UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DSIC DEPARTAMENT DE SISTEMES INFORMÀTICS I COMPUTACIÓ

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Dpto. de Sistemas Informáticos y Computación

Diseño e Implementacion de un Escaner de Malware Serverless

Trabajo Fin de Máster

Máster Universitario en Computación en la Nube y de Altas Prestaciones / Cloud and High-Performance Computing

AUTOR/A: Toro Ramirez, Diego Alejandro

Tutor/a: Moltó Martínez, Germán

CURSO ACADÉMICO: 2022/2023

Resumen

En la actual era marcada por la digitalización y auge tecnológico de las últimas décadas, aparecen nuevos retos como el consecuente crecimiento de las ciberamenazas, haciendo de la ciberseguridad algo necesario para todo tipo de empresas y organizaciones.

En este marco, el proyecto trata de abarcar el estudio, diseño e implementación de una estrategia para adoptar un escáner de malware en el contexto de solución cloud desplegada en plataforma cloud Amazon Web Services. El proyecto está enfocado a solventar la carencia de un escáner de malware nativo en el servicio Amazon S3 *(Simple Storage Service)*.

Dicha estrategia tendrá un enfoque de análisis dirigido por eventos, desencadenado por la subida de un fichero por parte del usuario a la aplicación. Este nuevo fichero pasará de forma temporal primeramente por un bucket de cuarentena de S3, antes de pasar al bucket real de producción, hasta que se verifique que es seguro mientras se lleva a cabo el análisis en busca de malware.

En caso de que el análisis detecte que el fichero está infectado, se le enviaría una notificación al administrador o administradores con rol de *"security champion"* de la cuenta, con el propósito de alertar sobre el posible ataque el cual podría escalar a otras tácticas. Al detectar el malware en el fichero, este permanece en el bucket de cuarentena, donde podría ser analizado por el equipo de seguridad. Pasado un tiempo, dicho malware sería eliminado de forma automática. Finalmente, en caso de que el fichero sea seguro pasaría a un bucket de producción, donde podría ser utilizado por la aplicación.

Para implementar dicha estrategia se va a seguir un enfoque donde la propuesta de valor sea un fácil despliegue y mantenimiento aprovechando la filosofía infraestructura como código, un coste reducido usando herramientas de código abierto y los propios servicios de Amazon Web Services, y una escalabilidad que permita su uso tanto en soluciones pequeñas como en soluciones más grandes.

Palabras clave: S3, Malware, Escáner, Antivirus, AWS, Sin Servidor, Escalable.

Abstract

In the current era marked by digitalization and the technological boom of the last decades, new challenges appear as the consequent growth of cyber threats, making cybersecurity something necessary for all types of companies and organizations.

Within this framework, the project aims to cover the study, design and implementation of a strategy to adopt a malware scanner in the context of a cloud solution deployed on Amazon Web Services cloud platform. The project is focused on solving the lack of a native malware scanner in the Amazon S3 service (Simple Storage Service).

This strategy will have an event-driven scanning approach, triggered by the upload of a file by the user to the application. This new file will temporarily pass first through an S3 quarantine bucket, before being moved to the actual production bucket, until it is verified as safe while scanning for malware.The scan detects that the file is infected, a notification would be sent to the administrator(s) in the "security champion" role for the account, for the purpose of alerting them to the potential attack which could escalate to other tactics. When malware is detected in the file, it remains in the quarantine bucket, where it could be analyzed by the security team. After some time, the malware would be automatically removed. Finally, if the file is safe, it would be moved to a production bucket, where it could be used by the application.

To implement this strategy, we will follow an approach where the value proposition is easy deployment and maintenance taking advantage of the infrastructure-as-code philosophy, a reduced cost using open-source tools and Amazon Web Services own services, and scalability that allows its use both in small solutions and in larger solutions.

Keywords: S3, Malware, Scanner, Antivirus, AWS, Serverless, Scalable.

Agradecimientos

Me gustaría dedicar este proyecto a mis padres por apoyarme durante toda esta etapa educativa.
A mis compañeros de máster por los buenos momentos y la amistad forjada.
Y en especial a Germán por la orientación, apoyo y dedicación que me han permitido aumentar mis conocimientos y crecer profesionalmente.

Tabla de contenidos

Índice de Figuras y Tablas

1. Introducción

1.1. Contexto actual - Importancia de ciberseguridad

La regulación europea define la ciberseguridad en [1] como *"Cybersecurity refers to the activities necessary to protect network and information systems, the users of such systems and other persons affected by cyber threats"*.

En la actual era digital, la ciberseguridad es más importante que nunca. Con el auge de la computación en nube, los dispositivos móviles y el Internet de las cosas, las empresas y gobiernos se enfrentan a un número creciente de amenazas a la seguridad. En general, según [2] los ciberataques mundiales aumentaron un 38% en 2022 en comparación con 2021.

Este aumento se explica, con factores relacionados con auge tecnológico, como el aumento de los usuarios y los entornos de trabajo en remoto tras la pandemia de COVID-19, o el desarrollo de tecnologías de inteligencia artificial que permiten acelerar la creación de ciberataques, o incluso más allá con factores sociopolíticos como la invasión rusa de Ucrania. La cual ha tenido un enorme impacto en el panorama de las ciberamenazas. Se estima que, desde el inicio de la guerra, como indica Google en su reporte *"Fog of war"* [3], se ha producido un aumento de más del 300% en las campañas de phishing rusas dirigidas contra usuarios de países de la OTAN.

Una violación de la ciberseguridad puede tener graves consecuencias para una empresa. Dependiendo del tipo de ataque, tamaño y alcance el impacto de un ciberataque puede variar. Se puede provocar desde la pérdida de datos confidenciales y las consecuentes responsabilidades legales si no protegen la información de sus clientes, hasta cuantiosas pérdidas financieras.

Según el informe de IBM de 2022, *"The average cost of a data breach."* [4] el coste medio de una violación de datos para una empresa es de 4,35 millones de dólares. Esto incluye la pérdida de ingresos y cuota de mercado, los honorarios legales y el coste de reparar los daños si el sistema de la empresa se ha visto comprometido. Aunado a esto, un ciberataque puede provocar la pérdida de confianza de los clientes, que puede ser difícil de recuperar.

Los ciberataques afectan a empresas de diferentes sectores y tamaños. Sin embargo, son las pequeñas empresas las que suelen ser más vulnerables a los ciberataques porque no disponen del mismo nivel de medidas de seguridad que las grandes compañías.

1.2. Justificación

Pese a la seguridad que puede ofrecer el Cloud, según el mismo informe de IBM, el 45% de las violaciones y brechas de seguridad estudiadas se produjeron en soluciones cuya infraestructura estaba basada completamente en plataformas *cloud*. Amazon S3, según [5] es un servicio de almacenamiento en la nube proporcionado por Amazon Web Services (AWS), que permite a las empresas y aplicaciones almacenar y recuperar grandes cantidades de datos de manera segura y escalable para una amplia variedad de casos de uso.
Por ejemplo, en el caso de una aplicación que permita a los usuarios almacenar y compartir sus fotos en la nube, los usuarios estarían subiendo archivos que se almacenarán en S3.

En este contexto un usuario malicioso podría subir algún tipo de malware, que podría afectar tanto a la propia infraestructura si no se ha configurado adecuadamente, como a otros usuarios que descargarán dichos archivos. De ahí la necesidad de una herramienta que analice los archivos que se van a almacenar en Amazon S3 en busca de malware u otras amenazas de seguridad.

En la presente fecha no existe un servicio gestionado por AWS para S3 que proporcione dicha protección, sólo algunas integraciones con terceros en su *marketplace* según [6][7][8], sin embargo, puede darse el caso que dichas integraciones no se ajusten a las necesidades específicas del negocio, a los costes estimados o a las políticas de la empresa.

Por tanto, la justificación de este proyecto, viene dada ante la necesidad de ofrecer una solución flexible, gestionada por la propia empresa que lo implemente, que permita garantizar la protección frente a ciberamenazas en forma de la ingesta de malware en el ámbito de uso de buckets de S3 en la plataforma Amazon Web Services.

1.3. Ámbito de aplicación del proyecto e impacto esperado

El ámbito de aplicación del proyecto abarcaría todas aquellas soluciones desplegadas en el proveedor cloud Amazon Web Services, en cuyo flujo de trabajo o lógica se tuviera la funcionalidad de subir ficheros/objetos a un bucket de S3 por parte de los usuarios de dicha solución, independientemente del resto de tecnologías usadas en la arquitectura de la solución.
El proyecto estaría acotado a un análisis dinámico de ficheros cuyo tamaño no exceda 4.2 GB, con un tiempo de espera desde 30 segundos hasta 2 minutos dependiendo del tamaño del fichero, donde la ejecución del escáner de malware estaría desencadenado a raíz del evento de creación de evento de creación del objeto en el bucket de S3, no abarcando un análisis retroactivo de los objetos ya existentes en el bucket de S3.
El proyecto está diseñado para aquellas arquitecturas con enfoque *serverless* y escalable, con una carga de trabajo variable, con la mínima configuración posible para su despliegue y mantenimiento, donde se precise implementar un escáner de malware en S3 a un bajo coste, teniendo en cuenta volúmenes de trabajo menores de 100 GB mensuales. Actualmente la base de este proyecto está implementada y está siendo usada en diferentes ambientes de desarrollo, integración y producción en una empresa multinacional estadounidense, para diversas soluciones y aplicaciones.

1.4. Objetivos del proyecto - Requisitos

El alcance del proyecto está acotado por los siguientes objetivos:

1. Detectar y mitigar la ingestión de malware a raíz del evento de creación de un objeto en un bucket de S3, y notificar la amenaza de ingestión de malware a los administradores de la solución.

2. Desarrollar una solución que permita analizar ficheros con una amplia variedad de formatos como imágenes, documentos, ejecutables, comprimidos, audio, texto plano.

3. Desarrollar el código de una función de AWS Lambda que permita interactuar con los diferentes recursos y cumplir con las tareas del escáner de malware como son obtener el fichero, analizarlo, y en función del caso realizar las acciones necesarias.

4. Desplegar la infraestructura necesaria para desplegar el escáner siguiendo un enfoque de Infraestructura como Código usando herramientas como Terraform/Terragrunt o CloudFormation.

5. Implementar la solución mediante un enfoque de coste reducido tanto en el cloud como en licencias.

6. Garantizar que la base de datos con las definiciones del malware esté actualizada con las últimas definiciones de virus/malware.

7. Diseñar una arquitectura con un enfoque escalable, fácil de mantener, desplegar en las arquitecturas existentes y con enfoque *serverless*.

1.5. Estructura del documento

El presente documento se divide en los siguientes capítulos:

1. **Introducción.** Se proporciona algo de contexto sobre el proyecto, se expone la necesidad del mismo, su objetivo, así como su ámbito de aplicación.

2. **Estado del Arte.** En este capítulo se presentan los conceptos de cloud computing, AWS y las soluciones actuales con respecto a un scanner de malware en S3.

3. **Desarrollo**. Se describe la arquitectura propuesta, las principales decisiones de diseño, las tecnologías usadas, así como aspectos técnicos sobre la implementación del proyecto.

4. **Pruebas.** En este capítulo se exponen la serie de pruebas y casos de uso para verificar el correcto funcionamiento de la implementación.

5. **Conclusiones y Trabajos Futuros**. Finalmente, se exponen las conclusiones y resultados obtenidos del desarrollo del proyecto, así como las posibles mejoras de este.

6. **Bibliografía.** Contiene una lista de las referencias de las diversas publicaciones, libros y medios sobre las que se ha apoyado el documento.

7. **Glosario.** Contiene una recopilación de definiciones o explicaciones cortas de conceptos usados durante el documento.

8. **Anexo - Implantación del proyecto**. Es un breve manual de uso con los pasos a seguir para desplegar e implantar el proyecto.

2. Estado del Arte

En este capítulo se exponen brevemente conceptos como cloud computing, antivirus y escáner de malware así como diversos servicios de Amazon Web Services y una crítica a las soluciones actuales en el contexto de ciberseguridad en el servicio de S3.

2.1 Cloud Computing

El concepto de computación en nube surgió entre las décadas de 1950 y 1960. Durante este periodo [9], los grandes ordenadores centrales se compartían entre múltiples usuarios, permitiéndoles acceder a los recursos informáticos de forma remota. Este modelo sentó las bases para el desarrollo de la computación en nube. Sin embargo, no fué hasta finales de la década de 1990 donde se acuñó el término, y su aplicación moderna empezó a tomar forma a principios de la década de 2000. Uno de los principales pioneros en este campo fue Amazon Web Services (AWS), que presentó su servicio *Elastic Compute Cloud* (EC2) en 2006.

El National Institute of Standards and Technology (NIST) en [10] define computación en la nube como: *"Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction."*

La aplicación moderna de cloud computing según [9] surgió a raíz de que la infraestructura informática tradicional requería importantes inversiones en hardware, software y mantenimiento. Además, ampliarla o reducirla para satisfacer demandas cambiantes era un proceso complejo y lento. La computación en nube surgió como solución a estos retos. En lugar de depender de la infraestructura local, se optó por un modelo basado en el uso de servidores y redes remotas, en las instalaciones externas de proveedores y centros de datos a los que se podía acceder a través de Internet.

Esto permitió a las empresas descargar la carga del mantenimiento de hardware y software a proveedores especializados, al tiempo que ofrecía ventajas como escalabilidad, flexibilidad y rentabilidad. Sin embargo, inicialmente la computación en nube se basaba en una infraestructura monolítica en el que todos los componentes de una aplicación estaban estrechamente acoplados y se ejecutaban en un único servidor o en un conjunto de servidores interconectados. Este enfoque de arquitectura monolítica seguía planteando limitaciones en términos de escalabilidad, tolerancia a fallos y agilidad de desarrollo. Cualquier cambio o actualización de un solo componente obligaba a volver a desplegar toda la aplicación, con el consiguiente tiempo de inactividad e ineficiencias.

Para superar estas limitaciones, la industria según [9] cambió hacia un enfoque más modular y flexible conocido como arquitectura de microservicios. Los microservicios descomponen las aplicaciones en componentes más pequeños, poco acoplados, que pueden desarrollarse, desplegarse y escalar de forma independiente. Cada microservicio realiza una función específica y se comunica con otros microservicios

Existen diferentes modelos de servicio en la nube. Por ejemplo, infraestructura como servicio (IaaS), que proporciona recursos virtualizados como almacenamiento, redes y máquinas virtuales. Con IaaS, los usuarios pueden ampliar o reducir fácilmente los recursos, pagar por lo que utilizan y tener más flexibilidad y control sobre su infraestructura. Esto sentó las bases de la Plataforma como Servicio (PaaS), que añadió una capa de desarrollo y despliegue, permitiendo a los desarrolladores centrarse en la codificación sin preocuparse de la infraestructura subyacente.

Por último, el software como servicio (SaaS), que ofrece aplicaciones totalmente funcionales accesibles a través de Internet, eliminando la necesidad de instalación y mantenimiento.

Esta evolución ha democratizado la tecnología, permitiendo a empresas de todos los tamaños aprovechar potentes recursos informáticos, escalar sin esfuerzo y reducir costes. La flexibilidad y escalabilidad de la computación en nube han fomentado la innovación y permitido el desarrollo de nuevos servicios, transformando el panorama digital.

2.2 Amazon Web Services

Algunos de los principales proveedores cloud en la actualidad son Microsoft Azure y Google Cloud Platform (GCP) y Amazon Web Services (AWS). Según el cuadrante mágico de Gartner 2022 que podemos observar en la figura 1, AWS se posiciona como el proveedor cloud líder, con una cuota de mercado de un 32%, más que sus competidores más cercanos, Microsoft Azure y Google Cloud, juntos (31%).

Figura 1-DevOps lifecycle
Imagen extraída de Gartner.com [11]

Según [12], Amazon Web Services es una plataforma de computación en la nube proporcionada por Amazon. Ofrece una amplia gama de servicios y recursos en la nube bajo demanda, lo que permite a particulares, empresas y organizaciones crear e implementar aplicaciones y servicios de forma rápida y segura. AWS proporciona una amplia gama de servicios, incluyendo potencia de cálculo, almacenamiento, bases de datos, aprendizaje automático, análisis, redes, seguridad y mucho más. Estos servicios están diseñados para ser altamente escalables, fiables y flexibles, lo que permite a los usuarios pagar solo por lo que utilizan sobre una base de pago por uso. Algunos de los servicios a tener en cuenta en este proyecto son:

- **EC2 (Elastic Compute Cloud).** Según [13], es un servicio que, entre otras funcionalidades, permite el despliegue de máquinas virtuales en la nube que permite ejecutar aplicaciones. Proporciona capacidad de computación redimensionable y ofrece diversos tipos de instancias. Las instancias EC2 pueden utilizarse para ejecutar una amplia gama de aplicaciones, desde simples servidores web hasta complejos clústeres de big data. EC2 cuenta con una característica denominada Auto Scaling Group, que según [18], permite ajustar automáticamente el número de instancias de un grupo o flota de EC2 en función de condiciones predefinidas. Ayuda a mantener el rendimiento y la disponibilidad deseados de las aplicaciones incluso durante fluctuaciones en la demanda

- **S3 (Simple Storage Service).** Según [5], es un servicio de almacenamiento de objetos que permite almacenar y recuperar grandes cantidades de datos. Es altamente duradero, escalable y seguro. S3 se utiliza habitualmente para realizar *backups* y restauraciones, archivar datos, distribuir contenidos y alojar sitios web estáticos.

- **Lambda.** Según [14], es un servicio de computación sin servidor (*serverless*) que le permite ejecutar código sin aprovisionar ni administrar servidores. Lambda escala automáticamente para gestionar las solicitudes entrantes. Se utiliza comúnmente para aplicaciones basadas en eventos y bajo demanda.

- **ECR (Elastic Container Registry).** Según [15], es un registro de contenedores totalmente gestionado que permite almacenar, administrar e implementar imágenes de contenedores Docker. Se integra con otros servicios de AWS como Amazon ECS (Elastic Container Service) y AWS Fargate.

- **IAM (Identity and Access Management Roles).** Según [16], es un servicio que permite a los administradores controlar el acceso a los recursos en la nube. Permite crear y gestionar identidades de usuarios, grupos y roles, asignar permisos granulares y políticas de acceso. Los roles agrupan permisos y acciones que determinan qué puede hacer un usuario, grupos u otros servicios sobre los recursos de AWS. Esto mejora la seguridad al otorgar acceso mínimo privilegio [49], reduciendo el riesgo de accesos no autorizados.

- **SNS (Simple Notification Service).** Según [17], es un servicio de mensajería totalmente administrado que le permite enviar notificaciones y mensajes a una variedad de puntos finales, como correo electrónico, SMS, notificaciones push móviles y más. Es compatible con la mensajería pub/sub (publicar/suscribir) y puede utilizarse para aplicaciones que requieren una comunicación basada en eventos.

