



Universidad Politécnica  
de Madrid



**Escuela Técnica Superior de  
Ingenieros Informáticos**

Grado en Ingeniería Informática

Trabajo Fin de Grado

**Desarrollo de un Chatbot Inteligente  
para la Gestión de Activos IT:  
Integración de IA Generativa con  
Amazon Bedrock en una Arquitectura  
Serverless**

Autor: Diego Cordero Gómez

Tutor(a): Julio García Martín

Madrid, junio 2025

Este Trabajo Fin de Grado se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

*Trabajo Fin de Grado*

*Grado en Ingeniería Informática*

*Título: Desarrollo de un Chatbot Inteligente para la Gestión de Activos IT:  
Integración de IA Generativa con Amazon Bedrock en una Arquitectura  
Serverless*

*Junio 2025*

*Autor: Diego Cordero Gómez*

*Tutor:*

Julio García Martín

Lenguajes y Sistemas Informáticos e Ingeniería del Software (LSIIS)

ETSI Informáticos

Universidad Politécnica de Madrid

## Resumen

El presente Trabajo de Fin de Grado tiene como objetivo el desarrollo de un sistema de gestión de activos tecnológicos (ITAM) que combine una arquitectura cloud basada en servicios de AWS con un subsistema de consulta inteligente controlado mediante un chatbot o agente conversacional basado en inteligencia artificial generativa. La solución ha sido diseñada para pequeñas y medianas empresas que necesitan gestionar de forma centralizada sus recursos informáticos, como hardware, licencias de software y líneas telefónicas, sin disponer de infraestructuras complejas ni conocimientos técnicos avanzados.

El sistema se estructura en dos componentes principales: una API REST para la gestión administrativa de los activos y un agente conversacional AI capaz de interpretar preguntas en lenguaje natural utilizando modelos fundacionales desplegados a través de Amazon Bedrock. Ambos subsistemas comparten una base de datos relacional implementada en Amazon RDS, y se integran mediante funciones AWS Lambda especializadas. El chatbot, ejecutado en local desde consola, permite realizar consultas dinámicas sin necesidad de interfaz gráfica, apoyándose en mecanismos de autenticación mediante AWS CLI y en un sistema controlado de memoria conversacional.

Los resultados obtenidos demuestran que el sistema es funcional, eficiente y fácilmente ampliable. Su diseño modular, el bajo coste de operación y la integración nativa de IA generativa lo convierten en una solución viable para contextos empresariales reales. Además, el trabajo incorpora consideraciones éticas, económicas y sostenibles, alineadas con varios Objetivos de Desarrollo Sostenible (ODS) de la Agenda 2030.

# Abstract

This Final Degree Project presents the development of an IT asset management (ITAM) system that combines a cloud-based architecture using AWS services with an intelligent query subsystem powered by a conversational AI chatbot or agent. The solution is aimed at small and medium-sized enterprises that require centralized control over their technological resources, such as hardware, software licenses, and phone lines, without relying on complex infrastructures or advanced technical knowledge.

The system is divided into two core components: a REST API for administrative asset management and a conversational agent based on artificial intelligence capable of interpreting natural language queries using foundation models deployed via Amazon Bedrock. Both subsystems share a relational database hosted on Amazon RDS and interact through specialized AWS Lambda functions. The chatbot, executed locally via the command line, enables dynamic querying without a graphical interface, relying on AWS CLI authentication and a custom-built memory mechanism to maintain conversational context.

The results confirm that the system is functional, efficient, and easily extensible. Its modular design, low operational cost, and integration of AI capabilities make it a viable solution for real-world business scenarios.

# Tabla de contenidos

<b>1</b>	<b>Introducción</b> .....	<b>1</b>
<b>2</b>	<b>Estado del arte</b> .....	<b>2</b>
2.1	Gestión de activos IT en empresas.....	2
2.1.1	Conceptos clave y alcance de ITAM.....	2
2.1.2	Ciclo de vida de los activos de TI.....	5
2.1.3	Beneficios y limitaciones actuales de la gestión de activos de TI.....	7
2.1.4	Modelos de Gobernanza y Mejores Prácticas .....	11
2.2	Tecnologías utilizadas en la gestión de activos IT .....	16
2.2.1	Principales herramientas comerciales y open source.....	16
2.2.2	Bases de datos de gestión de configuración (CMDB) y su relación con ITAM	17
2.2.3	Tendencias recientes en la gestión de activos IT.....	18
2.3	IA generativa como impulsora del cambio en los entornos IT.....	20
2.3.1	Fundamentos y evolución de la IA generativa.....	20
2.3.2	Aplicaciones de la IA generativa a la gestión de activos IT .....	25
2.3.3	Amazon Bedrock y su papel en el ecosistema cloud .....	27
<b>3</b>	<b>Desarrollo</b> .....	<b>29</b>
3.1	Requisitos del sistema.....	29
3.2	Arquitectura general del sistema .....	30
3.3	Diseño de la base de datos .....	33
3.4	Subsistema de consulta inteligente (Chatbot AI).....	36
3.4.1	Funciones Lambda especializadas por tipo de consulta .....	36
3.4.2	Interpretación dinámica con Bedrock .....	37
3.4.3	Optimización del uso de tokens .....	38
3.4.4	Gestión de la memoria conversacional .....	39
3.4.5	Interfaz de consola y autenticación mediante AWS CLI .....	39
3.5	Subsistema de gestión de recursos (API REST) .....	41
3.5.1	Diseño general del subsistema.....	41
3.5.2	Control de acceso y autenticación.....	42
3.5.3	Interfaz de inicio de sesión y flujo OAuth .....	43
3.5.4	Verificación del acceso autenticado.....	45
<b>4</b>	<b>Resultados y conclusiones</b> .....	<b>46</b>
4.1	Carga de datos .....	46
4.2	Pruebas del subsistema de consulta inteligente (Chatbot AI).....	48
4.2.1	Pruebas individuales de las funciones Lambda .....	48
4.2.2	Prueba integrada del chatbot conversacional .....	52
4.3	Pruebas del subsistema de gestión de recursos (API REST) .....	53
4.4	Conclusiones.....	57

<b>5</b>	<b>Análisis de Impacto .....</b>	<b>58</b>
<b>6</b>	<b>Bibliografía .....</b>	<b>60</b>
<b>7</b>	<b>Anexos.....</b>	<b>65</b>

# Índice de Figuras

Figura 1: Subconjuntos de la Gestión de activos IT [3] .....	3
Figura 2: Clasificación de los principales tipos de activos tecnológicos [4].....	4
Figura 3: Modelo de 5 etapas del ciclo de vida de los activos de IT .....	6
Figura 4: Resumen de los 5 beneficios clave de la Gestión de Activos IT [5].....	8
Figura 5: Desafíos tecnológicos para el futuro de la gestión de activos IT [8].....	9
Figura 6: Limitaciones actuales de los programas de gestión de activos IT en las organizaciones [8] .....	9
Figura 7: Resultados de la encuesta realizada acerca del gasto en auditorías de software en los últimos 3 años [10] .....	10
Figura 8: Ciclo de mejora continua PDCA adoptado por HDI AG para su gestión de licencias [13].....	12
Figura 9: 6 pasos fundamentales para identificar y registrar eficazmente los activos actuales en COBIT BAI09.01 [16] .....	12
Figura 10: Identificación de activos críticos en la organización según COBIT BAI09.02 [17] .....	13
Figura 11: Componentes clave para la gestión del ciclo de vida de los activos en COBIT BAI09.03 [18] .....	13
Figura 12: Componentes esenciales de COBIT BAI09.04 para implementar una gestión controlada de activos [19] .....	14
Figura 13: Elementos clave de la gestión de licencias en COBIT BAI09.05 [20].....	14
Figura 14: Mapa de calor de la contribución de ITAM a la cadena de valor del servicio de ITIL 4 [22].....	15
Figura 15: Tabla comparativa de las principales herramientas ITAM open source [23].	16
Figura 16: Flujo de información entre ITAM, ITSM y la CMDB en la gestión integral de activos y servicios de TI [29] .....	18
Figura 17: Diagrama conceptual del funcionamiento de un modelo de IA generativa [37] .....	21
Figura 18: Arquitectura y uso de los Foundation Models para tareas NLP [40].....	23
Figura 19: Evolución de los principales modelos LLM entre los años 2018 y junio de 2024 [41]. .....	24
Figura 20: Asistente de IA generativa "Torii" respondiendo a una consulta sobre licencias de software [43].....	26
Figura 21: Diagrama general de la arquitectura del sistema (Elaboración propia).....	32
Figura 22: Diagrama entidad-relación de la base de datos (Elaboración propia).....	34
Figura 23: Configuración de la instancia de la base de datos creada en AWS.....	35
Figura 24: Fragmento del código de chatbot_client.py en el que se muestra el prompt utilizado para la interpretación de las consultas. ....	37
Figura 25: Ejemplo de solicitud de la API de Amazon Bedrock para el modelo Claude 3.5 Sonnet v2 [49] .....	38
Figura 26: Resumen de creación de la API con sus rutas e integraciones .....	41
Figura 27: Creación del autorizador JWT en API Gateway.....	42
Figura 28: Asociación del autorizador CognitoAuthorizer a los endpoints.....	43
Figura 29: Configuración del Hosted UI de inicio de sesión de AWS Cognito .....	44
Figura 30: Página de inicio de sesión para la autenticación con Cognito .....	44
Figura 31: Configuración de autorización OAuth en Postman .....	45
Figura 32: Script de carga de datos utilizado para la realización de las pruebas del sistema.....	47
Figura 33: Resultado obtenido tras la ejecución de la Prueba 1 .....	48
Figura 34: Resultado obtenido tras la ejecución de la Prueba 2.....	49
Figura 35: Resultado obtenido tras la ejecución de la Prueba 3.....	51
Figura 36: Flujo de conversación mantenido con el Agente AI para la prueba integral del sistema.....	52

# 1 Introducción

La gestión eficiente de los activos tecnológicos se ha convertido en un componente esencial para el correcto funcionamiento de cualquier organización. En un entorno cada vez más digitalizado, donde los recursos informáticos crecen en complejidad y volumen, disponer de un sistema que permita controlar, optimizar y consultar dichos activos resulta imprescindible. Este Trabajo Fin de Grado se enmarca en ese contexto, con el objetivo de desarrollar un sistema de gestión de activos IT apoyado en tecnologías cloud y en capacidades avanzadas de inteligencia artificial generativa.

El sistema propuesto se basa en una arquitectura serverless utilizando servicios de Amazon Web Services (AWS), como Lambda, API Gateway, Amazon RDS (con MySQL) y Amazon Cognito para la autenticación. Una de las características más destacadas del proyecto es la incorporación de un chatbot inteligente que facilita el acceso a la información mediante lenguaje natural, permitiendo a los usuarios consultar de forma sencilla qué activos tienen asignados, su ubicación o sus especificaciones. Este chatbot se ha construido utilizando Amazon Bedrock, una plataforma que permite integrar modelos fundacionales de IA de última generación como Claude, Llama o Titan, sin necesidad de entrenar modelos propios. Tal y como se expone en el apartado 2.3.3, esta elección responde tanto a la disponibilidad de estos modelos como a su potencial para resolver consultas empresariales mediante lenguaje natural. Según un informe de McKinsey & Company, más del 25 % de las organizaciones ya han adoptado tecnologías de IA generativa en al menos una función empresarial, y su crecimiento continúa en expansión [1].

El desarrollo de esta solución ha partido de una fase de análisis del estado del arte en gestión de activos tecnológicos y en aplicaciones actuales de la inteligencia artificial generativa. A partir de ahí, se ha diseñado una base de datos relacional que centraliza la información sobre hardware, licencias y asignaciones, y se han construido las distintas piezas de la arquitectura en AWS. El enfoque modular y escalable del sistema busca no solo resolver una necesidad específica, sino también servir como ejemplo práctico de cómo integrar tecnologías emergentes con servicios cloud en el ámbito de la gestión IT.

El proyecto combina así dos vertientes complementarias: por un lado, una solución técnica orientada a la automatización y eficiencia de procesos IT, y por otro, una aplicación real de la inteligencia artificial generativa como interfaz conversacional para facilitar el acceso a la información tecnológica dentro de una organización. En las siguientes secciones se detallarán tanto los fundamentos teóricos como las decisiones de diseño adoptadas, junto con un análisis de las tecnologías empleadas y su impacto esperado en el entorno profesional.

## 2 Estado del arte

A medida que las organizaciones dependen cada vez más de la tecnología para gestionar sus operaciones, la correcta administración de los activos de TI se ha convertido en un aspecto crítico dentro de la estrategia empresarial. La evolución de las herramientas y metodologías para la gestión de estos activos ha ido de la mano con el desarrollo de normativas y estándares que buscan optimizar su uso, mejorar la seguridad y garantizar la sostenibilidad de los recursos tecnológicos. En este contexto, es fundamental examinar el estado actual de la gestión de activos IT, su regulación a nivel global y el papel creciente de la inteligencia artificial en este ámbito.

El presente capítulo aborda tres aspectos fundamentales en este campo. En primer lugar, se analizará la gestión de activos IT, incluyendo sus conceptos básicos, el ciclo de vida de los activos y las herramientas más utilizadas en la actualidad para su administración. Posteriormente, se presentará una revisión de las normativas y estándares más relevantes que rigen la gestión de estos activos, con especial énfasis en marcos de referencia ampliamente adoptados como ITIL e ISO 19770. Finalmente, se explorará el reciente avance de la inteligencia artificial y su impacto en la optimización de la gestión de activos IT, detallando sus aplicaciones actuales y las tendencias que definirán su evolución en los próximos años.

### 2.1 Gestión de activos IT en empresas

#### 2.1.1 Conceptos clave y alcance de ITAM

La gestión de activos de Tecnologías de la Información (IT Asset Management o ITAM) es un conjunto de prácticas y procesos que permiten a las organizaciones planificar, controlar y optimizar de manera eficaz y eficiente todos los activos de TI a lo largo de su ciclo de vida. Estos activos incluyen tanto hardware como software, y su gestión adecuada es esencial para maximizar el valor, controlar costos, gestionar riesgos y cumplir con requisitos regulatorios y contractuales [2].

Tal como define ITIL 4, el propósito principal de ITAM es planificar y gestionar el ciclo de vida completo de todos los activos de TI, con el fin de ayudar a la organización a:

- Maximizar el valor.
- Controlar costos.
- Gestionar riesgos.
- Apoyar la toma de decisiones sobre la compra, reutilización, retiro y disposición de activos de TI.
- Cumplir con requisitos regulatorios y contractuales.

Como se observa en la **figura 1**, ITAM se estructura en torno a varios subconjuntos especializados, siendo los más relevantes HAM (Hardware Asset Management) y SAM (Software Asset Management):

## What is ITAM?

- IT Asset Management
- Software Asset Management
- Hardware Asset Management
- Software License Management
- Software License Compliance
- IT Service Management
- Config. Management Database

© Anglepoint Group

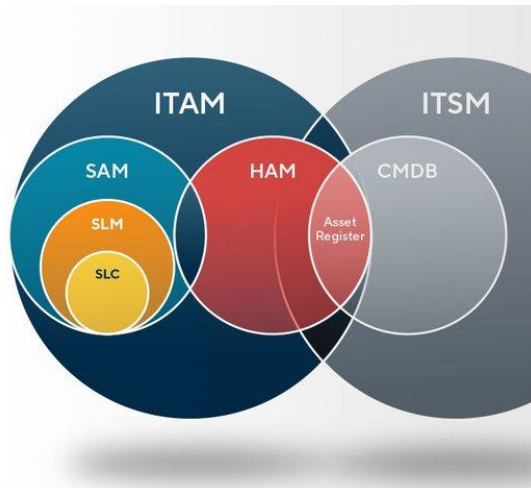


Figura 1: Subconjuntos de la Gestión de activos IT [3]

- **Gestión de Activos de Hardware (HAM):** Se centra en la administración de los componentes físicos de TI, como servidores, computadoras, dispositivos de red y otros equipos. Su objetivo es garantizar que el hardware se utilice de manera eficiente, se mantenga adecuadamente y se reemplace o elimine conforme a políticas establecidas [3].
- **Gestión de Activos de Software (SAM):** Está enfocada en el control y administración de las licencias de software y su uso dentro de la organización. SAM busca asegurar el cumplimiento de los acuerdos de licencia, optimizar el uso de software y reducir riesgos asociados a incumplimientos o uso ineficiente [3].

Además de estos subconjuntos, es importante considerar que ITAM se encuentra estrechamente relacionado con la **Gestión de Servicios de TI (ITSM)**, la cual se encarga de planificar y controlar la entrega de servicios tecnológicos al negocio. En este contexto, la **CMDB (Configuration Management Database)** actúa como un punto de conexión entre ITAM e ITSM, permitiendo alinear la gestión de activos con los procesos operativos de soporte y resolución. Más adelante se detallará su relación con ITAM.

Por otro lado, la norma **ISO/IEC 19770-1:2017**, en la que más adelante profundizaremos, proporciona una estructura estandarizada para implementar un sistema de gestión de activos IT (ITAMS). Este estándar establece una clasificación detallada de los distintos tipos de activos tecnológicos, mostrada en la **figura 2**.

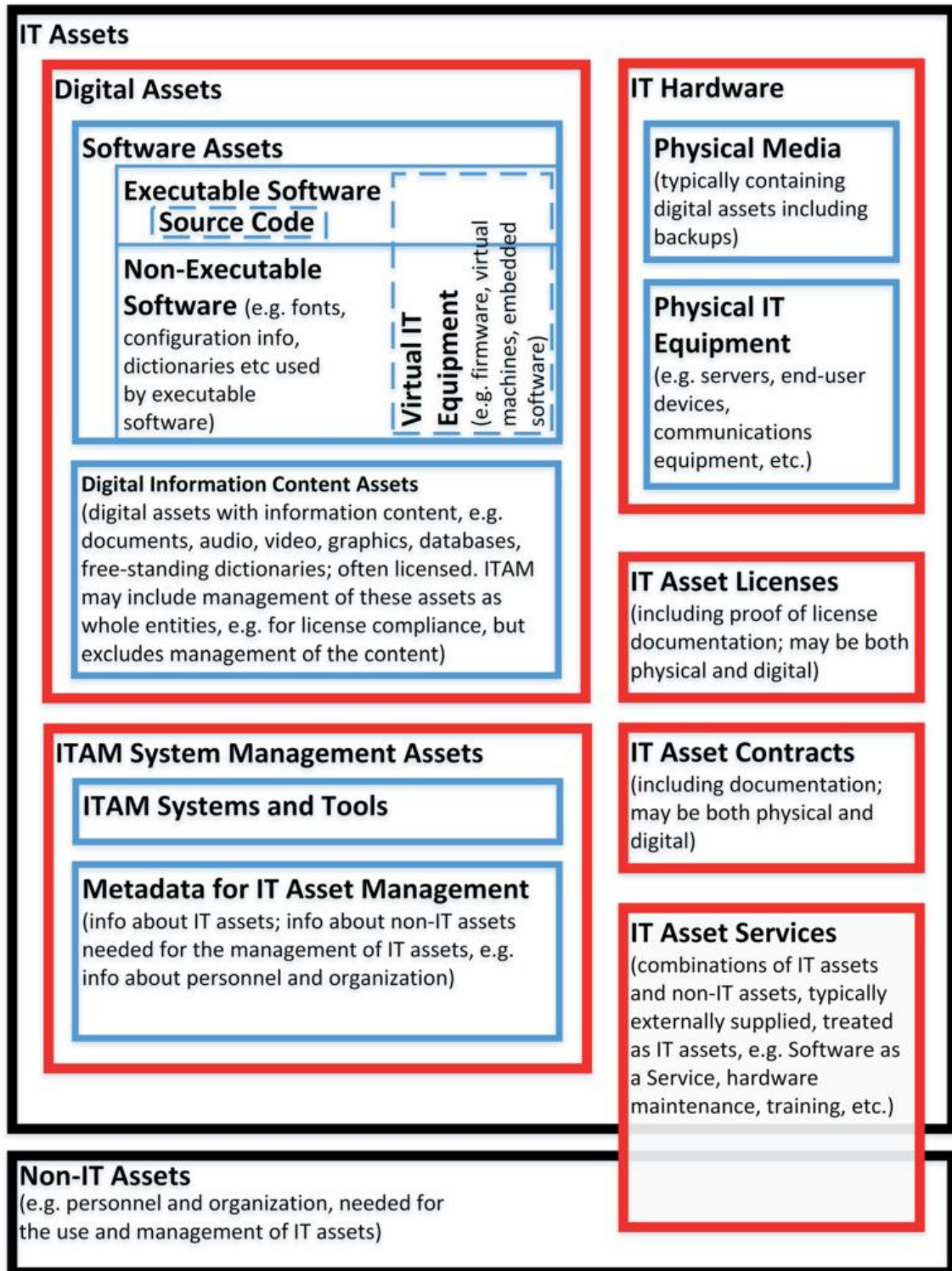


Figura 2: Clasificación de los principales tipos de activos tecnológicos [4]

Como se puede observar, se dividen los activos de TI en diferentes bloques:

- **Activos digitales:** incluyen tanto software ejecutable (como aplicaciones), software no ejecutable (p. ej. fuentes, configuraciones), equipamiento virtual (como máquinas virtuales, firmware...) y contenido digital como documentos, audio, vídeo o bases de datos [4].

- **Hardware IT:** medios físicos (discos, respaldos) y equipamiento como servidores, portátiles o routers [4].
- **Licencias y contratos:** documentación física o digital que respalda el uso legal del software y servicios adquiridos [4].
- **Servicios de activos IT:** como SaaS, mantenimiento externo o formaciones asociadas [4].
- **Activos de gestión ITAM:** herramientas utilizadas para la gestión de activos y metadatos que facilitan su administración (como información organizativa, usuarios responsables, etc.) [4].

Estos elementos permiten a la organización tener una visión integral y bien categorizada del conjunto de recursos IT, facilitando su control y alineación con los objetivos estratégicos.

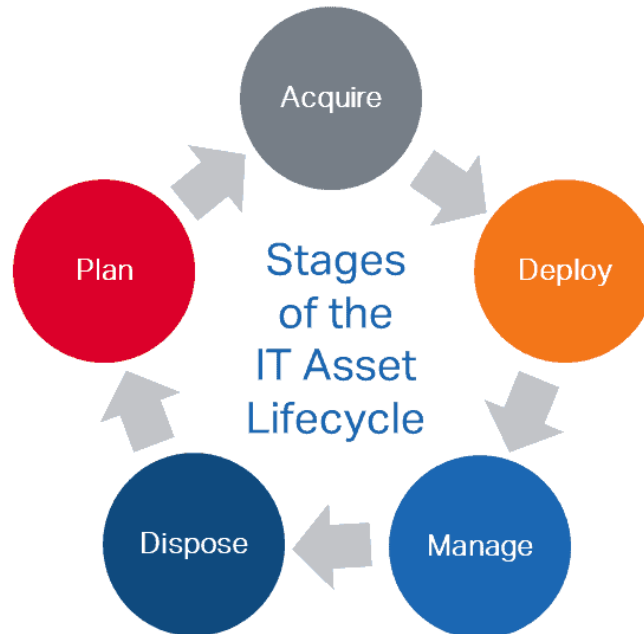
### 2.1.2 Ciclo de vida de los activos de TI

Un pilar de ITAM es la gestión integral del ciclo de vida de cada activo de TI, el cual es un proceso circular que abarca desde su planificación inicial hasta su disposición final. Aunque las etapas del ciclo de vida varían de una organización a otra y no existe un consenso definido, generalmente se compone de 5 o más fases que podemos encontrar en marcos de procesos como el COBIT 2019 (BAI09.03, Manage The Asset Life Cycle) o ITIL v4 :

- **Planificación:** identificación de necesidades tecnológicas, evaluación de opciones y presupuestación del activo antes de su adquisición. Se definen los requerimientos funcionales y se alinean con la estrategia del negocio y estándares vigentes (p. ej., compatibilidad, seguridad).
- **Adquisición:** proceso de compra o contratación del activo. Incluye selección de proveedores, negociaciones de contrato/licencias y alta en el inventario. Es crítico en esta fase asegurar que el activo cumpla especificaciones y condiciones acordadas (SLAs, garantías).
- **Implementación:** incorporación del activo al entorno de la organización. Implica su instalación física o despliegue lógico, configuración inicial y pruebas de funcionamiento. También abarca la capacitación de usuarios o administradores para usar el nuevo recurso. Al final de esta etapa el activo queda operativo y registrado (por ejemplo, asignado a un usuario o ubicación).
- **Mantenimiento:** periodo más extenso, en el que el activo es utilizado para las operaciones de la empresa. Se realizan actividades de monitoreo de su desempeño y uso (por ejemplo, seguimiento de costos de soporte, uptime, consumo de licencias), además de mantenimiento preventivo y correctivo para prolongar su vida útil. Aquí es esencial gestionar cambios (actualizaciones de software, reparaciones, parches de seguridad) y mantener los datos de inventario actualizados.
- **Retiro o Disposición:** fase final en la que el activo es dado de baja. Comprende procedimientos seguros de disposición: borrado de datos confidenciales, cancelación de licencias o contratos asociados, y reciclaje, venta o eliminación física del equipo según normativas ambientales.

También se actualizan los registros para reflejar la baja y se analizan costes residuales o lecciones aprendidas.

A continuación, en la **Figura 3** se ilustra un ciclo de vida típico de un activo de TI, desde la planificación inicial hasta su retiro:



*Figura 3: Modelo de 5 etapas del ciclo de vida de los activos de IT*

Una gestión rigurosa en cada etapa asegura que la organización obtiene el máximo provecho del activo y evita problemas como sobrecostes por compras innecesarias, brechas de seguridad en activos sin soporte, o pérdidas de datos en disposiciones inadecuadas.

### **2.1.3 Beneficios y limitaciones actuales de la gestión de activos de TI**

La gestión de activos IT ofrece beneficios estratégicos y operacionales que van mucho más allá del simple control del inventario tecnológico. A continuación se presentan los cinco **beneficios** clave para el negocio identificados por el ITAM Forum:

#### **Ahorro económico**

Se estima que en torno a un 30-40% del software en las organizaciones se encuentra infrautilizado o sin uso, lo que representa una oportunidad clara de ahorro al redistribuir licencias y reutilizar hardware existente antes de realizar nuevas compras. Un caso destacado es el de la cadena estadounidense CBS, que logró reducir en un 32% su gasto anual en licencias de Zoom tras implementar una estrategia de recolección y reasignación de licencias no utilizadas [5].

#### **Reducción de riesgos**

Al adquirir software, las organizaciones no compran el producto en sí, sino el derecho a usarlo bajo ciertas condiciones. El incumplimiento de estas condiciones puede suponer importantes sanciones legales, regulatorias o reputacionales. ITAM permite mitigar estos riesgos mediante una correcta gestión de licencias, auditorías internas y una mayor transparencia en el uso del software. Danske Bank es un claro ejemplo de esto. En menos de dos años, su equipo de ITAM transformó su enfoque de gestión de activos hacia uno estratégico alineado con la normativa del sector bancario, mejorando tanto la seguridad como el cumplimiento regulatorio [5].

#### **Eficiencia operativa**

Una correcta gestión de activos reduce redundancias, mejora la toma de decisiones y permite una planificación más ágil. Esto se traduce en una mayor productividad y un uso más racional de los recursos tecnológicos. El caso de CBS mencionado anteriormente también demuestra este beneficio: su equipo ITAM, además de ahorrar costes, logró eliminar herramientas con funcionalidades duplicadas, consolidar el stack tecnológico y asegurar que cada activo cumpliera con una función clara en el negocio [5]. Por otro lado, ITAM también facilita la automatización de procesos como el aprovisionamiento y retirada de equipos, lo que reduce la carga operativa sobre los equipos de TI y mejora los tiempos de respuesta a las necesidades de los usuarios [6].

#### **Mejora de la seguridad de la información**

Saber qué activos se tienen, dónde están y en qué estado se encuentran es esencial para protegerlos. ITAM colabora con los equipos de ciberseguridad para desactivar software fuera de soporte, prevenir brechas y gestionar vulnerabilidades. Como afirma Ron Brill, vicepresidente del ITAM Forum, sin ITAM no es posible alcanzar objetivos críticos como la reducción de costes o la seguridad de la información. También se garantiza una correcta retirada o reasignación de activos, evitando amenazas derivadas de dispositivos obsoletos o no autorizados [5].

## Sostenibilidad

La reutilización de equipos, la compra consciente y la gestión responsable del fin de vida de los activos permiten reducir el impacto ambiental. Se estima que fabricar un portátil nuevo genera hasta 316 kg de CO<sub>2</sub> y consume más de 190.000 litros de agua, el equivalente al gasto anual de una familia de 4 personas, por lo que prolongar su uso a través de ITAM reduce significativamente esta huella. Además, ITAM facilita la elección de proveedores sostenibles y prácticas de donación o reciclaje de dispositivos [5]. Según el Governance & Accountability Institute, el 90% de las empresas del S&P 500 ya publican informes de sostenibilidad, lo que refuerza la importancia de integrar prácticas de ITAM dentro de los objetivos ESG (*Environmental, Social & Governance*) de las compañías [7].

