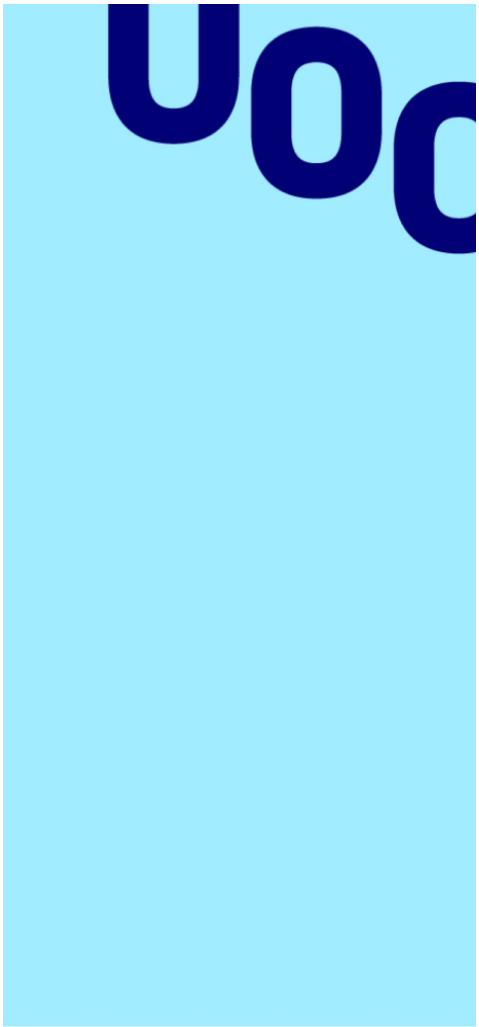
Diseño de arquitectura basada en microservicios en AWS para un sistema de CTI (Cyber Threat Intelligence)



Javier Sánchez García-Ochoa

Seguridad empresarial

Máster Universitario en Ciberseguridad y Privacidad

Nombre del director/a de TF:
Isaac Guasch García

Nombre del/de la PRA:
Víctor García Font

11 de junio de 2024

Universitat Oberta de Catalunya

Ficha Del Trabajo Final

Título del trabajo:	Diseño de arquitectura basada en microservicios en AWS para un sistema de CTI (Cyber Threat Intelligence)
Nombre del autor/a:	Javier Sánchez García-Ochoa
Nombre del director/a de TF:	Isaac Guasch García
Nombre del/de la PRA:	Víctor García Font
Fecha de entrega:	11 de junio de 2024
Titulación o programa:	Máster Universitario en Ciberseguridad y Privacidad
Área del trabajo final:	Seguridad empresarial
Idioma del trabajo:	Castellano
Palabras clave:	Amazon Web Services, Microservicios, Cyber Threat Intelligence

Resumen del trabajo

Este proyecto aborda el diseño, desarrollo y despliegue de una plataforma de ciberinteligencia de amenazas utilizando microservicios y el paradigma *serverless* en AWS, en respuesta a la oportunidad y necesidad de desarrollar soluciones modulares y escalables. La finalidad es proporcionar a los equipos de ciberinteligencia una herramienta que permita visualizar datos de manera clara y sencilla, facilitando decisiones informadas y rápidas.

El contexto se centra en la adopción de la nube pública, que ha transformado las operaciones tecnológicas de empresas de todos los tamaños, ofreciendo escalabilidad, flexibilidad y eficiencia. La metodología empleada incluye el uso de infraestructura como código (IaC) para asegurar una configuración reproducible y auditable, junto a la implementación de servicios en la nube para optimizar el rendimiento, los recursos energéticos y computacionales, y también los costes.

Los resultados destacan el éxito en la creación de una herramienta funcional, modular y escalable. A pesar de las limitaciones presupuestarias, se logra equilibrar la seguridad y los recursos disponibles, proporcionando una solución que ayuda a los ciberanalistas a visualizar indicadores y datos para tomar decisiones más precisas.

Las conclusiones resumen cómo se cumplen los objetivos del proyecto, demostrando la eficacia de la combinación de microservicios y el paradigma *serverless* en AWS. Como trabajo futuro, se sugiere ampliar la capacidad de recolección de datos, considerar un mayor presupuesto para mejorar la seguridad y adoptar tecnologías más avanzadas, y desarrollar un módulo de predicción basado en aprendizaje automático.

Abstract

This project addresses the design, development, and deployment of a cyber threat intelligence platform using microservices and the serverless paradigm on AWS, in response to the opportunity and need to develop modular and scalable solutions. The aim is to provide cyber intelligence teams with a tool that allows for clear and simple data visualization, enabling informed and rapid decision-making.

The context focuses on the adoption of public cloud, which has transformed the technological operations of companies of all sizes by offering scalability, flexibility, and efficiency. The methodology employed includes the use of Infrastructure as Code (IaC) to ensure a reproducible and auditable configuration, along with the implementation of cloud services to optimize performance, energy and computational resources, and costs.

The results highlight the success in creating a functional, modular, and scalable tool. Despite budgetary constraints, the project balances security and available resources, providing a solution that helps cyberintelligence analysts visualize indicators and data to make more accurate decisions.

The conclusions summarize how the project's objectives are met, demonstrating the effectiveness of combining microservices and the serverless paradigm on AWS. As future work, it is suggested to expand data collection capabilities, consider a larger budget to enhance security and adopt more advanced technologies, and develop a prediction module based on machine learning.

Índice general

Índice de acrónimos

AKS Azure Kubernetes Service

Amazon EBS Amazon Elastic Block Storage

Amazon EC2 Amazon Elastic Compute Cloud

Amazon ECS Amazon Elastic Container Service

Amazon EFS Amazon Elastic File System

Amazon EKS Amazon Elastic Kubernetes Service

Amazon Managed Grafana Amazon Managed Grafana

Amazon RDS Amazon Relational Database Service

Amazon S3 Amazon Simple Storage Service

Amazon VPC Amazon Virtual Private Cloud

AMI Imagen de máquina de Amazon (*Amazon Machine Image*)

API Interfaz de programación de aplicaciones (*Application Programming Interface*)

Application Composer AWS Application Composer

AWS Amazon Web Services

AWS CLI AWS Command Line Interface

AWS Pricing Calculator AWS Pricing Calculator

AWS SAM AWS Serverless Application Model

AWS SAM CLI AWS Serverless Application Model Command Line Interface

CaaS Contenedores como servicio (*Container-as-a-Service*)

CCEG Competencia de Compromiso Ético y Global

CERT Equipo de respuesta ante incidentes (*Computer Emergency Response Team*)

CI/CD Integración y entrega continuas (*Continuous Integration and Continuous Delivery*)

CIS Center for Internet Security

CLI Interfaz de línea de comandos (*Command Line Interface*)

CloudFormation AWS CloudFormation

CloudWatch Amazon CloudWatch

CPD Centro de Procesamiento de Datos

CPU Unidad central de procesamiento (*Central Processing Unit*)

CSP Proveedor de servicios *Cloud* (*Cloud Service Provider*)

CTI Ciberinteligencia de amenazas (*Cyber Threat Intelligence*)

DFIR Análisis forense digital y respuesta a incidentes (*Digital Forensics and Incident Response*)

DynamoDB Amazon DynamoDB

EDA Arquitectura basada en eventos (*Event-Driven Architecture*)

EventBridge Amazon EventBridge

FaaS Funciones como servicio (*Function-as-a-Service*)

GB Gigabyte

GCP Google Cloud Platform

GiB Gibibyte

HCL *HashiCorp Configuration Language*

HTTPS *Hypertext Transfer Protocol Secure*

IA Inteligencia artificial

IaC Infraestructura como código (*Infrastructure-as-Code*)

IAM Gestión de identidades y accesos (*Identity Access Management*)

IoC Indicador de compromiso (*Indicator of Compromise*)

IOPS Operaciones de entrada/salida por segundo (*Input/Output Operations Per Second*)

IP Dirección de protocolo de internet (*Internet Protocol*)

ISAC Centro de compartición y análisis de información (*Information Sharing and Analysis Center*)

JSON *JavaScript Object Notation*

Lambda AWS Lambda

MB Megabyte

MISP *Malware Information Sharing Platform*

ML Aprendizaje automático (*Machine Learning*)

MS-ISAC *Multi-State Information Sharing and Analysis Center*

MSA Arquitectura de microservicios (*Microservice Architecture*)

NAT Traducción de direcciones de red (*Network Address Translation*)

PoC Prueba de concepto (*Proof of Concept*)

RSS *Really Simple Syndication*

SOA Arquitectura orientada a servicios (*Service-Oriented Architecture*)

SOC Centro de operaciones de seguridad (*Security Operations Centre*)

SQL *Structured Query Language*

SSH *Secure Shell*

STIX *Structured Threat Information eXpression*

Systems Manager AWS Systems Manager

TAXII *Trusted Automated eXchange of Intelligence Information*

TIP Plataforma de inteligencia de amenazas (*Threat Intelligence Platform*)

TTPs Tácticas, técnicas y procedimientos

URL *Uniform Resource Locator*

XML Lenguaje de marcado extensible (*Extensible Markup Language*)

YAML *Yet Another Markup Language*

Índice de figuras

Índice de códigos

Índice de tablas

Capítulo 1

Introducción

1.1. Contexto del Trabajo Fin de Máster

En la última década, el panorama tecnológico ha experimentado una transformación radical, con la migración de cargas de trabajo a la nube pública, entre otras tendencias. Este cambio no solo ha estado impulsado por grandes corporaciones en busca de flexibilidad y eficiencia, sino que también ha brindado oportunidades sin precedentes para pequeñas empresas y *startups* que desean competir en un mercado globalizado y altamente dinámico [1]. La capacidad de escalar rápidamente, pagar solo por los recursos utilizados, acceder a infraestructuras desde cualquier lugar del mundo y el incremento de personal técnico cualificado son ventajas que impulsan la adopción de este modelo de computación.

Este cambio también ha impulsado el surgimiento de arquitecturas de microservicios (*Microservice Architectures*, MSA), que permiten a las organizaciones desarrollar y desplegar aplicaciones de manera más ágil, escalable y resiliente gracias a su modularidad y la combinación de la flexibilidad inherente a la nube.

En concreto, este trabajo estudia el desarrollo y despliegue de una plataforma de ciberinteligencia de amenazas propia, haciendo uso de microservicios en la nube. Para ello, se propondrá una posible arquitectura haciendo uso de la mayor cantidad de servicios nativos posibles de Amazon Web Services (AWS), la nube pública de Amazon, que es el proveedor líder en la industria [2]. Además, se emplearán herramientas de infraestructura como código (*Infrastructure-as-Code*, IaC), que permiten acortar el ciclo de vida del desarrollo de aplicaciones y proporcionar una entrega continua, gestionando los entornos de manera más eficiente y automatizada.

1.2. Objetivos

1.2.1. Objetivos generales

El principal objetivo de este trabajo es profundizar en las MSA, analizando sus ventajas y desafíos en relación a la herramienta a desarrollar, para así poder aprovechar todas sus capacidades, así como las de la computación en la nube. Haciendo uso de todas ellas se ha de diseñar y desarrollar un sistema de ciberinteligencia de amenazas (*Cyber Threat Intelligence*, CTI) útil y modular, adaptado a los paradigmas de la computación en la nube.

Para validar su funcionamiento, se analizará estáticamente el desarrollo e implementará tanto en una herramienta de desarrollo, LocalStack, como en la propia infraestructura de AWS, validando que el diseño es implementado íntegramente e integrado de principio a fin. Además, de manera transversal a todo el proceso, se tendrá en cuenta el principio de seguridad desde el diseño.

1.2.2. Objetivos específicos

En primer lugar, se tratará de adquirir un conocimiento previo sobre los sistemas distribuidos, así como las mejores prácticas en la implementación, gestión y escalabilidad de los mismos. De igual manera, se requiere comprender los servicios y modelos de despliegue propios de AWS enfocados en el desarrollo de MSA, así como explorar diferentes alternativas, como la computación sin servidor, y analizar sus diferencias.

Una vez conseguidas estas competencias, se continuará con el diseño de la arquitectura sobre la que se desplegará la herramienta. Para ello, será necesario definir y analizar paralelamente los requisitos de la plataforma de ciberinteligencia, las fuentes de datos que utilizará, cómo se mostrarán los datos e indicadores, etc. Con ello, y teniendo en cuenta los objetivos generales mencionados acerca de los servicios distribuidos y en la nube, se estudiará la elección de los diferentes productos, así como las herramientas de IaC, dando lugar a la arquitectura final.

1.2.3. Alcance

Forma parte de este proyecto definir e implementar una arquitectura MSA haciendo uso de servicios tanto nativos como de terceros en la nube pública de AWS. Además, también se contempla el despliegue en un laboratorio de desarrollo con LocalStack, una herramienta que simula los servicios de AWS y que puede ser ejecutada en local, evitando generar los costes que implica un despliegue en la arquitectura de Amazon. Una vez se compruebe la viabilidad de la herramienta en el entorno de desarrollo, se desplegará la arquitectura en la infraestructura de AWS para validar el trabajo en el entorno real.

En cuanto al producto, se pretende elaborar una prueba de concepto (*Proof of Concept*, PoC) de una plataforma de ciberinteligencia aprovechando los microservicios y ventajas de la nube pública en el desarrollo. No forma parte del proyecto elaborar una herramienta de ciberinteligencia completa y comercializable, ni pretende ser ninguna alternativa a los productos consolidados analizados en la Sección 2.3.1. Plataforma de inteligencia de amenazas (*Threat Intelligence Platform*, TIP).

Queda excluido del alcance de este proyecto detallar los procesos de configuración de los entornos de pruebas, tanto del entorno de desarrollo de LocalStack como del entorno de pre-producción en AWS. No obstante, su puesta en marcha son tareas que se deben desarrollar para poder completar las pruebas requeridas de la nueva solución. Por ello, se propondrán referencias bibliográficas útiles para poder replicar dichos procesos con éxito.

También queda fuera del alcance la configuración de la herramienta de visualización en LocalStack, así como otras herramientas cuyo funcionamiento y puesta en marcha sea similar en LocalStack y AWS. Estos procesos únicamente se abordarán en el despliegue en el entorno de pre-producción. Esto se debe a que LocalStack simula las interfaz de programación de aplicaciones (*Application Programming Interface*, API) de los servicios, pero no los replica en su

totalidad, por lo que su uso no puede ser completo como el que proporciona AWS.

1.3. Impacto en sostenibilidad, ético-social y de diversidad

Atendiendo a la guía transversal sobre la competencia ética y global, la Competencia de Compromiso Ético y Global (CCEG) se define a nivel de máster como «Actuar de manera honesta, ética, sostenible, socialmente responsable y respetuosa con los derechos humanos y la diversidad, tanto en la práctica académica como en la profesional, y diseñar soluciones para mejorar estas prácticas.» [3].

Siguiendo el citado documento, se han determinado impactos en las tres dimensiones: sostenibilidad, comportamiento ético y de responsabilidad social, y diversidad, género y derechos humanos.

1.3.1. Sostenibilidad

El uso de microservicios e infraestructura de nube pública contribuye al uso limitado y optimizado de recursos energéticos y computacionales. Esto se debe, principalmente, a una gran ventaja de la virtualización: los proveedores de servicios *Cloud* (*Cloud Service Providers*, CSPs) pueden repartir sus recursos físicos entre diferentes clientes, que emplean únicamente los recursos virtuales necesarios, pudiendo maximizar el uso de los equipos.

No obstante, los centros de datos de nube pública requieren de multitud de recursos esenciales para su funcionamiento, como el agua para refrigeración, energía eléctrica para alimentar todos los servidores, así como logística e infraestructura necesaria para que los sistemas estén en condiciones óptimas. Además, se generan multitud de residuos, por ejemplo, por el desgaste de elementos como discos duros.

1.3.2. Comportamiento ético y de responsabilidad social

El principal impacto social es el conocimiento del contexto que proporciona la herramienta de ciberinteligencia que se va a desarrollar. Como multitud de *software*, puede ser empleado tanto con fines positivos (por ejemplo, cuando una empresa busca mejorar su postura de seguridad y protegerse según la información obtenida) como negativos (cuando ciberdelincuentes aprovechan la información compartida para diseñar sus ataques).

Asimismo, la herramienta no pone en riesgo ningún puesto laboral, pues se pretende aportar recursos a los analistas, cuyo conocimiento experto es imprescindible para interpretar la información aglutinada y mostrada por la herramienta, sin el cual la misma carecería de utilidad.

1.3.3. Diversidad, género y derechos humanos

El uso de herramientas de inteligencia de amenazas busca preservar, entre otros, derechos básicos de los usuarios en Internet, garantizando los tres pilares básicos de la ciberseguridad: confidencialidad, integridad y disponibilidad. Asimismo, esto aplica también a la propiedad intelectual de las empresas, pues al emplearse estas herramientas se busca mejorar la postura

de seguridad de entidades en la red, mejorando su propia seguridad, así como la de sus usuarios, empleados, miembros, etc. No obstante, es imprescindible tener precaución y realizar un análisis detallado de las fuentes de información utilizadas, así como del procesamiento de los datos que se realice, pues se pueden introducir sesgos que afecten de manera desigual a las personas, creando posibles desigualdades.

1.4. Enfoque y método seguido

Este proyecto de Fin de Máster, como ya se ha introducido, busca desarrollar y desplegar una herramienta de ciberinteligencia, haciendo uno de microservicios en la nube. El producto, de nueva creación, no implica desarrollar funcionalidades nuevas o modificar las existentes de otros ejemplos de código abierto, sino centrar los esfuerzos en comprender cómo funcionan estos, qué y cómo se pueden replicar sus funcionalidades utilizando microservicios y comprender todo el proceso desde la extracción, transformación y visualización de los datos.

El primer objetivo, el estudio de las MSA y sistemas distribuidos, sirve como introducción al campo y permite comprender y analizar los conceptos que caracterizan a estas infraestructuras, así como sus diferencias respecto a aplicaciones monolíticas tradicionales. Conocer el estado del arte permite comprender los aspectos más importantes que se deben valorar para el diseño y despliegue de la nueva herramienta e infraestructura. Además, una vez conocida la tecnología y su funcionamiento, se puede entender cómo funciona su despliegue en la nube, las capacidades que brinda, así como posibles desventajas y factores nuevos que sea preciso valorar.

En el diseño de la arquitectura, se tomarán conceptos de las MSA y las arquitecturas basadas en eventos (*Event-Driven Architectures*, EDA). Además, se empleará un modelo de desarrollo propio de la nube, el modelo de computación sin servidor. Para ello, se utilizará AWS SAM, que es la concreción de dicho modelo para el entorno y servicios de AWS.

Por otro lado, se desarrollará toda la infraestructura como IaC haciendo uso de AWS CloudFormation para asegurar la consistencia, replicabilidad y buenas prácticas de configuración de los servicios empleados.

1.5. Planificación del trabajo

El trabajo a desempeñar se puede categorizar en tres fases interdependientes, aunque no necesariamente debe haberse completado una para poder comenzar con la siguiente. La Figura 1.1 muestra un detalle de las tareas concretas a realizar a lo largo del proyecto.

La primera consiste en una revisión bibliográfica del estado actual de las MSA, las diferentes tecnologías empleadas y cómo ayudan a desarrollar aplicaciones más desacopladas y escalables. Ello implica el estudio de las MSA en sí, de sus posibilidades en combinación con infraestructuras de nube pública. También es necesario realizar un análisis del fundamento de las herramientas de ciberinteligencia, así como las plataformas actuales y equiparables a la propuesta que se pretende desarrollar. Esta etapa es esencial para ahondar en los fundamentos de la herramienta que se propone desarrollar y poder identificar las oportunidades de mejora de las soluciones actuales, pudiendo aportar valor nuevo dentro del campo.

