.UBAfiuba
FACULTAD DE INGENIERÍA

Solución IoT para robot de exploración ambiental de datos críticos con almacenamiento en blockchain

Esp. Ing. Gonzalo Carreño

Carrera de Maestría en Internet de las Cosas

Director: Esp. Ing. Sergio Alberino (UTN-FRBA)

Jurados:

Jurado 1 (pertenencia)
Jurado 2 (pertenencia)
Jurado 3 (pertenencia)

Ciudad de Buenos Aires, abril de 2025

Resumen

En este trabajo se presenta la implementación de un proyecto personal que consiste en el desarrollo de una solución de Internet de las Cosas aplicada a un robot de exploración ambiental. Las implementación asegura la inmutabilidad y transparencia de los datos obtenidos por el robot.

Para su implementación se utilizaron conceptos y herramientas tales como el desarrollo de sistemas embebidos, mensajería asincrónica, almacenamiento y procesamiento distribuido en la nube, entre otros.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción general

Este capítulo presenta la motivación, alcance, objetivos y requerimientos del producto en el marco del estado del arte y su importancia en la industria.

1.1. Estado del arte

1.1.1. Introducción a las soluciones IoT

Las soluciones IoT (*Internet of Things* o Internet de las Cosas) se basan en la conexión de dispositivos físicos con aplicaciones informáticas para recopilar, transmitir y analizar datos en *streaming* y de forma *batch*. Esto mejora la automatización, observabilidad y toma de decisiones en diversos casos de uso.

Su arquitectura estándar en general incluye dispositivos y sensores para capturar datos, conectividad de red (Wi-Fi [1], 5G [2], LoRaWAN [3]) para su transmisión, una plataforma *backend* en la nube formada por sistemas distribuidos para el almacenamiento, procesamiento y análisis de datos, y una interfaz de usuario para la visualización de resultados. En ocasiones también puede incluir sistemas de publicación y distribuición de eventos en *streaming*. En general, las soluciones IoT generalmente estan organizadas en las siguientes capas:

- Capa de percepción (*Sensing Layer*): esta capa, la más cercana al entorno físico, captura datos del ambiente (temperatura, humedad, etc.) mediante dispositivos IoT y sensores. Generalmente, se usan sistemas embebidos con sensores y actuadores para interactuar con el entorno.
- Capa de red (*Network Layer*): se encarga de la transmisión de datos desde los dispositivos hasta los sistemas de procesamiento. Aquí es donde ocurre la conectividad mediante diversos protocolos de comunicación usando tecnologías inalámbricas (tales como Wi-Fi, Bluetooth [4], Zigbee [5], LoRaWAN, NB-IoT [6], etc.) y protocolos de red (por ejemplo, MQTT, CoAP, HTTP, etc.)
- Capa de procesamiento de borde (*Edge Computing Layer*): procesa datos cerca de donde se generan para reducir la latencia y el tráfico hacia la nube. Se toman decisiones inmediatas y solo los datos relevantes se envían a niveles superiores.
- Capa de almacenamiento y procesamiento *cloud* (*Data Storage/Cloud Layer*): almacena y procesa grandes volúmenes de datos recopilados en la nube, lo que permite realizar análisis más profundos, modelado de datos y aprendizaje automático. Se utilizan herramientas *cloud* (como AWS IoT Core [7], Azure Iot Hub [8], etc.) y Big Data.

- Capa de aplicación (*Application Layer*): es la interfaz que permite a los usuarios interactuar con el sistema IoT. Aquí se presentan los datos de manera visual o se automatizan acciones basadas en la información recibida. Se utilizan tecnologías web y mobile, orientadas a eventos *streaming* o dashboards para visualizar reportes *batch*.
- Capa de seguridad (*Security Layer*): esta capa es transversal a las capas anteriores y tiene como función segurar la protección de datos, dispositivos y redes en todas las capas del sistema IoT. Es fundamental para evitar vulnerabilidades y ataques. Utiliza algoritmos de encriptación (como por ejemplo TLS/SSL y AES), protocolos de seguridad (como OAuth, OpenID Connect, etc).

1.1.2. Soluciones IoT que utilizan robots exploradores

Existen casos de uso de IoT en los cuales se utilizan robots exploradores como dispositivos físicos para la recopilación de datos en la capa de percepción. Los robots exploradores son dispositivos robotizados capaces de moverse de forma autónoma y/o controlados a distancia que utilizan sensores avanzados, inteligencia artificial y comunicación en tiempo real para navegar y monitorear condiciones ambientales en entornos peligrosos, como minas, plataformas petrolíferas, espacios confinados o áreas afectadas por desastres, entre otros. En agricultura, pueden inspeccionar cultivos; en medio ambiente, pueden monitorear la calidad del aire, del agua; en el espacio y océanos, son capaces de explorar lugares inaccesibles para el ser humano.

Tanto en el ámbito académico como en la industria, existen diversos trabajos, proyectos e implementaciones comerciales de soluciones IoT que utilizan robots para mejorar la seguridad, la eficiencia y la toma de decisiones basada en datos. Por ejemplo:

- En Lotus Mountain, Jilin, China, se implementó un sistema de seguridad para estaciones de esquí que utiliza perros robóticos equipados con sensores y tecnología de imágenes 3D. Estos robots patrullan las pistas para identificar peligros como desprendimientos y bloqueos, mejorando así la seguridad de los esquiadores [9].
- El implementado por el Ayuntamiento de Bilbao [10] para la inspección y mantenimiento de redes de saneamiento, que por medio de drones y robots, busca mejorar la eficiencia operativa y la seguridad de los trabajadores al reducir la necesidad de intervenciones humanas en entornos subterráneos y potencialmente peligrosos.
- El proyecto Tecnobosque [11] en Cuenca, España, que utiliza drones equipados con sensores e inteligencia artificial para crear cortafuegos preventivos y reducir significativamente las hectáreas de bosques en casos de incendios.
- Spot [12], desarrollado por Boston Dynamics, un robot explorador cuadrupedo de propósito general capaz de explorar, almacenar y enviar información en tiempo real.

- BIKE [13], desarrollado por Waygate Technologies, un robot con ruedas magnéticas, muy utilizado en la industria de petróleo y gas entre otras, capaz de desplazarse por el interior de tuberías para poder realizar inspecciones y comunicar hallazgos.
- El prototipo robótico de exploración minera publicado en varios artículos [14], [15], e impulsado por el Instituto de Automática de la Facultad de Ingeniería de la Universidad Nacional de San Juan en el marco de un convenio con la Comisión Nacional de Energía Atómica y el Gobierno argentino [16].
- El robot de exploración terrestre denominado Geobot [17] desarrollado por los ingenieros Nelson Dario García Hurtado y Melvin Andrés González Pino, de la universidad de Pamplona, capaz de realizar reconocimiento de zonas y manipulación de muestras de manera autónoma o asistida.
- El robot minero MIN-SIS 1.0 SDG-STR [18] desarrollado por los ingenieros Hernán L. Helguero Velásquez y Rubén Medinaceli Tórrez de la Universidad Técnica de Oruro, capaz de detectar gases, almacenar datos locales y enviar video e imágenes al puesto de mando.

1.1.3. Soluciones IoT que utilizan blockchain

Para la exploración y monitoreo de áreas ambientalmente sensibles (reservas naturales, sitios de desastre ecológico), la recopilación de datos críticos (contaminación, temperatura, etc.) requiere el almacenamiento en un sistema que garantice la integridad y transparencia, como una cadena de bloques.

Una arquitectura blockchain [19] se basa en el agrupamiento de transacciones que luego de ser procesadas, son almacenadas en bloques encadenados de forman distribuida e inmutable, entre los nodos de una red. Esta estructura de datos se conoce como una cadena de bloques y sus datos almacenados forma un *distributed ledger* (o asiento contable distribuido). De esta manera, como los datos forman registros que no se pueden modificar una vez creados, se puede asegurar la inmutabilidad, y como el almacenamiento y procesamiento de la red se encuentran distribuidos, se puede garantizar su transparencia.

La mayoría de las redes blockchain constan de ciertas tecnologías para la implementación de código ejecutable en la misma red, que aunque su nombre puede cambiar dependiendo de la red, usualmente se los conoce como *smart contracts* [20]. La ejecución de estos componentes es realizada por los nodos de la red en el proceso que se conoce como minería o validación. La forma de interactuar con los *smart contracts* se realiza a través de otro componente conocido como dApps *(de-centralized applications)* [21] que por medio del uso de ciertas tecnologías invocan a estos componentes para almacenar y obtener datos en y desde el *distributed ledger*.

El uso de blockchain en arquitecturas IoT ofrece ventajas como descentralización, cifrado, seguridad y consenso, lo que aumenta la trazabilidad, la transparencia y la automatización mediante *smart contracts*.

Existen en la industria varias implementaciones de casos de uso IoT en los que se ha utilizado blockchain como por ejemplo:

- La solución basada en blockchain implementada por Walmart [22] para mejorar la trazabilidad de productos alimenticios en su cadena de suministro.

Al integrar dispositivos IoT, la empresa puede monitorear en tiempo real variables como temperatura y humedad durante el transporte y almacenamiento de productos perecederos. Estos datos se registran en una blockchain para garantizar la inmutabilidad y transparencia de la información.

- La solución implementada por ScanTrust [23] que utiliza códigos QR seguros para conectar productos físicos con el entorno digital. Al integrar IoT y blockchain, permite a las empresas y consumidores autenticar productos y rastrear su origen y cadena de suministro en tiempo real. Los códigos QR, impresos en los envases, se escanean con dispositivos móviles para proporcionar información detallada y asegurar la autenticidad del producto.
- La solución implementada por la empresa Saltoki en colaboración con EcoMT [24], que permite monitorizar y gestionar el consumo energético, para certificar la producción renovable y los ahorros obtenidos mediante tecnología blockchain.

1.2. Motivación del trabajo

La motivación del presente trabajo fue primeramente volcar y unificar en un emprendimiento personal los conceptos aprendidos en la Maestría en Internet de las Cosas.

Se diseñó una arquitectura robusta y flexible, con el fin de ser implementada en la industria, donde la integración de sistemas embebidos con blockchain es crucial para el almacenamiento transparente e inalterable de datos sensibles.

Asimismo, se procuró crear un producto que pudiera fomentar el conocimiento público y el estado del arte de proyectos de código abierto relacionados con soluciones IoT integradas a blockchain en Argentina.

1.3. Alcance y objetivos

El objetivo del trabajo es la integración del dispositivo robótico de exploración ambiental [25] desarrollado en el marco de la Carrera de Especialización en Sistemas Embebidos, con un sistema *backend* desplegado en la nube pública [26], y una red blockchain [19] a fin de poder asegurar la inmutabilidad y transparencia de las lecturas ambientales.

A continuación, se detallan las funcionalidades incluidas en el alcance del trabajo:

- La publicación del endpoint MQTT [27] para la recepción de los datos enviados por el robot.
- La adaptación del sistema embebido del robot de exploración ambiental para la conexión segura con el *backend* vía MQTT.
- La arquitectura e implementación de los sistemas *backend* y el modelo de datos necesario para el almacenamiento de las mediciones enviadas por el robot.
- La arquitectura, implementación y despliegue de la dApp [21] y *smart contracts* [20] necesarios para el almacenamiento de las mediciones en una red blockchain.

- La definición de métricas agregadas de valor y posterior arquitectura e implementación de los sistemas analíticos para procesar de forma *batch* y/o *real-time* utilizando herramientas de procesamiento paralelo basadas en Big Data.
- La implementación de la interfaz gráfica para poder visualizar los datos enviados y analíticas calculadas.

