UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DSIC
DEPARTAMENT DE SISTEMES INFORMÀTICS I COMPUTACIÓ

Departamento de Sistemas Informáticos y Computación
Universitat Politècnica de València

Sistema Serverless de Monitorización de Recursos para Plataformas de Cloud Público

TRABAJO FIN DE MÁSTER

Máster Universitario en Computación Paralela y Distribuida

Autor: Cristina Requena Casares

Tutor: Germán Moltó Martínez

Curso 2019-2020

Resumen

El presente trabajo fin de máster se centra en el desarrollo de un nuevo panel web accesible desde cualquier dispositivo que muestra los recursos que están en uso en una cuenta de Amazon Web Services, por parte de sus múltiples usuarios.

Con el fin de ahorrar costes derivados de adquirir y mantener los servidores, las empresas han empezado a desarrollar sus soluciones o mover sus centros de datos a servicios en la nube, es decir, al cloud computing. Esto es debido a las múltiples ventajas que aporta, como la falta de una gran inversión inicial, el pago por uso de los servicios, la fácil elasticidad y la mayor seguridad, entre otros.

Existen diferentes empresas que ofrecen estos servicios en la nube como Amazon, Microsoft, Alibaba o Google. Por tanto, las empresas que deseen alojar sus datos o servicios web deben elegir el proveedor de servicios en la nube más conveniente, basándose en la disponibilidad, la escalabilidad o el coste económico. No obstante, dado que el coste de estos servicios depende del uso, es necesario controlarlo para añadir o quitar recursos y así, evitar costes innecesarios.

En este proyecto se pretende mostrar la forma de crear un panel web utilizando el framework Vue.js, que internamente realiza peticiones a una API alojada en Amazon Web Services para ejecutar una función Lambda desplegada mediante el framework Serverless. El panel desarrollado permite al usuario consultar en tiempo real los principales recursos aprovisionados en una cuenta de AWS, así como su coste.

Palabras clave: panel web, cloud computing, Vue.js, API, Amazon AWS

Resum

El present treball fi de màster es centra en el desenvolupament d'un nou panel web accessible des de qualsevol dispositiu que mostra els recursos que estan en us en un compte de Amazon Web Services, per part dels seus múltiples usuaris.

Per tal d'estalviar costos derivats d'adquirir i mantenir els servidors, les empreses han començat a desenvolupar les seues solucions o moure els seus centres de dades a serveis al núvol, és a dir, al cloud computing. Això és degut als múltiples avantatges que aporta, com la falta d'una gran inversió inicial, el pagament per ús dels serveis, la fàcil elasticitat i la major seguretat, entre d'altres.

Existeixen varietat d'empreses que ofereixen aquests serveis al núvol com Amazon, Microsoft o Google. Per tant, les empreses que desitgen allotjar les seues dades o serveis web han de triar el proveïdor de serveis al núvol més convenient, basant-se en la disponibilitat, la escalabilitat o el cost econòmic. No obstant això, atès que el cost d'aquest serveis depèn de l'ús, cal controlar-lo per afegir o suprimir recursos i així, evitar costos innecessaris.

En aquest projecte es pretén mostrar la manera de crear un panel web utilitzant el framework Vue.js, que internament realitza peticions a una API allotjada a Amazon Web Services per a executar una funció Lambda desplegada mitjançant el framework Serverless. El panel desenvolupat permet a l'usuari consultar en temps real els principals recursos aprovisionats en un compte de AWS, així com el seu cost.

Paraules clau: panel web, cloud computing, Vue.js, API, Amazon AWS

Abstract

This master's thesis focuses on development of a new accessible web panel from any device that shows the resources in use in an Amazon Web Services account by multiple users.

In order to save costs derived from acquiring and maintaining servers, companies have started to develop their solutions or move their data centers to cloud services, that is to say, to cloud computing.

There are different companies that offer these cloud services such as Amazon, Microsoft or Google. Therefore, companies want to host their data or web services should choose the most convenient cloud service provider, based on availability, scalability or cost. However, since the cost of these services depends on the use, it's necessary to control it to add or remove resources and thus avoid unnecessary costs.

This project aims to show how to create a web panel using the Vue.js framework, which internally makes requests to an API hosted by Amazon Web Services to execute a Lambda function deployed using the Serverless framework. The developed panel allows the user to consult in real time the main resources provisioned in an AWS account, as well as their cost.

Key words: web panel, cloud computing, Vue.js, API, Amazon AWS

Índice general

Índice general VII
Índice de figuras IX
Índice de tablas IX

Agradecimientos XI
1 Introducción 1
1.1 Motivación . . . 1
1.2 Aplicaciones Serverless vs on-premises . . . 3
1.3 Objetivos . . . 4
1.4 Estructura del documento . . . 5
2 Preliminares: Tecnologías utilizadas 7
2.1 Amazon Web Services (AWS) . . . 7
2.1.1 Amazon Elastic Compute Cloud (EC2) . . . 8
2.1.2 Amazon Simple Storage Service (S3) . . . 9
2.1.3 Amazon Relational Database Service (RDS) . . . 9
2.1.4 Amazon Elastic Load Balancing (ELB) . . . 10
2.1.5 Amazon Cognito . . . 11
2.1.6 AWS Lambda . . . 12
2.1.7 AWS Pricing . . . 13
2.2 Python . . . 14
2.2.1 Boto3 . . . 14
2.2.1.1 Cliente ec2 . . . 14
2.2.1.2 Cliente s3 . . . 15
2.2.1.3 Cliente rds . . . 16
2.2.1.4 Clientes elb y elbv2 . . . 16
2.2.1.5 Cliente lambda . . . 17
2.2.1.6 Cliente autoscaling . . . 18
2.3 Javascript . . . 18
2.3.1 Vue.js . . . 19
2.4 Framework Serverless . . . 20
3 CloudTrail-Tracker 23
3.1 CloudTrail-Tracker-UI . . . 24
4 Oteador, monitorización serverless de recursos en AWS 27
4.1 Descripción del sistema . . . 27
4.2 Arquitectura del sistema . . . 27
4.3 Creación de la API de Oteador . . . 29
4.4 Integración con CloudTrail-Tracker-UI . . . 31
4.5 Funcionalidades . . . 33
4.5.1 Información sobre recursos y costes . . . 34

4.5.2 Listado de los buckets de Amazon S3 38
4.5.3 Listado de las instancias de Amazon EC2 41
4.5.4 Listado de las instancias de Amazon RDS 44
4.5.5 Listado de balanceadores de carga 46
4.5.6 Listado de grupos de auto-escalado creados 48
4.5.7 Listado de direcciones IP elásticas no conectadas a instancias 50
4.5.8 Listado de funciones Lambda 52

5 Conclusiones y Trabajos Futuros **55**

Glosario **57**

Bibliografía **59**

Apéndices

Anexo I: Definición de la API **63**

Índice de figuras

2.1 ¿Que es Amazon Cognito? [40] . . . 12
2.2 Proveedores de servicios serverless [11] . . . 20

3.1 Arquitectura del sistema CloudTrail-Tracker[46] . . . 24
3.2 Dashboard con el framework CloudTrail-Tracker-UI [47] . . . 25

4.1 Arquitectura del sistema . . . 28
4.2 Botón Oteador . . . 32
4.3 Panel Oteador . . . 33
4.4 Widgets Oteador . . . 34
4.5 Panel Oteador con información Amazon S3 . . . 40
4.6 Ciclo de vida de una instancia EC2 [48] . . . 41
4.7 Panel Oteador con información Amazon EC2 . . . 43
4.8 Panel Oteador con información Amazon RDS . . . 45
4.9 Panel Oteador con información Amazon Elastic Load Balancing . . . 47
4.10 Panel Oteador con información Amazon Auto Scaling Groups . . . 49
4.11 Panel Oteador con información Amazon Elastic IP . . . 51
4.12 Panel Oteador con información Amazon Lambda . . . 53

Índice de tablas

2.1 Clasificación de los servicios . . . 8

CAPÍTULO 1
Introducción

A lo largo de este capítulo veremos, en primer lugar, la motivación que justifica la necesidad de desarrollar una herramienta para monitorizar en tiempo real el uso de recursos de una cuenta de AWS. En segundo lugar, veremos un breve análisis comparativo entre aplicaciones serverless y aplicaciones on-premises. En tercer lugar, expondremos los objetivos de nuestro trabajo y finalmente, la estructura de los capítulos de este.

1.1 Motivación

La sociedad de hoy en día no tiene nada que ver con la sociedad del pasado, donde la presencia de los dispositivos electrónicos en la vida diaria era casi inexistente.

La mayoría de las empresas sólo tenían tienda física, y las empresas que también tenían su tienda online, no daban la opción de acceder a todo su catálogo.

Con el avance de la tecnología, las personas han ido adquiriendo mayor cantidad de estos dispositivos electrónicos, lo cual ha hecho que el sector crezca de forma exponencial en los últimos años. Esto a su vez, ha obligado a diversas empresas a adaptarse a los acontecimientos ofreciendo formas de llegar a las personas mediante los dispositivos electrónicos.

Algunas empresas optan por desarrollar sus aplicaciones web adquiriendo todos los recursos necesarios, mientras que otras optan por desarrollar sus aplicaciones alojándolas en plataformas en la nube, pagando por el uso que se haga de ellas.

Las plataformas en la nube nos permiten reducir costes de personal especializado en instalaciones o mantenimiento de los dispositivos usados para almacenar nuestro software, datos, etc. También permite convertir los costes fijos de contratar ciertos servicios, por variables dado que se pueden escalar de un día para otro según las necesidades. Por ello, muchas de ellas deciden introducirse en el cloud computing, es decir, los servicios en la nube.

Existen diversos proveedores de cloud computing:

- **Google Cloud Platform** (*Google*) [2]:

  Es una plataforma que ofrece productos, servicios y herramientas para diseñar, probar y lanzar aplicaciones con una gran escalabilidad y seguridad debido a la infraestructura proporcionada por Google, la cual se divide en regiones y zonas.

  Entre sus productos y servicios podemos destacar *App Engine*, un servicio para crear y poner en marcha aplicaciones; *Compute Engine*, una plataforma para instanciar máquinas en el cloud; *Cloud Virtual Network*, para crear maquinas virtuales; o *Cloud Storage*, un sistema de almacenamiento de objetos, que permite guardar datos no estructurados y ficheros de gran tamaño.

- **Microsoft Azure** (*Microsoft*) [4]

  Es una plataforma de servicios en la nube de Microsoft, que sirve para construir, probar, desplegar y administrar aplicaciones proporcionando gran escalabilidad y una alta disponibilidad. Asimismo, Azure cuenta con 160 centros de datos físicos que, a su vez, se encuentran repartidos en regiones.

  Entre sus productos podemos destacar: *Virtual Machines*, un servicio que se encarga de aprovisionar las máquinas virtuales; *App Service*, un servicio para crear aplicaciones o *Azure Functions*, un servicio que gestionar los eventos de proceso con código sin servidor.

- **Alibaba Cloud** (*Grupo Alibaba*) [5]

  Es la plataforma de servicios en la nube del Grupo Alibaba, que ofrece soluciones para crear aplicaciones en la nube dotando a estas de gran escalabilidad.

  Entre sus productos y servicios podemos destacar: *Elastic Compute Service*, para crear servidores virtuales o *Function Compute*, un servicio que permite ejecutar el código en un entorno serverless.

- **Amazon Web Services** (*Amazon*) [6]:

  Es la plataforma de servicios en la nube de Amazon, que proporciona un conjunto de herramientas para crear, probar y desplegar aplicaciones en la nube garantizando una gran escalabilidad, disponibilidad y seguridad.

  Entre sus productos y servicios podemos destacar: *Amazon EC2* [23], un servicio que permite crear, mantener o escalar instancias de cómputo; *Amazon RDS* [17], un servicio que ofrece distintos tipos de bases de datos; o *Amazon S3* [24], un servicio de almacenamiento y gestión de contenido potente y asequible.

Entre las empresas que han optado por desarrollar su aplicación haciendo uso de los servicios en la nube y transformando su modelo de negocio tenemos la empresa Atresmedia[7], la cual lleva años ofreciendo una gran variedad de contenido multimedia por televisión. Sin embargo, hace años crearon la aplicación Atresplayer, un servicio de transmisión por Internet y televisión disponible en una gran variedad de dispositivos como teléfonos móviles, tablets, ordenadores y Smart tv gracias a su desarrollo con Amazon Web Services.

Atresmedia necesitaba incrementar su audiencia digital y crear su plataforma Atresplayer en servicios en la nube, de forma que conseguirían gestionar mejor los picos de tráfico de los canales de TV.

Otro ejemplo es la empresa Netflix, la cual cambió totalmente su modelo de negocio de ofrecer un servicio de alquiler de DVDs a través de correo postal a ofrecer un catálogo amplio de series, películas o programas mediante su aplicación, la cual también está creada mediante Amazon Web Services.

En este caso, tras la popularización de sus servicios, Netflix se vio obligada a migrar sus centros de datos tradicionales a un servicio en la nube para poder hacer frente a los grandes picos de procesamiento de datos debido a la alta demanda; y además, les ha permitido expandirse en función de esa demanda dependiendo de su localización, ahorrando costes.

Pero todas estas aplicaciones que hacen uso de recursos que se pagan por uso, es necesario monitorizar su uso y así tomar las medidas pertinentes, con el fin de garantizar la rentabilidad de estas en cada negocio.

1.2 Aplicaciones Serverless vs on-premises

Las aplicaciones pueden ser ejecutadas en servidores propios, también llamadas aplicaciones on-premises. O bien, pueden ejecutarse de forma remota, de forma que el peso de la ejecución recaiga en un servidor externo. Dependiendo de cual sea la necesidad, se debe optar por un modelo de servicio u otro de informática en la nube: infraestructura como servicio (Iaas), plataforma como servicio (PaaS), software como servicio (SaaS), funciones como servicio (FaaS) y backend como servicio (BaaS).

A continuación, describiremos los modelos de servicio según la definición de NIST [12]:

- IaaS: La infraestructura como servicio es la capacidad de proporcionar al usuario el aprovisionamiento de unidades de procesamiento, almacenamiento y redes, para que este usuario pueda implementar y ejecutar software en dichos sistemas. Por ejemplo, el servicio Amazon EC2 de Amazon AWS.
- PaaS: La plataforma como servicio es la capacidad de proporcionar al usuario la opción de implementar aplicaciones en la infraestructura en la nube. Para ello, es necesario usar lenguajes de programación o herramientas compatibles con dicho proveedor de servicios en la nube. Por ejemplo, el servicio AWS Elastic Beanstalk [13] de Amazon AWS.
- SaaS: El software como servicio es la capacidad de proporcionar al usuario la opción de utilizar las aplicaciones del proveedor de servicios en la nube que se ejecutan en su propia infraestructura. Estas aplicaciones son accesibles a través de un navegador web o una interfaz de programa. Por ejemplo, las servicios Gmail [14] de Google o Microsoft Dynamics 365 [15] de Microsoft.
- FaaS: La función como servicio es la capacidad de proporcionar al usuario la posibilidad de desarrollar, ejecutar y administrar sus aplicaciones, en

forma de funciones; sin encargarse del aprovisionamiento o administración de los servidores o escalar la infraestructura. En este caso, el usuario solo paga cuando el programa se esta ejecutando. Por ejemplo, el servicio AWS Lambda [27] de Amazon AWS o Azure Functions de Microsoft.

- BaaS: El backend como servicio es la capacidad que permite integrar la aplicación con backends disponibles en la nube, de forma que se pueden consumir directamente a través de su API.

Las aplicaciones que se ejecutan sobre el modelo BaaS se llaman aplicaciones Serverless. Las aplicaciones Serverless son una nueva forma de desplegar aplicaciones en el mundo de los servicios de computación en la nube pagando por uso. Estas utilizan servicios de terceros alojados en la nube, siendo allí donde se ejecuta el programa. En cambio, las aplicaciones on-premises son las que se utilizaban tradicionalmente donde la lógica del programa se ejecutaba en un servidor propio.

Entre las ventajas de desarrollar aplicaciones on-premises tenemos que ante datos sensibles disponemos de mayor seguridad y además, solo necesitamos una red privada para funcionar. Sin embargo, las aplicaciones serverless aportan otras ventajas como son una mayor capacidad de escalabilidad y más facilidad de disponibilidad.

No obstante, las aplicaciones serverless ejecutadas en servicios de cloud computing, cuentan con la complejidad que su pago se realiza en función de la configuración contratada o el uso que se haya hecho de la aplicación alojada. Por ello, es necesario monitorizar los recursos utilizados en la nube.

Por tanto, en nuestro caso elegiremos la arquitectura Serverless dado que nos aporta las ventajas mencionadas anteriormente y también, nos abstrae de los inconvenientes del mantenimiento y adquisición del hardware. La aplicación Oteador se integra con la actual aplicación CloudTrail-Tracker [46] , una plataforma de código abierto para obtener un análisis de uso mejorado de una cuenta compartida de AWS. La herramienta de CloudTrail-Tracker esta orientada con fines educativos, de forma que la herramienta proporciona al profesor un panel visual, que muestra el uso agregado de los recursos por parte de los estudiantes, durante un período de tiempo determinado. Para más información acerca de la herramienta se puede consultar su documentación en su página web[1].

Sin embargo, con dicha herramienta no es posible obtener los recursos que están en uso en tiempo real, por lo que centraremos el desarrollo de Oteador en conseguir este requisito.

1.3 Objetivos

El objetivo de nuestro trabajo final de máster es crear un nuevo panel web llamado Oteador, dentro de la herramienta CloudTrail-Tracker, para la monitorización en tiempo real de los principales recursos aprovisionados por los usuarios

[1] http://www.grycap.upv.es/cloudtrail-tracker

dentro de una cuenta de usuario AWS, obteniendo también una estimación de su coste horario.

La elección de este tipo de aplicación viene dada por los siguientes motivos:

- **Abstracción en el acceso a la información.**

  Con esto se consigue que la interfaz de la aplicación se encargue únicamente de mostrar los datos que obtiene la aplicación serverless.

- **Integración con un herramienta existente.**

  Al integrar nuestro panel con la aplicación CloudTrail-Tracker, podemos obtener tanto los recursos que están en uso en ese instante, como los recursos que han estado en uso en un determinado tiempo.

- **Obtención del estado de diferentes recursos.**

  Con nuestra herramienta podemos obtener información sobre el uso que se da de diferentes recursos de Amazon AWS en forma de gráfica de barras y una tabla.

- **Obtención del coste horario.**

  También, se muestra el precio por hora de los recursos que su uso continuado con la configuración elegida, más coste conllevan.

- **Accesible mediante una dirección URL.**

  La aplicación es accesible mediante una dirección URL para ser consultada mediante el panel integrado en la aplicación CloudTrail-Tracker.

1.4 Estructura del documento

Con el fin de facilitar la lectura y el seguimiento de este trabajo final de master, a continuación mostramos los capítulos que contiene nuestro trabajo y una breve descripción del contenido de cada uno.

- **Capítulo 1. Introducción.**

  Una introducción sobre los motivos para realizar nuestra trabajo, una comparación sobre la diferencia entre aplicaciones on-premise y aplicaciones serverless y los objetivos marcados para conseguir nuestro fin.

- **Capítulo 2. Preliminares: Tecnologías utilizadas.**

  Una explicación de las tecnologías utilizadas involucradas en el desarrollo o uso de nuestro proyecto.

- **Capítulo 3. CloudTrail-Tracker**

  En este capítulo se describe la herramienta CloudTrail-Tracker con la que se integra Oteador, la cual permite obtener datos sobre el uso de los recursos incluyendo el portal web CloudTrail-Tracker-UI, que permite visualizar los datos que obtiene la API de CloudTrail-Tracker y Oteador.

- **Capítulo 4. Oteador, monitorización serverless de recursos en AWS**

  La definición de la arquitectura de la aplicación y la explicación de los pasos seguidos para lograr la implementación y desarrollo de la aplicación.

- **Capítulo 5. Conclusiones y Trabajos Futuros**

  Identificación de las posibles líneas de desarrollo para continuar evolucionando la aplicación, alcance de los objetivos y conclusiones.

- **Capítulo 6. Bibliografía y glosario.**

  Un listado con todo el material consultado en artículos o web de interés para la realización del proyecto.

