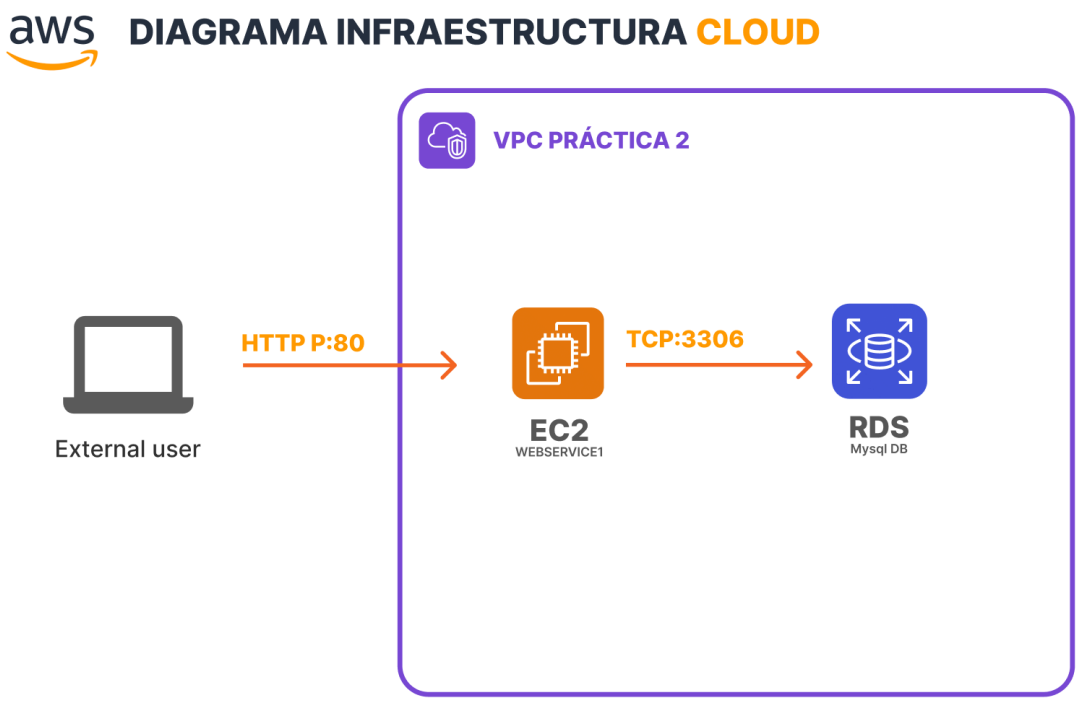
Actividad opcional

Introducción

La actividad opcional consistirá en el despliegue de nuestra API creada en la práctica 1 en una instancia EC2 utilizando Docker y CloudFormation. Para ello, crearemos un CloudFormation que contenga una instancia EC2 que descargue el código de la API desde GitHub utilizando un comando ***git pull****, y una base de datos RDS que funcionará como la capa de persistencia de datos. La instancia EC2 se encargará de ejecutar Docker, que contendrá nuestra API. Además, la base de datos RDS será utilizada para almacenar y gestionar los datos necesarios para el funcionamiento de la API. De esta manera, el entorno se automatiza completamente, asegurando que tanto la infraestructura como la aplicación estén listas para su ejecución con una sola plantilla.*