1.4. Requerimientos del producto

A continuación, se listan los requerimientos del producto:

1. Requerimientos funcionales
   a) El robot de exploración ambiental debe poder enviar a la plataforma datos de mediciones de parámetros ambientales, incluyendo los datos de fecha, hora, localización geográfica (que puede ser implementada como un valor *mock*) y la categorización si es o no un valor crítico.
   b) El robot de exploración ambiental debe incorporar una lógica para categorizar los valores medidos de cada parámetro ambiental como valores críticos si:
      1) Representan un máximo o mínimo global sensado hasta el momento.
      2) Representan un máximo o mínimo local durante el último día.
   c) La solución a desarrollar debe poder recibir y almacenar las mediciones de parámetros ambientales enviadas por el robot.
   d) Los datos considerados críticos deben ser almacenados en un sistema inmutable.
   e) La solución a desarrollar debe poder procesar las mediciones de parámetros ambientales enviadas por el robot para generar métricas de valor para el usuario de negocio.
   f) La solución a desarrollar debe brindar dos *frontend* con interfaz web:
      1) El *frontend* para el usuario de negocio.
      2) El *frontend* para el usuario administrador.
   g) El *frontend* para el usuario de negocio debe proveer métricas para visualizar:
      1) Las lecturas históricas almacenadas.
      2) Agregaciones (máximo, mínimo, promedio, etc.) de cada parámetro ambiental agrupado por frecuencias (ventanas de tiempo) y coordenadas geográficas.
      3) Las referencias a los datos persistidos en blockchain.
   h) El *frontend* para el usuario de administración debe permitir:
      1) Acceder a los diferentes recursos utilizados por la herramienta (topics MQTT, *smart contracts*, *buckets*, etc.).

2) Resetear valores y estado.

2. Requerimientos no funcionales

   a) La solución a desarrollar debe contar con al menos un *backend* de procesamiento y acceso a datos operacionales para la lógica de negocio.

   b) La solución a desarrollar debe contar con al menos un *backend* de acceso, procesamiento, almacenamiento de datos analíticos para la generación de métricas.

   c) El envío de los valores ambientales censados al *backend* debe ser mediante MQTT.

   d) Las lecturas ambientales categorizadas como críticas deben ser almacenadas en blockchain para garantizar fiabilidad e inmutabilidad.

   e) La gestión de datos almacenados en blockchain debe ser implementada mediante *smart contracts* desplegados en la red.

   f) La interacción con los *smart contracts* debe realizarse desde una dApp.

   g) Los sistemas de transferencia y almacenamiento de datos utilizados deben contar con seguridad, permitiendo encriptación, autenticación y autorización.

3. Requerimientos de documentación

   a) Video demostrativo.

   b) Documentación de arquitectura técnica del diseño del sistema.

   c) Manual de usuario.

   d) Memoria final.

4. Requerimientos de testing

   a) Se deben incluir tests de unitarios de componentes.

   b) Se deben incluir tests funcionales (*smoke test*) del producto general.

5. Requerimientos opcionales

   a) De infraestructura y despliegue:

      1) Se permite realizar el despliegue de la dApp en un IPFS (preferentemente) o en la nube.

      2) Se permite la incorporación de nuevo hardware al robot para la captura de datos adicionales.

      3) Se permite agregar automatización para la creación de la infraestructura como código.

   b) De datos:

      1) Se permite almacenar cualquier otro dato adicional sensado o derivado.

      2) Se permite agregar cualquier implementación de gobierno de datos.

3) Se permite almacenar cualquier otra métrica o gráfico de explotación de datos adicional.

Capítulo 2

Introducción específica

En este capítulo se presenta una breve introducción técnica a las herramientas de hardware y software utilizadas en el trabajo.

2.1. Tecnologías de hardware y firmware utilizadas

2.1.1. Robot de exploración ambiental

Como dispositivo físico en la capa de percepción, se utilizó el robot de exploración ambiental desarrollado en el marco de la Carrera de Especialización en Sistemas Embebidos [25]. En la figura 2.1 se puede apreciar una foto del robot.

FIGURA 2.1. Robot de exploración ambiental.

El robot de exploración ambiental es un sistema embebido desarrollado con el marco de desarrollo ESP-IDF [28] de Espressif Systems que utiliza un microcontrolador ESP32 [29] y FreeRTOS [30] como sistema operativo. Está diseñado para explorar terrenos mediante control de joystick y para obtener parámetros ambientales como luminosidad, presión atmosférica, temperatura y humedad.

2.1.2. Microcontrolador ESP32

ESP32 [29] es una serie de microcontroladores embebidos en un chip con Wi-Fi y Bluetooth integrados, de bajo costo y consumo, desarrollado por *Espressif Systems*. Emplea dos núcleos Xtensa® 32-bit LX6 CPU, incluye interruptores de antena, amplificador de potencia, amplificador de recepción de bajo ruido, un co-procesador ULP (*Ultra Low Power*), módulos de administración de energía y diversos periféricos.

2.1.3. Marco de trabajo ESP-IDF

El marco de trabajo ESP-IDF[28] de Espressif Systems proporciona un conjunto de recursos para facilitar el desarrollo de *firmware* para ESP32. Utiliza FreeRTOS como sistema operativo base y brinda varias utilidades de desarrollo y bibliotecas de código para poder acceder y controlar los diferentes recursos del microcontrolador.

2.1.4. FreeRTOS

FreeRTOS (*Free Real-Time Operating System*) [30] es un sistema operativo en tiempo real de código abierto, ligero y diseñado para microcontroladores y sistemas embebidos. Permite ejecutar múltiples tareas concurrentemente mediante planificación por prioridades, ofreciendo servicios como manejo de tareas, colas, temporizadores, semáforos y sincronización. Es ampliamente usado en aplicaciones IoT y sistemas críticos donde se requiere una respuesta rápida y predecible, y se puede portar fácilmente a una gran variedad de arquitecturas de hardware como ARM Cortex-M, RISC-V, entre otras.

2.2. Tecnologías de Amazon Web Services utilizadas

2.2.1. Amazon Web Services

AWS (*Amazon Web Services*) [31] es una de las principales plataformas de servicios en la nube pública proporcionada por Amazon, que ofrece una amplia gama de productos y herramientas para computación, almacenamiento, bases de datos, redes, inteligencia artificial, seguridad y herramientas de desarrollo.

2.2.2. AWS IoT Core

AWS IoT Core [7] es un servicio totalmente administrado de AWS que permite conectar dispositivos IoT a la nube de manera segura y confiable. Proporciona una infraestructura escalable para recopilar, procesar y analizar datos de dispositivos en tiempo real, así como para interactuar con otros servicios de AWS a los que se redirigen los mensajes recibidos mediante la configuración de reglas. Tiene soporte para varios protocolos de comunicaciones, entre los que se destacan principalmente MQTT, HTTP/S, WebSockets y LoRaWAN.

2.2.3. AWS App Runner

AWS App Runner [32] es un servicio completamente administrado de AWS que permite implementar y ejecutar aplicaciones web y servicios de forma rápida, sin tener que gestionar instancias de infraestructura. Está diseñado para simplificar el proceso de implementación y escalado automático de aplicaciones y permite aplicaciones directamente desde el código fuente o desde contenedores Docker [33].

2.2.4. AWS Glue

AWS Glue [34] es un servicio totalmente administrado de AWS diseñado para facilitar la extracción, transformación y carga de datos (ETL) en la nube. Permite a los usuarios descubrir, preparar y combinar datos de múltiples fuentes para su

análisis y almacenamiento en data lakes, data warehouses o bases de datos. Este servicio es útil para proyectos de Big Data y análisis de datos, ya que ofrece un catálogo de datos, asistencia para la generación de código ETL y ejecución de trabajos de procesamiento paralelo.

2.2.5. AWS S3

AWS S3 (*Simple Storage Service*) [35] es un servicio de almacenamiento de objetos totalmente administrado por AWS. Permite almacenar y recuperar cualquier cantidad de datos de forma segura, escalable y económica desde cualquier lugar. Es ideal para datos no estructurados de gran volumen, sitios web estáticos, archivos multimedia y copias de seguridad, entre otros.

2.2.6. AWS Athena

AWS Athena [36] es un servicio de análisis de datos sin servidor de AWS que permite consultar directamente datos almacenados en Amazon S3 y esquemas definidos en Glue, utilizando SQL estándar. No requiere configuración ni administración de servidores, y es ideal para analizar grandes volúmenes de datos de forma rápida y económica.

2.2.7. AWS SNS

Amazon SNS (*Amazon Simple Notification Service*) [37] es un servicio de mensajería totalmente gestionado que permite el envío de notificaciones desde aplicaciones a otros sistemas distribuidos, servicios en la nube o usuarios finales. Funciona bajo un modelo de publicación-suscripción (pub/sub), donde los productores de mensajes (publicadores) envían mensajes a un *topic*, y los consumidores (suscriptores) los reciben. Amazon SNS es ideal para aplicaciones que requieren alta disponibilidad, baja latencia y escalabilidad para distribuir eventos o alertas en tiempo real.

2.2.8. AWS SQS

Amazon SQS (*Amazon Simple Queue Service*) [38] es un servicio de colas de mensajes completamente gestionado que permite desacoplar y escalar componentes de aplicaciones distribuidas, eliminando la necesidad de administrar infraestructura de mensajería. SQS actúa como intermediario entre productores y consumidores de mensajes, almacenando mensajes de forma temporal hasta que los sistemas receptores estén listos para procesarlos. Amazon SQS es ideal para construir sistemas resilientes, desacoplados y altamente escalables, como pipelines de procesamiento de datos, flujos de trabajo asincrónicos o integración entre microservicios.

2.2.9. AWS Secrets Manager

AWS Secrets Manager [39] es un servicio gestionado que permite almacenar, gestionar, rotar y recuperar credenciales, claves API, contraseñas y otros secretos de forma segura. Brinda una API que permite a las aplicaciones recuperar secretos de forma programática, eliminando la necesidad de codificar credenciales directamente en el código fuente o archivos de configuración. Es compatible con varios servicios de AWS para la rotación automática de credenciales, y se integra muy

facilmente con todos los servicios de AWS, mejorando la seguridad, el cumplimiento y la gestión centralizada de secretos.

2.2.10. AWS Lambda

AWS Lambda [40] es un servicio de computación serverless ideal para crear aplicaciones escalables, desacopladas, reactivas y de baja latencia, que permite ejecutar código en respuesta a eventos sin necesidad de aprovisionar ni administrar servidores. Permite cargar funciones escritas en lenguajes como Python, Node.js, Java, entre otros, y Lambda se encarga automáticamente de ejecutar el código cuando se produce un evento, escalarlo según la demanda y gestionar la infraestructura subyacente. Lambda se integra de forma nativa con otros servicios de AWS como S3, DynamoDB, API Gateway, SNS y SQS, lo que permite construir arquitecturas basadas en eventos de manera sencilla y eficiente.

2.2.11. AWS DynamoDB

Amazon DynamoDB [41] es un servicio de base de datos NoSQL totalmente gestionado que ofrece almacenamiento de datos clave-valor y de documentos con alta disponibilidad, rendimiento escalable y baja latencia. Está diseñado para manejar cargas de trabajo intensivas y aplicaciones que requieren respuestas en tiempo real, ideal para casos de uso IoT por ese motivo. DynamoDB permite escalar automáticamente la capacidad de lectura y escritura, y ofrece opciones avanzadas como almacenamiento en caché, backup y restauración bajo demanda, replicación global y soporte para transacciones ACID.

