancing:us-east-1:004677159793:targetgroup/awseb-AWSEB-
192NTYPAD4V1F/99572707cb66a94b
2022-11-22 10:44:50    INFO    Created security group named: sg-02adcb0edf6c6fd6
2022-11-22 10:45:06    INFO    Created security group named: sg-033fcd0851f1b4a44
2022-11-22 10:45:06    INFO    Created Auto Scaling launch configuration named: awseb-e-
h7y7kxjexq-stack-AWSEBAutoScalingLaunchConfiguration-vJl4ZV7VY7Nh
2022-11-22 10:45:37    INFO    Created Auto Scaling group named: awseb-e-h7y7kxjexq-
stack-AWSEBAutoScalingGroup-QNCQATDB1H7I
2022-11-22 10:45:37    INFO    Waiting for EC2 instances to launch. This may take a few
minutes.
2022-11-22 10:45:52    INFO    Created Auto Scaling group policy named:
arn:aws:autoscaling:us-east-1:004677159793:scalingPolicy:095e2f4d-ac67-4285-af1c-
d4a9efb1de84:autoScalingGroupName/awseb-e-h7y7kxjexq-stack-AWSEBAutoScalingGroup-
QNCQATDB1H7I:policyName/awseb-e-h7y7kxjexq-stack-AWSEBAutoScalingScaleDownPolicy-
ZnG8C4EqrW1e
2022-11-22 10:45:52    INFO    Created Auto Scaling group policy named:
arn:aws:autoscaling:us-east-1:004677159793:scalingPolicy:f328b4c3-e7af-44b4-beb7-
5a9e12f9508e:autoScalingGroupName/awseb-e-h7y7kxjexq-stack-AWSEBAutoScalingGroup-
QNCQATDB1H7I:policyName/awseb-e-h7y7kxjexq-stack-AWSEBAutoScalingScaleUpPolicy-
VrQsv5Ib9xDy
2022-11-22 10:45:53    INFO    Created CloudWatch alarm named: awseb-e-h7y7kxjexq-stack-
AWSEBCloudwatchAlarmLow-108TFSJ0T42HX
2022-11-22 10:45:53    INFO    Created CloudWatch alarm named: awseb-e-h7y7kxjexq-stack-
AWSEBCloudwatchAlarmHigh-N0V3MAXWHATH
2022-11-22 10:46:57    INFO    Created load balancer named:
arn:aws:elasticloadbalancing:us-east-1:004677159793:loadbalancer/app/awseb-AWSEB-
10XLJ6S0WM1CL/7faec8107f5086a6

```

```

2022-11-22 10:47:12 INFO Created Load Balancer listener named:
arn:aws:elasticloadbalancing:us-east-1:004677159793:listener/app/awseb-AWSEB-
10XLJ6S0WM1CL/7faec8107f5086a6/ad7832fa68360c38
2022-11-22 10:47:31 INFO Instance deployment completed successfully.
2022-11-22 10:48:03 INFO Application available at workshop-devel.us-east-
1.elasticbeanstalk.com.
2022-11-22 10:48:03 INFO Successfully launched environment: development

```

Podemos fijarnos que, durante la creación del entorno, se han creado dos grupos de seguridad:

- Un **grupo de seguridad para el ALB**. Este grupo de seguridad va a permitir todo el tráfico HTTP que provenga desde Internet (0.0.0.0/0), así como todo el tráfico HTTP saliente (la buena práctica en materia de seguridad sería limitar el tráfico HTTP saliente siempre y cuando el destino sea el grupo de seguridad de las instancias EC2)
- Un **grupo de seguridad para las instancias EC2** del grupo de autoescalado. Este grupo de seguridad va a permitir todo el tráfico HTTP entrante que provenga del grupo de seguridad del ALB, así como todo el tráfico saliente.

- 13) Recordemos que, anteriormente, se creó un grupo de seguridad sobre la instancia RDS, que permitía el tráfico de entrada por el puerto 3306 TCP desde cualquier ubicación (0.0.0.0/0). Vamos a limitar ahora el tráfico entrante para que sólo se acepte aquél que provenga desde el grupo de seguridad de la capa de aplicación de nuestra infraestructura. Para ello, ejecutamos la siguiente orden para revocar los permisos anteriores:

```
$ aws ec2 revoke-security-group-ingress --group-id $RDS_SG --port 3306 --protocol tcp --region us-east-1
```

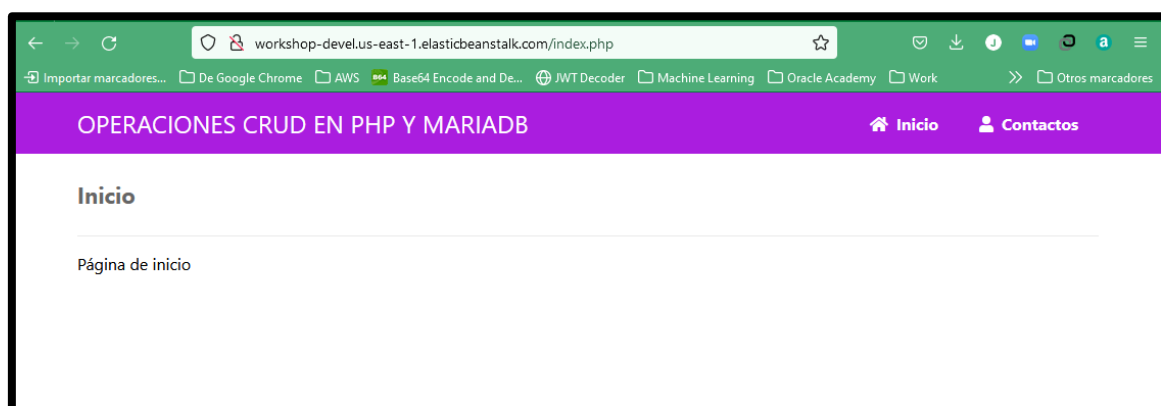
- 14) A continuación, alineándonos con las buenas prácticas en materia de seguridad, limitamos los permisos, estableciendo como origen permitido el grupo de seguridad de las instancias EC2 (se puede obtener fácilmente desde el log de creación del entorno; es el primer grupo de seguridad que se crea)

```
$ aws ec2 authorize-security-group-ingress --group-id $RDS_SG --protocol tcp --port 3306 --source-group sg-02adcb0edf6c6fd6 --region us-east-1
```

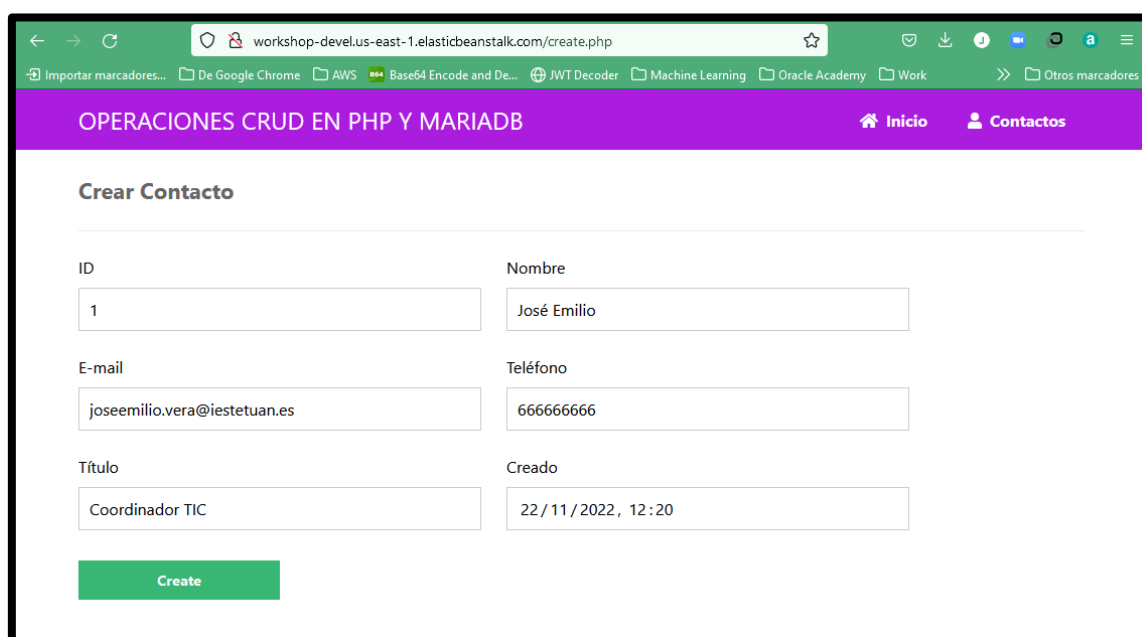
- 15) Tras esto, nuestra infraestructura tendrá definida una estrategia de defensa en profundidad con varios niveles. Para comprobar que nuestra aplicación funciona, tan sólo tendremos que introducir la orden:

```
$ eb open <entorno>
```

Con ello, se nos abrirá una ventana en el navegador mostrándonos el sitio web:



Navegamos por el menú de Contactos y podemos ver cómo funcionan las operaciones CRUD:



workshop-develus-east-1.elasticbeanstalk.com/create.php

OPERACIONES CRUD EN PHP Y MARIADB Inicio Contactos

### Crear Contacto

ID: 1

Nombre: José Emilio

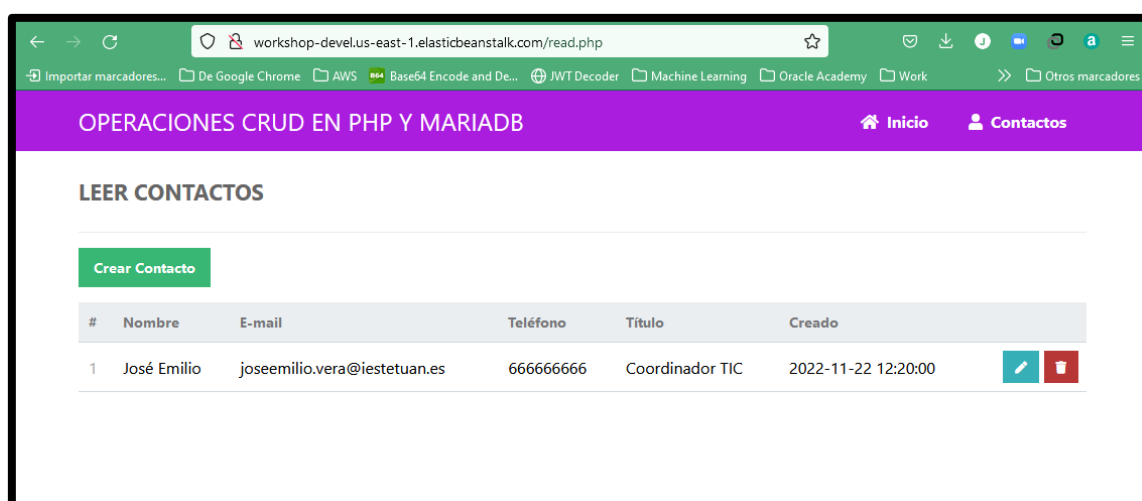
E-mail: joseemilio.vera@iestetuan.es

Teléfono: 666666666

Título: Coordinador TIC

Creado: 22/11/2022, 12:20

Create





workshop-develus-east-1.elasticbeanstalk.com/read.php

OPERACIONES CRUD EN PHP Y MARIADB Inicio Contactos

### LEER CONTACTOS

Crear Contacto

#	Nombre	E-mail	Teléfono	Título	Creado	
1	José Emilio	joseemilio.vera@iestetuan.es	666666666	Coordinador TIC	2022-11-22 12:20:00	 

## MONITORIZACIÓN DE LA APLICACIÓN EN AWS ELASTIC BEANSTALK

Una de las características del servicio *AWS Elastic Beanstalk* es la monitorización integrada de la aplicación web. La monitorización nos permitirá consultar diferentes métricas dentro de nuestro entorno, tanto nuestra aplicación como a nivel de sistema operativo.

El servicio monitoriza las siguientes métricas:

- **Aplicación**
  - *RequestCount*. Número de solicitudes recibidas por segundo
  - *Status2xx*. Número de respuestas con código de estado HTTP 2XX
  - *Status3xx*. Número de respuestas con código de estado HTTP 3XX
  - *Status4xx*. Número de respuestas con código de estado HTTP 4XX
  - *Status5xx*. Número de respuestas con código de estado HTTP 5XX

- **Sistema Operativo**

- *Running*. Tiempo de ejecución desde el lanzamiento
- *Load1*. Promedio de utilización de CPU en el último minuto
- *Load5*. Promedio de utilización de CPU en los últimos 5 minutos
- *User%*. Porcentaje de tiempo de CPU en espacio de usuario
- *Nice%*. Porcentaje de tiempo de CPU en procesos de baja prioridad
- *System%*. Porcentaje de tiempo de CPU en espacio de kernel
- *Idle%*. Porcentaje de tiempo de CPU en espera
- *I/O Wait%*. Porcentaje de tiempo de CPU en espera (sobre disco)

16) Para visualizar estas métricas y otra información adicional, podemos ejecutar la orden, sustituyendo el valor resaltado por el nombre asignado al entorno:

```
$ eb health <nombre-entorno>
```

Tras esto, visualizaremos una pantalla con las métricas en tiempo real:

developmentOk2022-11-22 19:52:57

WebServerPHP 8.1 running on 64bit Amazon Linux 2/3.5.1

total	ok	warning	degraded	severe	info	pending	unknown
2	2	0	0	0	0	0	0

instance-id	status	cause
Overall	OK	
i-0e955c4a44252a0e6	OK	
i-0c417dec59cb261c1	OK	

instance-id	r/sec	%2xx	%3xx	%4xx	%5xx	p99	p90	p75	p50	p10	requests
Overall	0.4	100.0	0.0	0.0	0.0	0.001*	0.001*	0.001	0.000	0.000	
i-0e955c4a44252a0e6	0.2	2	0	0	0	0.001*	0.001*	0.001	0.001	0.000	
i-0c417dec59cb261c1	0.2	2	0	0	0	0.000*	0.000*	0.000	0.000	0.000	

instance-id	type	az	running	load 1	load 5	user %	nice %	system %	idle %	iowait %	cpu
i-0e955c4a44252a0e6	t4g.micro	1b	2 hours	0.0	0.0	0.1	0.0	0.0	99.9	0.0	
i-0c417dec59cb261c1	t4g.micro	1a	2 hours	0.0	0.0	0.3	0.0	0.0	99.7	0.0	

instance-id	status	id	version	ago	deployments
i-0e955c4a44252a0e6	Deployed	1	app-8d40-221122_165359951680	2 hours	
i-0c417dec59cb261c1	Deployed	1	app-8d40-221122_165359951680	2 hours	

17) También es posible visualizar el estado en el que se encuentra un entorno de AWS Elastic Beanstalk, ejecutando la siguiente orden, y sustituyendo el valor resaltado por el nombre asignado al entorno:

```
$ eb status <nombre-entorno>
```

En esta ocasión, podremos visualizar como resultado un resumen del estado de nuestro entorno y comprobar si la aplicación está funcionando adecuadamente:

```
Environment details for: development
Application name: workshop
Region: us-east-1
Deployed Version: app-8d40-221122_165359951680
Environment ID: e-h8x5gaqrjg
Platform: arn:aws:elasticbeanstalk:us-east-1::platform/PHP 8.1 running on 64bit Amazon Linux 2/3.5.1
Tier: WebServer-Standard-1.0
CNAME: workshop-devel.us-east-1.elasticbeanstalk.com
Updated: 2022-11-22 16:57:51.729000+00:00
Status: Ready
Health: Green
```

El estado de salud de un entorno puede indicarse mediante colores:

- *Green* indica que las instancias EC2 (la mayoría de ellas) del entorno están pasando las comprobaciones de estado y no están reportando problemas.
- *Yellow* indica que se están reportando un número moderado de fallos a la hora de servir las solicitudes o algún otro problema con una instancia o el entorno.

- *Red* indica que se está produciendo un alto o muy alto número de fallos a la hora de servir las solicitudes o algún otro problema con una instancia o el entorno.
- *Grey* indica que hay alguna operación en progreso sobre una instancia o bien que no se disponen de datos para determinar el estado de salud del entorno.

## CAMBIOS EN LA CONFIGURACIÓN DEL ENTORNO

AWS Elastic Beanstalk también administra los cambios que se realicen en la configuración del entorno como, por ejemplo, cambios en el tipo de instancia EC2, tipo de entorno (alta disponibilidad, única instancia, ...), comprobaciones de estado, configuración de PHP, estrategia de despliegue, subredes, etc.

Los cambios en la configuración en la VPC o en las propias instancias EC2 provocar reemplazos en las instancias EC2 y es conveniente conocer de qué forma AWS Elastic Beanstalk los implementa estos cambios. Para ello, existen diferentes políticas para determinar cómo se produce el reemplazo de instancias EC2:

- **Actualizaciones continuas (*Rolling updates*).** Los cambios en la aplicación se aplican en lotes, manteniendo un mínimo de instancias en ejecución que sirven tráfico en todo momento. Esta aproximación previene la inactividad durante el proceso de actualización.
- **Actualizaciones inmutables.** Se lanza un grupo de autoescalado temporal fuera del entorno con un conjunto de instancias que tienen la nueva configuración. Estas instancias se ubican tras el balanceador de carga y comienzan a servir tráfico. Cuando están operativas, se mueven al grupo de autoescalado original y se eliminan tanto las instancias antiguas como el grupo de autoescalado temporal.
- **Deshabilitada.** No se evita la inactividad. Se terminan las instancias existentes y se reemplazan con instancias con la nueva configuración. No recomendable para entornos de producción.

18) Para poder visualizar la configuración del entorno creado, introducimos la orden siguiente, sustituyendo el valor resaltado por el nombre del entorno creado:

```
$ eb config <nombre-entorno>
```

Con ello visualizaremos un archivo YAML con todas las opciones configuradas. En concreto, buscaremos una opción llamada `aws:autoscaling:updatepolicy:rollingupdate` que, por defecto trae una configuración como la siguiente:

```
aws:autoscaling:updatepolicy:rollingupdate:
  MaxBatchSize: '1'
  MinInstancesInService: '2'
  PauseTime: null
  RollingUpdateEnabled: 'true'
  RollingUpdateType: Health
  Timeout: PT30M
```

La configuración anterior realiza las actualizaciones continuas del entorno, manteniendo un número mínimo de instancias de 2, y reemplazando las instancias en lotes de 1 instancia.

19) Para probar cómo funciona, vamos a sustituir el fragmento anterior del archivo de configuración por el siguiente:

```
aws:autoscaling:updatepolicy:rollingupdate:
  RollingUpdateEnabled: true
  RollingUpdateType: Immutable
```

Con el cambio anterior, el servicio *AWS Elastic Beanstalk* realizará una actualización inmutable del entorno, por lo que, en nuestro caso, aprovisionará dos instancias nuevas antes de terminar las antiguas.

- 20) Para probar el cambio, vamos a sustituir el tipo de instancia que, actualmente es una *t4g.micro*, por una instancia con más prestaciones como una *t4g.small*. Para ello, localizamos la sección del documento llamada *aws:autoscaling:launchconfiguration* y sustituimos el parámetro *InstanceType* por *t4g.small*, tal y como se muestra:

```
aws:autoscaling:launchconfiguration:
  BlockDeviceMappings: null
  DisableIMDSv1: 'false'
  EC2KeyName: vockey
  IamInstanceProfile: LabInstanceProfile
  ImageId: ami-0bf2f9cd51287e2d3
  InstanceType: t4g.small
  MonitoringInterval: 5 minute
```

Tras salvar el archivo de configuración, *AWS Elastic Beanstalk* comenzará a ejecutar la actualización del entorno mediante una política inmutable. Si nos fijamos en el *log* de eventos podremos comprobar el funcionamiento de dicha política:

```
Printing Status:
2022-11-23 19:48:31 INFO Environment update is starting.
2022-11-23 19:48:41 INFO Immutable deployment policy enabled. Launching one
instance with the new settings to verify health.
2022-11-23 19:49:12 INFO Created temporary auto scaling group awseb-e-9ed23mfc6a-
immutable-stack-AWSEBAutoScalingGroup-169F1432OJAFI.
2022-11-23 19:49:59 INFO Adding new instance(s) (i-0deaf16a6323ea552) to the load
balancer.
2022-11-23 19:50:00 INFO Waiting for instance(s) (i-0deaf16a6323ea552) to pass
health checks.
2022-11-23 19:52:10 INFO Test instance passed health checks. Launching remaining
new instances.
2022-11-23 19:53:19 INFO Added instance [i-0ec16b8261e20a491] to your environment.
2022-11-23 19:54:12 INFO Successfully launched all instances.
2022-11-23 19:54:13 INFO Adding new instance(s) (i-0ec16b8261e20a491) to the load
balancer.
2022-11-23 19:54:13 INFO Waiting for instance(s) (i-0deaf16a6323ea552,i-
0ec16b8261e20a491) to pass health checks.
2022-11-23 19:56:08 INFO Detached new instance(s) from temporary auto scaling
group awseb-e-9ed23mfc6a-immutable-stack-AWSEBAutoScalingGroup-169F1432OJAFI.
2022-11-23 19:56:11 INFO Attached new instance(s) to the permanent auto scaling
group awseb-e-9ed23mfc6a-stack-AWSEBAutoScalingGroup-5PMU4UQTUOH3.
2022-11-23 19:56:38 INFO Starting post-deployment configuration on new instances.
2022-11-23 19:56:49 INFO Instance deployment completed successfully.
2022-11-23 19:56:53 INFO Waiting for post-deployment configuration to complete.
2022-11-23 19:57:23 INFO Updating environment development's configuration
settings.
2022-11-23 19:57:39 INFO Created Auto Scaling launch configuration named: awseb-e-
9ed23mfc6a-stack-AWSEBAutoScalingLaunchConfiguration-sq6VoNSaJzcT
2022-11-23 19:58:27 INFO Deleted Auto Scaling launch configuration named: awseb-e-
9ed23mfc6a-stack-AWSEBAutoScalingLaunchConfiguration-xw7luZkh6ktw
2022-11-23 19:58:58 INFO Deployment succeeded. Terminating old instances and
temporary Auto Scaling group.
2022-11-23 20:01:18 INFO Removed instance [i-0a7ec0d535eeb8f8a] from your
environment.
2022-11-23 20:05:18 INFO Removed instance [i-067c621726ee85671] from your
environment.
2022-11-23 20:06:32 INFO Environment update completed successfully.
```

## ESTRATEGIAS DE DESPLIEGUE EN AWS ELASTIC BEANSTALK

Una de las características más importantes de *AWS Elastic Beanstalk* es, precisamente, que va a realizar la tarea de despliegue de las nuevas versiones de nuestra aplicación de forma completamente administrada.



Podemos mantener diferentes versiones de las aplicaciones que desplegamos en los diferentes entornos que puede administrar *AWS Elastic Beanstalk*. Las versiones de las aplicaciones se almacenan comprimidas en formato ZIP en un bucket de *Amazon S3* (que puede crearse previamente o dejar que el servicio lo cree directamente).

21) Para poder visualizar las versiones de la aplicación, podemos introducir la orden:

```
$ eb appversion
```

Tras ello podremos ver un resultado similar al siguiente:

development Application Name: workshop					
Environment Status: Ready Health Green					
Current version # deployed: 1					
#	Version Label	Date Created	Age	Description	appversion
1	app-322d-221123_192857000903	2022/11/23 19:28	52 mins	Subidos algunos ficheros	
(Commands: Quit, Delete, Lifecycle, ▼ ▲ ◀ ▶)					

22) Supongamos que deseamos desplegar una nueva versión de la aplicación. Para ello, haremos un pequeño cambio en el archivo `styles.css` y, en la línea 13, cambiaremos la propiedad CSS `background-color:blue` y guardaremos los cambios.

```
.navtop {
    background-color: blue;
    height: 60px;
    width: 100%;
    border: 0;
```

23) Por defecto, cuando creamos una nueva versión de nuestra aplicación en *AWS Elastic Beanstalk*, se considera la última versión del repositorio, por lo que haremos un commit para crear una nueva versión. Para ello introducimos las órdenes:

```
$ git add styles.css
$ git commit -m "Modificado styles.css"
```

24) Una vez confirmados los cambios en el sistema de control de versiones, procedemos a crear la nueva versión de la aplicación, para lo que ejecutamos la orden:

```
$ eb appversion --create
```

Obtendremos un resultado como el siguiente, donde aparece la etiqueta (*label*) de la última versión. La copiamos porque la utilizaremos más adelante:

```
Creating application version archive "app-aa08-221123_202937577284".
Uploading workshop/app-aa08-221123_202937577284.zip to S3. This may take a while.
Upload Complete.
```

25) Si ejecutamos de nuevo la orden `eb appversion`, veremos la nueva versión de nuestra aplicación:

development Application Name: workshop					
Environment Status: Ready Health Green					
Current version # deployed: 1					
#	Version Label	Date Created	Age	Description	appversion
2	app-aa08-221123_202937577284	2022/11/23 20:29	13 mins	Modificado styles.css	
1	app-322d-221123_192857000903	2022/11/23 19:28	1 hour	Subidos algunos ficheros	
(Commands: Quit, Delete, Lifecycle, ▼ ▲ ◀ ▶)					



*AWS Elastic Beanstalk* permite desplegar las aplicaciones web mediante diferentes estrategias que pueden suponer una mayor o menor disponibilidad de la aplicación durante el despliegue, mayor o menor tiempo de despliegue, o facilitar o no los *rollbacks* en caso de fallo.

Las estrategias de despliegue posibles son:

- **Todas a la vez** (*All at once*). Despliega la versión de la aplicación en todas las instancias EC2 de forma simultánea. No es apropiada para un entorno de producción ya que supone una inactividad importante. Es la opción por defecto.
- **Continua** (*Rolling*). Despliega la versión de la aplicación por lotes. Cada lote se saca fuera de servicio durante el despliegue, reduciendo la capacidad computacional del entorno en número de instancias EC2 igual al tamaño del lote.
- **Continua con un lote adicional** (*Rolling with additional batch*). Despliega la versión por lotes, pero en primer lugar se lanza un nuevo lote de instancias para asegurar la completa capacidad computacional del entorno durante el despliegue.
- **Inmutable** (*Immutable*). Despliega la nueva versión en un grupo nuevo de instancias realizando una actualización inmutable (la misma estrategia que en el apartado E de este *workshop*)
- **División de tráfico** (*Traffic Splitting*). Despliega la nueva versión a nuevo grupo de instancias y, temporalmente, divide el tráfico entrante a la aplicación entre la versión nueva de la aplicación y la existente.

26) En este caso, vamos a desplegar nuestra aplicación utilizando la estrategia de *División de tráfico*; para ello editaremos de nuevo la configuración del entorno mediante la orden:

```
$ eb config <nombre-entorno>
```

27) A continuación, buscaremos la opción `aws:elasticbeanstalk:command` y sustituiremos el bloque con la siguiente configuración:

```
aws:elasticbeanstalk:command:
  DeploymentPolicy: TrafficSplitting
```

Además, añadiremos un nuevo bloque a continuación con el siguiente contenido

