UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

# Portal Web de Analíticas de Uso para Cuentas Compartidas en AWS

Trabajo Fin de Grado

**Grado en Ingeniería Informática**

**Autor**: Jose Ramón Prieto Fontcuberta

**Tutor**: Germán Moltó Martínez

Curso 2017/2018

# Agradecimientos

A mis padres y hermana, ya que sin ellos no podría haber seguido estudiando.

A Germán, por la dedicación e interés en este proyecto.

A mis dos profesores de Grado Superior, Joan y Kico, ya que me motivaron a empezar el Grado.

# Resumen

La adopción de proveedores de Cloud público como Amazon Web Services (AWS), tanto en ámbitos académicos como empresariales, requiere, por lo general, el uso compartido de cuentas de usuario entre diferentes miembros de la institución. Es importante conocer el grado de utilización que los usuarios realizan de la plataforma, especialmente cuando se integra una plataforma Cloud en el ámbito educativo, donde es necesario conocer el grado de uso que los alumnos han hecho de los diferentes servicios de AWS.

Este Trabajo de Fin de Grado se ha centrado en el desarrollo de una aplicación distribuida, denominada CloudTrail-Tracker, para la recopilación, procesado y análisis de eventos que permite obtener la actividad de una cuenta de usuario de AWS, permitiendo conocer las actividades que han generado cambios en la infraestructura, mostrando por usuarios, servicios o añadiendo filtros más personalizados.

CloudTrail-Tracker se ha diseñado como una aplicación serverless de manera que se ejecuta completamente usando servicios de AWS y computación dirigida por eventos de manera que no es necesaria la gestión explícita de servidores. La aplicación ha sido liberada a la comunidad mediante una licencia de código abierto y está siendo utilizada actualmente en producción para soportar el análisis de las actividades de los alumnos en asignaturas de Cloud Computing de tres másteres de la Universitat Politècnica de València.

Se ha realizado un análisis de rendimiento y de coste de la herramienta, determinando que puede ser una solución efectiva al análisis del uso de cuentas de AWS tanto en términos de coste económico como en aplicabilidad a múltiples escenarios, no únicamente educativos.

**Palabras clave:** computación en la nube, serverless, AWS, eventos, seguimiento, monitorización



# Abstract

The adoption of public cloud providers such as Amazon Web Services (AWS), both in academic and business environments, generally requires account sharing among different members of the institution. It is important to gather usage analytics from the users of the platform, especially when integrating the Cloud in educational activities, where it is important to know the usage level of the students concerning the different AWS services.

This Bachelor's Degree Final Project has focused on the development of a distributed application, called CloudTrail-Tracker, for the collection, processing and analysis of events that allows to obtain the activity of an AWS user account, allowing one to know the activities that have generated infrastructure changes, aggregated by users, services or adding more personalized filters.

CloudTrail-Tracker has been designed as a serverless application running completely on AWS services and using event-driven computing so that explicit server management is not necessary. The application has been released to the community as an open source project and is currently being used in production to support the analysis of student activities in Cloud Computing subjects of three master's degrees from the Universitat Politècnica de València.

A performance and cost analysis of the tool has been carried out, determining that it can be an effective solution to the analysis of the use of AWS accounts both in terms of economic cost and its applicability to multiple scenarios and not only educational ones.

**Keywords:** Cloud Computing, serverless, AWS, events, tracking

# Resum

L'adopció dels proveïdors de Cloud públic com Amazon Web Services (AWS), tant en àmbits acadèmics com empresarials, requereix, per norma general, l'ús compartit de comptes d'usuari entre diferents membres de la institució. És important conèixer el grau d'utilització que els usuaris realitzen de la plataforma, especialment quan s'integra una plataforma Cloud en l'àmbit educatiu, on es necessari saber el grau d'ús que els alumnes han fet sobre els diferents serveis d'AWS.

Aquest Treball de Fi de Grau s'ha centrat en el desenvolupament d'una aplicació distribuïda, anomenada CloudTrail-Tracker, per a la recopìlació, processat i anàlisis d'events que permeteix obtindre l'activitat d'un compte d'usuari d'AWS, permitent conèixer les activitats que han generat canvis en la infraestructura, mostrant per usuarios, serveis o afe33333333gint filtres més personalitzats.

CloudTrail-Tracker s'ha dissenyat com una aplicació de serverless, de forma que s'executa completament emprant serveis d'AWS i computació dirigida per events, de forma que no es necessària la gestió explicita de servidors. L'aplicació ha estat alliberada a la comunitat mitjançant una llicència de codi obert i està essent utilitzada actualment en producció per suportar l'anàlisi de les activitats dels alumnes en assignatures de Cloud Computing de tres màsters de la Universitat Politècnica de València.

S'ha realitzat una anàlisi de rendiment i cost de l'eina, determinant que pot ser una solució efectiva a l'anàlisi de l'ús dels comptes d'AWS tant en terminis de cost econòmic com en aplicabilitat a múltiples escenaris, no únicament educatius.

**Paraules clau:** Informàtica en el núvol, serverless, AWS, events, monitorització

# Tabla de contenidos

# Índice de figuras

# 1. Introducción

"Es un método sencillo de obtener acceso a servidores, almacenamiento, bases de datos y una amplia gama de servicios de aplicaciones a través de Internet. Una plataforma de servicios en la nube, como Amazon Web Services, es propietaria y responsable del mantenimiento del hardware conectado en red necesario para dichos servicios de aplicaciones, mientras que usted se dedica a aprovisionar lo que necesite por medio de una aplicación web." [1]

Así es como define Amazon la informática en la nube.

Las tecnologías en la nube se están utilizando en gran cantidad de servicios hoy en día [2], donde cada vez va en aumento. Podemos ver por ejemplo cómo la cantidad de datos que genera la sociedad está creciendo, y es que todos los servicios y aplicaciones que usamos hoy en día se conectan o residen en la nube, y es debido a que usar servicios en la nube ofrece mayores beneficios que los servicios locales convencionales, como pueden ser: ahorro en los costes, escalabilidad, mayor almacenamiento, acceso en cualquier momento, mayor seguridad, ahorro de energía, etc.

Gracias a la computación en la nube se obtienen las siguientes ventajas:

- Se pasa de gastos fijos a gasto variables, pagando solo por lo que usamos
- Precios más asequibles al no tener que preocuparnos por la electricidad o la seguridad del hardware y gracias a la alta competencia en este sector.
- Permite aumentar o disminuir recursos rápidamente, estableciendo medidas de auto-escalado si fuera necesario, sin complicaciones.
- El usuario se despreocupa de la velocidad y tiempos de respuesta.

Estas ventajas permiten centrarse más en el desarrollo la calidad de la solución, y no tanto en el mantenimiento del hardware necesario.

De acuerdo a la definición del NIST sobre Cloud Computing [45], hay diferentes modelos de servicio relacionados con la computación en la nube. Tenemos infraestructura como servicio (I*nfrastructure as a Service*, IaaS), que permite el alquiler de recursos de cómputo, almacenaje y redes mediante un modelo de pago por uso. En lugar de comprar directamente el hardware, donde es necesario realizar la inversión inicial y pagar el mantenimiento, mediante IaaS se paga por la utilización de los recursos, teniendo una infraestructura fácilmente escalable.

A un nivel un poco más alto existen los servicios de plataforma (*Platform as a Service*, PaaS) dónde se proporciona al cliente un conjunto de herramientas prediseñadas para desarrollar, gestionar y distribuir aplicaciones sus propias aplicaciones. Estos servicios permiten abstraerse de la gestión de seguridad y el software de servidor, así como de las copias de seguridad. Permiten centrarse más en el desarrollo.

Subiendo un nivel más tenemos software como servicio (*Software as a Service*, SaaS). Se ofrecen aplicaciones basadas en Cloud, dónde el usuario no tiene que descargar ninguna aplicación, si no directamente usar una API o a través de la web. Estos servicios facilitan la colaboración online, suelen ofrecer un modelo de suscripción y evitan al usuario tener que gestionar, instalar o actualizar el software.

Por otro lado, también tenemos el modelo de servicio defunción como servicio (*Function as a Service*, FaaS). Este modelo permite ejecutar código encapsulado en función escritas en un determinado lenguaje de programación, sin administrar los servidores donde éste se ejecuta, solo lanzamos la petición, elegimos la potencia que necesitamos, y se paga por el tiempo de ejecución que ha estado en marcha dicha función. Se ha de tener en cuenta que el estado de ejecución se pierde una vez la función ha terminado.

Existen hoy en día numerosos proveedores Cloud que ofrecen múltiples servicios para soportar la computación en la nube. Entre los más conocidos y utilizados están Amazon Web Services (AWS), Microsoft Azure y Google Cloud Platform. Todos ellos ofrecen un ámplio catálogo de servicios que facilitan el desarrollo de arquitecturas de aplicaciones escalables en la nube. Por lo general, un usuario (o una organización) crea una cuenta en un proveedor Cloud y, dentro de ella, puede crear múltiples cuentas de usuario para asignar privilegios específicos y limitados en función de los diferentes roles que desempeñen los usuarios dentro de la organización.

En este sentido, los proveedores Cloud ofrecen servicios de trazabilidad para registrar la actividad realizada por los usuarios. En el caso de AWS, dicho servicio es AWS CloudTrail, el cual permite llevar un historial de eventos detallados de la actividad de la cuenta. Estos eventos almacenan información sobre las acciones que ocurren, ya hayan sido realizadas por usuarios o por los propios servicios, a petición del usuario. Se llegan a almacenar unos 25 atributos por evento, atributos muy útiles como pueden ser: el nombre del usuario que realiza la acción, el origen del evento, la región, dia y hora, y otros que no son de tanta utilidad, como pueden ser: versión del evento, dirección de origen, clave de acceso, varios identificadores, etc. Sin embargo, procesar dicho volumen de datos no resulta trivial, especialmente si los resultados se desean obtener en un tiempo razonable cuando se solicita.

## 1.1 Motivación

La idea de este TFG surge en el ámbito académico y en el contexto de diferentes asignaturas y cursos de formación online relacionados con Cloud Computing impartidos en la Universitat Politècnica de València, donde existe la necesidad de determinar el grado de uso que los alumnos realizan de los múltiples servicios de AWS. Sin embargo, rápidamente se advirtió que esta idea podría evolucionar a una plataforma general para el procesado de registros de uso en AWS, que pudiera utilizarse en ámbitos no exclusivamente educativos.

Dado que no existen herramientas de código libre que nos permitan analizar y realizar un seguimiento del uso de los servicios en AWS, surge la idea de un proyecto como el descrito en



este Trabajo Final de Grado, que se ha denominado CloudTrail-Tracker[1]. Además, las herramientas que actualmente existen, como se describe en el estado del arte, tienen un alto coste y no permiten un control total de los datos, ya que los extraen hacia su propia infraestructura para facilitar el procesado.

En este sentido, barajamos la posibilidad de realizar este proyecto con los servicios que ofrece AWS, implementando una solución completamente serverless, es decir, utilizando servicios FaaS , permitiendo la ejecución de funciones en respuesta a eventos y pagando sólo por el tiempo de ejecución de ésta, abaratando en gran medida los costes.

Dicha herramienta permitiría añadir, posteriormente, funcionalidades como crear seguimientos automáticos sobre usuarios, detección de anomalías o usos indebidos por parte de usuarios o servicios en general, la creación de alarmas y avisos complejos, informes de estadísticas con información agregada, entre otras posibilidades.

### 1.1.1 Motivación personal

Para mi este proyecto supone usar un gran número de tecnologías en la nube, las cuales no se han visto durante el grado, lo que me ha permitido aprender y profundizar sobre ellas como no lo podría haber hecho de otra forma.

La idea de trabajar en un proyecto así surge después realizar el Curso Online de Cloud Computing con Amazon Web Services (AWS) [3], impartido por el área de Grid, Cloud y Computación de Altas Prestaciones (GRyCAP) del Instituto de Instrumentación para Imagen Molecular (I3M), adquiriendo la mayor parte de los conocimientos necesarios para la realización de tal proyecto.

Lo que me hizo elegir este proyecto es coger experiencia en cuanto al uso y despliegue de soluciones en la nube, debido al abanico de posibilidades que abren dichos servicios y la alta demanda en el mundo laboral.

## 1.2 Objetivos

El principal objetivo del proyecto es diseñar, implementar y desplegar en producción una plataforma distribuida que permita recopilar, analizar y consultar las acciones que generen cambios en la infraestructura realizadas por usuarios de una cuenta de AWS derivadas de la utilización de los diferentes servicios.

Como complemento a este objetivo principal, se han marcado una serie de objetivos secundarios:

El primer objetivo consiste en diseñar la arquitectura de la aplicación en la nube usando servicios FaaS, mediante funciones dirigidas por eventos para simplificar el desarrollo, facilitar

1 CloudTrail-Tracker está disponible en https://github.com/grycap/cloudtrail-tracker

el escalado, abaratar los costes y eliminar la necesidad de desplegar, configurar y mantener servidores.

El segundo objetivo es ofrecer un mecanismo de consulta para poder obtener los eventos realizados por un usuario o servicio en una determinada franja de tiempo, soportando además la opción de buscar mediante una granularidad temporal fina, especificando hora, minutos y segundos a la consulta.

El tercer objetivo es ofrecer una forma amigable para la realización de las consultas, creando una API así como su documentación, de acuerdo a las buenas prácticas establecidas en la comunidad, y mostrando ejemplos de uso de ésta.

El cuarto objetivo consiste en optimizar las consultas para disminuir el tiempo de ejecución de las funciones, evitando así incurrir en gastos innecesarios, y evitar que la API responda en un tiempo razonable.

El quinto y último objetivo es liberar el proyecto como contribución a la comunidad con una licencia de código abierto, así como difundir los resultados del mismo en aquellos foros en los que se considere interesante su diseminación.

## 1.3 Impacto esperado

Actualmente, la herramienta CloudTrail-Tracker está siendo utilizada para dar soporte a la obtención de analíticas de aprendizaje de diferentes asignaturas de tres masters diferentes de la Universitat Politècnica de València (Máster Universitario en Computación Paralela y Distribuida, Máster Universitario en Gestión de la Información y Máster en Big Data Analytics) Asimismo, se está utilizando en el Curso Online de Cloud Computing con Amazon Web Services del GRyCAP.

Al ser un proyecto con licencia de código abierto y no haber otro proyecto de similares características se espera una amplia aceptación por parte de la comunidad de AWS. Cualquier usuario de AWS puede ejecutar la herramienta desarrollada en este trabajo en su cuenta de AWS y obtener a coste prácticamente cero analíticas detalladas del uso de los diferentes servicios por parte de los múltiples usuarios que puedan estar compartiendo dicha cuenta.

## 1.4 Estructura del documento

La estructura de este documento se divide en cinco bloques.

En el primer bloque hacemos un breve repaso del estado del arte sobre la computación en la nube, listando los principales proveedores Cloud. También repasamos cuales son las principales alternativas o propuestas similares a la nuestra. Seguidamente realizamos una crítica sobre estas alternativas y exponemos nuestra propuesta.

En el segundo bloque se describen las tecnologías empleadas en el desarrollo del servicio y cuál es la función de cada una de ellas. Identificamos los problemas y analizamos las posibles

soluciones a éstos. Seguidamente exponemos brevemente nuestra solución y finalmente mostramos el presupuesto o material necesario para la realización de este proyecto.

En el tercer bloque se muestra la arquitectura general del sistema y se explican los componentes del back-end y front-end y cómo se comunican entre ellos, donde entramos de forma detallada sobre ciertos aspectos técnicos y de diseño. Seguidamente, para finalizar este segundo bloque explicamos las dependencias necesarias para el sistema y cómo desplegarlo. Acabamos con algunos ejemplos de uso del mismo.

En el cuarto bloque referenciamos algunos casos de uso del servicio y ponemos a prueba el rendimiento y coste del mismo, probando varias consultas, cargando datos y analizando los tiempos.

Finalmente, en el quinto bloque explicamos la difusión del proyecto. Acabamos con las conclusiones, una lista de posibles trabajos futuros y explicando la relación del trabajo desarrollado con los estudios cursados.

# 2. Estado del arte

En este capítulo se describen los principales proveedores Cloud para introducir brevemente, el estado actual de la computación en la nube. A continuación, se centra el foco en algunas de las principales herramientas existentes para la recogida y seguimiento de eventos en AWS.

## 2.1 Principales proveedores de servicios en la nube

A continuación, vamos a analizar los principales proveedores de servicios en la nube.

### 2.1.1 Microsoft Azure

Es uno de los actores relevantes en computación en la nube según Forbes [4] debido a la profunda participación en las tres capas - infraestructura, plataforma y software como servicio (IaaS, PaaS y SaaS). Sus ingresos por los servicios Cloud ofrecidos se estiman en 16,7 mil millones en el año 2017.

En Microsoft Azure [5] se encuentran servicios Cloud de almacenamiento, bases de datos, redes, internet de las cosas y computación, entre otros. Trabajan con un total de 52 regiones, lo que permite acercar las aplicaciones en la nube a usuarios de todo el mundo, de este modo, se mantiene la residencia de los datos y agiliza los recursos. [6]

Como vemos en la figura 1, Microsoft Azure está en segunda posición en la tabla de tendencias en Stack Overflow.

### 2.1.2 Google Cloud

Google Cloud [7] destaca por sus bajos precios en computación y almacenamiento, siendo hasta un 25% más barato que Amazon (2 CPUs/8GB RAM por 52$/mes [47], mientras que AWS 69$/mes [8]), lo que hace que el servicio sea muy competitivo.
Tiene más de 50 servicios disponibles, donde destaca *Google Compute Engine*, servicio que permite lanzar máquinas virtuales a demanda.

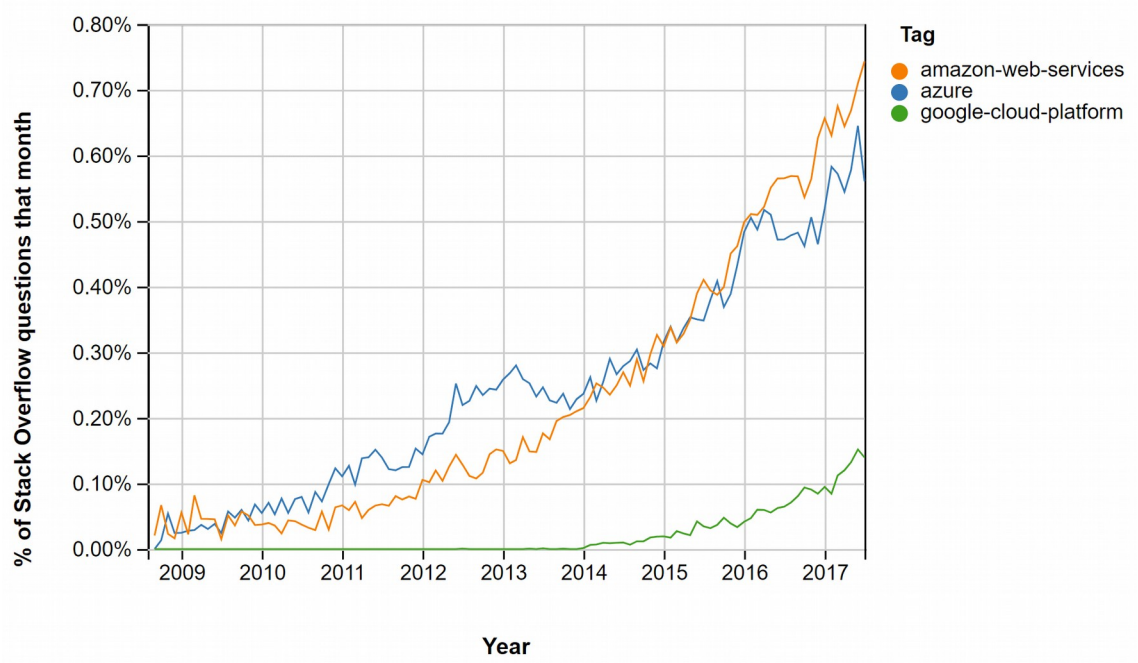



*Figura 1: Las tendencias de la nube estos últimos ocho años, según Stack Overflow [9]*

En la figura 1 podemos ver la tendencia en los últimos ocho años a usar los servicios en la nube de Google, Microsoft,y Amazon. Se observa que Google Cloud ha aumentado considerablemente estos últimos años,

También destaca por la mayor variedad de configuración de las instancias, siendo más flexible que otros servicios, además ofrece 12 meses de uso gratuito y 300$ de crédito para empezar a utilizar el servicio.

### 2.1.3 Amazon Web Services

Amazon Web Services (AWS [11], de ahora en adelante) es el proveedor que estrena más servicios al año, en favor a sus clientes, los cuales son presentados en AWS re:Invent, una conferencia anual.
Ofrece servicios flexibles, elásticos y escalables, requisitos fundamentales en los servicios en la nube, manteniendo una política invariable de disminuir los costos.

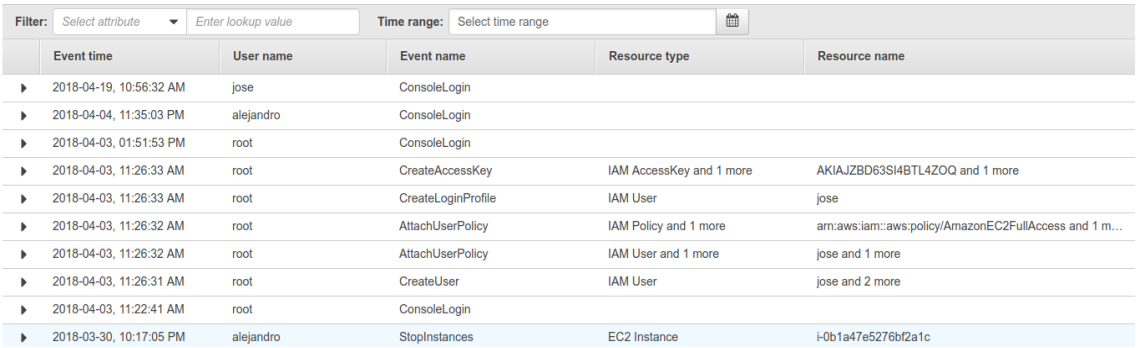
AWS es la plataforma de servicios en la nube que más dinero mueve, según Forbes [10], y que más servicios ofrece. Amazon es una de las empresas más potentes, con más experiencia y pioneras en este sector. Como vemos en la figura 1, es líder en menciones en Stack Overflow, sitio web muy frecuentado por desarrolladores informáticos y Gartner [38] habla de AWS como del líder indiscutible en servicios Cloud
Por estos motivos, hemos seleccionado AWS como plataforma en la que basarnos.

## 2.2 Monitorización de Eventos en la Nube

A continuación, vamos a ver algunos servicios relacionados con el análisis de eventos en AWS. Para ello, en primer lugar, abordaremos CloudTrail, el principal servicio de AWS para la recopilación de eventos.

### 2.2.1 CloudTrail

CloudTrail [12] es un servicio que nos permite recoger las llamadas a la API de AWS y almacenarlas en un fichero dentro de un bucket de S3.
Estos ficheros, estructurados en formato JSON, nos permiten hacer un seguimiento de toda acción que se realice en nuestra cuenta, permitiéndonos realizar un procesamiento y posterior análisis de las acciones, tanto de los usuarios como cambios en la infraestructura.