2.2.12. AWS IAM

AWS IAM (*AWS Identity and Access Management*) [42] es un servicio que permite gestionar de forma segura el acceso a los recursos de AWS mediante la creación y control de usuarios, grupos, roles y políticas de permisos. Con IAM se pueden definir quién puede acceder a qué recursos, en qué condiciones y con qué nivel de privilegio, aplicando el principio de mínimo privilegio. IAM permite la autenticación de identidades y la autorización de acciones tanto para usuarios internos como para aplicaciones y servicios de AWS. Además, admite políticas detalladas en formato JSON, control de acceso basado en atributos (ABAC) [43], autenticación multifactor (MFA) [44] y federación de identidades con directorios corporativos o proveedores externos.

2.2.13. AWS CloudWatch

Amazon CloudWatch [45] es un servicio de monitoreo y observabilidad totalmente gestionado que permite recopilar, visualizar y analizar métricas, logs y eventos de recursos y aplicaciones en AWS. Facilita la supervisión del rendimiento, la detección de anomalías y la generación de alertas en tiempo real a partir de datos generados por servicios como EC2, Lambda, DynamoDB, RDS, entre otros, así como por aplicaciones personalizadas mediante agentes o APIs.

2.3. Tecnologías de propósito general utilizadas

2.3.1. Node.js

Node.js [46] es un entorno de ejecución de código abierto basado en el motor V8 de Chrome, utilizado principalmente como servidor web. Emplea un modelo *single-thread* de *event loop* con I/O no bloqueante para gestionar múltiples solicitudes y paralelismo de tareas mediante callbacks asíncronos.

2.3.2. MQTT

MQTT (*Message Queuing Telemetry Transport*) [27] es un protocolo de comunicación asíncrono, ligero y orientado a mensajes, diseñado para dispositivos con recursos limitados y redes de baja ancho de banda. Ampliamente utilizado en IoT, permite la transmisión de datos en tiempo real entre dispositivos, sensores y aplicaciones mediante un modelo publish/subscribe y topics, donde los clientes se conectan a un broker para publicar o recibir notificaciones.

2.3.3. Swagger

Swagger [47] es un conjunto de herramientas para diseñar, documentar y consumir APIs RESTful. Utiliza el formato OpenAPI Specification (OAS) para describir de forma estandarizada la estructura de una API, permitiendo la generación automática de documentación interactiva, clientes SDK y pruebas.

2.4. Lenguajes de programación utilizados

2.4.1. Lenguaje de programación Python

Python es un lenguaje de programación de alto nivel, interpretado y de propósito general, conocido por su sintaxis clara y legible que favorece el desarrollo rápido y eficiente. Es ampliamente utilizado en áreas como desarrollo web, automatización, análisis de datos, inteligencia artificial y ciencia de datos, gracias a su gran ecosistema de bibliotecas, amplia documentación y comunidad activa. Fue utilizado en el trabajo en las funciones AWS Lambda.

2.4.2. Lenguaje de programación Javascript

JavaScript es un lenguaje de programación interpretado, de alto nivel y orientado a objetos, diseñado originalmente para el desarrollo web del lado del cliente. Se utiliza tanto en el frontend como en el backend gracias a entornos como Node.js. Fue utilizado en el trabajo para el desarrollo de la dApp como backend Node.js.

2.4.3. Lenguaje de programación C

C es un lenguaje de programación de propósito general, compilado y de bajo nivel, conocido por su eficiencia, velocidad y control directo sobre el hardware. Es ampliamente utilizado en sistemas operativos, software embebido y desarrollo de sistemas donde el rendimiento y el acceso a memoria son críticos. Fue utilizado en el trabajo para el desarrollo del firmware del sistema embebido.

2.4.4. Solidity

Solidity [48] es un lenguaje de programación de alto nivel, orientado a contratos inteligentes, específicamente diseñado para funcionar en la EVM (*Ethereum Virtual Machine*). Fue creado en 2014 por Gavin Wood, Christian Reitwiessner y otros desarrolladores de Ethereum. Su sintaxis es similar a JavaScript, Python y C++, lo que facilita el aprendizaje para desarrolladores familiarizados con esos lenguajes. Fue utilizado en el trabajo para el desarrollo de los *smart contracts*.

2.5. Tecnologías blockchain utilizadas

2.5.1. Ecosistema Ethereum

Ethereum [49] es una red blockchain pública diseñada para el procesamiento de transacciones de forma descentralizada con almacenamiento distribuido, inmutable y de acceso libre (*permissionless*) que se diferencia de otras redes blockchain, como Bitcoin, porque no es solo un *distributed ledger*, sino también una plataforma programable que permite desarrollar, desplegar y ejecutar aplicaciones. Se encuentra formada por distintos tipos de nodos EVM (*Ethereum Virtual Machine*) [50] que tienen como función procesar transacciones y almacenarlas en una estructura de datos basada en una cadena de bloques. Tiene como *token* el Ether, cuyo símbolo es ETH y tiene varios usos, pudiendo ser utilizado como criptomoneda de cambio, entre los usuarios finales de la red, pero también para pagar el *gas fee*, o costo de ejecución de transacciones. Existen varios despliegues de Ethereum disponibles para distintos propósitos. La Mainnet [51] es su red principal para usos productivos. Además existen múltiples *testnets*, o redes de prueba, como Sepolia [52] y Holesky [53] entre otras, disponibles para ser usadas durante el desarrollo y evaluación de soluciones sin necesidad de pagar con fondos reales.

Las aplicaciones desarrolladas en blockchain son los *smart contracts*. Estos componentes permiten ejecutar código directamente en la red para automatizar, verificar y hacer cumplir acuerdos. Pueden ser invocados desde afuera de la blockchain o disparados cuando se producen ciertos eventos. Una vez desplegados en la blockchain, no se pueden modificar, y todas las transacciones quedan registradas públicamente. Se desarrollan en alguno de los lenguajes de programación y SDK soportados, como por ejemplo Solidity y Truffle. Una vez desarrollados, tras el proceso de compilación se obtiene un ABI [54] que especifica el contrato en formato JSON. Posteriormente, se despliega el código binario (no el ABI) en una nueva transacción blockchain.

Para la invocación a los *smart contracts* se utilizan las dApps como componente de abstracción. Estas son aplicaciones que se ejecutan fuera de la blockchain, como por ejemplo un *backend* en Node.js o un *frontend* Javascript, e interactúan con la blockchain a través de ciertas bibliotecas, como por ejemplo, Web3.js [55]. Para poder acceder a la red, y posteriormente invocar el *smart contract*, la dApp necesita utilizar un endpoint RPC publicado por cualquier nodo de la red y disponer de la especificación ABI del *smart contract* obtenido durante su compilación (y no se disponible en la red).

Cada vez que un *smart contract* es invocado, se genera una nueva transacción en la red que debe ser validada y agrupada en un bloque. El proceso de validación de transacciones y generación de bloques, a partir de la versión 2.0 de Ethereum,

utiliza el protocolo PoS (*Proof-of-Stake*) [56] y propone un nuevo bloque aproximadamente cada doce segundos. El primer paso, es la recolección de transacciones enviadas por los usuarios a la red, luego un validador es seleccionado de forma aleatoria para proponer el siguiente bloque, y selecciona aquellas transacciones con tarifas de *gas* mas altas para maximizar su recompensa. Como resultado, el validador seleccionado confecciona un nuevo bloque que incluye las transacciones válidas, el estado actualizado del sistema, el *hash* del bloque anterior y los datos adicionales como la firma del bloque. Finalmente, un comité de validadores seleccionado de forma aleatoria revisa el bloque propuesto y verifica que las transacciones sean válidas, el bloque no esté duplicado o malicioso y sea coherente con el estado de la blockchain. Si más del 66,6 % de los validadores en el comité lo consideran válido, el bloque es agregado a la cadena de bloques de forma permanente.

El proceso de compensación y penalización de PoS retribuye a los validadores por diferentes acciones, con el fin de mantener la integridad y consenso de la red, aplica una técnica llamada *slashing* para la penalización por acciones malisiosas o incorrectas. El validador que propone el bloque recibe recompensas por bloque y tarifas de *gas*. Los validadores que votan correctamente para validar bloques también reciben recompensas proporcionales a su participación. Si el validador no presenta comportamiento malicioso o inactividad, no es penalizado. Sin embargo, los validadores pueden perder parte o todo su capital en *stack* si proponen múltiples bloques en un mismo slot, votan de manera inconsistente, o están inactivos durante largos períodos de tiempo.

2.5.2. Biblioteca Web3.js

Web3.js [55] es una biblioteca de JavaScript que permite interactuar con la blockchain de Ethereum y otros protocolos compatibles con EVM. Proporciona una forma sencilla de conectarse a nodos de Ethereum, realizar transacciones y leer datos de contratos inteligentes, directamente desde aplicaciones web o Node.js.

2.5.3. Ganache

Ganache [57] es una herramienta de desarrollo de Ethereum que permite crear una blockchain local para probar, desarrollar y depurar contratos inteligentes y dApps de forma rápida y segura. Es parte del conjunto de herramientas de Truffle Suite y es ampliamente utilizada por desarrolladores para simular una red Ethereum sin necesidad de usar una red pública como Mainnet o Testnets (Goerli, Sepolia).

2.5.4. Truffle

Truffle [58] es un framework de desarrollo para Ethereum y otras blockchains compatibles con EVM. Es parte de Truffle Suite y proporciona herramientas para compilar, desplegar y probar contratos inteligentes, además de facilitar la gestión de proyectos basados en Web3. Truffle automatiza gran parte del proceso de desarrollo de dApps, reduciendo errores y mejorando la eficiencia.

2.5.5. Alchemy

Alchemy [59] es una plataforma de desarrollo blockchain que proporciona herramientas e infraestructura para crear y gestionar dApps en Ethereum y otras redes compatibles con EVM. Ofrece nodos como servicio y herramientas para facilitar la interacción con la blockchain sin necesidad de que los desarrolladores configuren y mantengan sus propios nodos. En el trabajo actual, se utilizó Alchemy como punto de integración entre la dApp y los Smart Contracts.

2.5.6. Etherscan

Etherscan [60] es un explorador de bloques y plataforma de análisis para la red Ethereum. Permite a los usuarios buscar, verificar y rastrear transacciones, contratos inteligentes, direcciones de billeteras y otros datos en tiempo real. Es una herramienta fundamental para los desarrolladores y usuarios de Web3, ya que ofrece transparencia y acceso abierto a la información almacenada en la blockchain de Ethereum, tanto la Mainnet como las redes de prueba (Sepolia, Holesky, etc).

2.5.7. Metamask

MetaMask [61] es una billetera digital y extensión de navegador (también disponible como aplicación móvil) que permite a los usuarios interactuar con blockchains basadas en Ethereum y otros ecosistemas compatibles con Ethereum, como Binance Smart Chain (BSC) y Polygon. Es una herramienta fundamental para interactuar con aplicaciones descentralizadas (dApps), contratos inteligentes y realizar transacciones de criptomonedas directamente desde el navegador. En el presente trabajo se utilizó para almacenar los fondos en ETH obtenidos a través de *faucets* necesarios para pagar el gas de las transacciones.

2.6. Tecnologías de ingeniería de datos utilizadas

2.6.1. Microsoft Fabric

Microsoft Fabric es una plataforma unificada de analítica de datos en la nube de Azure que integra múltiples herramientas en un entorno gestionado. Está diseñada simplificar la integración, transformación, análisis y visualización de datos. Entre sus componentes principales se encuentran:

- Lakehouse [**link**]: combina las capacidades de un data lake y un data warehouse, permitiendo almacenar y procesar grandes volúmenes de datos estructurados y no estructurados, compatible con Spark, notebooks y consultas SQL.
- Warehouse [**link**]: un entorno de base de datos relacional orientado a análisis empresariales y modelado dimensional, que permite ejecutar consultas SQL sobre los datos almacenados en OneLake.
- Data Factory [**link**]: herramienta de orquestación de flujos datos que permite construir ETL/ELT mediante pipelines y actividades de copiado,.
- Dataflows Gen2 [**link**]: herramienta de transformaciones de datos complejas que brinda uuna interfaz visual sin necesidad de escribir código y utiliza el motor Spark de forma subyacente.