```
aws:elasticbeanstalk:trafficsplitting:
  EvaluationTime: '5'
  NewVersionPercent: '10'
```

Con la política anterior, estaremos implementando un despliegue *canary* en el que, durante 5 minutos, estaremos enviando un 10% del tráfico a la nueva versión de la aplicación, y el 90% del tráfico restante a la versión existente.

Una vez guardamos el archivo de configuración, se aplicarán los cambios.

28) Ahora sólo nos resta desplegar la nueva versión de la aplicación y comprobar cómo se comporta la estrategia de despliegue. Para ello ejecutamos la orden siguiente, sustituyendo los parámetros resaltados por el nombre del entorno y la etiqueta de la versión de la aplicación, respectivamente:

```
$ eb deploy <nombre-entorno> --version <etiqueta-version>
```

Podemos visualizar los *logs* de los eventos que van sucediendo:

```
2022-11-23 21:01:50 INFO Environment update is starting.
2022-11-23 21:02:00 INFO Traffic Splitting deployment policy enabled. Preparing a
new fleet for the new application version.
2022-11-23 21:02:31 INFO Created temporary auto scaling group awseb-e-9ed23mfc6a-
immutable-stack-AWSEBAutoScalingGroup-1T3QREKX45X0N.
```

```

2022-11-23 21:03:17 INFO Traffic-splitting deployment: successfully launched 2
test instance(s).
2022-11-23 21:03:19 INFO Traffic-splitting deployment: starting to shift 10% of
client traffic to the new application version.
2022-11-23 21:03:19 INFO Traffic-splitting deployment: routed 10% of client
traffic to the new application version.
2022-11-23 21:03:40 INFO Traffic-splitting deployment: all instances that have the
new application version passed load balancer health checks.
2022-11-23 21:03:40 INFO Traffic-splitting deployment: starting evaluation time
for 5 minute(s).
2022-11-23 21:03:40 INFO Waiting for instance(s) (i-0cacb6cc3def8c245,i-
05390052426f8c46b) to pass health checks.
2022-11-23 21:05:36 INFO Traffic-splitting deployment: all instances passed
enhanced health check.
2022-11-23 21:09:37 INFO Traffic-splitting deployment: evaluation is complete.
2022-11-23 21:09:37 INFO Traffic-splitting deployment: starting to shift all
client traffic to the new application version.
2022-11-23 21:09:37 INFO Traffic-splitting deployment: routed all client traffic
to the new application version.
2022-11-23 21:09:38 INFO Detached old instance(s) from environment auto scaling
group awseb-e-9ed23mfc6a-stack-AWSEBAutoScalingGroup-5PMU4UQTUOH3.
2022-11-23 21:10:08 INFO Detached new instance(s) from temporary auto scaling
group awseb-e-9ed23mfc6a-immutable-stack-AWSEBAutoScalingGroup-1T3QREKX45XON.
2022-11-23 21:10:12 INFO Attached new instance(s) to the permanent auto scaling
group awseb-e-9ed23mfc6a-stack-AWSEBAutoScalingGroup-5PMU4UQTUOH3.
2022-11-23 21:10:21 INFO Starting post-deployment configuration on new instances.
2022-11-23 21:10:31 INFO Instance deployment completed successfully.
2022-11-23 21:10:36 INFO Waiting for post-deployment configuration to complete.
2022-11-23 21:13:17 INFO New application version was deployed to running EC2
instances.
2022-11-23 21:13:17 INFO Environment update completed successfully.
2022-11-23 21:14:14 INFO Environment health has transitioned from Info to Ok.
Application update completed 67 seconds ago and took 10 minutes. 2 instances online
which meets Auto Scaling group desired capacity of 2.