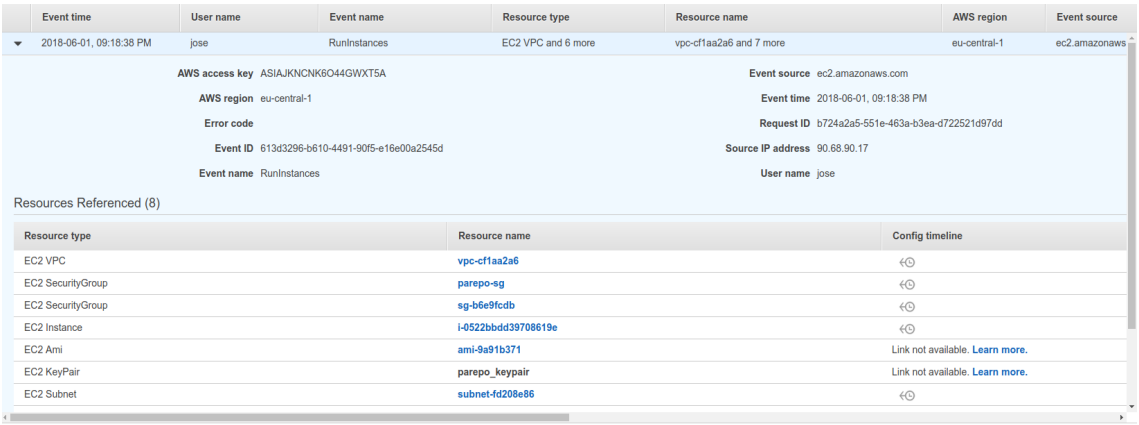


Filter: Select attribute | Enter lookup value | Time range: Select time range

| Event time | User name | Event name | Resource type | Resource name |
|---|---|---|---|---|
| 2018-04-19, 10:56:32 AM | jose | ConsoleLogin | | |
| 2018-04-04, 11:35:03 PM | alejandro | ConsoleLogin | | |
| 2018-04-03, 01:51:53 PM | root | ConsoleLogin | | |
| 2018-04-03, 11:26:33 AM | root | CreateAccessKey | IAM AccessKey and 1 more | AKIAJZBD63SI4BTL4ZOQ and 1 more |
| 2018-04-03, 11:26:33 AM | root | CreateLoginProfile | IAM User | jose |
| 2018-04-03, 11:26:32 AM | root | AttachUserPolicy | IAM Policy and 1 more | arn:aws:iam::aws:policy/AmazonEC2FullAccess and 1 m… |
| 2018-04-03, 11:26:32 AM | root | AttachUserPolicy | IAM User and 1 more | jose and 1 more |
| 2018-04-03, 11:26:31 AM | root | CreateUser | IAM User | jose and 2 more |
| 2018-04-03, 11:22:41 AM | root | ConsoleLogin | | |
| 2018-03-30, 10:17:05 PM | alejandro | StopInstances | EC2 Instance | i-0b1a47e5276bf2a1c |

*Figura 2: Servicio web de CloudTrail a la hora de realizar consultas.*

El servicio también ofrece información a través de una interfaz web de forma más amigable, aunque un poco limitada, como se puede ver en la figura 2, donde podemos entre un rango de fechas. No permite buscar eventos con más de 90 días de antigüedad.



| Event time | User name | Event name | Resource type | Resource name | AWS region | Event source |
|---|---|---|---|---|---|---|
| 2018-06-01, 09:18:38 PM | jose | RunInstances | EC2 VPC and 6 more | vpc-cf1aa2a6 and 7 more | eu-central-1 | ec2.amazonaws |

AWS access key: ASIAJKNCNK6O44GWXT5A
AWS region: eu-central-1
Error code:
Event ID: 613d3296-b610-4491-90f5-e16e00a2545d
Event name: RunInstances
Event source: ec2.amazonaws.com
Event time: 2018-06-01, 09:18:38 PM
Request ID: b724a2a5-551e-463a-b3ea-d722521d97dd
Source IP address: 90.68.90.17
User name: jose

Resources Referenced (8)

| Resource type | Resource name | Config timeline |
|---|---|---|
| EC2 VPC | vpc-cf1aa2a6 | |
| EC2 SecurityGroup | parepo-sg | |
| EC2 SecurityGroup | sg-b6e9fcdb | |
| EC2 Instance | i-0522bbdd39708619e | |
| EC2 Ami | ami-9a91b371 | Link not available. Learn more. |
| EC2 KeyPair | parepo_keypair | Link not available. Learn more. |
| EC2 Subnet | subnet-fd208e86 | |

*Figura 3: Detalles de un evento RunInstances desde CloudTrail Web*

En la figura 3 vemos parte de los detalles de un evento desde la web del servicio. Nos proporciona mucha información útil, como la fecha, la región, qué máquina se ha instanciado, etc. y otra información que en la mayoría de los casos no es tan útil, como el identificador de la petición o la clave de acceso. El evento se puede descargar en formato CSV y JSON para poder procesarlo.

Podemos crear filtros por usuario, identificador, nombre o origen del evento y tipo o nombre del recurso. También podemos hacer un filtrado en un rango de tiempo de no más de noventa días.

Este servicio es la base de donde extraemos toda la información que posteriormente procesamos en nuestra propuesta, la cual planteamos más adelante.

### 2.2.2 Amazon CloudWatch

CloudWatch [13] es el servicio de monitorización de recursos que ofrece AWS, el cual también permite registrar Logs y disparar eventos.

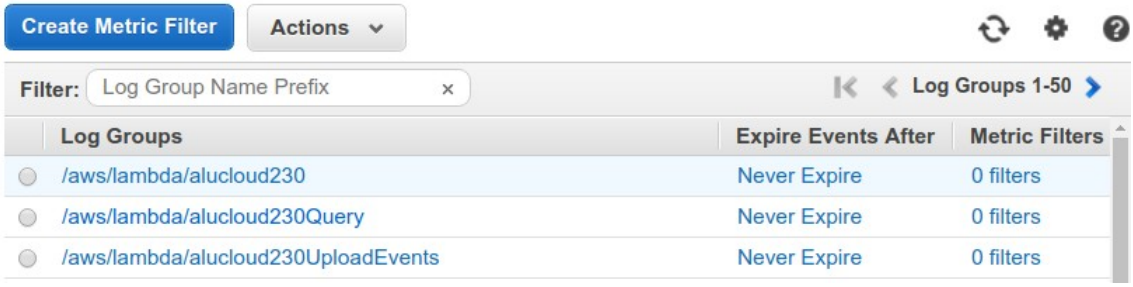



*Figura 4: Grupos de Logs usados en CloudTrail-Tracker, en CloudWatch*

Como vemos en la figura 4, podemos monitorizar algunos recursos concretos.
Nos permite crear grupos sobre diferentes servicios (no todos) de nuestra cuenta de AWS para posteriormente monitorizar los eventos y visualizarlos, como vemos en la siguiente figura.

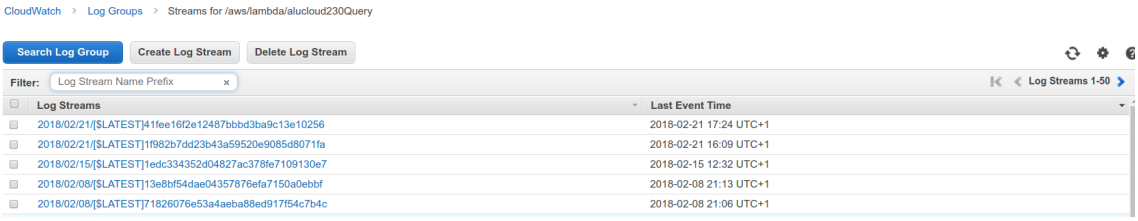



*Figura 5: Ejemplo de eventos monitorizados sobre una función Lambda*

Esta herramienta también es capaz de generar gráficas y métricas sobre la utilización de los diferentes recursos, como pueden ser uso del procesador, consumo de memoria, espacio libre en disco, uso de red y más métricas personalizables, siendo posible programar alarmas o lanzar funciones Lambda con esta información.

Con CloudWatch tenemos ciertos límites. Solo podemos poner 10 alarmas al mes y establecer 10 métricas al mes por cliente de forma gratuita, igual que tenemos 1.000.000 solicitudes a la API de forma gratuita y las métricas duran 15 meses, entre otras limitaciones menores.

Es importante destacar que CloudWatch es un servicio que está pensado para monitorizar el consumo de recursos en AWS, pero no está específicamente diseñado para conocer qué eventos han sucedido en una cuenta de AWS relativos al uso de los diferentes servicios.

A continuación, se muestran diferentes herramientas específicamente diseñadas para el procesado de eventos de AWS y, en particular, que involucren el uso de CloudTrail como fuente de dichos eventos, ya que serán las herramientas más similares a la propuesta de aplicación planteada en este trabajo.

### 2.2.3 Alert Logic Log Manager

Alert Logic [14] es una empresa especializada en crear herramientas de seguridad en la nube.
Su herramienta Log Manager [15] permite guardar eventos en su propia nube sobre diferentes plataformas: AWS, Azure, Office 365, IBM. Después normalizan toda la información y permiten hacer consultas en tiempo real.

Alert Logic Cloud Defender (US)

| Hosts | Description | 1 MONTH | 12 MONTHS | 24 MONTHS | 36 MONTHS |
|---|---|---|---|---|---|
| Tier 1: 25 H | Up to 25 Protected Hosts - 1 Month Term | $2979 | $31080 | $52836 | $74592 |
| Tier 2: 50 H | 26-50 Protected Hosts | $4589 | $47880 | $81396 | $114912 |
| Tier 3: 75 H | 51-75 Protected Hosts | $5739 | $59880 | $101796 | $143712 |
| Tier 4: 100 H | 76-100 Protected Hosts | $7809 | $81480 | $138516 | $195552 |
| Tier 5: 250 H | 101-250 Protected Hosts | $9649 | $100680 | $171156 | $241632 |
| Tier 6: 500 H | 251-500 Protected Hosts | $11834 | $123480 | $209916 | $296352 |
| Tier 7: 750 H | 501-750 Protected Hosts | $13329 | $139080 | $236436 | $333792 |
| Tier 8: 1000 H | 751-1000 Protected Hosts | $16089 | $167880 | $285396 | $402912 |

*Figura 6: Precio según Marketplace de AWS, Log Manager*

Es una opción a tener en cuenta si es necesario cumplir las normativas del Estándar de Seguridad de Datos para la Industria de Tarjeta de Pago (PCI) [16], la Ley de Transferencia y Responsabilidad de los Seguros Médicos (HIPAA) [17] o la Ley Sarbanes Oxley (SOX)[18] entre otras.

Tal y como se ve en la figura 6, los precios por mes para el uso del servicio son de casi 3000$ para 25 máquinas hasta dispararse a 16089$ para 1000 máquinas.

En el contexto de AWS permite monitorizar eventos en tiempo real de S3, EC2 e IAM y requiere tener acceso a los logs de CloudTrail. También ofrece la opción de identificar actividades sospechosas y vulnerabilidades.

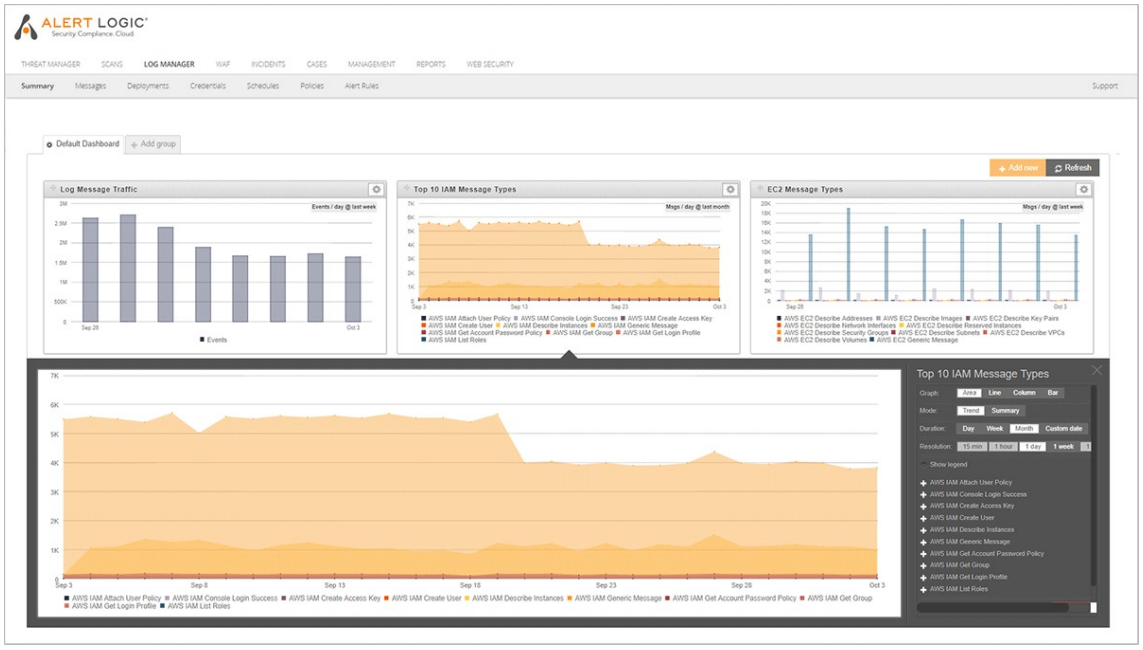



*Figura 7: Panel de visualización de visualización de Log Manager*

En la figura 7 vemos uno de los paneles de visualización en tiempo real que ofrece dicha herramienta. Ofrece paneles para la búsqueda de eventos, para monitorear y analizar en tiempo real y otros paneles para analizar eventos concretos.

### 2.2.4 AlienVault

Es una herramienta que almacena y analiza los eventos de CloudTrail, ofreciendo servicios de administración y monitorización de seguridad. Detecta amenazas automáticamente, también automatiza las respuestas si es necesario, en toda la cuenta de AWS.
Incluye informes predefinidos y personalizables. [19]

AlienVault Unified Security Management (USM)

| Data | Description | 1 MONTH |
|---|---|---|
| 250GB/Month | Monthly Data Volume Tier | $1095 |
| 500GB/Month | Monthly Data Volume Tier | $1495 |
| 1TB/Month | Monthly Data Volume Tier | $2195 |
| 1.5TB/Month | Monthly Data Volume Tier | $3295 |
| 2TB/Month | Monthly Data Volume Tier | $4495 |
| 3TB/Month | Monthly Data Volume Tier | $6395 |
| 4TB/Month | Monthly Data Volume Tier | $8795 |

*Figura 8: Precio según Marketplace de AWS, AlienVault:*

Como se ve en la figura 8, los precios al mes van de 1095$ por 250GB hasta llegar a 8795$, permitiendo 4TB de almacenamiento de datos.

Este servicio, con acceso a los eventos de CloudTrail, permite analizar y monitorizar en tiempo real nuestros servicios en AWS, permitiendo también crear alarmas. Como vemos en la figura 9, frecen paneles de visualización para la una cómoda y rápida visualización de los datos.

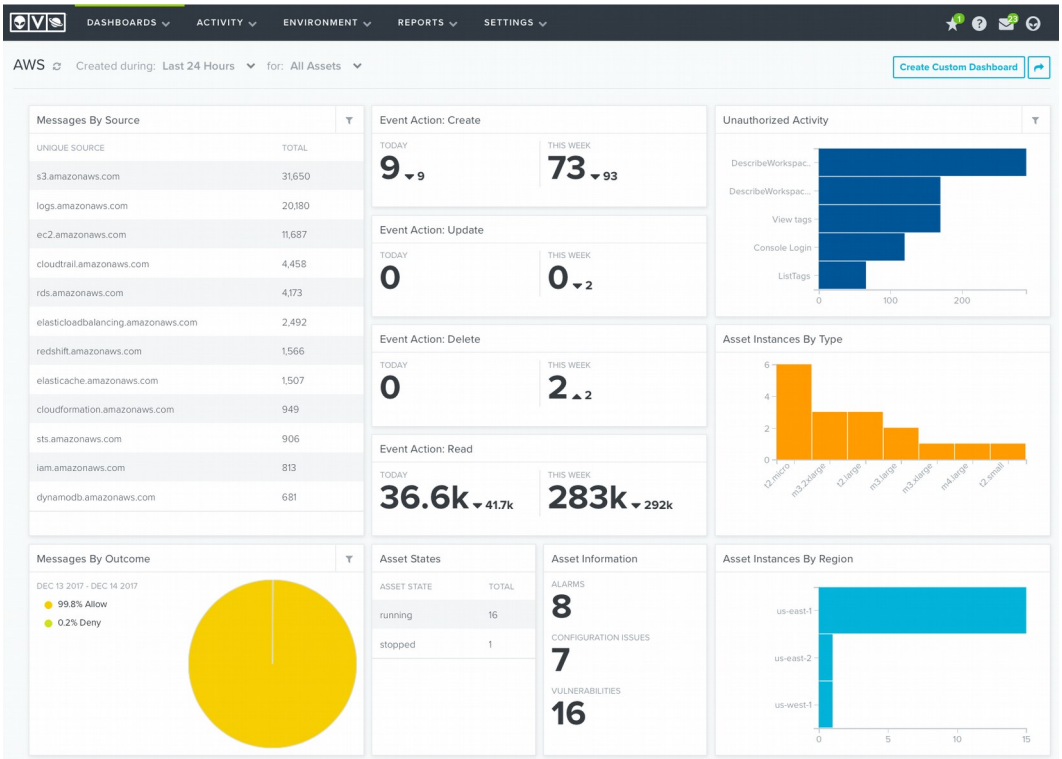



*Figura 9: Visualización de los datos en AlienVault*

La empresa que está detrás del servicio tiene un equipo de investigación sobre amenazas y, gracias a esto, analiza y detecta comportamientos sospechosos en la creación de instancias, creación de nuevos usuarios, elevación de permisos, modificación de grupos de seguridad, y más.

## Boundary

Boundary [20] es una herramienta que interpreta los datos provenientes de CloudTrail para monitorizar los cambios que se producen. Permite conocer en profundidad qué cambios hizo cada usuario y que provoca cada uno de éstos en tiempo real, permitiendo crear también alertas sobre los cambios.

No dispone del precio en el Marketplace de AWS y desde su página web no nos ofrecen ningún precio tampoco.

Este servicio se alimenta a base de los eventos almacenados por CloudTrail y, además, se especializan en monitorear no solo servicios en la nube como AWS, también bases de datos, servidores web, Big Data, y más.
En AWS se integra con 18 servicios, los más destacados son EC2, Billing, DynamoDB, EBS, RDS, AWS Linux, OpsWorks y Auto Scaling.


*Figura 10: Panel de Boudary para EC2*

Nos ofrecen diferentes paneles gráficos para la monitorización. En la figura 10 vemos el panel para EC2, con cantidad de gráficos diferentes.

## 2.3 Crítica al estado del arte

Tal y como hemos mencionado anteriormente, AWS CloudTrail, la solución nativa de Amazon, ofrece un panel de búsqueda, pero tiene ciertas limitaciones, dependiendo de las peticiones que necesites hacer. No permite realizar un análisis de varios servicios de forma simultánea o pedir información sobre un usuario en concreto. Tampoco permite añadir granularidad sobre las consultas y, además, dichas consultas están limitadas a 90 días.

El resto de herramientas mostradas, aunque potentes, no son de código abierto y tienen un coste elevado. Tales herramientas extraen la información de nuestra cuenta de AWS y la almacenan en una base de datos externa, por lo que no tenemos el control total de lo que hacen dichas herramientas con nuestra información. Tampoco permiten analizar todos los servicios que ofrece AWS, si no que se limitan a una lista propia del servicio.

## 2.4 Propuesta

A partir de la idea de crear una arquitectura capaz de procesar los registros que generan los usuarios al realizar acciones en la nube de Amazon, surge CloudTrail-Tracker, herramienta con licencia de código abierto, desplegada completamente en la nube, que permite facilitar el seguimiento de eventos generados por los usuarios de AWS en un tiempo razonable, sin coste añadido, solo pagando por los recursos que necesita para funcionar. En función del uso de la herramienta, ésta puede operar dentro de la capa gratuita (AWS Free Tier [39]) por lo que el coste de utilizar la herramienta es cero. Ofrecemos un control total sobre nuestros datos y la posibilidad de que cada usuario modifique la propia herramienta a su antojo.

# 3. Análisis del problema

Con este proyecto se pretende conseguir una herramienta para el almacenamiento, procesado y posterior análisis y consulta sobre las acciones realizadas por los múltiples usuarios de una cuenta de AWS. Para esto, hemos diseñado una arquitectura en la nube con la que poder llevar a cabo las diferentes tareas.

## 3.1 Tecnologías empleadas

A continuación, describimos las tecnologías y servicios empleados, así como otras posibles soluciones al problema de procesar y analizar en tiempos razonables tal cantidad de datos para su posterior consulta, sin aumentar considerablemente el coste.
Distinguimos entre los servicios ofrecidos por Amazon y las herramientas de terceros utilizadas para la creación de la herramienta.

### 3.1.1 Servicios de AWS

#### *AWS IAM*

AWS IAM (Identity and Access Management, [40]) es un servicio que permite administrar el acceso a los servicios y recursos en una cuenta de AWS de forma segura, se pueden administrar usuarios y grupos, así como utilizar permisos para conceder o denegar el acceso de estos a los recursos de AWS. A la vez, permite crear funciones de IAM, permitiendo delegar acceso a usuarios o servicios que no suelen tener acceso a los recursos de AWS de la organización.

Gracias a AWS IAM las organizaciones o grupos de trabajo se bastan con una cuenta en AWS para compartir todos sus recursos en la nube. Gracias a los grupos de usuario y a las funciones de IAM es mucho más fácil administrar los permisos.

IAM es una característica de la cuenta de AWS que se ofrece sin cargos adicionales.

#### *Amazon S3*

S3 (*Simple Storage Service*) [21] es un servicio de almacenamiento en la nube ofrecido por Amazon. Ofrece almacenamiento de ficheros basado en buckets (nomenclatura que se utiliza para la raíz de las carpetas de almacenamiento en S3) y está diseñado para ofrecer una durabilidad del 99,999999999%. Esta solución se integra con la mayoría de servicios de AWS y ofrece una API con la que poder integrarse con servicios de terceros de forma sencilla.

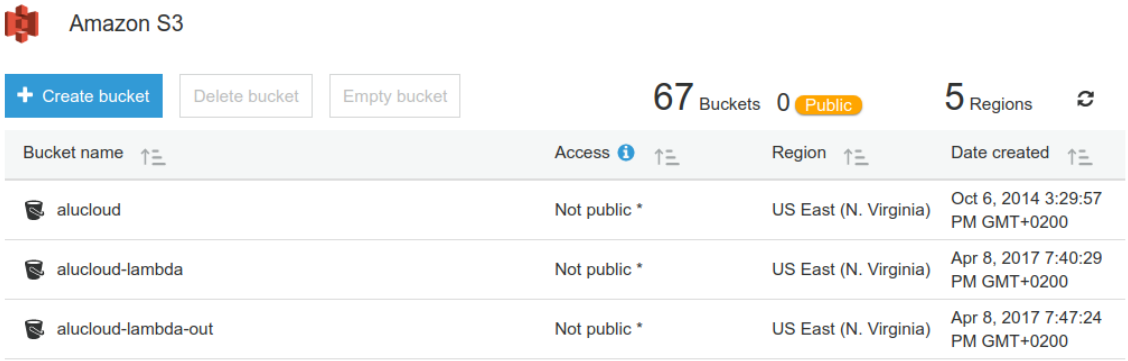



*Figura 11: Ejemplo de bucket S3*

La función de Amazon S3 en CloudTrail-Tracker será de guardar todos los eventos que ocurran en la cuenta de AWS en un bucket como los que vemos en la figura 11. Es el servicio CloudTrail el que se encargará periódicamente de depositar en dicho bucket los ficheros JSON comprimidos con la descripción de los eventos que han sucedido.

