

universidad de león



Escuela de Ingenierías I.I.

Industrial, Informática y Aeroespacial

MÁSTER UNIVERSITARIO EN
INVESTIGACIÓN EN CIBERSEGURIDAD

Trabajo de Fin de Máster

# IMPLEMENTACIÓN DEL PARADIGMA ZERO TRUST EN ENTORNOS CLOUD

**Autor: Alejandro Riera González**

**Tutor: Francisco Javier Rodríguez Lera**

(Julio, 2026)

**UNIVERSIDAD DE LEÓN**
**Escuela de Ingenierías I.I.**
**MÁSTER UNIVERSITARIO EN INVESTIGACIÓN EN CIBERSEGURIDAD**
**Trabajo de Fin de Máster**

**ALUMNO**: Alejandro Riera González

**TUTOR**: Francisco Javier Rodríguez Lera

**TÍTULO**: Implementación del paradigma Zero Trust en entornos cloud

**CONVOCATORIA**: Julio, 2026

**RESUMEN**:
Este Trabajo de Fin de Máster analiza la adopción del paradigma *Zero Trust* en entornos *cloud*, partiendo del estudio de sus principios generales y de su aplicación en infraestructuras modernas. Para ello, se diseña y despliega una aplicación base en AWS, representativa de una arquitectura *cloud* funcional, sobre la que se identifican sus principales puntos de mejora desde el punto de vista de la seguridad. A partir de este análisis, se realiza una migración incremental incorporando distintos principios *Zero Trust*, como la reducción de confianza implícita, el uso de identidades más específicas, la segmentación interna y la mejora de la monitorización. Finalmente, se comparan ambas arquitecturas mediante pruebas funcionales, de seguridad, rendimiento y coste, y se realiza una discusión objetiva de los resultados obtenidos.

**Palabras clave**: *Zero Trust*, ciberseguridad, arquitectura, *cloud*, AWS

| **Firma del alumno**: | **VºBº Tutor:** |
| --- | --- |
| | |

(July, 2026)

**UNIVERSITY OF LEÓN**
**School of Industrial, Computer and Aerospace Engineering**
**MASTER'S DEGREE IN CYBERSECURITY RESEARCH**
**Master's Thesis**

**STUDENT**: Alejandro Riera González

**SUPERVISOR**: Francisco Javier Rodríguez Lera

**TITLE**: Implementation of the Zero Trust paradigm in cloud environments

**CALL**: July, 2026

**ABSTRACT**:
This Master's Thesis analyses the adoption of the Zero Trust paradigm in cloud environments, beginning with a study of its general principles and its application in modern infrastructures. To this end, a base application is designed and deployed on AWS, representing a functional cloud architecture, against which its main areas for improvement from a security perspective are identified. Based on this analysis, an incremental migration is carried out, incorporating various Zero Trust principles, such as reducing implicit trust, using more specific identities, internal segmentation and improved monitoring. Finally, both architectures are compared through functional, security, performance and cost tests, and an objective discussion of the results obtained is presented.

**Keywords**: Zero Trust, cybersecurity, architecture, cloud, AWS

| **Student's signature**: | **Supervisor's approval:** |
| --- | --- |
| | |

# Índice general

**Índice de figuras** IV

**Índice de tablas** VII

**Glosario de términos** IX

**Introducción** 1

**1. Estudio del problema** 4
- 1.1. El contexto del problema . . . 4
- 1.2. El estado de la cuestión . . . 5
  - 1.2.1. De la seguridad perimetral al *Zero Trust* . . . 6
  - 1.2.2. Arquitectura de referencia y componentes lógicos . . . 7
  - 1.2.3. Desafíos de implementación en *cloud* . . . 10
  - 1.2.4. Tecnologías emergentes y orquestación de la arquitectura . . . 12
  - 1.2.5. Ejes de investigación dominantes en ZTA . . . 14
  - 1.2.6. Síntesis del estado de la cuestión . . . 18
- 1.3. La definición del problema . . . 19
- 1.4. Normativa . . . 21
  - 1.4.1. Estándares y marcos internacionales . . . 21
  - 1.4.2. Marco europeo . . . 22
  - 1.4.3. Normativa y legislación española . . . 22

**2. Gestión de proyecto *software*** 24
- 2.1. Alcance del proyecto . . . 24
  - 2.1.1. Definición del proyecto . . . 24
  - 2.1.2. Estimación de tareas y recursos . . . 24
  - 2.1.3. Presupuesto . . . 25
- 2.2. Plan de trabajo . . . 29

2.2.1. Identificación y estimación de tareas . . . 29
2.2.2. Planificación de tareas . . . 30
2.3. Gestión de riesgos . . . 32

**3. Solución base** **34**
3.1. Descripción de la solución . . . 34
3.2. El proceso de desarrollo . . . 35
3.2.1. Análisis . . . 35
3.2.2. Diseño . . . 46
3.2.3. Implementación . . . 51
3.2.4. Pruebas . . . 54
3.3. El producto del desarrollo . . . 57
3.3.1. Herramienta desarrollada . . . 57
3.3.2. Diseño final . . . 57
3.3.3. Funcionalidad de la aplicación . . . 58
3.3.4. Despliegue . . . 64

**4. Implementación de *Zero Trust*** **69**
4.1. Descripción de la migración . . . 69
4.2. El proceso de desarrollo . . . 70
4.2.1. Análisis . . . 70
4.2.2. Diseño . . . 73
4.2.3. Implementación . . . 75
4.3. Resultado . . . 92
4.3.1. Diseño final . . . 92
4.3.2. Estado final . . . 93

**5. Evaluación** **101**
5.1. Proceso de evaluación . . . 101
5.2. Forma de evaluación . . . 102
5.3. Casos de prueba . . . 105

**6. Análisis y discusión de resultados** **121**
6.1. Análisis . . . 121
6.1.1. Funcionalidad . . . 121
6.1.2. Seguridad . . . 122
6.1.3. Rendimiento . . . 123
6.1.4. Complejidad, despliegue y coste . . . 124

6.2. Discusión final . . . . . 125
6.3. Limitaciones de la aproximación realizada . . . . . 127

**Conclusión** **128**

**Lista de referencias** **131**

**A. Código fuente del trabajo** **134**

**B. Uso de herramientas de inteligencia artificial** **135**

**C. Inventario tecnológico final** **136**

# Índice de figuras

1.1. Evolución anual del número de publicaciones relacionadas con ZTA entre 2016 y 2025. Datos obtenidos de Web of Science (consulta realizada el 21 de enero de 2026). Fuente: elaboración propia. . . . . 6
1.2. Arquitectura lógica de referencia de ZT, mostrando los componentes de decisión, ejecución y las fuentes de contexto que intervienen en la evaluación continua de acceso. Fuente: NIST SP 800-207 [22]. . . . . 8
1.3. Entradas del *Trust Algorithm* en una arquitectura Zero Trust, incluyendo las variables que afectan a la toma de decisión. Fuente: NIST SP 800-207 [22]. . . . . 9
1.4. Taxonomía de los principales desafíos en la adopción e implementación de ZTA. Fuente: Zero Trust Architecture: A Systematic Literature Review [13]. . . . . 13
1.5. Arquitectura del sistema DR-TBAC para control de acceso dinámico basado en Zero Trust en entornos *cloud*. Fuente: Zero-trust Based Dynamic Access Control for Cloud Computing [28]. . . . . 15

2.1. Gráfico de barras del coste real de infraestructura *cloud* en AWS. Fuente: captura propia. . . . . 27
2.2. Planificación de tareas del proyecto mediante diagrama de Gantt. Fuente: elaboración propia. . . . . 31

3.1. Diagrama SysML de requisitos no funcionales de la aplicación base. Bloque de infraestructura, arquitectura y despliegue. Fuente: elaboración propia. . . . . 40
3.2. Diagrama SysML de requisitos no funcionales de la aplicación base. Bloque de seguridad, acceso, operación y calidad. Fuente: elaboración propia. . . . . 41
3.3. Diagrama UML de casos de uso de la aplicación base. Fuente: elaboración propia. . . . . 44

3.4. Diagrama UML de secuencia de la aplicación base. Fuente: elaboración propia. . . . . 45
3.5. Diagrama de arquitectura *cloud* de la aplicación base. Fuente: elaboración propia. . . . . 48
3.6. Diagrama entidad relación del modelo de persistencia de la aplicación base. Fuente: elaboración propia. . . . . 49
3.7. Diagrama definitivo de arquitectura *cloud* de la aplicación base. Fuente: elaboración propia. . . . . 58
3.8. *Login* y registro de la aplicación base. Fuente: captura propia. . . . . 59
3.9. Formulario de creación de evento y listado de eventos. Fuente: captura propia. . . . . 60
3.10. Detalle de evento con imagen publicada. Previo al procesamiento del servicio *batch*. Fuente: captura propia. . . . . 61
3.11. *Logout* y consulta de los procesos del servicio *batch*. Fuente: captura propia. . . . . 62
3.12. Detalle de evento con imagen publicada. Posterior al procesamiento del servicio *batch*. Fuente: captura propia. . . . . 63
3.13. Salida de terminal del estado de la infraestructura base desplegada. Fuente: captura propia. . . . . 65
3.14. Clúster EKS base en la consola de AWS. Fuente: captura propia. . . . . 66
3.15. Nodo del clúster EKS base en la consola de AWS. Fuente: captura propia. . . . . 66
3.16. *Pods* del nodo dentro del clúster EKS base en la consola de AWS. Fuente: captura propia. . . . . 66
3.17. VPC del despliegue de la aplicación base en la consola de AWS. Fuente: captura propia. . . . . 67
3.18. ALB del despliegue de la aplicación base en la consola de AWS. Fuente: captura propia. . . . . 67
3.19. RDS del despliegue de la aplicación base en la consola de AWS. Fuente: captura propia. . . . . 68
3.20. *Bucket* S3 del despliegue de la aplicación base en la consola de AWS. Fuente: captura propia. . . . . 68
3.21. ECR del despliegue de la aplicación base en la consola de AWS. Fuente: captura propia. . . . . 68

4.1. Diagrama de arquitectura *cloud* de la aplicación aplicando *Zero Trust*. Fuente: elaboración propia. . . . . 74

4.2. Diagrama definitivo de arquitectura *cloud* de la aplicación aplicando *Zero Trust*. Fuente: elaboración propia. . . . . 93
4.3. Estado de *pods*, servicios e *Ingress* tras la migración a *Zero Trust* consultado desde el bastión mediante AWS Systems Manager. Fuente: captura propia. . . . . 96
4.4. Instancia EC2 privada utilizada como bastión para la administración del clúster. Fuente: captura propia. . . . . 96
4.5. *Bucket* S3 utilizado para transferir artefactos Kubernetes al bastión privado. Fuente: captura propia. . . . . 97
4.6. Historial de sesiones de administración mediante AWS Systems Manager Session Manager. Fuente: captura propia. . . . . 97
4.7. Configuración privada del punto de enlace del servidor de la API de Amazon EKS. Fuente: captura propia. . . . . 98
4.8. Fallo esperado del acceso local directo al *endpoint* de Amazon EKS. Fuente: captura propia. . . . . 98
4.9. ServiceAccounts de los *workloads* asociados a roles IAM mediante IRSA. Fuente: captura propia. . . . . 99
4.10. Ausencia de credenciales S3 estáticas en el *pod* de la API. Fuente: captura propia. . . . . 99
4.11. NetworkPolicies aplicadas. Fuente: captura propia. . . . . 99
4.12. *Trail* de CloudTrail configurado para la auditoría de eventos de gestión de AWS. Fuente: captura propia. . . . . 100
4.13. *Logs* de auditoría de CloudTrail almacenados en el *bucket* S3 dedicado. Fuente: captura propia. . . . . 100

5.1. Distribución de tiempos de respuesta en las pruebas de rendimiento de la solución base. Fuente: elaboración propia. . . . . 112
5.2. Distribución de tiempos de respuesta en las pruebas de rendimiento de la solución *Zero Trust*. Fuente: elaboración propia. . . . . 112
5.3. R1: *endpoint* simple (base). Fuente: elaboración propia. . . . . 113
5.4. R1: *endpoint* simple (zt). Fuente: elaboración propia. . . . . 113
5.5. R2: autenticación de usuario (base). Fuente: elaboración propia. . . . 114
5.6. R2: autenticación de usuario (zt). Fuente: elaboración propia. . . . . 114
5.7. R3: subida de imagen (base). Fuente: elaboración propia. . . . . 115
5.8. R3: subida de imagen (zt). Fuente: elaboración propia. . . . . 115

# Índice de tablas

1.1. Síntesis de tecnologías y líneas de trabajo en arquitecturas *Zero Trust* en entornos *cloud* . . . . . 18

2.1. Coste real de infraestructura *cloud* en AWS . . . . . 26
2.2. Presupuesto total del proyecto . . . . . 29
2.3. Identificación y estimación de tareas del proyecto . . . . . 29
2.4. Riesgos identificados del proyecto . . . . . 32

3.1. Requisitos funcionales del sistema base . . . . . 35
3.2. Requisitos no funcionales del sistema base . . . . . 37
3.3. Casos de uso del sistema base . . . . . 43
3.4. Herramientas y tecnologías utilizadas en la implementación de la aplicación base . . . . . 52
3.5. Fases principales de implementación de la aplicación base . . . . . 53
3.6. Relación entre casos de uso y pruebas automáticas . . . . . 56

4.1. Análisis de brechas arquitectónicas para la migración hacia *Zero Trust* 71
4.2. Migración del modelo de acceso a S3 hacia IRSA . . . . . 76
4.3. Políticas de red aplicadas para la segmentación interna . . . . . 80
4.4. Controles de endurecimiento aplicados a los *workloads* . . . . . 84
4.5. Evolución del acceso al plano de control de Amazon EKS . . . . . 89
4.6. Mecanismos de trazabilidad orientados a *Zero Trust* . . . . . 91
4.7. Trazabilidad entre brechas identificadas y resultado final de la migración 94

5.1. Pruebas funcionales . . . . . 102
5.2. Pruebas de seguridad . . . . . 103
5.3. Pruebas de rendimiento mínimo . . . . . 103
5.4. Métricas de complejidad, despliegue y coste . . . . . 104
5.5. Resultados de pruebas funcionales . . . . . 105
5.6. Resultados de pruebas de seguridad . . . . . 107

5.7. Resultados de pruebas de rendimiento mínimo . . . . . 110
5.8. Resultados de métricas de complejidad, despliegue y coste . . . . . 116
5.9. Estimación teórica de coste mensual de la solución base en AWS Pricing Calculator . . . . . 116
5.10. Estimación teórica de coste mensual de la solución *Zero Trust* en AWS Pricing Calculator . . . . . 118

# Glosario de términos

Catálogo de términos específicos del contexto del trabajo.

**ABAC** : Attribute-Based Access Control. Modelo de seguridad que autoriza o deniega el acceso a recursos mediante la evaluación de reglas combinatorias aplicadas a un conjunto de características.

**API** : Application Programming Interface. Mecanismo que permite a dos componentes de *software* comunicarse entre sí mediante un conjunto de definiciones y protocolos.

**blockchain** : Cadena de bloques. Tecnología de registro digital descentralizado y sincronizado que permite almacenar información de manera inmutable y segura sin necesidad de intermediarios.

**BYOD** : Bring Your Own Device. Política donde los usuarios utilizan sus propios dispositivos para realizar las tareas necesarias en un entorno corporativo o educativo.

**CDM** : Continuous Diagnostics and Mitigation. Ecosistema de herramientas y procesos diseñados para proporcionar visibilidad en tiempo real sobre la postura de seguridad de los activos de una organización.

**COOP** : Continuity Of Operations Plan. Conjunto de procedimientos diseñados para garantizar que las funciones esenciales de una organización continúen operando durante y después de una interrupción.

**ciberseguridad** : Protección de los sistemas informáticos y de sus redes de comunicaciones, con el objetivo de mantener segura la información que procesan.

**cloud** : Modelo para permitir el acceso a la red a un conjunto escalable y elástico de recursos compartidos.

**contenedor** : Unidad de *software* estandarizada y ejecutable que empaqueta una aplicación con todo lo necesario para que funcione y la aísla del entorno.

**DR-TBAC** : Dynamic Reinforcement Trust-Based Access Control. Modelo de control de acceso basado en confianza que ajusta dinámicamente los permisos en tiempo real según el nivel de confianza de usuarios o entidades, calculado a partir de su comportamiento y contexto mediante aprendizaje por refuerzo.

**HWAM** : Hardware Assets Management. Proceso de control y monitorización de todos los dispositivos físicos en la red.

**IAM** : Identity and Access Management. Conjunto de procesos, políticas y tecnologías que garantiza que la identidad correcta tenga el acceso adecuado a los recursos específicos, en el momento necesario y por las razones válidas.

**IDS** : Intrusion Detection System. Herramienta diseñada para monitorizar el tráfico de red o la actividad de un sistema en busca de comportamientos sospechosos, vulnerabilidades o violaciones de las políticas de seguridad.

**IPS** : Intrusion Prevention System. Considerado una evolución del IDS, ya que aplica una capa de seguridad activa, es decir, es capaz de intervenir los comportamientos o violaciones de políticas de seguridad en tiempo real.

**IRSA** : IAM Roles for Service Accounts. Funcionalidad nativa de AWS que permite asociar roles de seguridad de AWS IAM directamente a cuentas de servicio de Kubernetes. Esto permite que los componentes (*pods*) obtengan permisos específicos y temporales para acceder a recursos de AWS (como S3 o DynamoDB) bajo el principio de menor privilegio, eliminando la necesidad de gestionar credenciales estáticas.

**JWT** : JSON Web Token. Estándar que define una forma compacta y autónoma de transmitir información de forma segura entre partes como un objeto JSON. Sirven para establecer sesiones y pueden aplicar tanto encriptación simétrica como asimétrica.

**legacy system** : Sistema heredado. Artefacto, aplicación o infraestructura tecnológica que ha quedado obsoleta en términos de arquitectura, lenguaje de programación o soporte técnico, pero que continúa en uso porque sigue cumpliendo una función de negocio crítica.

**logging** : Proceso de rastrear y grabar eventos, actividades o información de estado que ocurren dentro de un sistema informático o aplicación.

**MFA** : Multi-Factor Authentication. Método de seguridad que requiere que el usuario proporcione dos o más pruebas independientes para verificar su identidad.

**mTLS** : Mutual Transport Layer Security. Extensión del protocolo TLS estándar que garantiza una autenticación bidireccional, exigiendo que tanto el servidor como el cliente presenten y verifiquen sus certificados digitales.

**multi-tenant** : Arquitectura de *software* donde una única instancia de una aplicación y su infraestructura sirve a múltiples clientes, manteniendo sus datos y configuraciones lógicamente separados y seguros.

**NIST** : National Institute of Standards and Technology. Agencia del Departamento de Comercio de Estados Unidos que desarrolla estándares y mejores prácticas.

**PA** : Policy Administrator. Componente encargado de establecer o cerrar la vía de comunicación entre un sujeto y un recurso a través de los PEPs. Conforma junto al PE el PDP.

**PaC** : Policy as Code. Enfoque mediante el cual las políticas, que tradicionalmente se redactaban en forma de documentos, ahora se codifican en formatos legibles y ejecutables por máquinas.

**PDP** : Policy Decision Point. "Cerebro" del sistema Zero Trust. Es el componente lógico responsable de decidir si se otorga o no acceso a un recurso específico.

**PE** : Policy Engine. Componente responsable de la decisión final de conceder el acceso a un recurso para un sujeto determinado. Obtiene la decisión aplicando el TA. Conforma junto al PA el PDP.

**pentesting** : Pruebas de penetración. Conjunto de técnicas orientadas a evaluar la seguridad de un sistema mediante la simulación controlada de ataques, con el objetivo de identificar vulnerabilidades explotables.

**PEP** : Policy Enforcement Point. "Guardián" o ejecutor físico/lógico. Es el componente que se encarga de habilitar, supervisar y, finalmente, terminar las conexiones entre un sujeto y un recurso.

**PKI** : Public Key Infrastructure. Sistema responsable de generar y registrar los certificados emitidos por la organización a los recursos, los sujetos, los servicios y las aplicaciones.

**plano de control** : Infraestructura dedicada exclusivamente a la autenticación y toma de decisiones. Es donde reside el PA y PE, es decir, el PDP.

**plano de datos** : Canal por el cual se transmiten los datos reales del usuario. Es donde residen los PEPs, activos y recursos de la organización.

**SaaS** : Software as a Service. Modelo de entrega de *software* donde las aplicaciones están alojadas en la nube y se accede a través de Internet.

**SDP** : Software-Defined Perimeter. Enfoque de seguridad que oculta la infraestructura conectada a Internet, creando un "perímetro invisible" que solo se abre para usuarios y dispositivos autorizados.

**service mesh** : Capa de infraestructura dedicada dentro de una aplicación de *software* que gestiona la comunicación entre servicios.

**sidecar** : Patrón de diseño donde un componente auxiliar se ejecuta junto a una aplicación principal, compartiendo su ciclo de vida y recursos para añadir funcionalidades como *logs*, seguridad o gestión de credenciales, sin modificar el código de la aplicación original.

**SIEM** : Security Information and Event Management. Sistema que recoge información basada en la seguridad para su posterior análisis.

**SPIFFE** : Secure Production Identity Framework for Everyone. Conjunto de estándares de código abierto para identificar de forma segura sistemas de *software* en entornos dinámicos y heterogéneos.

**SWAM** : Software Asset Management. Proceso de gestión y control de las aplicaciones y programas instalados en los activos.

**TA** : Trust Algorithm. Proceso lógico que pondera varias señales de contexto para determinar la puntuación de confianza de una solicitud. Es el componente del PE que dicta la decisión final de acceso.

**VPC** : Virtual Private Cloud. Red virtual lógicamente aislada y segura, alojada dentro de la infraestructura de un proveedor de nube pública.

**VPN** : Virtual Private Network. Tecnología que crea un "túnel" cifrado y seguro sobre una red pública para conectar a un usuario con una red privada.

**ZT** : Zero Trust. Conjunto de conceptos e ideas diseñados para minimizar la inseguridad mediante la aplicación de decisiones de acceso precisas y de mínimos privilegios por solicitud, tanto en los sistemas de información como en los servicios ante una red que se considera comprometida.

**ZTA** : Zero Trust Architecture. Plan de ciberseguridad de una organización que utiliza los conceptos de confianza cero y engloba las relaciones de los componentes, la planificación del flujo de trabajo y las políticas de acceso.

**ZTNA** : Zero Trust Network Access. Sucesor de la VPN que proporciona acceso granular solo a aplicaciones específicas (no a la red) tras verificar identidad y contexto en cada sesión.

# Introducción

## Planteamiento del problema

La computación en la nube se ha consolidado como un modelo habitual para desplegar aplicaciones modernas, ya que permite utilizar infraestructura escalable, servicios gestionados y arquitecturas distribuidas basadas en contenedores, API y almacenamiento externo. Sin embargo, este tipo de entornos también introduce nuevos retos de seguridad, al desaparecer el perímetro tradicional claramente definido y aumentar el número de componentes, identidades, comunicaciones internas y servicios expuestos.

En este contexto, los modelos de seguridad basados únicamente en proteger la red interna resultan insuficientes. Una aplicación *cloud* puede estar correctamente desplegada y ser funcional, pero seguir manteniendo elementos de confianza implícita, como permisos demasiado amplios, credenciales persistentes, comunicaciones internas sin segmentación o planos de administración accesibles desde el exterior. Estos aspectos pueden facilitar el movimiento lateral, el abuso de identidades o la exposición innecesaria de recursos.

El paradigma *Zero Trust* propone abordar este problema eliminando la confianza por defecto y aplicando principios como verificación explícita, mínimo privilegio, segmentación, monitorización continua y control granular del acceso. No obstante, su adopción en entornos *cloud* no es ni directa ni sencilla, ya que implica tomar decisiones técnicas, operativas y económicas sobre la arquitectura existente respecto a un modelo arquitectónico relativamente reciente y todavía en evolución.

Por ello, este trabajo plantea el diseño e implementación de una arquitectura *cloud* base y su posterior migración incremental hacia una solución más alineada con *Zero Trust*. A partir de esta comparación, se busca analizar qué mejoras aporta este enfoque, qué impacto tiene sobre la funcionalidad, seguridad, rendimiento, complejidad y coste, y qué dificultades aparecen durante su aplicación práctica.

# Objetivos

La realización de este trabajo tiene varios objetivos relacionados con el *Zero Trust* y la computación en la nube. Tenemos un objetivo principal, que, a su vez, necesita de varios objetivos específicos para llevar a cabo su realización.

El **objetivo principal** es analizar el impacto técnico y económico de la adopción del paradigma *Zero Trust* en aplicaciones del ámbito *cloud*.

Para lograrlo, disponemos de una serie de **objetivos específicos** para conseguir lograr el principal, y estos son:

- $O_1$: Analizar los principios fundamentales del *Zero Trust*, así como sus componentes arquitectónicos principales y su aplicación en entornos de computación en la nube.
- $O_2$: Diseñar una arquitectura *cloud* de referencia sin *Zero Trust*, representativa de una aplicación empresarial sencilla.
- $O_3$: Definir y diseñar una arquitectura equivalente que incorpore los principios del modelo *Zero Trust*, incluyendo mecanismos de identidad, control de acceso, segmentación, verificación continua y monitorización.
- $O_4$: Implementar ambas arquitecturas en un entorno *cloud*, utilizando servicios de la nube que permitan reproducir los mecanismos de seguridad y control necesarios.
- $O_5$: Identificar los principales desafíos técnicos y operativos asociados a la adopción de *Zero Trust* en aplicaciones *cloud* a partir de arquitecturas no-ZT, así como sus implicaciones para organizaciones que deseen adoptar este paradigma.
- $O_6$: Evaluar y comparar los costes asociados al paso de migrar la arquitectura básica hacia la arquitectura ZT.

# Metodología

Para desarrollar este trabajo se ha seguido una metodología secuencial por fases, combinando investigación, diseño, implementación y evaluación. Esta aproximación permite partir de una base teórica sobre el paradigma *Zero Trust*, construir una arquitectura *cloud* de referencia y analizar posteriormente el impacto de su migración hacia una solución más alineada con dicho modelo.

En primer lugar, se realizó el estudio del problema y del estado de la cuestión, revisando los principios de *Zero Trust*, sus componentes principales, sus desafíos de adopción y la normativa relacionada.

A continuación, se diseñó e implementó una aplicación base desplegada en una infraestructura *cloud*. Sobre esta solución base se llevó a cabo una primera evaluación y un análisis de brechas, identificando los principales puntos de confianza implícita. A partir de estos resultados, se diseñó e implementó la migración hacia una arquitectura *Zero Trust*.

Finalmente, se evaluó la solución migrada mediante la misma batería de pruebas aplicada a la solución base, permitiendo comparar ambos entornos en términos de funcionalidad, seguridad, rendimiento, complejidad, despliegue y coste. Durante todo el desarrollo se mantuvieron reuniones periódicas con el tutor, para revisar avances, resolver dudas y validar las decisiones adoptadas.

# Estructura del trabajo

El trabajo se organiza en seis capítulos principales, además de la presente introducción y el capítulo final de conclusiones.

El Capítulo 1 presenta el estudio del problema, contextualizando la evolución desde modelos de seguridad perimetral hacia el paradigma *Zero Trust*, revisando el estado de la cuestión y definiendo el problema abordado.

El Capítulo 2 recoge la gestión del proyecto *software*, incluyendo el alcance, la estimación de tareas y recursos, el presupuesto, la planificación y la gestión de riesgos.

El Capítulo 3 describe la solución base desarrollada, detallando su análisis, diseño, implementación, pruebas y resultado final como arquitectura *cloud* de referencia.

El Capítulo 4 aborda la implementación de *Zero Trust*, partiendo de la solución base y describiendo el análisis de brechas, el diseño de la migración, los controles aplicados y el estado final obtenido.

El Capítulo 5 presenta el proceso de evaluación, las pruebas definidas y los resultados obtenidos para comparar la solución base y la solución *Zero Trust*.

El Capítulo 6 analiza y discute los resultados de la evaluación, interpretando el impacto de la migración en términos de funcionalidad, seguridad, rendimiento, complejidad, despliegue y coste, así como sus principales limitaciones.

# Capítulo 1

# Estudio del problema

## 1.1. El contexto del problema

La transformación digital de las organizaciones ha impulsado una adopción masiva de la computación en la nube como modelo dominante para el despliegue de infraestructuras, aplicaciones y servicios digitales. Plataformas *cloud* públicas, privadas e híbridas permiten una alta escalabilidad, elasticidad y optimización de costes, facilitando además el desarrollo de arquitecturas modernas basadas en microservicios, contenedores y servicios gestionados. Sin embargo, esta evolución tecnológica también ha introducido una superficie de ataque significativamente más amplia y compleja, tal y como señala el ENISA [12].

Los modelos tradicionales de seguridad perimetral, basados en la confianza implícita dentro de una red interna protegida por *firewalls* y mecanismos de control de acceso estáticos, resultan insuficientes en entornos *cloud* altamente distribuidos, dinámicos y expuestos a Internet [22]. En estos escenarios, los recursos ya no residen en un perímetro claramente definido. Los usuarios acceden desde ubicaciones remotas y los servicios interactúan entre sí de forma automática mediante API y canales cifrados, lo que dificulta la aplicación de controles de seguridad basados exclusivamente en la ubicación de red [22, 29].

Además, el incremento de ataques sofisticados, como el movimiento lateral, el compromiso de identidades, el abuso de credenciales válidas o los ataques a la cadena de suministro, ha puesto de manifiesto que asumir confianza implícita dentro de una red supone un riesgo crítico. Informes de incidentes reales muestran que una parte significativa de las brechas de seguridad actuales se producen tras la obtención de

credenciales legítimas, lo que permite a los atacantes operar sin ser detectados dentro de infraestructuras aparentemente confiables [27].

En este contexto surge el paradigma *Zero Trust Architecture* (ZTA), que propone un cambio de enfoque radical: no confiar en ningún usuario, dispositivo o servicio por defecto, independientemente de su ubicación o funcionalidad, y verificar continuamente cada solicitud de acceso a los recursos [22]. Este modelo rompe explícitamente con el concepto de perímetro e identificaciones tradicionales y establece mecanismos de verificación continua, control de acceso dinámico y segmentación de los recursos.

La aplicación de *Zero Trust* en entornos *cloud* se presenta, por tanto, como una respuesta estratégica a los desafíos actuales de la ciberseguridad en dicho campo concreto, pero también plantea interrogantes técnicos, organizativos y normativos que justifican un profundo análisis.

## 1.2. El estado de la cuestión

*Zero Trust Architecture* constituye una de las líneas de investigación más recientes y activas dentro del ámbito de la ciberseguridad. Aunque sus principios comenzaron a formularse a partir de enfoques industriales y propuestas conceptuales previas, no es hasta la publicación especial del NIST, "NIST SP 800-207", en el año 2020 cuando se establece un marco de referencia formal, sistemático y ampliamente aceptado para la definición de ZT, sus componentes arquitectónicos y sus principios operativos. Este hito marca un punto de inflexión en la investigación académica y en la adopción práctica del paradigma, impulsando un crecimiento significativo de trabajos científicos, revisiones sistemáticas e informes técnicos centrados en su aplicación a distintos dominios tecnológicos, verificándose en la Figura 1.1.

Desde 2020, la literatura refleja una clara intensificación del interés por *Zero Trust*, motivada principalmente por la adopción masiva de la computación en la nube, la consolidación de arquitecturas distribuidas y el incremento de amenazas basadas en el compromiso de identidades legítimas. En este contexto, *Zero Trust* deja de considerarse una aproximación teórica alternativa y pasa a analizarse como un modelo de referencia para la protección de infraestructuras modernas, especialmente en entornos *cloud* caracterizados por la elasticidad, la automatización y la ausencia de perímetros de red bien definidos [12, 29].

Las revisiones sistemáticas de la literatura publicadas entre 2024 y 2025 confirman que ZT ha alcanzado un grado notable de madurez conceptual, pero evidencian al

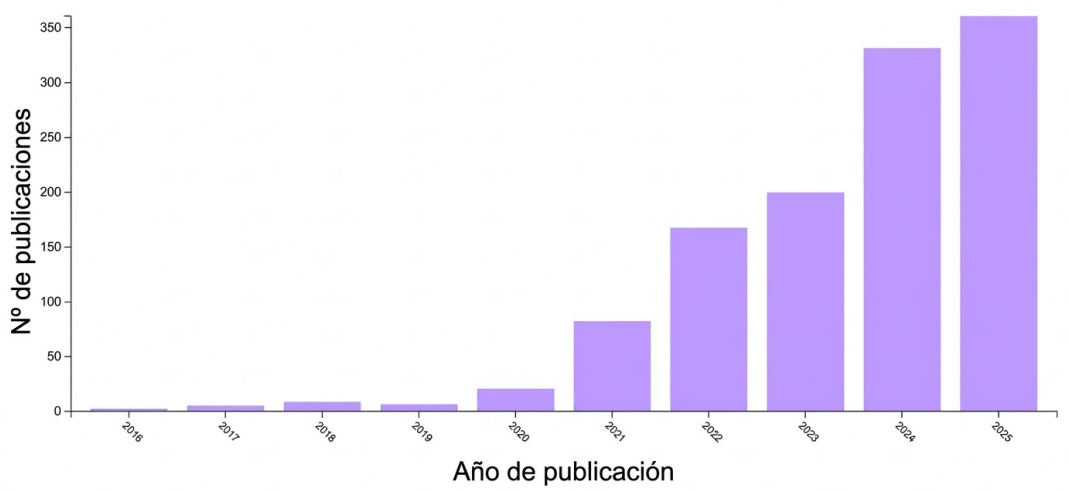



**Figura 1.1:** Evolución anual del número de publicaciones relacionadas con ZTA entre 2016 y 2025. Datos obtenidos de Web of Science (consulta realizada el 21 de enero de 2026). Fuente: elaboración propia.

mismo tiempo una implementación práctica desigual y fragmentada. En particular, la revisión "A Systematic Literature Review on the Implementation and Challenges of Zero Trust Architecture Across Domains" muestra que, aunque el dominio *cloud* es uno de los más estudiados y con mayor número de propuestas, muchas implementaciones se limitan a cubrir mecanismos básicos de autenticación y control de acceso, dejando sin resolver aspectos críticos como la automatización de políticas, la auditoría continua, la orquestación y la percepción contextual del entorno [21].

En consecuencia, el estado actual de la cuestión no se caracteriza por la ausencia de propuestas, sino por la coexistencia de múltiples enfoques parciales, heterogéneos y, en muchos casos, dependientes de soluciones propietarias.

### 1.2.1. De la seguridad perimetral al *Zero Trust*

Los modelos tradicionales de seguridad perimetral se basan en la premisa de que la red corporativa constituye un entorno confiable una vez superados determinados mecanismos de protección, como *firewalls*, redes privadas virtuales o sistemas de detección de intrusiones, es decir, IDS e IPS. Bajo este enfoque, el acceso se concede principalmente en función de la ubicación de red, asumiendo que los usuarios y servicios internos presentan un menor nivel de riesgo. Sin embargo, la literatura reciente coincide en que estos supuestos han dejado de ser válidos en el contexto de la computación en la nube y las arquitecturas distribuidas [22, 29].

La adopción de infraestructuras *cloud* públicas, privadas e híbridas ha introducido una pérdida efectiva del perímetro de red tradicional. Los recursos se distribuyen dinámicamente, los servicios se comunican de forma automática mediante API, los usuarios acceden desde ubicaciones remotas y los entornos *multi-tenant* comparten infraestructuras físicas y lógicas. Informes técnicos recientes, como el "ENISA Threat Landscape for Cloud Computing", señalan que este modelo amplía significativamente la superficie de ataque y favorece vectores basados en el abuso de credenciales legítimas, el movimiento lateral y los ataques a la cadena de suministro [12].

En este contexto, *Zero Trust* emerge como un cambio de paradigma que rompe explícitamente con la noción de confianza implícita asociada a la ubicación de red. Tal como define el "NIST SP 800-207", *Zero Trust* se basa en el principio de "nunca confiar, siempre verificar", asumiendo que ningún usuario, dispositivo o servicio debe ser considerado confiable por defecto, independientemente de su localización [22]. El acceso a los recursos se concede de forma granular, por sesión, y se encuentra sujeto a verificación continua en función de la identidad, el contexto y el nivel de riesgo asociado.

Asimismo, los trabajos más influyentes en la materia subrayan que una implementación efectiva de *Zero Trust* en *cloud* no puede limitarse al control de acceso inicial. Es necesario integrar mecanismos adicionales como infraestructuras de clave pública, monitorización continua, análisis de comportamiento, registro exhaustivo de eventos y sistemas de correlación centralizada, con el objetivo de detectar desviaciones, generar evidencias y adaptar las políticas de seguridad de forma dinámica [23, 24].

En conjunto, la transición desde la seguridad perimetral hacia *Zero Trust* en entornos *cloud* representa un cambio estructural en la forma de concebir la protección de sistemas. La literatura coincide en que *Zero Trust* no constituye una tecnología concreta ni un producto aislado, sino un enfoque arquitectónico que integra la verificación estricta de la identidad, la micro-segmentación de la red, la aplicación de políticas en el punto de acceso y la evaluación continua del riesgo.

### 1.2.2. Arquitectura de referencia y componentes lógicos

La arquitectura de referencia de *Zero Trust* se fundamenta en una separación clara entre los mecanismos de toma de decisiones de acceso y los mecanismos de aplicación de dichas decisiones, apoyándose en múltiples fuentes de información contextual. Esta separación permite implementar el principio de verificación continua

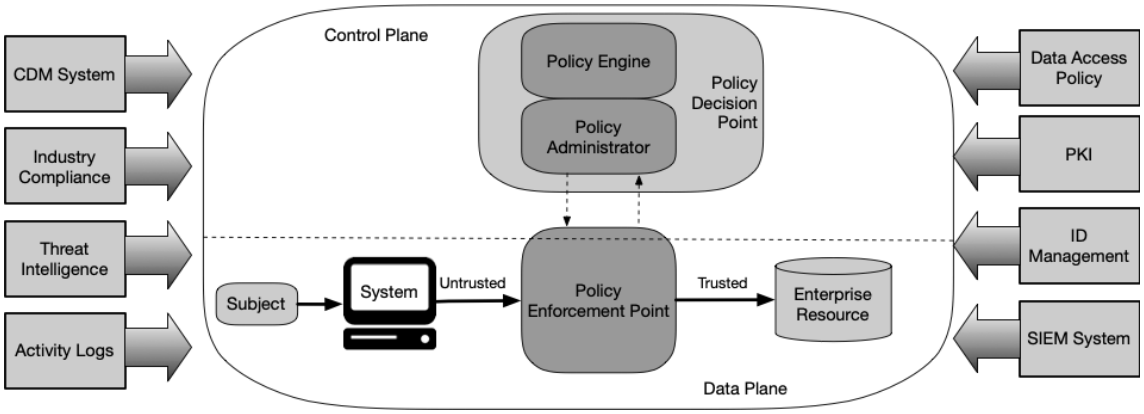



**Figura 1.2:** Arquitectura lógica de referencia de ZT, mostrando los componentes de decisión, ejecución y las fuentes de contexto que intervienen en la evaluación continua de acceso. Fuente: NIST SP 800-207 [22].

y control de acceso dinámico, independientemente de la ubicación de los recursos o de los sujetos que solicitan acceso.