2.3 Escáner de malware y antivirus

Los términos antivirus y escáner de malware suelen utilizarse indistintamente, pero presentan algunas diferencias sutiles en cuanto a su alcance y funcionalidad. Según [19], el término antivirus hace referencia a un software de seguridad diseñado para detectar, prevenir y eliminar varios tipos de software malicioso o malware en un sistema informático. Suele ofrecer protección en tiempo real ejecutándose en segundo plano analizando archivos y supervisa el comportamiento de conexiones de red y procesos en ejecución en búsqueda de acciones sospechosas, para detectar y bloquear amenazas antes de que puedan infectar el sistema. Por otro lado, según [19] un escáner de malware es un componente o función específica dentro del antivirus que se centra principalmente en la detección y eliminación de malware. Es una herramienta especializada que escanea archivos, directorios y la memoria que ya hay en el sistema en busca de indicadores o patrones de código malicioso.

La diferencia, según [20] radica en mientras que el software antivirus abarca una gama más amplia de funciones de seguridad, como la protección en tiempo real, la protección de la navegación web o las funciones de cortafuegos, un escáner de malware suele centrarse en la detección y erradicación del malware. Suele utilizarse para escaneos manuales del sistema o para realizar escaneos específicos de archivos o directorios sospechosos de estar infectados.

2.3.1 Técnicas de escaneo malware

Los escáneres de malware emplean diversas técnicas para identificar y eliminar las infecciones de malware, algunas destacadas según [19] son:

- **Detección basada en firmas.** Los escáneres de malware mantienen una base de datos de firmas o patrones de malware conocidos. Con esta técnica, el escáner compara los archivos examinados con las firmas de su base de datos. Si se encuentra una coincidencia, el archivo se marca como infectado.

- **Análisis heurístico.** Esta técnica consiste en analizar estructuras de archivos, patrones de código y otras características para identificar archivos potencialmente sospechosos o maliciosos que no tienen firmas conocidas. La heurística ayuda a detectar malware nuevo o modificado que puede no estar incluido aún en la base de datos de firmas.

Estas técnicas dependen de una base de datos la cual consiste en una colección de firmas, patrones, reglas de comportamiento y heurísticas que el escáner consulta durante el proceso de escaneado. Según [21], el malware evoluciona constantemente, y con frecuencia aparecen nuevas variantes y cepas. Para mantenerse al día de las últimas amenazas, los proveedores de escáneres de malware recopilan información sobre los nuevos programas maliciosos y crean firmas o reglas para detectarlos. Estas actualizaciones, según [22] a menudo denominadas "definiciones de virus" o "definiciones de malware", contienen las últimas firmas, patrones y heurística necesarios para identificar y bloquear las amenazas de malware más recientes. Pueden descargarse de manera local o consumirse en el cloud.

2.3.2 Motores de antivirus

Como se ha comentado anteriormente, un antivirus podría verse como una solución de seguridad integral que incluye un escáner de malware en su paquete de funcionalidades. Existen una gran cantidad de antivirus en el mercado, sin embargo, sólo suelen ofrecen el acceso al escáner de malware mediante GUI siendo productos que están enfocadas a usuarios finales, como Norton, McAfee, por lo que no son realmente interesantes para el contexto de este proyecto. A continuación se pueden apreciar una lista de antivirus que ofrecen la característica de acceder o invocar al escáner de malware mediante línea de comandos o mediante código:

- **ClamAV.** Es un proyecto *open source* que tiene el respaldo de Cisco Systems, y varias universidades e instituciones públicas. Cuenta con licencia de uso GPLv2. Destaca por su sencillez, flexibilidad, portabilidad y transparencia. Está disponible para múltiples sistemas operativos como Windows, GNU/Linux y macOS. Cuenta con SDK en múltiples lenguajes de programación. Permite el escaneo de archivos en múltiples formatos. Puede utilizarse como escáner independiente o integrado en otras aplicaciones de software. Está optimizado para ficheros pequeños, y aunque no tiene un tamaño máximo, se recomienda no exceder los 4 GB. El mirror de la base de datos está disponible de forma pública, es segura, y los usuarios pueden colaborar en actualizarla. Todos estos datos han sido obtenidos de [23].

- **Sophos**. Es un antivirus de pago. Funciona con un modelo de suscripción mensual de 50$ mensuales. Dispone de un escáner de línea de comandos. Está disponible sólo para plataformas Windows y macOS. No dispone de SDK. No menciona nada sobre un tamaño máximo de fichero. Su base de datos no es pública, las actualizaciones están gestionadas por el antivirus y se puede forzar un update por línea de comandos, solo en su versión de Windows. Todos estos datos han sido obtenidos de las fuentes [24] [25] [26] .

- **Kaspersky.** Es un antivirus de pago. Funciona con un modelo de suscripción mensual de 19$ mensuales. Dispone de un escáner de línea de comandos. Sin embargo, su uso está más enfocado en GUI, y no cuenta con demasiados parámetros de configuración en su versión línea de comandos. Está disponible sólo para plataformas Windows, macOS y plataformas móviles. No dispone de SDK. Todos estos datos han sido obtenidos de la fuente [27].

- **Avast**. Cuenta con un producto con una versión gratuita que incluye un escáner de malware. Está disponible sólo para plataformas Windows, macOS y plataformas móviles. No tiene SDK. Recopila estadísticas de su amplia red de usuarios para actualizar la base de datos con las definiciones de los virus. Se puede actualizar por línea de comandos. Todos estos datos han sido obtenidos de la fuente [28].

2.4 Crítica a las soluciones actuales para escanear malware en S3.

En la presente fecha, junio de 2023, según [7] no existe un servicio o característica nativa de escaneo de malware en el contexto de un bucket de S3. Existen soluciones de terceros que ofrecen este servicio, las cuales se pueden integrar mediante el *marketplace* de AWS. Las soluciones más relevantes en cuanto a críticas y uso, son:

- **Cloudstorage**: Ofrece servicios de análisis de malware para los servicios de S3, EBS y EFS. Entre sus servicios se pueden encontrar análisis dinámico, donde los ficheros son analizados conforme se introducen en el bucket,o un análisis retrospectivo bajo demanda, es decir, analizar los archivos existentes en el bucket. También ofrece servicios más premium como "*malware detonation",* que según [30] se trata de una técnica que consiste en un entorno aislado para estudiar cada amenaza. Su análisis de malware usa el motor de antivirus Sophos. Tiene un modelo de pago por uso, donde para los análisis comentados, los 100 primeros GB analizados de forma mensual tienen un coste mínimo de 49$ sin impuestos. Luego el precio por GB analizado va escalando, por ejemplo en el rango de los 101-500 GB el coste es de 0,4$. Por ejemplo, el precio para un TB sería de (49$ + 400GB*0,4$/GB + 500GB*0,35$/GB). Todos estos datos han sido obtenidos de la fuente [29].

- **bucketAV**: Solo ofrece servicios de análisis de malware para S3. Ofrece tanto servicios de análisis dinámico, como análisis retrospectivo de forma periódica como bajo demanda. A diferencia de Cloudstorage, éste ofrece análisis con dos tipos de antivirus, ClamAV para ficheros de un tamaño inferior a 4,1 GB y Sophos para ficheros de mayor tamaño y con mayores prestaciones en la velocidad del análisis. El precio de sus servicios varía en función del motor de antivirus seleccionado. Si se elige la opción de Sophos se ofrece funcionalidad adicional como un dashboard con las métricas de los análisis, etiquetado de los objetos como limpios. Se pueden configurar las notificaciones del resultado del análisis. Tiene un modelo de pago por uso, donde tiene un coste de entrada mensual de entorno a unos 42$ sin impuestos, y luego dependiendo del motor, en el caso de la opción de sophos se factura 0,2$ por GB analizado, y en el caso de ClamAV se factura 0,05$ por cada vCPU/hora de la máquina asignada que realice el escaneo de malware, donde el uso de la vCPU dependerá de varios factores como el tamaño de los ficheros, y la cantidad de peticiones simultáneas. Por ejemplo, el precio para la versión Sophos para un TB sería de (42$ + 1000*0,2). Para la versión de ClamAV la facturación se realizaría en función de la carga de trabajo, pero como mínimo tendría un coste de 42$ + 36,50$ (1 vCPU * 24 horas * 31 días). Todos estos datos han sido obtenidos de la fuente [31].

En la tabla 1, podemos ver un resumen comparativo entre el precio del modelo de pago por uso de cada una de estas alternativas.

	Cloudstorage	Bucketav-Sophos	Bucketav-ClamAV
Coste de entrada mensual	49 $	~ 42 $ *(sin impuestos)	~ 42 $ *(sin impuestos)
Coste 0-100 GB	0 $	0,2 $ *(sin impuestos)	na
Coste 101-500 GB	0,4 $	0,2 $ *(sin impuestos)	na
Coste vCPU per hour	na	na	0,05 *(sin impuestos)

Tabla 1-Comparativa de precios entre soluciones de terceros de pago.
Datos generados según las fuentes [29] y [31]

Pese a no tener precios excesivamente elevados, tienen un coste de entrada de alrededor de 50$, independientemente del uso, por lo que quizás se podría reducir este coste especialmente si pensamos en soluciones con volúmenes de trabajo de menos de 100 GB mensuales. Además, el uso de estas soluciones de pago implica tratar cuestiones como el tratamiento de datos por parte de estas empresas, salir del ecosistema de Amazon Web Services, pérdida del control sobre dependencia de la disponibilidad de terceros, integraciones que pueden ir en contra de las políticas de ciberseguridad de la empresa.

Aparte de estas soluciones de pago, existen proyectos, tutoriales e ideas desarrolladas por la comunidad, que plantean integrar una arquitectura propia en nuestro entorno de Amazon Web Services, la cual se encargue de realizar el escaneo de malware. Muchas de estas opciones son interesantes, pero en su mayoría siguen un enfoque más tradicional basado en servidor, donde se presentan problemas de escalabilidad y requiere de cierta configuración manual de alguno de los antivirus y un mantenimiento de los mismos, por ejemplo [32]. O en otros casos la base de datos con las definiciones de los virus no se actualizan de manera frecuente o bajo demanda, como ocurre en la solución descrita en [33].

2.4.1 Infraestructura como código

Además un punto a tener en cuenta con estas alternativas, es el coste de configuración e integración de las arquitecturas propuestas, ya que los procesos manuales tradicionales de aprovisionamiento de infraestructuras suelen ser propensos a errores e incoherencias, ya que dependen en gran medida de la intervención humana y la configuración manual.

La infraestructura como código (*Infrastructure as Code* - IaC), según [34] es un enfoque de ingeniería de software que hace hincapié en el uso de archivos de configuración legibles por máquina para automatizar el aprovisionamiento y la gestión de los recursos de infraestructura. Trata la infraestructura como software, permitiendo a las organizaciones definir y gestionar su infraestructura utilizando lenguajes de programación declarativos o imperativos. Este enfoque minimiza el error humano y facilita la creación de despliegues de infraestructura estandarizados y repetibles.

Los cambios en la infraestructura pueden rastrearse, auditarse y revertirse en caso necesario, lo que proporciona a las organizaciones un mayor control y flexibilidad. Al abstraer las configuraciones de infraestructura en código, las organizaciones pueden definir y desplegar fácilmente recursos de infraestructura en distintos entornos, como desarrollo, pruebas y producción La infraestructura como código pretende mejorar la eficiencia, coherencia, escalabilidad y fiabilidad de los procesos de despliegue y mantenimiento de infraestructuras.

En el contexto de Amazon Web Services, encontramos las siguientes alternativas de IaC:

- **AWS CloudFormation**. Según [35] es un servicio nativo de AWS que permite definir e implementar recursos de infraestructura mediante plantillas YAML o JSON. Proporciona un enfoque declarativo para aprovisionar y administrar recursos, permitiendo crear, actualizar y eliminar pilas enteras de recursos de AWS de forma fiable y automatizada. Usa el concepto de Stack el cual representa una colección de recursos de AWS que se crean, actualizan y eliminan como una sola unidad. Proporciona una forma de aprovisionar y administrar un grupo de recursos de manera repetible y automatizada.

- **AWS CDK (*Cloud Development Kit* - Kit de Desarrollo en la Nube).** Según [36] es un marco de desarrollo de software de código abierto proporcionado por AWS. Le permite definir su infraestructura utilizando lenguajes de programación conocidos, como TypeScript, Python o Java. Con CDK, puede escribir código para definir sus recursos de infraestructura y el marco generará las plantillas CloudFormation correspondientes para la implementación.

- **Terraform.** Aunque no es exclusivo de AWS, según [37] Terraform es una popular herramienta de código abierto que admite múltiples plataformas en la nube, incluida AWS. Utiliza un lenguaje de configuración declarativo denominado HashiCorp Configuration Language (HCL) para definir los recursos de infraestructura. Terraform proporciona un flujo de trabajo coherente para el aprovisionamiento y la gestión de recursos en diferentes proveedores de nubes, lo que lo hace adecuado para entornos de nubes múltiples o híbridas. Terraform sigue el workflow que podemos observar en la figura 2, donde podemos identificar los siguientes pasos:

    1. Write: Se define la infraestructura especificando el estado deseado de sus recursos en archivos de configuración utilizando el lenguaje de configuración de Terraform (HCL),

    2. Plan: Se genera un plan de ejecución utilizando el comando terraform plan. Terraform analiza la configuración, determina los cambios necesarios y muestra una vista previa de lo que se creará, modificará o destruirá.

    3. Apply: Se aplica el plan utilizando el comando terraform apply. Terraform aprovisiona y configura los recursos de la infraestructura descrita en el plan.

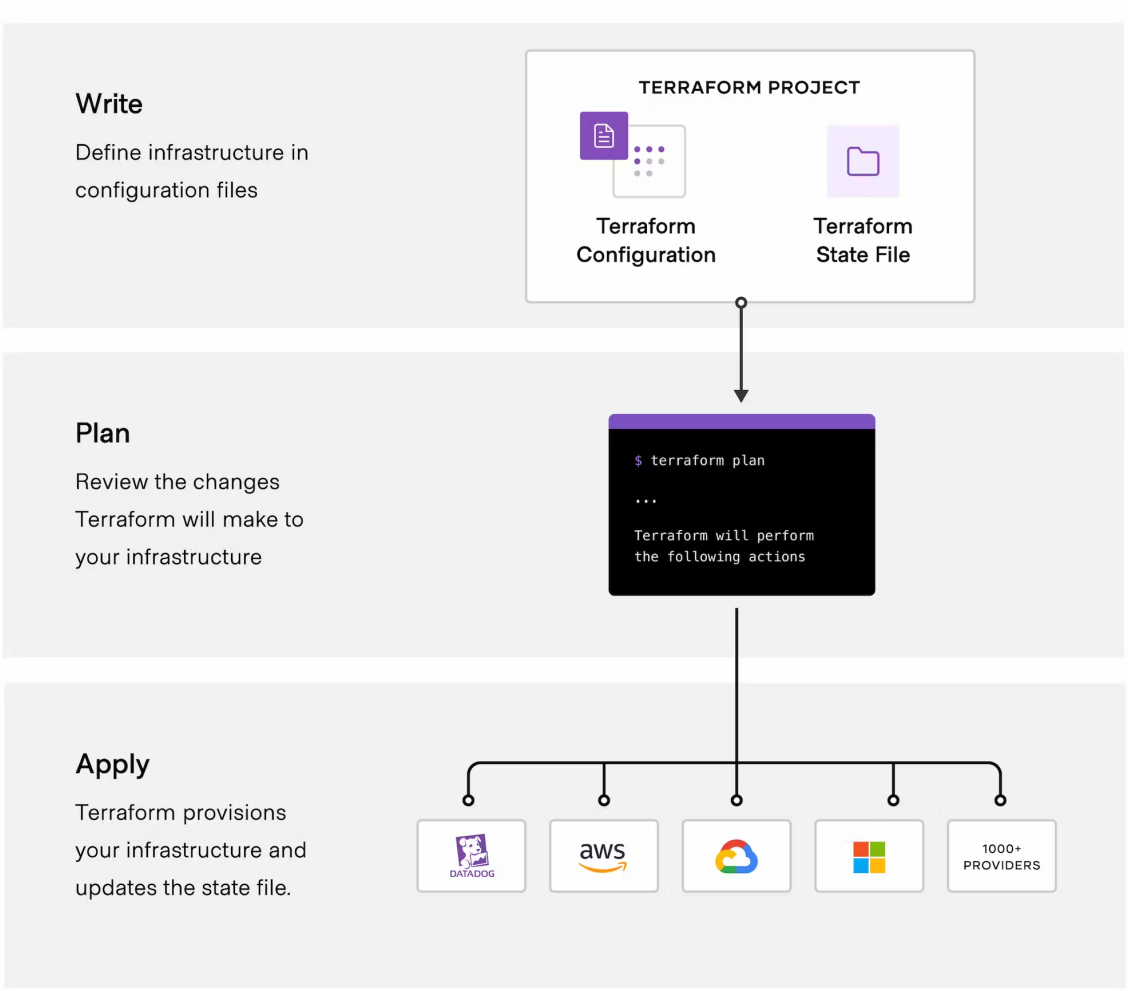

Figura 2-Terraform workflow.
Imagen extraída de hashicorp.com

2.5 Propuesta

La propuesta de este trabajo consiste en desarrollar una solución que permita integrar un escáner de malware que permita garantizar la protección frente a ciberamenazas en forma de la ingesta de malware en el ámbito de uso de buckets de S3 en la plataforma Amazon Web Services.

La integración del escáner de malware se va dividir en dos partes, por un lado, el despliegue de la arquitectura necesaria para el despliegue en AWS, y por otro lado, el desarrollo del controlador que se encargará de gestionar los eventos, hacer uso de alguno de los escáner de malware existentes para analizar los archivos, enviar notificaciones y asegurar de usar una base de datos de virus actualizada.

La arquitectura se va a plantear desde un enfoque *serverless*, escalable y de coste optimizado. Se utilizarán los principios de infraestructura como código facilitando un rápido y sencillo aprovisionamiento de la infraestructura necesaria en AWS.

3. Desarrollo

En este capítulo se detalla la arquitectura y se exponen las decisiones de diseño más relevantes.

3.1 Identificación y análisis de posibles soluciones

3.1.1 Estrategia de escaneo

Un bucket de S3, es un entorno estático, es decir, el malware no va a poder ejecutarse. Por lo que el mayor riesgo reside en que otros servicios o incluso otros usuarios obtengan el fichero infectado. Para evitar esto, se pueden adoptar diferentes estrategias de escaneo:

- **Escáner dirigido por evento**. Donde se escanea el fichero en cuanto se produce el evento de subida a S3.