La ventaja principal que aprovechamos de este servicio son las notificaciones cuando se crea un nuevo fichero. Estas notificaciones tienen la opción de ser entregadas directamente a AWS Lambda, servicio con el que podemos ejecutar funciones sin requerir un servidor dedicado esperando para ejecutar ésta y simplemente devolviendo el resultado, con lo que podremos iniciar un flujo de trabajo y procesar los eventos que van surgiendo, sin tener que levantar un servidor dedicado e ir vigilando si hay nuevos eventos.

| **Almacenamiento estándar en S3** | |
|---|---|
| Primeros 50 TB/mes | 0,024 USD por GB |
| Siguientes 450 TB/mes | 0,023 USD por GB |
| Más de 500 TB/mes | 0,022 USD por GB |

*Figura 12: Precios en AWS S3*

Usando este servicio sólo pagaremos por lo que utilicemos [43], sin tener que pagar un mínimo de cuotas en ningún momento, evitando así sobrecostes en los servicios. Como vemos en la figura 12, los precios por TBytes son relativamente bajos y, aunque depende de la región, no varían mucho entre ellas.
En la capa gratuita disponemos de 5 GBytes, 20.000 solicitudes Get, 2.000 solicitudes Put y 15 GB de transferencia de datos saliente al mes durante un año, por lo que con dicha capa cubrimos las necesidades del proyecto sin coste alguno.

### *AWS Lambda*

"*AWS Lambda es un servicio informático que permite ejecutar código sin aprovisionar ni administrar servidores. AWS Lambda ejecuta el código sólo cuando es necesario, y se escala de manera automática, pasando de pocas solicitudes al día a miles por segundo. Solo se paga el tiempo de computación que se consume; no hay ningún cargo mientras el código no se ejecuta.*" [22]

Así es como define Amazon el servicio AWS Lambda. Este servicio será una de las bases de nuestra propuesta para disminuir el coste total, al no tener un servidor dedicado siempre en marcha esperando peticiones, si no que pagaremos solo en función del tiempo de ejecución de las funciones y los recursos (en términos de memoria) asignados a ellas.

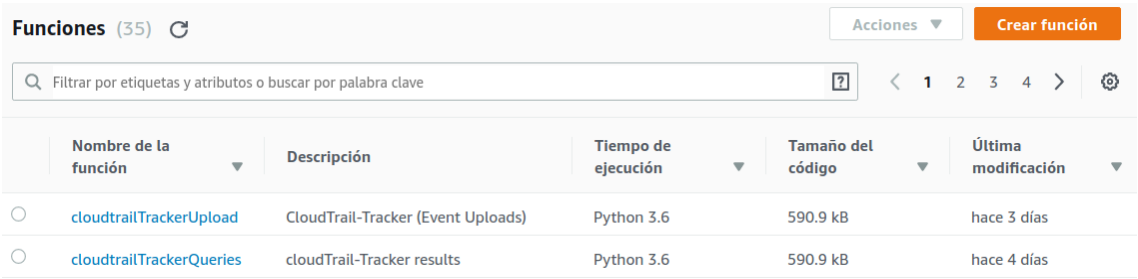



*Figura 13: Funciones Lambda usadas en CloudTrail-Tracker*

Podemos crear una serie de disparadores para que por cada objeto subido a un bucket de S3, se ejecute cierta función Lambda, con el bucket y la ruta del objeto como parámetros, entre otros. Las funciones en AWS Lambda pueden ejecutarse durante un máximo de 5 minutos y tener asignadas un máximo de 3008 MB de RAM [41].

Se puede configurar la potencia de estas funciones, dándoles más memoria y potencia de procesamiento. El coste de éstas depende de la potencia suministrada y de los milisegundos de tiempo que haya estado ejecutándose. Dicho coste se calcula a partir de una tabla [23], dependiendo de su duración y la memoria asignada, y el número de ejecuciones.

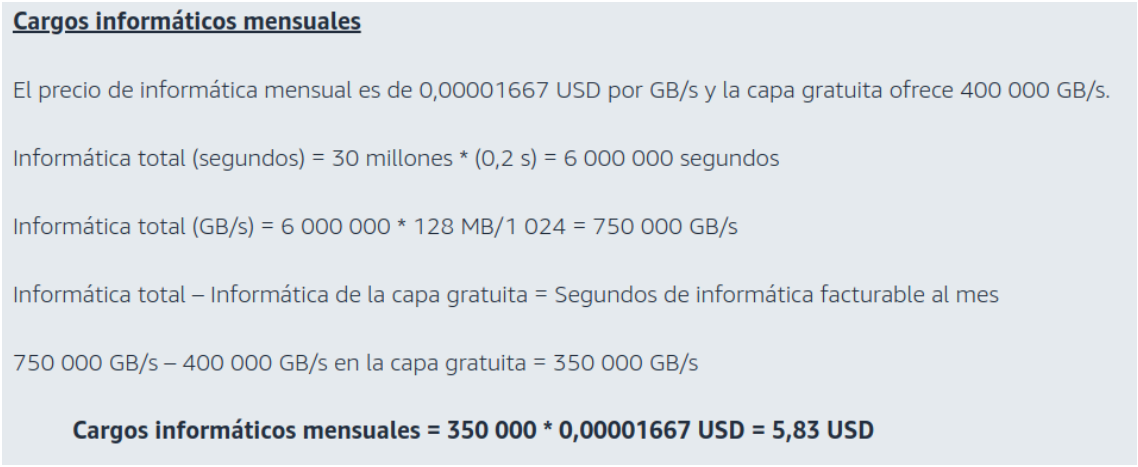



*Figura 14: Ejemplo de cargos mensuales en AWS Lambda*

En la figura 14 podemos ver un ejemplo del coste de una función Lambda si se le asigna 128 MBytes de memoria, se ejecuta 30 millones de veces en un mes y su ejecución dura 200 ms cada vez. El coste total es de 5.83 $.

AWS Lambda ofrece una capa gratuita de un millón de solicitudes al mes con 400.000 GBytes o segundos durante 12 meses, de forma indefinida. Dicha capa cubre las necesidades del presente proyecto.

### *Amazon DynamoDB*

DynamoDB [24] es el servicio de base de datos NoSQL proporcionado por Amazon, el cual proporciona un óptimo escalado de rendimiento y almacenamiento de forma automática, permitiendo realizar consultas consiguiendo respuestas consiguiendo respuestas con una latencia muy baja.
Es el servicio que nos soluciona la parte de almacenado de datos en nuestra propuesta.

Este servicio se basa en la creación de tablas sin esquemas fijos, permitiendo que cada una tenga un número de columnas diferente, enriqueciendo y flexibilizando el modelo de datos.
Nos permite también crear índices secundarios para añadir una mayor flexibilidad y velocidad a la hora de realizar consultas a la vez que ofrece sistemas de replicación automática entre regiones. Además, as unidades de lectura y escritura indican la capacidad de desempeño de ambas acciones. A mayor número de éstas, se garantiza un desempeño uniforme y disminuye la latencia.

El coste de DynamoDB [44] depende del número de unidades de lectura y escritura que tengamos contratados y del número de índices secundarios creados. El servicio no tiene en consideración el uso de CPU y memoria que realizará, como sí pasa con otras soluciones NoSQL, si no que aprovisiona automáticamente las previsiones de hardware para llegar al objetivo marcado por las unidades de lectura y escritura.
Una unidad de escritura proporciona hasta una escritura por segundo, suficiente para lograr 2,5 millones de escrituras al mes, tiene un coste de 0.47 $. Para las de lectura, que proporcionan hasta dos lecturas por segundo, suficiente para lograr 5,2 millones de lecturas al mes, tienen un

coste de 0.09 $. Además, el servicio cobra una tasa horaria por GByte de espacio de disco que consume su tabla, desde 0.25 $.

La capa gratuita de este servicio es de 25 unidades lectura, 25 unidades de escritura por cada índice y 25 GBytes de datos indexados. Esta capa cubre las necesidades del presente proyecto.

### *Amazon API Gateway*

Este servicio es la capa de API-management que ofrece AWS. Permite crear interfaces de programas de aplicaciones, (*Application Programming Interface*, API) definiendo los endpoints, métodos de acceso y las diferentes integraciones que tienen estos como, por ejemplo, una función Lambda.

API-Gateway [25] permite gestionar entornos (a los cuales les llama *stages*), crear variable de entorno, describir cada uno de los endpoints, añadir seguridad de acceso o restringir usuarios.
También ofrece herramientas de administración del tráfico, monitorización y análisis.
Es necesario tener en cuenta el límite máximo por petición, el cual es de 29 segundos [42]; pasado ese tiempo la API devolverá un error.

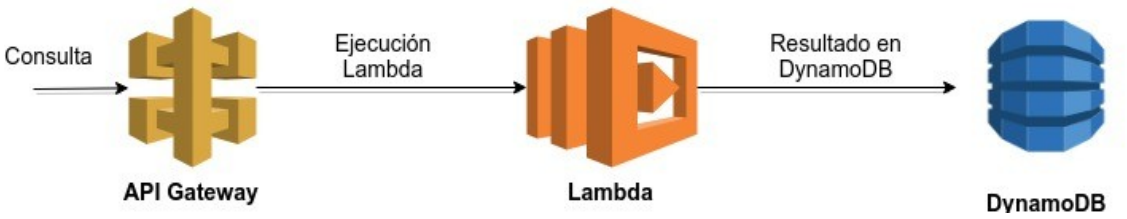



*Figura 15: Ejemplo de arquitectura usando API Gateway, Lambda y DynamoDB*

Un ejemplo de arquitectura usando API Gateway se puede ver en la figura 15. El usuario realiza una consulta a API Gateway, esta petición acciona una función Lambda con los parámetros que ha introducido el usuario y dicha función almacena los datos en DynamoDB. Si bien, podría tanto almacenar como consultar los datos, devolviendo así los resultados desde la misma API.

El coste de cada API depende del número de peticiones realizadas, sin ningún coste mínimo.
El coste actual que ofrece es de 3,50 $ por millón de llamadas al API recibidas, más el costo de los datos de salida transferidos en gigabytes, 0,09 $/GB por los primeros 10 TBytes y disminuyendo para los siguientes TBytes [45].
API Gateway ofrece una capa gratuita durante el primer año de uso. Proporciona un millón de llamadas sin coste alguno, lo que cubre las necesidades de nuestra propuesta sin coste adicional.

Cargos por llamadas a la API de Amazon API Gateway = 5 millones * 3,50 USD/millón = **17,50 USD**

Tamaño total de las transferencias de datos = 3 KB * 5 millones = 15 millones/KB = 14,3 GB

Cargos por las transferencias de datos de Amazon API Gateway = 14,3 GB * 0,09 USD = **1,29 USD**

**Cargos totales de Amazon API Gateway = 17,50 USD + 1,29 USD = 18,79 USD**

*Figura 16: Ejemplo de Coste en API Gateway*

En la figura 16 vemos un ejemplo de cómo se calcula el coste, con una que recibe cinco millones de llamadas a la API al mes, con cada llamada a la API que envía respuestas de 3 kilobytes (KB) de tamaño sin almacenamiento en caché.
Como vemos, el coste se calcula de forma mensual, teniendo en cuenta la transferencia de datos por un lado y el número de llamadas a la API por otro.

### *AWS CloudFormation*

AWS CloudFormation [26] proporciona un lenguaje que permite, a través de plantillas en formato de archivos de texto, modelar toda la infraestructura en la nube necesaria, describir cada uno de los componentes y aprovisionar de los recursos necesarios. A partir de esta plantilla se automatiza el aprovisionamiento, siendo el mismo servicio el que se encarga de determinar las operaciones correctas a realizarse al administra la pila, anulando los cambios de forma automática si se detecta algún error.

Además, AWS CloudFormation proporciona un diseñador de plantillas en la web [27], el cual permite seleccionar los componentes, arrastrarlos y configurarlos de forma sencilla y muy visual, generando posteriormente un fichero YAML o JSON para su ejecución.

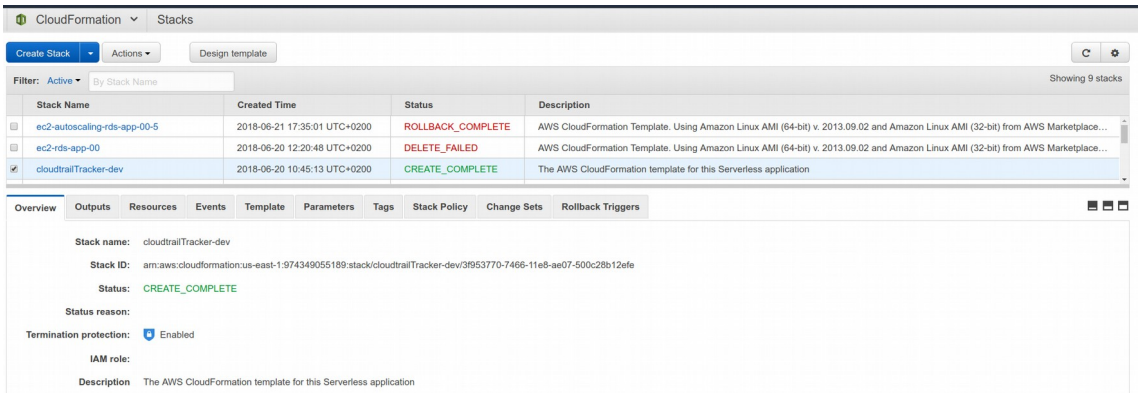



*Figura 17: Pilas en CloudFormation*

Por cada plantilla que se ejecute en AWS CloudFormation se crea una pila con el nombre de ésta. Como vemos en la figura 17, cada pila guarda toda la información y tiene un estado. Podemos ver como la plantilla *"cloudtrailTracker-dev"* se ha creado de forma satisfactoria, lo que quiere decir que se ha acabado de ejecutar sin errores y todos los componentes están operativos de la forma descrita en su plantilla.
También podemos ver como otra plantilla contiene errores y está en estado *"rollback complete"*, que quiere decir que ha deshecho los cambios realizados, dejando el sistema como estaba antes de ejecutarse. Esto es muy útil en caso de equivocarnos en la plantilla, ya que no deja un sistema inestable.
Hay otra plantilla con el estado *"delete failed"*, lo que podría ser por no tener permisos para borrar algunos de los componentes o porque se han modificado a mano posteriormente.

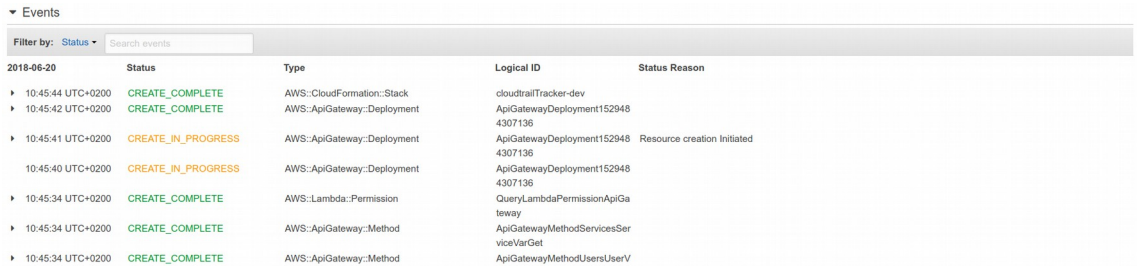



*Figura 18: Vista de la pila de CloudTrail-Tracker en CloudFormation*

En la figura 18 se ve parte del resultado de la ejecución de la plantilla creada para nuestra propuesta. Podemos observar como por cada componente hay mínimo un mensaje y un estado, teniendo en todo momento el conocimiento de lo que se está ejecutando en dicha receta.

AWS CloudFormation está disponible sin gastos adicionales, solo se paga por los recursos de AWS necesarios para ejecutar la infraestructura de la plantilla.

### 3.1.2 Herramientas y Lenguajes Utilizados

A continuación, vemos las herramientas y lenguajes utilizados en nuestra propuesta.

#### *Python*

Python [28] es un lenguaje de programación interpretado, multiparadigma, el cual posee una licencia de código abierto. Es un lenguaje ideal para scripting y rápido desarrollo de aplicaciones en la mayoría de plataformas.
En este proyecto hemos usado la versión 3.6 del lenguaje frente a otras posibilidades como Java o Node.js, lenguajes en los que también se puede desarrollar funciones en AWS Lambda [29], debido a la experiencia previa con dicho lenguaje y la poca curva de aprendizaje que tiene al usar nuevas librerías.

#### *Boto3*

Boto3 [30] es un paquete de Python que proporciona una interfaz para usar los servicios de AWS. Nos permite utilizar servicios como DynamoDB, S3 o EC2, entre otros.
Todas las funciones Lambda creadas en este proyecto se basan en código escrito sobre las funciones boto, así como todas las consultas a la base de datos de DynamoDB.

Para poder usar boto debemos tener configurada nuestra cuenta de AWS en nuestra línea de comandos. para ello, se debe editar el archivo **~/.aws/credentials** o ejecutar **aws credentials** [31].

#### *Serverless Framework*

Es un framework de código abierto, escrito en Node.js, el cual nos permite desplegar funciones Lambda junto con otros servicios como API Gateway o Cognito, configurando todos los aspectos de forma sencilla, basándose en CloudFormation. Gracias a CloudFormation, Serverless Framework puede modelar, automatizar e implementar toda la infraestructura a través de un fichero modelo en YAML o JSON.

Serverless proporciona una forma sencilla de crear y comunicar una API REST con funciones Lambda, especificando las rutas y parámetros en un mismo fichero, donde también se especifican las funciones Lambda a ejecutar y sus disparadores. Otra ventaja es la facilidad para añadir cambios y mantener el código. En definitiva, esta herramienta se ha utilizado para facilitar el despliegue de la arquitectura de aplicación propuesta sobre AWS.

#### *Swagger*

Es un framework para desarrollar la documentación de APIs RESTful (*Representational State Transfer*), lo que significa que trabaja sobre el protocolo HTTP y tiene en cuenta los códigos de



estado de dicho protocolo de forma correcta [32]. También proporciona una interfaz visual para consultar ejemplos de cada llamada y visualizar la documentación de forma sencilla y rápida. Swagger ayuda a crear una guía de la API a través de unas reglas y especificaciones para que todo el mundo pueda entenderla, aunque se tengan pocos conocimientos sobre éstas.

```
1  swagger: '2.0'
2  info:
3    description: API definition of CloudTrail-Tracker
4    version: 1.0.0
5    title: Swagger CloudTrail-Tracker
6    contact:
7      email: joprfon@inf.upv.es
8  basePath: '/{stage}/cloudtracking_querys'
9  tags:
10   - name: scan
11     description: General query
12   - name: services
13     description: General services query
14   - name: users
15     description: Information about user actions
16 schemes:
17   - http
18 paths:
19   /scan:
20     get:
21       tags:
22         - scan
23       summary: 'General query, no user or service filter'
24       description: 'General query, no user or service filter'
25       operationId: scan
26       produces:
27         - application/json
28       parameters:
29         - name: eventName
30           in: query
31           description: Event name to filter
32           required: false
33           type: string
34         - name: count
35           in: query
36           description: Return a number instead a json
37           required: false
38           default: false
39           type: boolean
```

*Figura 19: Documentación en Swagger*

En la figura 19 se ve parte de la documentación de la API que forma parte de nuestra solución. Se muestran tres rutas principales en el apartado “tags”, que son “scan”, “services” y “users”. A partir de la línea 19 se define la ruta de “scan”, en la figura se ven algunos parámetros usando el método “GET” (línea 20) como “eventName” y “count”, añadiendo información necesaria como si el parámetro va en la consulta de forma obligada o es opcional, el tipo y el valor por defecto. Más adelante, se proporcionarán detalles adicionales sobre el API.

## 3.2 Solución propuesta

Nos hemos decidido por una solución completamente en la nube, robusta y escalable, con los costos únicamente de los servicios que usamos, sin depender de tener un servidor configurado.
Vamos a describir la solución dividiéndola en tres capas, una para la API, otra para el procesamiento y una última para el almacenamiento.

### 3.2.1 Consultas

Para las consultas hemos seleccionado el servicio API Gateway de AWS, frente a la opción de tener un servidor dedicado únicamente a ejecutar una API y enrutar las peticiones. Nos permite abaratar costes en este sentido, al no tener hardware dedicado específicamente para esta función. Además, este servicio proporciona una conexión directa con los servicios en AWS y destaca también por las facilidades que aporta a la hora de configurar, ya sea de forma visual en la web o con frameworks como Serverless. También se puede exportar e importar la API en formato Swagger.

API Gateway proporciona una capa gratuita de un millón de consultas durante los 12 primeros meses, posteriormente el coste por cada millón de consultas es de 3,5 $ más el costo de los datos de salida transferidos en gigabytes, que viene siendo unos 0,09 USD/GB por los primeros 10 TBytes. Se tiene en cuenta que dicho servicio tiene una limitación de 30 segundos máximos por consulta, en caso de sobrepasar este límite la API responderá con un error.

### 3.2.2 Procesamiento

Nos hemos decantado por una solución serverless usando AWS Lambda, es decir, no necesitamos un servidor siempre ejecutándose y esperando peticiones, sino que, a través de la API, que recibirá las consultas, ejecutará una función Lambda y solo pagaremos por los recursos asignados y el tiempo de ejecución de la función. La función Lambda solo estará ejecutándose unos milisegundos, dependiendo de la consulta, a diferencia de tener un servidor dedicado, que debería estar ejecutándose todo el tiempo, aunque no haya consultas.

Las virtudes de esta tecnología son el bajo coste al no tener un servidor dedicado, como hemos mencionado anteriormente, y eliminamos los tiempos de mantenimiento del software, al no tener que preparar y mantener dichos servidores.

### 3.2.3 Almacenamiento

Hemos optado por el uso de bases de datos no relacionales, más concretamente DynamoDB, que es el servicio de bases de datos NoSQL que ofrece AWS. Para esta decisión nos hemos basado en la cantidad de datos que se disponen y en los tiempos de respuesta necesarios, siendo necesariamente menores de 29 segundos en cada consulta, debido al timeout de API Gateway.
Aprovechamos las ventajas derivadas de no tener que crear un esquema fijo de tabla, añadiendo eventos con multitud de campos diferentes, permitiendo adaptarse a nuevos eventos previamente no vistos o no tenidos en cuenta.

### 3.2.4 Presupuesto

Para la realización de este proyecto se han necesitado una alta cantidad de eventos generados por CloudTrail a lo largo del tiempo. Dichos datos han sido necesarios para:

- Comprobar los tiempos de carga de datos.
- El tamaño de la tabla en DynamoDB para conocer la cantidad de datos manejada.
- Análisis de la información de los eventos.
- Relación del número de eventos almacenados en la tabla frente al número de eventos almacenados en el bucket de s3.
- Recogida de tiempos de latencia en las consultas

Por ello, el Grupo de Grid y Computación de Altas Prestaciones (GRyCAP) nos ha proporcionado cuatro años de eventos históricos que equivale a un total de 1.61 GBytes de datos en la forma de cientos de miles de ficheros JSON que incluyen registros de las actividades prácticas realizadas por alumnos de las asignaturas Servicios en la Nube (Máster Universitario en Gestión de la Información), Infraestructuras de Cloud Público (Máster Universitario en Computación Paralela y Distribuida) así como del Curso Online de Cloud Computing con Amazon Web Services (AWS), entre otras, desde el curso académico 2013/2014. También se ha tenido acceso al bucket en el que se guardan periódicamente los nuevos eventos generados por el uso de los diferentes servicios.