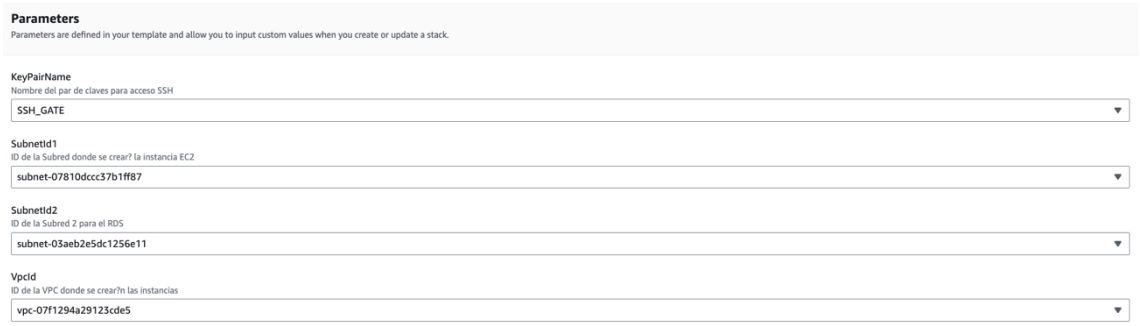
Fichero YAML

```yaml
AWSTemplateFormatVersion: "2010-09-09"
Description: Despliegue de API con Docker y RDS en EC2 usando CloudFormation.

Parameters:
  KeyPairName:
    Description: "Nombre del par de claves para acceso SSH"
    Type: AWS::EC2::KeyPair::KeyName
  VpcId:
    Description: "ID de la VPC donde se crearán las instancias"
    Type: AWS::EC2::VPC::Id
  SubnetId1:
    Description: "ID de la Subred donde se creará la instancia EC2"
    Type: AWS::EC2::Subnet::Id
  SubnetId2:
    Description: "ID de la Subred 2 para el RDS"
    Type: AWS::EC2::Subnet::Id

Resources:
  # Security Group para el EC2 y la API
  ApiSecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: "Permitir acceso a la API y SSH"
      VpcId: !Ref VpcId
      SecurityGroupIngress:
        - IpProtocol: "tcp"
          FromPort: "8080"
          ToPort: "8080"
          CidrIp: "0.0.0.0/0"
        - IpProtocol: "tcp"
          FromPort: "22"
          ToPort: "22"
          CidrIp: "0.0.0.0/0"

  # Security Group para el RDS
  RDSecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: "Permitir acceso a MySQL desde EC2"
      VpcId: !Ref VpcId
      SecurityGroupIngress:
        - IpProtocol: "tcp"
          FromPort: "3306"
          ToPort: "3306"
          SourceSecurityGroupId: !GetAtt ApiSecurityGroup.GroupId

  # Instancia RDS MySQL
  MySQLDatabase:
    Type: AWS::RDS::DBInstance
    Properties:
      AllocatedStorage: 20
      DBInstanceClass: db.t3.micro
      Engine: mysql
      EngineVersion: "8.0"
      MasterUsername: appuser
      MasterUserPassword: apppassword
      DBName: DrivingSchoolDB
      VPCSecurityGroups:
        - !Ref RDSecurityGroup
      DBSubnetGroupName: !Ref MySQLSubnetGroup
      PubliclyAccessible: true

  # Subnet Group para el RDS
  MySQLSubnetGroup:
    Type: AWS::RDS::DBSubnetGroup
    Properties:
      DBSubnetGroupDescription: "Subnets for MySQL RDS"
      SubnetIds:
        - !Ref SubnetId1
        - !Ref SubnetId2

  # Instancia EC2 para la API
  ApiServerInstance:
    Type: AWS::EC2::Instance
    DependsOn: MySQLDatabase
    Properties:
      InstanceType: t2.micro
      KeyName: !Ref KeyPairName
      ImageId: ami-0ebfd941bbafe70c6 # Amazon Linux 2
      SecurityGroupIds:
        - !Ref ApiSecurityGroup
      SubnetId: !Ref SubnetId1
      UserData:
        Fn::Base64: !Sub |
          #!/bin/bash
          # Actualizamos los paquetes
          sudo dnf update -y

          # Instalamos Docker y Git
          sudo dnf install docker -y
          sudo dnf install git -y
          sudo service docker start
          sudo usermod -a -G docker ec2-user

          # Clonamos el repositorio de GitHub
          sudo git clone https://github.com/EduardoOrtegaZerpa/FleetManager.git

          # Construimos la imagen Docker
          cd FleetManager
          sudo docker build -t api .

          # Ejecutamos el contenedor Docker de la API con las variables de entorno
          sudo docker run -d -p 8080:8080 \
            -e "DB_HOST=${MySQLDatabase.Endpoint.Address}" \
            -e "DB_NAME=DrivingSchoolDB" \
            -e "DB_USER=appuser" \
            -e "DB_PASS=apppassword" \
            -e "spring.application.name=Pepeducacion" \
            -e "server.port=8080" \
            -e
"spring.datasource.url=jdbc:mysql://${MySQLDatabase.Endpoint.Address}:3306/DrivingS
choolDB" \
            -e "spring.datasource.username=appuser" \
            -e "spring.datasource.password=apppassword" \
            -e "spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver" \
            -e "spring.jpa.hibernate.ddl-auto=update" \
            -e "spring.jpa.show-sql=true" \
            -e
"spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect" \
            api

Outputs:
  WebsiteURL:
    Description: "URL de la aplicación API"
    Value: !Sub "http://${ApiServerInstance.PublicIp}:8080"
```

