UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Sistema Serverless para la Detección de Incidencias en AWS

Trabajo Fin de Máster

Máster Universitario en Ciberseguridad y Ciberinteligencia

AUTOR/A: Montroy Ibáñez, Vicente

Tutor/a: Moltó Martínez, Germán

CURSO ACADÉMICO: 2022/2023

Resumen

Las cuentas de AWS a menudo son compartidas por diferentes usuarios (*muti-tenancy*). Esto implica que una fuga de credenciales o un abuso por parte de un usuario puede provocar una alteración en el uso habitual de recursos de dicha cuenta de AWS. Una adecuada implementación del principio de privilegio mínimo únicamente permite realizar ciertas acciones empleando determinados servicios de AWS. Sin embargo, no resulta fácil detectar cuándo un usuario ha desplegado de forma seguida un determinado número de instancias de Amazon EC2, el servicio de despliegue de máquinas virtuales o creado ciertos *buckets* de Amazon S3, el servicio de almacenamiento de objetos, o invocado ciertas veces una determinada función de AWS Lambda, el principal servicio de computación *serverless*. Sin embargo, esta información es especialmente relevante para el propietario de la cuenta de AWS con el objetivo de mitigar el impacto derivado del abuso.

Por ello, este trabajo de final de máster plantea la creación de un sistema *serverless* que permita, a partir de la definición de reglas y la evaluación de estas, alertar de la ocurrencia de ciertos eventos que puedan ser considerados anómalos en base a dichas reglas. Para ello se basará en *logs* de eventos obtenidos a través de CloudTrail, el servicio que proporciona AWS para monitorizar y registrar toda la actividad de una cuenta. Para llevar a cabo el desarrollo, se emplearán distintos servicios *serverless* ofrecidos por AWS, los cuales se conectarán y formarán una arquitectura capaz de detectar este tipo de comportamientos.

Palabras clave: AWS, FaaS, Serverless, Event-driven, Monitorización, Detección, Compliance

Abstract

AWS accounts are often shared by multiple users (multi-tenancy). This means that a credential leak or abuse can lead to a disruption in the normal resource usage of that AWS account. A proper implementation of the least privilege principle only allows a user to perform specific actions using certain AWS services. However, it is not easy to detect when a user has deployed several EC2 instances in a row, created certain number of S3 buckets or invoked and AWS Lambda function a certain number of times. Nevertheless, this information is particularly relevant for the AWS account owner in order to mitigate the impact of abuse.

For this reason, this final master's project proposes the creation of a serverless system which, based on the definition of rules and the evaluation of these, would alert to the ocurrence of certain events that could be considered anomalous on the basis of these rules. This will be based on event logs obtained through CloudTrail, the AWS service to monitor and record all the activity in an account. To carry out the development, different serverless services will be used. These services will be connected to each other, forming an architecture capable of detecting this type of behaviour.

Key words: AWS, FaaS, Serverless, Event-driven, Monitoring, Detection, Compliance

Tabla de contenidos

Tabla de figuras

1. Introducción

En el presente Trabajo de Final de Máster se pretende implementar un conjunto de reglas que permitan al propietario de una cuenta de Amazon Web Services (AWS), en la cual interactúen distintos usuarios, monitorizar y detectar un uso inapropiado de los recursos, ya sea por equivocación del usuario o motivado por otros objetivos. Dichas reglas serán capaces de detectar patrones en el comportamiento de cada usuario dentro del entorno de AWS y, en el caso de que este se salga de lo habitual, se notificará al administrador de la cuenta. Con ello, de una forma sencilla y sin depender de terceros, el propietario de la cuenta de AWS podrá minimizar o, en el mejor de los casos, evitar cualquier abuso de su infraestructura en la nube.

1.1. Objetivos

El punto clave del proyecto es que el sistema de detección de anomalías en el comportamiento de los usuarios se basará en lo que se conoce como s*erverless computing*, esto es, computación sin servidor. La principal ventaja que aportará este acercamiento será que, mientras ninguna regla de las existentes sea violada, no se generará ningún tipo de coste para el propietario de la cuenta de AWS. Esto choca directamente con el enfoque tradicional, en el cual, para desplegar un servicio cualquiera debe existir, al menos, un servidor que lo aloje, el cual generará gastos por el mero hecho de estar desplegado, se utilice o no. Evidentemente, cuando se habla de computación sin servidor, no hay que entenderlo de forma literal. Siempre habrá un equipo físico alojando un servicio, pero el modelo de negocio y de facturación pasa de ser "pago por despliegue" a "pago por uso". En el apartado correspondiente a *serverless computing* se abordará todo ello con más detalle.

Por tanto, el objetivo que se persigue es la implementación de un sistema basado en funciones Lambda (la solución de computación *serverless* de AWS) capaz de detectar anomalías en el comportamiento de un usuario, como puede ser el despliegue múltiple de instancias de EC2, y notificar al administrador de la cuenta de AWS para que pueda llevar a cabo las acciones pertinentes. Con ello, se busca minimizar el impacto que un mal uso del entorno, una fuga de credenciales o cualquier otro contratiempo pueda ocasionar al propietario de la cuenta de AWS. A lo largo del proyecto se hablará del enfoque *serverless* y de las herramientas existentes que permiten la creación de este tipo de reglas, y se detallará la implementación del sistema de monitorización en un entorno real.

1.2. Motivación

La motivación principal por la que elegí este trabajo fue mi interés por la computación en la nube, y más concretamente por el entorno Amazon Web Services. Un trabajo en el cual tuviera la oportunidad de trabajar con AWS me permitiría seguir aprendiendo sobre este entorno y las ventajas que puede ofrecer. Además de la posibilidad de profundizar en el mundo

serverless y las funciones como servicio, las cuales han venido para quedarse y es importante conocerlas bien y aprender a trabajar con ellas.

Por otro lado, el proyecto en sí me pareció muy interesante. Desarrollar un sistema capaz de detectar el mal uso de un entorno y ser capaz de reaccionar, todo ello de forma autónoma y a un coste reducido, ya que durante el tiempo en que el comportamiento sea normal este permanecerá a la espera, sin generar ningún tipo de coste. Este tipo de sistema puede resultar interesante para prácticamente cualquier cuenta de AWS que tenga que trabajar con múltiples usuarios, como pueden ser entornos educativos o empresariales.

2. *Serverless Computing*

Cuando se habla de *serverless computing*, cuya traducción literal es computación sin servidor, se hace referencia a un modelo de computación en el cual se deja de lado la infraestructura subyacente, centrándose en los servicios. Este hecho permite a los desarrolladores centrar sus esfuerzos en el desarrollo de las aplicaciones y no en la implementación de la infraestructura que las sustentará.

Este concepto puede dividirse en dos ramas principales: *Backend-as-a-Service* o *BaaS* y *Functions-as-a-Service* o *FaaS*. El primero de ellos engloba los diferentes servicios que una aplicación puede necesitar, como pueden ser bases de datos, servicios de autenticación o almacenamiento, entre otros. Por otro lado, *FaaS* es un concepto muy importante dentro de *serverless*. Las funciones como servicio son un tipo de servicio *cloud* que permite implementar computación basada en eventos. Se encuentra disponible en distintas nubes públicas, como Microsoft Azure (mediante el servicio Azure Functions) o AWS (mediante el servicio AWS Lambda), entre otras. Estas funciones son las encargadas de implementar la capacidad de cómputo a la informática sin servidor [1].

2.1. El modelo de servicio *FaaS*

Muchas veces tiende a confundirse el concepto de *serverless* con el de *FaaS* y a pensar que son lo mismo, pero *FaaS* es solamente una parte (fundamental) dentro del mundo *serverless*. De todos los servicios que pueden ofrecerse siguiendo la filosofía *serverless*, las *FaaS* son las que se encargan de ofrecer el servicio de computación en respuesta a eventos. Por tanto, *serverless* incluye *FaaS*, así como servicios de almacenamiento, bases de datos o autenticación, entre muchos otros.

Se podría decir que *Serverless* es un modelo de ejecución y desarrollo de aplicaciones basado en el uso de servicios gestionados y *FaaS* un modelo de servicio que involucra el uso de servicios de ejecución de funciones dirigidas por eventos. El proveedor nos permite ejecutar porciones de código, llamadas funciones, dependiendo de la ocurrencia de eventos concretos y sin preocuparnos de la infraestructura que hace posible ofrecer dicho servicio. Las funciones representan la unidad de despliegue, ejecución y escalado. Estas son *stateless*, es decir, no almacenan su estado una vez finaliza su ejecución. Si fuese necesario almacenar el estado de una función habría que hacerlo recurriendo a otro servicio.

Haciendo un paralelismo con una arquitectura basada en microservicios, una arquitectura basada en *FaaS* también se compondrá de pequeños elementos independientes, pero dichos elementos, en lugar de servicios, serán funciones. Es decir, los servicios que componen la arquitectura basada en microservicios se descompondrán en funciones autónomas (Figura 1). Este tipo de arquitectura aporta un nivel más de abstracción y pueden diseñarse sistemas complejos concatenando distintas funciones.

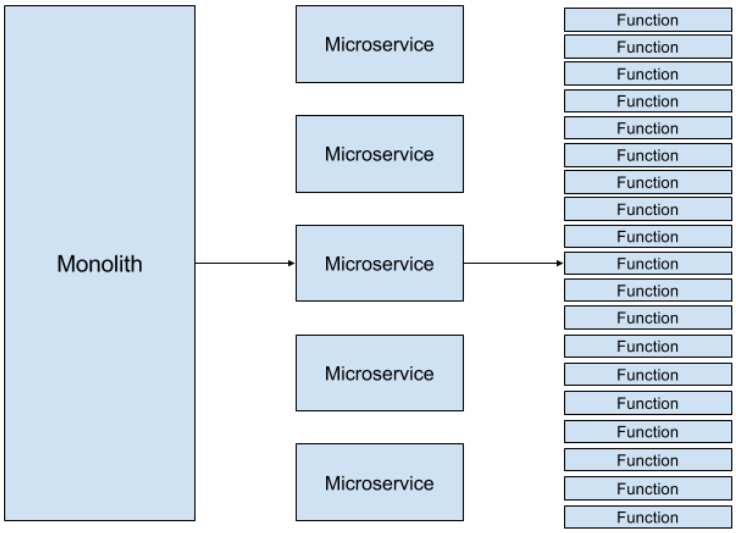


Figura 1. De arquitectura monolítica a funciones [2].

2.2. Ventajas y desventajas del modelo *FaaS*

Este modelo de computación cuenta con ciertas ventajas que han hecho que su adopción sea muy positiva. Entre sus ventajas principales se encuentra el escalado automático. Las funciones escalarán de forma automática, es decir, se aumentarán los recursos cuando sea necesario para llevar a cabo la tarea de forma óptima. El usuario no podrá equivocarse a la hora de aprovisionar la capacidad necesaria, ya que es el proveedor quien se encarga de esta tarea, aumentando o reduciendo los recursos según convenga.

Una de sus ventajas fundamentales es el pago por uso real. En otro tipo de servicios, el pago es por despliegue, esto es, se factura el tiempo que el servicio esté desplegado, esté trabajando o no. Esto no ocurre en las funciones como servicio, donde solamente se facturará el tiempo de cómputo, es decir, el tiempo durante el cual la función se esté ejecutando.

Por otro lado, el hecho de no tener que diseñar y desplegar una infraestructura hace que el tiempo de despliegue del servicio sea menor. Es decir, el "time to market" se reduce, lo que aumenta la ventaja competitiva del desarrollador. En esta misma línea, sin la necesidad de configurar la infraestructura que soporta el servicio, se elimina la necesidad de contar con equipo especializado que administre y mantenga dicha infraestructura, por lo que se eliminan, o se reducen en gran medida, los gastos asociados a este respecto. El desarrollador simplemente tiene que subir el código de la aplicación a la nube pública, el resto será tarea del proveedor.

Otra de las ventajas principales es la alta disponibilidad del servicio. Si los elementos que sustentan la aplicación fallan, el proveedor asignará nuevos recursos de forma automática. También se encargará de gestionar, mantener y actualizar los servidores, así como de aplicar los parches de seguridad que sean necesarios.

Las funciones como servicio también cuentan con algunos contrapuntos. Los principales se exponen a continuación.

En primer lugar, debido a que cada proveedor implementa *FaaS* de un modo distinto, la interoperabilidad entre ellos puede ser complicada y cara, lo que puede suponer una dependencia del proveedor que se esté empleando. Este hecho se conoce como "vendor lock-in".

Por otro lado, la parte negativa de abstraerse de la infraestructura subyacente es la pérdida del control total de la aplicación. Por ejemplo, no será posible (o será complicado) optimizar la aplicación para un *hardware* específico o quizá no se puedan configurar ciertos parámetros del modo óptimo. Desde el punto de vista de la seguridad, habrá que confiar en el proveedor.

Uno de los problemas principales de las funciones como servicio es lo que se conoce como "cold start". Esto es, cuando una función permanece inactiva durante un tiempo, el proveedor escala a cero los recursos asociados a dicha función, es decir, deja de asociarle los recursos encargados de ejecutarla, por lo que, cuando un nuevo evento la ejecute, pasará un tiempo hasta que esta pueda llevar a cabo su cometido, aumentando el tiempo total de ejecución. Existen distintos métodos para solucionar este problema y cada proveedor aplica el suyo.

Por último, hay que tener en cuenta que las funciones como servicio no son adecuadas cuando el tiempo de ejecución del código es elevado. Los tiempos de ejecución suelen estar acotados.

2.3. AWS Lambda

Durante la realización del presente trabajo, el proveedor de nube pública con el que se trabajará será Amazon Web Services o AWS. La solución de *FaaS* en AWS se conoce como Lambda [3] y sus funciones se ejecutan en base a la ocurrencia de ciertos eventos previamente definidos por el usuario.

AWS Lambda nace en 2014, hecho que hace ganar mucha popularidad al modelo *serverless* por la integración de las funciones como servicio en uno de los grandes proveedores. Las grandes tecnológicas comienzan a desarrollar y ofrecer sus propios servicios *serverless* y aparecen servicios como el propio Lambda o Cognito, de AWS; Google Cloud Functions, de Google o Azure Functions, de Microsoft.

Lambda es una de las soluciones más maduras y estables que encontramos en el mercado y es capaz de trabajar de forma nativa con los principales lenguajes de programación, como son Python, Go, Java o C#. Además, proporciona una API de tiempo de ejecución o *runtime* que permite emplear cualquier lenguaje de programación adicional para crear las funciones.

La facturación del servicio AWS Lambda es por milisegundos de ejecución de la función. El precio también dependerá de la región en la que se ejecute la función y de la arquitectura subyacente (x86 o ARM). Dentro de la capa gratuita de AWS, podrán realizarse 1 millón de solicitudes y 3,2 millones de segundos de tiempo de ejecución al mes. Puede asignarse a cada función un volumen de memoria entre 128 MB y 10240 MB, en incrementos de 1 MB y el tiempo máximo de ejecución de las funciones es de 15 minutos por invocación [4].

La disponibilidad en AWS Lambda es del 99,95% del tiempo para cada una de las regiones de AWS. Si dicha disponibilidad no es alcanzada en algún momento, AWS compensará al cliente con créditos para AWS Lamba, esto es, descuentos en el servicio [5].

En cuanto al problema de “cold start”, AWS Lambda lo previene haciendo uso de la “provisioned concurrency”, esto es, mantener inicializados cierto número de instancias que estarán listas de forma inmediata, sin pérdidas de tiempo por arranque. Emplear lenguajes de programación con un coste bajo de inicialización (p. ej. Python o Go) también es una buena medida para reducir este problema.

Por último, en cuanto a los tiempos de ejecución del código asociado a una función, en AWS Lambda el tiempo máximo de ejecución son 15 minutos.

3. Estado del arte y tecnología empleada

Amazon Web Services cuenta con una enorme cantidad de herramientas y funcionalidades para llevar a cabo distintas tareas. A lo largo del desarrollo del proyecto se emplearán algunas de estas herramientas, como puede ser CloudTrail, para la recolección de *logs* de las acciones realizadas por los usuarios en la cuenta de AWS durante su interacción con los múltiples servicios ofrecidos por este proveedor. Se va a trabajar también con herramientas externas a AWS, pero que pueden ser integradas en el entorno, como son Cloud Custodian, para la definición de políticas y su despliegue en funciones de Lambda.

3.1. Estado del arte

Existen diferentes herramientas o soluciones comerciales con características similares a las del presente proyecto: detección de amenazas, detecciones basadas en código (o reglas), respuesta antes dichas detecciones, aumento de la visibilidad del entorno o monitorización, entre otras. En este apartado se nombrarán algunas de ellas, señalando cuáles son sus puntos fuertes.

3.1.1. Panther

Panther [6] es una herramienta *cloud-native*, es decir, centrada para el uso en la nube. Su objetivo principal es transformar grandes cantidades de datos, como pueden ser los *logs* en crudo, en información bien estructurada, con el fin de detectar y responder en tiempo real ante un incidente. Permite desarrollar reglas a medida empleando el lenguaje Python (*detection-as-code*) y cuenta también con integraciones predefinidas para varias fuentes de datos.

Entre sus principales ventajas, el fabricante destaca: centrada en la seguridad, eliminando la necesidad de gestionar servidores, almacenamiento o actualizaciones; rapidez en la detección de incidentes, analizando los *logs* en el mismo momento que son ingestados; respuesta rápida gracias a la consulta de *logs* anteriores y búsqueda de IoCs (*Indicators of Compromise*).

3.1.2. Dynatrace

Dynatrace [7] es una plataforma de monitorización que pretende simplificar la complejidad de la nube y acelerar la transformación digital de las empresas. Aporta información

de interés sobre el desempeño de las aplicaciones y de la experiencia de los usuarios finales, así como de la infraestructura subyacente.

Entre sus principales características están: automatización, es decir, desde el despliegue, el descubrimiento de activos o la identificación de problemas se llevan a cabo de forma transparente para el usuario; es una herramienta "Full Stack", ya que monitoriza desde la experiencia de usuario hasta la infraestructura que sustenta la aplicación; se basa en la inteligencia artificial, esta no es una característica más, es el núcleo de la plataforma; ofrece escalabilidad gracias a ser una solución basada en la nube.

3.1.3. LogicMonitor

LogicMonitor [8] aporta a sus clientes la capacidad de monitorizar y ganar visibilidad sobre infraestructuras complejas y distribuidas. De este modo, pueden adelantarse a los problemas antes de que sucedan, mejorando así la experiencia del usuario final.

La solución de LogicMonitor cuenta con una sola plataforma central donde visualizar los datos referentes a infraestructura, *logs* y aplicaciones. Hace uso de inteligencia artificial para detectar anomalías, analizar cuál ha sido la raíz del problema e incluso es capaz de realizar pronósticos. Del mismo modo que la solución anterior, puede desplegarse y configurarse de forma automatizada.

3.1.4. Datadog

Datadog [9] ofrece distintas soluciones de monitorización: infraestructura, rendimiento de la red, contenedores, e incluso ofrece monitorización de las funciones *serverless* desplegadas. Cuenta con una línea de productos llamada "Datadog Cloud Security Platform", dentro de la cual se sitúa "Datadog Cloud SIEM".

Datadog Cloud SIEM ofrece detección de amenazas en entornos *cloud*, analizando *logs* en tiempo real y empleando reglas para su detección. Además, cuenta con una plataforma donde centraliza toda la información, haciendo más sencilla la tarea de investigación.

3.1.5. Securonix

Securonix [10] cuenta con "Cloud Security Monitoring", su solución para la detección y respuesta ante amenazas en entornos *cloud*. Entre otros proveedores, ofrece integración para AWS, donde se centra en: eventos de *login*, eventos de configuraciones de EC2 (Elastic Compute Cloud), *logs* de ELB (Elastic Load Balancing), *logs* de conexión a VPCs (Virtual Private Cloud) o actividades IAM (Identity Access and Management).

Entre los casos de uso que destaca la compañía para AWS, hay algunos muy similares a los que pretenden cubrirse con este proyecto, por ejemplo: detección anomalías en EC2, como picos de

creación o eliminación de instancias; Creaciones de usuarios sospechosas, cambios de privilegios, cambios en políticas de contraseña o actividades privilegiadas poco habituales; Conexiones sospechosas hacia APIs o desde direcciones IP sospechosas, entre otros.

Como se ha observado, debido al creciente uso de la nube pública, el número de herramientas o soluciones también ha aumentado. La parte diferencial que aporta el presente proyecto es la capacidad de obtener muchas de las ventajas descritas, pero minimizando al máximo el gasto para el propietario del entorno.