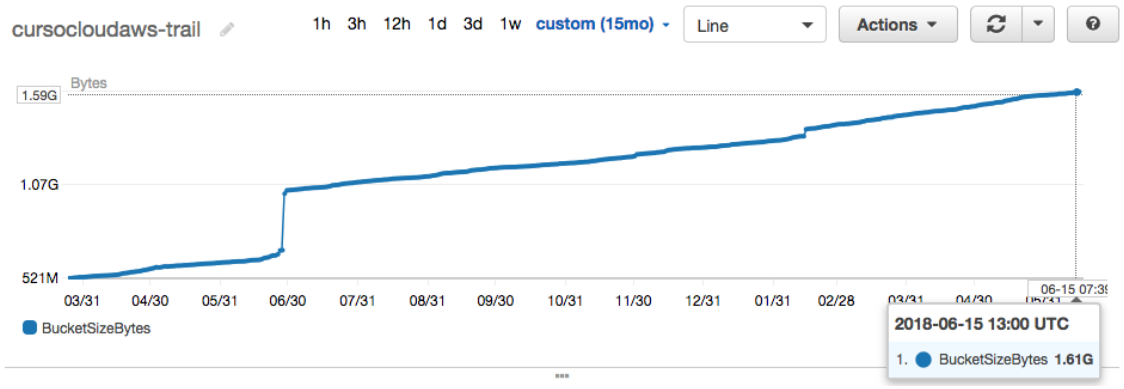



*Figura 20: Evolución del bucket S3 usado por CloudTrail -en un año de uso*

En la figura 20 se puede observar la evolución del bucket S3 usado por el servicio CloudTrail, al guardar los eventos. Se ve como en el último año y medio ha pasado de 512 MBytes a 1.6 GBytes.

Además, el GRyCAP nos ha proporcionado un usuario en la misma cuenta de AWS para poder crear y usar los servicios y herramientas necesarias, como han sido EC2, S3, funciones Lambda, entre otros.

Finalmente, hemos hecho una estimación del tiempo empleado por cada tarea junto a sus dependencias, como se puede ver en el diagrama de Gantt de la figura 21, también adjuntado en los anexos para una mejor visualización.

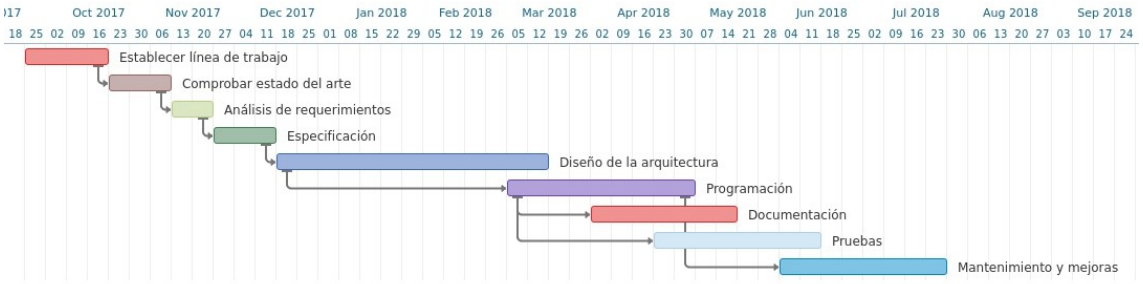



*Figura 21: Diagrama de Gantt*

Lo primero que hicimos fue establecer la línea de trabajo, precisado la idea del proyecto. Una vez acabada de definir la línea de trabajo, pasamos a hacer una actividad de búsqueda sobre el estado del arte, tanto de la computación en la nube como de otras soluciones para la monitorización de actividades en la nube.

El análisis de requerimientos del proyecto, dónde se incluyen los objetivos de éste, empieza al finalizar la búsqueda en el estado del arte. Una vez finalizado este análisis y a partir del resultado, se empieza con las especificaciones del proyecto, donde detallamos la forma en la que se comportará el servicio con el usuario.

A continuación, empieza la tarea de diseño de la arquitectura, en la cual se determina cómo funcionará el sistema de forma general sin entrar en detalles, que componentes lo forman y cuales se comunicarán entre ellos.

La fase de programación empieza sin la necesidad de haber terminado el diseño, ya que a medida que se saben algunos componentes se desarrollan prototipos funcionales, por lo que esta fase empieza antes de acabar la fase de diseño. De igual forma, a medida que llevamos ya desarrollado parte del código, se empieza a realizar la documentación del este, así como las pruebas que validan el correcto funcionamiento. Finalmente, una vez concluidas las pruebas, empieza la etapa de mantenimiento y mejoras del servicio, en la que realizamos propuestas de mejora, se valoran y, en un futuro, se implementan.

# 4. Desarrollo de CloudTrail-Tracker

En esta sección hablaremos de la arquitectura creada en el desarrollo de CloudTrail-Tracker, de los componentes del back-end como del front-end.

## 4.1 Arquitectura

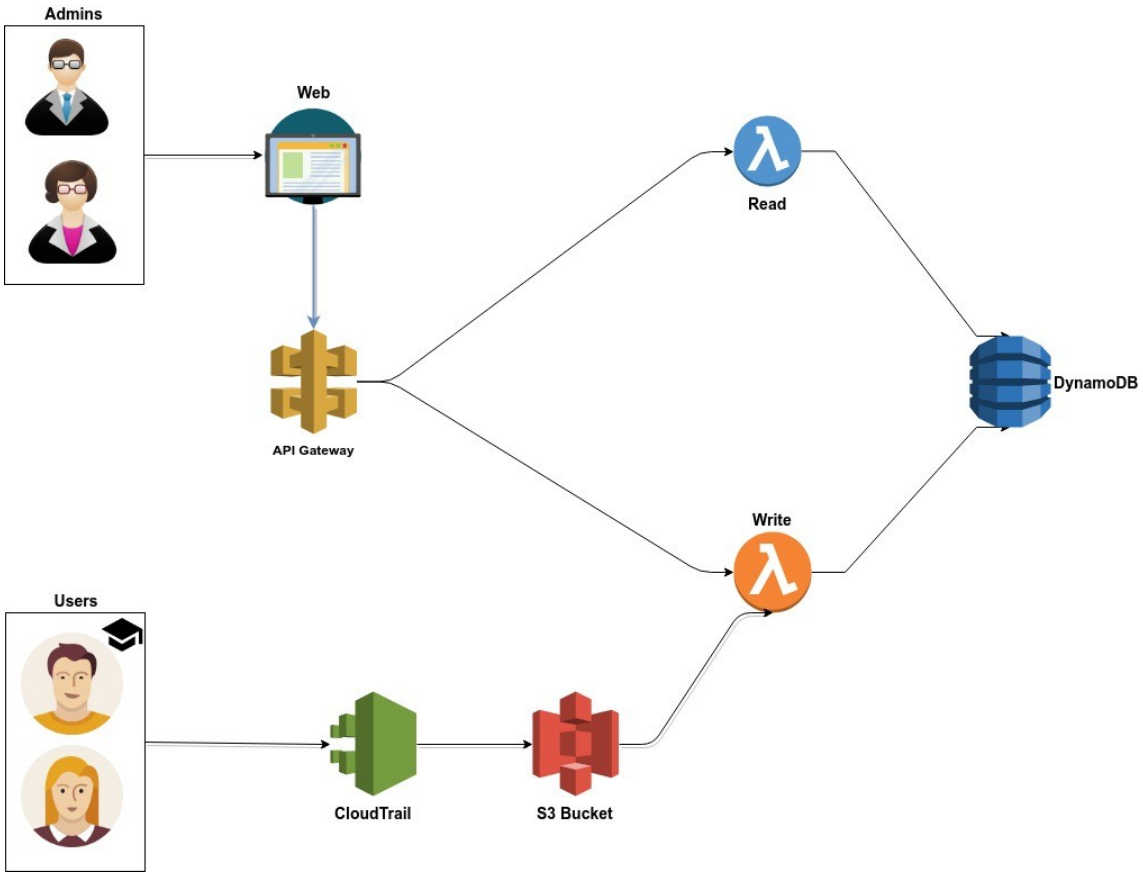



*Figura 22: Diagrama de la arquitectura de CloudTrail-Tracker.*

Tal y como se comentó anteriormente, cuando un usuario IAM utiliza algún servicio de AWS se genera un evento que se almacena en un bucket de S3, gracias a CloudTrail, ya que previamente ha sido configurado especificando el bucket destino donde se almacenarán los registros de eventos.

En primer lugar, para poder almacenar los eventos, se configura una función Lambda, añadiendo un disparador para que cada vez que se almacena un fichero acabado en *".json.gz"*, se ejecute, es decir, por cada acción del usuario o de algún servicio. Esta función es la que procesa el evento, elimina información innecesaria, como son algunos identificadores, el número de cuenta, y algunos tipos de eventos, configurables posteriormente. Finalmente almacena los datos en una tabla de DynamoDB.

En segundo lugar, para permitir consultar los eventos, hemos habilitado una API REST con API Gateway. Esta API, al recibir una consulta, lanza una llamada a otra función Lambda diferente a la explicada anteriormente, pasándole todos los parámetros proporcionados por el usuario, la cual acaba recogiendo y filtrando la información de la tabla de DynamoDB.

En la figura 22 tenemos un diagrama sobre la arquitectura del servicio, en el que podemos ver, de forma resumida, qué componentes se comunican entre sí y en qué orden.
Aparecen los usuarios, los cuales al realizar cualquier acción en la cuenta AWS, CloudTrail lo detecta y genera un archivo en el bucket S3 especificado. Aparece en color naranja la función Lambda de procesado y escritura en la base de datos DynamoDB y, en color azul, la función Lambda de lectura sobre la misma base de datos, la cual es invocada desde API Gateway.

Para realizar el desarrollo del servicio CloudTrail-Tracking se han definido los siguientes requisitos, divididos en requisitos funcionales, que son los que proveerá el sistema, y requisitos no funcionales, que son propiedades como fiabilidad, tiempos de respuesta, necesidades del usuario en cuanto al coste, etc.

### 4.1.1 Requisitos funcionales

- Diseñar y desarrollar una serie de consultas relevantes usando Boto, en Python.

- Crear un servicio serverless para la carga de los datos en el back-end de forma automática y otro para la consulta de los datos, evitando tener el gasto de un servidor siempre en marcha.

- Crear un script de carga de datos previos inicial, en caso de tener eventos de fechas anteriores al funcionamiento del servicio.

- Crear una API REST para poder realizar consultas de una forma sencilla desde cualquier navegador o herramienta.

- Implementar una estructura usando DynamoDB para el almacenamiento, permitiendo realizar consultas rápidas y eficientes.

- Facilitar el despliegue de la herramienta usando el framework Serverless para que sea de forma automática y configurable.

### 4.1.2 Requisitos no funcionales

- Realizar un análisis de los datos que se van a almacenar, omitiendo los datos no necesarios, reduciendo la carga y volumen.

- Definir y documentar dicha API con Swagger.

- Subir el código y documentación a un repositorio público del GRyCAP en GitHub, siguiendo las pautas de un proyecto de código libre.

## 4.2 Componentes del Back-end

En esta sección entramos en detalle a ver cómo funcionan los componentes del back-end, separando en escritura los componentes que se encargan de escribir en la tabla de DynamoDB y lectura los componentes que se encargan de realizar las consultas.

### 4.2.1 Escritura

Como hemos explicado anteriormente, cuando un usuario realiza cualquier acción en AWS se genera un evento gracias a CloudTrail. Dicho evento es un archivo con formato JSON que CloudTrail se encarga de almacenar en un bucket de S3. En la figura 23 vemos un ejemplo, una acción en EC2 realizada por el usuario *"alucloud00"*, el 01-06-2017.


```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "userName": "alucloud00",
    "type": "IAMUser",
    "principalId": "AIDAJ2QG6PUWMNOYKCICY",
    "arn": "arn:aws:iam::974349055189:user/alucloud00",
    "accountId": "974349055189",
    "accessKeyId": "AKIAJAIQMN42O7ADSC5A"
  },
  "eventTime": "2017-06-01T04:25:18Z",
  "eventSource": "ec2.amazonaws.com",
  "eventName": "DeleteSnapshot",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "52.206.222.117",
  "userAgent": "Boto/2.42.0 Python/2.7.6 Linux/3.13.0-36-generic",
  "errorCode": "Client.InvalidSnapshot.InUse",
  "errorMessage": "The snapshot snap-24d211a9 is currently in use by ami-20e9ad48",
  "requestParameters": {
    "snapshotId": "snap-24d211a9",
    "force": false
  },
  "responseElements": null,
  "requestID": "03a16002-f427-4d7b-97cb-63add7b60c19",
  "eventID": "58655f7b-82cd-41a1-a762-37a81cf85f80",
  "eventType": "AwsApiCall",
  "recipientAccountId": "974349055189"
},
```


*Figura 23: Ejemplo de evento generado por CloudTrail*

Los eventos no se generan al instante, si no que se van acumulando y a los pocos minutos dicho servicio los junta en un mismo fichero JSON, el cual comprime con formato 'gz'.
El nombre de este fichero se genera concatenando el identificador de la cuenta, añadiendo la región del evento, la fecha en la que se genera el evento y un id del fichero auto-generado. Se puede ver un ejemplo en la figura 24.


*Figura 24: Formato de un fichero creado por CloudTrail*

Este bucket de S3, donde se irán guardando los eventos, lanza una función Lambda por cada fichero que se almacene con nombre acabado en *"json.gz"*.
La función que hemos desarrollado procesa la información de los eventos, eliminando gran parte de información no necesaria para almacenar, como pueden ser datos de sesiones, tokens y versión del evento.

```
functions:
  EventUploads:
    handler: lambda/eventuploads/Upload.handler
    name: cloudtrailTrackerUploads
    description: CloudTrail-Tracker (Event Uploads)
    runtime: python3.6
    memorySize: 128
    timeout: 30
    reservedConcurrency: 5
```

*Figura 25: Configuración en archivo Serverless para la subida de eventos.*

En la figura 25 se muestra como se configura la función Lambda desde Serverless. En la ruta *lambda/eventuploads/Upload.py* nos encontramos con el código de la función de subida periódica. Asignamos 128MB de memoria principal para la ejecución, lo cual determinará, junto al tiempo de ejecución, el coste de dicha función.

Por otro lado, al generarse un evento por cada acción que realice un usuario de una cuenta de AWS, se genera excesiva información innecesaria, como logins en la página web, llamadas a la API de AWS para listar las máquinas de EC2 o generación de tokens. Por tanto, hemos sacado estadísticas de los eventos totales, como se ve en la figura 26, y hemos descartado los eventos que realmente no modifican la infraestructura.

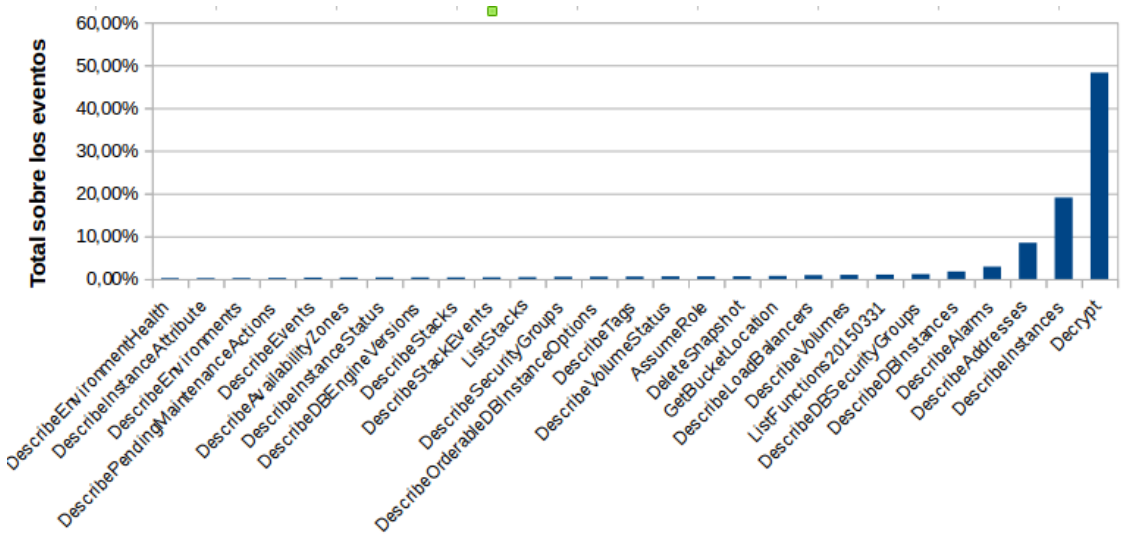



*Figura 26: Gráfico con lo eventos más usados.*

Como vemos, una gran parte de eventos son de acciones que no modifican la infraestructura, tales como "*list*" (acción de listar las máquinas de EC2 o pedir información de un servicio en la

web), "*describe"* (información sobre una máquina en EC2), *"decrypt"* (evento que salta cuando accedemos con un certificado), por lo que los descartaremos. Aun así, se han parametrizado dichos eventos y si en algún caso interesara almacenarlos, se pueden cambiar los filtros fácilmente desde el archivo de configuración, mostrado en la figura 27.

```
filterEventNames = ["get", "describe", "list", "info", "decrypt", "checkmfa", "head", "assumerole"]
```

*Figura 27: Lista de los eventos filtrados en el archivo settings/settings.py*

La ventaja principal de tal reducción en el número de eventos almacenados es mantener la rapidez de las consultas, eliminando más del 70% de los eventos. Además, en la figura 28 tenemos los parámetros que almacenamos de cada evento procesado, reduciendo aún más el tamaño de la base de datos. Esta lista se puede modificar si el usuario está interesado en guardar más información.

```
"""Values that we want to store of an event"""
self.selected = [
    "eventID",
    "userIdentity_userName",
    "awsRegion",
    "eventName",
    "eventSource",
    "eventTime",
    "eventType",
    "userAgent",
    "userIdentity_principalId",
    "requestParameters_bucketName",
    "requestParameters_dBInstanceIdentifier",
    "requestParameters_dBSecurityGroupName",
    "requestParameters_includeAllInstances",
    "requestParameters_includeDeleted",
    "requestParameters_instanceType",
    "requestParameters_roleSessionName",
    "requestParameters_volumeSet_items_0_volumeId",
    "responseElements_credentials_accessKeyId",
    "responseElements_credentials_expiration",
    "responseElements_credentials_sessionToken",
```

*Figura 28: Lista de parámetros por eventos*

En el almacenamiento, DynamoDB permite crear tablas sin una estructura fija, siempre indicando una clave de partición primaria por la cual podremos hacer consultas rápidas en muy poco tiempo. Aparte, podemos añadir sin coste una clave de ordenación, que ayuda a componer la clave primaria (permitiendo dos claves primarias idénticas pero con clave de ordenación distinta) y a poder hacer consultas más potentes y flexibles, como pueden ser consultas entre dos fechas, en el caso de que la clave de ordenación sean fechas, o en caso de que sean cadenas de caracteres, que empiecen o acaben con algún patrón.
También nos permite añadir un índice secundario global, junto con otra clave de ordenación, por la que podremos hacer consultas a partir de ésta igual de rápidas que con la clave primaria, con un coste extra.

En nuestro caso la clave primaria será el identificador único del evento, con nombre 'eventID', siendo una cadena de caracteres, y como clave de ordenación el nombre del usuario que ha

realizado el evento en cuestión, con nombre *'userIdentity_userName'*, también una cadena de caracteres. Esta clave primaria nos permite tener todos los eventos sin solaparse entre ellos y, gracias a la clave de ordenación, poder realizar consultas por usuario con tiempos de respuesta muy ajustados.

Hemos añadido también una clave secundaria global con el nombre como clave primaria y la fecha y hora como clave de ordenación (*"eventTime"*), lo cual nos permitirá realizar consultas entre dos franjas de tiempo.

Dado que en una tabla no podemos realizar consultas y agrupar los resultados, es decir, no podemos realizar una petición preguntado por una lista de usuarios sin que tenga que descargarse todos los eventos y hacer un recuento sobre éstos, lo cual es muy limitante y puede ser costoso dependiendo del número de eventos, por lo que hemos creado un ítem especial con un identificador reservado.
Este ítem, que tiene como identificador el número 1 y se puede ver en la figura 29, tendrá tres campo. El primero, llamado *"listUsers"*, guarda la lista de usuarios que han ido apareciendo en los eventos almacenamos, para que posteriormente tenga coste constante la petición de la lista. El segundo, llamado *"services"*, cumple la misma función que el anterior pero con los servicios, es otra lista que se va actualizando. El tercero y último es otra lista actualizada con los parámetros que han ido apareciendo. Éste último es útil porque permite listar los valores del campo *"param"* de la API, explicado en un apartado más adelante.

| eventID | userIde | event | cols | listUsers | services |
|---|---|---|---|---|---|
| 1 | all | 1 | { "awsRegion" : { "S" : "1" }, "eventID" : { "S" : "1" }, ... | { "acarrion" : { "S" : "1" }, "acm" : { "S" : "... | { "acm" : { "S" : "1" }, "apigateway" :... |

*Figura 29: Ítem especial con los usuarios, servicios y parámetros especiales disponibles.*

Esto tres campos son realmente diccionarios, con la clave con el valor que nos interesa. Esto es así para disminuir el coste de comprobar si el valor buscado está en dicha lista y agilizar así la función Lambda que ejecuta el código.
Aunque esta solución penaliza ligeramente las subidas de nuevos eventos a la tabla, ya que para cada evento se necesita consulta dicho ítem y se actualiza, permite, a través de una consulta directa, obtener esta información de forma instantánea sin tener que descargar toda la tabla.

### *Subida de datos históricos*

Hemos explicado que, al crearse un fichero en un bucket de S3, se suben eventos de forma periódica. También ofrecemos otra forma de cargar eventos teniendo un histórico de datos en un bucket de S3 o en una carpeta local.

Para ello, hemos desarrollado un script en Python que permite la carga de eventos históricos desde una de estas dos opciones, tal y como se haría desde la función Lambda. Hemos añadido dos parámetros para poder filtrar los eventos entre dos rangos de fechas.

A continuación, se muestra un ejemplo de carga de eventos desde un bucket con fecha límite del 2017-06-01:

```
~$ python dynamodb/Logs.py --bucket_name "cloudtrail_bucket" --
t "2017-06-01"
```

Es aconsejable ejecutar el script desde una máquina EC2 desplegada en la misma región geográfica en la que se ha creado el bucket de S3, ya que los tiempos de carga disminuyen considerablemente al tener una latencia menor entre componentes.

### 4.2.2 Lectura

Para poder consultar los datos de DynamoDB hemos desarrollado una serie de funciones en Python 3.6 y Boto 3. Estas funciones están en el fichero dynamodb/Querys.py, las cuales se llamarán desde la API.