La Figura 1.2 muestra la arquitectura lógica de referencia definida por el NIST, en la que se identifican los componentes principales de decisión y ejecución, así como las fuentes de datos que alimentan el proceso de evaluación de confianza [22].

El núcleo de la arquitectura está constituido por el *Policy Decision Point* (PDP), responsable de evaluar las solicitudes de acceso y emitir decisiones en función de políticas dinámicas y del contexto disponible. El PDP se descompone, a su vez, en dos componentes lógicos diferenciados: el *Policy Engine* (PE) y el *Policy Administrator* (PA). El PE es el encargado de evaluar las políticas de acceso, calcular el nivel de riesgo asociado a cada solicitud y determinar si el acceso debe ser permitido, denegado o revocado. El PA, por su parte, actúa como intermediario entre el motor de decisión y los puntos de aplicación, traduciendo las decisiones del PE en acciones concretas que deben ser ejecutadas por el sistema. Tanto el PE como el PA conviven en un espacio lógico denominado *Control Plane*.

La aplicación efectiva de las decisiones de acceso recae en el *Policy Enforcement Point* (PEP), que se sitúa cerca del recurso protegido. El PEP es responsable de permitir, bloquear o finalizar las conexiones en función de las instrucciones recibidas desde el PDP, garantizando que las políticas se apliquen de forma coherente y consistente en todos los accesos. En entornos *cloud*, este componente puede materializarse en distintos elementos, como *proxies*, *gateways* de API, *sidecars* de microservicios o mecanismos nativos de control de tráfico. Tanto los PEPs como todos los activos implicados en la arquitectura, conviven en el espacio lógico denominado *Data Plane*.

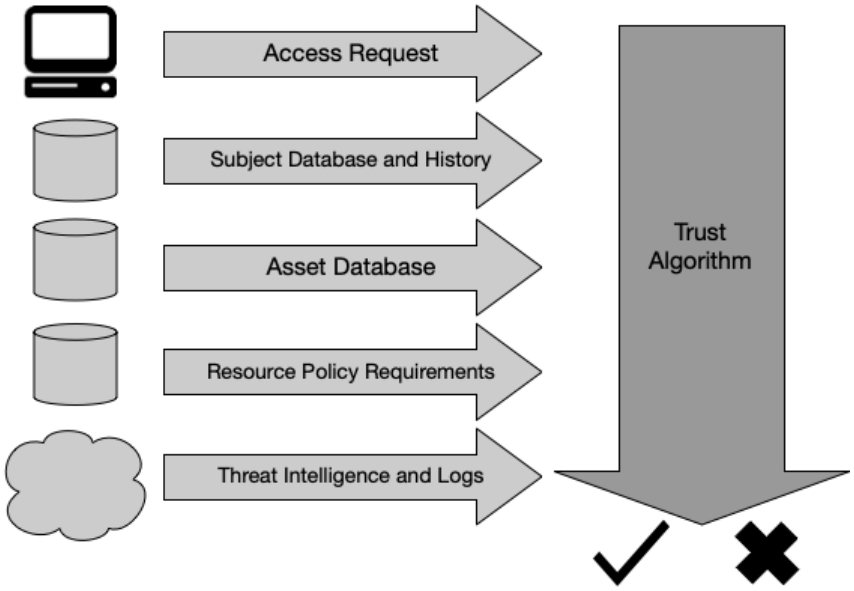



**Figura 1.3:** Entradas del *Trust Algorithm* en una arquitectura Zero Trust, incluyendo las variables que afectan a la toma de decisión. Fuente: NIST SP 800-207 [22].

La toma de decisiones en *Zero Trust* se apoya en un conjunto amplio de fuentes de información contextual que proporcionan una visión continua del estado del sistema. Entre estas fuentes se incluyen los sistemas de gestión de identidades y accesos, que aportan información sobre la identidad y los atributos de los sujetos; los mecanismos de evaluación de la postura de dispositivos y *workloads*; los registros de actividad y telemetría generados por los recursos protegidos; las fuentes de inteligencia de amenazas; y los repositorios de políticas. De acuerdo con el "NIST SP 800-207", todas estas variables constituyen las entradas del *Trust Algorithm* (TA), el componente lógico encargado de evaluar el nivel de confianza asociado a cada solicitud de acceso dentro del PE [22].

El TA procesa de forma continua estas señales contextuales con el objetivo de estimar el riesgo y determinar si una sesión debe ser autorizada, restringida o revocada, permitiendo así que el PDP adapte dinámicamente sus decisiones en función de la evolución del contexto y del comportamiento observado. La Figura 1.3 ilustra las principales fuentes de información que alimentan el TA y su papel central en la toma de decisiones de acceso en una arquitectura ZTA.

En conjunto, esta arquitectura lógica proporciona una base flexible y extensible para la implementación de Zero Trust en entornos *cloud*, al desacoplar la toma de decisiones de su aplicación y permitir la integración de nuevos mecanismos de evaluación, automatización y análisis avanzado. Sobre esta arquitectura de referencia

se construyen las propuestas más avanzadas del estado del arte, incluyendo aquellas que incorporan técnicas de aprendizaje automático y orquestación dinámica, como las analizadas en secciones posteriores.

### 1.2.3. Desafíos de implementación en *cloud*

La adopción de ZTA en la nube parte de una base favorable: los principales proveedores (AWS, Azure y Google) ofrecen mecanismos maduros de identidad, controles nativos de segmentación y un ecosistema amplio de herramientas de seguridad [2, 14, 19]. Sin embargo, la literatura coincide en que la traslación de los principios de *Zero Trust* a infraestructuras elásticas, distribuidas y *multi-tenant* introduce fricciones técnicas y operativas específicas [13, 23].

#### Complejidad y orquestación de políticas

En *cloud*, la superficie de ataque y el número de entidades que requieren control de acceso crecen rápidamente: usuarios, identidades de servicio, *workloads* efímeros, funciones *serverless* y microservicios con comunicaciones intensivas. Esto obliga a pasar de reglas gruesas a políticas finas y contextuales (p. ej., ABAC), lo que incrementa el esfuerzo de definición, validación, despliegue y mantenimiento. Distintos estudios sintetizados en revisiones sistemáticas señalan que, aunque las capacidades de autenticación/autorización suelen estar bien cubiertas, la orquestación automática (aplicar cambios de política de forma consistente y dinámica en múltiples componentes) continúa siendo una debilidad recurrente, especialmente cuando las dependencias entre microservicios evolucionan a gran velocidad. Esta complejidad no es solo respecto a la implementación. También es un coste sostenido, porque la matriz de reglas crece con el número de servicios y contextos posibles, y puede saturar la capacidad del equipo de seguridad para auditar y ajustar políticas con suficiente cadencia [23].

#### Escalabilidad del plano de control y rendimiento

Un principio esencial de ZTA es la verificación por solicitud y la reevaluación continua del nivel de confianza. En entornos de gran escala (organizaciones distribuidas, acceso remoto masivo y servicios en múltiples regiones), este requisito ejerce presión sobre la arquitectura de control: el acoplamiento entre el PDP y los PEPs puede convertirse en un cuello de botella conforme aumenta el volumen de transacciones y la complejidad del conjunto de políticas, introduciendo latencia y degradación de rendimiento. La literatura específica sobre *cloud* indica que la micro-segmentación

y los mecanismos *Zero Trust* no siempre implican un impacto "enorme" en el plano de datos, pero sí obligan a diseñar cuidadosamente el plano de control para minimizar retrasos de autenticación y mantener continuidad del negocio. Además, cuando se pretende evolucionar hacia control de acceso dinámico y adaptativo, aparece un reto adicional: muchas propuestas existentes tienden a definir políticas estáticas o de granularidad insuficiente, lo que limita la capacidad de respuesta ante cambios rápidos del contexto (comportamiento, sensibilidad del recurso, etc.) [28].

**Gestión dinámica de identidades y cargas efímeras**

En entornos basados en la nube, es frecuente que los componentes "nazcan y mueran" en minutos, ya sea por autoescalado o actualizaciones constantes, y que las identidades relevantes no sean solo humanas, sino también de servicio y de carga de trabajo. Esto exige un ciclo de vida robusto de identidades (emisión, rotación, revocación) y su integración con decisiones de acceso condicionadas por el riesgo. Las propuestas de acceso dinámico basadas en ZTA subrayan que la precisión de la autorización depende tanto de métodos de evaluación de confianza (basados en comportamiento y contexto) como de la mejora continua de las políticas para incorporar retroalimentación del entorno. En la práctica, implementar esta visión implica instrumentación (telemetría suficiente para "alimentar" el cálculo de confianza) y un correcto manejo de la gestión para evitar inconsistencias entre usuarios cuando los atributos y señales cambian [21, 28].

**Observabilidad y auditoría**

La evidencia revisada muestra que los mecanismos de *logging* aparecen de forma casi ubicua en propuestas de ZTA [13], porque almacenan "pruebas" necesarias para decisiones futuras de confianza y para la aplicación de políticas. Sin embargo, otra conclusión persistente en revisiones transversales es que la auditoría suele ser *post hoc* y no verdaderamente continua o proactiva, y que muchas soluciones en *cloud* carecen de capas completas de auditoría y orquestación a pesar de los requisitos de cumplimiento. En entornos multi-cuenta/multi-región, consolidar trazas y evidencias (quién accedió, desde dónde, con qué postura, a qué recurso, bajo qué política efectiva) es crítico tanto para investigación forense como para demostrar conformidad normativa; pero lograrlo implica coste, normalización de eventos y evitar puntos ciegos entre servicios gestionados [27].

#### Interoperabilidad y dependencia del proveedor

Los trabajos revisados identifican como retos de seguridad en redes *cloud* la interoperabilidad entre arquitecturas heterogéneas, las vulnerabilidades del plano de virtualización y prácticas deficientes de control/visibilidad sobre los datos en la industria. A esto se suma un problema específicamente destacado para el dominio *cloud*: aunque existe soporte maduro por parte de proveedores, persisten preocupaciones de privatización y dificultad para extender un modelo ZTA consistente cuando se opera en *multi-cloud* [23]. De hecho, se observa la necesidad de mayor soporte para aplicaciones desplegadas como microservicios y contenedores, así como para escenarios *multi-cloud* con *hardware* y *software* de distintos fabricantes. En términos de implementación, esto obliga a definir un "mínimo común" de atributos, decisiones y evidencias o, en caso contrario, aceptar una mayor dependencia de controles específicos del proveedor.

#### Limitaciones organizativas y adopción parcial

No todos los actores cuentan con los mismos recursos. Se destaca que, aunque gobiernos y grandes proveedores impulsan marcos y soluciones de *Zero Trust*, organizaciones pequeñas y medianas pueden carecer de tiempo, recursos o capacidad para adoptar estas arquitecturas en sus ecosistemas. Esto conecta con un riesgo práctico: una adopción parcial que cubra autenticación y segmentación, pero deje sin resolver orquestación, auditoría continua o percepción contextual [13].

En este contexto, diversos trabajos han propuesto taxonomías que estructuran los principales retos asociados a la adopción de ZT, permitiendo agruparlos por dimensiones técnicas, organizativas y operativas. Estas taxonomías ofrecen una visión de conjunto que facilita la comprensión de por qué la implementación de ZTA en la nube sigue siendo un desafío abierto, a pesar de la madurez de muchas de sus tecnologías subyacentes. La Figura 1.4 resume de forma sintética estos desafíos.

### 1.2.4. Tecnologías emergentes y orquestación de la arquitectura

Desde el punto de vista de la orquestación, el paradigma de *Policy-as-Code* (PaC) ha emergido como la tecnología habilitadora fundamental. Herramientas como *Open Policy Agent* (OPA) o lenguajes de autorización como Cedar [1], permiten que las reglas de seguridad se definan de forma programática y se evalúen de manera centralizada en un PDP. Esta aproximación permite superar las limitaciones de los sistemas

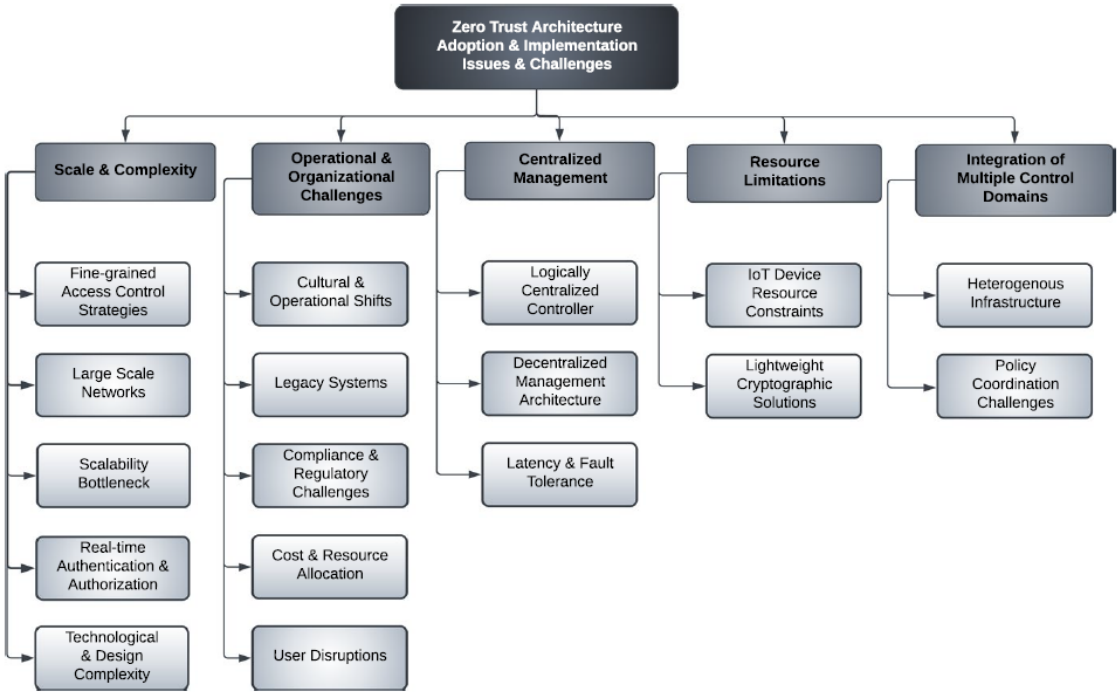



**Figura 1.4:** Taxonomía de los principales desafíos en la adopción e implementación de ZTA. Fuente: Zero Trust Architecture: A Systematic Literature Review [13].

tradicionales basados en rígidas listas de control de acceso (ACL), facilitando una respuesta automática ante cambios en el estado de seguridad de los activos o del entorno.

En paralelo, la integración de tecnologías de Acceso Verificado, algunas ya consolidadas en la industria, pero que aplican principios de ZT y brindan mayor confianza de seguridad; son de obligado uso en esta arquitectura. Mecanismos, siendo claves los reconocimientos biométricos, las llaves de acceso (PassKeys), códigos OTP, entre otros; están redefiniendo la exposición de servicios informáticos, especialmente en la nube. A diferencia de las arquitecturas convencionales que utilizan redes privadas virtuales para otorgar confianza implícita a nivel de red, las soluciones emergentes proponen la creación de túneles cifrados efímeros y específicos para cada aplicación. Este modelo de orquestación valida de forma continua tanto la identidad del solicitante como el estado de cumplimiento del dispositivo antes de cada transacción, garantizando que el acceso sea granular y limitado estrictamente al recurso necesario [9].

Finalmente, los marcos de trabajo actuales subrayan la necesidad de una identidad de carga de trabajo que trascienda los límites de la infraestructura física. Estándares abiertos como SPIFFE [8] permiten el arranque y la iteración de identidades

criptográficas de corta duración en entornos dinámicos y heterogéneos. Este enfoque desplaza el modelo de seguridad tradicional basado en topología de red y direcciones IP hacia uno de autenticación mutua (mTLS) y verificación de *tokens* (JWT) en el nivel de aplicación. Al desacoplar la identidad del servicio de su ubicación en la VPC, se facilita una microsegmentación lógica que permite una orquestación de seguridad escalable, limitando eficazmente el movimiento lateral y reduciendo la superficie de ataque en arquitecturas de microservicios [3].

### 1.2.5. Ejes de investigación dominantes en ZTA

A partir de las SLR y de los artículos investigados, incluidos en la bibliografía; se observan cuatro líneas técnicas recurrentes en ZTA para *cloud*:

#### IAM dinámico y acceso adaptativo

Una parte importante de la investigación intenta resolver el gran punto débil de *cloud*: el IAM (Identity and Access Management) clásico (RBAC/MAC/DAC/A-BAC) no escala bien en entornos elásticos, *multi-tenant* y con cambios constantes. En ese contexto, se proponen enfoques de confianza dinámica (*trust score*) y reglas adaptativas para ajustar privilegios en tiempo real.

Un ejemplo muy representativo es el que se indica en "Zero-trust based dynamic access control for cloud computing" [28], que propone un modelo TBAC (Trust-Based Access Control) y su extensión DR-TBAC: calcula confiabilidad en tiempo real desde *logs* (con LSTM) y usa Deep Reinforcement Learning (DQN) para ajustar dinámicamente umbrales de confianza y activar/revocar permisos. La Figura 1.5 muestra la arquitectura general del sistema DR-TBAC, donde la evaluación continua de confianza basada en telemetría se combina con técnicas de aprendizaje profundo y aprendizaje por refuerzo para la toma dinámica de decisiones de acceso.

Además, se aterriza en un entorno realista con OpenStack (Victoria), integrándose como módulo de seguridad, es decir, un PDP que reevalúa permisos tras cada actualización de reglas, evitando el patrón de revocar después del incidente (*post hoc*). En resultados comparativos, DR-TBAC mejora métricas frente a *baselines* (incluyendo RBAC), alcanzando valores altos de *precision/recall/F1-score* en su evaluación.

La idea clave desemboca en que la tendencia fuerte 2020–2025 es pasar de políticas estáticas y métricas dinámicas a política y umbrales dinámicos, con aprendizaje a partir de telemetría (*logs*, comportamiento, contexto).

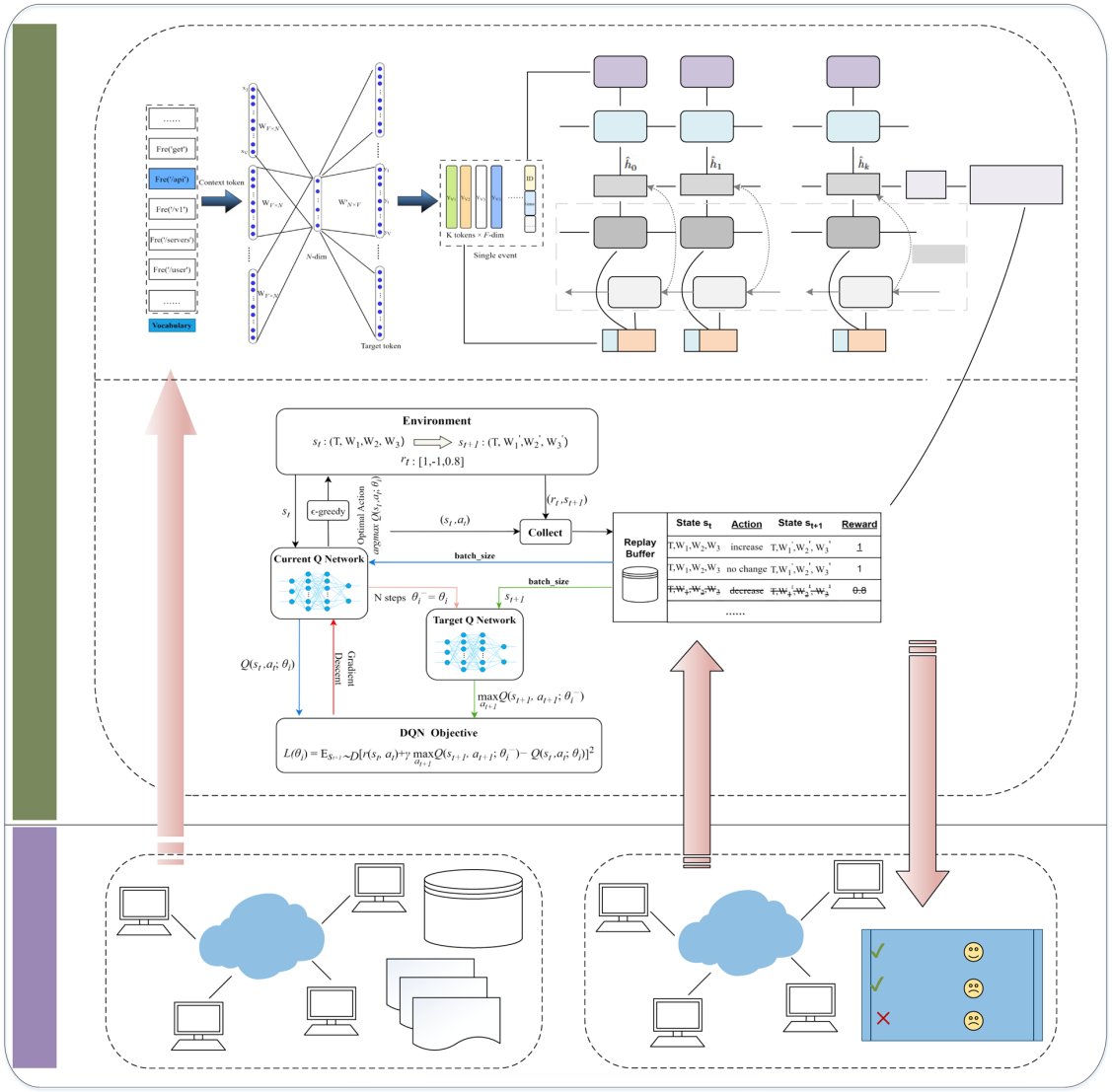



**Figura 1.5:** Arquitectura del sistema DR-TBAC para control de acceso dinámico basado en Zero Trust en entornos *cloud*. Fuente: Zero-trust Based Dynamic Access Control for Cloud Computing [28].

### Micro-segmentación y aplicación nativa en la nube

En el paradigma actual de computación en la nube, el perímetro de seguridad se ha desplazado desde los límites de la red física hacia las cargas de trabajo individuales. La literatura científica coincide en que el recurso a proteger ha evolucionado de máquinas virtuales aisladas a ecosistemas de microservicios efímeros e interconectados. En este contexto, la micro-segmentación se erige como un pilar fundamental de la ZTA.

Según el artículo "Zero Trust: The What, How, Why, and When"; la micro-segmentación no debe entenderse simplemente como una división de la red, sino como un mecanismo crítico para minimizar la superficie de ataque, reducir drásticamente el movimiento lateral y delimitar la zona afectada ante una posible brecha de seguridad [24]. Estos autores subrayan que, a diferencia de la segmentación tradicional basada en VLAN o subredes, la micro-segmentación ZT permite una visibilidad y un control granular que son independientes de la topología de red subyacente.

Por su parte, la revisión sistemática "A Systematic Literature Review on the Implementation and Challenges of Zero Trust Architecture Across Domains" destaca que la implementación efectiva en entornos *cloud-native* (especialmente en Kubernetes y arquitecturas de microservicios) se basa en la convergencia de tres ejes: la autenticación robusta, la segmentación funcional y el aislamiento de contenedores [21]. La investigación en este dominio demuestra que las propuestas más representativas abogan por una aplicación dinámica de políticas. Esto se materializa mediante el despliegue de PEPs situados en la proximidad inmediata del recurso, utilizando tecnologías como *sidecars* dentro de un *service mesh*, *gateways* de aplicación y políticas de seguridad a nivel de *pod*.

Un punto crítico identificado en la literatura reciente es la transición del concepto de segmento de red hacia la segmentación basada en la identidad. Como sugieren los estudios analizados:

- La segmentación es funcional: se define por el propósito del servicio y la identidad de la carga de trabajo, no por su dirección IP.
- La aplicación de políticas es distribuida: el control se ejecuta en cada punto de interacción, garantizando que cada solicitud de comunicación sea verificada explícitamente antes de conceder el acceso.

Esta capacidad de integrar la seguridad directamente en la lógica de orquestación de la infraestructura permite que el control de acceso sea tan dinámico y escalable como los propios servicios que protege.

**Automatización y orquestación de políticas**

A pesar de su importancia, la revisión por dominios identifica la orquestación como una de las dimensiones con menor grado de implementación y madurez técnica en la literatura actual, lo que supone un obstáculo crítico para el despliegue de ZTA a escala empresarial [21].

En el contexto de la nube, la automatización es indispensable debido a la volatilidad de los microservicios y la necesidad de respuestas en tiempo real. Según el artículo "Automation and Orchestration of Zero Trust Architecture: Potential Solutions and Challenges" [4], las soluciones de orquestación actuales están evolucionando hacia un modelo de inteligencia distributiva, donde la Inteligencia Artificial juega un papel habilitador fundamental para superar los cuellos de botella de los sistemas tradicionales:

- Evaluación de confianza adaptativa: Frente a las políticas estáticas, se propone el uso de modelos de aprendizaje profundo y aprendizaje por refuerzo para calcular el nivel de confianza de cada solicitud de acceso de forma continua. Esto permite que la orquestación reaccione a cambios sutiles en el comportamiento del usuario o de la carga de trabajo, ajustando los privilegios de manera proactiva.
- Gestión inteligente del ciclo de vida de las políticas: La orquestación en la nube debe gestionar la creación, actualización y revocación de miles de reglas de acceso. La literatura reciente aboga por el uso de técnicas de procesamiento de lenguaje natural y razonamiento automatizado para verificar la consistencia de las políticas y evitar configuraciones erróneas que puedan ser explotadas durante el movimiento lateral.

El futuro de la orquestación en la nube reside en el concepto de seguridad autonómica. Esto implica que el sistema no solo debe aplicar políticas, sino ser capaz de "auto-optimizarse" y "auto-protegerse", utilizando la telemetría constante de la red para reconfigurar los PEPs sin intervención humana. Esta transición hacia una orquestación impulsada por IA es, según los autores, el único camino viable para garantizar el cumplimiento del principio de verificación explícita en infraestructuras de nube de escala masiva.

### 1.2.6. Síntesis del estado de la cuestión

A partir de la revisión realizada, se observa que la investigación actual sobre *Zero Trust* en entornos *cloud* se articula en torno a varios ejes tecnológicos recurrentes. La Tabla 1.1 resume las principales tecnologías y líneas de trabajo identificadas.

**Tabla 1.1:** Síntesis de tecnologías y líneas de trabajo en arquitecturas *Zero Trust* en entornos *cloud*

| **Eje** | **Tecnologías o mecanismos asociados** | **Papel en *Zero Trust*** |
|---|---|---|
| Identidad y acceso adaptativo | IAM, MFA, RBAC, ABAC, TBAC, *trust score*, autenticación continua | Permiten verificar explícitamente a usuarios, servicios y cargas de trabajo antes de conceder acceso a recursos. |
| Microsegmentación | *NetworkPolicies*, grupos de seguridad, *service mesh*, *sidecars*, *gateways* | Reduce la confianza implícita entre componentes y limita el movimiento lateral dentro de la arquitectura. |
| Protección de credenciales | Credenciales temporales, federación de identidad, STS, rotación de claves, gestión de secretos | Evita el uso de credenciales persistentes y reduce el impacto de una posible exposición. |
| Orquestación y políticas | *Policy-as-Code*, OPA, Cedar, PDP, PEP, automatización de políticas | Facilita la definición, evaluación y aplicación coherente de políticas de acceso en entornos dinámicos. |
| Observabilidad y auditoría | *Logs*, telemetría, CloudWatch, CloudTrail, SIEM, SOAR, correlación de eventos | Proporciona evidencias para auditoría, investigación, detección de anomalías y revisión continua de la postura de seguridad. |
| Administración segura | Bastiones privados, acceso remoto controlado, SSM, restricción de *endpoints*, canales administrativos privados | Reduce la exposición de los planos de administración y limita el acceso operativo a vías controladas. |

*Continúa en la siguiente página*

**Tabla 1.1:** (continuación)

| **Eje** | **Tecnologías o mecanismos asociados** | **Papel en *Zero Trust*** |
|---|---|---|
| Automatización e infraestructura reproducible | Terraform, Kubernetes, CI/CD, *scripts* de despliegue, infraestructura como código | Permite aplicar controles de forma repetible, auditable y coherente en infraestructuras cambiantes. |

## 1.3. La definición del problema

La presente investigación parte de la observación de que, a pesar de la madurez conceptual alcanzada por ZTA como paradigma de seguridad diseñado para entornos *cloud*, existe una brecha clara entre la definición teórica de *Zero Trust* y su aplicación práctica en sistemas reales que tradicionalmente han sido concebidos bajo modelos de seguridad perimetral.

En contextos *cloud*, como los proporcionados por Amazon Web Services, las organizaciones suelen desplegar aplicaciones modernas basadas en microservicios, API, identidades dinámicas y componentes distribuidos. A pesar de ello, muchas de estas aplicaciones no aprovechan plenamente los principios de *Zero Trust*, sino que aplican controles híbridos o parciales (autenticación reforzada, segmentación básica), manteniendo implícitos supuestos de confianza dentro de elementos de su arquitectura que, por su naturaleza basada en la nube, ya no resultan seguras.

Esto da lugar a una doble problemática:

- Un déficit metodológico para evaluar y guiar la transición desde una arquitectura tradicional con controles de seguridad convencionales hacia una ZTA completamente alineada con los principios de referencia, medible con modelos de madurez apropiados.
- Una carencia de evidencia empírica y procedimientos sistematizados que permitan demostrar, cuantificar y justificar las diferencias entre un estado "básico" y un estado "optimizado por *Zero Trust*" en términos de seguridad, coste, complejidad de implementación y operación en un entorno en la nube.

En otras palabras, a pesar de que numerosos trabajos describen componentes individuales de ZTA, así como los desafíos técnicos ligados a su implementación en entornos *cloud*, no existe un trabajo que combine (i) una evaluación de estado inicial,

(ii) una estrategia de migración documentada, (iii) un modelo de medición aplicable, y (iv) evidencia clara del efecto de la migración aplicada a un caso real en *cloud*.

Esta carencia se observa, de forma explícita, en las siguientes evidencias derivadas de la literatura revisada:

1. El enfoque académico en propuestas parciales, donde la implementación de *Zero Trust* se limita a aspectos específicos como IAM o micro-segmentación sin integrar métricas de comparación entre estados [21].

2. La ausencia de procedimientos replicables para la transformación de arquitecturas existentes, de manera que los equipos de seguridad puedan documentar y medir de forma objetiva la transición de un modelo tradicional a *Zero Trust* [23].

3. La falta de modelos de evaluación ampliamente aceptados que guíen y cuantifiquen la madurez de *Zero Trust* en sistemas reales, más allá de conceptos teóricos y métricas aisladas, como el que se presenta en "Zero Trust Maturity Model Version 2.0" [10].

4. La limitada evidencia empírica que demuestre, en un contexto práctico, el impacto cuantificable de adoptar ZT dentro de una plataforma *cloud* pública tanto en términos de beneficios de seguridad como de costes, complejidad y carga operativa.

Desde la perspectiva de ingeniería, esta situación plantea varios retos concretos:

- La necesidad de caracterizar de forma objetiva qué significa que una aplicación "no aplica *Zero Trust*" en términos de controles efectivamente implementados, métricas de seguridad y supuestos de confianza.

- La definición de criterios de comparación claros, medibles y replicables que permitan evaluar de forma sistemática la efectividad de la migración hacia *Zero Trust*.

- El desarrollo de procedimientos y pasos de migración aplicables en entornos *cloud*, con recomendaciones prácticas y guías basadas en evidencia.

En consecuencia, el problema investigado en este trabajo se formula como la falta de metodologías y modelos aplicables para evaluar y guiar la migración de una arquitectura tradicional de aplicación hacia una arquitectura *Zero Trust* en la nube, así como la ausencia de evidencia concreta y replicable de los efectos de esa

migración en términos de seguridad operativa, coste de implementación y factores de desempeño.

## 1.4. Normativa

Abordar la normativa que se aplica a *Zero Trust* puede resultar en una ardua tarea. Este paradigma es tan reciente que no dispone de ninguna ISO donde se mencione explícitamente cómo debe llevarse a cabo su desarrollo, implementación, gestión, etc. Sin embargo, se pueden relacionar varias normas y leyes existentes con ZTA, al aplicarse principios de seguridad indicados en varios documentos oficiales. El orden de mención de dichos documentos se basa en ir desde el ámbito más global, ISOs y estándares, hasta el más específico, como pueden ser leyes o normativa españolas.

### 1.4.1. Estándares y marcos internacionales

#### ISO/IEC 27001 – Sistema de Gestión de la Seguridad de la Información

Aunque la norma ISO 27001 no mencione explícitamente *Zero Trust*, su enfoque basado en riesgos es el pilar fundamental para cualquier arquitectura ZTA. La norma exige la identificación de activos y la evaluación de riesgos, lo que en ZT se traduce en la definición de la superficie de protección. El ciclo PDCA (*Plan-Do-Check-Act*) asegura que la verificación continua de la identidad y la integridad del dispositivo, núcleo de ZT, se mantenga actualizada frente a amenazas cambiantes [16].

#### ISO/IEC 27002 – Controles de seguridad

Este código de prácticas proporciona la guía para implementar los controles de la ISO 27001. En el contexto de ZT, son especialmente relevantes los controles revisados en su versión de 2022, como la gestión de identidades y accesos, el control de acceso físico y lógico, y la gestión de derechos de acceso privilegiado. Estos controles materializan la política de mínimo privilegio necesaria para segmentar la red de forma efectiva [17].

#### ISO/IEC 27017 – Seguridad en servicios en la nube

Dado que la mayoría de las implementaciones de *Zero Trust* se apoyan en entornos híbridos o *cloud*, esta norma es vital. Proporciona controles específicos para la nube, como la separación de entornos y la visibilidad del tráfico, que son requisitos previos

para implementar un PDP que gestione el acceso a recursos que ya no residen en un perímetro físico tradicional [15].

#### NIST SP 800-207 – Zero Trust Architecture

Este es el único documento oficial que trata explícitamente sobre *Zero Trust*. Se ha hecho referencia al mismo durante todo el transcurso del capítulo. Sienta las bases conceptuales para conseguir el paso de arquitecturas tradicionales a sistemas que implementen los principios ZTA [22].

### 1.4.2. Marco europeo

#### Reglamento (UE) 2016/679 – RGPD

El RGPD impone la obligación de aplicar medidas técnicas y organizativas apropiadas para garantizar la seguridad de los datos personales. *Zero Trust* cumple directamente con los principios de protección de datos desde el diseño y por defecto. Al segmentar el acceso y autenticar cada solicitud, se reduce drásticamente la probabilidad de una brecha de datos masiva, cumpliendo con la exigencia de confidencialidad e integridad del Reglamento [25].

#### Directiva (UE) 2022/2555 – NIS2

La directiva NIS2 amplía los sectores considerados críticos y endurece las obligaciones de ciberseguridad. ZT se alinea con el requisito de higiene cibersegura y seguridad de la cadena de suministro. NIS2 obliga a las entidades a implementar controles de acceso estrictos y autenticación multifactor, elementos que son, por definición, los componentes básicos de una arquitectura ZT [26].

### 1.4.3. Normativa y legislación española

#### Ley Orgánica 3/2018 – LOPDGDD

Esta ley adapta el RGPD al ordenamiento español. En su disposición adicional primera, establece que las medidas de seguridad deben ser proporcionales al riesgo, remitiendo en muchos casos al Esquema Nacional de Seguridad. *Zero Trust* actúa aquí como la herramienta técnica para garantizar que el acceso a datos sensibles esté restringido solo al personal autorizado, reforzando el control de trazabilidad que exige la ley [18].

### Real Decreto 311/2022 – Esquema Nacional de Seguridad (ENS)

La actualización de 2022 del ENS está profundamente alineada con los principios de *Zero Trust*. Introduce el concepto de perímetro dinámico y refuerza la necesidad de vigilancia continua y respuesta ante incidentes. El ENS obliga a una segmentación de red que evite el movimiento lateral, lo cual es el objetivo principal de las arquitecturas de microsegmentación dentro de ZT [20].

### CCN-STIC 800 / 801 / 802

Estas guías del Centro Criptológico Nacional (CCN) proporcionan las pautas para la implementación del ENS. La serie 800 es clave para entender cómo aplicar configuraciones seguras. En particular, las guías sobre seguridad en entornos *cloud* y acceso remoto han comenzado a incorporar terminología de *Zero Trust*, recomendando explícitamente la eliminación de la confianza ciega en las redes internas [5–7].

# Capítulo 2

# Gestión de proyecto *software*

## 2.1. Alcance del proyecto

### 2.1.1. Definición del proyecto

El proyecto consiste en el desarrollo y despliegue de una aplicación *cloud* sencilla, diseñada como caso de estudio para analizar posteriormente su evolución hacia un modelo de seguridad basado en *Zero Trust*. La solución inicial incluye una aplicación web con un *frontend*, una API *backend*, un servicio *batch*, una base de datos relacional y almacenamiento de objetos para la gestión de imágenes o ficheros asociados.

La aplicación se despliega sobre una infraestructura *cloud*, utilizando servicios gestionados y un entorno de contenedores basado en Kubernetes. Esta primera versión representa una arquitectura *cloud* funcional, pero con ciertos elementos de confianza implícita habituales en despliegues iniciales, como permisos amplios, exposición administrativa o una segmentación limitada entre componentes. A partir de esta solución base, el proyecto plantea una migración incremental hacia una arquitectura *Zero Trust*. Se busca introducir controles de seguridad adicionales sobre la infraestructura existente.

### 2.1.2. Estimación de tareas y recursos

Las tareas principales del proyecto se agrupan en los siguientes bloques:

- Análisis inicial de requisitos funcionales, técnicos y de seguridad.
- Diseño de la arquitectura *cloud* base y definición de los componentes de aplicación.

- Desarrollo de la aplicación web, la API *backend* y el servicio *batch*.
- Definición de la infraestructura *cloud* mediante infraestructura como código.
- Despliegue de la solución base en Kubernetes sobre AWS.
- Ejecución de pruebas funcionales, de seguridad, de rendimiento y de coste para la solución base.
- Identificación de brechas de seguridad respecto a los principios *Zero Trust*.
- Implementación de la migración incremental hacia *Zero Trust*.
- Ejecución de pruebas funcionales, de seguridad, de rendimiento y de coste para la solución final.
- Documentación técnica del proyecto y análisis de resultados.