CAPÍTULO 2
Preliminares: Tecnologías utilizadas

En esta sección comentaremos brevemente las diferentes tecnologías que hemos utilizado para desarrollar nuestra aplicación Oteador y la manera en la que se han integrado.

2.1 Amazon Web Services (AWS)

Amazon Web Services (AWS) es una colección de servicios web que en conjunto forman una plataforma de computación en la nube completa y en constante evolución proporcionada por Amazon. Ofrece una combinación de infraestructura como servicio (IaaS), plataforma como servicio (PaaS), software como servicio (SaaS) y funciones como servicio (FaaS).

AWS fue una de las primeras empresas en introducir un modelo de computación en la nube de pago por uso el cual se puede ampliar para proporcionar a los usuarios el cálculo, el almacenamiento o el rendimiento según sea necesario.

Los servicios de AWS se reparten en docenas de centros de datos distribuidos por zonas de disponibilidad (AZ) en regiones de todo el mundo. Una AZ representa una ubicación que normalmente contiene múltiples centros de datos físicos, mientras que una región es una colección de AZ próximas entre sí y conectadas por enlaces de red de baja latencia. Por ello, un cliente de AWS puede activar máquinas virtuales (VM) y replicar datos en diferentes AZ para lograr una infraestructura altamente confiable que resista las caídas de servidores individuales o de un centro de datos completo.

En nuestro caso utilizaremos algunos servicios para implementar la aplicación de Oteador, en otros los usaremos para obtener la información relevante y finalmente, también en algunos otros se usarán tanto para implementar la aplicación como para obtener la información de interés:

Amazon Web Services			
	Implementados en Oteador	Obtenidos en Oteador	Ambos
Amazon EC2		•	
Amazon RDS		•	
Amazon Elastic Load Balancing		•	
AWS Auto Scaling		•	
Amazon S3			•
Amazon Lambda			•
AWS Pricing	•		

Tabla 2.1: Clasificación de los servicios

Como podemos ver en la Tabla 2.1, el servicio de AWS Pricing se utiliza para desarrollar la aplicación de Oteador; los servicios de Amazon EC2, Amazon RDS, Amazon Elastic Load Balancing y AWS Auto Scaling se utilizan para obtener la información relevante pero no se implementan en la aplicación; y finalmente, los servicios de Amazon S3 y Amazon Lambda se utilizan con ambas finalidades. A continuación, se proporciona una breve descripción de cada servicio.

2.1.1. Amazon Elastic Compute Cloud (EC2)

Amazon EC2 es un servicio web que permite a los usuarios alquilar máquinas virtuales (VM) de tamaño modificable para ejecutar sus propias aplicaciones. Por ello, esta diseñado para simplificar la escalabilidad web en la nube a los desarrolladores. Además, tiene una interfaz de servicios web que permite obtener y configurar las máquinas virtuales.

En nuestro caso, en la aplicación Oteador podemos obtener las instancias de máquinas según su estado:

- Pendiente (pending).
- En uso (running).
- Parándose (stopping).
- Paradas (stopped).
- Terminándose (shutting-down)
- Terminadas (terminated).

Además, también podemos obtener el usuario que desplegó la instancia. De esta forma, es posible monitorizar las máquinas en uso.

2.1.2. Amazon Simple Storage Service (S3)

Amazon S3 es un servicio de almacenamiento en la nube que permite a los usuarios almacenar objetos ofreciendo escalabilidad, disponibilidad de datos, seguridad y rendimiento.

Este servicio almacena los datos en contenedores de objetos, llamados buckets. Para cada bucket se puede controlar el acceso y los permisos para crear, eliminar o listar los objetos de este.

Amazon S3 tiene una interfaz de servicios web que permite almacenar y recuperar los datos desde cualquier parte de la web y en cualquier momento.

Asimismo, si comparamos Amazon S3 frente a sus competidores en el mercado, vemos que el alojamiento de aplicaciones web en Amazon S3 es la opción más rentable, dado que para el tipo de almacenamiento estándar (S3 Estándar) el precio de 50 TB/mes es de 0,023 USD por GB; en cambio, uno de sus mayores competidores, Google Cloud Storage [25] ofrece el mismo almacenamiento por 0,026 USD por GB. Además, es la opción más segura dado que no hay problemas de intrusiones y la opción más escalable dado que permite múltiples accesos concurrentes.

En nuestro caso, en la aplicación Oteador el servicio de Amazon S3 se utiliza de diversas formas:

- La versiones de la aplicación serverless se almacenan en un bucket de Amazon S3.
- Por otro lado, también obtenemos los buckets de Amazon S3 que están creados.

2.1.3. Amazon Relational Database Service (RDS)

Amazon RDS es un servicio de base de datos relacional distribuido de AWS. Está diseñado para simplificar la configuración, el funcionamiento y el escalamiento de una base de datos relacional para su posterior uso en aplicaciones en la nube.

El servicio proporciona la capacidad para que sea rentable y escalable al mismo tiempo que automatiza todas las tareas administrativas, como el abastecimiento de hardware, la configuración de las bases de datos, la implementación de parches y la creación de copias de seguridad.

Amazon RDS dispone de varios tipos de instancias de base de datos: optimizadas para memoria, rendimiento u operaciones de escritura y lectura. También proporciona seis motores de bases de datos conocidos como PostgreSQL [19], MySQL [20], Oracle Database [21], SQL Server [22], entre otros.

Además, se puede migrar o replicar las bases de datos existentes a Amazon RDS mediante el servicio AWS Database Migration Service [18].

En nuestro caso, en la aplicación Oteador podemos obtener las instancias de base de datos según su estado:

- En funcionamiento (available).

- Creando copia de seguridad (backing-up).
- Habilitando/des-habilitando la monitorización mejorada (configuring-enhanced-monitoring).
- Habilitando/des-habilitando la autentificación (configuring-iam-database-auth).
- Habilitando/des-habilitando la publicación de archivos de Amazon CloudWatch Logs (configuring-log-exports).
- Convirtiendo a instancia que no está en una Amazon VPC (converting-to-vpc).
- Creándose (creating).
- Eliminandose (deleting).
- Error (failed).
- En mantenimiento (maintenance).
- Modificándose (modifying).
- Reiniciándose (rebooting).
- Iniciándose (starting).
- Detenida (stopped).

Además, también podemos obtener los datos del usuario que desplegó la instancia. De esta forma, nos es posible monitorizar las bases de datos en uso y posteriormente, evaluar su coste horario.

2.1.4. Amazon Elastic Load Balancing (ELB)

Amazon Elastic Load Balancing [26] es el balanceador de carga de Amazon Web Services (AWS), el cual distribuye de forma automática el tráfico de las aplicaciones entrantes dependiendo de su destino: instancias de Amazon EC2, contenedores, direcciones IP y funciones Lambda.

Además, ofrece tres tipos de balanceadores de carga aportando alta disponibilidad, escalabilidad automática y seguridad para garantizar que sus aplicaciones son tolerantes a errores. Las tres opciones que nos ofrece son:

- Balanceador de carga de aplicaciones. El balanceador de carga de aplicaciones permite equilibrar la carga del trafico HTTP y HTTPS; y proporciona un direccionamiento de solicitudes avanzado dirigido a arquitecturas de aplicaciones modernas incluyendo microservicios y contenedores.
- Balanceador de carga de red. El balanceador de carga de red permite equilibrar la carga del tráfico del protocolo de control de transmisión (TCP), del protocolo de datagramas de usuario (UDP) y de la seguridad de la capa de transporte (TLS).

- Balanceador de carga clásico. El balanceador de carga clásico permite equilibrar la carga básica de varias instancias de Amazon EC2 funcionando tanto a nivel de solicitud como a nivel de conexión. Este tipo de balanceador está diseñado para aplicaciones que se crearon dentro de la red EC2-Classic.

En nuestro caso, en la aplicación Oteador podemos obtener los balanceadores de carga que se han creado y clasificarlos según su tipo: aplicación, de red o clásico.

2.1.5. Amazon Cognito

Amazon Cognito [40] es un servicio de control de cuentas de usuario de AWS que se ejecuta en la nube y proporciona autenticación, autorización y administración de usuarios a las aplicaciones móviles y web.

El servicio se encarga de guardar y sincronizar los datos del usuario, con lo que permite acelerar el proceso de desarrollo de aplicaciones.

Los dos componentes principales de este servicio son los grupos de usuarios y los grupos de identidades. A continuación, explicaremos con más detalle en qué consisten cada uno de ellos:

- Los grupos de usuarios son directorios de usuarios que permiten a los usuarios de las aplicaciones finales poder registrarse e iniciar sesión en estas mediante Amazon Cognito. Aunque, también pueden federarse mediante un proveedor de identidad (IdP) de terceros.

  Por tanto, los grupos de usuarios proporcionan:

  - El servicio de registrarse e iniciar sesión.
  - Una interfaz web personalizable e integrada para que los usuarios finales puedan iniciar sesión.
  - El inicio de sesión a través de redes sociales con Facebook, Google, Login con Amazon e Inicio de sesión con Apple, o por medio de proveedores de identidad SAML y OIDC desde su grupo de usuarios.
  - La administración de directorios de usuarios y sus respectivos perfiles de usuario.
  - La autentificación multifactor (MFA), lo que permite garantizar una cierta seguridad mediante comprobaciones de credenciales filtradas, protección de posesión de la cuenta y verificación del correo electrónico y teléfono.
  - El flujo de trabajo personalizado y migración de los usuarios a través de disparadores de AWS Lambda.

- Los grupos de identidades permiten que los usuarios puedan acceder a otros servicios de AWS mediante la obtención de credenciales temporales de AWS.

Estos grupos admiten usuarios invitados anónimos y proveedores de identidad para autenticar a los usuarios. Entre los proveedores de identidad tenemos:

- Proveedores de OpenID Connect (OIDC).
- Proveedores de identidad SAML.
- Identidades autenticadas por el desarrollador.

En nuestro caso, en la aplicación Oteador necesitamos autenticar el usuario y, a continuación, permitirle el acceso a otro servicio de AWS.

1. En primer lugar, el usuario inicia sesión a través del User Pool de Amazon Cognito [41], es decir, un grupo de usuarios que se autentifica mediante un proveedor de identidad y recibe los tokens de grupos de usuarios.
2. Luego la aplicación intercambia estos tokens del grupo de usuarios por las credenciales de AWS a través del grupo de identidades.
3. Finalmente, el usuario de la aplicación puede utilizar las credenciales de AWS para acceder a otros servicios de AWS.

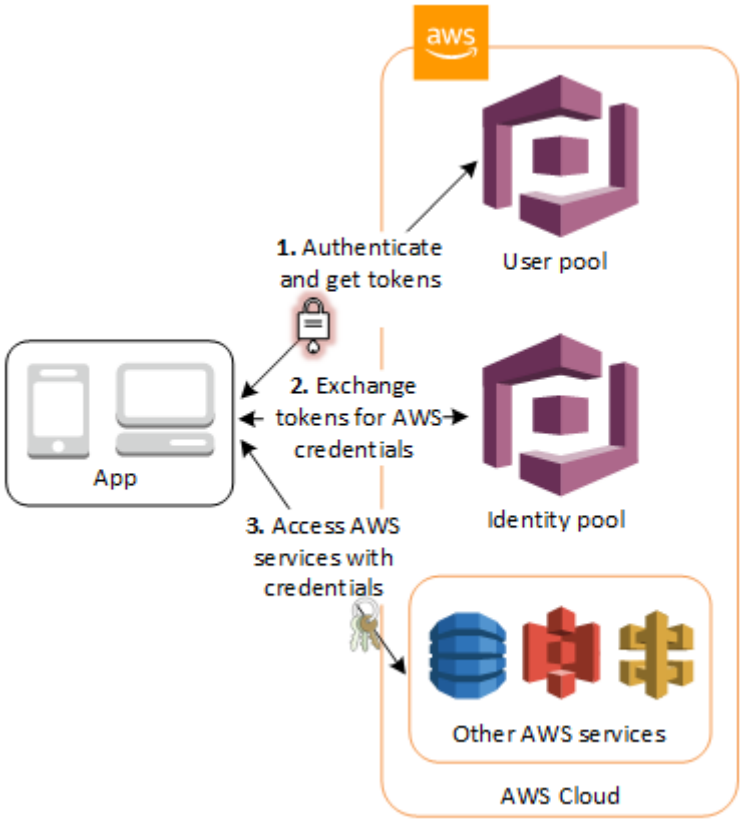

Figura 2.1: ¿Que es Amazon Cognito? [40]

2.1.6. AWS Lambda

AWS Lambda [27] es un servicio que permite ejecutar código automáticamente sin servidor dependiendo de varios eventos, como solicitudes HTTP a través de Amazon API Gateway [30], modificaciones realizadas en los objetos de los buckets de Amazon S3 y actualizaciones en tablas en Amazon DynamoDB.

Además, permite administrar automáticamente los recursos subyacentes, ampliar la funcionalidad de otros servicios de AWS o bien crear servicios back-end propios garantizando un cierto nivel de seguridad, rendimiento y escalabilidad de AWS.

AWS Lambda ejecuta el programa en una infraestructura de alta disponibilidad y se encarga de la administración de los recursos, incluido el mantenimiento del servidor y del sistema operativo, el aprovisionamiento de capacidad y el escalado automático, la implementación de parches de seguridad y código, así como la monitorización de código y los registros.

De modo que, al ejecutar una función, se crea una instancia de dicha función y si antes de que se devuelva una respuesta se recibe otra petición a la función, se crea una instancia adicional. Cuando el numero de solicitudes disminuye, se detienen las instancias que no se utilizan para liberar la capacidad de escalado para otras funciones.

Asimismo, en AWS Lambda se paga por las funciones que se ejecutan, por tanto la facturación se realiza en base a la cantidad de solicitudes y el tiempo necesario para ejecutar estas funciones. La duración se calcula a partir del instante en el cual el código empieza a ejecutarse hasta que finaliza, y esta se redondea a los 100 ms más cercanos.

En nuestro caso, en la aplicación Oteador el servicio de AWS Lambda se utiliza de diversas formas:

- Creamos una función Lambda que obtenga algunos recursos de AWS para una región concreta.
- Por otro lado, también obtenemos las funciones de AWS Lambda que están desplegadas.

2.1.7. AWS Pricing

AWS Pricing [29] es un servicio de AWS para consultar los precios de los distintos recursos de AWS como Amazon EC2, Amazon RDS, etc.

Dado que en AWS se paga por los recursos en uso, es de vital importancia conocer que instancias tenemos en funcionamiento y su coste.

En la aplicación Oteador, el servicio de AWS Pricing se utiliza para obtener el precio por hora de los servicios:

- Amazon EC2 (estado: running)
- Amazon RDS (estado: available)
- Amazon Elastic Load Balancing
- Amazon Lambda

El motivo por el cual se han escogido estos servicios para monitorizar su coste horario, es debido a que son los más utilizados para desarrollar aplicaciones en la nube y dependiendo del estado en el que se encuentren pueden tener un alto coste.

2.2 Python

Python [36] es un lenguaje de programación interpretado, orientado a objetos, de alto nivel, de tipado dinámico, es decir, las variables no requieren ser definidas asignando su tipo de datos. Las estructuras de datos integrados de alto nivel combinadas con el tipado dinámico y el enlace dinámico, lo hacen versátil para el desarrollo de aplicaciones, scripts o conectar otros componentes. Entre sus características podemos destacar:

- La sintaxis simple y fácil de aprender que enfatiza la legibilidad y reduce el coste del mantenimiento del programa.
- La posibilidad de añadir módulos y paquetes que ayuda a modularizar el programa y reutilizar código.

En nuestro caso, en la aplicación Oteador utilizaremos el lenguaje de programación Python para desarrollar la aplicación serverless que obtendrá los recursos de AWS y que se ejecutará en una función Lambda. No obstante, para poder acceder a la información necesitamos la librería Boto3 de Python.

2.2.1. Boto3

Boto [38] es el kit de desarrollo de software (SDK) de Amazon AWS para Python. Este permite a los desarrolladores de Python crear, configurar y administrar servicios de AWS (EC2, S3, RDS, etc.).

Además, proporciona una API orientada a objetos fácil de usar y acceso a bajo nivel a los servicios de AWS. De esta forma, podemos obtener la información de interés en tiempo real.

Las funciones de su API las podemos usar dependiendo del cliente que inicialicemos. En nuestro caso, sólo inicializaremos estos clientes de los múltiples que tiene:

- ec2
- s3
- rds
- elb / elbv2
- lambda
- autoscaling

2.2.1.1. Cliente ec2

Con el cliente 'ec2' podemos realizar acciones sobre las distintas instancias de las máquinas de Amazon EC2: obtener información o realizar modificaciones sobre las instancias.

Este es un ejemplo sencillo de carga de cliente y listar las instancias de EC2:

```
import boto3

ec2 = boto3.client('ec2')

response = ec2.describe_instances()

print(response)
```

Listing 2.1: Boto3: Listar instancias de EC2

Tras ejecutar la función *describe_instances* podemos obtener información relevante a la instancia como:

- *AmiLaunchIndex*: Es el índice del lanzamiento de la AMI.
- *ImageId*: El ID de la AMI usada para lanzar la instancia.
- *InstanceId*: El ID de la instancia.
- *InstanceType*: El tipo de instancia.
- *KernelId*: El kernel asociado con la instancia.
- *LaunchTime*: El tiempo que la instancia está lanzada.
- *Placement*: La localización de la instancia desplegada.
- *PrivateIpAddress*: La dirección IP privada asignada a la instancia.
- *PublicIpAddress*: La dirección IP pública asignada a la instancia.
- *State*: El estado en el que está la instancia.
- *IamInstanceProfile*: El perfil de IAM asociado a la instancia.

En nuestro caso, en la aplicación Oteador necesitamos monitorizar las instancias de Amazon EC2, por lo que no realizamos modificaciones, sólo obtendremos la información de las instancias.

2.2.1.2. Cliente s3

Con el cliente 's3' también podemos realizar acciones sobre los buckets de Amazon S3 para obtener información o realizar modificaciones sobre los buckets.

En nuestro caso, en la aplicación Oteador necesitamos monitorizar los buckets de Amazon S3 creados, por lo que no realizamos modificaciones.

Este es un ejemplo sencillo de carga del recurso y listar los buckets de S3:

```
import boto3

s3 = boto3.client('s3')

response = s3.list_buckets()

print(response)
```

Listing 2.2: Boto3: Listar buckets de S3

2.2.1.3. Cliente rds

Con el cliente 'rds' podemos realizar acciones sobre las distintas instancias de bases de datos relacionales en Amazon RDS: obtener información o realizar modificaciones sobre las instancias.

Este es un ejemplo sencillo de carga del cliente y listar las bases de datos de RDS:

```
import boto3

rds = boto3.client('rds')

response = rds.describe_db_instances()

print(response)
```

Listing 2.3: Boto3: Listar bases de datos de RDS

Tras ejecutar la función *describe_db_instances* podemos obtener información relevante a la instancia de base de datos como:

- *DBInstanceIdentifier*: Contiene el identificador de la instancia.
- *DBInstanceClass*: Contiene el nombre de la clase de capacidad de cálculo y memoria de la instancia.
- *Engine*: El motor de la instancia de base de datos.
- *DBInstanceStatus*: El estado en el que está la instancia.
- *DBName*: El nombre del motor de base de datos que utiliza la instancia.
- *Endpoint*: El punto de acceso a la instancia.
- *InstanceCreateTime*: El día y hora de creación de la instancia.

En nuestro caso, en la aplicación Oteador necesitamos monitorizar las instancias que tenemos en Amazon RDS, por lo que no realizamos modificaciones, tan sólo obtendremos la información relevante.

2.2.1.4. Clientes elb y elbv2

Con el cliente 'elb' podemos obtener los balanceadores de carga clasicos, mientras que con el cliente 'elbv2' obtenemos los balanceadores de carga de aplicación y de red; del servicio Elastic Load Balancers de AWS.