- Power BI [**link**]: la herramienta de BI (Business Intelligence o inteligencia de negocios) en la nube de Azure, integrada de forma nativa en Fabric, permite crear visualizaciones, dashboards e informes interactivos directamente sobre datos almacenados en Lakehouse o Warehouse, mediante la abstracción de un Semantic Model que define métricas, jerarquías y relaciones, sin necesidad de mover ni duplicar los datos.
- OneLake [**link**]: actúa como repositorio de datos único para todos los servicios de Fabric, asegurando gobierno y seguridad de datos.

2.7. Tecnologías de desarrollo utilizadas

2.7.1. Plataforma Docker

Docker [33] es un proyecto de código abierto que automatiza el despliegue de aplicaciones en contenedores, utilizando características de aislamiento del kernel Linux (cgroups y namespaces) para ejecutar contenedores ligeros y aislados, evitando la sobrecarga de máquinas virtuales.

2.7.2. Plataforma de CI/CD

Durante el proceso de desarrollo del producto se utilizó CI/CD (*continuous integration / continuous delivery*) mediante la integración de las siguientes herramientas:

- Github [62]: servicio de repositorio y control de versiones de código fuente.
- AWS CodePipeline [63]: servicio de compilación, empaquetado y ejecución *builds*.
- AWS Elastic Container Registry [64]: servicio de repositorio y control de versiones de imágenes Docker.

El objetivo de esta configuración de servicios es permitir que por cada cambio en el código fuente versionado en el controlador de versiones Github, se dispare un proceso de compilación y ejecución de tests unitarios notificando en tiempo real si dicho cambio agrega o no una falla al actual estado del desarrollo. En caso de pasar satisfactoriamente la compilación y ejecución de los tests entonces se genera una nueva imagen Docker con la última versión del codigo compilado y se versiona en AWS Elastic Container Registry.

2.7.3. Visual Studio Code

Visual Studio Code [65] es un editor de código fuente desarrollado por Microsoft para Windows, Linux, macOS y Web. Incluye soporte para la depuración, control integrado de Git, resaltado de sintaxis, finalización inteligente de código, fragmentos y refactorización de código.

2.7.4. Sistema operativo Ubuntu

Ubuntu [66] es una distribución Linux basada en Debian GNU/Linux y patrocinado por Canonical, que incluye principalmente software libre y de código abierto. Puede utilizarse en ordenadores y servidores, está orientado al usuario promedio, con un fuerte enfoque en la facilidad de uso y en mejorar la experiencia del usuario.

Capítulo 3

Diseño e implementación

En este capítulo se presentan los detalles técnicos de diseño e implementación de la solución IoT que se tuvieron en cuenta durante el desarrollo del trabajo.

3.1. Arquitectura de software del sistema

El sistema cuenta con una arquitectura multi-cloud en la que se integra como hardware el robot de exploración ambiental [25] desarrollado en el marco de la Carrera de Especialización en Sistemas Embebidos, con un sistema *back-end* desplegado en la nube de Amazon Web Services [31], Smart Contracts desplegados en la red Blockchain Ethereum [49], y un plataforma analítica de datos desplegada en la nube de Azure [67] utilizando la herramienta Microsoft Fabric [68].

En la siguiente figura 3.1 podemos apreciar la arquitectura de la solución.

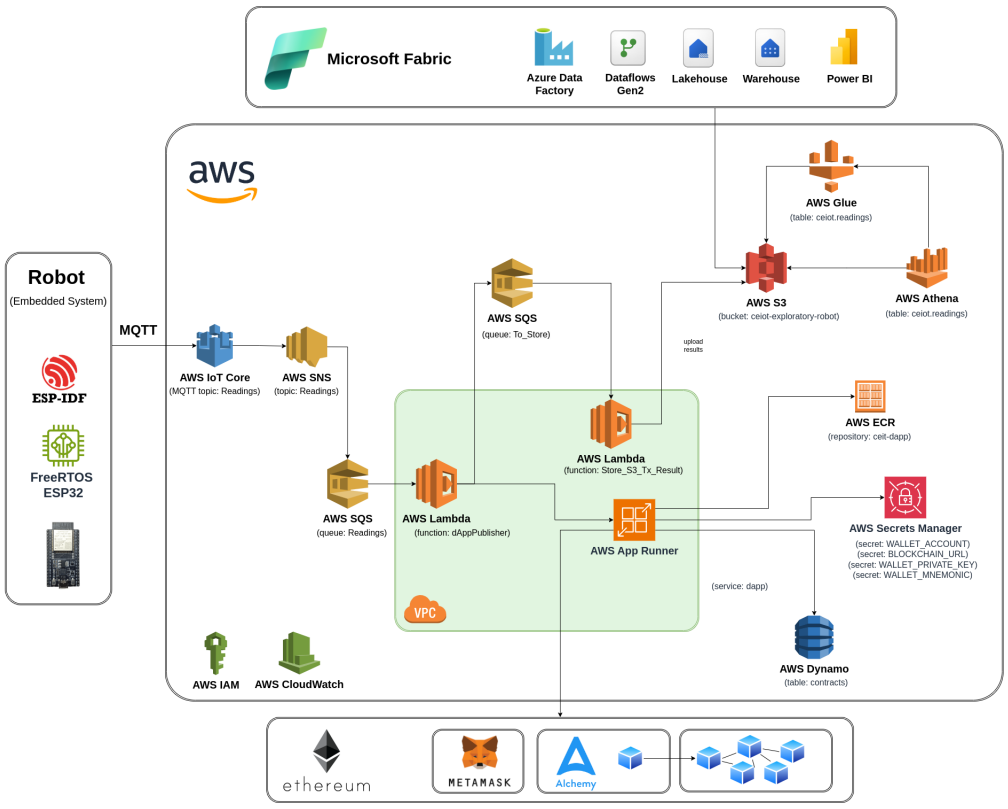

FIGURA 3.1. Arquitectura de la solución.

3.2. Hardware e infraestructura del sistema

<A desarrollar>

3.3. Integración de los módulos y subsistemas

3.3.1. Capa de percepción

El desarrollo e integración de los componentes de esta capa consistió en la adaptación del firmware desplegado en el robot explorador para extender sus funcionalidades y enviar las lecturas de parámetros ambientales al *topic* MQTT. Dentro de las funcionalidades que se le agregaron al robot explorador se encontraron:

- Capturar fecha y hora local del sistema.
- Generación de coordenadas geograficas (con datos *mock*).
- Conexión segura con tópico MQTT y envío de los datos generados.

La configuración de la fecha y hora se realizó por medio del uso del servicio SNTP [69] que permite la sincronización del hardware de una red con la fecha y hora provista por servicios externos en estandar en una zona horaria. Esta configuración se realizó incluyendo el encabezado **esp_sntp.h** en el código del robot. Una vez realizado esto fue posible obtener la fecha y hora local invocando a la función *localtime*.

La generación de las coordenadas geograficas con datos *mock* se realizo mediante la generación de las ecuaciones 3.1 y 3.2 a continuación.

$$MockLat = \left(\frac{rand()}{RAND\_MAX}\right)(LAT\_MAX - LAT\_MIN) + LAT\_MIN \quad (3.1)$$

$$MockLong = \left(\frac{rand()}{RAND\_MAX}\right)(LONG\_MAX - LONG\_MIN) + LONG\_MIN \quad (3.2)$$

Finalmente, los datos capturados fueron enviados en formato JSON al tópico MQTT con la estructura de la tabla:

TABLA 3.1. Tabla de objetos AWS

Nombre del campo	Tipo del campo	Descripción
deviceId	string	Id del dispositivo
type	string	Tipo de lectura
value	string	Valor de la lectura
geoLat	string	Latitud geográfica
geoLong	string	Longitud geográfica
date	string	Fecha
time	string	Hora

A continuación podemos apreciar un valor de ejemplo del objeto JSON enviado por el robot:

```
{
        "deviceId": "12ad-dao23-ux23",
        "type": "Temperature",
        "value": "0.00",
        "geoLat": "-26.056772",
        "geoLong": "-64.014824",
        "date": "2025-04-1",
        "time": "11:23:59"
}
```


3.3.2. Capa de red

El desarrollo de los componentes de esta capa consistió en la publicación de un *topic* MQTT desde el servicio AWS IoT Core y la configuración de la lógica de redirección y almacenamiento de los mensajes recibidos en AWS S3.

Para la conexión segura con el tópico MQTT se configuró el servicio AWS IoT Core, donde se creó una nueva instancia de un dispositivo remoto con el nombre ESP32. Una vez creado este dispositivo se descargaron e instalaron en el código del robot los certificados listados a continuación:

- AmazonRootCA1.pem, renombrado a brokerCA.crt: Es la autoridad certificadora que AWS usa para firmar certificados de sus servidores. El dispositivo lo necesita para verificar la identidad del servidor AWS IoT al conectarse.
- dev-certificate.pem.crt, renombrado a client.crt: Contiene la clave pública correspondiente a la clave privada (.key) y está firmado por AWS (o por una CA en la que AWS confía) para verificar la identidad del dispositivo.
- dev-private.pem.key, renombrado a client.key: Usada por el dispositivo para firmar su identidad durante la conexión TLS. Nunca se comparte ni se sube a AWS. Tu dispositivo la usa para autenticar su certificado (.crt).

Una vez realizada la configuración del servicio AWS IoT core e integrado el robot con el tópico, se probó la recepción de lecturas con el cliente de prueba provisto por AWS suscrito al tópico *readings* como puede apreciarse en la figura 3.2.

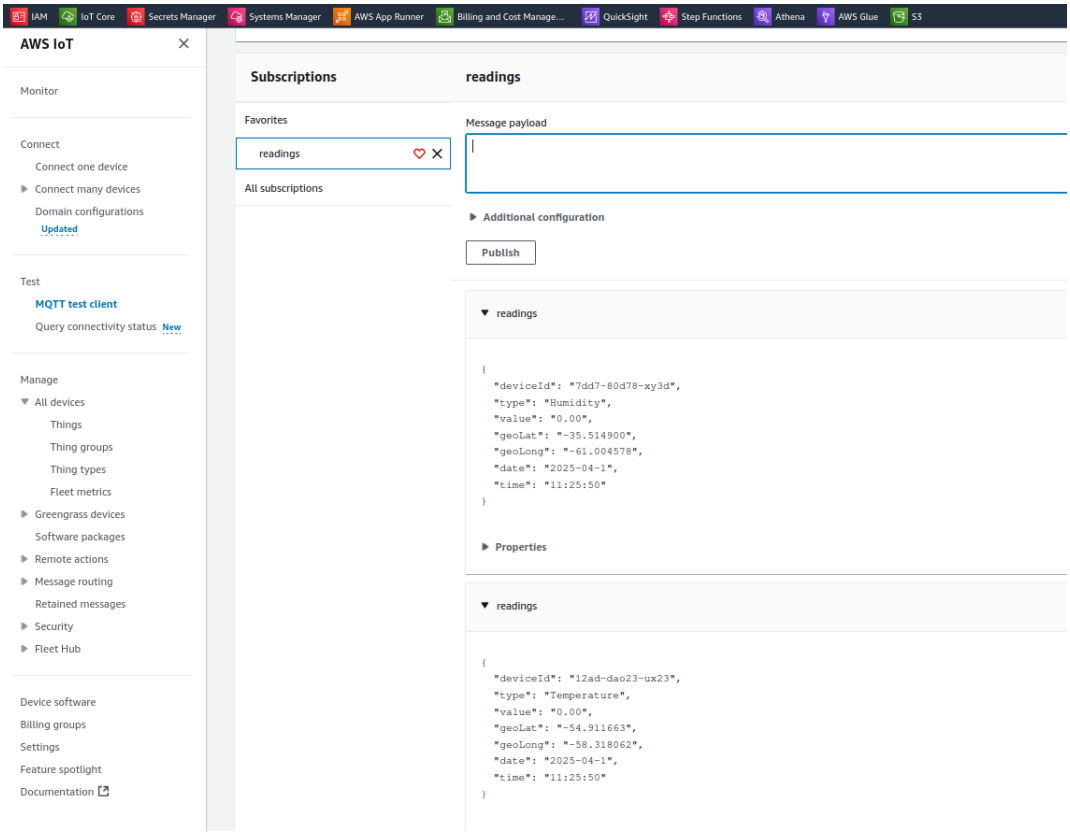

FIGURA 3.2. Prueba de recepción de mensajes MQTT.