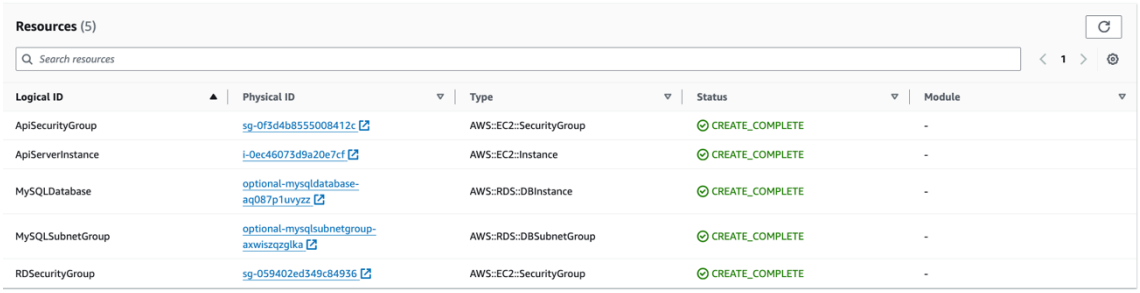
El archivo YAML define la infraestructura en AWS mediante CloudFormation para desplegar una aplicación web basada en Spring Boot. La aplicación se ejecuta en una instancia EC2 que, al iniciarse, utiliza Docker para clonar un repositorio de GitHub, construir una imagen de la aplicación y ejecutarla en un contenedor. Este contenedor se lanza con variables de entorno que configuran parámetros esenciales como el nombre de la aplicación, el puerto del servidor y la conexión a una base de datos MySQL, alojada en RDS. El archivo también incluye la creación de dos grupos de seguridad: uno para la instancia EC2 que permite el acceso SSH y HTTP desde cualquier lugar, y otro para la base de datos MySQL que restringe el acceso únicamente desde la instancia EC2. La instancia RDS se configura con MySQL 8.0 y se conecta a la aplicación utilizando las variables de entorno que contienen las credenciales de la base de datos, el nombre del esquema y el endpoint de la misma. La configuración asegura que la base de datos esté dentro de una subred privada, mientras que la instancia EC2 expone la API en el puerto 8080 para el acceso externo.

Creación del stack

Dado que la creación del stack sigue los mismos pasos que en despliegues anteriores, omitimos las instrucciones comunes y nos enfocamos en las diferencias específicas. La principal diferencia es que, en este caso, hemos configurado una serie de parámetros de entrada personalizables desde la consola de Amazon al crear el stack de CloudFormation. Estos parámetros incluyen el par de claves SSH para acceder a la instancia EC2, las IDs de las subredes donde se desplegarán los recursos, y la ID del VPC, lo que permite adaptar el stack a distintos entornos de red de manera flexible.



Una vez configurados estos parámetros, el resto del proceso se realiza de manera automática y podemos verificar que el despliegue se haya completado correctamente revisando la pestaña de recursos del stack en la consola de AWS, donde se listan todos los elementos desplegados, como la instancia EC2 y la base de datos RDS, asegurando que todo ha sido creado y configurado exitosamente.