Este es un ejemplo sencillo de carga de ambos clientes y listar los balanceadores de carga:

```
import boto3

elb = boto3.client('elb')
elbv2 = boto3.client('elbv2')

response_elb = elb.describe_load_balancers()
```

```
response_elbv2 = elbv2.describe_load_balancers()

print(response_elb)
print(response_elbv2)
```

Listing 2.4: Boto3: Listar balanceadores de carga de ELB y ELBV2

Tras ejecutar la función *describe_load_balancers* podemos obtener información de los balanceadores de carga como:

- *LoadBalancerName*: El nombre del balanceador de carga.
- *Instances*: Los IDs de las instancias de EC2 para el balanceador de carga.
- *DNSName*: El nombre del DNS del balanceador de carga.
- *CreatedTime*: El día y hora de creación del balanceador de carga.
- *Type*: El tipo del balanceador de carga.

En nuestro caso, en la aplicación Oteador necesitamos monitorizar los balanceadores de carga que tenemos en Elastic Load Balancers de AWS, por lo que no realizamos modificaciones, tan sólo obtendremos la información importante.

2.2.1.5. Cliente lambda

Con el cliente 'lambda' podemos obtener las funciones de AWS Lambda desplegadas.

Este es un ejemplo sencillo de carga del cliente y listar las funciones Lambda:

```
import boto3

lambda = boto3.client('lambda')

response = lambda.list_functions()

print(response)
```

Listing 2.5: Boto3: Listar funciones Lambda

Tras ejecutar la función *list_functions* podemos obtener información de las funciones como:

- *FunctionName*: El nombre de la función.
- *Runtime*: El entorno de ejecución de la función.
- *Role*: El role de ejecución de la función.
- *Handler*: La función a la que que Lambda invoca al ejecutarse.
- *CodeSize*: El tamaño de la implementación de la función.
- *LastModified*: El día y hora de la última modificación en la función.

En nuestro caso, en la aplicación Oteador necesitamos monitorizar las funciones que tenemos en AWS Lambda, por lo que sólo obtendremos la información de interés.

2.2.1.6. Cliente autoscaling

Con el cliente 'autoscaling' podemos obtener los grupos de auto-escalado en AWS Auto Scaling.

Este es un ejemplo sencillo de carga del cliente y listar los grupos de auto-escalado:

```
1 import boto3
2
3 autoscaling = boto3.client('autoscaling')
4
5 response = autoscaling.describe_auto_scaling_groups()
6
7 print(response)
```

Listing 2.6: Boto3: Listar grupos de auto-escalado

Tras ejecutar la función *describe_auto_scaling_groups()* podemos obtener información de los grupos de auto-escalado como:

- *AutoScalingGroupName*: El nombre del grupo de auto-escalado.
- *LaunchConfigurationName*: El nombre de la configuración de inicio asociada.
- *DesiredCapacity*: El tamaño deseado del grupo.
- *MinSize*: El tamaño mínimo del grupo.
- *MaxSize*: El tamaño máximo del grupo.
- *Instances*: Las instancias de EC2 asociadas con el grupo.

En nuestro caso, en la aplicación Oteador necesitamos monitorizar los grupos de auto-escalado tenemos en AWS Auto Scaling, por tanto obtendremos la información relevante.

2.3 Javascript

Javascript es un lenguaje de programación interpretado, orientado a objetos, basado en prototipos e imperativo. También es un lenguaje de tipado débil o dinámico.

Además, Javascript permite realizar diferentes actividades complejas en una página web, por lo que no muestra sólo información estática. Entre las distintas opciones que nos ofrece tenemos:

- Actualización de contenido en tiempo real.
- Interactuar con mapas o animaciones gráficas 2D/3D.
- Controlar archivos multimedia.

En las aplicaciones cliente-servidor, este lenguaje se utiliza principalmente en la parte cliente (front-end) como parte del navegador o aplicación web. Aunque también puede utilizarse en la parte servidor (back-end), ejecutándose con Node.js [34]. Node.js es un entorno en tiempo de ejecución multiplataforma de código abierto, asíncrono y con, entrada y salida de datos en una arquitectura orientada a eventos.

No obstante, hay muchas herramientas que pueden llegar a complementar al propio lenguaje de programación de Javascript: librerías y frameworks.

- **Las librerías (biblioteca):**

  Las librerías hacen referencia a una colección de programas, tanto si contiene clases, componentes u otros subprogramas. Su función es facilitar el desarrollo de las aplicaciones, con el uso de funciones que ya están implementadas en las librerías.

  Una librería se desarrolla para ser usada de una manera determinada y por tanto, sus funciones se han desarrollado ajustándose las unas a las otras. Por ejemplo, la librería D3.js, permite la visualización de datos, la realización de pequeñas tablas, diagramas o estadísticas e incluso representaciones gráficas complejas que incluyan animaciones y opciones para la interacción.

  Otra de las limitaciones que poseen las librerías es que siempre están vinculadas a un software que accede a las funciones correspondientes de dicha librería de programas, es decir, las bibliotecas solo funcionan en el entorno de un programa y no se pueden ejecutar de manera independiente.

- **Los frameworks:**

  El framework es un tipo especial de biblioteca de clases y representan la arquitectura de software de una aplicación. De esta forma, el framework determina el proceso de desarrollo de las aplicaciones.

  Los frameworks poseen modelos concretos de desarrollo que, a su vez, cuentan con diversas funciones o bien, varias librerías; y se usan para desarrollar aplicaciones nuevas e independientes. Por ejemplo, el framework Vue.js, en el cual se va a basar la aplicación de CloudTrail-Tracker-UI.

2.3.1. Vue.js

Vue.js es un framework de código abierto de Javascript que parte del patrón modelo-vista-controlador, es decir, trata de desacoplar lo máximo posible la interfaz de usuario de la lógica de la aplicación.

Mediante Vue.js podemos construir interfaces de usuario y aplicaciones de página única (SPA), es decir, una aplicación web que cabe en una sola página con el propósito de dar fluidez en la navegación de la aplicación.

A diferencia de otros frameworks menos flexibles, Vue.js permite diseñar las aplicaciones desde cero e ir gradualmente adoptando a los nuevos requerimientos.

Además, la librería básica se centra sólo en la capa de vista y es fácil de integrar con otras librerías o proyectos existentes.

Por tanto, dado que Vue.js es un framework muy potente que permite utilizarlo tanto para algo sencillo como algo más complejo, la aplicación CloudTrail-Tracker-UI esta desarrollada con este, permitiendo seguir desarrollando el proyecto y mantener su código.

2.4 Framework Serverless

El framework serverless [9] es una interfaz de línea de comandos de código abierto para crear y desplegar aplicaciones serverless.

Con el framework serverless puedes definir toda la aplicación utilizando tecnologías populares de serverless como AWS Lambda mediante un archivo de configuración. Además, mediante su simple e intuitiva interfaz de línea de comandos se puede desarrollar y desplegar aplicaciones para plataformas en la nube como AWS, Microsoft Azure [4], Google Cloud Platform [2] o IBM OpenWhisk [43].

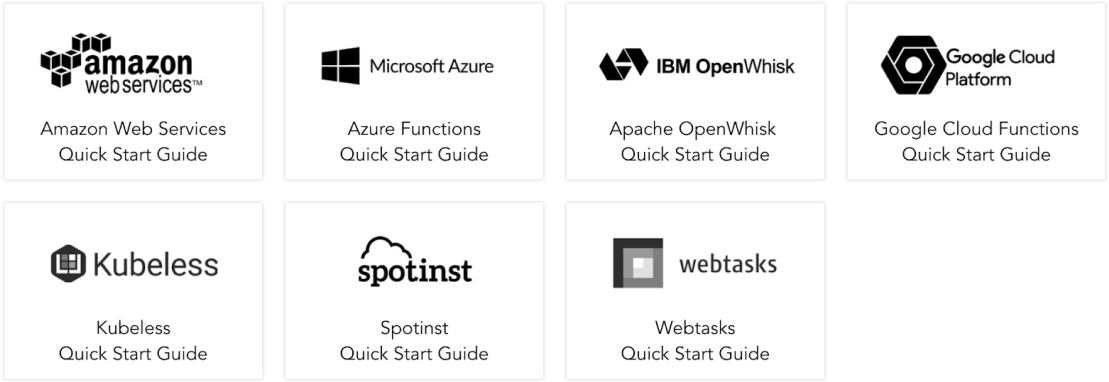

Figura 2.2: Proveedores de servicios serverless [11]

Las aplicaciones serverless aportan tres grandes ventajas frente a las aplicaciones alojadas en servidores tradicionales:

- **Bajo mantenimiento:** No necesitan mucho mantenimiento porque no se tiene un servidor que administrar.
- **Bajo coste:** También tiene un menor coste dado que pagas por cada petición de consulta a la dirección URL donde se aloja la aplicación, de forma que solo se paga cuando se utiliza la aplicación.
- **Fácil escalabilidad:** Es más fácil escalar nuestra aplicación ya que AWS Lambda amplia el servicio dependiendo de la demanda. Además, la aplicación se basa en la tendencia JAMStack [44], basada en el uso de Javascript para el front-end, APIs re-utilizables e independientes y un Markup pre-calculado. La naturaleza estática de una aplicación JAMStack facilita su escalabilidad, debido a la segmentación de esta.

En la aplicación Oteador utilizaremos el framework serverless para crear el template de CloudFormation [45] a partir de un fichero de configuración YAML y así, desplegar la API de forma automática en AWS.

CAPÍTULO 3
CloudTrail-Tracker

CloudTrail-Tracker [46] es una herramienta que proporciona información rápida y beneficiosa sobre el uso de recursos que hacen múltiples usuarios de una misma cuenta de AWS [42], todos ellos con sus propias credenciales de seguridad y con distintos tipos de permisos para controlar las operaciones a realizar sobre los servicios de AWS.

El sistema de la aplicación está compuesto por:

- Una aplicación servidor (back-end) serverless con AWS Lambda, que se activa con los registros de eventos almacenados por AWS CloudTrail en Amazon S3.

  Además, los almacena en Amazon DynamoDB, un servicio de base de datos NoSQL, de esta forma permite un acceso más rápido y una mejor capacidad de consulta.

- Un servicio basado en la tecnología REST que proporciona API Gateway de Amazon Web Services (AWS) con el fin de consultar los eventos que han sido almacenados en la base de datos no relacional DynamoDB de AWS.

- CloudTrail-Tracker-UI, un portal web realizado con el framework Vue.js. Este portal web permite realizar consultas a una API REST para representar de forma visual la información de alto nivel sobre el uso de los recursos en Amazon Web Services (AWS) por todos los usuarios en función de los eventos almacenados en la base de datos no relacional DynamoDB.

Esta herramienta está diseñada como una aplicación serverless, es decir, la lógica de la aplicación se ejecuta utilizando los servicios de AWS, con el fin de minimizar costes respecto al uso de los servicios de Amazon Web Services y al mismo tiempo, mantener una mayor escalabilidad y lograr de manera eficiente las métricas agregadas.

Además, el portal web se puede implementar como un sitio web estático en un bucket de Amazon S3 y si se hace un uso responsable de la herramienta, la plataforma se puede ejecutar de manera gratuita con la cuenta de AWS.

Otra de las ventajas de hacer uso de los servicios de Amazon Web Services para obtener información acerca del uso de estos mismos, es que los datos de uso nunca salen de la propia cuenta de AWS, por lo que mantenemos una cierta seguridad.

En la siguiente figura se describe la arquitectura:

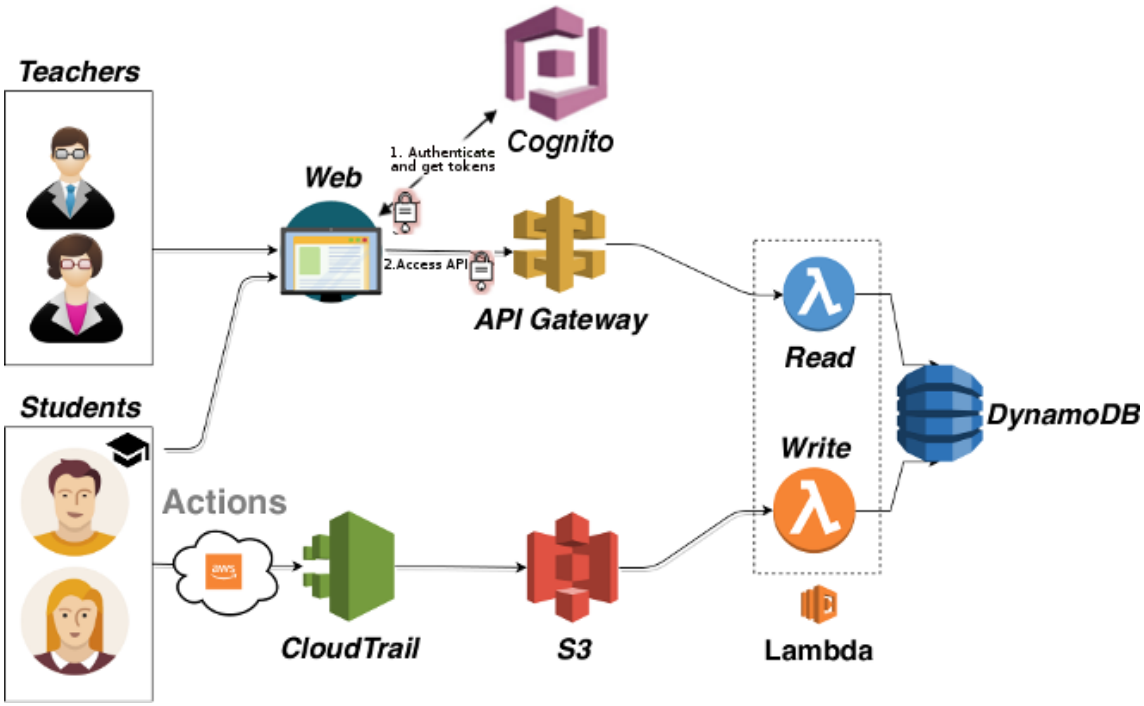

Figura 3.1: Arquitectura del sistema CloudTrail-Tracker[46]

A continuación, hablaremos de la aplicación CloudTrail-Tracker-UI, la cual se encarga de invocar a la herramienta CloudTrail-Tracker y mostrar los resultados en forma de gráficas y tablas para su correcta interpretación.

3.1 CloudTrail-Tracker-UI

CloudTrail-Tracker-UI [47] es un portal web basado en el framework Vue.js que consulta la API de la aplicación CloudTrail-Tracker.

Mediante este portal web se muestra visualmente la información agregada de alto nivel relacionada con el uso que se hace de los recursos en Amazon Web Services (AWS) por los diferentes usuarios que acceden a dicha cuenta de AWS y en función, de la información de los eventos.

Para poder hacer uso de dicho portal web es necesario tener un pool de usuarios en la cuenta de AWS. Por ello, se requiere el uso de un Cognito User Pool existente de AWS.

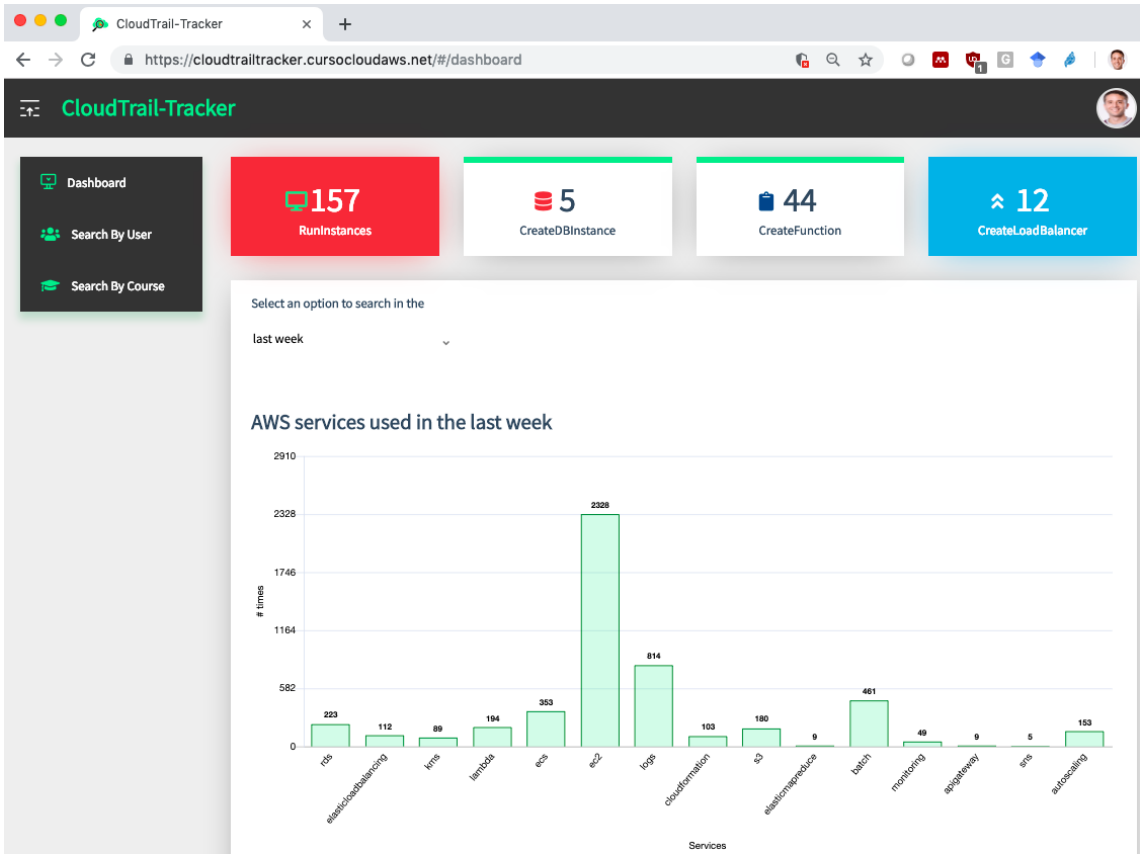

Figura 3.2: Dashboard con el framework CloudTrail-Tracker-UI [47]

En nuestro caso, en la aplicación Oteador usaremos el mismo pool de usuarios que usa la aplicación de CloudTrail-Tracker.

CAPÍTULO 4
Oteador, monitorización serverless de recursos en AWS

En este apartado se describe en qué consiste el sistema desarrollado, el patrón arquitectónico que sigue y las dificultades encontradas para integrar el API de Oteador a la aplicación CloudTrail-Tracker-UI.

4.1 Descripción del sistema

Oteador es una aplicación serverless encargada de consultar en tiempo real, en qué estado y qué uso se hace de los servicios de Amazon Web Services (AWS) con el fin de supervisar el uso de la cuenta de AWS.

Los servicios de AWS que pueden ser consultados son:

- Amazon EC2
- Amazon RDS
- Amazon S3
- Amazon Auto Scaling Groups
- Amazon Elastic Load Balancing
- Amazon Elastic IP
- Amazon Lambda

La aplicación se invoca mediante llamadas al API de Oteador desplegada en la nube o bien, en el portal web de CloudTrail-Tracker donde se ha añadido un nuevo apartado de acceso a ella. Ademas, en el panel añadido a CloudTrail-Tracker también se muestran los costes asociados a algunos servicios.

4.2 Arquitectura del sistema

La arquitectura del sistema implementada en nuestro sistema está basada en las arquitecturas serverless donde la lógica de negocio se ejecuta en los servicios

alojados en la plataforma de cloud público, de forma que el navegador tan sólo debe hacer llamadas a los puntos de acceso a estos servicios.