Para la parte de las consultas nos hemos decidido por API Gateway, ya que nos permite crear una API REST con la que poder crear una capa de abstracción sobre los demás componentes.
La definición de dicha API se describe en el fichero YAML de Serverless, ubicado en la raíz del proyecto.

```
events:
  - http:
      path: scan/
      method: get
      cors: true
      integration: lambda
      request:
        parameters:
          querystrings:
            count: false
            eventName: false
            from: false
            to: false
            param: false
            value: false
        template:
          application/json: '{ "scan" : "scan",
                               "count" : "$input.params(''count'')",
                               "eventName" : "$input.params(''eventName'')",
                               "from" : "$input.params(''from'')",
                               "to" : "$input.params(''to'')",
                               "param" : "$input.params(''param'')",
                               "value" : "$input.params(''value'')"}'
```

*Figura 30: Ejemplo de la configuración de una de las rutas de la API desde Serverless*

En la figura 30 se muestra cómo se describe una de las rutas de la función Lambda para la consulta de eventos. Dicha ruta es */scan*, la cual se explicará a continuación, junto al resto de rutas. Se especifica el tipo de método que va a utilizar la API para esta ruta, el tipo de integración y los diferentes parámetros que hemos añadido a la función, para crear consultas más complejas.

Esta API se ha documentado totalmente con el framework Swagger. A continuación, se muestran capturas de la documentación, donde se definen las diferentes rutas de la API, con sus respectivos parámetros y los valores a devolver.



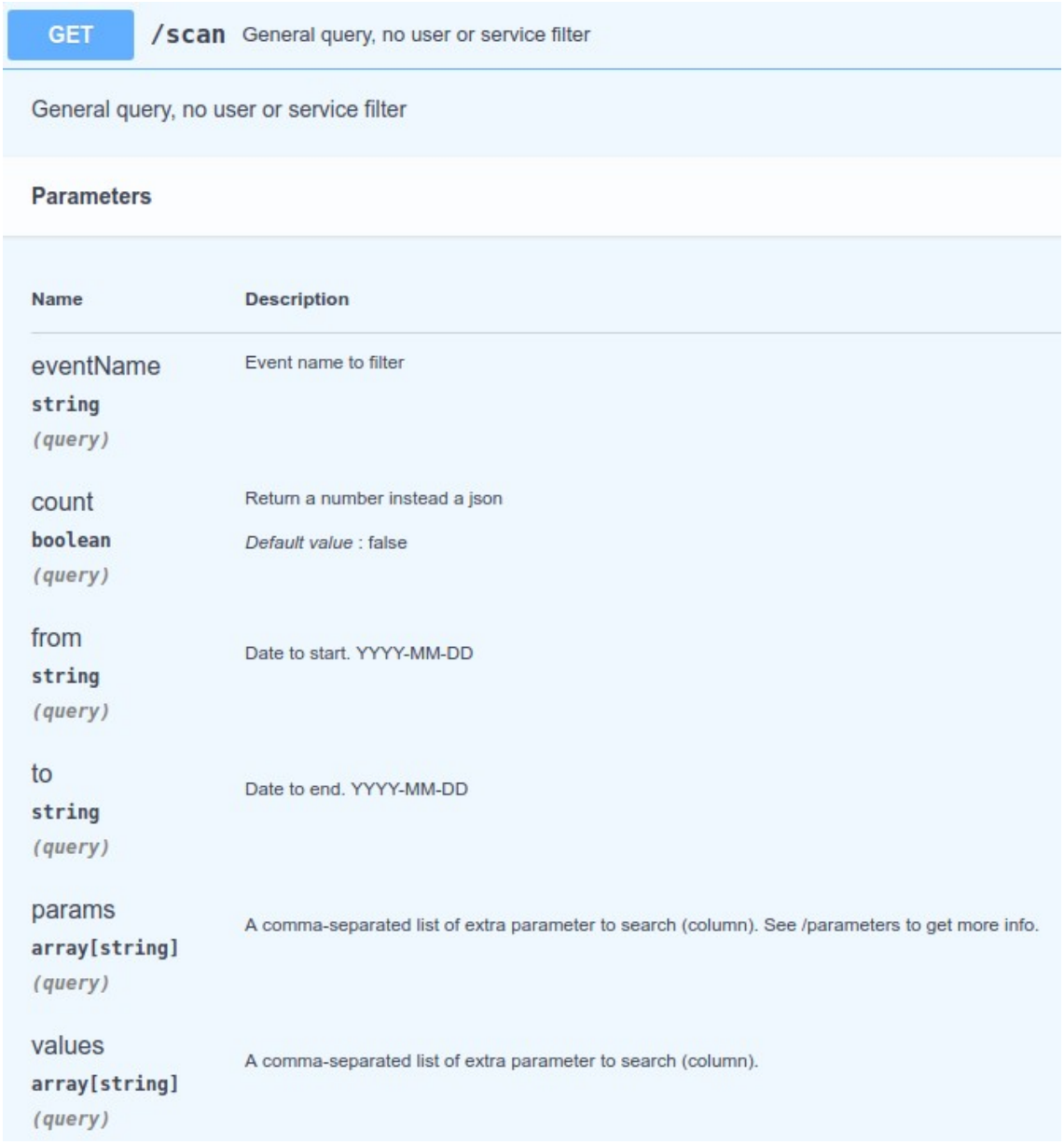



*Figura 31: Ruta scan de la API.*

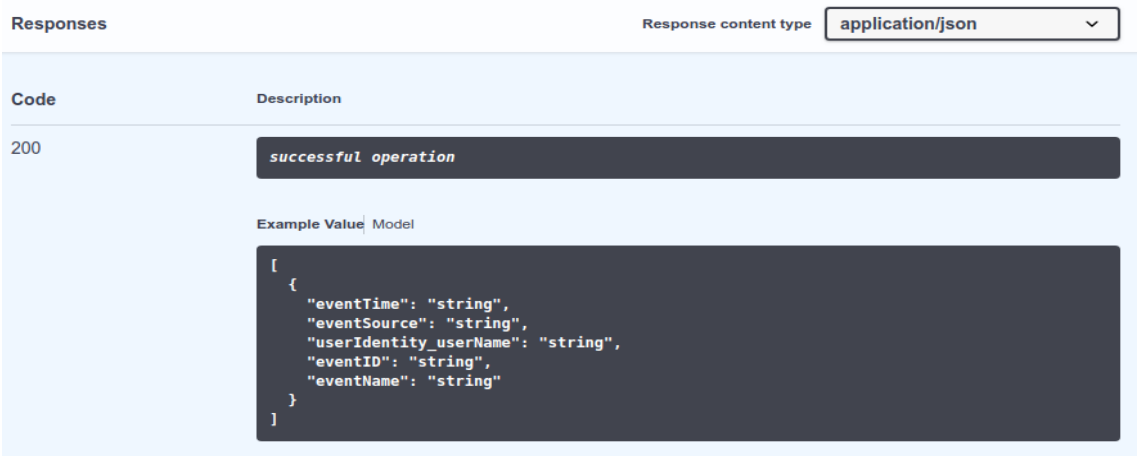



*Figura 32: Resultado de la ruta scan de la API.*

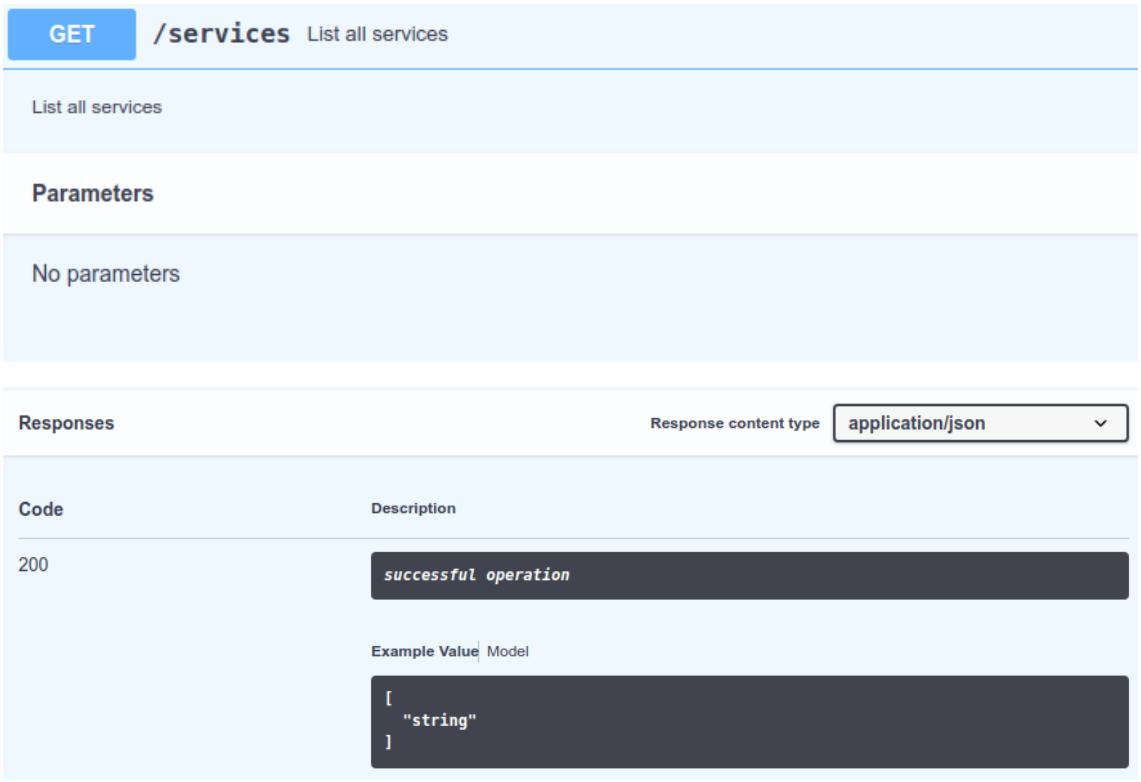



*Figura 33: Ruta y resultado de services de la API.*

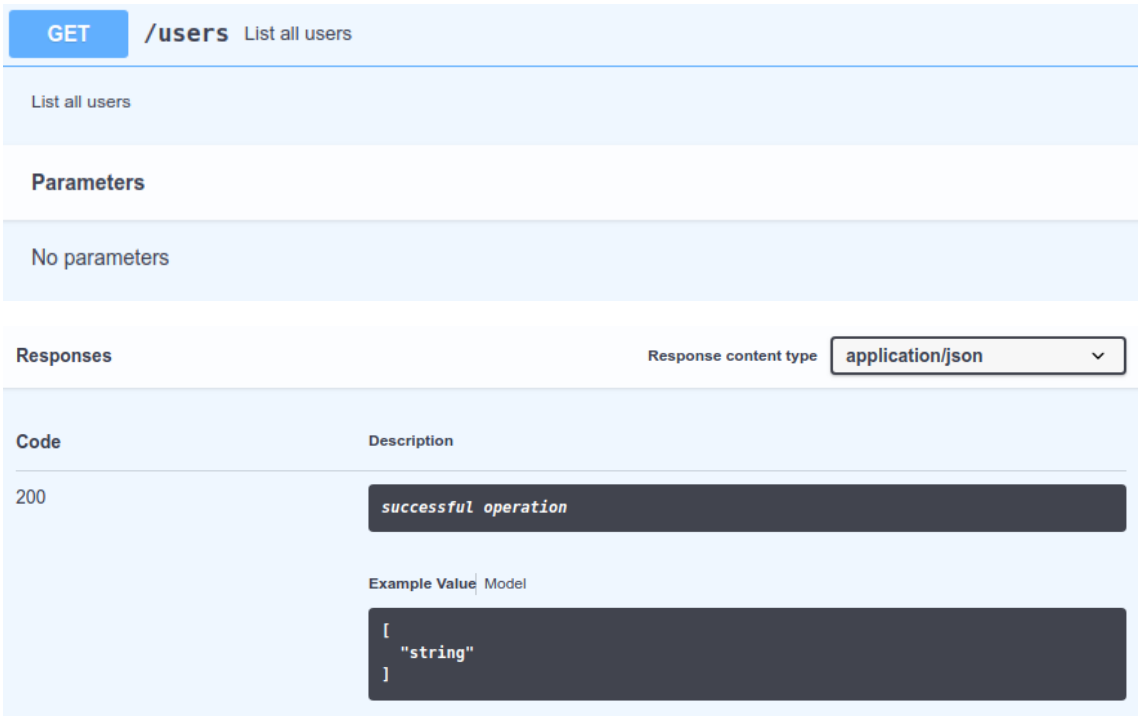



*Figura 34: Ruta y resultado de users de la API.*

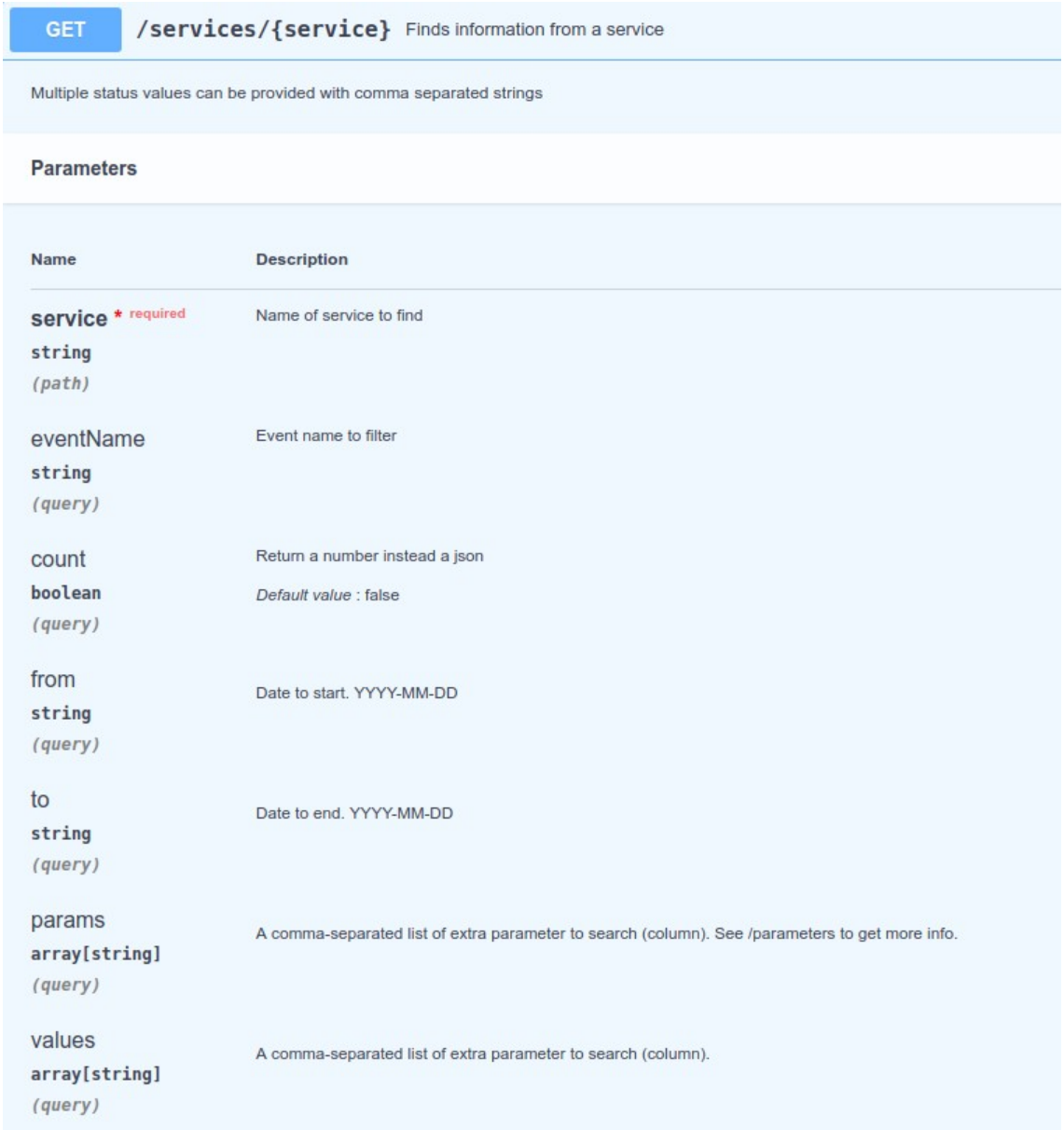



*Figura 35: Ruta services/service de la API.*

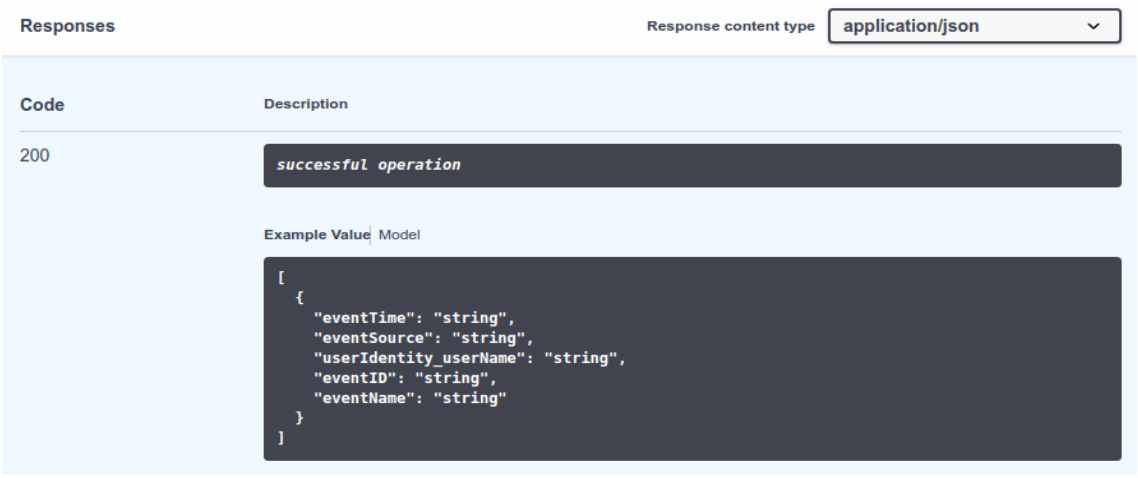



*Figura 36: Resultado de la ruta paratemers de la API.*

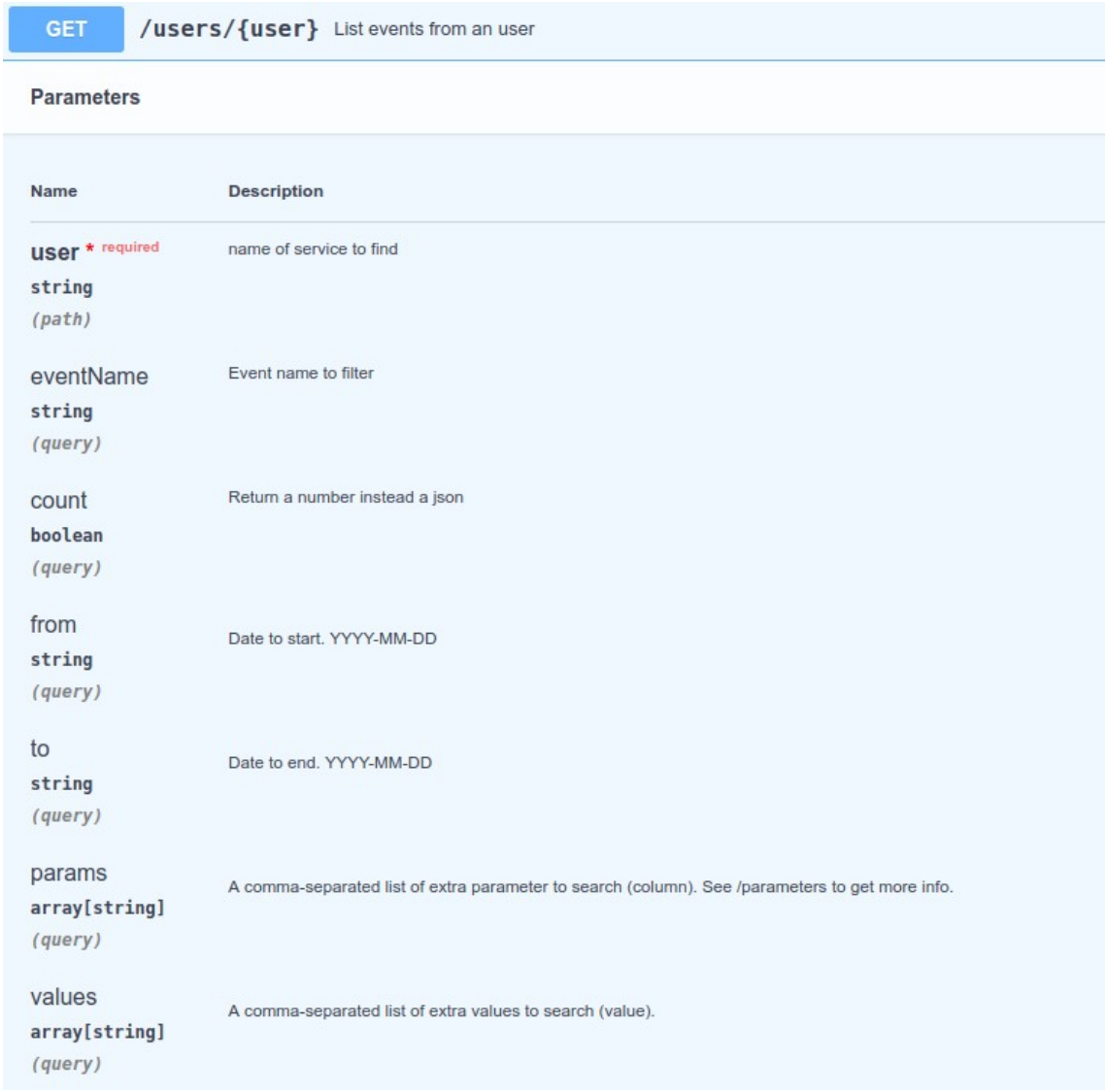



*Figura 37: Ruta users/user de la API.*

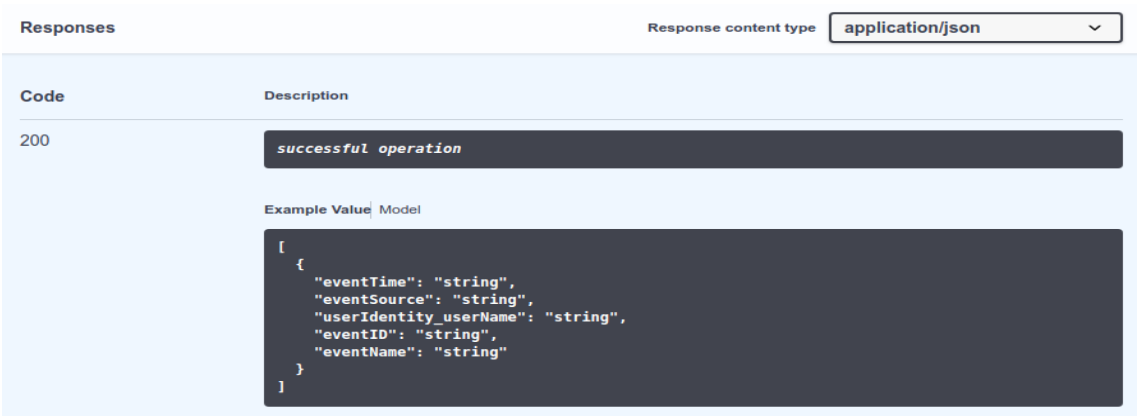



*Figura 38: Resultado de la ruta users/user de la API.*

En la API podemos diferenciar tres métodos importantes para obtener la información, y otros tres métodos que complementan la información. Los tres métodos importantes son scan, services/{service} y users/{user}. Los otros métodos complementarios son *services*, *users* y *parameters*.