En la siguiente figura se muestra un cuadro resumen de los 5 beneficios clave de la gestión de activos TI:



Figura 4: Resumen de los 5 beneficios clave de la Gestión de Activos IT [5]

A pesar de los numerosos beneficios que ofrece la gestión de activos tecnológicos, su implementación y evolución presentan importantes **desafíos** que muchas organizaciones aún no han conseguido resolver del todo. Estos retos afectan tanto a la estrategia como a la operativa diaria de la gestión de activos. A continuación, se presentan los más relevantes:

### Complejidad tecnológica creciente

Uno de los principales retos para los programas de ITAM radica en la creciente sofisticación y diversidad del entorno tecnológico. Como se observa en la **figura 5**, las plataformas cloud (44%), la complejidad en las licencias de procesadores

(38%) y el fenómeno del “Shadow IT” (32%) encabezan la lista de tecnologías que más obstáculos presentan para la gestión eficiente de activos [8].



Figura 5: Desafíos tecnológicos para el futuro de la gestión de activos IT [8]

Otros factores como la adopción de dispositivos móviles, IoT, virtualización de escritorios o tecnologías como el edge computing y los contenedores también dificultan el seguimiento del ciclo de vida completo de los activos. Esta diversidad introduce numerosos puntos ciegos y complica la integración de datos entre sistemas [8].

Las herramientas tradicionales de ITAM no siempre están preparadas para gestionar entornos tan dinámicos, lo que provoca una pérdida de visibilidad sobre parte de los activos, especialmente aquellos que no son adquiridos por la empresa (BYOD, Bring Your Own Device) o que se encuentran en entornos distribuidos como sucede en el teletrabajo [9].

### Barreras organizativas y de proceso

Además de los desafíos técnicos, las organizaciones también deben enfrentarse a limitaciones estructurales internas. En la **figura 6**, extraída del informe de Deloitte de 2021, se destacan barreras como la falta de procesos consistentes (73%), carencia de personal con las competencias necesarias (68%) y la ausencia de herramientas adaptadas a los nuevos requerimientos (67%) [8].



Figura 6: Limitaciones actuales de los programas de gestión de activos IT en las organizaciones [8]

También es especialmente relevante la dificultad para obtener datos precisos y completos sobre hardware, despliegues de software y derechos de uso (64%), así como la falta de apoyo explícito del liderazgo ejecutivo (63%). Estos factores reducen la capacidad del equipo de ITAM para influir en la toma de decisiones estratégicas y dificultan el desarrollo de una cultura de gobernanza tecnológica transversal [8].

### Visibilidad, auditorías y sostenibilidad económica

En el informe *State of ITAM 2023* publicado por Flexera se confirma esta problemática, donde más del 50% de los profesionales de ITAM declaran no tener visibilidad total de las inversiones tecnológicas de su empresa, lo que repercute directamente en la capacidad para optimizar costes y controlar riesgos [10].

Además, esta falta de control repercute en auditorías cada vez más costosas. Según el mismo estudio, el 32% de las empresas ha gastado más de 5 millones de dólares en auditorías de software en los últimos tres años, y un 22% ha superado los 10 millones [10]. En la siguiente figura se muestran los resultados de la encuesta realizada por flexera:

How much do you estimate your organization has paid in software vendor audits over the past 3 years?

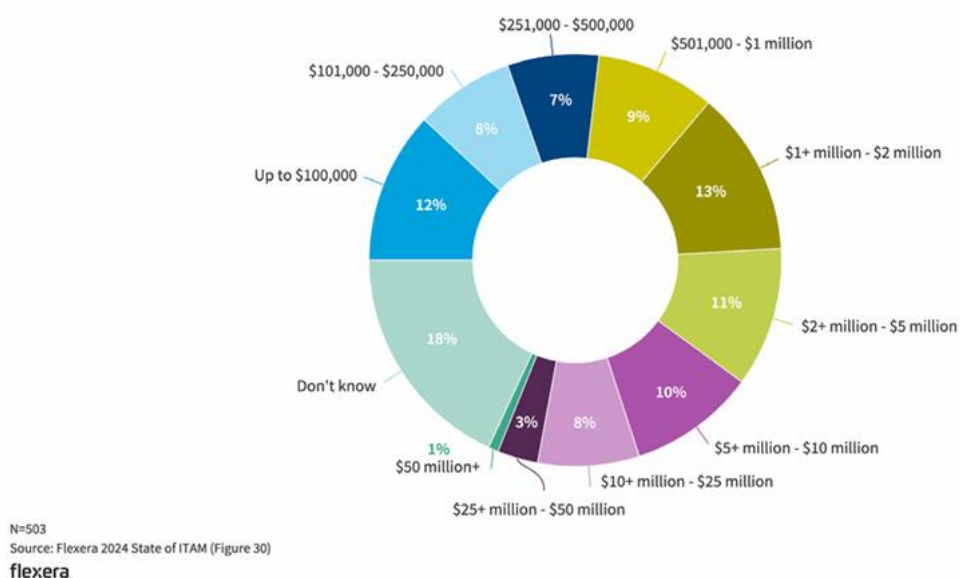


Figura 7: Resultados de la encuesta realizada acerca del gasto en auditorías de software en los últimos 3 años [10]

### Desafíos derivados del trabajo remoto y nuevas tecnologías

La generalización del trabajo remoto ha hecho que muchas organizaciones pierdan capacidad de rastrear y controlar los activos utilizados por los empleados fuera del perímetro tradicional, lo que introduce nuevos riesgos de seguridad y cumplimiento normativo [11] [9].

Además, tecnologías emergentes como dispositivos IoT, gafas de realidad aumentada o sensores inteligentes requieren enfoques adaptativos en la estrategia de ITAM para garantizar su correcta integración, seguimiento y seguridad [9].

## **Cultura del cambio y restricciones presupuestarias**

Según Brightfin, muchos programas de ITAM fracasan por falta de compromiso interno, resistencia al cambio y visión a corto plazo. La implementación de soluciones efectivas requiere inversión en herramientas y formación especializada, algo que no siempre está alineado con los presupuestos asignados o con la prioridad estratégica que recibe el ITAM dentro de la organización [11].

### **2.1.4 Modelos de Gobernanza y Mejores Prácticas**

La gestión eficaz de los activos de Tecnologías de la Información (TI) es esencial para optimizar recursos, controlar costos y garantizar el cumplimiento normativo. Diversos marcos y estándares internacionales ofrecen directrices y mejores prácticas para lograr una gestión eficiente y alineada con los objetivos organizacionales. A continuación, se presentan algunos de los más relevantes:

#### **ISO/IEC 19770-1:2017**

Este estándar internacional establece los requisitos para implementar un sistema de gestión de activos de TI (ITAM) eficaz, auditable y alineado con otras normas de gestión reconocidas como ISO 9001 o ISO/IEC 27001. Su enfoque se basa en procesos estructurados, responsabilidades definidas y mejora continua, lo que permite garantizar el uso responsable y controlado de los activos tecnológicos [4].

Su principal ventaja es que permite a las organizaciones alinear sus procesos ITAM con estándares de gobernanza reconocidos, lo cual es especialmente útil en entornos regulados o con necesidades de certificación [12].

#### **Aspectos clave del ISO/IEC 19770-1:2017:**

- Establece un sistema de gestión de activos de TI (ITAMS) basado en procesos repetibles, roles definidos y control documentado [12] [4].
- Incorpora el ciclo de mejora continua Plan-Do-Check-Act (PDCA), facilitando auditorías y mejoras progresivas [13].
- Permite gestionar no solo hardware, sino también software, servicios, licencias y activos virtuales [4].
- Enfatiza la trazabilidad y el registro completo de cada activo, desde su adquisición hasta su retiro [12].

En la siguiente figura se presenta un ejemplo de diagrama PDCA que fue utilizado por la empresa alemana HDI AG para la gestión de licencias:

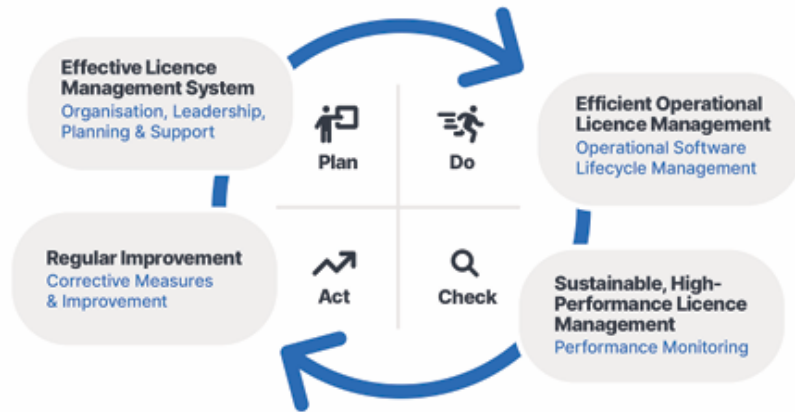


Figura 8: Ciclo de mejora continua PDCA adoptado por HDI AG para su gestión de licencias [13]

## COBIT 2019 – BAI09

COBIT 2019 es un marco de referencia para la gobernanza y gestión de TI desarrollado por ISACA. Dentro del dominio “BAI” (Build, Acquire and Implement) se encuentra el objetivo **BAI09: Manage Assets (Gestionar los activos)**, que establece una serie de buenas prácticas para la gestión de activos IT, asegurando que los recursos tecnológicos se utilicen de manera óptima y segura [14] [15].

Las 5 buenas prácticas del proceso BAI09 son las siguientes:

1. **Identificar y registrar los activos:** Consiste en descubrir y mantener un inventario completo de todos los activos tecnológicos. Este inventario debe ser preciso, actualizado y alineado con las necesidades del negocio. Su correcta gestión permite reducir riesgos y optimizar decisiones sobre los activos [16].

A continuación, en la siguiente figura se muestran los 6 pasos fundamentales para identificar y registrar eficazmente los activos:

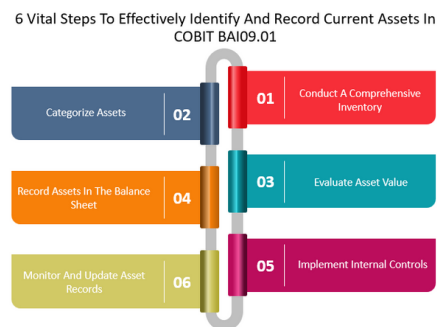


Figura 9: 6 pasos fundamentales para identificar y registrar eficazmente los activos actuales en COBIT BAI09.01 [16]

2. **Gestionar los activos críticos:** Una vez registrados, es necesario clasificar los activos según su importancia estratégica, impacto en el negocio y nivel de riesgo asociado. Esta práctica permite asignar prioridades y recursos de forma inteligente [17].

En la siguiente figura se muestran los criterios clave para identificar activos críticos:

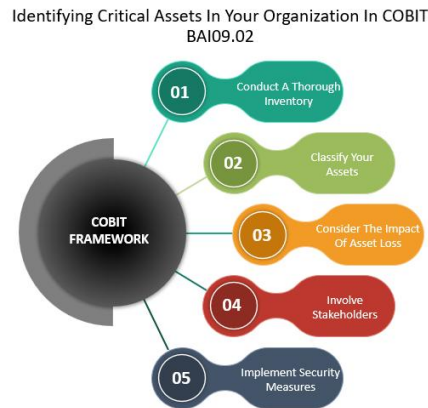


Figura 10: Identificación de activos críticos en la organización según COBIT BAI09.02 [17]

3. **Gestionar el ciclo de vida de los activos:** Esta práctica promueve una visión integral de cada activo a lo largo de su vida útil: desde la planificación y adquisición, pasando por la operación y mantenimiento, hasta su disposición final [18].

La figura 11 resume los 7 componentes clave de la gestión del ciclo de vida de TI según el framework COBIT 2019:

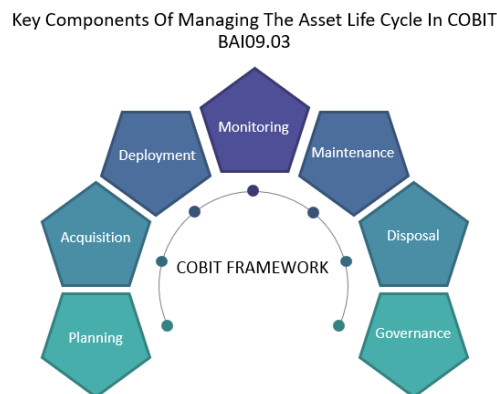


Figura 11: Componentes clave para la gestión del ciclo de vida de los activos en COBIT BAI09.03 [18]

4. **Optimizar el valor de los activos:** Implica revisar periódicamente el conjunto de activos para detectar redundancias, infrautilización o necesidades de renovación, con el fin de maximizar su valor para el negocio [19].

La siguiente figura muestra los componentes principales para implementar esta práctica:

Vital Components Of COBIT BAI09.04 For Implementing Managed Assets

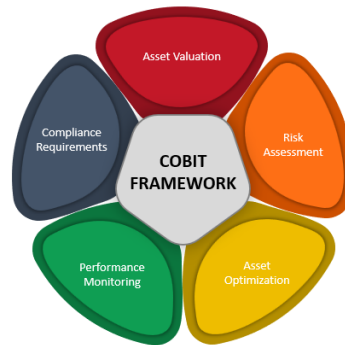


Figura 12: Componentes esenciales de COBIT BAI09.04 para implementar una gestión controlada de activos [19]

5. **Gestionar licencias y contratos:** Se centra en administrar de manera proactiva las licencias de software y los contratos, evitando tanto incumplimientos que puedan generar multas, como el desperdicio económico por licencias infrautilizadas [20].

La siguiente figura resume los elementos clave en la gestión de licencias, como el seguimiento de licencias y la gestión de contratos:

Key Elements Of License Management Under COBIT BAI 09.05

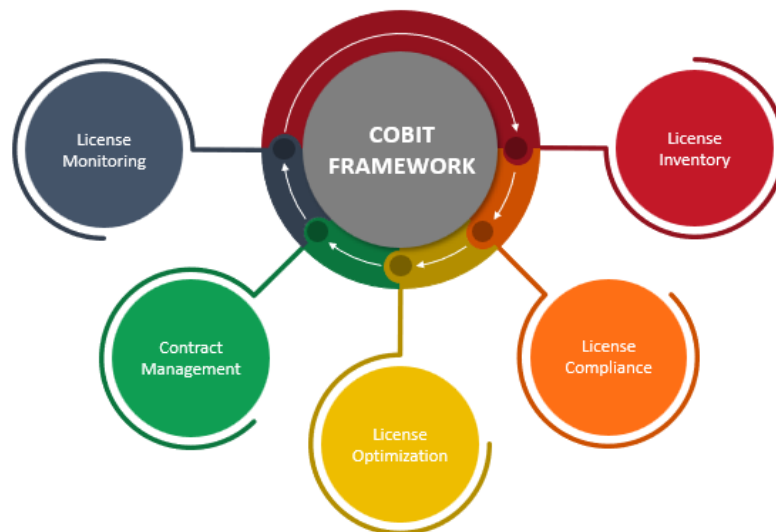


Figura 13: Elementos clave de la gestión de licencias en COBIT BAI09.05 [20]

## ITIL 4 y la Gestión de Activos de TI

A diferencia de ISO o COBIT 2019, **ITIL 4** no es un estándar rígido, sino un marco de buenas prácticas diseñado para ayudar a las organizaciones a ofrecer valor a través de servicios de TI eficientes. Uno de los pilares fundamentales de ITIL 4 es la **cadena de valor del servicio (Service Value Chain)**, que estructura todas las actividades que transforman la demanda en valor mediante la entrega y mejora de servicios [2] [21].

En este contexto, la **gestión de activos de TI** es una de las 34 prácticas de ITIL 4 y tiene un rol específico dentro de esta cadena. Su objetivo es planificar y gestionar el ciclo de vida de los activos para maximizar su valor y minimizar riesgos, costes o ineficiencias [21].

En la siguiente figura se muestra un mapa de calor que representa el grado de contribución de la gestión de activos IT a cada una de las actividades de la cadena de valor del servicio de ITIL 4, destacando en color más intenso aquellas áreas donde aporta mayor impacto:

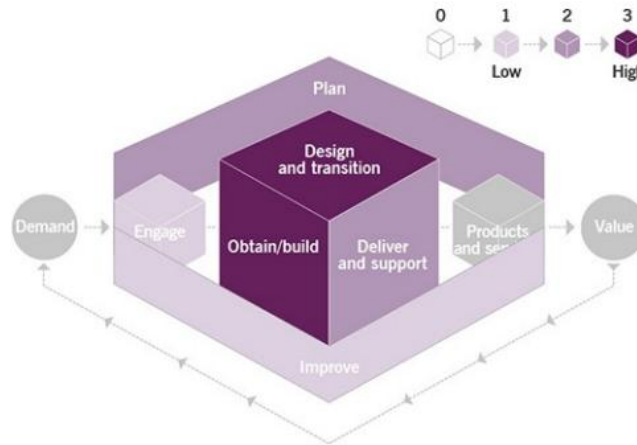


Figura 14: Mapa de calor de la contribución de ITAM a la cadena de valor del servicio de ITIL 4 [22]

Algunas mejores prácticas de ITIL 4 en la gestión de activos son:

- **Inventario detallado:** Mantener un registro preciso de todos los activos, incluyendo detalles como ubicación, estado y propietario.
- **Integración con otros procesos de ITSM:** Alinear la gestión de activos con procesos como gestión de incidentes y cambios para una respuesta más efectiva.
- **Monitoreo continuo:** Implementar herramientas que permitan el seguimiento en tiempo real del uso y rendimiento de los activos.
- **Gestión de licencias:** Asegurar que el software utilizado esté debidamente licenciado y en conformidad con los acuerdos establecidos.
- **Evaluación y mejora continua:** Revisar periódicamente las prácticas de gestión de activos para identificar áreas de mejora y adaptarse a cambios en el entorno tecnológico [22].

Estos marcos no son excluyentes sino complementarios. Las organizaciones suelen inspirarse de varios: por ejemplo, **ITIL 4** para implementar los procesos operativos diarios de gestión de activos e integrarlos con la mesa de servicio; **ISO 19770-1** para establecer un sistema formal auditable y mejorar la disciplina en documentación y métricas; y **COBIT** para asegurar que la alta dirección supervise y reciba valor de ITAM alineado con la estrategia. La **Tabla 2** presenta una comparación simplificada de estos marcos y estándares, destacando su enfoque principal en relación con ITAM.

## 2.2 Tecnologías utilizadas en la gestión de activos IT

### 2.2.1 Principales herramientas comerciales y open source

En la actualidad, existen múltiples soluciones diseñadas para facilitar la gestión de activos tecnológicos (ITAM) y el control de software (SAM). Estas herramientas suelen incorporar funcionalidades clave como:

- Inventario automatizado de hardware y software.
- Gestión de licencias y contratos.
- Asignación de activos a usuarios o unidades organizativas.
- Integración con directorios corporativos (como Active Directory).
- Auditorías, alertas y reportes de cumplimiento.
- Integración con CMDB y herramientas ITSM (como ServiceNow o Jira) [23].

Se pueden clasificar en dos grandes categorías:

- **Soluciones comerciales:**

Herramientas robustas como *ServiceNow ITAM*, *Ivanti Neurons for ITAM*, *ManageEngine AssetExplorer* o *Flexera One* ofrecen una amplia integración con entornos corporativos complejos. Son especialmente recomendables para grandes organizaciones con necesidades de gobernanza avanzadas [24] [25].

- **Soluciones open source o de menor coste:**

Alternativas como *Snipe-IT*, *GLPI* o *OCS Inventory NG* proporcionan funcionalidades suficientes para medianas empresas. Aunque suelen requerir más personalización y conocimientos técnicos, ofrecen mayor control sobre la infraestructura [23].

En la siguiente figura se muestra una tabla comparativa de las principales herramientas open source:

	Asset Tracking	Maintenance Management	Configuration Management	Compliance Tracking	Contract/License Management	Requisition Management	Procurement Management
Snipe-IT	✓	✓	✓	✓	✓	✓	✓
SysAid	✓	✗	✓	✓	✓	✓	✓
GLPI	✓	✓	✓	✗	✓	✗	✗
Spiceworks	✓	✓	✓	✓	✓	✓	✗
ERPNext	✓	✓	✗	✓	✓	✓	✗
Open-Audit	✓	✓	✓	✓	✓	✓	✗
CMDBuild	✓	✗	✓	✓	✗	✗	✓

Figura 15: Tabla comparativa de las principales herramientas ITAM open source [23]

### 2.2.2 Bases de datos de gestión de configuración (CMDB) y su relación con ITAM

Un concepto clave en la gestión de activos tecnológicos es la distinción entre las bases de datos de configuración (CMDB) y los sistemas de gestión de activos (ITAM). Aunque ambos almacenan información sobre componentes de TI, su enfoque y utilidad son diferentes.

Una **CMDB (Configuration Management Database)** es una base de datos que registra elementos de configuración (ECs) y sus relaciones, siguiendo las buenas prácticas de ITIL. Su propósito es reflejar cómo los componentes tecnológicos interactúan para ofrecer servicios, lo cual resulta esencial para entender el impacto de cambios o incidencias en el entorno de TI [26].

Por otro lado, la **gestión de activos IT (ITAM)** se centra en el ciclo de vida, valor y estado de cada activo: desde su adquisición hasta su retirada. Incluye información sobre su coste, propietario, contratos y ubicación. Es decir, aporta una perspectiva financiera y administrativa del entorno tecnológico [26] [27].

En la práctica, hay un solapamiento inevitable entre ambos enfoques. Por ejemplo, un servidor puede figurar en ITAM con su fecha de compra, valor contable y responsable, mientras que en la CMDB aparece como un EC con sus atributos técnicos y relaciones con otras aplicaciones o infraestructuras [26] [27]. La diferencia fundamental se encuentra en el objetivo:

- La **CMDB** enfatiza las relaciones entre componentes y la configuración técnica detallada (versiones, parches, dependencias), siendo clave para la gestión de cambios, problemas e incidencias [26].
- El **ITAM** ofrece una visión centrada en la propiedad, el coste y el seguimiento administrativo del activo, facilitando la planificación financiera, el cumplimiento normativo y las auditorías [27].

Muchas herramientas modernas, como ServiceNow, integran ambos enfoques en un único sistema donde cada EC incluye información tanto técnica como financiera [28]. Otras organizaciones prefieren mantener sistemas separados pero sincronizados, de modo que los cambios en uno se reflejen en el otro [26]. Una buena práctica es vincular ambas fuentes, por ejemplo, cuando un activo se da de baja en ITAM, su estado en la CMDB también debe actualizarse. Del mismo modo, si se detecta un cambio técnico en la CMDB (como una ampliación de memoria), ese cambio debería reflejarse en el sistema de activos [27].

Como ya se mencionó en el apartado 2.1.1, la CMDB también cumple un papel esencial como nexo de unión entre ITAM y la Gestión de Servicios de TI (ITSM). Esta conexión se puede apreciar en la **figura 16**, donde se muestra cómo los datos de activos y configuración fluyen entre ambos entornos, permitiendo una visión unificada tanto técnica como operativa:

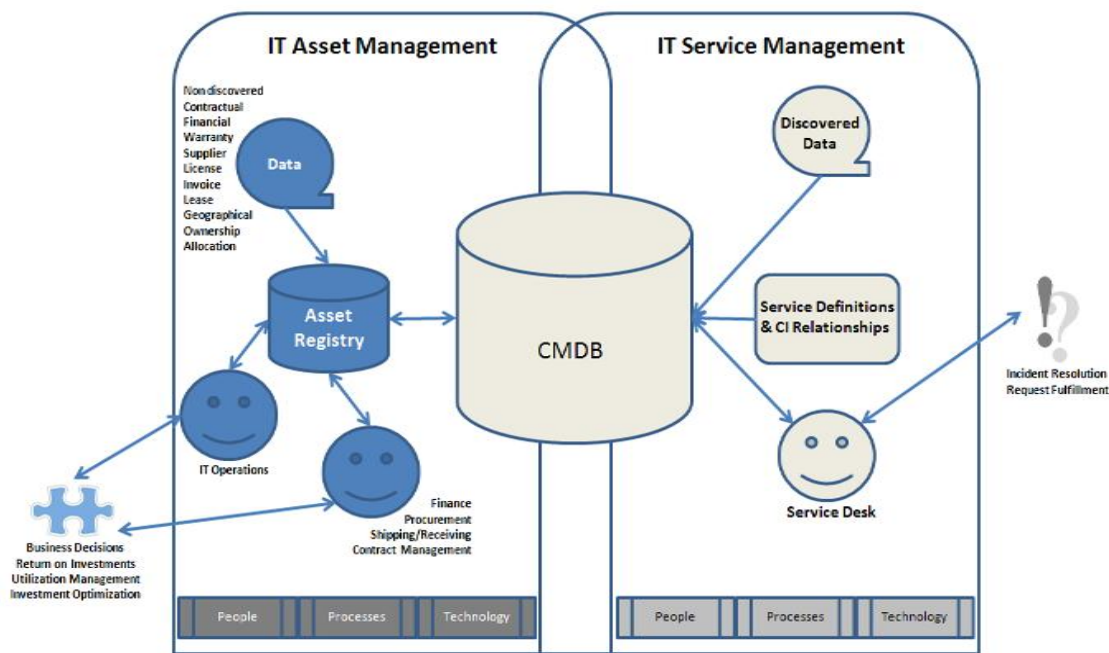


Figura 16: Flujo de información entre ITAM, ITSM y la CMDB en la gestión integral de activos y servicios de TI [29]

### 2.2.3 Tendencias recientes en la gestión de activos IT

La gestión de activos tecnológicos (ITAM) está evolucionando para adaptarse a entornos cada vez más complejos, híbridos y orientados al negocio. A continuación, se presentan las principales tendencias que están marcando esta transformación:

- **Gestión de activos en la nube y SaaS:** La migración a la nube ha ampliado el concepto de "activo", incluyendo instancias virtuales, contenedores y suscripciones SaaS. Actualmente, el 94% de las organizaciones utilizan servicios en la nube, y se espera que el mercado de SaaS alcance los 344 mil millones de dólares para 2027 [30]. Las herramientas modernas de ITAM se integran con APIs de proveedores cloud para descubrir y monitorear estos activos, facilitando su control y optimización.
- **Convergencia entre ITAM y FinOps:** La creciente adopción de la nube y el modelo SaaS ha difuminado las fronteras entre ITAM y FinOps. Ambas disciplinas buscan optimizar el gasto tecnológico y mejorar la gobernanza financiera. Según Flexera, el 32% de los equipos de ITAM colaboran activamente con FinOps, frente al 25% en 2023 [10]. Esta colaboración permite una visión unificada del gasto en tecnología, abarcando desde recursos en la nube hasta licencias de software [31].
- **Integración de dispositivos IoT:** El auge del Internet de las Cosas (IoT) ha introducido una nueva categoría de activos, como sensores y dispositivos inteligentes. La gestión de estos dispositivos requiere políticas específicas de ciclo de vida y seguridad, así como capacidades de analítica para mantenimiento predictivo [32].

- **Automatización e inteligencia artificial:** La IA y el aprendizaje automático están comenzando a aplicarse en ITAM para tareas como la normalización de datos, la reconciliación de activos duplicados y la predicción de fallos. Además, la automatización robótica de procesos (RPA) se utiliza para extraer datos de facturas y actualizar sistemas de activos, reduciendo la intervención manual [33] [32].
- **Enfoque en la experiencia del empleado (DEX):** La gestión de activos se orienta cada vez más a mejorar la experiencia digital del empleado. Esto incluye portales de autoservicio para gestionar activos asignados, encuestas automatizadas tras la entrega de nuevos dispositivos y chatbots que permiten consultar información sobre activos mediante lenguaje natural [34].

Entre estas tendencias, destaca la adopción de la inteligencia artificial como un cambio disruptivo en la gestión de activos IT. La IA permite automatizar procesos, mejorar la toma de decisiones y optimizar el uso de recursos. Este avance será explorado en detalle en el siguiente capítulo, donde se analizarán los fundamentos de la IA generativa y su aplicación en entornos empresariales.

## **2.3 IA generativa como impulsora del cambio en los entornos IT**

### **2.3.1 Fundamentos y evolución de la IA generativa**

La inteligencia artificial generativa se refiere a técnicas de machine learning capaces de crear contenido original (texto, imágenes, audio, código, etc.) a partir de los patrones aprendidos en grandes conjuntos de datos [35]. En lugar de limitarse a clasificar datos o predecir valores (como la IA tradicional discriminativa), la IA generativa produce nuevos datos que parecen creados por humanos. Esto es posible gracias a modelos de aprendizaje profundo (deep learning) entrenados con enormes volúmenes de información, identificando relaciones y estructuras presentes en los datos para luego utilizarlas en la generación de contenidos novedosos [35]. Un ejemplo cotidiano de esta tecnología es ChatGPT, un popular chatbot de OpenAI lanzado en 2022, que ha demostrado la capacidad de un modelo generativo de lenguaje para entablar conversaciones, escribir poemas o responder preguntas de forma coherente y natural [36]. Su aparición ha marcado un hito en la masificación de la IA generativa, atrayendo la atención mundial hacia estas capacidades. De hecho, la última oleada de avances ha pasado de la generación de imágenes realistas, como la lograda por redes generativas adversarias (GAN) que transformaban selfies en retratos de estilo renacentista en 2017, a impresionantes progresos en procesamiento de lenguaje natural (NLP), donde grandes modelos de lenguaje pueden improvisar sobre casi cualquier tema [36]. Estas nuevas aplicaciones van más allá del entretenimiento: los modelos generativos pueden aprender también la “gramática” de otros dominios, como el código de software, moléculas químicas o secuencias de ADN, generando resultados útiles en cada campo [36].