La segunda consiste en el diseño de la solución a alto nivel, de manera agnóstica a tecnologías, aunque teniendo en mente que se trata de un despliegue en nube pública. Para ello, se

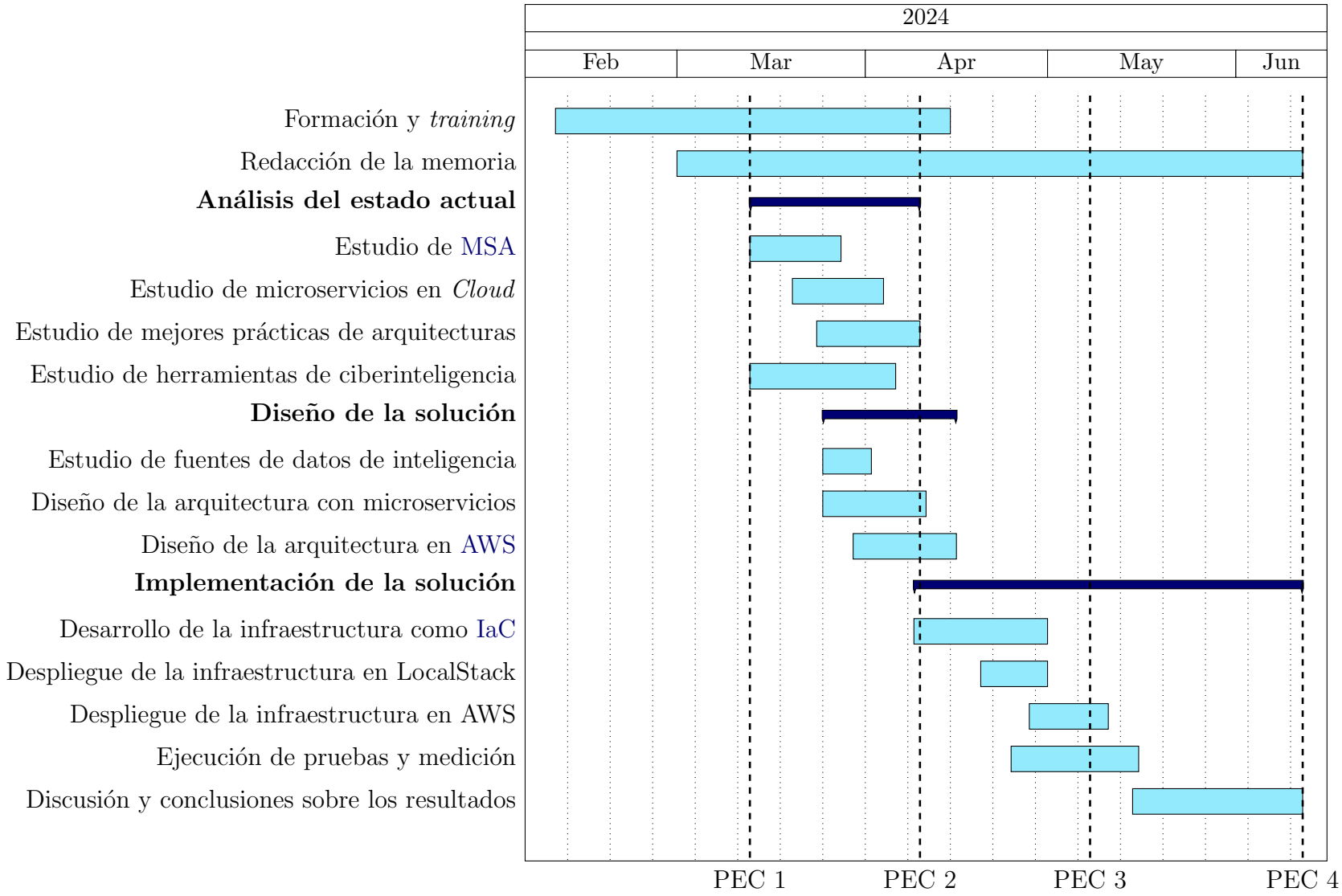


Figura 1.1: Planificación del trabajo

han de analizar posibles fuentes públicas de datos de ciberinteligencia que aporten valor a la herramienta. También se deben revisar las tecnologías de visualización de los mismos, así como idear la manera en que se orquestará la obtención de datos.

La tercera y última consiste en la implementación de dicha propuesta en la nube pública de AWS. Para ello se debe en primer lugar desarrollar la infraestructura como IaC y probar el despliegue en un entorno de desarrollo como es LocalStack. Seguidamente, se puede validar el despliegue en AWS, para finalizar con un apartado de medición y validación que proporcione información para identificar tanto las fortalezas como las posibles deficiencias de la solución.

Hay dos tareas, reflejadas también en la Figura 1.1, que son transversales al resto de fases: (I) la formación previa sobre los entornos y herramientas, necesaria para poder abordar el proyecto con solvencia, y predominante en las dos primeras fases del proyecto, (II) la redacción de la memoria, proyectada de manera paralela a la ejecución de cada tarea.

1.6. Estructura del documento

El resto de este documento sigue una estructura en capítulos que desarrollan las tres etapas de trabajo mencionadas anteriormente: la revisión del estado del arte, la fase de diseño de la misma, y su implementación y validación.

El capítulo dos incluye una introducción a las arquitecturas de microservicios, las tecnologías que los hacen posible, así como su naturaleza y capacidades en infraestructuras de nube pública. Asimismo, introduce la ciberinteligencia de amenazas y las arquitecturas empleadas en los despliegues en la nube por parte de las herramientas de código abierto más conocidas actualmente.

El capítulo tres comprende el diseño de la arquitectura, centrado en la elección de la fuente de datos de ciberinteligencia, así como los requisitos del resto de elementos esenciales para que la nueva herramienta sea completa y totalmente funcional.

En el capítulo cuatro se avanza con el diseño de la arquitectura, escogiendo tecnologías y servicios concretos, dados los requisitos y posibilidades de los mismos, así como sus costes. Además, se implementa toda la arquitectura como IaC, y se despliega tanto en un entorno de desarrollo como en la propia infraestructura de AWS. Durante todo el despliegue se valida el éxito del mismo, empleando asimismo otros métodos como herramientas de análisis estático del código.

En el capítulo cinco se realiza una valoración económica del laboratorio, analizando la capa gratuita de AWS y realizando una estimación de costes.

Por último, en el capítulo seis se exponen las conclusiones técnicas y personales del proyecto, analizando la consecución de los objetivos y las líneas de trabajo futuro derivadas.

Capítulo 2

Estado del arte

A lo largo de este capítulo se analizan las características más importantes de las MSA, su relación con las infraestructuras de nube pública, y las características de seguridad relacionadas. Además, se realiza un estudio del estado del arte actual de plataformas de ciberinteligencia, su funcionamiento y utilidad dentro del panorama de amenazas actual.

2.1. Arquitectura de microservicios (*Microservice Architecture*, MSA)

El diseño de aplicaciones ha ido cambiando a lo largo del tiempo, desde las arquitecturas monolíticas cliente-servidor, hacia las arquitecturas orientada a servicios (*Service-Oriented Architectures*, SOA), un enfoque en el que los componentes individuales del sistema son diseñados para ser servicios reutilizables y conectables, y son accedidos y utilizados de manera independiente a través de una red [4]. Derivadas de las SOA surgen las MSA, un nuevo paradigma de diseño de arquitecturas propio de la computación en la nube. Estas organizan conjuntos de componentes pequeños y granulares que pueden ser implementados de manera independiente en diferentes plataformas y tecnologías. Cada uno de ellos ejecuta sus propios procesos y se comunica con el resto por medio de APIs [5].

En arquitecturas monolíticas, los procesos están estrechamente relacionados, teniendo una única aplicación, aunque pueda estar compuesta de diversos módulos. Sin embargo, estos módulos no pueden ejecutarse de manera independiente. Además, al tratarse de un conjunto único, cualquier pico de demanda de un proceso requiere escalar toda la arquitectura. Por otro lado, conforme crece la aplicación y se añaden funcionalidades, el desarrollo y mantenimiento se vuelve insostenible. Todo ello hace que el impacto de cualquier error afecte a toda la aplicación, al ser procesos dependientes y vinculados estrechamente, y limita la flexibilidad para realizar cambios al tener que desplegar toda la aplicación en su conjunto [6, 7].

Por otro lado, como ya se ha introducido previamente, los microservicios son elementos que pueden ejecutarse de manera independiente y que desempeñan una funcionalidad concreta, sin tener en cuenta los requisitos de otras capas en su desarrollo, por lo que no dependen estrechamente entre sí. Para comunicarse, se utilizan APIs bien definidas que exponen los métodos apropiados. Al ejecutarse de manera independiente, se pueden desplegar, actualizar y escalar según las necesidades y la demanda sin interferir en el resto de componentes [7]. Esto

ha causado que su uso sea mayor en los últimos años, dada su facilidad de desarrollo, escalado y facilidad de despliegue en entornos *cloud*, aprovechando sus ventajas como la escalabilidad y el pago por uso [6]. La Figura 2.1 enfrenta un ejemplo de arquitectura monolítica con una de microservicios, mostrando el acoplamiento y relación entre las capas de la monolítica con los microservicios independientes.

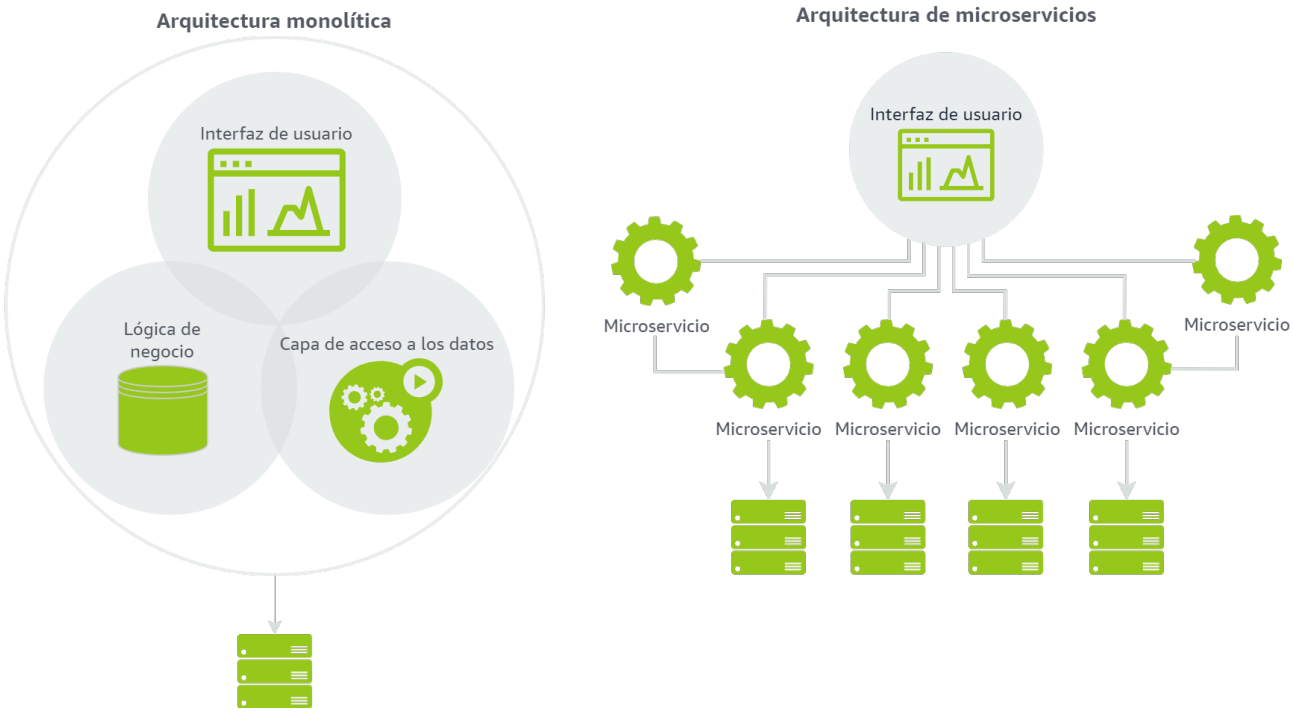


Figura 2.1: Arquitectura monolítica frente a microservicios (basado en [8])

La creciente popularidad de los microservicios está relacionada con el auge de la filosofía DevOps, ya que les permiten desarrollar productos con mayor velocidad y entregas rápidas, haciendo uso de metodologías ágiles de desarrollo de *software*. Son habituales en estos contextos las prácticas de integración y entrega continuas (*Continuous Integration and Continuous Delivery*, CI/CD) para automatizar los procesos de desarrollo e integración, reduciendo los tiempos de entrega, así como añadir herramientas de pruebas y controles de seguridad, etc. [9].

2.1.1. Virtualización basada en contenedores

Todas estas características de las arquitecturas de microservicios no serían posibles sin las tecnologías de virtualización. Estas permiten la abstracción de los recursos físicos en objetos lógicos desligados del *hardware*, logrando una mayor modularidad, flexibilidad y eficiencia en el uso de los recursos físicos.

Uno de los diversos tipos de virtualización existentes es la basada en contenedores, mediante la cual se empaquetan las aplicaciones y todo lo que necesitan para ejecutarse en una unidad estándar que puede ser posteriormente desplegada con facilidad en multitud de entornos [10]. Al encapsular todas las dependencias precisadas y sus sistemas de archivos, no están supeditados de ningún modo a los componentes instalados en el sistema host, permitiendo que los diferentes

contenedores desplegados en un sistema compartan el mismo sistema operativo y kernel. Por ello, la eficiencia es mayor que con otras tecnologías de virtualización en las que se replica un sistema operativo para cada entorno virtual, denominada en este caso máquina virtual. La Figura 2.2 muestra la evolución en el tiempo desde los despliegues tradicionales, pasando por la virtualización basada en máquinas virtuales, hasta la virtualización basada en contenedores, mostrando los diferentes elementos necesarios y disponibles en cada caso.

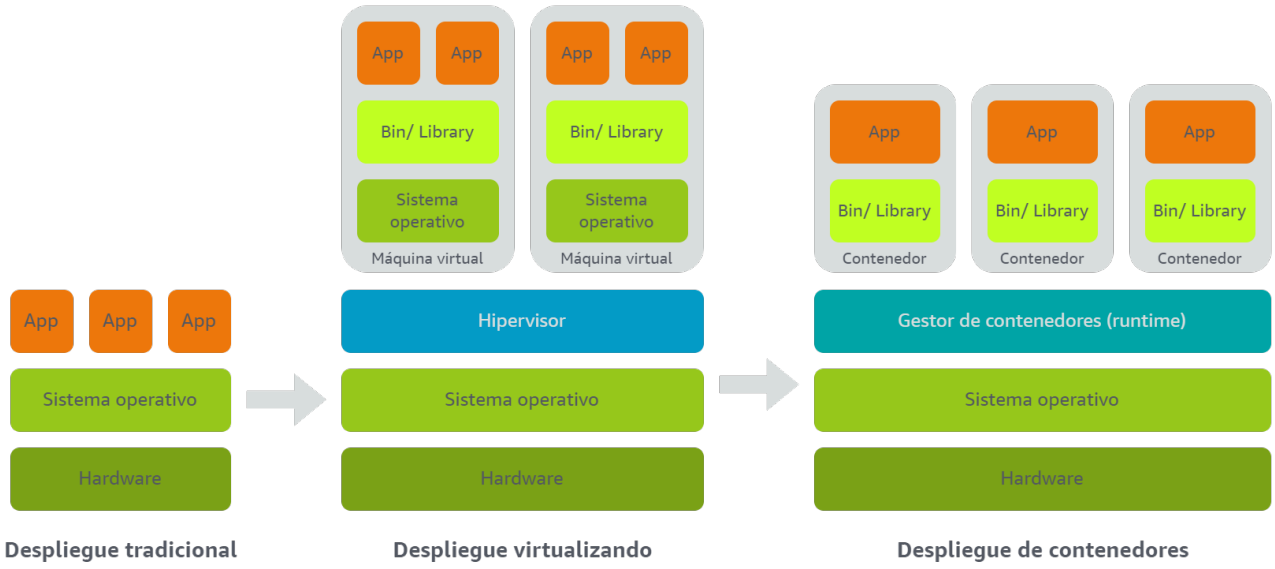


Figura 2.2: Evolución de los despliegues con virtualización y contenedores (basado en [11])

Hay diversas tecnologías que hacen posible este paradigma. Una de ellas es Docker, una plataforma de código abierto que permite empaquetar y ejecutar las aplicaciones en contenedores de manera aislada. Además, proporciona herramientas para gestionar todo el ciclo de vida de los mismos, incluyendo el desarrollo de la aplicación y de los componentes y dependencias que necesita, la distribución y pruebas de la aplicación, y su despliegue en entornos de producción tanto en entornos locales como en proveedores *cloud* o entornos híbridos, bien como contenedores o como servicios orquestados [12]. Hay otras tecnologías alternativas a Docker como Podman, RKT u OpenVz.

Sin embargo, la gestión de multitud de contenedores puede ser complicada haciendo uso de Docker exclusivamente. En este punto, aparece Kubernetes, otra plataforma de código abierto enfocada en la gestión de cargas de trabajo de contenedores. Con ella, se pueden escalar los servicios creando diferentes nodos de un clúster y balancear la carga entre ellos, para asegurar la disponibilidad de la aplicación. Es capaz de gestionar los recursos físicos disponibles según se le indique, y además se encarga de responder ante errores de los contenedores, reiniciándolos si fallan, reemplazándolos o destruyéndolos si no son capaces de responder ante las comprobaciones que se definan. La disponibilidad también está asegurada en procesos de actualización, ya que Kubernetes se encarga de no reemplazar el contenedor antiguo hasta que el nuevo está totalmente disponible [11]. Existen además otras tecnologías de orquestadores como Docker Swarm, Service Fabric o DC/OS.

2.2. Microservicios en la nube pública

En los despliegues de nube pública, los CSPs proveen a sus clientes con multitud de servicios para el despliegue de MSA. Principalmente, se diferencian dos enfoques posibles:

1. Utilizar un orquestador de contenedores que se ejecutan en nodos dedicados, como Kubernetes.

2. Utilizar un nuevo paradigma conocido como computación sin servidor.

Para desarrollar aplicaciones utilizando el primer modelo, existen servicios gestionados que podríamos denominar como contenedores como servicio (*Container-as-a-Service*, CaaS), siguiendo la nomenclatura habitual de los modelos de gestión de la nube. En ellos, los CSPs despliegan la infraestructura y la plataforma, incluyendo el orquestador y los recursos computacionales de los nodos, según las indicaciones que dé el usuario, encargado posteriormente de la gestión de las aplicaciones y *software*. Ejemplos de estos servicios son Amazon Elastic Kubernetes Service (Amazon EKS) o Azure Kubernetes Service (AKS).

El paradigma de computación sin servidor es una evolución más, posible gracias a la distribución de responsabilidades entre CSP y cliente. El proveedor gestiona todo el trabajo operacional de la infraestructura y la pone a disposición para que los desarrolladores únicamente tengan que escribir el código de las aplicaciones. De esta manera, los clientes evitan los costes y tareas de aprovisionamiento y mantenimiento, y solo tienen que hacer uso de la infraestructura necesaria para ejecutar su flujo de trabajo [13]. Es un paradigma diferente a los servicios orquestados de contenedores, ya que la computación sin servidor funciona a demanda basada en eventos y sin una infraestructura subyacente disponible.

Para ello, el cómputo se hace mediante funciones como servicio (*Function-as-a-Service*, FaaS), servicios que ejecutan fragmentos de código funcionales que no necesitan una infraestructura completa. Por supuesto, también están involucrados otros servicios, como los encargados de gestionar notificaciones y mensajes entre servicios, además del almacenamiento, las APIs usadas para la comunicación, etc. Sin embargo, se pierde control sobre la infraestructura subyacente a todo el flujo, aunque esta característica es también una ventaja, pues la complejidad es mucho menor que en la gestión de un clúster de Kubernetes. Ambos enfoques tienen características diferentes que los hacen óptimos para casos de usos distintos.

Dado el auge que está teniendo este modelo, están surgiendo también herramientas de FaaS de código abierto autogestionadas [14]. Estas permiten disponer de entornos de computación sin servidor en entornos locales o de nube privada, y al no depender de las tecnologías de los CSPs, evitan el *vendor lock-in* y facilitan también la portabilidad, así como el control sobre la infraestructura.