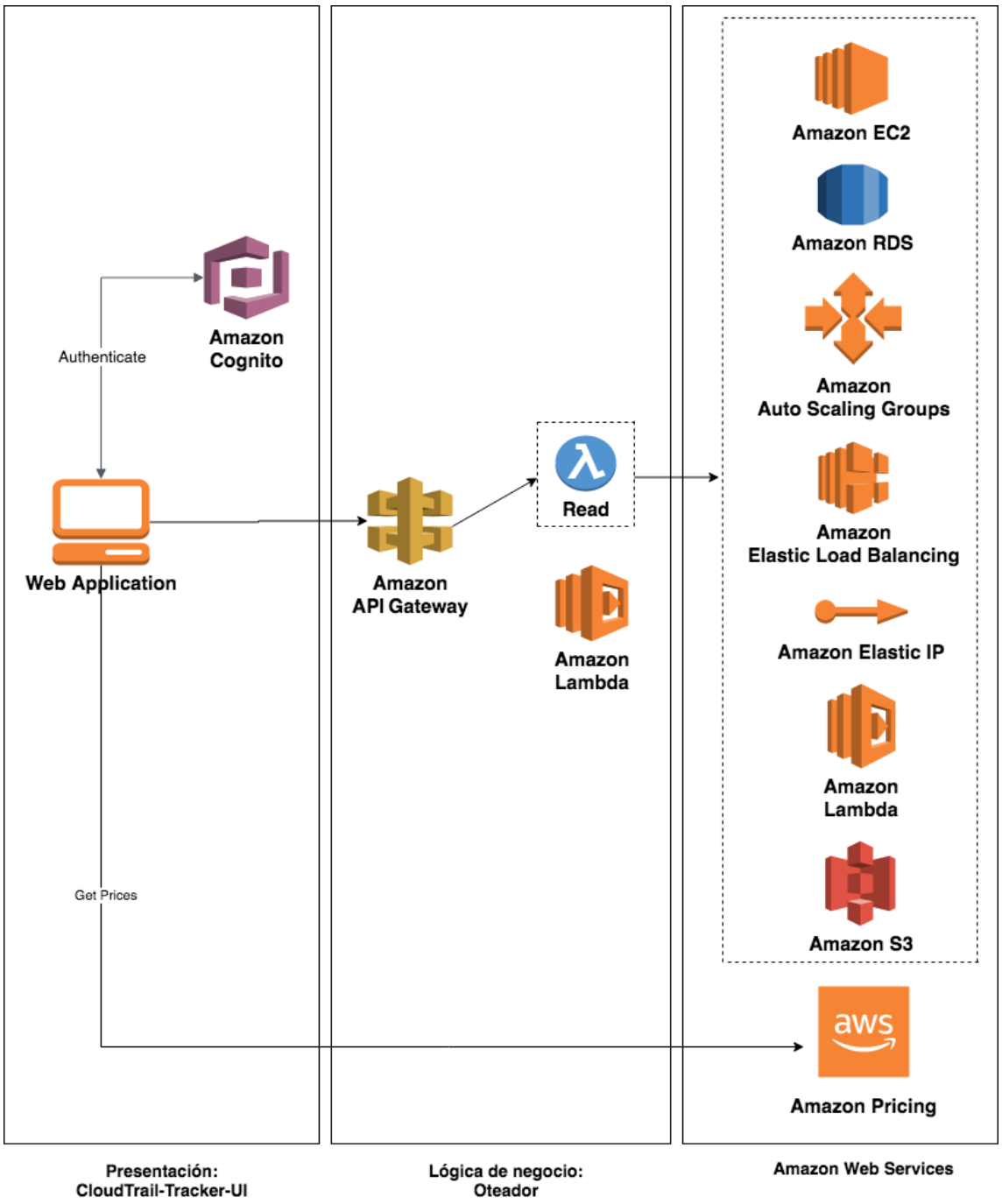

Figura 4.1: Arquitectura del sistema

Como se muestra en la figura 4.1, las diferentes capas de la arquitectura del sistema son:

- **Presentación:** La capa de presentación abarca la interfaz gráfica HTML, los controladores de las vistas en JavaScript, los componentes propios o externos y la comunicación con la capa de la lógica de negocio mediante la invocación a la API de Oteador realizada en Amazon API Gateway. También se

encarga de mostrar los resultados obtenidos tras la invocación a la capa de la lógica de negocio.

Para esta capa, la aplicación CloudTrail-Tracker ya tenía el proyecto asociado CloudTrail-Tracker-UI, con el que nos hemos integrado y así, añadir un apartado más dentro de la aplicación CloudTrail-Tracker.

- **Lógica de negocio:** La capa de la lógica de negocio se encarga de convertir las llamadas a los diferentes endpoints de la API de Oteador en consultas a los servicios de Amazon AWS y devolver el resultado a la capa de presentación.

  La aplicación Oteador se ejecuta en una función Lambda de AWS para consultar el estado y uso que han los usuarios de los recursos de la misma plataforma.

A continuación, explicaremos cómo hemos integrado nuestra aplicación Oteador con el portal web CloudTrail-Tracker-UI.

4.3 Creación de la API de Oteador

La API de Oteador contiene todas las consultas que se realizan a la API de AWS para obtener la información sobre los distintos recursos.

En primer lugar, se ha creado una función AWS Lambda con Python y la librería boto3 para obtener la información de los recursos. Esta función lambda se invoca mediante eventos HTTP.

Para desplegar la función AWS Lambda hemos utilizado el framework Serverless, el cual crea automáticamente cualquier infraestructura necesario para cada tipo de evento, en este caso el punto de acceso en Amazon API Gateway; y configura la función AWS Lambda para que espere a ser invocada por los eventos.

En nuestro caso el fichero para desplegar con el framework Serverless contiene tres eventos de tipo HTTP:

```
1
2  functions:
3    Consultas:
4      handler: lambda_functions/handler.handler
5      runtime: python3.6
6      events:
7        - http:
8            path: services/region/{region}
9            method: get
10           cors: true
11           integration: lambda
12           request:
13             parameters:
14               querystrings:
15                 count: false
16             template:
17               application/json: '{"region":"$input.params(''region''
                     )"}'
18       - http:
```

```
19          path: services/{service}/region/{region}
20          method: get
21          cors: true
22          integration: lambda
23          request:
24            parameters:
25              querystrings:
26                count: false
27            template:
28              application/json: '{"service" : "$input.params(''
                    service'')", "region":"$input.params(''region'')"}
                    '
29
30      - http:
31          path: services/{service}/region/{region}/state/{state}
32          method: get
33          cors: true
34          integration: lambda
35          request:
36            parameters:
37              querystrings:
38                count: false
39            template:
40              application/json: '{"service" : "$input.params(''
                    service'')", "region":"$input.params(''region'')",
                     "state":"$input.params(''state'')"}'
```

Listing 4.1: Fichero Serverless.yml

Estos eventos pueden invocarse dependiendo del path al que se acceda y los parámetros de entrada (service, region y state):

- /services/region/{region}
- /services/{service}/region/{region}
- /services/{service}/region/{region}/state/{state}

Por tanto, la funcion AWS Lambda realizará unas consultas u otras, dependiendo de los eventos de entrada:

```
1  def handler(event, context):
2
3      if event['service'] == "AllBuckets":
4          return action("AllBuckets",event.get('region'))
5
6      elif event['service'] == 'AllInstancesEC2':
7          return action("AllInstancesEC2",event.get('region'))
8
9      elif event['service'] == 'AllInstancesRDS':
10         return action("AllInstancesRDS",event.get('region'))
11
12     elif event['service'] == 'InstancesByStateEC2':
13       if event.get('state'):
14         return action("InstancesByStateEC2",event.get('region'),
               event.get('state'))
15
16     elif event['service'] == 'InstancesByStateRDS':
```

```
      if event.get('state'):
          return action("InstancesByStateRDS",event.get('region'),
              state=event.get('').get('state'))

    elif event['service'] == 'ElasticLoadBalancing':
        return action("ElasticLoadBalancing",event.get('region'))

    elif event['service'] == 'AutoScallingGroups':
        return action("AutoScallingGroups",event.get('region'))

    elif event['service'] == 'ElasticIP':
        return action("ElasticIP",event.get('region'))

    else:
        return 'No accion'
```

Listing 4.2: Handler.py

Una vez se ha desplegado la API correctamente, podemos configurarla en en servicio de API Gateway para que los usuarios se autentifiquen mediante un pool de usuarios de Cognito User Pool previamente creado. Además, en la herramienta API Gateway también podemos obtener la definición de la API con Swagger tal y como aparece en el Anexo I : Definición de la API.

4.4 Integración con CloudTrail-Tracker-UI

La integración de nuestra aplicación con CloudTrail-Tracker-UI no ha sido una tarea trivial dado que había que tener en cuenta diferentes factores:

- La versión instalada de Node.js.
- La versión instalada de npm.
- Los plugins de Vue.js cargados.

En primer lugar, para ejecutar el proyecto CloudTrail-Tracker-UI es necesario tener en cuenta la versión de Node.js, que debe ser la versión 12.

En segundo lugar, también debemos tener en cuenta la versión de npm instalada, que debe ser la versión 6.

En tercer lugar, debido a un cambio de compatibilidad en Bootstrap 4, la biblioteca multiplataforma de código abierto que utiliza Vue.js, se requiere realizar estos cambios en el archivo *build/webpack.dev.conf.js*:

- Hemos importado y cargado el plugin VueLoaderPlugin de Vue.js.
- Hemos inicializado el plugin VueLoaderPlugin dentro de la sección plugins del archivo.

```
const VueLoaderPlugin = require('vue-loader/lib/plugin')
...
new VueLoaderPlugin()
```

Listing 4.3: Cargar e inicializar VueLoaderPlugin

Finalmente, con el portal web CloudTrail-Tracker-UI ejecutándose hemos realizado la integración con nuestra aplicación Oteador mediante la realización de los siguientes pasos:

- Modificación del archivo *src/api.js* para añadir la URL de nuestra API.

```
module.exports = {
  url: {
    "general": "https://api.cursocloudaws.net/tracker/"
    "oteador": "https://api.cursocloudaws.net/oteador/"
  }
}
```

Listing 4.4: Añadir API Oteador

- Se ha utilizado el mismo pool de usuarios de Cognito User Pool configurado en el archivo *src/env.js*. De esta forma, ambas APIs se validan con los mismos usuarios.

- Creación de un botón "Oteador" en el panel izquierdo para redireccionar a nuestro panel.

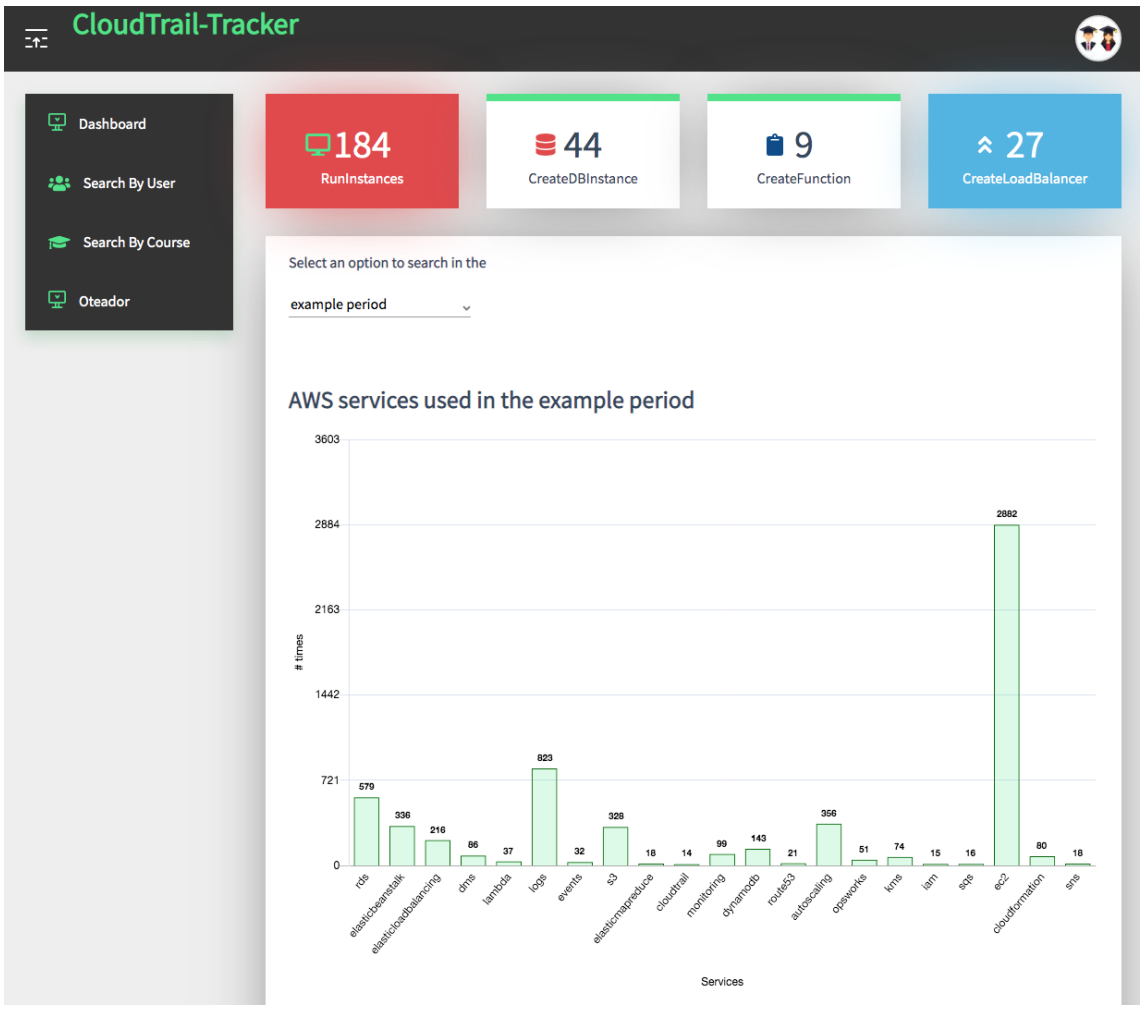

Figura 4.2: Botón Oteador

- Se ha añadido la re-dirección del nuevo panel en el archivo */src/router/index.js*:

```
1
2 import Oteador from '@/components/Oteador/Oteador.vue'
3 ...
4 { path: '/oteador', component: Oteador, beforeEnter:
    requireAuth}
```

Listing 4.5: Re-direccionamiento a Oteador

- Creación del nuevo panel Oteador:

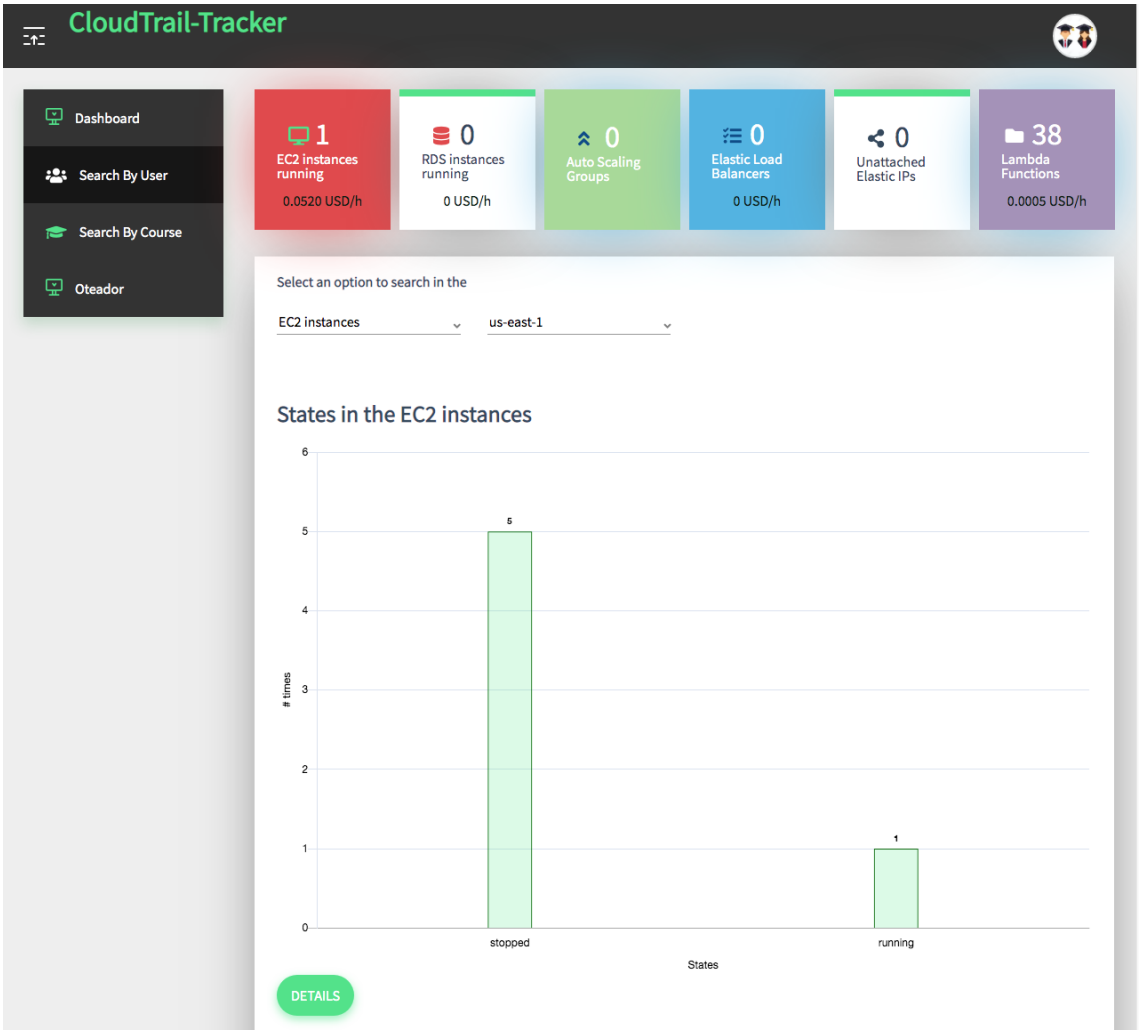

Figura 4.3: Panel Oteador

4.5 Funcionalidades

El sistema implementado para monitorizar los servicios desplegados en Amazon AWS aporta varias funcionalidades, ya que su finalidad es mostrar de manera sencilla las instancias de los servicios creados.

4.5.1. Información sobre recursos y costes

Al acceder a la aplicación nos aparece el panel de Oteador donde podemos ver en la parte de arriba seis widgets:

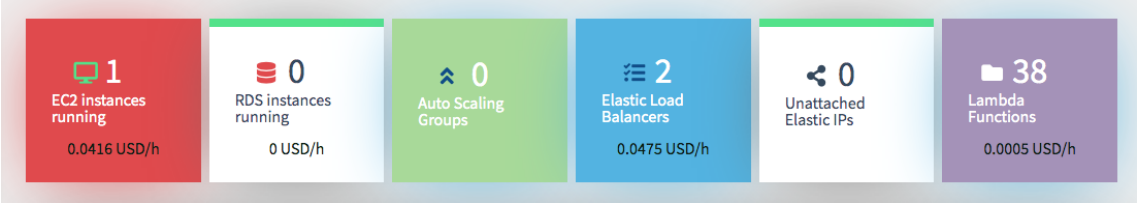

Figura 4.4: Widgets Oteador

Como podemos ver la Figura 4.4 nos aporta la siguiente información:

- *EC2 instances running*, en este widget nos aparece el número de instancias de Amazon EC2 que se encuentran en estado "running", y su coste horario.
- *RDS instances running*, nos aparece el número de instancias de Amazon RDS que se encuentran en estado "running" , y su coste horario.
- *Auto Scaling Groups*, en el siguiente widget nos aparece el número de grupos de auto-escalado de AWS Auto Scaling, que tienen como tamaño de grupo un valor mayor que cero. Dado que no se aplican cargos adicionales por el uso de AWS Auto Scaling no se muestra información sobre costes horarios.
- *Elastic Load Balancers*, nos aparece el número de balanceadores de carga de Amazon Elastic Load Balancing y su coste horario.
- *Unattached Elastic IPs*, en este widget nos aparece el número de direcciones IP elásticas de Amazon EC2 que no están conectadas a una instancia.

En primer lugar, debemos cargar el cliente de pricing e instanciar la clase para poder acceder a las distintas funciones de pricing.

```
1 var Pricing = require("aws-sdk/clients/pricing");
2
3 var pricing = new Pricing({
4     region: "us-east-1",
5     credentials: AWS.config.credentials
6 });
```

Listing 4.6: Cargar cliente pricing e instanciación

En segundo lugar, llamaremos a la función *getProducts* para obtener los precios de los distintos servicios.