Para poder disfrutar de las soluciones anteriormente nombradas, estas deberán integrarse en el entorno y habrá que pagar por su uso. Por el contrario, una solución implementada por el propietario del entorno aportará ventajas como: no depender de terceros; minimizar el coste, ya que no habrá gasto alguno si no se detecta ninguna anomalía (este gasto puede incluso ser de cero si las detecciones a lo largo de un mes no son muy numerosas); o evitar que los datos salgan del propio entorno, lo que aporta seguridad y privacidad.

3.2. Tecnología empleada

3.2.1. Cloud Custodian

La herramienta Cloud Custodian [11] unifica distintas herramientas y *scripts* que muchas empresas emplean a la hora de gestionar sus cuentas en la nube pública. Su funcionamiento se basa en el establecimiento de reglas para la definición de políticas y una de sus ventajas es que es capaz de integrarse de forma adecuada con el modelo *serverless*.

Entre las distintas políticas que pueden implementarse con esta herramienta encontramos, por ejemplo, políticas de seguridad, de etiquetado, de gestión de recursos no utilizados o de costes, todo ello unificado en una misma herramienta.

Las políticas de Cloud Custodian se definen empleando el formato YAML y todas ellas incluyen el tipo de recurso sobre el que se aplicará la política, filtros que permitirán reducir el número de recursos sobre los que se aplicará y por último las acciones que se aplicarán. Cuando se define una nueva política, esta puede ser probada y validada antes de desplegarse en un entorno en producción.

Se trata de una herramienta multiplataforma, es decir, puede ser empleada en distintos proveedores, entre ellos AWS.

En este punto puede introducirse el concepto de “Governance as Code”, ya que la función de Cloud Custodian es implementar las distintas políticas que permiten gestionar la gobernanza de una organización mediante líneas de código. A la hora de gestionar una infraestructura en la nube, es necesaria una optimización constante para poder mantener el rendimiento, la seguridad, la disponibilidad o la optimización de costes en los niveles adecuados. Gestionar cada uno de esos aspectos de forma manual puede ser complicado e incluso, en algunos casos, inviable, por lo que el “Governance as Code” puede ayudar a automatizar esta tarea [12].

3.2.2. Servicios de AWS

A continuación, se proporciona una breve descripción de los servicios de AWS involucrados en el desarrollo de la solución propuesta para la detección de incidencias en AWS.

3.2.2.1. CloudTrail

CloudTrail [13] es un servicio de AWS cuyo objetivo principal es monitorizar y registrar la actividad de una cuenta en toda la infraestructura de AWS. CloudTrail recoge toda la actividad y uso de la API de cada usuario para cada servicio de AWS, almacenando los *logs* en *buckets* de S3. Algunos casos de uso interesantes que nos indica Amazon son: control de actividad, esto es, monitorización, almacenamiento y validación de eventos; o identificación de incidentes relativos a la seguridad, es decir, detecciones de accesos no autorizados.

En resumen, se trata de un servicio de auditoría que ofrece el propio Amazon. Todas las acciones llevadas a cabo por usuarios, roles o servicios de AWS son recogidas por CloudTrail. Estos eventos incluyen tanto aquellos llevados a cabo en la consola de AWS, como mediante la interfaz de línea de comandos, las API o el SDK, creando para cada actividad un nuevo evento de CloudTrail [14].

Dentro de la arquitectura propuesta será la herramienta encargada de monitorizar y registrar la actividad de una cuenta en toda la infraestructura de AWS. Es capaz de registrar los movimientos y eventos generados por cada usuario y puede integrarse con AWS EventBridge.

3.2.2.2. S3 (Simple Storage Service)

Simple Storage Service o S3 [15] es el servicio de almacenamiento de objetos que proporciona AWS. S3 está enfocado al almacenamiento de objetos y ofrece un alto nivel de escalabilidad y disponibilidad. Se trata de un servicio que puede adaptarse a distintos casos de uso y dentro de la arquitectura propuesta será el servicio encargado de almacenar todos los *logs* que genere CloudTrail.

3.2.2.3. DynamoDB

DynamoDB [16] es una base de datos NoSQL *serverless* de clave-valor que nos permite ejecutar aplicaciones a cualquier escala. Entre otras características, ofrece copias de seguridad continuas y replicación automatizada en varias regiones de AWS. Este servicio será el encargado de almacenar los eventos que están siendo monitorizados para poder obtener el número de ocurrencias de un evento en un rango de tiempo determinado. Lambda será el servicio encargado de interactuar con la tabla de la base de datos de DynamoDB, tanto para añadir los nuevos eventos como para determinar el número de ocurrencias.

3.2.2.4. SNS (Simple Notification Service)

Simple Notification Service o SNS [17] es un servicio *serverless* de mensajería para la comunicación entre aplicaciones o entre aplicación y usuario. Puede enviar notificaciones a distintos destinos, como puede ser una función Lambda u otros servicios de AWS, así como transmitir dichas notificaciones vía SMS o correo electrónico. Este servicio se empleará dentro de la arquitectura propuesta para notificar al administrador si alguna de las reglas implementadas se dispara.

3.2.2.5. Lambda

Lambda es el servicio *serverless* de AWS que nos permitirá incorporar la lógica a la arquitectura. Será el servicio encargado de ejecutar acciones en respuesta a ciertos eventos recibidos desde EventBridge. A su vez, Lambda interactuará con otros servicios con el fin de almacenar la información recibida y notificar al administrador de la cuenta en caso de ser necesario.

Las funciones Lambda también pueden emplearse para integrar políticas de Cloud Custodian, como se detallará en el siguiente apartado.

3.2.2.6. EventBridge

EventBridge [18] es un bus de eventos *serverless* que nos permitirá definir los eventos que se quieren monitorizar y permanecerá a la espera y sin incurrir gastos mientras que no se genere uno de estos eventos. Una vez se haya producido un evento que está siendo monitorizado, EventBridge se encarga de distribuirlo hacia los destinos que se hayan definido. Pueden generarse reglas de enrutamiento y determinar hacia donde se enviarán dichos datos. Es capaz integrarse con múltiples fuentes de eventos, tanto internas como externas.

Dentro de la arquitectura propuesta desarrollará la función de transmitir los eventos generados por CloudTrail hacia las funciones Lambda, las cuales procesarán la información.

4. Diseño del sistema

4.1. Arquitectura

La arquitectura sobre la que se va a desarrollar el proyecto será del tipo “event-driven” u orientada a eventos. Esto es, se va a emplear la ocurrencia de ciertos eventos para iniciar la interacción entre los distintos servicios que formen parte de la arquitectura. Un evento puede ser un cambio de estado, una actualización, una creación de cierto elemento dentro del entorno *cloud*, como puede ser una instancia de EC2, o un *login*, entre muchos otros. Este tipo de arquitecturas cuentan con tres componentes principales: la fuente del evento, los elementos que gestionan y transportan la información acerca de dicho evento y los consumidores de eventos [19].

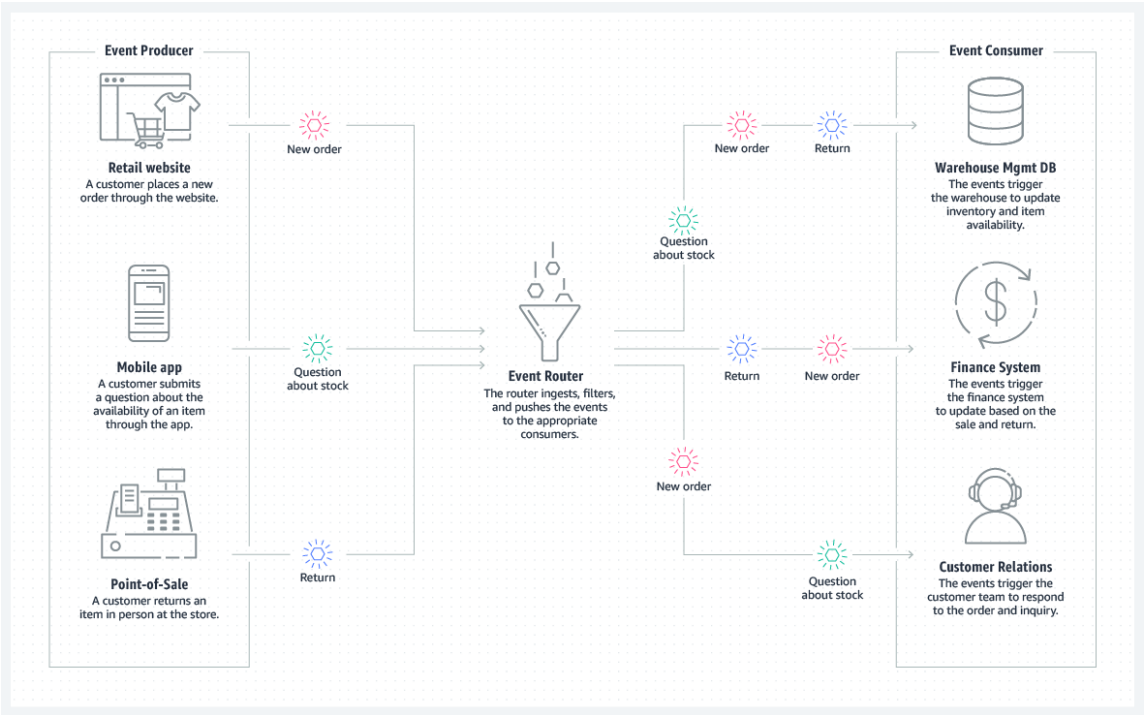


Figura 2. Ejemplo de arquitectura orientada a eventos [19].

Una vez descrito el tipo de arquitectura con el que se va a trabajar, se muestra en la Figura 3 la arquitectura propuesta para el sistema de detección. En esta ocasión, la fuente de los eventos será el servicio CloudTrail; el servicio encargado de transportar los eventos será EventBridge y, por último, el consumidor de los eventos será Lambda. En el siguiente apartado se detallará el proceso de implementación del sistema en una cuenta de AWS.

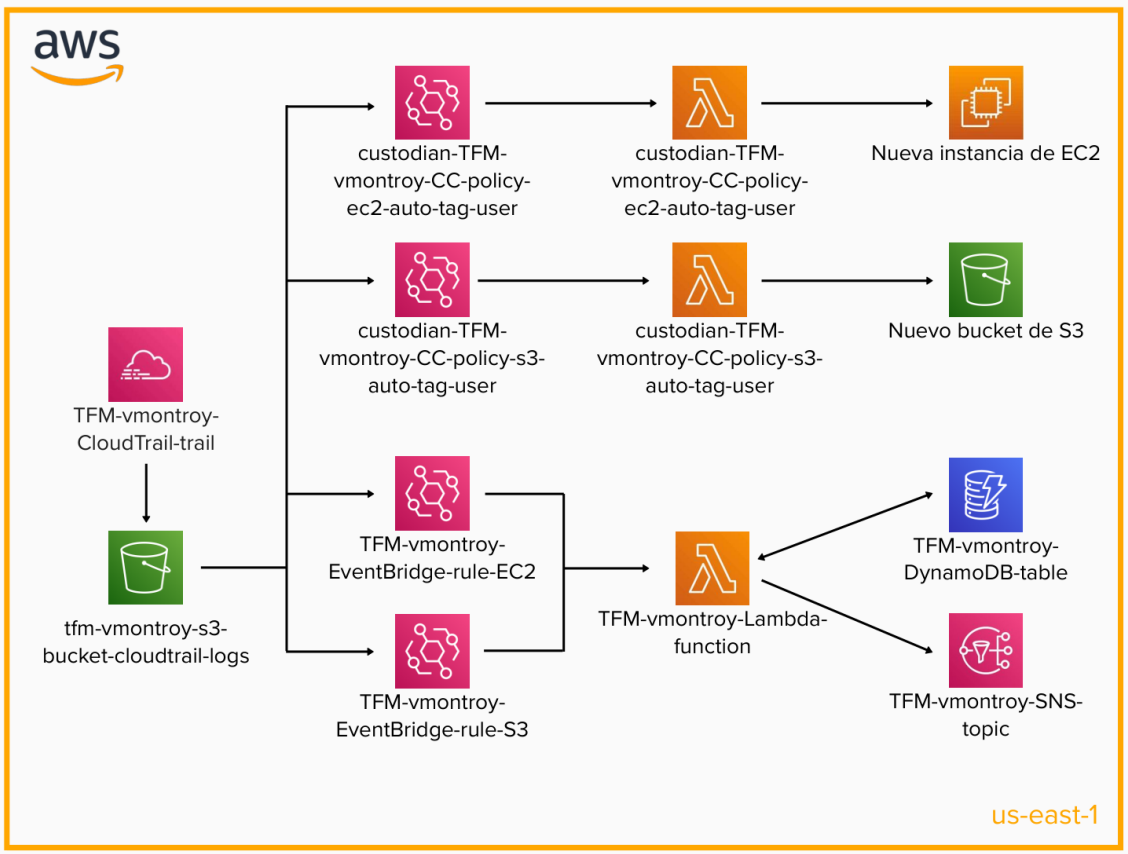


Figura 3. Diagrama de la arquitectura del sistema [Creación propia].

4.2. Implementación de la arquitectura

En este apartado se va a detallar paso por paso el despliegue de cada uno de los elementos que forman parte de la arquitectura propuesta, detallados en el apartado 3.2.2. del presente proyecto, así como la integración entre cada uno de los servicios.

El primer paso antes de desplegar el sistema de detección será contar con una cuenta de AWS. Habrá que crear una nueva cuenta o emplear una existente. Si se opta por trabajar con una cuenta recién creada, será posible beneficiarse del *free tier* o capa gratuita de AWS. Esto es, durante el primer año, Amazon nos permite utilizar ciertos servicios de forma limitada sin incurrir en ningún gasto. Algunos de estos servicios cuentan con un nivel de uso gratuito que no está limitado al primer año. Este punto se detallará para cada uno de los servicios empleados.

4.2.1. Configuración de CloudTrail y S3

El primer servicio que se va a configurar es CloudTrail, ya que será el encargado de capturar los eventos que se quieran monitorizar. Por defecto, AWS crea un *trail* de CloudTrail para capturar los eventos que ocurren en el entorno, pero es importante indicar que no es posible emplear este *trail* creado por defecto en la arquitectura propuesta, habrá que crear uno nuevo.

Esto se debe a que el *trail* que se crea por defecto no tiene activada la opción de *logging*, lo cual es necesario para capturar eventos del tipo "AWS API Call via CloudTrail". Dichos *logs* se almacenarán en un *bucket* de S3.

Dentro de la capa gratuita de AWS se incluye la creación de un *trail* que envíe una copia de cada evento de administración al *bucket* de S3 asociado [20]. Si se requiere más de una copia por evento o capturar eventos que no sean de administración se incurrirá en gastos, pero para la arquitectura propuesta es posible trabajar dentro de la capa gratuita. Respecto al *bucket* de S3 que se creará, AWS ofrece hasta 5 GB de almacenamiento, 20 000 solicitudes GET (obtención de datos) y 2000 solicitudes PUT (subidas de datos) [21]. Durante el desarrollo de este proyecto, el número de solicitudes PUT gratuitas se ha superado, pero el gasto incurrido ha sido de $0.10, es decir, el gasto que puede llegar suponer este sistema de detección es menor.

Para crear el nuevo *trail* habrá que dirigirse a la configuración de CloudTrail en la consola de configuración de AWS, como se observa en la Figura 4.

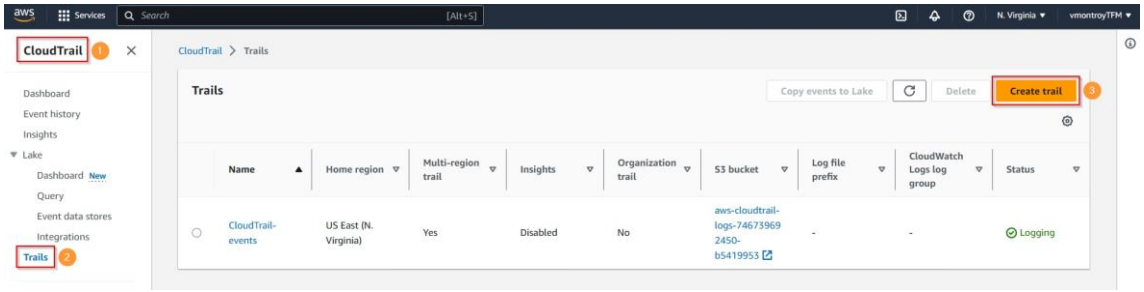


Figura 4. Creación de un nuevo trail de CloudTrail [Creación propia].

Una vez se ha iniciado la creación del *trail*, habrá que configurarlo. El primer paso es darle un nombre al *trail* y al *bucket* S3 asociado que se creará para almacenar los *logs*, como puede apreciarse en la Figura 5. Además, habrá que desmarcar la opción de cifrado de logs. Todas las demás opciones pueden dejarse por defecto y puede procederse al siguiente paso.

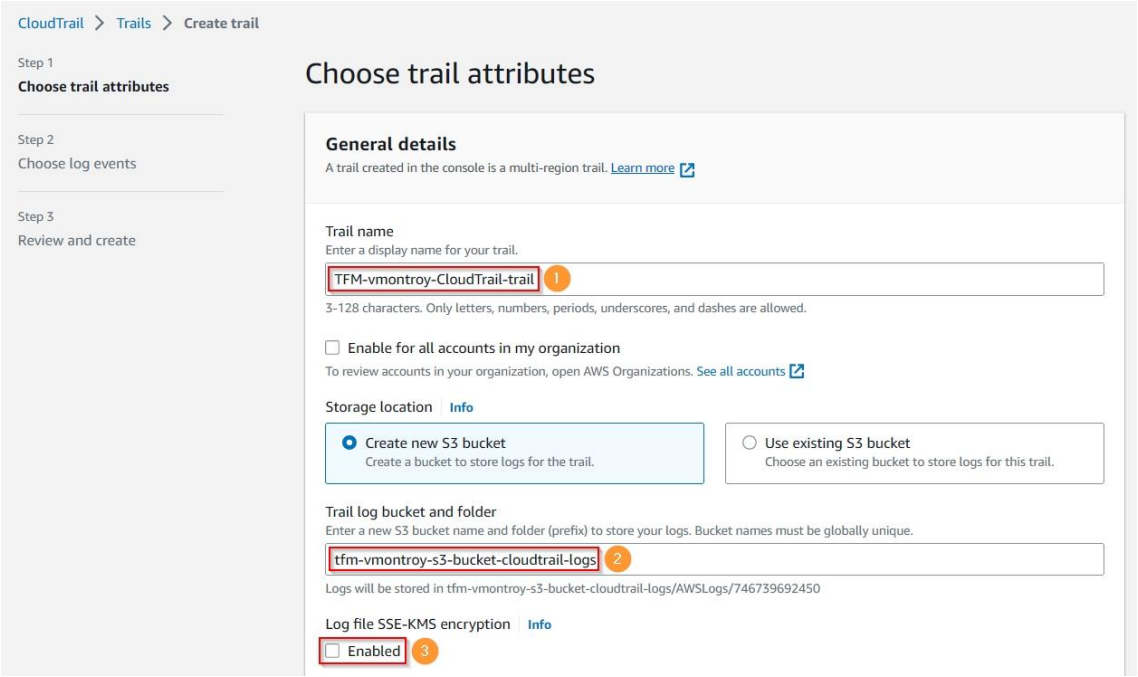


Figura 5. Configuración del nuevo trail - paso 1 [Creación propia].

Una vez se han nombrado el *trail* y el *bucket*, habrá que indicar que tipo de evento se quiere monitorizar. En esta ocasión se van a monitorizar eventos de gestión, los cuales engloban eventos como los de creación o eliminación de instancias de EC2 o de *buckets* de S3, entre muchos otros. En la Figura 6 se observa la opción seleccionada. Los elementos restantes pueden dejarse con la configuración por defecto antes de proceder al último paso.