2.3. Ciberinteligencia de amenazas

Tal y como indica su nombre, la ciberinteligencia de amenazas o inteligencia de ciberamenazas hace referencia a la aplicación de inteligencia a todas aquellas amenazas que atañen al ciberespacio. La acuñación del término se remonta al siglo V, relacionado casi siempre con el ámbito militar. El «Libro de los Números» de la Biblia católica y el tratado «El arte de la

guerra» del general chino Sun Tzu son los primeros textos que lo mencionan, ambos atribuidos a dicho siglo [15, 16].

Según [17] se define la inteligencia de amenazas en general como sigue: «la inteligencia de amenazas es el conocimiento basado en la evidencia, incluyendo el contexto, mecanismos, indicadores, implicaciones y consejos prácticos, sobre una amenaza o peligro existente o emergente para los activos que puede utilizarse para fundamentar las decisiones sobre la respuesta del sujeto a dicha amenaza o peligro.». Recientemente, el mismo término se ha aplicado a las ciberamenazas, por lo que es preciso combinar esta definición de inteligencia tradicional con el dominio del ciberespacio.

El objetivo de este proceso de recopilación de información y generación de conocimiento es dar soporte al proceso de análisis y gestión del riesgo de una organización, ya que es esencial conocer y comprender las amenazas para poder gestionarlas adecuadamente. Por tanto, la utilidad de esta inteligencia radica en su aplicación para poder evitar impedimentos a la hora de lograr los objetivos de una organización. Vivimos en un panorama complejo y dinámico, con adversarios cada vez más preparados y ambiciosos, y con posturas de seguridad de las organizaciones que cambian día a día (nuevas vulnerabilidades descubiertas, nuevos controles de seguridad añadidos, nuevos parches disponibles, etc.). Es vital para las organizaciones comprender todos estos cambios y conocer las nuevas tácticas, técnicas y procedimientos (TTPs) que usan los adversarios para poder defenderse adecuadamente, destinar los recursos apropiados, y en definitiva, tomar decisiones informadas de la mejor manera posible [16].

Los tipos de información que la inteligencia de amenazas puede proporcionar se suelen categorizar en tres o cuatro opciones [16, 18, 19]:

1. Estratégica: describe las tendencias a alto nivel y los objetivos de los adversarios a largo plazo, afectando a la estrategia de seguridad y la toma de decisiones de negocio a largo plazo. Sirve a la alta dirección y al consejo ejecutivo.

2. Operacional: describe los cambios a corto y medio plazo en el panorama de amenazas y las TTPs utilizadas actualmente para decidir y priorizar las operaciones de seguridad. Es útil para los analistas y expertos del centro de operaciones de seguridad (*Security Operations Centre*, SOC), el equipo de análisis forense digital y respuesta a incidentes (*Digital Forensics and Incident Response*, DFIR), así como la gestión de vulnerabilidades y amenazas internas.

3. Táctica y técnica: describe situaciones concretas que están sucediendo actualmente en el panorama de amenazas, que a menudo son procesadas por los sistemas de defensa para gestionar situaciones automáticamente, tales como indicadores de compromiso (*Indicators of Compromise*, IoCs) que se deben bloquear o resultados de análisis de malware. Facilita el trabajo de los operadores y analistas del SOC y operadores de plataformas e infraestructuras.

2.3.1. Plataforma de inteligencia de amenazas (*Threat Intelligence Platform*, TIP)

Además de los ejemplos aportados en la Sección 2.3. Ciberinteligencia de amenazas, cualquier fuente de datos que un analista considere útil para sus objetivos es susceptible de ser

empleada en el proceso de ciberinteligencia. Sin embargo, para poder hacer uso de una gran cantidad de datos de forma eficiente, es necesario automatizar la extracción y enriquecimiento de los datos, de manera que el analista pueda centrar sus esfuerzos y conocimientos en extraer inteligencia de los mismos, en lugar de en realizar un tedioso proceso de extracción manual. Para ello, además, es esencial prestar atención a la presentación y visualización de los mismos, utilizando indicadores útiles que ayuden en la toma de decisiones. Y por último, ser capaces de compartir e intercambiar inteligencia con otras organizaciones reconocidas, equipos de respuesta ante incidentes (*Computer Emergency Response Teams*, CERTs) y demás actores involucrados, es esencial para que el proceso sea eficiente y efectivo.

Para lograr todos los retos anteriores, existen multitud de TIPs tanto de código abierto como privadas. Estas automatizan la extracción de datos de todo tipo, la presentan de manera gráfica y útil, y en ocasiones incluyen también la gestión de la misma mediante estándares que estructuran la información para su compartición, como *Structured Threat Information eXpression* (STIX) o *Trusted Automated eXchange of Intelligence Information* (TAXII). La Tabla 2.1 recoge una comparativa de funcionalidades entre varias de las herramientas más populares en el sector [18].

Tabla 2.1: Comparativa de las principales TIP

Característica	**MISP**	**OpenCTI**	**Splunk**	**Mandiant Advantage**	**ThreatConnect**
Open-source	✓	✓			
Gestión de alertas		✓	✓	✓	✓
Puntuación de amenazas		✓	✓	✓	✓
Integración con *sandbox*			✓		✓
Mapeo MITRE ATT&CK		✓	✓	✓	✓
Integración SIEM		✓	✓	✓	✓
Soporte API	✓	✓	✓		✓
Dashboard global		✓	✓	✓	✓
Informes y reportes		✓	✓	✓	✓

Como muestra la Tabla 2.1, cada herramienta tiene distintas funcionalidades y características, e incluso integraciones entre ellas, que pueden motivar la elección de una u otra según las necesidades a cubrir. No obstante, como este proyecto se enfoca en el despliegue de la arquitectura en la nube, se ha realizado un análisis más en detalle de estos aspectos para las dos

herramientas *open-source*, ya que son las únicas que, dada su naturaleza, lo permiten: OpenCTI y *Malware Information Sharing Platform* (MISP). Además, el hecho de ser *open-source* no disminuye su uso o soporte. Multitud de organizaciones reconocidas, como la *Agència de Ciberseguretat de Catalunya*, emplean MISP en sus operaciones [20]. Por su parte, OpenCTI fue desarrollada inicialmente por la *Agence nationale de la sécurité des systèmes d'information* de Francia y el CERT de la Unión Europea (CERT-UE), y múltiples fabricantes y terceros desarrollan conectores para esta solución. Esto evidencia que hay una gran comunidad de organizaciones y empresas tanto públicas como privadas que usan o desarrollan componentes de las herramientas, avalando su soporte por parte de la comunidad.

En relación a las opciones de instalación de las dos herramientas, en primer lugar, OpenCTI, ofrece cuatro formas diferentes de despliegue [21]:

1. Desplegando directamente contenedores Docker.
2. Añadiendo Kubernetes como orquestador de los contenedores, haciendo uso de *charts* de Helm.
3. Desplegando microservicios en AWS, Azure o Google Cloud Platform (GCP) utilizando Terraform.
4. Instalando manualmente las dependencias y paquetes.

El método más relevante en este caso es el tercero, orientado a despliegues en nube pública. Se proponen además dos alternativas, una básica que realiza la instalación manual en máquinas virtuales, disponible para los tres CSPs [22], y otra más avanzada, que aprovecha más servicios específicos de AWS, entre los que destacan los siguientes [23, 24, 25]:

- Clúster de Amazon Elastic Container Service (Amazon ECS) para el cómputo principal de la herramienta
- Clúster de Redis para almacenar información de sesiones de usuarios y estado de las operaciones de procesamiento de los datos, entre otros.
- OpenSearch para almacenar los datos de ciberinteligencia.
- RabbitMQ conectado a funciones Lambda y a un volumen Amazon Elastic File System (Amazon EFS) para la gestión de mensajes durante la ingesta.
- *Buckets* de Amazon Simple Storage Service (Amazon S3) para almacenar registros de auditoría (*logs*) y objetos del clúster de Kubernetes.
- Servicios de red como *gateways* de Internet y de traducción de direcciones de red (*Network Address Translation*, NAT) y balanceadores de carga.

Como refleja la Figura 2.3, la infraestructura desplegada con Terraform para AWS sigue la filosofía de los microservicios, disgregando componentes según funcionalidad, haciendo uso de servicios específicos para cada fin y empleando algún servicio *serverless* como las funciones Lambda. Esto resulta en una arquitectura modular, flexible, escalable, eficiente en el consumo

de recursos, y además de sencilla de gestionar, pudiendo sustituir componentes y servicios en el desarrollo con facilidad.

Por otro lado, MISP ofrece multitud de opciones de despliegue, aupadas por la comunidad [26]:

- Utilizando máquinas virtuales con MISP preinstalado:
  - Imágenes para pruebas en VirtualBox y VMware (desplegable también con Vagrant).
  - Imágenes de máquina de Amazon (*Amazon Machine Images*, AMI) para instancias de Amazon EC2.
- Desplegando contenedores Docker con la herramienta dividida en módulos.
- Instalando las dependencias y paquetes, pudiendo hacer uso de multitud de herramientas como Puppet, Ansible, AutoMISP, paquetes RPM, etc.

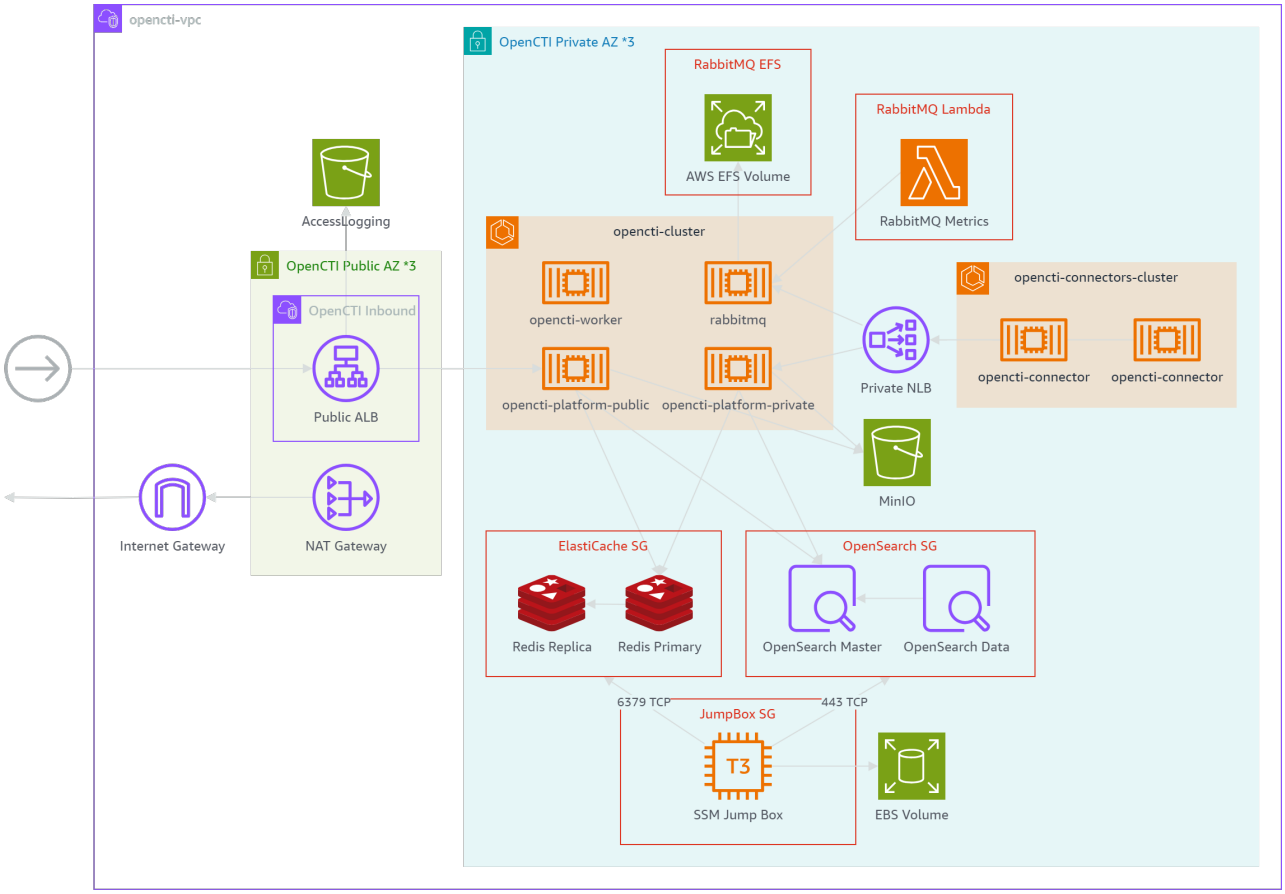


Figura 2.3: Diagrama de la arquitectura de OpenCTI en AWS (basado en [23])

De todas estas alternativas, existes dos que pueden estar más enfocadas a entornos *cloud*: las AMI, cuyo despliegue es análogo al de las máquinas virtuales, pero en el entorno de AWS, por lo que no utilizan apenas las ventajas de la nube, y el despliegue de contenedores Docker. Este último, del que existen varias versiones mantenidas por distintos usuarios y organismos,

está más enfocado a microservicios. La Figura 2.4 contiene un diagrama con la arquitectura de contenedores Docker de la versión oficial [27]. Existen otras alternativas con arquitecturas muy similares, como la desarrollada por la *National Cyber and Information Security Agency* de la República Checa en [28], centrada en el rendimiento y la seguridad, y con una arquitectura muy similar (integra el servicio de correo en el núcleo de MISP). En ambos casos, se emplea un contenedor con el núcleo de la herramienta, y sendos contenedores para las bases de datos Redis y MySQL, además de un contenedor adicional con módulos autónomos que incrementan las funcionalidades de MISP, con servicios como el enriquecimiento y la importación y exportación de datos [29]. No obstante, al desplegar esta herramienta en un entorno *cloud* con las opciones actualmente disponibles, apenas se estarían aprovechando ventajas del mismo, al contrario de como sucede con OpenCTI, que aprovecha las capacidades de Kubernetes y servicios específicos de AWS.

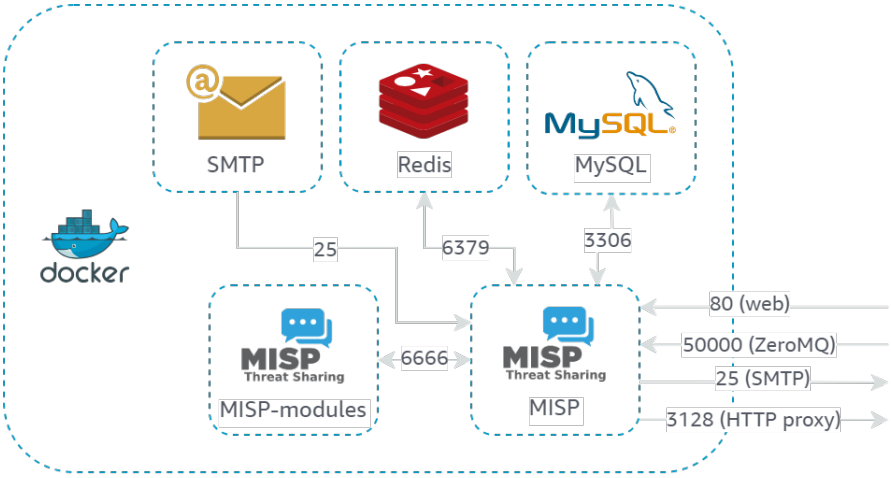


Figura 2.4: Diagrama de la arquitectura de MISP en Docker desarrollada en [27]

A la vista del análisis realizado, OpenCTI ha desarrollado una arquitectura pensada en su despliegue en la nube, escalable, fácilmente gestionable y teniendo en cuenta las ventajas tanto de la nube como de los microservicios. Por otro lado, MISP tiene una arquitectura ligeramente modular, pero sin aprovechar aún las ventajas de, por ejemplo, orquestadores como Kubernetes. Sin embargo, en ambos casos se disponen de servicios en ejecución permanente, incluso para la ingesta y procesado de datos, un proceso que no necesariamente debe ser continuo al nivel de segundos. Por ello, se tratará de desarrollar una propuesta de TIP que emplee la computación sin servidor basada en eventos para aquellos procesos en que sea posible hacerlo.

Capítulo 3

Plataforma de inteligencia de amenazas propuesta

A lo largo de este capítulo se describe la nueva TIP propuesta como desarrollo de este proyecto. Para ello, se han estudiado diferentes modelos novedosos y propios de la computación en la nube. Estos aportan valor respecto a las soluciones ya presentes en el mercado y son la base de la arquitectura desarrollada en este capítulo.

Haciendo uso de estas ventajas en el desarrollo de aplicaciones *cloud* y la seguridad de las mismas, se ha definido una arquitectura sencilla, pero altamente escalable y modificable para una nueva TIP que pueda crecer y equipararse a las ya existentes. Para ello, se han estudiado diferentes fuentes de datos de ciberinteligencia, optando por un indicador de riesgo definido por el Center for Internet Security (CIS). Seguidamente, se han analizado las necesidades del resto de elementos necesarios para extraer, almacenar y visualizar los datos, así como el método que determina cuándo se realiza la extracción.

3.1. Motivación

Tal y como se ha introducido desde el Capítulo 1. Introducción, en este trabajo se busca desarrollar una nueva TIP que cuente con una arquitectura diseñada pensando en las capacidades y oportunidades de la computación en la nube. Es habitual encontrar despliegues en la nube que son copias exactas de los tradicionales despliegues *on-premises*, con multitud de servidores con su sistema operativo. No obstante, estos desarrollos terminan generando más costes y aprovisionando recursos que no son necesarios, ya que se dispone de servidores en ejecución con recursos de cómputo inutilizados.

Teniendo en cuenta todas las capacidades que ofrece el *cloud* y la multitud de servicios específicos disponibles, desarrollar una arquitectura basada en servidores virtuales no es óptima en la mayoría de casos. Por supuesto que este producto tiene muchas aplicaciones útiles, pero se debe desarrollar una estrategia de migración que contemple también la modernización para escoger los servicios más apropiados a las necesidades y aprovechar todas las posibilidades de la nube. Sin embargo, en muchos casos este proceso requiere refactorizar código y aplicaciones, lo que demanda recursos humanos y dilata la migración.

Un primer paso es contar con aplicaciones basadas en microservicios, como es el caso de

OpenCTI (véase la Figura 2.3), que emplea varios servicios gestionados de AWS y permite escalar en función de la demanda cada uno de ellos. Además, si ocurren fallos o actualizaciones, no se requiere redesplegar toda la aplicación. No obstante, en este caso siguen existiendo recursos de cómputo en continua ejecución, como *pods* de Kubernetes, por los cuales los CSPs facturan.

Para evitarlo, se puede ir un paso más allá con las MSA y las EDA. Haciendo uso de estos paradigmas se consigue reducir el cómputo facturado al mínimo: únicamente los recursos empleados para ejecutar el código necesario en los momentos oportunos. En este proyecto, en concreto, se exploran estas soluciones para la ingesta de datos a las TIPs. Este proceso puede ser periódico (para recabar información todos los días, por ejemplo) o también desencadenado por situaciones de operaciones de una organización (para descargar los IoC más recientes durante una respuesta a incidentes). No es preciso disponer de un servicio en ejecución continua cuando de antemano se pueden definir estas situaciones y momentos periódicos en los que ejecutar el código, evitando además la gestión y escalabilidad de la infraestructura para ello.

3.2. Modelos a utilizar

A continuación se introducen diferentes modelos y patrones de diseño novedosos y altamente relacionados con la computación en la nube que serán empleados en la propuesta de este proyecto. Todos ellos tienen relación directa con los ya citados microservicios (véase la Sección 2.1. Arquitectura de microservicios (*Microservice Architecture*, MSA)), que suponen el fundamento a la vez que la novedad y valor de la nueva arquitectura a desarrollar.

3.2.1. AWS Serverless Application Model (AWS SAM)

El modelo AWS SAM es la concreción del paradigma de computación sin servidor dentro del entorno de AWS. Este marco consiste en un conjunto de herramientas diseñadas para facilitar el desarrollo rápido de aplicaciones, aprovechando los servicios sin servidor disponibles en AWS [30].