El auge de la IA generativa en los últimos años ha sido impulsado por la evolución de la arquitectura Transformer (introducida en 2017), que permitió escalar modelos con miles de millones de parámetros. Gracias a esta arquitectura, los modelos generativos actuales pueden mantener coherencia en largas secuencias de texto y capturar dependencias complejas. Así, un modelo generativo moderno no selecciona respuestas de un catálogo predefinido, sino que construye palabra por palabra (o píxel a píxel en el caso de imágenes) su salida, calculando en cada paso la continuación más probable según lo aprendido. El resultado son sistemas capaces de crear contenido nuevo que frecuentemente es indistinguible del producido por humanos [35]. Por supuesto, lograr esto requiere un entrenamiento a gran escala: típicamente se emplean conjuntos masivos de datos no estructurados (por ejemplo, todo el texto disponible en Internet, imágenes recopiladas de bancos públicos, etc.) para que el modelo aprenda patrones generales del lenguaje o de las imágenes. Este entrenamiento suele ser auto supervisado, el modelo aprende sin intervención humana directa, por ejemplo, intentando predecir la siguiente palabra de una frase una y otra vez hasta afinar su “entendimiento” estadístico del lenguaje [35].

En síntesis, la IA generativa brinda la capacidad de sintetizar información nueva a partir del conocimiento existente en los datos. Sus aplicaciones van desde la generación de texto (chatbots, asistentes virtuales, traducción automática),

pasando por la creación de imágenes y videos sintéticos (arte digital, simulaciones, deepfakes), hasta la generación de código y música. La **figura 17** ilustra este concepto: un modelo generativo recibe una entrada (por ejemplo, una instrucción en lenguaje natural) y produce una salida creativa (una respuesta textual, una imagen, etc.) basándose en lo aprendido durante su entrenamiento. Este enfoque contrasta con los sistemas tradicionales programados manualmente, y explica por qué la IA generativa puede adaptarse a tantas tareas distintas.

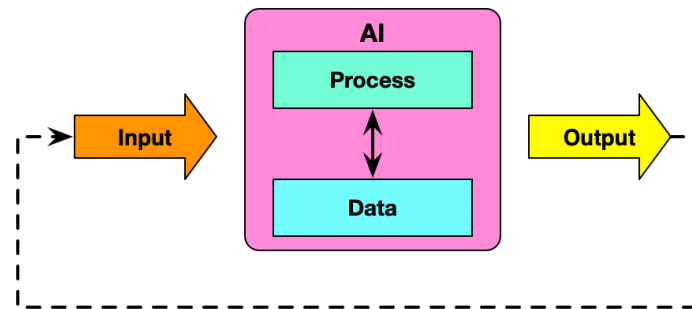


Figura 17: Diagrama conceptual del funcionamiento de un modelo de IA generativa [37]

El impacto de esta tecnología está siendo considerable en múltiples sectores. Se ha observado una adopción acelerada en el ámbito empresarial: según un estudio global de McKinsey, un tercio de las organizaciones ya utilizan IA generativa de forma regular en al menos una función de su negocio [1]. Gartner, por su parte, proyecta que para 2026 más del 80% de las compañías habrán desplegado aplicaciones basadas en IA generativa [38]. Estas cifras reflejan la magnitud de la tendencia y las expectativas puestas en la IA generativa. No obstante, también conllevan nuevos desafíos en cuanto a la calidad, veracidad y ética de los contenidos generados, lo que exige desarrollar medidas como filtros, validaciones y controles para su uso responsable. Pese a ello, el consenso es que estamos ante un cambio de paradigma en la informática, ya que la IA generativa ofrece formas más naturales de interactuar con la información y los sistemas, habilitando soluciones innovadoras que antes no eran posibles.

### Foundation Models y NLP

El reciente éxito de la IA generativa se sostiene sobre una nueva categoría de modelos de aprendizaje automático denominados modelos fundacionales (foundation models). Según investigadores del Stanford Institute for Human-Centered Artificial Intelligence, un modelo fundacional es “*cualquier modelo de IA entrenado con datos amplios y heterogéneos (generalmente mediante auto-supervisión a gran escala) que puede adaptarse a una amplia gama de tareas finales*” [39]. En otras palabras, son modelos de propósito general, no diseñados para una tarea específica, sino preentrenados para captar conocimientos básicos del lenguaje, la visión u otros dominios, de tal forma que luego puedan reutilizarse como base (*foundation*) para resolver diversos problemas.

Estos modelos aprovechan ideas de transfer learning. En primer lugar, aprenden patrones generales a gran escala, y después se afinan (*fine-tuning*) o se configuran para tareas particulares con mucho menos esfuerzo que entrenar un modelo desde cero. El **transfer learning** permite reutilizar el conocimiento adquirido por un modelo en un dominio diferente al que fue entrenado inicialmente. Por otro lado, el **fine-tuning** consiste en reentrenar una parte del modelo, normalmente las capas superiores, con datos específicos del nuevo dominio para adaptarlo mejor a la nueva tarea sin perder lo aprendido anteriormente.

Algunas características destacadas de estos modelos son:

- Se entrenan con datos no etiquetados, en volúmenes masivos, y sin intervención humana directa.
- Pueden adaptarse fácilmente a tareas concretas con un esfuerzo computacional reducido.
- Exhiben capacidades emergentes, es decir, habilidades complejas que no fueron programadas de forma explícita pero que surgen como resultado del entrenamiento a gran escala [39].

En la práctica, estos modelos se basan en redes neuronales de gran escala, con miles de millones de parámetros ajustables. Su entrenamiento exige una potencia computacional elevada y conjuntos masivos de datos. El resultado son modelos capaces de ofrecer un rendimiento sobresaliente en múltiples tareas una vez adaptados [39]. Dada su eficacia, estos modelos se han convertido en un estándar sobre el cual se construyen hoy en día muchos sistemas de IA. Este enfoque ha provocado un cambio de paradigma en la industria. En lugar de construir numerosos modelos especializados para tareas concretas, muchas organizaciones están optando por estandarizar el uso de unos pocos modelos fundacionales versátiles. Esta estrategia permite concentrar los esfuerzos en mejorar su robustez, reducir sesgos y optimizar el rendimiento general. Sin embargo, también implica riesgos importantes: cualquier defecto o sesgo presente en el modelo base podría replicarse en todas sus aplicaciones [39].

El impacto de estos modelos en el procesamiento del lenguaje natural (NLP) ha sido especialmente notable. Los modelos de lenguaje de gran tamaño (*Large Language Models*, o LLM) se han convertido en el eje central de muchas de las aplicaciones actuales de NLP [35]. Se entrenan con cantidades ingentes de texto extraído de libros, artículos, páginas web o foros, lo que les permite aprender las regularidades del lenguaje humano y entender su contexto. Su funcionamiento se asemeja al de un “cerebro lingüístico” preentrenado, que puede adaptarse a casi cualquier tarea relacionada con el lenguaje con una mínima cantidad de datos adicionales.

Como puede observarse en la **figura 18**, un modelo fundacional en NLP puede utilizarse para tareas como generación de texto, respuesta a preguntas, análisis de sentimientos o reconocimiento de objetos, entre muchas otras.

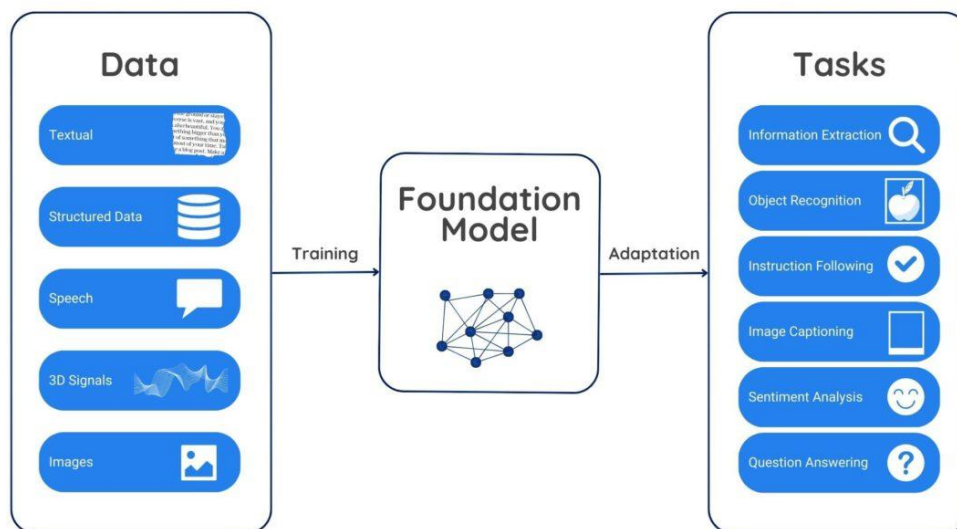


Figura 18: Arquitectura y uso de los Foundation Models para tareas NLP [40]

Por ejemplo, BERT (de Google, 2018) fue uno de los primeros modelos fundacionales en NLP. Fue entrenado de forma bidireccional para entender el contexto de las palabras y, tras su publicación, se empezó a reutilizar ampliamente para tareas como análisis de sentimientos, extracción de información o respuestas a preguntas mediante un ligero ajuste con datos etiquetados específicos. Pocos años después, GPT-3 (OpenAI, 2020) llevó el concepto al extremo opuesto con 175 mil millones de parámetros entrenados para generar texto, demostrando que un único modelo era capaz de redactar artículos, escribir código o entablar diálogos con solo recibir instrucciones en lenguaje natural (prompting), sin necesidad de reentrenamiento explícito para cada tarea. Este fenómeno, conocido como aprendizaje en contexto (few-shot learning), evidenció el poder de los modelos fundacionales de gran escala para generalizar conocimientos y resolver problemas nuevos con mínimos ejemplos [36].

Hoy en día existen numerosos modelos fundacionales de lenguaje desarrollados por distintas organizaciones, muchos de ellos de código abierto. Google introdujo su familia PaLM (Pathways Language Model, 2022) con hasta 540 mil millones de parámetros, enfocada en razonamiento y multiidioma, y Meta AI lanzó LLaMA (2023) con versiones de hasta 65 mil millones de parámetros, optimizado para funcionar en entornos con recursos más limitados. Estos modelos, junto con GPT-4 de OpenAI (2023) que incorporó capacidades multimodales al aceptar también imágenes como entrada, representan el estado del arte en NLP, alcanzando o superando el nivel humano en tareas como comprensión de lectura, composición de textos y resolución de problemas complejos [39]. Grandes empresas tecnológicas (Google, Microsoft, Meta, Amazon) invierten intensivamente en el desarrollo y despliegue de estos modelos fundacionales en sus productos, desde buscadores web hasta asistentes virtuales [39]. Al mismo tiempo, startups especializadas (Anthropic, AI21 Labs, Cohere, Hugging Face, entre otras) se han sumado a la carrera, ofreciendo sus propios modelos o facilitando el acceso a modelos abiertos, lo que ha democratizado en cierta medida el acceso a esta tecnología.

En la siguiente figura que se muestra a continuación puede verse la evolución de los principales LLM a lo largo del tiempo:

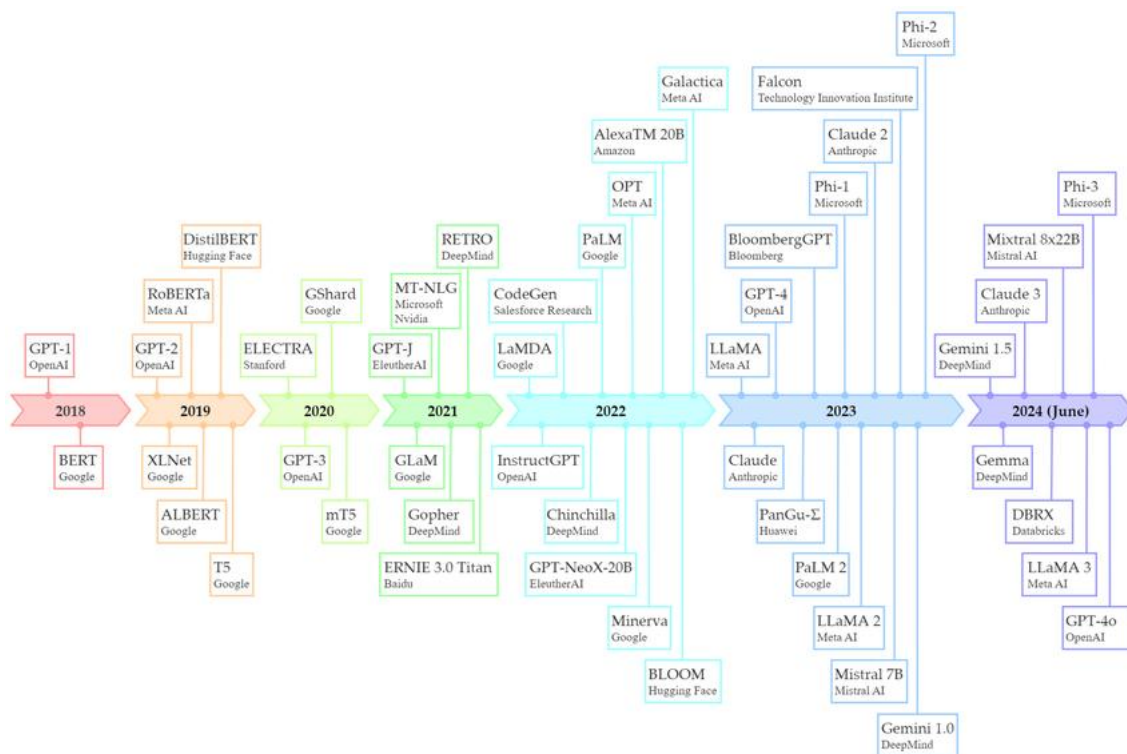


Figura 19: Evolución de los principales modelos LLM entre los años 2018 y junio de 2024 [41].

El papel de los modelos fundacionales en NLP se resume en su capacidad de servir como infraestructura básica sobre la cual construir soluciones. En lugar de programar reglas lingüísticas a mano o entrenar un modelo diferente para cada tarea de lenguaje, una organización puede aprovechar un LLM preentrenado y adaptarlo a sus necesidades. Esto ha simplificado enormemente la creación de aplicaciones de lenguaje natural, desde traductores automáticos hasta sistemas de resumen de documentos o chatbots avanzados. Por ejemplo, un mismo LLM puede, con ajustes mínimos, dar vida a un asistente virtual capaz de contestar preguntas frecuentes, y al mismo tiempo resumir informes extensos o analizar el tono de opiniones de clientes.

Los modelos fundacionales han cambiado las reglas del juego en NLP. Su capacidad para concentrar conocimiento lingüístico general y luego transferirlo a casos específicos ha elevado el desempeño en comprensión y generación de lenguaje a niveles inimaginables. Gracias a ellos, hoy es posible contar con sistemas conversacionales y asistentes inteligentes realmente competentes, que entienden preguntas complejas y producen respuestas útiles, todo ello construido sobre la misma base. Mas adelante en el proyecto se mostrará como esta tecnología puede aplicarse en la creación de una herramienta inteligente enfocada a la gestión de activos IT.

### 2.3.2 Aplicaciones de la IA generativa a la gestión de activos IT

Hasta hace poco, la gestión de activos tecnológicos dependía en gran medida de procesos manuales, como el registro de activos en hojas de cálculo, la revisión de contratos o la atención directa a los empleados que solicitaban soporte. La llegada de la IA generativa, especialmente a través de modelos de lenguaje como ChatGPT o Claude, está transformando este panorama al introducir nuevas formas de interactuar con los sistemas y automatizar tareas rutinarias.

Gracias a los modelos de lenguaje de gran tamaño (LLMs), es posible construir asistentes virtuales que entienden el lenguaje natural y pueden responder preguntas como: “¿Qué portátiles tengo asignados?” o “¿Cuándo vence la licencia de mi antivirus?”. Estos asistentes no solo hacen más accesible la información, sino que también permiten ahorrar tiempo a los equipos de soporte técnico y mejorar la experiencia de los empleados.

Algunas de las aplicaciones más relevantes de la IA generativa en el ámbito de ITAM son:

- **Asistentes virtuales conversacionales:** permiten realizar consultas en lenguaje natural sobre el estado, la ubicación o el historial de los activos. Por ejemplo, pueden responder automáticamente a preguntas sobre qué dispositivos están sin garantía o cuántas licencias están activas por departamento [42].
- **Soporte 24/7 para usuarios y técnicos:** herramientas como *GroWbot 2.0*, desarrollado por GroWrk, utilizan modelos como GPT-4 para ofrecer asistencia continua. Este chatbot es capaz de informar del estado de envío de los equipos, detectar problemas recurrentes e incluso abrir tickets automáticamente cuando no puede resolver un incidente por sí mismo [42].
- **Gestión de licencias y contratos:** algunas plataformas de gestión de SaaS, como *Torii*, han incorporado asistentes con IA que permiten consultar en lenguaje natural detalles como cuántas licencias están en uso, qué usuarios las utilizan o cuándo expiran los contratos asociados [42].
- **Optimización de recursos y decisiones FinOps:** *Snow Copilot*, desarrollado por Snow Software, ofrece un ejemplo claro de integración de IA generativa para ayudar en la toma de decisiones sobre el entorno IT. Este asistente puede identificar servidores infrautilizados, responder preguntas estratégicas sobre inventarios o proponer acciones para reducir costes en la nube [42].

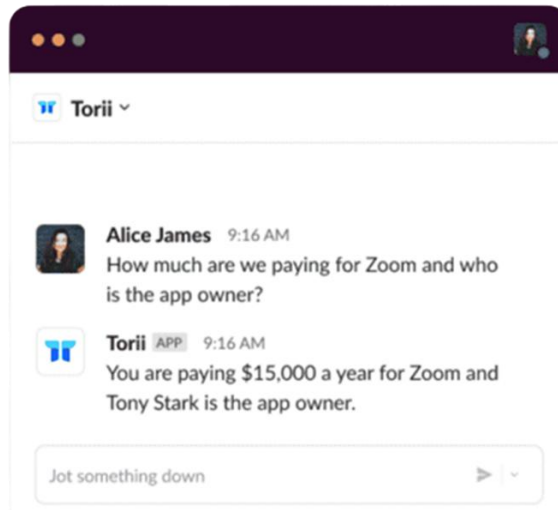


Figura 20: Asistente de IA generativa "Torii" respondiendo a una consulta sobre licencias de software [43].

Además de ayudar en la interacción con los usuarios, los modelos generativos también pueden analizar grandes volúmenes de texto, como registros de incidencias o descripciones de fallos, para detectar patrones y sugerir soluciones en tiempo real. Esto permite reducir el tiempo de diagnóstico y mejora la eficiencia del soporte técnico. También pueden utilizarse para generar artículos de ayuda o guías internas a partir del conocimiento ya existente en la organización, facilitando la actualización continua de la documentación técnica [44].

Entre los principales beneficios que aporta la IA generativa a la gestión de activos IT, destacan los siguientes:

- **Reducción del esfuerzo manual y repetitivo:** los asistentes basados en IA pueden encargarse de tareas como la búsqueda de información técnica, la recopilación de datos de inventario o la generación de tickets de soporte, lo que permite liberar a los técnicos de tareas operativas y centrarse en actividades de mayor valor [44].
- **Mayor agilidad en la obtención de datos relevantes:** al permitir consultas en lenguaje natural, los sistemas impulsados por IA generativa eliminan la necesidad de navegar por menús complejos o recordar comandos específicos. Esto acelera la localización de información sobre activos, licencias o incidencias [42] [44].
- **Mejora de la experiencia de usuario:** cualquier empleado, independientemente de su perfil técnico, puede interactuar con estos sistemas de forma intuitiva. Esto facilita el acceso a los datos de IT y reduce la dependencia del personal especializado [44].
- **Apoyo en la toma de decisiones:** la IA generativa puede resumir información compleja, identificar patrones en grandes volúmenes de datos y proponer recomendaciones concretas. Esto facilita la evaluación de riesgos, la optimización de recursos y la planificación de acciones correctivas o preventivas [44].

En conjunto, estos avances están transformando el rol de ITAM, pasando de ser una función centrada en tareas administrativas a un proceso más inteligente, proactivo y orientado a la mejora continua. Por todo ello, todo apunta a que aquellas organizaciones que implementen soluciones basadas en IA generativa no solo ganarán en eficiencia operativa, sino que también podrán ofrecer un servicio más accesible, flexible y moderno tanto a empleados como a clientes.

### 2.3.3 Amazon Bedrock y su papel en el ecosistema cloud

A medida que la IA generativa se consolida como una tecnología estratégica, los grandes proveedores de servicios en la nube (cloud) han comenzado a integrarla en sus plataformas para facilitar su adopción masiva. Amazon Web Services (AWS), líder en servicios cloud, presentó en 2023 Amazon Bedrock como parte de su apuesta por la IA generativa. Amazon Bedrock es un servicio que ofrece acceso bajo demanda a modelos fundacionales de alto rendimiento desarrollados por múltiples empresas líderes en IA [45].

Amazon Bedrock permite acceder a modelos fundacionales de texto e imagen de varios proveedores sin necesidad de gestionar infraestructura propia. Se trata de un servicio completamente gestionado que facilita a las empresas la integración de modelos generativos en sus aplicaciones mediante una única API [45] [46].

Entre los principales modelos disponibles destacan los siguientes:

- **Anthropic:** Claude 3.5 Sonnet v2, Claude 3.5 Haiku, Claude 4 Sonnet y Claude 4 Opus.
- **Meta:** LLaMA 3.1, 3.2 y 3.3, y los nuevos LLaMA 4 Maverick y LLaMA 4 Scout.
- **Amazon:** Titan, Nova Micro, Lite, Premier y Pro.
- **DeepSeek:** modelo Deepseek R1.
- **Cohere:** modelo Rerank 3.5.

Esta variedad convierte a Bedrock en una plataforma flexible y adaptable a distintos casos de uso, desde generación de texto conversacional hasta procesamiento de imágenes o clasificación de documentos. Además, al ser un servicio serverless, no requiere aprovisionar ni mantener servidores, lo que reduce las barreras técnicas para su adopción [46].

Dentro del ecosistema cloud de Amazon, Bedrock no es una solución aislada, sino que forma parte de un conjunto más amplio de herramientas ofrecidas por AWS para construir aplicaciones inteligentes. La nube de Amazon proporciona servicios especializados que pueden combinarse con Bedrock para desarrollar soluciones completas:

- **Amazon S3:** servicio de almacenamiento de objetos altamente escalable, ideal para guardar documentación técnica, archivos de configuración, resultados generados por el modelo o bases de conocimiento en bruto [47].

- **AWS Lambda:** permite ejecutar funciones en la nube en respuesta a eventos sin necesidad de servidores. Es una pieza clave para conectar consultas del usuario con funciones específicas del sistema, como acceder a una base de datos o transformar datos antes de mostrarlos [47].
- **Amazon API Gateway:** facilita la creación y gestión de APIs seguras que permiten que aplicaciones web, móviles u otros servicios se comuniquen con los componentes del backend, como funciones Lambda o bases de datos [47].
- **Amazon RDS y DynamoDB:** permiten almacenar información estructurada de forma segura y eficiente. RDS ofrece bases de datos relacionales como MySQL o PostgreSQL, mientras que DynamoDB es una base NoSQL orientada a documentos y clave-valor [47].
- **Amazon Cognito:** gestiona el inicio de sesión, el registro y el control de acceso de los usuarios de forma centralizada. Permite diferenciar roles y proteger el acceso a recursos sensibles dentro de la aplicación [47].

Esta combinación de servicios permite extender las capacidades de Bedrock e integrarlo dentro de una solución cloud robusta, flexible y alineada con los requisitos reales de los entornos empresariales modernos. Bedrock, en este contexto, actúa como la puerta de entrada al uso de modelos de lenguaje, mientras que otras piezas del ecosistema completan la arquitectura necesaria para resolver casos reales.

Desde su lanzamiento, Amazon Bedrock ha ampliado progresivamente sus capacidades para facilitar su integración en entornos corporativos. Actualmente ofrece funcionalidades como la personalización de modelos mediante ajustes privados y el acceso a datos internos mediante técnicas de recuperación aumentada (*Retrieval-Augmented Generation*), incluyendo conexiones automatizadas con fuentes estructuradas como Amazon RDS. Sin embargo, en escenarios donde se requiere una gestión precisa del contexto conversacional, la coordinación con flujos de negocio definidos o la integración con múltiples servicios de backend, sigue siendo necesario diseñar arquitecturas específicas que combinen Bedrock con otros componentes de AWS. Al estar diseñado como un servicio sin estado, cada interacción se procesa de forma independiente, por lo que el manejo del contexto, la interpretación de consultas y la orquestación de respuestas deben implementarse externamente. Esta flexibilidad permite adaptar el comportamiento del sistema a las necesidades concretas de cada organización, como por ejemplo cuando se desea ofrecer respuestas personalizadas basadas en información registrada en una base de datos [46] [48].

En el siguiente capítulo se mostrará cómo se ha desarrollado una solución basada en inteligencia artificial orientada a la gestión de activos IT, integrando de forma coordinada varios servicios de AWS en una arquitectura funcional diseñada para facilitar consultas eficientes y accesibles a los datos tecnológicos de una organización.

## 3 Desarrollo

Tras haber revisado en el capítulo anterior los fundamentos teóricos, estándares, herramientas y tecnologías asociadas a la gestión de activos tecnológicos, en este capítulo se describe en detalle la solución implementada. Se abordarán los requisitos funcionales y no funcionales del sistema, la arquitectura general en la nube, el diseño de la base de datos y el funcionamiento de los distintos componentes desarrollados. Todo ello con el objetivo de ofrecer una solución práctica, eficiente y fácilmente desplegable para entornos empresariales.

### 3.1 Requisitos del sistema

En este apartado se detallan los requisitos fundamentales que debe cumplir el sistema de gestión de activos IT, tanto a nivel funcional como no funcional. Dicho sistema está orientado principalmente a su uso en empresas pequeñas y medianas (pymes), con el objetivo de facilitar al personal técnico el control y la consulta de activos tecnológicos mediante una solución cloud integrada con servicios de AWS y un chatbot basado en inteligencia artificial generativa.

#### Requisitos funcionales (RF)

Los siguientes requisitos funcionales definen el conjunto de acciones que el sistema debe ser capaz de realizar desde el punto de vista del usuario administrador:

- **RF1.** El sistema debe permitir consultar activos, usuarios y ubicaciones mediante un chatbot que acepte preguntas formuladas en lenguaje natural.
- **RF2.** El chatbot debe ser capaz de interpretar consultas abiertas, sin depender de comandos estructurados ni expresiones específicas.
- **RF3.** El sistema debe permitir realizar operaciones CRUD (crear, leer, actualizar, eliminar) sobre activos, usuarios y ubicaciones a través de una API REST.
- **RF4.** El sistema debe ofrecer respuestas coherentes y completas a partir de la información almacenada en su base de datos.
- **RF5.** El acceso a las funciones de gestión debe estar restringido a usuarios autenticados.

#### Requisitos no funcionales (RNF)

Los siguientes requisitos no funcionales establecen las características de calidad que debe tener el sistema en cuanto a disponibilidad, rendimiento, seguridad, eficiencia y coste:

- **RNF1.** El procesamiento de las consultas y la ejecución de las funciones del sistema deben realizarse íntegramente en la nube mediante servicios gestionados de AWS.
- **RNF2.** El sistema debe utilizar componentes escalables y sin servidor, que reduzcan los costes operativos y simplifiquen el mantenimiento.

- **RNF3.** La autenticación de usuarios debe integrarse en el sistema para proteger las operaciones sensibles.
- **RNF4.** La arquitectura debe facilitar tiempos de respuesta bajos en condiciones normales de uso.
- **RNF5.** El diseño debe priorizar un consumo eficiente de recursos, especialmente en lo relativo a invocaciones y procesamiento de lenguaje natural.
- **RNF6.** El sistema debe ser económicamente sostenible y mantenerse operativo dentro de un presupuesto reducido, sin requerir infraestructura propia por parte de la empresa usuaria.

## 3.2 Arquitectura general del sistema

El sistema de gestión de activos tecnológicos ha sido diseñado siguiendo una arquitectura modular basada en servicios sin servidor (*serverless*) ofrecidos por Amazon Web Services (AWS). Esta estructura permite una implementación ágil, escalabilidad automática y una reducción significativa del mantenimiento del backend, lo que resulta especialmente conveniente en proyectos con recursos limitados como es el caso.

Todos los componentes principales del sistema se han desplegado en la región **us-west-2** (Oregón) por varios motivos:

- Se trata de una región incluida en el AWS Free Tier, lo que permite acceder gratuitamente a servicios como Lambda, API Gateway, Amazon RDS y Cognito dentro de los límites de uso establecidos.
- Alberga una de las mayores disponibilidades de modelos fundacionales en Amazon Bedrock, incluyendo modelos como Claude 3.5 Sonnet v2 o DeepSeek R1, lo que proporciona flexibilidad a la hora de elegir el modelo más adecuado para el sistema conversacional.
- Ofrece buena disponibilidad y baja latencia tanto para usuarios en Europa como en América, lo que la convierte en una ubicación óptima para una solución accesible desde múltiples localizaciones sin necesidad de red privada virtual (VPN).

La única excepción es la gestión de alertas de coste, ya que Amazon CloudWatch solo permite configurar alarmas presupuestarias en la región *us-east-1*. Por ello, dicha funcionalidad se ha centralizado en esa región, mientras que el resto de los servicios permanecen desplegados en *us-west-2*.

El sistema se divide en dos grandes capas: el **subsistema de consulta inteligente mediante lenguaje natural (Chatbot AI)** y el **subsistema de gestión de recursos (API REST)**. Ambos subsistemas comparten la misma base de datos relacional (Amazon RDS) pero operan de forma desacoplada, lo que permite desarrollar, mantener y escalar cada uno de manera independiente.