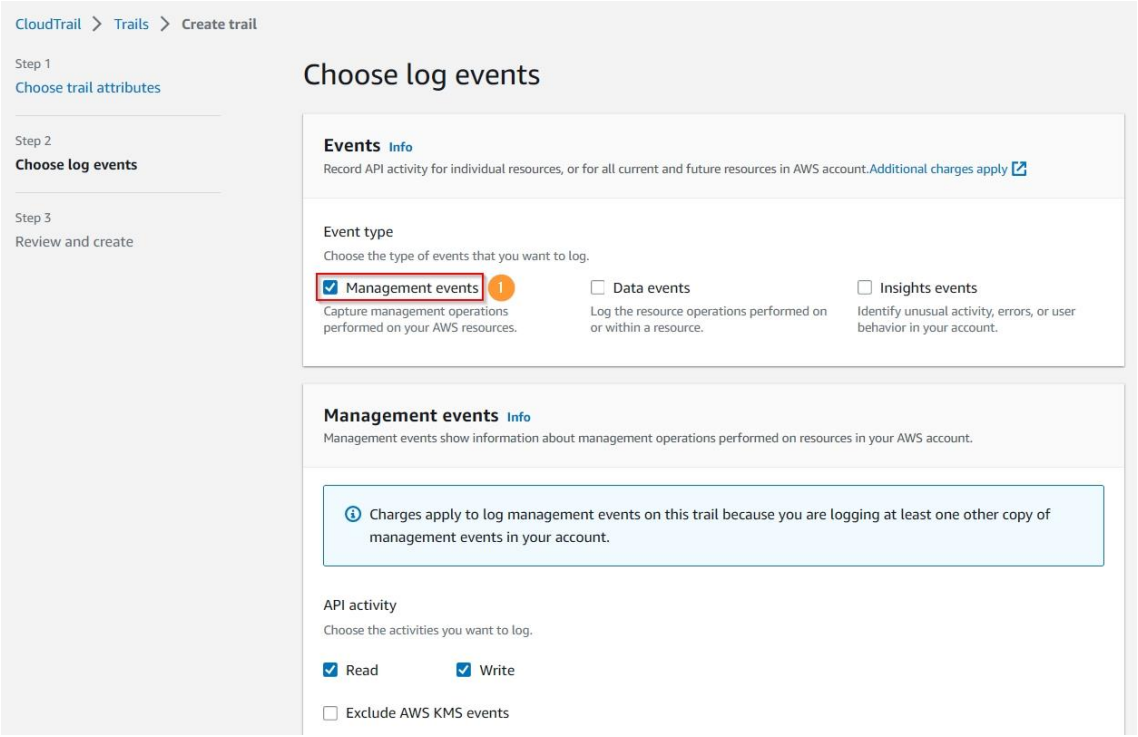


Figura 6. Configuración del nuevo trail - paso 2 [Creación propia].

En la Figura 7 se muestra el resumen de la configuración del nuevo *trail*. Si todo es correcto se procederá a su creación pulsando al botón de crear.

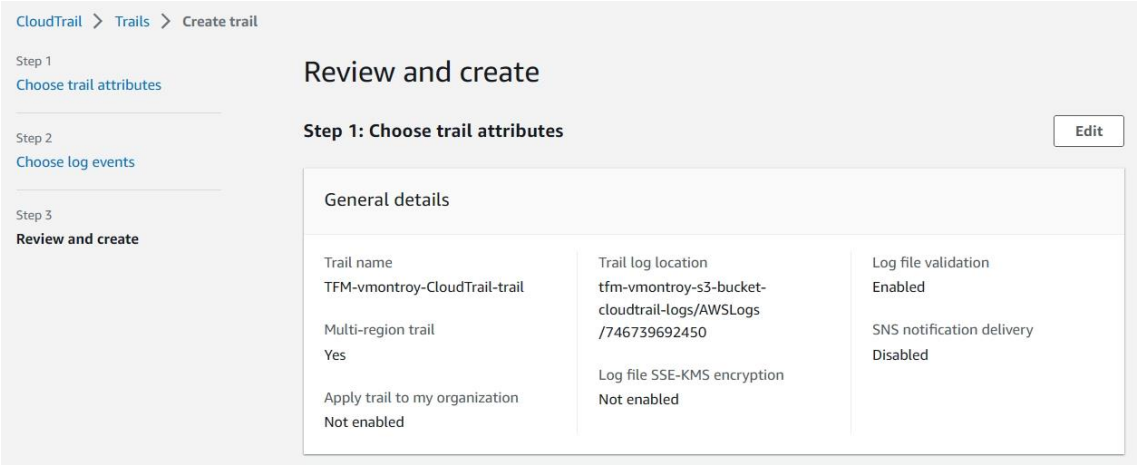


Figura 7. Configuración del nuevo trail - paso 3 [Creación propia].

En las Figuras 8 y 9 pueden observarse el nuevo *trail* desplegado y en funcionamiento, y el *bucket* de S3 donde CloudTrail almacenará todos los eventos monitorizados.

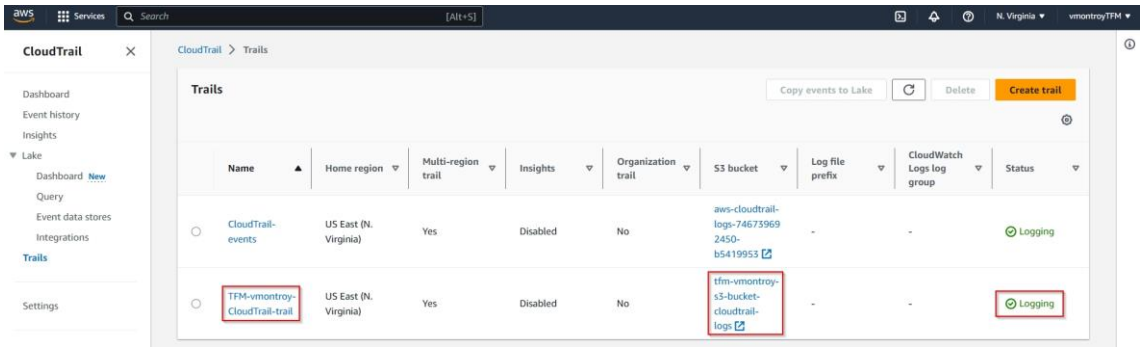


Figura 8. Nuevo trail de CloudTrail en funcionamiento [Creación propia].

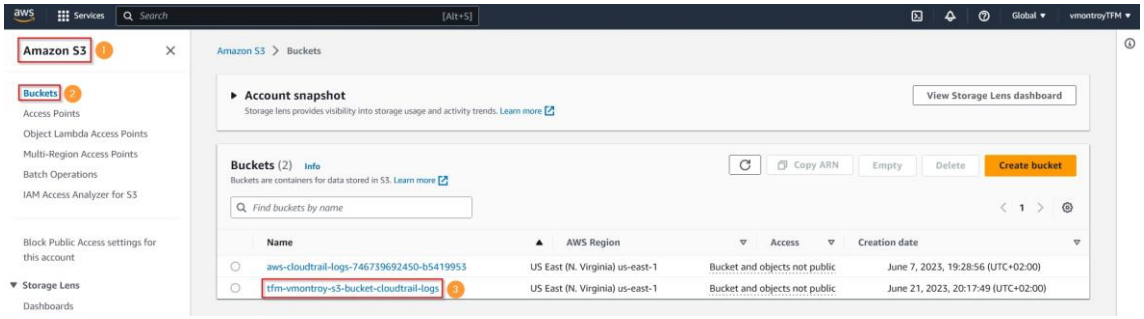


Figura 9. Bucket de S3 asociado al nuevo trail [Creación propia].

A continuación, se muestra un evento de CloudTrail a modo de ejemplo. En este caso, un usuario llamado "Alice" ha empleado la CLI (interfaz de línea de comandos) de AWS para llamar a la acción "StartInstances" de EC2, ejecutando el comando `ec2-start-instances`, con la intención de iniciar una instancia de EC2 con identificador "i-ebeaf9e2" [22]:

```
{"Records": [{
    "eventVersion": "1.0",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::123456789012:user/Alice",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "accountId": "123456789012",
        "userName": "Alice"
    },
    "eventTime": "2014-03-06T21:22:54Z",
    "eventSource": "ec2.amazonaws.com",
    "eventName": "StartInstances",
    "awsRegion": "us-east-2",
    "sourceIPAddress": "205.251.233.176",
    "userAgent": "ec2-api-tools 1.6.12.2",
    "requestParameters": {"instancesSet": {"items": [{"instanceId": "i-
ebeaf9e2"}]}},
    "responseElements": {"instancesSet": {"items": [{
        "instanceId": "i-ebeaf9e2",
        "currentState": {
```

```
            "code": 0,
            "name": "pending"
        },
        "previousState": {
            "code": 80,
            "name": "stopped"
        }
    }]}}
}]}
```


En el evento se observan ciertos campos que pueden resultar de interés para la detección de anomalías, como son: “userName”, que contiene el nombre del usuario que ha iniciado la instancia; “eventTime”, que contiene la fecha y hora en las que se ha iniciado la instancia; “eventName”, que indica el tipo de evento en cuestión; o “instanceId”, que contiene el identificador de la instancia que se ha iniciado.

4.2.2. Configuración de DynamoDB

Este servicio se empleará para almacenar la información de interés que contenga cada uno de los eventos monitorizados. En esta ocasión se va a configurar una tabla de DynamoDB que contará con los siguientes atributos:

- Identificador del elemento desplegado (“elementID”). Actuará como clave de partición, es decir, será el identificador único para cada elemento de la tabla. Al tratarse de la clave de partición de la tabla, no podrán existir dos “elementID” iguales. En el caso de que se introdujera un nuevo valor con el mismo “elementID”, este valor sobrescribiría al anterior. Si fuera necesario contar con dos elementos de la tabla con la misma clave de partición, habría que configurar otro atributo de la tabla como clave de ordenación y que el valor del atributo fuera distinto para los dos elementos [23]. En esta ocasión, como no habrá dos identificadores iguales para dos recursos desplegados distintos, no será necesario configurar una clave de ordenación.

- Nombre del usuario que ha desplegado el recurso (“userName”). Se almacenará el nombre de usuario para poder obtener el número de recursos que ha desplegado y notificar al administrador en caso de superar el límite.

- Nombre del evento (“eventName”). Será necesario almacenar el tipo de evento que se ha capturado para poder contar el número de ocurrencias en el tiempo especificado.

- Fecha y hora del despliegue (“eventTime”). Habrá que almacenar el momento en el que se ha desplegado un recurso para poder obtener el número de eventos ocurridos en un periodo de tiempo determinado. En esta ocasión, por sencillez, se almacenará en formato *timestamp*.

Una vez definidos todos los atributos con los que contará cada elemento de la tabla de DynamoDB, es momento de configurarlo en la consola de AWS. Para ello habrá que dirigirse a

la configuración de DynamoDB en la consola de configuración de AWS, como puede observarse en la Figura 10.

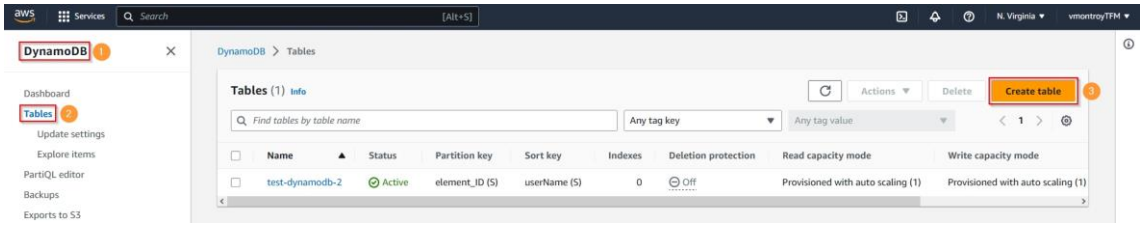


Figura 10. Creación de una nueva tabla de DynamoDB [Creación propia].

La configuración de la tabla es muy sencilla. Bastará con introducir un nombre para la nueva tabla y definir una clave de partición, en esta ocasión ha sido "elementID" (Figura 11). De manera opcional puede crearse una clave de ordenación, como podría ser el nombre del usuario. Los demás elementos pueden dejarse con la configuración por defecto.

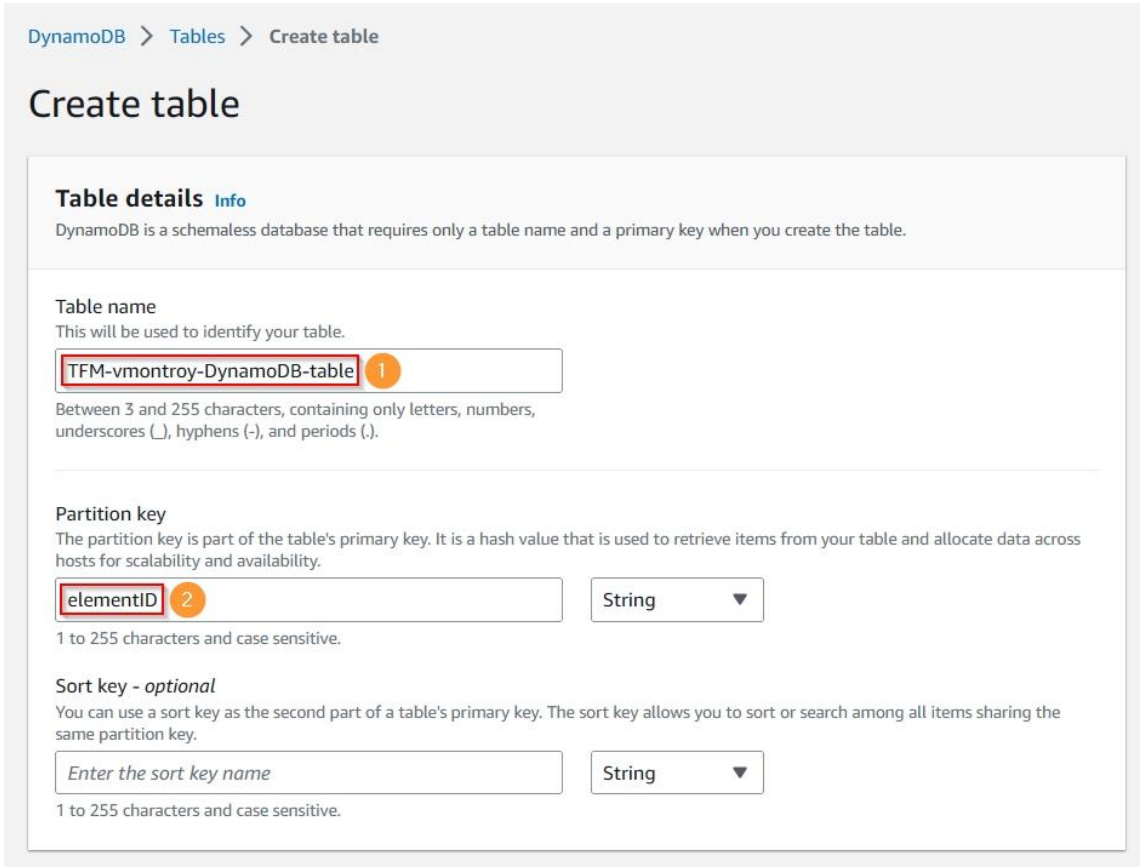


Figura 11. Configuración de la nueva tabla [Creación propia].

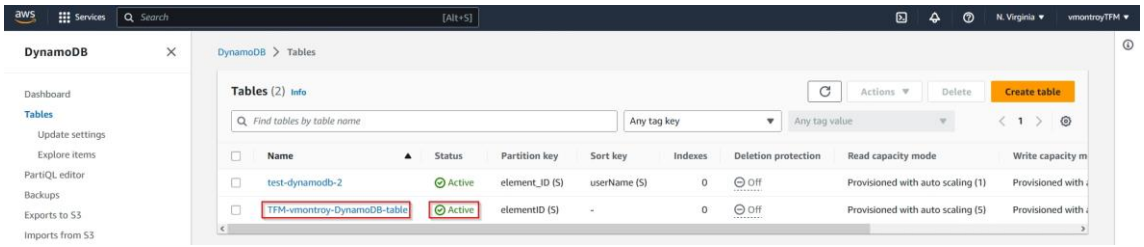


Figura 12. Nueva tabla de DynamoDB en funcionamiento [Creación propia].

Una vez creada la tabla se observará que solamente se cuenta con uno de los atributos que se han definido anteriormente, "elementID". Esto ocurre porque, al tratarse de una base de datos no relacional, los atributos de cada elemento pueden ser distintos y crearse de forma dinámica. Es posible definir un esquema de datos para una tabla, con sus atributos y el tipo de datos que contendrá cada uno, pero en esta ocasión, con el objetivo de simplificar el proceso y facilitar la configuración, no se ha definido un esquema para la tabla.

Respecto a la capa gratuita, AWS ofrece 25 GB de almacenamiento en DynamoDB, así como 25 unidades de capacidad de escritura (WCU) y de lectura (RCU) de datos [24], que determinan el número de operaciones por unidad de tiempo que admite la tabla.

4.2.3. Configuración de SNS

SNS se encargará de enviar las alertas generadas al administrador de la cuenta de AWS. Este servicio permite enviar notificaciones a través de distintas vías, como son SMS o correos electrónicos. En esta ocasión se configurará para enviar un correo electrónico, el cuál contendrá el nombre del usuario que ha generado la alerta, así como los identificadores de los recursos que han sido desplegados. De este modo, el administrador podrá localizar de forma sencilla los elementos desplegados y detener su ejecución o eliminarlos, así como bloquear la cuenta del usuario implicado o ponerse en contacto con este.

Para configurar este servicio habrá que dirigirse a la configuración de SNS en la consola de AWS. El primer pasó será la creación de un nuevo *topic*, como se observa en la Figura 13. Podemos definir un *topic* como una cola de eventos que serán posteriormente transmitidos a todas las suscripciones que existan para ese *topic*.

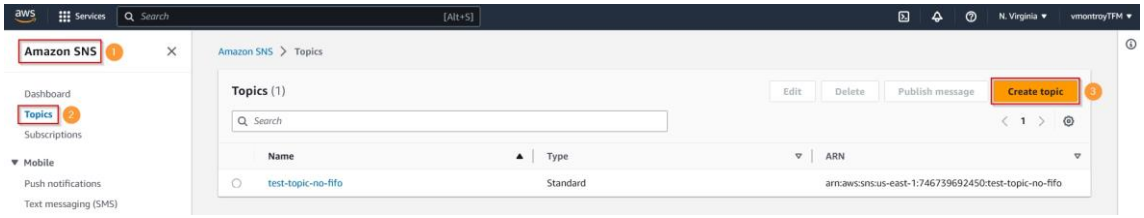


Figura 13. Creación de un nuevo topic de SNS [Creación propia].

En la Figura 14 se define la configuración para el nuevo *topic*. Para poder alertar empleando el correo electrónico habrá que seleccionar el tipo estándar y nombrar el nuevo *topic*. El último de los campos definirá el nombre que aparecerá en el remitente del correo electrónico, en esta ocasión "Alerta AWS". La configuración restante puede dejarse por defecto.

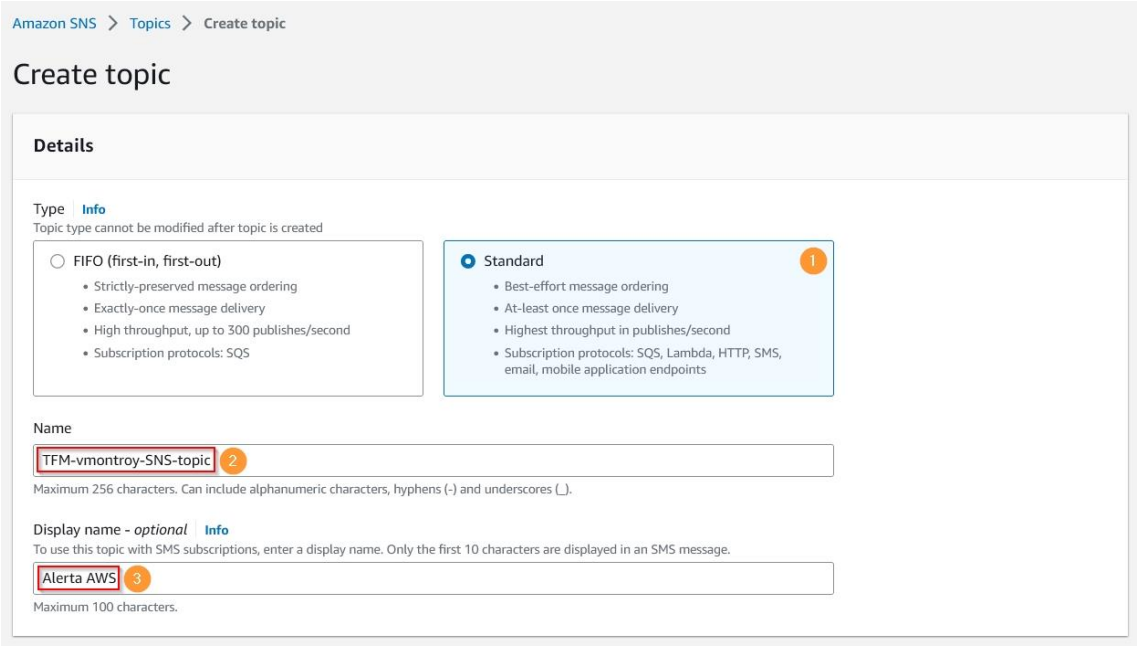


Figura 14. Configuración del nuevo topic [Creación propia].