Para ello, AWS SAM proporciona plantillas predefinidas de IaC que se despliegan desde CloudFormation. Existe también un nuevo elemento que se puede emplear en estas plantillas, que son los conectores de AWS SAM. Estos se emplean para definir los permisos de lectura y/o escritura entre distintos servicios *serverless*, de manera unidireccional. De esta manera, el modelo se encarga de gestionar automáticamente las políticas de acceso necesarias para que las interacciones definidas se puedan llevar a cabo, permitiendo a los desarrolladores centrar sus esfuerzos en el diseño y desarrollo de la propia arquitectura de la aplicación.

La Figura 3.1 muestra un diagrama de una aplicación *serverless* de ejemplo, en el que una función AWS Lambda escribe datos en una tabla de la base de datos Amazon DynamoDB. En este caso, el elemento conector se define como parte de la función Lambda, con destino la tabla de DynamoDB (que debe haberse declarado previamente), y con permiso de escritura [31].

Además, AWS SAM consta también de una interfaz de línea de comandos (*Command Line Interface*, CLI) que permite controlar todo el ciclo de vida de las aplicaciones, como generar los ficheros de configuración necesarios, desplegarlas, probarlas localmente, sincronizar con la versión en la nube, etc. [32].

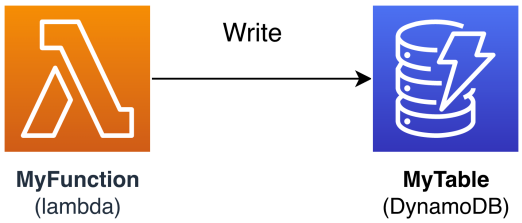


Figura 3.1: Diagrama de ejemplo de conector de AWS SAM [31]

3.2.2. Arquitectura basada en eventos (*Event-Driven Architecture*, EDA)

La EDA es un patrón de arquitectura moderna estrechamente relacionado con los microservicios y la computación sin servidor. Su fundamento son los cambios de estado de los elementos del sistema o del entorno, como el envío de un formulario web, un consumo de memoria determinado de un servidor, o un momento temporal. Estas relaciones entre servicios facilitan un acoplamiento flexible entre elementos, facilitando el despliegue, escalabilidad y actualización de los mismos de manera independiente [33, 34].

Existen tres elementos principales y desacoplados en una EDA:

1. Productores de eventos: generan un flujo de eventos desencadenando toda la respuesta posterior.
2. Enrutadores de eventos: ingieren, filtran, ordenan y almacenan los eventos hasta que son procesados.
3. Consumidores de eventos: responden a los eventos realizando acciones cuando estos suceden.

La Figura 3.2 muestra la estructura de un ejemplo básico de EDA, con los tres componentes mencionados. A la izquierda se distinguen varios productores de eventos que generan el nuevo pedido, la pregunta sobre la disponibilidad y la devolución. Los eventos pasan posteriormente a un enrutador que los filtra y envía a los consumidores apropiados, situados a la derecha, que los procesan y efectúan las acciones necesarias.

3.3. Arquitectura de la solución

Como se ha definido en el alcance de este proyecto, durante este proyecto se desarrolla tan solo una PoC, aportando la estructura de microservicios para que de manera sencilla y con poco esfuerzo se puedan agregar fuentes de datos u otras funcionalidades según se desee.

La arquitectura de esta PoC consta de los siguientes elementos (véase la Figura 3.3):

- Una fuente de datos de ciberinteligencia.
- Una base de datos para almacenar los datos extraídos.
- Una plataforma de visualización de datos.

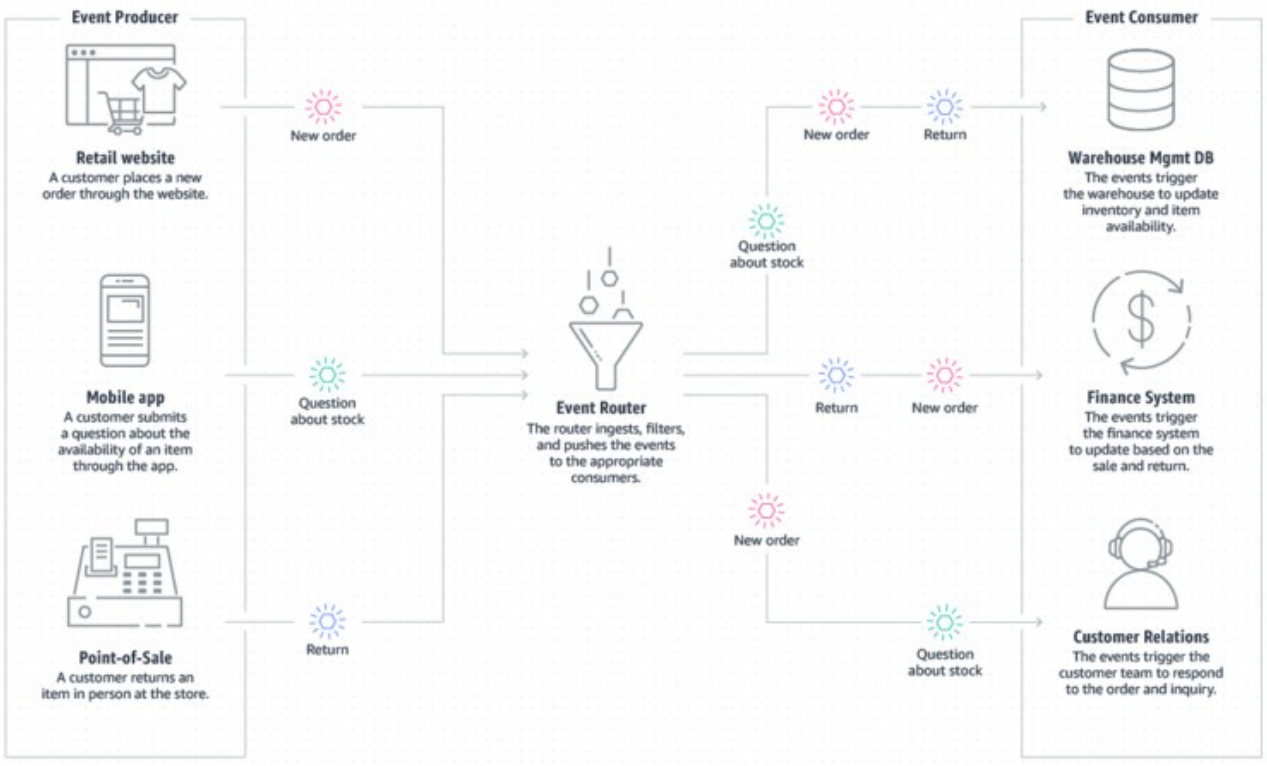


Figura 3.2: Diagrama de ejemplo de una EDA [33]

- Un *script* recolector de datos que los obtiene de la fuente y los almacena en la base de datos.

Figura 3.3: Arquitectura de la solución propuesta

3.3.1. Fuente de datos de ciberinteligencia

Existen multitud de organizaciones, tanto públicas como privadas, de reconocido prestigio dentro del sector de la ciberseguridad que publican datos de ciberinteligencia. No todas las fuentes de datos son abiertas, como es el caso de las herramientas de terceros, que venden la

información, o los centros de compartición y análisis de información (*Information Sharing and Analysis Centers*, ISACs), que son organizaciones sin ánimo de lucro que recopilan información de inteligencia y facilitan el intercambio, generalmente, solo entre sus miembros [35]. Sin embargo, hay otros grupos como CERTs diversos y organizaciones que sí hacen pública la información que procesan y generan.

Es también el caso del Center for Internet Security (CIS), entidad reconocida globalmente, que dispone de tres canales *Really Simple Syndication* (RSS), un formato de archivo basado en lenguaje de marcado extensible (*Extensible Markup Language*, XML) para distribuir contenido estructurado de manera periódica. En ellos, publica información generada a través de su *Multi-State Information Sharing and Analysis Center* (MS-ISAC) [36, 37], enfocado en la ciberseguridad de Estados Unidos:

1. `https://www.cisecurity.org/feed/alert`: proporciona información sobre el nivel de alerta de amenazas, cuándo se decidió, y la argumentación del mismo.

2. `https://www.cisecurity.org/feed/advisories`: recopila los avisos de seguridad que el CIS emite para diferentes tecnologías y productos.

3. `https://www.cisecurity.org/feed/blog`: contiene las noticias del blog del CIS.

El primero de ellos, con información sobre el nivel de alerta, ha sido el escogido para el desarrollo de esta PoC. Este define cinco posibles niveles, representados con cinco colores distintos para indicar la severidad, y calculados en base a sus miembros [38]. Se valoran cuatro indicadores diferentes [38]: (I) el daño potencial de la amenaza (letalidad), (II) el valor del objetivo (criticidad), (III) las medidas preventivas desplegadas en los *hosts* (contramedidas del sistema) y (IV) las medidas preventivas desplegadas en las redes (contramedidas de la red). Estos cuatro indicadores pueden tomar valores entre 1 y 5, siendo 1 el menos crítico o probable, y 5 el caso opuesto. La combinación de los mismos para obtener la severidad, con valores posibles entre -8 y $+8$, es como sigue:

$$Severidad = (Criticidad + Letalidad) - (Contramedidas\ sistema + Contramedidas\ red)$$

En función del valor de la severidad, se establece la siguiente escala cualitativa, que es la empleada por el CIS y distribuida en su canal RSS, página web, etc.:

1. Bajo (entre -8 y -5): no existe ninguna actividad inusual más allá de la preocupación normal por actividades maliciosas conocidas.

2. Vigilado (entre -4 y -2): existe la posibilidad de actividades maliciosas, pero no se han identificado explotaciones, o no han producido impactos significativos.

3. Elevado (entre -1 y $+2$): existen vulnerabilidades conocidas que están siendo explotadas con un nivel de impacto moderado, o la probabilidad de impacto es alta.

4. Alto (entre $+3$ y $+5$): las vulnerabilidades están siendo explotadas con un nivel de impacto alto, o la probabilidad de un impacto grave es alta.

5. Severo (entre +6 y +8): las vulnerabilidades están siendo explotadas con un impacto severo, generalizado o con interrupción en activos de infraestructuras críticas.

Este indicador de alerta ofrece una forma cuantitativa para conocer rápidamente el nivel actual de amenazas y su gravedad, teniendo en cuenta tanto su impacto como su capacidad para causar daño en activos importantes. Además, al restar el efecto de las contramedidas implementadas tanto a nivel de sistemas como de redes, se obtiene una medida más precisa de la verdadera amenaza enfrentada. Esto permite una gestión priorizada más efectiva, facilitando la asignación de recursos y la toma de decisiones informadas sobre las estrategias de las organizaciones. No obstante, al evaluar los cuatro indicadores que componen la ecuación en relación con los miembros del MS-ISAC, puede no tenerse un panorama preciso en otro tipo de organizaciones. Este aspecto, sin embargo, es totalmente comprensible y aceptable, ya que un ISAC actúa en relación a las características y condiciones que definen quién puede formar parte de sus miembros, pero debe conocerse a la hora de tomar decisiones.

Además de la idoneidad del nivel de alerta del CIS para cualquier TIP, la elección de este indicador para este proyecto se fundamenta también en su fácil acceso público, a través del canal RSS, y la simplicidad de los datos publicados en el mismo. Este aspecto, sin quitar valor a los datos, facilita la implementación del *script* recolector y la estructura de tablas de la base de datos, pues su complejidad no forma parte de los objetivos de este proyecto.

3.3.2. Ingesta y visualización de datos

El proceso de ingesta de datos encaja completamente en el paradigma de computación sin servidor, pues se realiza únicamente en los momentos que se definan. No es necesario aprovisionar un servidor (o equivalente) en el que instalar las dependencias necesarias para ejecutar el código y tenerlo operativo constantemente, cuando únicamente se va a emplear en instantes señalados. En contraposición, es posible desarrollar la ingesta por medio de funciones sin servidor, ya que son efímeras y de corta duración, y pueden escalar en función de la demanda si se añaden más fuentes de datos en un futuro. Además, se elimina la necesidad de mantener una infraestructura operativa constantemente, lo que ahorra costos y reduce la complejidad del sistema, a la par que aumenta la flexibilidad del mismo.

En este caso, dicho instante es un evento programado, con el fin de actualizar el nivel de alerta diariamente. Se puede configurar la periodicidad como se desee, pero la naturaleza del indicador recogido hace que no fluctúe con frecuencia de horas o menor. Podría incluso disminuirse la frecuencia y recoger los datos tan solo una vez a la semana, pero un cambio en el indicador puede implicar cambios dentro de las organizaciones miembro del MS-ISAC, y un periodo de una semana para detectarlo e iniciar los procedimientos oportunos puede ser demasiado alto, ya que, entre otras, se trata de organizaciones gubernamentales.

La plataforma de visualización sí debe ser una aplicación web disponible en todo momento para atender la demanda de los usuarios. Esta tendrá definida diferentes paneles y gráficos que muestren de manera clara y útil la información de ciberinteligencia extraída (el nivel de alerta, en este caso), como soporte a la toma de decisiones.

Un aspecto importante en la herramienta diseñada es la posibilidad de añadir fuentes de datos de manera sencilla. Para ello, tan solo hay que replicar los recolectores de datos, creando tantas funciones *serverless* como se desee y las tablas de la base de datos acordes. Posteriormente, únicamente queda definir los nuevos gráficos en la aplicación de visualización, que se

nutrirá de los datos añadidos a la base de datos por las funciones. Estas facilidades aseguran que la plataforma pueda evolucionar y adaptarse a medida que cambian las necesidades de ciberseguridad de la organización.

Capítulo 4

Implementación y validación

A lo largo de este capítulo se describe de manera detallada el proceso de desarrollo de la nueva TIP, prestando especial atención a las ventajas que ofrece el paradigma de la computación en la nube. Tras haber expuesto en el Capítulo 3. Plataforma de inteligencia de amenazas propuesta la arquitectura de la propuesta a alto nivel, se describe a continuación el diseño e implementación de la misma en AWS. Para ello, se describen los servicios empleados, así como los motivos por los que se han escogido, y se definen como IaC haciendo uso de CloudFormation.

En primer lugar, se aborda el enfoque de la infraestructura como código (*Infrastructure-as-Code*, IaC) y se analizan las ventajas que proporciona, así como la tecnología a utilizar y los motivos que impulsan dicha decisión. Seguidamente, se analiza en detalle la arquitectura propuesta, así como su implementación en código paso a paso. También se estudian los servicios adicionales necesarios para conectar y orquestar toda la infraestructura.

Seguidamente, se aborda el despliegue de la solución tanto en LocalStack como en AWS, de manera paralela a su validación tanto mediante análisis estático del código como verificando el correcto despliegue de cada recurso con los parámetros definidos.

Por último, se valoran diferentes aspectos de la propuesta que condicionan su puesta en producción, su versatilidad para otros casos de uso y la flexibilidad para portar la solución a otros entornos o CSPs.

4.1. Arquitectura e implementación de la solución

A continuación se describen los aspectos clave y las decisiones tomadas durante el desarrollo de la propuesta, con el objetivo de proporcionar una visión clara del proceso y garantizar una comprensión completa de la estrategia adoptada, facilitando así su replicabilidad, modificación, mejora y migración a otro CSP.

4.1.1. Infraestructura como código (*Infrastructure-as-Code*, IaC)

El enfoque de IaC es la práctica de definir una infraestructura deseada en forma de código. Esta ofrece numerosas ventajas, especialmente en términos de seguridad del despliegue. Gracias a la IaC, se elimina la posibilidad de errores humanos durante un despliegue manual y se garantiza que la infraestructura se aprovisione de manera consistente y reproducible, lo que reduce significativamente los riesgos de seguridad asociados con configuraciones incorrectas o

incompletas. Además, permite a los equipos definir y versionar su infraestructura de manera similar a como se realiza con el código de las aplicaciones, lo que facilita la colaboración y el control de versiones, así como la integración de los despliegues en los procesos de CI/CD.

Elección de solución

Para el desarrollo de este proyecto se ha optado por AWS CloudFormation, la herramienta nativa de AWS para la gestión de IaC. Esta proporciona un lenguaje declarativo para describir un *stack*: el conjunto de recursos de AWS deseados y que se van a crear y administrar de manera conjunta. Por ejemplo, un *stack* puede definir todos los recursos necesarios para ejecutar una aplicación web (servidor, base de datos, reglas de red, etc.), pudiendo ejecutarlo o eliminarlo todo en conjunto cuando sea necesario [39, 40].

Gracias a su integración con multitud de servicios y paradigmas de AWS, aparecen los ya mencionados conectores de AWS SAM (véase la Sección 3.2.1. AWS Serverless Application Model (AWS SAM)), que se encargan de gestionar las conexiones entre servicios, liberando al desarrollador de toda la configuración de permisos necesaria.

No obstante, si bien CloudFormation es la solución nativa de AWS, existen otras herramientas con mucha popularidad, como es el caso de Terraform, desarrollado por HashiCorp. Su principal ventaja es que admite múltiples proveedores de nube con un lenguaje común, en contraposición a CloudFormation. Aparte de opciones específicas de AWS, como los conectores de AWS SAM y otras integraciones y optimizaciones, ambas plataformas ofrecen capacidades similares en términos de definición de infraestructura como código y seguridad del despliegue.

Mientras que CloudFormation emplea los lenguajes *JavaScript Object Notation* (JSON) y *Yet Another Markup Language* (YAML), HashiCorp ha desarrollado un lenguaje propio, *HashiCorp Configuration Language* (HCL), similar y compatible con JSON. Terraform es modular, con la posibilidad de definir variables en diferentes archivos con distintas prioridades, anidar módulos, definir los datos de salida en archivos diferentes, etc., lo que en parte facilita el desarrollo, pero también añade complejidad al código. CloudFormation, en contraposición, ofrece menos opciones en cuanto a las variables, que se proporcionan en tiempo de ejecución, y orquestando todo el *stack* desde un mismo fichero fácilmente (aunque también se pueden anidar *stacks*), ganando sencillez.

De acuerdo con las ventajas y desventajas de cada herramienta, la elección entre Terraform y CloudFormation depende de las necesidades específicas del proyecto y organización, como por ejemplo la integración con diferentes proveedores. En este caso, dado que únicamente se van a aprovisionar recursos dentro de AWS, la integración de CloudFormation y su sencillez, así como la oportunidad de aprender una nueva tecnología, han sido claves a la hora de escoger esta plataforma.

Proceso de aprovisionamiento

Para poder aprovisionar recursos haciendo uso de CloudFormation, en primer lugar, es preciso definir el estado deseado en JSON o YAML, indistintamente. Para ello, se definen los tipos de recursos, acompañados de diferentes parámetros de configuración que permiten especificar las diferentes opciones, de igual manera que se pueden escoger en la consola web [41]. Con dicha plantilla, bien desde una consola en local o almacenándola en un *bucket* de Amazon S3, se eje-

cuta en CloudFormation, que mediante llamadas a las APIs de AWS, aprovisiona y configura todos los recursos definidos. Este proceso se muestra en la Figura 4.1.

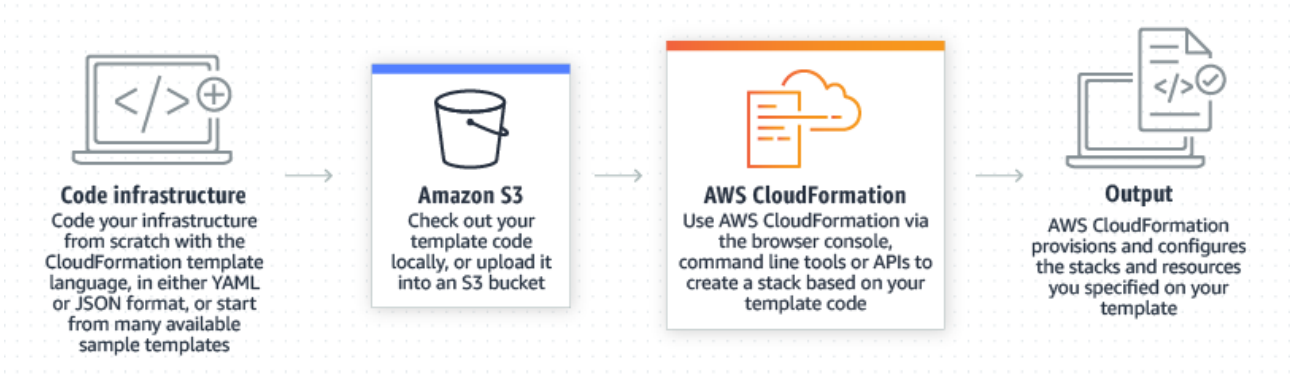


Figura 4.1: Proceso de aprovisionamiento de una infraestructura con CloudFormation [42]

La plantilla define lo que se conoce como *stack*, que es el conjunto de recursos que se gestionan desde de CloudFormation como una única unidad. Por medio de la plantilla, desde CloudFormation se puede tanto aprovisionar y crear los recursos como actualizarlos o eliminarlos todos en conjunto, como muestra la Figura 4.2. Esta separación lógica se utiliza para agrupar recursos relacionados, como por ejemplo una aplicación web que incluya el servidor, la base de datos, elementos de red, etc. [43].