## Subsistema de consulta inteligente (Chatbot AI)

Este subsistema permite realizar consultas en lenguaje natural sobre los activos tecnológicos y usuarios registrados en el sistema, así como sus ubicaciones. Está formado por los siguientes componentes:

- **Cliente local (chatbot\_client.py):** script en Python que gestiona la interacción con el usuario desde la línea de comandos. Se encarga de enviar la consulta a Bedrock y llamar a las funciones lambda para recibir los datos desde la base de datos.
- **Amazon Bedrock (Claude 3.5 Sonnet v2):** interpreta la intención del usuario, determina qué función Lambda debe ejecutarse y, tras obtener los datos, genera una respuesta en lenguaje natural adaptada a la consulta del usuario.
- **Funciones AWS Lambda:** ejecutan las consultas SQL necesarias sobre la base de datos, en función del tipo de pregunta (por usuario, activo, ubicación o global).
- **Amazon RDS (MySQL):** almacena toda la información relativa a activos, usuarios y ubicaciones. Es accedida por las funciones Lambda para obtener los datos solicitados por Bedrock.

## Subsistema de gestión de recursos (API REST)

Este subsistema permite gestionar los recursos del sistema de forma estructurada mediante una interfaz REST. Esta formado por los siguientes componentes:

- **Cliente REST (Postman):** empleado para consumir manualmente los endpoints de gestión expuestos por la API. Permite realizar operaciones como crear o modificar registros directamente desde un entorno de pruebas.
- **Funciones AWS Lambda:** encargadas de realizar operaciones CRUD sobre los datos de usuarios, activos y ubicaciones.
- **Amazon API Gateway:** expone de forma controlada los endpoints necesarios para interactuar con las funciones Lambda.
- **Amazon Cognito:** proporciona autenticación de usuarios para proteger el acceso a los endpoints REST definidos.
- **Amazon RDS (MySQL):** base de datos común donde se almacenan los datos gestionados por ambos subsistemas.

A diferencia del chatbot, esta capa está orientada a operaciones administrativas internas (alta, baja y modificación de registros) y cuenta con control de acceso mediante autenticación.

El funcionamiento general del sistema puede resumirse en dos flujos diferenciados:

- En el **primer flujo**, un usuario técnico utiliza el chatbot desde su máquina local para realizar una consulta en lenguaje natural. El cliente local comunica la petición a Bedrock, que interpreta la consulta, llama a la función Lambda correspondiente y genera una respuesta en lenguaje natural a partir de los datos extraídos de la base de datos.
- En el **segundo flujo**, un usuario autenticado accede a la API REST y realiza una operación administrativa sobre los datos del sistema (por ejemplo, registrar un nuevo activo), utilizando herramientas como Postman. La petición se autentica con Cognito y se redirige a la función Lambda correspondiente, que modifica la base de datos.

Este enfoque garantiza una clara separación de responsabilidades, permite escalar cada subsistema de forma independiente y facilita un control granular de costes, todos ellos aspectos clave para una solución diseñada dentro del marco del AWS Free Tier.

Como se puede observar en la **figura 21**, el sistema sigue una arquitectura distribuida y desacoplada, donde los servicios gestionados de AWS actúan como bloques funcionales que se comunican mediante eventos.

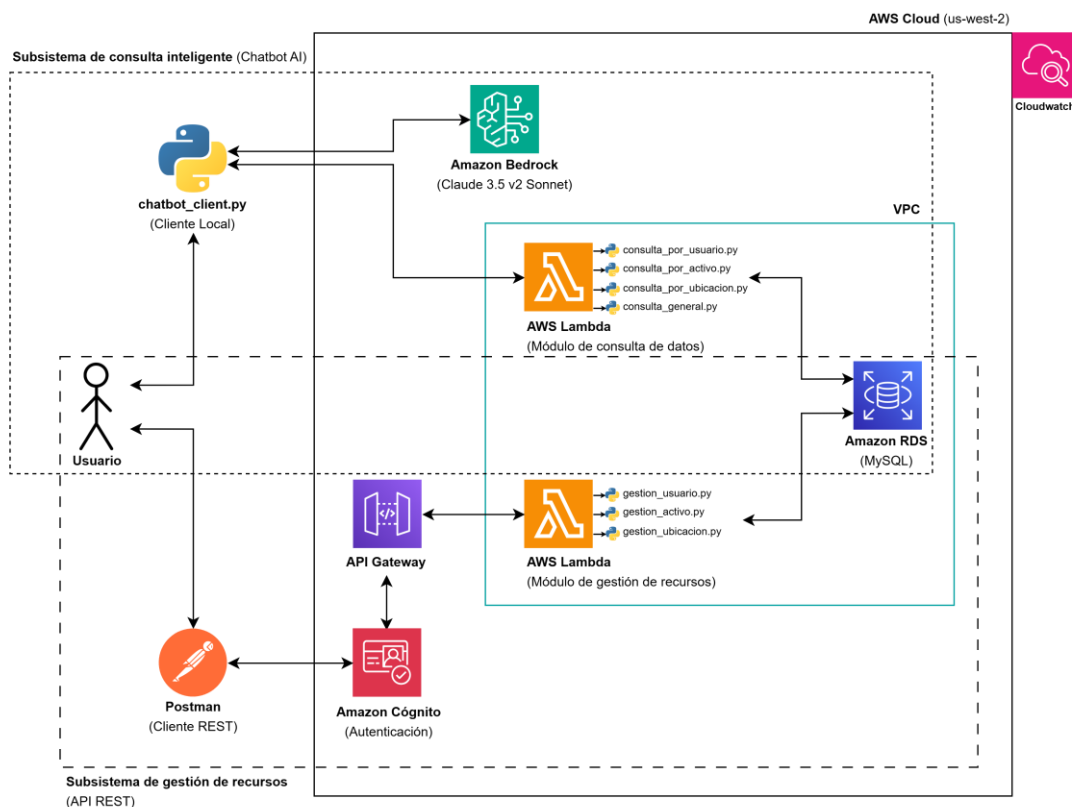


Figura 21: Diagrama general de la arquitectura del sistema (Elaboración propia)

### 3.3 Diseño de la base de datos

La base de datos del sistema se ha diseñado siguiendo un enfoque relacional normalizado, orientado a representar de forma clara los distintos componentes de un sistema de gestión de activos tecnológicos en una organización. Esta base de datos actúa como núcleo compartido entre los dos subsistemas principales: la capa de gestión de recursos y el chatbot conversacional. Su diseño se ha enfocado buscando un equilibrio entre normalización, eficiencia y simplicidad de consultas, especialmente teniendo en cuenta el procesamiento posterior de las respuestas por parte del modelo fundacional de Amazon Bedrock.

#### Modelo lógico y estructura

El diseño adopta un enfoque de **herencia de supertipo-subtipo** para representar los diferentes tipos de activos tecnológicos:

- La entidad Activo actúa como supertipo general y recoge atributos comunes como `id_activo`, `tipo`, `estado`, `fecha_adquisición`, `coste`, `recurrente` (si el coste lo es), `nombre_proveedor` y `email_propietario`.

A partir de esta tabla principal se derivan tres subtipos especializados, cada uno con sus propios atributos:

- **Hardware:** incluye información física como número de serie, modelo, marca, tipo (ordenador, impresora, etc.) y ubicación. Este subtipo es el único que contiene el campo `ubicacion_nombre`, que actúa como referencia directa a la entidad Ubicación.
- **Licencia:** representa productos de software asignados a usuarios. Contiene los campos `nombre_licencia`, `producto` y `numero_licencia`.
- **Línea Telefónica:** almacena las líneas móviles y sus datos técnicos: número, operadora, plan, PIN, PUK, SIM y el ID del dispositivo hardware al que están asociadas (`id_hardware`), si aplica.

Además, el sistema cuenta con dos entidades principales adicionales:

- **Usuario:** contiene los datos del personal, incluyendo `email`, `nombre_completo`, `telefono`, `estado`, `tipo_usuario`, `puesto`, `departamento` y `ubicacion`. El campo `ubicacion` actúa como clave foránea hacia la tabla de la entidad Ubicación.
- **Ubicación:** representa los espacios físicos donde se sitúan usuarios y activos. Incluye atributos como `nombre`, `pais`, `calle`, `número`, `localidad`, `provincia`, `código_postal` y `n_empleados`. Este último campo se actualiza automáticamente mediante **triggers** cada vez que se añade o elimina un usuario.

La creación detallada de todas estas tablas, claves primarias y relaciones puede consultarse en el script `consulta_creacion_tablas.sql`, incluido en el **Anexo** del presente documento.

Durante el diseño de la base de datos una de las decisiones clave ha sido evitar relaciones N:M o estructuras excesivamente normalizadas entre entidades. En su lugar, se ha optado por un diseño más simple y directo con relaciones 1:N explícitas. Esto ha permitido:

- **Reducir el número de joins** necesarios para obtener los datos relevantes en cada consulta.
- **Segmentar la base de datos**, lo que facilita la generación de respuestas enfocadas y con menor volumen de datos irrelevantes.
- **Optimizar el consumo de tokens** durante la interacción con el modelo de lenguaje, ya que las funciones Lambda pueden devolver únicamente la información esencial para cada caso.
- **Disminuir los tiempos de respuesta**, especialmente en consultas lanzadas desde el chatbot.

Además, este enfoque se adapta mejor a la arquitectura serverless del sistema, minimizando la complejidad del backend y mejorando la trazabilidad de los datos devueltos.

Como se puede observar en la **figura 22**, el modelo entidad-relación final refleja esta estructura jerárquica centrada en el tipo de activo, con relaciones simples, claves foráneas bien definidas y una organización que permite escalar y mantener el sistema con facilidad.

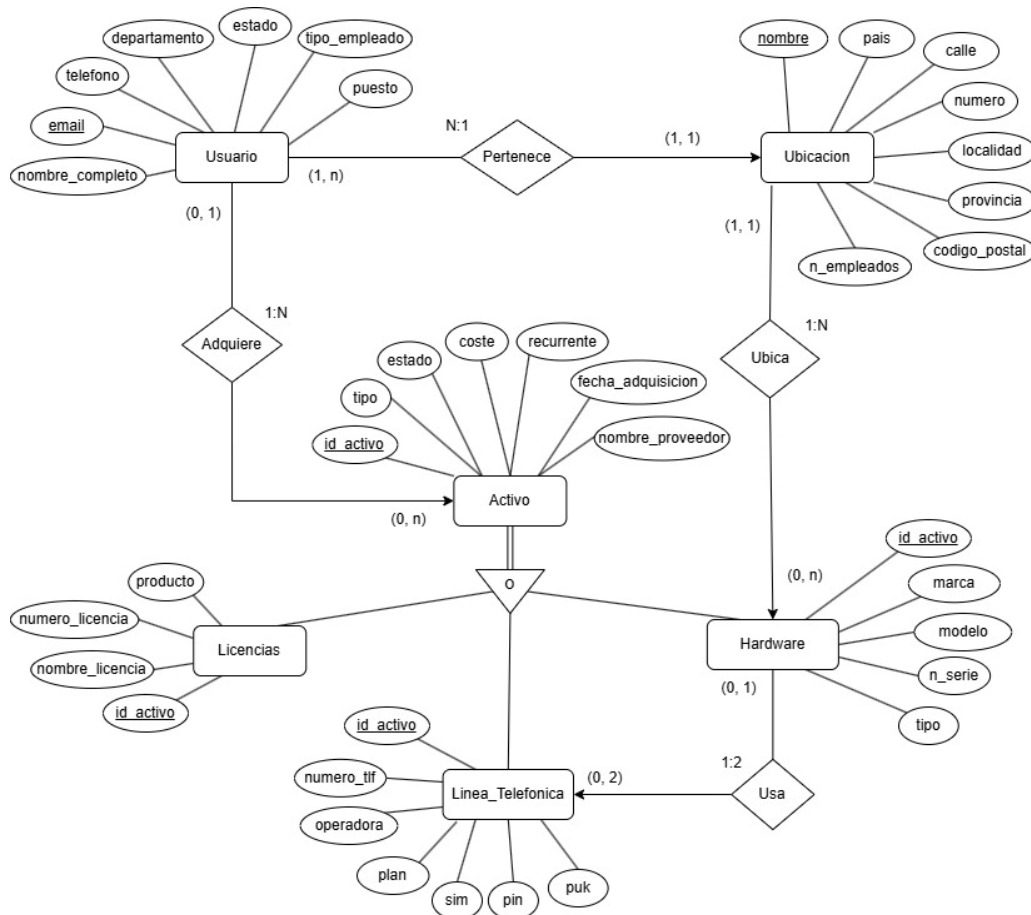


Figura 22: Diagrama entidad-relación de la base de datos (Elaboración propia)

## Implementación en la nube

La base de datos ha sido desplegada en Amazon RDS, utilizando el motor MySQL 8.0.40 en la región us-west-2 (Oregón). Se ha configurado como una instancia db.t4g.micro, incluida en el AWS Free Tier, con 2 vCPU, 1 GB de RAM y 20 GiB de almacenamiento SSD generalizado.

La instancia se encuentra dentro de una VPC privada, compartida con las funciones AWS Lambda del sistema, lo que permite una comunicación directa y segura entre ambos componentes. Además, se ha asociado al Security Group por defecto, desde el cual es posible restringir el acceso a rangos de IP específicos. No obstante, para facilitar el desarrollo y las pruebas desde distintas ubicaciones, se ha habilitado temporalmente el acceso desde cualquier dirección IP.

Durante la configuración también se habilitó el cifrado en reposo con claves gestionadas por AWS KMS, mientras que funcionalidades como la replicación Multi-AZ o la autenticación mediante IAM fueron desactivadas, ya que no están incluidas en el plan gratuito y no eran necesarias en este entorno.

El acceso a la base de datos desde las funciones Lambda se gestiona mediante variables de entorno, evitando exponer las credenciales en el código fuente. Esta configuración ofrece un equilibrio adecuado entre seguridad, facilidad de desarrollo y coste cero dentro de los límites del entorno gratuito de AWS.

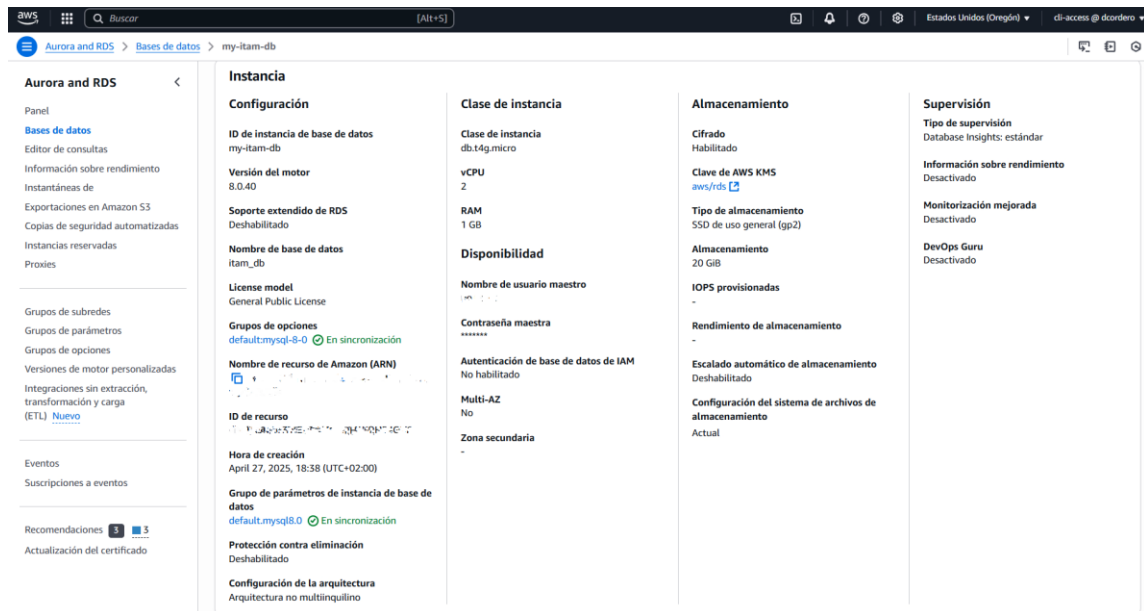


Figura 23: Configuración de la instancia de la base de datos creada en AWS

## 3.4 Subsistema de consulta inteligente (Chatbot AI)

El subsistema de consulta inteligente constituye el componente central del sistema desarrollado, al permitir que los usuarios interactúen con la base de datos utilizando lenguaje natural. Gracias a la integración con Amazon Bedrock, es posible realizar preguntas abiertas sobre la base de datos sin necesidad de conocer estructuras formales de consulta. El propio modelo fundacional se encarga de interpretar la intención de cada mensaje, activar la función Lambda correspondiente y generar una respuesta adaptada a la consulta original.

### 3.4.1 Funciones Lambda especializadas por tipo de consulta

Una de las decisiones clave durante el diseño fue separar la lógica de extracción de datos en cuatro funciones Lambda especializadas, en lugar de una función única con múltiples condiciones internas. Esta separación permite que cada función esté optimizada para un único tipo de consulta, lo que mejora la legibilidad del código, facilita su mantenimiento y permite escalar individualmente cada función si fuera necesario. Además, se reduce el tiempo de ejecución, ya que cada Lambda opera con una lógica acotada y bien definida.

Las funciones desarrolladas son:

- **consulta\_por\_usuario:** devuelve únicamente los datos del usuario solicitado, sus activos y su ubicación. El usuario puede ser identificado tanto por su nombre completo como por su correo electrónico. Ej:

*“Dime a que departamento pertenece el usuario david ramirez gil y que activos tiene asignados”*

- **consulta\_por\_activo:** localiza un activo a partir de su ID de activo, número de serie, número de licencia o número de línea telefónica, y devuelve la información de dicho activo, su propietario y su ubicación. Ej:

*“A que usuario pertenece el numero de teléfono 666555444 ?”*

*“Dime el coste y la fecha de adquisición del activo ORD01 ”*

- **consulta\_por\_ubicacion:** devuelve toda la información de los usuarios y activos asociados a una determinada ubicación física así como los datos de la propia ubicación. Ej:

*“Dime cual es el gasto mensual total en licencias de Microsoft en la oficina de madrid”*

*“Dime que usuarios de la oficina de Madrid tienen el cargo de administrador de sistemas”*

- **consulta\_general:** devuelve el estado completo de la base de datos, sin aplicar ningún filtro específico. Es la función más costosa en términos de consumo de tokens, pero necesaria para consultas amplias. Ej:

*“Dime qué usuarios tienen líneas de Yoigo y a que equipos de hardware están asociadas dichas líneas”*

*“Dime que usuarios de la oficina con más empleados utilizan ordenadores HP”*

### 3.4.2 Interpretación dinámica con Bedrock

Durante la fase de desarrollo se valoró inicialmente la posibilidad de identificar las consultas del usuario mediante keywords o estructuras predefinidas. Sin embargo, esta opción fue descartada rápidamente al evidenciarse su complejidad creciente y su escasa flexibilidad ante variaciones en el lenguaje. Finalmente se optó por aprovechar directamente la capacidad interpretativa del modelo Claude 3.5 Sonnet v2 puesto que esta alternativa resultó mucho más precisa, eficiente y fácil de mantener.

El sistema funciona mediante un prompt diseñado específicamente para que Bedrock actúe como un intérprete oculto: analiza la consulta del usuario y decide qué función Lambda debe llamarse y con qué parámetro. Esta operación se realiza en una primera llamada al modelo.

Posteriormente, tras obtener los datos correspondientes desde la base de datos, se realiza una segunda llamada a Bedrock (con el mismo agente) que genera la respuesta final en lenguaje natural. El modelo no tiene conciencia de que ha sido él mismo quien ha interpretado la consulta previamente, ya que cada llamada es completamente independiente. Esto permite mantener las dos responsabilidades de interpretación y respuesta dentro de un mismo agente, sin necesidad de compartir estado o duplicar sesiones.

A continuación, se muestra el fragmento del script *chatbot\_client.py* donde se define el prompt de interpretación:

```
10 # Prompt para interpretación
11 SYSTEM_PROMPT = """
12 Eres un agente especializado en interpretar consultas sobre activos informáticos realizadas en lenguaje natural.
13
14 Tu única tarea es analizar la consulta del usuario y devolver únicamente el nombre de la función Lambda a ejecutar
15 y su argumento (si es necesario).
16
17 Funciones disponibles:
18 1. consulta_por_activo,<identificador>
19 2. consulta_por_usuario,<identificador>
20 3. consulta_por_ubicacion,<identificador>
21 4. consulta_general
22
23 Reglas:
24 - El identificador puede contener espacios (por ejemplo: "Oficina Málaga").
25 - Si no hay identificador (consulta general), devuelve solo el nombre de la función.
26 - Formato de salida: solo texto, sin comillas, sin explicaciones, una sola línea, usa una coma si hay identificador.
27
28 Ejemplos:
29 Consulta: ¿Quién tiene el activo SN001?
30 Respuesta: consulta_por_activo,SN001
31
32 Consulta: Muéstrame que activos tiene elena gomez santos
33 Respuesta: consulta_por_usuario,Elena Gómez Santos
34
35 Consulta: ¿Qué usuarios están en las oficinas de malaga?
36 Respuesta: consulta_por_ubicacion,Oficina Málaga
37
38 Consulta: ¿Que usuarios tienen lineas de yoigo?
39 Respuesta: consulta_general
40 """
```

Figura 24: Fragmento del código de *chatbot\_client.py* en el que se muestra el prompt utilizado para la interpretación de las consultas.

Cabe destacar que el cuerpo de la solicitud a la API de Bedrock varía ligeramente en función del modelo fundacional seleccionado, ya que cada proveedor define su propia estructura y parámetros. No obstante, Amazon proporciona en su consola la documentación detallada y actualizada para cada uno de ellos, lo que permite adaptar el cliente de forma sencilla. Gracias a esta flexibilidad, el chatbot puede ser ajustado para funcionar con cualquier otro modelo compatible disponible en Bedrock (como Mistral, Meta Llama o Amazon Titan) realizando únicamente pequeñas modificaciones en el formato de la solicitud.

A continuación, se muestra el formato de solicitud de la API para el modelo fundacional utilizado en el proyecto:

```
{
  "modelId": "anthropic.claude-3-5-sonnet-20241022-v2:0",
  "contentType": "application/json",
  "accept": "application/json",
  "body": {
    "anthropic_version": "bedrock-2023-05-31",
    "max_tokens": 200,
    "top_k": 250,
    "stop_sequences": [],
    "temperature": 1,
    "top_p": 0.999,
    "messages": [
      {
        "role": "user",
        "content": [
          {
            "type": "text",
            "text": "hello world"
          }
        ]
      }
    ]
  }
}
```

Figura 25: Ejemplo de solicitud de la API de Amazon Bedrock para el modelo Claude 3.5 Sonnet v2 [49]

### 3.4.3 Optimización del uso de tokens

Uno de los aspectos más relevantes a nivel de eficiencia ha sido la optimización del uso de tokens, tanto por razones de coste como de rendimiento. Para lograrlo, las funciones Lambda están diseñadas para devolver únicamente los datos estrictamente necesarios para que Bedrock pueda generar una respuesta precisa y contextualizada. Por ejemplo, cuando se consulta por un usuario concreto, se devuelven sus activos y ubicación, pero no se incluye información de otros usuarios ni de activos no relacionados. Esta lógica se ha mantenido para las funciones de tipo individual (*consulta\_por\_activo*, *consulta\_por\_usuario* y *consulta\_por\_ubicacion*), mientras que la función *consulta\_general* (la más costosa en términos de tokens) solo se invoca en situaciones donde no es posible acotar la búsqueda. Este diseño permite controlar de forma precisa el volumen de datos que se envían al modelo, lo que repercute directamente en un menor consumo de recursos y tiempos de respuesta reducidos.

### 3.4.4 Gestión de la memoria conversacional

Aunque cada llamada a Bedrock es completamente independiente, el chatbot desarrollado implementa un mecanismo de memoria conversacional manual que permite mantener la coherencia entre preguntas encadenadas. Esta memoria no es nativa del modelo, sino que se gestiona directamente desde el cliente local `chatbot_client.py` y se pasa como contexto en cada invocación del modelo.

Se han definido dos tipos de memoria, cada una adaptada al rol que asume Bedrock en cada momento:

- **Memoria del intérprete:** utilizada por el modelo en su primera función, la de interpretar la consulta del usuario y decidir qué función Lambda debe ejecutarse. En este caso, se mantiene únicamente la consulta actual y la anterior. Esta estrategia permite interpretar correctamente preguntas encadenadas o que hacen referencia al turno anterior.

*Ejemplo:*

Consulta 1: “¿A qué departamento pertenece Cristina Pérez?”

Consulta 2: “¿Qué activos tiene asignados?”

- **Memoria del agente:** utilizada en la segunda llamada a Bedrock, donde el modelo actúa como agente conversacional. Aquí se conserva el historial completo de la conversación (consultas y respuestas), lo que permite generar interacciones más naturales y coherentes con el flujo del diálogo. Sin embargo, esta memoria no incluye los datos devueltos por las funciones Lambda en llamadas anteriores, por lo que Bedrock no conserva memoria sobre información técnica previamente extraída de la base de datos si no se le vuelve a proporcionar. Esta estrategia evita duplicidades innecesarias y también contribuye a reducir significativamente el consumo de tokens en cada llamada.

Esta gestión explícita del contexto ha permitido mantener una experiencia conversacional satisfactoria sin necesidad de almacenamiento persistente, y con un control total sobre la información compartida en cada etapa.

### 3.4.5 Interfaz de consola y autenticación mediante AWS CLI

En cuanto a la interfaz, se ha optado por implementar el cliente del chatbot como una aplicación local en línea de comandos (cmd). Aunque en un principio se consideró desplegar esta funcionalidad como una función Lambda en la nube, finalmente esta opción fue descartada por varias razones técnicas:

- Habría sido necesario implementar un sistema adicional para almacenar el historial conversacional y mantener el estado entre sesiones.
- La autenticación de usuarios requeriría una gestión personalizada de sesiones, tokens y permisos.
- El coste y la complejidad del despliegue habrían aumentado significativamente sin aportar valor funcional adicional.

Además, el entorno local ofrece una ventaja adicional en términos de autenticación. Al estar basado en AWS CLI, el cliente solo puede ejecutarse en sistemas donde el usuario haya configurado previamente sus credenciales IAM mediante el comando *aws configure*. Esto actúa como un mecanismo de autenticación implícito, ya que si el usuario no ha introducido sus claves de acceso, no podrá acceder al sistema ni ejecutar las consultas. De este modo, se evita implementar un sistema de login adicional, ya que el acceso queda restringido de forma automática a usuarios autenticados por la CLI.

Por todo ello, el entorno en línea de comandos resulta una solución simple, eficaz y coherente con los objetivos del proyecto. El script `chatbot_client.py`, que encapsula toda esta lógica, puede encontrarse en el **anexo** del documento.

## 3.5 Subsistema de gestión de recursos (API REST)

El sistema desarrollado contempla un segundo subsistema, complementario al chatbot, que permite realizar operaciones estructuradas sobre los datos del sistema. Este subsistema tiene como objetivo ofrecer una interfaz estructurada para realizar operaciones administrativas sobre los activos tecnológicos, usuarios y ubicaciones almacenados en el sistema. A diferencia del chatbot, que está orientado a la consulta mediante lenguaje natural, este subsistema se centra en proporcionar un punto de entrada uniforme y controlado para modificar y registrar información crítica.

### 3.5.1 Diseño general del subsistema

El subsistema se ha construido como una API REST sin servidor, desplegada íntegramente sobre servicios de AWS. El componente central de esta arquitectura es Amazon API Gateway, que actúa como punto de entrada para las peticiones externas. Cada endpoint de la API está asociado a una función AWS Lambda, la cual implementa la lógica de negocio necesaria para ejecutar las operaciones CRUD (crear, leer, actualizar y eliminar) sobre los datos.

En concreto, se han definido tres rutas principales:

- /usuario para la gestión de usuarios
- /activo para el control de activos tecnológicos y sus subtipos (hardware, licencias y líneas telefónicas)
- /ubicacion para administrar ubicaciones físicas

Estas rutas están enlazadas a las funciones *gestion\_usuario.py*, *gestion\_activo.py* y *gestion\_ubicacion.py*, respectivamente. Tal y como se muestra en la **figura 26**, las funciones Lambda han sido integradas en la API durante su proceso de creación.