En cuanto a los **recursos humanos**, se plantea un equipo formado por un ingeniero informático especializado en ciberseguridad. Al tratarse de un proyecto de alcance reducido, este perfil asume las tareas de análisis, desarrollo, despliegue, seguridad, pruebas y documentación.

Los **recursos técnicos** necesarios incluyen un entorno de desarrollo local, las herramientas *software* necesarias para el desarrollo y despliegue del proyecto, y los servicios *cloud* de AWS utilizados para alojar la infraestructura y evaluar la solución.

### 2.1.3. Presupuesto

A continuación se detalla un presupuesto estimado para el coste total del proyecto. Al tratarse de una aplicación desplegada sobre infraestructura *cloud*, no se contempla la adquisición de servidores físicos propios. El coste de cómputo, red, almacenamiento y servicios gestionados se recoge dentro del apartado de infraestructura *cloud*. Como recurso *hardware* local únicamente se considera el equipo utilizado para el desarrollo, aplicado como coste amortizado.

#### Coste de personal

Para la estimación del coste de personal se considera un único perfil técnico: un ingeniero informático especializado en ciberseguridad. Este perfil asume las tareas de análisis, diseño, desarrollo, despliegue, migración, pruebas y documentación del proyecto.

La estimación de esfuerzo se toma a partir de la planificación de tareas definida en la Tabla 2.3, donde se obtiene una dedicación total de 171 horas. Se estima un coste de 30 EUR/hora, al tratarse de un perfil técnico con conocimientos de desarrollo *software*, infraestructura *cloud*, Kubernetes y seguridad.

Por tanto, el coste total de personal se calcula de la siguiente forma:

$$171 \text{ horas} \times 30 \text{ EUR/hora} = 5130 \text{ EUR}$$

De este modo, el coste estimado de personal para el proyecto supone **5130 EUR**.

### Coste de *hardware*

Como recurso *hardware* local únicamente se considera el equipo utilizado para el desarrollo, sin imputarlo como coste directo al no haber sido adquirido específicamente para el proyecto.

### Coste de infraestructura

El coste de infraestructura *cloud* se calcula a partir del gasto real registrado en AWS durante el periodo de desarrollo, despliegue y pruebas del proyecto. Este coste corresponde al uso experimental de los servicios necesarios para construir, desplegar y validar la solución, por lo que no debe interpretarse como el coste mensual estable de una arquitectura en producción.

El periodo considerado comprende desde abril de 2026 hasta junio de 2026. Durante este intervalo, el coste total registrado fue de 31,19 USD, con un coste medio mensual aproximado de 10,40 USD. En la Tabla 2.1 se muestra el desglose por servicios de AWS, exportado directamente desde AWS Cost Explorer. Cabe destacar que se reflejan todos los servicios aunque su gasto real sea 0,00 USD, para mantenerse fiel al uso realizado.

**Tabla 2.1:** Coste real de infraestructura *cloud* en AWS

| **Servicio AWS** | **Coste total** |
|---|---|
| Elastic Container Service for Kubernetes | 10,41 USD |
| EC2-Otros | 6,00 USD |
| Impuestos | 5,41 USD |

*Continúa en la siguiente página*

**Tabla 2.1:** (continuación)

| **Servicio AWS** | **Coste total** |
|---|---|
| EC2-Instancias | 3,78 USD |
| Elastic Load Balancing | 2,42 USD |
| Relational Database Service | 1,74 USD |
| VPC | 1,22 USD |
| Cost Explorer | 0,10 USD |
| EC2 Container Registry (ECR) | 0,06 USD |
| S3 | 0,05 USD |
| Secrets Manager | 0,00 USD |
| Glue | 0,00 USD |
| Key Management Service | 0,00 USD |
| SNS | 0,00 USD |
| SQS | 0,00 USD |
| CloudWatch | 0,00 USD |
| CloudTrail | 0,00 USD |
| **Total** | **31,19 USD** |

Asimismo, en la Figura 2.1 se puede visualizar el gráfico de barras de los servicios que más gasto reflejan en el total.

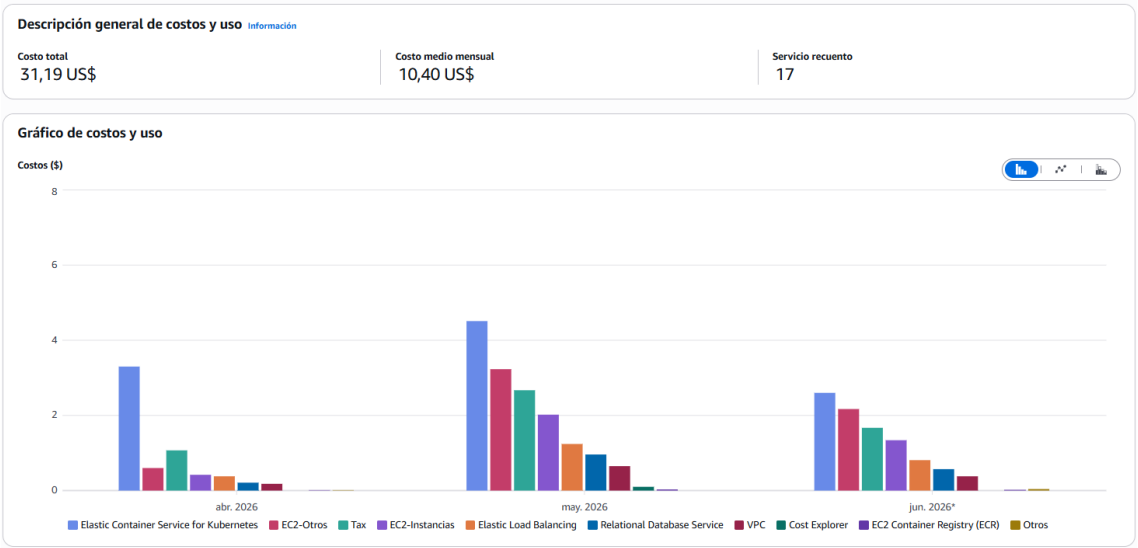



**Figura 2.1:** Gráfico de barras del coste real de infraestructura *cloud* en AWS. Fuente: captura propia.

Para trabajar en la misma moneda, se realiza conversión de USD a EUR. Considerando una equivalencia de 1 EUR = 1.1467 USD, valor referencia publicado por el Banco Central Europeo para el dólar estadounidense [11], el coste total de infraestructura *cloud* registrado es:

$$1 \text{ USD} = \frac{1}{1{,}1467} \text{ EUR} = 0{,}8713 \text{ EUR}$$

$$31{,}19 \text{ USD} \times 0{,}8713 = 27{,}18 \text{ EUR}$$

De este modo, el coste de infraestructura para el proyecto supone **27,18 EUR**.

### Coste total

El coste total del proyecto se obtiene sumando el coste estimado de personal y el coste real de infraestructura *cloud* registrado durante el periodo de desarrollo, despliegue y pruebas. El coste de *hardware* no se imputa como coste directo, ya que el desarrollo se realiza desde un equipo portátil ya disponible y no adquirido específicamente para el proyecto.

Con el objetivo de obtener un presupuesto más completo, además de los costes directos se incorpora una estimación de costes indirectos. Estos costes representan gastos generales asociados al desarrollo del proyecto que no han sido medidos de forma individual, como consumo eléctrico, conexión a Internet, uso del entorno de trabajo, herramientas auxiliares y otros recursos de apoyo. Dado que estos costes no se han registrado de manera separada, se estima un porcentaje del 15 % sobre los costes directos del proyecto como criterio presupuestario.

Finalmente, se aplica el IVA correspondiente al tipo general vigente en España, fijado en el 21 %, con el objetivo de obtener el presupuesto total del proyecto con impuestos incluidos.

A partir de estos criterios, los costes directos del proyecto se obtienen sumando el coste de personal y el coste real de infraestructura *cloud*. Sobre esta cantidad se calcula posteriormente el porcentaje correspondiente a los costes indirectos estimados. Se aplica finalmente el IVA del 21 %, obteniendo así el coste total del proyecto con impuestos incluidos. A partir de los valores estimados, el presupuesto total del proyecto queda recogido en la Tabla 2.2.

**Tabla 2.2:** Presupuesto total del proyecto

| **Concepto** | **Coste** |
|---|---|
| Coste de personal | 5130,00 EUR |
| Coste real de infraestructura *cloud* | 27,18 EUR |
| Coste de *hardware* | No imputado |
| **Costes directos** | **5157,18 EUR** |
| Costes indirectos estimados (15 %) | 773,58 EUR |
| **Subtotal sin IVA** | **5930,76 EUR** |
| IVA (21 %) | 1245,46 EUR |
| **Total del proyecto** | **7176,22 EUR** |

## 2.2. Plan de trabajo

El plan de trabajo se organiza en fases consecutivas, siguiendo una evolución incremental del proyecto. En primer lugar se construye una solución *cloud* base funcional y, posteriormente, se analiza y ejecuta su migración hacia un modelo de seguridad basado en *Zero Trust*.

### 2.2.1. Identificación y estimación de tareas

La Tabla 2.3 recoge las tareas principales del proyecto junto con una estimación inicial de esfuerzo en horas.

**Tabla 2.3:** Identificación y estimación de tareas del proyecto

| **ID** | **Tarea** | **Horas** |
|---|---|---|
| **Análisis** | | |
| T1 | Estudio inicial de *Zero Trust* | 11 |
| T2 | Análisis de requisitos funcionales, técnicos y de seguridad | 15 |
| T3 | Diseño de la arquitectura *cloud* base | 8 |
| **Base *cloud*** | | |
| T4 | Desarrollo de la aplicación web, API *backend* y servicio *batch* | 23 |
| T5 | Despliegue base en AWS/EKS | 14 |
| T6 | Infraestructura como código | 11 |
| T7 | Pruebas de la solución base | 24 |

*Continúa en la siguiente página*

**Tabla 2.3:** (continuación)

| ID | Tarea | Horas |
|---|---|---|
| ***Zero Trust*** | | |
| T8 | Identificación de brechas *Zero Trust* | 2 |
| T9 | Diseño de la migración incremental | 4 |
| T10 | Implementación de controles *Zero Trust* | 13 |
| T11 | Pruebas de la solución final | 10 |
| **Documentación** | | |
| T12 | Documentación y análisis de resultados | 36 |
| **Total** | – | **171** |

### 2.2.2. Planificación de tareas

En la Figura 2.2 se muestra el diagrama de Gantt correspondiente a la planificación estimada del proyecto.

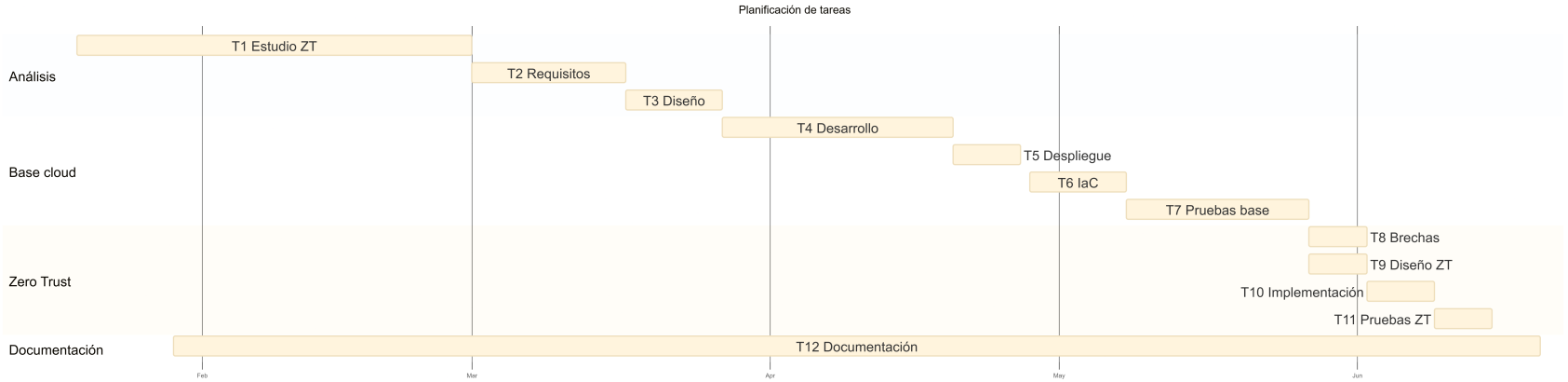



**Figura 2.2:** Planificación de tareas del proyecto mediante diagrama de Gantt. Fuente: elaboración propia.

## 2.3. Gestión de riesgos

La gestión de riesgos tiene como objetivo identificar los principales eventos que podrían afectar al desarrollo, despliegue o evaluación del proyecto. Al tratarse de una solución desplegada sobre infraestructura *cloud*, los riesgos principales están relacionados con la complejidad técnica, el coste de los servicios, la configuración de seguridad y la disponibilidad de evidencias suficientes para justificar los resultados. En la Tabla 2.4 se puede visualizar el desglose de posibles riesgos identificados junto con su probabilidad de suceso, impacto y correspondiente medida de mitigación.

**Tabla 2.4:** Riesgos identificados del proyecto

| ID | Riesgo | Prob. | Impacto | Medida de mitigación |
|---|---|---|---|---|
| R1 | Incremento inesperado del coste de servicios *cloud*. | Media | Medio | Apagar recursos cuando no se utilicen y revisar periódicamente AWS Cost Explorer. |
| R2 | Errores de configuración en infraestructura o Kubernetes. | Media | Alto | Utilizar infraestructura como código, pruebas incrementales y revisión de configuraciones antes del despliegue. |
| R3 | Exposición accidental de credenciales o secretos. | Baja | Alto | Evitar credenciales estáticas, usar servicios de gestión de secretos e identidades IAM. |

*Continúa en la siguiente página*

**Tabla 2.4:** (continuación)

| ID | Riesgo | Prob. | Impacto | Medida de mitigación |
|---|---|---|---|---|
| R4 | Fallos durante la migración hacia *Zero Trust*. | Media | Alto | Realizar una migración incremental, validando cada bloque mediante pruebas funcionales y de seguridad. |
| R5 | Pérdida o insuficiencia de evidencias para la evaluación. | Media | Medio | Registrar resultados, capturas, salidas de comandos y ficheros de pruebas durante cada fase. |
| R6 | Sobrecarga de tareas al concentrar el proyecto en un único perfil técnico. | Media | Medio | Priorizar tareas críticas, acotar el alcance y documentar la planificación real del trabajo. |
| R7 | Problemas de disponibilidad o cambios en servicios gestionados de AWS. | Baja | Medio | Diseñar la solución con servicios estándar y mantener alternativas de despliegue y documentación de configuración. |

A partir de los riesgos identificados, se observa que los riesgos de mayor impacto están asociados a la configuración de seguridad, la migración hacia *Zero Trust* y la posible exposición de credenciales. Por ello, las medidas de mitigación se centran en aplicar una migración incremental, utilizar infraestructura como código, reducir el uso de secretos estáticos y aplicar mínimos privilegios.

# Capítulo 3

# Solución base

## 3.1. Descripción de la solución

La solución desarrollada consiste en una aplicación web desplegada sobre infraestructura *cloud*, diseñada como sistema base para el posterior estudio de una migración hacia una arquitectura *Zero Trust*. El objetivo de esta solución no es construir desde el inicio una arquitectura completamente *Zero Trust*, sino disponer de una aplicación funcional, modular y representativa de un entorno *cloud* moderno sobre el que posteriormente puedan aplicarse mecanismos de mejora en seguridad, control de accesos, segmentación y verificación continua.

La aplicación base permite la gestión de eventos por parte de usuarios autenticados. Sus funcionalidades principales incluyen el registro e inicio de sesión de usuarios, la creación, consulta, edición y eliminación de eventos, la subida de imágenes asociadas a dichos eventos y la consulta del estado de procesamiento de la información. Además, el sistema incorpora un servicio interno de procesamiento *batch*, encargado de tratar periódicamente los eventos pendientes y generar información derivada a partir de los datos almacenados.

Desde el punto de vista arquitectónico, la solución se ha planteado como una aplicación distribuida compuesta por varios elementos diferenciados: una interfaz web, una API de *backend*, una base de datos relacional, un sistema de almacenamiento de objetos y un servicio de procesamiento interno. Esta separación permite representar de forma sencilla un escenario habitual en aplicaciones *cloud*: componentes desacoplados, comunicaciones internas entre servicios, almacenamiento persistente, exposición controlada hacia Internet y dependencia de servicios gestionados de pro-

veedor. La infraestructura de despliegue se ha implementado sobre AWS, utilizando diferentes servicios que se detallarán en las siguientes secciones.

Esta solución base constituye, por tanto, el punto de partida práctico del trabajo. A partir de ella se podrá analizar qué carencias presenta una arquitectura *cloud* convencional desde el punto de vista *Zero Trust* y qué cambios serían necesarios para evolucionarla hacia un modelo basado en mínima confianza, segmentación más estricta, autenticación y autorización continuas, control granular de comunicaciones y observabilidad avanzada.

## 3.2. El proceso de desarrollo

El desarrollo de la solución base se ha llevado a cabo siguiendo un proceso incremental, orientado a construir primero una aplicación funcional en entorno local y, posteriormente, desplegarla sobre infraestructura *cloud*. Este enfoque permite separar de forma clara las decisiones propias del desarrollo *software* de las decisiones asociadas al despliegue, la infraestructura y la operación del sistema. Por lo tanto, la sección se divide en cuatro grandes bloques, que son: análisis inicial, diseño, implementación de la aplicación base mediante proceso incremental y pruebas que validan el desarrollo.

### 3.2.1. Análisis

#### Definición y especificación de requisitos

La definición de requisitos se divide en requisitos funcionales y no funcionales, teniendo en cuenta la descripción anterior. Los requisitos funcionales, dispuestos en la Tabla 3.1 se centran en lo que hace la aplicación para usuarios y servicios internos, mientras que los requisitos no funcionales, mostrados en la Tabla 3.2 se centran en propiedades de calidad, despliegue *cloud*, seguridad base, operación y escalabilidad.

**Tabla 3.1:** Requisitos funcionales del sistema base

| RF | Título | Descripción |
|---|---|---|
| RF-1 | Registro de usuarios | El sistema debe permitir a un usuario no registrado crear una nueva cuenta proporcionando la información necesaria para su alta en la plataforma. |

*Continúa en la siguiente página*

**Tabla 3.1:** (continuación)

| RF | Título | Descripción |
|---|---|---|
| RF-2 | Autenticación de usuarios | El sistema debe permitir a un usuario registrado iniciar sesión mediante credenciales válidas, verificando su identidad. |
| RF-3 | Finalización de sesión | El sistema debe permitir a un usuario autenticado cerrar su sesión activa de forma segura. |
| RF-4 | Creación de eventos | El sistema debe permitir a los usuarios autenticados crear nuevos eventos con su información asociada (título, descripción, fuente, fecha, país, idioma y estado). |
| RF-5 | Consulta de listado de eventos | El sistema debe permitir a los usuarios autenticados visualizar el conjunto de eventos registrados en la plataforma. |
| RF-6 | Consulta de detalle de evento | El sistema debe permitir a los usuarios visualizar la información completa de un evento concreto. |
| RF-7 | Edición de eventos | El sistema debe permitir a los usuarios autenticados modificar la información de eventos existentes. |
| RF-8 | Eliminación de eventos | El sistema debe permitir a los usuarios autenticados eliminar eventos previamente registrados. |
| RF-9 | Subida de imágenes a eventos | El sistema debe permitir a los usuarios autenticados subir imágenes y asociarlas a un evento existente. |
| RF-10 | Visualización de imágenes de eventos | El sistema debe permitir consultar y visualizar las imágenes asociadas a un evento. |
| RF-11 | Procesamiento *batch* de eventos | El sistema debe disponer de un servicio *batch* que procese periódicamente los eventos pendientes de tratamiento. |
| RF-12 | Generación automática de metadatos | El servicio *batch* debe generar metadatos o descripciones automáticas para los eventos procesados a partir de la información disponible. |

*Continúa en la siguiente página*

**Tabla 3.1:** (continuación)

| RF | Título | Descripción |
|---|---|---|
| RF-13 | Consulta del estado del procesamiento | El sistema debe permitir consultar el estado de ejecución de los procesos *batch* y su progreso. |
| RF-14 | Registro de ejecuciones *batch* | El sistema debe almacenar información de cada ejecución *batch*, incluyendo fechas, estado y resultados del procesamiento. |
| RF-15 | Gestión de eventos procesados | El sistema debe registrar el resultado del procesamiento *batch* en los eventos, incluyendo metadatos generados y estado de procesamiento. |
| RF-16 | Asociación de imágenes a eventos | El sistema debe mantener la relación entre las imágenes almacenadas y los eventos a los que pertenecen. |
| RF-17 | Almacenamiento de imágenes | El sistema debe almacenar los ficheros de imagen en el servicio de almacenamiento y registrar su información asociada en base de datos. |
| RF-18 | Recuperación de imágenes | El sistema debe permitir recuperar las imágenes almacenadas a partir de su referencia para su visualización. |
| RF-19 | Integración entre servicios | La aplicación debe coordinar la comunicación entre la API, la base de datos, el servicio de almacenamiento y el servicio *batch* para completar las operaciones del sistema. |
| RF-20 | Asociación de autoría | El sistema debe registrar qué usuario ha creado cada evento y cada imagen, garantizando la trazabilidad de las acciones. |

**Tabla 3.2:** Requisitos no funcionales del sistema base

| RNF | Título | Descripción |
|---|---|---|
| RNF-1 | Despliegue en entorno *cloud* | El sistema debe desplegarse sobre infraestructura *cloud*, concretamente en AWS, utilizando una VPC como entorno de red aislado. |

*Continúa en la siguiente página*

**Tabla 3.2:** (continuación)

| RNF | Título | Descripción |
|---|---|---|
| RNF-2 | Arquitectura distribuida | El sistema debe estar organizado en componentes desacoplados (Web/API, base de datos, almacenamiento y *batch*). |
| RNF-3 | Contenerización | Los componentes de la aplicación deben ejecutarse en contenedores para garantizar portabilidad y consistencia. |
| RNF-4 | Orquestación de contenedores | La gestión y despliegue de contenedores debe realizarse mediante un orquestador como AWS EKS. |
| RNF-5 | Separación de red | La infraestructura debe dividirse en subredes públicas y privadas, aislando los componentes internos del acceso directo desde Internet. |
| RNF-6 | Acceso externo controlado | El acceso a la aplicación desde Internet debe realizarse a través de servicios controlados como balanceadores de carga y *gateways*. |
| RNF-7 | Persistencia gestionada | La base de datos debe desplegarse como un servicio gestionado que garantice la persistencia y disponibilidad de los datos. |
| RNF-8 | Almacenamiento escalable | El sistema debe utilizar un servicio de almacenamiento de objetos escalable para gestionar las imágenes (por ejemplo, S3). |
| RNF-9 | Disponibilidad del sistema | La aplicación debe estar disponible para los usuarios durante su uso normal y permitir la ejecución periódica de procesos *batch*. |
| RNF-10 | Escalabilidad horizontal | La arquitectura debe permitir escalar los componentes de aplicación horizontalmente según la carga de trabajo. |
| RNF-11 | Rendimiento | Las operaciones principales (*login*, consulta de eventos, subida de imágenes) deben ejecutarse en tiempos de respuesta adecuados. |
| RNF-12 | Seguridad de credenciales | Las contraseñas de usuario deben almacenarse utilizando algoritmos de hash seguros y nunca en texto plano. |

*Continúa en la siguiente página*

**Tabla 3.2:** (continuación)

| RNF | Título | Descripción |
|---|---|---|
| RNF-13 | Gestión de secretos | Las credenciales y configuraciones sensibles deben gestionarse mediante un servicio específico como AWS Secrets Manager. |
| RNF-14 | Gestión de identidades y accesos | El acceso a los recursos *cloud* debe controlarse mediante políticas y roles definidos en AWS IAM. |
| RNF-15 | Monitorización | El sistema debe disponer de mecanismos de monitorización y registro de *logs* para supervisar su estado y detectar incidencias. |
| RNF-16 | Trazabilidad | El sistema debe registrar información suficiente para auditar acciones relevantes como creación de eventos o ejecución de *batch*. |
| RNF-17 | Integridad de datos | El sistema debe garantizar la coherencia entre los datos almacenados en base de datos y los ficheros almacenados en el sistema de objetos. |
| RNF-18 | Fiabilidad del procesamiento *batch* | Los procesos *batch* deben registrar correctamente su estado, resultados y posibles errores para facilitar su análisis y recuperación. |
| RNF-19 | Mantenibilidad | El sistema debe estar diseñado de forma modular para facilitar su evolución, mantenimiento y pruebas. |
| RNF-20 | Base para evolución a *Zero Trust* | La arquitectura debe permitir su evolución futura hacia un modelo *Zero Trust* mediante la clara separación de componentes y control de accesos. |

Estos requisitos se plasman en el diagrama de las Figuras 3.1 y 3.2, con el objetivo de mejorar la legibilidad y facilitar la trazabilidad con los componentes del sistema.

### Casos de uso de la aplicación

En los casos de uso que presenta la aplicación base, aparecen representados **cuatro actores** diferentes, los cuales son:

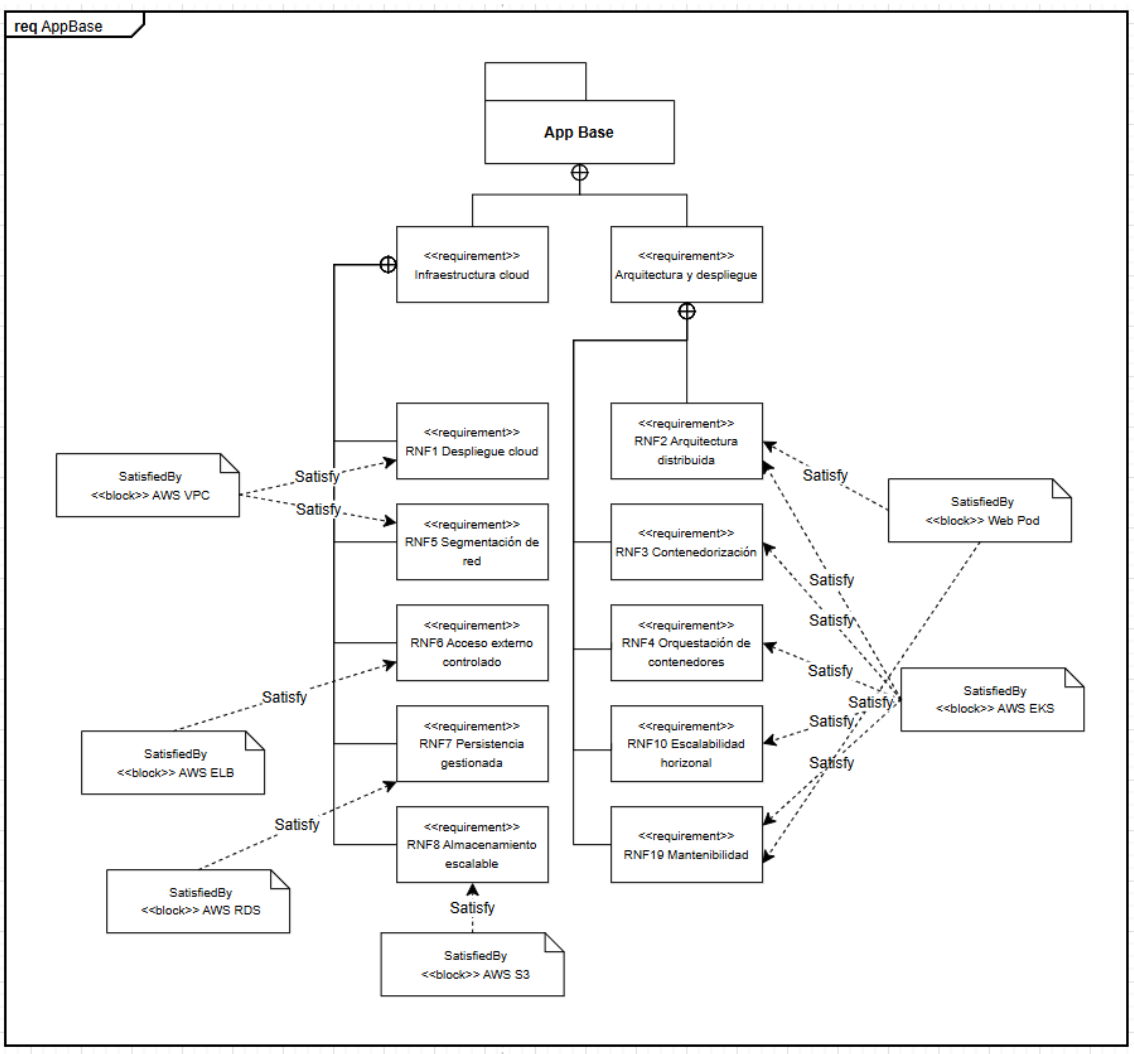



**Figura 3.1:** Diagrama SysML de requisitos no funcionales de la aplicación base. Bloque de infraestructura, arquitectura y despliegue. Fuente: elaboración propia.

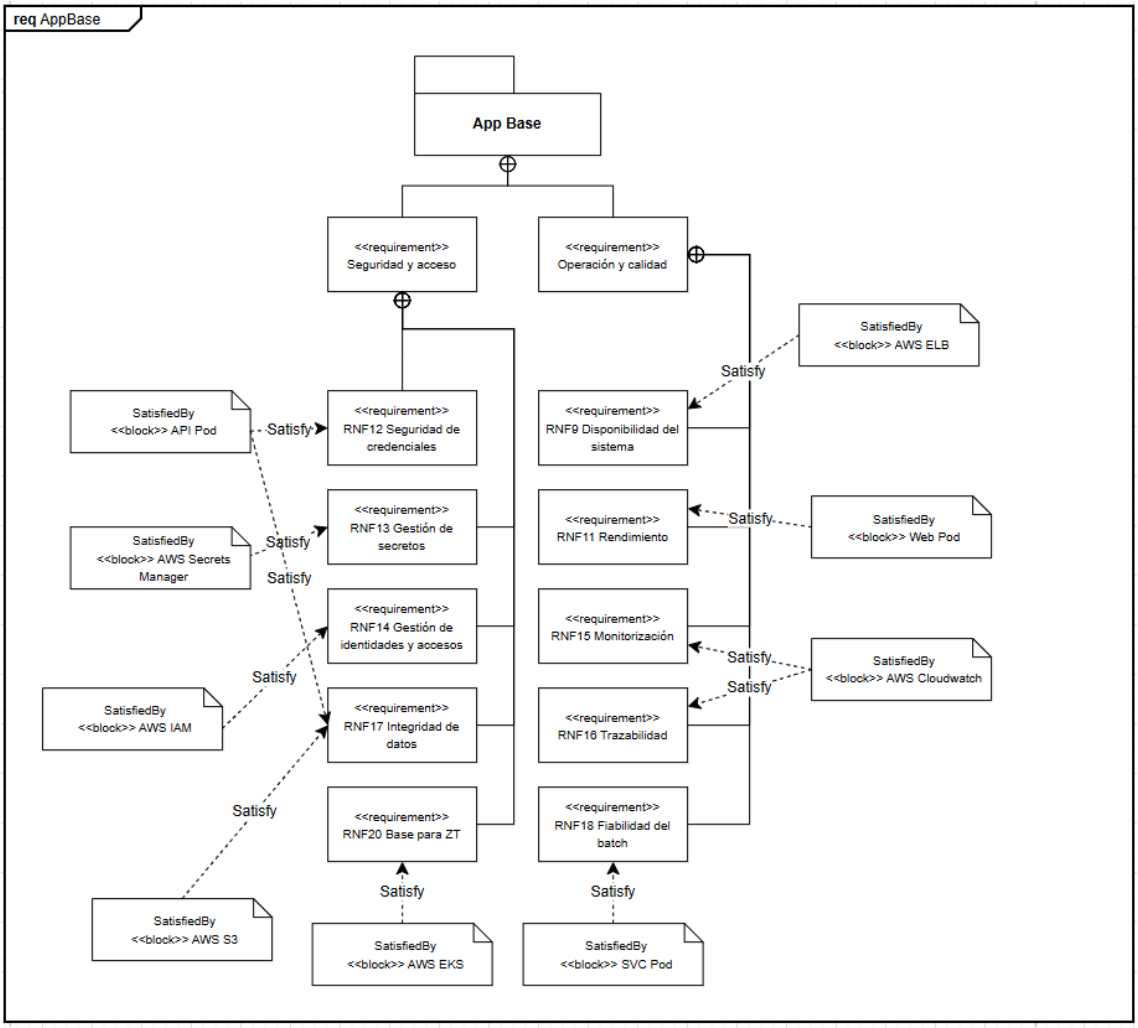



**Figura 3.2:** Diagrama SysML de requisitos no funcionales de la aplicación base. Bloque de seguridad, acceso, operación y calidad. Fuente: elaboración propia.

- $A_1$: Usuario no registrado. Corresponde a cualquier usuario que accede a la aplicación sin disponer todavía de una cuenta.
- $A_2$: Usuario registrado. Representa a un usuario que dispone de una cuenta válida en el sistema y que ha sido autenticado correctamente. Este actor puede acceder a las funcionalidades principales de la aplicación.
- $A_3$: SVC *Batch*. Servicio interno encargado de ejecutar procesos automáticos de forma periódica. Su función principal es procesar los eventos almacenados en el sistema con el objetivo de generar metadatos derivados a partir de la información disponible.
- $A_4$: SVC Almacenamiento. Servicio responsable de gestionar el almacenamiento de los archivos asociados a la aplicación.

La descripción de los casos de uso implicados se detalla en la Tabla 3.3.

**Tabla 3.3:** Casos de uso del sistema base

| UC | Título | Descripción |
|---|---|---|
| UC-1 | Registrar usuario | Permite crear una nueva cuenta en el sistema |
| UC-2 | Iniciar sesión | Permite a un usuario autenticarse en la aplicación mediante sus credenciales |
| UC-3 | Cerrar sesión | Permite al usuario finalizar su sesión activa en la aplicación |
| UC-4 | Crear evento | Permite a un usuario registrado crear un nuevo evento |
| UC-5 | Listar eventos | Permite al usuario visualizar el listado de sus eventos registrados |
| UC-6 | Ver detalle de evento | Permite consultar la información completa de un evento |
| UC-7 | Editar evento | Permite modificar la información de un evento existente |
| UC-8 | Eliminar evento | Permite eliminar un evento previamente creado |
| UC-9 | Subir imagen a evento | Permite asociar una imagen a un evento existente |
| UC-10 | Ver imágenes de evento | Permite visualizar las imágenes asociadas a un evento |
| UC-11 | Procesar eventos en *batch* | Permite al servicio interno procesar periódicamente los eventos para generar metadatos |
| UC-12 | Generar metadatos del evento | El *batch* genera los metadatos automáticamente para un evento a partir de las imágenes |
| UC-13 | Consultar estado del procesamiento | Permite comprobar las variables involucradas en el *batch* |

Finalmente, el diagrama de casos de uso del sistema se muestra en la Figura 3.3.