- **Escáner retrospectivo.** Donde se realiza un escaneo del bucket completo en busca de ficheros infectados.

- **Escáner bajo demanda:** Donde se realiza un escaneo del fichero antes de ser consumido o descargado por otros servicios o usuarios.

La implementación de cada uno de los distintos enfoques no es excluyente; al contrario son complementarias. Sin embargo, la primera línea de defensa es el enfoque de escáner dirigido por el evento de creación del fichero ya que va a mitigar cuanto antes el posible ataque y va a ayudar al equipo de ciberseguridad del sistema a ponerse en alerta ante otras posibles estrategias de ciberataque. Por lo que, esta va a ser la estrategia que vamos a implementar en el proyecto. Adicionalmente, esta estrategia se puede completar integrando un bucket de cuarentena o de paso donde el fichero permanecerá temporalmente hasta que termine de ser analizado por el escáner, evitando mezclar el fichero con malware con los ficheros ya escaneados y que estén limpios en el bucket de producción.

3.1.2 Elección del motor de antivirus

El motor de escaner de malware que se usará será ClamAV. Por varios motivos:

- Adecuación. Como en un bucket de S3, no tenemos procesos en ejecución, con la funcionalidad que nos ofrece un escáner de malware se estarían cubriendo las necesidades planteadas. Se podría usar un antivirus, pero por lo general estos necesitan más recursos en cuanto a espacio, y por tanto más coste y tiempo en desplegarse. Los escáner de malware que hemos visto, a excepción de ClamAV, no tienen una versión independiente de su antivirus haciendo que no sean del todo adecuados si queremos desarrollar un sistema que sea escalable, ya que por ejemplo penalizará la ejecución del análisis en nuevas instancias EC2 en un grupo de autoescalado, y el caso de funciones Lambda se estarían usando más recursos de memoria de los necesarios.

- Coste. ClamAV tiene licencia GPLv2, con lo cual su uso es gratuito. Avast también es gratuito. Con respecto a Sophos y Kaspersky, no son interesantes a nivel de coste, ya llegan a tener costes incluso mayores que las soluciones de terceros.

- Base de datos de virus. La base de datos de virus ClamAV cuenta con el respaldo de Cisco Systems, universidades, instituciones públicas, centros de investigación, y miles de usuarios, se actualiza casi de forma diaria. Tiene un catálogo más extenso que el resto de soluciones. Además cuenta con la herramienta *freshclam* para actualizarla.

- Herramientas de desarrollo. ClamAV a diferencia del resto, al ser de código abierto cuenta con diversas librerías en múltiples lenguajes e incluso SDK. Por tanto el desarrollo de un controlador es bastante sencillo. En cuanto al resto de antivirus, solo disponen de herramientas de línea de comandos para invocar sus escáneres de malware, por lo que para desarrollar un controlador, se tiene que realizar una llamada al sistema invocando dicha herramienta. Esto no es muy buena práctica y puede llevar a cometer errores.

3.1.3 Server vs Serverless

El enfoque más tradicional basado en servidor es interesante si se va a realizar un uso intensivo del escáner. Sin embargo, es complicado adaptar las cargas de trabajo. Por un lado, una escalabilidad horizontal por ejemplo usando instancias EC2 con un grupo de autoescalado, presenta retos como la coherencia en su base de datos entre las distintas instancias, la cual aunque no es de gran tamaño, es local para cada instancia. Se podría pensar en externalizar, pero estaríamos añadiendo más complejidad a la infraestructura. Y en cuanto escalado vertical, podría llegar a ser interesante pero de todas maneras la instancia tendría un mínimo uso en los periodos donde no recibirá cargas de trabajo.

Un enfoque *serverless* basado en funciones Lambda o contenedores, puede ser más interesante ya que por naturaleza estos recursos son más escalables. Entre ambos servicios, vamos a adoptar el uso de las funciones Lambda, ya que estas son ideales para tareas pequeñas y efímeras que se escalan automáticamente según la demanda. Los contenedores son más adecuados para aplicaciones más complejas que requieran más recursos y tengan un ciclo de vida de ejecución más largo. Los contenedores proporcionan más control y flexibilidad, pero también requieren más administración. Si se compara el despliegue de una instancia EC2, con una Lambda, está siempre será bastante más rápida. Ya que a diferencia de las instancias EC2 que utilizan un sistema operativo completo, el runtime o entorno de ejecución de una Lambda se optimiza específicamente para ejecutar código de forma rápida y eficiente en la nube, sin proporcionar todas las características y funcionalidades innecesarias de un sistema operativo tradicional para un entorno de ejecución no deja de ser un contenedor efímero. Existen admite varios runtimes, como Node.js, Python, Java, C#, Go e incluso se pueden crear entornos de ejecución a partir de una imagen Docker.

Finalmente, otro punto a tener en cuenta, aunque paradójico, es la seguridad del sistema que estamos implementando. El runtime de Lambda es un entorno aislado, donde después del análisis, los recursos asociados a la invocación de la función Lambda dejan de estar disponibles transcurrido un tiempo. Mediante este enfoque podemos asegurarnos que el malware que estamos analizando no infecte el recurso que se encargue de realizar el análisis, por falta de recursos y tiempo de ejecución y aunque lo haga no tenga ningún impacto dado el carácter efímero de las invocaciones a funciones Lambda. En siguientes apartados explicaremos cómo podemos reforzar esta seguridad mediante el uso de roles y políticas. Con enfoque basado en un servidor es algo que tenemos que tener en cuenta.

3.2 Arquitectura del Sistema

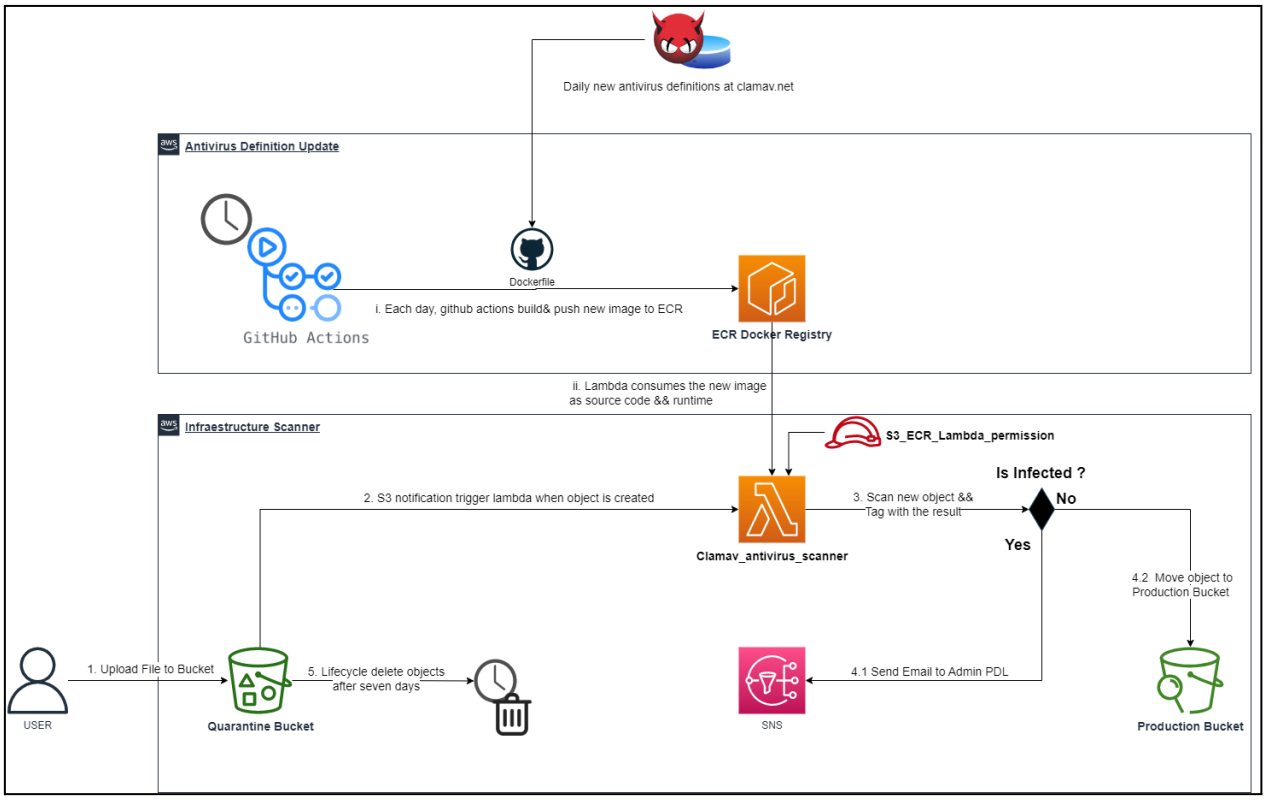

Figura 3-Arquitectura serverless propuesta.

A continuación se va a explicar la arquitectura planteada del sistema, componente a componente siguiendo el workflow que podemos apreciar en la figura 3.

1. El flujo empieza con la subida de un fichero por parte de algún usuario a la aplicación. Independientemente de la lógica del *frontend* o *backend* implementados para que el usuario pueda realizar esta acción, el fichero terminará en un bucket de S3. Lo único a tener en cuenta por parte de los desarrolladores es cambiar el bucket destino inicial sustituyendo el bucket de producción o *backend* que se estaba usando por el bucket de cuarentena que se ha creado para integrar este sistema.

2. El evento de creación del fichero, en este contexto llamado objeto, creará una notificación que servirá de *trigger* para invocar la función Lambda de escaneo.

3. La función Lambda asumirá el rol correspondiente, con el cual obtendrá los permisos necesarios para obtener el objeto del bucket de S3 de cuarentena, analizarlo usando el escáner de malware ClamAV y crear una tag en el objeto con el resultado.

4. Dependiendo del resultado del análisis:

    a. Si el archivo está infectado, la función Lambda envía un correo electrónico utilizando un topic SNS para notificar a los administradores del sistema que el archivo está infectado y necesita ser tratado.

    b. Si el archivo está limpio, la función Lambda lo mueve al bucket de producción para su uso en el entorno de producción.

5. Los archivos infectados permanecerán en el bucket de cuarentena durante 7 días para ser investigados. Pasada esta fecha, debido a una *life policy* del bucket, el fichero es eliminado de forma automática por Amazon S3. De esta forma se evitan costes de almacenamiento innecesarios.

Paralelamente también hay un proceso programado que se ejecuta con el fin de construir el runtime de la Lambda para garantizar que el escáner de *malware* tenga una de las versiones más recientes de la base de datos con las definiciones de los virus. Para ello:

I. Cada medianoche, mediante un pipeline que utiliza acciones de GitHub, se construye la nueva imagen Docker incluyendo la última versión de la base de datos de virus disponible en ese momento. La imagen se etiqueta y se sube a un repositorio privado de ECR.

II. Se actualiza la función Lambda del escáner de malware con esta nueva versión.

3.3 Diseño detallado

3.3.1 Infraestructura como código

El código de la infraestructura desplegada se encuentra disponible en un repositorio público de la plataforma GitHub, al cual se puede acceder mediante el siguiente enlace:

```
https://github.com/toromuu/Serverless_Malware_Scanner_Terragrunt-Infrastructure-Live
```

En la siguiente figura 4 podemos observar la estructura de carpetas del repositorio, en este caso nos centramos en la definición de la infraestructura. Bajo carpeta `account1`, podemos observar varios niveles lógicos de organización, por un lado `account1` estaríamos hablando a nivel de cuenta, `us-east-1` se corresponde con la región, y `dev` con el entorno de desarrollo, y finalmente, el nivel de componentes donde encontramos `malware_scanner` , con la definición de los recursos de la arquitectura del escáner de malware.

En cada uno de estos niveles, encontramos ficheros de configuración, (`account.hlc`, `region.hcl`, `env.hcl`) donde podemos encontrar variables que pueden ser consumidas por los recursos que se encuentren en dicho alcance. Estas variables comunes ayudan a mantener el código limpio y seguir el principio de diseño DRY-Code (ver Glosario).

Los recursos que crea este repositorio son:

- Un bucket de cuarentena, donde el fichero va a permanecer hasta que sea analizado. Contiene una política que elimina los objetos tras 7 días desde su creación. Se corresponde con el recurso `./account1/us-east-1/dev/malware_scanner/S3/Quarantine`.

- Un bucket de producción, donde se va a mover el fichero una vez haya pasado satisfactoriamente el escáner de malware y no se encuentre infectado. Se corresponde con el recurso `./account1/us-east-1/dev/malware_scanner/S3/Production`.

- Un repositorio ECR privado para subir la imagen que usará la función Lambda como runtime. Se corresponde con el recurso `./account1/us-east-1/dev/malware_scanner/ECR`.

- Un rol IAM para conceder permisos a la función Lambda para obtener la imagen de ECR, obtener objetos del bucket de cuarentena y moverlos al bucket de producción. Se corresponde con el recurso `./account1/us-east-1/dev/malware_scanner/IAM`..

- Un topic de SNS que usará la Lambda para enviar la notificación de detección de malware por correo electrónico a direcciones de correo electrónico específicas. Se corresponde con el recurso `./account1/us-east-1/dev/malware_scanner/SNS`.

- La función Lambda que va a realizar la función de escáner de malware en el bucket de cuarentena y va a interactuar con el resto de recursos anteriormente descritos. Se corresponde con el recurso `./account1/us-east-1/dev/malware_scanner/Lambda`.

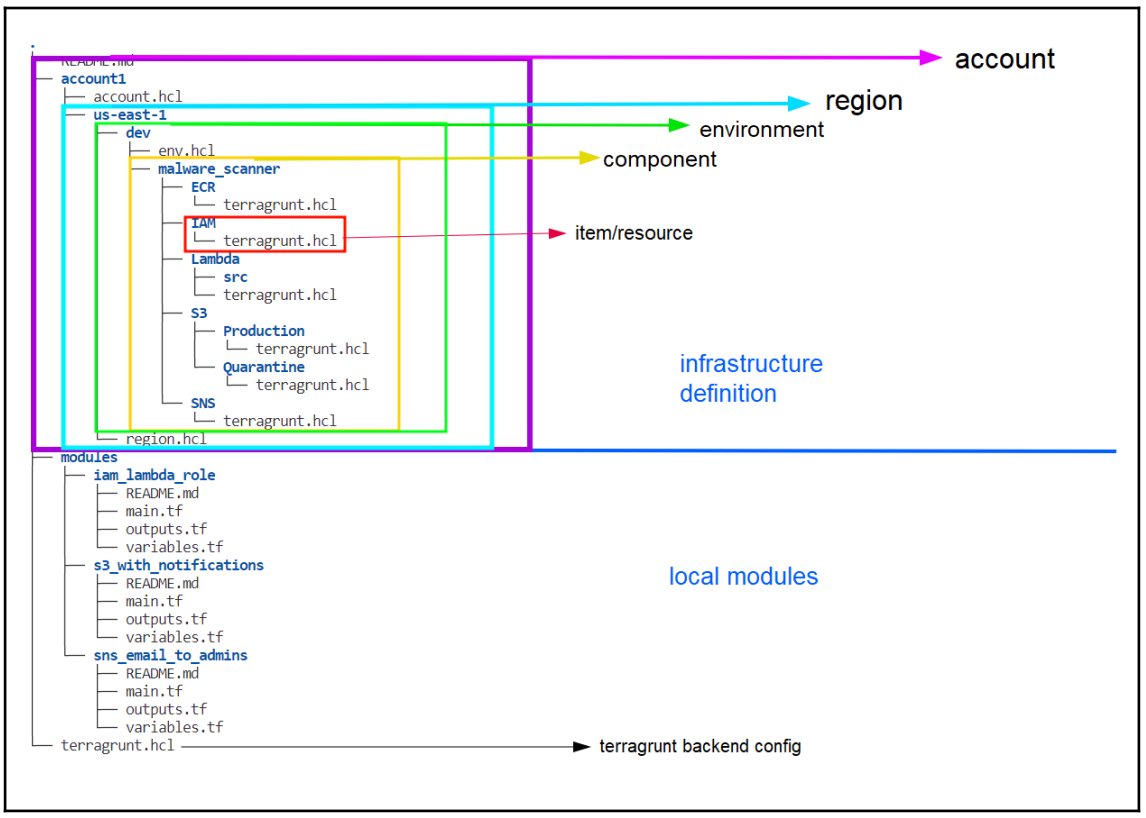

Figura 4-Estructura de carpetas en el proyecto IaC.

La tecnología escogida para desplegar esta infraestructura como código es Terraform en conjunto a un wrapper llamado Terragrunt, ver fuente [38]. Estas dos tecnologías ofrecen ciertas características que han sido realmente interesantes para el proyecto:

3.3.1.1 Gestión de espacios de trabajo.

Se refiere a la capacidad de crear y gestionar múltiples entornos aislados dentro de una única configuración de Terraform. Cada espacio de trabajo representa una instancia distinta de su infraestructura, lo que le permite gestionar diferentes despliegues, entornos o variaciones de su infraestructura dentro de la misma base de código. Terragrunt simplifica la creación de espacios de trabajo automatizando y gestionando la configuración backend por espacio de trabajo y facilita el cambio entre espacios de trabajo. Esta funcionalidad facilita la gestión de entornos separados como desarrollo, *staging* y producción. En comparación, según [39] y [40] AWS CloudFormation y AWS CDK no ofrecen capacidades de gestión de espacios de trabajo integradas comparables. En la figura 5 podemos apreciar un diagrama que explica este concepto.

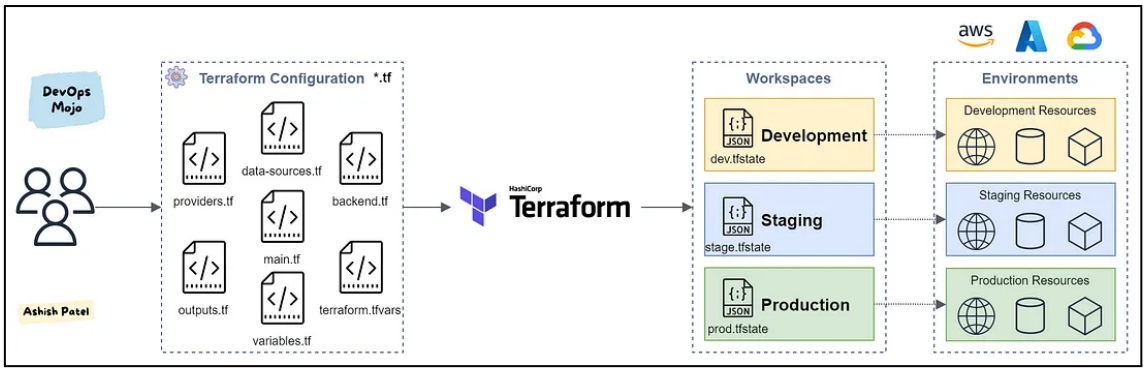

Figura 5-Terraform workspaces.
Imagen extraída de medium.com [41].