Con el nuevo *topic* creado, habrá que añadir una nueva suscripción (Figura 15). Existen distintos medios por los que se puede enviar una notificación, pero en este caso se ha elegido el correo electrónico. En la Figura 16 se observa la configuración de esta nueva suscripción. En el campo de protocolo se elegirá "Email", aunque existe otra opción para notificaciones por correo electrónico, "Email-JSON". Este segundo tipo envía un mensaje en formato JSON con información adicional y puede ser de utilidad si quieren automatizarse tareas posteriores a la recepción de la alerta. Tras introducir la dirección de correo electrónico a la cuál llegará la notificación y dejando el resto de las opciones por defecto, se creará la suscripción.

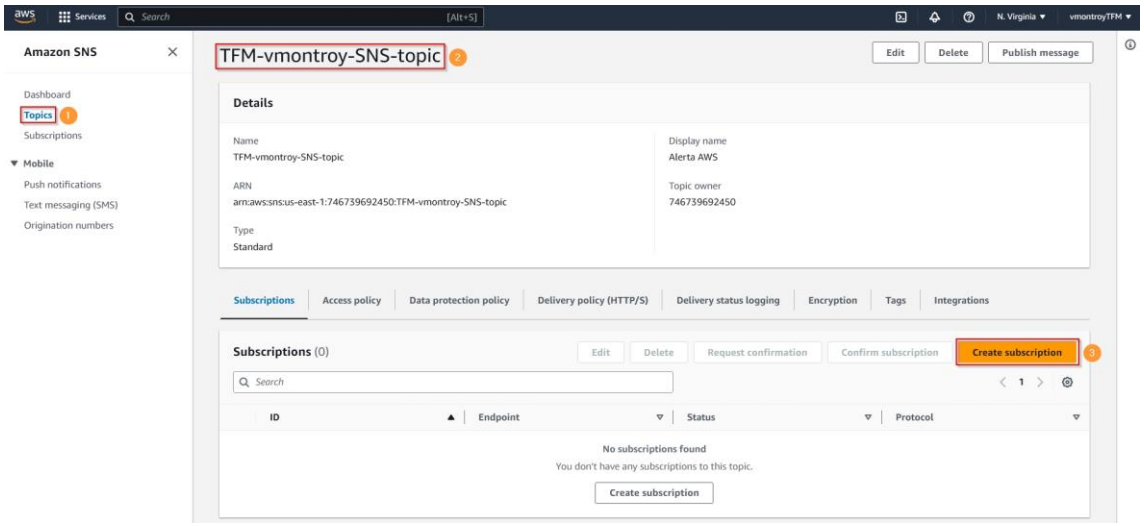


Figura 15. Nueva suscripción en el topic [Creación propia].

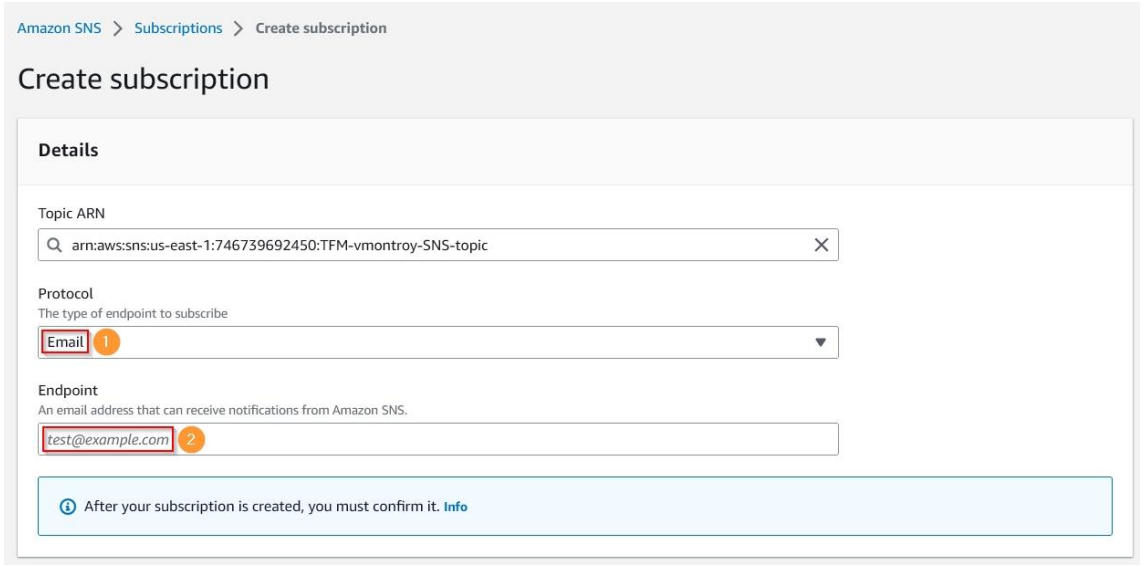


Figura 16. Configuración de la suscripción [Creación propia].

Tras la creación de la suscripción, será necesario confirmarla. A la dirección de correo electrónico introducida durante la creación de la suscripción llegará un correo electrónico como el de la Figura 17. Para aceptar la suscripción al *topic* solamente habrá que seguir el enlace de confirmación de la suscripción.

Figura 17. Correo electrónico de confirmación de la suscripción [Creación propia].

En la Figura 18 se observa el nuevo *topic* y la suscripción a este. Si se desea notificar a más de una persona, pueden añadirse nuevas suscripciones siguiendo el proceso detallado.

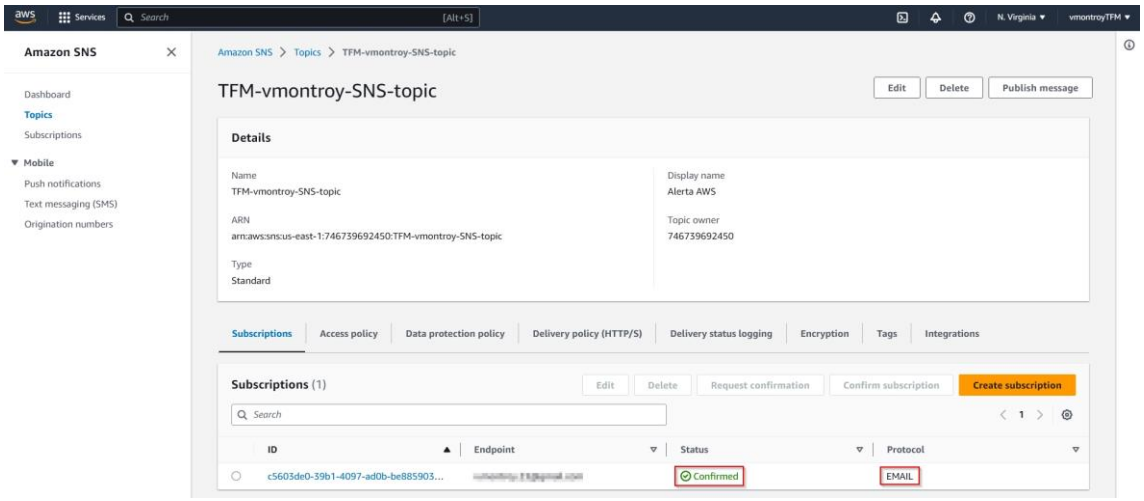


Figura 18. Suscripción al topic confirmada [Creación propia].

Dentro de su capa gratuita, AWS ofrece hasta 1000 notificaciones por correo electrónico al mes [25].

4.2.4. Configuración de Lambda

AWS Lambda será el servicio que nos permitirá procesar los eventos y alertar al administrador en caso de ser necesario. Cuando la función Lambda reciba un nuevo evento, lo procesará y extraerá la información de interés de dicho evento. Seguidamente, introducirá la información extraída en la tabla de DynamoDB y realizará una consulta a esta misma tabla para determinar si el usuario asociado está dentro de los límites permitidos o ha sobrepasado alguna de las reglas. En caso de que haya desplegado más recursos de los permitidos para el periodo de

tiempo definido, enviará una notificación al *topic* SNS que se ha creado y este la trasmitirá al administrador vía correo electrónico.

La fuente de eventos para la función Lambda será el servicio EventBridge de AWS. La configuración de este servicio se detallará en el siguiente apartado, ya que para configurarlo debe existir la función Lambda con la cual se comunicará.

En este apartado se detallará la integración de Cloud Custodian con AWS Lambda. En esta ocasión, se han desplegado políticas de Cloud Custodian que se encargan de etiquetar de forma automatizada recursos en el momento de su despliegue. Dicha etiqueta contendrá el nombre del usuario que ha desplegado el recurso, haciendo más sencilla la identificación de los recursos asociados a cada usuario.

Respecto a la capa gratuita, AWS ofrece hasta 1 millón de solicitudes gratuitas por mes, así como 3,2 millones de segundos de tiempo de ejecución de las funciones desplegadas [26].

4.2.4.1. Función Lambda principal

El procesado de los eventos se centralizará en una sola función Lambda. Para configurarla habrá que dirigirse a la configuración de Lambda en la consola de configuración de AWS (Figura 19).

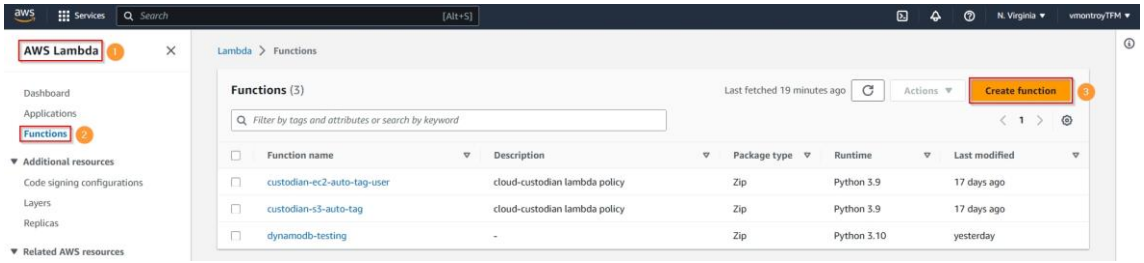


Figura 19. Creación de una nueva función Lambda [Creación propia].

Se configurará la nueva función desde cero, como se observa en la Figura 20. Habrá que introducir un nombre para la función, así como el lenguaje de programación que se empleará para programarla. En esta ocasión se va a trabajar con Python en su versión 3.10. Los elementos restantes de la configuración pueden dejarse por defecto.

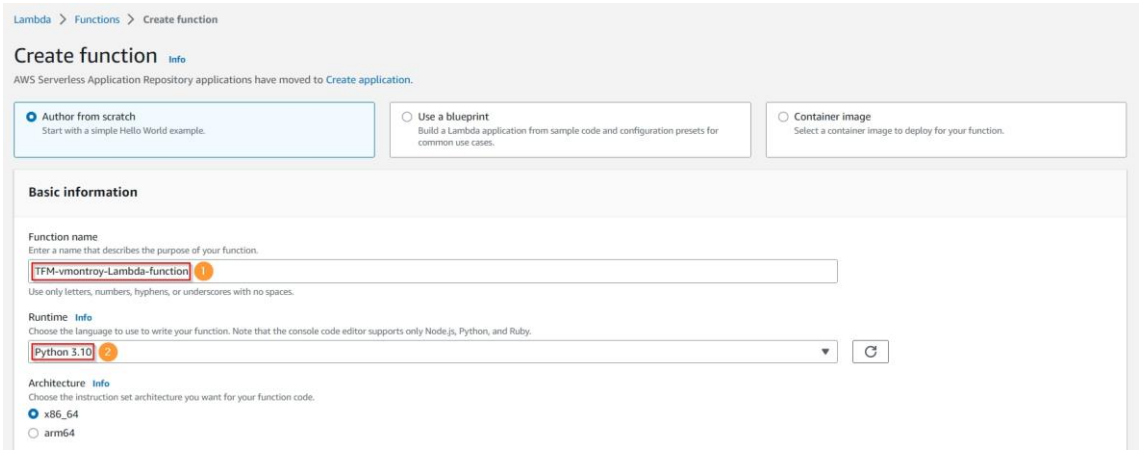


Figura 20. Configuración de la nueva función [Creación propia].

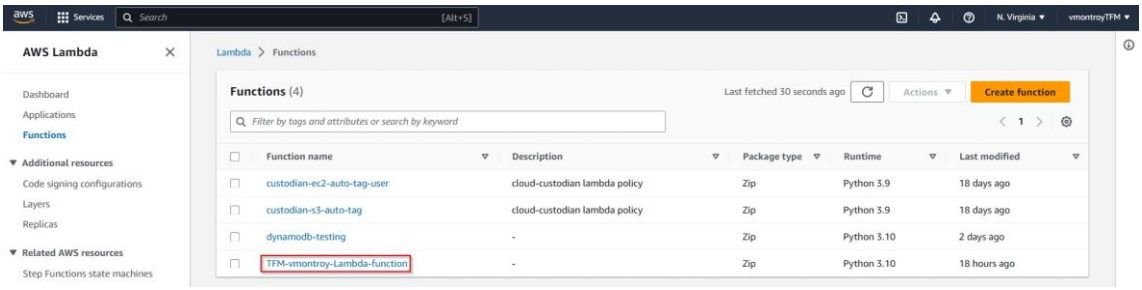


Figura 21. Nueva función Lambda creada [Creación propia].

Una vez creada la nueva función, al acceder a ella se podrá encontrar el código fuente que se ejecutará cuando se dispare la función. En la Figura 22 se muestra el código fuente por defecto, contenido en el fichero "lambda_function.py". Este es el fichero que habrá que modificar para que la función lleve a cabo las funciones pertinentes.

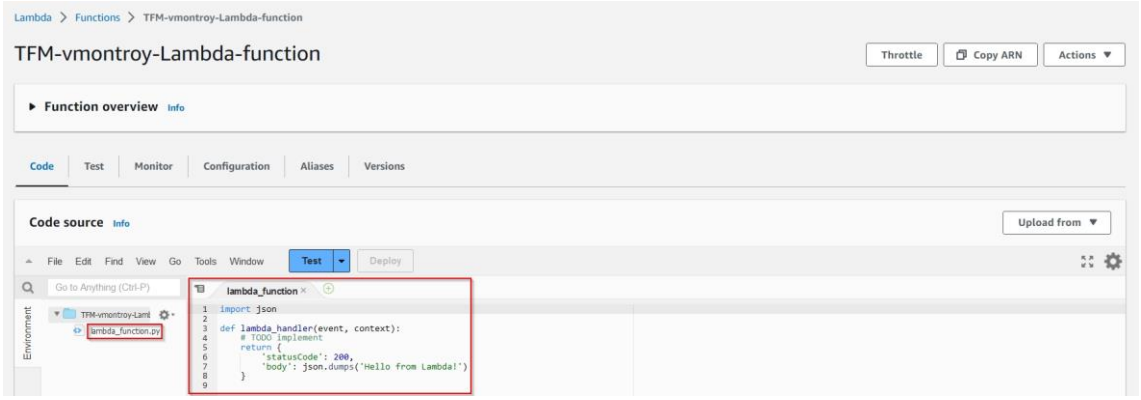


Figura 22. Código fuente por defecto de la nueva función [Creación propia].

El código fuente que se ha desarrollado se encargará de extraer la información de interés de los eventos de despliegue de una nueva instancia de EC2 (“RunInstances”) y de creación de una nuevo *bucket* de S3 (“CreateBucket”) que lleguen a la función Lambda desde el servicio EventBridge, e introducir dicha información en la tabla de DynamoDB. Tras registrar un nuevo evento, realizará una consulta a la tabla para obtener el número de eventos del mismo tipo registrados en la última hora. En caso de superarse el límite, establecido en 5 despliegues o más, se notificará al administrador mediante el servicio SNS.

Cabe destacar que es posible adaptar el código fuente a las necesidades de cada administrador. Puede aumentarse el margen de tiempo, el número de despliegues máximos o ampliar el número de eventos monitorizados. Es decir, el código proporcionado es solamente un ejemplo de uso del sistema, pudiéndose desarrollar un código fuente distinto basándose en el funcionamiento del sistema.

A continuación, se muestra el código fuente. Cada una de las tareas que realiza están comentadas en el mismo código fuente. Además, se ha añadido este mismo código en formato de texto en el Anexo I.

```python
import json
import boto3
from boto3.dynamodb.conditions import Key, Attr
import time
from datetime import datetime, timedelta
from dateutil import parser

# Se inician dos instancias para poder interactuar con DynamoDB y SNS.
dynamodb = boto3.resource('dynamodb')
client = boto3.client('sns')

# Función principal que se ejecutará cuando se dispare la función Lambda.
def lambda_handler(event, context):

    # Se obtiene la información referente al usuario y al evento.
    user_name = event['detail']['userIdentity']['userName']
    event_name = event['detail']['eventName']

    # Se establece la respuesta ante la detección del evento "RunInstances".
    if event_name == "RunInstances":

        # Se obtiene la hora de ejecución y el ID de la instancia lanzada. Puede que se lancen varias instancias EC2 al mismo tiempo.
        # En este caso solo se generará un evento y habrá que iterar dentro de él.
        if len(event['detail']['responseElements']['instancesSet']['items']) > 1:

            for i in range(len(event['detail']['responseElements']['instancesSet']['items'])):

                instance_launch_time = event['detail']['responseElements']['instancesSet']['items'][i]['launchTime']
                instance_ID = event['detail']['responseElements']['instancesSet']['items'][i]['instanceId']

                print(instance_launch_time)
                print(instance_ID)

                # Se llama a la función encargada de introducir los datos en la tabla de DynamoDB.
                database_entry(event_name, user_name, instance_launch_time, instance_ID)

        else:

            instance_launch_time = event['detail']['responseElements']['instancesSet']['items'][0]['launchTime']
            instance_ID = event['detail']['responseElements']['instancesSet']['items'][0]['instanceId']

            # Se llama a la función encargada de introducir los datos en la tabla de DynamoDB.
            database_entry(event_name, user_name, instance_launch_time, instance_ID)

        # Se obtienen el número de eventos "RunInstances" ocurridos en la última hora y los IDs asociados a las instancias EC2 desplegadas.
        events = event_count(user_name, event_name)

        # Si el número de eventos "RunInstances" es igual o superior a 5 se llamará a la función encargada de notificar al administrador.
        if events[0] >= 3:
            send_alert(user_name, event_name, events[1], "/!\ Alerta EC2")
```

```python

    # Se establece la respuesta ante la detección del evento "RunInstances".
    if event_name == "CreateBucket":

        # Se obtiene la hora de ejecución y el ID del bucket creado.
        bucket_launch_time = event['detail']['eventTime']
        bucket_ID = event['detail']['requestParameters']['bucketName']

        # La hora de creación del bucket no viene en formato timestamp, se convierte a este formato antes de introducirse en la tabla de DynamoDB.
        bucket_launch_time_norm = int(str(datetime.timestamp(parser.parse(bucket_launch_time))).split(".")[0])*1000

        # Se llama a la función encargada de introducir los datos en la tabla de DynamoDB.
        database_entry(event_name, user_name, bucket_launch_time_norm, bucket_ID)

        # Se obtienen el número de eventos "CreateBucket" ocurridos en la última hora y los IDs asociados a los buckets S3 creados.
        events = event_count(user_name, event_name)

        # Si el número de eventos "CreateBucket" es igual o superior a 5 se llamará a la función encargada de notificar al administrador.
        if events[0] >= 5:
            send_alert(user_name, event_name, events[1], "/!\ Alerta S3")

# Se define la función encargada de introducir la información en la tabla de DynamoDB.
def database_entry(event_name, user_name, launch_time, element_ID):

    # Se define un objeto Table para poder interactuar con la tabla de DynamoDB.
    table = dynamodb.Table('TFM-vmontroy-DynamoDB-table')

    # Se introduce la información en las distintas columnas de la tabla DynamoDB.
    table.put_item(
        Item={
            'elementID': element_ID,
            'userName': user_name,
            'eventName': event_name,
            'eventTime': launch_time
            }
        )

# Se define la función encargada de consultar el número de eventos ocurridos a la tabla de DynamoDB.
def event_count(user_name, event_name):

    # Se define un objeto Table para poder interactuar con la tabla de DynamoDB.
    table = dynamodb.Table('TFM-vmontroy-DynamoDB-table')

    # Como quiere obtenerse el número de ocurrencias de los eventos durante la última hora, habrá que obtener el timestamp de una hora antes de la ejecución
    # de la función Lambda.
    time_1h_ago_timestamp=int(str(datetime.timestamp(datetime.now()-timedelta(hours=1))).split(".")[0])*1000

    # Se obtiene el número de eventos para el usuario implicado en la última hora.
    event_count = table.scan(FilterExpression=Attr('userName').eq(user_name) & Attr('eventName').eq(event_name) & Attr('eventTime').gt(time_1h_ago_timestamp))

    # Se obtiene el número de eventos ocurridos en la última hora.
    events = event_count['Count']

    # Se define una lista para almacenas los IDs de los elementos implicados.
    element_IDs = []

    # Se rellena la lista con los IDs.
    for i in range(events):
        element_IDs.append(event_count['Items'][i]['elementID'])

    # La función devuelve el número de eventos ocurridos y la lista de IDs implicados.
    return(events, element_IDs)

# Se define la función encargada de notificar al administrados las alertas.
def send_alert(user_name, event_name, element_IDs, subject):

    # Se crea el cuerpo del correo electrónico que recibirá el administrador.
    message = f"""
    El usuario {user_name} ha superado el límite establecido para el evento "{event_name}".

    A continuación se listan los IDs de los elementos deplegados:

    {*element_IDs,}
    """

    # Se envía el mensaje al topic de SNS encargado de mandar el correo electrónico al administrador.
    send_alert = client.publish(TopicArn='arn:aws:sns:us-east-1:746739692450:TFM-vmontroy-SNS-topic', Message=message, Subject=subject)
```

Figura 23. Código fuente de la función Lambda principal [Creación propia].