2022-11-23 21:15:14 INFO Removed instances [i-0deaf16a6323ea552, i-
0ec16b8261e20a491] from your environment.

```

- 29) Si vamos accediendo al entorno desde el navegador y refrescamos muchas veces podremos ver que, eventualmente, el tráfico impacta sobre la nueva versión de la aplicación:



## CIFRADO DE DATOS EN TRÁNSITO

*AWS Elastic Beanstalk* crea de forma automática (aunque es configurable) los grupos de seguridad para la capa del balanceador de carga y para la capa computacional (instancias EC2) atendiendo a las mejores prácticas de seguridad según el Marco de la Buena Arquitectura de AWS.

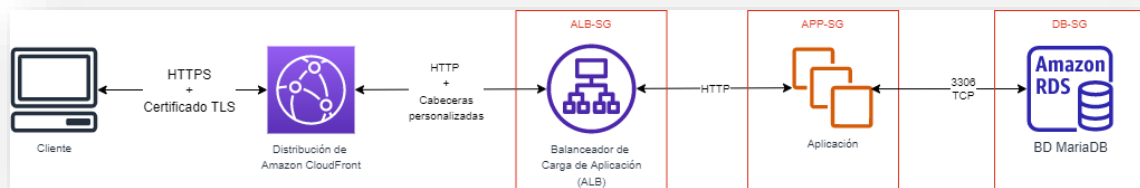
Sin embargo, el eslabón débil en la cadena de seguridad de nuestra infraestructura es precisamente el protocolo HTTP que está utilizando el agente de escucha del balanceador de carga. Para un mayor nivel de seguridad en nuestra aplicación, debemos garantizar el cifrado de los datos en tránsito y utilizar un protocolo HTTPS con certificados TLS. Para ello, tenemos dos posibilidades:

- (a) Si disponemos de un certificado emitido por una CA (*Certificate Authority*) de confianza o un certificado autofirmado, podremos importarlo con *AWS Certificate Manager* y poder asociarlo a nuestro ALB. Dado el carácter educativo de este taller, es bastante probable que en las aulas no

dispongamos de certificados emitidos por una CA de confianza, y en caso de disponer de un certificado autofirmado, ésta tampoco sería una opción muy deseable desde el punto de vista de la seguridad.

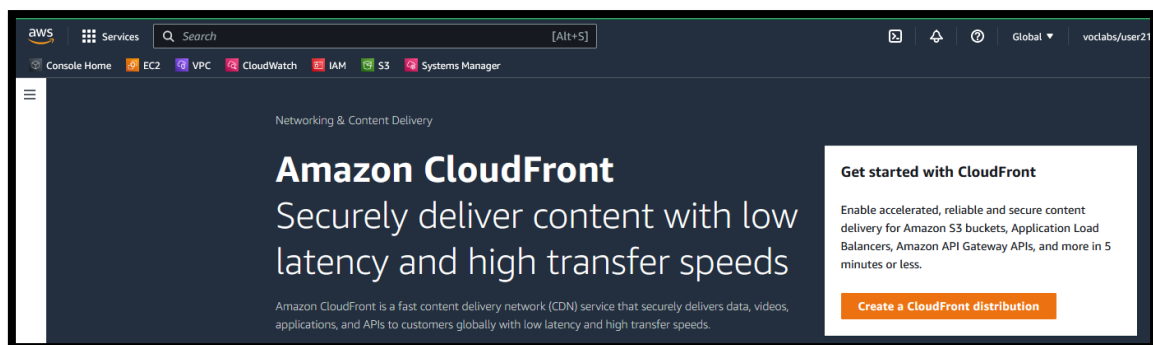
- (b) Utilizar una distribución de *Amazon CloudFront* delante de nuestro ALB que utilice un certificado TLS emitido por una CA de confianza para forzar el cifrado de la información en tránsito entre el cliente y la propia distribución de CloudFront con el protocolo HTTPS. En este caso se deberá garantizar que todo el tráfico pasa, necesariamente, por la distribución de CloudFront y no se pueda acceder directamente al ALB. Para ello, se puede definir que CloudFront envíe una cabecera personalizada con un valor secreto al ALB, y que el ALB sólo acepte las solicitudes HTTP que lleven dicha cabecera con el valor secreto elegido.

En este taller implementaremos esta última opción, de forma que nuestra defensa en profundidad quedaría reflejada en el siguiente esquema:

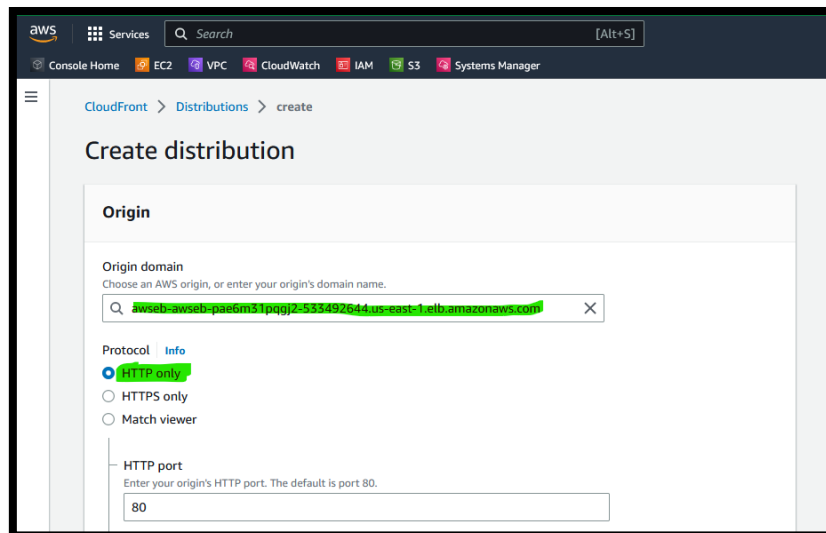


**NOTA:** Lo ideal sería incluir un agente de escucha en el ALB que soporte un certificado emitido por una CA de confianza, junto con la distribución de *Amazon CloudFront*; sin embargo, esta posibilidad no es viable en el aula ya que *Amazon CloudFront*, a día de hoy, sólo soporta la integración con ALBs configurados con un agente de escucha HTTPS con un certificado emitido por una CA de confianza.

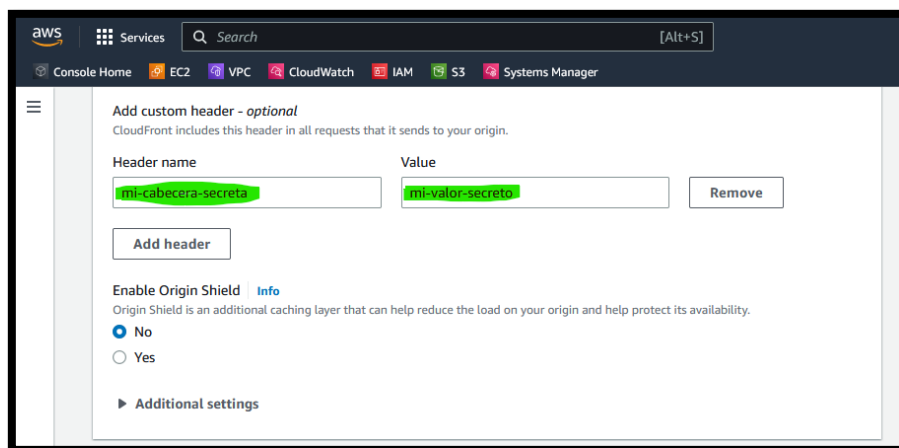
- 30) Para crear una distribución de, lo haremos a través de la Consola de Administración de AWS. Para ello abrimos la consola del servicio de *Amazon CloudFront* y presionamos el botón **Create a CloudFront distribution**:



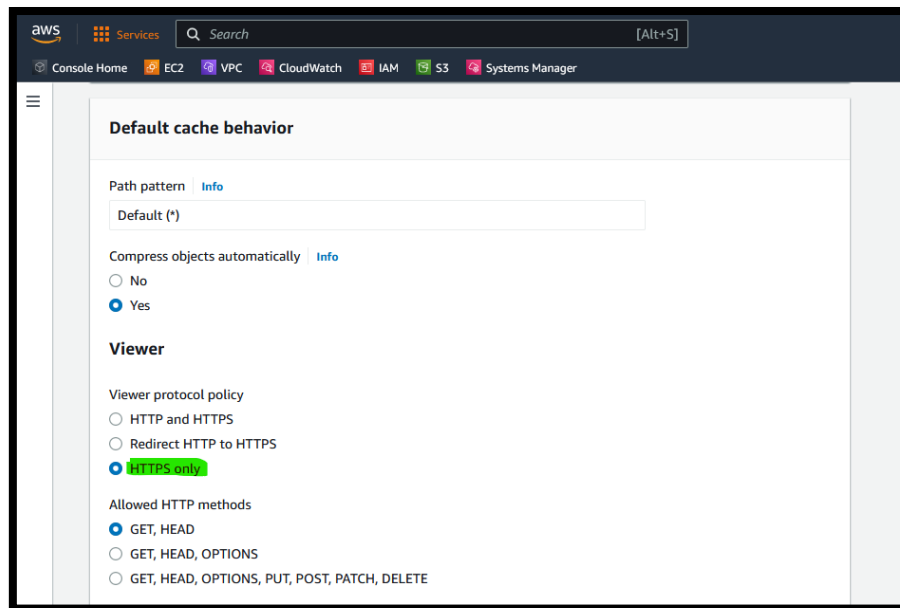
- 31) Desde la ventana de la creación de la distribución, seleccionaremos como dominio de origen nuestro balanceador de carga de aplicación (ALB) y seleccionaremos el protocolo HTTP que es el que acepta el agente de escucha del ALB:



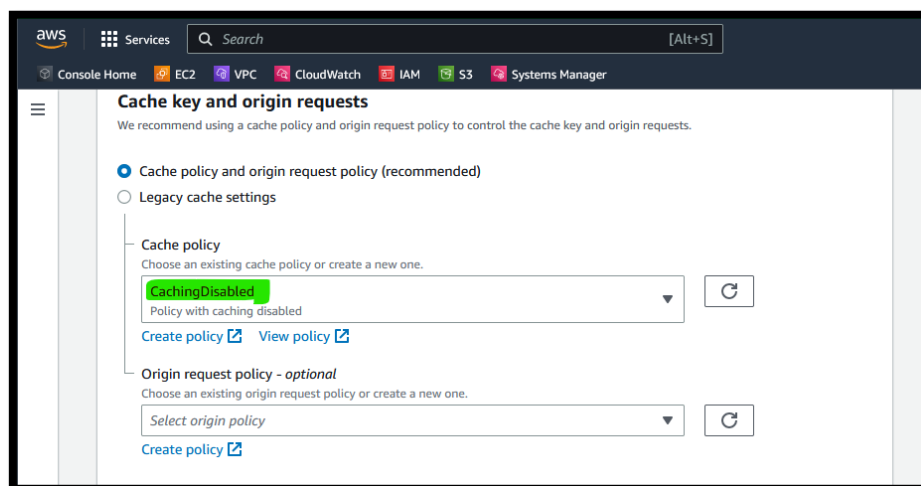