La gestión de espacios con Terragrunt permite crear espacios de trabajo dependiendo del nivel en el que nos encontremos. De esta forma se puede desplegar desde un único recurso como podría ser la función Lambda, hasta el escaner completo u otra abstracción como un servidor web, de forma independiente, o todos los recursos de la región, hasta el nivel de cuenta. Es posible mediante carpetas, tener un entorno de dev, stg o prod, usando la misma configuración. Y por otro lado, permite cambiar de cuenta, y desplegar esta misma arquitectura, con los mismos ficheros de configuración si se tienen los permisos suficientes en cada cuenta.

3.3.1.2 Modularidad, reutilización y recursos creados.

Terraform, permite la definición de módulos reutilizables, encapsulando patrones comunes de infraestructura y facilitando la reutilización del código. Este enfoque modular simplifica la creación de configuraciones de infraestructura consistentes y mantenibles. Terragrunt se basa íntegramente en el uso de estos módulos. Los módulos pueden ser tanto locales, como obtenerse de un repositorio de la propia organización, como obtenerse de fuentes oficiales como el *registry* de Terraform accesible desde la dirección https://registry.terraform.io/.

En concreto en el proyecto se han usado dos módulos de esta fuente, uno para para la definición del recurso ECR, y otro para la función Lambda.

Módulo ECR. El módulo se encuentra disponible en la siguiente dirección:

```
https://registry.terraform.io/modules/terraform-aws-modules/ecr/aws/latest
```

Para la creación de este recurso se ha definido la política que podemos ver en la figura 6 que solo permite tener dos imágenes del repositorio ECR al mismo tiempo. De tal manera que cuando se construye una nueva imagen, se etiqueta con el *tag* de "latest", y se sube al repositorio, la imagen actual es desplazada quedando sin tag. No se elimina directamente ya que en caso de fallar la imagen que se acaba de subir, se podría usar la desplazada como backup. Con la próxima subida de otra nueva imagen, esta imagen sin tag es eliminada del repositorio, y el puesto de imagen de backup pasa a ser de la imagen que la sustituyó en su momento. De esta forma se ahorran costes en ECR pues no se mantienen una gran cantidad de imágenes que con poca probabilidad se vayan a usar.

```
  expireUntagged = jsonencode({
    rules = [
      {
        action = {
          type = "expire"
        },
        selection = {
          countType   = "imageCountMoreThan",
          countNumber = 1,
          tagStatus   = "untagged"
        },
        description  = "Expire untagged images",
        rulePriority = 1
      }
    ]
  })
}
```

Figura 6-Política ECR para optimizar el número de imágenes simultáneas.

Módulo Lambda. El módulo se encuentra disponible en la siguiente dirección:

```
https://registry.terraform.io/modules/terraform-aws-modules/lambda/aws/latest
```

La figura 7 muestra el input de la función Lambda para este módulo. El timeout está aumentado a 900 segundos o 15 minutos siendo este el límite máximo de ejecución de la Lambda. El `ephemeral_storage_size` , es decir el tamaño máximo que puede tener la función Lambda, es de 8 GB. Por estos dos parámetros, los ficheros que puede llegar a analizar el escáner de malware están limitados por tamaño. Sin embargo, comentar que no es buena práctica permitir a los usuarios que realicen subidas de grandes ficheros en la mayoría de sistemas, ya que podrían ocupar sistemas de análisis como este, de forma malintencionada, incurriendo en un mayor coste y causando un cuello de botella. Con esta limitación el sistema no se va a ver saturado por ficheros grandes.

Otros puntos interesantes de la Lambda, es la definición de variables de entorno a partir de los outputs de otros módulos. Estas variables de entorno pueden ser consumidas por el controlador, en este caso para saber cuál es el bucket de destino y el topic de SNS.

```
inputs = {
  function_name = local.environment_vars.locals.lambda_antivirus_scanner_name
  //handler = "index.handler"
  package_type                    = "Image"
  runtime                         = "nodejs16.x"
  timeout                         = 900
  memory_size                     = 3008
  ephemeral_storage_size          = 8096
  cloudwatch_logs_retention_in_days = 90
  create_package                  = false
  create_role                     = false

  # Set the function code and dependencies
  image_uri = "${local.account_id}.dkr.ecr.${local.aws_region}.amazonaws.com/${local.environment_vars.locals.ecr_repository_name}:latest"

  # Set the environment variables for ClamAV and the quarantine bucket
  environment_variables = {
    S3_PRODUCTION_BUCKET = local.environment_vars.locals.s3_production_bucket_name
    SNS_ARN_TOPIC        = dependency.sns.outputs.sns_topic_arn
  }

  # Set the IAM role for the function to allow it to access S3 and write to the production bucket
  lambda_role = dependency.lambda_role.outputs.iam_role_arn
  tags        = local.environment_vars.locals.tags
}
```

Figura 7-Inputs o parámetros de configuración de la Lambda.

Siempre es recomendable intentar usar un módulo oficial, ya que cuentan con una amplia funcionalidad, están testeados y están respaldados por la comunidad, en la cual podríamos buscar apoyo si encontramos algún problema en su uso. Sin embargo, el uso de un módulo oficial o uno local, dependerá de las necesidades del sistema.

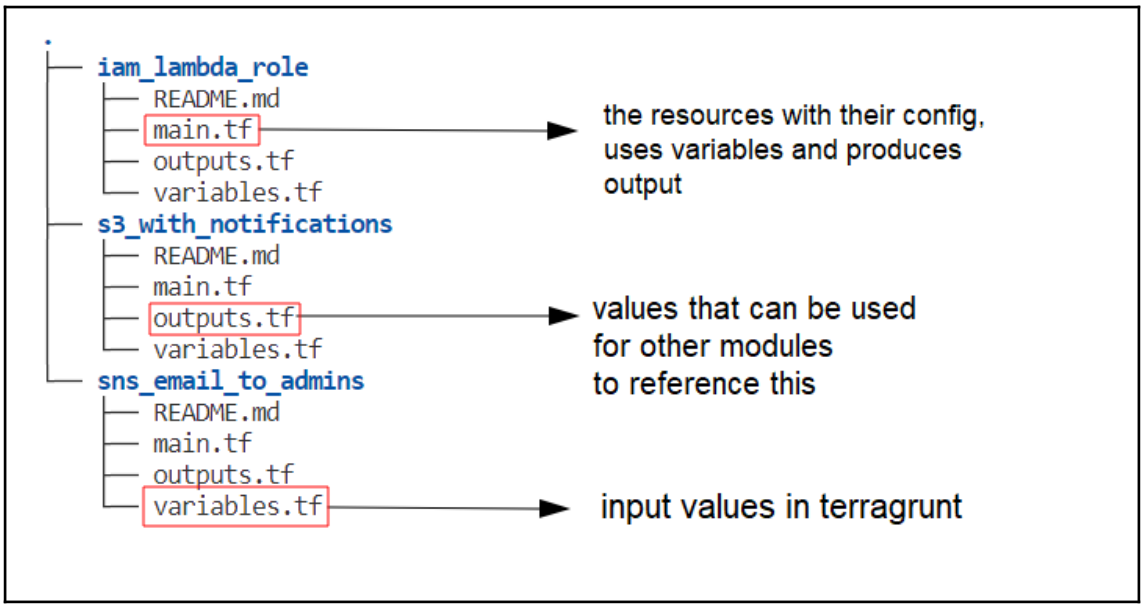

Figura 8-Estructura de ficheros de la carpeta /modules.

Por otro lado, el proyecto ha necesitado definir 3 módulos locales que añaden funcionalidad que el módulo base oficial no contempla, como definir un rol y enlazar una política personalizada al mismo tiempo, tener bucket de s3 con la posibilidad de activar notificaciones y una política de ciclo de vida según el caso, o suscribir emails a un determinado topic de SNS. Los módulos locales se pueden encontrar bajo la carpeta `/modules`. Cada módulo se define por 3 archivos, que podemos ver en la figura 8.
Los módulos definidos son:

Módulo iam_lambda_role. Crea un nuevo rol y le adjunta una nueva política. Esta tiene los permisos con mínimo privilegio para que la función Lambda tenga permisos de get, put y tag, solo sobre los dos buckets de s3 y sus objetos. También para que pueda obtener la imagen del repositorio de ECR, y pueda publicar en el topic de SNS. Este nivel de granularidad es una buena práctica ya que nos va a aportar más seguridad en el sistema, limitando el impacto de la Lambda si se consiguiera explotar alguna vulnerabilidad. El único punto negativo, es que para mantener este nivel de granularidad, se debe mantener el nombre por defecto de los recursos de la arquitectura, o en su defecto ajustarlos en este módulo si se realiza algún cambio de nomenclatura. Podemos ver esta política al detalle en la figura 9.

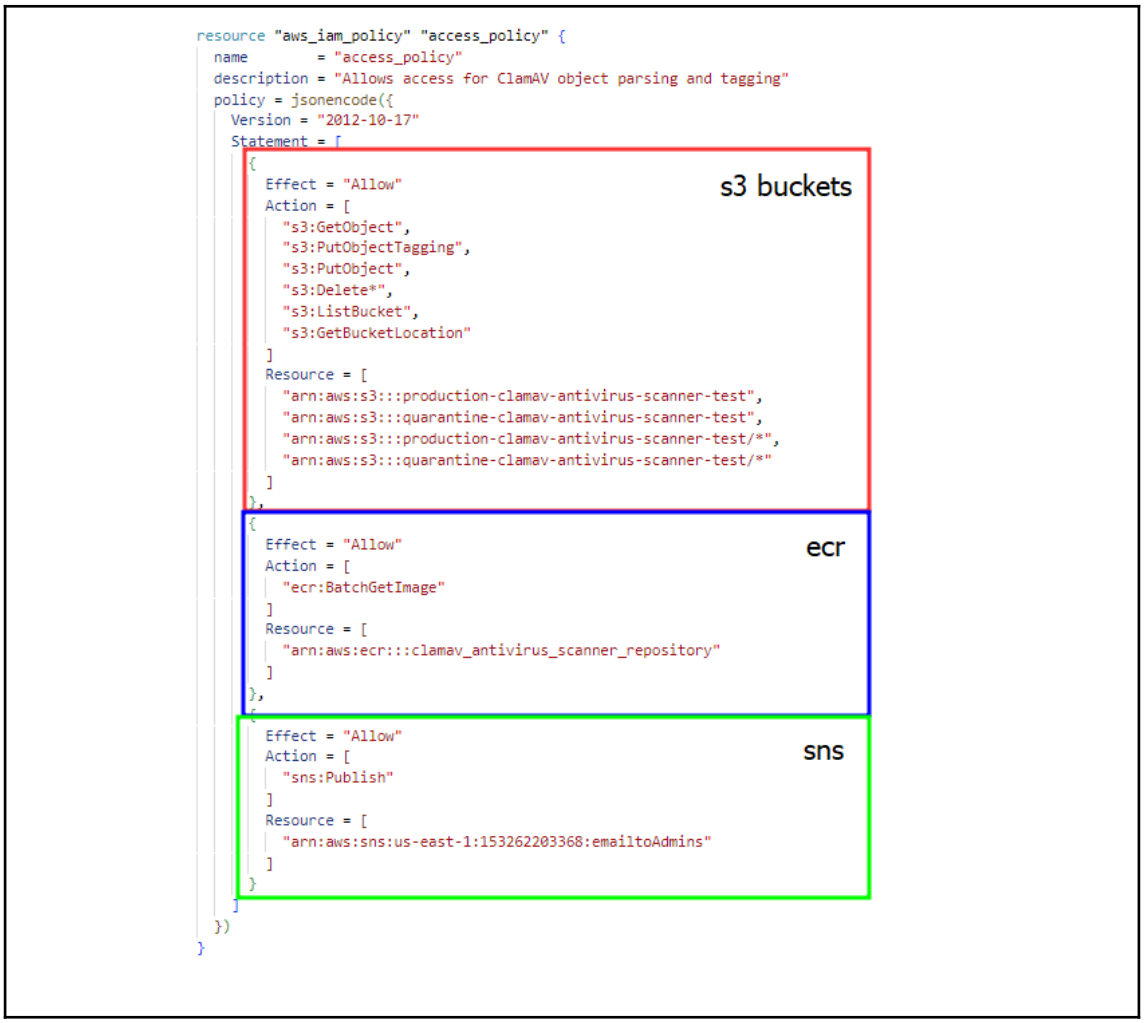

Figura 9-Política definida del módulo iam_lambda_role.

Módulo s3_with_notifications. Este módulo se usa para crear los dos buckets, con la peculiaridad que para el bucket de cuarentena, se puede seleccionar de manera opcional si añadir una política de ciclo de vida para destruir los objetos transcurridos ciertos días después de su creación. Se puede apreciar esta política en la figura 10.

```
# The objects that remain in the quarantine bucket will be deleted
resource "aws_s3_bucket_lifecycle_configuration" "bucket_lifecycle_configuration" {
  count = var.create_lifecycle_configuration ? 1 : 0

  rule {
    id     = "delete-objects-after-${var.lifecycle_expiration_days}-days"
    status = "Enabled"

    expiration {
      days = var.lifecycle_expiration_days
    }
  }

  bucket = aws_s3_bucket.bucket.id
}
```

Figura 10-Política de ciclo de vida del bucket de S3 de cuarentena.

Por otro lado también se puede añadir una notificación que activará la Lambda cuando se cree un objeto en el bucket. En la figura 11 podemos observar cómo esta opción es opcional pudiendo reutilizar el módulo para crear un bucket normal sin notificaciones.

```
# Define the trigger that is the object created in the bucket
resource "aws_s3_bucket_notification" "bucket_notification" {
  count  = var.create_notification ? 1 : 0
  bucket = aws_s3_bucket.bucket.id

  lambda_function {
    lambda_function_arn = var.notification_lambda_function_arn
    events              = ["s3:ObjectCreated:*"]
  }
}
```

Figura 11-Recurso para crear una notificación de S3 en el módulo S3_with_notifications.

Módulo sns_email_to_admins. Crea un topic de sns y añade como *end-point* una lista de emails que recibe como input. En este caso será una lista de correos con los administradores del sistema. En la figura 12 se observa cómo se añade la lista de correos como endpoint.

```
# The new topic
resource "aws_sns_topic" "sns_topic" {
  name = var.sns_topic_name
}

# Add as endpoint a list of admin pdl emails
resource "aws_sns_topic_subscription" "sns_subscription" {
  count     = length(var.email_addresses)
  topic_arn = aws_sns_topic.sns_topic.arn
  protocol  = "email"
  endpoint  = var.email_addresses[count.index]
}
```

Figura 12-Definición del recurso de suscripción al topic SNS en el módulo sns_emails_to_admins

3.3.1.3 Gestión de dependencias

Terragrunt ofrece funcionalidad para gestionar las dependencias entre los módulos de Terraform. Al definir explícitamente las dependencias entre módulos, Terragrunt garantiza el aprovisionamiento de los recursos necesarios en el orden adecuado. Esta capacidad es especialmente valiosa cuando se trata de despliegues de infraestructura complejos, ya que proporciona un medio para agilizar su gestión. Además nos permite usar los outputs de dichos módulos antes de su creación. Por el contrario, según [39] y [40] AWS CloudFormation y AWS CDK no proporcionan funciones integradas de gestión de dependencias tan amplias como las que ofrece Terragrunt. En la siguiente figura 13 podemos ver un bloque de dependencias de la función Lambda. En concreto depende del rol IAM para su creación.

```
dependency "lambda_role" {
  config_path = "../IAM"

  mock_outputs_merge_strategy_with_state  = "shallow"
  mock_outputs_allowed_terraform_commands = ["validate", "plan", "destroy"]
  mock_outputs = {
    iam_role_arn = "ecr-arn-2123"
  }
}
```

Figura 13-Ejemplo de definición de dependencias en el código del recurso Lambda.

3.3.1.4 Legibilidad del código.

Terraform adopta un enfoque declarativo de la infraestructura como código, permitiendo a los usuarios definir el estado deseado de su infraestructura mediante código. Este enfoque mejora la comprensibilidad y la capacidad de mantenimiento de las configuraciones de infraestructura a lo largo del tiempo. Según [39] AWS CloudFormation también adopta un modelo declarativo, mientras que según [40] AWS CDK se inclina más hacia un enfoque imperativo, requiriendo que el código se escriba en un lenguaje de programación. El código de Terragrunt se puede dividir en tres bloques: import, variables de entorno e inputs.

En el import se indica la fuente del módulo que se va a usar. Los módulos se pueden versionar, esto ayuda a tener un mayor seguimiento de los cambios del código. En la figura 14 se aprecia como el módulo de la Lambda está usando la versión 4.14.0.

```
terraform {
  source = "tfr:///terraform-aws-modules/lambda/aws//.?version=4.14.0"
}
```

Figura 14-Ejemplo de import de un módulo en Terragrunt.

En la figura 15, se puede apreciar como las variables de entorno se obtienen de los ficheros de configuración `account.hlc, region.hcl, env.hcl`.

```
# Include the necessary variables
locals {
  # Automatically load account-level variables
  account_vars = read_terragrunt_config(find_in_parent_folders("account.hcl"))

  # Automatically load region-level variables
  region_vars = read_terragrunt_config(find_in_parent_folders("region.hcl"))

  # Automatically load environment-level variables
  environment_vars = read_terragrunt_config(find_in_parent_folders("env.hcl"))

  # Extract the variables we need for easy access
  account_name = local.account_vars.locals.account_name
  account_id   = local.account_vars.locals.aws_account_id
  aws_region   = local.region_vars.locals.aws_region
}

dependency "ecr_repository" {
  config_path = "../ECR"

  mock_outputs_merge_strategy_with_state  = "shallow"
  mock_outputs_allowed_terraform_commands = ["validate", "plan", "destroy"]
  mock_outputs = {
    repository_arn = "ecr-arn-2123"
  }
}
```

Figura 15-Ejemplo de variables de entorno y dependencias en Terragrunt.

Las dependencias con otros módulos también sirven para exportar valores. En las dependencias se declaran mocks los cuales establecen valores por defecto, en caso de que el recurso dependiente no esté creado y no se pueda exportar dicho valor. Esto es útil para un despliegue de toda la infraestructura, ya que estos valores se usan en la fase del plan.

Y por último los input de configuración, donde se configura el módulo con los valores necesarios para su despliegue. Se puede ver un ejemplo en la figura 7.