- **Obtención de los precios de Amazon EC2**

  Para obtener los precios de Amazon EC2 previamente necesitamos asignar unos parámetros que se le pasan a la función *getProducts* para así filtrar los resultados correctamente.

```
var params = {
    Filters: [
        {
            Field: "instanceType",
            Type: "TERM_MATCH",
            Value: type_service
        },
        {
            Field: "location",
            Type: "TERM_MATCH",
            Value: locationDescription
        },
        {
            Field: "operatingSystem",
            Type: "TERM_MATCH",
            Value: "Linux"
        },
        {
            Field: "tenancy",
            Type: "TERM_MATCH",
            Value: "Shared"
        },
        {
            Field: "capacitystatus",
            Type: "TERM_MATCH",
            Value: "Used"
        },
        {
            Field: "preInstalledSw",
            Type: "TERM_MATCH",
            Value: "NA"
        }
    ],
    FormatVersion: "aws_v1",
    MaxResults: 1,
    ServiceCode: "AmazonEC2"
};

pricing.getProducts(params, function (err, data) {
    if (err) console.log(err, err.stack);
    else {
        var priceList = data["PriceList"];
        var priceOnDemand = priceList[0]["terms"]["OnDemand"];
        var priceDimensions = Object.values(priceOnDemand)[0][
            "priceDimensions"
        ];
        var pricePerUnit = Object.values(priceDimensions)[0][
            "pricePerUnit"
        ];
        return pricePerUnit["USD"];
    }
});
```

Listing 4.7: Obtener precios de Amazon EC2

- **Obtención de los precios de Amazon RDS**

  Para obtener los precios de Amazon RDS también necesitamos asignar otros parámetros de entrada a la función *getProducts* para así filtrar los resultados correctamente.

```
var params = {
    Filters: [
        {
            Field: "instanceType",
            Type: "TERM_MATCH",
            Value: type_service
        },
        {
            Field: "databaseEngine",
            Type: "TERM_MATCH",
            Value: "MySQL"
        },
        {
            Field: "location",
            Type: "TERM_MATCH",
            Value: locationDescription
        }
    ],
    FormatVersion: "aws_v1",
    MaxResults: 1,
    ServiceCode: "AmazonRDS"
};

pricing.getProducts(params, function (err, data) {
    if (err) console.log(err, err.stack);
    else {
        var priceList = data["PriceList"];
        var priceOnDemand = priceList[0]["terms"]["OnDemand"];
        var priceDimensions = Object.values(priceOnDemand)[0][
            "priceDimensions"
        ];
        var pricePerUnit = Object.values(priceDimensions)[0][
            "pricePerUnit"
        ];

        return pricePerUnit["USD"];
    }
});
```

Listing 4.8: Obtener precios de Amazon RDS

- **Obtención de los precios de Amazon Elastic Load Balancing**

  Para obtener los precios de Amazon Elastic Load Balancing necesitamos asignar parámetros de entrada, siendo uno de ellos el que filtra por tipo de balanceador de carga y así, invocar a la función *getProducts* y obtener los resultados esperados.

```
var params = {
    Filters: [
        {
            Field: "location",
            Type: "TERM_MATCH",
```

```
            Value: locationDescription
        },
        {
            Field: "usagetype",
            Type: "TERM_MATCH",
            Value: "LoadBalancerUsage"
        }
    ],
    FormatVersion: "aws_v1",
    MaxResults: 1,
    ServiceCode: "AWSELB"
};

switch (type_service) {
    case "classic":
        params.Filters.push({
            Field: "operation",
            Type: "TERM_MATCH",
            Value: "LoadBalancing"
        });
        break;
    case "application":
        params.Filters.push({
            Field: "operation",
            Type: "TERM_MATCH",
            Value: "LoadBalancing:Application"
        });
        break;
    case "network":
        params.Filters.push({
            Field: "operation",
            Type: "TERM_MATCH",
            Value: "LoadBalancing:Network"
        });
        break;
    default:
}

pricing.getProducts(params, function (err, data) {
    if (err) console.log(err, err.stack);
    else {
        var priceList = data["PriceList"];
        var priceOnDemand = priceList[0]["terms"]["OnDemand"];
        var priceDimensions = Object.values(priceOnDemand)[0][
            "priceDimensions"
        ];
        var pricePerUnit = Object.values(priceDimensions)[0][
            "pricePerUnit"
        ];
        return pricePerUnit["USD"];
    }
});
```

Listing 4.9: Obtener precios de Amazon Elastic Load Balancing

- **Obtención de los precios de Amazon Lambda**

  Para obtener los precios de Amazon Lambda también asignaremos unos parámetros de entrada para llamar a la función *getProducts*.

```
var params = {
    Filters: [
        {
            Field: "location",
            Type: "TERM_MATCH",
            Value: locationDescription
        },
        {
            Field: "usagetype",
            Type: "TERM_MATCH",
            Value: "Lambda-GB-Second"
        }
    ],
    FormatVersion: "aws_v1",
    MaxResults: 1,
    ServiceCode: "AWSLambda"
};
pricing.getProducts(params, function (err, data) {
    if (err) console.log(err, err.stack);
    else {
        var priceList = data["PriceList"];
        var priceOnDemand = priceList[0]["terms"]["OnDemand"];
        var priceDimensions = Object.values(priceOnDemand)[0][
            "priceDimensions"
        ];
        var pricePerUnit = Object.values(priceDimensions)[0][
            "pricePerUnit"
        ];

        return pricePerUnit["USD"];
    }
});
```

Listing 4.10: Obtener precios de Amazon Lambda

4.5.2. Listado de los buckets de Amazon S3

Una de las funcionalidades de nuestro sistema es mostrar de forma gráfica todos los buckets de Amazon S3 que están siendo usados.

Mediante la librería boto3 mencionada anteriormente, realizaremos diversas llamadas a la API proporcionada por Amazon AWS.

En primer lugar, debemos cargar el cliente de Amazon S3 de la librería de boto3.

```
import boto3
s3 = boto3.client('s3', region_name = region)
```

Listing 4.11: Cargar cliente s3 para la región dada

En segundo lugar, obtendremos los buckets de nuestra cuenta de Amazon Web Services con la función *list_buckets*, la cual nos permite listar todos los buckets que tenemos en uso.

```
def list_buckets():
  list_buckets = []
  try:
    for bucket in s3.list_buckets():
    returned.append(bucket.name)

  except Exception as e: print(e)

  return list_buckets;
```

Listing 4.12: Obtener información s3

Finalmente, tras realizarse la consulta a Amazon Web Services podemos obtener un listado de los buckets de Amazon S3.

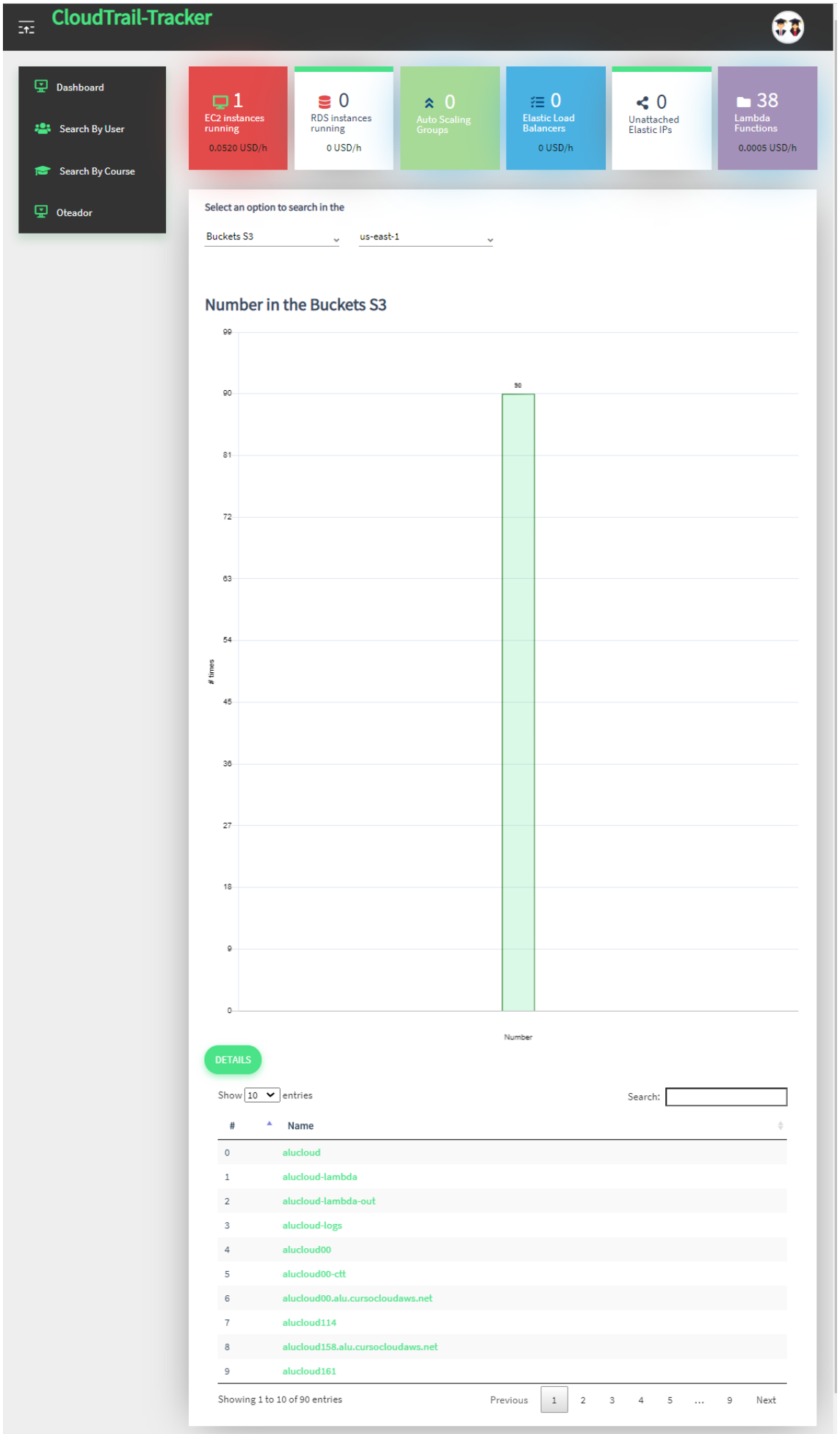

Figura 4.5: Panel Oteador con información Amazon S3

Como podemos ver en la Figura 4.5, en la parte de abajo nos aparece el listado de buckets con la posibilidad de acceder pulsando sobre cada uno de ellos, a la consola de aws del servicio de Amazon S3 para ver o modificar los elementos del bucket seleccionado.

4.5.3. Listado de las instancias de Amazon EC2

Otra de las funcionalidades de nuestro sistema es mostrar de forma gráfica las instancias de EC2 en función de su estado.

Las instancias de EC2 pueden pasar por diferentes estados en su ciclo de vida: ”pending”, ”running”, "shutting-down”, ”stopping”, ”stopped”, ”terminated” o ”rebooting”.

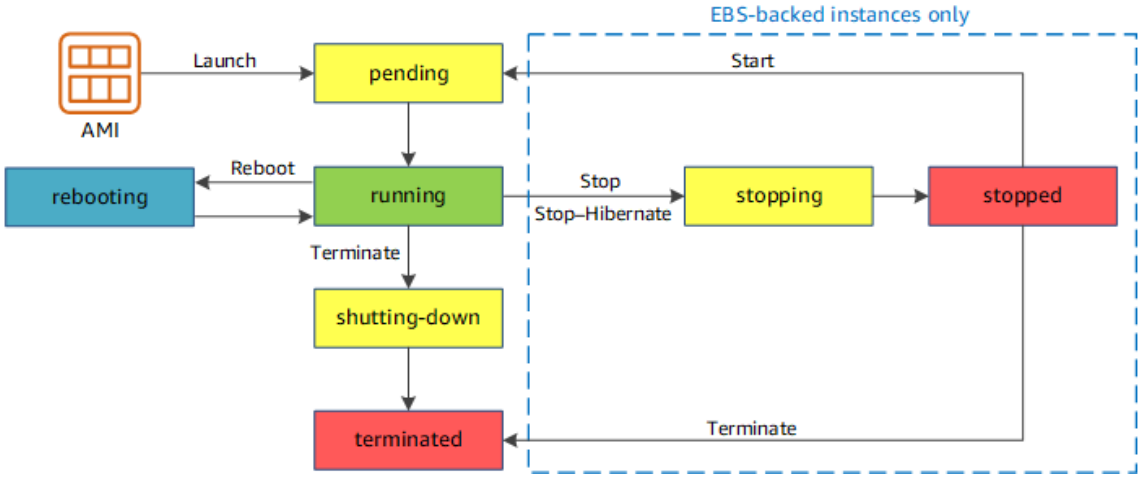

Figura 4.6: Ciclo de vida de una instancia EC2 [48]

En primer lugar, debemos cargar el cliente de Amazon EC2 de la librería de boto3.

```
import boto3
ec2 = boto3.client('ec2', region_name = region)
```

Listing 4.13: Cargar cliente ec2 para la región dada

En segundo lugar, para obtener la información general de las instancias usaremos la función *describe_instances*, la cual nos da información necesaria como el id de la instancia (“InstanceId”), el tipo de instancia (“InstanceType”), el tiempo que lleva lanzado (“LaunchTime”) y el estado en el que se encuentra la instancia (“State”).

En nuestro caso, en la aplicación Oteador vamos a obtener toda la información disponible con la función *describe_instances*, guardándonos la que tenga propietario (“owner”).

```
def list_instances_EC2(region):
  list_instances = []
  try:
```

```
    response = ec2.describe_instances()
    for reservation in response['Reservations']:
      for instance in reservation['Instances']:
        owner = ''
        if ('Tags') in instance):
          for tag in instance['Tags']:
            if 'owner' in tag['Key']:
              owner = tag['Value']
        list_instances.append(getInfoInstanceEC2(instance,owner,
           region))

  except Exception as e: print(e)

  return list_instances;
```

Listing 4.14: Obtener información ec2

Finalmente, tras la realizarse la consulta a Amazon Web Services podemos obtener las instancias de Amazon EC2 desplegadas en una región en concreto y una clasificación por estados.

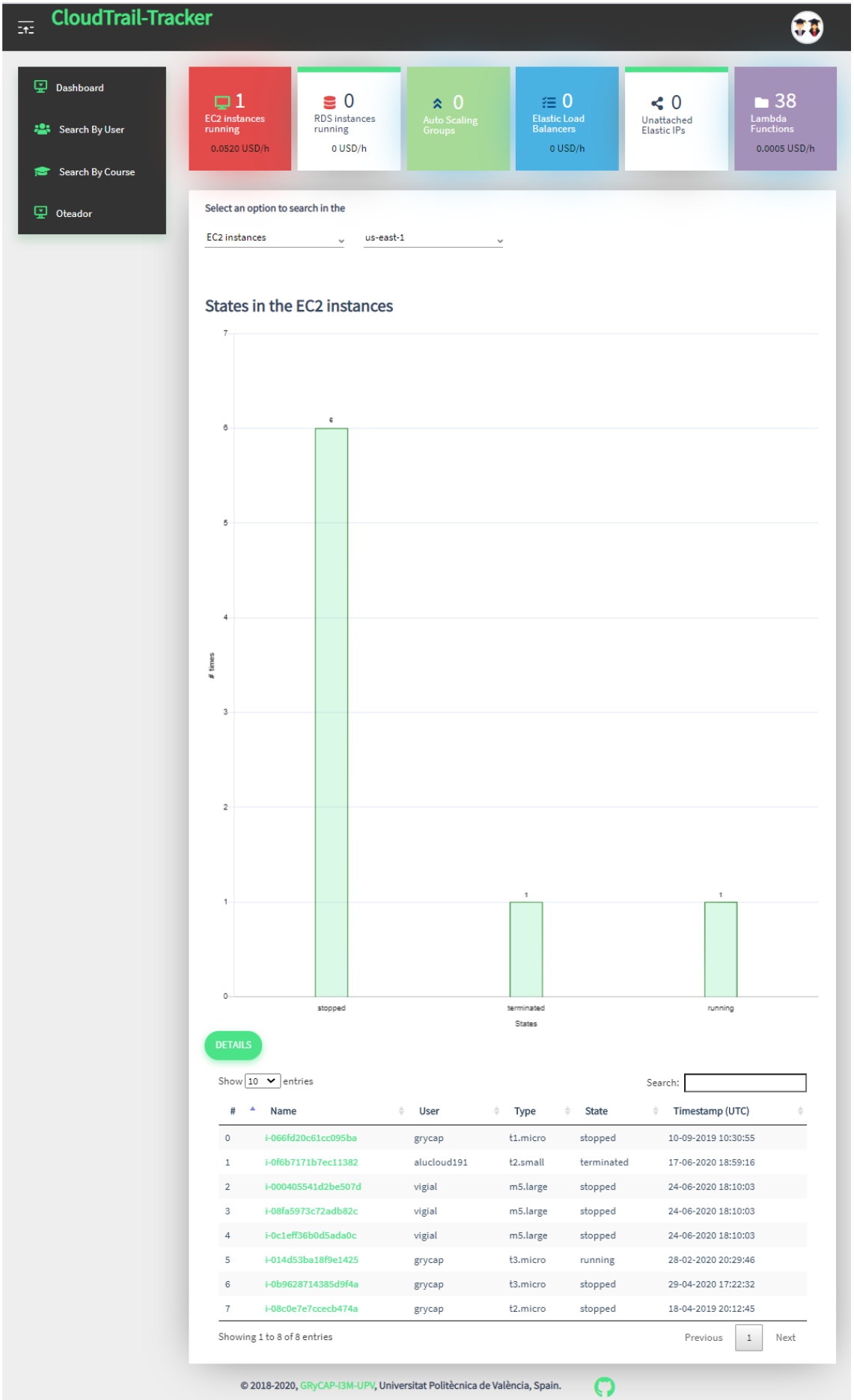

Figura 4.7: Panel Oteador con información Amazon EC2

Como podemos ver en la Figura 4.7, en la parte de abajo nos aparece el listado de las instancias de Amazon EC2 donde nos indica el usuario que ha desplegado la máquina, el tipo de máquina, el estado y el día y hora que se desplegó. Además, podemos seleccionar cada instancia y acceder directamente a la consola de AWS del servicio de Amazon EC2 para ver o modificar el estado de la instancia.

4.5.4. Listado de las instancias de Amazon RDS

También disponemos de la funcionalidad de mostrar de forma gráfica las instancias de Amazon RDS en ejecución, paradas o terminadas.

En primer lugar, debemos cargar el cliente de Amazon RDS de la librería de boto3.

```
import boto3
rds = boto3.client('rds', region_name = region)
```

Listing 4.15: Cargar cliente rds para la región dada

En segundo lugar, obtendremos la información general de las instancias de bases de datos usando la función *describe_db_instances*, la cual nos da información necesaria como el id de la base de datos ("DBInstanceIdentifier"), el tipo de base de datos ("DBInstanceClass"), el estado en el que se encuentra la base de datos ("DBInstanceStatus") que puede ser "available", "creating", "deleting", "modifying", etc.

En nuestro caso, en la aplicación Oteador vamos a obtener toda la información disponible con la función *describe_db_instances*, guardándonos la información de interés.

```
def list_instances_RDS(region):
  list_instances = []
  try:
    response = rds.describe_db_instances()
    for dbinstance in response['DBInstances']:
      list_instances.append(getInfoInstanceRDS(dbinstance,region))
  except Exception as e: print(e)

  return list_instances;
```

Listing 4.16: Obtener información rds

Finalmente, tras la realizarse la consulta a Amazon Web Services podemos obtener las instancias de Amazon RDS desplegadas en una región concreta y una clasificación por estados.