- 32) En el apartado de las cabeceras personalizadas, aprovecharemos para enviar una cabecera HTTP al ALB con un valor secreto. Posteriormente configuraremos el ALB para que sólo atienda las solicitudes HTTP que lleven dicha cabecera con el valor apropiado:



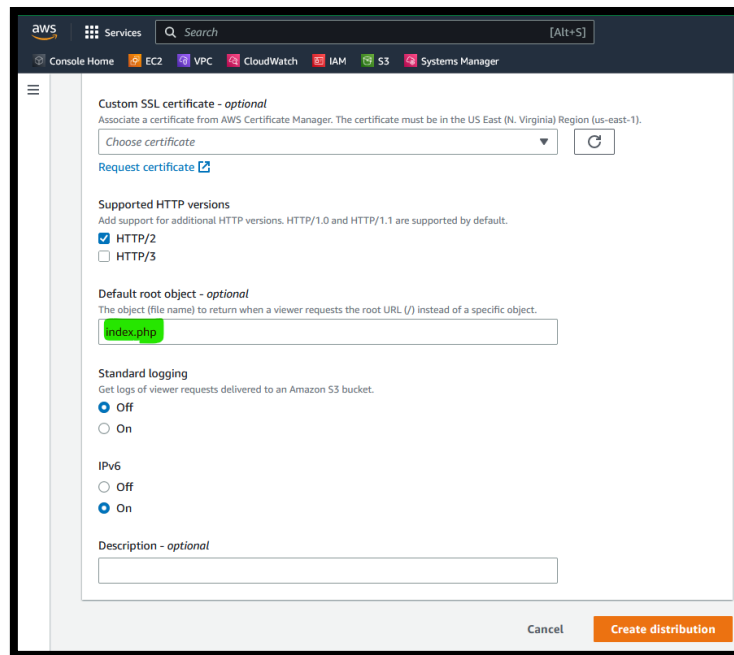
- 33) En el comportamiento de la caché por defecto, haremos que nuestra distribución de CloudFront sólo acepte solicitudes HTTPS desde los usuarios finales, para lo cual seleccionaremos la opción marcada:



- 34) En el apartado de solicitudes al origen y claves de caché, deshabilitaremos la política de caché, ya que estamos cacheando contenido dinámico:



- 35) Por último, en el apartado de configuración, indicaremos el nombre del objeto raíz, que se devolverá cuando se solicite el acceso a la URL raíz. En este caso indicaremos el archivo `index.php` de nuestra aplicación. Por último, presionaremos el botón **Create distribution**, para terminar el proceso.



Custom SSL certificate - optional

Associate a certificate from AWS Certificate Manager. The certificate must be in the US East (N. Virginia) Region (us-east-1).

Choose certificate

[Request certificate](#)

Supported HTTP versions

Add support for additional HTTP versions. HTTP/1.0 and HTTP/1.1 are supported by default.

☒ HTTP/2

☐ HTTP/3

Default root object - optional

The object (file name) to return when a viewer requests the root URL (/) instead of a specific object.

Standard logging

Get logs of viewer requests delivered to an Amazon S3 bucket.

☒ Off

☐ On

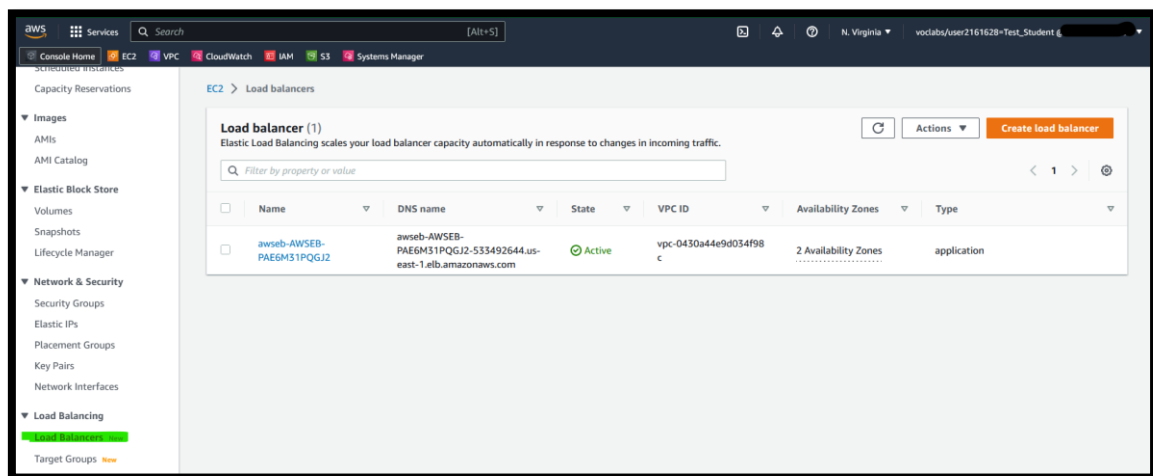
IPv6

☐ Off

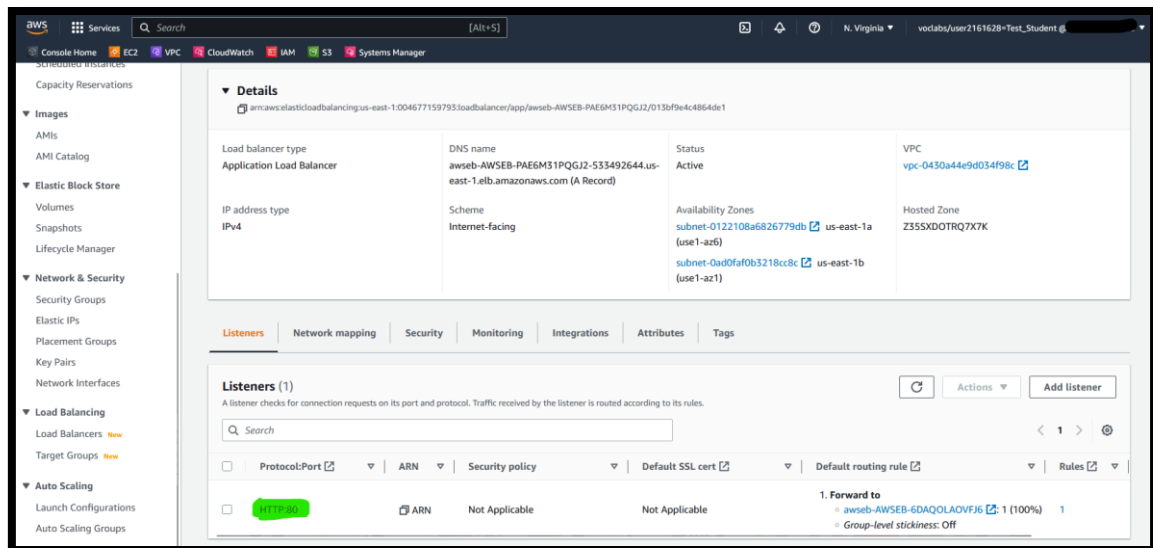
☒ On

Description - optional

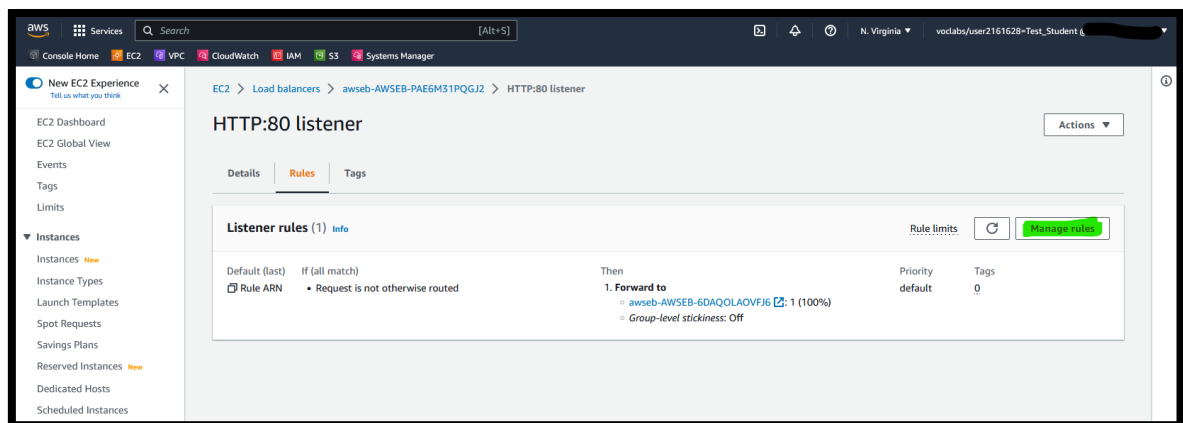
36) Tras unos 10 minutos, tendremos nuestra distribución preparada. Mientras tanto, vamos a modificar la configuración del agente de escucha de nuestro ALB para que sólo acepte las solicitudes que lleven incluida la cabecera especificada anteriormente. Para ello, desde la consola de Amazon EC2, seleccionamos la opción **Load Balancers**:



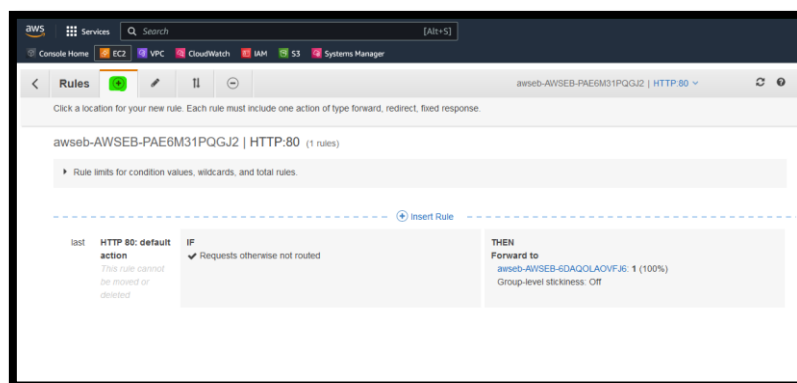
37) Seleccionamos nuestro ALB y hacemos clic en el listener HTTP:80:



38) Desde la siguiente ventana, seleccionamos la pestaña de **Rules** y presionamos el botón **Manage rules**:

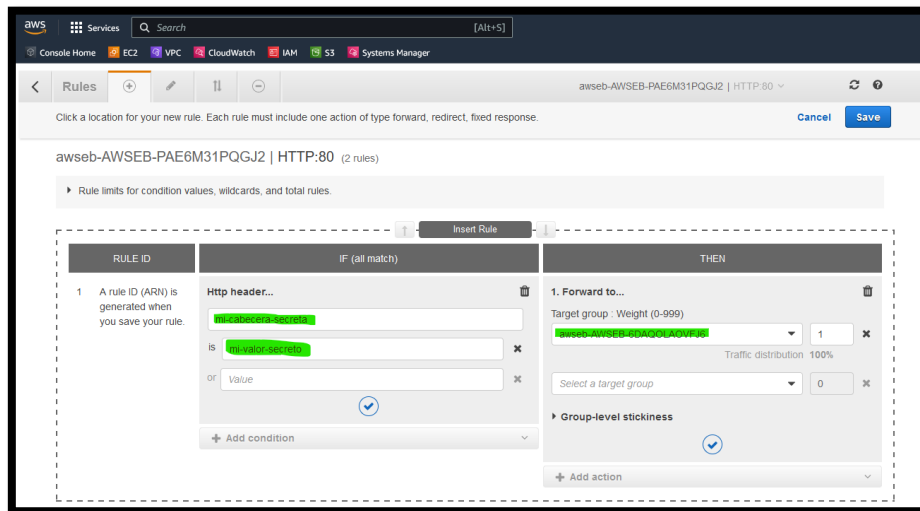


39) A continuación, añadimos una nueva regla al agente de escucha presionando el botón "+":

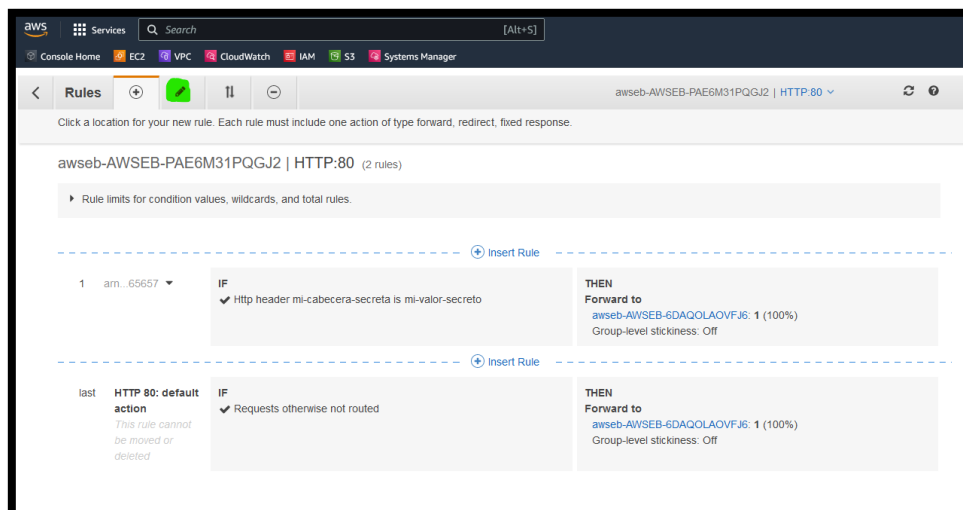


40) Presionamos el botón **Insert Rule** y añadimos una condición **Http header** rellenando con la información de la cabecera personalizada que enviará la distribución de CloudFront. En la parte

derecha de la regla, seleccionamos la opción **Forward to** y elegimos el grupo de destinos existente y presionamos el botón **Save**:

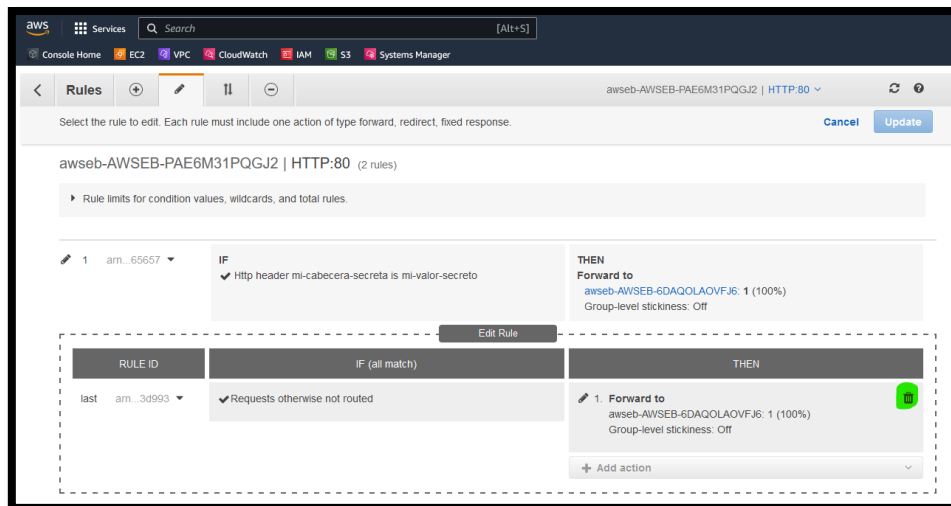


- 41) Por último, debemos editar la regla por defecto, en este caso deberemos entregar un mensaje de error de la serie HTTP 5XX evitando el acceso a la aplicación. Para ello editamos las reglas, presionando el botón del “lápiz”:

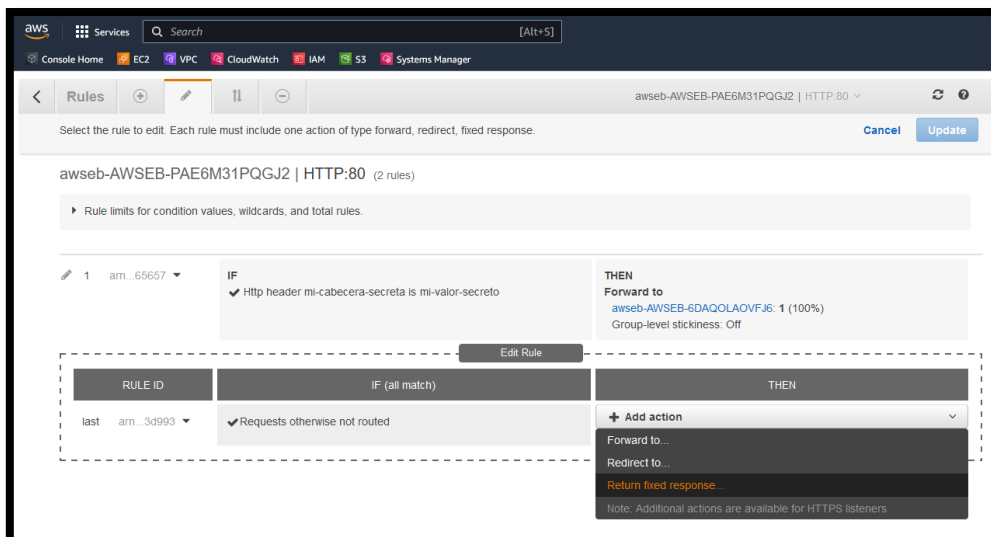


- 42) Presionamos de nuevo el botón del “lápiz” en la regla por defecto y eliminamos la parte derecha de la regla haciendo clic en el botón de la “papelera”:

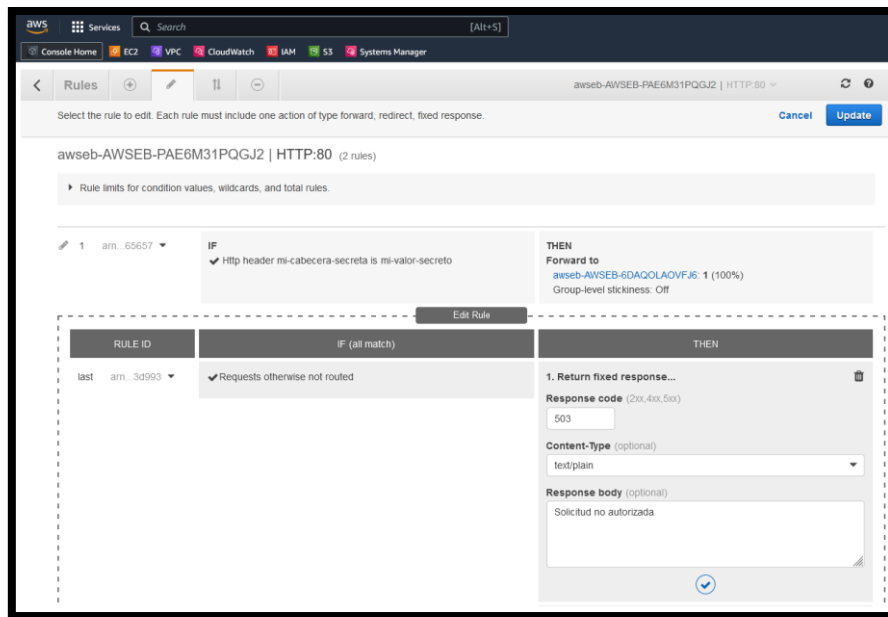