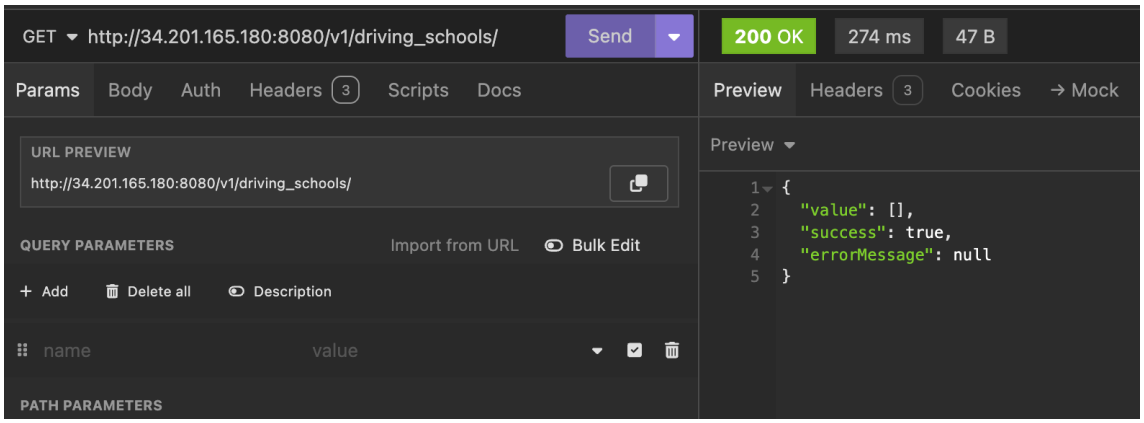


Validación

Para validar el correcto funcionamiento del despliegue, simplemente necesitamos verificar que la API esté operativa y que exista persistencia en la base de datos. Utilizaremos el cliente Insomnia para realizar varias solicitudes HTTP a nuestra API REST. A través de estas peticiones, comprobaremos que los datos persisten correctamente, es decir, que los registros se mantienen en la base de datos tras realizar operaciones como crear, leer o actualizar recursos. Si la API responde adecuadamente a las solicitudes y los datos persisten en memoria, podemos concluir que el despliegue ha sido exitoso y que la conexión con la base de datos MySQL en RDS funciona correctamente.

GET /v1/driving_schools/

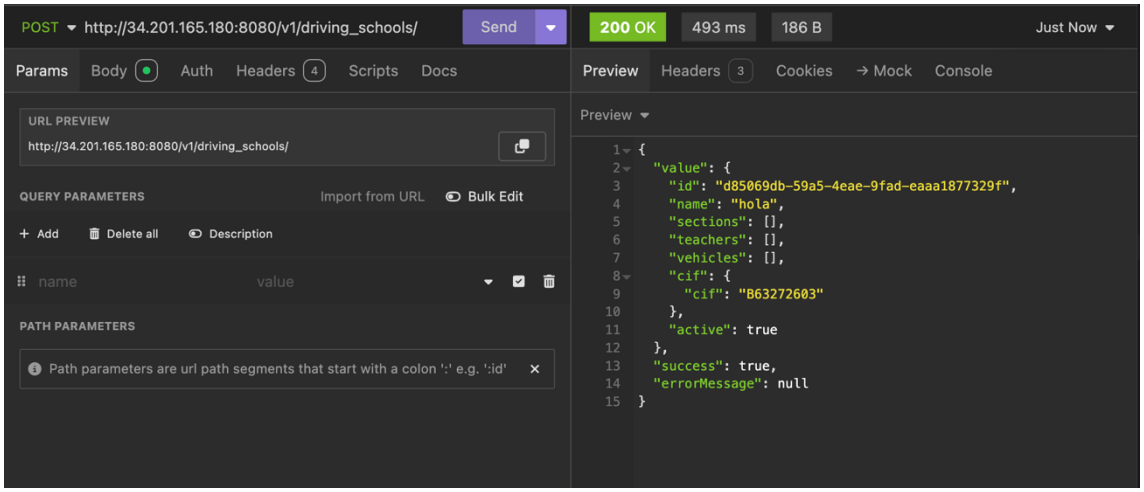
*La petición **GET /v1/driving_schools/** se realiza de manera exitosa, lo que demuestra que la API está funcionando correctamente. Al recibir una respuesta válida, podemos confirmar que la instancia EC2 está ejecutando la aplicación como se esperaba y que el servidor web es accesible desde el exterior. Este resultado también indica que la base de datos está correctamente configurada y la API puede acceder a los datos almacenados, permitiendo validar que el despliegue ha sido exitoso.*