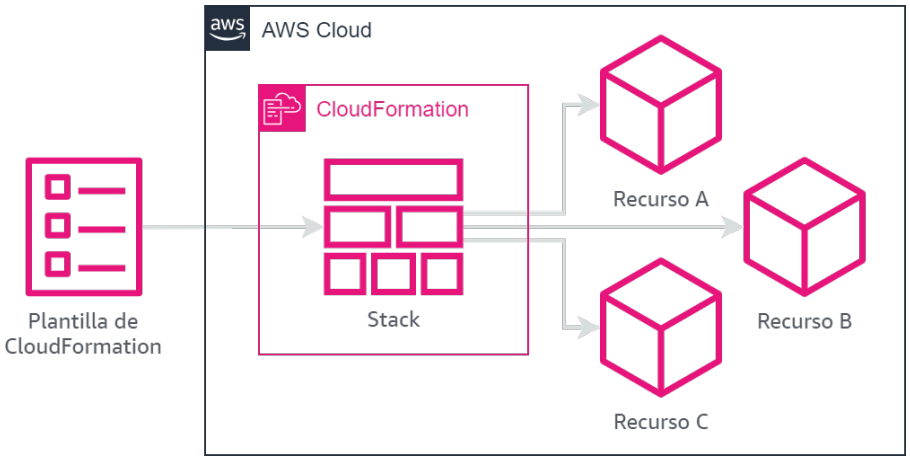


Figura 4.2: Funcionamiento de CloudFormation

4.1.2. Arquitectura de la solución

En la Sección 3.3. Arquitectura de la solución se definen los elementos básicos de la arquitectura de la herramienta. Además, se detalla cómo el recolector de datos encaja a la perfección con las funciones *serverless*, por lo que este elemento se implementa por medio de funciones Lambda.

En cuanto a la base de datos, se pueden emplear tanto bases de datos relacionales como no relacionales. Las primeras forman parte de Amazon Relational Database Service (Amazon RDS),

que proporciona ocho motores de bases de datos diferentes. No obstante, estas no están indicadas para aplicaciones *serverless* que tengan picos de demanda porque están basadas en conexión. En estos casos, es más apropiado emplear bases de datos de clave-valor y/o documentos y de alto rendimiento, como DynamoDB. En este caso de uso ambas opciones son apropiadas, dado que las ingestas de datos no van a ser excesivamente frecuentes ni en paralelo. Sin embargo, a pesar de que DynamoDB está integrada en el modelo AWS SAM, siendo más sencilla su implementación en código y el uso de los conectores, así como el escalado futuro de la herramienta, no se han encontrado herramientas de visualización que provean de una conexión sencilla y gratuita con esta base de datos dentro de sus licencias gratuitas. Por ello, se ha optado por Amazon RDS, empleando como motor PostgreSQL debido a su robustez, flexibilidad y sencillez, su inclusión en la capa gratuita de AWS [44], así como los conocimientos y experiencia del autor.

Por último, como plataforma de visualización, se ha escogido Grafana, una plataforma de visualización de datos y análisis que permite explorar, visualizar y compartir de manera fácil y rápida los datos almacenados en multitud de fuentes a través de una interfaz web. Su elección ha estado condicionada y motivada por su bajo consumo de recursos y su compatibilidad con PostgreSQL dentro de su licencia gratuita, ofreciendo una amplia gama de paneles y gráficos personalizables de forma sencilla a través de consultas SQL. No obstante, se han valorado previamente otras alternativas también populares como Metabase (más intuitiva pero mayor consumo de recursos) o Kibana (implementación compleja con Elasticsearch y muy alto consumo de recursos)

La arquitectura a alto nivel, incluyendo las tecnologías escogidas, es la mostrada en la Figura 4.3, con los elementos previamente definidos en la Figura 3.3.

Figura 4.3: Arquitectura de la solución con las tecnologías empleadas a alto nivel

A continuación se analiza la implementación de cada producto en IaC, así como las necesidades que tienen de otros servicios y elementos auxiliares para su funcionamiento.

Implementación de la base de datos

Para crear una instancia de Amazon RDS en CloudFormation, se debe usar el recurso `AWS::RDS::DBInstance`. Para configurar correctamente la base de datos, se han añadido unos parámetros que se deben establecer en el momento del despliegue, con el usuario, contraseña y

nombre de la base de datos. Además, se han especificado otras opciones de configuración para asegurar que la instancia se incluye dentro de los límites de la capa gratuita de AWS [44], como el tipo de almacenamiento, tipo de instancia, etc (véase el Apéndice B. Código del *stack* de AWS CloudFormation completo).

Además, para lograr que el recolector de datos pueda tener acceso a la instancia, se ha configurado como accesible públicamente, y se ha creado un grupo de seguridad abierto a cualquier direción dirección de protocolo de internet (*Internet Protocol*, IP). Cabe destacar que esto es una mala configuración, y en ningún caso debe replicarse en entornos de producción. Existen soluciones como agregar un Amazon RDS proxy o configurar el recolector dentro de la misma red y desplegar una puerta de enlace de Internet, pero todas ellas requieren de servicios con costes inasumibles dentro de este proyecto.

Implementación del recolector de datos

Para implementar el *script* recolector de datos hay que desarrollar en primer lugar el código de la función. Este debe realizar una petición `HTTP GET` al *endpoint* del canal RSS, y posteriormente crear e insertar los datos en la tabla de Amazon RDS. En este caso, se ha empleado el lenguaje Python debido a la familiaridad y habilidad del autor con el mismo, lo que garantiza un desarrollo más rápido y sencillo. Los detalles de la implementación se encuentran en el Apéndice A. Código de la función AWS Lambda recolectora de datos.

Para que el Código 4 funcione correctamente, hay que añadir una «capa» con la biblioteca `psycopg2`, empleada para interactuar con PostgreSQL. Este proceso requiere utilizar el recurso `AWS::Serverless::LayerVersion`, generar un archivo comprimido con la biblioteca con el Código 1, y subirlo a un *bucket* de Amazon S3 o situarlo en un directorio local desde el que se tomará durante el despliegue.

```
mkdir dependencies
cd dependencies
mkdir python
cd python
pip install psycopg2-binary -t ./
cd ..
zip ./psycopg2_package.zip .
```

Código 1: Generación de la capa con la librería `psycopg2`

Por último, hay que definir una regla de Amazon EventBridge que, siguiendo una programación diaria, cause la ejecución de la función Lambda. Para ello, hay que crear dos recursos nuevos: `AWS::Events::Rule`, que define la propia regla, y `AWS::Lambda::Permission`, para que la misma pueda invocar la ejecución de la Lambda. El código completo con estos dos elementos, además de la propia función Lambda, se encuentra en el Apéndice B. Código del *stack* de AWS CloudFormation completo.

Implementación de la plataforma de visualización

Grafana dispone de varias formas de instalación [45]. En este caso, se ha optado por desplegar contenedores Docker por su sencillez, aislamiento y consistencia [46]. Además, esta opción es perfecta en entornos *cloud*, ya que hay servicios gestionados de contenedores. No obstante, en el caso de AWS no están incluidos dentro de la capa gratuita [44], por lo que se ha optado por aprovisionar un servidor Amazon EC2, en el que se ha instalado Docker y se ha levantado un contenedores para Grafana. Para instalar estas dependencias al iniciar la instancia de Amazon EC2, se puede incluir un *script* de *shell* como datos de usuario (véase el Código 2), de manera que se ejecutará durante el primer arranque del servidor [47].

```
apt update -y
apt upgrade -y
apt autoremove -y

curl -fsSL https://get.docker.com -o install-docker.sh
sudo sh install-docker.sh
sudo groupadd docker
sudo usermod -aG docker ${USER}
newgrp docker

docker pull grafana/grafana
docker run -d -p 3000:3000 --name=grafana grafana/grafana
```

Código 2: Instalación de Docker, Elasticsearch y Grafana en Amazon Elastic Compute Cloud

Como AMI del servidor se ha empleado `ami-09e647bf7a368e505`, correspondiente a Amazon Linux 2023 x86_64, en la región de `Europa (Fránkfurt) (eu-central-1)`, liberada el 24 de mayo de 2024. Aunque inicialmente se ha valorado desplegar los recursos en la región de `Europa (España) (eu-south-2)`, se tuvo que descartar por problemas del despliegue en LocalStack.

Además, la instancia requiere de almacenamiento persistente, un grupo de seguridad que controle las conexiones de red entrantes y salientes y un par de claves para poder establecer conexión remota mediante SSH. Ambos elementos se han creado junto al servidor en el *stack* de CloudFormation (véase el Apéndice B. Código del *stack* de AWS CloudFormation completo).

En primer lugar, el almacenamiento persistente se puede obtener de Amazon Elastic Block Storage (Amazon EBS), un servicio de almacenamiento en bloque escalable que proporciona el volumen que monta la instancia Amazon EC2.

El grupo de seguridad, por su parte, permite conexiones entrantes únicamente desde la IP del autor (para reducir la superficie de exposición) para los puertos 22 (SSH) y 3000, el empleado por Grafana. Los rangos de IP deben ser modificados según el caso de uso.

El par de claves crea una clave pública, que se almacena en el servidor, y una clave privada que se debe usar para establecer conexión remota, guardada en el *Parameter Store* de AWS Systems Manager [48]. Durante el proceso de despliegue (véase la Sección 4.2. Despliegue y

validación de la solución) se muestra cómo recuperar y emplear la clave para obtener acceso remoto al servidor.

Una vez desplegado Grafana, en la Sección 4.2. Despliegue y validación de la solución se configurará la conexión a la base de datos y se creará un *dashboard* con gráficos apropiados a la fuente de datos escogida del CIS.

Arquitectura final

Teniendo en cuenta todos los elementos que se han añadido a medida que se han implementado en código los diferentes recursos seleccionados, resulta la infraestructura mostrada en la Figura 4.4. Se detallan, además, el tipo y nombre (cuando existe) de los recursos tal y como se definen en la plantilla de CloudFormation que implementa la infraestructura, descrita en el Apéndice B. Código del *stack* de AWS CloudFormation completo. Esta define, además, diferentes valores de salida o *outputs* que muestran datos útiles para acceder a la infraestructura, como la dirección IP pública de la instancia Amazon EC2, la *Uniform Resource Locator* (URL) de Grafana, así como identificadores de otros servicios.

El diagrama contiene todos los elementos analizados en la Sección 3.3. Arquitectura de la solución: (I) la fuente de datos, (II) el recolector, conectado a la regla temporal (con su política de permisos), (III) la base de datos y (IV) la plataforma de visualización dentro del servidor (conectado con el almacén de claves SSH).

Figura 4.4: Arquitectura final de la solución

4.2. Despliegue y validación de la solución

En las siguientes secciones se detalla el proceso de despliegue del *stack* de CloudFormation tanto en LocalStack, un entorno de desarrollo que emula el de AWS, como en la propia infraestructura de Amazon. Aunque el procedimiento es muy similar, se pretende proporcionar una alternativa muy útil en las etapas de desarrollo para probar las aplicaciones en un entorno sencillo de gestionar y sin riesgo de incurrir en costes innecesarios.

Durante la fase de despliegue se valida además que la infraestructura funciona, puesto que se demuestra cómo se aprovisionan exitosamente los recursos definidos mediante IaC, se ejecuta el recolector de datos y se visualiza el indicador de diferentes maneras útiles para el análisis de datos de ciberinteligencia. Además, respecto a la plantilla de IaC, tanto desde LocalStack como en AWS se realiza un análisis estático del código, comprobando los tipos de datos, opciones de configuración inexistentes, etc.

4.2.1. LocalStack

LocalStack es una herramienta que simula el entorno de AWS en local. Pretende servir como entorno de desarrollo, evitando los costes de recursos en el proveedor en esta fase del ciclo de vida, simplificando la gestión de cuentas de usuario para desarrolladores, la limpieza de recursos, etc. Aunque es de código abierto, ofrece ciertas características o prestaciones de pago. Además, cabe destacar que no consta de todos los servicios disponibles en AWS, sino que únicamente emula 83 de estos [49].

Esta herramienta, entre sus múltiples funcionalidades, implementa un *wrapper* de la AWS Serverless Application Model Command Line Interface (AWS SAM CLI). Para ello, crea el comando `samlocal`, que únicamente modifica los *endpoints* del comando `sam` con el objetivo de desplegar y gestionar aplicaciones AWS SAM dentro de LocalStack en lugar de AWS [50].

Una de las utilidades de `sam` y, por tanto, también de `samlocal`, es analizar estáticamente una plantilla de CloudFormation, empleando `cfn-lint` [51]. De esta manera, se consiguen detectar y eliminar errores de formato, sangría, usos incorrectos de los recursos y variables, etc. Para la plantilla de este proyecto, incluida en el Apéndice B. Código del *stack* de AWS CloudFormation completo, la validación es correcta, mostrándose el mensaje del Código 3.

Otro comando muy útil antes del despliegue es `sam list resources`, que muestra los recursos que se van a desplegar según la plantilla, así como su estado si ya se ha desplegado y se incluye la opción `--stack-name`. Esto permite visualizar qué cambios se van a realizar, para garantizar que se está conforme con el proceso, evitando generar recursos por descuido que generen tanto costes como problemas de seguridad, al no ser conscientes de su existencia.

El último paso es el propio despliegue, por medio de `samlocal deploy`, aunque su salida completa se omite en el Código 3 por simplicidad.

Durante el despliegue, se ha empleado la opción guiada, que realiza varias preguntas de configuración como el nombre del *stack*, la región, así como los parámetros definidos en la plantilla. Esta guarda todas las opciones de configuración de la AWS SAM CLI en el fichero `samconfig.toml` [52]. El despliegue del *stack* se puede verificar tanto en la salida del comando `samlocal deploy` como desde LocalStack Desktop [53], en caso de tenerlo instalado y configurado.

Además, al finalizar el despliegue se muestran todas las variables de salida definidas en la

plantilla (véase el Apéndice B. Código del *stack* de AWS CloudFormation completo). Cabe destacar que, por ejemplo, en LocalStack la instancia EC2 es simulada y no se puede acceder a Grafana a través de la URL obtenida.

```
ubuntu@ubuntu-tfm:~$ samlocal validate --lint --template template.yml
/home/ubuntu/template.yml is a valid SAM Template
ubuntu@ubuntu-tfm:~$ samlocal list resources --template template.yml
...
Resources
-------------------------------------------------------------------------
Logical ID                                              Physical ID
-------------------------------------------------------------------------
MyDBSecurityGroup                                       -
MyPostgreSQL                                            -
MyScheduledRule                                         -
PermissionForEventsToInvokeLambda                       -
MyEC2SecurityGroup                                      -
MyKeyPair                                               -
MyEC2Instance                                           -
MyLambdaFunction                                        -
MyLambdaFunctionRole                                    -
MyLambdaDependencies5972cb1644                          -
-------------------------------------------------------------------------
ubuntu@ubuntu-tfm:~$ samlocal deploy --guided --template template.yml
...
```

Código 3: Validación y despliegue de la plantilla de AWS SAM con `samlocal`

4.2.2. AWS

Al pasar a la infraestructura de AWS, se debe emplear de nuevo la AWS SAM CLI para poder subir el archivo comprimido con la biblioteca `psycopg2`. Si se desea realizar el proceso desde la consola de administración web, se debe subir el archivo a un *bucket* de Amazon S3 y modificar el Apéndice B. Código del *stack* de AWS CloudFormation completo para tomar el código desde ahí. Para el despliegue desde la AWS SAM CLI, es necesario (I) crear un usuario de IAM no administrador (recomendado pero no imprescindible), (II) crearle un par de claves de acceso [54], (III) asignarle permisos necesarios (véase el Apéndice C. Política de IAM para lanzar el *stack* de AWS CloudFormation), (IV) instalar la AWS Command Line Interface (AWS CLI) y (V) configurar las credenciales previamente generadas [55].

Dentro de AWS existe una herramienta que facilita el desarrollo de *stacks* de CloudFormation, denominada AWS Application Composer. Esta permite diseñar aplicaciones visualmente mediante bloques, permitiendo comprender de una manera sencilla la arquitectura, configuración y relaciones entre recursos. Al importar la plantilla del Apéndice B. Código del *stack* de AWS

CloudFormation completo en ella, se obtiene el diagrama contenido en la Figura 4.5, similar a la Figura 4.4, otorgando una validación visual del *stack*. Además, desde esta pantalla también se puede ejecutar el análisis estático con `cfn-lint`.

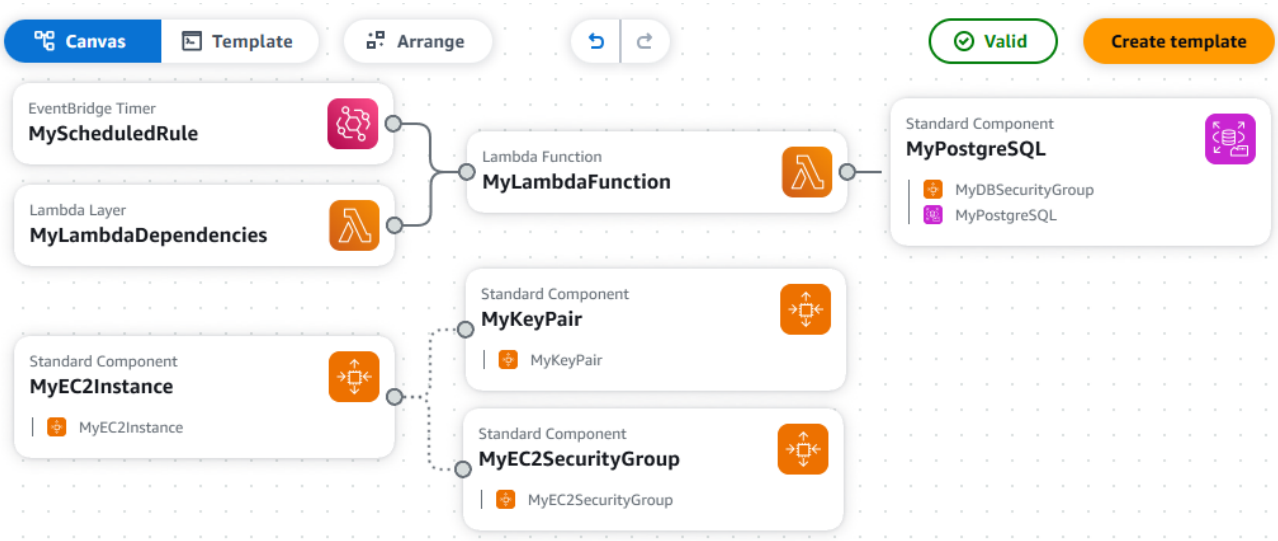


Figura 4.5: Diagrama de la infraestructura en AWS Application Composer

Para desplegar la plantilla, se debe ejecutar el comando `sam deploy` igual que en el Código 3 para `samlocal`. De nuevo se debe prestar atención a la estructura del directorio, debiendo existir el fichero `dependencies/psycopg2_package.zip` con la biblioteca `psycopg2` (véase el Código 1).

El despliegue culmina exitosamente cuando se obtiene un resultado similar al de la Figura 4.6, que muestra desde la pestaña de «Recursos» de la consola de CloudFormation el proceso de creación completado [1]. Tras esto, en el apartado de «Salidas» (véase la Figura 4.7) se muestran diferentes datos identificativos útiles tal y como se ha definido en la plantilla (véase el Apéndice B. Código del *stack* de AWS CloudFormation completo). Entre otros, se obtiene la URL de acceso al panel de Grafana.