En la figura 31 tenemos la ruta para el método scan. Este método nos muestra información general sin filtrar por usuario ni por servicio. Por defecto, la ruta /scan nos devuelve un listado con los eventos registrados por CloudTrail-Tracker en los últimos siete días. En la imagen observamos también los diferentes parámetros que coinciden, en su mayoría, con los parámetros de los siguientes métodos. Con los parámetros podemos filtrar nuestra consulta. Con eventName filtramos por nombre de evento, podemos decidir entre un rango de fechas con *from* y *to*, y podemos añadir parámetros auxiliares con *param* y *value*. Si ponemos el parámetro *count* al valor True, nos devolverá el número de eventos en lugar de un JSON, como veremos en un ejemplo más adelante.

En la figura 32 podemos ver cómo devuelve el resultado el método scan, coincidiendo también con el resultado de los métodos services/{service} y users/{users}. Este resultado será una lista de diccionarios ordenada por tiempo. En la figura 33 tenemos el método /services. Éste nos devuelve una lista con los servicios de los cuales tiene eventos registrados. Cada uno de estos servicios son los que darán resultados en el método services/{service}, siendo el parámetro {service} uno de estos métodos. El método /users, mostrado en la figura 34, funciona de forma similar al método de la ruta /services, devolviendo una lista de usuarios. En la misma figura vemos el formato del resultado.

Los métodos de las rutas services/{service} y users/{user} funcionan de forma similar entre ellos, cambiando el parámetro por un servicio o por un usuario respectivamente, obtenidos en los métodos explicados anteriormente, services y users. Los parámetros opcionales son los mismos que en el método scan. Vemos que el resultado de ambos métodos también es el mismo que el del método scan, una lista de diccionarios ordenados por fecha, formando un resultado en JSON.

## 4.3 Componentes del Front-end

La parte del front-end se ha desarrollado en forma de portal web[2], por parte de de Diana María Naranjo Delgado, investigadora predoctoral del Grupo de Grid y Computación de Altas Prestaciones [33] con la que se ha colaborado para ofrecer el soporte necesario para acceder al API y mantener actualizado el servicio.

Para elaborar el portal web se ha usado Vue.js, un framework sobre Javascript para el desarrollo de interfaces web. Dicha web es completamente serverless y estática, ya que está alojada en un bucket de S3 y se basa en Jenkins para el despliegue de las versiones estables de forma automática. Además, integra Cognito para gestionar el registro de los usuarios de forma automática.

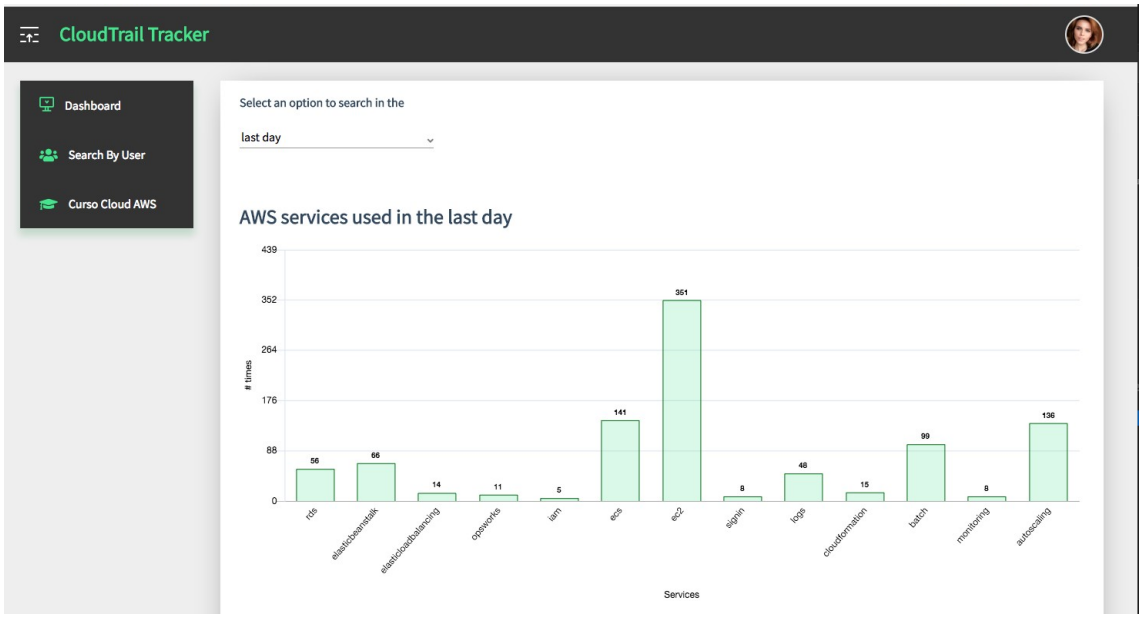



*Figura 39: Listado de los servicios usados en el último día desde la web*

2 http://cloudtrailtracker.cursocloudaws.net/ (accesible con las credenciales demo / demoDem0!)

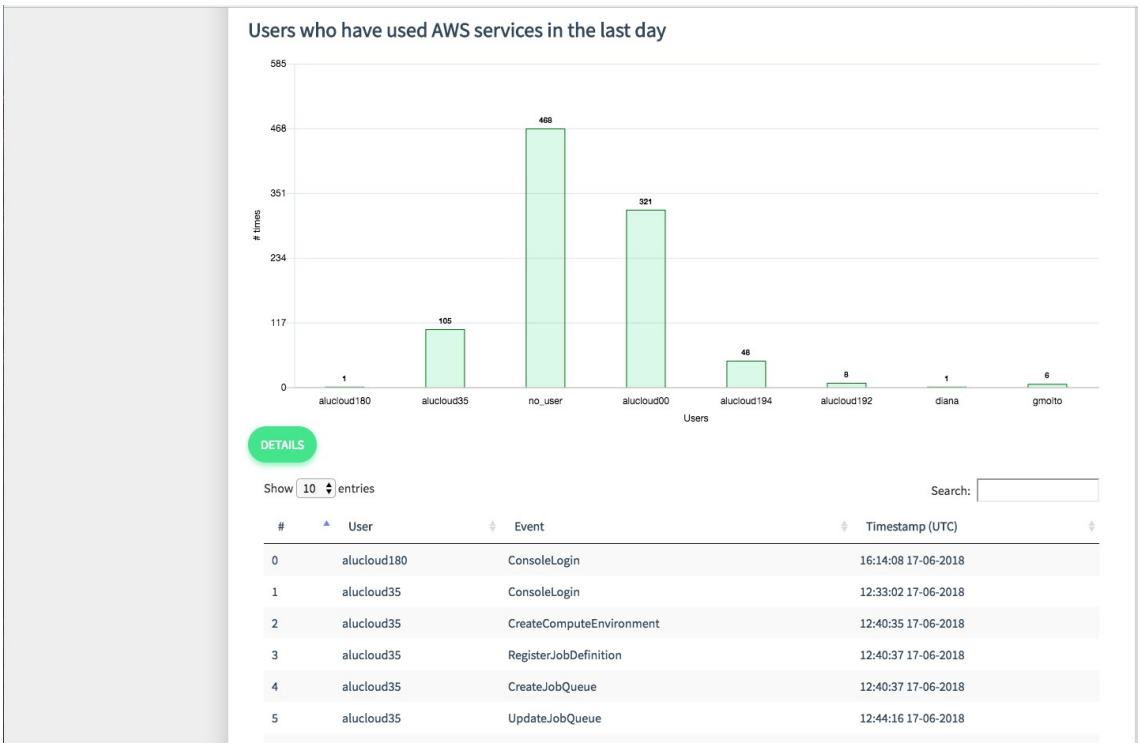



*Figura 40: Usuarios activos en el último día*

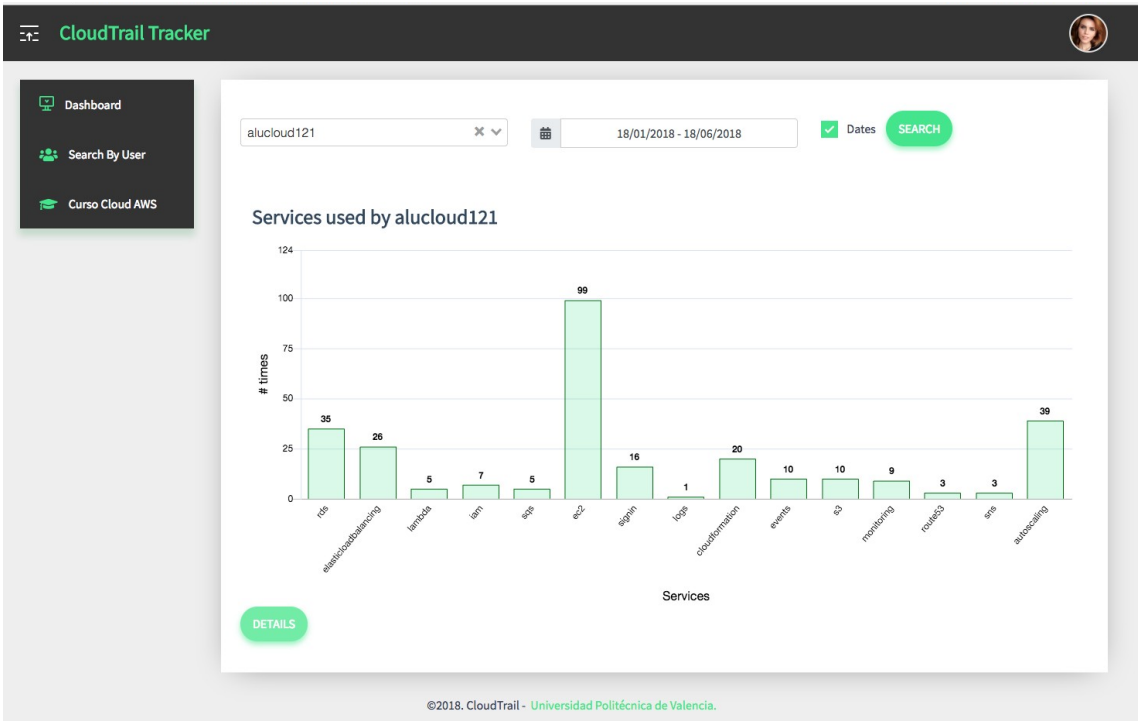



*Figura 41: Servicios usados por el usuario alucloud121 entre dos rangos de fechas*

En las figuras 39, 40 y 41 podemos ver ejemplos de uso del servicio desde la página web. En la figura 39 vemos los servicios usados en el último día. En la figura 40 vemos los usuarios activos en el último día, mostrando debajo del gráfico las acciones de los usuarios con más detalle. Finalmente, en la figura 41 hemos realizado una consulta por un usuario específico entre un rango de fechas.

## 4.4 Dependencias y despliegue

A continuación, mostramos las dependencias necesarias y cómo desplegar todo el servicio.

### 4.4.1 Dependencias

Entre las dependencias del proyecto se encuentran Python 3.6 y pip 3 para poder instalar los paquetes de la forma que se muestra a continuación, aunque hay formas alternativas de hacerlo con otros paquetes de instalación, aquí mostramos una de ellas.

Necesitamos instalar el paquete boto3 junto con el framework serverless, lo podemos hacer de la siguiente forma:

```
~$ npm install -g serverless
~$ pip3 install boto3
```

También necesitamos un bucket con permisos de lectura, ya que serverless creará una carpeta en él y un rol en AWS con los permisos que se muestran en la figura 42.

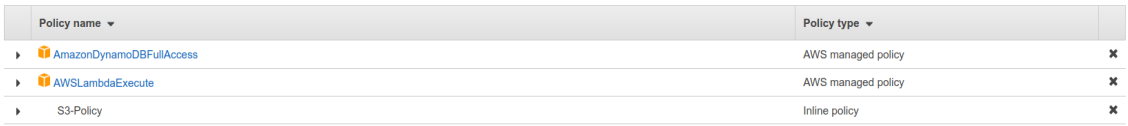



*Figura 42: Rol necesario para completar las dependencias.*

Tanto el bucket como el rol se añadirán en la configuración del archivo serverless.yaml.

Finalmente, clonamos el proyecto desde nuestro repositorio con el siguiente comando:

```
~$ git clone https://github.com/grycap/cloudtrail-tracker
~$ cd cloudtrail-tracker
```

Aclarar que todos los comandos que se ejecuten de aquí en adelante serán desde la carpeta del proyecto clonado.

### 4.4.2 Despliegue

Para el despliegue de todos los servicios hemos desarrollado scripts en boto3 para la parte de la base de datos y el framework Serverless para las funciones Lambda y la API.

El primer paso es configurar desde el archivo settings/settings.py, el cual se ve en la figura 43, los parámetros de nuestros servicios, como pueden ser la región de AWS en la que queramos desplegar el proyecto, los nombres de las funciones Lambda y de la API, los roles, etc.

```
AWS_REGION = 'us-east-1'
#Lambda function name for querying
lambda_func_name = "cloudtrailTrackerQueries"
#Lambda function name for automatic event upload
lambda_func_name_trigger = "cloudtrailTrackerUpload"
#Stage name for API Gateway
stage_name = "cloudtrailtrackerStage"
#DynamoDB Table name
table_name = "cloudtrailtrackerdb"
#Preconfigured S3 bucket by CloudTrail
bucket_name = "cursocloudaws-trail"
#API name
API_name = "cloudtrailTrackerAPI"
#eventNames that we DO NOT want to store - Filter
filterEventNames = ["get", "describe", "list", "info", "decrypt", "checkmfa", "head", "assumerole"]
#Index name for DynamoDB Table - Dont modify if is not necessary
index = 'userIdentity_userName-eventTime-index'
```

*Figura 43: Especificaciones en el archivo settings/settings.py*

Seguidamente creamos la tabla en DynamoDB ejecutando script mostrado en la figura 44.

```
(TFG) jose@jose-Lenovo-ideapad-700-15ISK ~/Documentos/TFG/TFG $ python dynamodb/Database.py
Creating table EventoCloudTrail_TFG
. . .
Table created
Creating row for users, services and parameters list...
. . .
. . .
. . .
Succeeded
Finished
```

*Figura 44: Creación de la tabla en DynamoDB*

Esto creará la tabla correspondiente, insertando el primer elemento con un identificador especial para guardar un diccionario, mostrado en la figura 45, donde en posteriores consultas podremos obtener la lista de usuarios, de servicios y de parámetros disponibles.

| eventID | userIdentity_use | eventTime | cols | listUsers | services |
|---|---|---|---|---|---|
| 1 | all | 1 | { "awsRegion" : { "S" : "1" }, "eventID" : {... | { "acarrion" : { "S" : "1" }, "acm... | { "apigateway" : { "S" : "1" }, "autoscaling" : { "S" : "1" }, "cloudf... |

*Figura 45: Ítem especial con información de los usuarios, eventos y parámetros*

A continuación, creamos las funciones Lambda, junto a la API REST ejecutando el comando con el framework Serverless mostrado en la figura 46.

```
(TFG) jose@jose-Lenovo-ideapad-700-15ISK ~/Documentos/TFG/TFG $ sls deploy
Serverless: Packaging service...
Serverless: Excluding development dependencies...
Serverless: Uploading CloudFormation file to S3...
Serverless: Uploading artifacts...
Serverless: Uploading service .zip file to S3 (582.29 KB)...
Serverless: Validating template...
Serverless: Creating Stack...
Serverless: Checking Stack create progress...
.....
Serverless: Stack create finished...
Service Information
service: cloudtrail-tracker
stage: dev
region: us-east-1
stack: cloudtrail-tracker-dev
api keys:
  None
endpoints:
  GET - https://a49m410egc.execute-api.us-east-1.amazonaws.com/dev/scan
  GET - https://a49m410egc.execute-api.us-east-1.amazonaws.com/dev/users
  GET - https://a49m410egc.execute-api.us-east-1.amazonaws.com/dev/users/{user}
  GET - https://a49m410egc.execute-api.us-east-1.amazonaws.com/dev/services
  GET - https://a49m410egc.execute-api.us-east-1.amazonaws.com/dev/parameters
  GET - https://a49m410egc.execute-api.us-east-1.amazonaws.com/dev/services/{service}
functions:
  EventUploads: cloudtrail-tracker-dev-EventUploads
  Query: cloudtrail-tracker-dev-Query
```

*Figura 46: Desplegando con serverless*

1. Serverless habrá hecho los siguientes pasos por nosotros, a partir de la plantilla en formato YAML:
2. Comprimir todo lo necesario para que el proyecto funcione, como el código y dependencias.
3. Crear una plantilla en CloudFormation.
4. Subir la compresión y la plantilla al bucket S3, creando una carpeta llamada serverless en éste.
5. Ejecutar la plantilla desde CloudFormation.

También nos muestra los puntos de acceso a la API y las funciones Lambda creadas.

El framework Serverless no permite modificar la infraestructura actual, por lo cual desde la configuración del propio archivo YAML no permite la creación de un nuevo bucket, sino que obliga a utilizar uno ya existente. Por esta razón hemos creado un bucket previamente y, además, hemos configurado CloudTrail para que use dicho bucket, almacenando los eventos en éste.

```
(TFG) jose@jose-Lenovo-ideapad-700-15ISK ~/Documentos/TFG/TFG $ python lambda/eventuploads/trigger.py
Creating trigger S3 -> Lambda . . . Done!
```

*Figura 47: Creando disparador entre bucket S3 y función Lambda*

Algo que tampoco es posible desde el framework es agregar un disparador al bucket, por lo que con el siguiente script agregamos el disparador que se lanzará cuando se suba un fichero acabado en *'.json.gz'* , que llamará a una función Lambda con el nombre del fichero como parámetro. Esto se puede ver en la figura 47.

### 4.4.3 Ejemplos de uso

A continuación, mostramos algunos ejemplos de uso del servicio.
Los ejemplos se listan utilizando curl, y en el caso de los usuarios se mostrará con curl y desde la interfaz web.
En cada caso deberemos cambiar {ENDPOINT} por el endpoint que nos habrá devuelto serverless en su ejecución y {STAGE} por el "stage" configurado en serverless.yaml .

Cabe destacar que los resultados siempre tendrán la misma forma, una lista de diccionarios ordenada por fecha. Cada diccionario contiene lo siguiente:

- **eventID**: identificador único de cada evento
- **eventName**: nombre del evento en cuestión
- **eventSource**: origen del evento, refiriendo a un servicio como EC2, S3, etc.
- **eventTime**: fecha del evento.
- **userIdentity_userName**: nombre de usuario del evento.

En la figura 48 vemos un ejemplo de consulta, donde se muestran dos eventos con dicha información.


```
[
  ...
{
    "eventID": "6bcee565-5cf2-4f3e-bef9-6b250b091a80",
    "eventName": "CreateDBSubnetGroup",
    "eventSource": "rds.amazonaws.com",
    "eventTime": "2018-06-11T15:42:21Z",
    "userIdentity_userName": "alucloud131"
  },
  {
    "eventID": "d4f51839-b19d-46b3-b370-1b7833492aca",
    "eventName": "DeleteDBInstance",
    "eventSource": "rds.amazonaws.com",
    "eventTime": "2018-06-04T05:10:25Z",
    "userIdentity_userName": "alucloud139"
  },
  ...
]
```


*Figura 48: Resultado de ejemplo mostrando dos eventos..*

En caso de querer listar los eventos sobre los últimos siete días, basta con usar el método "scan".

```
~$ curl {ENDPOINT}/{STAGE}/scan
```

Si quisiéramos añadir parámetros a la consulta, a parte de los propios parámetros que ofrece la API, podemos consultar la lista de parámetros con:

```
~$ curl {ENDPOINT}/{STAGE}/parameters
```

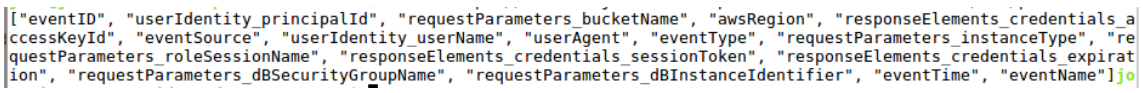



*Figura 49: Resultado del método "parameters"*

Cada elemento de la lista devuelta por esta consulta se puede usar como valor en el parámetro "*param*".
Un ejemplo del método "scan" con un rango de fechas y usando el parámetro "*params*" sería:

```
~$curl      {ENDPOINT}/{STAGE}/scan?from=2018-01-01&to=2018-05-
01&param=['awsRegion']&value=['us-east-1']
```

En caso de querer listar por servicios, podemos listar, por defecto la actividad de los últimos siete días, aunque también disponemos de diferentes parámetros, como en el caso del método "scan":

```
~$ curl {ENDPOINT}/{STAGE}/services/ec2
```

También podemos listar los servicios disponibles con la siguiente ruta:

```
~$ curl {ENDPOINT}/{STAGE}/services
```

Para listar información sobre un solo usuario, lo hacemos de la siguiente forma:

```
~$ curl {ENDPOINT}/{STAGE}/users/alucloud230
```

Esta última ruta mostrará los últimos siete días de actividad del usuario alucloud230. Podemos añadir diferentes parámetros de la misma forma que en los otros métodos. Un ejemplo sería:

```
~$curl      {ENDPOINT}/{STAGE}/users/alucloud230?from=2018-01-
01&to=2018-05-01&param=['awsRegion']&value=['us-east-
1']&eventName=CreateResource
```

De esta forma preguntamos qué instancia de EC2 ha desplegado el usuario *alucloud230* en la región *"us-east-1"* entre el 2018-01-01 y el 2018-05-01. Si queremos saber sólo cuántos eventos hay, bastaría añadir el parámetro "count" de la siguiente forma, devolviendo solo un número:

```
~$curl        {ENDPOINT}/{STAGE}/users/alucloud230?from=2017-01-
01&to=2018-01-
01&eventName=RunInstances&param=['awsRegion']&value=['us-east-
1']&count=True
```

Dicha consulta devuelve 29, que son el número de acciones realizadas entre esas dos fechas con los filtros aplicados.

Para listar información sobre los usuarios también podemos hacerlo desde la página web.

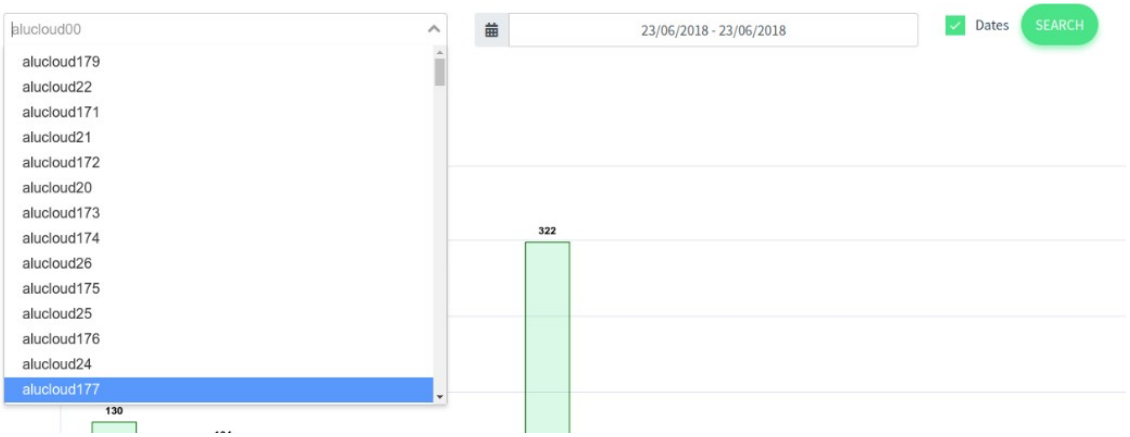



*Figura 50: Web de CloudTrail-Tracker listando los usuarios.*

Como vemos en a la figura 50, podemos seleccionar un usuario entre los existentes y nos muestra los eventos ocurridos entre un rango de fechas previamente seleccionado.