Una vez la función ya está configurada y el código fuente desarrollado, habrá que modificar los permisos con los que cuenta la función, ya que esta interactuará con distintos elementos dentro de AWS y sin los permisos adecuados no se podrá llevar a cabo esta interacción.

Para ello, cuando se crea una nueva función, es posible añadir un rol de IAM [27] (*Identity and Access Management*) previamente configurado, o puede crearse uno nuevo junto con la función. Esta es la opción por defecto, por tanto, en esta ocasión, se ha creado uno nuevo que habrá que configurar con los permisos adecuados.

Los únicos permisos que vienen por defecto al crearse el rol de IAM asociado a la función son aquellos que le permiten generar *logs*, para poder monitorizar su funcionamiento. Además de estos, habrá que añadir los permisos necesarios para interactuar con DynamoDB y SNS. Para ello, como muestra la Figura 24, desde la propia configuración de la función puede accederse a la configuración del rol asociado.

Figura 24. Acceso al rol de IAM desde la configuración de la función [Creación propia].

En la Figura 25 se muestra la configuración del rol. El nombre que se le asigna al rol cuando se crea de forma automática es el nombre de la función, seguido de "role" y una cadena alfanumérica. La única política que se crea por defecto es "AWSLambdaBasicExecutionRole", la cual permitirá a la función escribir *logs*. Para añadir nuevos permisos habrá que dirigirse a la pestaña de añadir permisos y crear una nueva política que agrupe todos los permisos necesarios para que la función Lambda pueda trabajar. Para ello, se va a seguir el principio del privilegio mínimo y solamente se definirán los permisos estrictamente necesarios: añadir nuevos elementos a la tabla de DynamoDB, realizar consultas a la tabla y publicar mensajes en el *topic* de SNS. En la Figura 26 se observa cómo se inicia la creación de una nueva política.

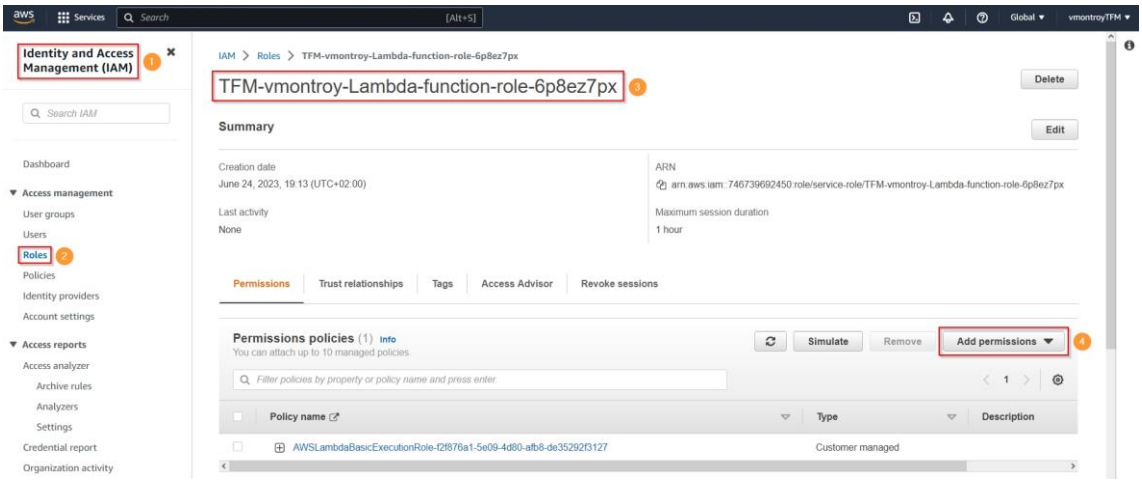


Figura 25. Configuración de los permisos asociado al rol de IAM [Creación propia].

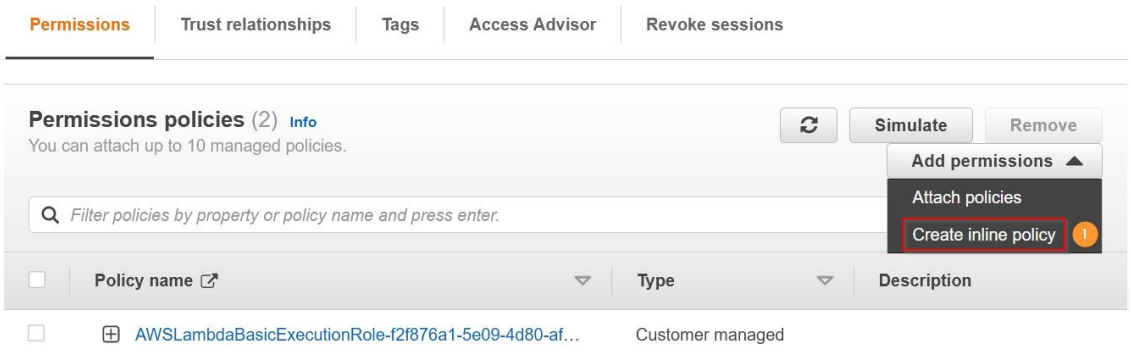


Figura 26. Creación de una nueva política con los permisos necesarios [Creación propia].

Tras comenzar la creación de una nueva política, nos dará la posibilidad de crearla con una interfaz gráfica o introduciendo el texto en formato JSON que contenga los permisos. En esta ocasión se ha optado por definir los permisos en formato JSON, por lo que bastará con copiar la definición de la política y crearla para contar con los nuevos permisos. A continuación, se muestra la política que se ha configurado:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "TFMLambdaPolicy",
            "Effect": "Allow",
            "Action": [
                "sns:Publish",
                "dynamodb:PutItem",
                "dynamodb:Scan"
            ],
            "Resource": [
                "arn:aws:sns:us-east-1:746739692450:TFM-vmontroy-SNS-topic",
                "arn:aws:dynamodb:us-east-1:746739692450:table/TFM-vmontroy-
DynamoDB-table"
            ]
        }
    ]
}
```

4.2.4.2. Integración de Cloud Custodian con AWS Lambda

Como se ha detallado en el apartado 3.2.1., Cloud Custodian es una herramienta que permite al propietario de una cuenta de AWS aplicar políticas de gestión y gobernanza. Esta herramienta puede integrarse con AWS Lambda, pudiendo desplegar políticas de Cloud Custodian como funciones Lambda individuales.

En esta ocasión, se ha optado por el despliegue de dos políticas de Cloud Custodian que se encargan de etiquetar de forma automatizada las nuevas instancias de EC2 y *buckets* de S3, añadiendo la etiqueta “owner”, que contiene el nombre del usuario que ha desplegado el

recurso. De este modo, el propietario de la cuenta puede consultar rápidamente quién ha desplegado cada recurso. Se ha optado por desplegar solamente dos políticas a modo de ejemplo, una para EC2 y la otra para S3, pero estas políticas también podrían configurarse para otros recursos.

Para poder desplegar las funciones Lambda con las políticas de Cloud Custodian habrá que realizar los siguientes pasos:

1. En un equipo con sistema operativo Windows 10, se creará un entorno virtual de Python. Para ello puede consultarse la guía oficial de Python en [28].

2. Una vez desplegado el entorno virtual, habrá que descargar la librería de Cloud Custodian para Python con el comando: `pip install c7n`.

3. Desde este punto ya contamos con la herramienta operativa, pero para poder interactuar con el entorno de AWS habrá que añadir a las variables de entorno de Windows la información necesaria para que el equipo pueda interactuar con la API de AWS. En [29] se detalla el proceso de creación de esta clave en AWS, la cual contará con un identificador y un secreto. El secreto solamente será visible en el momento de su creación, por lo que habrá que guardarlo en un lugar seguro. Una vez creada la clave, se añadirá a las variables de entorno de Windows con los comandos: `set AWS_ACCESS_KEY_ID=<ID_VALUE>` y `set AWS_SECRET_ACCESS_KEY=<SECRET_VALUE>`.

4. Una vez configurado el entorno, habrá que definir las políticas de Cloud Custodian y validarlas, para asegurar que no existan errores. Si está todo correcto podrán desplegarse en la cuenta de AWS.

   A continuación, se muestran las dos políticas en formato YAML creadas para el etiquetado automático de los recursos:

```
policies:
- name: TFM-vmontroy-CC-policy-ec2-auto-tag-user
  resource: ec2
  mode:
    type: cloudtrail
    role: arn:aws:iam::{account_id}:role/TFM-vmontroy-resource-auto-
tagger-role
    events:
     - RunInstances
  filters:
    - tag:owner: absent
  actions:
    - type: auto-tag-user
      tag: owner
```

```
policies:
- name: TFM-vmontroy-CC-policy-s3-auto-tag-user
  resource: aws.s3
  mode:
    type: cloudtrail
    role: arn:aws:iam::{account_id}:role/TFM-vmontroy-resource-auto-
```

```
tagger-role
    events:
      - source: s3.amazonaws.com
        event: CreateBucket
        ids: requestParameters.bucketName
  filters:
     - tag:owner: absent
  actions:
    - type: auto-tag-user
      tag: owner
```

5. Para comprobar si una política es correcta, dentro del entorno virtual, puede emplearse el comando: `custodian validate <policy.yaml>`. Si la política es correcta, solo quedará un último paso antes de desplegar la función. Como se observa en las definiciones de las políticas en el punto anterior, estas tienen asociado un rol de IAM. Este rol deberá existir en el entorno de AWS antes del despliegue ya que, en caso contrario, Cloud Custodian mostrará un error.

6. El proceso de creación del rol de IAM es sencillo. Para ello, habrá que dirigirse a la configuración de IAM en la consola de configuración de AWS y comenzar la creación de un nuevo rol (Figura 27).

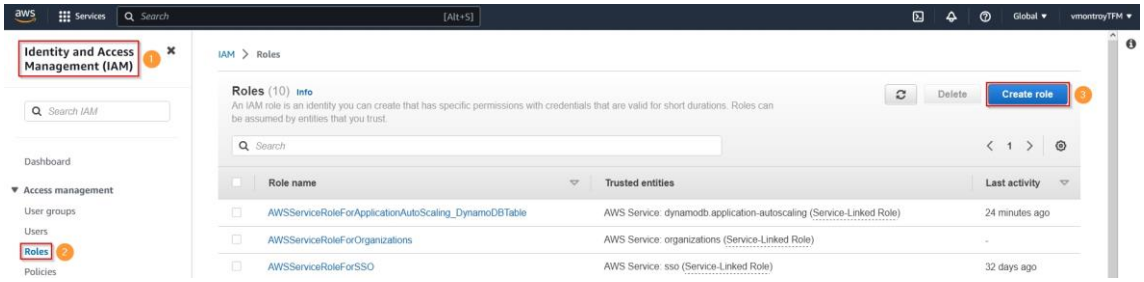


Figura 27. Creación de un nuevo rol de IAM [Creación propia].

En la configuración del rol se seleccionará Lambda como caso de uso, como se observa en la Figura 28.

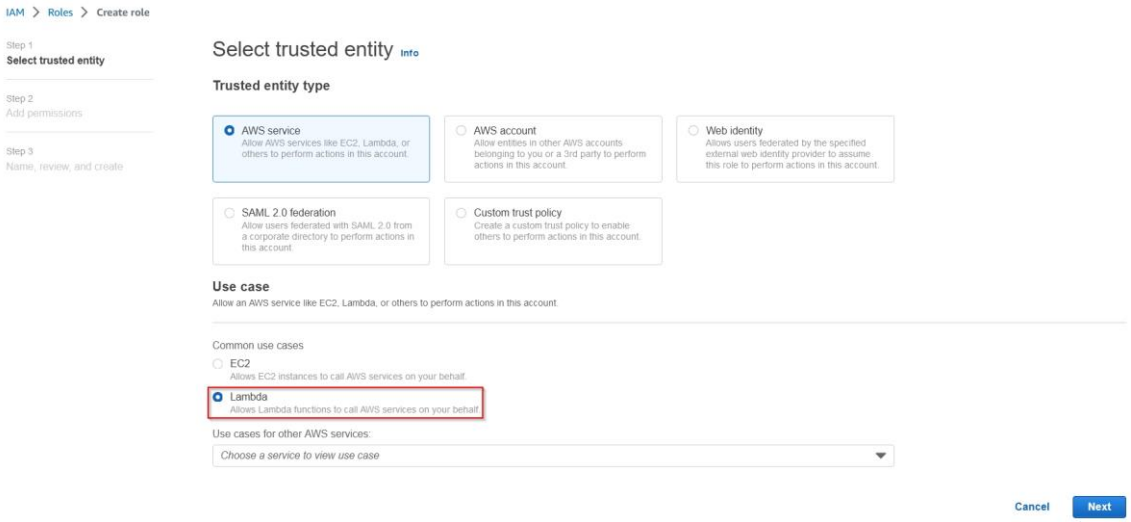


Figura 28. Configuración del nuevo rol de IAM – paso 1 [Creación propia].

El siguiente paso será añadir las políticas necesarias al nuevo rol. Como los recursos que se van a etiquetar serán instancias de EC2 y *buckets* de S3, se creará una política que permita etiquetar dichos recursos siguiendo el principio del privilegio mínimo. Además, se añadirá la política "AWSLambdaBasicExecutionRole", la cual permitirá a la función Lambda crear *logs*. Para añadir los permisos, en el paso 2 (Figura 29) habrá que buscar la política "AWSLambdaBasicExecutionRole" y añadirla. A continuación, habrá que seleccionar la opción de crear una nueva política y se añadirá la siguiente definición en formato JSON:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "TFMAllowAutoTagging",
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeInstances",
                "ec2:CreateTags",
                "s3:ListAllMyBuckets",
                "s3:GetBucketTagging",
                "s3:PutBucketTagging"
            ],
            "Resource": "*"
        }
    ]
}
```

Figura 29. Configuración del nuevo rol de IAM – paso 2 [Creación propia].

El último paso será añadir un nombre al rol y crearlo (Figura 30). Hay que tener en cuenta que el nombre que se le dé al rol debe coincidir con el que aparece en las políticas de Cloud Custodian. En la Figura 31 se observa el nuevo rol con las dos políticas añadidas.

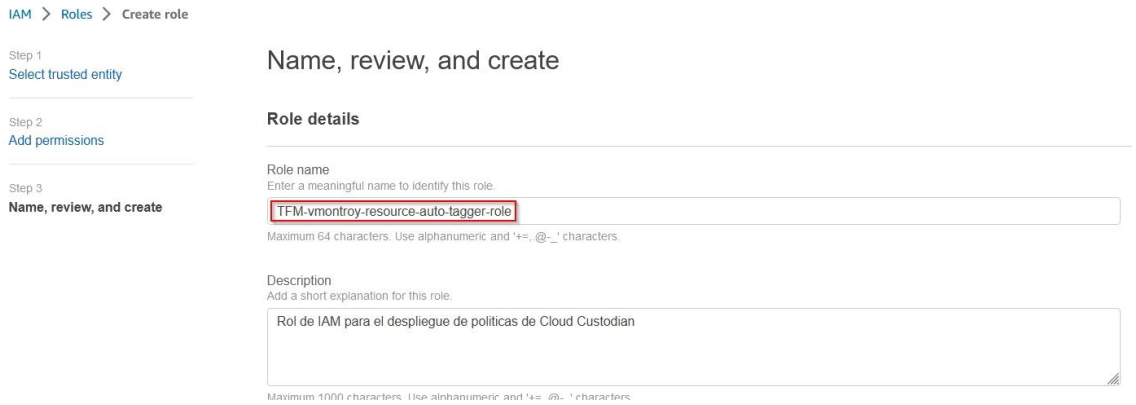


Figura 30. Configuración del nuevo rol de IAM – paso 3 [Creación propia].

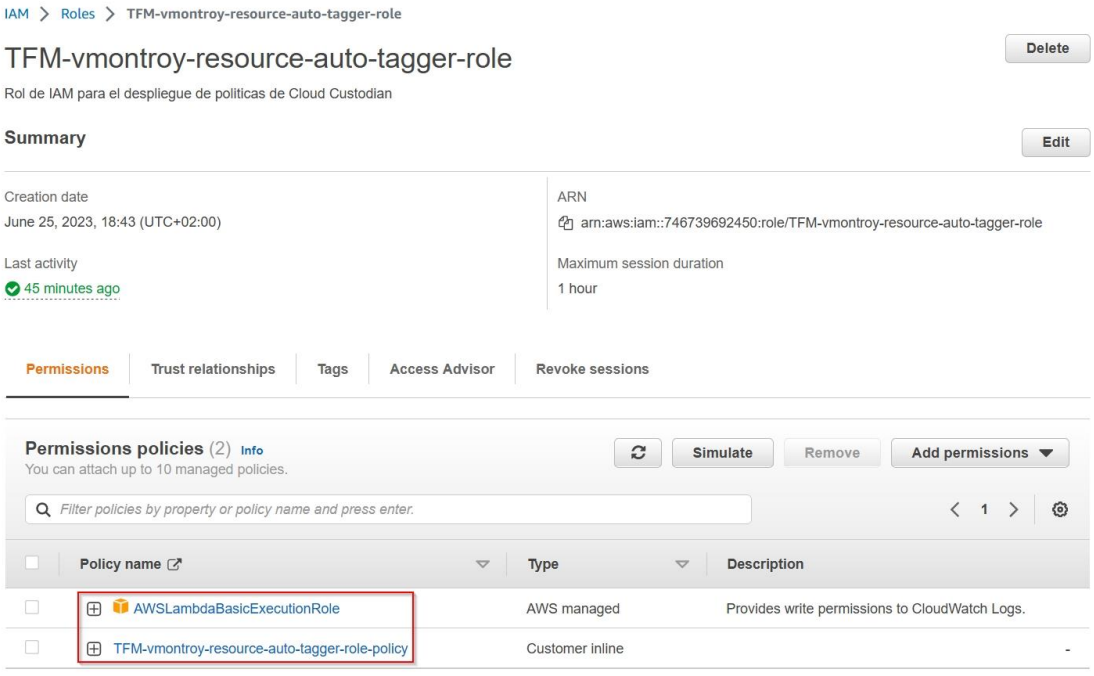


Figura 31. Nuevo rol de IAM creado con los permisos configurados [Creación propia].

7. Tras la creación y configuración del nuevo rol de IAM, el último paso será desplegar las políticas en AWS. Para ello, dentro del entorno virtual, se emplearán los comandos: `custodian run -s Output -r us-east-1 EC2-auto-tag-policy.yaml` y `custodian run -s Output -r us-east-1 S3-auto-tag-policy.yaml`, siendo "us-east-1" la región donde se desplegará la función Lambda y "Output" el directorio donde se guardarán los logs de la ejecución del comando.

Tras seguir los pasos anteriores, en la cuenta de AWS se habrán creado dos nuevas funciones Lambda, así como dos nuevas reglas de EventBridge, las cuales se encargarán de llamar a las funciones Lambda cuando se despliegue una nueva instancia de EC2 o un nuevo *bucket* de S3. En las Figuras 32 y 33 se observan los nuevos recursos desplegados.