Sin embargo, para poder visualizar el indicador en Grafana, primero hay que simular una extracción de datos, fuera del periodo definido (lunes 3:00 AM), de manera que se pueda crear un *dashboard* en Grafana que represente al menos una entrada en Amazon RDS. Desde la consola de administración de Lambda, se puede generar un evento de prueba con un JSON vacío, no es necesario que contenga ningún dato.

Tras esto, se puede acceder a Grafana y configurar una conexión con PostgreSQL con los datos introducidos por parámetros al desplegar el *stack* y las variables de salida. En concreto, hay que rellenar los siguientes campos:

1. `Host URL`: `tippostgresql.cbeuoig6udpp.eu-central-1.rds.amazonaws.com:5432`
2. `Database name`: `postgres`

[1] Se muestran las salidas desde la consola web en vez de emplear capturas o transcripciones del texto de la terminal porque así se facilita la claridad y comprensión del lector.

ID lógico	ID físico	Tipo	Estado
MyDBSecurityGroup	sam-tip-tfm-MyDBSecurityGroup-u27C9xklSyS7	AWS::EC2::SecurityGroup	CREATE_COMPLETE
MyEC2Instance	i-09f2b852d47f047ce	AWS::EC2::Instance	CREATE_COMPLETE
MyEC2SecurityGroup	kibana-ec2-sg	AWS::EC2::SecurityGroup	CREATE_COMPLETE
MyKeyPair	kibana-ec2-keys	AWS::EC2::KeyPair	CREATE_COMPLETE
MyLambdaDependenciesed29c4f602	arn:aws:lambda:eu-central-1:767397671557:layer:sam-tip-dependencies:3	AWS::Lambda::LayerVersion	CREATE_COMPLETE
MyLambdaFunction	data-extractor	AWS::Lambda::Function	CREATE_COMPLETE
MyLambdaFunctionRole	sam-tip-tfm-MyLambdaFunctionRole-Pjho22PHJdWe	AWS::IAM::Role	CREATE_COMPLETE
MyPostgreSQL	tippostgresql	AWS::RDS::DBInstance	CREATE_COMPLETE
MyScheduledRule	cis-rss-trigger-rule	AWS::Events::Rule	CREATE_COMPLETE
PermissionForEventsToInvokeLambda	sam-tip-tfm-PermissionForEventsToInvokeLambda-K2c6rSGd9kAo	AWS::Lambda::Permission	CREATE_COMPLETE

Figura 4.6: Creación completa de los recursos del *stack*

Clave	Valor	Descripción
EC2InstanceID	i-09f2b852d47f047ce	The EC2 instance ID
EC2InstancePublicIP	3.71.5.93	The public IP of the EC2 instance
EC2SecurityGroup	kibana-ec2-sg	The security group ID for the EC2 instance
GrafanaURL	http://ec2-3-71-5-93.eu-central-1.compute.amazonaws.com:3000	The URL to access Grafana
LambdaFunctionArn	arn:aws:lambda:eu-central-1:767397671557:function:data-extractor	The ARN of the Lambda function
RDSInstanceURL	tippostgresql.cbeuoig6udpp.eu-central-1.rds.amazonaws.com	The RDS PostgreSQL URL
ScheduledRuleId	cis-rss-trigger-rule	The ID of the EventBridge rule

Figura 4.7: Variables de salida del *stack*

3. `Username`: `postgres`
4. `Password`: `*******`
5. `TLS/SSL Mode`: `require`

4.2.3. Desarrollo de gráficos e indicadores de visualización

Una vez desplegado Grafana y configurado con la conexión a PostgreSQL, es momento de representar gráficamente el indicador recolectado. Dentro de esta PoC se han desarrollado dos visualizaciones o paneles diferentes que muestran de manera sencilla e intuitiva el nivel de alerta del CIS, facilitando su análisis y uso por parte del ciberanalista.

El primero tiene forma de manómetro o velocímetro, y muestra el nivel de alerta actual. Este indicador es crucial en el contexto de ciberinteligencia porque proporciona de manera instantánea el último valor, pudiendo actuar en consecuencia y tomar decisiones relacionadas con el nivel, como implementar ciertas medidas de seguridad, comunicar el estado a otros departamentos, etc.

El segundo gráfico es una serie temporal que permite mostrar cómo ha cambiado el nivel de alerta a lo largo del tiempo. Esto permite analizar tendencias o patrones, observando si el indicador cambia en determinadas épocas del año, o si hay alguna tendencia ascendente que pueda implicar, por ejemplo, cambios en las políticas de ciberseguridad de la compañía. Además, con el auge de la inteligencia artificial, la serie temporal no sólo es útil para el análisis retrospectivo, sino que también puede emplearse para realizar predicciones futuras, permitiendo anticiparse al nivel de riesgo. No obstante, los algoritmos de aprendizaje automático (*Machine Learning*, ML) requerirían de más variables de entrada para las predicciones, como el estado de las medidas de seguridad, histórico de ciberincidentes, etc. Este panorama podría plantearse dentro de una compañía, pero sería más desafiante a la vez que valioso realizarlo desde el MS-ISAC para sus miembros, pudiendo alertarles del posible futuro nivel. Si este horizonte se alcalzase, el MS-ISAC podría incluso generar recomendaciones que los miembros y otras organizaciones pudieran aplicar para evitar que el nivel de riesgo alcanzase el nivel predicho.

Ambos gráficos emplean la escala numérica de niveles y la escala de colores definidas en la Sección 3.3.1. Fuente de datos de ciberinteligencia. La Figura 4.8 muestra el resultado de ambos paneles, mientras que el detalle de las sentencias SQL utilizadas se detalla en el Apéndice E. Consultas SQL de los paneles de Grafana. Cabe destacar que la herramienta no se ha mantenido desplegada el tiempo necesario como para obtener varios valores diferentes del indicador que pudieran reflejarse en la serie temporal, por lo que únicamente se puede ver un punto.

4.2.4. Escalabilidad

En este apartado se pretende valorar los cambios necesarios para trasladar la propuesta a un entorno de producción real y que sea capaz de adaptarse a las variaciones en la demanda habituales en un equipo de ciberinteligencia.

La arquitectura propuesta es totalmente compatible con un entorno en producción. La elección de los servicios, así como el uso de Grafana, que tiene un consumo de recursos muy bajo, permiten que el despliegue y uso de la TIP propuesta sea directo. No obstante, hay algunos cambios que se recomiendan acometer para mejorar la resiliencia del sistema, especialmente

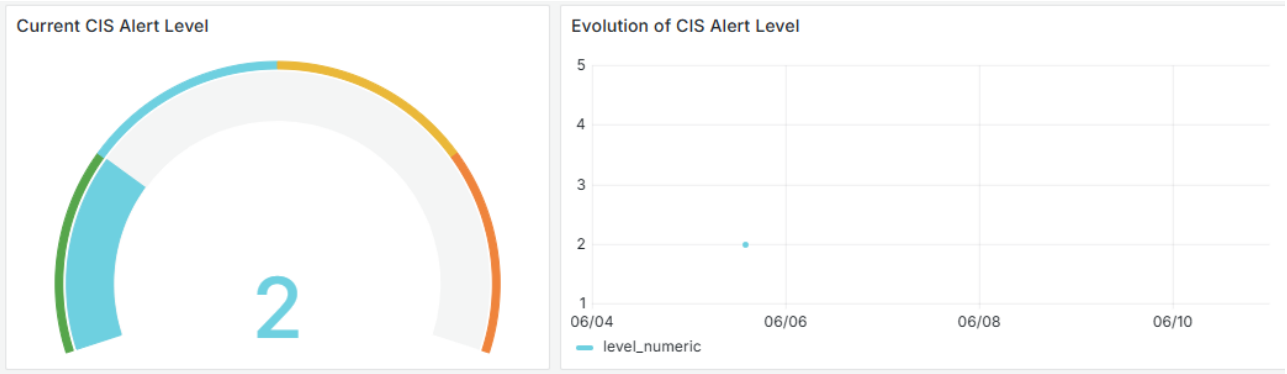


Figura 4.8: Paneles definidos en Grafana

en los servicios no incluidos en el paradigma *serverless*. El más importante es garantizar la disponibilidad, tanto de la herramienta de visualización como de la base de datos. El primero debe admitir conexiones simultáneas de múltiples analistas, bien configurando un grupo de autoescalado de Amazon EC2 o utilizando el servicio gestionado Amazon Managed Grafana. Para la base de datos, el cambio más sencillo es mejorar el tipo de instancia, de manera que se garantice que los recolectores puedan escribir concurrentemente a la vez que se ejecutan múltiples consultas SQL desde Grafana.

Sin embargo, de nada sirve garantizar la disponibilidad si se despliega la propuesta con configuraciones de seguridad deficientes. Aunque en este proyecto ha sido inevitable introducirlas (véase la Capítulo 5. Valoración económica), se deben corregir para evitar problemas en producción. En concreto, el problema es la regla de entrada del grupo de seguridad de la base de datos, que la expone públicamente a todo Internet con el rango `0.0.0.0/0`. Para evitarlo, se puede configurar la función Lambda dentro de una red Amazon Virtual Private Cloud (Amazon VPC) con una puerta de enlace a Internet, o configurar un Amazon RDS Proxy que gestione las conexiones. También es apropiado configurar *Hypertext Transfer Protocol Secure* (HTTPS) en Grafana para garantizar la privacidad de las comunicaciones con la herramienta [56].

Finalmente, es crucial considerar la administración de usuarios en todos los niveles de la infraestructura de la herramienta. Esto abarca principalmente la gestión de los usuarios de Grafana necesarios para los ciberanalistas y el usuario empleado para la inserción de datos en PostgreSQL, proceso que se ha realizado con el usuario administrador en este proyecto.

4.2.5. Usabilidad

Al emplear el paradigma de computación sin servidor, las EDA y el desarrollo mediante IaC, la versatilidad de la propuesta para otros casos de uso es máxima. Se ha desarrollado una arquitectura totalmente modular, basada en microservicios que implica cambios mínimos para modificarla y/o ampliarla según las necesidades. Con unos mínimos conocimientos y comprensión del proceso desarrollado en este proyecto se puede modificar la plantilla de CloudFormation para incluir nuevas fuentes de datos añadiendo recolectores de datos (funciones Lambda), tablas de la base de datos y paneles de Grafana, reemplazar unos servicios por otros, etc.

El mayor desafío de este proceso es principalmente la implementación del recolector, que

en ocasiones será sencilla si se dispone de APIs de proveedores de ciberinteligencia, pero puede complicarse si se debe realizar una extracción más manual mediante *web scraping*, por ejemplo. Este último escenario puede ser especialmente desafiante en determinados contextos, ya que la obtención de información de fuentes como foros *underground* o *dark web*, que a menudo contienen indicadores de ciberinteligencia muy valiosos, suele estar mucho más monitorizado y restringido.

4.2.6. Flexibilidad

En relación con la flexibilidad y portabilidad de la propuesta de TIP desarrollada en este proyecto, se han seguido conceptos y paradigmas presentes en otros CSP, como principios de arquitectura y mejores prácticas tanto del propio desarrollo de aplicaciones como de aspectos de ciberseguridad, principalmente garantizando la disponibilidad. No obstante, la implementación se ha realizado adaptada a AWS, empleando sus servicios, así como su servicio de IaC. Esta decisión asegura una óptima integración de los servicios y una alta eficiencia en aspectos de la gestión y despliegue de la infraestructura, aunque implica que la implementación es muy dependiente de este proveedor.

Para llevar a cabo una implementación y despliegue exitosos en otros proveedores, se debe estudiar detalladamente su cartera de servicios, especialmente los que implementan la computación sin servidor, así como sus herramientas de IaC (o bien escoger una solución *multi-cloud* como Terraform). Existen componentes que se pueden reutilizar, como el código Python que extrae el indicador del CIS o la instalación de Grafana con Docker.

Una vez sustituidos los servicios, es esencial comprender su configuración para poder realizar un despliegue seguro, evitando las configuraciones por defecto que puedan causar problemas de seguridad como exponer el servidor virtual o la base de datos públicamente, permitir que cualquiera desencadene la ejecución de las funciones *serverless*, etc.

Capítulo 5

Valoración económica

Una de las ventajas de la nube es el pago por uso de los recursos consumidos, que permite gestionar el presupuesto más eficientemente al preveer los gastos, evitar gastos innecesarios y, en definitiva, optimizar los costes operativos. Para ello, es fundamental realizar una evaluación de costes en AWS de manera previa a la implantación de un proyecto. Además, el enfoque de la IaC también contribuye a evitar facturas inesperadas, al permitir una gestión y previsión más precisa de los recursos y sus configuraciones, y la facilidad que supone poder administrar las aplicaciones y *stacks* de manera centralizada.

Al igual que en este proyecto, el coste de la infraestructura es uno de los principales condicionantes en el diseño de cualquier arquitectura, pues el cumplir los objetivos presupuestarios es esencial para poder desarrollar la propuesta.

En este caso, ha sido crucial minimizar los costes todo lo posible, adaptando los servicios seleccionados al catálogo de la capa gratuita que AWS pone a disposición de sus clientes [44]. Además, se ha cometido una configuración de seguridad inadecuada también por problemas de costes, como es exponer a todo Internet (rango `0.0.0.0/0`) la base de datos Amazon RDS, para que la función Lambda y la instancia Amazon EC2 tengan acceso a la misma sin conocer su dirección IP en cada ejecución.

Dada la arquitectura propuesta, representada en la Figura 4.4 y la Figura 4.5, es necesario valorar los siguiente cuatro servicios para la evaluación de los costes del laboratorio: (I) Amazon Relational Database Service, (II) AWS Lambda, (III) Amazon EventBridge y (IV) Amazon Elastic Compute Cloud.

A continuación, se realiza un análisis de la influencia de la capa gratuita en estos cuatro servicios, pudiendo así comprender la elección de los recursos, tipos de instancias y opciones de configuración.

Además, como segunda parte de este ejercicio se realiza una estimación empleando la herramienta AWS Pricing Calculator, con la misma configuración y caso de uso de este proyecto, y excluyendo los servicios gratuitos. Ambos procesos se efectúan para la región de `Europa (Fránkfurt) (eu-central-1)`, en la que se ha desplegado el proyecto.

5.1. Análisis de la capa gratuita

En primer lugar se analizan las posibilidades de la capa gratuita de AWS en relación con los servicios utilizados, tratando de ofrecer una explicación a algunas decisiones del diseño y configuración de los mismos.

5.1.1. Amazon Relational Database Service

En el primer caso, durante los primeros 12 meses, para Amazon RDS se proporcionan 750 horas al mes de instancias `db.t2.micro`, `db.t3.micro` y `db.t4g.micro` en una sola zona de disponibilidad, para los motores MySQL, MariaDB y PostgreSQL. Estas instancias son las más básicas y todas tienen 1 CPU virtual, y entre 1 y 2 GiB de memoria. El rendimiento de red y los créditos por hora de CPU fluctúa ligeramente entre instancias [57]. Además, el tamaño no puede exceder de 20 GB.

Estos recursos son suficientes para la PoC desarrollada, por lo que todo el despliegue de la base de datos no generaría ningún coste incluso si la instancia está en ejecución continua todo el mes, siempre y cuando sea la única. No obstante, necesitarían incrementarse los valores si aumentan los accesos concurrentes a la instancia o si se desea configurar replicación en múltiples zonas de disponibilidad o copias de seguridad, entre otros.

Cabe destacar que el uso del grupo de seguridad asociado a la instancia de Amazon RDS no tiene ningún coste.

5.1.2. AWS Lambda

En el caso de Lambda, de manera indefinida todas las cuentas disponen de 1.000.000 solicitudes gratuitas al mes, y 3,2 millones de segundos de ejecución. Esto es más que suficiente para el caso de uso desarrollado, incluso añadiendo multitud de conectores a la plataforma TIP.

Si se exceden estos valores, hasta 6 billones de GB-segundos, el coste de funciones de arquitectura x86 sigue siendo muy bajo: 0,0000166667 $/GB-segundo de duración y 0,20 $ por cada millón de peticiones. Si se superan los 6 billones de GB-segundos en un mes, que sería un caso extremo considerando la herramienta desarrollada, el coste del GB-segundo se vería disminuido, y si la arquitectura es Arm, todas estas cantidades se reducen más aún [58].

El uso de librerías adicionales mediante capas tampoco tiene ningún coste asociado, por lo que emplear `psycopg2` no supone ningún cargo complementario. Tampoco supone ningún coste el permiso creado para que EventBridge pueda disparar la ejecución de la función.

5.1.3. Amazon EventBridge

Dentro de EventBridge se emplea una regla temporal que envía un evento disparador de la función Lambda. En la capa gratuita se proporcionan indefinidamente 1 millón de eventos de entrada al mes, lo que es muy superior a la demanda de este proyecto, incluso aunque se incrementase la cantidad de recolectores de datos. El coste adicional al exceder esta cantidad es de 1 $ por cada millón adicional, por lo que el coste sería ínfimo [59].

5.1.4. Amazon Elastic Compute Cloud

Por último, respecto al servidor Amazon EC2 existen también multitud de tipos de instancias, con cantidades de recursos diferentes, al igual que en Amazon RDS. Esto implica que el cálculo exacto del coste depende mucho de las opciones elegidas.

Dentro de la capa gratuita, los primeros 12 meses de uso de la cuenta se disponen de 750 horas de ejecución mensual de instancias de tipo `t2.micro` o `t3.micro` (cuando la región no disponga de `t2.micro`). Ambas disponen de 1 CPU virtual, y 1 GiB y 2 GiB de memoria, respectivamente [60]. Este tiempo equivale a tener una única instancia en ejecución continua durante todo el mes, y los recursos son suficentes para la ejecución de Grafana sin problemas.

La capa gratuita también incluye 30 GB de almacenamiento persistente en Amazon EBS, de los cuales únicamente se han empleado 10 GB. El precio para consumo adicional es de 0,0952 $/GB-mes. También se deben considerar los operaciones de entrada/salida por segundo (*Input/Output Operations Per Second*, IOPS) del disco, siendo gratuito hasta 3.000 IOPS, y el rendimiento (*throughput*), con 125 MB/s gratuitos [61, 62].

5.2. Estimación de costes

Dentro del ejercicio de evaluación económica del despliegue, se ha realizado una estimación de costes usando AWS Pricing Calculator. Esta herramienta detalla todos los parámetros de configuración que influyen en las facturas, de manera que el usuario los puede identificar fácilmente y completar con los valores que desee según sus necesidades específicas. Además, al mostrar los diferentes recursos y opciones junto a su tarifa asociada, permite comparar diferentes alternativas y seleccionar las opciones más económicas y adecuadas.

Para esta actividad, se han tenido en cuenta los mismos parámetros empleados en la propuesta (véase el Apéndice B). Aunque en un despliegue en producción quizá no sean los valores óptimos, se pretende ofrecer un análisis real del coste del mismo laboratorio desarrollado, sin contar con el descuento de la capa gratuita, proporcionando una guía del análisis que se ha de realizar al escalar o modificar la solución propuesta.

La Tabla 5.1 contiene el desglose por servicio del coste mensual y anual, calculado según los siguientes parámetros, obtenidos directamente del informe de AWS Pricing Calculator:

1. Amazon RDS: *Storage amount (20 GB), Storage volume (General Purpose SSD (gp2)), Nodes (1), Instance Type (db.t3.micro), Utilization (On-Demand only) (100 %Utilized/Month), Deployment Option (Single-AZ), Pricing Model (OnDemand).*

2. Lambda: *Architecture (x86), Amount of ephemeral storage allocated (512 MB), Architecture (x86), Invoke Mode (Buffered), Number of requests (4 per month).*

3. EventBridge: *Size of the payload (1 KB), Number of custom events (5 per month), Number of invocations (5 per month).*

4. Amazon EC2: *Tenancy (Shared Instances), Operating system (Linux), Workload (Consistent, Number of instances: 1), Advance EC2 instance (t2.micro), Pricing strategy (On-Demand Utilization: 100 %Utilized/Month), Enable monitoring (disabled), EBS Storage amount (10 GB), DT Inbound: Not selected (0 TB per month), DT Outbound: Not selected (0 TB per month), DT Intra-Region: (0 TB per month).*

Tabla 5.1: Resumen de la estimación de costes con AWS Pricing Calculator

Servicio	**Concepto**	**Coste mensual**	**Coste anual**
Amazon Relational Database Service	Instancia Amazon RDS PostgreSQL	15,33 $	183,96 $
	Almacenamiento	2,74 $	32,88 $
AWS Lambda	Cómputo	0,00 $	0,00 $
Amazon EventBridge	Eventos programados	0,00 $	0,00 $
Amazon Elastic Compute Cloud	Instancia Amazon EC2 *On-Demand*	9,78 $	117,36 $
	Almacenamiento Amazon EBS	0,95 $	11,40 $
Total	**Total**	**28,80 $**	**345,60 $**

Los detalles de la estimación están disponibles hasta el 11 de junio de 2025 en [63].