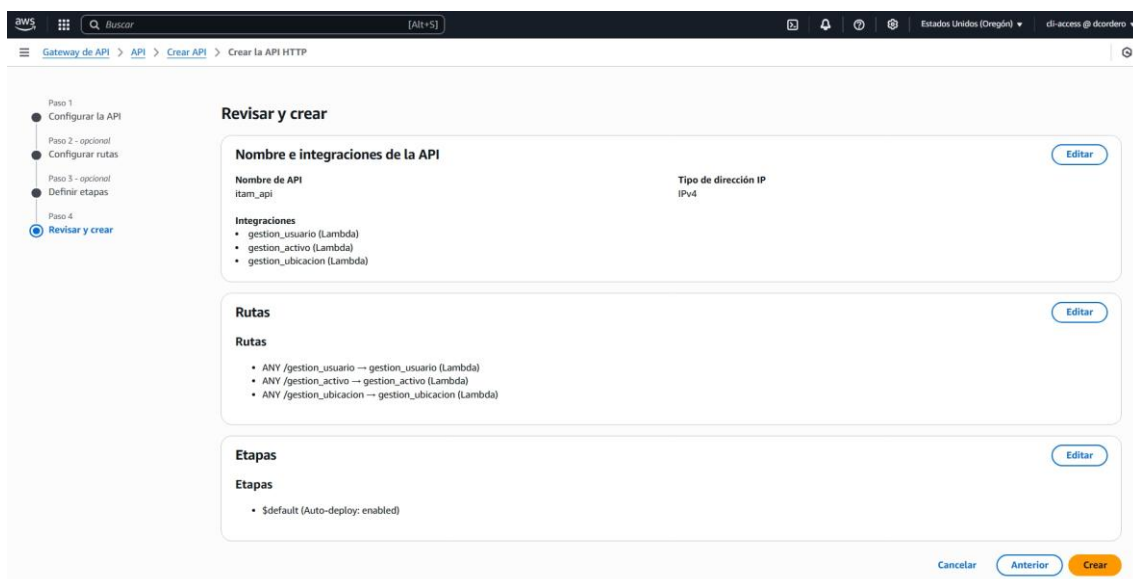


Figura 26: Resumen de creación de la API con sus rutas e integraciones

Durante la fase de diseño se evaluaron las dos opciones disponibles en Amazon API Gateway: REST API y HTTP API. Finalmente se optó por **HTTP API** debido a las siguientes ventajas:

- **Simplicidad en la configuración:** HTTP API ofrece una interfaz más simple y directa, especialmente adecuada para aplicaciones ligeras con lógica desacoplada como esta.
- **Menor coste:** el coste es considerablemente inferior en HTTP API, lo que permite operar dentro del AWS Free Tier durante el desarrollo y las pruebas.
- **Baja latencia:** HTTP API proporciona mejor rendimiento en comparación con REST API en entornos sin autenticación compleja ni transformaciones avanzadas.

El diseño de la API no incluye un frontend asociado, ya que su finalidad es facilitar la administración y evolución del sistema, siendo accesible desde herramientas de prueba como Postman o desde futuros clientes web o móviles que puedan desarrollarse a posteriori. Esta separación permite mantener una arquitectura limpia, desacoplada y fácilmente ampliable.

### 3.5.2 Control de acceso y autenticación

Dado que esta API permite modificar información sensible, se ha integrado un sistema de autenticación basado en **Amazon Cognito**, que actúa como proveedor de identidad. El acceso a cada ruta está restringido mediante un authorizer JWT configurado en API Gateway, el cual valida que cada petición contenga un token válido emitido por Cognito. De este modo, se garantiza que solo los usuarios autenticados puedan invocar las funciones Lambda asociadas.

El proceso comienza con la creación del autorizador, tal como se observa en la **figura 27**. En él se especifica la URL del emisor (issuer) correspondiente al User Pool de Cognito, así como el ID del cliente (audience) autorizado a acceder a la API.

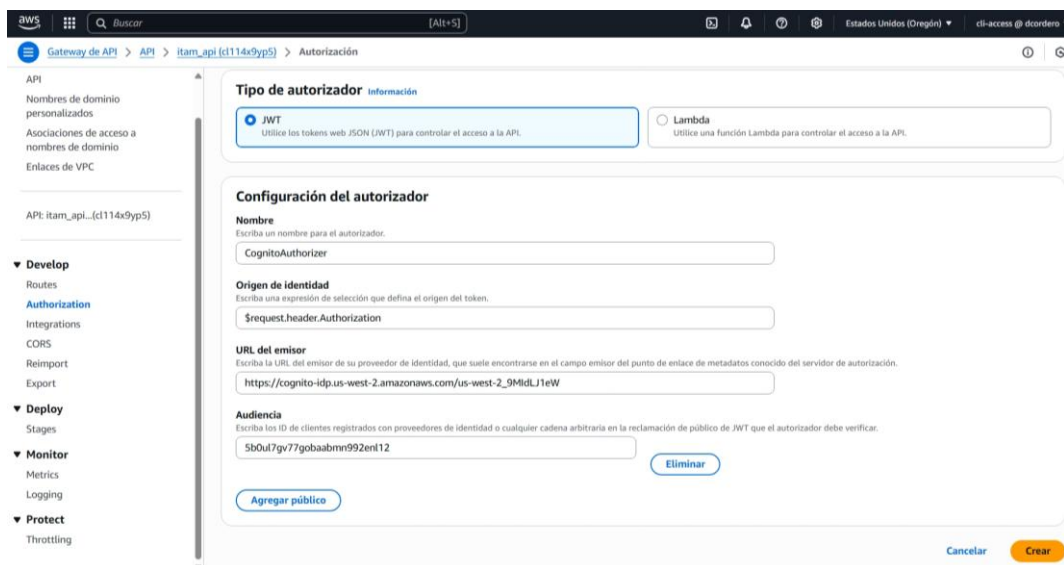


Figura 27: Creación del autorizador JWT en API Gateway

Posteriormente, este autorizador se asocia a cada una de las rutas, como se muestra en la **figura 28**. A partir de ese momento, cualquier intento de acceso a la API que no incluya un token válido será denegado automáticamente con un error 401.

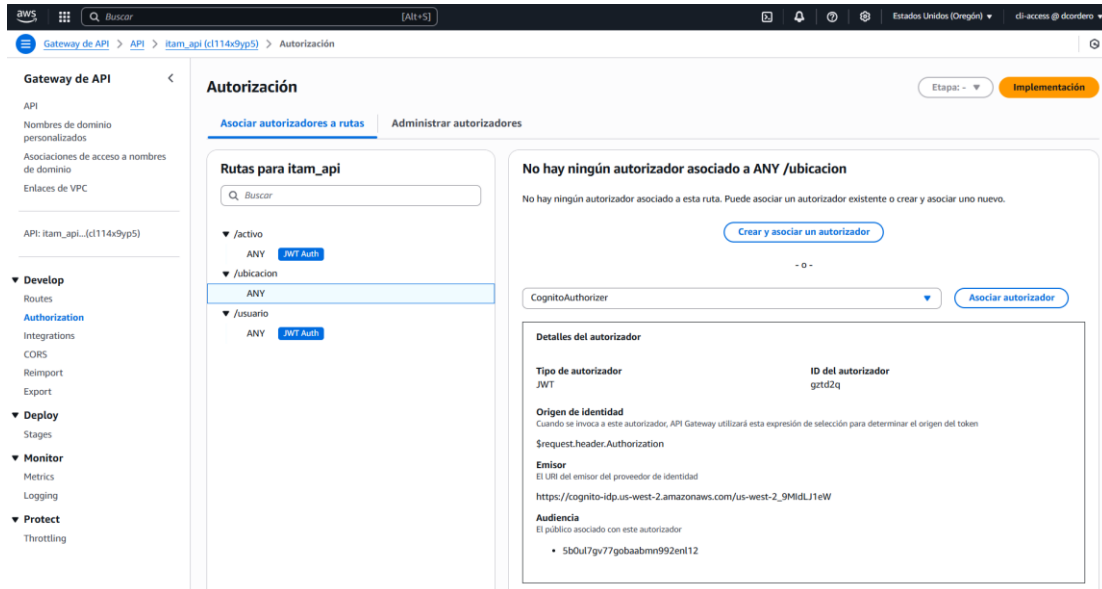


Figura 28: Asociación del autorizador CognitoAuthorizer a los endpoints

El uso de Cognito responde a la necesidad de aplicar un control de acceso robusto y gestionado sin tener que implementar un sistema de login propio. A diferencia del chatbot, que se ejecuta en un entorno autenticado previamente con AWS CLI, este subsistema está diseñado para ser accesible desde cualquier cliente externo, lo que exige un sistema de autenticación explícito y seguro.

### 3.5.3 Interfaz de inicio de sesión y flujo OAuth

Para facilitar el acceso a la API por parte de usuarios externos como Postman, se ha configurado el **Hosted UI de Amazon Cognito**, que proporciona una interfaz de inicio de sesión totalmente gestionada. Esta interfaz se basa en un dominio personalizado y permite que los usuarios introduzcan sus credenciales (correo electrónico y contraseña) para obtener un token JWT válido.

La **figura 29** muestra la configuración de la página de inicio de sesión, donde se establecieron la URL de retorno (`http://localhost`), los scopes permitidos (`openid, email`) y el flujo de autorización implícita (`response_type=token`). La URL de retorno se eligió por ser una dirección local reconocida por Cognito para pruebas y desarrollo, lo que permite recibir el token directamente sin necesidad de un servidor de backend. Esta configuración permite que Postman pueda interceptar automáticamente la redirección a `http://localhost` y extraer el token desde el navegador. En cuanto a los scopes, se seleccionaron `openid` y `email` por ser los mínimos necesarios para identificar al usuario de forma segura y acceder a su dirección de correo electrónico, que es el atributo clave en el sistema. Finalmente, se optó por el flujo de autorización implícita debido a que permite obtener el `access_token` directamente tras el login sin necesidad de realizar un segundo paso de intercambio, lo que resulta más adecuado en entornos donde no se dispone de una aplicación backend que gestione dicho intercambio, como ocurre en este proyecto.

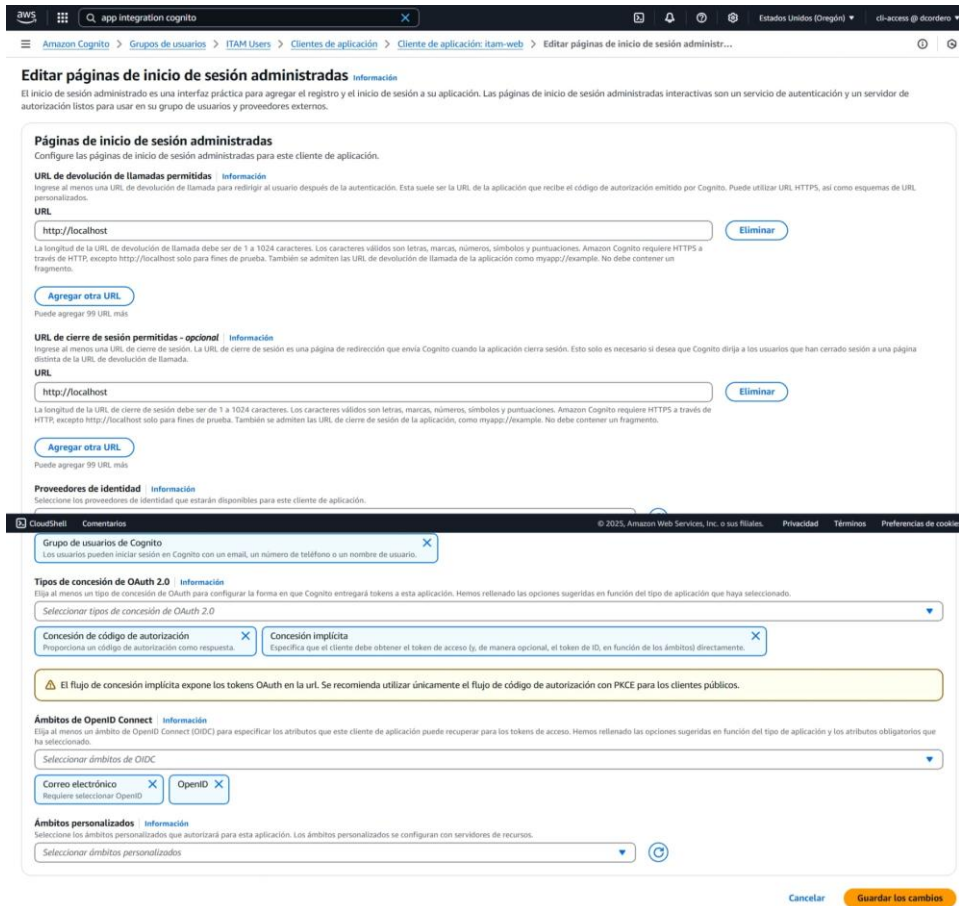


Figura 29: Configuración del Hosted UI de inicio de sesión de AWS Cognito

Cuando un usuario accede a la URL de login generada por Cognito, se le redirige al portal de autenticación, tal y como se muestra en la **figura 30**. Una vez completado el inicio de sesión, el navegador redirige automáticamente al cliente con el token JWT necesario.

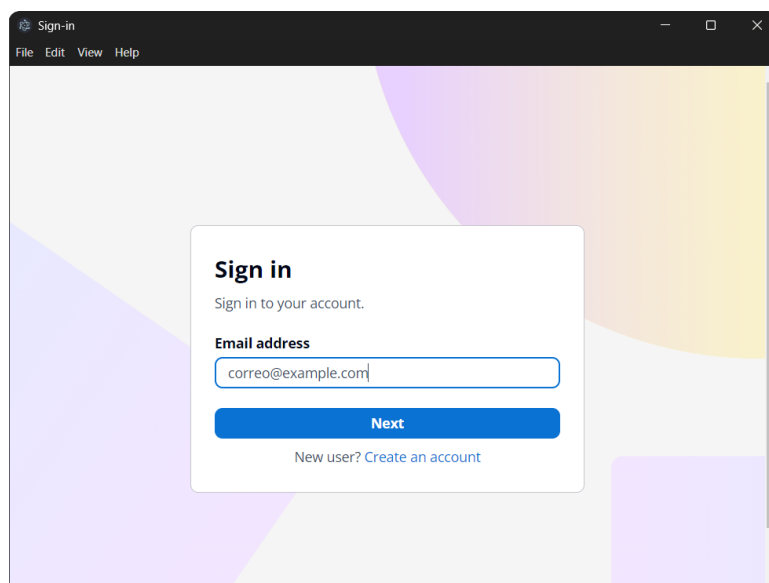


Figura 30: Página de inicio de sesión para la autenticación con Cognito

### 3.5.4 Verificación del acceso autenticado

Para realizar pruebas sobre los endpoints protegidos, se utilizó **Postman**, aprovechando su integración con OAuth 2.0. En la pestaña de autorización, que se muestra en la **figura 31**, se configuró el flujo Implícito, con el Client ID, la URL de autorización y los scopes definidos. Gracias a esto, Postman es capaz de abrir la interfaz de inicio de sesión, recibir el token tras el login y aplicarlo automáticamente a todas las peticiones posteriores mediante el encabezado Authorization: Bearer.

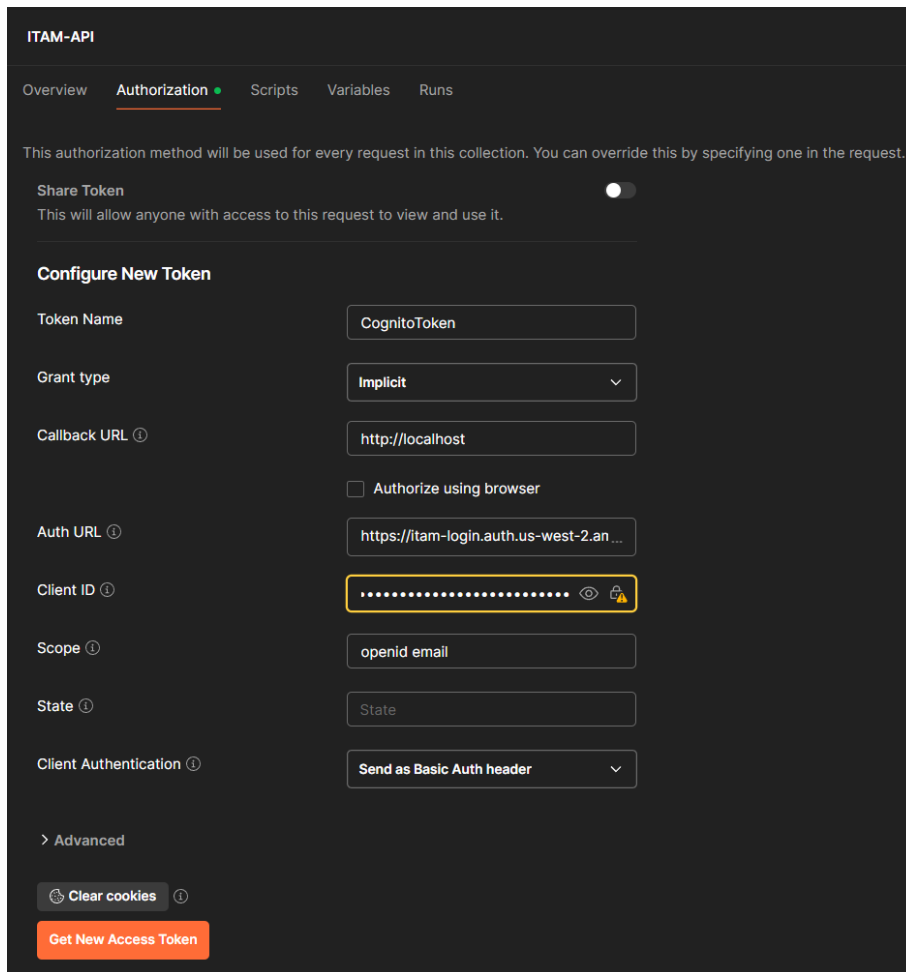


Figura 31: Configuración de autorización OAuth en Postman

Además, se ha validado que al expirar el token (o revocar la sesión en Cognito), la API responde con un error “401 Unauthorized”, lo que garantiza que no se permite el acceso sin una autenticación válida.

Este enfoque permite realizar pruebas completamente realistas del sistema sin necesidad de desarrollar un frontend propio. Además, deja abierta la posibilidad de utilizar la API en futuras aplicaciones cliente, gracias a que el flujo de autenticación es estándar y bien documentado.

## 4 Resultados y conclusiones

Tras haber analizado en detalle la arquitectura, diseño y componentes del sistema en el capítulo anterior, en este apartado se presentan los resultados obtenidos mediante una serie de pruebas funcionales. El objetivo es validar que tanto el chatbot inteligente como la API REST de gestión cumplen con los requisitos establecidos, ofreciendo respuestas precisas y un comportamiento coherente frente a distintos tipos de consulta y operación.

### 4.1 Carga de datos

Antes de realizar las pruebas funcionales sobre los subsistemas desarrollados, fue necesario poblar la base de datos con un conjunto realista de datos de ejemplo. Para ello se creó el script *consulta\_carga\_datos.sql*, que contiene todas las instrucciones necesarias para insertar ubicaciones, usuarios y activos tecnológicos, tanto hardware como licencias y líneas telefónicas, en sus respectivas tablas.

El objetivo de este script es simular un entorno empresarial básico que permita validar el comportamiento del sistema en distintos escenarios. Se han definido tres ubicaciones físicas principales (Madrid, Málaga y Barcelona), cada una con un número equilibrado de usuarios asignados, representando diversos departamentos y perfiles profesionales. Cada usuario dispone al menos de:

- Un equipo hardware (ordenador) asignado y ubicado físicamente en su oficina correspondiente.
- Una línea telefónica vinculada a su número personal.
- Al menos una licencia de software activa.

Además, se han incluido algunos activos sin asignar (líneas y licencias en estado *disponible*) para facilitar pruebas de inserción, reasignación y verificación de disponibilidad.

La estructura del script sigue una lógica ordenada por bloques:

1. Inserción de registros en la tabla *Ubicacion*, con sus atributos de dirección.
2. Creación de usuarios en la tabla *Usuario*, incluyendo claves foráneas hacia ubicaciones.
3. Inserción de activos de tipo hardware, *linea\_telefonica* y licencia en la tabla *Activo*, con los datos de proveedor, coste, estado y propietario.
4. Registro de los atributos específicos en las tablas de subtipo: *Hardware*, *Linea\_Telefonica* y *Licencia*.

A continuación, se muestra el contenido completo del script de carga de datos utilizado:

```

USE itam_db;

-- 1. Insertar Ubicaciones
INSERT INTO Ubicacion (nombre, pais, calle, numero, localidad, provincia, codigo_postal)
VALUES
('Oficina Madrid', 'España', 'Calle Alcalá', 45, 'Madrid', 'Madrid', '28014'),
('Oficina Málaga', 'España', 'Avda Andalucía', 12, 'Málaga', 'Málaga', '29007'),
('Oficina Barcelona', 'España', 'Gran Vía', 85, 'Barcelona', 'Barcelona', '08013');

-- 2. Insertar Usuarios
INSERT INTO Usuario (email, nombre_completo, telefono, estado, tipo_usuario, puesto, departamento, ubicacion)
VALUES
('ana.garcia@example.com', 'Ana García Pérez', '60011222', 'activo', 'interno', 'Administradora Sistemas', 'IT', 'Oficina Madrid'),
('carlos.lopez@example.com', 'Carlos López Díaz', '600333444', 'activo', 'interno', 'Técnico de Soporte', 'IT', 'Oficina Madrid'),
('laura.martin@example.com', 'Laura Martín Rutz', '600555666', 'activo', 'externo', 'Consultora', 'Consultoría', 'Oficina Málaga'),
('jose.sanchez@example.com', 'José Sánchez Torres', '600777888', 'activo', 'interno', 'Programador', 'Desarrollo', 'Oficina Málaga'),
('maria.fernandez@example.com', 'María Fernández Morales', '600888999', 'activo', 'interno', 'Arquitecta de Software', 'Desarrollo', 'Oficina Madrid'),
('david.ramirez@example.com', 'David Ramírez Gil', '600999000', 'activo', 'externo', 'Analista de Datos', 'Datos', 'Oficina Málaga'),
('elena.gomez@example.com', 'Elena Gómez Santos', '60111222', 'activo', 'interno', 'DevOps Engineer', 'Infraestructura', 'Oficina Barcelona'),
('francisco.ruiz@example.com', 'Francisco Ruiz Cano', '601333444', 'activo', 'interno', 'Gestor de Proyecto', 'PMO', 'Oficina Barcelona'),
('lucia.molina@example.com', 'Lucía Molina Pascual', '601555666', 'activo', 'interno', 'Responsable de IT', 'IT', 'Oficina Barcelona'),
('manuel.navarro@example.com', 'Manuel Navarro Ortega', '601777888', 'activo', 'externo', 'QA Tester', 'Calidad', 'Oficina Barcelona');

-- 3. Insertar Activos
INSERT INTO Activo (id_activo, tipo, estado, coste, recurrente, fecha_adquisicion, nombre_proveedor, email_propietario) VALUES
('ORD001', 'hardware', 'asignado', 1050, 'no', '2024-01-01', 'Dell', 'ana.garcia@example.com'),
('ORD002', 'hardware', 'asignado', 720, 'no', '2024-01-01', 'HP', 'carlos.lopez@example.com'),
('ORD003', 'hardware', 'asignado', 700, 'no', '2024-01-01', 'Dell', 'laura.martin@example.com'),
('ORD004', 'hardware', 'asignado', 829, 'no', '2024-01-01', 'Lenovo', 'jose.sanchez@example.com'),
('ORD005', 'hardware', 'asignado', 939, 'no', '2024-01-01', 'HP', 'maria.fernandez@example.com'),
('ORD006', 'hardware', 'asignado', 1000, 'no', '2024-01-01', 'Dell', 'david.ramirez@example.com'),
('ORD007', 'hardware', 'asignado', 849, 'no', '2024-01-01', 'HP', 'elena.gomez@example.com'),
('ORD008', 'hardware', 'asignado', 795, 'no', '2024-01-01', 'Lenovo', 'francisco.ruiz@example.com'),
('ORD009', 'hardware', 'asignado', 919, 'no', '2024-01-01', 'HP', 'lucia.molina@example.com'),
('ORD010', 'hardware', 'asignado', 1230, 'no', '2024-01-01', 'Lenovo', 'manuel.navarro@example.com'),
('TLF001', 'linea_telefonica', 'asignado', 20, 'si', '2024-01-05', 'Movistar', 'ana.garcia@example.com'),
('TLF002', 'linea_telefonica', 'asignado', 20, 'si', '2024-01-05', 'Vodafone', 'carlos.lopez@example.com'),
('TLF003', 'linea_telefonica', 'asignado', 20, 'si', '2024-01-05', 'Orange', 'laura.martin@example.com'),
('TLF004', 'linea_telefonica', 'asignado', 20, 'si', '2024-01-05', 'Yoigo', 'jose.sanchez@example.com'),
('TLF005', 'linea_telefonica', 'asignado', 20, 'si', '2024-01-05', 'Movistar', 'maria.fernandez@example.com'),
('TLF006', 'linea_telefonica', 'asignado', 20, 'si', '2024-01-05', 'Vodafone', 'david.ramirez@example.com'),
('TLF007', 'linea_telefonica', 'asignado', 20, 'si', '2024-01-05', 'Orange', 'elena.gomez@example.com'),
('TLF008', 'linea_telefonica', 'asignado', 20, 'si', '2024-01-05', 'Yoigo', 'francisco.ruiz@example.com'),
('TLF009', 'linea_telefonica', 'asignado', 20, 'si', '2024-01-05', 'Movistar', 'lucia.molina@example.com'),
('TLF010', 'linea_telefonica', 'asignado', 20, 'si', '2024-01-05', 'Vodafone', 'manuel.navarro@example.com'),
('TLF011', 'linea_telefonica', 'disponible', 20, 'si', '2024-01-05', 'Movistar', null),
('TLF012', 'linea_telefonica', 'disponible', 20, 'si', '2024-01-05', 'Vodafone', null),
('LIC001', 'licencia', 'asignado', 20, 'si', '2024-01-01', 'Microsoft', 'ana.garcia@example.com'),
('LIC002', 'licencia', 'asignado', 50, 'si', '2024-01-01', 'Adobe', 'carlos.lopez@example.com'),
('LIC003', 'licencia', 'asignado', 40, 'si', '2024-01-01', 'JetBrains', 'laura.martin@example.com'),
('LIC004', 'licencia', 'asignado', 20, 'si', '2024-01-01', 'Microsoft', 'jose.sanchez@example.com'),
('LIC005', 'licencia', 'asignado', 20, 'si', '2024-01-01', 'Microsoft', 'maria.fernandez@example.com'),
('LIC006', 'licencia', 'asignado', 15, 'si', '2024-01-01', 'Atlassian', 'david.ramirez@example.com'),
('LIC007', 'licencia', 'asignado', 20, 'si', '2024-01-01', 'Microsoft', 'elena.gomez@example.com'),
('LIC008', 'licencia', 'asignado', 20, 'si', '2024-01-01', 'Microsoft', 'francisco.ruiz@example.com'),
('LIC009', 'licencia', 'asignado', 20, 'si', '2024-01-01', 'Microsoft', 'lucia.molina@example.com'),
('LIC010', 'licencia', 'asignado', 20, 'si', '2024-01-01', 'Microsoft', 'manuel.navarro@example.com'),
('LIC011', 'licencia', 'disponible', 20, 'si', '2024-01-01', 'Microsoft', null),
('LIC012', 'licencia', 'disponible', 20, 'si', '2024-01-01', 'Microsoft', null);

-- 4. Insertar Subtipos

-- Hardware
INSERT INTO Hardware (id_activo, n_serie, modelo, marca, tipo, ubicacion_nombre) VALUES
('ORD001', 'SN001', 'Latitude 5500', 'Dell', 'ordenador', 'Oficina Madrid'),
('ORD002', 'SN002', 'EliteBook 840', 'HP', 'ordenador', 'Oficina Madrid'),
('ORD003', 'SN003', 'ThinkPad E14', 'Lenovo', 'ordenador', 'Oficina Málaga'),
('ORD004', 'SN004', 'ProBook 450', 'HP', 'ordenador', 'Oficina Málaga'),
('ORD005', 'SN005', 'XPS 13', 'Dell', 'ordenador', 'Oficina Madrid'),
('ORD006', 'SN006', 'Spectre x360', 'HP', 'ordenador', 'Oficina Málaga'),
('ORD007', 'SN007', 'IdeaPad 5', 'Lenovo', 'ordenador', 'Oficina Barcelona'),
('ORD008', 'SN008', 'Pavilion 15', 'HP', 'ordenador', 'Oficina Barcelona'),
('ORD009', 'SN009', 'ThinkCentre M720', 'Lenovo', 'ordenador', 'Oficina Barcelona'),
('ORD010', 'SN010', 'EliteBook 830', 'HP', 'ordenador', 'Oficina Barcelona');

-- Líneas Telefónicas
INSERT INTO Linea_Telefonica (id_activo, numero_tlf, operadora, plan, pin, puk, sim, id_hardware) VALUES
('TLF001', '60011222', 'Movistar', 'Pro Max', '1111', '111111', 'SIM001', 'ORD001'),
('TLF002', '600333444', 'Vodafone', 'Full Data', '2222', '222222', 'SIM002', 'ORD002'),
('TLF003', '600555666', 'Orange', 'Smart Plan', '3333', '333333', 'SIM003', 'ORD003'),
('TLF004', '600777888', 'Yoigo', 'Top Plan', '4444', '444444', 'SIM004', 'ORD004'),
('TLF005', '600888999', 'Movistar', 'Pro Max', '5555', '555555', 'SIM005', 'ORD005'),
('TLF006', '600999000', 'Vodafone', 'Full Data', '6666', '666666', 'SIM006', 'ORD006'),
('TLF007', '60111222', 'Orange', 'Smart Plan', '7777', '777777', 'SIM007', 'ORD007'),
('TLF008', '601333444', 'Yoigo', 'Top Plan', '8888', '888888', 'SIM008', 'ORD008'),
('TLF009', '601555666', 'Movistar', 'Pro Max', '9999', '999999', 'SIM009', 'ORD009'),
('TLF010', '601777888', 'Vodafone', 'Full Data', '0000', '000000', 'SIM010', 'ORD010'),
('TLF011', '605911666', 'Movistar', 'Pro Max', '9819', '997199', 'SIM011', null),
('TLF012', '601879178', 'Vodafone', 'Full Data', '0072', '0067200', 'SIM012', null);

-- Licencias
INSERT INTO Licencia (id_activo, nombre_licencia, producto, numero_licencia) VALUES
('LIC001', 'Microsoft Office 365 Premium', 'Office', '0365-001'),
('LIC002', 'Adobe Pro', 'Adobe', 'ADOBE-002'),
('LIC003', 'IntelliJ IDEA', 'JetBrains', 'JB-003'),
('LIC004', 'Microsoft Office 365 Premium', 'Office', '0365-004'),
('LIC005', 'Microsoft Office 365 Premium', 'Office', '0365-005'),
('LIC006', 'Jira', 'Atlassian', 'JIRA-006'),
('LIC007', 'Microsoft Office 365 Premium', 'Office', '0365-007'),
('LIC008', 'Microsoft Office 365 Premium', 'Office', '0365-008'),
('LIC009', 'Microsoft Office 365 Premium', 'Office', '0365-009'),
('LIC010', 'Microsoft Office 365 Premium', 'Office', '0365-010'),
('LIC011', 'Microsoft Office 365 Premium', 'Office', '0365-011'),
('LIC012', 'Microsoft Office 365 Premium', 'Office', '0365-012');