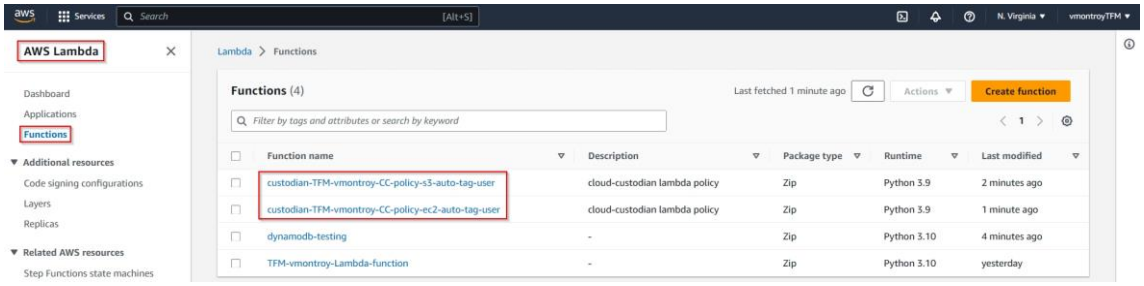


Figura 32. Funciones Lambda desplegadas por Cloud Custodian [Creación propia].

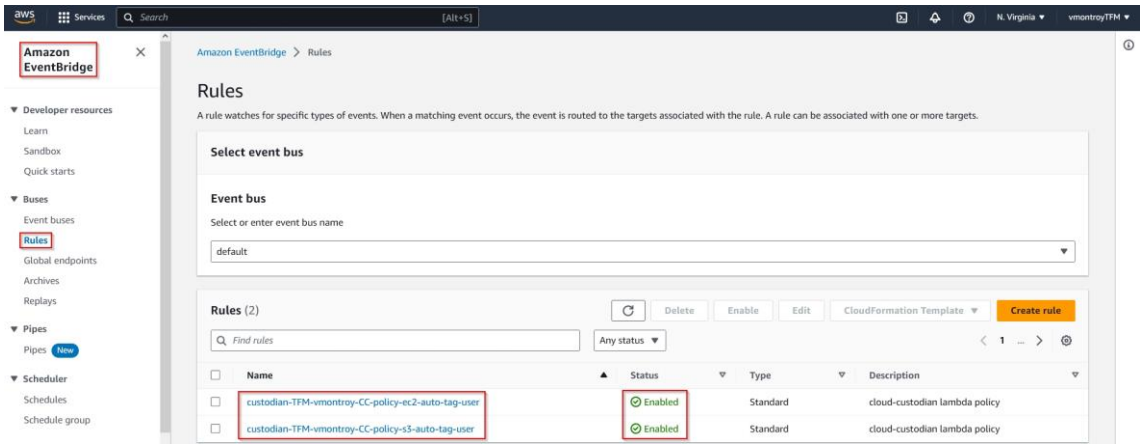


Figura 33. Reglas de EventBridge creadas por Cloud Custodian [Creación propia].

4.2.5. Configuración de EventBridge

El último servicio que habrá que configurar será EventBridge, el cual permitirá capturar solamente los eventos específicos sobre los que se quiera hacer un seguimiento y transmitirlos a la función Lambda principal para su procesado. En esta ocasión se configurará para capturar los eventos asociados a la creación de instancias de EC2 y *buckets* de S3.

El primer paso para crear las nuevas reglas será acceder a la configuración de EventBridge en la consola de configuración de AWS y comenzar la creación de una nueva regla (Figura 34).

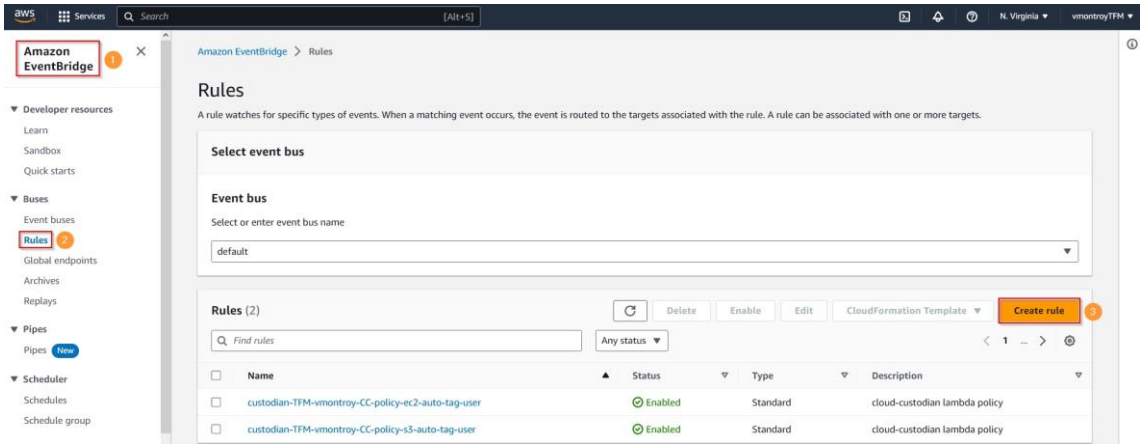


Figura 34. Creación de una nueva regla de EventBridge [Creación propia].

En primer lugar, habrá que darle un nombre a la nueva regla (Figura 35). El siguiente paso será definir la fuente de los eventos que monitorizará la regla. Por defecto, esta fuente son los eventos generados por los servicios de AWS, como se observa en la Figura 36.

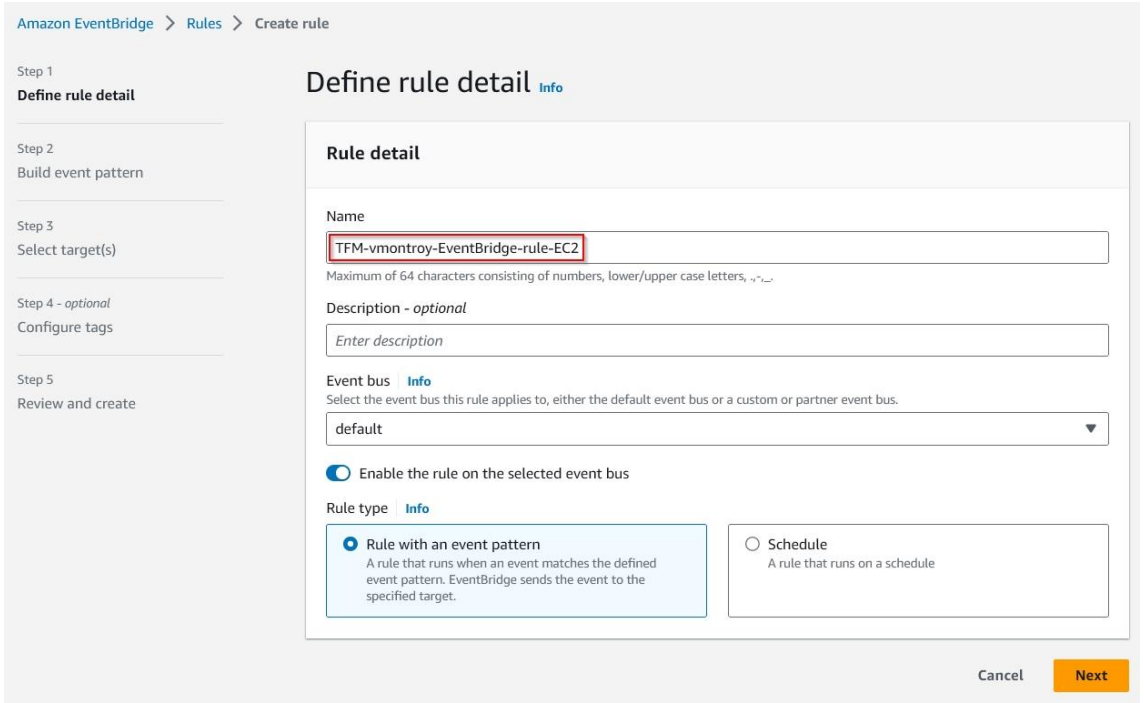


Figura 35. Configuración de la nueva regla - paso 1 [Creación propia].

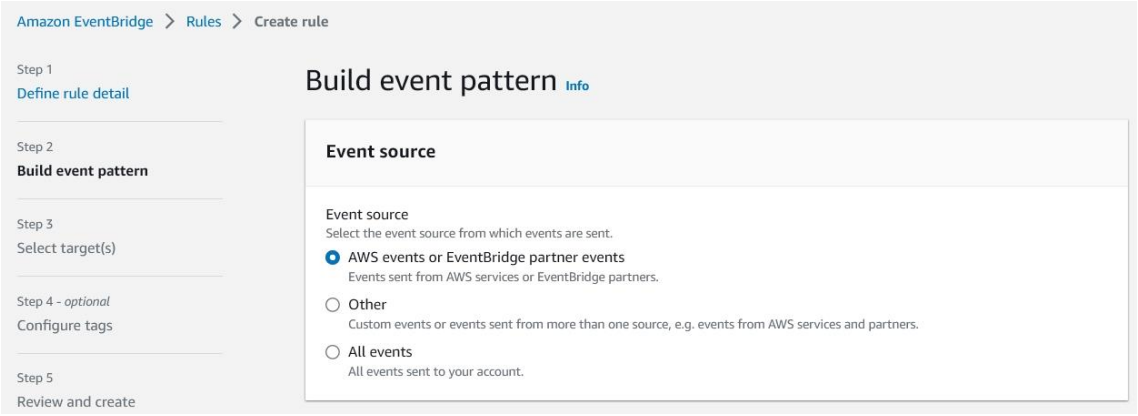


Figura 36. Configuración de la nueva regla - paso 2 [Creación propia].

En el paso 2 (Figura 36), tras elegir la fuente del evento, habrá que definir el patrón que detectará la regla de EventBridge. Para esta primera regla, EventBridge deberá capturar todas las llamadas a la API de AWS registradas en CloudTrail cuya fuente sea EC2 y quedarse solamente con los eventos del tipo “RunInstances”. Este evento es el que se genera cada vez que se despliega una nueva instancia de EC2. En la Figura 37 se muestra el proceso de configuración del patrón.

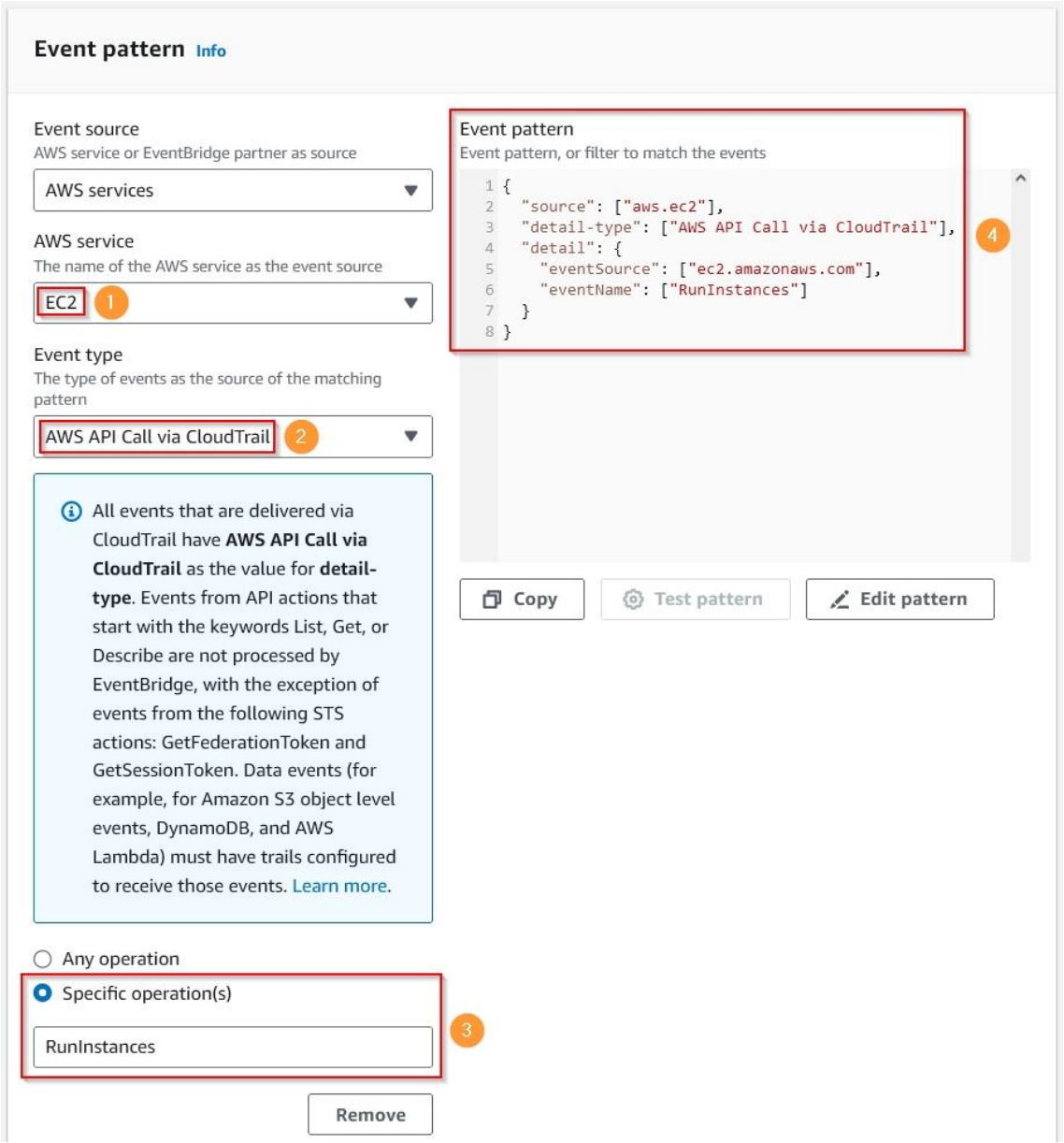


Figura 37. Definición del evento de EC2 que monitorizará la regla [Creación propia].

A continuación, habrá que definir el destino hacia el que la regla enviará los eventos capturados. En esta ocasión, el destino será la función Lambda principal, previamente creada y configurada. En la Figura 38 se observa la selección de la función Lambda "TFM-vmontroy-Lambda-function" como destino. Tras la definición del destino, los demás pasos se pueden dejar configurados por defecto y crear la nueva regla.

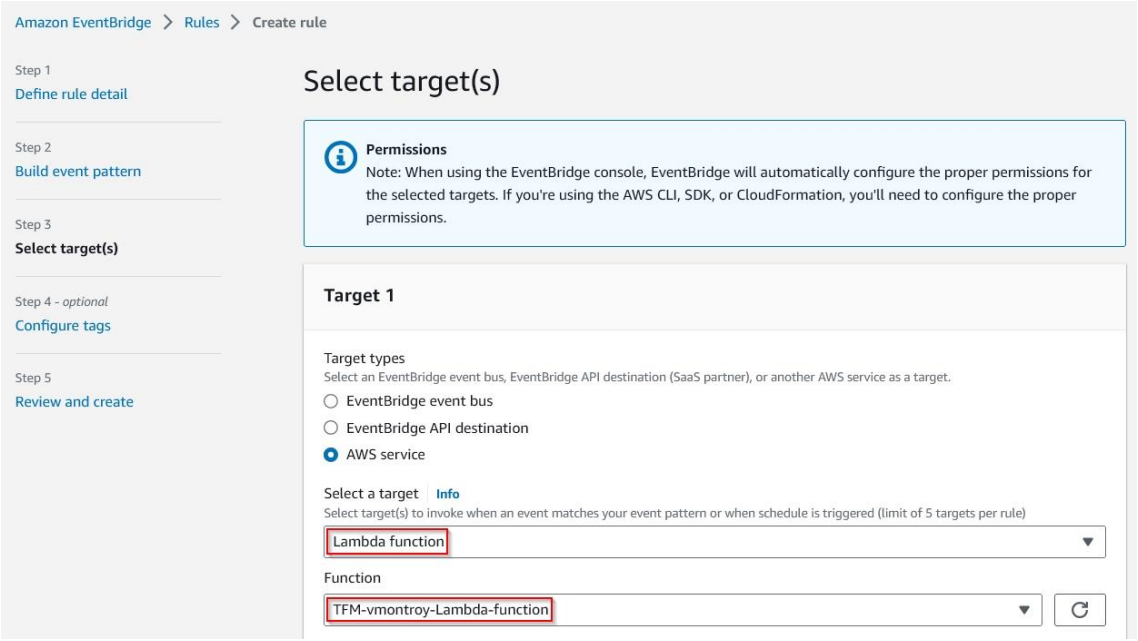


Figura 38. Configuración de la nueva regla - paso 3 [Creación propia].

De este modo puede configurarse una regla de EventBridge. Como en nuestro sistema se van a monitorizar tanto el despliegue de instancias de EC2 como de *buckets* de S3, habrá que configurar una segunda regla para capturar los eventos de creación de nuevos *buckets*. Para ello, basta con seguir los pasos descritos en este apartado, modificando solamente el nombre de la regla y el patrón de eventos que capturará, el cual se define en la Figura 39.

La capa gratuita de AWS permite capturar los eventos de modificación de estado publicados por los servicios de AWS sin coste [30].

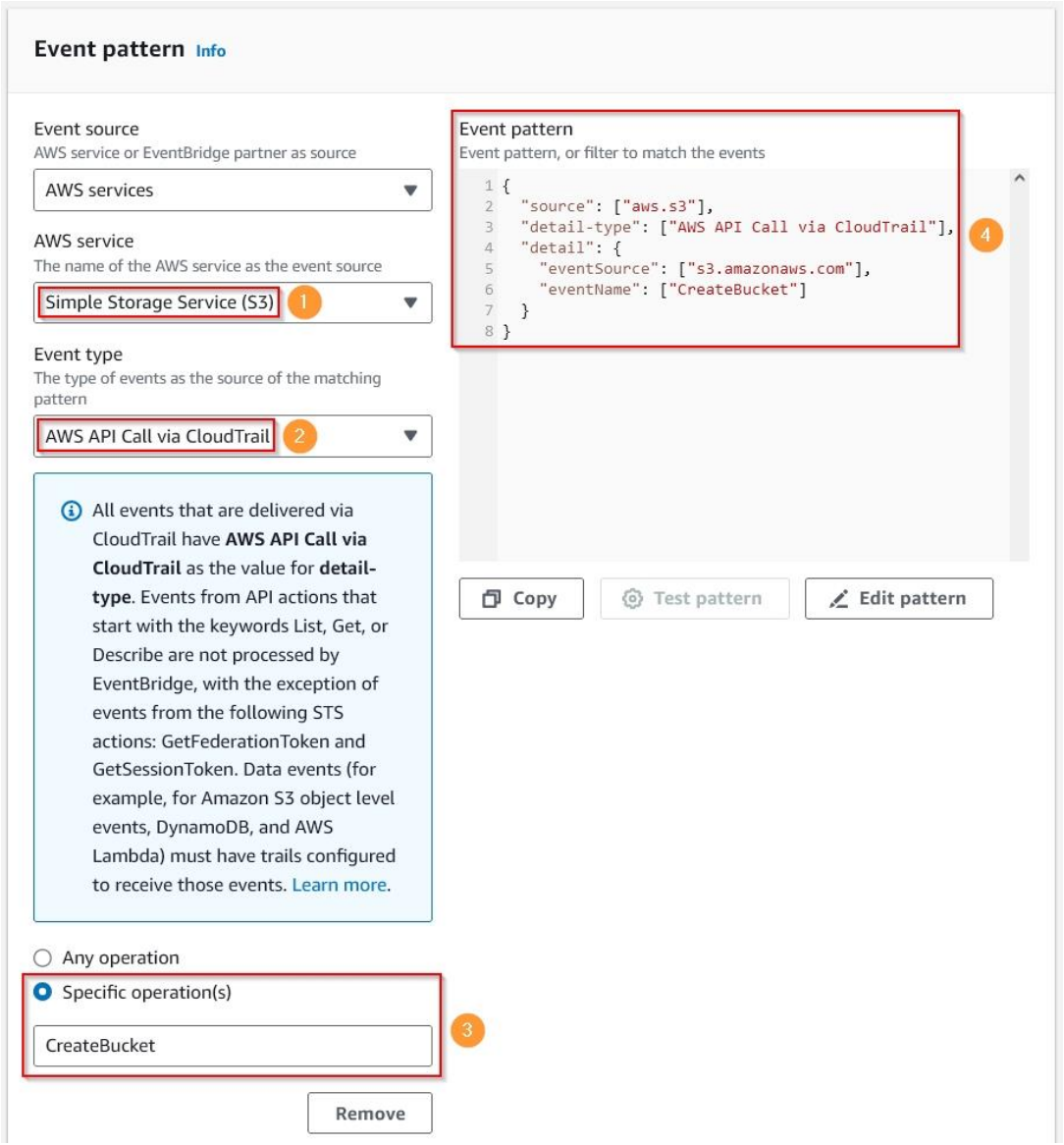


Figura 39. Definición del evento de S3 que monitorizará la regla [Creación propia].