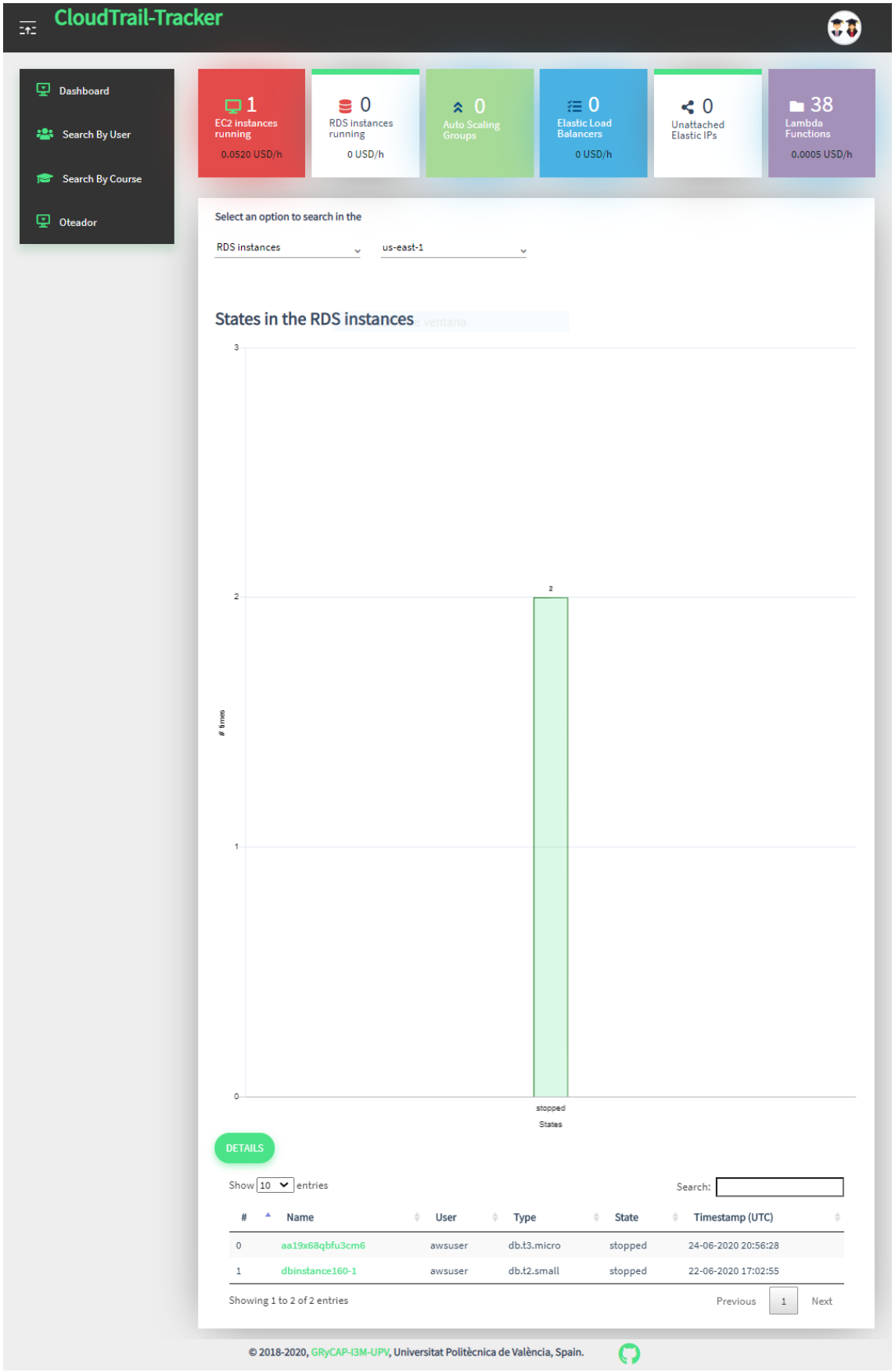

Figura 4.8: Panel Oteador con información Amazon RDS

Como podemos ver en la Figura 4.8, en la parte de abajo nos aparece el listado con las bases de datos de Amazon RDS donde nos indica el usuario que ha desplegado la base de datos, el tipo de base de datos, el estado y el día y hora que se desplegó. Además, podemos seleccionar cada instancia y acceder directamente a la consola de AWS del servicio de Amazon RDS para ver o modificar el estado de la base de datos.

4.5.5. Listado de balanceadores de carga

Otra funcionalidad de la que disponemos es mostrar de forma gráfica los balanceadores de carga de Amazon Elastic Load Balancing.

En primer lugar, debemos cargar los dos clientes de Elastic Load Balancers de la librería de boto3. El cliente elb nos aporta la información de los balanceadores de carga clásicos, mientras que el cliente elbv2 nos muestra la información de los balanceadores de carga de aplicación.

Por tanto, debemos cargar ambos clientes filtrando por la región.

```
1 import boto3
2 elb = boto3.client('elb', region_name = region)
3 elbv2 = boto3.client('elbv2', region_name = region)
```

Listing 4.17: Cargar cliente elb y elbv2 para la región dada

En segundo lugar, obtendremos la información general de las instancias de los balanceadores de carga usando la función *describe_load_balancers*, la cual nos da información necesaria como el nombre del balanceador de carga ("LoadBalancerName"), el nombre del DNS ("DNSName"), el estado en el que se encuentra el balanceador de carga ("State") o bien, el tipo de balanceador de carga ("Type") que puede ser application, network o classic.

En nuestro caso, en la aplicación Oteador vamos a obtener toda la información disponible con la función *describe_load_balancers*, sin aplicar ningún filtro en la búsqueda guardándonos la información de interés.

```
1  def list_elastic_load_balancers(region):
2    list_load_balancers = []
3    try:
4      response = elb.describe_load_balancers()
5      for elbalancer in response['LoadBalancerDescriptions']:
6        list_load_balancers.append(getInfoElasticLoadBalancer(
           elbalancer,region))
7      response2 = elbv2.describe_load_balancers()
8      for elbalancerv2 in response2['LoadBalancers']
         list_load_balancers.append(getInfoElasticLoadBalancer(
         elbalancerv2,region))
9
10   except Exception as e: print(e)
11
12   return list_load_balancers;
```

Listing 4.18: Obtener información elb y elbv2

Finalmente, tras la realizarse la consulta a Amazon Web Services podemos obtener los balanceadores de carga desplegados.

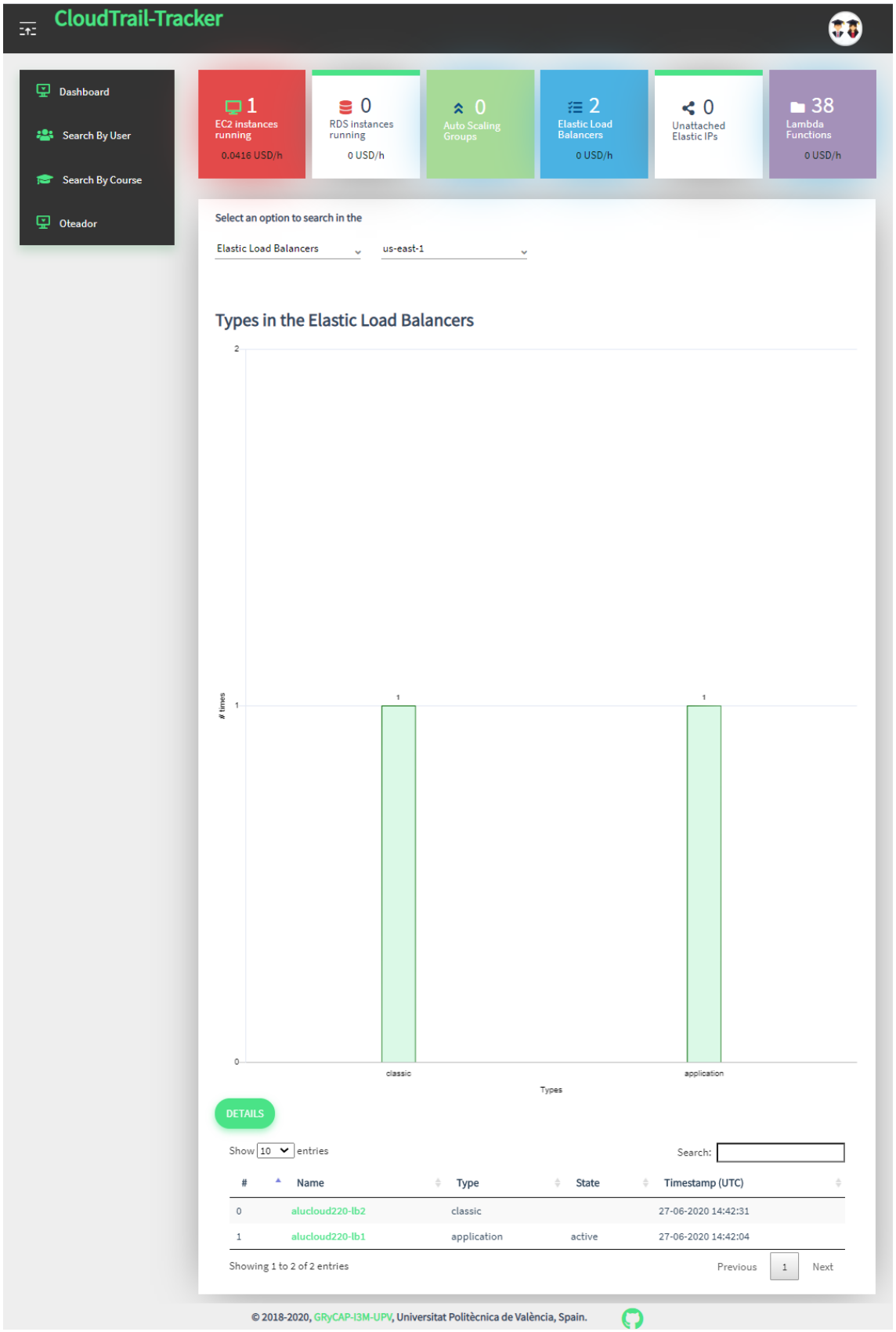

Figura 4.9: Panel Oteador con información Amazon Elastic Load Balancing

Como podemos ver en la Figura 4.9, en la parte de abajo nos aparece el listado con los balanceadores de carga de Amazon Elastic Load Balancing donde nos in-

dica el tipo de balanceador de carga, el estado (los balanceadores de carga de tipo classic no tienen estado) y el día y hora que se desplegó. Además, podemos seleccionar cada balanceador de carga y acceder directamente a la consola de AWS del servicio de Amazon Elastic Load Balancing para ver o modificar el estado del balanceador de carga.

4.5.6. Listado de grupos de auto-escalado creados

También disponemos de la funcionalidad de mostrar de forma gráfica los grupos de auto-escalado de Amazon EC2 Auto Scaling.

En primer lugar, debemos cargar el cliente de EC2 Auto Scaling de la librería de boto3 para una región dada.

```
import boto3
autoscaling = boto3.client('autoscaling', region_name = region)
```

Listing 4.19: Cargar cliente autoscaling para la región dada

En segundo lugar, obtendremos la información general de los grupos de auto-escalado usando la función *describe_auto_scaling_groups*, la cual nos da información necesaria como el nombre del grupo de auto-escalado ("AutoScalingGroupName"), entre otros datos.

En nuestro caso, en la aplicación Oteador vamos a obtener toda la información disponible con la función *describe_auto_scaling_groups*, sin aplicar ningún filtro en la búsqueda guardándonos la información de interés.

```
def list_auto_scaling_groups(REGION):
  list_autoscaling = []
  try:
    response = autoscaling.describe_auto_scaling_groups()
    for autoscalinggroup in response['AutoScalingGroups']:
      list_autoscaling.append(getInfoAutoScalingGroups(
          autoscalinggroup,region))

  except Exception as e: print(e)

  return list_autoscaling;
```

Listing 4.20: Obtener información autoscaling

Finalmente, tras la realizarse la consulta a Amazon Web Services podemos obtener las instancias de Amazon EC2 Auto Scaling desplegadas para una región concreta.

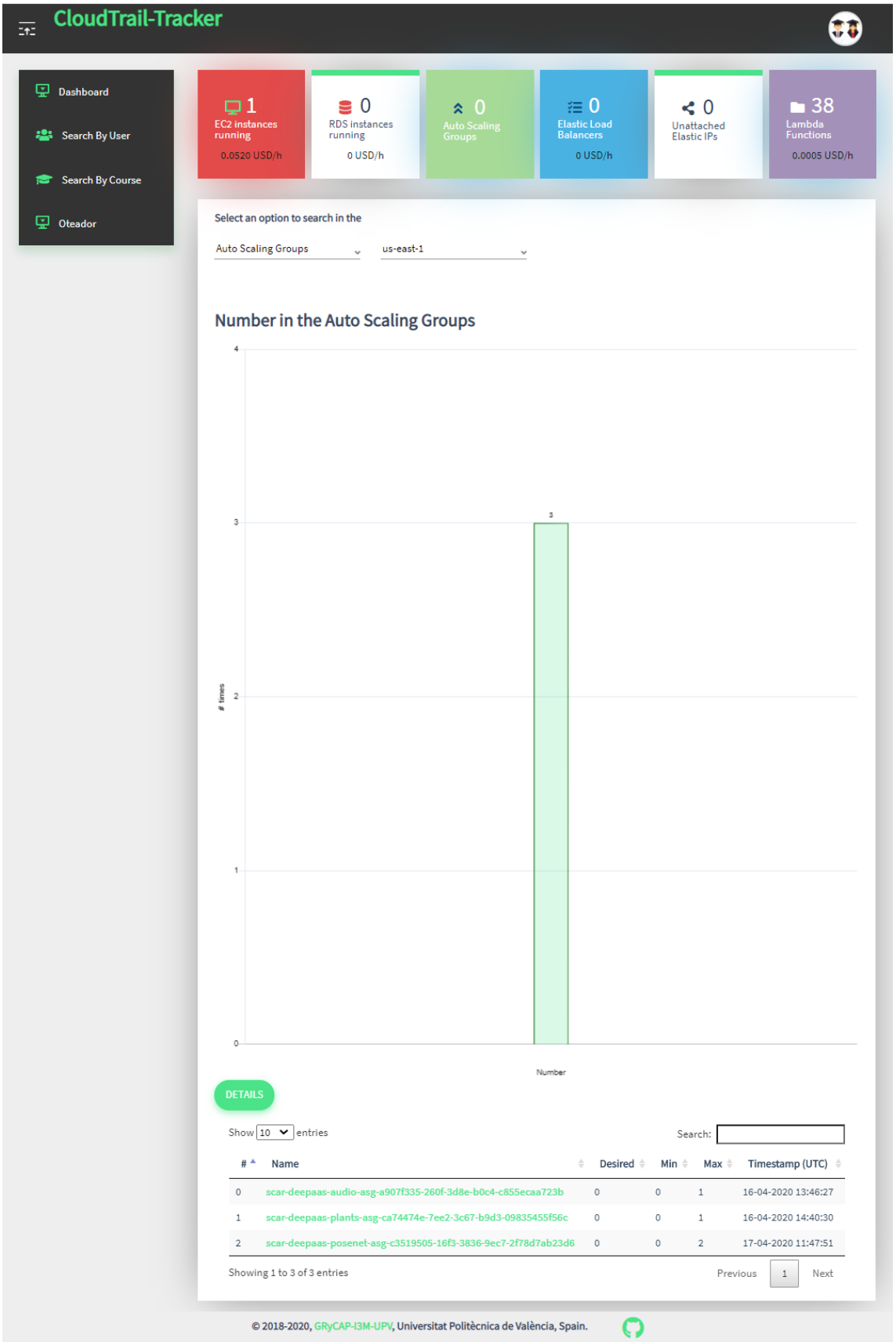

Figura 4.10: Panel Oteador con información Amazon Auto Scaling Groups

Como podemos ver en la Figura 4.10, en la parte de abajo nos aparece el listado con los grupos de auto-escalado de Amazon Auto Scaling donde nos indica el

nombre del grupo de auto-escalado, el tamaño del grupo o desired, el mínimo y el máximo del grupo; y el día y hora de creación. Además, podemos seleccionar cada grupo de auto-escalado y acceder directamente a la consola de AWS del servicio de Amazon Auto Scaling para ver o modificar el grupo de auto-escalado.

4.5.7. Listado de direcciones IP elásticas no conectadas a instancias

Asimismo, disponemos de la funcionalidad de mostrar de forma gráfica las direcciones IP elásticas de Amazon EC2.

En primer lugar, debemos volver a cargar el cliente de Amazon EC2 de la librería boto3.

En segundo lugar, obtendremos la información general de las direcciones IP elásticas usando la función *describe_addresses*, la cual nos da información necesaria como la dirección privada ("PrivateIPAddress") o la dirección pública ("PublicIp").

En nuestro caso, en la aplicación Oteador vamos a obtener toda la información disponible con la función *describe_addresses*, sin aplicar ningún filtro en la búsqueda guardándonos la información de interés.

```
1  def list_elasticIP (region,isEmpty):
2      list_elasticIP = []
3      regionName = get_region_name(region)
4
5      try:
6          response = ec2.describe_addresses()
7          for eIP in response['Addresses']:
8              if (isEmpty == "true"):
9                  if (eIP['InstanceId'] == ''): list_elasticIP.append
                       (getInfoElasticIP(eIP,regionName))
10             else:
11                 list_elasticIP.append(getInfoElasticIP(eIP,
                       regionName))
12
13     except Exception as e: print(e)
14
15     return list_elasticIP
```

Listing 4.21: Obtener direcciones IP elásticas

Finalmente, tras la realizarse la consulta a Amazon Web Services podemos obtener las direcciones IP elásticas de Amazon EC2 desplegadas con una región dada.

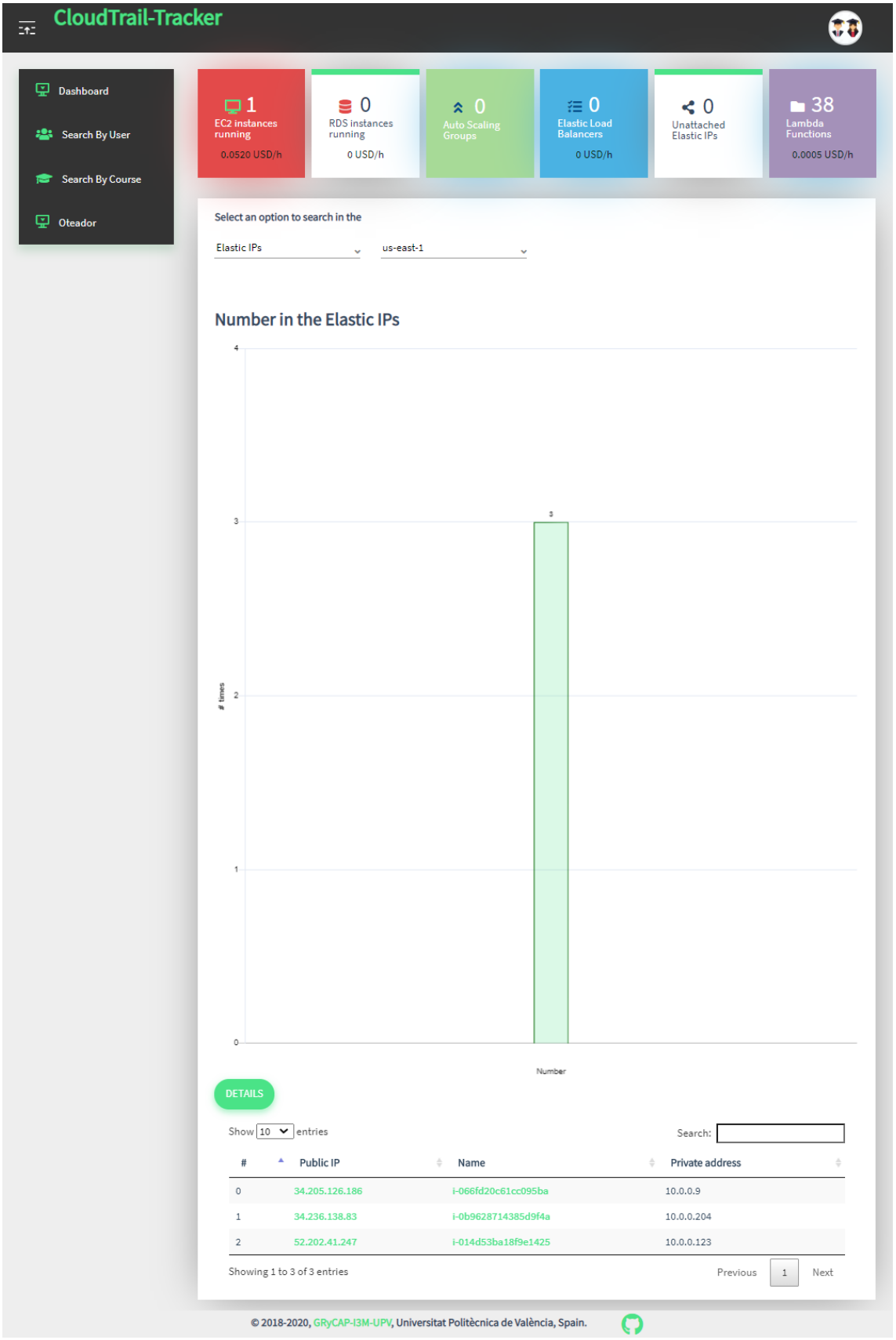

Figura 4.11: Panel Oteador con información Amazon Elastic IP

Como podemos ver en la Figura 4.11, en la parte de abajo nos aparece el listado con las direcciones IP elásticas de Amazon EC2 donde nos indica la dirección

IP pública, el nombre de la instancia en EC2 conectada (puede no haya una instancia conectada), y la dirección IP privada. Además, podemos seleccionar cada dirección IP elástica y acceder directamente a la consola de AWS del servicio de Amazon EC2 para conectar una instancia de EC2 o bien eliminarla.