Capítulo 6

Conclusiones y trabajos futuros

En este proyecto, se ha llevado a cabo el desarrollo de una plataforma TIP empleando microservicios, en concreto el paradigma *serverless* sobre el entorno de AWS. A continuación, se diferencian las conclusiones técnicas y personales derivadas de la realización del mismo.

6.1. Conclusiones técnicas

Durante la implementación de la solución para equipos de ciberinteligencia se han identificado distintas soluciones y tecnologías de microservicios que facilitan el trabajo tanto a los desarrolladores como al equipo de seguridad durante todo el ciclo de vida de las aplicaciones. Empleando paradigmas y modelos de desarrollo novedosos, gracias a la computación en la nube, se ha conseguido implementar y desplegar una TIP totalmente funcional, modular, escalable y flexible que puede ser empleada por analistas para visualizar todo tipo de datos de forma clara y sencilla, de manera que les permite tomar decisiones de una manera informada y rápida.

De manera añadida, al haber implementado la solución como IaC, se consigue tener una herramienta reproducible y fácil de auditar en relación a la seguridad y configuración de los servicios empleados. Además, esta opción resulta muy útil para gestionar todo el ciclo de vida de la aplicación, facilitando tanto despliegues rápidos y sencillos como un control continuo de los recursos hasta su eliminación de manera conjunta.

Por otro lado, el proyecto ha servido como una oportunidad para analizar detalladamente y comprender la multitud de configuraciones de los servicios *cloud*, buscando garantizar un despliegue tan seguro como fuera posible dentro del escaso presupuesto disponible. Sin embargo, se ha enfrentado el desafío de tener que equilibrar la implementación de medidas y configuraciones de seguridad deseables con las restricciones económicas. Esta dicotomía no es más que la realidad diaria de los equipos de ingeniería, que deben gestionar el presupuesto para cumplir tanto los requisitos funcionales como los de seguridad.

Además, en relación con los impactos en sostenibilidad, ético-social y de diversidad, creando una herramienta útil para ciberanalistas pero sin destruir en absoluto puestos laborales, para ayudarles a preservar derechos básicos de los usuarios en Internet utilizando fuentes de información que no causan sesgos que afecten de manera desigual a las personas, y que contribuye al uso limitado y optimizado de recursos energéticos y computacionales. Como ya se adelantaba en el Capítulo 1. Introducción, a pesar de que los Centro de Procesamiento de Datos

(CPD) consumen una gran cantidad de estos recursos, este se realiza de manera compartida y optimizada entre multitud de usuarios.

Todos los hechos y éxitos descritos previamente evidencian el cumplimiento de los objetivos tanto generales como específicos enunciados en el Capítulo 1. Introducción.

6.2. Reflexiones personales

A nivel personal, este proyecto ha sido un desafío que ha implicado un continuo aprendizaje, combinando las tareas de desarrollo con la visión de ciberseguridad a lo largo de todo el proceso. Desde el estudio de los microservicios y la investigación de indicadores en fuentes abiertas que fueran de interés para la ciberinteligencia de amenazas, hasta el desarrollo de IaC de toda la infraestructura de manera segura. El desarrollo ha requerido de un extenso estudio también de los diferentes servicios gestionados de AWS, así como de herramientas de visualización gestionadas y de terceros integrables con los demás productos. Este último paso ha supuesto mucho trabajo de pruebas fallidas con herramientas como Elasticsearch y Kibana, bases de datos como DynamoDB, etc.

Sin embargo, el desafío en ningún caso ha impedido lograr por completo el desarrollo de la solución planteada en el Capítulo 1. Introducción y el Capítulo 2. Estado del arte, aportando una versión preliminar de una herramienta totalmente útil para realizar inteligencia de amenazas para proteger cualquier organización a través de indicadores útiles. Además, los paneles de visualización desarrollados en Grafana generan una oportunidad para aplicar algoritmos de inteligencia artificial (IA) y ML para realizar predicción y recomendación, tan presentes en cualquier ámbito hoy en día.

En relación a la planificación, es necesario mencionar que ha sido complicado cumplirla en su totalidad debido tanto a imprevistos acontencidos, especialmente en el desarrollo del *stack* y la implementación de la propuesta, como al seguimiento semanal de la titulación y el resto de asignaturas.

6.3. Líneas de trabajo futuro

Debido a las mejoras y actualizaciones constantes de la oferta de servicios de los CSPs, este debe ser valorado de manera previa a cualquier replicación de este proyecto, para comprobar que el diseño y desarrollo en IaC sigue siendo apropiado y funcional.

La forma más sencilla y directa de mejorar la herramienta es aumentar su capacidad creando más recolectores de datos. Estas nuevas funciones Lambda obtendrían información de otras fuentes complementarias, enriqueciendo el panel de visualización y proporcionando a los analistas una visión más completa y diversa para tomar decisiones informadas. Ampliar el conjunto de datos disponibles mejorará significativamente la capacidad de la plataforma para detectar y analizar amenazas de manera más precisa y efectiva.

Otra posible línea de trabajo futuro es, como ya se ha introducido, la adición de un módulo de predicción. Este módulo utilizaría técnicas de ML para anticipar posibles amenazas o niveles de alerta, proporcionando a los ciberanalistas herramientas más avanzadas para tomar decisiones proactivas en lugar de reactivas.

Otra área de mejora es rehacer el diseño con un límite presupuestario mayor. Esto permitiría aumentar los requisitos de seguridad en relación con la configuración de los servicios, así como utilizar servicios más adecuados que no se pudieron emplear inicialmente debido a restricciones económicas (véase la Capítulo 5. Valoración económica). Esto incluiría la integración configuraciones de seguridad que se descartaron en este proyecto, mayor redundancia y escalabilidad, y el uso de servicios gestionados más sencillos de operar.

Bibliografía

[1] Eurostat. (Marzo 2022) "*Cloud computing services.*" Luxemburgo, Luxemburgo. [En línea]. Disponible en: https://ec.europa.eu/eurostat/databrowser/view/isoc_cicce_use/default/bar?lang=en (Último acceso: 01-04-2024).

[2] R. Bala, B. Gill, D. Smith, D. Wright, y K. Ji, "*Magic Quadrant for Cloud Infrastructure and Platform Services*," *Gartner Magic Quadrant*, Julio 2021. [En línea]. Disponible en: https://www.gartner.com/doc/reprints?id=1-271OE4VR&ct=210802

[3] Universitat Oberta de Catalunya. (Septiembre 2022) "*Guía transversal para el estudiantado de TF de los Estudios de Informática, Multimedia y Telecomunicación.*" (Último acceso: 05-03-2024).

[4] A. Nehme, V. Jesus, K. Mahbub, y A. Abdallah, "*Securing Microservices*," *IT professional*, vol. 21, no. 1, pp. 42–49, Marzo 2019.

[5] M. Waseem, P. Liang, y M. Shahin, "*A Systematic Mapping Study on Microservices Architecture in DevOps*," *The Journal of systems and software*, vol. 170, p. 110798, Diciembre 2020.

[6] V. Velepucha y P. Flores, "*A Survey on Microservices Architecture: Principles, Patterns and Migration Challenges*," *IEEE access*, vol. 11, pp. 88 339–88 358, Agosto 2023.

[7] Amazon Web Services. (2024) "*What are Microservices?*" [En línea]. Disponible en: https://aws.amazon.com/microservices/ (Último acceso: 01-04-2024).

[8] Oracle Cloud. (2024) "*Learn About the Microservices Architecture.*" [En línea]. Disponible en: https://docs.oracle.com/en/solutions/learn-architect-microservice/index.html (Último acceso: 01-04-2024).

[9] Amazon Web Services. (2024) "*What is DevOps?*" [En línea]. Disponible en: https://aws.amazon.com/devops/what-is-devops/ (Último acceso: 01-04-2024).

[10] M. Portnoy, "*Understanding Applications in a Virtual Machine*," en *Virtualization Essentials*. Estados Unidos: John Wiley & Sons, Incorporated, 2023.

[11] Kubernetes, *Overview*, 2024, Kubernetes Documentation. [En línea]. Disponible en: https://kubernetes.io/docs/concepts/overview/ (Último acceso: 05-04-2024).

[12] Docker Inc., *Docker overview*, 2024, Docker Docs. [En línea]. Disponible en: https://docs.docker.com/get-started/overview/ (Último acceso: 05-04-2024).

[13] S. Newman, *Monolith to microservices: evolutionary patterns to transform your monolith*, 1ª ed. O'Reilly Media, Noviembre 2019.

[14] S. CM, *Architecting cloud native serverless solutions: design, build, and operate serverless solutions on cloud and open-source platforms*, 1ª ed. Packt Publishing, Junio 2023.

[15] J. J. McDermott, *Reading the Pentateuch: An Historical Introduction.* Paulist Press, Julio 2002.

[16] M. Lee, *Cyber Threat Intelligence*, 1ª ed. Wiley, Abril 2023.

[17] Gartner Research y R. McMillan. (Mayo 2013) "*Definition: Threat Intelligence.*" [En línea]. Disponible en: https://www.gartner.com/en/documents/2487216 (Último acceso: 06-04-2024).

[18] A. Roberts, *Cyber Threat Intelligence: The No-Nonsense Guide for CISOs and Security Managers*, 1ª ed. Berkeley, California, EEUU: Apress L. P, 2021.

[19] K. Bauker. (Marzo 2023) "*What is Cyber Threat Intelligence?*" [En línea]. Disponible en: https://www.crowdstrike.com/cybersecurity-101/threat-intelligence/ (Último acceso: 07-04-2024).

[20] MISP Project. (Abril 2024) "*MISP Communities and MISP Feeds.*" [En línea]. Disponible en: https://www.misp-project.org/communities/ (Último acceso: 08-04-2024).

[21] C. Neto, S. Hassine, 2xyo, I. A. Marugán, A. Henry Jard, y S. Bocahu, *Installation*, Marzo 2024, OpenCTI Documentation. [En línea]. Disponible en: https://docs.opencti.io/latest/deployment/installation/ (Último acceso: 08-04-2024).

[22] C. Void y C. Goettel. (Marzo 2021) "*newcontext-oss/opencti-terraform.*" [En línea]. Disponible en: https://github.com/newcontext-oss/opencti-terraform/

[23] ReadyElbow. (Octubre 2022) "*QinetiQ-Cyber-IntelligenceOpenCTI-Terraform.*" [En línea]. Disponible en: https://github.com/QinetiQ-Cyber-Intelligence/OpenCTI-Terraform

[24] J. Richard, "*Our design and development journey to Redis cluster support in OpenCTI*," Julio 2023. [En línea]. Disponible en: https://blog.filigran.io/our-design-and-development-journey-to-redis-cluster-support-in-opencti-af2e3e58cab

[25] J. Richard, "*OpenCTI platform performances*," Marzo 2021. [En línea]. Disponible en: https://blog.filigran.io/opencti-platform-performances-e3431b03f822

[26] S. Ortolani, C. Vandeplas, F. Takahashi, S. Clement, y A. Dulaunoy, *Download and Install MISP*, Diciembre 2023. [En línea]. Disponible en: https://www.misp-project.org/download/ (Último acceso: 08-04-2024).

[27] MISP Project. (Marzo 2024) "*MISP/misp-docker.*" [En línea]. Disponible en: https://github.com/MISP/misp-docker (Último acceso: 08-04-2024).

[28] Národní úřad pro kybernetickou a informační bezpečnost (NUKIB). (Abril 2024) "*NUKIB/misp.*" [En línea]. Disponible en: https://github.com/NUKIB/misp (Último acceso: 08-04-2024).

[29] Team MISP Project, *Extending MISP with Python modules*, Febrero 2023, MISP Threat Sharing. [En línea]. Disponible en: https://www.misp-project.org/misp-training/3.1-misp-modules.pdf (Último acceso: 08-04-2024).

[30] Amazon Web Services, *What is the AWS Serverless Application Model (AWS SAM)?*, 2024, Developer Guide. [En línea]. Disponible en: https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/what-is-sam.html (Último acceso: 29-04-2024).

[31] Amazon Web Services, *Managing resource permissions with AWS SAM connectors*, 2024, Developer Guide. [En línea]. Disponible en: https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/managing-permissions-connectors.html (Último acceso: 03-05-2024).

[32] Amazon Web Services, *Using the AWS SAM CLI*, 2024, Developer Guide. [En línea]. Disponible en: https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/using-sam-cli.html (Último acceso: 03-05-2024).

[33] Amazon Web Services, *What is EDA (Event-Driven Architecture)?*, 2024. [En línea]. Disponible en: https://aws.amazon.com/what-is/eda/ (Último acceso: 03-05-2024).

[34] Azure Architecture Center, *Event-driven architecture style*, Abril 2024, Microsoft Learn. [En línea]. Disponible en: https://learn.microsoft.com/en-us/azure/architecture/guide/architecture-styles/event-driven (Último acceso: 03-05-2024).

[35] European Union Agency for Cybersecurity (ENISA). "*Information Sharing and Analysis Centers (ISACs).*" Atenas, Grecia. [En línea]. Disponible en: https://www.enisa.europa.eu/topics/national-cyber-security-strategies/information-sharing (Último acceso: 06-05-2024).

[36] Center for Internet Security (CIS). (2024) "*RSS Syndication.*" East Greenbush, New York, EEUU. [En línea]. Disponible en: https://www.cisecurity.org/rss-syndication (Último acceso: 01-05-2024).

[37] Center for Internet Security (CIS). (2024) "*Multi-State Information Sharing and Analysis Center.*" East Greenbush, New York, EEUU. [En línea]. Disponible en: https://www.cisecurity.org/ms-isac (Último acceso: 06-05-2024).

[38] Center for Internet Security (CIS). (2024) "*Alert Level Information.*" East Greenbush, New York, EEUU. [En línea]. Disponible en: https://www.cisecurity.org/cybersecurity-threats/alert-level (Último acceso: 06-05-2024).

[39] Amazon Web Services, *What is AWS CloudFormation?*, 2024, Developer Guide. [En línea]. Disponible en: https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/Welcome.html (Último acceso: 06-05-2024).

[40] Amazon Web Services, *Working with stacks*, 2024, Developer Guide. [En línea]. Disponible en: https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/stacks.html (Último acceso: 06-05-2024).

[41] Amazon Web Services, *Template reference*, 2024, Developer Guide. [En línea]. Disponible en: https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/template-reference.html (Último acceso: 20-05-2024).

[42] Amazon Web Services, *Infrastructure As Code Provisioning Tool - AWS CloudFormation*, 2024. [En línea]. Disponible en: https://aws.amazon.com/cloudformation/ (Último acceso: 20-05-2024).

[43] Amazon Web Services, *Working with stacks*, 2024, Developer Guide. [En línea]. Disponible en: https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/stacks.html (Último acceso: 20-05-2024).

[44] Amazon Web Services, *Free Cloud Computing Services - AWS Free Tier*, 2024. [En línea]. Disponible en: https://aws.amazon.com/free/ (Último acceso: 10-06-2024).

[45] Grafana Labs, *Install Grafana*, Junio 2024, Grafana documentation. [En línea]. Disponible en: https://grafana.com/docs/grafana/latest/setup-grafana/installation/ (Último acceso: 10-06-2024).

[46] Grafana Labs, *Run Grafana Docker image*, Enero 2024, Grafana documentation. [En línea]. Disponible en: https://grafana.com/docs/grafana/latest/setup-grafana/installation/docker/ (Último acceso: 10-06-2024).

[47] Amazon Web Services, *Run commands on your Linux instance at launch*, 2024, Developer Guide. [En línea]. Disponible en: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/user-data.html (Último acceso: 07-05-2024).

[48] Amazon Web Services, *Create a key pair for your Amazon EC2 instance - Create a key pair using AWS CloudFormation*, 2024, Developer Guide. [En línea]. Disponible en: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/create-key-pairs.html#create-key-pair-cloudformation (Último acceso: 07-05-2024).

[49] LocalStack. (2024) "*LocalStack.*" Uetlibergstrasse 95, 8045 Zurich, Switzerland. [En línea]. Disponible en: https://www.localstack.cloud/ (Último acceso: 05-06-2024).

[50] LocalStack, *AWS Serverless Application Model (SAM)*, Mayo 2024, Docs. [En línea]. Disponible en: https://docs.localstack.cloud/user-guide/integrations/aws-sam/ (Último acceso: 05-06-2024).

[51] AWS CloudFormation. (Junio 2024) "*aws-cloudformation/cfn-lint.*" [En línea]. Disponible en: https://github.com/aws-cloudformation/cfn-lint

[52] Amazon Web Services, *AWS SAM CLI configuration file*, 2024, Developer Guide. [En línea]. Disponible en: https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/serverless-sam-cli-config.html (Último acceso: 06-06-2024).

[53] LocalStack, *LocalStack Desktop*, Mayo 2024, Docs. [En línea]. Disponible en: https://docs.localstack.cloud/user-guide/tools/localstack-desktop/ (Último acceso: 05-06-2024).

[54] Amazon Web Services, *Managing access keys for IAM users*, 2024, Developer Guide. [En línea]. Disponible en: https://docs.aws.amazon.com/IAM/latest/UserGuide/id_credentials_access-keys.html (Último acceso: 10-06-2024).

[55] Amazon Web Services, *Installing the AWS SAM CLI*, 2024, Developer Guide. [En línea]. Disponible en: https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/install-sam-cli.html (Último acceso: 08-06-2024).

[56] Grafana Labs, *Set up Grafana HTTPS for secure web traffic*, Octubre 2023, Grafana documentation. [En línea]. Disponible en: https://grafana.com/docs/grafana/latest/setup-grafana/set-up-https/ (Último acceso: 11-06-2024).

[57] Amazon Web Services, *Amazon RDS Instance Types*, 2024. [En línea]. Disponible en: https://aws.amazon.com/rds/instance-types/ (Último acceso: 11-06-2024).

[58] Amazon Web Services, *AWS Lambda Pricing*, 2024. [En línea]. Disponible en: https://aws.amazon.com/lambda/pricing/ (Último acceso: 11-06-2024).

[59] Amazon Web Services, *Amazon EventBridge Pricing*, 2024. [En línea]. Disponible en: https://aws.amazon.com/eventbridge/pricing/ (Último acceso: 11-06-2024).

[60] Amazon Web Services, *Amazon EC2 Instance Types*, 2024. [En línea]. Disponible en: https://aws.amazon.com/ec2/instance-types/ (Último acceso: 11-06-2024).

[61] Amazon Web Services, *Amazon EBS Pricing*, 2024. [En línea]. Disponible en: https://aws.amazon.com/ebs/pricing/ (Último acceso: 11-06-2024).

[62] Amazon Web Services, *Amazon EBS I/O characteristics and monitoring*, 2024, Developer Guide. [En línea]. Disponible en: https://docs.aws.amazon.com/ebs/latest/userguide/ebs-io-characteristics.html (Último acceso: 11-06-2024).

[63] Amazon Web Services, *SAM TIP TFM UOC EstimateEdit*, 2024, AWS Pricing Calculator. [En línea]. Disponible en: https://calculator.aws/#/estimate?id=ce977d0f9b1d9e16a00553e17f49ffe63d0a741b (Último acceso: 11-06-2024).

[64] Amazon Web Services, *Lambda function handler in Python*, 2024, Developer Guide. [En línea]. Disponible en: https://docs.aws.amazon.com/lambda/latest/dg/python-handler.html (Último acceso: 07-05-2024).