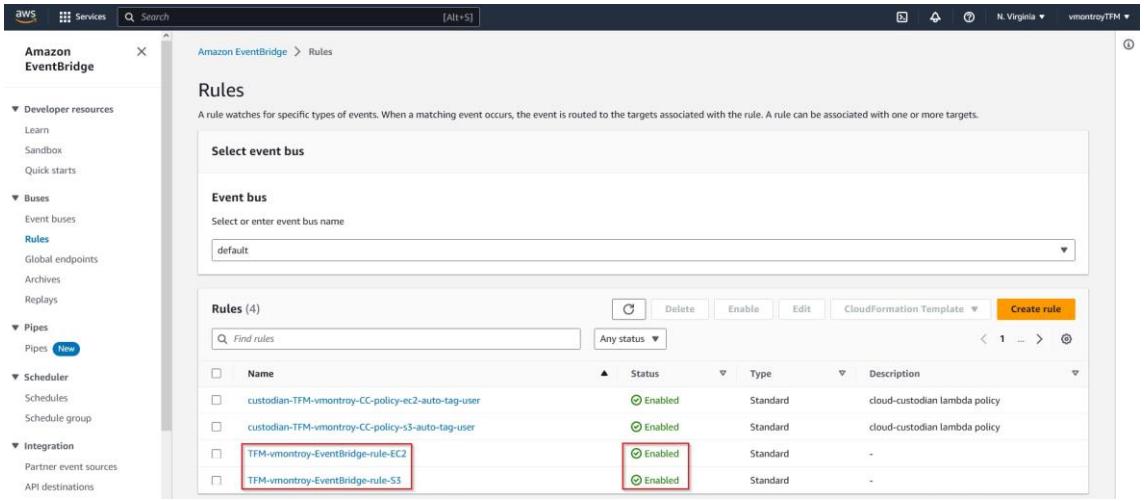


Figura 40. Nuevas reglas de EventBridge creadas [Creación propia].

5. Pruebas del sistema

Tras la implementación de la arquitectura propuesta en una cuenta de AWS, en este apartado se van a realizar una serie de pruebas para mostrar el funcionamiento del sistema propuesto.

El objetivo inicial del proyecto era implementar un sistema de detección de comportamientos anómalos por parte de los usuarios de una cuenta de AWS, como puede ser el despliegue de múltiples instancias EC2 en un periodo de tiempo determinado, y actuar de forma reactiva, es decir, permanecer a la espera mientras que no se produzca ningún evento que pueda resultar de interés, reduciendo de forma notable el coste asociado a la solución. Para ello, se han seleccionado servicios de AWS que basan su funcionamiento en el modelo *serverless*.

Con el objetivo de realizar las pruebas del sistema, se ha creado un nuevo usuario llamado "TFM_User-1", cuyos permisos se limitan a la interacción con los servicios EC2 y S3. El principio del privilegio mínimo, o limitación los permisos de cada usuario a aquellos estrictamente necesarios, es importante cuando se trabaja en entornos con distintos usuarios. Aunque se siga este principio a la hora de establecer los permisos de los usuarios, esto no evita que un usuario pueda abusar de aquellos permisos que se le han otorgado. En esta ocasión, el usuario "TFM_User-1" solo puede interactuar con dos servicios, pero no existe una limitación en cuanto al número de recursos que este puede desplegar. Es aquí donde la monitorización cobra sentido y es el motivo por el que se ha diseñado esta solución.

Un punto importante a destacar es que AWS trabaja con regiones, es decir, ubicaciones físicas donde se sitúan los servidores de AWS. Cuando se despliega un servicio, este se despliega en una región determinada. El sistema que se ha implementado en el apartado 4.2. se ha desplegado en la región "us-east-1" la cuál corresponde al Norte de Virginia [31] y solo detectará anomalías en esa misma región, es decir, si un usuario despliega más instancias de las permitidas, pero lo hace en otra región, el sistema no lo detectará. Para evitar que esto ocurra, se ha limitado al usuario "TFM_User-1" a trabajar solamente en la región "us-east-1" [32]. Este no podrá desplegar ningún recurso fuera de la región especificada. Hay que tener en cuenta este detalle a la hora de definir los usuarios o de desplegar es sistema. Si se requiere trabajar en más de una región, es posible desplegar el sistema en varias regiones de forma simultánea.

En la Figura 41 se observa una instancia de EC2 desplegada por el usuario "TFM_User-1" en la región "us-east-1d", cuyo identificador es "i-0578c59bc4f926d92". Se observa también la presencia de la etiqueta "owner", la cual contiene el nombre del usuario que ha desplegado la instancia. Dicha etiqueta ha sido creada de forma automatizada gracias a una de las políticas de Cloud Custodian definidas en el apartado 4.2.4.2.

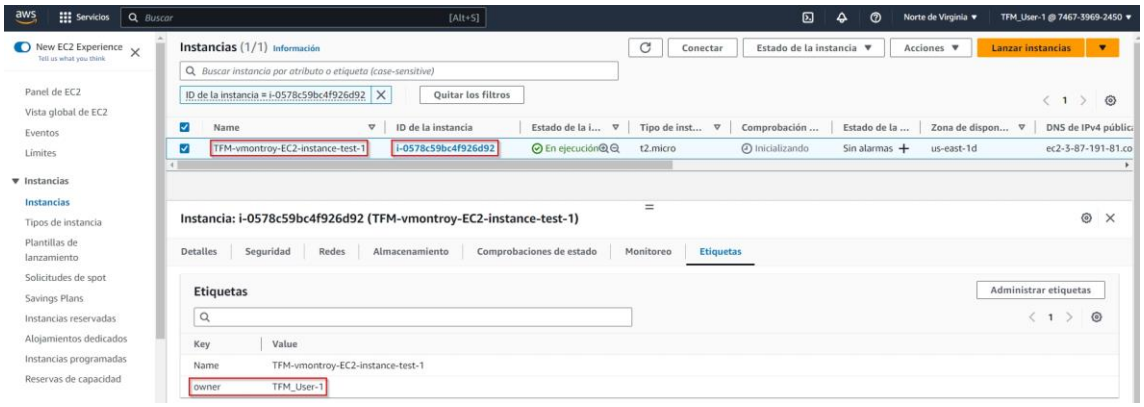


Figura 41. Despliegue de una instancia de EC2 [Creación propia].

A continuación, en la Figura 42 se observa la tabla "TFM-vmontroy-DynamoDB-table" de DynamoDB, la cual contiene la información referente a la instancia que se acaba de desplegar. Cada una de las instancias o *buckets* que se desplieguen en la región se verán reflejados en la tabla.

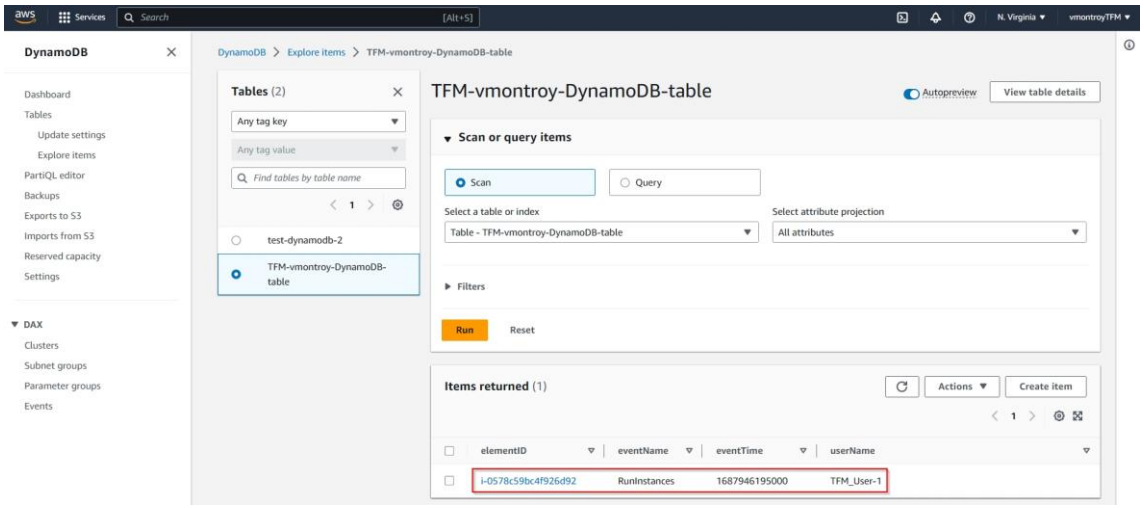


Figura 42. Tabla de DynamoDB con la información del evento "RunInstances" [Creación propia].

Del mismo modo, tras crear un nuevo *bucket* de S3 (Figura 43), la información del evento asociado a su creación, "CreateBucket", se verá reflejada en la tabla de DynamoDB, como puede observarse en la Figura 44.

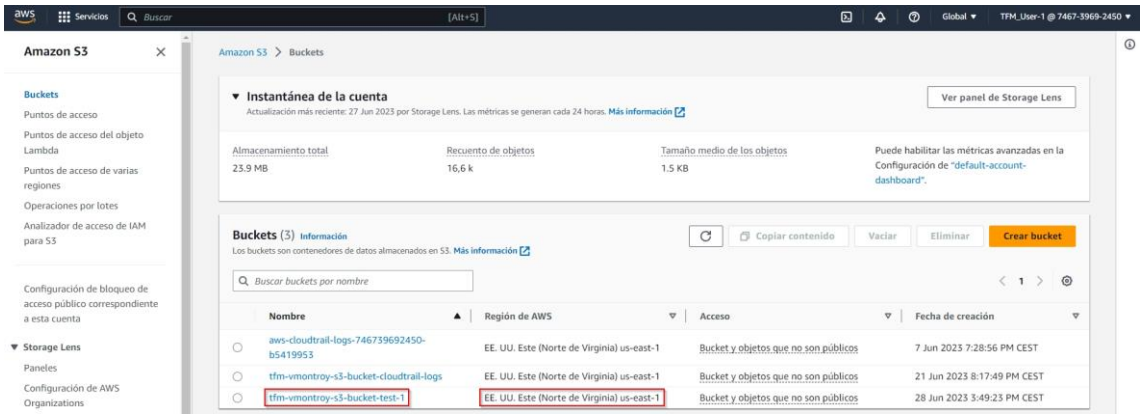


Figura 43. Despliegue de un bucket de S3 [Creación propia].

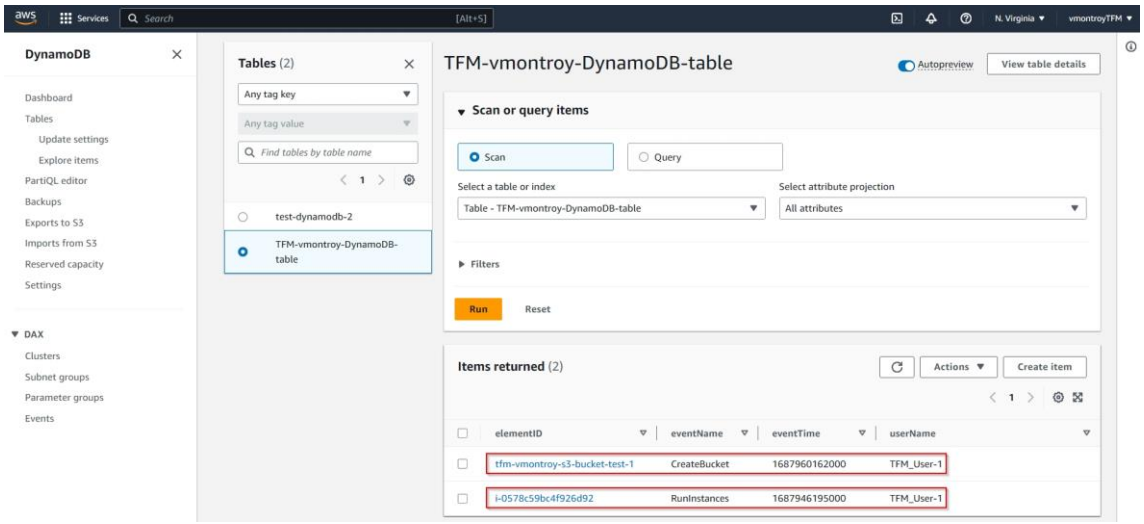


Figura 44. Tabla de DynamoDB con la información del evento "CreateBucket" [Creación propia].

Este procedimiento será el que el sistema llevará a cabo cada vez que se detecte uno de los eventos monitorizados. Además, cada vez que la función Lambda introduzca un nuevo evento en la tabla, comprobará el número de eventos del mismo tipo para el mismo usuario en la última hora (o la franja de tiempo especificada en el código fuente de la función Lambda principal).

A continuación, se va a mostrar un ejemplo de alerta. Se van a desplegar cinco instancias de EC2 de forma simultánea. Como la función Lambda está programada para detectar el despliegue de cinco o más instancias en un periodo de tiempo de una hora para un mismo usuario, esta actividad hará que salte una de las alertas, notificando al administrador de la cuenta. Cabe destacar que, cuando se despliega más de una instancia de forma simultánea, solamente se genera un evento del tipo “RunInstances”, el cual contiene la información relativa a todas las instancias desplegadas. Este comportamiento está contemplado en el código fuente de la función Lambda principal, la cual extraerá la información para cada instancia y generará un nuevo elemento de la tabla de DynamoDB para cada una de ellas.

En la Figura 45 se observa el despliegue de las cinco instancias de EC2. Al ser desplegadas de forma simultánea, el nombre asociado es el mismo para todas, “TFM-vmontroy-EC2-instance-

test-bulk", pero el identificador es único para cada una de ellas. Por ello, en la tabla de DynamoDB se crearán cinco nuevos elementos, como puede verse en la Figura 46.

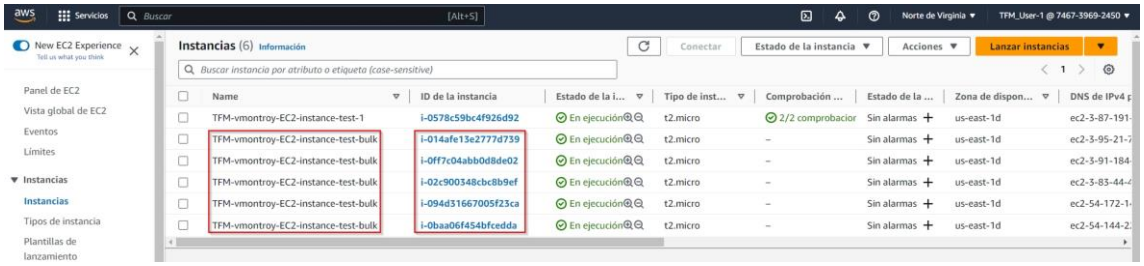


Figura 45. Despliegue de varias instancias de EC2 de forma simultánea [Creación propia].

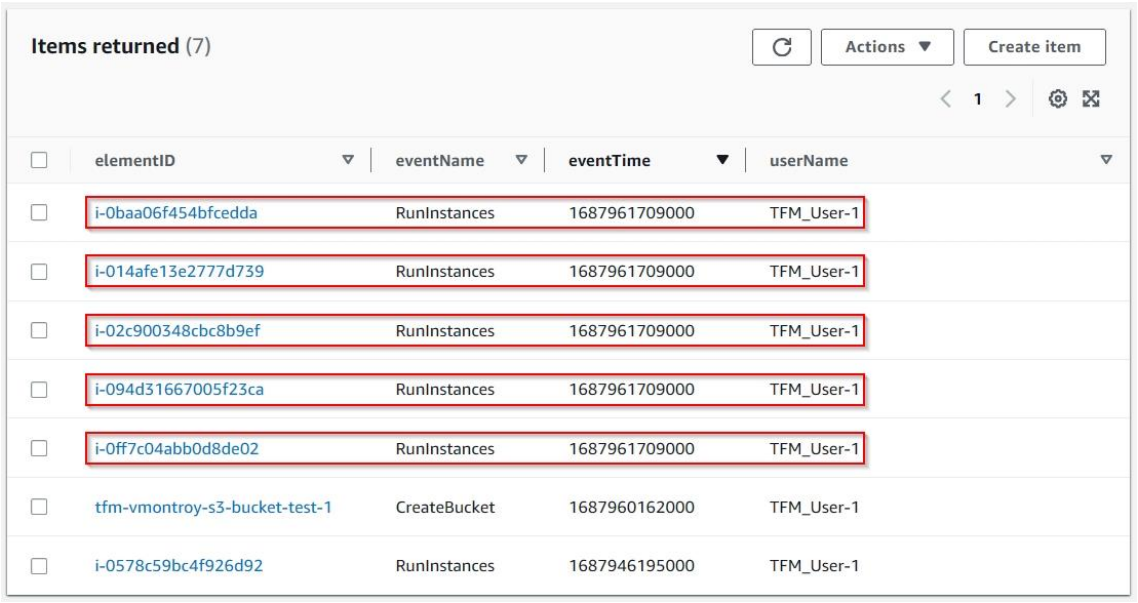


Figura 46. Tabla de DynamoDB con la información de las nuevas instancias de EC2 [Creación propia].

Al haberse superado el número máximo de instancias desplegadas en una hora, tras añadir la información de los eventos en la tabla de DynamoDB, la función Lambda mandará una notificación al servicio SNS, la cual contendrá un mensaje y una lista con los identificadores de los recursos implicados. A su vez, SNS enviará la notificación al administrador de la cuenta, como se observa en la Figura 47.

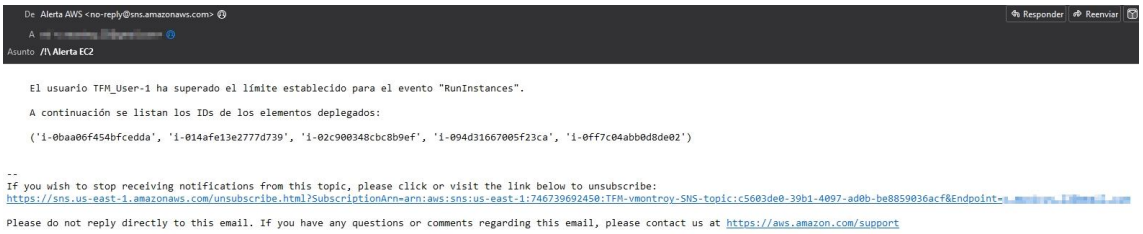


Figura 47. Correo electrónico alertando del despliegue de múltiples instancias [Creación propia].

Cabe destacar que, con la configuración actual, la función Lambda detectará un comportamiento anómalo cuando un usuario despliegue cinco instancias o más, es decir, cuando el número de instancias desplegadas en una hora sea igual o mayor que cinco, se notificará al administrador. Esta configuración puede provocar que si, por ejemplo, el usuario despliega diez instancias, el administrador reciba seis correos electrónicos. Si se quiere evitar que esto suceda, bastará con modificar el código fuente de la función Lambda principal y adaptarlo a las necesidades del administrador, pudiéndose definir, por ejemplo, que se notifique solamente cuando se hayan detectado exactamente cinco despliegues o que se notifique cada cierto número de despliegues.

El funcionamiento para el servicio AWS S3 es el mismo que el descrito anteriormente. En esta ocasión, la creación múltiple de *buckets* de S3 no será posible, ya que cada *bucket* debe configurarse y desplegarse de forma individual. A continuación, se va a mostrar otro ejemplo del funcionamiento del sistema, pero en esta ocasión se crearán cinco *buckets* de S3.

En la Figura 48 se observa la creación de los cinco *buckets* de S3. Al tener configurada la función Lambda para que alerte en caso de detectar cinco o más despliegues en un periodo de una hora, esta actividad generará una nueva alerta.