3.3.1.5 Gestión de estados.

Según [37], Terraform mantiene un archivo de estado que registra el estado de los recursos que gestiona. Este archivo de estado se utiliza para realizar un seguimiento del estado actual de la infraestructura y ayuda a Terraform a comprender los cambios entre los estados deseados y reales de la infraestructura. El archivo de estado se almacena localmente por defecto, pero también soporta backends de estado remotos, como Amazon S3. Terragrunt simplifica la configuración de este backend y proporciona características adicionales para el bloqueo de estado, lo que ayuda a prevenir modificaciones concurrentes y garantiza la seguridad del estado de la infraestructura.

El almacenamiento remoto de estados se recomienda para flujos de trabajo colaborativos y para compartir estados entre equipos. El uso del estado remoto permite que varios usuarios trabajen en la misma base de código de infraestructura, lo que facilita la colaboración y evita conflictos.

Por lo que aunque AWS CloudFormation también gestiona el estado, la gestión de estado de Terraform es más flexible, lo que permite compartir y versionar más fácilmente los estados de la infraestructura.

En el proyecto, podemos identificar el uso de los ficheros de bloqueo, y la creación de un bucket de S3 de forma automática, la primera vez que aplicamos Terraform en la cuenta con este proyecto. El encargado de este backend es el fichero `terragrunt.hcl` en la carpeta raíz en la figura 4.

3.3.1.6 Gestión del código y trabajo con terraform-terragrunt.

Surge como una propiedad emergente de las características anteriores. Hacen que los proyectos se puedan gestionar realmente como código, mediante un repositorio de GitHub y un pipeline de CI/CD (Continuous Integration and Continuous Delivery/Deployment). Donde varios usuarios pueden integrar cambios a la rama principal, que otros usuarios revisen y validen dicho código y el plan que generaría. Y tras aprobar dichos cambios se desplegarán en la infraestructura, en un determinado entorno. Teniendo la posibilidad de un *rollback* si el cambio dejará inconsistente el sistema, mediante algo tan sencillo como un uncommit del último cambio.

3.3.2 Runtime de la Lambda.

La función Lambda usará un runtime custom, con la base de datos de virus y el controlador. El runtime tiene un impacto en el tiempo de ejecución y despliegue de la Lambda, así que para crear este runtime custom, se usará como punto de partida alguno de los runtime nativos, los cuales ya están optimizados para su ejecución en una Lambda. Los runtimes nativos soportados por AWS son Python, Nodejs, Java, .Net, Go y Ruby.

En la siguiente figura 16, extraída de la fuente [42], podemos ver una comparativa entre los distintos tiempos de ejecución medios según el runtime escogido. Como podemos ver el lenguaje más rápido es Go, seguido de Python y luego Nodejs.

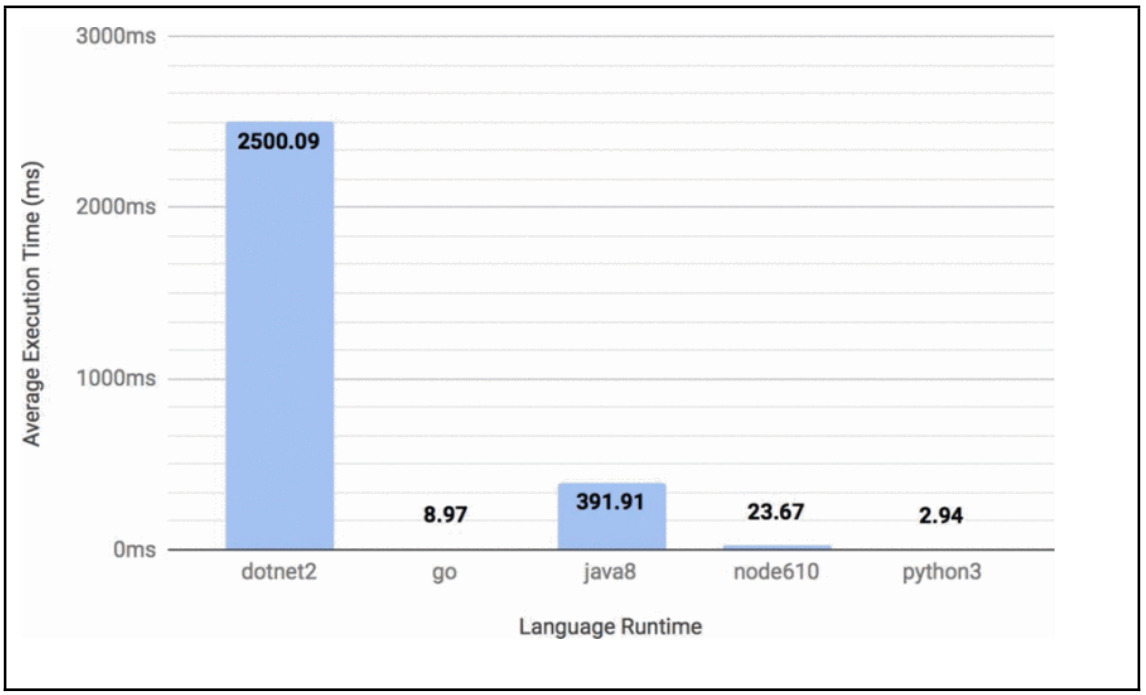

Figura 16-Tiempo promedio de ejecución (ms) en todas las pruebas de arranque en frío en AWS Lambda.
Imagen extraída de [42]

Otras fuentes más recientes, como [43], realizan una comparativa en este caso del tiempo de despliegue en frío, teniendo en cuenta el caso específico de una Lambda cuyo código a ejecutar se encuentra en la propia imagen del runtime. Este caso coincide con el contexto de nuestro proyecto donde el handler formará parte del runtime, y no estará encapsulado en ningún .zip ni artefacto. En este caso, tal y como se aprecia en la figura 17, solo se compara Nodejs, Python y Java. Resultando más favorable para Nodejs, seguido de Python y Java. La explicación que da el autor es el uso de nuevas versiones de cada uno de los runtimes.

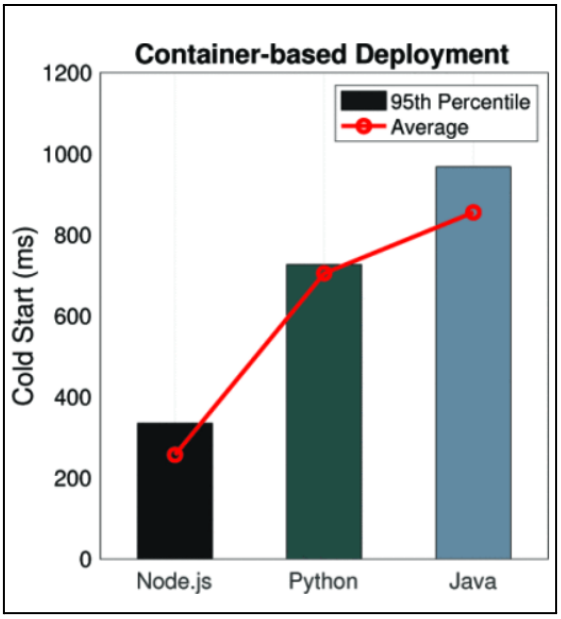

Figura 17-Función factorial con diferentes tiempos de ejecución de lenguaje y configuración de memoria de 128 MB.
Imagen extraída de [43]

Por otro lado, además del tiempo de ejecución, se debe tener en cuenta el tamaño de la imagen del runtime, ya que se va a agregar la base de datos, y esto hará que el despliegue sea más costoso. En la siguiente tabla 2, podemos ver una comparativa del tamaño de los runtime más interesantes, donde el runtime con menor tamaño es Nodejs, seguido de la versión de Python 3, y posteriormente Go, y Python 2.7.

	public.ecr.aws/lambda/go:latest	**public.ecr.aws/lambda/nodejs:16-x86_64**	**public.ecr.aws/lambda/python:2.7**	**public.ecr.aws/lambda/python:3**
Tamaño	335 MB	153 MB	342 MB	185 MB (max.)

Tabla 2-Comparativa de tamaños por defecto de cada runtime.
Datos generados según las fuentes [44],[45] y [46]

Finalmente, se debe evaluar si existe alguna librería que facilite el uso del escáner de malware de ClamAV en estos lenguajes. Para cada estos runtime respectivamente se encuentran las siguientes librerías o wrapper que en mayor o menor medida facilitan el uso básico de la herramienta.

- clamd-python. Disponible en https://pypi.org/project/clamd/.
- go-clamav. Disponible en https://pkg.go.dev/github.com/ca110us/go-clamav.
- clamscan. Disponible en https://www.npmjs.com/package/clamscan. Incluye bastantes parámetros de configuración en comparación con las anteriores.

Según los datos expuestos, aunque el tiempo de ejecución con Python o Go es menor, la memoria y el tiempo de despliegue es menor en Nodejs, por lo que podría ser más adecuado para este contexto. Aunque los tres lenguajes son válidos candidatos, también se tienen en cuenta otros factores como la familiaridad y afinidad con el lenguaje y el stack tecnológico.

3.3.3 Proceso de update y código del controlador

El código del controlador se encuentra disponible en un repositorio público de la plataforma GitHub, al cual se puede acceder mediante el siguiente enlace:

```
https://github.com/toromuu/Serverless_Malware_Scanner-Lambda-Runtime
```

En la figura 18, se puede apreciar la estructura de ficheros de este repositorio. Donde encontramos tres componentes, por un lado, la definición del trabajo de GitHub que se encargará de construir la imagen de forma recurrente, por otro el Dockerfile con la definición del proceso de construcción de la imagen del runtime de la Lambda, y finalmente el código del controlador y sus dependencias.

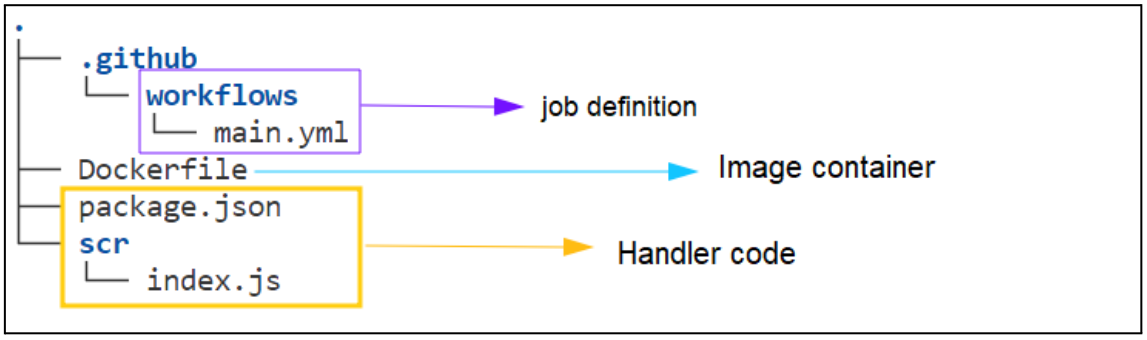

Figura 18- Estructura de ficheros del repositorio con la definición del runtime de la Lambda.

3.3.3.1 Construcción de la imagen

Para construir la imagen con el runtime de la Lambda, se hará uso de la tecnología de contenedores Docker. En la siguiente figura 19 se puede encontrar el `dockerfile` con el cual se construirá el runtime de la Lambda.Para ello, se parte desde el runtime `public.ecr.aws/lambda/nodejs:16`. A continuación se instala el escáner de malware. Se actualiza la definición de la base de datos de virus con el comando `freshclam`. Se copian al contenedor los ficheros necesarios, como dependencias de Javascript y el propio controlador. Finalmente, se indica cuál va a ser el controlador del runtime.

```
FROM public.ecr.aws/lambda/nodejs:16

# Install Clamav Engine
RUN yum update -y && yum install -y clamav

# Update local database virus definitions
RUN freshclam

# Copy the dependencies required
COPY package*.json ./

# Copy the handler
COPY ./scr/index.js .

# Install the dependencies
RUN npm install

CMD ["index.handler"]
```

Figura 19- Dockerfile del runtime de la Lambda.

3.3.3.2 GitHub Actions CI/CD.

Dicha imagen, debe construirse de forma periódica para actualizar la base de datos de virus.

Para ello necesitamos una plataforma o sistema que nos permita ejecutar un job que cumpla esta función de construcción. La gran mayoría de empresas cuentan con plataformas de CI/CD como Jenkins, Azure Devops entre otros. En este caso, se usa la plataforma GitHub Actions, que permitirá de manera gratuita , con una sencilla configuración, ejecutar trabajos y al mismo tiempo gestionar el repositorio del código del runtime de la Lambda.

Para ello se ha definido el archivo `main.yml` en `./.git/workflows`. Este fichero, figura 20, está estructurado en serie de steps o jobs. Cada job realiza una tarea concreta. Con cierta abstracción se puede apreciar que las tareas a realizar son en primer lugar identificarse en ECR y en AWS para obtener los permisos necesarios para realizar las siguientes tareas. Para ello se usarán secretos y variables. La configuración de estos secretos se detalla en el Anexo I. A continuación se construye la imagen, se le añade un tag, y se sube al repositorio. Una vez subida, se fuerza a la Lambda a actualizar su referencia a la nueva imagen, de tal forma que cuando se vuelva a invocar lo haga esta vez usando el nuevo runtime que se acaba de subir.

Por otro lado, todas estas tareas se pueden realizar bajo demanda, o en este caso con un CRON, a media noche, coincidiendo con menor tráfico de usuarios, haciendo que el trabajo anteriormente descrito se ejecute en GitHub Actions.

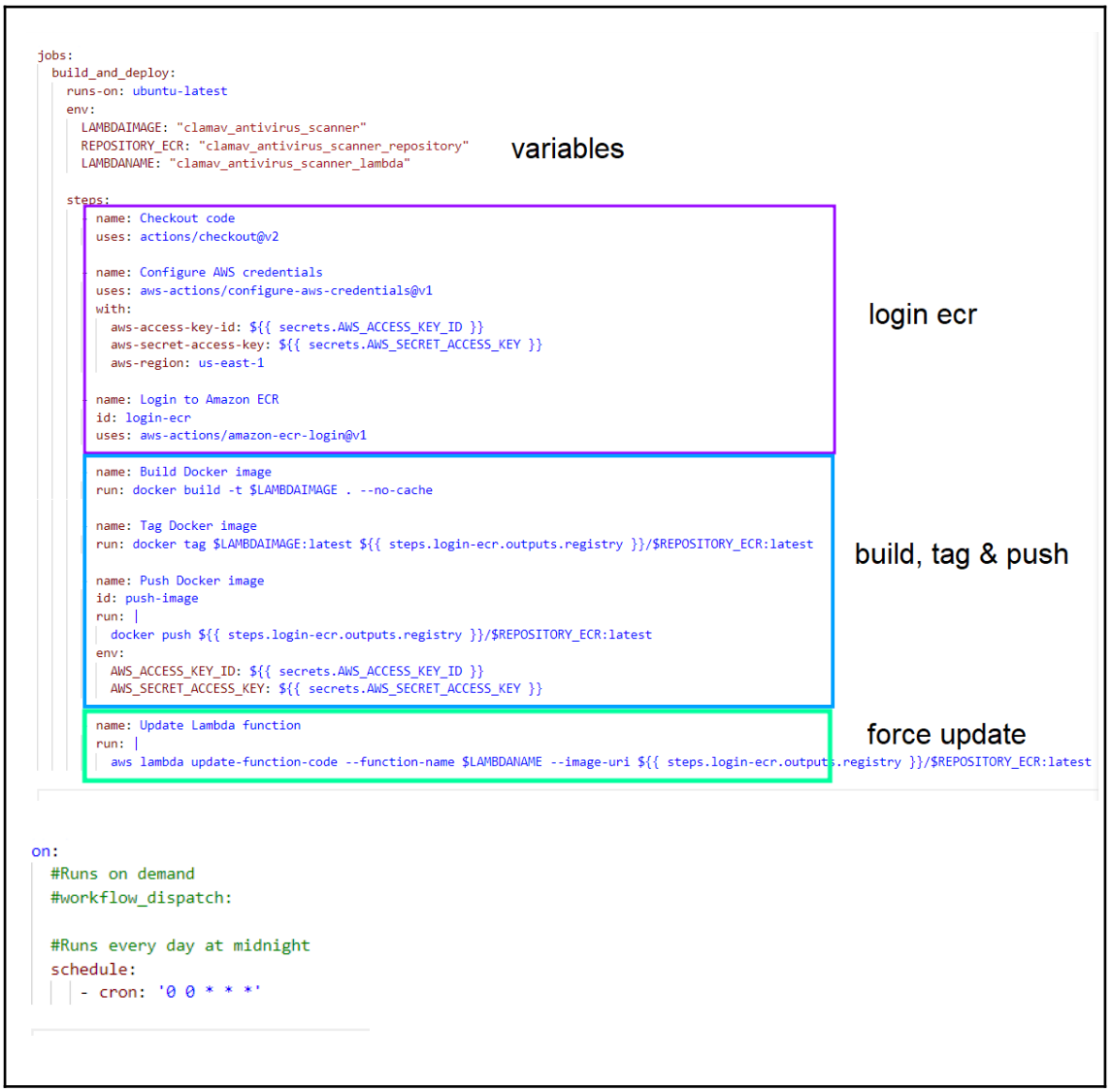

Figura 20-Fichero ./.git/workflows.

3.3.3.3 Código del controlador

Cómo se usará un runtime basado en Nodejs, el controlador será desarrollado usando la tecnología de Javascript. El controlador está definido en el archivo `index.js`.

Este controlador usa solo las dependencias que podemos observar en la figura 21, con el fin de ser lo más ligero posible. Solo hace uso del SDK de AWS para las operaciones con los buckets y el envío de la notificación. El módulo de clamscan para manejar el escáner de malware, y el módulo fs para crear ficheros temporales.

```
"aws-sdk": "^2.964.0",   //Import NodeClam module for malware scanning
"clamscan": "^2.1.2",    //Import AWS SDK for AWS operations
"fs": "^0.0.1-security" //Import fs module for file system operations
```

Figura 21- Dependencias del controlador. Fichero package.json.

En la figura 22, se muestra cómo pueden recuperarse las variables de entorno que se indican en la creación del recurso de la Lambda.

```
const region = process.env.AWS_REGION;  // Get AWS region from environment variable
const production_bucket = process.env.S3_PRODUCTION_BUCKET;  // Get production or destionation S3 bucket from environment variable
const sns_topic_arn = process.env.SNS_ARN_TOPIC; // Get topic for send email
```

Figura 22 - Código del controlador index.js. Variables de entorno.