Para el almacenamiento en AWS S3 de los mensajes recibidos, se configuro una *routing rule* o regla de redirección en AWS IoT Core, indicando mediante una consulta con sintaxis SQL, que todos los mensajes recibidos en el tópico *readings* deben almacernarce en el bucket S3 ceiot-exploratory-robot. En la figura 3.3 puede apreciarse esta configuración.

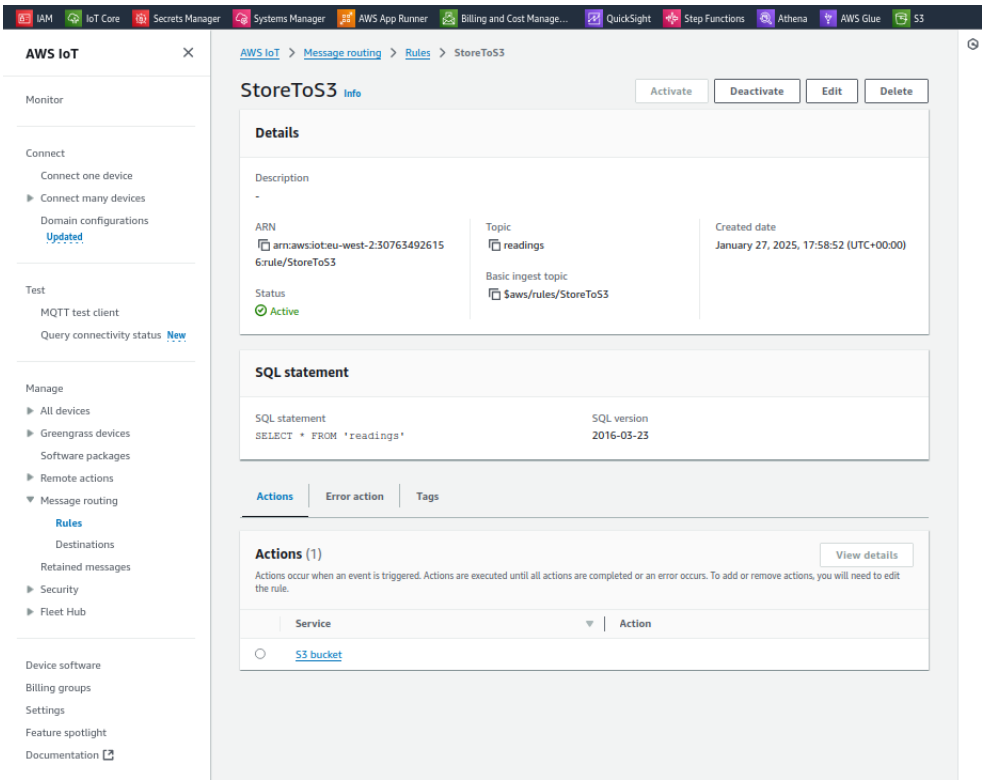

FIGURA 3.3. Configuración de redirección de mensajes MQTT.

Como resultado de la configuración realizada, los mensajes recibidos en MQTT fueron redirigidos y almacenados en AWS S3 como puede apreciarse en la figura 3.4.

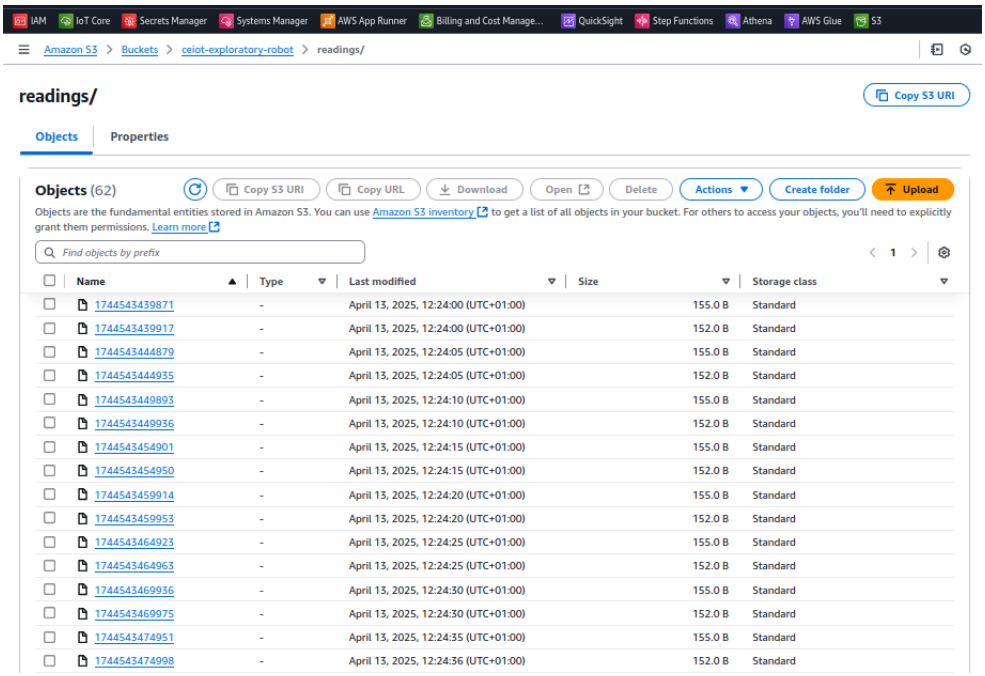

FIGURA 3.4. Almacenamiento de mensajes JSON en AWS S3.

3.3.3. Capas de procesamiento y almacenamiento - cloud

Una vez realizadas las configuraciones de ingesta de datos en *streaming* se realizaron las configuraciones para poder administrarlos y procesarlos. Para ello se crearon una base de datos y una tabla en AWS Glue para representar el esquema de datos almacenados en AWS S3 en formato JSON, como se puede apreciar en la figura 3.5.

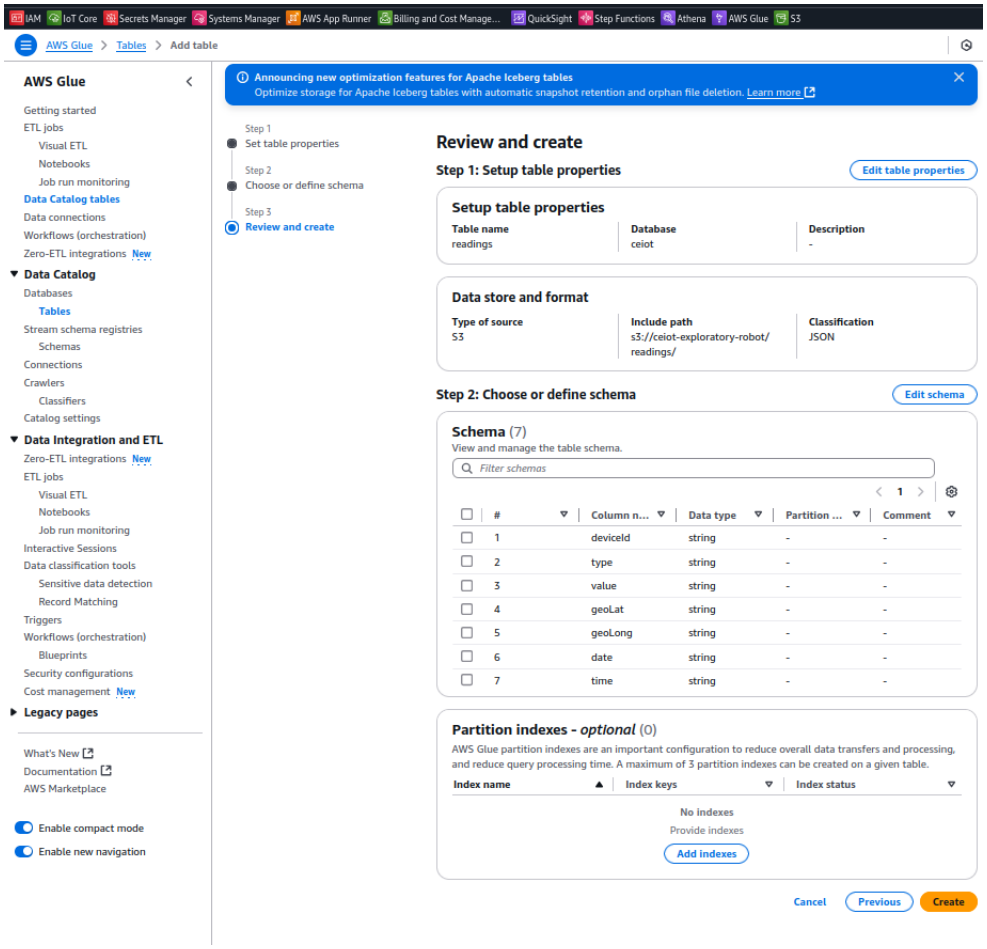

FIGURA 3.5. Creación de base datos, tabla y esquema AWS Glue.

Con el esquema de datos definido en el catálogo de AWS Glue, fue posible realizar consultas SQL sobre los datos almacenados en AWS S3 desde AWS Athena, como se puede apreciar en la figura

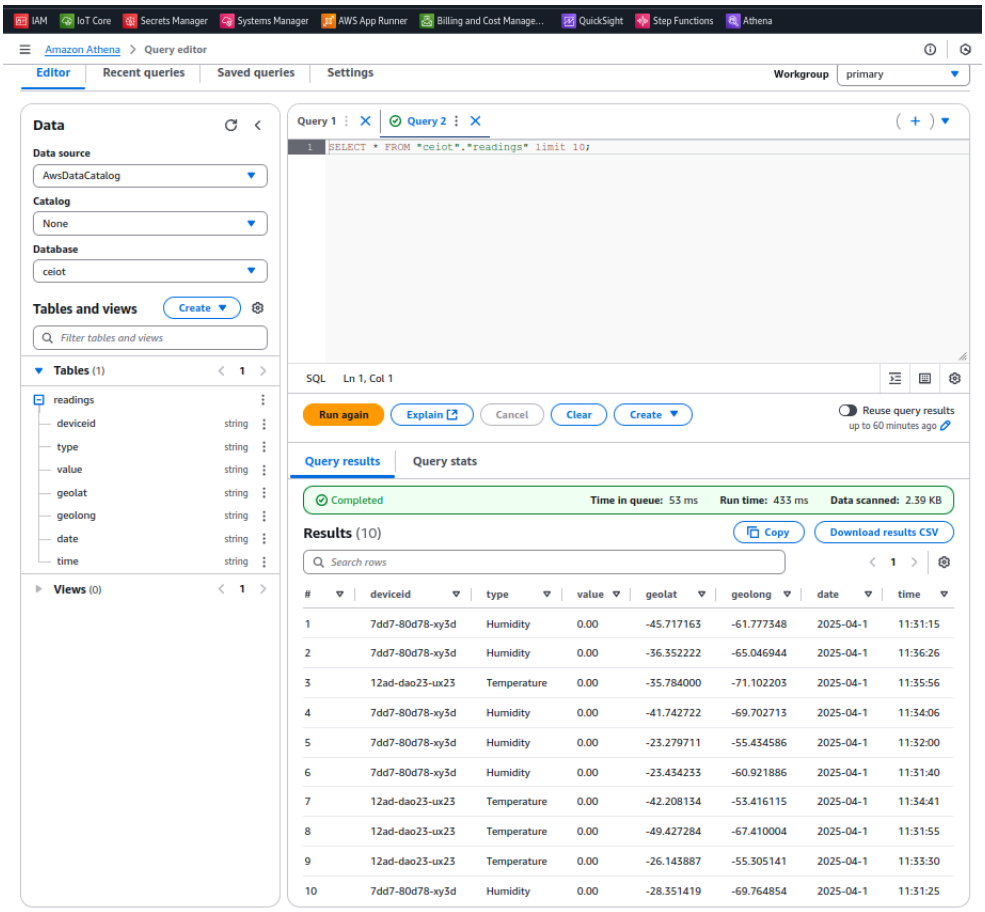

FIGURA 3.6. Consulta de datos SQL desde AWS Athena.