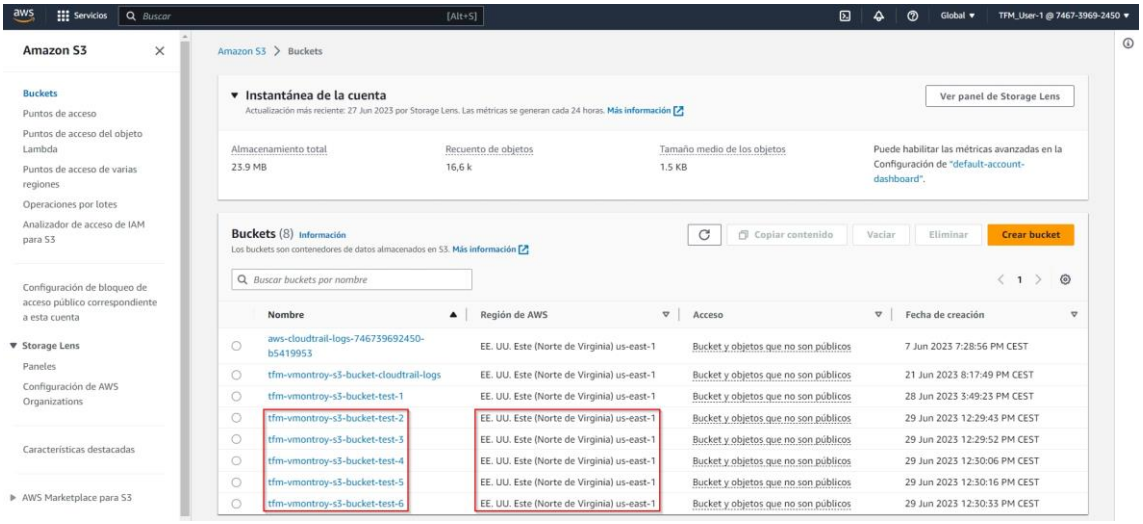
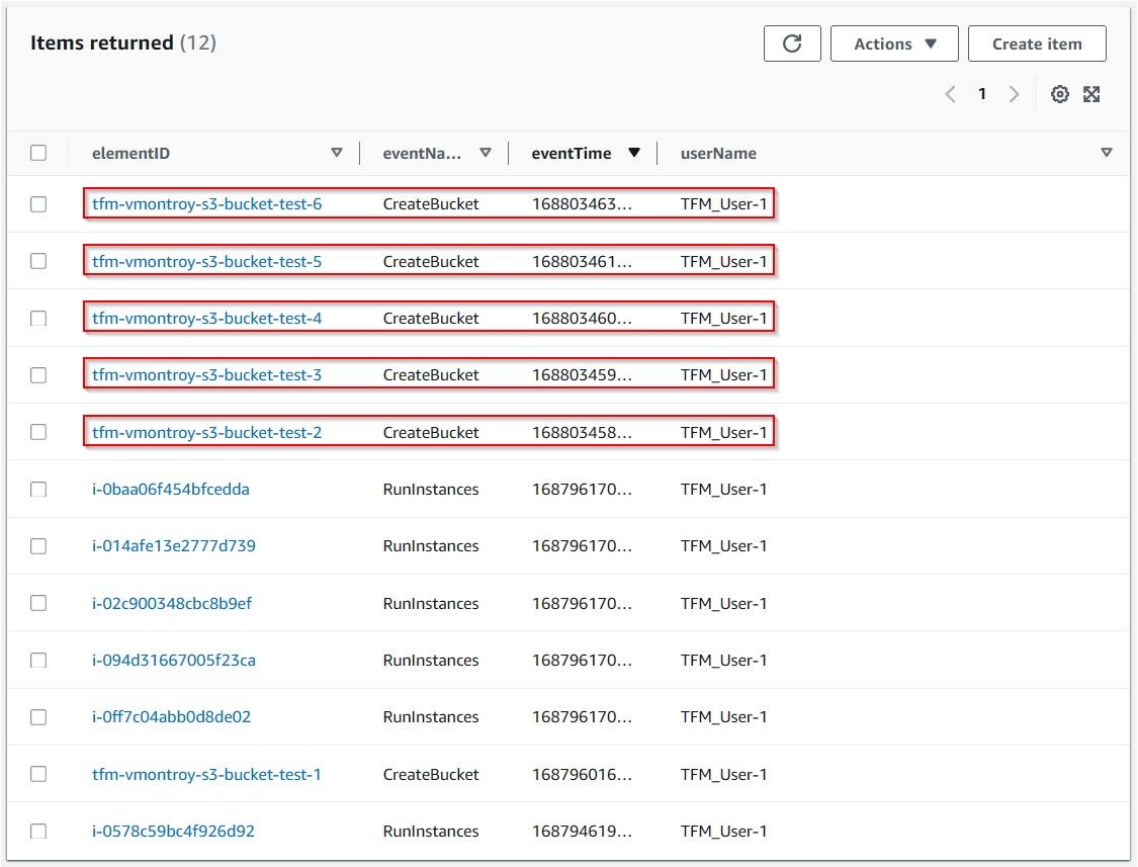


Figura 48. Despliegue de varios buckets de S3 [Creación propia].

Tras introducir la información obtenida de los eventos “CreateBucket” en la tabla de DynamoDB (Figura 49), la función Lambda generará un nuevo mensaje de alerta y lo

transmitirá al servicio SNS. Tras ello, SNS notificará al administrador de la cuenta, como se observa en la Figura 50.



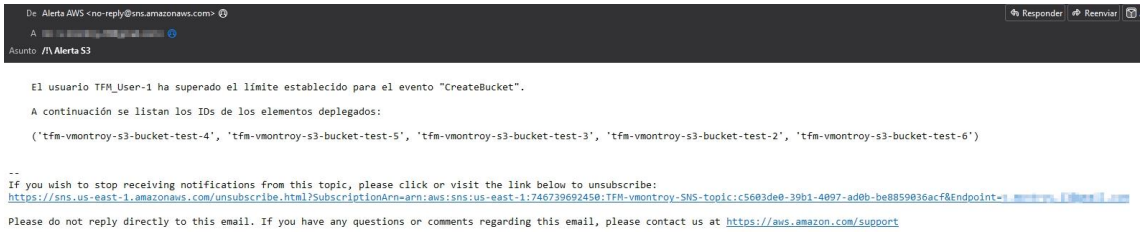
Items returned (12)

Actions ▼ | Create item

1

elementID	eventNa...	eventTime	userName
tfm-vmontroy-s3-bucket-test-6	CreateBucket	168803463...	TFM_User-1
tfm-vmontroy-s3-bucket-test-5	CreateBucket	168803461...	TFM_User-1
tfm-vmontroy-s3-bucket-test-4	CreateBucket	168803460...	TFM_User-1
tfm-vmontroy-s3-bucket-test-3	CreateBucket	168803459...	TFM_User-1
tfm-vmontroy-s3-bucket-test-2	CreateBucket	168803458...	TFM_User-1
i-0baa06f454bfcedda	RunInstances	168796170...	TFM_User-1
i-014afe13e2777d739	RunInstances	168796170...	TFM_User-1
i-02c900348cbc8b9ef	RunInstances	168796170...	TFM_User-1
i-094d31667005f23ca	RunInstances	168796170...	TFM_User-1
i-0ff7c04abb0d8de02	RunInstances	168796170...	TFM_User-1
tfm-vmontroy-s3-bucket-test-1	CreateBucket	168796016...	TFM_User-1
i-0578c59bc4f926d92	RunInstances	168794619...	TFM_User-1

Figura 49. Tabla de DynamoDB con la información de los nuevos buckets de S3 [Creación propia].



De Alerta AWS <no-reply@sns.amazonaws.com>

A [redacted]

Asunto /!\ Alerta S3

Responder Reenviar

El usuario TFM_User-1 ha superado el límite establecido para el evento "CreateBucket".

A continuación se listan los IDs de los elementos deplegados:

('tfm-vmontroy-s3-bucket-test-4', 'tfm-vmontroy-s3-bucket-test-5', 'tfm-vmontroy-s3-bucket-test-3', 'tfm-vmontroy-s3-bucket-test-2', 'tfm-vmontroy-s3-bucket-test-6')

--

If you wish to stop receiving notifications from this topic, please click or visit the link below to unsubscribe:
https://sns.us-east-1.amazonaws.com/unsubscribe.html?SubscriptionArn=arn:aws:sns:us-east-1:746739692450:TFM-vmontroy-SNS-topic:c5603de0-39b1-4097-ad0b-be8859036acf&Endpoint=[redacted]

Please do not reply directly to this email. If you have any questions or comments regarding this email, please contact us at https://aws.amazon.com/support

Figura 50. Correo electrónico alertando del despliegue de múltiples buckets [Creación propia].

6. Conclusiones y trabajo futuro

Tras las pruebas realizadas en el apartado anterior, puede considerarse que se ha alcanzado el objetivo propuesto inicialmente. Gracias al sistema que se ha diseñado y configurado, será posible detectar ciertos comportamientos que pueden resultar perjudiciales para el propietario de una cuenta de AWS.

A lo largo del proyecto, la monitorización y detección se ha centrado únicamente en los servicios EC2 y S3, ya que son altamente utilizados y prácticamente en cualquier cuenta de AWS se hará uso de ellos, pero esto no significa que el sistema propuesto se limite a la monitorización de estos servicios. Las reglas de detección propuestas son solamente dos ejemplos para mostrar el funcionamiento del sistema, pero cada administrador puede modificarlas o añadir nuevas reglas centradas en otros servicios.

Por otro lado, se ha optado por responder de forma conservadora ante las detecciones. En el sistema propuesto, cuando salta una alerta, la única acción que se lleva a cabo es el envío de una notificación al administrador. De nuevo, este comportamiento puede modificarse y adaptarse a las necesidades de cada administrador. Por ejemplo, la función Lambda principal, además de notificar, podría detener las instancias detectadas o eliminar los *buckets* desplegados, sin necesidad de que el administrador intervenga.

Como puede apreciarse, se trata de un sistema que puede adaptarse a las necesidades de cada administrador. Con los servicios desplegados y configurados, bastará con modificar el código fuente de la función Lambda principal para añadir, eliminar o modificar las acciones que esta realizará. Dependiendo de estas modificaciones, cabe la posibilidad de que haya que adaptar algún otro servicio aparte de Lambda, o incluso configurar uno nuevo y añadirlo a la arquitectura, pero este supuesto escapa al alcance de este proyecto.

En cuanto al trabajo futuro, podría configurarse una plantilla de CloudFormation [33], el servicio de AWS que permite implementar infraestructura como código, el cual desplegara y configurara todos los servicios descritos en el apartado 4.2. del proyecto. De este modo, bastaría con ejecutar la plantilla de CloudFormation para llevar a cabo este proceso.

7. Bibliografía

[1] https://www.redhat.com/es/topics/cloud-native-apps/what-is-faas

[2] https://mycamer.net/everything-you-need-to-know-about-serverless-architecture/

[3] https://aws.amazon.com/es/lambda/

[4] https://aws.amazon.com/es/lambda/pricing/

[5] https://d1.awsstatic.com/legal/AWS_Lambda/AWS%20Lambda%20Service%20Level%20Agreement-ES.pdf

[6] https://docs.panther.com/

[7] https://www.dynatrace.com/support/help/get-started/what-is-dynatrace

[8] https://www.logicmonitor.com/enterprise

[9] https://www.datadoghq.com/product/security-platform/cloud-siem/

[10] https://www.securonix.com/solutions/cloud-security-monitoring/securonix-for-amazon-web-services-aws/

[11] https://cloudcustodian.io/docs/

[12] https://www.nexastack.com/blog/governance-as-a-code

[13] https://aws.amazon.com/es/cloudtrail/

[14] https://docs.aws.amazon.com/awscloudtrail/latest/userguide/cloudtrail-user-guide.html

[15] https://aws.amazon.com/es/s3/

[16] https://aws.amazon.com/es/dynamodb/

[17] https://aws.amazon.com/es/sns/

[18] https://aws.amazon.com/es/eventbridge/

[19] https://aws.amazon.com/es/event-driven-architecture/

[20] https://aws.amazon.com/es/cloudtrail/pricing/

[21] https://aws.amazon.com/es/s3/pricing/

[22] https://docs.aws.amazon.com/awscloudtrail/latest/userguide/cloudtrail-log-file-examples.html

[23] https://docs.aws.amazon.com/es_es/amazondynamodb/latest/developerguide/HowItWorks.CoreComponents.html

[24] https://aws.amazon.com/es/dynamodb/pricing/

[25] https://aws.amazon.com/es/sns/pricing/

[26] https://aws.amazon.com/es/lambda/pricing/

[27] https://aws.amazon.com/es/iam/

[28] https://docs.python.org/3/library/venv.html

[29] https://docs.aws.amazon.com/es_es/IAM/latest/UserGuide/id_credentials_access-keys.html?icmpid=docs_iam_console#Using_CreateAccessKey

[30] https://aws.amazon.com/es/eventbridge/pricing/

[31] https://aws.amazon.com/es/about-aws/global-infrastructure/

[32] https://aws.amazon.com/es/blogs/security/easier-way-to-control-access-to-aws-regions-using-iam-policies/

[33] https://aws.amazon.com/es/cloudformation/

Anexo I – Código fuente de la función Lambda principal

```
import json
import boto3
from boto3.dynamodb.conditions import Key, Attr
import time
from datetime import datetime, timedelta
from dateutil import parser

# Se inician dos instancias para poder interactuar con DynamoDB y SNS.
dynamodb = boto3.resource('dynamodb')
client = boto3.client('sns')

# Función principal que se ejecutará cuando se dispare la función Lambda.
def lambda_handler(event, context):

    # Se obtiene la información referente al usuario y al evento.
    user_name = event['detail']['userIdentity']['userName']
    event_name = event['detail']['eventName']

    # Se establece la respuesta ante la detección del evento "RunInstances".
    if event_name == "RunInstances":

        # Se obtiene la hora de ejecución y el ID de la instancia lanzada. Puede que se lancen varias instancias EC2
al mismo tiempo. En este caso solo se generará un evento y habrá que iterar dentro de él.
        if len(event['detail']['responseElements']['instancesSet']['items']) > 1:

            for i in range(len(event['detail']['responseElements']['instancesSet']['items'])):

                instance_launch_time = event['detail']['responseElements']['instancesSet']['items'][i]['launchTime']
                instance_ID = event['detail']['responseElements']['instancesSet']['items'][i]['instanceId']

                print(instance_launch_time)
                print(instance_ID)

                # Se llama a la función encargada de introducir los datos en la tabla de DynamoDB.
                database_entry(event_name, user_name, instance_launch_time, instance_ID)

        else:

            instance_launch_time = event['detail']['responseElements']['instancesSet']['items'][0]['launchTime']
            instance_ID = event['detail']['responseElements']['instancesSet']['items'][0]['instanceId']

            # Se llama a la función encargada de introducir los datos en la tabla de DynamoDB.
            database_entry(event_name, user_name, instance_launch_time, instance_ID)

        # Se obtienen el número de eventos "RunInstances" ocurridos en la última hora y los IDs asociados a las
instancias EC2 desplegadas.
        events = event_count(user_name, event_name)

        # Si el número de eventos "RunInstances" es igual o superior a 5 se llamará a la función encargada de
notificar al administrador.
        if events[0] >= 5:
            send_alert(user_name, event_name, events[1], "/!\ Alerta EC2")

    # Se establece la respuesta ante la detección del evento "RunInstances".
    if event_name == "CreateBucket":

        # Se obtiene la hora de ejecución y el ID del bucket creado.
        bucket_launch_time = event['detail']['eventTime']
        bucket_ID = event['detail']['requestParameters']['bucketName']

        # La hora de creación del bucket no viene en formato timestamp, se convierte a este formato antes de
introducirse en la tabla de DynamoDB.
        bucket_launch_time_norm = int(str(datetime.timestamp(parser.parse(bucket_launch_time))).split(".")[0])*1000

        # Se llama a la función encargada de introducir los datos en la tabla de DynamoDB.
        database_entry(event_name, user_name, bucket_launch_time_norm, bucket_ID)

        # Se obtienen el número de eventos "CreateBucket" ocurridos en la última hora y los IDs asociados a los
buckets S3 creados.
        events = event_count(user_name, event_name)

        # Si el número de eventos "CreateBucket" es igual o superior a 5 se llamará a la función encargada de
notificar al administrador.
        if events[0] >= 5:
```

```
            send_alert(user_name, event_name, events[1], "/!\ Alerta S3")

# Se define la función encargada de introducir la información en la tabla de DynamoDB.
def database_entry(event_name, user_name, launch_time, element_ID):

    # Se define un objeto Table para poder interactuar con la tabla de DynamoDB.
    table = dynamodb.Table('TFM-vmontroy-DynamoDB-table')

    # Se introduce la información en las distintas columnas de la tabla DynamoDB.
    table.put_item(
        Item={
            'elementID': element_ID,
            'userName': user_name,
            'eventName': event_name,
            'eventTime': launch_time
            }
        )

# Se define la función encargada de consultar el número de eventos ocurridos a la tabla de DynamoDB.
def event_count(user_name, event_name):

    # Se define un objeto Table para poder interactuar con la tabla de DynamoDB.
    table = dynamodb.Table('TFM-vmontroy-DynamoDB-table')

    # Como quiere obtenerse el número de ocurrencias de los eventos durante la última hora, habrá que obtener el
timestamp de una hora antes de la ejecución de la función Lambda.
    time_1h_ago_timestamp=int(str(datetime.timestamp(datetime.now()-timedelta(hours=1))).split(".")[0])*1000

    # Se obtiene el número de eventos para el usuario implicado en la última hora.
    event_count = table.scan(FilterExpression=Attr('userName').eq(user_name) & Attr('eventName').eq(event_name) &
Attr('eventTime').gt(time_1h_ago_timestamp))

    # Se obtiene el número de eventos ocurridos en la última hora.
    events = event_count['Count']

    # Se define una lista para almacenas los IDs de los elementos implicados.
    element_IDs = []

    # Se rellena la lista con los IDs.
    for i in range(events):
        element_IDs.append(event_count['Items'][i]['elementID'])

    # La función devuelve el número de eventos ocurridos y la lista de IDs implicados.
    return(events, element_IDs)

# Se define la función encargada de notificar al administrados las alertas.
def send_alert(user_name, event_name, element_IDs, subject):

    # Se crea el cuerpo del correo electrónico que recibirá el administrador.
    message = f"""
    El usuario {user_name} ha superado el límite establecido para el evento "{event_name}".

    A continuación se listan los IDs de los elementos deplegados:

    {*element_IDs,}
    """

    # Se envía el mensaje al topic de SNS encargado de mandar el correo electrónico al administrador.
    send_alert = client.publish(TopicArn='arn:aws:sns:us-east-1:746739692450:TFM-vmontroy-SNS-topic', Message=message,
Subject=subject)
```

Anexo II – Objetivos de Desarrollo Sostenible (ODS)

En este anexo de definirá la relación del Trabajo de Final de Máster "*Sistema Serverless para la Detección de Incidencias en AWS*" con los Objetivos de Desarrollo Sostenible (ODS).

Tabla 1. Grado de relación del TFM con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenibles	**Alto**	**Medio**	**Bajo**	**No Procede**
ODS 1. **Fin de la pobreza.**				**X**
ODS 2. **Hambre cero.**				**X**
ODS 3. **Salud y bienestar.**				**X**
ODS 4. **Educación de calidad.**				**X**
ODS 5. **Igualdad de género.**				**X**
ODS 6. **Agua limpia y saneamiento.**				**X**
ODS 7. **Energía asequible y no contaminante.**				**X**
ODS 8. **Trabajo decente y crecimiento económico.**		**X**		
ODS 9. **Industria, innovación e infraestructuras.**	**X**			
ODS 10. **Reducción de las desigualdades**.				**X**
ODS 11. **Ciudades y comunidades sostenibles**.				**X**
ODS 12. **Producción y consumo responsables.**		**X**		
ODS 13. **Acción por el clima.**			**X**	
ODS 14. **Vida submarina.**				**X**
ODS 15. **Vida de ecosistemas terrestres.**				**X**
ODS 16. **Paz, justicia e instituciones sólidas.**				**X**
ODS 17. **Alianzas para lograr objetivos.**				**X**

El Trabajo de Final de Máster propuesto tiene como objetivo principal el desarrollo de un sistema de detección de anomalías en el comportamiento de los usuarios de una cuenta de Amazon Web Services (AWS). El sistema se basa en la monitorización de eventos específicos para cada usuario, detectando cuando el número de eventos supere un umbral para un determinado periodo de tiempo y notificando al administrador de la cuenta.

El empleo de un sistema de este tipo dificulta en gran medida que un usuario malintencionado o una fuga de credenciales ocasionen un uso inadecuado de los recursos disponibles, no solo por el perjuicio económico que esto causaría al propietario de la cuenta, sino por el uso innecesario de recursos y el gasto energético asociado. Se recuerda que, pese a ser un entorno *cloud*, estos servicios se ejecutan en servidores físicos que consumen energía. Por ello, es conveniente limitar el uso de estos recursos al estrictamente necesario, evitando malgastar energía.

Además, si somos capaces de desarrollar un sistema que no dependa de terceros, se reducirá también el gasto energético, ya que no habrá más de una parte implicada en la implementación de la solución.

Por otro lado, las infraestructuras *cloud* e híbridas son cada vez más populares y muchas empresas de todos los tamaños optan por usarlas. Además, la concienciación en cuanto a la seguridad informática es cada vez mayor y los usuarios se preocupan por que sus sistemas sean lo más seguros y están lo más protegidos posible. Por ello, el desarrollo de soluciones que consigan aumentar el nivel de seguridad a los entornos en la nube aportará valor a la industria.