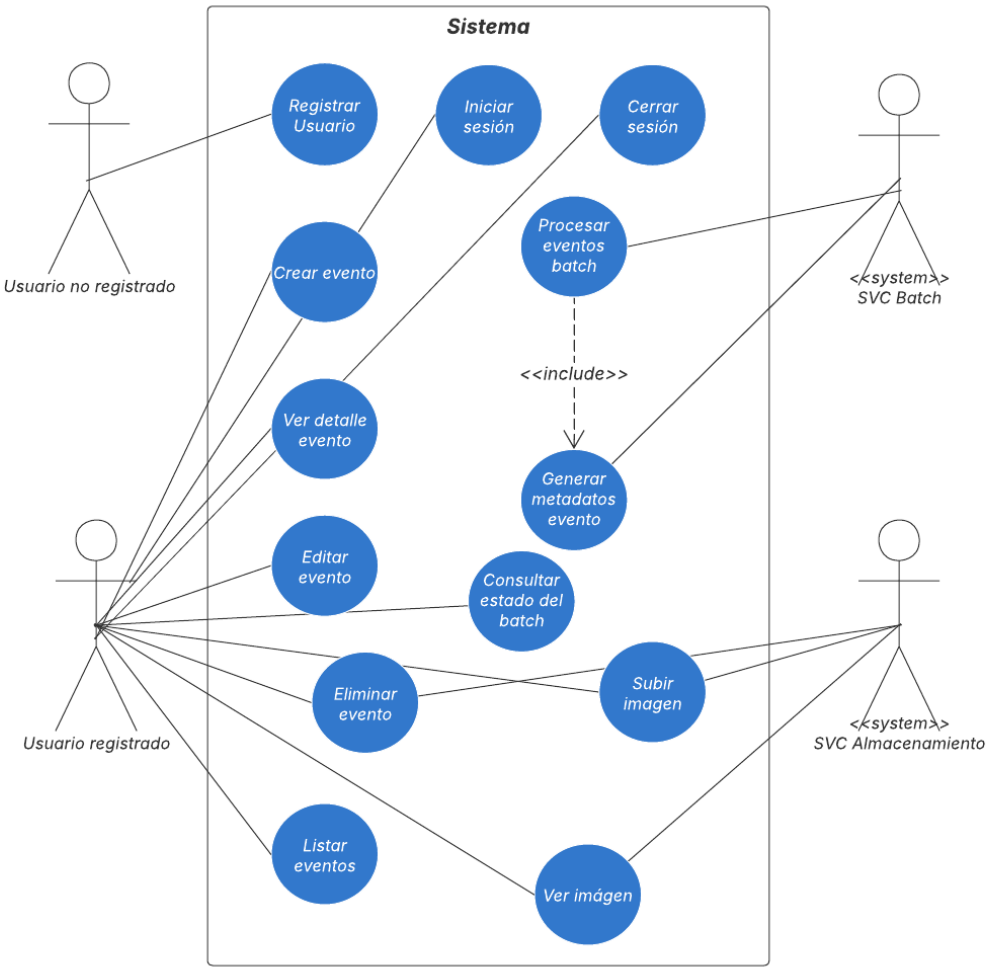



**Figura 3.3:** Diagrama UML de casos de uso de la aplicación base. Fuente: elaboración propia.

### Diagrama de secuencia

El diagrama de secuencia detallado en la Figura 3.4 muestra el flujo completo del uso de un usuario en la aplicación. El funcionamiento principal se puede dividir en seis grandes caminos:

- 1–4: Registro
- 5–8: *Login*
- 9–16: Gestión de eventos
- 17–26: Gestión de imágenes
- 27–33: *Batch*
- 34–37: Consulta del estado

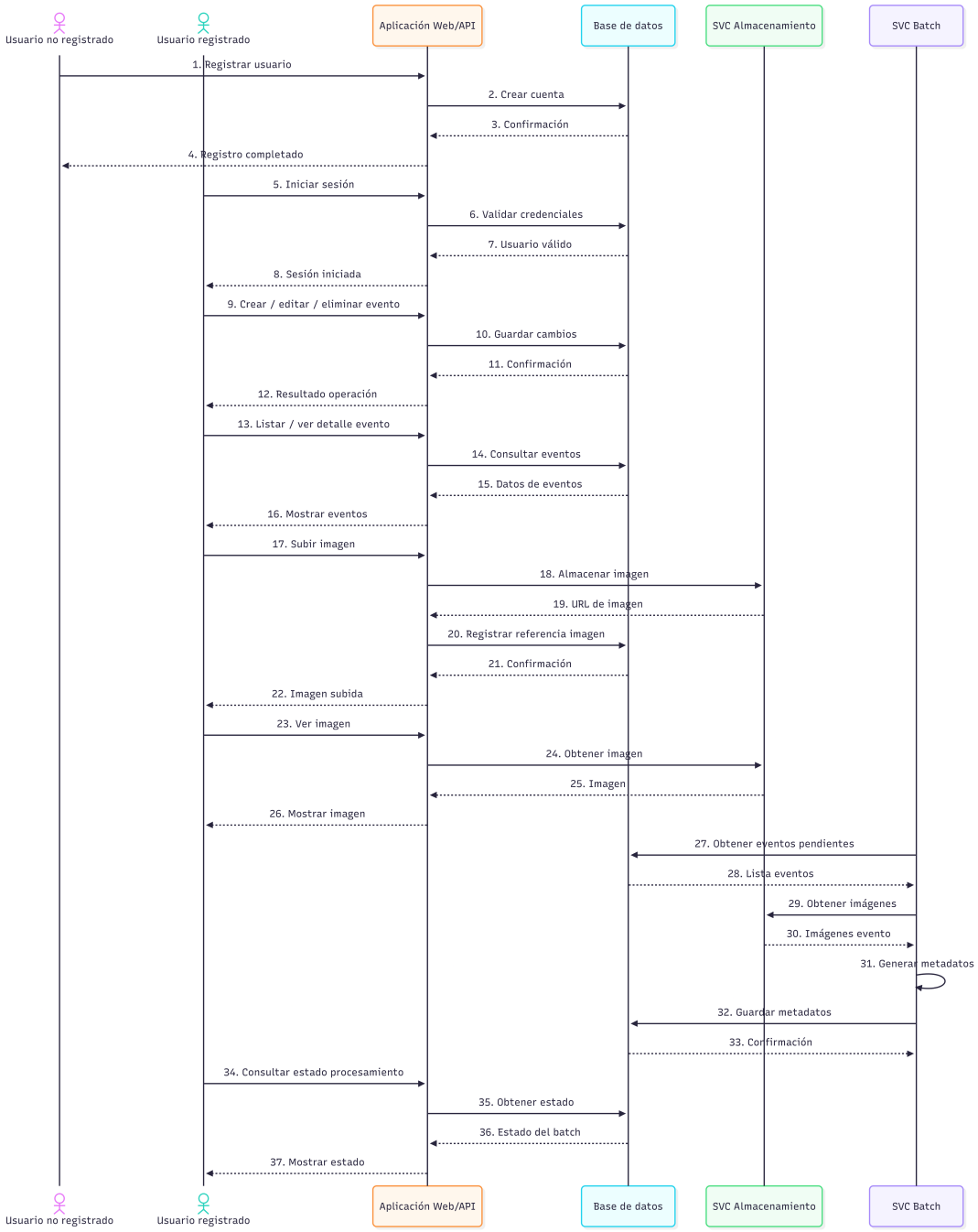



**Figura 3.4:** Diagrama UML de secuencia de la aplicación base. Fuente: elaboración propia.

### 3.2.2. Diseño

#### Diseño de sistema

En esta fase se establecen los componentes principales, las relaciones entre ellos y los servicios *cloud* necesarios para cumplir los requisitos no funcionales y ejecutar la solución en un entorno AWS. El objetivo de este diseño es disponer de una arquitectura modular, desplegable en la nube y suficientemente representativa de una aplicación *cloud* moderna, de forma que posteriormente pueda utilizarse como punto de partida para su evolución hacia un modelo *Zero Trust*. En este diseño inicial se contemplan los siguientes elementos, servicios y tecnologías:

- Amazon Route 53
- Amazon VPC (Virtual Private Cloud)
- Internet Gateway (IGW)
- Subred pública
- Elastic Load Balancing (ELB)
- NAT Gateway
- Subred privada
- Amazon EKS (Elastic Kubernetes Service)
- Amazon RDS (Relational Database Service)
- Amazon S3 (Simple Storage Service)
- Amazon CloudWatch
- AWS IAM (Identity and Access Management)
- AWS Secrets Manager

La arquitectura se organiza en torno a una red privada virtual en AWS, dentro de la cual se diferencian zonas públicas y privadas. Los elementos expuestos hacia Internet se sitúan en la parte pública de la infraestructura, mientras que el resto de los servicios desplegados dentro de la VPC se ubican en zonas privadas, reduciendo así la exposición directa de los componentes internos. Esta separación permite establecer un primer nivel de aislamiento mediante subredes, tablas de rutas y grupos de seguridad.

Desde el punto de vista funcional, el sistema se compone de tres servicios principales de aplicación: el servicio web, la API y el servicio *batch*. Estos componentes se despliegan como contenedores dentro de un clúster de Kubernetes gestionado mediante EKS.

La persistencia de la información se divide en dos mecanismos diferenciados. Por un lado, Amazon RDS proporciona una base de datos relacional PostgreSQL. Por otro lado, Amazon S3 se utiliza como almacenamiento de objetos.

El acceso externo a la aplicación se plantea a través de servicios controlados de entrada. Amazon Route 53 se encarga de la resolución DNS, mientras que el tráfico procedente de Internet accede a la VPC mediante un Internet Gateway y es dirigido hacia un balanceador de carga. Este balanceador actúa como punto de entrada hacia los servicios desplegados en Kubernetes, evitando la exposición directa de los *pods* de aplicación.

Además, la arquitectura incorpora servicios de soporte transversales. AWS IAM se utiliza para gestionar permisos y roles de acceso a los recursos de AWS. AWS Secrets Manager permite almacenar credenciales y configuraciones sensibles sin incluirlas directamente en el código fuente o en los manifiestos de despliegue. Por último, Amazon CloudWatch proporciona capacidades de observabilidad mediante la recolección de *logs* y métricas de los servicios desplegados.

El diagrama de arquitectura del sistema base se presenta en la Figura 3.5.

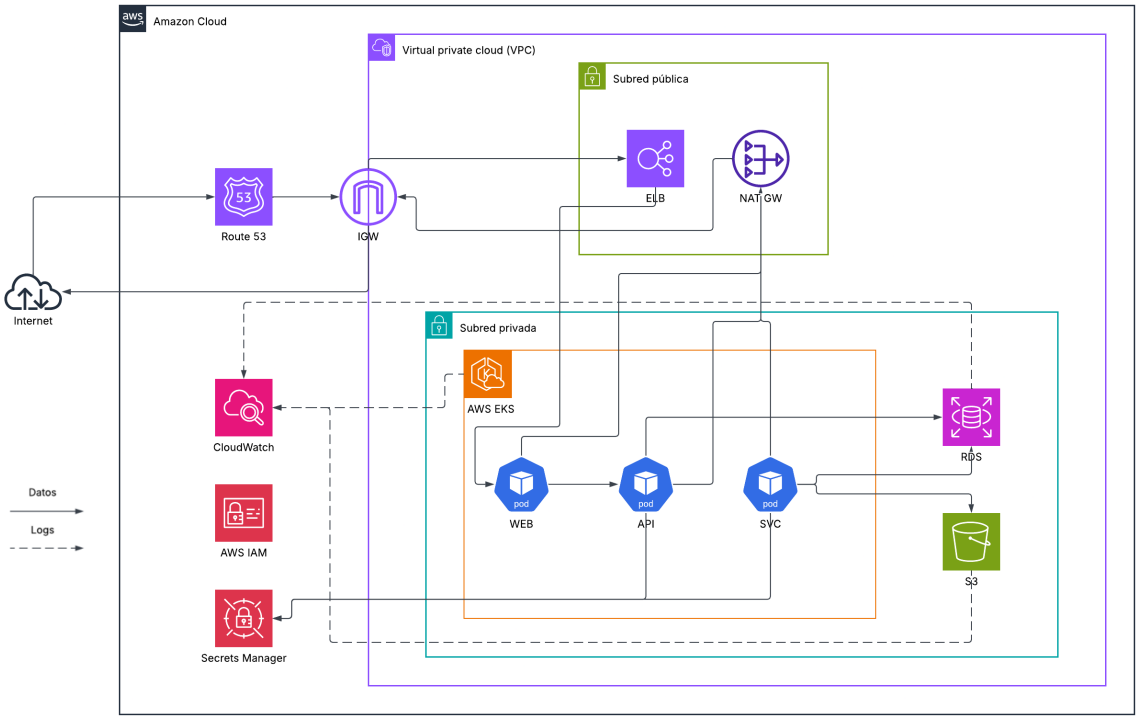



**Figura 3.5:** Diagrama de arquitectura *cloud* de la aplicación base. Fuente: elaboración propia.

### Diseño detallado

En este apartado se concreta la organización interna de la aplicación base a partir de la arquitectura definida en el diseño de sistema. Para ello, se describen las principales capas lógicas del sistema: la capa de persistencia, encargada del almacenamiento de datos; la capa de modelo y lógica de negocio, responsable de representar las entidades principales y coordinar las operaciones de la aplicación; y la capa de interfaz, que permite la interacción del usuario con las funcionalidades disponibles.

La **capa de persistencia** del sistema se encarga de almacenar y gestionar de forma estructurada toda la información utilizada por la aplicación. La tabla *users* almacena la información necesaria para la autenticación y gestión de los usuarios del sistema. La tabla *events* representa los eventos registrados en la aplicación, incluyendo tanto la información introducida manualmente por los usuarios como los datos generados posteriormente por el proceso *batch*. Las imágenes asociadas a cada evento se gestionan mediante la tabla *images*, que mantiene una relación de 1-N con la tabla de eventos. Finalmente, la tabla *batch_ executions* registra las ejecuciones del proceso *batch*, permitiendo almacenar información sobre el estado de cada ejecución, las métricas de procesamiento y los posibles errores producidos. De esta forma, el modelo permite diferenciar claramente entre los datos introducidos manualmente y la

información generada automáticamente durante el procesamiento. En la Figura 3.6 se pueden visualizar las relaciones y atributos de las tablas implicadas.




**Figura 3.6:** Diagrama entidad relación del modelo de persistencia de la aplicación base. Fuente: elaboración propia.

La **capa de modelo** representa los conceptos principales del dominio de la aplicación y actúa como punto intermedio entre la persistencia y las operaciones expuestas por la API. En esta capa se definen las entidades fundamentales del sistema, como usuarios, eventos, imágenes y ejecuciones *batch*, junto con las relaciones existentes entre ellas. El usuario se asocia a los eventos que crea, cada evento puede tener una o varias imágenes asociadas y las ejecuciones *batch* permiten registrar el procesamiento periódico realizado sobre los datos almacenados.

Sobre este modelo se construye la lógica de negocio de la aplicación. Las operaciones principales incluyen la validación de usuarios, la gestión del ciclo de vida de los eventos, la asociación de imágenes, la actualización del estado de procesamiento y el registro de los resultados generados por el servicio *batch*. Esta separación permite

que la API no acceda directamente a la base de datos de forma desorganizada, sino a través de una estructura coherente de entidades, servicios y operaciones.

El diseño del modelo también facilita la separación entre acciones realizadas por usuarios y acciones realizadas por servicios internos. Las operaciones de creación, edición, consulta o eliminación de eventos corresponden principalmente a la interacción del usuario autenticado con la aplicación, mientras que la generación de metadatos y la actualización del estado de procesamiento corresponden al servicio *batch*. Esta distinción resulta importante para mantener la trazabilidad de las acciones y para preparar la arquitectura de cara a futuras mejoras de control de acceso.

La **capa de interfaz** está formada por la aplicación web utilizada por los usuarios para interactuar con el sistema. Esta interfaz proporciona las pantallas necesarias para el registro e inicio de sesión, la consulta del listado de eventos, la creación de nuevos eventos, la edición de eventos existentes, la visualización de su detalle y la gestión de las imágenes asociadas. Su función principal es ofrecer una entrada sencilla a las capacidades de la API, ocultando al usuario la complejidad de la infraestructura *cloud* y de los servicios internos.

La comunicación entre la interfaz y el *backend* se realiza mediante peticiones HTTP hacia la API. De esta forma, la interfaz no accede directamente ni a la base de datos ni al almacenamiento de objetos, sino que delega todas las operaciones en el *backend*. Este enfoque permite centralizar la lógica de validación, autenticación y autorización en la API, manteniendo una separación clara entre presentación y lógica de negocio.

Desde el punto de vista de la **capa de infraestructura de ejecución**, las capas lógicas anteriores se materializan sobre los servicios *cloud* definidos en el diseño de sistema. La interfaz se despliega como un componente web contenerizado, la lógica de negocio se expone mediante la API y el procesamiento interno se ejecuta mediante el servicio *batch*; todos ellos se ejecutan como cargas de trabajo dentro del clúster de Kubernetes gestionado por Amazon EKS. La persistencia estructurada se delega en Amazon RDS, mientras que los ficheros de imagen se almacenan en Amazon S3. Además, los servicios auxiliares de AWS proporcionan soporte transversal para la gestión de permisos, secretos y observabilidad.

En conjunto, el diseño detallado mantiene una separación entre persistencia, modelo, interfaz e infraestructura de ejecución. La base de datos conserva la información estructurada, el modelo organiza las entidades y operaciones principales, la interfaz proporciona el punto de interacción con el usuario y la infraestructura *cloud*

materializa estos componentes en un entorno distribuido. Esta división favorece la mantenibilidad de la aplicación y facilita la evolución independiente de cada parte del sistema.

### 3.2.3. Implementación

La implementación de la aplicación base se realizó de forma incremental, separando el desarrollo funcional de la aplicación, su contenerización, el despliegue inicial en AWS y la posterior automatización mediante infraestructura como código. Esta separación permitió validar primero el comportamiento de la aplicación en un entorno local controlado y, posteriormente, trasladarla progresivamente a una arquitectura *cloud* más cercana al escenario final del trabajo. El proceso completo puede agruparse en tres grandes bloques de trabajo:

- Desarrollo de la aplicación en entorno local, utilizando contenedores para simular los servicios principales necesarios: *backend*, *frontend*, base de datos, almacenamiento de objetos y servicio de procesamiento interno. Esta fase permitió implementar y validar los casos de uso principales sin depender todavía de la infraestructura *cloud*. Una vez comprobado el funcionamiento local, se adaptó el proyecto para su ejecución en AWS, incorporando imágenes Docker, manifiestos de Kubernetes, configuración para servicios gestionados y *scripts* auxiliares de despliegue.
- Despliegue manual en AWS mediante la consola del proveedor. Esta fase permitió comprender de forma práctica las dependencias entre los servicios utilizados, como la red VPC, las subredes públicas y privadas, los grupos de seguridad, el clúster EKS, los repositorios ECR, la base de datos RDS, el *bucket* S3 y los permisos IAM.
- Despliegue automático de la infraestructura mediante Terraform, transformando la configuración desplegada en recursos gestionados como infraestructura como código (IaC), consiguiendo así un mayor control y versionado de los recursos utilizados.

#### Herramientas y tecnologías utilizadas

Para el desarrollo y despliegue de la aplicación base se ha hecho uso de un gran repertorio de herramientas y tecnologías. Aunque ya se hayan comentado previamente, en la Tabla 3.4 se detallan el ámbito y uso de dichas herramientas en el ciclo de implementación.

**Tabla 3.4:** Herramientas y tecnologías utilizadas en la implementación de la aplicación base

| **Ámbito** | **Herramientas y tecnologías** | **Uso en el proyecto** |
|---|---|---|
| *Frontend* | React, Vite | Desarrollo de la interfaz web de usuario para la gestión de eventos, autenticación y visualización de imágenes. |
| *Backend* | FastAPI, SQLAlchemy, Alembic | Implementación de la API, lógica de negocio, acceso a datos y gestión de migraciones de base de datos. |
| Base de datos | PostgreSQL | Almacenamiento estructurado de usuarios, eventos, imágenes y ejecuciones del proceso *batch*. |
| Almacenamiento de objetos | MinIO, Amazon S3 | MinIO se utilizó en local para simular un servicio compatible con S3; Amazon S3 se utilizó en el despliegue *cloud*. |
| Procesamiento interno | Servicio *svc* en Python | Ejecución de tareas periódicas de procesamiento sobre eventos e imágenes almacenadas. |
| Contenerización | Docker, Docker Compose | Construcción y ejecución local de los servicios de la aplicación mediante contenedores. |
| Orquestación | Kubernetes, Amazon EKS | Despliegue de los componentes de aplicación como cargas de trabajo contenerizadas en un clúster gestionado. |
| Registro de imágenes | Amazon ECR | Almacenamiento de las imágenes Docker necesarias para desplegar los servicios en AWS. |
| Infraestructura como código | Terraform | Definición, importación y automatización de los recursos de infraestructura en AWS. |
| Automatización auxiliar | PowerShell, *scripts* Shell | Automatización de tareas de construcción, etiquetado, subida de imágenes y despliegue de componentes. |

*Continúa en la siguiente página*

**Tabla 3.4:** (continuación)

| **Ámbito** | **Herramientas y tecnologías** | **Uso en el proyecto** |
|---|---|---|
| Observabilidad y permisos | CloudWatch, IAM, AWS Secrets Manager | Gestión de *logs*, permisos de acceso, roles y secretos utilizados por la aplicación y la infraestructura. |
| Control de versiones | Git, GitHub | Gestión del histórico de cambios, ramas, documentación y evolución incremental del proyecto. |

### Proceso de implementación

Como ya se ha indicado anteriormente, el proceso de implementación ha sido ejecutado de manera incremental, cambiando de fase una vez conseguido una serie de hitos. Las fases principales se detallan en la Tabla 3.5.

**Tabla 3.5:** Fases principales de implementación de la aplicación base

| **Fase** | **Descripción** |
|---|---|
| Inicialización del proyecto | Creación del repositorio, configuración inicial, documentación básica y definición de la estructura general del proyecto. |
| Desarrollo funcional local | Implementación del inicio de sesión, creación básica de eventos, operaciones de creación, consulta, modificación y eliminación, y almacenamiento de imágenes en MinIO como simulación local de S3. |
| Servicio *batch* | Desarrollo del servicio interno de procesamiento, conexión con base de datos y almacenamiento de objetos, y actualización del estado de los eventos procesados. |
| Contenerización | Creación de Dockerfiles para los servicios de aplicación, reducción del tamaño de imágenes y preparación de los contenedores para su ejecución local y en la nube. |
| Adaptación a entorno *cloud* | Incorporación de manifiestos de Kubernetes, cambios de configuración necesarios para AWS y *scripts* de construcción y subida de imágenes a Amazon ECR. |

*Continúa en la siguiente página*

**Tabla 3.5:** (continuación)

| Fase | Descripción |
|---|---|
| Despliegue manual en AWS | Creación y configuración manual de los recursos principales: VPC, subredes, grupos de seguridad, EKS, ECR, RDS, S3, balanceador de carga, IAM y servicios auxiliares. |
| Automatización con Terraform | Importación y definición progresiva de la infraestructura en Terraform, incluyendo red, ECR, S3, RDS, NAT, VPC Endpoint, EKS, grupos de nodos e IAM. |
| Automatización mediante *scripts* | Desarrollo de *scripts* PowerShell y Shell para automatizar la construcción de imágenes, subida a ECR y despliegue de la aplicación en el entorno *cloud*. |
| Documentación final | Elaboración de documentación específica para despliegue manual, despliegue automático, *scripts*, Terraform, Kubernetes y README principal del repositorio. |

#### Automatización del despliegue

El proceso de automatización con Terraform se realizó de forma progresiva. Parte de la infraestructura fue creada inicialmente de forma manual para validar la arquitectura y comprender las dependencias entre recursos. Después, estos recursos fueron incorporados al estado de Terraform mediante importaciones y definiciones equivalentes en código. Este enfoque permitió evitar la destrucción y recreación innecesaria de recursos, manteniendo la infraestructura existente y convirtiéndola gradualmente en infraestructura gestionada como código.

Finalmente, se desarrollaron *scripts* auxiliares para reducir tareas repetitivas del despliegue. Estos *scripts* permiten construir las imágenes Docker, etiquetarlas, autenticarse contra Amazon ECR, subir las imágenes al registro y facilitar la actualización de los componentes desplegados. De esta forma, las tareas principales del proceso de despliegue quedan automatizadas y documentadas, reduciendo errores manuales y mejorando la reproducibilidad del entorno.

### 3.2.4. Pruebas

La validación de la aplicación base se realizó mediante una combinación de pruebas funcionales, pruebas automáticas y pruebas de sistema. Las pruebas descritas en esta sección se centran en comprobar que los casos de uso definidos en la fase de aná-

lisis se cumplen correctamente y que los componentes de la aplicación interactúan de forma adecuada.

El proceso de pruebas siguió el flujo definido en el diagrama de secuencia de la Figura 3.4. A medida que se completaban las funcionalidades de la aplicación, se verificó que cada operación producía el resultado esperado y que no generaba errores en los servicios implicados. Esta validación incluyó tanto el entorno local como el entorno desplegado en AWS, permitiendo comprobar la transición desde una ejecución basada en Docker Compose hasta una ejecución sobre infraestructura *cloud*.

Las pruebas se agruparon en tres bloques principales: pruebas de autenticación, pruebas de gestión de eventos e imágenes, y pruebas de integración con los servicios internos y externos. En las pruebas de autenticación se comprobó el registro y el inicio de sesión de usuarios. El cierre de sesión no se validó como operación de *backend*, ya que la API no expone un *endpoint* específico para ello y la sesión se gestiona mediante un *token stateless* con expiración. En las pruebas de eventos se validaron las operaciones de creación, consulta, modificación y eliminación. Finalmente, en las pruebas de integración se verificó la subida y visualización de imágenes, la comunicación con el almacenamiento de objetos, la persistencia en base de datos y la ejecución del servicio *batch*.

Para validar los casos de uso definidos en la fase de análisis, se implementó una batería de pruebas automáticas basada en *pytest*. Estas pruebas se centraron en la API, el servicio de procesamiento *batch* y un flujo completo de integración, evitando dependencias innecesarias con la interfaz gráfica o con servicios *cloud* externos. El objetivo fue comprobar de forma reproducible el comportamiento funcional del sistema en un entorno local controlado. La Tabla 3.6 muestra la relación entre los casos de uso y las pruebas automáticas implementadas.

**Tabla 3.6:** Relación entre casos de uso y pruebas automáticas

| UC | Prueba automática | Resultado |
|---|---|---|
| UC-1 | test_register_user_success | ✓ |
| UC-2 | test_login_user_success | ✓ |
| UC-3 | No aplicable a nivel *backend*. La API no expone *endpoint* de *logout* y la sesión se gestiona mediante *token stateless* con expiración | No aplica |
| UC-4 | test_create_event | ✓ |
| UC-5 | test_list_events | ✓ |
| UC-6 | test_get_event_detail | ✓ |
| UC-7 | test_update_event | ✓ |
| UC-8 | test_delete_event | ✓ |
| UC-9 | test_upload_image_to_event | ✓ |
| UC-10 | test_list_event_images | ✓ |
| UC-11 | test_run_batch_once_processes_event | ✓ |
| UC-12 | test_generate_event_description _includes_metadata | ✓ |
| UC-13 | test_get_batch_status | ✓ |
| Flujo completo | test_full_event_flow | ✓ |

Además de la validación funcional, se realizaron pruebas de sistema para comprobar el comportamiento conjunto de los componentes. Estas pruebas tuvieron como objetivo verificar que la aplicación no solo funcionaba de forma aislada, sino también como sistema distribuido, con comunicación entre *frontend*, API, base de datos, almacenamiento de objetos y servicio *batch*.

En el entorno local se realizaron dos tipos de validaciones. Por un lado, las pruebas automáticas se ejecutaron con una base de datos SQLite de test y con el almacenamiento S3 simulado mediante *mocks*, evitando dependencias externas. Por otro lado, las pruebas funcionales y de sistema se realizaron sobre el entorno contenerizado con Docker Compose, PostgreSQL y MinIO, reproduciendo el comportamiento general del sistema antes de su despliegue en AWS. Posteriormente, las mismas operaciones principales se validaron en el entorno *cloud*, sustituyendo los servicios locales por los recursos gestionados correspondientes, como Amazon RDS, Amazon S3, Amazon EKS y Amazon ECR.

Durante estas pruebas se revisaron también los *logs* de los servicios desplegados, los estados de los *pods* de Kubernetes y la conectividad entre componentes. En particular, se comprobó que los servicios de aplicación podían acceder correctamente a la base de datos, que las imágenes podían almacenarse y recuperarse desde el almacenamiento de objetos y que el servicio *batch* podía ejecutar sus operaciones sobre los datos persistidos.

## 3.3. El producto del desarrollo

### 3.3.1. Herramienta desarrollada

La herramienta desarrollada materializa la solución base planteada en el trabajo mediante una aplicación web funcional y desplegable en la nube. Su finalidad es ofrecer un escenario práctico sobre el que representar una arquitectura *cloud* convencional, incorporando interacción de usuario, persistencia de datos, almacenamiento de ficheros y procesamiento interno. De este modo, el producto obtenido no se limita a una demostración aislada, sino que integra los elementos habituales de una aplicación distribuida: interfaz web, API, base de datos, almacenamiento de objetos, servicio *batch* y recursos de despliegue.

### 3.3.2. Diseño final

El diseño final de la aplicación base, dispuesto en la Figura 3.7, recoge la arquitectura resultante tras el proceso de implementación, despliegue manual y automatización de la infraestructura. A diferencia del diseño inicial, esta versión refleja únicamente los servicios finalmente utilizados en la solución desplegada, ajustando la posición de algunos recursos y simplificando aquellos elementos que no formaron parte del producto final, concretamente:

- Se suprimen dos servicios que no se han llegado a utilizar, tales como AWS Secrets Manager y Amazon Route 53, al no ser necesarios para validar los objetivos de la solución base. En el caso de Route 53, el acceso a la aplicación pudo realizarse directamente mediante el punto de entrada proporcionado por el balanceador de carga (ALB). En el caso de AWS Secrets Manager, la gestión de configuración sensible se mantuvo mediante mecanismos más simples.
- El almacenamiento S3 aparece representado fuera de la VPC, ya que se trata de un servicio regional gestionado por AWS y no de un recurso desplegado dentro de las subredes de la red privada virtual, como se pensó al principio. La

comunicación con dicho servicio se realiza desde los componentes de aplicación autorizados, manteniendo la separación entre la infraestructura de cómputo y el almacenamiento de objetos.

- Aparece el servicio ECR, indicando que en los repositorios que se almacenan en el mismo, están disponibles las imágenes utilizadas por el clúster para los correspondientes *pods*.

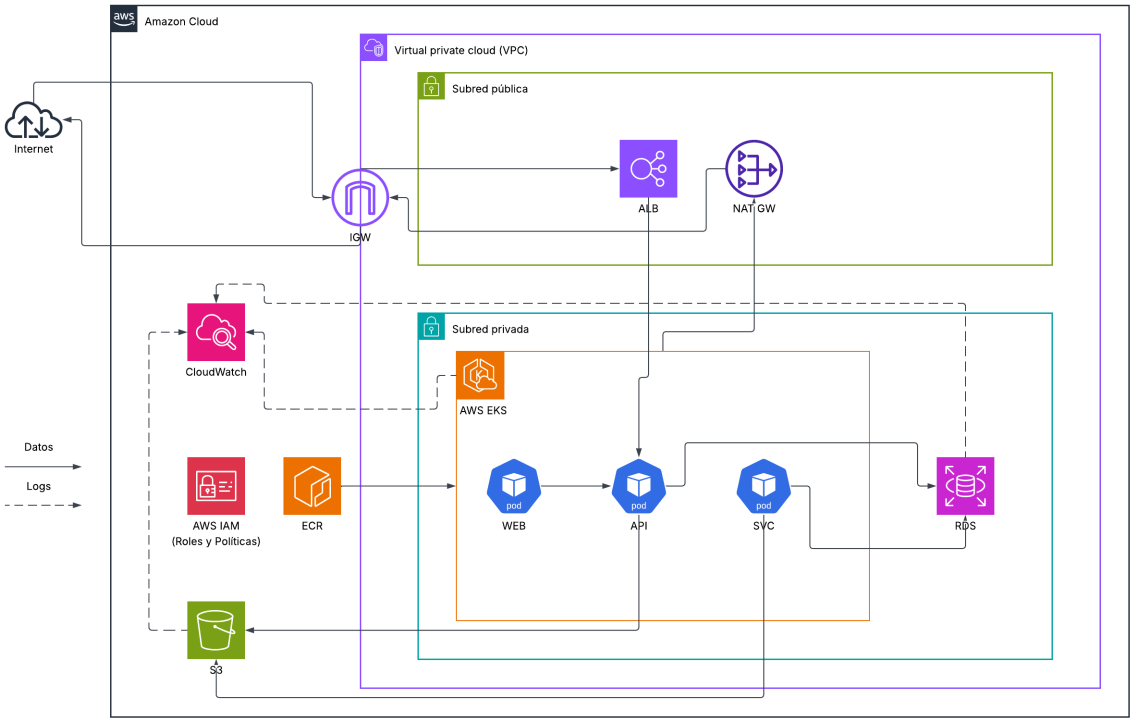



**Figura 3.7:** Diagrama definitivo de arquitectura *cloud* de la aplicación base. Fuente: elaboración propia.

### 3.3.3. Funcionalidad de la aplicación

La aplicación desarrollada proporciona una interfaz web desde la que los usuarios pueden acceder a las funcionalidades principales del sistema. Tras registrarse o iniciar sesión, el usuario puede crear eventos, consultar el listado de los ya existentes, acceder a su detalle, asociar imágenes y comprobar el estado del procesamiento interno realizado por el servicio *batch*. De esta forma, la interfaz permite recorrer los casos de uso principales definidos durante la fase de análisis y verificar de manera visual el funcionamiento conjunto de los distintos componentes de la solución.

El flujo habitual de uso se ha mantenido con respecto al análisis realizado. Por lo tanto, el flujo coincide con el representado en el diagrama de secuencia en la Figura 3.4.

La Figura 3.8 muestra las pantallas de acceso inicial a la aplicación, desde las que un usuario puede registrarse o autenticarse.

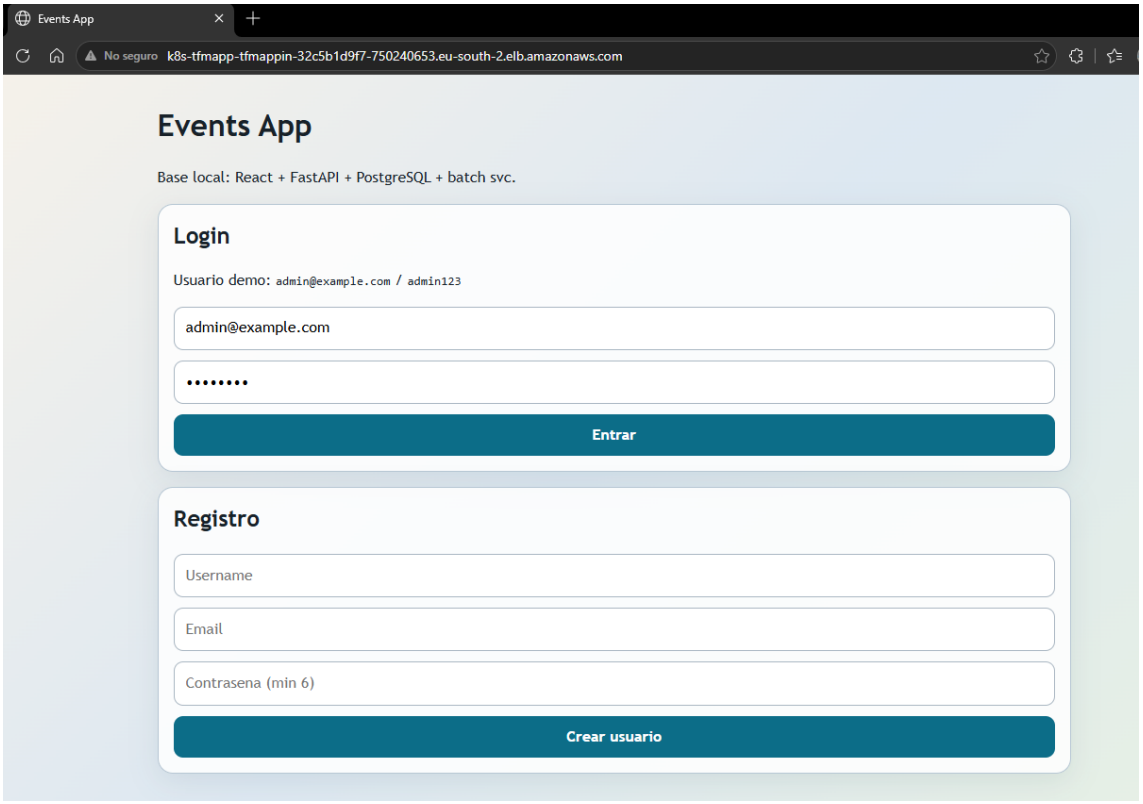



**Figura 3.8:** *Login* y registro de la aplicación base. Fuente: captura propia.

Una vez autenticado, el usuario accede a la gestión de eventos. Desde esta vista puede crear nuevos registros y consultar el listado de eventos existentes, tal y como se muestra en la Figura 3.9.

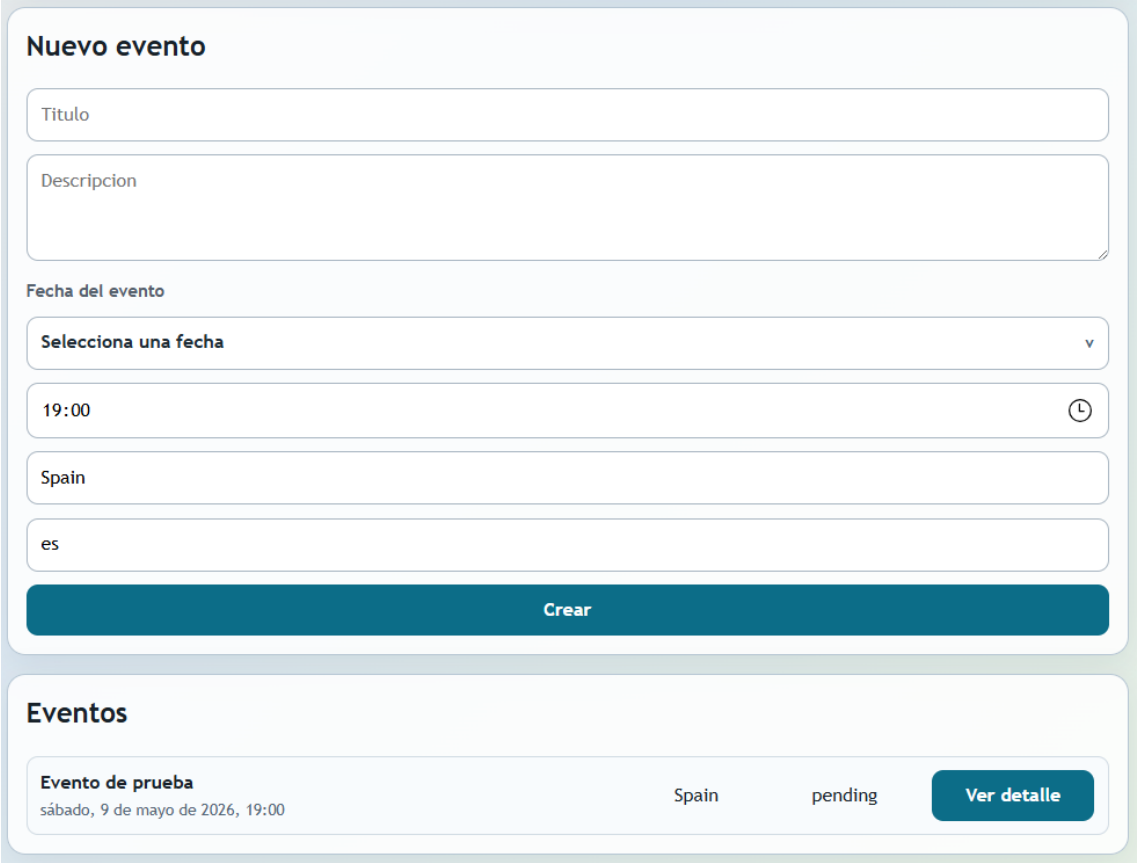



**Figura 3.9:** Formulario de creación de evento y listado de eventos. Fuente: captura propia.

La Figura 3.10 muestra el detalle de un evento tras su creación y la publicación de una imagen asociada, antes de que haya sido tratado por el servicio *batch*.




**Figura 3.10:** Detalle de evento con imagen publicada. Previo al procesamiento del servicio *batch*. Fuente: captura propia.

La aplicación también permite consultar el estado de las ejecuciones del servicio interno, así como finalizar la sesión activa del usuario, funcionalidades representadas en la Figura 3.11.




**Figura 3.11:** *Logout* y consulta de los procesos del servicio *batch*. Fuente: captura propia.

Tras la ejecución del procesamiento interno, la información generada por el servicio *batch* queda reflejada en el detalle del evento, como puede observarse en la Figura 3.12.




**Figura 3.12:** Detalle de evento con imagen publicada. Posterior al procesamiento del servicio *batch*. Fuente: captura propia.

### 3.3.4. Despliegue

La aplicación puede ejecutarse en dos entornos principales: un entorno local de desarrollo y un entorno *cloud* desplegado en AWS. El entorno local se utiliza para el desarrollo y validación rápida de funcionalidades, mientras que el entorno *cloud* representa el despliegue objetivo de la solución base.

- En local, la ejecución se apoya en Docker Compose, levantando los servicios de aplicación junto con una base de datos PostgreSQL y un servicio MinIO compatible con S3. Este entorno permite probar la aplicación sin depender de los servicios gestionados de AWS y facilita la depuración durante el desarrollo.
- En AWS, la ejecución se realiza mediante contenedores desplegados en Amazon EKS. Las imágenes Docker de los servicios se construyen y publican en Amazon ECR, desde donde son utilizadas por los manifiestos de Kubernetes para crear las cargas de trabajo correspondientes. La base de datos se ejecuta sobre Amazon RDS y el almacenamiento de imágenes se realiza mediante Amazon S3.