3.3.4. Capas de procesamiento y almacenamiento - blockchain

Uno de los primeros pasos para poder comenzar a desarrollar los componentes blockchain fue la obtención de tokens para poder realizar despliegues y ejecutar la aplicación en las redes de prueba de Ethereum sin utilizar fondos reales. Para Para poder realizar esto, primero fue necesario crear un *wallet* o billetera digital, para lo que se utilizó el servicio Metamask. Luego, para la obtención de créditos se utilizaron los *faucets* de Google [70]. Como se puede apreciar en las figura 3.7 y tras seleccionar la dirección del *wallet* y 3.8, tras realizar las transacciones en el *faucet* se reciben los fondos en la billeta digital.

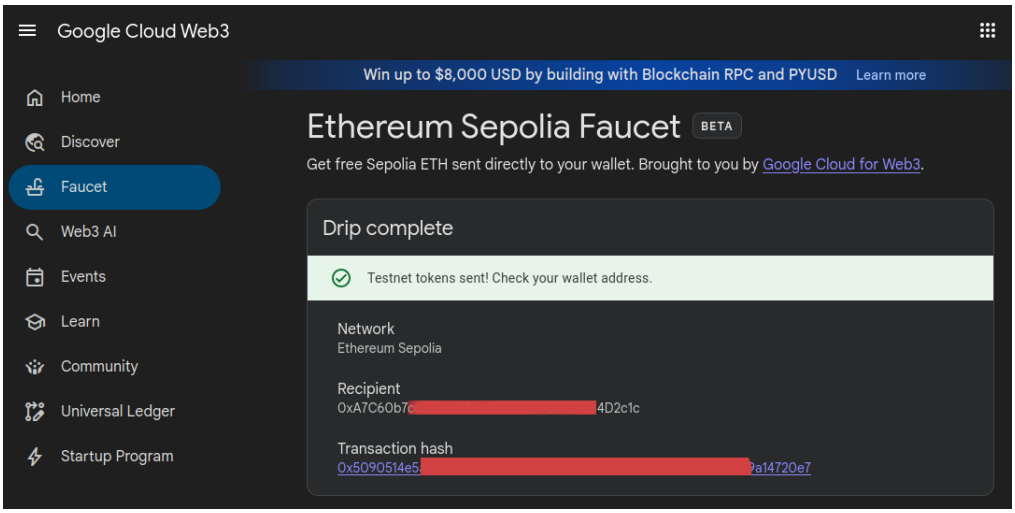

FIGURA 3.7. Obtención de créditos mediante Google Web3.

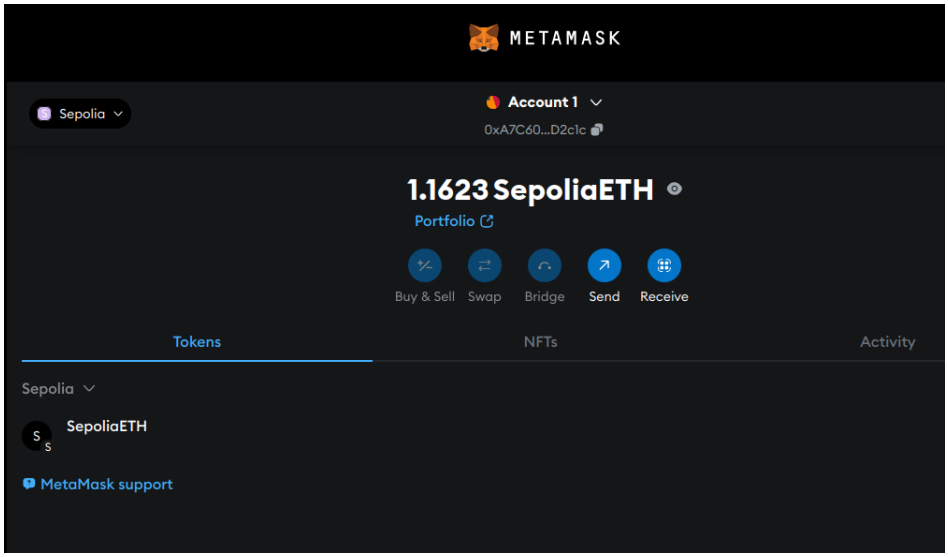

FIGURA 3.8. Saldo en Metamask.

Posteriormente como podemos apreciar en la figura, en el servicio Etherscan las transacciones realizadas para generar fondos en la dirección de la billetera quedan publicadas en la red.

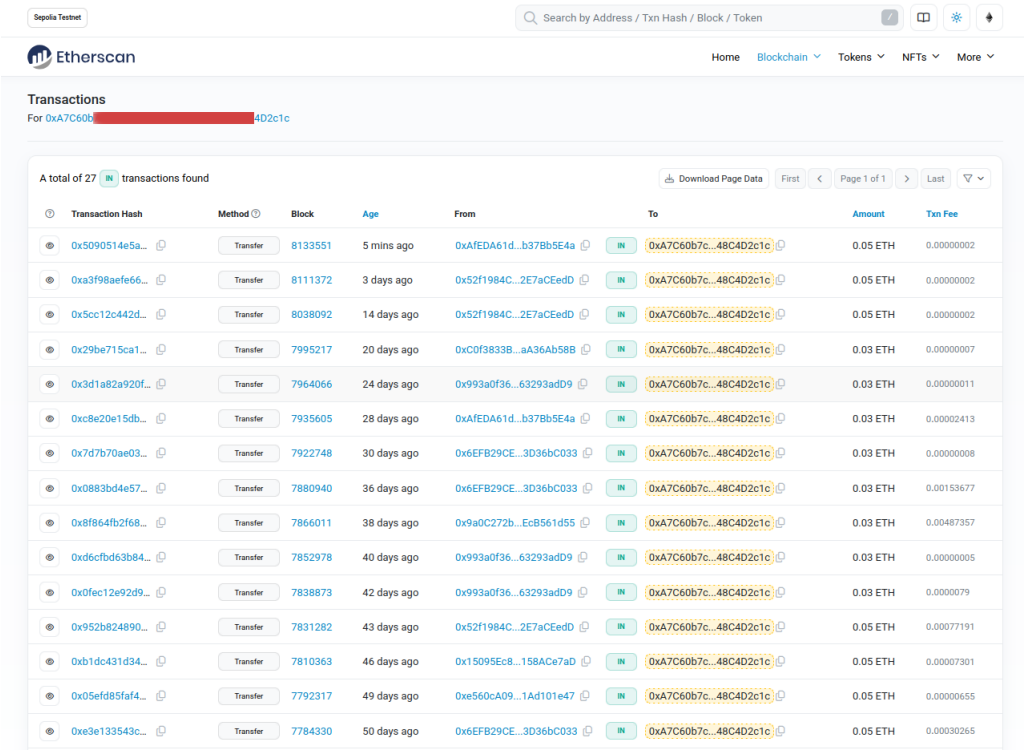

FIGURA 3.9. Transacciones de generación de fondos de prueba.

Una vez obtenidos los fondos de prueba en la billetera se procedió con el desarrollo de los componentes blockchain. Para el desarrollo de los *smart contracts* se utilizó Solidity como lenguaje de programanción y Truffle como herramienta de gestión de configuración, compilación, empaquetado y despliegue. Truffle utiliza una configuración basada en archivos Javascript para la descripción de las tareas, y para realizar el despliegue a diferentes redes, como por ejemplo de forma local a Ganache o de forma remota a redes como Sepolia, Holesky y Mainnet. En la los archivos de configuración de Truffle se agregaron entradas para poder desplegar a Ganache y a Sepolia como se puede apreciar en la figura 3.10.

```
networks: {

  development: {
    host: "ganache", // Dirección de Ganache en el host local
    port: 7545,        // Puerto donde Ganache está corriendo
    network_id: "*",   // Conecta a cualquier red
  },


  sepolia: {
    provider: () => new HDWalletProvider(
      MNEMONIC, // Frase secreta de MetaMask
      `${BLOCKCHAIN_URL}` // URL RPC de la Red a la cual se despliega
    ),
    gas: 3000000, // Límite de gas
    gasPrice: 5000000000, // 5 Gwei
    network_id: 11155111, // ID de la red Sepolia
    skipDryRun: true, // Evita correr una simulación en dry-run antes del despliegue

    // confirmations: 2, // Nº de confirmaciones antes de considerar la transacción válida
    // timeoutBlocks: 200, // Nº de bloques de espera antes de fallar

    pollingInterval: 1800000,
    disableConfirmationListener: true,
  },
```

FIGURA 3.10. Configuración de redes de despliegue en Truffle.

Como se puede apreciar en la figura 3.11, durante el proceso de despliegue se presenta cierta información por pantalla:

- La red a la cual se esta realizando el despliegue.
- Los *smart contracts* incluidos en el despliegue y la dirección que toman una vez desplegados.
- El *hash* o identificador de la transacción.
- El número de bloque en el que se encuentra la transacción de despliegue.
- La dirección de la billetera digital y el balance disponible previo a la transacción.
- La cantidad de gas utilizado en el despliegue, el costo unitario y el costo total de la transacción.

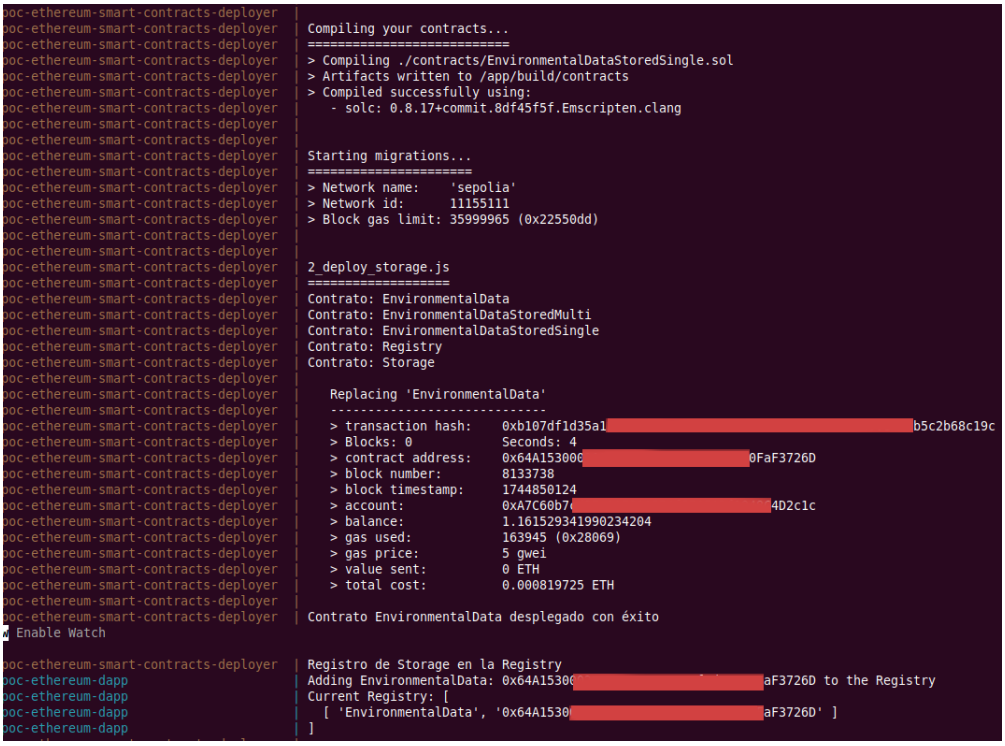

FIGURA 3.11. Salida por pantalla durante el proceso de despliegue de los componentes blockchain.