4.5.8. Listado de funciones Lambda

Por último, se ha implementado la funcionalidad de mostrar de forma gráfica las funciones Lambda de Amazon Lambda.

En primer lugar, debemos cargar el cliente de Amazon Lambda de la librería boto3.

```
1 import boto3
2 lambda = boto3.client('lambda',region_name=region)
```

Listing 4.22: Cargar cliente lambda

En segundo lugar, obtendremos la información general de las funciones lambda desplegadas usando la función *list_functions*, la cual nos da información necesaria como el nombre de la función ("FunctionName") o la fecha de la última modificación realizada ("LastModified").

En nuestro caso, en la aplicación Oteador vamos a obtener toda la información disponible con la función *list_functions*, sin aplicar ningún filtro en la búsqueda guardándonos la información de interés.

```
1  def list_lambda(region):
2      list_lambda = []
3      regionName = get_region_name(region)
4
5      try:
6          response = lambda.list_functions()
7          for function in response['Functions']:
8              list_lambda.append(getInfoLambda(function,regionName))
9
10     except Exception as e: print(e)
11
12     return list_lambda
```

Listing 4.23: Obtener funciones Lambda

Tras la realizarse la consulta a Amazon Web Services podemos obtener las funciones de Amazon Lambda desplegadas con una región dada.

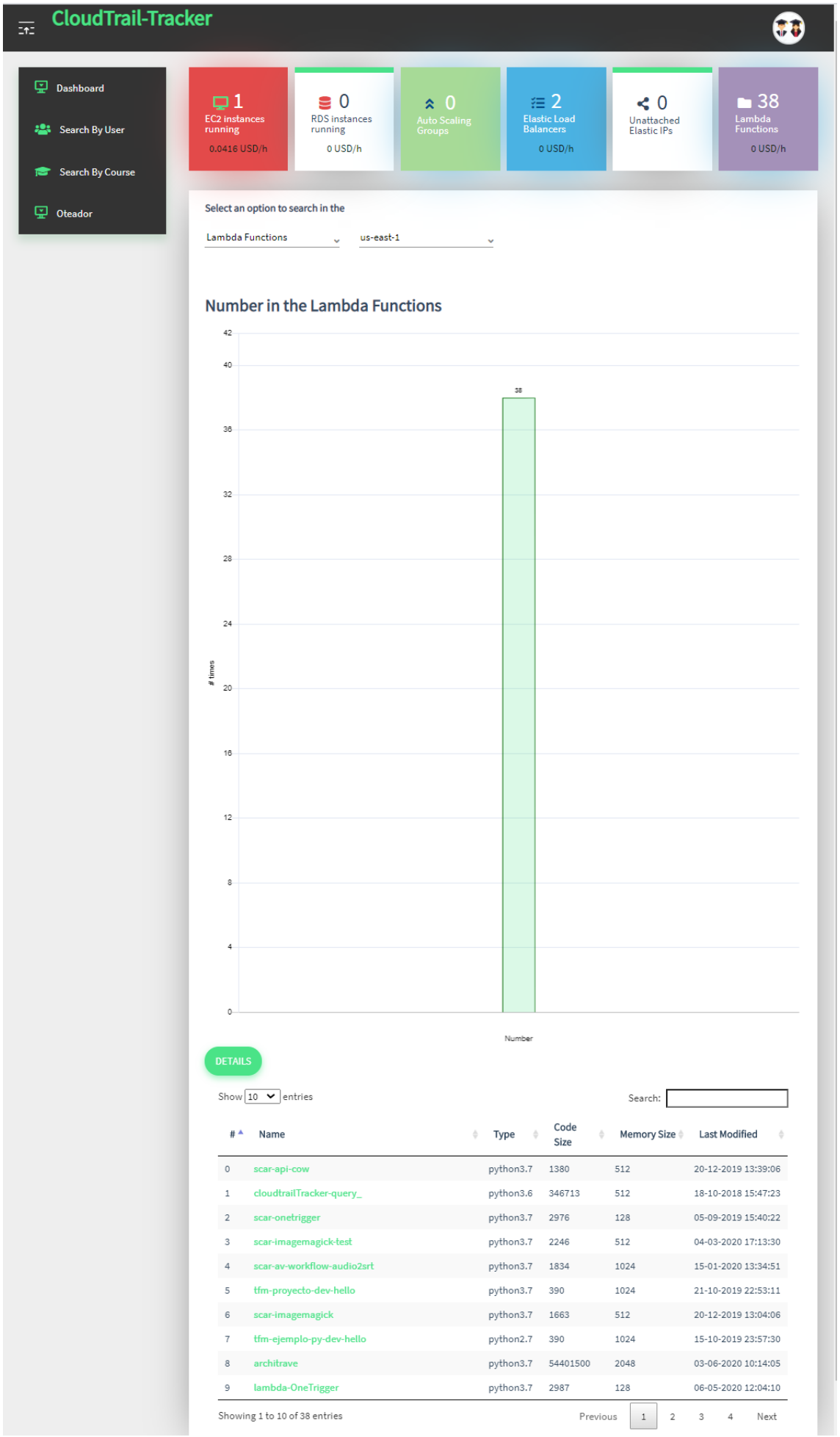

Figura 4.12: Panel Oteador con información Amazon Lambda

Como podemos ver en la Figura 4.12, en la parte de abajo nos aparece el listado con las funciones de Amazon Lambda donde nos indica el nombre, el tipo de entorno de programación, el tamaño del código, la memoria asignada a la función y el día y hora que se realizó la última modificación. Además, también podemos seleccionar cada función y acceder directamente a la consola de AWS del servicio de Amazon Lambda para ver y modificar la función seleccionada.

CAPÍTULO 5
Conclusiones y Trabajos Futuros

Este capítulo esta dedicado a las conclusiones finales del proyecto, extraídas a lo largo del proceso de desarrollo del mismo.

De los objetivos planteados al inicio del trabajo se han podido cumplir todos:

- Abstracción en el acceso a la información.
- Integración con un herramienta existente.
- Obtención del estado de diferentes recursos.
- Obtención del coste horario.
- Accesibilidad mediante una dirección URL.

Gracias a ellos se ha logrado crear un panel web que monitoriza en tiempo real los recursos en uso en AWS y que ha sido integrado con la aplicación CloudTrail-Tracker. Además se ha añadido una funcionalidad nueva, que permite el acceso a la consola de cada recurso monitorizado en AWS para poder gestionar dicha instancia.

Cabe destacar que al inicio del proyecto, hubo que conocer y estudiar la aplicación CloudTrail-Tracker-UI, ya que uno de los objetivos era la integración con esta herramienta ya existente. Esto definió parte del desarrollo técnico del proyecto, para seguir con las estructuras y metodologías ya vigentes, con la finalidad de que una vez completada la integración no se diferenciara la version anterior con la que incluye a Oteador. De forma que el usuario final no percibe dos paneles tan diferentes, sino una única herramienta con más funcionalidades.

Atendiendo al desarrollo técnico del proyecto, al principio hubo que conocer la aplicación CloudTrail-Tracker-UI, porque finalmente es un proyecto finalizado que debes comprender su desarrollo antes de poder integrarse con él. Por lo que las primeras fases fueron de conocimiento y estudio de la herramienta antes de desarrollar la aplicación.

Como posibles mejoras y ampliaciones en el futuro de este proyecto pueden destacarse diferentes líneas de trabajo:

- La primera de ellas es la búsqueda de nuevos elementos visuales que puedan ayudar a mostrar la información de diferentes formas adaptándose a las necesidades.

- Otro de los posibles aspectos a mejorar es que se pueden obtener mas recursos de AWS que resulten de interés como Amazon Simple Queue Service (SQS) para obtener las colas de mensajes que permiten desacoplar y ajustar la escala de microservicios, sistemas distribuidos y aplicaciones sin servidor; o Amazon Elastic Container Service (ECS) para obtener los contenedores desplegados.

Glosario

- **API.** *Application Programming Interface.* Interfaz de programación de aplicaciones: es el conjunto de funciones y procedimientos que ofrece una biblioteca para ser utilizada por otro software abstrayendo de su desarrollo interno.
- **Framework.** Estructura conceptual y tecnológica de soporte definido que puede servir de base para la organización y desarrollo de software.
- **REST.** *Representational state transfer.* Arquitectura software para sistemas hipermedia distribuidos como la World Wide Web o red informática mundial.
- **Endpoint.** Es una dirección URL que puede ser exponer un servicio para ser accedido remotamente.
- **Backend** Es la capa de acceso a datos que permite abstraer la lógica del programa de la interfaz de usuario.
- **AMI.** *Amazon Machine Images.* es una imagen que proporciona la información necesaria para lanzar una instancia en AWS.
- **Kernel.** Software que constituye una parte fundamental del sistema operativo y se ejecuta en modo privilegiado o modo núcleo.
- **DNS.** *Domain Name System.* Método de denominación empleado para nombrar a los dispositivos que se conectan a una red a través de una IP.
- **Widgets** Pequeñas aplicaciones cuyo objetivo es dotar de información visual y facilitar el acceso a las funciones que se utilizan de forma frecuente.

Bibliografía

[1] David Cierco. **Cloud computing: Retos y Oportunidades**. Fundación Ideas, 2011.

[2] **Google Cloud** Consulta a https://cloud.google.com/

[3] **¿Qué es google cloud platform?** Consultado a https://www.doctormetrics.com/google-cloud-platform/

[4] **Microsoft Azure** Consulta a https://azure.microsoft.com/en-us/

[5] **Alibaba Cloud** Consulta a https://eu.alibabacloud.com/

[6] **AWS** Consulta a https://aws.amazon.com/

[7] **Caso práctico de AWS: Atresmedia.** Consultado a https://aws.amazon.com/es/solutions/case-studies/atresmedia/

[8] **Netflix & Amazon Kinesis Data Streams Case Study.** Consultado a https://aws.amazon.com/es/solutions/case-studies/netflix-kinesis-data-streams/

[9] **Serverless Framework Open Source** Consultado a https://serverless.com/open-source

[10] **¿Por qué crear aplicaciones serverless?** Consultado a https://serverless-stack.com/chapters/es/why-create-serverless-apps.html

[11] **Serverless: the time has come toreplace servers with code.** Consultado a https://en.paradigmadigital.com/techbiz/serverless-the-time-has-come-to-replace-servers-with-code/

[12] Mell, Peter, and Tim Grance. **The NIST Definition of Cloud Computing**. NIST Special Publication 800-145, 2011. http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf

[13] **AWS Elastic Beanstalk** Consultado a https://aws.amazon.com/elasticbeanstalk/

[14] **Gmail** Consultado a https://www.google.com/gmail/

[15] **Microsoft Dynamics 365** Consultado a https://dynamics.microsoft.com/

[16] **Margaret Rouse.** Amazon Web Services (AWS) Consultado a https://searchaws.techtarget.com/definition/Amazon-Web-Services

[17] **Amazon Relational Database Service (RDS)** Consultado a https://aws.amazon.com/rds

[18] **AWS Database Migration Service)** Consultado a https://aws.amazon.com/dms/

[19] **PostgreSQL: The World's Most Advanced Open Source Relational Database** Consultado a https://www.postgresql.org/

[20] **MySQL** Consultado a https://www.mysql.com/

[21] **Oracle Database** Consultado a https://www.oracle.com/database/

[22] **Microsoft Data platform** Consultado a https://www.microsoft.com/en-us/sql-server

[23] **Amazon EC2** Consultado a https://aws.amazon.com/ec2

[24] **Amazon S3** Consultado a https://aws.amazon.com/s3

[25] **Cloud Storage pricing** Consultado a https://cloud.google.com/storage/pricing/

[26] **Elastic Load Balancing** Consultado a https://aws.amazon.com/elasticloadbalancing/

[27] **AWS Lambda** Consultado a https://aws.amazon.com/lambda/

[28] **Escalado de funciones de AWS Lambda** Consultado a https://docs.aws.amazon.com/es_es/lambda/latest/dg/invocation-scaling.html

[29] **AWS Pricing** Consultado a https://aws.amazon.com/pricing/

[30] **Amazon API Gateway** Consultado a https://aws.amazon.com/api-gateway/

[31] **Javascript.com•** Consultado a https://www.javascript.com/about

[32] **What is JavaScript?** Consultado a https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript

[33] **JavaScript.** Consultado a https://en.wikipedia.org/wiki/JavaScript

[34] **Node.js** Consultado a https://en.wikipedia.org/wiki/Node.js

[35] **What is Vue.js?** Consultado a https://vuejs.org/v2/guide/

[36] textbfPython Consultado a https://wiki.python.org/moin/BeginnersGuide/Overview

[37] **Python** Consultado a https://es.wikipedia.org/wiki/Python

[38] **Boto 3 Documentation** Consultado a https://boto3.amazonaws.com/v1/documentation/api/latest/index.html

[39] Juan María Fiz **Serverless: ha llegado el momento de reemplazar servidores por código** Consultado a https://www.paradigmadigital.com/techbiz/serverless-ha-llegado-el-momento-de-reemplazar-servidores-por-codigo/

[40] **¿Que es Amazon Cognito?** Consulta a https://docs.aws.amazon.com/cognito/latest/developerguide/what-is-amazon-cognito.html

[41] **Amazon Cognito User Pools** Consulta a https://docs.aws.amazon.com/cognito/latest/developerguide/cognito-user-identity-pools.html

[42] **Manage IAM users** Consulta a https://aws.amazon.com/iam/features/manage-users/

[43] **Apache OpenWhisk** Consulta a https://developer.ibm.com/technologies/microservices/projects/openwhisk/

[44] Jesús Tramullas. **Elaboración de productos de información con JAMstack: del sistema de gestión de contenidos al web estático.** *Anuario ThinkEPI, v. 14, e14f05.* Consulta a https://recyt.fecyt.es/index.php/ThinkEPI/article/view/thinkepi.2020.e14f05

[45] **AWS CloudFormation** Consulta a https://aws.amazon.com/cloudformation/

[46] Naranjo, Diana M., José R. Prieto, Germán Moltó, and Amanda Calatrava. 2019. **A Visual Dashboard to Track Learning Analytics for Educational Cloud Computing.** Sensors 19(13): 2952. https://www.mdpi.com/1424-8220/19/13/2952/htm

[47] **CloudTrail-Tracker-UI** Consulta a https://github.com/grycap/cloudtrail-tracker-ui

[48] **Instance lifecycle** Consulta a https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-instance-lifecycle.html