En la figura 23, se muestra como el objeto se obtiene a partir del evento de creación en S3. Si es un fichero temporal en bucket, el análisis fallará porque no podrá recuperarlo. El fichero tiene que recuperarse como un fichero temporal, ya que el scanner no admite un stream de datos.

```
// Get source bucket and object key from S3 event
const sourceBucket = s3Event.Records[0].s3.bucket.name;
const objectKey = decodeURIComponent(s3Event.Records[0].s3.object.key.replace(/\+/g, " "));

// Download object from S3 and save it to a localtemporal file
const objectToScan = await s3.getObject({
  Bucket: sourceBucket,
  Key: objectKey,
}).promise();

// Need to recreate the same folder structure if user upload entire folder
const localPath = `/tmp/${objectKey}`;
const localDir = path.dirname(localPath);
fs.mkdirSync(localDir, { recursive: true });
fs.writeFileSync(`/tmp/${objectKey}`, objectToScan.Body);
console.log(`File ${objectKey} written successfully\n`);
```

Figura 23-Código del controlador index.js. Obtención del objeto de S3.

En la figura 24 se puede observar cómo se inicializa el scanner. Destacar que en cada ejecución se muestra la versión de la base de datos de virus, esta versión está identificada por la fecha. El comportamiento esperado si se está actualizando de manera correcta, es que la versión coincida con la fecha actual o un par de días anteriores.

```
// Initialize virus scanner and perform scan
const ClamScan = new NodeClam();

await ClamScan.init(options)
  .then(async (clamscan) => {
   try {
    console.log(clamscan);

    const version = await clamscan.getVersion();
    // Log the version of virus database, like 1-2 days is updated with new definitions
    console.log(`ClamAV Version: ${version}`);

    // Scan the object
    const { err, file, isInfected, viruses } = await clamscan.isInfected(localPath);

    // If malware is detected, maintain the object in the quarantine bucket
    if (isInfected === true) {
      console.log("INFECTED");
```

Figura 24-Código del controlador index.js. Inicialización del scanner y análisis.

Los objetos se etiquetan con el resultado del análisis,la marca de tiempo y el virus encontrado para su posterior estudio, tal y como se muestra en la figura 25.

```
    // Add tags to the objects in S3
    await s3.putObjectTagging({
      Bucket: sourceBucket,
      Key: objectKey,
      Tagging: {
        TagSet: [
          { Key: "Status", Value: "INFECTED" },
          { Key: "Timestamp", Value: timestamp },
          { Key: "Virus", Value: viruses.toString() },
            ],
          },
        })
        .promise();
      console.log(`Virus found, ${objectKey}`);
```

Figura 25-Etiquetado de los objetos analizados.

El código hace uso buenas prácticas como el uso de bloques *try-catch* para controlar todos los posibles errores tal y como se muestra en la figura 26.

```
const params = {
    Message: message,
    Subject: 'AWS Notification',
    TopicArn: sns_topic_arn
};
try {
    const result = await sns.publish(params).promise();
    console.log('SNS message sent:', result);
    return {
        statusCode: 200,
        body: 'SNS message sent successfully'
    };
} catch (error) {
    console.error(error);
    return {
        statusCode: 500,
        body: 'Error sending SNS message'
    };
}
```

Figura 26-Código del controlador index.js. Control de errores.

El último aspecto a destacar es que el mensaje que se envía a los administradores o *security champions* en caso de que se encuentre un fichero infectado, está definido como una constante tal y como se muestra en la figura 27. Es sencillo, pero informa del objeto infectado, del virus, y la marca de tiempo.

```
const message = `[Critical AWS]: Virus Found in S3 Bucket

Hello Security Champions,

This email is to notify you that a virus has been found in the S3 bucket. Immediate action is required t

  Details:

  Object: ${objectKey}
  Virus name:${viruses.toString()}
  Timestamp: ${timestamp}

Please take appropriate measures to secure the S3 bucket and remove the virus promptly.

Best regards,
ClamAV Antivirus Scanner`;
```

Figura 27-Código del controlador index.js. Email para los administradores.

4. Pruebas

En este capítulo se llevaban a cabo las validaciones del proyecto.

4.1 Actualización del runtime de la función Lambda

En esta prueba se desea comprobar el correcto funcionamiento de la actualización del runtime de la Lambda y la base de datos, en primer lugar se tomará como referencia la última actualización de la imagen en el repositorio de ECR. Esta actualización se corresponde al 29 de mayo, tal y como se muestra en la figura 28.

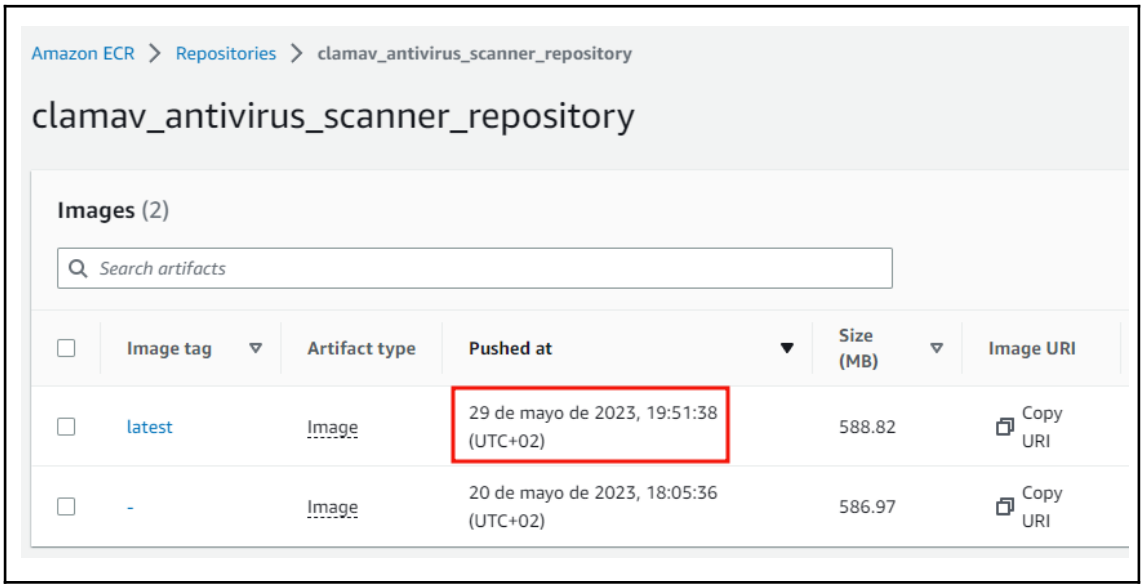

Figura 28-Última imagen en el repositorio ECR antes de la actualización.

En el repositorio del runtime de la Lambda, tal y como se muestra en la figura 29, en las opciones 1.Actions, 2.Docker Build and Deploy y 3.Run workflow, se lanzará una actualización bajo demanda.

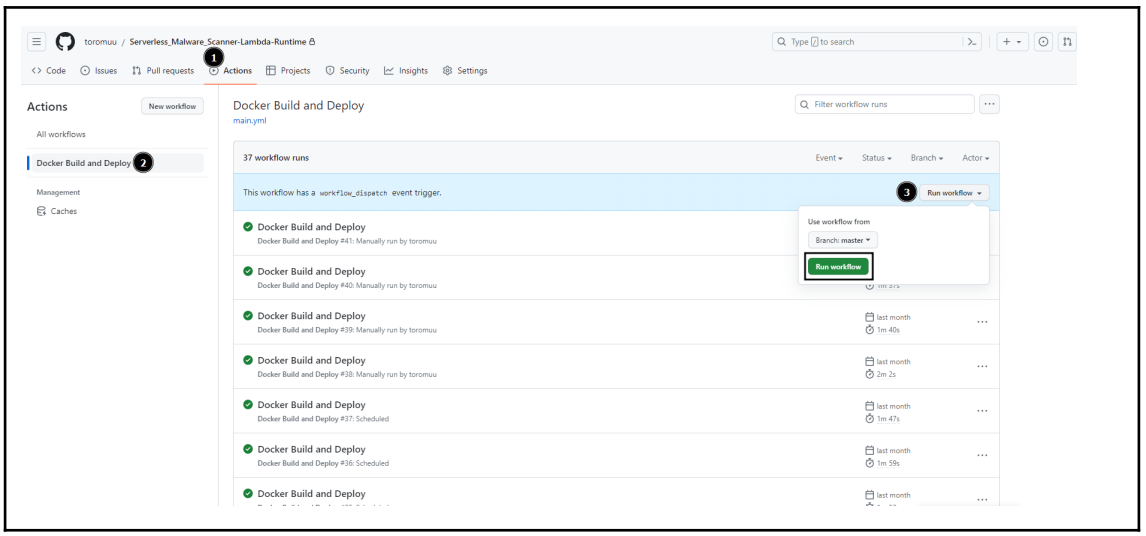

Figura 29-Menú de GitHub Actions del repositorio.

Como se puede apreciar en la figura 30, hay una nueva ejecución en marcha.

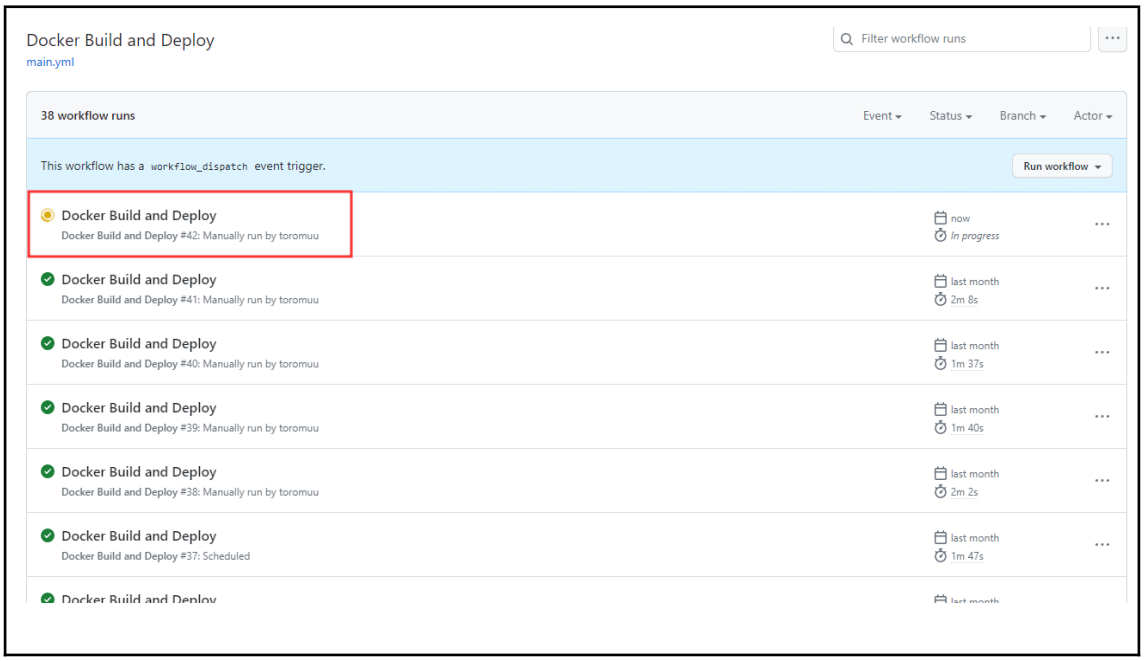

Figura 30-Ejecución del workflow de GitHub Actions.

El paso de esta nueva ejecución que se corresponde con la actualización de la base de datos con la definición de los virus es `Build Docker Image`, concretamente `RUN freshclam`. Como se aprecia en la figura 31, se esta actualización la base de datos de virus a una nueva versión correspondiente con la fecha del lunes 26 de junio de 2023.

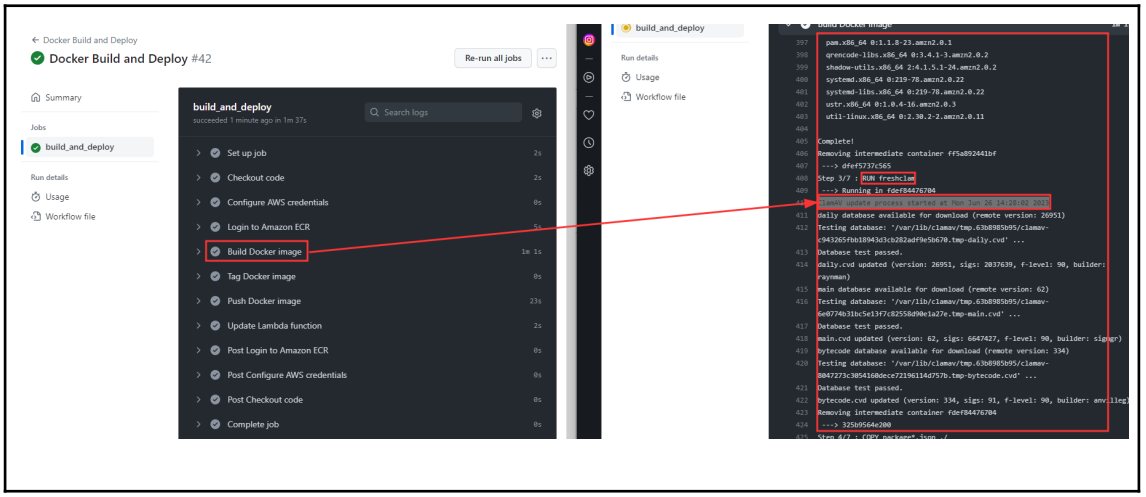

Figura 31-Step de actualización de la base de datos de virus.

Tras completarse esta ejecución, si se vuelve al repositorio de ECR , en la figura 32 se puede confirmar como última imagen subida se corresponde con la del 26 de junio, reemplazando a la del 29 de mayo.

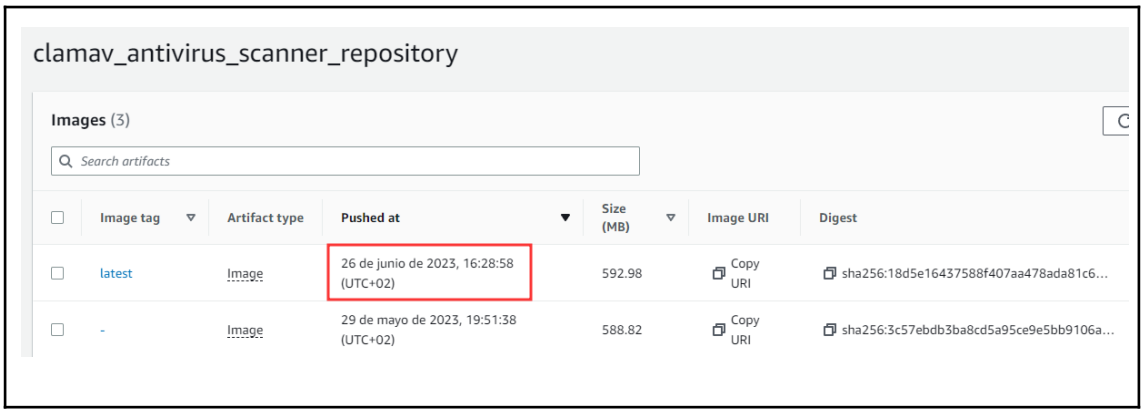

Figura 32-Última imagen en el repositorio ECR después de la actualización.

Tras confirmar que la última imagen se ha subido, queda confirmar que la Lambda la está consumiendo para su runtime, para ello se comprobará si se está usando la versión de la base de datos correspondiente.

Para ello se insertará el fichero eicar.com.txt que podemos encontrar en la fuente [47]. Se trata de un fichero de prueba desarrollado por EICAR ó el European Institute for Computer Anti-Virus Research . Dicho fichero está registrado como malware en las bases de datos de virus. Mediante el siguiente comando se insertará el fichero en el bucket de cuarentena.

```
aws s3 cp eicar.com.txt s3://quarantine-clamav-antivirus-scanner-test
```

Como se puede apreciar en la figura 33, la versión de ClamAV se corresponde con el 26 de junio de 2023 por lo que se puede deducir que se está usando la última imagen subida con la última versión de la base de datos de virus en ese momento.

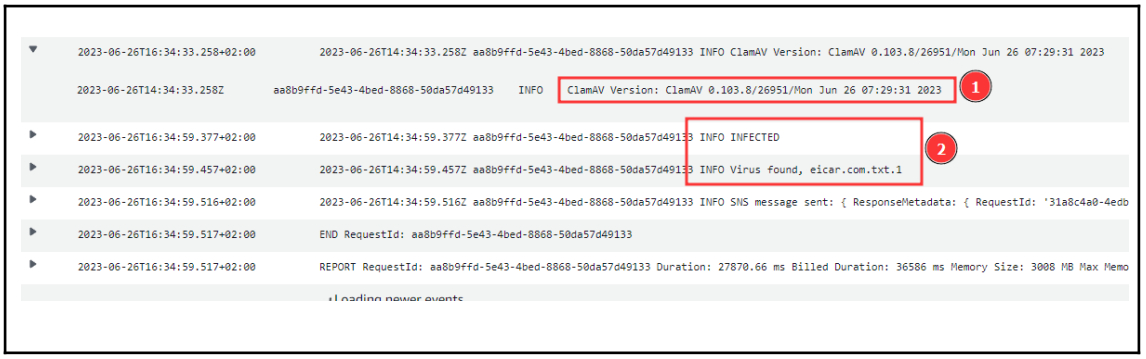

Figura 33-Logs de la Lambda escáner tras la subida del fichero eicar.

Tras finalizar el escaner de malware, se ha recibido un correo alertando de la detección y eliminación del fichero. El correo se visualiza en la figura 34.

AWS Notification from AWS Notifications <no-reply@sns.amazonaws.com>

Text (1 KB)

[Critical AWS]: Virus Found in S3 Bucket

Hello Security Champions,

This email is to notify you that a virus has been found in the S3 bucket. Immediate action is required to investigate and mitigate the issue.

Details:

Object: eicar.com.txt.1
Virus name:Win.Test.EICAR_HDB-1
Timestamp: 2023-06-26T14:34:59.377Z

Please take appropriate measures to secure the S3 bucket and remove the virus promptly.

Best regards,
ClamAV Antivirus Scanner

--
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-1:153262203368:emailtoAdmins:d3a09c13-79ac-4550-878f-31e780b4dae7&Endpoint=datorram@posgrado.upv.es

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 34-Correo de detección de malware.

4.2 Funcionalidad

En esta prueba se desea testear si el escáner es capaz de por un lado, detectar ficheros infectados con malware y por otro lado, si es capaz de analizar distintos tipos de ficheros. Para ello, se van a utilizar dos scripts.