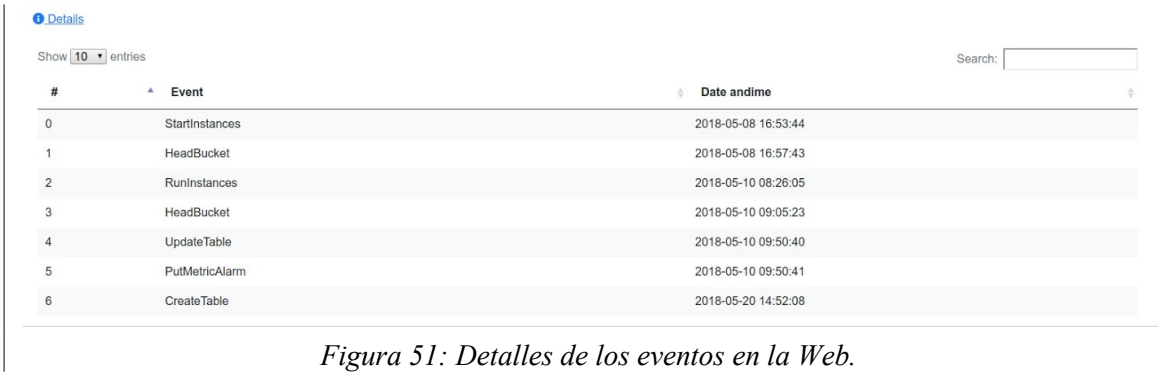



*Figura 51: Detalles de los eventos en la Web.*

También podemos ver los detalles de dichos eventos y, si fuera necesario, buscar entre ellos, tal y como se ve en la figura 51

# 5. Rendimiento y casos de uso

A continuación, exponemos algunos casos de uso en los que este servicio ya se está utilizando o se utilizará. Seguidamente analizamos el coste económico y rendimiento del servicio, realizando carga de datos y consultas, analizando los tiempos de respuesta y el coste.

## 5.1 Asignaturas de Cloud

Este servicio actualmente se usa en tres asignaturas de tres másters diferentes de la Universidad Politécnica de Valencia:

- Infraestructuras de Cloud Público (ICP) del Máster Universitario en Computación Paralela y Distribuida.
- Computación y Gestión de Datos en la Nube para Big Data del Máster en Big Data Analytics.
- Servicios en la Nube (SEN) del Máster Universitario en Gestión de la Información.

Como hemos mencionado en el impacto esperado, también se está utilizando en el Curso Online de Cloud Computing con Amazon Web Services (AWS) del GRyCAP, un curso de formación online asíncrona, de pago y ofertado públicamente, que desde Julio de 2013 ha formado a más de 700 personas.

La principal ventaja de integrar una herramienta de estas características consiste en que el profesor disponga de un panel web centralizado en el que puede acceder a información sobre el grado de utilización de los diferentes servicios de AWS, que es uno de los aspectos más relevantes de las prácticas. El objetivo será evolucionar el portal web para ofrecer una visualización personalizada por alumno en función de las diferentes prácticas de la asignatura, para asistir al proceso de evaluación de la parte práctica, que requiere que los alumnos hayan utilizado los diferentes servicios de AWS.

## 5.2 Coste y rendimiento

En este subapartado analizamos el rendimiento y el coste de los diferentes servicios usados en el proyecto.

### 5.2.1 Carga histórica de datos

Para poder realizar este análisis se dispone de cuatro años de eventos históricos de uso de la cuenta de AWS por el GRyCAP, almacenados por CloudTrail en un bucket de S3.
Hemos necesitado una máquina en EC2 para ejecutar el script. la cual no tiene por qué ser muy potente, ya que no hay una excesiva carga de trabajo.

La máquina EC2 ha tardado 20h en cargar un total de 700.000 eventos, resultado de haber filtrado un número mucho mayor de eventos. Esto se podría haber acelerado subiendo las unidades de escritura de la tabla de DynamoDB, pero se han dejado a 20 unidades, siendo suficiente para la mayoría de los casos. Es importante destacar que DynamoDB ofrece una capa de uso gratuito que incluye 25 unidades de lectura y 25 unidades de escritura. Si no se superan estos valores, el coste derivado de uso de DynamoDB para soportar esta aplicación es cero.

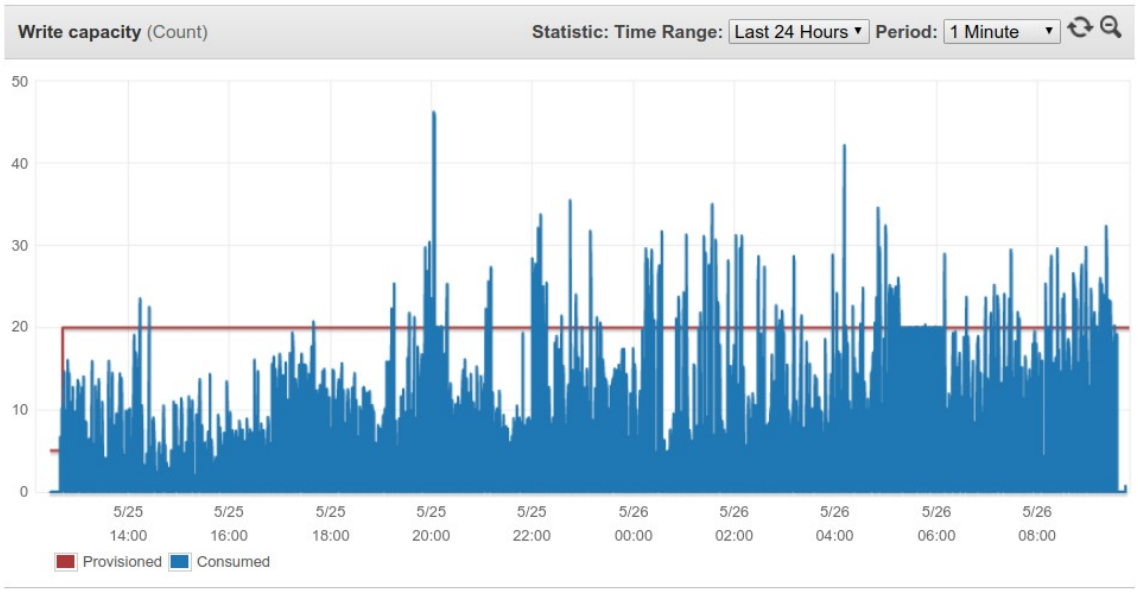



*Figura 52: Unidades de escritura usadas en DynamoDB por la clave principal*

Como se puede ver en la figura 52, en algunos momentos se sobrepasaba el límite de unidades de escritura, pero esto no supone un problema mientras no sobrepase este límite durante varios minutos seguidos, ya que entonces DynamoDB devuelve un error.

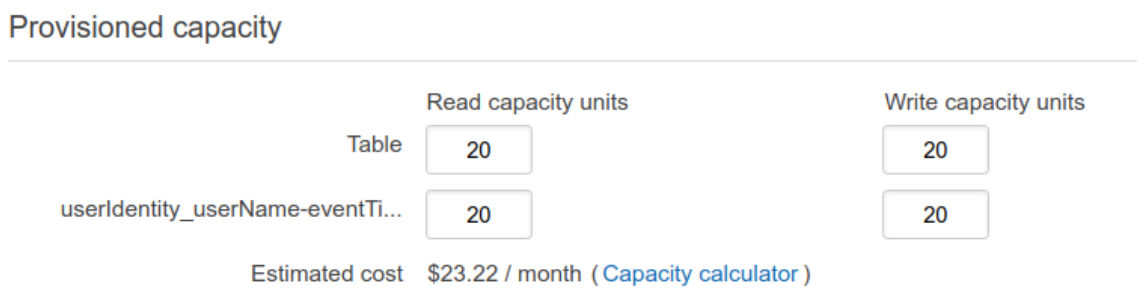



*Figura 53: Unidades de lectura y su coste*

En la figura 53 se puede ver el coste mensual de tener 20 unidades de lectura por cada clave. Estas unidades entran dentro de la capa gratuita de DynamoDB al no superar las 25 unidades, como hemos descrito en el apartado del servicio.

Table details

| | |
|---|---|
| **Table name** | cloudtrailtrackerdb |
| **Primary partition key** | eventID (String) |
| **Primary sort key** | userIdentity_userName (String) |
| **Point-in-time recovery** | DISABLED **Enable** |
| **Encryption** | DISABLED |
| **Time to live attribute** | DISABLED **Manage TTL** |
| **Table status** | Active |
| **Creation date** | May 25, 2018 at 2:26:17 PM UTC+2 |
| **Provisioned read capacity units** | 20 (Auto Scaling Disabled) |
| **Provisioned write capacity units** | 20 (Auto Scaling Disabled) |
| **Last decrease time** | June 8, 2018 at 10:12:35 AM UTC+2 |
| **Last increase time** | June 7, 2018 at 9:09:13 PM UTC+2 |
| **Storage size (in bytes)** | 214.20 MB |
| **Item count** | 698,703 |
| **Region** | US East (N. Virginia) |
| **Amazon Resource Name (ARN)** | arn:aws:dynamodb:us-east-1:974349055189:table/cloudtrailtrackerdb |

*Figura 54: Resultado al cargar los eventos a la Tabla*

En la figura 55 se puede ver el resultado de subir los eventos de 4 años, aplicando los filtros explicados en el apartado del back-end, han sido 214 MBytes y un total de casi 700.000 eventos.

### 5.2.2 Carga de eventos periódica

Cada vez que se generan eventos en nuestra cuenta de AWS, se agrupan y se cargan en un bucket de S3. Por cada archivo que se carga en este bucket se ejecuta una función Lambda, la cual tiene un coste dependiendo de la memoria asignada y el tiempo de ejecución.

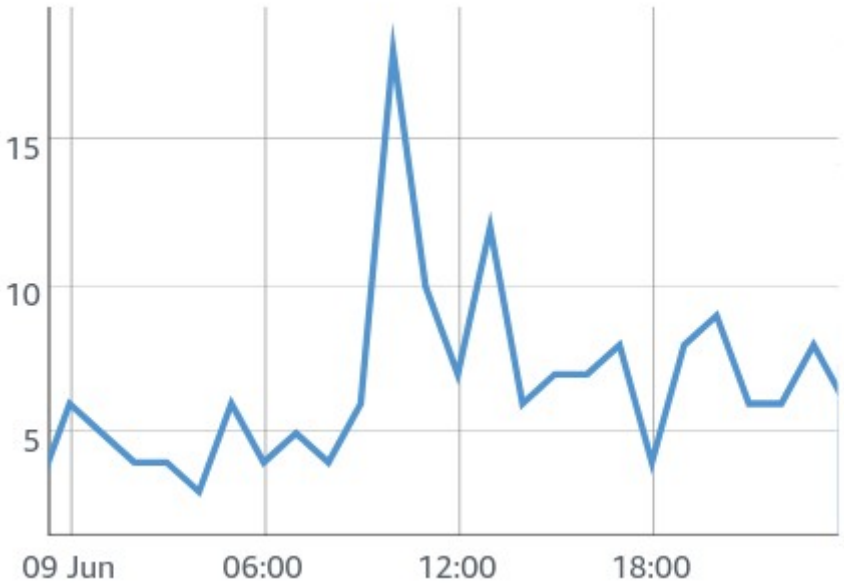



*Figura 55: Número de invocaciones de la función lambda de carga de datos en el tiempo*

En la figura 55 podemos ver el número de veces que se ha ejecutado dicha función Lambda, con un mínimo de 3 invocaciones y un máximo de 18 invocaciones, a las 10:00h de la mañana en un día entre semana. Hay un total de 172 eventos al día, de media.

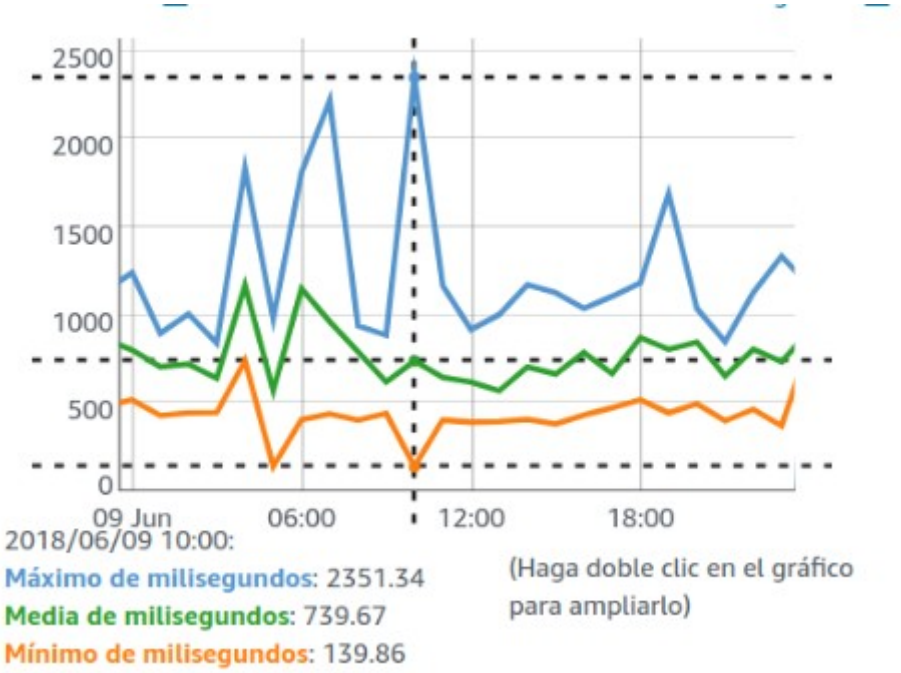



*Figura 56: Duración (ms) de las invocaciones de la función lambda de carga*

Como vemos en la figura 56, la duración media, aunque depende del momento del día, es de 739ms, el mínimo es de 139ms y la duración máxima es de 2351ms. Estos tiempos son la suma de las invocaciones por hora, siendo, a las 10:00h de la mañana, el pico de actividad de la cuenta.

Estos picos de duración surgen debido al inicio en frío (*Cold Start*) de la función. "Un inicio en frío ocurre cuando una función Lambda es invocada después de no ser usada por un largo periodo de tiempo, resultando en una invocación con una latencia más alta." *[37]*

El tiempo de enfriamiento ronda entre los 45 y 60 minutos, según [37], después de la primera ejecución, los tiempos disminuyen notablemente, y con ello el coste económico.

### 5.2.3 Consultas

Para elaborar el análisis de las consultas hemos realizado consultas sobre un usuario activo, y para el rendimiento de las consultas sobre servicios lo haremos con dos servicios, uno más usado que otro, para ver la diferencia de tiempo entre las diferentes consultas.

### *Scan*

Empezamos con la consulta básica "scan", una consulta general en los últimos siete días. Hay que tener en cuenta que la consulta se está ejecutando desde un equipo fuera de AWS por lo que la latencia de acceso a la red también influye, aunque en menor medida, en el tiempo de obtención de la respuesta.

```
jose@jose-Lenovo-ideapad-700-15ISK ~ $ time curl https://dme1mwt5je.execute-api.us-east-1.amazonaws.com/dev/scan > res
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  467k  100  467k    0     0  51817      0  0:00:09  0:00:09 --:--:--  110k

real    0m9.249s
user    0m0.016s
sys     0m0.004s
```

*Figura 57: Consulta desde curl*

El tiempo real para el usuario utilizando curl ha sido de 9,249 segundos. Es importante destacar que el tiempo de ejecución de dicha función no será constante, pues depende principalmente del número de datos a devolver por la consulta que, a su vez, está directamente relacionado con la actividad que haya habido en la cuenta de AWS por parte de sus usuarios.

```
▸ 07:48:29  START RequestId: d46b5a5b-6d4b-11e8-bc34-39af293ae750 Version: $LATEST
▸ 07:48:29  2018-06-11T00:00:00Z
▸ 07:48:29  <class 'str'>
▸ 07:48:33  END RequestId: d46b5a5b-6d4b-11e8-bc34-39af293ae750
▾ 07:48:33  REPORT RequestId: d46b5a5b-6d4b-11e8-bc34-39af293ae750 Duration: 3695.96 ms Billed Duration: 3700 ms Memory Size: 512 MB Max Memory Used: 44 MB

REPORT RequestId: d46b5a5b-6d4b-11e8-bc34-39af293ae750 Duration: 3695.96 ms Billed Duration: 3700 ms Memory Size: 512 MB Max Memory Used: 44 MB
```

*Figura 58: Tiempo de la función Lambda sobre la consulta "scan" para los últimos siete días*

En la figura 58 se puede ver la información que nos proporciona CloudWatch sobre la función Lambda. La consulta ha durado 3695 milisegundos, lo que se redondea a 3700 milisegundos para el coste. En cuanto a la tabla en DynamoDB, ha consumido 0.1166 unidades de lectura, lo cual no supone un coste adicional.

También se ha probado a realizar una consulta con "scan" y un rango de fechas más elevado, de 6 meses, como se ve en la siguiente imagen.

```
jose@jose-Lenovo-ideapad-700-15ISK ~ $ time curl 'https://dme1mwt5je.execute-api.us-east-1.amazonaws.com/dev/scan
?from=2017-06-01&to=2018-01-01' > /dev/null
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100    41  100    41    0     0      1      0  0:00:41  0:00:29  0:00:12    10

real    0m29.240s
user    0m0.016s
sys     0m0.008s
```

*Figura 59: Consulta de seis meses con "scan"*

La consulta ha durado 29,2 segundos de cara al usuario. A continuación, lo vemos desde la perspectiva de la tabla en DynamoDB.

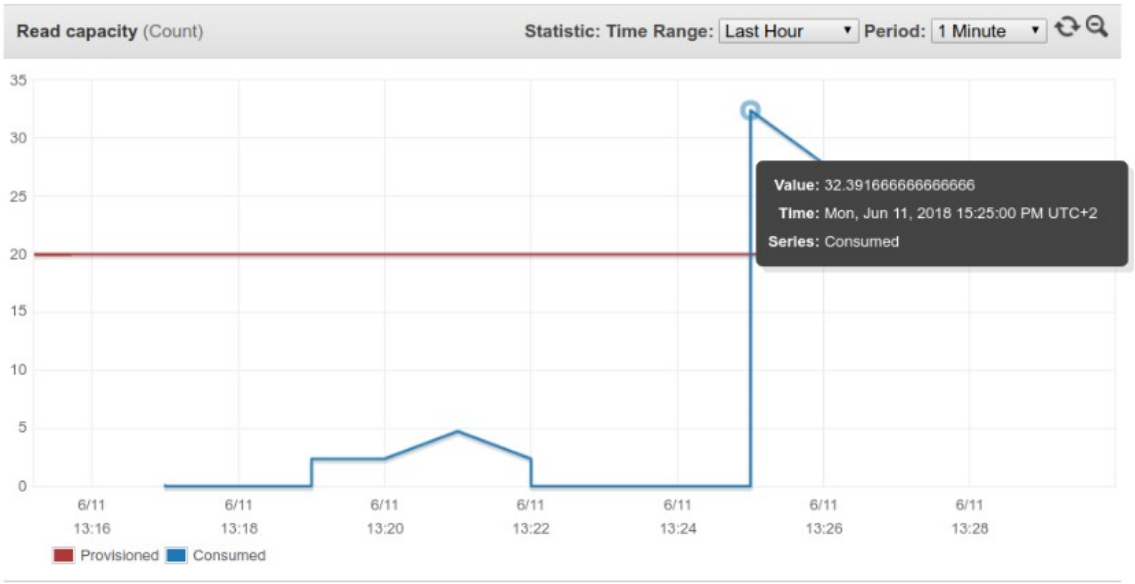



*Figura 60: Imagen de las unidades de lectura de la taba en el método "scan" de seis meses.*

En la figura 60 vemos que el consumo llega a 32 unidades de lectura, sobrepasando las 20 que había configuradas. Esto crea un cuello de botella, solucionable aumentando el número de unidades de lectura de la tabla de forma manual o con el auto-escalado.
Si se superan las unidades de lectura por varios minutos, DynamoDB devolverá un error, impidiendo dicha lectura. Por otro lado, podemos activar el auto-escalado, lo que hará que DynamoDB aumente o disminuya el número de unidades tanto de lectura como de escritura, lo que garantizará unos tiempos de respuesta menores en caso de superarse las unidades, o incluso un menor consumo al disminuir las unidades, si el servicio lo viera necesario.

### *Usuarios*

Para el caso de los usuarios se han hecho dos consultas sobre el usuario alucloud230. Las consultas específicas sobre usuarios son más rápidas, ya que se consultan sobre una clave específica sobre ellos.

```
jose@jose-Lenovo-ideapad-700-15ISK ~/Documentos/TFG $ time curl 'https://api.cursocloudaws
.net/tracker/users/alucloud230?from=2017-08-01&to=2017-09-01' > /dev/null
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 36177  100 36177    0     0  57723      0 --:--:-- --:--:-- --:--:-- 58350

real    0m0.640s
user    0m0.027s
sys     0m0.011s
```

*Figura 61: Consulta sobre el usuario alucloud230 en julio de 2017*

La primera de ellas, en la figura 61, es una consulta sobre todo un mes de uso del usuario. Se observa que la consulta ha durado 0,64 segundos.

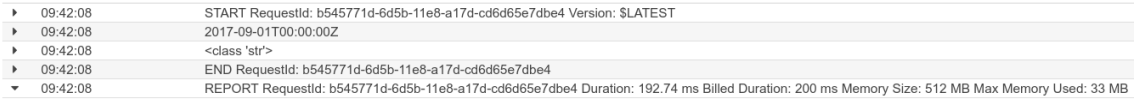



*Figura 62: Detalles de la consulta en CloudWatch*

La duración real de la consulta con la función lambda es de 200 milisegundos, como nos muestra CloudWatch en la figura 62.

Si hacemos una consulta insertando más filtros, como se ve en la figura 63, no aumenta el tiempo.

```
jose@jose-Lenovo-ideapad-700-15ISK ~ $ time curl 'https://dme1mwt5je.execute-api.us-east-1.amazonaws.com/dev/users
/alucloud230?from=2017-08-01&to=2017-09-01&params=eventSource&values=ec2' > /dev/null
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 36177  100 36177    0     0  74795      0 --:--:-- --:--:-- --:--:-- 81296

real    0m0.493s
user    0m0.016s
sys     0m0.000s
```

*Figura 63: Consulta sobre el usuario alucloud230 con filtros.*

Hemos probado, una vez más, a realizar una consulta pesada. Esta vez, tal y como se observa en las dos siguientes imágenes, hemos aumentado el tiempo a 3 años en el rango de tiempo.

```
jose@jose-Lenovo-ideapad-700-15ISK ~ $ time curl 'https://dme1mwt5je.execute-api.us-east-1.amazonaws.com/dev/user
s/alucloud230?from=2015-01-01&to=2018-01-01' > /dev/null
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  316k  100  316k    0     0   314k      0  0:00:01  0:00:01 --:--:--  315k

real    0m1.012s
user    0m0.020s
sys     0m0.000s
```

*Figura 64: Consulta sobre el usuario alucloud230 en un rango de 3 años.*

Como vemos, el tiempo de la consulta ha sido de alrededor de 1 segundo de cara al usuario, aun cuando el rango de búsqueda ha aumentado.
Las consultas directas sobre usuarios son casi instantáneas ya que, en la tabla, disponemos de un índice sobre los usuarios.