Anexo I: Definición de la API

En este apartado se muestra la definición de la API de Oteador, la cual se acceder mediante la url "api.cursocloudaws.net/oteador" y el path concreto de la función a ejecutar.


```
{
  "swagger": "2.0",
  "info": {
    "version": "2020-06-19T14:17:01Z",
    "title": "dev-oteador"
  },
  "host": "api.cursocloudaws.net",
  "basePath": "/oteador",
  "schemes": [
    "https"
  ],
  "paths": {
    "/services/region/{region}": {
      "get": {
        "consumes": [
          "application/json",
          "application/x-www-form-urlencoded"
        ],
        "parameters": [
          {
            "name": "count",
            "in": "query",
            "required": false,
            "type": "string"
          }
        ],
        "responses": {
          "200": {
            "description": "200 response",
            "headers": {
              "Access-Control-Allow-Origin": {
                "type": "string"
              }
            }
          },
          "400": {
            "description": "400 response",
            "headers": {
              "Access-Control-Allow-Origin": {
                "type": "string"
              }
            }
```

```
            },
            "401": {
              "description": "401 response",
              "headers": {
                "Access-Control-Allow-Origin": {
                  "type": "string"
                }
              }
            },
            "403": {
              "description": "403 response",
              "headers": {
                "Access-Control-Allow-Origin": {
                  "type": "string"
                }
              }
            },
            "404": {
              "description": "404 response",
              "headers": {
                "Access-Control-Allow-Origin": {
                  "type": "string"
                }
              }
            },
            "422": {
              "description": "422 response",
              "headers": {
                "Access-Control-Allow-Origin": {
                  "type": "string"
                }
              }
            },
            "500": {
              "description": "500 response",
              "headers": {
                "Access-Control-Allow-Origin": {
                  "type": "string"
                }
              }
            },
            "502": {
              "description": "502 response",
              "headers": {
                "Access-Control-Allow-Origin": {
                  "type": "string"
                }
              }
            },
            "504": {
              "description": "504 response",
              "headers": {
                "Access-Control-Allow-Origin": {
                  "type": "string"
                }
              }
            }
          },
          "x-amazon-apigateway-integration": {
```

```
102             "uri": "arn:aws:apigateway:us-east-1:lambda:path/2015-03-
                    31/functions/arn:aws:lambda:us-east-1:974349055189:
                    function:oteador-dev-query/invocations",
103             "responses": {
104               "[\\s\\S]*\\[404\\][\\s\\S]*": {
105                 "statusCode": "404",
106                 "responseParameters": {
107                   "method.response.header.Access-Control-Allow-Origin
                          ": "'*'"
108                 }
109               },
110               "[\\s\\S]*\\[502\\][\\s\\S]*": {
111                 "statusCode": "502",
112                 "responseParameters": {
113                   "method.response.header.Access-Control-Allow-Origin
                          ": "'*'"
114                 }
115               },
116               "default": {
117                 "statusCode": "200",
118                 "responseParameters": {
119                   "method.response.header.Access-Control-Allow-Origin
                          ": "'*'"
120                 }
121               },
122               "[\\s\\S]*\\[401\\][\\s\\S]*": {
123                 "statusCode": "401",
124                 "responseParameters": {
125                   "method.response.header.Access-Control-Allow-Origin
                          ": "'*'"
126                 }
127               },
128               "[\\s\\S]*\\[403\\][\\s\\S]*": {
129                 "statusCode": "403",
130                 "responseParameters": {
131                   "method.response.header.Access-Control-Allow-Origin
                          ": "'*'"
132                 }
133               },
134               "[\\s\\S]*\\[400\\][\\s\\S]*": {
135                 "statusCode": "400",
136                 "responseParameters": {
137                   "method.response.header.Access-Control-Allow-Origin
                          ": "'*'"
138                 }
139               },
140               "([\\s\\S]*\\[504\\][\\s\\S]*)|(.*Task timed out after
                      \\d+\\.\\d+ seconds$)": {
141                 "statusCode": "504",
142                 "responseParameters": {
143                   "method.response.header.Access-Control-Allow-Origin
                          ": "'*'"
144                 }
145               },
146               "[\\s\\S]*(Process\\s?exited\\s?before\\s?completing\\s
                      ?request|\\[500\\])[\\s\\S]*": {
147                 "statusCode": "500",
148                 "responseParameters": {
```

```
149                 "method.response.header.Access-Control-Allow-Origin
                        ": "'*'"
150               }
151             },
152             "[\\s\\S]*\\[422\\][\\s\\S]*": {
153               "statusCode": "422",
154               "responseParameters": {
155                 "method.response.header.Access-Control-Allow-Origin
                        ": "'*'"
156               }
157             }
158           },
159           "requestParameters": {
160             "integration.request.querystring.count": "method.
                    request.querystring.count"
161           },
162           "requestTemplates": {
163             "application/json": "{\"region\":\"$input.params('
                    region')\"}",
164             "application/x-www-form-urlencoded": "\n    #define(
                    $body )\n      {\n      #foreach( $token in $input.
                    path('$').split('&') )\n        #set( $keyVal =
                    $token.split('=') )\n        #set( $keyValSize =
                    $keyVal.size() )\n        #if( $keyValSize >= 1 )\n
                              #set( $key = $util.escapeJavaScript($util.
                    urlDecode($keyVal[0])) )\n          #if( $keyValSize
                     >= 2 )\n            #set($val = $util.
                    escapeJavaScript($util.urlDecode($keyVal[1])).
                    replaceAll(\"\\\\'\",\"'\"))\n          #else\n
                              #set( $val = '' )\n          #end\n
                              \"$key\": \"$val\"#if($foreach.hasNext),#
                    end\n        #end\n      #end\n      }\n    #end\n\n
                        \n  #define( $loop )\n    {\n    #foreach($key
                    in $map.keySet())\n        #set( $k = $util.
                    escapeJavaScript($key) )\n        #set( $v = $util.
                    escapeJavaScript($map.get($key)).replaceAll
                    (\"\\\\'\", \"'\") )\n        \"$k\":\n          \"
                    $v\"\n          #if( $foreach.hasNext ) , #end\n
                    #end\n    }\n  #end\n\n  {\n    \"body\": $body,\n
                       \"method\": \"$context.httpMethod\",\n    \"
                    principalId\": \"$context.authorizer.principalId\",\
                    n    \"stage\": \"$context.stage\",\n\n    \"
                    cognitoPoolClaims\" : {\n        \n        \"sub\": \"
                    $context.authorizer.claims.sub\"\n    },\n\n    #set
                    ( $map = $context.authorizer )\n    \"
                    enhancedAuthContext\": $loop,\n\n    #set( $map =
                    $input.params().header )\n    \"headers\": $loop,\n\
                    n    #set( $map = $input.params().querystring )\n
                       \"query\": $loop,\n\n    #set( $map = $input.
                    params().path )\n    \"path\": $loop,\n\n    #set(
                    $map = $context.identity )\n    \"identity\": $loop,
                    \n\n    #set( $map = $stageVariables )\n    \"
                    stageVariables\": $loop,\n\n    \"requestPath\": \"
                    $context.resourcePath\"\n  }\n\n  "
165           },
166           "passthroughBehavior": "never",
167           "httpMethod": "POST",
168           "type": "aws"
169         }
```

```
    },
    "options": {
      "consumes": [
        "application/json"
      ],
      "produces": [
        "application/json"
      ],
      "responses": {
        "200": {
          "description": "200 response",
          "headers": {
            "Access-Control-Allow-Origin": {
              "type": "string"
            },
            "Access-Control-Allow-Methods": {
              "type": "string"
            },
            "Access-Control-Allow-Headers": {
              "type": "string"
            }
          }
        }
      },
      "x-amazon-apigateway-integration": {
        "responses": {
          "default": {
            "statusCode": "200",
            "responseParameters": {
              "method.response.header.Access-Control-Allow-
                 Methods": "'OPTIONS,GET'",
              "method.response.header.Access-Control-Allow-
                 Headers": "'Content-Type,X-Amz-Date,
                 Authorization,X-Api-Key,X-Amz-Security-Token,X-
                 Amz-User-Agent'",
              "method.response.header.Access-Control-Allow-Origin
                 ": "'*'"
            },
            "responseTemplates": {
              "application/json": "#set($origin = $input.params
                 (\"Origin\"))\n#if($origin == \"\") #set($origin
                  = $input.params(\"origin\")) #end\n#if($origin.
                 matches(\".+\")) #set($context.responseOverride.
                 header.Access-Control-Allow-Origin = $origin) #
                 end"
            }
          }
        },
        "requestTemplates": {
          "application/json": "{statusCode:200}"
        },
        "passthroughBehavior": "when_no_match",
        "contentHandling": "CONVERT_TO_TEXT",
        "type": "mock"
      }
    }
  },
  "/services/{service}/region/{region}": {
    "get": {
```

```
        "consumes": [
          "application/json",
          "application/x-www-form-urlencoded"
        ],
        "parameters": [
          {
            "name": "count",
            "in": "query",
            "required": false,
            "type": "string"
          }
        ],
        "responses": {
          "200": {
            "description": "200 response",
            "headers": {
              "Access-Control-Allow-Origin": {
                "type": "string"
              }
            }
          },
          "400": {
            "description": "400 response",
            "headers": {
              "Access-Control-Allow-Origin": {
                "type": "string"
              }
            }
          },
          "401": {
            "description": "401 response",
            "headers": {
              "Access-Control-Allow-Origin": {
                "type": "string"
              }
            }
          },
          "403": {
            "description": "403 response",
            "headers": {
              "Access-Control-Allow-Origin": {
                "type": "string"
              }
            }
          },
          "404": {
            "description": "404 response",
            "headers": {
              "Access-Control-Allow-Origin": {
                "type": "string"
              }
            }
          },
          "422": {
            "description": "422 response",
            "headers": {
              "Access-Control-Allow-Origin": {
                "type": "string"
              }
```

```
278             }
279           },
280           "500": {
281             "description": "500 response",
282             "headers": {
283               "Access-Control-Allow-Origin": {
284                 "type": "string"
285               }
286             }
287           },
288           "502": {
289             "description": "502 response",
290             "headers": {
291               "Access-Control-Allow-Origin": {
292                 "type": "string"
293               }
294             }
295           },
296           "504": {
297             "description": "504 response",
298             "headers": {
299               "Access-Control-Allow-Origin": {
300                 "type": "string"
301               }
302             }
303           }
304         },
305         "x-amazon-apigateway-integration": {
306           "uri": "arn:aws:apigateway:us-east-1:lambda:path/2015-03-
                31/functions/arn:aws:lambda:us-east-1:974349055189:
                function:oteador-dev-query/invocations",
307           "responses": {
308             "[\\s\\S]*\\[404\\][\\s\\S]*": {
309               "statusCode": "404",
310               "responseParameters": {
311                 "method.response.header.Access-Control-Allow-Origin
                      ": "'*'"
312               }
313             },
314             "[\\s\\S]*\\[502\\][\\s\\S]*": {
315               "statusCode": "502",
316               "responseParameters": {
317                 "method.response.header.Access-Control-Allow-Origin
                      ": "'*'"
318               }
319             },
320             "default": {
321               "statusCode": "200",
322               "responseParameters": {
323                 "method.response.header.Access-Control-Allow-Origin
                      ": "'*'"
324               }
325             },
326             "[\\s\\S]*\\[401\\][\\s\\S]*": {
327               "statusCode": "401",
328               "responseParameters": {
329                 "method.response.header.Access-Control-Allow-Origin
                      ": "'*'"
330               }
```

```
            },
            "[\\s\\S]*\\[403\\][\\s\\S]*": {
              "statusCode": "403",
              "responseParameters": {
                "method.response.header.Access-Control-Allow-Origin
                    ": "'*'"
              }
            },
            "[\\s\\S]*\\[400\\][\\s\\S]*": {
              "statusCode": "400",
              "responseParameters": {
                "method.response.header.Access-Control-Allow-Origin
                    ": "'*'"
              }
            },
            "([\\s\\S]*\\[504\\][\\s\\S]*)|(.*Task timed out after
                \\d+\\.\\d+ seconds$)": {
              "statusCode": "504",
              "responseParameters": {
                "method.response.header.Access-Control-Allow-Origin
                    ": "'*'"
              }
            },
            "[\\s\\S]*(Process\\s?exited\\s?before\\s?completing\\s
                ?request|\\[500\\])[\\s\\S]*": {
              "statusCode": "500",
              "responseParameters": {
                "method.response.header.Access-Control-Allow-Origin
                    ": "'*'"
              }
            },
            "[\\s\\S]*\\[422\\][\\s\\S]*": {
              "statusCode": "422",
              "responseParameters": {
                "method.response.header.Access-Control-Allow-Origin
                    ": "'*'"
              }
            }
          },
          "requestParameters": {
            "integration.request.querystring.count": "method.
                request.querystring.count"
          },
          "requestTemplates": {
            "application/json": "{\"service\" : \"$input.params('
                service')\", \"region\":\"$input.params('region')\"}
                ",
            "application/x-www-form-urlencoded": "\n    #define(
                $body )\n      {\n      #foreach( $token in $input.
                path('$').split('&') )\n        #set( $keyVal =
                $token.split('=') )\n        #set( $keyValSize =
                $keyVal.size() )\n        #if( $keyValSize >= 1 )\n
                          #set( $key = $util.escapeJavaScript($util.
                urlDecode($keyVal[0])) )\n          #if( $keyValSize
                 >= 2 )\n            #set($val = $util.
                escapeJavaScript($util.urlDecode($keyVal[1])).
                replaceAll(\"\\\\'\",\"'\"))\n          #else\n
                            #set( $val = '' )\n          #end\n
                          \"$key\": \"$val\"#if($foreach.hasNext),#
```

```
                end\n        #end\n      #end\n      }\n    #end\n\n
                    \n  #define( $loop )\n    {\n    #foreach($key
                in $map.keySet())\n        #set( $k = $util.
                escapeJavaScript($key) )\n        #set( $v = $util.
                escapeJavaScript($map.get($key)).replaceAll
                (\"\\\\’\", \"’\") )\n        \"$k\":\n          \"
                $v\"\n          #if( $foreach.hasNext ) , #end\n
                #end\n    }\n  #end\n\n  {\n    \"body\": $body,\n
                   \"method\": \"$context.httpMethod\",\n    \"
                principalId\": \"$context.authorizer.principalId\",\
                n    \"stage\": \"$context.stage\",\n\n    \"
                cognitoPoolClaims\" : {\n       \n       \"sub\": \"
                $context.authorizer.claims.sub\"\n    },\n\n    #set
                ( $map = $context.authorizer )\n    \"
                enhancedAuthContext\": $loop,\n\n    #set( $map =
                $input.params().header )\n    \"headers\": $loop,\n\
                n    #set( $map = $input.params().querystring )\n
                   \"query\": $loop,\n\n    #set( $map = $input.
                params().path )\n    \"path\": $loop,\n\n    #set(
                $map = $context.identity )\n    \"identity\": $loop,
                \n\n    #set( $map = $stageVariables )\n    \"
                stageVariables\": $loop,\n\n    \"requestPath\": \"
                $context.resourcePath\"\n  }\n\n  "
      },
      "passthroughBehavior": "never",
      "httpMethod": "POST",
      "type": "aws"
    }
  },
  "options": {
    "consumes": [
      "application/json"
    ],
    "produces": [
      "application/json"
    ],
    "responses": {
      "200": {
        "description": "200 response",
        "headers": {
          "Access-Control-Allow-Origin": {
            "type": "string"
          },
          "Access-Control-Allow-Methods": {
            "type": "string"
          },
          "Access-Control-Allow-Headers": {
            "type": "string"
          }
        }
      }
    },
    "x-amazon-apigateway-integration": {
      "responses": {
        "default": {
          "statusCode": "200",
          "responseParameters": {
            "method.response.header.Access-Control-Allow-
               Methods": "’OPTIONS,GET’",
```

```
404                 "method.response.header.Access-Control-Allow-
                        Headers": "'Content-Type,X-Amz-Date,
                        Authorization,X-Api-Key,X-Amz-Security-Token,X-
                        Amz-User-Agent'",
405                 "method.response.header.Access-Control-Allow-Origin
                        ": "'*'"
406               },
407               "responseTemplates": {
408                 "application/json": "#set($origin = $input.params
                        (\"Origin\"))\n#if($origin == \"\") #set($origin
                         = $input.params(\"origin\")) #end\n#if($origin.
                        matches(\".+\")) #set($context.responseOverride.
                        header.Access-Control-Allow-Origin = $origin) #
                        end"
409               }
410             }
411           },
412           "requestTemplates": {
413             "application/json": "{statusCode:200}"
414           },
415           "passthroughBehavior": "when_no_match",
416           "contentHandling": "CONVERT_TO_TEXT",
417           "type": "mock"
418         }
419       }
420     },
421     "/services/{service}/region/{region}/state/{state}": {
422       "get": {
423         "consumes": [
424           "application/json",
425           "application/x-www-form-urlencoded"
426         ],
427         "parameters": [
428           {
429             "name": "count",
430             "in": "query",
431             "required": false,
432             "type": "string"
433           }
434         ],
435         "responses": {
436           "200": {
437             "description": "200 response",
438             "headers": {
439               "Access-Control-Allow-Origin": {
440                 "type": "string"
441               }
442             }
443           },
444           "400": {
445             "description": "400 response",
446             "headers": {
447               "Access-Control-Allow-Origin": {
448                 "type": "string"
449               }
450             }
451           },
452           "401": {
453             "description": "401 response",
```

```
            "headers": {
              "Access-Control-Allow-Origin": {
                "type": "string"
              }
            }
          },
          "403": {
            "description": "403 response",
            "headers": {
              "Access-Control-Allow-Origin": {
                "type": "string"
              }
            }
          },
          "404": {
            "description": "404 response",
            "headers": {
              "Access-Control-Allow-Origin": {
                "type": "string"
              }
            }
          },
          "422": {
            "description": "422 response",
            "headers": {
              "Access-Control-Allow-Origin": {
                "type": "string"
              }
            }
          },
          "500": {
            "description": "500 response",
            "headers": {
              "Access-Control-Allow-Origin": {
                "type": "string"
              }
            }
          },
          "502": {
            "description": "502 response",
            "headers": {
              "Access-Control-Allow-Origin": {
                "type": "string"
              }
            }
          },
          "504": {
            "description": "504 response",
            "headers": {
              "Access-Control-Allow-Origin": {
                "type": "string"
              }
            }
          }
        },
        "x-amazon-apigateway-integration": {
          "uri": "arn:aws:apigateway:us-east-1:lambda:path/2015-03-
              31/functions/arn:aws:lambda:us-east-1:974349055189:
              function:oteador-dev-query/invocations",
```

```
511             "responses": {
512               "[\\s\\S]*\\[404\\][\\s\\S]*": {
513                 "statusCode": "404",
514                 "responseParameters": {
515                   "method.response.header.Access-Control-Allow-Origin
                          ": "'*'"
516                 }
517               },
518               "[\\s\\S]*\\[502\\][\\s\\S]*": {
519                 "statusCode": "502",
520                 "responseParameters": {
521                   "method.response.header.Access-Control-Allow-Origin
                          ": "'*'"
522                 }
523               },
524               "default": {
525                 "statusCode": "200",
526                 "responseParameters": {
527                   "method.response.header.Access-Control-Allow-Origin
                          ": "'*'"
528                 }
529               },
530               "[\\s\\S]*\\[401\\][\\s\\S]*": {
531                 "statusCode": "401",
532                 "responseParameters": {
533                   "method.response.header.Access-Control-Allow-Origin
                          ": "'*'"
534                 }
535               },
536               "[\\s\\S]*\\[403\\][\\s\\S]*": {
537                 "statusCode": "403",
538                 "responseParameters": {
539                   "method.response.header.Access-Control-Allow-Origin
                          ": "'*'"
540                 }
541               },
542               "[\\s\\S]*\\[400\\][\\s\\S]*": {
543                 "statusCode": "400",
544                 "responseParameters": {
545                   "method.response.header.Access-Control-Allow-Origin
                          ": "'*'"
546                 }
547               },
548               "([\\s\\S]*\\[504\\][\\s\\S]*)|(.*Task timed out after
                     \\d+\\.\\d+ seconds$)": {
549                 "statusCode": "504",
550                 "responseParameters": {
551                   "method.response.header.Access-Control-Allow-Origin
                          ": "'*'"
552                 }
553               },
554               "[\\s\\S]*(Process\\s?exited\\s?before\\s?completing\\s
                     ?request|\\[500\\])[\\s\\S]*": {
555                 "statusCode": "500",
556                 "responseParameters": {
557                   "method.response.header.Access-Control-Allow-Origin
                          ": "'*'"
558                 }
559               },
```

```
560             "[\\s\\S]*\\[422\\][\\s\\S]*": {
561               "statusCode": "422",
562               "responseParameters": {
563                 "method.response.header.Access-Control-Allow-Origin
                      ": "'*'"
564               }
565             }
566           },
567           "requestParameters": {
568             "integration.request.querystring.count": "method.
                  request.querystring.count"
569           },
570           "requestTemplates": {
571             "application/json": "{\"service\" : \"$input.params('
                  service')\", \"region\":\"$input.params('region')\",
                   \"state\":\"$input.params('state')\"}",
572             "application/x-www-form-urlencoded": "\n    #define(
                  $body )\n      {\n      #foreach( $token in $input.
                  path('$').split('&') )\n        #set( $keyVal =
                  $token.split('=') )\n        #set( $keyValSize =
                  $keyVal.size() )\n        #if( $keyValSize >= 1 )\n
                           #set( $key = $util.escapeJavaScript($util.
                  urlDecode($keyVal[0])) )\n          #if( $keyValSize
                   >= 2 )\n            #set($val = $util.
                  escapeJavaScript($util.urlDecode($keyVal[1])).
                  replaceAll(\"\\\\'\",\"'\"))\n          #else\n
                            #set( $val = '' )\n          #end\n
                           \"$key\": \"$val\"#if($foreach.hasNext),#
                  end\n        #end\n      #end\n      }\n    #end\n\n
                      \n  #define( $loop )\n    {\n    #foreach($key
                  in $map.keySet())\n        #set( $k = $util.
                  escapeJavaScript($key) )\n        #set( $v = $util.
                  escapeJavaScript($map.get($key)).replaceAll
                  (\"\\\\'\", \"'\") )\n        \"$k\":\n          \"
                  $v\"\n          #if( $foreach.hasNext ) , #end\n
                  #end\n    }\n  #end\n\n  {\n    \"body\": $body,\n
                     \"method\": \"$context.httpMethod\",\n    \"
                  principalId\": \"$context.authorizer.principalId\",\
                  n    \"stage\": \"$context.stage\",\n\n    \"
                  cognitoPoolClaims\" : {\n       \n       \"sub\": \"
                  $context.authorizer.claims.sub\"\n    },\n\n    #set
                  ( $map = $context.authorizer )\n    \"
                  enhancedAuthContext\": $loop,\n\n    #set( $map =
                  $input.params().header )\n    \"headers\": $loop,\n\
                  n    #set( $map = $input.params().querystring )\n
                     \"query\": $loop,\n\n    #set( $map = $input.
                  params().path )\n    \"path\": $loop,\n\n    #set(
                  $map = $context.identity )\n    \"identity\": $loop,
                  \n\n    #set( $map = $stageVariables )\n    \"
                  stageVariables\": $loop,\n\n    \"requestPath\": \"
                  $context.resourcePath\"\n  }\n\n  "
573           },
574           "passthroughBehavior": "never",
575           "httpMethod": "POST",
576           "type": "aws"
577         }
578       },
579       "options": {
580         "consumes": [
```

```
            "application/json"
          ],
          "produces": [
            "application/json"
          ],
          "responses": {
            "200": {
              "description": "200 response",
              "headers": {
                "Access-Control-Allow-Origin": {
                  "type": "string"
                },
                "Access-Control-Allow-Methods": {
                  "type": "string"
                },
                "Access-Control-Allow-Headers": {
                  "type": "string"
                }
              }
            }
          },
          "x-amazon-apigateway-integration": {
            "responses": {
              "default": {
                "statusCode": "200",
                "responseParameters": {
                  "method.response.header.Access-Control-Allow-
                     Methods": "'OPTIONS,GET'",
                  "method.response.header.Access-Control-Allow-
                     Headers": "'Content-Type,X-Amz-Date,
                     Authorization,X-Api-Key,X-Amz-Security-Token,X-
                     Amz-User-Agent'",
                  "method.response.header.Access-Control-Allow-Origin
                     ": "'*'"
                },
                "responseTemplates": {
                  "application/json": "#set($origin = $input.params
                     (\"Origin\"))\n#if($origin == \"\") #set($origin
                      = $input.params(\"origin\")) #end\n#if($origin.
                     matches(\".+\")) #set($context.responseOverride.
                     header.Access-Control-Allow-Origin = $origin) #
                     end"
                }
              }
            },
            "requestTemplates": {
              "application/json": "{statusCode:200}"
            },
            "passthroughBehavior": "when_no_match",
            "contentHandling": "CONVERT_TO_TEXT",
            "type": "mock"
          }
        }
      }
    }
}
```

Listing 1: Definición de la API con Swagger