POST /v1/driving_schools/

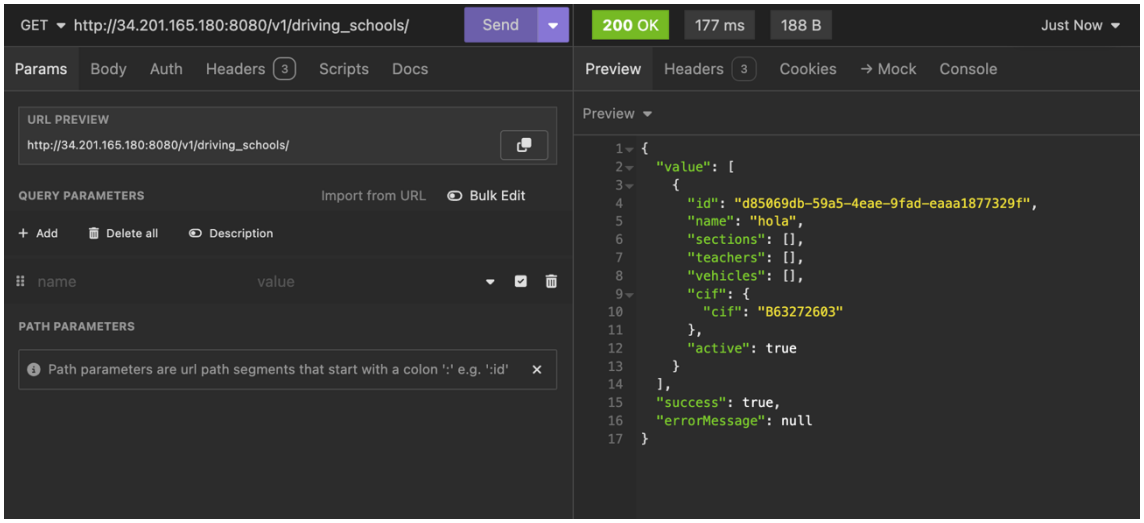
*La petición **POST /v1/driving_schools/** ha sido procesada correctamente, lo que indica que la API está permitiendo la creación de nuevos recursos sin ningún tipo de error. Esto confirma que no solo la API está funcionando bien, sino que también hay una comunicación fluida con la base de datos MySQL en RDS, donde los datos se almacenan de manera persistente. El éxito en la creación de una nueva autoescuela sin errores demuestra que tanto el backend como la lógica de persistencia están*

funcionando correctamente en este entorno desplegado.

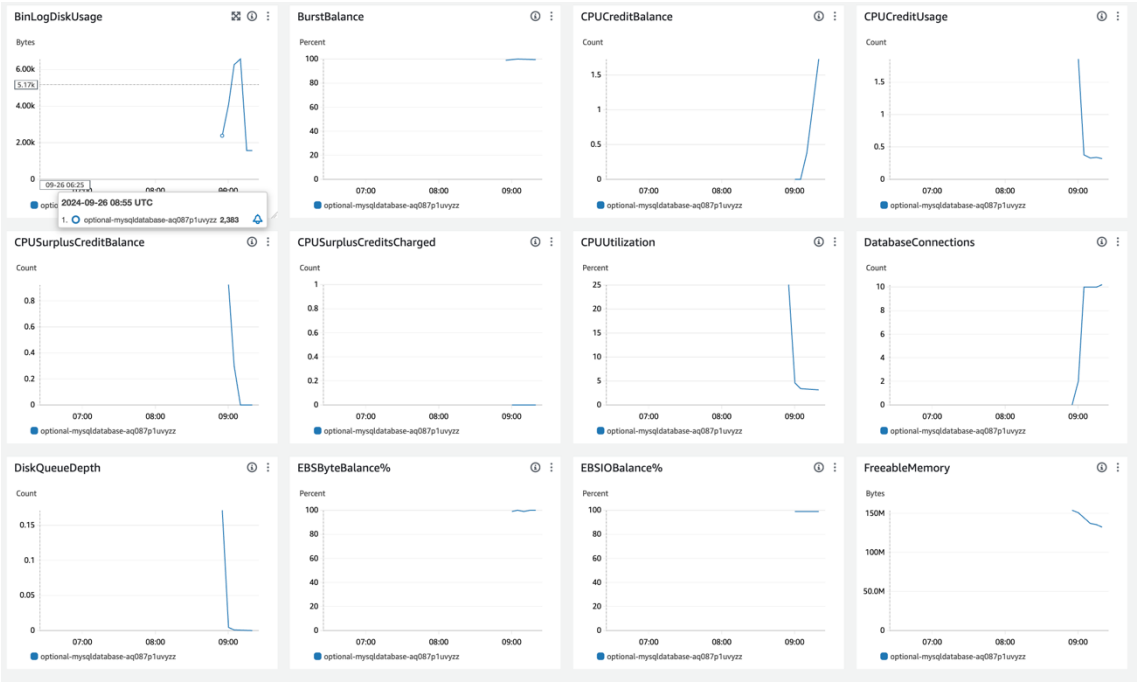


GET /v1/driving_schools/

*Al realizar nuevamente la petición **GET /v1/driving_schools/**, observamos que el listado devuelto está actualizado y muestra la nueva autoescuela creada, lo que confirma que la persistencia de datos funciona correctamente.*