El despliegue de la infraestructura se automatiza mediante Terraform, que define los recursos principales de red, cómputo, almacenamiento, base de datos, permisos y registro de imágenes. Además, se dispone de ***scripts* auxiliares**, disponibles en el repositorio del proyecto, para automatizar tareas repetitivas como la construcción de imágenes, el etiquetado, la autenticación frente a ECR y la publicación de nuevas versiones.

Una vez ejecutado el despliegue automatizado, el estado completo de la solución puede comprobarse mediante el propio *script* de gestión desarrollado para el proyecto. La Figura 3.13 muestra la salida del comando de consulta de estado, en la que se verifica que la infraestructura real se encuentra sincronizada con la definición de Terraform, que el clúster de EKS está activo, que los *pods* de la aplicación se ejecutan correctamente y que los recursos asociados, como RDS y el NAT Gateway, se encuentran disponibles.




**Figura 3.13:** Salida de terminal del estado de la infraestructura base desplegada. Fuente: captura propia.

Además de la comprobación realizada desde terminal, el despliegue puede verificarse directamente desde la consola de AWS. Las siguientes Figuras recogen los principales recursos asociados a Amazon EKS: el clúster activo 3.14, el nodo de

cómputo utilizado para ejecutar las cargas de trabajo 3.15 y los *pods* correspondientes a los servicios de la aplicación desplegados correctamente 3.16.

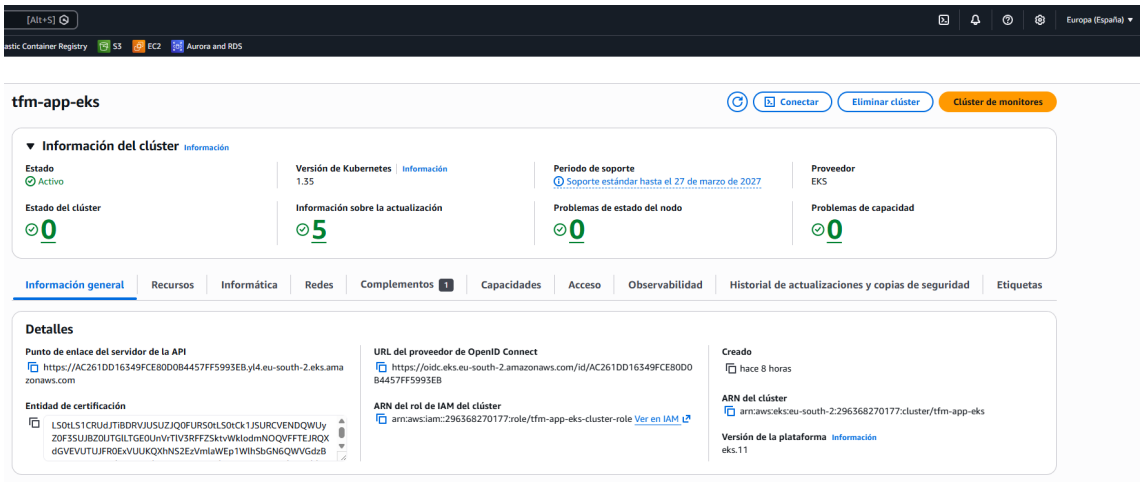



**Figura 3.14:** Clúster EKS base en la consola de AWS. Fuente: captura propia.




**Figura 3.15:** Nodo del clúster EKS base en la consola de AWS. Fuente: captura propia.

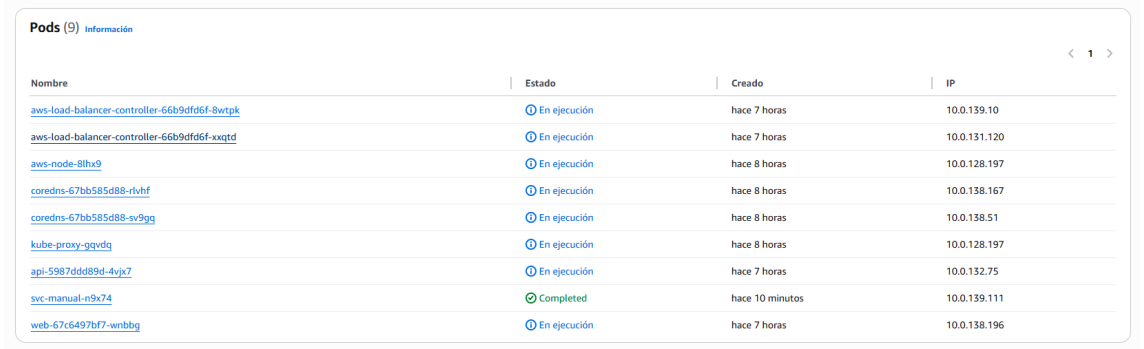



**Figura 3.16:** *Pods* del nodo dentro del clúster EKS base en la consola de AWS. Fuente: captura propia.

Por último, las siguientes figuras muestran los servicios gestionados que completan el despliegue de la aplicación base. La VPC constituye el entorno de red aislado sobre

el que se despliegan los recursos principales de la solución, tal y como se muestra en la Figura 3.17. Sobre esta infraestructura, el balanceador de carga proporciona el punto de acceso externo a la aplicación, como recoge la Figura 3.18; Amazon RDS aloja la base de datos PostgreSQL, mostrada en la Figura 3.19; Amazon S3 almacena las imágenes asociadas a los eventos, como se observa en la Figura 3.20; y Amazon ECR conserva las imágenes Docker utilizadas por los distintos servicios, según se muestra en la Figura 3.21.

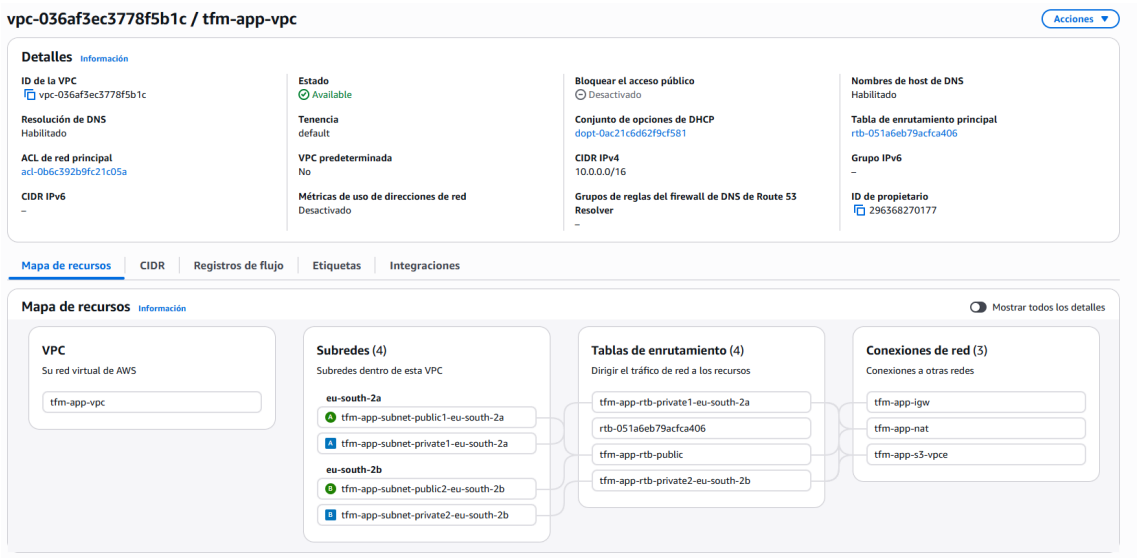



**Figura 3.17:** VPC del despliegue de la aplicación base en la consola de AWS. Fuente: captura propia.

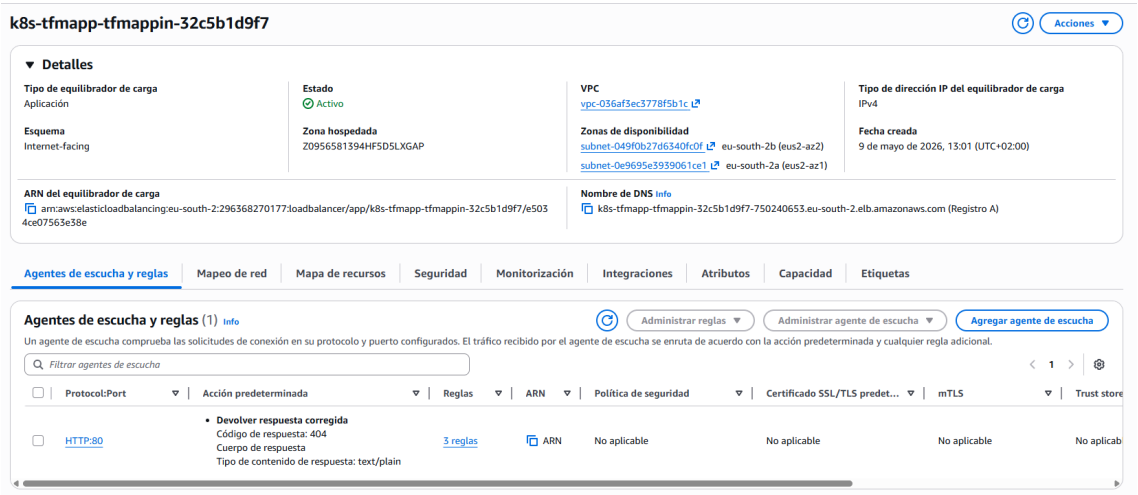



**Figura 3.18:** ALB del despliegue de la aplicación base en la consola de AWS. Fuente: captura propia.

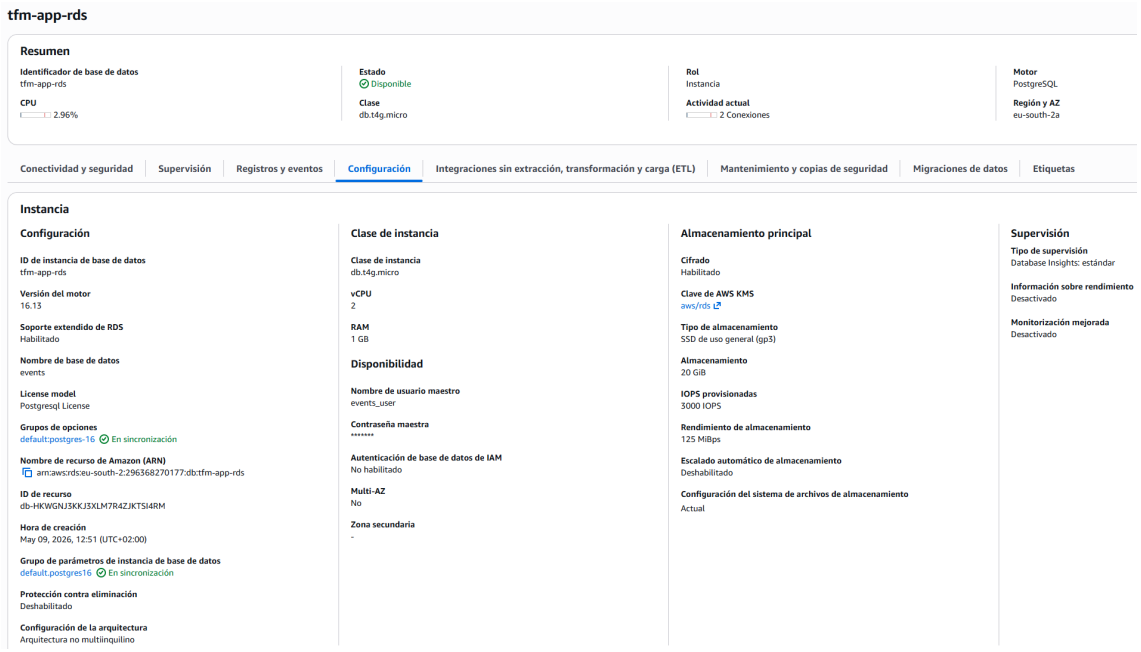



**Figura 3.19:** RDS del despliegue de la aplicación base en la consola de AWS. Fuente: captura propia.

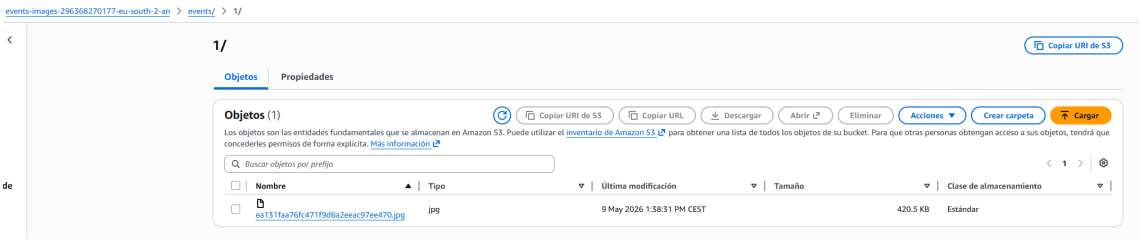



**Figura 3.20:** *Bucket* S3 del despliegue de la aplicación base en la consola de AWS. Fuente: captura propia.

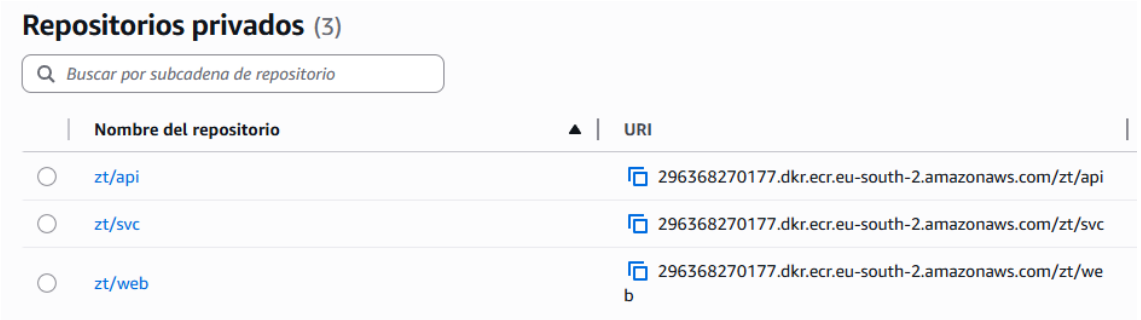



**Figura 3.21:** ECR del despliegue de la aplicación base en la consola de AWS. Fuente: captura propia.

# Capítulo 4

# Implementación de *Zero Trust*

## 4.1. Descripción de la migración

La migración de la arquitectura base hacia un enfoque *Zero Trust* se plantea como una evolución incremental de la solución *cloud* desarrollada previamente, y no como una sustitución completa de la infraestructura existente. El objetivo principal de esta fase es reducir la confianza implícita entre los distintos componentes del sistema mediante la incorporación de controles de identidad, mínimo privilegio, segmentación de red, protección de credenciales y observabilidad.

La solución base ya proporciona una arquitectura *cloud* funcional, desplegada sobre AWS y apoyada en sus servicios. Sin embargo, una arquitectura funcional no implica necesariamente una arquitectura alineada con los principios de *Zero Trust*. Por ello, esta fase parte de la infraestructura existente y analiza qué elementos pueden reforzarse para limitar la exposición de credenciales, reducir permisos excesivos, controlar mejor las comunicaciones internas y mejorar la capacidad de verificación del sistema.

El enfoque seguido se basa en identificar brechas concretas de seguridad en la arquitectura inicial y asociar cada una de ellas con un control técnico específico. De esta forma, la migración no se aborda como una aplicación genérica del paradigma *Zero Trust*, sino como un proceso trazable en el que cada modificación responde a una debilidad previamente detectada.

En términos prácticos, la migración se organiza en tres líneas principales de actuación:

1. Se refuerza la identidad de los *workloads* y el modelo de permisos, sustituyendo el uso de credenciales estáticas por un mecanismo basado en roles asociados a cuentas de servicio de Kubernetes.
2. Se introduce segmentación de red dentro del clúster para restringir las comunicaciones entre componentes a aquellas estrictamente necesarias.
3. Se endurece la configuración de los *pods* y se mejora la generación de evidencias mediante pruebas y comprobaciones orientadas a validar los controles implantados.

La solución resultante no pretende representar una implementación completa de todos los mecanismos posibles de *Zero Trust*, sino una migración realista y medible de una arquitectura *cloud* base hacia un modelo más restrictivo, verificable y alineado con los principios de *Zero Trust*.

## 4.2. El proceso de desarrollo

### 4.2.1. Análisis

Antes de realizar la migración, se ha llevado a cabo un análisis de la infraestructura base con el objetivo de identificar los principales puntos de mejora respecto a los principios de *Zero Trust*.

La Tabla 4.1 resume las brechas identificadas, el riesgo asociado a cada una de ellas, el principio de *Zero Trust* relacionado y la mejora para la migración. Estas líneas actúan como orientación inicial de la migración.

**Tabla 4.1:** Análisis de brechas arquitectónicas para la migración hacia *Zero Trust*

| ID | Brecha identificada | Riesgo asociado | Principio *Zero Trust* | Mejora |
|---|---|---|---|---|
| ZT1 | Uso de credenciales estáticas de Amazon S3 dentro de Kubernetes. | Exposición de claves persistentes que podrían reutilizarse si se ven comprometidas. | Verificación explícita y protección de credenciales. | Eliminar las claves estáticas y utilizar IRSA para asociar credenciales temporales a los *workloads* que acceden a S3. |
| ZT2 | Permisos de acceso a S3 asociados al rol de los nodos de EKS. | Posible exceso de privilegios, ya que los permisos quedan vinculados al nodo y no al servicio concreto que accede al recurso. | Mínimo privilegio y acceso por recurso. | Sustituir el modelo basado en permisos del nodo por roles IAM específicos para cada *workload*, limitando las acciones y recursos permitidos. |
| ZT3 | Ausencia de políticas de red internas en Kubernetes. | Comunicación este-oeste demasiado amplia entre *pods*, facilitando el movimiento lateral ante el compromiso de un componente. | Segmentación y reducción de confianza implícita. | Aplicar *NetworkPolicies* con enfoque de denegación por defecto y reglas explícitas para permitir únicamente las comunicaciones necesarias entre componentes. |

*Continúa en la siguiente página*

**Tabla 4.1:** (continuación)

| ID | Brecha identificada | Riesgo asociado | Principio *Zero Trust* | Mejora |
|---|---|---|---|---|
| ZT4 | Configuración de ejecución de *pods* con endurecimiento limitado. | Mayor superficie de ataque dentro del contenedor y posibilidad de abuso de privilegios en caso de compromiso. | Mínimo privilegio y reducción de superficie de ataque. | Incorporar *securityContext*, ejecución como usuario no privilegiado, desactivación de escalada de privilegios, límites de recursos y sondas de estado. |
| ZT5 | *Endpoint* público del plano de control de Amazon EKS. | Mayor exposición del plano de administración del clúster frente a accesos no deseados desde redes externas. | Verificación explícita y reducción de superficie expuesta. | Restringir el acceso al *endpoint* público mediante rangos CIDR concretos, manteniendo la operatividad del entorno de administración. |

*Continúa en la siguiente página*

**Tabla 4.1:** (continuación)

| ID | Brecha identificada | Riesgo asociado | Principio *Zero Trust* | Mejora |
|---|---|---|---|---|
| ZT6 | Observabilidad no orientada específicamente a los controles de seguridad implantados. | Menor capacidad para revisar el estado de los controles *Zero Trust*, detectar comportamientos anómalos o justificar la postura de seguridad de la solución migrada. | Monitorización continua, auditoría y evaluación dinámica de la confianza. | Reutilizar la integración existente con CloudWatch y complementarla con eventos de Kubernetes, *logs* de aplicación y evidencias operativas de IRSA, *NetworkPolicies*, endurecimiento de *pods* y restricción del *endpoint* de EKS. |

### 4.2.2. Diseño

A partir del análisis de brechas realizado, se definió una arquitectura objetivo orientada a reducir los principales puntos de confianza implícita de la solución base. El diseño plantea un rediseño parcial de los mecanismos de identidad, autorización, segmentación, exposición administrativa y observabilidad. Esta decisión permite mantener la funcionalidad de la aplicación y, al mismo tiempo, introducir controles alineados con los principios de *Zero Trust*.

El diseño propuesto se basa en trasladar la confianza desde elementos amplios de infraestructura, como los nodos del clúster o la red interna de Kubernetes, hacia identidades, permisos y comunicaciones explícitamente definidos. En la arquitectura base, determinados permisos se encontraban asociados al entorno de ejecución y parte de las comunicaciones internas se permitían de forma implícita. En la arquitectura migrada, cada *workload* relevante dispone de una identidad propia, los accesos a recursos *cloud* se conceden mediante políticas específicas, y las comunicaciones internas se restringen mediante reglas declarativas.

Los servicios gestionados de AWS y los mecanismos nativos de Kubernetes se utilizan para materializar parcialmente las funciones lógicas descritas por NIST para una arquitectura *Zero Trust*. En este trabajo no se despliega un motor de políticas independiente, sino que la toma de decisiones y la aplicación de políticas se distribuyen entre servicios existentes. AWS IAM, IRSA y las políticas IAM intervienen en la autorización de acceso a recursos *cloud*; las cuentas de servicio de Kubernetes permiten asociar identidad a los *workloads*; las *NetworkPolicies* y los grupos de seguridad actúan como puntos de aplicación de políticas de red; y la configuración del *endpoint* de EKS limita el acceso al plano de administración. De esta forma, la arquitectura incorpora funciones equivalentes de decisión y *enforcement* sin introducir una plataforma adicional de orquestación *Zero Trust*.

En la Figura 4.1 se puede visualizar el diseño objetivo de la arquitectura *Zero Trust* tras el análisis descrito en el apartado anterior.

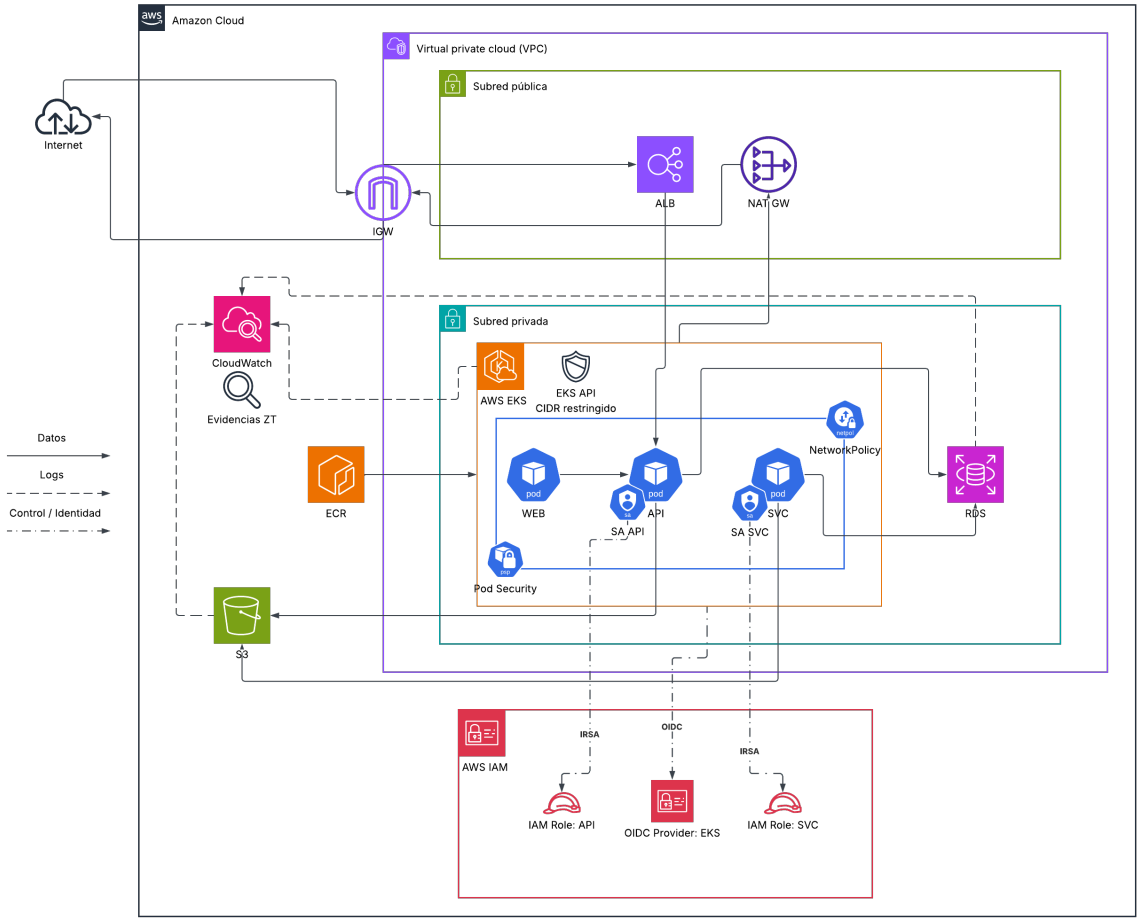



**Figura 4.1:** Diagrama de arquitectura *cloud* de la aplicación aplicando *Zero Trust*. Fuente: elaboración propia.

La explicación de los nuevos cambios introducidos es la siguiente:

- Identidad y permisos por *workload*: se incorporan cuentas de servicio específicas para los componentes que necesitan acceder a recursos de AWS. Estas cuentas se asocian a roles IAM mediante IRSA, evitando el uso de credenciales estáticas y permitiendo aplicar permisos mínimos a cada servicio.
- Segmentación interna: se introduce una *NetworkPolicy* dentro del clúster de Kubernetes para controlar las comunicaciones entre los distintos *pods*. De esta forma, se reduce la confianza implícita dentro de EKS y solo se permiten los flujos necesarios para el funcionamiento de la aplicación.
- Endurecimiento de *workloads*: se refuerza la configuración de ejecución de los *pods* mediante medidas de seguridad a nivel de contenedor, como el uso de *securityContext*, la ejecución con usuarios no privilegiados, la desactivación de la escalada de privilegios y la definición de límites de recursos.
- Restricción del plano de control: se limita el acceso administrativo al *endpoint* de la API de EKS mediante rangos CIDR autorizados. Con ello se reduce la exposición del plano de administración del clúster sin modificar el flujo funcional de acceso a la aplicación.
- Trazabilidad orientada a *Zero Trust*: se mantiene la integración existente con CloudWatch, pero se utiliza como soporte para revisar evidencias operativas de los nuevos controles implantados. Esta trazabilidad permite comprobar el estado de IRSA, las *NetworkPolicies*, el endurecimiento de *pods* y la configuración restringida del *endpoint* de EKS.

### 4.2.3. Implementación

La implementación de la arquitectura *Zero Trust* se llevó a cabo aplicando los cambios definidos en la fase de diseño sobre la infraestructura *cloud* base descrita en el capítulo anterior. Las modificaciones se organizaron en cinco bloques principales, alineados con los controles definidos en el diseño.

#### Identidad y permisos por *workload*

El primer cambio aplicado durante la migración consistió en sustituir el modelo de acceso a Amazon S3 basado en credenciales estáticas por un modelo de identidad federada mediante *IAM Roles for Service Accounts* (IRSA). Esta modificación responde a las brechas identificadas en la arquitectura base relativas al uso de claves

persistentes dentro de Kubernetes y a la asignación de permisos de S3 al rol IAM de los nodos de Amazon EKS.

En la solución base, los componentes que necesitaban acceder al *bucket* de imágenes recibían las credenciales de S3 mediante variables de entorno obtenidas desde secretos de Kubernetes. En particular, los despliegues de la API y del servicio *batch* utilizaban valores como `S3_ACCESS_KEY` y `S3_SECRET_KEY`. Aunque este enfoque permitía el acceso funcional al almacenamiento de objetos, introducía un riesgo relevante, ya que las claves eran credenciales persistentes almacenadas dentro del entorno Kubernetes. En caso de exposición de un secreto, de un manifiesto o de un *pod* comprometido, dichas credenciales podrían reutilizarse fuera del contexto previsto.

Además, la solución base mantenía permisos de acceso a S3 asociados al rol IAM de los nodos de EKS. Este planteamiento generaba una confianza implícita excesiva en el entorno de ejecución, ya que el permiso no quedaba vinculado al servicio concreto que necesitaba acceder al recurso, sino a la infraestructura subyacente donde se ejecutaban los *pods*. Desde una perspectiva *Zero Trust*, este modelo no ofrece una separación suficientemente precisa entre los distintos *workloads*, puesto que no diferencia de forma explícita qué componente debe tener acceso a cada recurso *cloud*.

La Tabla 4.2 resume la diferencia entre el modelo de acceso utilizado en la solución base y el modelo adoptado tras la migración mediante IRSA.

**Tabla 4.2:** Migración del modelo de acceso a S3 hacia IRSA

| **Elemento** | **Solución base** | **Solución migrada** |
|---|---|---|
| Credenciales AWS | Claves estáticas almacenadas en secretos de Kubernetes | Credenciales temporales obtenidas mediante IRSA |
| Identidad de acceso a S3 | Usuario IAM con claves estáticas y permisos asociados al rol de nodos | Roles IAM específicos para cada *workload* |
| API | Acceso mediante variables `S3_ACCESS_KEY` y `S3_SECRET_KEY` | Acceso mediante el *ServiceAccount* `api` y el rol `tfm-app-api-irsa-role` |

*Continúa en la siguiente página*

**Tabla 4.2:** (continuación)

| **Elemento** | **Solución base** | **Solución migrada** |
|---|---|---|
| Servicio *batch* | Acceso mediante las mismas credenciales estáticas | Acceso mediante el *ServiceAccount* `svc` y el rol `tfm-app-svc-irsa-role` |
| Rol de nodos de EKS | Incluía permisos de acceso a S3 | Se elimina la política S3 del rol de nodos |
| Principio aplicado | Confianza implícita en secretos y nodos | Identidad explícita, credenciales temporales y mínimo privilegio |

Para corregir esta situación, se introdujo un modelo basado en identidades específicas por *workload*. En Terraform se definieron dos nuevos roles IAM, asociados a los componentes que realmente necesitaban acceder a Amazon S3:

- Un rol IAM para la API.
- Un rol IAM para el servicio *batch*.

Estos roles se asociaron respectivamente a los *ServiceAccounts* `api` y `svc` del *namespace* `tfm-app`. La relación de confianza de cada rol se configuró utilizando el proveedor OIDC del clúster EKS y la acción `sts:AssumeRoleWithWebIdentity`, limitando la asunción del rol al *ServiceAccount* correspondiente mediante la condición `sub`.

De esta forma, el rol de la API solo puede ser asumido por el *ServiceAccount* asociado a dicho componente, mientras que el rol del servicio *batch* queda restringido a su propia cuenta de servicio. Las identidades utilizadas en la condición `sub` fueron las siguientes:

- API: `system:serviceaccount:tfm-app:api`
- Servicio *batch*: `system:serviceaccount:tfm-app:svc`

Junto a los roles IAM, se definieron políticas específicas de acceso a Amazon S3. La política asociada a la API permite las operaciones necesarias para gestionar las imágenes de los eventos, incluyendo lectura, escritura y eliminación de objetos. Por su parte, la política asociada al servicio *batch* se limitó a las operaciones necesarias para su procesamiento, evitando conceder permisos adicionales que no fueran reque-

ridos. Con ello, el acceso al *bucket* deja de depender de una identidad compartida o del rol de los nodos, y pasa a estar ligado a identidades concretas de aplicación.

En Kubernetes se incorporó el archivo `serviceaccounts.yaml`, donde se declararon los *ServiceAccounts* utilizados por los componentes con acceso a S3. Cada uno de ellos incluye la anotación `eks.amazonaws.com/role-arn`, que permite vincular el *ServiceAccount* de Kubernetes con el rol IAM correspondiente. Posteriormente, los manifiestos de los componentes se ajustaron de la siguiente forma:

- El despliegue de la API se actualizó para utilizar `serviceAccountName: api`.
- El *CronJob* del servicio *batch* se modificó para utilizar `serviceAccountName: svc`.

Como parte de la migración, se eliminaron de los manifiestos las variables de entorno que inyectaban las credenciales estáticas de S3. También se limpió el archivo de secretos de Kubernetes para retirar las claves persistentes asociadas al usuario IAM anterior.

Sin embargo, se mantuvieron los valores de configuración no sensibles, ya que estos parámetros no constituyen credenciales y siguen siendo necesarios para el funcionamiento de la aplicación:

- Nombre del *bucket*.
- Región.
- Uso de SSL.
- Tiempo de validez de las URL generadas.

A nivel de infraestructura, también se eliminó el usuario IAM utilizado previamente para el acceso a S3, junto con su política asociada. Del mismo modo, se retiró la política de acceso a S3 que estaba vinculada al rol de los nodos de EKS. Esta modificación reduce los privilegios del rol de nodos y evita que el acceso al *bucket* esté disponible desde la infraestructura de ejecución de forma general.

En consecuencia, los nodos conservan únicamente los permisos necesarios para operar dentro del clúster, mientras que el acceso a S3 queda delegado exclusivamente en los *workloads* autorizados mediante IRSA.

Tras aplicar los cambios, se comprobó que los componentes principales continuaban desplegándose correctamente y que la API recibía las variables propias del mecanismo IRSA, como `AWS_ROLE_ARN` y `AWS_WEB_IDENTITY_TOKEN_FILE`, sin recibir ya

las variables `S3_ACCESS_KEY` ni `S3_SECRET_KEY`. También se ejecutó manualmente el *CronJob* del servicio *batch*, verificando que el *pod* generado utilizaba el *ServiceAccount* `svc` y que este se encontraba anotado con el rol IAM correspondiente.

Este cambio aporta una mejora significativa en términos de *Zero Trust*. En concreto, permite reforzar varios aspectos de la arquitectura:

- Elimina credenciales persistentes de AWS dentro de Kubernetes, sustituyéndolas por credenciales temporales emitidas por AWS STS.
- Aplica el principio de mínimo privilegio al separar los permisos de la API y del servicio *batch*.
- Reduce la confianza implícita en los nodos de EKS, ya que estos dejan de disponer de permisos directos sobre S3.
- Limita el impacto potencial de un compromiso, puesto que cada *pod* solo puede asumir el rol asociado a su propio *ServiceAccount* y únicamente acceder a los recursos autorizados para dicho *workload*.

Con esta modificación, la arquitectura migrada refleja un modelo en el que la identidad y la autorización se definen de forma explícita por componente, en lugar de depender de credenciales compartidas o permisos asociados al entorno de ejecución.

#### Segmentación interna mediante *NetworkPolicies*

El segundo cambio aplicado durante la migración consistió en introducir segmentación interna dentro del clúster de Kubernetes mediante *NetworkPolicies*. El objetivo de esta medida fue reducir la confianza implícita existente entre los *pods* del *namespace* de la aplicación, limitando el tráfico entrante a los componentes únicamente a aquellos flujos necesarios para el funcionamiento del sistema.

En la solución base, los *pods* desplegados en el clúster podían comunicarse internamente sin restricciones específicas a nivel de Kubernetes. Este comportamiento es habitual en despliegues por defecto, pero desde una perspectiva *Zero Trust* implica una superficie de movimiento lateral demasiado amplia. Si un componente se viera comprometido, podría intentar comunicarse con otros servicios internos del mismo *namespace*, aunque dicha comunicación no formara parte del flujo funcional esperado.

La segmentación aplicada se centró en el *namespace* `tfm-app`, donde se ejecutan los principales componentes de la aplicación:

- `web`: despliegue del *frontend*, identificado mediante la etiqueta `app: web` y expuesto internamente en el puerto `8080`.
- `api`: despliegue del *backend*, identificado mediante la etiqueta `app: api` y expuesto internamente en el puerto `8000`.
- `svc`: servicio *batch* desplegado como *CronJob*, sin exposición de tráfico entrante.

Antes de aplicar las políticas, fue necesario habilitar el soporte efectivo de *NetworkPolicy* en Amazon EKS. Para ello, se modificó la configuración del *addon* Amazon VPC CNI mediante Terraform, incorporando el parámetro `enableNetworkPolicy = "true"`. Esta configuración permite que las políticas declaradas en Kubernetes sean aplicadas realmente por el componente de red del clúster, evitando que los manifiestos de *NetworkPolicy* queden únicamente definidos a nivel declarativo sin efecto práctico sobre el tráfico.

La estrategia adoptada se basó en un enfoque de denegación por defecto para el tráfico entrante. A partir de esa restricción inicial, se añadieron reglas explícitas para permitir únicamente los flujos necesarios:

- tráfico desde el ALB hacia el *frontend* `web`;
- tráfico desde el ALB hacia la API, necesario para las rutas `/api` y para el *health check*;
- tráfico interno legítimo desde `web` hacia `api`.

La Tabla 4.3 resume las políticas implantadas y su función dentro del modelo de segmentación.

**Tabla 4.3:** Políticas de red aplicadas para la segmentación interna

| **Política** | ***Pods* afectados** | **Objetivo** |
|---|---|---|
| `default-deny-ingress` | Todos los *pods* del *namespace* `tfm-app` | Deniega por defecto todo el tráfico entrante hacia los *pods* del *namespace*. Sirve como base para aplicar un modelo de autorización explícita. |
| *Continúa en la siguiente página* | | |

**Tabla 4.3:** (continuación)

| **Política** | ***Pods* afectados** | **Objetivo** |
|---|---|---|
| `allow-alb-to-web` | *Pods* con etiqueta `app: web` | Permite tráfico TCP hacia el puerto `8080` desde las subredes públicas donde se ubica el ALB. Mantiene accesible el *frontend* desde el balanceador. |
| `allow-alb-and-web-to-api` | *Pods* con etiqueta `app: api` | Permite tráfico TCP hacia el puerto `8000` desde las subredes públicas del ALB y desde *pods* con etiqueta `app: web`. Mantiene las rutas `/api`, el *health check* y la comunicación interna `web` → `api`. |

La primera política, `default-deny-ingress`, selecciona todos los *pods* del *namespace* mediante `podSelector: {}` y define únicamente el tipo `Ingress`. Con ello, cualquier tráfico entrante hacia los *pods* queda bloqueado salvo que otra política lo permita explícitamente. Esta decisión cambia el modelo por defecto del clúster, pasando de una comunicación interna abierta a un esquema de acceso basado en reglas concretas.

La segunda política, `allow-alb-to-web`, permite que el balanceador de carga de AWS acceda al *frontend*. En este despliegue, el *Ingress* utiliza `ingressClassName: alb` y la anotación `alb.ingress.kubernetes.io/target-type: ip`, por lo que el ALB enruta el tráfico directamente hacia las IPs de los *pods*. Por ese motivo, la regla permite tráfico desde los rangos CIDR de las subredes públicas donde se encuentra el ALB hacia el puerto `8080` de los *pods* `web`.