```

Figura 32: Script de carga de datos utilizado para la realización de las pruebas del sistema

## 4.2 Pruebas del subsistema de consulta inteligente (Chatbot AI)

### 4.2.1 Pruebas individuales de las funciones Lambda

Antes de evaluar el comportamiento del chatbot como sistema conversacional, es fundamental comprobar que las funciones Lambda sobre las que se apoya funcionan correctamente de forma aislada. A continuación, se presentan las pruebas realizadas sobre cada una de las cuatro funciones disponibles:

#### Prueba 1 – Función consulta\_por\_usuario.py

- Evento JSON enviado:

```
{
  "identificador": "David Ramírez Gil"
}    o    {
  "identificador": "david.ramirez@example.com"
}
```

- Resultado esperado:
  - Devuelve los datos del usuario (nombre, email, teléfono...).
  - Incluye los datos de los activos ORD006, TLF006, LIC006 correctamente.
  - Incluye los datos de la ubicación Oficina Málaga.
- Resultado obtenido:

```
statusCode: 200
body: {
  "Usuario": [
    {
      "email": "david.ramirez@example.com",
      "nombre_completo": "David Ramírez Gil",
      "telefono": "600999000",
      "estado": "activo",
      "tipo_usuario": "externo",
      "puesto": "Analista de Datos",
      "departamento": "Datos",
      "ubicacion": "Oficina Málaga"
    }
  ],
  "Ubicacion": [
    {
      "nombre": "Oficina Málaga",
      "pais": "España",
      "calle": "Avda Andalucía",
      "numero": 12,
      "localidad": "Málaga",
      "provincia": "Málaga",
      "codigo_postal": "29007",
      "n_empleados": 3
    }
  ],
  "Activo_Hardware": [
    {
      "id_activo": "ORD006",
      "tipo_activo": "hardware",
      "estado": "asignado",
      "coste": "1000.00",
      "recurrente": "no",
      "fecha_adquisicion": "2024-01-01",
      "nombre_proveedor": "Dell",
      "email_propietario": "david.ramirez@example.com",
      "n_serie": "SN006",
      "modelo": "Spectre x360",
      "marca": "HP",
      "tipo_hardware": "ordenador",
      "ubicacion_nombre": "Oficina Málaga"
    }
  ],
  "Activo_Licencia": [
    {
      "id_activo": "LIC006",
      "tipo_activo": "licencia",
      "estado": "asignado",
      "coste": "15.00",
      "recurrente": "si",
      "fecha_adquisicion": "2024-01-01",
      "nombre_proveedor": "Atlassian",
      "email_propietario": "david.ramirez@example.com",
      "nombre_licencia": "Jira",
      "producto": "Atlassian",
      "numero_licencia": "JIRA-006"
    }
  ],
  "Activo_Linea_Telefonica": [
    {
      "id_activo": "TLF006",
      "tipo_activo": "linea_telefonica",
      "estado": "asignado",
      "coste": "20.00",
      "recurrente": "si",
      "fecha_adquisicion": "2024-01-05",
      "nombre_proveedor": "Vodafone",
      "email_propietario": "david.ramirez@example.com",
      "numero_tlf": "600999000",
      "operadora": "Vodafone",
      "plan": "Full Data",
      "pin": "6666",
      "puk": "666666",
      "sim": "SIM006",
      "id_hardware": "ORD006"
    }
  ]
}
```

Figura 33: Resultado obtenido tras la ejecución de la Prueba 1

## Prueba 2 – Función consulta\_por\_activo.py

- Evento JSON enviado:

```
{ "identificador": "SN005" } o { "identificador": "ORD005" }
```

- Resultado esperado:
  - Identifica correctamente el tipo de activo a partir del id\_activo o del identificador indirecto del subtipo.
  - Devuelve todos los atributos del activo ORD005 del tipo hardware.
  - Devuelve también los datos del propietario del activo y su ubicación asociada.
- Resultado obtenido:

```
statusCode: 200
body: {
  "Activo_Hardware": [
    {
      "id_activo": "ORD005",
      "tipo_activo": "hardware",
      "estado": "asignado",
      "coste": "939.00",
      "recurrente": "no",
      "fecha_adquisicion": "2024-01-01",
      "nombre_proveedor": "HP",
      "email_propietario": "maria.fernandez@example.com",
      "n_serie": "SN005",
      "modelo": "XPS 13",
      "marca": "Dell",
      "tipo_hardware": "ordenador",
      "ubicacion_nombre": "Oficina Madrid"
    }
  ],
  "Usuario": [
    {
      "email": "maria.fernandez@example.com",
      "nombre_completo": "María Fernández Morales",
      "telefono": "600888999",
      "estado": "activo",
      "tipo_usuario": "interno",
      "puesto": "Arquitecta de Software",
      "departamento": "Desarrollo",
      "ubicacion": "Oficina Madrid"
    }
  ],
  "Ubicacion": [
    {
      "nombre": "Oficina Madrid",
      "pais": "España",
      "calle": "Calle Alcalá",
      "numero": "45",
      "localidad": "Madrid",
      "provincia": "Madrid",
      "codigo_postal": "28014",
      "n_empleados": 3
    }
  ]
}
```

Figura 34: Resultado obtenido tras la ejecución de la Prueba 2

### Prueba 3 – Función consulta\_por\_ubicacion.py

- Evento JSON enviado:

```
{
  "identificador": " Oficina Barcelona"
}
```

- Resultado esperado:

- Devuelve correctamente la información de todos los usuarios que pertenecen a la oficina de Barcelona.
- Devuelve toda la información los activos de hardware ubicados físicamente en esta ubicación.
- Devuelve toda la información de las licencias y líneas telefónicas asignadas a los usuarios pertenecientes a dicha localización.

- Resultado obtenido:

```
statusCode: 200
body: {
  "Ubicacion": [
    {
      "nombre": "Oficina Barcelona",
      "pais": "España",
      "calle": "Gran Vía",
      "numero": 85,
      "localidad": "Barcelona",
      "provincia": "Barcelona",
      "codigo_postal": "08013",
      "n_empleados": 4
    }
  ],
  "Usuario": [
    {
      "email": "elena.gomez@example.com",
      "nombre_completo": "Elena Gómez Santos",
      "telefono": "601111222",
      "estado": "activo",
      "tipo_usuario": "interno",
      "puesto": "DevOps Engineer",
      "departamento": "Infraestructura",
      "ubicacion": "Oficina Barcelona"
    },
    {
      "email": "francisco.ruiz@example.com",
      "nombre_completo": "Francisco Ruiz Cano",
      "telefono": "601333444",
      "estado": "activo",
      "tipo_usuario": "interno",
      "puesto": "Gestor de Proyecto",
      "departamento": "PMO",
      "ubicacion": "Oficina Barcelona"
    },
    {
      "email": "lucia.molina@example.com",
      "nombre_completo": "Lucía Molina Pascual",
      "telefono": "601555666",
      "estado": "activo",
      "tipo_usuario": "interno",
      "puesto": "Responsable de IT",
      "departamento": "IT",
      "ubicacion": "Oficina Barcelona"
    },
    {
      "email": "manuel.navarro@example.com",
      "nombre_completo": "Manuel Navarro Ortega",
      "telefono": "601777888",
      "estado": "activo",
      "tipo_usuario": "externo",
      "puesto": "QA Tester",
      "departamento": "Calidad",
      "ubicacion": "Oficina Barcelona"
    }
  ]
}
```

```
"Activo_Hardware": [
  {
    "id_activo": "ORD007",
    "tipo_activo": "hardware",
    "estado": "asignado",
    "coste": "849.00",
    "recurrente": "no",
    "fecha_adquisicion": "2024-01-01",
    "nombre_proveedor": "HP",
    "email_propietario": "elena.gomez@example.com",
    "n_serie": "SN007",
    "modelo": "IdeaPad 5",
    "marca": "Lenovo",
    "tipo_hardware": "ordenador",
    "ubicacion_nombre": "Oficina Barcelona"
  },
  {
    "id_activo": "ORD008",
    "tipo_activo": "hardware",
    "estado": "asignado",
    "coste": "795.00",
    "recurrente": "no",
    "fecha_adquisicion": "2024-01-01",
    "nombre_proveedor": "Lenovo",
    "email_propietario": "francisco.ruiz@example.com",
    "n_serie": "SN008",
    "modelo": "Pavilion 15",
    "marca": "HP",
    "tipo_hardware": "ordenador",
    "ubicacion_nombre": "Oficina Barcelona"
  },
  {
    "id_activo": "ORD009",
    "tipo_activo": "hardware",
    "estado": "asignado",
    "coste": "919.00",
    "recurrente": "no",
    "fecha_adquisicion": "2024-01-01",
    "nombre_proveedor": "HP",
    "email_propietario": "lucia.molina@example.com",
    "n_serie": "SN009",
    "modelo": "ThinkCentre M720",
    "marca": "Lenovo",
    "tipo_hardware": "ordenador",
    "ubicacion_nombre": "Oficina Barcelona"
  },
  {
    "id_activo": "ORD010",
    "tipo_activo": "hardware",
    "estado": "asignado",
    "coste": "1230.00",
    "recurrente": "no",
    "fecha_adquisicion": "2024-01-01",
    "nombre_proveedor": "Lenovo",
    "email_propietario": "manuel.navarro@example.com",
    "n_serie": "SN010",
    "modelo": "EliteBook 830",
    "marca": "HP",
    "tipo_hardware": "ordenador",
    "ubicacion_nombre": "Oficina Barcelona"
  }
]
```

```

"Activo_Licencia": [
  {
    "id_activo": "LIC007",
    "tipo_activo": "licencia",
    "estado": "asignado",
    "coste": "20.00",
    "recurrente": "si",
    "fecha_adquisicion": "2024-01-01",
    "nombre_proveedor": "Microsoft",
    "email_propietario": "elena.gomez@example.com",
    "nombre_licencia": "Microsoft Office 365 Premium",
    "producto": "Office",
    "numero_licencia": "0365-007"
  },
  {
    "id_activo": "LIC008",
    "tipo_activo": "licencia",
    "estado": "asignado",
    "coste": "20.00",
    "recurrente": "si",
    "fecha_adquisicion": "2024-01-01",
    "nombre_proveedor": "Microsoft",
    "email_propietario": "francisco.ruiz@example.com",
    "nombre_licencia": "Microsoft Office 365 Premium",
    "producto": "Office",
    "numero_licencia": "0365-008"
  },
  {
    "id_activo": "LIC009",
    "tipo_activo": "licencia",
    "estado": "asignado",
    "coste": "20.00",
    "recurrente": "si",
    "fecha_adquisicion": "2024-01-01",
    "nombre_proveedor": "Microsoft",
    "email_propietario": "lucia.molina@example.com",
    "nombre_licencia": "Microsoft Office 365 Premium",
    "producto": "Office",
    "numero_licencia": "0365-009"
  },
  {
    "id_activo": "LIC010",
    "tipo_activo": "licencia",
    "estado": "asignado",
    "coste": "20.00",
    "recurrente": "si",
    "fecha_adquisicion": "2024-01-01",
    "nombre_proveedor": "Microsoft",
    "email_propietario": "manuel.navarro@example.com",
    "nombre_licencia": "Microsoft Office 365 Premium",
    "producto": "Office",
    "numero_licencia": "0365-010"
  }
],
"Activo_Linea_Telefonica": [
  {
    "id_activo": "TLF007",
    "tipo_activo": "linea_telefonica",
    "estado": "asignado",
    "coste": "20.00",
    "recurrente": "si",
    "fecha_adquisicion": "2024-01-05",
    "nombre_proveedor": "Orange",
    "email_propietario": "elena.gomez@example.com",
    "numero_tlf": "60111222",
    "operadora": "Orange",
    "plan": "Smart Plan",
    "pin": "7777",
    "puk": "777777",
    "sim": "SIM007",
    "id_hardware": "ORD007"
  },
  {
    "id_activo": "TLF008",
    "tipo_activo": "linea_telefonica",
    "estado": "asignado",
    "coste": "20.00",
    "recurrente": "si",
    "fecha_adquisicion": "2024-01-05",
    "nombre_proveedor": "Yoigo",
    "email_propietario": "francisco.ruiz@example.com",
    "numero_tlf": "601333444",
    "operadora": "Yoigo",
    "plan": "Top Plan",
    "pin": "8888",
    "puk": "888888",
    "sim": "SIM008",
    "id_hardware": "ORD008"
  },
  {
    "id_activo": "TLF009",
    "tipo_activo": "linea_telefonica",
    "estado": "asignado",
    "coste": "20.00",
    "recurrente": "si",
    "fecha_adquisicion": "2024-01-05",
    "nombre_proveedor": "Movistar",
    "email_propietario": "lucia.molina@example.com",
    "numero_tlf": "601555666",
    "operadora": "Movistar",
    "plan": "Pro Max",
    "pin": "9999",
    "puk": "999999",
    "sim": "SIM009",
    "id_hardware": "ORD009"
  },
  {
    "id_activo": "TLF010",
    "tipo_activo": "linea_telefonica",
    "estado": "asignado",
    "coste": "20.00",
    "recurrente": "si",
    "fecha_adquisicion": "2024-01-05",
    "nombre_proveedor": "Vodafone",
    "email_propietario": "manuel.navarro@example.com",
    "numero_tlf": "601777888",
    "operadora": "Vodafone",
    "plan": "Full Data",
    "pin": "0000",
    "puk": "000000",
    "sim": "SIM010",
    "id_hardware": "ORD010"
  }
]
}

```

Figura 35: Resultado obtenido tras la ejecución de la Prueba 3

#### Prueba 4 – Función consulta\_general.py

Esta función Lambda no requiere ningún parámetro de entrada y su objetivo es devolver el estado completo del sistema en un único bloque de datos estructurado. Durante la prueba se verificó que la salida JSON incluye correctamente todas las entidades definidas en la base de datos: usuarios, ubicaciones y activos de tipo hardware, licencias y líneas telefónicas. Aunque no se adjunta una figura debido a la longitud del resultado, se ha comprobado que el formato de los datos es consistente con el esperado y que refleja con precisión el contenido actual de las tablas.

## 4.2.2 Prueba integrada del chatbot conversacional

Una vez validadas las funciones Lambda de forma individual, se ha procedido a realizar una prueba integrada ejecutando el cliente local del chatbot (*chatbot\_client.py*) desde consola. Esta prueba tiene como objetivo comprobar el flujo completo de interpretación, extracción de datos y generación de respuestas mediante lenguaje natural, incluyendo también la correcta gestión de la memoria conversacional.

Durante esta prueba se han formulado distintas preguntas que permiten cubrir todos los tipos de consulta definidos en el sistema: por usuario, por activo, por ubicación y generales. Además, se ha incluido una secuencia encadenada de preguntas para validar que el modelo es capaz de mantener el contexto y ofrecer respuestas coherentes en función del historial conversacional.

A continuación, se muestra la conversación mantenida con el agente para la ejecución de la prueba:

```
C:\Users\diego\Documents\TFG-II>python chatbot_client.py
Bienvenido. Escribe tu consulta sobre activos IT. Escribe 'salir' para terminar.

Tú: dime a que departamento y ubicacion pertenece laura martin ruiz
Claude: Laura Martín Ruiz pertenece al departamento de Consultoría y está ubicada en la Oficina Málaga.

Tú: que activos tiene?
Claude: Laura Martín Ruiz tiene asignados los siguientes activos:

- Un ordenador Lenovo ThinkPad E14
- Una licencia de IntelliJ IDEA de JetBrains
- Una línea telefónica de Orange con el número 600555666

Tú: Cuantos empleados tiene la oficina de Malaga y quienes son?
Claude: La oficina de Málaga tiene 3 empleados:

- David Ramírez Gil (Analista de Datos)
- José Sánchez Torres (Programador)
- Laura Martín Ruiz (Consultora)

Tú: quien tiene asignado el activo SN007 ? donde esta ubicado?
Claude: El activo con número de serie SN007 está asignado a Elena Gómez Santos, quien trabaja como DevOps Engineer y está ubicada en la Oficina Barcelona.

Tú: dime cuantas licencias se encuentran disponibles en la organización y a quien me recomendarías asignárselas en base a las necesidades de su rol o función dentro de la empresa
Claude: Hay 2 licencias de Microsoft Office 365 Premium disponibles (LIC011 y LIC012) que no están asignadas actualmente.

Basándome en los roles y funciones de los empleados, recomendaría asignar estas licencias a:

1. José Sánchez Torres (Programador): Si bien ya tiene una licencia Office, podría necesitar una adicional para gestionar documentación técnica y colaborar en proyectos.
2. Elena Gómez Santos (DevOps Engineer): Como DevOps Engineer, podría beneficiarse de una licencia adicional para manejar documentación de infraestructura y procedimientos.

Estas recomendaciones se basan en que ambos roles requieren frecuentemente trabajar con documentación, hojas de cálculo y presentaciones para sus responsabilidades diarias.

Sin embargo, la decisión final debería basarse en las necesidades específicas actuales de cada empleado y las políticas de la empresa sobre asignación de licencias múltiples.

Tú: calcula el coste mensual total en licencias y líneas telefónicas de la oficina de madrid
Claude: Para la oficina de Madrid, el coste mensual total en licencias y líneas telefónicas es de 150€, desglosado de la siguiente manera:

Licencias (90€):
- Ana García: Microsoft Office 365 Premium (20€)
- Carlos López: Adobe Pro (50€)
- María Fernández: Microsoft Office 365 Premium (20€)

Líneas telefónicas (60€):
- Ana García: línea Movistar (20€)
- Carlos López: línea Vodafone (20€)
- María Fernández: línea Movistar (20€)

Tú: salir
Fin del chat.
```

Figura 36: Flujo de conversación mantenido con el Agente AI para la prueba integral del sistema

Como se observa en la figura anterior, el chatbot es capaz de interpretar correctamente cada pregunta, identificar qué función Lambda debe ejecutarse en segundo plano y generar una respuesta clara y adaptada a la consulta. En el caso de preguntas encadenadas, la memoria del intérprete permite resolver referencias al turno anterior, mientras que la memoria del agente garantiza que el flujo de la conversación se mantenga natural.

Estos resultados confirman que el sistema funciona correctamente de extremo a extremo y que es capaz de ofrecer una experiencia conversacional robusta sobre la base de datos de activos tecnológicos.

### **4.3 Pruebas del subsistema de gestión de recursos (API REST)**

Una vez validado el funcionamiento del subsistema de consulta inteligente, se ha llevado a cabo la prueba del segundo subsistema del sistema desarrollado: la API REST orientada a operaciones de gestión. Esta prueba tiene como finalidad comprobar que las funciones Lambda asociadas a los endpoints de usuario, activo y ubicación permiten realizar correctamente las operaciones CRUD (crear, leer, actualizar y eliminar) sobre los datos del sistema, y que los mecanismos de autenticación basados en Amazon Cognito se comportan como se espera.

Todas las pruebas salvo la primera se han realizado utilizando la herramienta Postman previamente configurada con el flujo de autenticación OAuth 2.0 tal y como se ha detallado previamente en el capítulo **3.5.4**. A través de su interfaz se han generado tokens válidos mediante el Hosted UI de Cognito, y se han utilizado para acceder a los distintos recursos protegidos. A continuación, se muestran los resultados obtenidos al invocar el endpoint /activo, ya que es el que presenta mayor complejidad por su estructura de supertipo-subtipo.

#### **Prueba 1 – Consulta específica de un activo por ID (token inválido)**

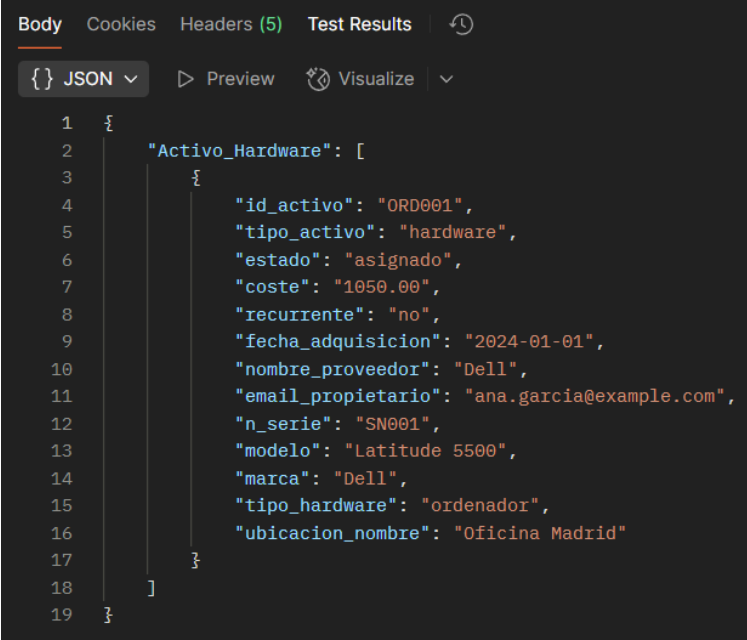
En esta primera prueba se ha intentado acceder a la información de un activo concreto mediante su identificador único (id\_activo) utilizando un token JWT ya expirado. Esta prueba permite comprobar que la API deniega correctamente el acceso a usuarios no autenticados.

- Operación: **GET**
- URL:  
`https://cl114x9yp5.execute-api.us-west-2.amazonaws.com/activo?id_activo=ORD001`
- Resultado: El sistema ha devuelto el error **401 Unauthorized**

## Prueba 2 – Consulta específica de un activo por ID (token válido)

Tras obtener un nuevo token de acceso accediendo con las credenciales de un usuario autorizado se ha solicitado la información detallada del activo ORD001 utilizando su identificador. En esta prueba se pretende verificar que el sistema es capaz de devolver únicamente los datos del activo solicitado, sin incluir información adicional no relevante.

- Operación: **GET**
- URL:  
*[https://cl114x9yp5.execute-api.us-west-2.amazonaws.com/activo?id\\_activo=ORD001](https://cl114x9yp5.execute-api.us-west-2.amazonaws.com/activo?id_activo=ORD001)*
- Resultado: El sistema ha devuelto el código **200 OK** y el siguiente contenido:



```
1  {
2      "Activo_Hardware": [
3          {
4              "id_activo": "ORD001",
5              "tipo_activo": "hardware",
6              "estado": "asignado",
7              "coste": "1050.00",
8              "recurrente": "no",
9              "fecha_adquisicion": "2024-01-01",
10             "nombre_proveedor": "Dell",
11             "email_propietario": "ana.garcia@example.com",
12             "n_serie": "SN001",
13             "modelo": "Latitude 5500",
14             "marca": "Dell",
15             "tipo_hardware": "ordenador",
16             "ubicacion_nombre": "Oficina Madrid"
17         }
18     ]
19 }
```

## Prueba 3 – Inserción de un nuevo activo

En esta prueba se ha realizado una operación de tipo POST para registrar un nuevo activo de tipo hardware. Esta prueba permite comprobar que el sistema acepta correctamente la creación de nuevos registros cuando se proporcionan todos los campos requeridos para la entidad general y su subtipo asociado.

- Operación: **POST**
- URL: *<https://cl114x9yp5.execute-api.us-west-2.amazonaws.com/activo>*
- Cuerpo de la petición:

```

{
  "id_activo": "ORD999",
  "tipo": "hardware",
  "estado": "asignado",
  "coste": 870,
  "recurrente": "no",
  "fecha_adquisicion": "2024-05-01",
  "nombre_proveedor": "HP",
  "email_propietario": "ana.garcia@example.com",
  "n_serie": "SN999",
  "modelo": "ProBook 450",
  "marca": "HP",
  "tipo_hardware": "ordenador",
  "ubicacion_nombre": "Oficina Madrid"
}

```

- Resultado: El sistema ha devuelto el código **201 Created** y el mensaje "Activo creado correctamente."

#### Prueba 4 – Inserción de un nuevo activo (datos incompletos)

En esta prueba se ha intentado insertar un nuevo activo omitiendo el campo obligatorio `id_activo`. El objetivo de la misma es validar que el sistema detecta errores en las peticiones entrantes y responde con mensajes de error adecuados.

- Operación: **POST**
- URL: <https://cl114x9yp5.execute-api.us-west-2.amazonaws.com/activo>
- Cuerpo de la petición:

```

{
  "tipo": "hardware",
  "estado": "asignado",
  "coste": 800,
  "recurrente": "no",
  "fecha_adquisicion": "2024-05-01",
  "nombre_proveedor": "HP",
  "email_propietario": "ana.garcia@example.com",
  "n_serie": "SN000",
  "modelo": "EliteBook",
  "marca": "HP",
  "tipo_hardware": "ordenador",
  "ubicacion_nombre": "Oficina Madrid"
}

```

- Resultado: El sistema ha devuelto el error **500 Internal Server Error** y el siguiente contenido:

```

{
  "error": "id_activo"
}

```

## Prueba 5 – Actualización de un activo existente

Se ha utilizado una operación PUT para actualizar los datos del activo previamente creado ORD999, modificando su estado y modelo. Esta prueba sirve para comprobar que el sistema permite editar correctamente los valores de un activo ya existente sin afectar a otras entidades.

- Operación: **PUT**
- URL: <https://cl114x9yp5.execute-api.us-west-2.amazonaws.com/activo>
- Cuerpo de la petición:

```
{
  "id_activo": "ORD999",
  "tipo": "hardware",
  "estado": "mantenimiento",
  "coste": 870,
  "recurrente": "no",
  "fecha_adquisicion": "2024-05-01",
  "nombre_proveedor": "HP",
  "email_propietario": "ana.garcia@example.com",
  "n_serie": "SN999",
  "modelo": "ProBook 455",
  "marca": "HP",
  "tipo_hardware": "ordenador",
  "ubicacion_nombre": "Oficina Madrid"
}
```

- Resultado: El sistema ha devuelto el código **200 OK** y el mensaje "Activo y subtipo actualizados correctamente."

## Prueba 6 – Eliminación de un activo

Finalmente, se ha eliminado el activo ORD999 mediante una operación DELETE, como parte del ciclo completo de pruebas CRUD. Esta acción permite validar que los registros pueden ser eliminados correctamente del sistema y que no vuelven a aparecer en posteriores consultas.

- Operación: DELETE
- URL:  
[https://cl114x9yp5.execute-api.us-west-2.amazonaws.com/activo?id\\_activo=ORD999](https://cl114x9yp5.execute-api.us-west-2.amazonaws.com/activo?id_activo=ORD999)
- Resultado: El sistema ha devuelto el código **200 OK** y el mensaje "Activo eliminado correctamente."

Estas pruebas confirman que los mecanismos de autenticación funcionan correctamente y que los endpoints definidos permiten gestionar con éxito los recursos del sistema. Además, se ha verificado que el sistema responde de forma controlada ante peticiones malformadas o no autorizadas. El uso de Postman ha facilitado la validación completa del subsistema, replicando de forma realista el comportamiento que tendría una futura interfaz web o aplicación cliente al interactuar con la API. Finalmente, aunque en este apartado se han documentado únicamente las pruebas sobre el endpoint `/activo` por motivos de espacio y relevancia técnica, los otros dos endpoints (`/usuario` y `/ubicacion`) han sido validados siguiendo el mismo procedimiento, obteniendo resultados correctos y coherentes con la lógica del sistema.

## 4.4 Conclusiones

El sistema de gestión de activos tecnológicos desarrollado a lo largo de este proyecto representa una solución eficaz, accesible y técnicamente sólida, especialmente adaptada a las necesidades de pequeñas y medianas empresas. Gracias a una arquitectura modular basada en servicios gestionados de AWS, se ha logrado construir una plataforma robusta y flexible que opera íntegramente en la nube, aprovechando los recursos del entorno gratuito sin comprometer la seguridad, la escalabilidad ni la experiencia del usuario. La separación en dos subsistemas bien diferenciados, el chatbot conversacional y la API REST administrativa, ha permitido atender distintos perfiles de uso sin aumentar la complejidad general, garantizando una gestión completa del inventario digital a través de interfaces independientes pero complementarias.

La integración de Amazon Bedrock como motor de interpretación y generación de respuestas, combinada con funciones Lambda especializadas, ha demostrado ser una estrategia eficaz para ofrecer una experiencia conversacional fluida y coherente. El chatbot es capaz de procesar preguntas abiertas, mantener un contexto básico entre turnos y generar respuestas adaptadas con un consumo controlado de tokens. Además, la memoria conversacional, el diseño del cliente local en consola y el uso de AWS CLI para la autenticación han simplificado tanto el desarrollo como la gestión del estado, sin necesidad de interfaces gráficas ni almacenamiento adicional. El sistema admite también una fácil adaptación a cualquier otro modelo fundacional disponible en Bedrock, bastando con modificar ligeramente el formato de las solicitudes según la documentación oficial, lo que refuerza su escalabilidad y su proyección a futuro.