*Como podemos observar en el panel de administración de **RDS**, la base de datos está siendo utilizada activamente, lo cual confirma que **Hibernate** está ejecutando correctamente las operaciones de escritura y lectura en la base de datos. El tráfico y la actividad mostrada en el panel validan que cada vez que se realizan solicitudes a la API, como la creación y recuperación de autoescuelas, Hibernate interactúa adecuadamente con la base de datos MySQL en RDS. Esto ratifica que la configuración de persistencia de datos es funcional y que la conexión entre la API y la base de datos está operando sin problemas.*



Conclusión

*En conclusión, a lo largo de esta práctica hemos aprendido a utilizar **CloudFormation** para automatizar el despliegue de aplicaciones en AWS de manera eficiente. A través de distintos casos de uso, hemos comprendido cómo configurar y desplegar recursos como instancias EC2, bases de datos RDS y grupos de seguridad, todo mediante plantillas de infraestructura como código. Además, hemos implementado una API REST que se ejecuta en un contenedor Docker, asegurando la conexión y persistencia de datos con una base de datos MySQL en RDS. Este proceso nos ha permitido ver el poder de la automatización en AWS y cómo gestionar infraestructuras completas de manera reproducible y escalable.*

Anexo

Actividad 1

Fichero YAML

```
AWSTemplateFormatVersion: "2010-09-09"

Resources:

 # No necesitamos crear una KeyPair aquí, ya que se usa una existente llamada SSH_GATE

 SSHSecurityGroup:

  Type: "AWS::EC2::SecurityGroup"

  Properties:

   GroupDescription: "Enable SSH access from anywhere"

   SecurityGroupIngress:

    - IpProtocol: "tcp"

      FromPort: "22"

      ToPort: "22"

      CidrIp: "0.0.0.0/0"


 WebSecurityGroup:

  Type: "AWS::EC2::SecurityGroup"

  Properties:

   GroupDescription: "Allow HTTP and SSH traffic"

   SecurityGroupIngress:

    - IpProtocol: "tcp"

      FromPort: "80"

      ToPort: "80"

      CidrIp: "0.0.0.0/0"

    - IpProtocol: "tcp"

      FromPort: "22"

      ToPort: "22"
```

```
      CidrIp: "0.0.0.0/0"

  SSHGateInstance:
    Type: "AWS::EC2::Instance"
    Properties:
      InstanceType: "t2.micro"
      KeyName: "SSH_GATE"  # Usamos la KeyPair SSH_GATE existente
      SecurityGroups:
        - !Ref SSHSecurityGroup
      ImageId: "ami-0ebfd941bbafe70c6"  # El nuevo ID de AMI

  WebServerInstance:
    Type: "AWS::EC2::Instance"
    Properties:
      InstanceType: "t2.micro"
      KeyName: "SSH_GATE"  # Usamos la KeyPair SSH_GATE existente
      SecurityGroups:
        - !Ref WebSecurityGroup
      ImageId: "ami-0ebfd941bbafe70c6"  # El nuevo ID de AMI
      UserData:
        Fn::Base64:
          !Sub |
            #!/bin/bash
            yum update -y
            yum -y install httpd
            systemctl enable httpd
            systemctl start httpd
            echo "<html><h1>Instancia Web ID: $(curl http://169.254.169.254/latest/meta-data/instance-id)</h1></html>" > /var/www/html/index.html
```