A lo largo de los sucesivos apéndices se proporcionan detalles respecto al código que implementa la TIP desarrollada en este proyecto, así como fragmentos útiles para obtener, por ejemplo, acceso remoto mediante el protocolo SSH a la instancia Amazon EC2.

Apéndice A

Código de la función AWS Lambda recolectora de datos

El Apéndice A contiene todo el código Python para obtener los datos del RSS del CIS e insertarlos en la base de datos. Los parámetros de configuración se han empleado como variables de entorno, que se definen desde CloudFormation.

Para indicar a la función Lambda el punto de entrada, se debe definir un controlador o *handler*. Para ello, simplemente hay que situar el código principal dentro de la función `def handler_name(event, context)` [64]. Se han definido también otras funciones adicionales para crear la tabla de PostgreSQL e insertar los datos en ella, dotando de mayor claridad al código.

La función Lambda se define mediante el recurso `AWS::Serverless::Function` en el *stack* de CloudFormation. Este debe incluir la definición de las variablas de entorno empleadas en el código, que son obtenidas (I) de los parámetros de entrada del *stack* definidos por el usuario, (II) de la creación del recurso de Amazon RDS y (III) de valores que se pueden parametrizar pero que se han codificado («*hardcodeado*») para no aumentar la complejidad del código.

Cabe destacar que el manejo de errores en el código es muy elemental, pues no forma parte de los objetivos de este proyecto el elaborar un código complejo y robusto. Tan solo se comprueba si existen los valores necesarios tras la petición web mediante cláusulas `if`. Queda como trabajo futuro hacer más resiliente el código en este aspecto.

```
1  import psycopg2
2  from urllib.request import Request, urlopen
3  import xml.etree.ElementTree as ET
4  import os
5
6  ENDPOINT=os.environ["ENDPOINT"]
7  PORT=os.environ["PORT"]
8  USER=os.environ["USER"]
9  PASSWORD=os.environ["PASSWORD"]
10 REGION=os.environ["REGION"]
11 DBNAME=os.environ["DBNAME"]
12
```

```
13 def create_table(conn):
14     with conn.cursor() as cursor:
15         cursor.execute("""CREATE TABLE IF NOT EXISTS cis_alert (id SERIAL
           ↪  PRIMARY KEY, date TIMESTAMP NOT NULL, level VARCHAR(25) NOT
           ↪  NULL, description TEXT)""")
16         conn.commit()
17 
18 def insert(conn, date, level, description):
19     with conn.cursor() as cursor:
20         cursor.execute("""INSERT INTO cis_alert (date, level, description)
           ↪  VALUES (%s, %s, %s)""", (date, level, description))
21         conn.commit()
22     print(f"Inserted {date}: {level}, {description}")
23 
24 def lambda_handler(event, context):
25     try:
26         conn = psycopg2.connect(host=ENDPOINT, port=PORT, database=DBNAME,
           ↪  user=USER, password=PASSWORD)
27         create_table(conn)
28 
29         request = Request("https://www.cisecurity.org/feed/alert")
30         request.add_header("User-Agent", "Mozilla/5.0 (Windows NT 10.0;
           ↪  Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
           ↪  Chrome/124.0.0.0 Safari/537.36 Edg/124.0.0.0",)
31         with urlopen(request) as response:
32             data = response.read()
33             root = ET.fromstring(data)
34             item = root.find("./channel/item")
35             if item is not None:
36                 title = item.find("title").text.replace("Threat Level - ",
                   ↪  "")
37                 description = item.find("description").text
38                 pubDate = item.find("pubDate").text
39                 if title and description and pubDate:
40                     insert(conn, pubDate, title, description)
41 
42     except Exception as e:
43         print("Database connection failed due to {}".format(e))
```

Código 4: Código de la función AWS Lambda recolectora de datos

Apéndice B

Código del *stack* de AWS CloudFormation completo

El Código 5 contiene la implementación del *stack* de CloudFormation, que aprovisiona todos los recursos necesarios para desplegar la propuesta desarrollada en este trabajo. La Figura 4.3 contiene una representación gráfica de los recursos aprovisionados en este *stack*. Cabe destacar que este *stack* se ha desarrollado para la región `Europa (Frankfurt) (eu-central-1)`, ya que el despliegue en LocalStack ha presentado problemas en la región de `Europa (España) (eu-south-2)`, más próxima geográficamente.

En el recurso `MyEC2SecurityGroup` se ha eliminado la dirección IP de las reglas por motivos de privacidad. Para replicar el *stack*, se deben descomentar los valores de las claves `CidrIp` y reemplazar la cadena `<MI_IP>` con la dirección o rangos deseados.

```
1  AWSTemplateFormatVersion: "2010-09-09"
2  Description: This stack creates a Lambda function triggered by a cron
   ↪  EventBridge rule to fetch CIS RSS data and upload it to RDS PostgreSQL.
   ↪  It also provisions an EC2 instance with Grafana for data visualization.
3  Transform: AWS::Serverless-2016-10-31
4
5  Parameters:
6    DatabaseUser:
7      Type: "String"
8      Default: "postgres"
9      Description: "Database admin account name"
10     MinLength: "5"
11     MaxLength: "16"
12     AllowedPattern: "[a-zA-Z][a-zA-Z0-9]*"
13     ConstraintDescription: "Name must begin with a letter and contain only
       ↪  alphanumeric characters."
14
15   DatabasePassword:
16     NoEcho: "true"
17     Type: "String"
```

```
    Description: "Database admin account password"
    MinLength: "6"
    MaxLength: "41"
    AllowedPattern: "[a-zA-Z0-9]*"
    ConstraintDescription: "Password must contain only alphanumeric
    ↪  characters."

  DatabaseName:
    Type: "String"
    Default: "postgres"
    Description: "Database name"
    MinLength: "1"
    MaxLength: "30"
    AllowedPattern: "[a-zA-Z][a-zA-Z0-9]*"
    ConstraintDescription: "Name must begin with a letter and contain only
    ↪  alphanumeric characters."

Resources:
  MyDBSecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupName: rds-sg
      GroupDescription: Security Group for PostgreSQL RDS instance
      SecurityGroupIngress:
        - IpProtocol: tcp
          FromPort: 5432
          ToPort: 5432
          CidrIp: 0.0.0.0/0

  MyPostgreSQL:
    Type: AWS::RDS::DBInstance
    Properties:
      Engine: "postgres"
      EngineVersion: "16.3"
      DBInstanceIdentifier: "TIPpostgreSQL"
      DBName: !Ref DatabaseName
      MasterUsername: !Ref DatabaseUser
      MasterUserPassword: !Ref DatabasePassword
      DBInstanceClass: "db.t3.micro"
      AllocatedStorage: "20"
      StorageType: "gp2"
      MultiAZ: false
      BackupRetentionPeriod: 0
      MonitoringInterval: 0
```

```
      PubliclyAccessible: true
      VPCSecurityGroups:
        - !GetAtt MyDBSecurityGroup.GroupId

  MyLambdaDependencies:
    Type: AWS::Serverless::LayerVersion
    Properties:
      LayerName: psycopg2
      Description: psycopg2 package for sam TIP TFM app
      ContentUri: dependencies/psycopg2_package.zip
      CompatibleRuntimes:
        - python3.12
      RetentionPolicy: Delete

  MyLambdaFunction:
    Type: AWS::Serverless::Function
    Properties:
      FunctionName: data-extractor
      Runtime: python3.12
      Handler: index.lambda_handler
      Timeout: 10
      Layers:
        - !Ref MyLambdaDependencies
      InlineCode: |
        import psycopg2
        from urllib.request import Request, urlopen
        import xml.etree.ElementTree as ET
        import os

        ENDPOINT=os.environ["ENDPOINT"]
        PORT=os.environ["PORT"]
        USER=os.environ["USER"]
        PASSWORD=os.environ["PASSWORD"]
        REGION=os.environ["REGION"]
        DBNAME=os.environ["DBNAME"]

        def create_table(conn):
            with conn.cursor() as cursor:
                cursor.execute("""
                CREATE TABLE IF NOT EXISTS cis_alert (
                    id SERIAL PRIMARY KEY,
                    date TIMESTAMP NOT NULL,
                    level VARCHAR(25) NOT NULL,
                    description TEXT
```

```
            )
            """)
            conn.commit()

        def insert(conn, date, level, description):
            with conn.cursor() as cursor:
                cursor.execute("""
                INSERT INTO cis_alert (date, level, description)
                VALUES (%s, %s, %s)
                """, (date, level, description))
                conn.commit()
            print(f"Inserted {date}: {level}, {description}")

        def lambda_handler(event, context):
            try:
                conn = psycopg2.connect(
                    host=ENDPOINT,
                    port=PORT,
                    database=DBNAME,
                    user=USER,
                    password=PASSWORD
                )

                create_table(conn)

                request = Request("https://www.cisecurity.org/feed/alert")
                request.add_header(
                    "User-Agent",
                    "Mozilla/5.0 (Windows NT 10.0; Win64; x64)
                    ↪ AppleWebKit/537.36 (KHTML, like Gecko)
                    ↪ Chrome/124.0.0.0 Safari/537.36 Edg/124.0.0.0",
                )
                with urlopen(request) as response:
                    data = response.read()
                    root = ET.fromstring(data)
                    item = root.find("./channel/item")
                    if item is not None:
                        title = item.find("title").text.replace("Threat
                        ↪ Level - ", "")
                        description = item.find("description").text
                        pubDate = item.find("pubDate").text
                        if title and description and pubDate:
                            insert(conn, pubDate, title, description)
```

```
            except Exception as e:
                print("Database connection failed due to {}".format(e))

    Environment:
      Variables:
        ENDPOINT: !GetAtt MyPostgreSQL.Endpoint.Address
        PORT: "5432"
        USER: !Ref DatabaseUser
        PASSWORD: !Ref DatabasePassword
        REGION: "eu-central-1"
        DBNAME: !Ref DatabaseName

MyScheduledRule:
  Type: AWS::Events::Rule
  Properties:
    Name: cis-rss-trigger-rule
    Description: Triggers Lambda function every Monday at 3 AM
    ScheduleExpression: "cron(0 3 ? * MON *)"
    State: ENABLED
    Targets:
      - Arn:
          Fn::GetAtt:
            - "MyLambdaFunction"
            - "Arn"
        Id: "TargetFunctionV1"

PermissionForEventsToInvokeLambda:
  Type: AWS::Lambda::Permission
  Properties:
    FunctionName: !Ref "MyLambdaFunction"
    Action: "lambda:InvokeFunction"
    Principal: "events.amazonaws.com"
    SourceArn:
      Fn::GetAtt:
        - "MyScheduledRule"
        - "Arn"

MyEC2SecurityGroup:
  Type: AWS::EC2::SecurityGroup
  Properties:
    GroupName: grafana-ec2-sg
    GroupDescription: Security group for EC2 instance
    SecurityGroupIngress:
      - IpProtocol: tcp
```

```
        FromPort: 22
        ToPort: 22
        CidrIp: # MI_IP
      - IpProtocol: tcp
        FromPort: 3000
        ToPort: 3000
        CidrIp: # MI_IP

MyKeyPair:
  Type: AWS::EC2::KeyPair
  Properties:
    KeyName: grafana-ec2-keys

MyEC2Instance:
  Type: AWS::EC2::Instance
  Properties:
    ImageId: ami-09e647bf7a368e505 # Amazon Linux 2023 amd64 eu-central-1
    AvailabilityZone:
      Fn::Select:
        - 0
        - Fn::GetAZs: !Ref 'AWS::Region'
    InstanceType: t2.micro
    KeyName: !Ref MyKeyPair
    SecurityGroupIds:
      - !Ref MyEC2SecurityGroup
    BlockDeviceMappings:
      - DeviceName: "/dev/sda1"
        Ebs:
          DeleteOnTermination: true
          VolumeSize: 10
    UserData:
      Fn::Base64: !Sub |
        #!/bin/bash
        yum update -y
        yum upgrade -y
        yum autoremove -y

        yum install -y docker
        service docker start
        groupadd docker
        usermod -a -G docker ec2-user
        newgrp docker

        docker pull grafana/grafana
```

```
          docker run -d -p 3000:3000 --name=grafana grafana/grafana
      Tags:
        - Key: Name
          Value: Grafana-EC2
Outputs:
  RDSInstanceURL:
    Description: The RDS PostgreSQL URL
    Value: !GetAtt MyPostgreSQL.Endpoint.Address
  EC2InstanceID:
    Description: The EC2 instance ID
    Value: !Ref MyEC2Instance
  EC2InstancePublicIP:
    Description: The public IP of the EC2 instance
    Value: !GetAtt MyEC2Instance.PublicIp
  GrafanaURL:
    Description: The URL to access Grafana
    Value: !Sub "http://${MyEC2Instance.PublicDnsName}:3000"
  EC2SecurityGroup:
    Description: The security group ID for the EC2 instance
    Value: !Ref MyEC2SecurityGroup
  LambdaFunctionArn:
    Description: The ARN of the Lambda function
    Value: !GetAtt MyLambdaFunction.Arn
  ScheduledRuleId:
    Description: The ID of the EventBridge rule
    Value: !Ref MyScheduledRule
```

Código 5: Código del *stack* de AWS CloudFormation

Apéndice C

Política de IAM para lanzar el *stack* de AWS CloudFormation

El Código 6 contiene la política de IAM que necesita ser asociada al usuario que ejecute la AWS SAM CLI para desplegar el *stack* del Apéndice B en AWS. En concreto, se asignan los siguientes permisos al usuario:

1. **CloudFormation**: permite al usuario crear, actualizar, describir, y eliminar *stacks* de CloudFormation.
2. **Amazon EC2**: permite la creación y manejo de instancias Amazon EC2, grupos de seguridad, y pares de llaves.
3. **Amazon RDS**: permite la creación y manejo de instancias Amazon RDS y grupos de seguridad relacionados.
4. **Lambda**: permite la creación, actualización, invocación y eliminación de funciones Lambda, así como la gestión de permisos.
5. **EventBridge**: permite la creación y manejo de reglas y *targets* de EventBridge.
6. **Amazon S3**: permite la creación y manejo de *buckets* y objetos en Amazon S3, necesarios para almacenar la plantilla de CloudFormation.
7. **CloudWatch Logs**: permite la creación y manejo de grupos de logs y streams, necesarios para la salida de logs de Lambda.
8. **IAM**: permite el uso de `PassRole` para que los roles IAM puedan ser utilizados por los recursos creados, especialmente Lambda y Amazon RDS.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
```

```
            "Effect": "Allow",
            "Action": [
                "cloudformation:CreateStack",
                "cloudformation:UpdateStack",
                "cloudformation:DescribeStacks",
                "cloudformation:DescribeStackEvents",
                "cloudformation:DeleteStack",
                "cloudformation:DescribeStackResources",
                "cloudformation:GetTemplate",
                "cloudformation:ValidateTemplate",
                "ec2:CreateSecurityGroup",
                "ec2:AuthorizeSecurityGroupIngress",
                "ec2:CreateTags",
                "ec2:DescribeInstances",
                "ec2:RunInstances",
                "ec2:TerminateInstances",
                "ec2:DescribeSecurityGroups",
                "ec2:DeleteSecurityGroup",
                "ec2:DescribeKeyPairs",
                "ec2:CreateKeyPair",
                "ec2:DeleteKeyPair",
                "ec2:DescribeImages",
                "ec2:DescribeRegions",
                "ec2:DescribeAvailabilityZones",
                "rds:CreateDBInstance",
                "rds:DeleteDBInstance",
                "rds:DescribeDBInstances",
                "rds:ModifyDBInstance",
                "rds:CreateDBSecurityGroup",
                "rds:DeleteDBSecurityGroup",
                "rds:AuthorizeDBSecurityGroupIngress",
                "rds:DescribeDBSecurityGroups",
                "lambda:CreateFunction",
                "lambda:DeleteFunction",
                "lambda:InvokeFunction",
                "lambda:CreateEventSourceMapping",
                "lambda:DeleteEventSourceMapping",
                "lambda:UpdateFunctionCode",
                "lambda:UpdateFunctionConfiguration",
                "lambda:GetFunction",
                "lambda:ListFunctions",
                "lambda:AddPermission",
                "lambda:RemovePermission",
                "events:PutRule",
```

```
                "events:DeleteRule",
                "events:DescribeRule",
                "events:EnableRule",
                "events:DisableRule",
                "events:PutTargets",
                "events:RemoveTargets",
                "s3:CreateBucket",
                "s3:DeleteBucket",
                "s3:PutObject",
                "s3:GetObject",
                "s3:DeleteObject",
                "s3:ListBucket",
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "logs:DescribeLogGroups",
                "logs:DescribeLogStreams",
                "logs:DeleteLogGroup",
                "iam:PassRole"
            ],
            "Resource": "*"
        }
    ]
}
```

Código 6: Código de la política de IAM

Apéndice D

Obtención de la clave privada para conexión SSH a Amazon Elastic Compute Cloud

En caso de desear establecer conexión remota por SSH con la instancia Amazon EC2, hay que obtener la clave privada que se crea al desplegar el *stack*. Como se menciona en la Sección 4.1.2, esta se almacena en el *Parameter Store* de Systems Manager. Para recuperarla, se puede acceder a Systems Manager a través de la consola, o ejecutar el Código 7 desde la AWS CLI [48].

```
1 ubuntu@ubuntu-tfm:~$ aws ec2 describe-key-pairs --key-names grafana-ec2-keys
   ↪  --query KeyPairs[*].KeyPairId --output text
2 key-084c551d66a57e5b7
3 ubuntu@ubuntu-tfm:~$ aws ssm get-parameter --name
   ↪  /ec2/keypair/key-084c551d66a57e5b7 --with-decryption --query
   ↪  Parameter.Value --output text > grafana-ec2-keys.pem
4 ubuntu@ubuntu-tfm:~$ chmod 400 grafana-ec2-keys.pem
```

Código 7: Obtención de la clave privada para SSH

Apéndice E

Consultas SQL de los paneles de Grafana

En la herramienta de visualización Grafana se han definido dos paneles diferentes para mostrar el indicador escogido, disponibles en la Figura 4.8.

El primero, con forma de manómetro o velocímetro, es de tipo *Gauge* dentro de Grafana. Para que se muestre el semicírculo de forma precisa, se han configurado el valor mínimo 1 y el máximo 5. Además, se ha configurado la escala de colores mediante umbrales, asignando el verde como base, el azul al valor 2, el amarillo al 3, el naranja al 4 y el rojo al 5. Cabe destacar que este tipo de gráfico únicamente puede representar valores numéricos, por lo que se ha traducido cada posible valor textual a su equivalente numérico como muestra el Código 8.

```sql
SELECT
  CASE
    WHEN level = 'LOW' THEN 1
    WHEN level = 'GUARDED' THEN 2
    WHEN level = 'ELEVATED' THEN 3
    WHEN level = 'HIGH' THEN 4
    WHEN level = 'SEVERE' THEN 5
    ELSE 0
  END AS level_numeric
FROM cis_alert
ORDER BY date DESC
LIMIT 1
```

Código 8: Consulta SQL para el velocímetro en Grafana

En el segundo, un panel de tipo *Time series*, también se han configurado el valor mínimo, el máximo y los umbrales para los colores. El eje de ordenadas representa valores numéricos, por lo que se ha relacionado también el valor en texto con cada indicador numérico. Respecto a la fecha, se debe seleccionar como `"time"` para que Grafana sea capaz de dibujar el gráfico temporal correctamente, así como aplicar el filtro `$__timeFilter()` en la cláusula `WHERE`

para poder utilizar el filtro de rangos temporales de la interfaz de Grafana. La consulta SQL completa se detalla en el Código 9.

```sql
SELECT
  date AS "time",
  CASE
    WHEN level = 'LOW' THEN 1
    WHEN level = 'GUARDED' THEN 2
    WHEN level = 'ELEVATED' THEN 3
    WHEN level = 'HIGH' THEN 4
    WHEN level = 'SEVERE' THEN 5
    ELSE 0
  END AS level_numeric
FROM cis_alert
WHERE $__timeFilter(date)
ORDER BY date ASC
```

Código 9: Consulta SQL para la serie temporal en Grafana