Luego de haber realizado el despliegue de los *smart contracts* a la red Sepolia, se puede evidenciar desde Etherscan las transacciones reportadas en la salida por pantalla (figura 3.12) y comparar los valores. Como se puede apreciar en la figura 3.13, al ingresar a los detalles de la transacción podemos obtener mas información del contrato.

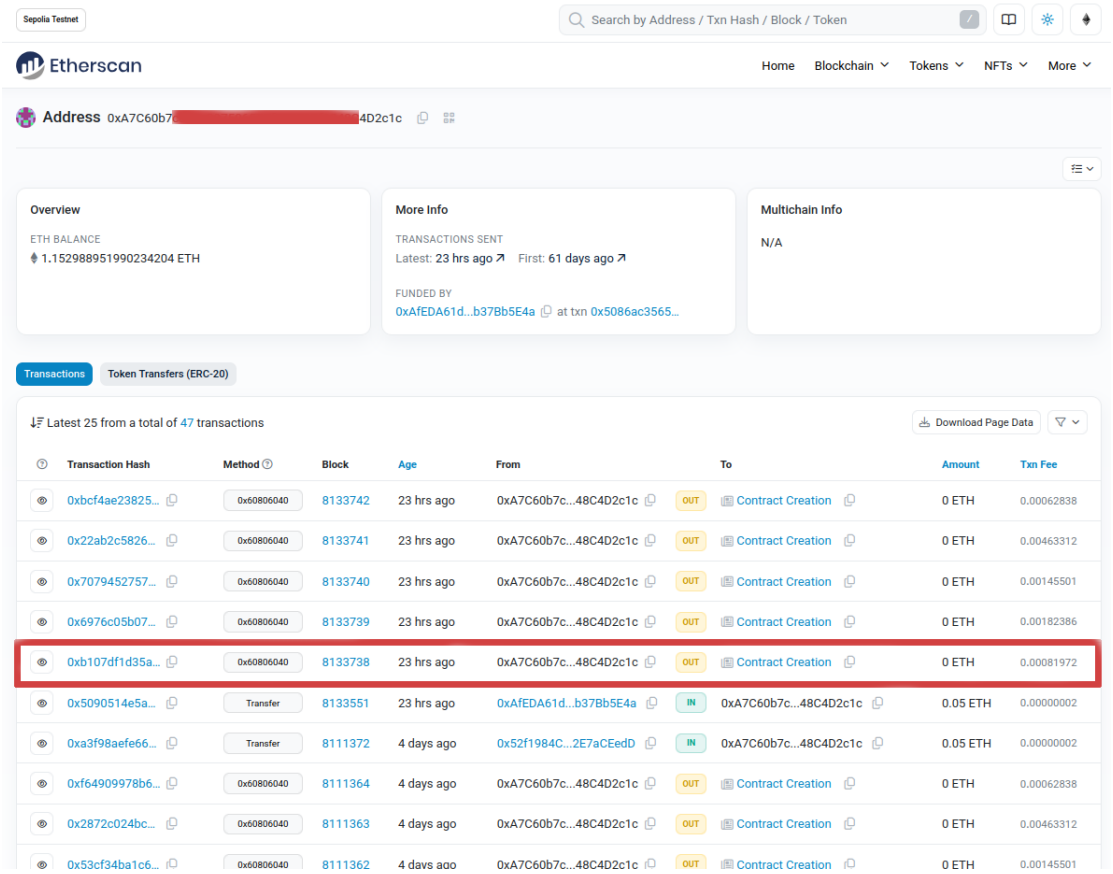

FIGURA 3.12. Transacciones de despliegue en Sepolia.

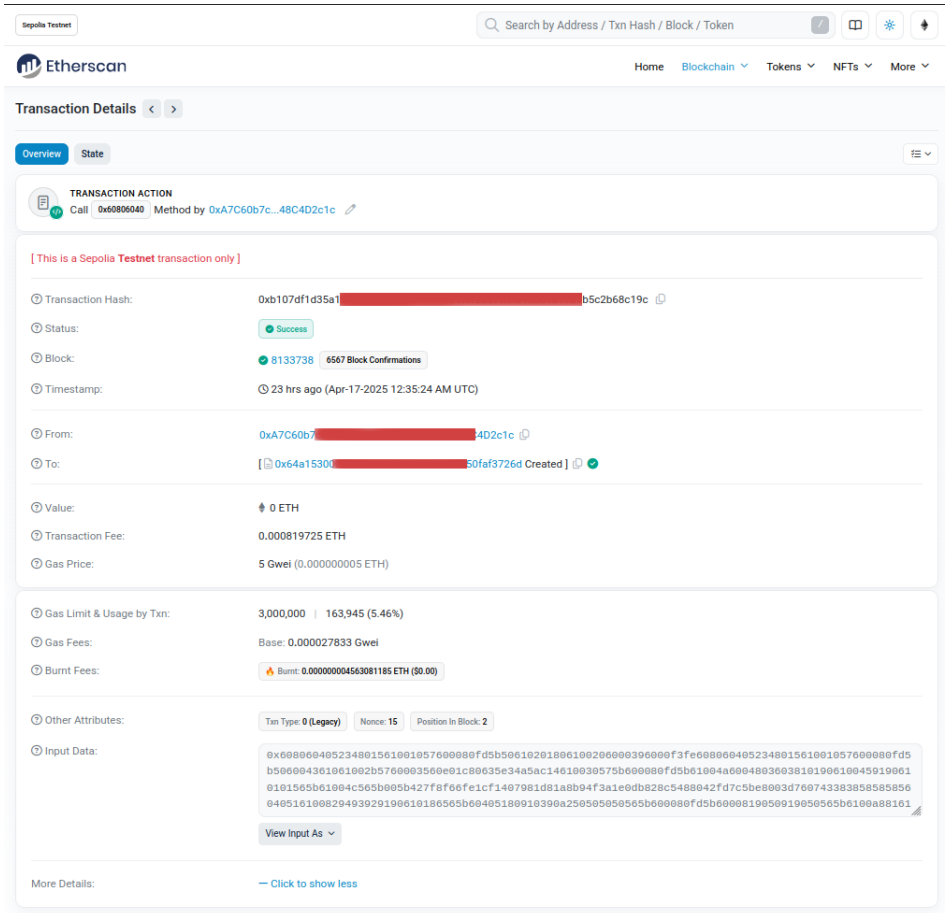

FIGURA 3.13. Detalles del contrato desplegado en Sepolia.

Desde el punto de vista del *backend* blockchain, se desarrolló la dApp utilizando Node.js y la biblioteca Javascript Web3.js para la comunicación con los *smart contracts*. Para la dApp, se desarrollaron varios *endpoints* y se expusieron mediante el estandar Open API [47] utilizando la biblioteca Swagger, que como se puede apreciar en la figura 3.14, genera una interfaz gráfica documenta y brinda un cliente de prueba para poder invocar los *endpoints* expuestos.

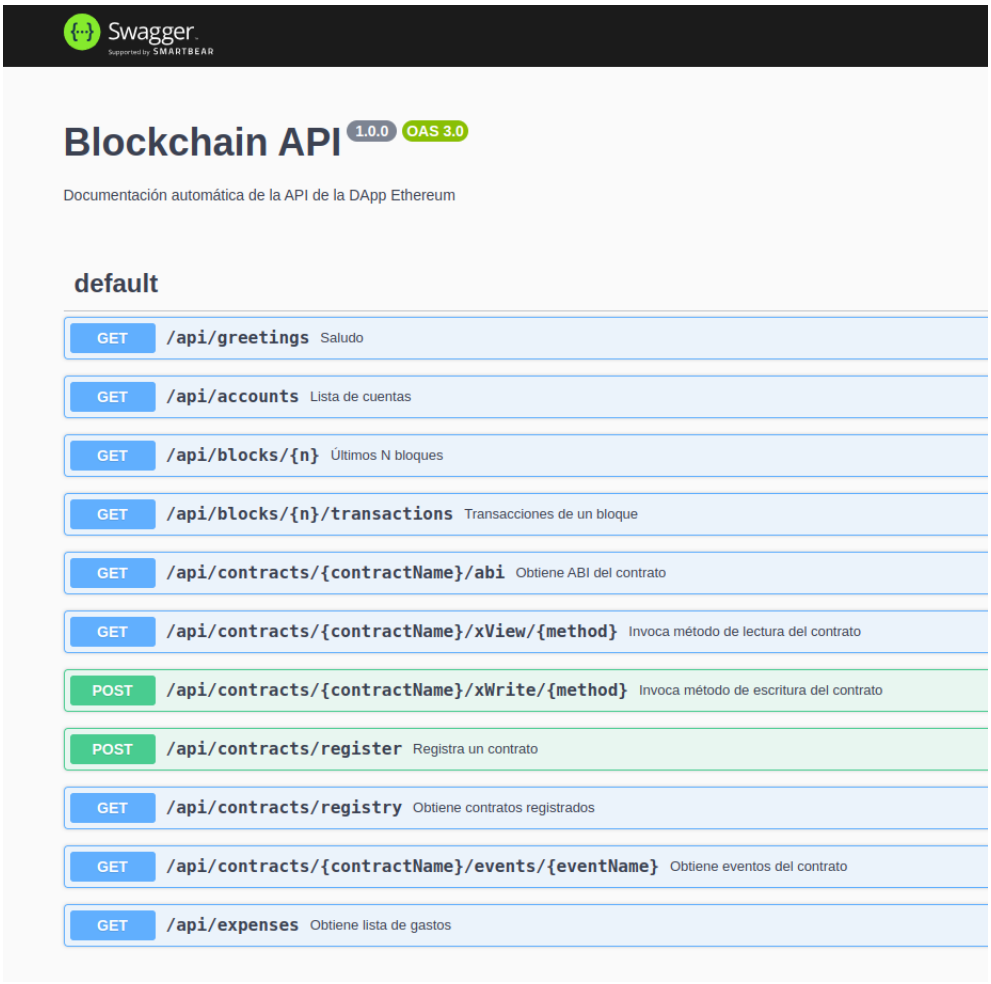

FIGURA 3.14. Endpoints expuestos por la dApp.

Para que la dApp pueda interactuar con la red debe poder conectarse a un endpoint RPC publicado desde cualquier nodo de la misma. Por este motivo se utilizó el servicio Alchemy que brinda publica endpoints para distintas redes, incluyendo Ethereum. Primero se creó un perfil en este servicio de manera gratuita y se creó un nuevo proyecto con el nombre iot_robots_storage. Una vez hecho esto se pudo acceder a los diferentes endpoints disponibles de Ethereum para las tres redes principales (Mainnet, Holesky y Sepolia) como se puede apreciar en la figura 3.15. Posteriormente, se configuró en la inicialización de la biblioteca Web3.js en la dApp, el endpoint provisto por Alchemy y el app_key del proyecto iot_robots_storage.