Actividad 2

Fichero YAML

```
AWSTemplateFormatVersion: "2010-09-09"

Description: Stack con dos instancias EC2 detrás de un Load Balancer y un Auto-Scaling Group.


Parameters:

 KeyName:

  Description: Nombre del par de claves para acceder a las instancias por SSH

  Type: String

  Default: "SSH_GATE" # Cambia esto por el nombre de tu key pair


Resources:

 # Grupo de seguridad para el Load Balancer

 LoadBalancerSG:

  Type: AWS::EC2::SecurityGroup

  Properties:

   GroupDescription: Seguridad para el Load Balancer

   VpcId: "vpc-07f1294a29123cde5"

   SecurityGroupIngress:

    - IpProtocol: tcp

      FromPort: 80

      ToPort: 80

      CidrIp: 0.0.0.0/0


 # Load Balancer (ALB)

 LoadBalancer:

  Type: AWS::ElasticLoadBalancingV2::LoadBalancer

  Properties:

   Subnets:
```

```
      - subnet-0ba2ce3373b4a615c
      - subnet-0c60fabd025f28af3
    SecurityGroups:
      - !Ref LoadBalancerSG

# Listener para el ALB
HTTPListener:
  Type: AWS::ElasticLoadBalancingV2::Listener
  Properties:
    DefaultActions:
      - Type: forward
        TargetGroupArn: !Ref WebTargetGroup
    LoadBalancerArn: !Ref LoadBalancer
    Port: 80
    Protocol: HTTP

# Grupo de destino (Target Group) para los servidores web
WebTargetGroup:
  Type: AWS::ElasticLoadBalancingV2::TargetGroup
  Properties:
    VpcId: "vpc-07f1294a29123cde5"
    Port: 80
    Protocol: HTTP
    TargetType: instance
    HealthCheckProtocol: HTTP
    HealthCheckPort: "80"
    HealthCheckPath: "/"
```

```
  # Auto Scaling Launch Configuration
  WebLaunchConfig:
    Type: AWS::AutoScaling::LaunchConfiguration
    Properties:
      ImageId: "ami-0ebfd941bbafe70c6"
      InstanceType: t2.micro
      SecurityGroups:
        - !Ref LoadBalancerSG
      KeyName: !Ref KeyName
      UserData:
        Fn::Base64: !Sub |
          #!/bin/bash
          yum update -y
          yum -y install httpd
          systemctl enable httpd
          systemctl start httpd
          TOKEN=$(curl -X PUT "http://169.254.169.254/latest/api/token" -H "X-aws-ec2-metadata-token-ttl-seconds: 21600")
          echo "<html><h1>Servidor Web - Instance ID: $(curl -H 'X-aws-ec2-metadata-token: $TOKEN' http://169.254.169.254/latest/meta-data/instance-id)</h1><p>IP: $(curl http://169.254.169.254/latest/meta-data/local-ipv4)</p></html>" > /var/www/html/index.html


  # Auto Scaling Group
  WebASG:
    Type: AWS::AutoScaling::AutoScalingGroup
    Properties:
      VPCZoneIdentifier:
        - subnet-0ba2ce3373b4a615c
        - subnet-0c60fabd025f28af3
      LaunchConfigurationName: !Ref WebLaunchConfig
```

```
    MinSize: 1
    MaxSize: 2
    TargetGroupARNs:
      - !Ref WebTargetGroup
    MetricsCollection:
      - Granularity: "1Minute"

# Política de Escalado - Aumentar instancias cuando el uso de CPU sea mayor al 50%
ScaleUpPolicy:
  Type: AWS::AutoScaling::ScalingPolicy
  Properties:
    AutoScalingGroupName: !Ref WebASG
    PolicyType: SimpleScaling
    AdjustmentType: ChangeInCapacity
    Cooldown: 300
    ScalingAdjustment: 1

# Alarma de CloudWatch para aumentar instancias
CPUAlarmHigh:
  Type: AWS::CloudWatch::Alarm
  Properties:
    AlarmDescription: "Escalar hacia arriba cuando el uso de CPU sea mayor al 50%"
    MetricName: CPUUtilization
    Namespace: AWS/EC2
    Statistic: Average
    Period: 60
    EvaluationPeriods: 2
    Threshold: 50
```

```
      ComparisonOperator: GreaterThanThreshold
      Dimensions:
        - Name: AutoScalingGroupName
          Value: !Ref WebASG
      AlarmActions:
        - !Ref ScaleUpPolicy

Outputs:
  LoadBalancerDNS:
    Description: URL del Load Balancer
    Value: !GetAtt LoadBalancer.DNSName
```