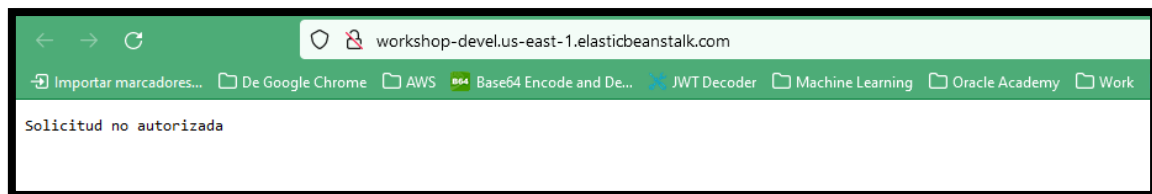
- 43) A continuación, añadimos una nueva acción, pulsando el botón **Add action** seleccionando la opción **Return fixed response**:



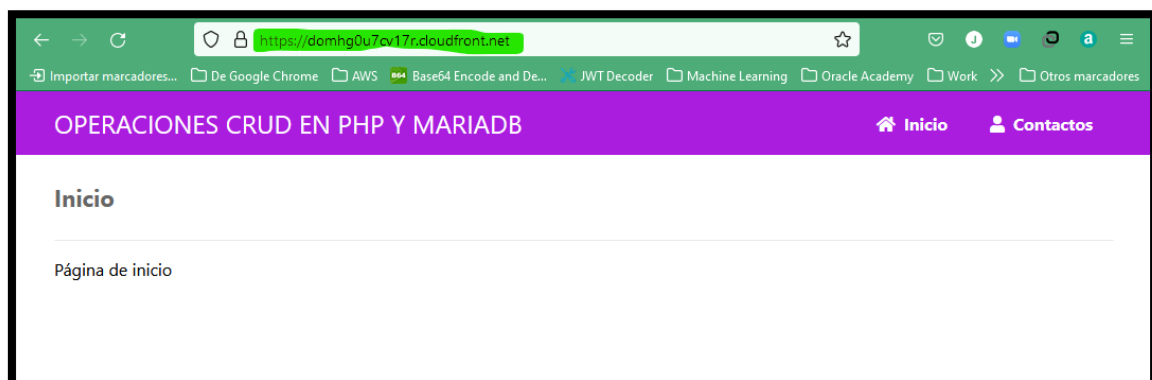
- 44) Por último, seleccionamos el código de estado correspondiente al error HTTP 503 y en el campo del cuerpo de la respuesta escribimos el mensaje de error que devolverá al cliente. Para terminar, presionamos el botón **Update**:



- 45) Si accedemos ahora al nombre DNS de nuestra pila de aplicación desde *AWS Elastic Beanstalk* comprobaremos que nos da el error correspondiente:

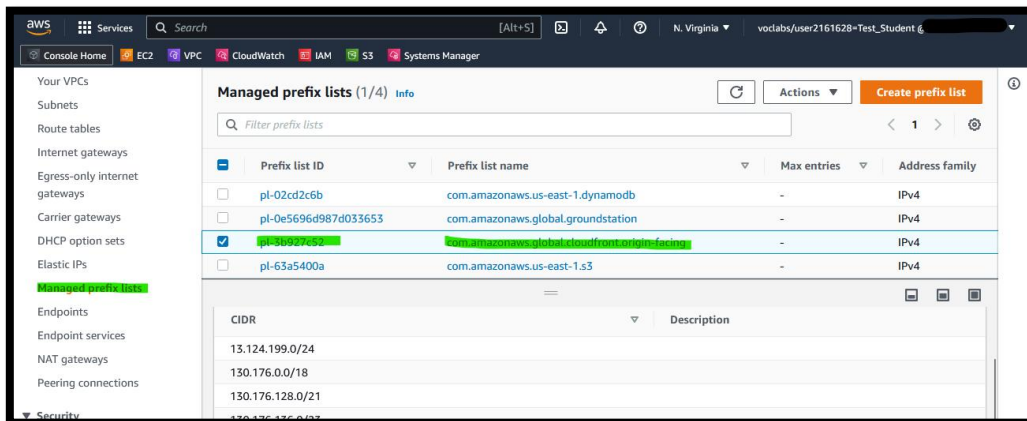


- 46) Volviendo a la distribución de *Amazon CloudFront* creada anteriormente, copiamos el nombre de dominio de nuestra distribución y, abriendo una ventana nueva del navegador, accedemos por HTTPS a dicho dominio, esta vez con el tráfico cifrado en tránsito mediante un certificado TLS emitido por Amazon:



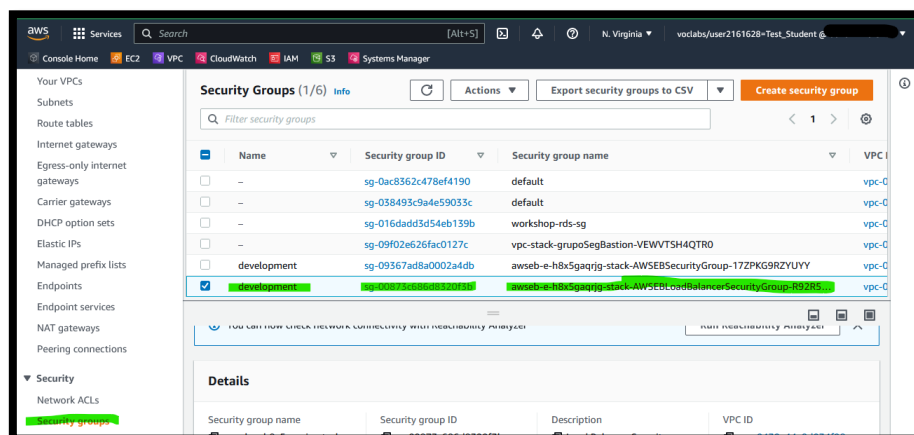
- 47) Para terminar de asegurar nuestra infraestructura vamos a permitir que el ALB únicamente acepte solicitudes de *Amazon CloudFront*. El servicio de *Amazon VPC* mantiene una lista completamente administrada de prefijos de diferentes servicios de AWS, entre ellos *Amazon CloudFront*, *Amazon S3* y *Amazon DynamoDB*. Podemos acceder a estos prefijos desde la consola

de Amazon VPC en la sección **Managed prefix lists**. Desde allí podremos ver las direcciones IPs asignadas al servicio de *Amazon CloudFront*:

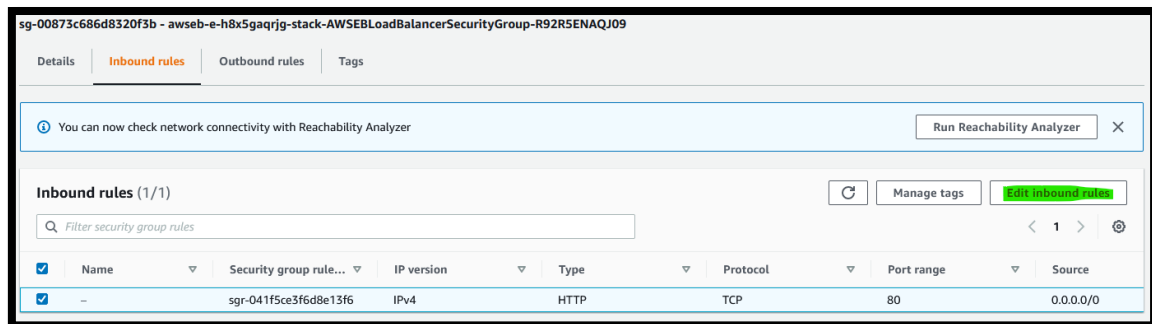


48) Copiamos el ID de la lista de prefijos correspondiente al servicio de *Amazon CloudFront*.

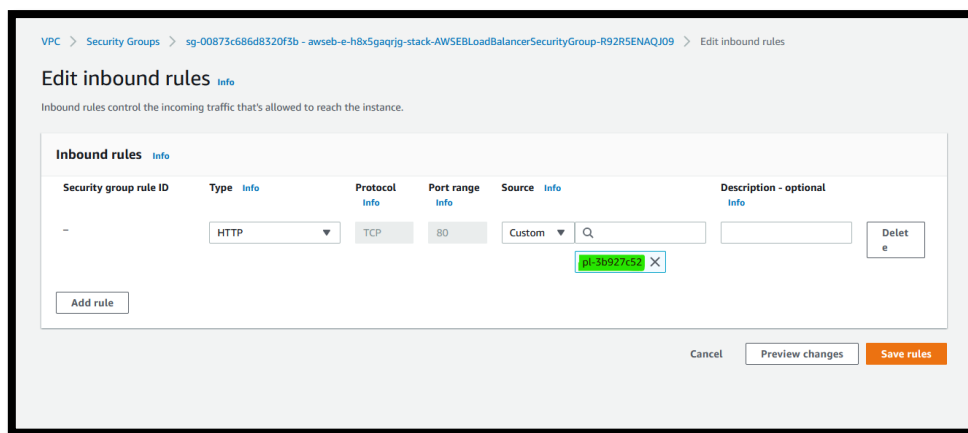
49) Podemos aprovechar estos prefijos y modificar el grupo de seguridad del ALB para que sólo acepte solicitudes que provengan de los prefijos de *Amazon CloudFront*. Para ello, desde la consola del servicio de Amazon VPC elegimos la opción **Security Groups** y seleccionamos el grupo de seguridad asignado al ALB (es fácil reconocerlo porque como parte de su nombre aparece *AWSEBLoadBalancer*):



50) A continuación, en la parte inferior seleccionamos la pestaña **Inbound rules** y presionamos el botón **Edit inbound rules**:



- 51) En la siguiente ventana, eliminamos la regla existente y añadimos una nueva regla que permita el tráfico HTTP, pegando la lista de prefijos de Amazon CloudFront como **Source**. Por último, presionamos el botón **Save rules**:



- 52) Tras todas estas acciones, habremos garantizado que nuestro ALB sólo aceptará solicitudes de *Amazon CloudFront* y además dichas solicitudes deben tener una cabecera HTTP específica, definida en la propia distribución de *CloudFront*.
- 53) Es posible establecer medidas adicionales de seguridad como el cifrado a nivel de campo (*field-level encryption*), que permite que el servicio *Amazon CloudFront* cifre en el borde los campos de una solicitud HTTP con un sistema de cifrado asimétrico RSA. Lamentablemente, este tipo de medida de seguridad queda fuera del alcance de este *workshop*.

### Limpieza de la Práctica:

Es importante eliminar adecuadamente el entorno creado para no incurrir en costes adicionales una vez concluido este *workshop*.

- Para proceder a la limpieza del entorno, en primer lugar, eliminaremos nuestra instancia de RDS. Para ello ejecutamos la orden:

```
$ aws rds delete-db-instance --db-instance-identifier workshop-rds --skip-final-snapshot
--region us-east-1
$ aws rds wait db-instance-deleted --region us-east-1
```

- La última orden ejecutada esperará a que la instancia esté completamente eliminada. Transcurridos unos minutos, cuando recuperemos el control del prompt, eliminamos el grupo de seguridad asociado a la instancia RDS ejecutando la orden:

```
$ aws ec2 delete-security-group --group-id $RDS_SG --region us-east-1
```

- El siguiente paso es eliminar el grupo de subredes, para lo que ejecutamos la orden:

```
$ aws rds delete-db-subnet-group --db-subnet-group-name workshop-subnet-group --region us-east-1
```

- A continuación, eliminamos el parámetro de AWS Systems Manager Parameter Store mediante la orden:

```
$ aws ssm delete-parameter --name RDS-secret --region us-east-1
```

- A continuación, eliminamos nuestra aplicación y el entorno creados, para lo que ejecutamos la instrucción y seguimos el asistente:

```
$ eb terminate -all
```

- Una vez se haya completado la eliminación de la aplicación de AWS Elastic Beanstalk, eliminamos la infraestructura de red mediante la orden, sustituyendo el valor resaltado por el nombre de la pila utilizado en el proceso de creación:

```
aws cloudformation delete-stack --stack-name <nombre-pila> --region us-east-1
```

- Por último, deshabilitamos y eliminamos la distribución de CloudFront que hemos creado desde la consola de Amazon CloudFront.