4.2.1 case_base.sh

Este script realiza una subida de los distintos ficheros de malware de prueba `eicar` que podemos encontrar en la fuente [47] y un fichero de texto sin malware al bucket de cuarentena para su análisis. Se puede apreciar el código del script en la figura 35.

```bash
#!/bin/bash

# Array of test file URLs to download from https://www.eicar.org/download-anti-malware-testfile/
urls=(
    "https://secure.eicar.org/eicarcom2.zip"
    "https://secure.eicar.org/eicar_com.zip"
    "https://secure.eicar.org/eicar.com.txt"
    "https://secure.eicar.org/eicar.com"
    "https://fegalaz.usc.es/~gamallo/aulas/lingcomputacional/corpus/quijote-es.txt"
)

# S3 bucket name
bucket_name="s3://quarantine-clamav-antivirus-scanner-test"

# Download and upload function
# Download and upload function
function download_and_upload {
    url="$1"
    file_name="$(basename "$url")"

    # Download the file
    echo "Downloading $url..."
    curl -s -o "$file_name" "$url"

    # Upload the file to S3
    echo "Uploading $file_name to S3..."
    aws s3 cp "$file_name" "$bucket_name"

    # Remove the downloaded file
    rm "$file_name"
}

# Loop through the URLs and call the download_and_upload function
for url in "${urls[@]}"; do
    download_and_upload "$url"
done
```

Figura 35-Script case_base.sh.

El resultado tras el escaner es el esperado, en la figura 36, se aprecia como los 4 ficheros infectados permanecen etiquetados en el bucket de cuarentena, mientras que el único fichero que encontramos en producción es el fichero limpio.

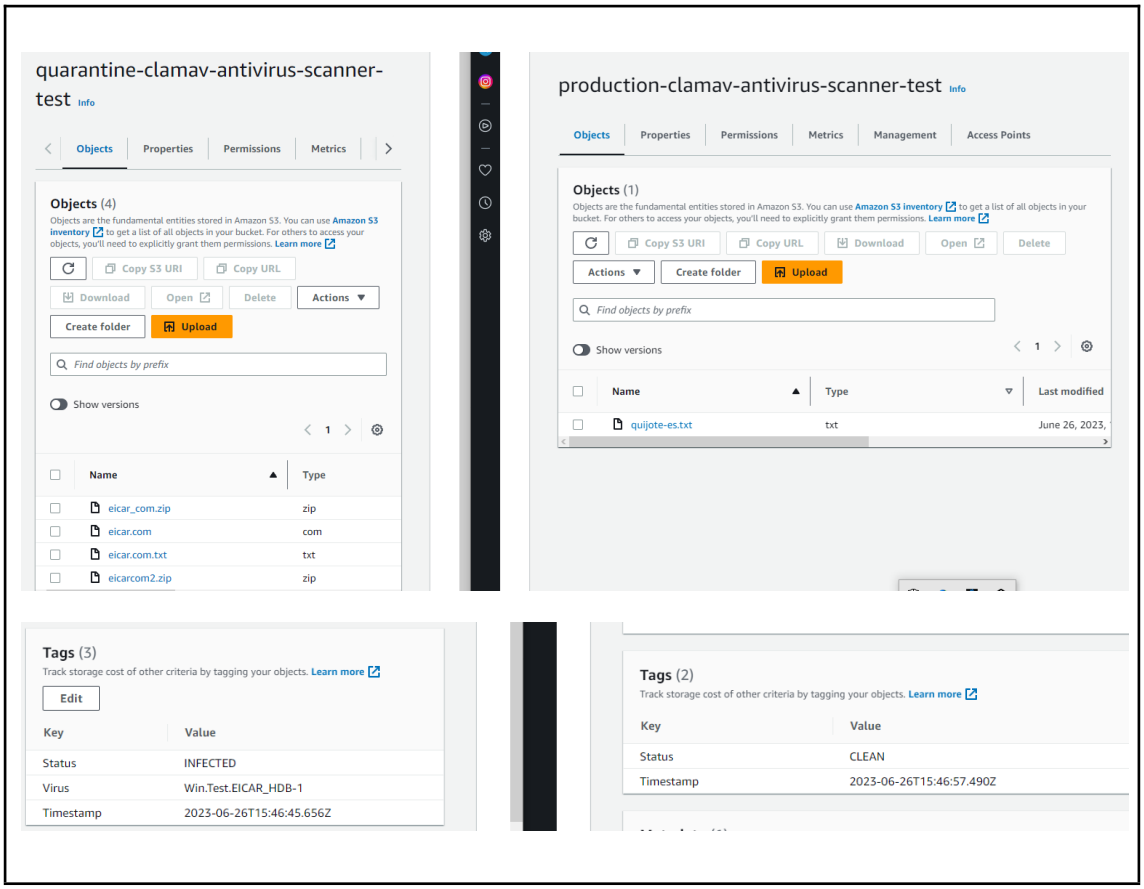

Figura 36-Estado de los buckets tras el escaneo.

4.2.2 multiple_extensions.sh

Este test consiste en comprobar si el sistema es capaz de analizar ficheros con distintas extensiones. Existe una enorme cantidad de extensiones, de las cuales se han escogido algunas de las más utilizadas: mp4, mp3, pdf, ppt, csv, doc, xls, txt, jpg, bmp, png, zip, tar.

```
# Array of file URLs to download
urls=(
    "https://filesamples.com/samples/video/mp4/sample_960x400_ocean_with_audio.mp4"
    "https://filesamples.com/samples/audio/mp3/sample4.mp3"
    "https://filesamples.com/samples/document/pdf/sample2.pdf"
    "https://filesamples.com/samples/document/ppt/sample2.ppt"
    "https://filesamples.com/samples/document/csv/sample4.csv"
    "https://filesamples.com/samples/document/doc/sample1.doc"
    "https://filesamples.com/samples/document/xls/sample2.xls"
    "https://filesamples.com/samples/document/txt/sample1.txt"
    "https://filesamples.com/samples/image/jpg/sample_640%C3%97426.jpg"
    "https://filesamples.com/samples/image/bmp/sample_640%C3%97426.bmp"
    "https://filesamples.com/samples/image/png/sample_640%C3%97426.png"
    "https://buenosaires.gob.ar/areas/educacion/quijote/pdf/don_quijote_de_la_mancha.zip"
    "https://getsamplefiles.com/download/tar/sample-1.tar"
)
```

Figura 37-Ficheros con diversas extensiones.

En la figura 37 se pueden observar los ficheros específicos que se han usado en este test. En la figura 38 se puede apreciar todos los ficheros analizados, si el escáner no fuera capaz de procesarlos no serían movidos al bucket de producción demostrando el correcto funcionamiento del escáner.

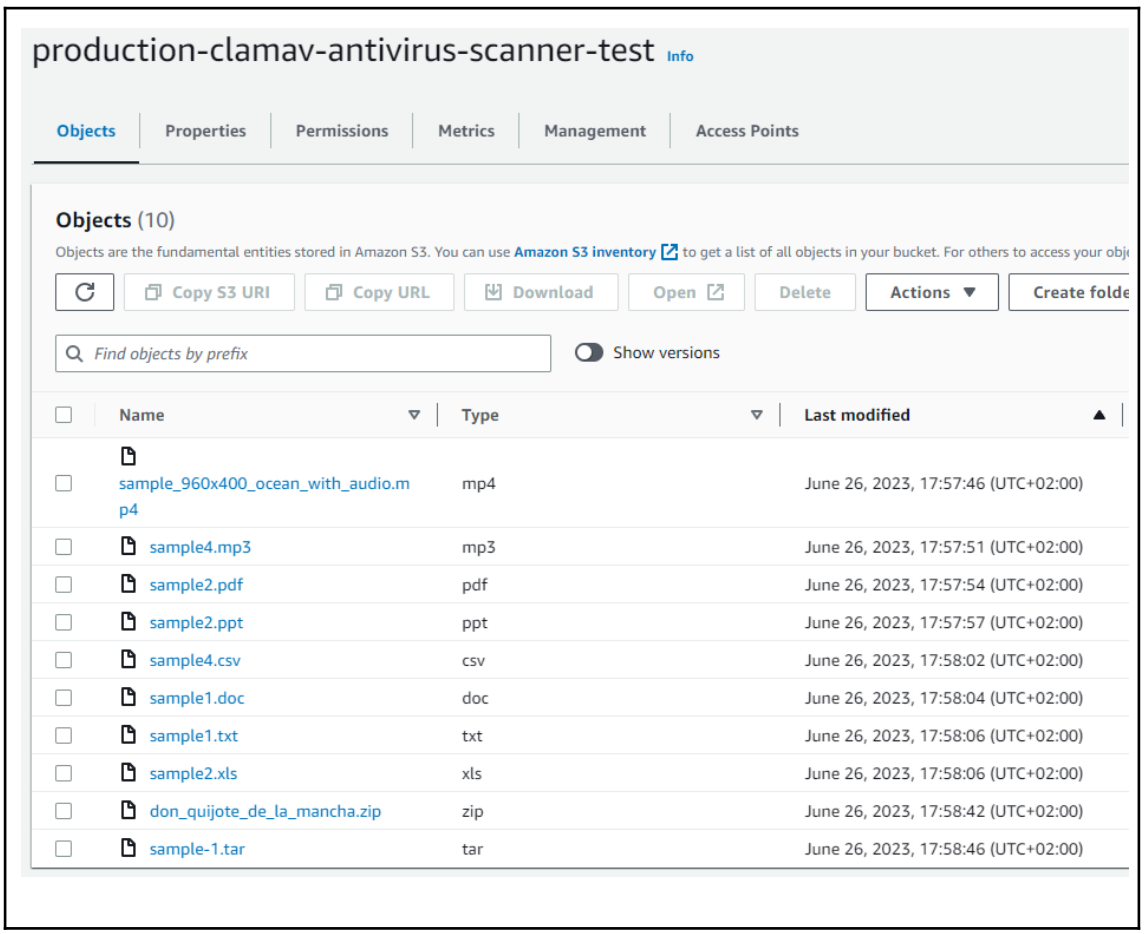

Figura 38-Ficheros con diferente extensión en el bucket de producción.

4.3 Análisis de costes

La última prueba consiste en realizar un análisis de costes de la infraestructura. En la figura 39 se puede ver un resumen de dicha estimación para el caso de un volumen de 100 GB analizados al mes. Estos 100 GB se van a subir al sistema en ficheros de media de 10 mb, resultando en unas 10.000 peticiones.

Tiene un coste mensual de 6.44 dólares. El grueso del importe radica en el cómputo de la Lambda. La Lambda tendría una configuración de arquitectura x86, un ephemeral storage allocated de hasta 8096 mb, un memory size de 3008 mb y una concurrencia máxima de 10 instancias. En cuanto al resto de recursos su precio es despreciable entrando dentro de la capa gratuita de Amazon Web Services, por ejemplo el repositorio de ECR como mucho va a almacenar dos imágenes que no suelen llegar a los 600 mb de media.

Con respecto al bucket de cuarentena S3, solo se ha considerado 5 GB ya que únicamente va a almacenar el malware detectado el cuál además debido a su política de ciclo de vida, va a ser eliminado. Ha sido configurado con un volumen de 100.000 peticiones del tipo GET, PUT y DELETE, ya que es solo un bucket de paso. Para esta estimación se ha tenido en cuenta la región de US East (N. Virginia) y no se ha incluido el coste del bucket de producción ya que se asume como un gasto que ya estaba antes de establecer el sistema.

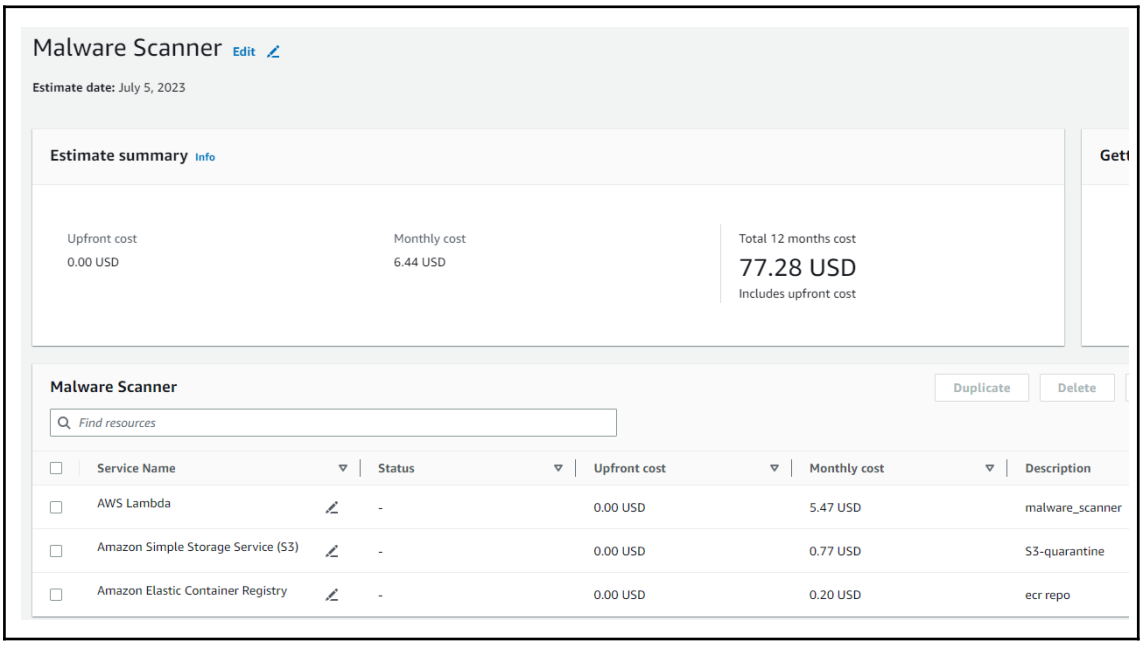

Figura 39- Estimación de costes de la infraestructura del escáner de malware.
Datos generados según la fuente [48]

5. Conclusiones y Trabajos futuros

El trabajo cumple con los objetivos propuestos en la sección de forma satisfactoria. En definitiva es una solución que permite de forma sencilla, integrar un escáner de malware capaz de analizar detectar malware en archivos con diferentes tipos de extensiones, de forma escalable y notificar de las posibles amenazas, haciendo uso de un enfoque de infraestructura como código.

El proyecto me ha servido para reforzar conocimientos transversales vistos en el máster, pues abarca desde realizar un estudio del mercado, valorar diferentes alternativas hasta decisiones de diseño en infraestructura en el cloud, buenas prácticas en el desarrollo de políticas IAM, desarrollo de código, conceptos básicos de seguridad, profundizar en el concepto de infraestructura como código. Actualmente la base de este proyecto está implementado y está siendo usado en diferentes ambientes de desarrollo, integración y producción en una empresa multinacional estadounidense, para diversas soluciones y aplicaciones.

El proyecto tiene varios puntos de mejora y enfoques a explorar. Por ejemplo, la limitación de duración de 15 mins de las funciones Lambda, limita el tamaño máximo del archivo a escanear. En la mayoría de casos de uso debería ser suficiente, y como hemos comentado no es muy buena práctica permitir a los usuarios subir archivos de gran tamaño a la plataforma, sin embargo podría ser interesante, explorar una opción basada en contenedores, empleando servicios como AWS Fargate, para analizar archivos mayores.

Aunque el sistema ofrece una versión actualizada de la base de datos de virus en el momento de analizar el fichero, cada día se descubren nuevos tipos de malware, por lo que sería interesante integrar además del análisis que se está realizando, programar un análisis retrospectivo en el bucket para identificar malware en ficheros que anteriormente fueron considerados limpios porque la versión de la base de datos aún no contemplaba el descubrimiento de estas nuevas definiciones de malware.

Por otro lado, para la descarga de la base de datos de malware se consume una fuente oficial, la cual bloquea las peticiones que no respeten cierto intervalo de tiempo. Actualmente se realiza una actualización programada a media noche, para respetar el consumo de esta fuente. Se podría valorar integrar un mirror de esta fuente y realizar actualizaciones del runtime con la base de datos de forma más frecuente, por ejemplo cada vez que se publicará una nueva versión.

Finalmente otro aspecto interesante sería valorar el desarrollo de un nuevo runtime y handler basado en otros lenguajes como Go o Python, y realizar una comparativa de rendimiento más ajustada a este caso concreto.

6. Anexo I - Implantación del proyecto

En el siguiente anexo se explica de forma detallada a modo de manual de uso cómo desplegar e implementar el proyecto en una cuenta de AWS. Para ello por un lado se desplegará la infraestructura y por otro lado el repositorio con el código del controlador.

6.1 Infraestructura

En este apartado se va a usar el código del repositorio, por lo que es recomendable clonarlo mediante el siguiente comando:

```
git clone https://github.com/toromuu/Serverless_Malware_Scanner_Terragrunt-Infrastructure-Live.git
```

6.1.1. Requisitos Infraestructura

Para desplegar la infraestructura como código se debe cumplir con los siguientes requisitos previos:

- Tener instalado en local Terraform versión 1.1.4 y Terragrunt versión v0.36.0 o más reciente.
- Tener credenciales de AWS con los permisos necesarios para la creación, destrucción o modificación de los recursos que se van a desplegar.
- Configurar el archivo `/account1/account.hcl` con el `Account-ID` y el `Account-Name` de la cuenta, tal y como se muestra en la figura 40.

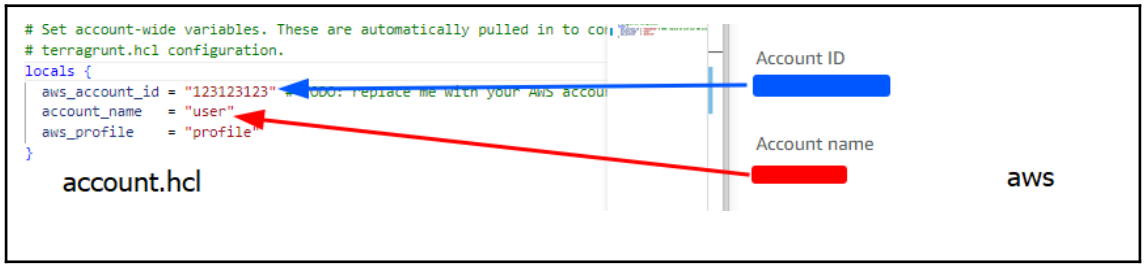

Figura 40-Configuración del fichero account.hcl.

6.1.2. Configuración del fichero de environment

Se debe configurar el valor de las variables `admins_pdl` en el fichero `env.hcl` con los emails en los cuales se desea notificar la detección de malware. También se puede modificar la variable `s3_production_bucket_name` con el nombre del bucket que se desea escanear. Además, se pueden modificar los tags de los recursos que se van a crear. Podemos ver la configuración por defecto de este fichero en la figura 41.

```
# Set common variables for the environment. This is automatically pulled in in the root terragrun
# feed forward to the child modules.
locals {
  environment                 = "dev"
  tags                        = { "Solution" : "S3_Antivirus_Scanner" }
  ecr_repository_name         = "clamav_antivirus_scanner_repository"
  lambda_antivirus_scanner_name = "clamav_antivirus_scanner_lambda"
  s3_production_bucket_name   = "production-clamav-antivirus-scanner-test"
  s3_quarantine_bucket_name   = "quarantine-clamav-antivirus-scanner-test"
  admins_pdl = [
    "admin1@enterprise.com",
    "admin2@enterpise.com"
  ]
  sns_topic_name = "emailtoAdmins"
}
```

Figura 41-Configuración del fichero env.hcl.