Actividad Opcional

Github Repository

https://github.com/josericardopenase/fleet_management_ddd

Fichero YAML

```
AWSTemplateFormatVersion: "2010-09-09"

Description: Despliegue de API con Docker y RDS en EC2 usando CloudFormation.


Parameters:

  KeyPairName:

    Description: "Nombre del par de claves para acceso SSH"

    Type: AWS::EC2::KeyPair::KeyName

  VpcId:

    Description: "ID de la VPC donde se crearán las instancias"

    Type: AWS::EC2::VPC::Id

  SubnetId1:

    Description: "ID de la Subred donde se creará la instancia EC2"

    Type: AWS::EC2::Subnet::Id

  SubnetId2:

    Description: "ID de la Subred 2 para el RDS"

    Type: AWS::EC2::Subnet::Id


Resources:

  # Security Group para el EC2 y la API

  ApiSecurityGroup:

    Type: AWS::EC2::SecurityGroup

    Properties:

      GroupDescription: "Permitir acceso a la API y SSH"

      VpcId: !Ref VpcId

      SecurityGroupIngress:
```

```
      - IpProtocol: "tcp"
        FromPort: "8080"
        ToPort: "8080"
        CidrIp: "0.0.0.0/0"
      - IpProtocol: "tcp"
        FromPort: "22"
        ToPort: "22"
        CidrIp: "0.0.0.0/0"

# Security Group para el RDS
RDSecurityGroup:
  Type: AWS::EC2::SecurityGroup
  Properties:
    GroupDescription: "Permitir acceso a MySQL desde EC2"
    VpcId: !Ref VpcId
    SecurityGroupIngress:
      - IpProtocol: "tcp"
        FromPort: "3306"
        ToPort: "3306"
        SourceSecurityGroupId: !GetAtt ApiSecurityGroup.GroupId

# Instancia RDS MySQL
MySQLDatabase:
  Type: AWS::RDS::DBInstance
  Properties:
    AllocatedStorage: 20
    DBInstanceClass: db.t3.micro
    Engine: mysql
```

```
    EngineVersion: "8.0"
    MasterUsername: appuser
    MasterUserPassword: apppassword
    DBName: DrivingSchoolDB
    VPCSecurityGroups:
      - !Ref RDSecurityGroup
    DBSubnetGroupName: !Ref MySQLSubnetGroup
    PubliclyAccessible: true

# Subnet Group para el RDS
MySQLSubnetGroup:
  Type: AWS::RDS::DBSubnetGroup
  Properties:
    DBSubnetGroupDescription: "Subnets for MySQL RDS"
    SubnetIds:
      - !Ref SubnetId1
      - !Ref SubnetId2

# Instancia EC2 para la API
ApiServerInstance:
  Type: AWS::EC2::Instance
  DependsOn: MySQLDatabase
  Properties:
    InstanceType: t2.micro
    KeyName: !Ref KeyPairName
    ImageId: ami-0ebfd941bbafe70c6 # Amazon Linux 2
    SecurityGroupIds:
      - !Ref ApiSecurityGroup
```

```
SubnetId: !Ref SubnetId1
UserData:
  Fn::Base64: !Sub |
    #!/bin/bash
    # Actualizamos los paquetes
    sudo dnf update -y

    # Instalamos Docker y Git
    sudo dnf install docker -y
    sudo dnf install git -y
    sudo service docker start
    sudo usermod -a -G docker ec2-user

    # Clonamos el repositorio de GitHub
    sudo git clone https://github.com/EduardoOrtegaZerpa/FleetManager.git

    # Construimos la imagen Docker
    cd FleetManager
    sudo docker build -t api .

    # Ejecutamos el contenedor Docker de la API con las variables de entorno
    sudo docker run -d -p 8080:8080 \
      -e "DB_HOST=${MySQLDatabase.Endpoint.Address}" \
      -e "DB_NAME=DrivingSchoolDB" \
      -e "DB_USER=appuser" \
      -e "DB_PASS=apppassword" \
      -e "spring.application.name=Pepeducacion" \
      -e "server.port=8080" \
```

```
        -e
"spring.datasource.url=jdbc:mysql://${MySQLDatabase.Endpoint.Address}:3306/DrivingSchoolDB" \
        -e "spring.datasource.username=appuser" \
        -e "spring.datasource.password=apppassword" \
        -e "spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver" \
        -e "spring.jpa.hibernate.ddl-auto=update" \
        -e "spring.jpa.show-sql=true" \
        -e "spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect" \
        api


Outputs:
  WebsiteURL:
    Description: "URL de la aplicación API"
    Value: !Sub "http://${ApiServerInstance.PublicIp}:8080"
```