Las pruebas realizadas han confirmado el correcto funcionamiento de todos los componentes, tanto en escenarios aislados como en interacciones completas de extremo a extremo. Las funciones Lambda operan con estabilidad, la base de datos responde con agilidad y los mecanismos de autenticación protegen eficazmente los accesos. En conjunto, este proyecto demuestra que es posible desarrollar una solución práctica, sostenible y técnicamente madura para la automatización de procesos clave en la gestión de activos IT, integrando de forma natural inteligencia artificial generativa con servicios cloud de última generación. Además de validar su aplicabilidad en contextos reales, el proyecto sienta una base sólida para futuras evoluciones del sistema, como la incorporación de interfaces web, análisis predictivos o integración con herramientas externas de gestión IT.

## 5 Análisis de Impacto

La realización de este Trabajo de Fin de Grado ha supuesto un ejercicio integral no solo desde el punto de vista técnico, sino también en cuanto a la reflexión sobre los impactos que una solución tecnológica puede generar en distintos ámbitos. A continuación, se presenta un análisis de los principales efectos potenciales, tanto positivos como negativos, de la solución desarrollada, así como su alineación con los Objetivos de Desarrollo Sostenible (ODS) de la Agenda 2030.

### **Impacto personal**

Desde una perspectiva personal, el desarrollo de este sistema ha representado un importante reto académico y profesional. La integración de servicios avanzados de inteligencia artificial y arquitectura en la nube ha permitido consolidar conocimientos en áreas clave como el diseño de bases de datos, servicios serverless, seguridad cloud y procesamiento de lenguaje natural. Asimismo, ha fomentado una visión crítica sobre las decisiones de diseño y sus implicaciones más allá del plano técnico.

### **Impacto empresarial**

El sistema propuesto está especialmente orientado a pequeñas y medianas empresas, que suelen carecer de soluciones avanzadas para la gestión de activos tecnológicos. Al combinar una interfaz conversacional con una API estructurada, se facilita tanto el control como la consulta del inventario IT, promoviendo una mejor organización interna y reduciendo los costes operativos. Su arquitectura modular, de bajo coste y fácil mantenimiento, lo convierte en una solución accesible que puede adaptarse a distintas organizaciones con un esfuerzo mínimo.

### **Impacto social y cultural**

Aunque el alcance inmediato del sistema se limita al entorno empresarial, su uso puede contribuir indirectamente a mejorar las condiciones laborales del personal técnico, al reducir tareas repetitivas y facilitar el acceso a la información. Además, promueve la cultura del dato y la digitalización en entornos que habitualmente no disponen de herramientas tan avanzadas. Esta democratización tecnológica puede considerarse un aporte al ODS 9 (*Industria, innovación e infraestructura*), al fomentar entornos más tecnológicamente resilientes y sostenibles.

### **Impacto económico**

El uso mayoritario de servicios dentro del AWS Free Tier, junto con una arquitectura serverless, ha permitido mantener los costes asociados al desarrollo y operación en niveles muy reducidos. Este enfoque contribuye a la sostenibilidad económica de la solución, haciéndola viable para empresas con recursos limitados. Además, al automatizar procesos de consulta y gestión, se libera tiempo del personal técnico, generando indirectamente una mejora de la productividad.

### **Impacto medioambiental**

La optimización del consumo de tokens y la reducción del volumen de datos procesados por el modelo de lenguaje no solo repercute en la eficiencia del sistema, sino también en su huella de carbono. Aunque el uso de modelos fundacionales conlleva un consumo energético considerable, el diseño implementado favorece un uso responsable y eficiente de estos recursos, en línea con el ODS 12 (*Producción y consumo responsables*). La decisión de no almacenar historial conversacional en servicios persistentes, y de evitar interfaces gráficas innecesarias, también responde a una filosofía de simplicidad y bajo impacto.

### **Impacto ético y en la toma de decisiones**

Durante el desarrollo del sistema se han tomado diversas decisiones considerando sus posibles implicaciones. Por ejemplo, se ha optado por mantener todo el procesamiento del chatbot en el entorno local del usuario autenticado, evitando la exposición de datos sensibles en aplicaciones web o canales públicos. Además, la autenticación basada en AWS CLI y Cognito garantiza un control explícito sobre los accesos, reforzando la privacidad y la trazabilidad del sistema.

Para finalizar, este proyecto no solo ha cumplido con sus objetivos funcionales, sino que ha sido ideado y desarrollado desde una perspectiva responsable, teniendo en cuenta su impacto en distintos niveles. Su diseño busca aportar valor real sin generar dependencias innecesarias ni impactos negativos significativos, y se alinea con los principios de sostenibilidad, accesibilidad y eficiencia que promueven los ODS de la Agenda 2030.

## 6 Bibliografía

- [1] McKinsey & Company, «The state of AI in 2023: Generative AI's breakout year,» McKinsey, 1 Agosto 2023. [En línea]. Available: <https://www.mckinsey.com/capabilities/quantumblack/our-insights/the-state-of-ai-in-2023-generative-ais-breakout-year#/>. [Último acceso: 20 Abril 2025].
- [2] AXELOS, ITIL 4: IT Asset Management, Nicosia, Cyprus: PeopleCert International Limited, 2023.
- [3] Licenseware, «licenseware.io,» 16 Enero 2023. [En línea]. Available: <https://licenseware.io/a-brief-history-of-software-licensing/>. [Último acceso: 13 Abril 2025].
- [4] ISO/IEC, «ISO/IEC 19770-1:2017 – Part 1: IT Asset Management Systems – Requirements,» International Organization for Standardization/International Electrotechnical Commission, Noviembre 2017. [En línea]. Available: <https://www.iso.org/standard/68531.html>.
- [5] ITAM Forum, «The 5 Key Business Benefits of IT Asset Management,» *itamf.org*, 2021.
- [6] M. Buenning, «What Is IT Asset Management (ITAM): A Comprehensive Guide,» *NinjaOne*, 25 Octubre 2024.
- [7] Governance & Accountability Institute, «90% of S&P 500 Index Companies Publish Sustainability Reports in 2019, G&A Announces in its Latest Annual 2020 Flash Report,» *GlobeNewswire News Room*, 16 Julio 2020.
- [8] Deloitte, «IT Asset Management Global Survey 2021: Changing expectations in a post-pandemic paradigm,» Junio 2021. [En línea]. Available: <https://www.deloitte.com/global/en/services/risk-advisory/research/it-asset-management-itam-global-survey-2021.html>. [Último acceso: 13 Abril 2025].
- [9] E. Glozshtein, «IT Asset Management Strategy: Current Challenges,» *Alloy Software*, 23 Agosto 2024.
- [10] Flexera, «2024 State of ITAM Report,» *flexera.com*, 2024.
- [11] K. Dawe, «Avoid These 9 Common IT Asset Management Challenges,» *Brightfin*, 23 Mayo 2022.
- [12] FOURMATT, «ISO 19770-1: Conozca sus beneficios,» 4 Mayo 2022. [En línea]. Available: <https://4matt.com.br/es-mx/iso-19770-1-conheca-os-processos-e-seus-beneficios>.
- [13] M. Bohlen, «Thanks to ISO/IEC 19770-1 certified licence management, HDI reduces risks and saves costs,» TIMETOACT Group, Febrero 2023. [En línea]. Available: <https://www.timetoact-group.com/TTA/Referenzen/HDI/HDI-success-story.pdf>.
- [14] M. Thomas, «IT Asset Management and COBIT 5: Strategic ingredients for effective Governance of Enterprise IT,» *APMG International*, Enero 2018.

- [15] M. I. Cardozo, «El reto de gestionar los activos de TI,» CSI, [En línea]. Available: <https://www.csi-col.com.co/index.php/blog/69-el-reto-de-gestionar-los-activos-de-ti>. [Último acceso: 11 Abril 2025].
- [16] A. Kempwad, «COBIT BAI09.01 - Identify And Record Current Assets,» ITSM Docs, 31 Mayo 2024. [En línea]. Available: <https://www.itsm-docs.com/blogs/cobit-framework/cobit-bai09-01-identify-and-record-current-assets>.
- [17] A. Kempwad, «COBIT BAI09.02 - Manage Critical Assets,» ITSM Docs, 4 Junio 2024. [En línea]. Available: <https://www.itsm-docs.com/blogs/cobit-framework/cobit-bai09-02-manage-critical-assets>.
- [18] A. Kempwad, «COBIT BAI09.03 - Manage The Asset Life Cycle,» ITSM Docs, 4 Junio 2024. [En línea]. Available: <https://www.itsm-docs.com/blogs/cobit-framework/cobit-bai09-03-manage-the-asset-life-cycle>.
- [19] A. Kempwad, «COBIT BAI09.04 - Optimize Asset Value,» ITSM Docs, 4 Junio 2024. [En línea]. Available: <https://www.itsm-docs.com/blogs/cobit-framework/cobit-bai09-04-optimize-asset-value>.
- [20] A. Kempwad, «COBIT BAI09.05 - Manage Licenses,» ITSM Docs, 4 Junio 2024. [En línea]. Available: <https://www.itsm-docs.com/blogs/cobit-framework/cobit-bai09-05-manage-licenses>.
- [21] A. Martins Antonio, «What is ITIL 4? Definitive Guide to ITIL 4,» 29 Marzo 2023. [En línea]. Available: <https://itsmhouse.com/what-is-til-4-definitive-guide-to-til-4/>. [Último acceso: 10 Abril 2025].
- [22] interpolados.wordpress.com, «ITIL 4: PRÁCTICAS DE GESTIÓN DE ITIL: GESTIÓN DE ACTIVOS DE TI,» 22 Septiembre 2020. [En línea]. Available: <https://interpolados.wordpress.com/2020/09/22/til-4-practicas-de-gestion-de-til-gestion-de-activos-de-ti/>. [Último acceso: 10 Abril 2025].
- [23] J. Rodriguez, «The Best 7 Free and Open Source IT Asset Management Software,» GoodFirms, 29 Mayo 2020. [En línea]. Available: <https://www.goodfirms.co/it-asset-management-software/blog/best-free-open-source-it-asset-management-software>. [Último acceso: 14 Abril 2025].
- [24] ServiceNow, «IT Asset Management (ITAM),» [En línea]. Available: <https://www.servicenow.com/products/it-asset-management.html>. [Último acceso: 14 Abril 2025].
- [25] T. Tremblay, «ITAM Tools: Top 10 IT Asset Management Software for 2024,» 2024 Septiembre 10. [En línea]. Available: <https://www.kohezion.com/blog/itam-tools>. [Último acceso: 14 Abril 2025].
- [26] Device42, «CMDB vs Asset Management: Key Distinctions Explained,» 22 Abril 2024. [En línea]. Available: <https://www.device42.com/cmdb-best-practices/cmdb-vs-asset-management/>. [Último acceso: 14 Abril 2025].

- [27] Ivanti, «ITAM or CMDB? Why You Need Both,» 3 Diciembre 2024. [En línea]. Available: <https://www.ivanti.com/blog/itam-vs-cmdb>. [Último acceso: 14 Abril 2025].
- [28] M. Shaw, «What comes first: ITAM or configuration management?,» <https://www.servicenow.com/blogs/2022/itam-configuration-management-first> Mayo 2022. [En línea]. Available: <https://www.servicenow.com/blogs/2022/itam-configuration-management-first>. [Último acceso: 14 Abril 2025].
- [29] P. Salfi, «ITAM: A Prerequisite for Building a CMDB (Part 2/6) – “The Link between ITAM and ITSM”,» 20 Noviembre 2009. [En línea]. Available: <https://itassetmanagement.net/2009/11/20/itam-a-prerequisite-for-building-a-cmdb-part-26-%E2%80%93-%E2%80%9Cthe-link-between-itam-and-itsm%E2%80%9D/>. [Último acceso: 15 Abril 2025].
- [30] M. Verlaque, «Integrating SaaS Management into Traditional ITAM Frameworks,» The ITAM Review, 11 Abril 2024. [En línea]. Available: <https://itassetmanagement.net/2024/04/11/integrating-saas-management-into-traditional-itam-frameworks/>. [Último acceso: 16 Abril 2025].
- [31] B. Trevino, «FinOps and ITAM: A unified approach to optimizing technology investments,» Flexera, 5 Marzo 2025. [En línea]. Available: <https://www.flexera.com/blog/finops/finops-and-itam-a-unified-approach-to-optimizing-technology-investments/>. [Último acceso: 16 Abril 2025].
- [32] S. Danby, «The Future of IT Asset Management: 9 ITAM Trends For 2025,» Invgate, 11 Diciembre 2023. [En línea]. Available: <https://blog.invgate.com/it-asset-management-trends-2024>. [Último acceso: 16 Abril 2025].
- [33] A. Mehrotra, «The Next Generation of IT Asset Management: Insights for 2024,» Medium, 2 Febrero 2024. [En línea]. Available: <https://arvind-mehrotra.medium.com/the-next-generation-of-it-asset-management-insights-for-2024-535ed4147f0a>. [Último acceso: 16 Abril 2025].
- [34] R. Stefani, «Digital Employee Experience (DEX): What it is and why you should care,» The ITAM Review, 4 Septiembre 2024. [En línea]. Available: <https://itassetmanagement.net/2024/09/04/gartner-magic-quadrant-digital-employee-experience/>. [Último acceso: 16 Abril 2025].
- [35] C. Stryker y M. Scapicchio, «¿Qué es la IA generativa?,» IBM, 22 Marzo 2024. [En línea]. Available: <https://www.ibm.com/mx-es/think/topics/generative-ai>. [Último acceso: 19 Abril 2025].
- [36] K. Martineau, «What is generative AI?,» IBM Research, 20 Abril 2023. [En línea]. Available: <https://research.ibm.com/blog/what-is-generative-AI>. [Último acceso: 19 Abril 2025].
- [37] C. Quinn, «AI as a System,» Learnlets, 7 Mayo 2024. [En línea]. Available: <https://blog.learnlets.com/2024/05/ai-as-a-system/>. [Último acceso: 20 Abril 2025].

- [38] Gartner, «Gartner Says More Than 80% of Enterprises Will Have Used Generative AI APIs or Deployed Generative AI-Enabled Applications by 2026,» Gartner, 11 Octubre 2023. [En línea]. Available: <https://www.gartner.com/en/newsroom/press-releases/2023-10-11-gartner-says-more-than-80-percent-of-enterprises-will-have-used-generative-ai-apis-or-deployed-generative-ai-enabled-applications-by-2026>. [Último acceso: 20 Abril 2025].
- [39] R. Bommasani y P. Liang, «Reflections on Foundation Models,» Stanford University, 18 Octubre 2021. [En línea]. Available: <https://hai.stanford.edu/news/reflections-foundation-models>. [Último acceso: 20 Abril 2025].
- [40] N. Klingler, «Foundation Models in Modern AI Development (2024 Guide),» viso.ai, 20 Marzo 2024. [En línea]. Available: <https://viso.ai/deep-learning/foundation-models/>. [Último acceso: 29 Mayo 2025].
- [41] Future Internet, «The timeline of the development of LLMs from 2018 to 2024 (June), showcasing key advancements and notable models,» Diciembre 2024. [En línea]. Available: [https://www.researchgate.net/figure/The-timeline-of-the-development-of-LLMs-from-2018-to-2024-June-showcasing-key\\_fig3\\_384694535](https://www.researchgate.net/figure/The-timeline-of-the-development-of-LLMs-from-2018-to-2024-June-showcasing-key_fig3_384694535).
- [42] M. Quintanilla, «Why your IT asset management needs AI,» groWrk, 11 Mayo 2024. [En línea]. Available: <https://growrk.com/blog/why-your-it-asset-management-needs-ai>. [Último acceso: 1 Abril 2025].
- [43] A. J. Witt, «Torii announces new AI & developer capabilities,» ITAM Review, 13 Septiembre 2023. [En línea]. Available: <https://itassetmanagement.net/2023/09/13/saas-management-veteran-torii-announces-new-ai-developer-capabilities/>. [Último acceso: 1 Abril 2025].
- [44] P. Blackstaffe, «GenAI IT Service Management Use Case Examples,» ITSM Tools, 6 Noviembre 2024. [En línea]. Available: <https://itsm.tools/genai-itsm/>. [Último acceso: 2 Abril 2025].
- [45] D. Dominguez, «AWS Enters the Generative AI Race with Bedrock and Titan Foundation Models,» InfoQ, 19 Abril 2023. [En línea]. Available: <https://www.infoq.com/news/2023/04/aws-bedrock-titan/>. [Último acceso: 3 Abril 2025].
- [46] Amazon Web Services, «Amazon Bedrock: La forma más sencilla de crear y escalar aplicaciones de IA generativa con modelos básicos,» AWS, 2025. [En línea]. Available: <https://aws.amazon.com/es/bedrock/>. [Último acceso: 21 Abril 2025].
- [47] P. Vogel, «Building a serverless document chat with AWS Lambda and Amazon Bedrock,» Amazon Web Services, 4 Octubre 2023. [En línea]. Available: <https://aws.amazon.com/es/blogs/compute/building-a-serverless-document-chat-with-aws-lambda-and-amazon-bedrock/>. [Último acceso: 24 Abril 2025].
- [48] Amazon Web Services, «Retrieve data and generate AI responses with Amazon Bedrock Knowledge Bases,» Amazon Bedrock User Guide, [En línea]. Available:

<https://docs.aws.amazon.com/bedrock/latest/userguide/knowledge-base.html>. [Último acceso: 25 Abril 2025].

- [49] «Amazon Bedrock: Claude 3.5 Sonnet v2,» Amazon Web Services, 2025. [En línea]. Available: <https://us-west-2.console.aws.amazon.com/bedrock/home?region=us-west-2#/model-catalog/serverless/anthropic.claude-3-5-sonnet-20241022-v2:0>. [Último acceso: 20 Mayo 2025].

## 7 Anexos

consulta\_creacion\_tablas.sql:

```
USE itam_db;

-- Tabla Ubicacion
CREATE TABLE Ubicacion (
  nombre VARCHAR(255) PRIMARY KEY,
  pais VARCHAR(100),
  calle VARCHAR(255),
  numero INT,
  localidad VARCHAR(100),
  provincia VARCHAR(100),
  codigo_postal VARCHAR(10),
  n_empleados INT DEFAULT 0
);

-- Tabla Usuario
CREATE TABLE Usuario (
  email VARCHAR(255) PRIMARY KEY,
  nombre_completo VARCHAR(255) NOT NULL,
  telefono VARCHAR(20),
  estado ENUM('activo', 'baja', 'vacaciones', 'suspendido') NOT NULL,
  tipo_usuario ENUM('interno', 'externo') NOT NULL,
  puesto VARCHAR(100),
  departamento VARCHAR(100),
  ubicacion VARCHAR(255),
  FOREIGN KEY (ubicacion) REFERENCES Ubicacion(nombre) ON DELETE SET NULL
);

-- Tabla Activo (Supertipo)
CREATE TABLE Activo (
  id_activo VARCHAR(100) PRIMARY KEY,
  tipo ENUM('hardware', 'licencia', 'linea_telefonica') NOT NULL,
  estado ENUM('asignado', 'disponible', 'en_reparacion', 'baja') NOT NULL,
  coste DECIMAL(10,2),
  recurrente ENUM('si', 'no') DEFAULT 'no',
  fecha_adquisicion DATE,
  nombre_proveedor VARCHAR(100),
  email_propietario VARCHAR(255),
  FOREIGN KEY (email_propietario) REFERENCES Usuario(email) ON DELETE SET NULL
);

-- Tabla Hardware (Subtipo de Activo)
CREATE TABLE Hardware (
  id_activo VARCHAR(100) PRIMARY KEY,
  n_serie VARCHAR(100),
  modelo VARCHAR(100),
  marca VARCHAR(100),
  tipo ENUM(
    'smartphone',
    'tablet',
    'ordenador',
    'impresora',
    'servidor',
    'router',
    'accesorio',
    'switch',
    'firewall',
    'monitor'
  ),
  ubicacion_nombre VARCHAR(255),
  FOREIGN KEY (id_activo) REFERENCES Activo(id_activo) ON DELETE CASCADE,
  FOREIGN KEY (ubicacion_nombre) REFERENCES Ubicacion(nombre) ON DELETE SET NULL
);
```

```

-- Tabla Licencia (Subtipo de Activo)
CREATE TABLE Licencia (
    id_activo VARCHAR(100) PRIMARY KEY,
    nombre_licencia VARCHAR(100),
    producto VARCHAR(100),
    numero_licencia VARCHAR(100),
    FOREIGN KEY (id_activo) REFERENCES Activo(id_activo) ON DELETE CASCADE
);

-- Tabla Linea_Telefonica (Subtipo de Activo)
CREATE TABLE Linea_Telefonica (
    id_activo VARCHAR(100) PRIMARY KEY,
    numero_tlf VARCHAR(20),
    operadora VARCHAR(50),
    plan VARCHAR(100),
    pin VARCHAR(10),
    puk VARCHAR(10),
    sim VARCHAR(50),
    id_hardware VARCHAR(100),
    FOREIGN KEY (id_activo) REFERENCES Activo(id_activo) ON DELETE CASCADE,
    FOREIGN KEY (id_hardware) REFERENCES Hardware(id_activo) ON DELETE SET NULL
);

-- Trigger: al insertar un nuevo usuario, incrementar el contador en la ubicación
DELIMITER //
CREATE TRIGGER trg_incrementar_empleados
AFTER INSERT ON Usuario
FOR EACH ROW
BEGIN
    IF NEW.ubicacion IS NOT NULL THEN
        UPDATE Ubicacion
        SET n_empleados = n_empleados + 1
        WHERE nombre = NEW.ubicacion;
    END IF;
END;
//
DELIMITER ;

-- Trigger: al eliminar un usuario, decrementar el contador en la ubicación
DELIMITER //
CREATE TRIGGER trg_decrementar_empleados
AFTER DELETE ON Usuario
FOR EACH ROW
BEGIN
    IF OLD.ubicacion IS NOT NULL THEN
        UPDATE Ubicacion
        SET n_empleados = n_empleados - 1
        WHERE nombre = OLD.ubicacion;
    END IF;
END;
//
DELIMITER ;

```

## consulta\_por\_usuario.py:

```
import pymysql
import os
import json

# Mapeo de tipo a query SQL
join_map = {
    "hardware": """
        SELECT
            a.id_activo,
            a.tipo AS tipo_activo,
            a.estado,
            a.coste,
            a.recurrente,
            a.fecha_adquisicion,
            a.nombre_proveedor,
            a.email_propietario,
            h.n_serie,
            h.modelo,
            h.marca,
            h.tipo AS tipo_hardware,
            h.ubicacion_nombre
        FROM Activo a
        JOIN Hardware h ON a.id_activo = h.id_activo
        WHERE a.email_propietario = %s
    """,
    "licencia": """
        SELECT
            a.id_activo,
            a.tipo AS tipo_activo,
            a.estado,
            a.coste,
            a.recurrente,
            a.fecha_adquisicion,
            a.nombre_proveedor,
            a.email_propietario,
            l.nombre_licencia,
            l.producto,
            l.numero_licencia
        FROM Activo a
        JOIN Licencia l ON a.id_activo = l.id_activo
        WHERE a.email_propietario = %s
    """,
    "linea_telefonica": """
        SELECT
            a.id_activo,
            a.tipo AS tipo_activo,
            a.estado,
            a.coste,
            a.recurrente,
            a.fecha_adquisicion,
            a.nombre_proveedor,
            a.email_propietario,
            lt.numero_tlf,
            lt.operadora,
            lt.plan,
            lt.pin,
            lt.puk,
            lt.sim,
            lt.id_hardware
        FROM Activo a
        JOIN Linea_Telefonica lt ON a.id_activo = lt.id_activo
        WHERE a.email_propietario = %s
    ""
}
```

```

def lambda_handler(event, context):
    identificador = event.get("identificador")
    if not identificador:
        return {
            "statusCode": 400,
            "body": json.dumps({"error": "No se proporcionó ningún identificador"})
        }

    conn = pymysql.connect(
        host=os.environ["DB_HOST"],
        user=os.environ["DB_USER"],
        password=os.environ["DB_PASS"],
        database=os.environ["DB_NAME"],
        cursorclass=pymysql.cursors.DictCursor
    )

    try:
        with conn.cursor() as cursor:
            # Determinar si es email o nombre completo
            if "@" in identificador:
                cursor.execute("SELECT * FROM Usuario WHERE email = %s", (identificador,))
            else:
                cursor.execute("SELECT * FROM Usuario WHERE nombre_completo = %s", (identificador,))

            usuario_data = cursor.fetchone()

            if not usuario_data:
                return {
                    "statusCode": 404,
                    "body": json.dumps({"error": "Usuario no encontrado"})
                }

            email_usuario = usuario_data["email"]

            # Obtener ubicación
            ubicacion_data = None
            if usuario_data.get("ubicacion"):
                cursor.execute("SELECT * FROM Ubicacion WHERE nombre = %s", (usuario_data["ubicacion"],))
                ubicacion_data = cursor.fetchone()

            # Obtener activos por tipo
            cursor.execute(join_map("hardware"), (email_usuario,))
            activos_hardware = cursor.fetchall()

            cursor.execute(join_map("licencia"), (email_usuario,))
            activos_licencia = cursor.fetchall()

            cursor.execute(join_map("linea_telefonica"), (email_usuario,))
            activos_linea = cursor.fetchall()

            # Construir respuesta final
            response = {
                "Usuario": [usuario_data],
                "Ubicacion": [ubicacion_data] if ubicacion_data else [],
                "Activo_Hardware": activos_hardware,
                "Activo_Licencia": activos_licencia,
                "Activo_Linea_Telefonica": activos_linea
            }

            return {
                "statusCode": 200,
                "body": json.dumps(response, default=str)
            }

        finally:
            conn.close()

    # Prueba local
    if __name__ == "__main__":
        evento_prueba = {"identificador": "ana.garcia@example.com"} # o nombre completo
        resultado = lambda_handler(evento_prueba, None)
        print("Código de estado:", resultado["statusCode"])
        print(json.dumps(json.loads(resultado["body"]), indent=2, ensure_ascii=False))

```

## consulta\_por\_activo.py:

```
import pymysql
import os
import json

# Mapeo de tipo a query SQL
join_map = {
    "hardware": """
        SELECT
            a.id_activo,
            a.tipo AS tipo_activo,
            a.estado,
            a.coste,
            a.recurrente,
            a.fecha_adquisicion,
            a.nombre_proveedor,
            a.email_propietario,
            h.n_serie,
            h.modelo,
            h.marca,
            h.tipo AS tipo_hardware,
            h.ubicacion_nombre
        FROM Activo a
        JOIN Hardware h ON a.id_activo = h.id_activo
        WHERE a.id_activo = %s
    """,
    "licencia": """
        SELECT
            a.id_activo,
            a.tipo AS tipo_activo,
            a.estado,
            a.coste,
            a.recurrente,
            a.fecha_adquisicion,
            a.nombre_proveedor,
            a.email_propietario,
            l.nombre_licencia,
            l.producto,
            l.numero_licencia
        FROM Activo a
        JOIN Licencia l ON a.id_activo = l.id_activo
        WHERE a.id_activo = %s
    """,
    "linea_telefonica": """
        SELECT
            a.id_activo,
            a.tipo AS tipo_activo,
            a.estado,
            a.coste,
            a.recurrente,
            a.fecha_adquisicion,
            a.nombre_proveedor,
            a.email_propietario,
            lt.numero_tlf,
            lt.operadora,
            lt.plan,
            lt.pin,
            lt.puk,
            lt.sim,
            lt.id_hardware
        FROM Activo a
        JOIN Linea_Telefonica lt ON a.id_activo = lt.id_activo
        WHERE a.id_activo = %s
    """,
}

# Mapeo de tipo al nombre de tabla para la respuesta
tabla_map = {
    "hardware": "Activo_Hardware",
    "licencia": "Activo_Licencia",
    "linea_telefonica": "Activo_Linea_Telefonica"
}
```

```

# Función auxiliar para identificar tipo si no se pasa el id_activo directamente
def identificar_activo(conn, identificador):
    with conn.cursor() as cursor:
        sql = """
            SELECT a.id_activo, a.tipo
            FROM Activo a
            LEFT JOIN Hardware h ON a.id_activo = h.id_activo
            LEFT JOIN Licencia l ON a.id_activo = l.id_activo
            LEFT JOIN Linea_Telefonica lt ON a.id_activo = lt.id_activo
            WHERE a.id_activo = %s OR h.n_serie = %s OR l.numero_licencia = %s OR lt.numero_tlf = %s
            LIMIT 1;
        """
        cursor.execute(sql, (identificador, identificador, identificador, identificador))
        result = cursor.fetchone()
        return (result['id_activo'], result['tipo']) if result else (None, None)

def lambda_handler(event, context):
    identificador = event.get("identificador")

    conn = pymysql.connect(
        host=os.environ["DB_HOST"],
        user=os.environ["DB_USER"],
        password=os.environ["DB_PASS"],
        database=os.environ["DB_NAME"],
        cursorclass=pymysql.cursors.DictCursor
    )

    try:
        with conn.cursor() as cursor:
            # Identificar tipo e id_activo
            cursor.execute("SELECT tipo FROM Activo WHERE id_activo = %s", (identificador,))
            row = cursor.fetchone()
            if row:
                id_activo = identificador
                tipo = row["tipo"]
            else:
                id_activo, tipo = identificar_activo(conn, identificador)

            if not id_activo or not tipo:
                return {
                    "statusCode": 404,
                    "body": json.dumps({"error": "Activo no encontrado"})
                }

            # Obtener join Activo + Subtipo
            sql_join = join_map.get(tipo)
            if not sql_join:
                return {
                    "statusCode": 500,
                    "body": json.dumps({"error": "Tipo de activo no soportado"})
                }

            cursor.execute(sql_join, (id_activo,))
            activo_extendido = cursor.fetchone()

            # Obtener suario propietario
            cursor.execute("SELECT * FROM Usuario WHERE email = %s", (activo_extendido["email_propietario"],))
            usuario_data = cursor.fetchone()

            # Obtener ubicación del activo si es tangible o la del propietario si es intangible
            ubicacion_data = None

            if tipo == "hardware" and activo_extendido.get("ubicacion_nombre"):
                cursor.execute("SELECT * FROM Ubicacion WHERE nombre = %s", (activo_extendido["ubicacion_nombre"],))
                ubicacion_data = cursor.fetchone()
            elif usuario_data and usuario_data.get("ubicacion"):
                cursor.execute("SELECT * FROM Ubicacion WHERE nombre = %s", (usuario_data["ubicacion"],))
                ubicacion_data = cursor.fetchone()

            # Construcción de la respuesta
            tipo_tabla = tabla_map[tipo]
            response = {
                tipo_tabla: [activo_extendido],
                "Usuario": [usuario_data] if usuario_data else [],
                "Ubicacion": [ubicacion_data] if ubicacion_data else []
            }

            return {
                "statusCode": 200,
                "body": json.dumps(response, default=str)
            }

    finally:
        conn.close()

# Prueba local
if __name__ == "__main__":
    evento_prueba = {"identificador": "ORD001"} # ID o identificador real
    resultado = lambda_handler(evento_prueba, None)
    print("Código de estado:", resultado["statusCode"])
    print(json.dumps(json.loads(resultado["body"]), indent=2, ensure_ascii=False))