6.1.3. Despliegue

Para desplegar la infraestructura es recomendable seguir el workflow de Terraform-Terragrunt, que consiste en los dos siguientes pasos:

1º Plan

Sitúese en el directorio raíz account1. Ejecute el siguiente comando, esto creará el plan con todos los cambios que va a aplicar. Se puede ver un ejemplo de un plan en la figura 43.

```
terragrunt run-all plan
```

2ºApply

A continuación, en el mismo directorio, ejecute el siguiente comando, si está conforme con el plan de despliegue.

```
terragrunt run-all apply
```

El despliegue requiere de una aprobación manual, figura 42. Tras esto se puede ver un ejemplo tras el despliegue en la figura 44, donde se muestran los cambios realizados en el sistema.

```
INFO[0000] The stack at /home/dtororam/TFM/ClamAV_Antivirus_Scanner_Terragrunt-Infrastructure-Live/non-prod/us-east-1 will be processed in the following order for command apply:
Group 1
- Module /home/dtororam/TFM/ClamAV_Antivirus_Scanner_Terragrunt-Infrastructure-Live/non-prod/us-east-1/dev/ECR
- Module /home/dtororam/TFM/ClamAV_Antivirus_Scanner_Terragrunt-Infrastructure-Live/non-prod/us-east-1/dev/IAM
- Module /home/dtororam/TFM/ClamAV_Antivirus_Scanner_Terragrunt-Infrastructure-Live/non-prod/us-east-1/dev/S3/Production
- Module /home/dtororam/TFM/ClamAV_Antivirus_Scanner_Terragrunt-Infrastructure-Live/non-prod/us-east-1/dev/SNS

Group 2
- Module /home/dtororam/TFM/ClamAV_Antivirus_Scanner_Terragrunt-Infrastructure-Live/non-prod/us-east-1/dev/Lambda

Group 3
- Module /home/dtororam/TFM/ClamAV_Antivirus_Scanner_Terragrunt-Infrastructure-Live/non-prod/us-east-1/dev/S3/Quarantine

Are you sure you want to run 'terragrunt apply' in each folder of the stack described above? (y/n) y
```

Figura 42-Aprobación de un despliegue usando Terragrunt.

```
# aws_ecr_repository_policy.this[0] will be updated in-place
~ resource "aws_ecr_repository_policy" "this" {
      id          = "clamav_antivirus_scanner_repository"
    ~ policy      = jsonencode(
        ~ {
            ~ Statement = [
                  {
                      Action    = [
                          "ecr:ListTagsForResource",
                          "ecr:ListImages",
                          "ecr:GetRepositoryPolicy",
                          "ecr:GetLifecyclePolicyPreview",
                          "ecr:GetLifecyclePolicy",
                          "ecr:GetDownloadUrlForLayer",
                          "ecr:GetAuthorizationToken",
                          "ecr:DescribeRepositories",
                          "ecr:DescribeImages",
                          "ecr:DescribeImageScanFindings",
                          "ecr:BatchGetImage",
                          "ecr:BatchCheckLayerAvailability",
                      ]
                      Effect    = "Allow"
                      Principal = {
                          AWS = "arn:aws:iam::153262203368:root"
                      }
                      Sid       = "PrivateReadOnly"
                  },
                - {
                    - Action    = [
                        - "ecr:BatchGetImage",
                        - "ecr:GetDownloadUrlForLayer",
                        - "ecr:SetRepositoryPolicy",
                        - "ecr:DeleteRepositoryPolicy",
                        - "ecr:GetRepositoryPolicy",
                      ]
                    - Condition = {
                        - StringLike = {
                            - "aws:sourceArn" = "arn:aws:lambda:us-east-1:153262203368:function:*"
                          }
                      }
                    - Effect    = "Allow"
                    - Principal = {
                        - Service = "lambda.amazonaws.com"
                      }
                    - Sid       = "LambdaECRImageRetrievalPolicy"
                  },
              ]
              # (1 unchanged attribute hidden)
          }
      )
      # (2 unchanged attributes hidden)
  }

Plan: 0 to add, 1 to change, 0 to destroy.
```

Figura 43- Ejemplo de un plan en Terragrunt.

```
Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
Acquiring state lock. This may take a few moments...
aws_s3_bucket.bucket: Refreshing state... [id=quarantine-clamav-antivirus-scanner-test]
aws_s3_bucket_versioning.versioning: Refreshing state... [id=quarantine-clamav-antivirus-scanner-test]
aws_s3_bucket_acl.acl: Refreshing state... [id=quarantine-clamav-antivirus-scanner-test,private]
aws_lambda_permission.allow_s3_invoke_lambda[0]: Refreshing state... [id=AllowS3InvokeLambda]
aws_s3_bucket_notification.bucket_notification[0]: Refreshing state... [id=quarantine-clamav-antivirus-scanner-test]
aws_s3_bucket_server_side_encryption_configuration.encryption: Refreshing state... [id=quarantine-clamav-antivirus-scanner-test]
aws_s3_bucket_lifecycle_configuration.bucket_lifecycle_configuration[0]: Refreshing state... [id=quarantine-clamav-antivirus-scanner-test]

No changes. Your infrastructure matches the configuration.

Terraform has compared your real infrastructure against your configuration
and found no differences, so no changes are needed.
Releasing state lock. This may take a few moments...

Apply complete! Resources: 0 added, 0 changed, 0 destroyed.

Outputs:

bucket_arn = "arn:aws:s3:::quarantine-clamav-antivirus-scanner-test"
bucket_id = "quarantine-clamav-antivirus-scanner-test"
```

Figura 44- Ejemplo de un despliegue en Terragrunt.

6.2 Runtime update

En este apartado se va a usar el código del controlador, por lo que es recomendable clonarlo mediante el siguiente comando:

```
git clone https://github.com/toromuu/Serverless_Malware_Scanner-Lambda-Runtime.git
```

El flujo de trabajo necesita las credenciales de AWS para realizar el push a ECR y actualizar la Lambda. Así que tiene que ser configurado manualmente, tal y como se puede ver en la figura 45.

1.Abre tu repositorio de GitHub

2. Ve a la pestaña "Settings".

3. Haz clic en "Secrets/Actions" en la barra lateral izquierda.

4. Haz clic en "New Repository Secret" para crear un nuevo secreto.

5. Nombra tu secreto como "AWS_ACCESS_KEY_ID" y pega el ID de tu clave de acceso de AWS en el campo "Value".Haz click en "Add Secret" para guardar el secreto.

6. Repite el paso anterior para el secreto "AWS_SECRET_ACCESS_KEY" y pega tu clave de acceso secreta de AWS en el campo "Value".

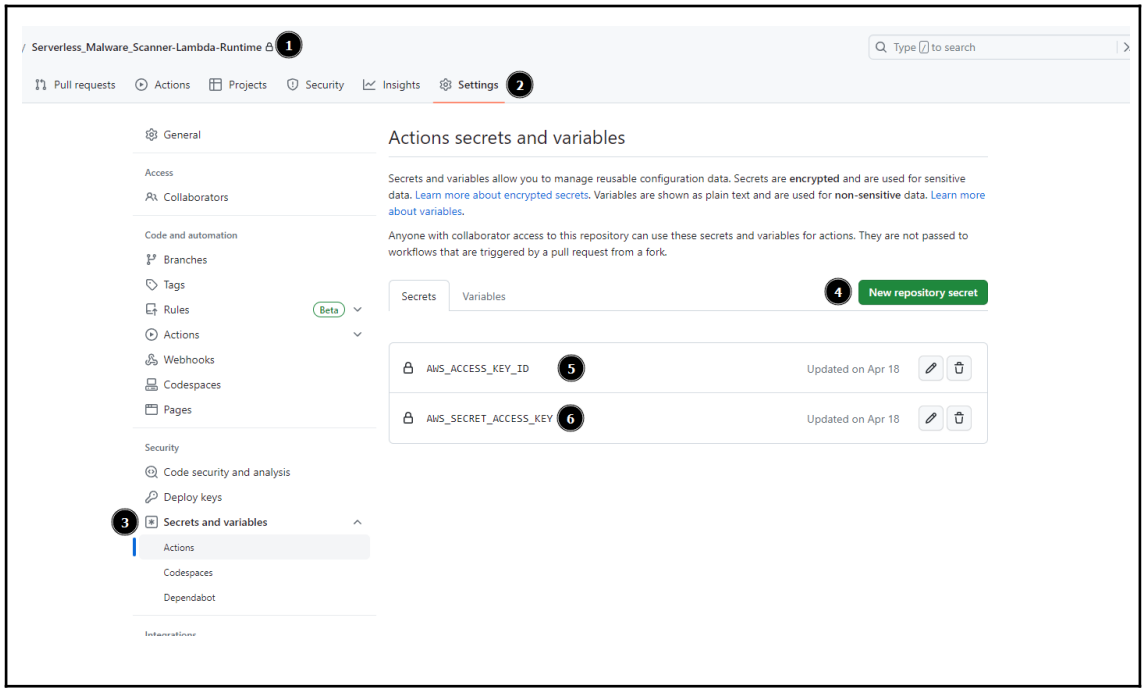

Figura 45-Configuración de los secretos en el repositorio runtime de Lambda.

Referencias

[1] REGULATION (EU) 2019/881 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL of 17 April 2019 on ENISA (the European Union Agency for Cybersecurity) and on information and communications technology cybersecurity certification and repealing Regulation (EU) No 526/2013 (Cybersecurity Act)

[2] Check point software.(2023) Cyber security report 2023. https://pages.checkpoint.com/cyber-security-report-2023.html

[3] Google. (2023, February). Fog of war research report. https://services.google.com/fh/files/blogs/google_fog_of_war_research_report.pdf

[4] IBM (2022) Cost of a Data Breach Report 2022. https://www.ibm.com/downloads/cas/3R8N1DZJ#:~:text=Average%20total%20cost%20of%20a

[5] Amazon Web Services (2022). S3. https://docs.aws.amazon.com/s3/index.html

[6] Amazon Web Services (2022).Repost.aws. https://repost.aws/questions/QUlkzLibOoTXGq2HXhdjCbBw/questions/QUlkzLibOoTXGq2HXhdjCbBw/enabling-anti-virus-on-an-s3-bucket

[7] Amazon Web Services (2022). Blogs. https://aws.amazon.com/blogs/apn/integrating-amazon-s3-malware-scanning-into-your-application-workflow-with-cloud-storage-security/

[8] Amazon Web Services (2022). MarketPlace https://aws.amazon.com/marketplace/pp/prodview-sykoblbsdgw2o

[9] J. Surbiryala and C. Rong, "Cloud Computing: History and Overview," 2019 IEEE Cloud Summit, Washington, DC, USA, 2019, pp. 1-7, doi: 10.1109/CloudSummit47114.2019.00007.

[10] Mell, P., & Grance, T. (2011). National Institute of Standards and Technology (NIST) Cloud Computing Definition. NIST Special Publication, 800-145.

[11] Gartner.com. (2022). Magic Quadrant for Cloud Infrastructure and Platform Services.https://www.gartner.com/doc/reprints?id=1-29B7RDWN&ct=220304&st=sb

[12] Amazon Web Services (2023). What is aws. https://aws.amazon.com/what-is-aws/

[13] Amazon Web Services (2023). EC2. https://docs.aws.amazon.com/ec2/index.html

[14] Amazon Web Services (2023). Lambda. https://docs.aws.amazon.com/lambda/index.html

[15] Amazon Web Services (2023). ECR. https://docs.aws.amazon.com/ecr/index.html

[16] Amazon Web Services (2022). IAM. https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles.html

[17] Amazon Web Services (2023). SNS. https://docs.aws.amazon.com/sns/index.html

[18] Amazon Web Services (2022). Autoscaling.
https://docs.aws.amazon.com/autoscaling/index.html

[19] Koret, J., & Bachaalany, E. (2015). The antivirus hacker's handbook. John Wiley & Sons.
https://www.kneda.net/documentos/The%20Antivirus%20Hacker%27s.pdf

[20] Kaspersky. (2023) Malware Remover vs. Antivirus Software.
https://www.kaspersky.com/resource-center/preemptive-safety/malware-remover-vs-antivirus-software

[22] CISA.GOV. (2023) Understanding Anti-Virus Software.
https://www.cisa.gov/news-events/news/understanding-anti-virus-software

[23] ClamAV(2023) Documentation. https://docs.clamav.net

[24] Sophos (2023) Documentation.
https://docs.sophos.com/esg/SAV-Linux/help/en-us/PDF/sav_linux_cg.pdf

[25] Sophos (2023) Scan from CLI
.https://docs.sophos.com/esg/endpoint/help/en-us/help/Scan-from-CLI/index.html#wildcards

[26] Sophos (2023) Support.
https://support.sophos.com/support/s/article/KB-000033886?language=en_US

[27] Kaspersky (2023) Support.
https://support.kaspersky.com/KIS4Mac/16.0/en.lproj/pgs/26820.htm

[28] Avast (2023) CommandLineUpdatesScan.
https://businesshelp.avast.com/Content/Products/AfB_Antivirus/ConfiguringSettings/CommandLineUpdatesScans.htm

[29] Group-in (2023) What exactly does it mean to detonate malware.
https://www.group-ib.com/resources/knowledge-hub/malware-detonation-platform/https://security.stackexchange.com/questions/157559/what-exactly-does-it-mean-to-detonate-malware

[30] Cloudstoragesec (2023) AWS Antivirus. https://cloudstoragesec.com/aws-antivirus

[31] Bucketav (2023)Bucketav. https://bucketav.com

[32] Amazon Web Services (2023) Near Real-Time DLP Scan and Malware Scan for AWS
https://aws.amazon.com/blogs/architecture/how-usaa-built-an-amazon-s3-malware-scanning-solution/ ,
https://success.myshn.net/Skyhigh_Cloud_Infrastructure_(CNAPP)/CSPM/CSPM_Near_Real-Time_DLP_Scans/Near_Real-Time_DLP_Scan_and_Malware_Scan_for_AWS

[33] Amazon Web Services (2023) Running a Serverless Malware Scanner at Scale Using AWS Lambda and AWS Step Functions.
https://aws.amazon.com/blogs/apn/running-a-serverless-malware-scanner-at-scale-using-aws-lambda-and-aws-step-functions/

[34] Alshuqayran, N., & Bahsoon, R. (2018). Infrastructure as code: A systematic mapping study. In 2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR) (pp. 446-450). IEEE. https://doi.org/10.1145/3196398.3196432

[35] Amazon Web Services (2023) Cloud Formation. https://aws.amazon.com/cloudformation/

[36] Amazon Web Services (2023) CDK. https://aws.amazon.com/cdk/

[37] Hashicorp (2023) Terraform Introduction.https://developer.hashicorp.com/terraform/intro

[38] Terragrunt (2023) Terragrunt doc. https://terragrunt.gruntwork.io/docs/#reference

[39] Geekflare (2023) Understanding IaC Tools: Cloudformation vs Terraform. https://geekflare.com/cloudformation-vs-terraform/

[40] Geekflare (2023) Descripción de las herramientas de IaC: AWS CDK frente a Terraform https://geekflare.com/es/aws-cdk-vs-terraform/

[41] Medium.com (2023) Terraform — Workspaces Overview https://medium.com/devops-mojo/terraform-workspaces-overview-what-is-terraform-workspace-introduction-getting-started-519848392724

[42] D. Jackson and G. Clynch, "An Investigation of the Impact of Language Runtime on the Performance and Cost of Serverless Functions," 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion), Zurich, Switzerland, 2018, pp. 154-160, doi: 10.1109/UCC-Companion.2018.00050.

[43] J. Dantas, H. Khazaei and M. Litoiu, "Application Deployment Strategies for Reducing the Cold Start Delay of AWS Lambda," 2022 IEEE 15th International Conference on Cloud Computing (CLOUD), Barcelona, Spain, 2022, pp. 1-10, doi: 10.1109/CLOUD55607.2022.00016.

[44] Amazon Web Services (2023) AWS Lambda/python. https://gallery.ecr.aws/lambda/python

[45] Amazon Web Services (2023) AWS Lambda/go. https://gallery.ecr.aws/lambda/go

[46] Amazon Web Services (2023) AWS Lambda/nodejs. https://gallery.ecr.aws/lambda/nodejs

[47] European Institute for Computer Anti-Virus Research (EICAR) (2023) ANTI MALWARE TESTFILE. https://www.eicar.org/download-anti-malware-testfile/

[48] Amazon Web Services (2023) Calculator. https://calculator.aws/#/estimate?id=e96ae66cf9393d3d36c29a15ecc1730433c6b1df

[49] Wikipedia (2023) Principio de mínimo privilegio. https://es.wikipedia.org/wiki/Principio_de_m%C3%ADnimo_privilegio

Glosario

CI/CD (Continuous Integration/Continuous Delivery): Automatización del proceso de integración y entrega de código para una entrega rápida y segura.

CRON: Sistema de programación de tareas en sistemas Unix para ejecutar comandos o scripts en momentos específicos.

Docker: Plataforma de virtualización ligera que empaqueta aplicaciones y sus dependencias en contenedores portátiles.

DRY (Don't Repeat Yourself): Principio de desarrollo de software que promueve la reutilización y evita la duplicación de código.

EICAR (European Institute for Computer Anti-Virus Research): Instituto de investigación de antivirus que también ofrece un archivo de prueba para evaluar programas antivirus.

Endpoint: Dispositivo final o nodo en una red que puede enviar o recibir datos.

GUI (Graphical User Interface): Interfaz de usuario gráfica que permite la interacción mediante elementos visuales.

Malware: Software malicioso diseñado para dañar o infiltrarse en sistemas informáticos.

Mirror: Réplica exacta de un repositorio o sitio web para descargas distribuidas y eficientes.

Registry: Repositorio centralizado para almacenar y gestionar metadatos o configuraciones importantes en un sistema.

Roadmap: Representación visual y planificada de objetivos y fechas clave de un proyecto o producto.

Rollback: Revertir una acción o cambio en un sistema a un estado anterior o punto de restauración.

SDK (Software Development Kit): Conjunto de herramientas para desarrolladores que facilita la creación de aplicaciones para una plataforma específica.

Security Champion: Defensor y promotor de prácticas de seguridad dentro de una organización.

Stack: Conjunto de tecnologías o software utilizados en conjunto para crear una aplicación o sistema.

Trigger: Evento o condición que inicia una acción automatizada en un sistema o software.

Wrapper: Capa de software que envuelve o encapsula una funcionalidad existente para facilitar su uso o integración.

Mínimo privilegio: Principio que establece que los usuarios y entidades deben tener solo los privilegios necesarios para realizar sus tareas específicas, minimizando así el acceso a recursos y funcionalidades adicionales.