La tercera política, `allow-alb-and-web-to-api`, permite el acceso legítimo a la API. Por un lado, se autoriza el tráfico desde las subredes públicas del ALB hacia el puerto `8000`, ya que el *Ingress* enruta las rutas bajo `/api` directamente al servicio `api` y utiliza `/api/health` como ruta de comprobación de estado. Por otro lado, se

permite la comunicación interna desde *pods* con la etiqueta `app: web`, habilitando el flujo funcional `web` → `api` sin abrir el acceso a cualquier otro *pod* del *namespace*.

El servicio `svc`, al ejecutarse como *CronJob*, no expone un servicio de entrada ni requiere recibir tráfico desde otros componentes. Por tanto, no se añadió una regla específica de entrada para este componente. Bajo el modelo de denegación por defecto, queda protegido frente a accesos laterales entrantes, manteniendo únicamente las comunicaciones salientes necesarias para su ejecución.

En esta fase no se aplicó una política restrictiva de *egress*. Esta decisión fue intencionada, ya que tanto la API como el servicio *batch* requieren conectividad saliente hacia servicios externos y gestionados de AWS, entre ellos:

- Amazon RDS, para el acceso a la base de datos PostgreSQL.
- Amazon S3, para el almacenamiento de imágenes.
- AWS STS, necesario para la obtención de credenciales temporales mediante IRSA.
- DNS/CoreDNS, necesario para la resolución de nombres dentro y fuera del clúster.

Restringir el tráfico saliente en esta fase podría introducir indisponibilidad o errores difíciles de diagnosticar, especialmente porque las *NetworkPolicies* estándar de Kubernetes no permiten definir reglas basadas en nombres de dominio completos. Esto complica la limitación directa de servicios como S3 o STS, cuyos *endpoints* pueden resolverse a direcciones IP variables. Por este motivo, el control fino de *egress* se considera una evolución posterior, que podría apoyarse en mecanismos como *endpoints* privados de AWS, grupos de seguridad, políticas IAM e IRSA, AWS Network Firewall, capacidades avanzadas de CNI o proxies de salida.

La integración de esta medida en el proceso de despliegue se realizó incorporando el manifiesto `networkpolicy.yaml`. De esta forma, las políticas de red pasan a formar parte del despliegue reproducible de la solución migrada. En el flujo de despliegue, el manifiesto se aplica junto con el resto de recursos de Kubernetes, mientras que en el proceso de parada se contempla su eliminación junto al resto de objetos del *namespace*. Asimismo, el estado de las políticas puede consultarse junto con los *pods*, servicios e *Ingress* desplegados.

Con esta modificación, la arquitectura reduce la comunicación lateral implícita entre componentes. El acceso a los *pods* deja de estar abierto por defecto y pasa a

depender de reglas explícitas que reflejan los flujos funcionales reales de la aplicación. En concreto, el *frontend* sigue siendo accesible a través del ALB, la API mantiene el acceso necesario para las rutas públicas y para la comunicación desde el *frontend*, y el servicio *batch* permanece sin exposición entrante.

En conjunto, este bloque introduce un avance relevante hacia el modelo *Zero Trust*, al sustituir la comunicación interna implícita por un esquema de autorización explícita entre componentes. Aunque no constituye una segmentación completa de todos los flujos de red, establece una primera barrera efectiva frente al movimiento lateral dentro del clúster y deja preparado el entorno para futuras restricciones más finas del tráfico saliente.

### Endurecimiento de *workloads*

Esta fase aborda la reducción de privilegios en tiempo de ejecución. El objetivo es limitar las capacidades disponibles dentro de los contenedores, reducir la superficie de ataque y contener el impacto potencial de un compromiso de alguno de los *pods*.

En la solución base, los manifiestos de Kubernetes no definían controles explícitos de endurecimiento a nivel de *pod* o contenedor. La ejecución de los contenedores dependía principalmente de la configuración incluida en las propias imágenes, sin declarar en Kubernetes restricciones como `securityContext`, eliminación de capacidades Linux, perfil `seccomp`, límites de recursos o sondas de salud. Aunque este enfoque permitía el funcionamiento correcto de la aplicación, dejaba parte del comportamiento de seguridad del *runtime* sin definir de forma declarativa en los manifiestos.

Desde una perspectiva *Zero Trust*, no basta con limitar qué identidad tiene cada servicio o qué comunicaciones están permitidas entre componentes. También es necesario reducir lo que un contenedor puede hacer una vez se encuentra en ejecución. Por ello, esta fase incorpora controles orientados a aplicar mínimo privilegio dentro del entorno de ejecución de Kubernetes.

Los controles aplicados se agrupan en cuatro líneas principales:

- Restricción de privilegios del contenedor, mediante la desactivación de la escalada de privilegios.
- Reducción de capacidades Linux disponibles, eliminando las capacidades no necesarias.

- Uso del perfil `seccomp` por defecto del *runtime*, para limitar llamadas al sistema potencialmente peligrosas.
- Definición de límites y peticiones de recursos, para reducir el impacto de consumos anómalos de CPU o memoria.

Además, en los componentes de tipo servicio se incorporaron sondas de salud para mejorar el control operativo del despliegue. En el caso del servicio *batch*, al tratarse de un *CronJob*, se aplicaron controles específicos sobre la ejecución de trabajos.

La Tabla 4.4 resume los principales controles aplicados a cada *workload*.

**Tabla 4.4:** Controles de endurecimiento aplicados a los *workloads*

| ***Workload*** | **Controles aplicados** | **Objetivo** |
|---|---|---|
| `web` | Ejecución no root verificable, perfil `seccomp`, bloqueo de escalada de privilegios, eliminación de capacidades Linux, límites de recursos y sondas HTTP. | Reducir privilegios del *frontend*, controlar su consumo de recursos y comprobar su disponibilidad antes de recibir tráfico. |
| `api` | Perfil `seccomp`, bloqueo de escalada de privilegios, eliminación de capacidades Linux, límites de recursos y sondas HTTP sobre `/api/health`. | Limitar el impacto de una posible explotación del *backend* y mejorar su control operativo sin modificar la imagen ni el código. |
| `svc` | Perfil `seccomp`, bloqueo de escalada de privilegios, eliminación de capacidades Linux, límites de recursos y controles específicos de *CronJob*. | Restringir privilegios del proceso *batch*, evitar ejecuciones concurrentes y limitar el impacto de trabajos fallidos o prolongados. |

En el *workload* `web` se aplicó un `securityContext` a nivel de *pod* con `runAsNonRoot:true` y `seccompProfile` configurado como `RuntimeDefault`. A nivel de contenedor se deshabilitó la escalada de privilegios mediante `allowPrivilegeEscalation: false` y se eliminaron las capacidades Linux mediante `capabilities.drop: ["ALL"]`. También se definieron peticiones y límites de recursos, concretamente `50m` de CPU y `64Mi` de memoria como valores solicitados, y `250m` de CPU y `256Mi`

de memoria como límites. Finalmente, se añadieron sondas HTTP de disponibilidad y vida sobre la ruta `/` y el puerto `8080`.

En el caso de la API, se aplicó un endurecimiento equivalente en aquellos controles compatibles con la imagen actual. Se configuró el perfil `seccomp` como `RuntimeDefault`, se deshabilitó la escalada de privilegios y se eliminaron las capacidades Linux no necesarias. Además, se añadieron peticiones de recursos de `100m` de CPU y `128Mi` de memoria, así como límites de `500m` de CPU y `512Mi` de memoria. También se incorporaron sondas HTTP de disponibilidad y vida utilizando la ruta `/api/health` sobre el puerto `8000`, coherente con el *endpoint* de salud ya utilizado por el balanceador de carga.

El servicio `svc`, al ejecutarse como *CronJob*, recibió un tratamiento específico. Se aplicaron los mismos controles de reducción de privilegios compatibles con su imagen: `seccompProfile: RuntimeDefault`, `allowPrivilegeEscalation: false`, eliminación de capacidades Linux y límites de recursos. Además, se añadieron controles propios de la ejecución batch, como `concurrencyPolicy:Forbid`, `successfulJobsHistoryLimit:3`, `failedJobsHistoryLimit: 3`, `backoffLimit: 2` y `activeDeadlineSeconds: 1800`. Estas opciones permiten evitar ejecuciones concurrentes del proceso, limitar el histórico de trabajos y acotar la duración máxima de una ejecución.

Sin embargo, durante la aplicación de estos controles se detectó una limitación relevante relacionada con `runAsNonRoot`. Este control pudo aplicarse correctamente al *workload* `web`. Sin embargo, al aplicarlo inicialmente sobre la API, Kubernetes generó un error de tipo `CreateContainerConfigError`. El evento asociado indicaba que el contenedor tenía configurado `runAsNonRoot`, pero la imagen declaraba un usuario no numérico, `app`, por lo que Kubernetes no podía verificar que dicho usuario no correspondiera a `root`.

El problema no implica necesariamente que la imagen de la API se ejecute como usuario privilegiado. El Dockerfile define `USER app`, pero al tratarse de un identificador simbólico y no de un UID numérico, Kubernetes no puede comprobar de forma estática que ese usuario sea distinto de `0`. Por este motivo, y para evitar introducir cambios no validados en las imágenes o en los Dockerfiles, se retiró `runAsNonRoot` de la API. Dado que el servicio `svc` sigue un patrón equivalente, también se evitó forzar este control en su manifiesto.

Esta decisión mantiene un enfoque prudente y coherente con el alcance de la migración. Al fin y al cabo, el objetivo es minimizar la modificación de las imágenes

de la app y centrarnos en modificaciones respecto a la arquitectura y despliegue. Por lo tanto, no se fijaron valores arbitrarios de `runAsUser` o `runAsGroup`, no se asumió un UID sin validar y se mantuvieron el resto de controles de endurecimiento compatibles con los *workloads* actuales.

En conjunto, este bloque refuerza la postura de seguridad de la arquitectura migrada al limitar las capacidades disponibles para los contenedores y reducir su margen de actuación en caso de compromiso.

### Restricción del plano de control de Amazon EKS

Esta fase aborda específicamente el acceso al plano de control de Kubernetes. El objetivo fue evitar que el *endpoint* de la API de EKS permaneciera accesible públicamente desde Internet y trasladar la administración del clúster a una vía privada, autenticada y controlada.

En la solución base, el clúster mantenía habilitado tanto el *endpoint* público como el *endpoint* privado del plano de control. Aunque el acceso efectivo a Kubernetes seguía protegido mediante los mecanismos de autenticación y autorización correspondientes, como IAM y RBAC, el servidor API continuaba siendo alcanzable desde redes externas. Desde una perspectiva *Zero Trust*, esta situación representa una superficie administrativa expuesta que conviene reducir, ya que el acceso al plano de control debe limitarse al mínimo conjunto de orígenes y mecanismos necesarios.

La implantación de este bloque se realizó de forma progresiva en tres fases, con el fin de evitar la pérdida de acceso operativo al clúster durante la migración.

**Fase 1. Preparación de una vía privada de administración** Antes de deshabilitar el *endpoint* público de EKS, se creó una vía alternativa de administración dentro de la propia VPC. Para ello, se desplegó una instancia EC2 privada destinada a tareas administrativas sobre Kubernetes. Esta instancia se ubicó sin IP pública, no expone el puerto SSH y se accede a ella mediante AWS Systems Manager Session Manager. Además, su grupo de seguridad no define reglas de entrada, reduciendo así la exposición directa de la instancia.

El bastión dispone únicamente de la salida necesaria para comunicarse con servicios de AWS y con el *endpoint* privado del plano de control de EKS. Durante el arranque de la instancia se instalaron automáticamente las herramientas necesarias para la administración del clúster, principalmente AWS CLI, `kubectl` y `helm`. De

esta forma, el bastión queda preparado como punto de operación interno antes de modificar la exposición del servidor API.

Para permitir su gestión mediante Session Manager, se creó un rol IAM específico asociado a la instancia. Este rol incluye la política administrada `AmazonSSMManagedInstanceCore`, necesaria para que la instancia pueda registrarse y ser accedida mediante SSM. Además, se añadieron permisos mínimos de AWS para describir el clúster EKS, así como permisos de lectura sobre el *bucket* S3 utilizado para almacenar temporalmente los artefactos de Kubernetes.

El acceso del bastión al propio clúster se configuró mediante una entrada de acceso de EKS. Para ello, el rol IAM de la instancia se asoció al clúster mediante *EKS Access Entry* y se vinculó a una política administrativa de EKS. Esta decisión simplifica la operativa del entorno del TFM, ya que el bastión actúa como estación administrativa del clúster. En un entorno productivo, esta política podría sustituirse por permisos más granulares en función de los perfiles de operación definidos.

Finalmente, se añadió una regla de grupo de seguridad para permitir que el bastión pudiera comunicarse con el plano de control de EKS por TCP 443. Con ello, antes de cerrar el *endpoint* público ya existía una vía de administración privada y funcional dentro de la VPC.

**Fase 2. Automatización de la operación Kubernetes desde el bastión** Una vez validado el acceso manual desde el bastión, se adaptó el *script* automatizado para operar los recursos de Kubernetes de forma remota mediante SSM. Esta adaptación permite mantener la ejecución de Terraform desde el equipo local, mientras que las operaciones que dependen de `kubectl` se trasladan al bastión privado.

En el flujo de despliegue, la infraestructura continúa gestionándose desde el equipo local, pero los manifiestos de Kubernetes se empaquetan y se transfieren a un *bucket* S3 privado. Posteriormente, el bastión descarga el artefacto y aplica los recursos dentro del clúster utilizando `kubectl`. En este mismo flujo remoto también se contempla la instalación o actualización de AWS Load Balancer Controller mediante `helm`, evitando depender del acceso local al *endpoint* de Kubernetes.

En la consulta de estado, la información asociada a AWS y Terraform se obtiene desde el equipo local, mientras que el estado de Kubernetes se consulta desde el bastión. De esta forma, los nodos, *pods*, servicios, *Ingress*, *NetworkPolicies* y componentes como AWS Load Balancer Controller se revisan desde dentro de la VPC. En el flujo de parada, los recursos de Kubernetes se eliminan primero desde el bastión y,

posteriormente, se destruyen los recursos de infraestructura correspondientes, como EKS, RDS, NAT y el propio bastión mediante Terraform.

Para transportar los manifiestos de Kubernetes se creó un *bucket* S3 privado de artefactos. Este *bucket* tiene bloqueo de acceso público, cifrado SSE-S3 y una regla de ciclo de vida para expirar artefactos temporales. El rol del bastión solo dispone de permisos de lectura sobre dicho *bucket*. Esta decisión evita clonar el repositorio completo dentro del bastión y elimina la necesidad de utilizar SSH o SCP, manteniendo la transferencia de artefactos dentro de mecanismos controlados mediante IAM, S3 y SSM.

Adicionalmente, se añadió un modo auxiliar de contingencia para ejecutar únicamente la parte de infraestructura sin realizar operaciones sobre Kubernetes. Este mecanismo permite separar la gestión de Terraform de la gestión de manifiestos cuando sea necesario, por ejemplo, durante tareas de recuperación o mantenimiento.

**Fase 3. Deshabilitación del *endpoint* público de EKS** Tras validar la administración desde el bastión, se modificó la configuración del clúster EKS para deshabilitar el *endpoint* público del plano de control y mantener habilitado el *endpoint* privado. La configuración objetivo quedó expresada mediante los siguientes valores:

- `endpoint_public_access = false`
- `endpoint_private_access = true`

Con esta configuración, el servidor API de Kubernetes deja de estar accesible directamente desde Internet. La administración del clúster debe realizarse desde el bastión privado, accedido mediante AWS Systems Manager Session Manager.

Este cambio modifica de forma significativa el modelo operativo del clúster. Antes de la migración, el equipo local podía ejecutar `kubectl` directamente contra el *endpoint* público de EKS. Tras la restricción, dicho acceso directo deja de estar disponible, y las operaciones administrativas sobre Kubernetes se canalizan a través del bastión. Por tanto, el plano de control queda separado de la red pública y se reduce la exposición de uno de los componentes más sensibles de la arquitectura.

La Tabla 4.5 resume la evolución del modelo de administración del plano de control.

**Tabla 4.5:** Evolución del acceso al plano de control de Amazon EKS

| **Elemento** | **Situación inicial** | **Situación migrada** |
|---|---|---|
| *Endpoint* público de EKS | Habilitado | Deshabilitado |
| *Endpoint* privado de EKS | Habilitado | Habilitado |
| Origen de administración | Equipo local mediante `kubectl` | Bastión privado dentro de la VPC |
| Canal de acceso administrativo | Acceso directo al *endpoint* público | AWS Systems Manager Session Manager |
| Exposición del servidor API | Alcanzable desde Internet | Accesible únicamente desde la VPC o redes conectadas |
| Transferencia de manifiestos | Uso local de `infra/k8s` | Artefactos temporales en S3 privado y ejecución remota desde bastión |

No obstante, la solución presenta algunas limitaciones. En primer lugar, la instancia privada utiliza salida hacia servicios de AWS a través de NAT Gateway, lo que simplifica la operación del entorno pero mantiene dependencia de conectividad saliente general. Una versión más estricta podría sustituir o complementar este enfoque mediante VPC *endpoints* para servicios como SSM, STS, EKS, S3 u otros servicios necesarios. En segundo lugar, el rol del bastión dispone de permisos administrativos sobre el clúster para simplificar la operativa del TFM; en un entorno productivo sería recomendable separar estos permisos en roles más granulares. En tercer lugar, el *bucket* S3 de artefactos utiliza cifrado SSE-S3, suficiente para este escenario, aunque podría reforzarse mediante claves KMS gestionadas por el cliente. Finalmente, el flujo de administración pasa a depender del correcto funcionamiento de SSM y del bastión, por lo que estos componentes se convierten en elementos críticos de operación.

Este bloque representa una de las modificaciones más relevantes de la migración hacia *Zero Trust*, ya que elimina la exposición pública del plano de control de Kubernetes y traslada la administración a un canal privado, autenticado y auditable. Con ello, la arquitectura reduce la confianza implícita en la accesibilidad pública del clúster y establece una vía de operación más controlada.

**Implementación de trazabilidad y auditoría orientada a *Zero Trust***

El último bloque de la migración se centró en incorporar una capa de trazabilidad y auditoría sobre la infraestructura desplegada en AWS y Amazon EKS. Esta fase se orienta a disponer de fuentes de información que permitan revisar, atribuir e investigar operaciones administrativas realizadas sobre el entorno.

Desde una perspectiva *Zero Trust*, la trazabilidad es un elemento complementario a los controles de acceso. No impide por sí misma que se produzca una acción indebida, pero permite asociar las operaciones a identidades concretas, revisar cambios realizados sobre recursos críticos y disponer de información para auditoría o investigación posterior.

Antes de esta fase, el clúster EKS ya tenía habilitados *logs* del plano de control para los tipos `api`, `audit` y `authenticator`. Estos *logs* permiten revisar actividad asociada al servidor API de Kubernetes, eventos de auditoría y autenticación contra el clúster. Además, la administración de Kubernetes ya se realizaba mediante un bastión privado gestionado con AWS Systems Manager Session Manager, lo que centraliza la operación administrativa en una vía privada y controlada. Sin embargo, no existía todavía un recurso CloudTrail gestionado ni un *bucket* S3 dedicado para almacenar *logs* de auditoría.

Para cubrir esta necesidad, se añadió una configuración específica de CloudTrail mediante Terraform. El recurso principal está orientado a registrar eventos de gestión de la cuenta relacionados con los servicios utilizados en la arquitectura.

También se añadieron recursos S3 específicos para almacenar los *logs* generados por CloudTrail. Para ello, se creó un nuevo *bucket*, configurado con bloqueo de acceso público, controles de propiedad, versionado, cifrado en reposo mediante SSE-S3 con AES256 y una regla de ciclo de vida para expirar los *logs* a los 30 días. Asimismo, se definió una política de *bucket* con los permisos mínimos necesarios para que CloudTrail pueda comprobar la ACL del *bucket* y escribir los objetos de log correspondientes.

La Tabla 4.6 resume los principales mecanismos de trazabilidad considerados en esta fase.

**Tabla 4.6:** Mecanismos de trazabilidad orientados a *Zero Trust*

| **Mecanismo** | **Recurso implicado** | **Finalidad** |
|---|---|---|
| *Logs* del plano de control de EKS | Amazon EKS y CloudWatch *logs* | Registrar actividad del servidor API, autenticación y eventos de auditoría de Kubernetes. |
| Auditoría de operaciones AWS | AWS CloudTrail | Registrar llamadas a la API de AWS realizadas por usuarios, roles, servicios, AWS CLI o Terraform. |
| Almacenamiento de *logs* de auditoría | Amazon S3 | Conservar los ficheros generados por CloudTrail en un *bucket* privado, cifrado y con ciclo de vida limitado. |
| Administración remota controlada | Bastión privado y AWS Systems Manager | Centralizar operaciones administrativas sobre Kubernetes en una instancia privada accesible mediante SSM. |
| Trazabilidad de artefactos operativos | *Bucket* S3 de artefactos Kubernetes | Permitir transferencia controlada de manifiestos hacia el bastión sin clonar el repositorio ni utilizar SSH/SCP. |

El *trail* se configuró como multi-región, con inclusión de eventos de servicios globales y validación de integridad de los ficheros de log. Además, se registran tanto operaciones de lectura como de escritura realizadas sobre la API de AWS.

No se configuraron *data events*, CloudTrail Lake ni CloudTrail Insights. Esta decisión mantiene el alcance del bloque acotado y evita convertir la solución en una plataforma de monitorización o análisis avanzado. El objetivo es disponer de una fuente base de auditoría de bajo coste para operaciones de gestión, suficiente para revisar actividad administrativa sobre servicios como IAM, STS, EKS, EC2, SSM, S3, RDS y llamadas realizadas mediante Terraform o AWS CLI.

El *trail* quedó configurado como una fuente base de auditoría de la arquitectura migrada, por lo que no se parametrizó mediante una variable de activación opcional. Se considera parte de la postura mínima de trazabilidad del entorno *Zero Trust*.

Tras aplicar los cambios, la arquitectura dispone de una fuente centralizada de auditoría para las operaciones realizadas sobre AWS. CloudTrail almacena sus registros en un *bucket* S3 privado, cifrado y con una política de ciclo de vida que limita la conservación de los *logs* a 30 días. Esta configuración permite mantener trazabilidad suficiente para el contexto del TFM sin introducir una solución de observabilidad sobredimensionada ni costes innecesarios.

El alcance de esta solución se mantuvo deliberadamente acotado. No se añadieron herramientas pesadas de observabilidad, como Prometheus, Grafana, OpenSearch, Fluent Bit o un SIEM completo. Tampoco se incorporó CloudTrail Lake ni se configuraron eventos de datos, ya que estos mecanismos habrían incrementado el coste y la complejidad del entorno. La decisión adoptada proporciona una base de auditoría suficiente para la arquitectura propuesta.

En conjunto, este bloque completa la implementación *Zero Trust* añadiendo una capa de responsabilidad y visibilidad sobre las operaciones realizadas en AWS y Kubernetes. La arquitectura no solo restringe identidades, comunicaciones y exposición del plano de control, sino que también conserva registros que permiten revisar la actividad administrativa del entorno.

## 4.3. Resultado

### 4.3.1. Diseño final

La Figura 4.2 muestra el diseño final de la arquitectura tras la migración. En ella se reflejan los principales elementos incorporados durante la implementación: CloudTrail y su *bucket* S3 dedicado para *logs* de auditoría, el bastión EC2 privado accesible mediante AWS Systems Manager Session Manager, el *bucket* S3 de artefactos Kubernetes, el *endpoint* privado de Amazon EKS, los roles IRSA asociados a los *workloads* `api` y `svc`, las `NetworkPolicies` aplicadas en el *namespace* de la aplicación y los mecanismos de endurecimiento de *workloads*.

Además, el diagrama diferencia los flujos principales de la arquitectura resultante. Los usuarios externos acceden a la aplicación a través del balanceador de carga, mientras que las tareas administrativas se canalizan mediante AWS Systems Manager y el bastión privado. CloudWatch queda representado como servicio asociado a

los *logs* del plano de control de EKS, mientras que CloudTrail registra eventos de gestión de AWS y almacena dichos registros en un *bucket* S3 dedicado.

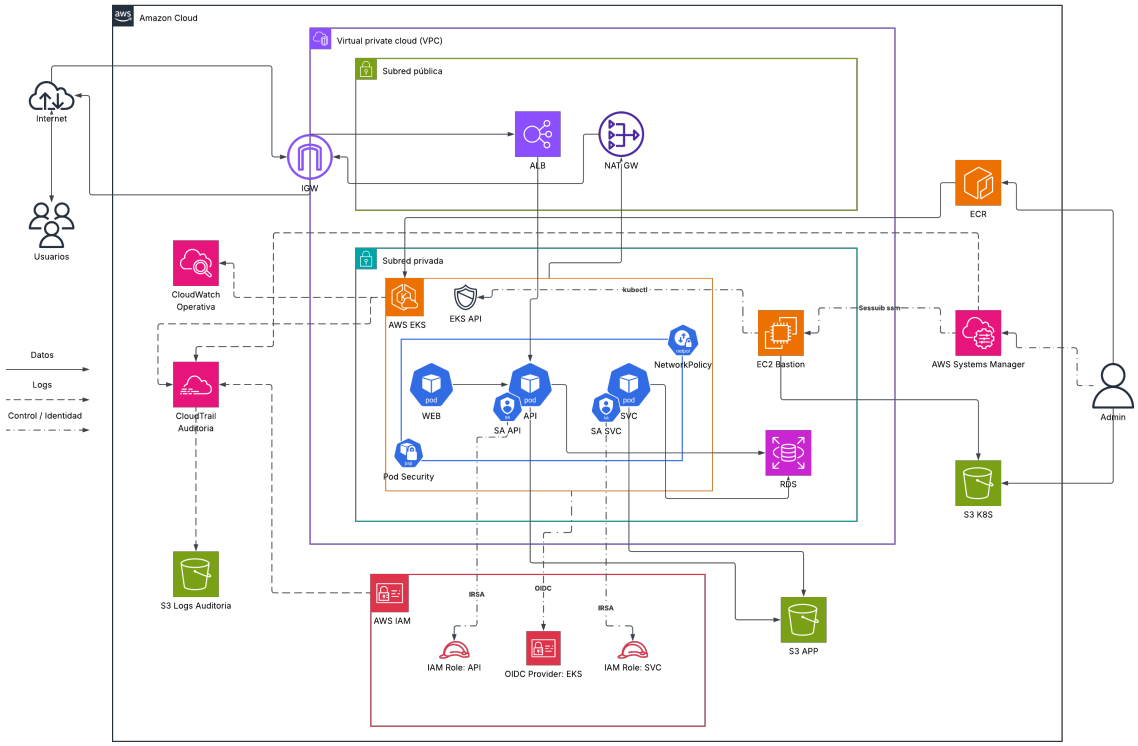



**Figura 4.2:** Diagrama definitivo de arquitectura *cloud* de la aplicación aplicando *Zero Trust*. Fuente: elaboración propia.

### 4.3.2. Estado final

Una vez completada la implementación de los controles definidos durante la migración, se comprobó el estado final de la solución desplegada.

En primer lugar, se verificó que los *workloads* principales de la aplicación se desplegaban correctamente en Amazon EKS y que los servicios `web`, `api` y `svc` mantenían su comportamiento esperado. Asimismo, se comprobó que el acceso externo a la aplicación seguía realizándose a través del flujo previsto, utilizando el balanceador de carga y el `Ingress`, sin exponer directamente los *pods* de la aplicación.

También se revisaron los cambios relacionados con la identidad de los *workloads*. En particular, se comprobó que los `ServiceAccounts` utilizados por `api` y `svc` estaban asociados a sus respectivos roles IAM mediante IRSA, y que las variables de credenciales estáticas de S3 ya no estaban presentes en los manifiestos ni en los *pods* desplegados. De esta forma, el acceso al almacenamiento de objetos pasó a

depender de identidades asociadas al *workload* y no de secretos estáticos definidos en Kubernetes.

Respecto a la segmentación interna, se verificó la presencia de las `NetworkPolicies` aplicadas en el *namespace* de la aplicación. Estas políticas permiten representar el cambio desde un modelo interno más permisivo hacia una comunicación explícitamente definida entre componentes, manteniendo únicamente los flujos necesarios para el funcionamiento de la solución.

Por último, se verificó el nuevo modelo de administración privada del clúster y la trazabilidad de acciones en AWS. El acceso directo desde el equipo local al plano de control de EKS dejó de estar disponible tras deshabilitar el *endpoint* público, mientras que la administración mediante el bastión privado y AWS Systems Manager Session Manager permitió ejecutar comandos `kubectl` desde dentro de la VPC. Además, CloudTrail quedó configurado para registrar eventos de gestión de AWS y almacenar sus *logs* en un *bucket* S3 dedicado, complementando los *logs* del plano de control de EKS disponibles en CloudWatch.

La Tabla 4.7 relaciona las brechas identificadas durante el análisis inicial con el resultado finalmente aplicado.

**Tabla 4.7:** Trazabilidad entre brechas identificadas y resultado final de la migración

| ID | Brecha identificada | Resultado aplicado | Estado |
|---|---|---|---|
| ZT1 | Uso de credenciales estáticas de Amazon S3 dentro de Kubernetes. | Se eliminaron las variables de credenciales estáticas de los *workloads* y el acceso a S3 pasó a realizarse mediante IRSA. | Aplicado |
| ZT2 | Permisos de acceso a S3 asociados al rol de los nodos de EKS. | Se retiraron los permisos S3 del rol de nodos y se crearon roles IAM específicos para los *workloads* `api` y `svc`. | Aplicado |

*Continúa en la siguiente página*

**Tabla 4.7:** (continuación)

| ID | Brecha identificada | Resultado aplicado | Estado |
|---|---|---|---|
| ZT3 | Ausencia de políticas de red internas en Kubernetes. | Se habilitó el soporte de `NetworkPolicy` y se aplicaron políticas de denegación por defecto y reglas explícitas para los flujos necesarios. | Aplicado |
| ZT4 | Configuración de ejecución de *pods* con endurecimiento limitado. | Se añadieron controles de `securityContext`, perfil `seccomp`, eliminación de capacidades, bloqueo de escalada de privilegios, límites de recursos y sondas de estado. | Aplicado parcialmente |
| ZT5 | *Endpoint* público del plano de control de Amazon EKS. | La medida evolucionó respecto al planteamiento inicial: se deshabilitó el *endpoint* público y la administración pasó a realizarse desde un bastión privado mediante AWS Systems Manager Session Manager. | Aplicado |
| ZT6 | Observabilidad no orientada específicamente a los controles de seguridad implantados. | Se mantuvieron los *logs* del plano de control de EKS en CloudWatch y se incorporó CloudTrail con almacenamiento de *logs* de auditoría en un *bucket* S3 dedicado. | Aplicado parcialmente |

A continuación se recogen las evidencias visuales del estado final de la solución tras la migración:

- La Figura 4.3 muestra el estado general de los recursos Kubernetes consultados desde el bastión.

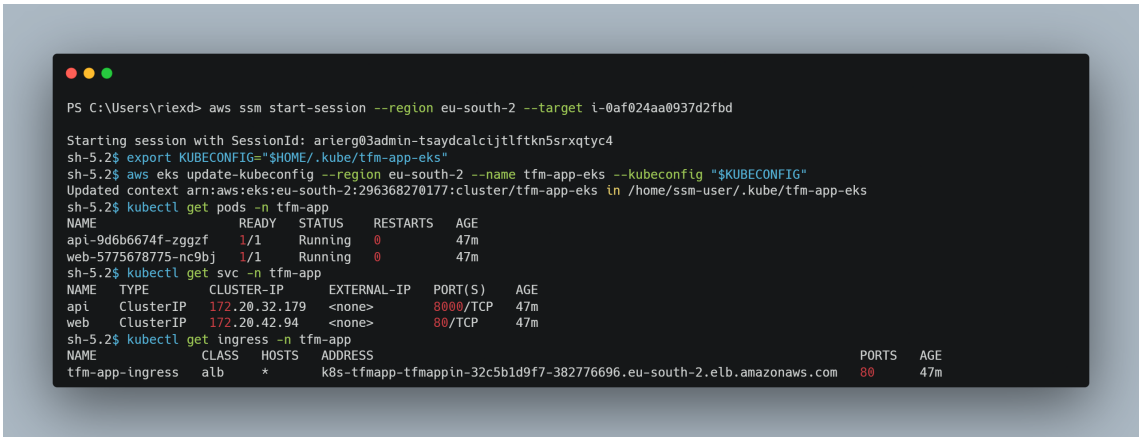



**Figura 4.3:** Estado de *pods*, servicios e *Ingress* tras la migración a *Zero Trust* consultado desde el bastión mediante AWS Systems Manager. Fuente: captura propia.

- Las Figuras 4.4, 4.5 y 4.6 documentan la nueva vía privada de administración, basada en una instancia EC2 sin exposición pública, un *bucket* S3 de artefactos Kubernetes y sesiones de AWS Systems Manager Session Manager.

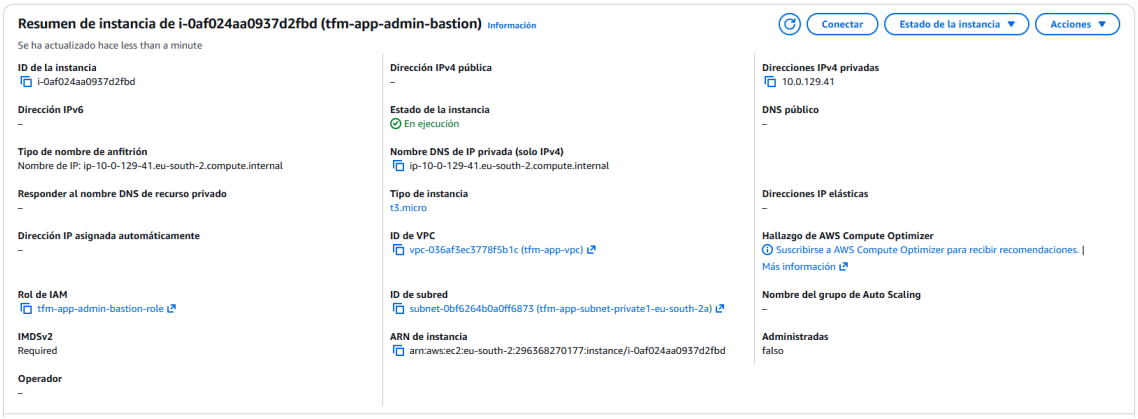



**Figura 4.4:** Instancia EC2 privada utilizada como bastión para la administración del clúster. Fuente: captura propia.

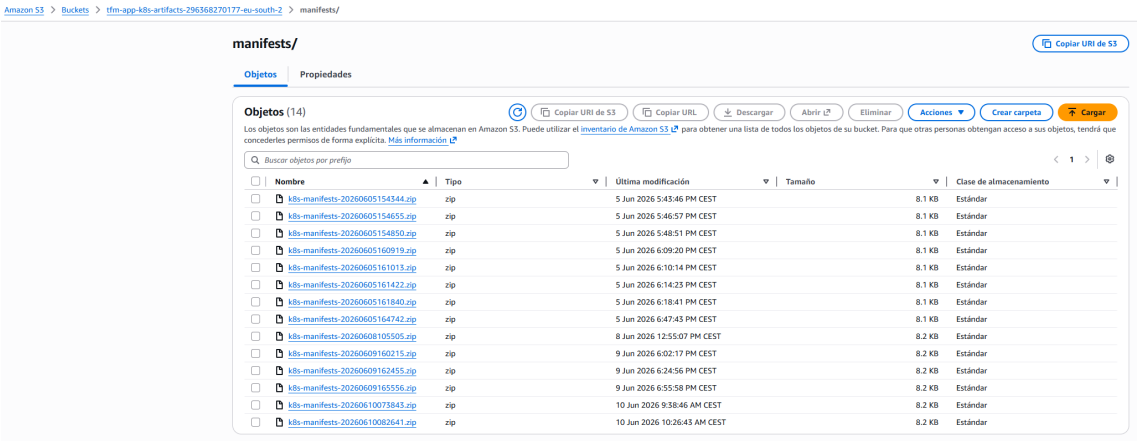



**Figura 4.5:** *Bucket* S3 utilizado para transferir artefactos Kubernetes al bastión privado. Fuente: captura propia.

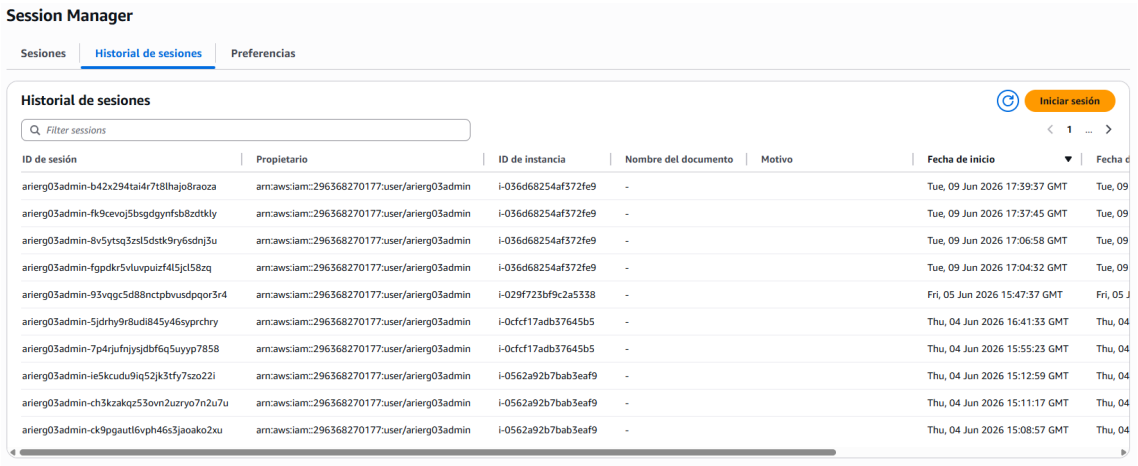



**Figura 4.6:** Historial de sesiones de administración mediante AWS Systems Manager Session Manager. Fuente: captura propia.

- Las Figuras 4.7 y 4.8 muestran la configuración privada del *endpoint* de Amazon EKS y el fallo esperado del acceso local directo.