### *Servicios*

Para evaluar las consultas sobre los servicios también se han hecho tres consultas.

En la primera, una consulta con parámetros sobre un servicio que suele ser más usado, EC2.

```
jose@jose-Lenovo-ideapad-700-15ISK ~ $ time curl 'https://dme1mwt5je.execute-api.us-east-1.amazonaws.com/dev/services
/ec2?from=2018-03-01&to=2018-04-01&eventName=RunInstances' > /dev/null
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 92025  100 92025    0     0  21437      0  0:00:04  0:00:04 --:--:-- 21451

real    0m4.298s
user    0m0.008s
sys     0m0.004s
```

*Figura 65: Consulta en un rango de fechas de un mes sobre EC2*

En la consulta de la figura 65 se piden las instancias de EC2 desplegadas en un mes. La consulta ha tardado 4,3 segundos en realizarse.


*Figura 66: Detalles de la consulta sobre EC2 en CloudWatch.*

Vemos que realmente la duración de la función Lambda es de 3100 milisegundos.

Al realizar la consulta sobre otro servicio similar, como S3, nos salen resultados similares, como se puede ver en la siguiente imagen.

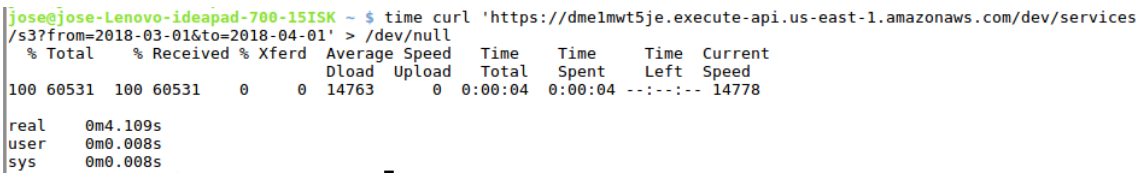



*Figura 67: Consulta sobre S3.*

Hemos realizado una consulta más, involucrando mayor rango de tiempo en EC2, para probar de poner el servicio al límite.
La consulta, como se puede ver en la figura 68, es sobre dos años de uso, y ha necesitado 29 segundos para poder mostrarse.

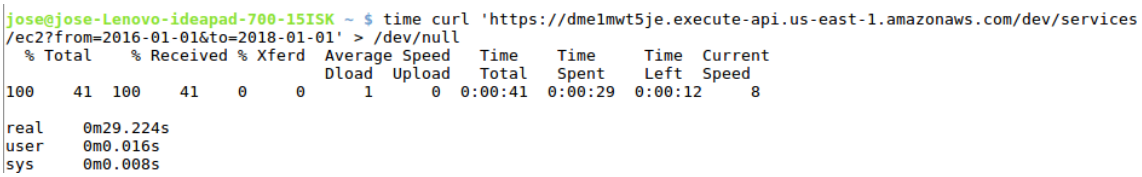



*Figura 68: Consulta de 2 años sobre EC2*

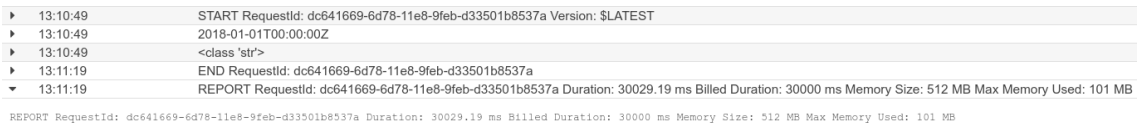



*Figura 69: Datos de CloudWatch sobre la consulta*

Como vemos en los logs de CloudWatch, la consulta ha durado 30 segundos, se ha ajustado al límite que tiene API Gateway, en caso de haber tardado un segundo más, API Gateway habría devuelto un error. Esa consulta tiene un coste más alto.

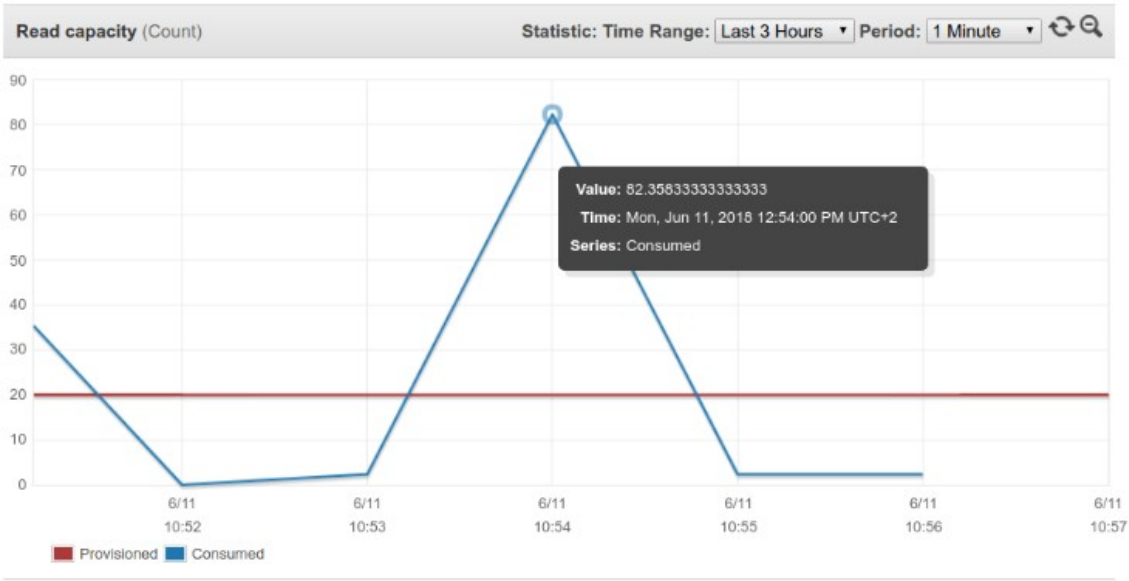



*Figura 70: Unidades de lectura usadas en DynamoDB con servicios durante dos años*

En la figura 70 podemos ver que el problema de que la consulta casi exceda el tiempo proviene de la cantidad de unidades de lectura asignadas, siendo 20 unidades las asignadas y 82 las usadas. Esto provoca, otra vez un cuello de botella y se puede plantear el subirlas o dejar el auto-escalado que ofrece AWS. En todo caso, para consultas más ligeras no hace falta tenerlo muy alto, pero si tenemos en mente realizar consultas más pesadas conviene subir este número.

# 6. Conclusiones y trabajos futuros

## 6.1 Conclusiones

El objetivo principal de este proyecto era diseñar e implementar una plataforma de recopilación y procesado de eventos que generen cambios en la infraestructura en una cuenta de AWS por parte de los múltiples usuarios IAM vinculados a dicha cuenta. Para ello habíamos marcado una serie de objetivos, sobre los cuales podemos sacar conclusiones.

El primer objetivo planteado, usar servicios FaaS (*Functions as a Service*), se ha cumplido al diseñar e implementar la solución con el framework serverless y completamente en la nube, automatizando el proceso de despliegue del sistema, facilitando así la adopción por parte de otros usuarios y abaratando los costes

El segundo objetivo planteado ha sido ofrecer un mecanismo de consulta para poder obtener los eventos realizados por un usuario o servicio entre dos fechas. Dado que en la API hemos parametrizado las fechas, incluyendo una granularidad fina de horas, minutos y segundos, podemos dar este objetivo por cumplido.

El uso de una *API REST* bien documentada con Swagger, siguiendo unas buenas prácticas, permite tener una forma sencilla de consulta de los eventos. Al documentar paso por paso el despliegue de todo el sistema, podemos dar por satisfechos el tercer objetivo.

Otro objetivo era el optimizar las consultas al máximo posible. Dado que hemos eliminado un gran número de eventos que, al no generar cambios en la infraestructura, no eran útiles, se han conseguido reducir los tiempos de consulta sin tener que aumentar las unidades de lectura, y por ende, el coste.

Dado que el proyecto está disponible públicamente en el repositorio del GRyCAP y con licencia de código abierto, cumplimos también con el quinto y último objetivo marcado.

Como reflexión general, estamos convencidos que el desarrollo de una aplicación de estas características tiene utilidad para cualquier usuario de AWS ya que permite visualizar información sobre el uso de los servicios en AWS. Haber implementado una solución completamente serverless donde tanto el back-end como el front-end funcionan sin necesidad de operar ninguna instancia de EC2, es decir, ninguna máquina virtual, y que opera en coste prácticamente cero, constituye un patrón de diseño de aplicación en la nube que dominará gran parte de las aplicaciones futuras en la nube.



Finalmente, mencionar que durante el grado en Ingeniería Informática no se han tratado las tecnologías en la nube. Este proyecto ha permitido aprender y profundizar en el uso de estas tecnologías, además de contribuir a la comunidad de AWS con una herramienta de código libre.

## 6.2 Trabajos futuros

Como mejoras futuras se presentan las siguientes opciones:

- Estudiar la posibilidad de añadir otro índice en la tabla de DynamoDB para los servicios.
- Uso de dos tablas de DynamoDB, una para usuarios y otra para servicios, aumentando el coste y la complejidad de las consultas, pero consiguiendo una reducción instantánea de las consultas a lo servicios.
- Añadir autenticación (*Cognito*) en el despliegue del servicio.
- Añadir paginación a la API.
- Crear un sistema de alertas sobre eventos.
- Estudiar la posibilidad de aplicar técnicas de Machine Learning para detectar comportamientos anómalos.
- La utilización de herramientas de plantillas para la configuración.

## 6.3 Relación del trabajo desarrollado con los estudios cursado

En el Grado en Ingeniería Informática no hemos dado materias relacionadas con las tecnologías utilizadas en este proyecto, por lo que dentro del plan del tiempo del proyecto hemos tenido una curva de aprendizaje sobre dichas tecnologías.

No obstante, en el Grado hemos adquirido conocimientos más generales y cierta base, lo cual posteriormente nos permite aprender fácilmente otras tecnologías o adoptar otros paradigmas de forma rápida.

Cabe mencionar algunas competencias desarrolladas en el Grado, las cuales se han requerido y puesto en práctica en este proyecto. Al estar en un nuevo entorno sin experiencia en él, se ha requerido de cierto análisis y saber de resolución de problemas, resolviendo problemas con procedimientos estructurados cuando se ha necesitado, así como la aplicación y pensamiento práctico, permitiendo adaptarnos al nuevo entorno. Dado que el trabajo se ha basado en el diseño de un nuevo servicio, se ha promovido el trabajo autónomo.

Finalmente, se ha usado instrumental específica al buscar soluciones concretas basadas en la nube y se ha favorecido al aprendizaje permanente aprendiendo a usar dicho instrumental.

## 6.4 Difusión

Se ha presentado CloudTrail-Tracker como póster de material docente en la jornada de innovación docente (JIDINF'18 [34]). Dicho poster fue seleccionado para participar en la mesa redonda final y realizar una presentación del mismo a la audiencia. Se pretende presentar una

contribución en SIGCSE 2019, un congreso sobre educación informática organizado por ACM SIGCSE, en Minnesota. Este congreso atrae a más de 1500 investigadores y profesores. [35]
Este proyecto está publicado en el repositorio del GRyCAP: https://github.com/grycap/cloudtrail-tracker .

## 6.5 Disponibilidad y Licencia

CloudTrail-Tracker, disponible en https://github.com/grycap/cloudtrail-tracker, es un proyecto de código abierto bajo la licencia Apache 2.0 [36], lo cual permite uso comercial, modificaciones, distribución y uso privado. Los derechos de autor deben conservarse tanto en el código fuente como en los binarios.

# 7. Referencias

[1] Amazon, "Informática en la nube. Ventajas y Beneficios." [Online]. Available: https://aws.amazon.com/es/what-is-cloud-computing/. [Accessed: 25-Apr-2018].

[2] Joy Tan, "Cloud Computing Is Crucial To The Future Of Our Societies -- Here's Why." [Online]. Available: https://www.forbes.com/sites/joytan/2018/02/25/cloud-computing-is-the-foundation-of-tomorrows-intelligent-world/#3bdae34a4073. [Accessed: 25-Apr-2018].

[3] GRyCAP, "Curso Online de Cloud Computing con Amazon Web Services (AWS)." [Online]. Available: http://www.grycap.upv.es/cursocloudaws/index.php. [Accessed: 02-May-2018].

[4] Bob Evans, "The Top 5 Cloud-Computing Vendors: #1 Microsoft, #2 Amazon, #3 IBM, #4 Salesforce, #5 SAP." [Online]. Available: https://www.forbes.com/sites/bobevans1/2017/11/07/the-top-5-cloud-computing-vendors-1-microsoft-2-amazon-3-ibm-4-salesforce-5-sap/#fbc712c6f2eb. [Accessed: 03-May-2018].

[5] Microsoft, "Microsoft Azure: plataforma y servicios de informática en la nube." [Online]. Available: https://azure.microsoft.com/es-es/. [Accessed: 02-Jul-2018].

[6] Microsoft Azure, "Regiones de Azure." [Online]. Available: https://azure.microsoft.com/es-es/global-infrastructure/regions/. [Accessed: 05-May-2018].

[7] Google, "Cloud Computing, servicios de alojamiento y APIs de Google Cloud." [Online]. Available: https://cloud.google.com/. [Accessed: 05-May-2018].

[8] Amazon, "Amazon Web Services Simple Monthly Calculator." [Online]. Available: https://calculator.s3.amazonaws.com/index.html. [Accessed: 10-May-2018].

[9] Stack Overflow, "Stack Overflow Trends." [Online]. Available: https://insights.stackoverflow.com/trends?tags=azure%2Camazon-web-services%2Cgoogle-cloud-platform. [Accessed: 12-May-2018].

[10] Bob Evans, "Amazon To Become #1 In Cloud-Computing Revenue By Beating IBM's $17 Billion." [Online]. Available: https://www.forbes.com/sites/bobevans1/2018/01/26/amazon-to-become-1-in-cloud-computing-revenue-by-beating-ibms-17-billion/#183f8e7d6b3e. [Accessed: 15-May-2018].

[11] Amazon, "Cloud Computing - Servicios de informática en la nube." [Online]. Available: https://aws.amazon.com/es/. [Accessed: 16-May-2018].

[12] Amazon, "AWS CloudTrail." [Online]. Available: https://aws.amazon.com/es/cloudtrail/. [Accessed: 20-May-2018].

[13] Amazon, "AWS CloudWatch – Servicios de monitorización de la red y la nube." [Online]. Available: https://aws.amazon.com/es/cloudwatch/. [Accessed: 21-May-2018].

[14] Alert Logic, “Security-as-a-Service - Cyber Security Solutions - Cloud Security Solutions.” [Online]. Available: https://www.alertlogic.com/solutions. [Accessed: 21-May-2018].

[15] Amazon, “AWS CloudTrail - Alert Logic.” [Online]. Available: https://aws.amazon.com/es/cloudtrail/partners/alert-logic/. [Accessed: 30-May-2018].

[16] Security Standards Council, “PCI (industria de tarjetas de pago).”

[17] University System of New Hampshire, “Ley de Transferencia y Responsabilidad de los Seguros Médicos (HIPAA).”

[18] Universidad EAFIT, “Ley Sarbanes Oxley (SOX).”

[19] AlienVault, “AWS Security Monitoring & Compliance Management.” [Online]. Available: https://www.alienvault.com/solutions/aws-security-and-compliance-management. [Accessed: 24-May-2018].

[20] BMC, “TrueSight Pulse.” [Online]. Available: https://truesightpulse.bmc.com/. [Accessed: 25-May-2018].

[21] Amazon, “Almacenamiento de datos seguro en la nube (S3).” [Online]. Available: https://aws.amazon.com/es/s3/. [Accessed: 25-May-2018].

[22] Amazon, “¿Qué es AWS Lambda? - AWS Lambda.” [Online]. Available: https://docs.aws.amazon.com/es_es/lambda/latest/dg/welcome.html. [Accessed: 25-May-2018].

[23] Amazon, “AWS Lambda – Precios.” [Online]. Available: https://aws.amazon.com/es/lambda/pricing/. [Accessed: 26-May-2018].

[24] Amazon, “Servicio de base de datos gestionada NoSQL (DynamoDB).” [Online]. Available: https://aws.amazon.com/es/dynamodb/. [Accessed: 27-May-2018].

[25] Amazon, “API Gateway.” [Online]. Available: https://aws.amazon.com/es/api-gateway/. [Accessed: 27-May-2018].

[26] Amazon, “AWS CloudFormation – Infraestructura como código y aprovisionamiento de recursos de AWS.” [Online]. Available: https://aws.amazon.com/es/cloudformation/. [Accessed: 27-May-2018].

[27] Amazon, “¿Qué es AWS CloudFormation Designer? - AWS CloudFormation.” [Online]. Available: https://docs.aws.amazon.com/es_es/AWSCloudFormation/latest/UserGuide/working-with-templates-cfn-designer.html. [Accessed: 01-Jun-2018].

[28] Python Software Foundation, “Introducción — Tutorial de Python 3.6.3 documentation.” [Online]. Available: http://docs.python.org.ar/tutorial/3/real-index.html. [Accessed: 02-Jun-2018].

[29] Amazon, “SDK y conjuntos de herramientas de programación para AWS.” [Online]. Available: https://aws.amazon.com/es/tools/. [Accessed: 03-Jun-2018].



[30] Amazon, “Boto 3 - The AWS SDK for Python.” [Online]. Available: https://github.com/boto/boto3. [Accessed: 03-Jun-2018].

[31] Amazon, “Quickstart — Boto 3 Docs 1.7.48 documentation.” [Online]. Available: https://boto3.readthedocs.io/en/latest/guide/quickstart.html. [Accessed: 03-Jun-2018].

[32] Weblantropia, “RESTful API: ¿Qué es y para que sirve?” [Online]. Available: http://www.weblantropia.com/2016/05/24/restful-api-que-es/. [Accessed: 05-Jun-2018].

[33] GRyCAP, “Graphical tool for CloudTrail-Tracker.” [Online]. Available: https://github.com/grycap/cloudtrail-tracker-ui.

[34] ETSINF UPV, “JIDINF’18 | Jornada de Innovación Docente ETSINF 2018 (3 de julio).” [Online]. Available: http://jidinf.webs.upv.es/. [Accessed: 25-Jun-2018].

[35] ACM SIGCSE, “SIGCSE 2019.” [Online]. Available: https://sigcse2019.sigcse.org/. [Accessed: 05-Jun-2018].

[36] Free Software Directory, “License:Apache2.0.” [Online]. Available: https://directory.fsf.org/wiki/License:Apache2.0. [Accessed: 06-Jun-2018].

[37] Yan Cui, “How long does AWS Lambda keep your idle functions around before a cold start?” [Online]. Available: https://read.acloud.guru/how-long-does-aws-lambda-keep-your-idle-functions-around-before-a-cold-start-bf715d3b810. [Accessed: 08-Jun-2018].

[38] Matt Asay, “Gartner: AWS Now Five Times The Size Of Other Cloud Vendors Combined.” [Online]. Available: https://readwrite.com/2013/08/21/gartner-aws-now-5-times-the-size-of-other-cloud-vendors-combined/#awesm=~ofa4PMQ8ODXA2W. [Accessed: 08-Jun-2018].

[39] Amazon, “Capa gratuita de AWS.” [Online]. Available: https://aws.amazon.com/es/free/. [Accessed: 08-Jun-2018].

[40] Amazon, “Administración de identidades | IAM.” [Online]. Available: https://aws.amazon.com/es/iam/. [Accessed: 30-Jun-2018].

[41] Amazon, “Límites de AWS Lambda.” [Online]. Available: https://docs.aws.amazon.com/es_es/lambda/latest/dg/limits.html. [Accessed: 25-Jun-2018].

[42] Amazon, “Límites y problemas conocidos de Amazon API Gateway.” [Online]. Available: https://docs.aws.amazon.com/es_es/apigateway/latest/developerguide/limits.html. [Accessed: 30-Jun-2018].

[43] Amazon, “Precios Amazon Web Service S3.” [Online]. Available: https://aws.amazon.com/es/s3/pricing/. [Accessed: 28-Jun-2018].

[44] Amazon, “Precios de Amazon DynamoDB.” [Online]. Available: https://aws.amazon.com/es/dynamodb/pricing/. [Accessed: 29-Jun-2018].

[45] Amazon, “API Gateway – Precios.” [Online]. Available: https://aws.amazon.com/es/api-gateway/pricing/. [Accessed: 30-Jun-2018].

[46] P. M. Mell and T. Grance, “The NIST definition of cloud computing,” 2011.

[47] Google Cloud, “Google Cloud Platform Pricing Calculator.” [Online]. Available: https://cloud.google.com/products/calculator/. [Accessed: 02-Jul-2018].



# 8. Anexo

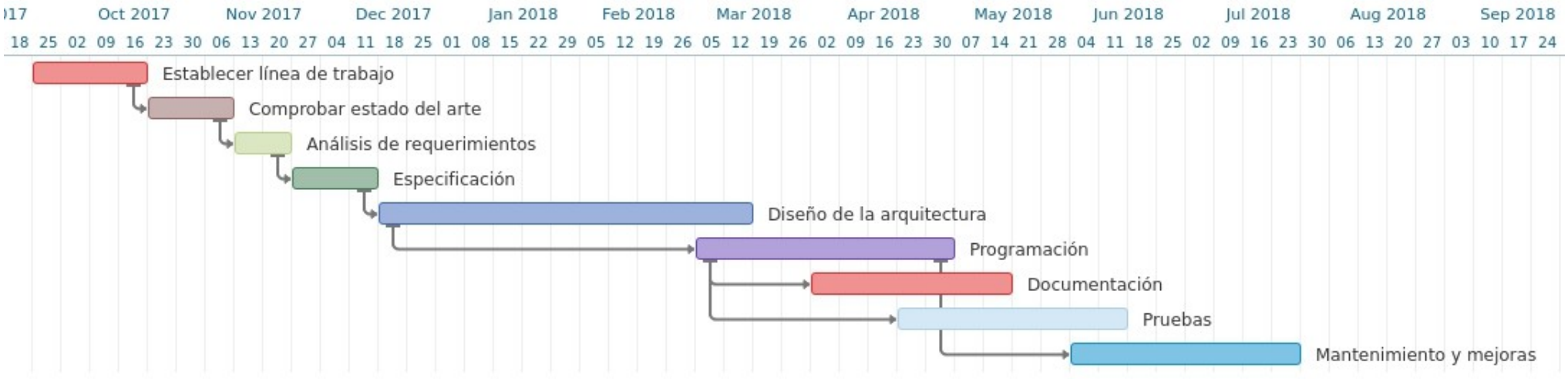



Diagrama de Gantt, estimación del tiempo empleado por cada tarea junto a sus dependencias.