```

## consulta\_por\_ubicacion.py:

```
import pymysql
import os
import json

# Queries SQL
query_usuarios = """
SELECT * FROM Usuario WHERE ubicacion = %s
"""

query_hardware = """
SELECT
    a.id_activo,
    a.tipo AS tipo_activo,
    a.estado,
    a.coste,
    a.recurrente,
    a.fecha_adquisicion,
    a.nombre_proveedor,
    a.email_propietario,
    h.n_serie,
    h.modelo,
    h.marca,
    h.tipo AS tipo_hardware,
    h.ubicacion_nombre
FROM Activo a
JOIN Hardware h ON a.id_activo = h.id_activo
WHERE h.ubicacion_nombre = %s
"""

query_licencias = """
SELECT
    a.id_activo,
    a.tipo AS tipo_activo,
    a.estado,
    a.coste,
    a.recurrente,
    a.fecha_adquisicion,
    a.nombre_proveedor,
    a.email_propietario,
    l.nombre_licencia,
    l.producto,
    l.numero_licencia
FROM Activo a
JOIN Licencia l ON a.id_activo = l.id_activo
WHERE a.email_propietario IN (
    SELECT email FROM Usuario WHERE ubicacion = %s
)
"""

query_lineas = """
SELECT
    a.id_activo,
    a.tipo AS tipo_activo,
    a.estado,
    a.coste,
    a.recurrente,
    a.fecha_adquisicion,
    a.nombre_proveedor,
    a.email_propietario,
    lt.numero_tlf,
    lt.operadora,
    lt.plan,
    lt.pin,
    lt.puk,
    lt.slm,
    lt.id_hardware
FROM Activo a
JOIN Linea_Telefonica lt ON a.id_activo = lt.id_activo
WHERE a.email_propietario IN (
    SELECT email FROM Usuario WHERE ubicacion = %s
)
"""
```

```

def lambda_handler(event, context):
    nombre_ubicacion = event.get("identificador")
    if not nombre_ubicacion:
        return {
            "statusCode": 400,
            "body": json.dumps({"error": "No se proporcionó ninguna ubicación"})
        }

    conn = pymysql.connect(
        host=os.environ["DB_HOST"],
        user=os.environ["DB_USER"],
        password=os.environ["DB_PASS"],
        database=os.environ["DB_NAME"],
        cursorclass=pymysql.cursors.DictCursor
    )

    try:
        with conn.cursor() as cursor:
            # Obtener información de la ubicación solicitada
            cursor.execute("SELECT * FROM Ubicacion WHERE nombre = %s", (nombre_ubicacion,))
            ubicacion_data = cursor.fetchone()

            # Obtener usuarios en la ubicación
            cursor.execute(query_usuarios, (nombre_ubicacion,))
            usuarios = cursor.fetchall()

            # Obtener hardware ubicado allí
            cursor.execute(query_hardware, (nombre_ubicacion,))
            activos_hardware = cursor.fetchall()

            # Obtener licencias de los usuarios de esa ubicación
            cursor.execute(query_licencias, (nombre_ubicacion,))
            activos_licencia = cursor.fetchall()

            # Obtener líneas telefónicas de los usuarios de esa ubicación
            cursor.execute(query_lineas, (nombre_ubicacion,))
            activos_linea = cursor.fetchall()

            # Construcción de la respuesta
            response = {
                "Ubicacion": [ubicacion_data] if ubicacion_data else [],
                "Usuario": usuarios,
                "Activo_Hardware": activos_hardware,
                "Activo_Licencia": activos_licencia,
                "Activo_Linea_Telefonica": activos_linea
            }

            return {
                "statusCode": 200,
                "body": json.dumps(response, default=str)
            }

        finally:
            conn.close()

# Prueba local
if __name__ == "__main__":
    evento_prueba = {"identificador": "Oficina Barcelona"}
    resultado = lambda_handler(evento_prueba, None)
    print("Código de estado:", resultado["statusCode"])
    print(json.dumps(json.loads(resultado["body"]), indent=2, ensure_ascii=False))

```

## consulta\_general.py:

```
import pymysql
import os
import json

# Queries SQL
query_usuarios = "SELECT * FROM Usuario"
query_ubicaciones = "SELECT * FROM Ubicacion"

query_hardware = """
SELECT
    a.id_activo,
    a.tipo AS tipo_activo,
    a.estado,
    a.coste,
    a.recurrente,
    a.fecha_adquisicion,
    a.nombre_proveedor,
    a.email_propietario,
    h.n_serie,
    h.modelo,
    h.marca,
    h.tipo AS tipo_hardware,
    h.ubicacion_nombre
FROM Activo a
JOIN Hardware h ON a.id_activo = h.id_activo
"""

query_licencias = """
SELECT
    a.id_activo,
    a.tipo AS tipo_activo,
    a.estado,
    a.coste,
    a.recurrente,
    a.fecha_adquisicion,
    a.nombre_proveedor,
    a.email_propietario,
    l.nombre_licencia,
    l.producto,
    l.numero_licencia
FROM Activo a
JOIN Licencia l ON a.id_activo = l.id_activo
"""

query_lineas = """
SELECT
    a.id_activo,
    a.tipo AS tipo_activo,
    a.estado,
    a.coste,
    a.recurrente,
    a.fecha_adquisicion,
    a.nombre_proveedor,
    a.email_propietario,
    lt.numero_tlf,
    lt.operadora,
    lt.plan,
    lt.pin,
    lt.puk,
    lt.sim,
    lt.id_hardware
FROM Activo a
JOIN Linea_Telefonica lt ON a.id_activo = lt.id_activo
"""
```

```

def lambda_handler(event, context):
    conn = pymysql.connect(
        host=os.environ["DB_HOST"],
        user=os.environ["DB_USER"],
        password=os.environ["DB_PASS"],
        database=os.environ["DB_NAME"],
        cursorclass=pymysql.cursors.DictCursor
    )

    try:
        with conn.cursor() as cursor:
            cursor.execute(query_usuarios)
            usuarios = cursor.fetchall()

            cursor.execute(query_ubicaciones)
            ubicaciones = cursor.fetchall()

            cursor.execute(query_hardware)
            activos_hardware = cursor.fetchall()

            cursor.execute(query_licencias)
            activos_licencia = cursor.fetchall()

            cursor.execute(query_lineas)
            activos_linea = cursor.fetchall()

            response = {
                "Usuario": usuarios,
                "Ubicacion": ubicaciones,
                "Activo_Hardware": activos_hardware,
                "Activo_Licencia": activos_licencia,
                "Activo_Linea_Telefonica": activos_linea
            }

            return {
                "statusCode": 200,
                "body": json.dumps(response, default=str)
            }

    finally:
        conn.close()

# Prueba local
if __name__ == "__main__":
    evento_prueba = {}
    resultado = lambda_handler(evento_prueba, None)
    print("Código de estado:", resultado["statusCode"])
    print(json.dumps(json.loads(resultado["body"]), indent=2, ensure_ascii=False))

```

## chatbot\_client.py:

```
import boto3
import json

# Clientes
bedrock = boto3.client("bedrock-runtime", region_name="us-west-2")
lambda_client = boto3.client("lambda", region_name="us-west-2")

MODEL_ID = "anthropic.claude-3-5-sonnet-20241022-v2:0"

# Prompt para interpretación
SYSTEM_PROMPT = """
Eres un agente especializado en interpretar consultas sobre activos informáticos realizadas en lenguaje natural.

Tu única tarea es analizar la consulta del usuario y devolver únicamente el nombre de la función Lambda a ejecutar y su argumento (si es necesario).

Funciones disponibles:
1. consulta_por_activo,<identificador>
2. consulta_por_usuario,<identificador>
3. consulta_por_ubicacion,<identificador>
4. consulta_general

Reglas:
- El identificador puede contener espacios (por ejemplo: "Oficina Málaga").
- Si no hay identificador (consulta general), devuelve solo el nombre de la función.
- Formato de salida: solo texto, sin comillas, sin explicaciones, una sola línea, usa una coma si hay identificador.

Ejemplos:
Consulta: ¿Quién tiene el activo SN001?
Respuesta: consulta_por_activo,SN001

Consulta: Muéstrame que activos tiene elena gomez santos
Respuesta: consulta_por_usuario,Elena Gómez Santos

Consulta: ¿Qué usuarios están en las oficinas de malaga?
Respuesta: consulta_por_ubicacion,Oficina Málaga

Consulta: ¿Que usuarios tienen líneas de yoigo?
Respuesta: consulta_general
"""

# Interpretación de la consulta
def interpretar_funcion(pregunta_usuario, pregunta_anterior=None):
    texto = ""
    if pregunta_anterior:
        texto += f"Consulta previa: {pregunta_anterior.strip()}\n"
    texto += f"Consulta actual: {pregunta_usuario.strip()}"

    payload = {
        "anthropic_version": "bedrock-2023-05-31",
        "max_tokens": 50,
        "temperature": 0,
        "top_k": 1,
        "top_p": 1,
        "messages": [
            {
                "role": "user",
                "content": [
                    {
                        "type": "text",
                        "text": f"{SYSTEM_PROMPT.strip()}\n\n{texto.strip()}"
                    }
                ]
            }
        ]
    }

    response = bedrock.invoke_model(
        modelId=MODEL_ID,
        contentType="application/json",
        accept="application/json",
        body=json.dumps(payload)
    )

    salida = json.loads(response["body"].read())["content"][0]["text"].strip()

    if "," in salida:
        nombre_funcion, identificador = salida.split(",", 1)
        payload_lambda = {"identificador": identificador.strip()}
```

```

else:
    nombre_funcion = salida.strip()
    payload_lambda = {}

    return nombre_funcion, payload_lambda

# Invocación de la función Lambda
def ejecutar_lambda(nombre_funcion, payload_lambda):
    result = lambda_client.invoke(
        FunctionName=nombre_funcion,
        InvocationType="RequestResponse",
        Payload=json.dumps(payload_lambda)
    )
    return json.loads(result["Payload"]).read()

# Generación de la respuesta final de Claude con los datos
def generar_respuesta(pregunta_usuario, datos_json, historial):
    mensajes = []

    for turno in historial:
        mensajes.append({"role": "user", "content": [{"type": "text", "text": turno["user"]}]}))
        mensajes.append({"role": "assistant", "content": [{"type": "text", "text": turno["assistant"]}]}))

    mensajes.append({
        "role": "user",
        "content": [
            {"type": "text", "text": f""Esta es la consulta del usuario:\n{pregunta_usuario.strip()}\n
Estos son los datos del sistema para responder (en JSON):\n{json.dumps(datos_json, ensure_ascii=False, indent=2)}\n
Con base en estos datos, responde en lenguaje natural (no reveles su estructura).""}
        ]
    })

    payload = {
        "anthropic_version": "bedrock-2023-05-31",
        "max_tokens": 500,
        "temperature": 0.7,
        "top_p": 0.9,
        "messages": mensajes
    }

    response = bedrock.invoke_model(
        modelId=MODEL_ID,
        contentType="application/json",
        accept="application/json",
        body=json.dumps(payload)
    )

    return json.loads(response["body"]).read()[0]["text"].strip()

# Bucle principal de conversación
def iniciar_chat():
    historial = []
    print("👋 Bienvenido. Escribe tu consulta sobre activos IT. Escribe 'salir' para terminar.\n")

    while True:
        pregunta = input("👤 Tú: ").strip()
        if pregunta.lower() in ["salir", "exit"]:
            print("👋 Fin del chat.")
            break

        try:
            pregunta_anterior = historial[-1]["user"] if historial else None
            funcion, parametros = interpretar_funcion(pregunta, pregunta_anterior)
            datos = ejecutar_lambda(funcion, parametros)
            respuesta = generar_respuesta(pregunta, datos.get("body", datos), historial)

            print(f"👤 Claude: {respuesta}\n")
            historial.append({"user": pregunta, "assistant": respuesta})

        except Exception as e:
            print(f"❌ Error: {e}\n")

# Ejecutar
if __name__ == "__main__":
    iniciar_chat()

```

## gestion\_usuario.py:

```
import pymysql
import os
import json

# Conexión con la base de datos
def conectar_db():
    return pymysql.connect(
        host=os.environ["DB_HOST"],
        user=os.environ["DB_USER"],
        password=os.environ["DB_PASS"],
        database=os.environ["DB_NAME"],
        cursorclass=pymysql.cursors.DictCursor
    )

def lambda_handler(event, context):
    method = event.get("requestContext", {}).get("http", {}).get("method", "GET")
    print("Método recibido:", event.get("httpMethod"))

    if not method:
        print("⚠ No se recibió httpMethod en el evento. Evento completo:")
        print(json.dumps(event, indent=2))
        return {"statusCode": 400, "body": json.dumps({"error": "Método HTTP no especificado"})}

    body = event.get("body")
    print("Cuerpo crudo:", event.get("body"))

    # Cargar el cuerpo de la petición
    if body and isinstance(body, str):
        body = json.loads(body)

    conn = conectar_db()

    try:
        with conn.cursor() as cursor:

            # Crear un nuevo usuario
            if method == "POST":
                sql = """
                INSERT INTO Usuario (email, nombre_completo, telefono, estado, tipo_usuario, puesto, departamento, ubicacion)
                VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
                """
                values = {
                    body["email"],
                    body["nombre_completo"],
                    body["telefono"],
                    body["estado"],
                    body["tipo_usuario"],
                    body["puesto"],
                    body["departamento"],
                    body["ubicacion"]
                }
                cursor.execute(sql, values)
                conn.commit()
                return {"statusCode": 201, "body": json.dumps({"mensaje": "Usuario creado correctamente."})}

            # Obtener uno o todos los usuarios
            elif method == "GET":
                email = event.get("queryStringParameters", {}).get("email")
                if email:
                    cursor.execute("SELECT * FROM Usuario WHERE email = %s", (email,))
                    resultado = cursor.fetchone()
                else:
                    cursor.execute("SELECT * FROM Usuario")
                    resultado = cursor.fetchall()
                return {"statusCode": 200, "body": json.dumps(resultado, default=str)}

            # Actualizar un usuario existente
            elif method == "PUT":
                sql = """
                UPDATE Usuario
                SET nombre_completo=%s, telefono=%s, estado=%s, tipo_usuario=%s, puesto=%s, departamento=%s, ubicacion=%s
                WHERE email=%s
                """
                values = {
                    body["nombre_completo"],
                    body["telefono"],
                    body["estado"],
                    body["tipo_usuario"],
                    body["puesto"],
                    body["departamento"],
                    body["ubicacion"],
                    body["email"]
                }
                cursor.execute(sql, values)
                conn.commit()
                return {"statusCode": 200, "body": json.dumps({"mensaje": "Usuario actualizado correctamente."})}
```

```
# Eliminar un usuario
elif method == "DELETE":
    email = event.get("queryStringParameters", {}).get("email")
    cursor.execute("DELETE FROM Usuario WHERE email = %s", (email,))
    conn.commit()
    return {"statusCode": 200, "body": json.dumps({"mensaje": "Usuario eliminado correctamente."})}

# Método no soportado
else:
    return {"statusCode": 405, "body": json.dumps({"error": "Método HTTP no permitido."})}

except Exception as e:
    return {"statusCode": 500, "body": json.dumps({"error": str(e)})}
finally:
    conn.close()
```

## gestion\_ubicacion.py:

```
import pymysql
import os
import json

# Conexión con la base de datos
def conectar_db():
    return pymysql.connect(
        host=os.environ["DB_HOST"],
        user=os.environ["DB_USER"],
        password=os.environ["DB_PASS"],
        database=os.environ["DB_NAME"],
        cursorclass=pymysql.cursors.DictCursor
    )

def lambda_handler(event, context):
    method = event.get("requestContext", {}).get("http", {}).get("method", "GET")
    print("Método recibido:", event.get("httpMethod"))

    if not method:
        print("⚠ No se recibió httpMethod en el evento. Evento completo:")
        print(json.dumps(event, indent=2))
        return {"statusCode": 400, "body": json.dumps({"error": "Método HTTP no especificado"})}

    body = event.get("body")
    print("Cuerpo crudo:", event.get("body"))
    if body and isinstance(body, str):
        body = json.loads(body)

    conn = conectar_db()

    try:
        with conn.cursor() as cursor:

            # Crear una nueva ubicación
            if method == "POST":
                cursor.execute("""
                    INSERT INTO Ubicacion (nombre, direccion, ciudad, pais, planta, responsable)
                    VALUES (%s, %s, %s, %s, %s, %s)
                """, (
                    body["nombre"],
                    body["direccion"],
                    body["ciudad"],
                    body["pais"],
                    body["planta"],
                    body["responsable"]
                ))
                conn.commit()
                return {"statusCode": 201, "body": json.dumps({"mensaje": "Ubicación creada correctamente."})}

            # Obtener una o todas las ubicaciones
            elif method == "GET":
                nombre = event.get("queryStringParameters", {}).get("nombre")
                if nombre:
                    cursor.execute("SELECT * FROM Ubicacion WHERE nombre = %s", (nombre,))
                    resultado = cursor.fetchone()
                else:
                    cursor.execute("SELECT * FROM Ubicacion")
                    resultado = cursor.fetchall()
                return {"statusCode": 200, "body": json.dumps(resultado, default=str)}

            # Actualizar una ubicación
            elif method == "PUT":
                cursor.execute("""
                    UPDATE Ubicacion
                    SET direccion=%s, ciudad=%s, pais=%s, planta=%s, responsable=%s
                    WHERE nombre=%s
                """, (
                    body["direccion"],
                    body["ciudad"],
                    body["pais"],
                    body["planta"],
                    body["responsable"],
                    body["nombre"]
                ))
                conn.commit()
                return {"statusCode": 200, "body": json.dumps({"mensaje": "Ubicación actualizada correctamente."})}

            # Eliminar una ubicación
            elif method == "DELETE":
                nombre = event.get("queryStringParameters", {}).get("nombre")
                cursor.execute("DELETE FROM Ubicacion WHERE nombre = %s", (nombre,))
                conn.commit()
                return {"statusCode": 200, "body": json.dumps({"mensaje": "Ubicación eliminada correctamente."})}

            # Método no soportado
            else:
                return {"statusCode": 405, "body": json.dumps({"error": "Método HTTP no permitido."})}

    except Exception as e:
        return {"statusCode": 500, "body": json.dumps({"error": str(e)})}
    finally:
        conn.close()
```

## gestion\_activo.py:

```
import pymysql
import os
import json

# Conexión con la base de datos
def conectar_db():
    return pymysql.connect(
        host=os.environ["DB_HOST"],
        user=os.environ["DB_USER"],
        password=os.environ["DB_PASS"],
        database=os.environ["DB_NAME"],
        cursorclass=pymysql.cursors.DictCursor
    )

def lambda_handler(event, context):
    method = event.get("requestContext", {}).get("http", {}).get("method", "GET")
    print("Método recibido:", event.get("httpMethod"))

    if not method:
        print("⚠ No se recibió httpMethod en el evento. Evento completo:")
        print(json.dumps(event, indent=2))
        return {"statusCode": 400, "body": json.dumps({"error": "Método HTTP no especificado"})}

    body = event.get("body")
    print("Cuerpo crudo:", event.get("body"))
    if body and isinstance(body, str):
        body = json.loads(body)

    conn = conectar_db()

    try:
        with conn.cursor() as cursor:

            # Crear nuevo activo
            if method == "POST":
                tipo = body["tipo"]
                cursor.execute("""
                    INSERT INTO Activo (id_activo, tipo, estado, coste, recurrente, fecha_adquisicion, nombre_proveedor,
email_propietario)
                    VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
                    """, (
                    body["id_activo"],
                    tipo,
                    body["estado"],
                    body["coste"],
                    body["recurrente"],
                    body["fecha_adquisicion"],
                    body["nombre_proveedor"],
                    body["email_propietario"]
                ))

            if tipo == "hardware":
                cursor.execute("""
                    INSERT INTO Hardware (id_activo, n_serie, modelo, marca, tipo, ubicacion_nombre)
                    VALUES (%s, %s, %s, %s, %s, %s)
                    """, (
                    body["id_activo"],
                    body["n_serie"],
                    body["modelo"],
                    body["marca"],
                    body["tipo_hardware"],
                    body["ubicacion_nombre"]
                ))

            elif tipo == "licencia":
                cursor.execute("""
                    INSERT INTO Licencia (id_activo, nombre_licencia, producto, numero_licencia)
                    VALUES (%s, %s, %s, %s)
                    """, (
                    body["id_activo"],
                    body["nombre_licencia"],
                    body["producto"],
                    body["numero_licencia"]
                ))

            elif tipo == "linea_telefonica":
                cursor.execute("""
                    INSERT INTO Linea_Telefonica (id_activo, numero_tlf, operadora, plan, pin, puk, sim, id_hardware)
                    VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
                    """, (
                    body["id_activo"],
                    body["numero_tlf"],
                    body["operadora"],
                    body["plan"],
                    body["pin"],
                    body["puk"],
                    body["sim"],
                    body["id_hardware"]
                ))

        conn.commit()
        return {"statusCode": 201, "body": json.dumps({"mensaje": "Activo creado correctamente."})}
```

```

# Obtener uno o todos los activos
elif method == "GET":
    id_activo = event.get("queryStringParameters", {}).get("id_activo")

    if id_activo:
        # Obtener tipo del activo
        cursor.execute("SELECT tipo FROM Activo WHERE id_activo = %s", (id_activo,))
        row = cursor.fetchone()

        if not row:
            return {"statusCode": 404, "body": json.dumps({"error": "Activo no encontrado"})}

        tipo = row["tipo"]

        if tipo == "hardware":
            cursor.execute("""
                SELECT
                    a.id_activo, a.tipo AS tipo_activo, a.estado, a.coste, a.recurrente,
                    a.fecha_adquisicion, a.nombre_proveedor, a.email_propietario,
                    h.n_serie, h.modelo, h.marca, h.tipo AS tipo_hardware, h.ubicacion_nombre
                FROM Activo a
                JOIN Hardware h ON a.id_activo = h.id_activo
                WHERE a.id_activo = %s
            """, (id_activo,))
            resultado = {"Activo_Hardware": [cursor.fetchone()]}

        elif tipo == "licencia":
            cursor.execute("""
                SELECT
                    a.id_activo, a.tipo AS tipo_activo, a.estado, a.coste, a.recurrente,
                    a.fecha_adquisicion, a.nombre_proveedor, a.email_propietario,
                    l.nombre_licencia, l.producto, l.numero_licencia
                FROM Activo a
                JOIN Licencia l ON a.id_activo = l.id_activo
                WHERE a.id_activo = %s
            """, (id_activo,))
            resultado = {"Activo_Licencia": [cursor.fetchone()]}

        elif tipo == "linea_telefonica":
            cursor.execute("""
                SELECT
                    a.id_activo, a.tipo AS tipo_activo, a.estado, a.coste, a.recurrente,
                    a.fecha_adquisicion, a.nombre_proveedor, a.email_propietario,
                    lt.numero_tlf, lt.operadora, lt.plan, lt.pin, lt.puk, lt.sim, lt.id_hardware
                FROM Activo a
                JOIN Linea_Telefonica lt ON a.id_activo = lt.id_activo
                WHERE a.id_activo = %s
            """, (id_activo,))
            resultado = {"Activo_Linea_Telefonica": [cursor.fetchone()]}

        else:
            return {"statusCode": 400, "body": json.dumps({"error": "Tipo de activo no válido"})}

    return {"statusCode": 200, "body": json.dumps(resultado, default=str)}

else:
    # Obtener todos los activos
    cursor.execute("""
        SELECT
            a.id_activo, a.tipo AS tipo_activo, a.estado, a.coste, a.recurrente,
            a.fecha_adquisicion, a.nombre_proveedor, a.email_propietario,
            h.n_serie, h.modelo, h.marca, h.tipo AS tipo_hardware, h.ubicacion_nombre
        FROM Activo a
        JOIN Hardware h ON a.id_activo = h.id_activo
    """)
    hardware = cursor.fetchall()

    cursor.execute("""
        SELECT
            a.id_activo, a.tipo AS tipo_activo, a.estado, a.coste, a.recurrente,
            a.fecha_adquisicion, a.nombre_proveedor, a.email_propietario,
            l.nombre_licencia, l.producto, l.numero_licencia
        FROM Activo a
        JOIN Licencia l ON a.id_activo = l.id_activo
    """)
    licencias = cursor.fetchall()

    cursor.execute("""
        SELECT
            a.id_activo, a.tipo AS tipo_activo, a.estado, a.coste, a.recurrente,
            a.fecha_adquisicion, a.nombre_proveedor, a.email_propietario,
            lt.numero_tlf, lt.operadora, lt.plan, lt.pin, lt.puk, lt.sim, lt.id_hardware
        FROM Activo a
        JOIN Linea_Telefonica lt ON a.id_activo = lt.id_activo
    """)
    lineas = cursor.fetchall()

    return {
        "statusCode": 200,
        "body": json.dumps({
            "Activo_Hardware": hardware,
            "Activo_Licencia": licencias,
            "Activo_Linea_Telefonica": lineas
        }, default=str)
    }
}

```

```

# Actualizar activo
elif method == "PUT":
    tipo = body["tipo"]
    cursor.execute("""
        UPDATE Activo
        SET tipo=%s, estado=%s, coste=%s, recurrente=%s, fecha_adquisicion=%s,
            nombre_proveedor=%s, email_propietario=%s
        WHERE id_activo=%s
    """, (
        tipo,
        body["estado"],
        body["coste"],
        body["recurrente"],
        body["fecha_adquisicion"],
        body["nombre_proveedor"],
        body["email_propietario"],
        body["id_activo"]
    ))

    if tipo == "hardware":
        cursor.execute("""
            UPDATE Hardware
            SET n_serie=%s, modelo=%s, marca=%s, tipo=%s, ubicacion_nombre=%s
            WHERE id_activo=%s
        """, (
            body["n_serie"],
            body["modelo"],
            body["marca"],
            body["tipo_hardware"],
            body["ubicacion_nombre"],
            body["id_activo"]
        ))

    elif tipo == "licencia":
        cursor.execute("""
            UPDATE Licencia
            SET nombre_licencia=%s, producto=%s, numero_licencia=%s
            WHERE id_activo=%s
        """, (
            body["nombre_licencia"],
            body["producto"],
            body["numero_licencia"],
            body["id_activo"]
        ))

    elif tipo == "linea_telefonica":
        cursor.execute("""
            UPDATE Linea_Telefonica
            SET numero_tlf=%s, operadora=%s, plan=%s, pin=%s, puk=%s, sim=%s, id_hardware=%s
            WHERE id_activo=%s
        """, (
            body["numero_tlf"],
            body["operadora"],
            body["plan"],
            body["pin"],
            body["puk"],
            body["sim"],
            body["id_hardware"],
            body["id_activo"]
        ))

    conn.commit()
    return {"statusCode": 200, "body": json.dumps({"mensaje": "Activo y subtipo actualizados correctamente."})}


# Eliminar activo (el subtipo se elimina en cascada por la FK)
elif method == "DELETE":
    id_activo = event.get("queryStringParameters", {}).get("id_activo")
    cursor.execute("DELETE FROM Activo WHERE id_activo = %s", (id_activo,))
    conn.commit()
    return {"statusCode": 200, "body": json.dumps({"mensaje": "Activo eliminado correctamente."})}

else:
    return {"statusCode": 405, "body": json.dumps({"error": "Método HTTP no permitido."})}

except Exception as e:
    return {"statusCode": 500, "body": json.dumps({"error": str(e)})}
finally:
    conn.close()

```

Este documento esta firmado por



<b>Firmante</b>	CN=tfgm.fi.upm.es, OU=CCFI, O=ETS Ingenieros Informaticos - UPM, C=ES
<b>Fecha/Hora</b>	Wed Jun 04 19:58:56 CEST 2025
<b>Emisor del Certificado</b>	EMAILADDRESS=camanager@etsiinf.upm.es, CN=CA ETS Ingenieros Informaticos, O=ETS Ingenieros Informaticos - UPM, C=ES
<b>Numero de Serie</b>	561
<b>Metodo</b>	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)