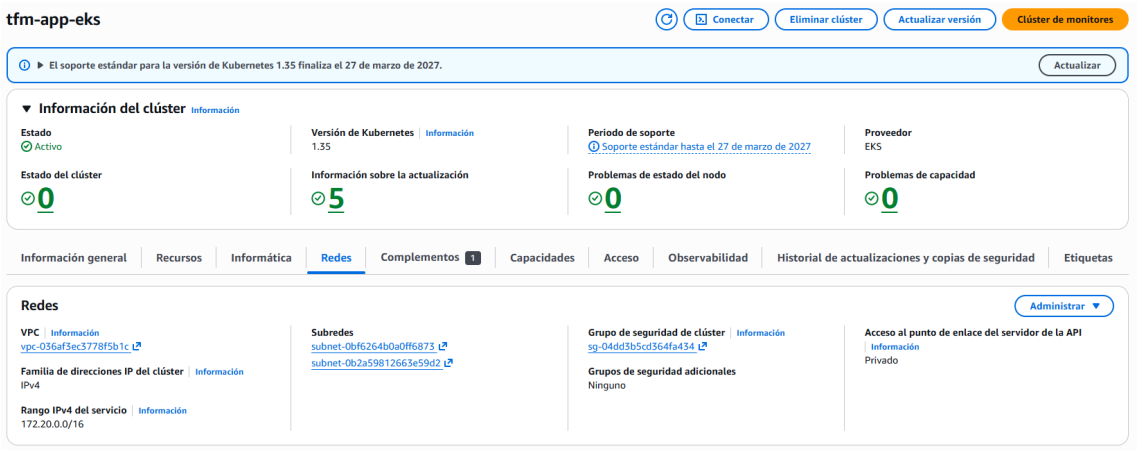



**Figura 4.7:** Configuración privada del punto de enlace del servidor de la API de Amazon EKS. Fuente: captura propia.

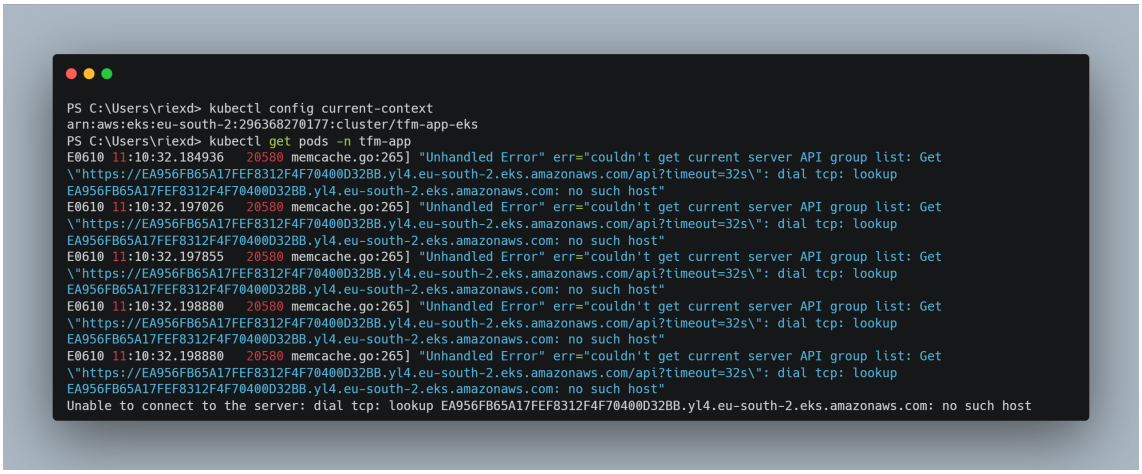



**Figura 4.8:** Fallo esperado del acceso local directo al *endpoint* de Amazon EKS. Fuente: captura propia.

- Las Figuras 4.9, 4.10 y 4.11 recogen las evidencias asociadas a identidad mediante IRSA, eliminación de credenciales estáticas y segmentación interna mediante `NetworkPolicies`.




**Figura 4.9:** ServiceAccounts de los *workloads* asociados a roles IAM mediante IRSA. Fuente: captura propia.




**Figura 4.10:** Ausencia de credenciales S3 estáticas en el *pod* de la API. Fuente: captura propia.




**Figura 4.11:** NetworkPolicies aplicadas. Fuente: captura propia.

- Finalmente, las Figuras 4.12 y 4.13 muestran la configuración de CloudTrail y la generación de *logs* de auditoría en el *bucket* S3 dedicado.

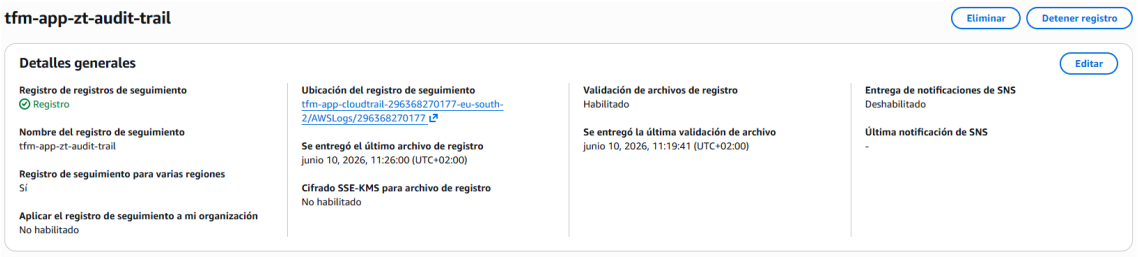



**Figura 4.12:** *Trail* de CloudTrail configurado para la auditoría de eventos de gestión de AWS. Fuente: captura propia.

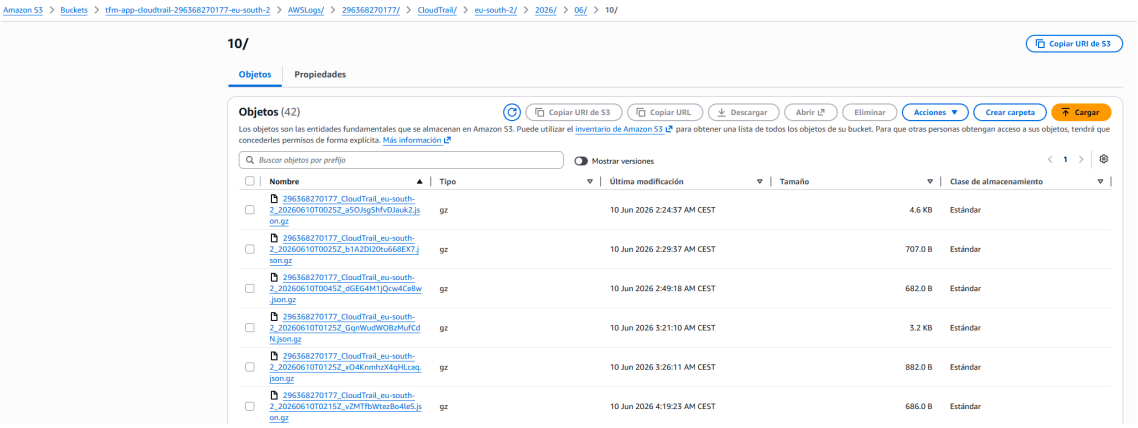



**Figura 4.13:** *Logs* de auditoría de CloudTrail almacenados en el *bucket* S3 dedicado. Fuente: captura propia.

# Capítulo 5

# Evaluación

Una vez implementadas la arquitectura base y su evolución hacia una solución alineada con *Zero Trust*, resulta necesario establecer un proceso de evaluación que permita comparar ambos estados de forma objetiva.

Para ello, se define una batería común de pruebas y métricas aplicada sobre las dos versiones de la solución. La utilización de los mismos criterios de evaluación permite identificar qué aspectos se mantienen, qué controles mejoran tras la migración y qué impacto adicional introduce la adopción de mecanismos *Zero Trust*. De este modo, los resultados obtenidos sirven como base para el análisis comparativo desarrollado en el capítulo posterior.

## 5.1. Proceso de evaluación

Se evalúan dos versiones de una misma solución *cloud*:

- Solución base: despliegue funcional en AWS con EKS, RDS, S3, ECR, ALB, VPC, IAM y Terraform.
- Solución *Zero Trust*: evolución de la misma solución, manteniendo la aplicación prácticamente inalterada, pero reforzando identidad, mínimo privilegio, segmentación, exposición pública, gestión de secretos, observabilidad y control de acceso entre componentes.

La comparación se realiza mediante una batería común de pruebas funcionales, de infraestructura, seguridad, rendimiento básico, coste y complejidad operativa.

## 5.2. Forma de evaluación

Para evaluar la solución se define una batería común de pruebas y métricas, aplicada tanto a la solución *cloud* base como a la solución *Zero Trust*. Esta batería se organiza en cuatro bloques:

- Funcionalidad.
- Seguridad.
- Rendimiento mínimo.
- Complejidad, despliegue y coste.

En primer lugar, la Tabla 5.1 recoge las pruebas funcionales, orientadas a comprobar que la aplicación mantiene su comportamiento esperado en ambos despliegues. Su función no es evaluar tiempos de respuesta ni realizar cálculos, sino simplemente verificar que las dos infraestructuras realizan su funcionalidad básica. A continuación, la Tabla 5.2 define las pruebas de seguridad, centradas en la exposición de la aplicación, los controles de acceso, los permisos y la segmentación entre componentes. La Tabla 5.3 recoge las pruebas de rendimiento mínimo, utilizadas para comparar tiempos de respuesta en operaciones representativas. Por último, la Tabla 5.4 presenta las métricas de complejidad, despliegue y coste, con las que se evalúa el esfuerzo necesario para implantar y operar cada arquitectura.

**Tabla 5.1:** Pruebas funcionales

| ID | Caso de prueba | Objetivo / qué mide |
|---|---|---|
| F1 | Comprobación de disponibilidad | Verificar que el *frontend* y la API están accesibles. |
| F2 | Autenticación de usuario | Verificar que un usuario puede iniciar sesión correctamente. |
| F3 | Creación de evento | Verificar que se puede crear un evento en el sistema. |
| F4 | Subida de imagen | Verificar que se puede asociar una imagen a un evento. |
| F5 | Consulta de evento y resultado *batch* | Verificar que los datos creados pueden recuperarse y que el servicio *batch* opera sobre ellos. |

**Tabla 5.2:** Pruebas de seguridad

| ID | Prueba | Objetivo / qué mide |
|---|---|---|
| S1 | Acceso sin token a *endpoint* privado | Comprobar que los *endpoints* protegidos no permiten el acceso a usuarios no autenticados. |
| S2 | Acceso con token inválido o caducado | Verificar que la API rechaza credenciales no válidas o expiradas, devolviendo una respuesta de error adecuada. |
| S3 | Intento de acceso directo a componentes internos desde el exterior | Evaluar si recursos como la API, la base de datos o el almacenamiento pueden ser accedidos directamente desde fuera del flujo previsto de la aplicación. |
| S4 | Intento de acceso directo a componentes internos desde el interior | Evaluar si recursos como la API, la base de datos o el almacenamiento pueden ser accedidos directamente desde dentro de la arquitectura de la aplicación. |
| S5 | Comprobación de privilegios IAM | Analizar si los permisos asignados a los servicios son amplios o si se ajustan al principio de mínimo privilegio. |
| S6 | Comprobación de segmentación y comunicación entre componentes | Verificar qué comunicaciones están permitidas entre los componentes de la arquitectura y si existen restricciones de red adecuadas. |

**Tabla 5.3:** Pruebas de rendimiento mínimo

| ID | Prueba | Objetivo / qué mide |
|---|---|---|
| R1 | Tiempo medio de respuesta de un *endpoint* simple | Medir la latencia básica de la aplicación mediante peticiones a un *endpoint* representativo, como `/health` o la consulta de eventos. |

*Continúa en la siguiente página*

**Tabla 5.3:** (continuación)

| ID | Prueba | Objetivo / qué mide |
|---|---|---|
| R2 | Tiempo de autenticación de usuario | Medir el tiempo necesario para completar el proceso de inicio de sesión y evaluar el posible impacto de los controles de autenticación. |
| R3 | Tiempo de subida de imagen | Medir el impacto temporal de una operación completa que involucra API, almacenamiento de objetos y persistencia de metadatos. |

**Tabla 5.4:** Métricas de complejidad, despliegue y coste

| ID | Métrica | Objetivo / qué mide |
|---|---|---|
| C1 | Tiempo de despliegue | Medir el tiempo necesario para desplegar la infraestructura y la aplicación, incluyendo la ejecución de Terraform y el despliegue de los manifiestos de Kubernetes. |
| C2 | Número de recursos *cloud* desplegados | Cuantificar la cantidad de recursos AWS necesarios para cada versión de la solución. |
| C3 | Número de políticas IAM, grupos de seguridad y reglas | Medir la complejidad de la configuración de seguridad asociada a permisos, identidades y comunicaciones de red. |
| C4 | Líneas de configuración y automatización | Estimar la complejidad técnica de la solución mediante el número de líneas en ficheros Terraform, manifiestos Kubernetes y *scripts* de despliegue. |
| C5 | Coste mensual teórico | Comparar el coste económico asociado a cada arquitectura a partir de la estimación obtenida en AWS. |

## 5.3. Casos de prueba

Las pruebas funcionales se ejecutaron mediante un *script* Python que realiza una secuencia de peticiones contra la aplicación desplegada en AWS, comenzando por la comprobación de disponibilidad, continuando con el inicio de sesión de un usuario de prueba, la creación de un evento, la subida de una imagen asociada y la consulta posterior del evento junto con el estado o resultado del procesamiento *batch*. La Tabla 5.5 muestra el estado, la observación y el tiempo de ejecución de cada prueba para cada entorno.

**Tabla 5.5:** Resultados de pruebas funcionales

| ID | Entorno | Estado | Observación | Tiempo |
|---|---|---|---|---|
| F1 | Base | Superada | El *frontend* y el *endpoint* de salud de la API respondieron correctamente con código HTTP 200. | 169,74 ms |
| | *Zero Trust* | Superada | El *frontend* y el *endpoint* de salud de la API respondieron correctamente con código HTTP 200. | 123,07 ms |
| F2 | Base | Superada | El inicio de sesión del usuario de prueba fue correcto, recibiéndose un *token* de tipo *bearer*. | 152,76 ms |
| | *Zero Trust* | Superada | El inicio de sesión del usuario de prueba fue correcto, recibiéndose un *token* de tipo *bearer*. | 144,25 ms |
| F3 | Base | Superada | El sistema permitió crear correctamente un evento, devolviendo código HTTP 201 e identificador de evento válido. | 43,10 ms |
| | *Zero Trust* | Superada | El sistema permitió crear correctamente un evento, devolviendo código HTTP 201 e identificador de evento válido. | 32,49 ms |

*Continúa en la siguiente página*

**Tabla 5.5:** (continuación)

| ID | Entorno | Estado | Observación | Tiempo |
|---|---|---|---|---|
| F4 | Base | Superada | La imagen de prueba se subió correctamente y quedó asociada al evento creado, devolviendo código HTTP 201 y URL de imagen. | 306,40 ms |
| | *Zero Trust* | Superada | La imagen de prueba se subió correctamente y quedó asociada al evento creado, devolviendo código HTTP 201 y URL de imagen. | 277,21 ms |
| F5 | Base | Superada | El evento creado pudo consultarse correctamente mediante la API. No existía todavía una ejecución *batch* disponible, por lo que el estado *batch* devolvió código HTTP 404 controlado. | 69,83 ms |
| | *Zero Trust* | Superada | El evento creado pudo consultarse correctamente mediante la API y el servicio *batch* se ejecutó correctamente desde el bastión mediante SSM, generando una ejecución con estado *success*. Se detectaron 4 eventos, se procesaron 4 correctamente y no se registraron fallos. | 12616,84 ms |

Las pruebas de seguridad se dividieron en dos grupos. Las pruebas S1, S2 y S3 se ejecutaron mediante *script* Python realizando peticiones HTTP contra la aplicación desplegada. Estas pruebas verifican que los *endpoints* protegidos rechazan peticiones sin token, que los *tokens* inválidos no permiten el acceso y que no existen *endpoints* internos o administrativos accesibles directamente desde el exterior. Se registra el tiempo de respuesta.

Las pruebas S4, S5 y S6 no se basan en peticiones funcionales a la aplicación, sino en la inspección de la configuración *cloud* desplegada. Para ello se recopilaron automáticamente intentos de conexión de manera interna, evidencias de IAM, grupos de seguridad, configuración del clúster EKS y recursos Kubernetes. Estas evidencias

permiten comparar la amplitud de permisos y el grado de segmentación entre la solución base y la solución *Zero Trust*. Al contrario que las tres pruebas anteriores, estas al no ser HTTP, no se registra el tiempo de respuesta. La Tabla 5.6 muestra los resultados obtenidos.

**Tabla 5.6:** Resultados de pruebas de seguridad

| ID | Entorno | Estado | Observación | Tiempo |
|---|---|---|---|---|
| S1 | Base | Superada | El *endpoint* privado `/api/events` rechazó correctamente el acceso sin *token*, devolviendo HTTP 401. | 55,28 ms |
| | *Zero Trust* | Superada | El *endpoint* privado `/api/events` rechazó correctamente el acceso sin *token*, devolviendo HTTP 401. | 112,15 ms |
| S2 | Base | Superada | El *endpoint* privado `/api/events` rechazó correctamente el acceso con un *token* inválido, devolviendo HTTP 401. | 66,50 ms |
| | *Zero Trust* | Superada | El *endpoint* privado `/api/events` rechazó correctamente el acceso con un *token* inválido, devolviendo HTTP 401. | 47,42 ms |
| S3 | Base | No superada | Se detectaron cinco *endpoints* internos o administrativos accesibles públicamente con HTTP 200: `/metrics`, `/admin`, `/docs`, `/openapi.json` y `/api/docs`. En cambio, la base de datos RDS no fue accesible directamente desde el exterior y el *bucket*/objeto S3 rechazó el acceso anónimo con HTTP 403. | 5889,72 ms |

*Continúa en la siguiente página*

**Tabla 5.6:** (continuación)

| ID | Entorno | Estado | Observación | Tiempo |
|---|---|---|---|---|
| | *Zero Trust* | Superada | No se detectaron *endpoints* internos accesibles públicamente. La base de datos RDS no fue accesible directamente desde el exterior y el *bucket*/objeto S3 rechazó el acceso anónimo con HTTP 403. | 5885,15 ms |
| S4 | Base | No superada | Desde un *pod* interno se pudo acceder directamente al servicio interno de salud de la API, a la documentación interna de la API y abrir conexión TCP directa con RDS. En cambio, el *bucket* y el objeto S3 rechazaron el acceso anónimo con HTTP 403. | – |
| | *Zero Trust* | Superada | Desde un *pod* interno no se pudo acceder directamente al servicio interno de salud de la API ni a la documentación interna de la API. Tampoco se pudo abrir conexión TCP directa con RDS, y el *bucket* y el objeto S3 rechazaron el acceso anónimo con HTTP 403. | – |
| S5 | Base | Superada parcial-mente | Se identificaron 11 roles IAM en la cuenta, de los cuales 6 resultan potencialmente relevantes para la solución. No se detectaron políticas de tipo administrador asociadas a estos roles, aunque el rol de nodos de EKS concentra permisos de EKS, ECR y una política *inline* de acceso a S3. | – |

*Continúa en la siguiente página*

**Tabla 5.6:** (continuación)

| ID | Entorno | Estado | Observación | Tiempo |
|---|---|---|---|---|
| | *Zero Trust* | Superada | Se identificaron 14 roles IAM en la cuenta, de los cuales 9 resultan potencialmente relevantes para la solución. No se detectaron políticas de tipo administrador asociadas a estos roles. Los permisos de acceso a S3 se separan mediante roles IRSA específicos para la API y el servicio *batch*, y el bastión privado utiliza un rol propio con permisos de SSM y políticas *inline* específicas. | – |
| S6 | Base | No superada completamente | La solución base presenta segmentación parcial mediante grupos de seguridad y servicios Kubernetes de tipo `ClusterIP`, pero no define `NetworkPolicies`. Además, existe un grupo de seguridad con entrada pública, todos los grupos analizados mantienen salida abierta a `0.0.0.0/0` y el *endpoint* público de EKS está habilitado para `0.0.0.0/0`. | – |

*Continúa en la siguiente página*

**Tabla 5.6:** (continuación)

| ID | Entorno | Estado | Observación | Tiempo |
|---|---|---|---|---|
| | *Zero Trust* | Superada | La solución *Zero Trust* presenta segmentación mediante servicios Kubernetes de tipo `ClusterIP` y 3 `NetworkPolicies`. Se identificaron 2 *pods*, 2 servicios internos y 1 *Ingress*. El *endpoint* público de EKS está deshabilitado, el *endpoint* privado está habilitado y no existen CIDR públicos asociados al acceso al plano de control. Se analizaron 6 grupos de seguridad, con 1 grupo con entrada pública asociada al Application Load Balancer. | – |

Las pruebas de rendimiento mínimo se ejecutaron mediante un *script* automatizado que realiza varias peticiones consecutivas, concretamente 20, sobre operaciones representativas de la aplicación. Para cada prueba se registraron los tiempos de respuesta individuales y se calcularon métricas agregadas como el valor mínimo, máximo, medio, mediano y percentil 95. El objetivo de estas pruebas no es realizar una prueba de carga intensiva, sino obtener una referencia básica del impacto temporal de las operaciones principales en ambas arquitecturas. La Tabla 5.7 muestra todas las métricas de manera ordenada para cada entorno analizado.

**Tabla 5.7:** Resultados de pruebas de rendimiento mínimo

| ID | Prueba | Métrica | Base | ***Zero Trust*** |
|---|---|---|---|---|
| R1 | *Endpoint* simple `/api/health` | Estado | Superada | Superada |
| | | Iteraciones correctas | 20/20 | 20/20 |
| | | Tiempo medio | 32,47 ms | 26,88 ms |
| | | Mediana | 32,41 ms | 26,61 ms |
| | | Mínimo | 31,34 ms | 25,37 ms |
| | | Máximo | 34,58 ms | 33,79 ms |

*Continúa en la siguiente página*

**Tabla 5.7:** (continuación)

| ID | Prueba | Métrica | Base | *Zero Trust* |
|---|---|---|---|---|
| | | Percentil 95 | 33,73 ms | 27,71 ms |
| R2 | Autenticación de usuario | Estado | Superada | Superada |
| | | Iteraciones correctas | 20/20 | 20/20 |
| | | Tiempo medio | 138,56 ms | 156,19 ms |
| | | Mediana | 135,19 ms | 162,67 ms |
| | | Mínimo | 131,70 ms | 122,54 ms |
| | | Máximo | 157,94 ms | 191,89 ms |
| | | Percentil 95 | 157,52 ms | 184,65 ms |
| R3 | Subida de imagen | Estado | Superada | Superada |
| | | Iteraciones correctas | 20/20 | 20/20 |
| | | Tiempo medio | 186,79 ms | 185,39 ms |
| | | Mediana | 183,56 ms | 166,77 ms |
| | | Mínimo | 137,31 ms | 140,09 ms |
| | | Máximo | 275,76 ms | 346,43 ms |
| | | Percentil 95 | 245,03 ms | 282,63 ms |

Para complementar los valores agregados de la Tabla 5.7, se generaron representaciones gráficas de los tiempos de respuesta obtenidos durante la ejecución de las pruebas. En primer lugar, las Figuras 5.1 y 5.2 resumen la distribución de tiempos de las tres operaciones evaluadas en la solución base y en la solución *Zero Trust*, respectivamente, permitiendo comparar su dispersión, mediana y posibles valores atípicos.

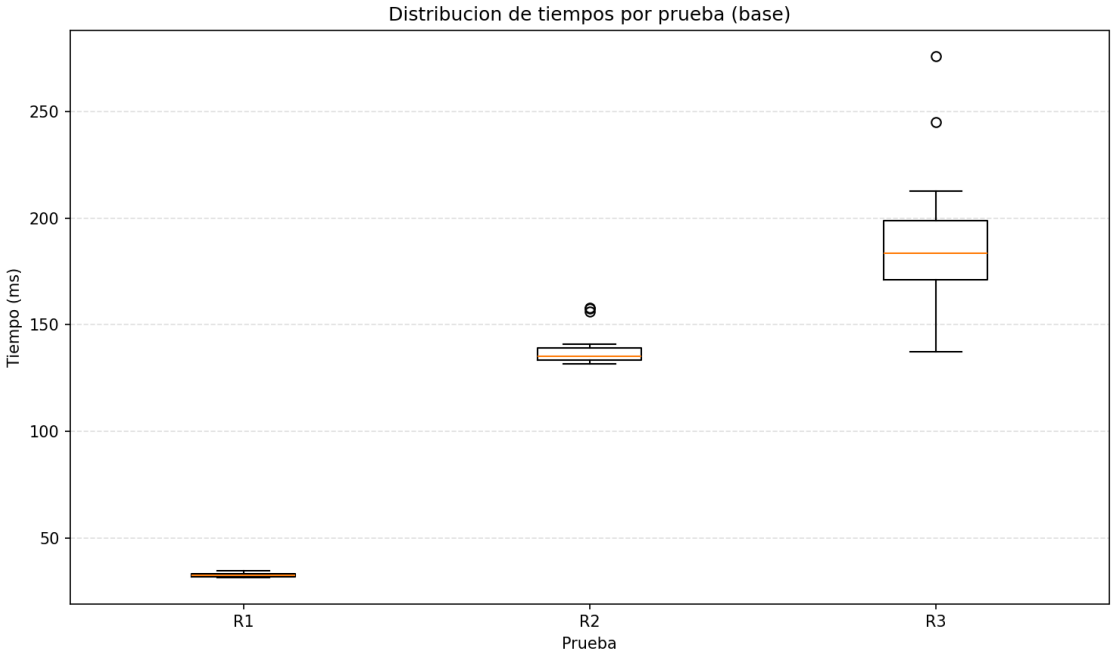



**Figura 5.1:** Distribución de tiempos de respuesta en las pruebas de rendimiento de la solución base. Fuente: elaboración propia.

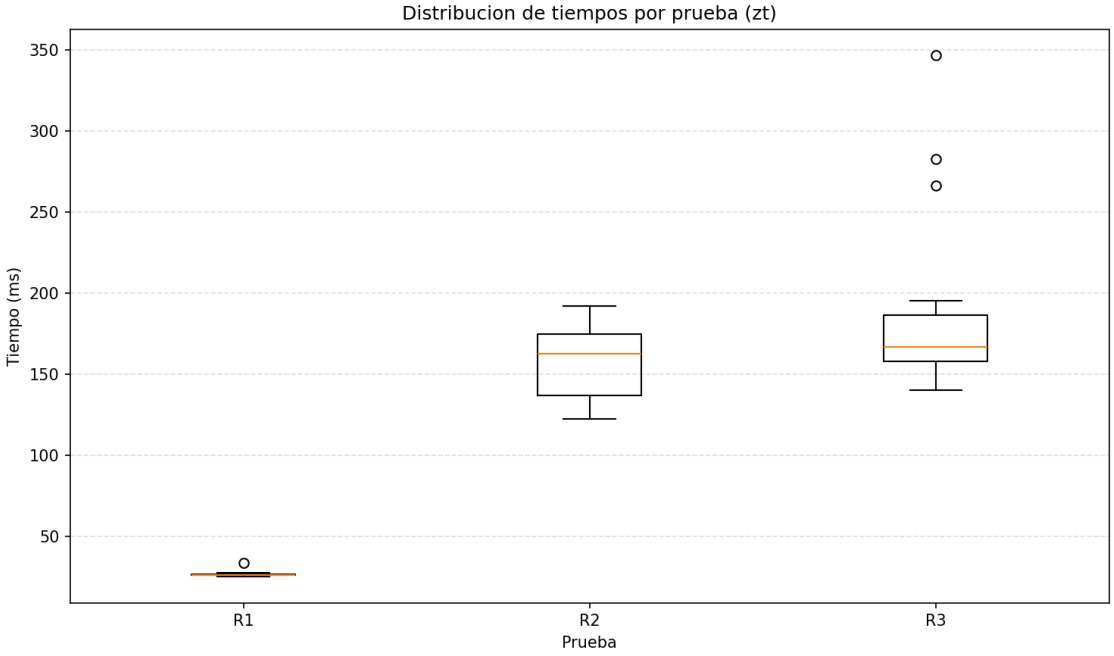



**Figura 5.2:** Distribución de tiempos de respuesta en las pruebas de rendimiento de la solución *Zero Trust*. Fuente: elaboración propia.

Posteriormente, las Figuras 5.3 y 5.4 muestran la evolución de los tiempos de respuesta del *endpoint* simple.

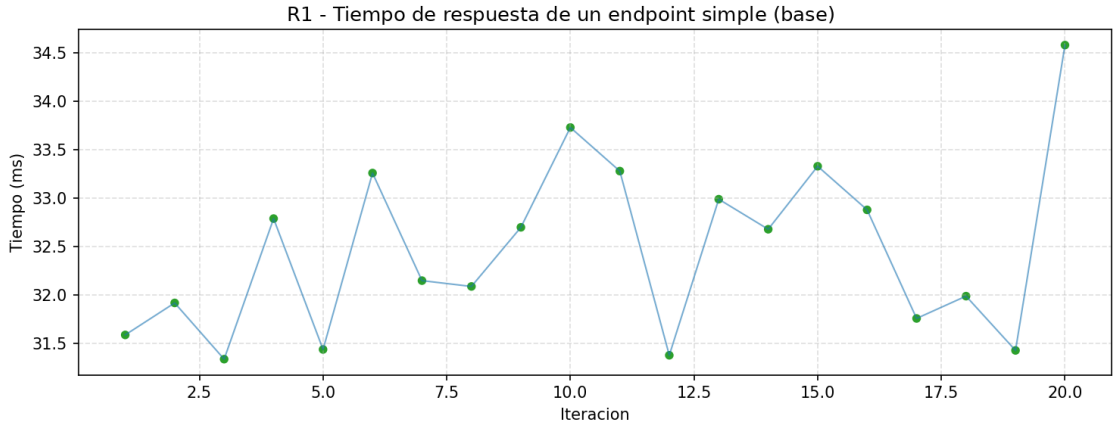



**Figura 5.3:** R1: *endpoint* simple (base). Fuente: elaboración propia.

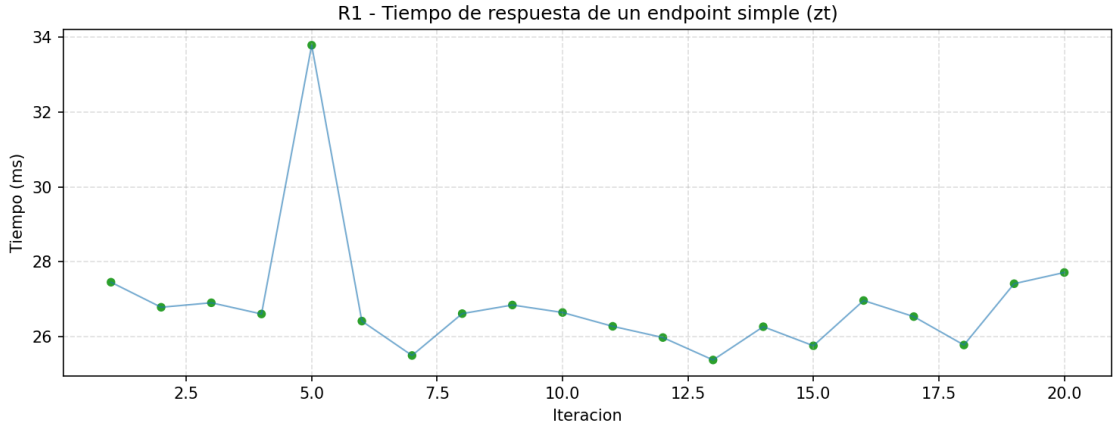



**Figura 5.4:** R1: *endpoint* simple (zt). Fuente: elaboración propia.

Las Figuras 5.5 y 5.6 recogen los resultados de autenticación de usuario.

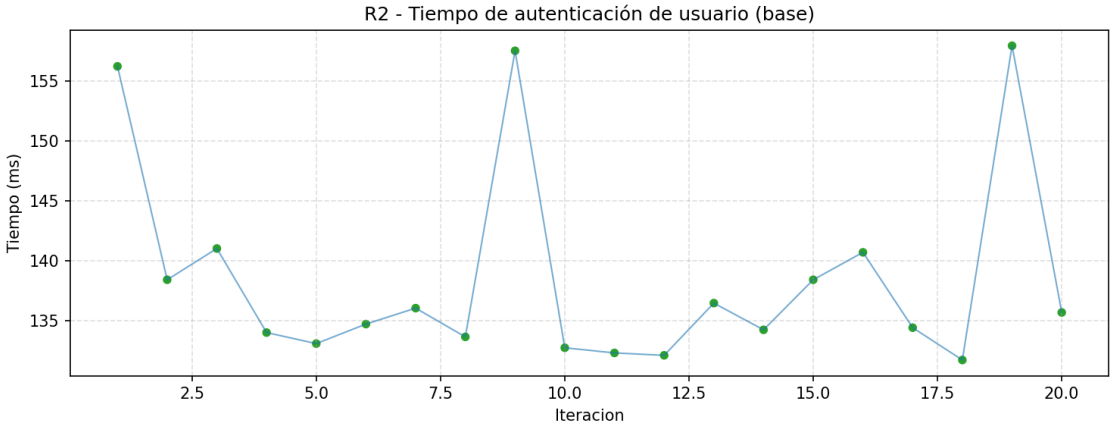



**Figura 5.5:** R2: autenticación de usuario (base). Fuente: elaboración propia.

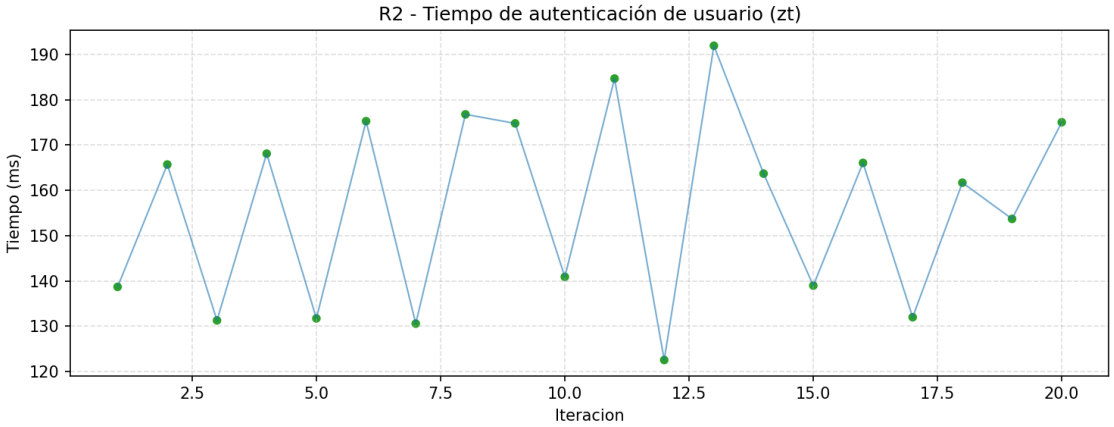



**Figura 5.6:** R2: autenticación de usuario (zt). Fuente: elaboración propia.

Finalmente, las Figuras 5.7 y 5.8 muestran los tiempos asociados a la subida de imagen.

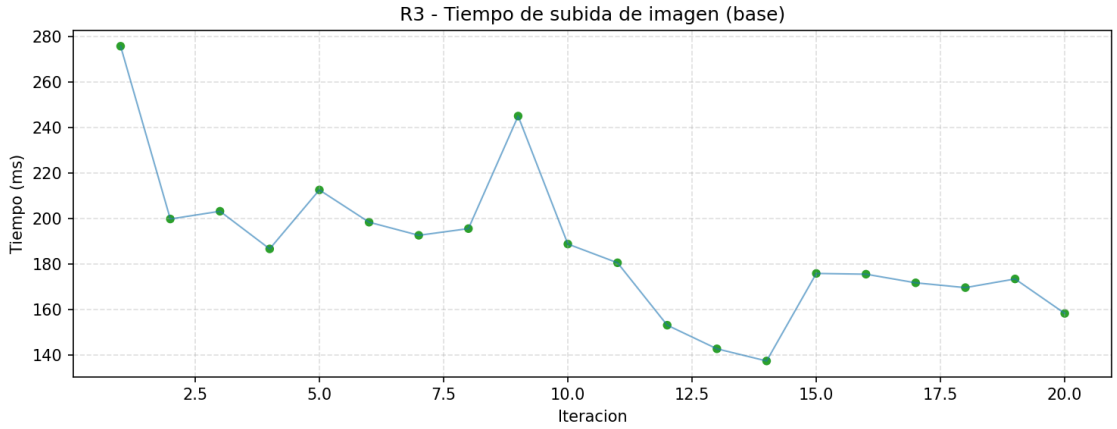



**Figura 5.7:** R3: subida de imagen (base). Fuente: elaboración propia.

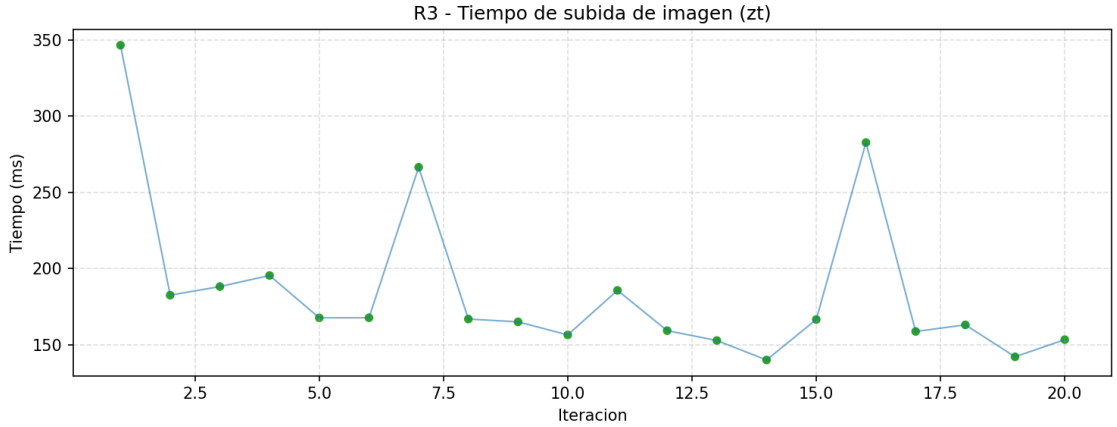



**Figura 5.8:** R3: subida de imagen (zt). Fuente: elaboración propia.

Las métricas de complejidad, despliegue y coste se recopilaron mediante un *script* automatizado basado en PowerShell y AWS CLI. Este *script* obtiene evidencias del estado de Terraform, del número de recursos gestionados, de la configuración de seguridad asociada a IAM y grupos de seguridad, del volumen de ficheros de configuración y automatización y del coste mensual. Dicho coste se ha calculado de forma teórica mediante AWS Pricing Calculator, reproduciendo los servicios y características principales de cada infraestructura desplegada.

Estas métricas permiten comparar el impacto operativo de la solución base y de la solución *Zero Trust*. En este bloque no se evalúa la funcionalidad de la aplicación, sino el esfuerzo técnico y económico necesario para desplegar, mantener y operar cada arquitectura. La Tabla 5.8 muestra los resultados obtenidos.

**Tabla 5.8:** Resultados de métricas de complejidad, despliegue y coste

| ID | Métrica | Base | *Zero Trust* |
|---|---|---|---|
| C1 | Tiempo medio de despliegue | 666,33 s / 11,11 min | 690,62 s / 11,51 min |
| | Ejecuciones | 6 | 8 |
| | Tiempo mínimo de despliegue | 628 s | 637 s |
| | Tiempo máximo de despliegue | 688 s | 735 s |
| C2 | Recursos *cloud* desplegados | 46 | 73 |
| C3 | Roles IAM relevantes | 6 | 9 |
| | Políticas asociadas | 8 + 1 *inline* | 11 + 2 *inline* |
| | Grupos de seguridad de red | 4 | 5 |
| | Reglas de entrada | 4 | 5 |
| | Reglas de salida | 4 | 7 |
| C4 | Ficheros de configuración y automatización | 27 | 31 |
| | Líneas totales | 2541 | 3527 |
| | Líneas efectivas | 2127 | 2968 |
| | Líneas Terraform | 658 | 1073 |
| | Líneas Kubernetes | 307 | 441 |
| | Líneas de *scripts* de automatización | 1162 | 1454 |
| C5 | Coste teórico mensual | 177,64 USD | 186,69 USD |
| | Coste teórico diario | 5,92 USD | 6,22 USD |

La Tabla 5.9 recoge la estimación teórica de coste mensual de la solución base calculada mediante AWS Pricing Calculator, mientras que la Tabla 5.10 muestra la estimación teórica de la migración *Zero Trust*.

**Tabla 5.9:** Estimación teórica de coste mensual de la solución base en AWS Pricing Calculator

| Grupo | Servicio | Concepto estimado | Cantidad | Coste mensual |
|---|---|---|---|---|
| Base de datos | Amazon RDS | Instancia PostgreSQL de tipo `db.t4g.micro`. | 730 h | 13,14 USD |

*Continúa en la siguiente página*

**Tabla 5.9:** (continuación)

| **Grupo** | **Servicio** | **Concepto estimado** | **Cantidad** | **Coste mensual** |
|---|---|---|---|---|
| Base de datos | Amazon RDS | Almacenamiento gp3 asociado a la base de datos. | 20 GB-mes | 2,54 USD |
| Cómputo | Amazon EBS | Volumen gp3 asociado a la infraestructura de cómputo. | 20 GB-mes | 1,76 USD |
| Cómputo | Amazon EC2 | Nodo de trabajo `t3.medium` utilizado por el clúster. | 730 h | 33,29 USD |
| Cómputo | Amazon EKS | Plano de control del clúster EKS. | 730 h | 73,00 USD |
| Red | Elastic Load Balancing | Funcionamiento mensual del Application Load Balancer. | 730 h | 18,40 USD |
| Red | Elastic Load Balancing | Consumo de LCU del Application Load Balancer. | 29,2 LCU-h | 0,23 USD |
| Red | Amazon VPC / NAT Gateway | Funcionamiento mensual del NAT Gateway. | 730 h | 35,04 USD |
| Red | Amazon VPC / NAT Gateway | Procesamiento de datos del NAT Gateway regional. | 1 GB | 0,05 USD |
| Red | Amazon VPC / NAT Gateway | Uso horario adicional del NAT Gateway regional. | 1 h | 0,05 USD |
| Almace-namiento | Amazon S3 | Almacenamiento de objetos de la aplicación. | 5 GB-mes | 0,12 USD |

*Continúa en la siguiente página*

**Tabla 5.9:** (continuación)

| **Grupo** | **Servicio** | **Concepto estimado** | **Cantidad** | **Coste mensual** |
|---|---|---|---|---|
| Almacenamiento | Amazon S3 | Solicitudes de tipo Tier 1. | 1000 solicitudes | 0,01 USD |
| Almacenamiento | Amazon S3 | Solicitudes de tipo Tier 2. | 1000 solicitudes | 0,00 USD |
| Almacenamiento | Amazon ECR | Almacenamiento de imágenes Docker. | 0,2 GB-mes | 0,02 USD |
| Observabilidad | Amazon CloudWatch | Almacenamiento mensual de *logs* y métricas. | 0,15 GB-mes | 0,00 USD |
| | | | **Coste mensual estimado total** | **177,64 USD** |

**Tabla 5.10:** Estimación teórica de coste mensual de la solución *Zero Trust* en AWS Pricing Calculator

| **Grupo** | **Servicio** | **Concepto estimado** | **Cantidad** | **Coste mensual** |
|---|---|---|---|---|
| Base de datos | Amazon RDS | Instancia PostgreSQL de tipo `db.t4g.micro`. | 730 h | 13,14 USD |
| Base de datos | Amazon RDS | Almacenamiento gp3 asociado a la base de datos. | 20 GB-mes | 2,54 USD |
| Cómputo | Amazon EBS | Volumen gp3 asociado a la infraestructura de cómputo. | 20 GB-mes | 1,76 USD |
| Cómputo | Amazon EC2 | Nodo de trabajo `t3.medium` utilizado por el clúster. | 730 h | 33,29 USD |
| Cómputo | Amazon EKS | Plano de control del clúster EKS. | 730 h | 73,00 USD |

*Continúa en la siguiente página*

**Tabla 5.10:** (continuación)

| **Grupo** | **Servicio** | **Concepto estimado** | **Cantidad** | **Coste mensual** |
|---|---|---|---|---|
| Administración | Amazon EC2 | Bastión privado de administración `t3.micro`, sin IP pública y operado mediante AWS Systems Manager Session Manager. | 730 h | 8,32 USD |
| Administración | Amazon EBS | Volumen gp3 asociado al bastión privado de administración. | 8 GB-mes | 0,70 USD |
| Red | Elastic Load Balancing | Funcionamiento mensual del Application Load Balancer. | 730 h | 18,40 USD |
| Red | Elastic Load Balancing | Consumo de LCU del Application Load Balancer. | 29,2 LCU-h | 0,23 USD |
| Red | Amazon VPC / NAT Gateway | Funcionamiento mensual del NAT Gateway. | 730 h | 35,04 USD |
| Red | Amazon VPC / NAT Gateway | Procesamiento de datos del NAT Gateway regional. | 1 GB | 0,05 USD |
| Red | Amazon VPC / NAT Gateway | Uso horario adicional del NAT Gateway regional. | 1 h | 0,05 USD |
| Almacenamiento | Amazon S3 | Almacenamiento de objetos de la aplicación. | 5 GB-mes | 0,12 USD |

*Continúa en la siguiente página*

**Tabla 5.10:** (continuación)

| **Grupo** | **Servicio** | **Concepto estimado** | **Cantidad** | **Coste mensual** |
|---|---|---|---|---|
| Almacenamiento | Amazon S3 | Almacenamiento de *logs* de CloudTrail en *bucket* privado con ciclo de vida de 30 días. | 0,5 GB-mes | 0,01 USD |
| Almacenamiento | Amazon S3 | Almacenamiento temporal de artefactos Kubernetes en *bucket* privado con ciclo de vida de 7 días. | 0,1 GB-mes | 0,00 USD |
| Almacenamiento | Amazon S3 | Solicitudes de tipo Tier 1 asociadas a los *buckets* S3 estimados. | 3000 solicitudes | 0,02 USD |
| Almacenamiento | Amazon S3 | Solicitudes de tipo Tier 2 asociadas a los *buckets* S3 estimados. | 3000 solicitudes | 0,00 USD |
| Almacenamiento | Amazon ECR | Almacenamiento de imágenes Docker. | 0,2 GB-mes | 0,02 USD |
| Observabilidad | Amazon CloudWatch | Almacenamiento mensual de *logs* y métricas, incluyendo los registros habilitados del plano de control de EKS. | 0,15 GB-mes | 0,00 USD |
| | | **Coste mensual estimado total** | | **186,69 USD** |

# Capítulo 6

# Análisis y discusión de resultados

## 6.1. Análisis

Una vez realizadas las baterías de pruebas para evaluar los dos entornos, tanto el base como el migrado a *Zero Trust*, el siguiente paso es analizar e interpretar los datos obtenidos. Se realiza un análisis separado de cada categoría evaluada, para a continuación, en la siguiente sección, presentar la discusión global.

### 6.1.1. Funcionalidad

Los resultados de las pruebas funcionales muestran que ambas arquitecturas mantienen correctamente el comportamiento esperado de la aplicación y la migración no introduce cambios de comportamiento. Esto resulta relevante, ya que una de las condiciones necesarias de la migración era reforzar la postura de seguridad de la arquitectura sin modificar de forma significativa la lógica de la aplicación ni comprometer su funcionamiento.

Los tiempos de respuesta obtenidos en las pruebas F1, F2, F3 y F4 son similares entre ambas soluciones. En algunos casos, la solución *Zero Trust* presenta tiempos incluso inferiores a los de la solución base. No obstante, estas diferencias no deben interpretarse como una mejora directa causada por los controles *Zero Trust*, ya que pueden depender de condiciones puntuales del entorno, del estado de los recursos *cloud* o de la variabilidad normal de las peticiones.

La prueba F5 requiere una interpretación particular. En la solución base, la consulta del evento se realiza correctamente, aunque no existe todavía una ejecución *batch* disponible y el sistema devuelve un código HTTP 404 controlado para dicho

estado. En cambio, en la solución *Zero Trust* se valida también la ejecución del servicio *batch* desde el bastión con estado *success*. Por este motivo, el tiempo registrado en F5 para la solución *Zero Trust* es considerablemente superior, ya que no representa únicamente una petición de consulta, sino una operación más completa que incluye la ejecución del proceso *batch*.

En conjunto, los resultados funcionales permiten concluir que la migración hacia la arquitectura *Zero Trust* se ha realizado de forma satisfactoria desde el punto de vista operativo.

### 6.1.2. Seguridad

Los resultados de las pruebas de seguridad muestran diferencias más significativas entre la solución base y la solución *Zero Trust*. Mientras que ambas arquitecturas incorporan controles básicos de autenticación a nivel de API, la migración introduce mejoras claras en la reducción de exposición pública, la segmentación interna, la separación de privilegios y la protección del plano de administración.

En primer lugar, las pruebas S1 y S2 indican que la solución base ya disponía de mecanismos mínimos de control de acceso a nivel de aplicación. La migración a *Zero Trust* no modifica de forma sustancial este comportamiento.

La diferencia principal aparece en la prueba S3, centrada en la exposición externa de componentes internos o auxiliares. En la solución base se detectan varios *endpoints* accesibles públicamente, como `/metrics`, `/admin`, `/docs`, `/openapi.json` y `/api/docs`. Aunque la base de datos RDS y el almacenamiento S3 no son accesibles directamente desde el exterior, la existencia de estos *endpoints* expuestos amplía la superficie de ataque de la aplicación. En la solución *Zero Trust*, por el contrario, no se detectan *endpoints* internos accesibles públicamente, lo que refleja una mejora directa en el principio de exposición mínima.

La prueba S4 permite observar la diferencia entre ambas arquitecturas desde el punto de vista interno. En la solución base, un *pod* interno puede acceder directamente al servicio de salud de la API, a la documentación interna y abrir conexión TCP directa con RDS. Este comportamiento evidencia un nivel elevado de confianza implícita dentro del entorno, ya que un componente interno dispone de capacidad de comunicación con otros servicios sin restricciones específicas. En la solución *Zero Trust*, estos accesos quedan bloqueados, lo que confirma que la segmentación interna introducida reduce la posibilidad de movimiento lateral dentro del clúster y limita las comunicaciones no previstas entre componentes.

En cuanto a la prueba S5, la solución base no presenta políticas de tipo administrador asociadas a los roles analizados, por lo que no se identifica un exceso extremo de privilegios. Sin embargo, el rol de nodos de EKS concentra permisos de varios servicios, incluyendo una política *inline* de acceso a S3. Este diseño implica que los permisos de acceso al almacenamiento quedan asociados al nodo y no al *workload* concreto que los necesita. En la solución *Zero Trust*, los permisos de S3 se separan mediante roles IRSA específicos para la API y el servicio *batch*, aplicando un modelo más cercano al principio de mínimo privilegio y reduciendo la dependencia de credenciales o permisos compartidos a nivel de nodo.

La prueba S6 confirma también una mejora relevante en la segmentación y administración del entorno. La solución base se apoya principalmente en grupos de seguridad y servicios internos de Kubernetes de tipo `ClusterIP`, pero no define `NetworkPolicies`. Además, mantiene habilitado el *endpoint* público de EKS para `0.0.0.0/0`, lo que incrementa la exposición del plano de control. En la solución *Zero Trust*, se incorporan `NetworkPolicies`, se deshabilita el *endpoint* público del clúster y se habilita únicamente el *endpoint* privado, de forma que la administración pasa a realizarse desde el bastión privado mediante AWS Systems Manager.

En conjunto, los resultados de seguridad muestran que la migración consigue reducir varios puntos de confianza implícita presentes en la arquitectura base. La solución *Zero Trust* no solo mantiene los controles de autenticación existentes, sino que mejora la postura de seguridad mediante una menor exposición pública, una separación más precisa de permisos, una segmentación interna más restrictiva y una administración del clúster menos expuesta.

### 6.1.3. Rendimiento

Los resultados de rendimiento muestran que ambas arquitecturas responden correctamente en las tres operaciones evaluadas durante las 20 iteraciones ejecutadas.

En la prueba R1, correspondiente al tiempo de respuesta de un *endpoint* simple, la solución *Zero Trust* obtiene un tiempo medio inferior al de la solución base. La media pasa de 32,47 ms en la solución base a 26,88 ms en la solución *Zero Trust*, y el percentil 95 también disminuye de 33,73 ms a 27,71 ms. Las gráficas asociadas muestran además una distribución bastante estable en ambos casos, con tiempos reducidos y sin variaciones significativas. Este resultado indica que los controles incorporados en la migración no afectan negativamente a una operación básica de consulta del estado de la API.

En la prueba R2, relativa a la autenticación de usuario, la solución *Zero Trust* presenta tiempos algo superiores. El tiempo medio aumenta de 138,56 ms a 156,19 ms, y el percentil 95 pasa de 157,52 ms a 184,65 ms. Además, las gráficas muestran una mayor dispersión en la solución *Zero Trust*, con oscilaciones más marcadas entre iteraciones. No obstante, los valores registrados siguen situándose en rangos reducidos para una operación de autenticación, por lo que no se considera que exista una penalización significativa.

En la prueba R3, correspondiente a la subida de imagen, los tiempos medios son muy similares entre ambas arquitecturas: 186,79 ms en la solución base y 185,39 ms en la solución *Zero Trust*. Sin embargo, la solución *Zero Trust* presenta una dispersión mayor, con un máximo de 346,43 ms y un percentil 95 de 282,63 ms, frente a los 275,76 ms de máximo y 245,03 ms de percentil 95 en la solución base. Esta variabilidad puede deberse a factores propios de la operación, ya que la subida de imagen involucra varios componentes, como la API, el almacenamiento de objetos y la persistencia de metadatos.

A partir de estos resultados, no puede afirmarse que la migración a *Zero Trust* mejore el rendimiento de la aplicación, ya que las diferencias observadas pueden depender de la variabilidad normal del entorno *cloud* y del momento de ejecución de las pruebas. Sin embargo, sí puede concluirse que los controles introducidos no provocan una degradación relevante en las operaciones evaluadas. Ambas arquitecturas mantienen tiempos de respuesta reducidos y comportamiento estable en las pruebas de rendimiento.

### 6.1.4. Complejidad, despliegue y coste

Los resultados de complejidad, despliegue y coste muestran que la migración hacia la solución *Zero Trust* implica un incremento claro del esfuerzo técnico necesario para desplegar y operar la arquitectura.

En primer lugar, el tiempo medio de despliegue aumenta de 666,33 segundos en la solución base a 690,62 segundos en la solución *Zero Trust*. La diferencia es de 24,29 segundos, lo que supone un incremento reducido en términos relativos. Por tanto, el aumento de complejidad no se traduce en una penalización relevante en el tiempo de despliegue.

La diferencia es más visible en el número de recursos *cloud* gestionados. La solución base contiene 46 recursos definidos mediante Terraform, mientras que la solución *Zero Trust* alcanza 73 recursos. Este incremento se debe principalmente a

los elementos añadidos durante la migración, como los roles IAM específicos para *workloads*, el bastión privado, los recursos asociados a la administración mediante Systems Manager, los *buckets* auxiliares y los componentes de auditoría. Desde el punto de vista operativo, esto implica una arquitectura más amplia y con más elementos que mantener, revisar y actualizar.

También aumenta la complejidad de la configuración de seguridad. En la solución base se identifican 6 roles IAM relevantes, 8 políticas asociadas y 1 política *inline*, mientras que en la solución *Zero Trust* se identifican 9 roles, 11 políticas asociadas y 2 políticas *inline*. Este incremento no debe interpretarse únicamente como una desventaja, ya que responde a una separación más precisa de responsabilidades.

En cuanto al coste económico, la solución base tiene un coste teórico estimado de 177,64 USD mensuales, mientras que la solución *Zero Trust* alcanza 186,69 USD mensuales. La diferencia es de 9,05 USD al mes, equivalente aproximadamente a un incremento del 5,09 %. Este aumento se debe principalmente al bastión privado de administración y a los recursos auxiliares añadidos para almacenamiento de *logs*, artefactos y trazabilidad. Sin embargo, la mayor parte del coste mensual continúa estando asociada a componentes que ya existían en la arquitectura base. Por tanto, desde el punto de vista económico, la migración presenta un impacto moderado en el escenario evaluado. La adopción de controles *Zero Trust* no multiplica el coste de la infraestructura.

En conjunto, los resultados muestran que el principal impacto de la migración no se encuentra en el tiempo de despliegue ni en el coste mensual, sino en la complejidad operativa. La solución *Zero Trust* requiere gestionar más recursos, más políticas, más ficheros de configuración y más relaciones entre componentes.

## 6.2. Discusión final

La evaluación realizada permite observar que la migración hacia la arquitectura *Zero Trust* supone una mejora clara de la postura de seguridad de la solución, manteniendo al mismo tiempo la funcionalidad principal de la aplicación y sin introducir una penalización relevante en el rendimiento medido.

Desde el punto de vista funcional, la migración puede considerarse satisfactoria, ya que la aplicación sigue operando correctamente tras la incorporación de los nuevos controles.

El mayor beneficio de la migración se observa en el ámbito de seguridad. La solución base ya disponía de ciertos controles, como autenticación a nivel de API, uso de servicios internos de Kubernetes y aislamiento parcial. Sin embargo, mantenía varios elementos de confianza implícita. La solución *Zero Trust* reduce estos puntos débiles mediante una combinación de controles nativos de AWS y Kubernetes, aplicando una aproximación más alineada con principios como mínimo privilegio, exposición mínima y segmentación.

En rendimiento, los resultados no evidencian un impacto negativo significativo. Algunas operaciones presentan tiempos ligeramente mejores en la solución *Zero Trust*, mientras que otras muestran valores algo superiores o mayor dispersión. No obstante, estas diferencias se mantienen en rangos reducidos y pueden estar condicionadas por la variabilidad normal del entorno *cloud*. Por ello, la conclusión principal respecto al rendimiento es que *Zero Trust* ni mejora ni empeora el rendimiento.

El coste económico también resulta asumible en el escenario analizado. Solo un incremento aproximado del 5,09 % es una diferencia muy reducida teniendo en cuenta las mejoras introducidas en términos de seguridad, trazabilidad, control de acceso y administración privada. Además, la mayor parte del coste total sigue estando asociada a componentes ya presentes en la arquitectura base.

El principal coste de la adopción de *Zero Trust* se encuentra en la complejidad operativa. La arquitectura migrada incorpora más recursos *cloud*, más roles IAM, más políticas, más manifiestos Kubernetes y más dependencias entre componentes. Además, el modelo de administración se vuelve menos directo, ya que el acceso al clúster deja de realizarse desde el exterior y pasa a depender del bastión privado. Esto mejora la seguridad del plano de control, pero también exige mayor disciplina operativa, una gestión más cuidadosa de permisos y una comprensión más profunda de la infraestructura desplegada.

Por tanto, los resultados reflejan el equilibrio habitual entre seguridad y complejidad. La solución *Zero Trust* no es necesariamente más sencilla de operar que la solución base, pero sí ofrece una postura de seguridad más robusta y reduce varios riesgos asociados a la confianza implícita. En este trabajo, dicho aumento de complejidad se considera aceptable, ya que no viene acompañado de un incremento económico elevado ni de una degradación funcional o de rendimiento significativa.

Por tanto, queda claro que la adopción de *Zero Trust* no debe entenderse únicamente como la activación de mecanismos de seguridad aislados, sino como un proceso que requiere diseño, mantenimiento y revisión continua.

## 6.3. Limitaciones de la aproximación realizada

Existen una serie de limitaciones asociadas al alcance del proyecto, al entorno utilizado y a las decisiones adoptadas. Estas son:

- **Entorno experimental acotado.** La evaluación se ha realizado sobre una aplicación *cloud* representativa, pero de tamaño reducido. La arquitectura incluye componentes habituales en aplicaciones desplegadas en la nube, como *frontend*, API, base de datos o Kubernetes, pero no reproduce toda la complejidad de un entorno empresarial real.
- **Carga de trabajo limitada.** Las pruebas funcionales y de rendimiento se han ejecutado con un número controlado de peticiones. Sin embargo, no equivalen a una prueba de carga intensiva ni a una validación de comportamiento bajo alta concurrencia de usuarios.
- **Estimación teórica de costes.** El coste mensual de ambas arquitecturas se ha calculado mediante AWS Pricing Calculator, reproduciendo los servicios y configuraciones principales de cada despliegue. Esta aproximación permite realizar una comparación homogénea entre la solución base y la solución *Zero Trust*, pero no sustituye a una medición prolongada basada en facturación real.
- **Pruebas de seguridad no equivalentes a una auditoría profesional.** Las pruebas de seguridad realizadas permiten identificar diferencias claras entre ambas arquitecturas en términos de exposición, permisos, segmentación y acceso interno. No obstante, no constituyen una auditoría de seguridad completa ni un ejercicio de *pentesting* avanzado.
- **Observabilidad y auditoría básicas.** La solución *Zero Trust* incorpora mecanismos de registro y trazabilidad mediante servicios como CloudTrail y CloudWatch. Sin embargo, no se ha desplegado una plataforma avanzada de correlación, detección y respuesta como un SIEM o SOAR. Por ello, la evaluación se centra en la existencia de evidencias y registros, no en la detección automática de amenazas ni en la respuesta ante incidentes.
- **Dependencia del proveedor *cloud* utilizado.** La implementación se ha realizado sobre AWS, utilizando servicios específicos de la misma. Aunque los principios aplicados son trasladables a otros entornos *cloud*, los detalles técnicos, costes, configuraciones y servicios equivalentes podrían variar en otros proveedores como Azure o Google Cloud Platform.

# Conclusión

Una vez finalizado el desarrollo del presente trabajo, se puede realizar una valoración global del trabajo llevado a cabo y de los resultados obtenidos. En esta última parte se recogen las principales aportaciones realizadas, se plantean posibles líneas de trabajo futuro y se describen algunas de las dificultades encontradas durante el desarrollo del proyecto. Finalmente, se incluye una valoración personal sobre el proceso seguido y el aprendizaje adquirido.

## Aportaciones realizadas

Las principales aportaciones realizadas en este trabajo, orientadas a analizar de forma práctica la adopción del paradigma *Zero Trust* en una arquitectura *cloud*, son las siguientes:

- Diseño e implementación reproducible de una aplicación *cloud*.
- Análisis de los puntos de confianza implícita en una arquitectura *cloud*.
- Metodología de migración de arquitectura *cloud* tradicional a arquitectura *cloud Zero Trust* utilizando controles nativos de AWS y Kubernetes.
- Definición y aplicación de batería de evaluación de arquitecturas base frente a arquitecturas *Zero Trust*.

## Trabajos futuros

A partir del trabajo realizado, se identifican varias líneas de trabajo futuro que permitirían ampliar el alcance de la investigación:

- Estudiar la aplicación del paradigma *Zero Trust* en entornos *cloud* distintos a los grandes proveedores comerciales habituales, como lo son AWS, Azure o

Google Cloud Platform; analizando alternativas basadas en nubes privadas, entornos híbridos o infraestructuras más descentralizadas.

- Ampliar la aproximación hacia escenarios de IoT, robótica o sistemas ciberfísicos, donde la aplicación de *Zero Trust* presenta retos adicionales relacionados con dispositivos de recursos limitados, comunicaciones heterogéneas y necesidad de verificación continua.
- Incorporar mecanismos más avanzados de microsegmentación y comunicación segura entre servicios, como el uso de *service mesh* y mTLS interno entre componentes.
- Desarrollar una evaluación más completa, incluyendo *pentesting* avanzado para la seguridad y pruebas de concurrencia exhaustivas para el rendimiento.
- Integrar capacidades avanzadas de monitorización, correlación de eventos y respuesta automática mediante soluciones SIEM, SOAR o mecanismos equivalentes de detección y respuesta.

# Problemas encontrados

Durante el desarrollo del trabajo se encontraron una serie de dificultades, tanto técnicas como metodológicas, que condicionaron algunas decisiones de diseño e implementación:

- **Búsqueda y selección de información relevante.** Existe una gran cantidad de información sobre *Zero Trust*, pero parte de ella es muy teórica, comercial o orientada a IA y ML. Por ello, fue necesario filtrar la documentación disponible y priorizar referencias que permitieran conectar los principios del modelo con controles técnicos concretos aplicables en un entorno *cloud* basado en AWS y Kubernetes.
- **Delimitación del alcance de *Zero Trust*.** Una de las principales dificultades fue definir hasta qué punto debía llegar la migración. El paradigma *Zero Trust* puede abarcar numerosos controles, tecnologías y niveles de madurez, por lo que fue necesario acotar el trabajo para mantenerlo dentro del alcance razonable de un TFM.
- **Administración del clúster mediante *endpoint* privado y bastión.** Al deshabilitar el *endpoint* público de EKS, la administración del clúster dejó de poder realizarse directamente desde el entorno local. Aunque se ha convertido

en una de las mejoras más potentes ofrecidas por *Zero Trust*, ha obligado a modificar tanto algunas pruebas para conseguir que evalúen de manera correcta, como la manera de administrar la solución en general.

## Opinión personal

Desde mi punto de vista, la realización de este trabajo ha supuesto un reto considerable, tanto por la amplitud del tema tratado como por la cantidad de tecnologías implicadas.

El desarrollo del trabajo también ha requerido más dedicación de la que se había previsto. No obstante, considero que esta dedicación ha sido positiva, ya que ha permitido adquirir una visión mucho más práctica sobre tecnologías *cloud* y sobre los problemas reales que aparecen al trasladar principios de seguridad teóricos a una infraestructura desplegada. Tecnologías como AWS, Kubernetes, Terraform, redes *cloud* y seguridad aplicada están a la orden del día y combinar todas ellas desde cero para aplicar el paradigma *Zero Trust* se ha convertido en una tarea ardua.

Finalmente, me ha permitido comprobar que mejorar la seguridad de una arquitectura no consiste únicamente en añadir controles, sino en analizar de manera más completa sus efectos sobre la operación, el coste, el rendimiento y la mantenibilidad del sistema, mejorando en la organización de las tareas a realizar.

# Lista de referencias


[1] Amazon Web Services: Cedar Policy Language Documentation. Amazon Web Services, Inc., https://docs.cedarpolicy.com/

[2] Amazon Web Services: Zero trust security, https://aws.amazon.com/es/security/zero-trust/

[3] Amazon Web Services: AWS Security Reference Architecture (AWS SRA) (2024), https://docs.aws.amazon.com/prescriptive-guidance/latest/aws-design-principles-security-reference-architecture/welcome.html, documentación técnica oficial

[4] Cao, Y., Pokhrel, S., Zhu, Y., Doss, R., Li, G.: Automation and orchestration of zero trust architecture: Potential solutions and challenges. Machine Intelligence Research **21** (01 2024). https://doi.org/10.1007/s11633-023-1456-2

[5] Centro Criptológico Nacional: Guía CCN-STIC 800: Esquema Nacional de Seguridad. Guía de aplicación (2022), https://www.ccn-cert.cni.es/

[6] Centro Criptológico Nacional: Guía CCN-STIC 801: Responsabilidades y Funciones en el ENS (2022), https://www.ccn-cert.cni.es/

[7] Centro Criptológico Nacional: Guía CCN-STIC 802: Auditoría del Esquema Nacional de Seguridad (2022), https://www.ccn-cert.cni.es/

[8] Cloud Native Computing Foundation: Spiffe overview: Secure production identity framework for everyone (2024), https://spiffe.io/docs/latest/spiffe-about/overview/

[9] Cloud Security Alliance (CSA): Continuous controls monitoring: A reference guide for cloud security. Tech. rep., Cloud Security Alliance (2023), https://cloudsecurityalliance.org/artifacts/continuous-controls-monitoring-a-reference-guide/

[10] Cybersecurity and Infrastructure Security Agency (CISA): Zero Trust Maturity Model Version 2.0. U.S. Department of Homeland Security (April 2023), https://www.cisa.gov/resources-tools/resources/zero-trust-maturity-model

[11] European Central Bank: Euro foreign exchange reference rates: Us dollar (usd). https://www.ecb.europa.eu/stats/policy_and_exchange_rates/euro_reference_exchange_rates/html/eurofxref-graph-usd.en.html (2026), consultado el 22 de junio de 2026

[12] European Union Agency for Cybersecurity (ENISA): Enisa threat landscape for cloud computing (2025), https://www.enisa.europa.eu/publications/enisa-threat-landscape-2025

[13] Gambo, M.L., Almulhem, A.: Zero trust architecture: A systematic literature review. Journal of Network and Systems Management **34** (11 2025). https://doi.org/10.1007/s10922-025-09998-x

[14] Google Cloud: Beyondcorp: Zero trust security model, https://cloud.google.com/beyondcorp

[15] International Organization for Standardization: ISO/IEC 27017:2015. Information technology — Security techniques — Code of practice for information security controls based on ISO/IEC 27002 for cloud services. ISO (2015)

[16] International Organization for Standardization: ISO/IEC 27001:2022. Information security, cybersecurity and privacy protection — Information security management systems — Requirements. ISO (2022)

[17] International Organization for Standardization: ISO/IEC 27002:2022. Information security, cybersecurity and privacy protection — Information security controls. ISO (2022)

[18] Jefatura del Estado: Ley orgánica 3/2018, de 5 de diciembre, de protección de datos personales y garantía de los derechos digitales (2018), https://www.boe.es/eli/es/lo/2018/12/05/3

[19] Microsoft: Modelo de seguridad zero trust, https://learn.microsoft.com/es-es/azure/security/fundamentals/zero-trust

[20] Ministerio de la Presidencia, Relaciones con las Cortes y Memoria Democrática: Real decreto 311/2022, de 3 de mayo, por el que se regula el esquema nacional de seguridad (2022), https://www.boe.es/eli/es/rd/2022/05/03/311

[21] Mushtaq, S., Mohsin, M., Mushtaq, M.: A systematic literature review on the implementation and challenges of zero trust architecture across domains. Sensors **25**, 6118 (10 2025). https://doi.org/10.3390/s25196118

[22] National Institute of Standards and Technology: Zero trust architecture. Special Publication SP 800-207, NIST (2020), `https://doi.org/10.6028/NIST.SP.800-207`

[23] Sarkar, S., Choudhary, G., Shandilya, S.K., Hussain, A., Kim, H.: Security of zero trust networks in cloud computing: A comparative review. Sustainability **14**, 11213 (09 2022). https://doi.org/10.3390/su141811213

[24] Shore, M., Zeadally, S., Keshariya, A.: Zero trust: The what, how, why, and when. Computer **54**, 26–35 (11 2021). https://doi.org/10.1109/MC.2021.3090018

[25] Unión Europea: Reglamento (ue) 2016/679 del parlamento europeo y del consejo, de 27 de abril de 2016, relativo a la protección de las personas físicas en lo que respecta al tratamiento de datos personales (2016), `https://www.boe.es/eli/es/re/2016/04/27/2016/679/con`

[26] Unión Europea: Directiva (ue) 2022/2555 del parlamento europeo y del consejo, de 14 de diciembre de 2022, relativa a medidas destinadas a garantizar un elevado nivel común de ciberseguridad en toda la unión (2022), `https://www.boe.es/doue/2022/333/L00080-00152.pdf`

[27] Verizon: 2024 data breach investigations report (2024), `https://www.verizon.com/business/resources/reports/dbir/`

[28] Wang, R., Li, C., Zhang, K., Tu, B.: Zero-trust based dynamic access control for cloud computing. Cybersecurity **8** (02 2025). https://doi.org/10.1186/s42400-024-00320-x

[29] Ward, B., Beyer, B.: Beyondcorp: A new approach to enterprise security. USENIX ;login: (2014), `https://research.google/pubs/pub43231/`

# Anexo A

# Código fuente del trabajo

La parte práctica perteneciente a este proyecto se ha realizado mediante control de versiones utilizando la herramienta Git, y siendo GitHub la plataforma donde se almacena.

En el repositorio `arierg03/zt-cloud-tfm` se puede encontrar todo el trabajo práctico realizado.

`https://github.com/arierg03/zt-cloud-tfm`

# Anexo B

# Uso de herramientas de inteligencia artificial

Se han utilizado algunas herramientas basadas en inteligencia artificial generativa como apoyo auxiliar en tareas concretas. En particular, se han empleado las siguientes herramientas:

- **Codex**[1]: se ha utilizado como apoyo en tareas relacionadas con revisión de fragmentos de código y depuración de errores. El código y las propuestas de mejoras generadas fueron revisadas, adaptadas y validadas manualmente antes de incorporarse al proyecto.
- **ChatGPT**[2]: se ha utilizado como herramienta de apoyo para la redacción y revisión de fragmentos en LaTeX, mejora gramatical y estilística del documento y ayuda en la búsqueda, identificación y síntesis inicial de artículos relacionados con *Zero Trust*, computación en la nube y arquitecturas de seguridad. El contenido incorporado al documento fue posteriormente revisado, contrastado y ajustado por el autor.
- **Gemini**[3]: se ha utilizado como herramienta complementaria para contrastar explicaciones, obtener segundas propuestas de redacción y apoyar la comprensión de conceptos técnicos relacionados con servicios *cloud*, ciberseguridad y arquitectura *Zero Trust*. Su uso tuvo carácter auxiliar y no sustituyó la revisión crítica del autor.

[1] `https://openai.com/es-ES/codex/`
[2] `https://chatgpt.com/`
[3] `https://gemini.google.com/`

# Anexo C

# Inventario tecnológico final

## Aplicación

***Frontend.*** React `^18.3.1`. React DOM `^18.3.1`. Vite `^5.4.19`. `@vitejs/plugin-react` `^4.4.0`.

***Backend.*** Python `3.12-slim`. FastAPI `0.115.14`. Uvicorn `0.35.0`. SQLAlchemy `2.0.39`. Alembic `1.15.2`. Psycopg `3.2.9`. Boto3 `1.39.0`. Pillow `11.3.0`. Python Multipart `0.0.20`.

**Servicio *batch*.** Python `3.12-slim`. Psycopg `3.2.9`. Boto3 `1.42.91`.

**Base de datos.** PostgreSQL `16`.

- Local: imagen Docker `postgres:16`
- AWS: Amazon RDS PostgreSQL `16`

**Almacenamiento de objetos.** Tipos:

- Local: MinIO `latest` y `minio/mc:latest`
- AWS: *Buckets* S3

**Pruebas.** pytest `8.3.5`. HTTPX `0.28.1`. requests `2.32.5`. matplotlib `3.10.9`.

## Contenerización y orquestación

**Docker.** Versión `29.3.1`. Imágenes utilizadas:

- `python:3.12-slim`

- `node:20-alpine`
- `nginxinc/nginx-unprivileged:1.27-alpine`
- `postgres:16`
- `minio/minio:latest`
- `minio/mc:latest`

**Docker Compose.** Versión `v5.1.1`.

**Kubernetes.** Manifiestos con APIs `apps/v1`, `batch/v1`, `v1` y `networking.k8s.io/v1`.

**Amazon EKS.** Kubernetes `1.35`. Add-ons:

- Amazon VPC CNI `v1.21.1-eksbuild.1`
- kube-proxy `v1.35.3-eksbuild.2`
- CoreDNS `v1.13.2-eksbuild.4`

**kubectl.** Binario de Amazon EKS `1.35.3`.

**Helm.** Helm `v3.15.4`, instalado en el bastión privado.

**eksctl.** Referencia `0.225.0` en etiquetas del OIDC Provider.

**AWS Load Balancer Controller.** Versión `v3.4.0`. Instalado mediante Helm desde el repositorio `eks/aws-load-balancer-controller`.

## Infraestructura *cloud*

**Proveedor.** AWS.

**Región.** `eu-south-2`.

**Terraform.** Versión requerida `>= 1.6.0`. Proveedor AWS `~> 6.0`.

**Red.** VPC `10.0.0.0/16`; subredes públicas en `eu-south-2a` y `eu-south-2b`; subredes privadas en `eu-south-2a` y `eu-south-2b`; Internet Gateway; tablas de rutas públicas y privadas; NAT Gateway opcional; VPC Gateway Endpoint para S3.

**Cómputo.** Amazon EKS con Managed Node Group en subredes privadas. Tipo de instancia de nodos: `t3.medium`. AMI type `AL2023_x86_64_STANDARD`.

**Bastión privado.** Amazon EC2 `t3.micro` con Amazon Linux 2023, sin IP pública, sin reglas de entrada, administrado mediante AWS Systems Manager Session Manager e IMDSv2 obligatorio.

**Base de datos.** Amazon RDS PostgreSQL `16`; clase `db.t4g.micro`; almacenamiento `gp3`; 20 GB; cifrado habilitado; acceso público deshabilitado.

**Registro de imágenes.** Repositorios Amazon ECR con cifrado AES256.

**Balanceo.** Application Load Balancer generado por AWS Load Balancer Controller a partir del Ingress de Kubernetes.

**Buckets S3.** Nombres:

- *Bucket* de imágenes `events-images-296368270177-eu-south-2-an`
- *Bucket* de artefactos Kubernetes `tfm-app-k8s-artifacts-296368270177-eu-south-2`
- *Bucket* de *logs* CloudTrail `tfm-app-cloudtrail-296368270177-eu-south-2`

**IAM.** Roles del clúster EKS, nodos EKS, AWS Load Balancer Controller, API, servicio *batch* y bastión privado. Políticas específicas para acceso a S3, administración de EKS, lectura de artefactos Kubernetes y permisos gestionados de AWS.

**Auditoría y observabilidad.** CloudTrail multi-región con validación de integridad de *logs*. Registros de plano de control de EKS habilitados para `api`, `audit` y `authenticator`, integrados con CloudWatch.