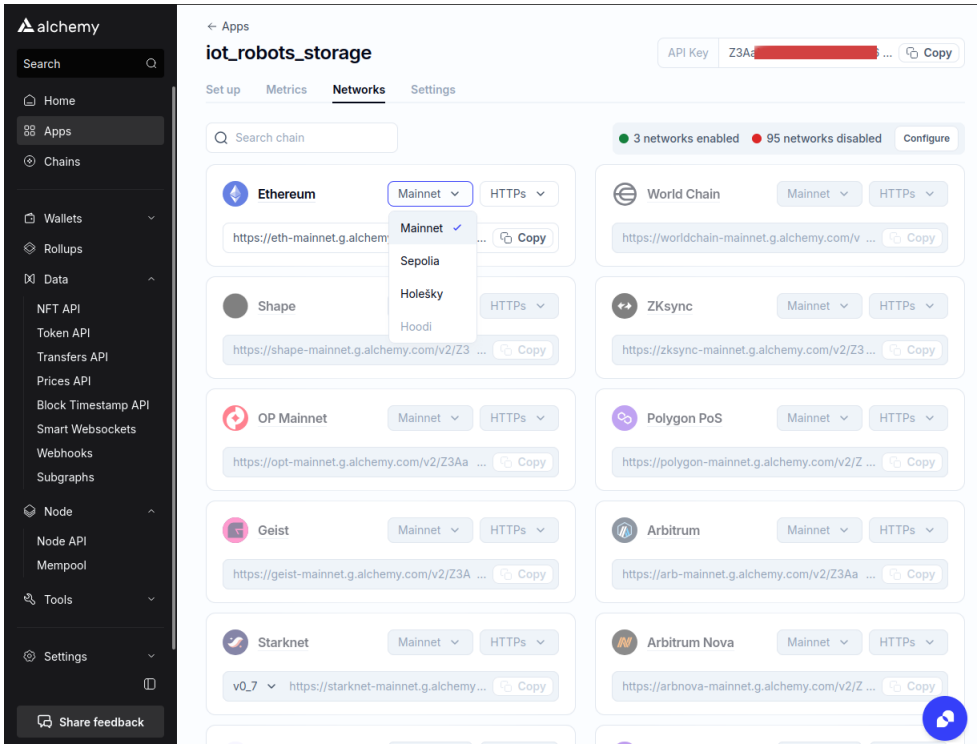

FIGURA 3.15. Frontend the Alchemy.

Como se mencionó en el capítulo 1, la red blockchain cumple la funcion de almacenamiento transparente e inmutable en este caso de uso. Para su integración con el flujo de eventos provenientes del tópico MQTT, a fin de poder almacenar las lecturas en la red, se invocó...

3.3.5. Capas de procesamiento y almacenamiento - *pipeline* analítico

Para

3.4. Plataforma de desarrollo y despliegue

3.5. Tabla de todos los objetos AWS creados

TABLA 3.2. Tabla de objetos AWS

Servicio	Propósito	Nombre de objeto
AWS IoT Core	*Thing*	ESP32
AWS IoT Core	*MQTT topic*	readings
AWS IoT Core	*Routing Rule*	StoreToS3
AWS S3	Bucket	ceiot-exploratory-robot
AWS Glue	*Base de datos*	ceit
AWS Glue	Tabla	*Readings*

Capítulo 4

Ensayos y resultados

Todos los capítulos deben comenzar con un breve párrafo introductorio que indique cuál es el contenido que se encontrará al leerlo. La redacción sobre el contenido de la memoria debe hacerse en presente y todo lo referido al proyecto en pasado, siempre de modo impersonal.

4.1. Pruebas funcionales del hardware

La idea de esta sección es explicar cómo se hicieron los ensayos, qué resultados se obtuvieron y analizarlos.

Capítulo 5

Conclusiones

Todos los capítulos deben comenzar con un breve párrafo introductorio que indique cuál es el contenido que se encontrará al leerlo. La redacción sobre el contenido de la memoria debe hacerse en presente y todo lo referido al proyecto en pasado, siempre de modo impersonal.

5.1. Conclusiones generales

La idea de esta sección es resaltar cuáles son los principales aportes del trabajo realizado y cómo se podría continuar. Debe ser especialmente breve y concisa. Es buena idea usar un listado para enumerar los logros obtenidos.

En esta sección no se deben incluir ni tablas ni gráficos.

Algunas preguntas que pueden servir para completar este capítulo:

- ¿Cuál es el grado de cumplimiento de los requerimientos?
- ¿Cuán fielmente se puedo seguir la planificación original (cronograma incluido)?
- ¿Se manifestó algunos de los riesgos identificados en la planificación? ¿Fue efectivo el plan de mitigación? ¿Se debió aplicar alguna otra acción no contemplada previamente?
- Si se debieron hacer modificaciones a lo planificado ¿Cuáles fueron las causas y los efectos?
- ¿Qué técnicas resultaron útiles para el desarrollo del proyecto y cuáles no tanto?

5.2. Próximos pasos

Acá se indica cómo se podría continuar el trabajo más adelante.

Bibliografía

[1] Wikipedia. *Wi-Fi*. URL: https://es.wikipedia.org/wiki/Wifi.

[2] Wikipedia. *5G*. URL: https://es.wikipedia.org/wiki/5G.

[3] Wikipedia. *LoraWAN*. URL: https://es.wikipedia.org/wiki/LoRaWAN.

[4] Wikipedia. *Bluetooth*. URL: https://en.wikipedia.org/wiki/Bluetooth.

[5] Wikipedia. *Zigbee*. URL: https://en.wikipedia.org/wiki/Zigbee.

[6] Wikipedia. *Narrowband IoT*. URL: https://en.wikipedia.org/wiki/Narrowband_IoT.

[7] Amazon Web Service. *AWS Athena*. URL: https://aws.amazon.com/iot-core/.

[8] Azure. *Azure IoT Hub*. URL: https://azure.microsoft.com/en-gb/products/iot-hub.

[9] As.com | Marc Fontrodona. *Una estación de esquí china despliega una patrulla de perros robot*. URL: https://as.com/deportes_accion/snow/una-estacion-de-esqui-china-despliega-una-patrulla-de-perros-robot-n.

[10] Cadenaser.com | Radio Bilbao. *Bilbao inspecciona sus redes de saneamiento con drones y robots para mejorar la eficiencia y la seguridad*. URL: https://cadenaser.com/euskadi/2024/12/18/bilbao-inspecciona-sus-redes-de-saneamiento-con-drones-y-robots-para-mejorar-la-eficiencia-y-la-seguridad-radio-bilbao/.

[11] Los40.com | Dani Cabezas. *Así se gestiona un "tecnobosque"*. URL: https://los40.com/2024/12/10/asi-se-gestiona-un-tecnobosque/.

[12] Boston Dynamics. *Spot*. URL: https://www.bostondynamics.com/products/spot.

[13] Waygate Technologies. *BIKE - An advanced crawler robot for remote visual inspection*. URL: https://www.bakerhughes.com/waygate-technologies/robotic-inspection/bike.

[14] Latam Mining. *Robots y minería: Gobierno argentino quiere implementarlos*. URL: https://www.latam-mining.com/robots-y-mineria-gobierno-argentino-quiere-implementarlos/.

[15] Diario de Cuyo. *Gobierno pone la mira en el desarrollo de robots para la actividad minera*. URL: https://www.diariodecuyo.com.ar/politica/Gobierno-pone-la-mira-en-el-desarrollo-de-robots-para-la-actividad-minera-20200202-0052.html.

[16] Universidad Nacional de San Juan. *Robots en la minería*. URL: http://www.unsj.edu.ar/home/noticias_detalles/4810/1.

[17] Ing. Nelson Dario García Hurtado e Ing. Melvin Andrés González Pino. *Robot de exploración terrestre Geobot*. URL: https://www.unipamplona.edu.co/unipamplona/portalIG/home_40/recursos/01_general/revista_1/09102011/v01_09.pdf.

[18] Ing. Hernán L. Helguero Velásquez1 e Ing. Rubén Medinaceli Tórrez. *Robot Minero: Sistema Detector de Gases utilizando Sensores en Tiempo Real MIN – SIS 1.0 SDG-STR*. URL: http://www.scielo.org.bo/scielo.php?script=sci_arttext&pid=S2519-53522020000100003.

[19] Wikipedia. *Blockchain*. URL: https://en.wikipedia.org/wiki/Blockchain.
[20] Wikipedia. *Smart Contracts*. URL: https://en.wikipedia.org/wiki/Smart_contract.
[21] Wikipedia. *DApp - Decentralized Application*. URL: https://en.wikipedia.org/wiki/Decentralized_application.
[22] Walmart | Global Tech. *Blockchain in the food supply chain - What does the future look like?* URL: https://tech.walmart.com/content/walmart-global-tech/en_us/blog/post/blockchain-in-the-food-supply-chain.html?utm_source=chatgpt.com.
[23] ledgerinsights | Nicky Morris. *ScanTrust's anti-counterfeit solution isn't just about blockchain*. URL: https://www.ledgerinsights.com/scantrust-anti-counterfeit-blockchain/.
[24] Paula Eiroa Interempresas | Julio Lema. *Mejorar la eficiencia energética con IoT y blockchain*. URL: https://www.interempresas.net/Energia/Articulos/446423-Mejorar-la-eficiencia-energetica-con-IoT-y-blockhain.html.
[25] Esp. Ing. Gonzalo Carreño. *LSE-FIUBA - Trabajo Final CESE- Robot de exploración ambiental*. URL: https://lse-posgrados-files.fi.uba.ar/tesis/LSE-FIUBA-Trabajo-Final-CESE-Gonzalo-Carreno-2024.pdf.
[26] Amazon Web Services. *¿Qué es una nube pública?* URL: https://aws.amazon.com/es/what-is/public-cloud/.
[27] OASIS. *MQTT Protocol Specification*. URL: https://mqtt.org/mqtt-specification/.
[28] Espressif. *ESP-IDF Programming Guide | Get Started*. URL: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/get-started/.
[29] Espressif. *ESP32*. URL: https://www.espressif.com/en/products/socs/esp32.
[30] FreeRTOS. *FreeRTOS | Real-time operating system for microcontrollers and small microprocessors*. URL: https://www.freertos.org/.
[31] Amazon Web Service. *Amazon Web Service*. URL: https://aws.amazon.com/.
[32] Amazon Web Service. *AWS Athena*. URL: https://aws.amazon.com/apprunner/.
[33] Docker. *Docker*. URL: https://docker.com/.
[34] Amazon Web Service. *AWS Glue*. URL: https://aws.amazon.com/glue/.
[35] Amazon Web Service. *AWS Athena*. URL: https://aws.amazon.com/s3/.
[36] Amazon Web Service. *AWS Athena*. URL: https://aws.amazon.com/athena/.
[37] Amazon Web Service. *AWS SNS*. URL: https://aws.amazon.com/sns/.
[38] Amazon Web Service. *AWS SQS*. URL: https://aws.amazon.com/sqs/.
[39] Amazon Web Service. *AWS Secrets Manager*. URL: https://aws.amazon.com/secrets-manager/.
[40] Amazon Web Service. *AWS Lambda*. URL: https://aws.amazon.com/lambda/.
[41] Amazon Web Service. *AWS Dynamo*. URL: https://aws.amazon.com/dynamodb/.
[42] Amazon Web Service. *AWS IAM*. URL: https://aws.amazon.com/iam/.
[43] Wikipedia. *ABAC*. URL: https://en.wikipedia.org/wiki/Attribute-based_access_control.
[44] Wikipedia. *MFA*. URL: https://en.wikipedia.org/wiki/Multi-factor_authentication.
[45] Amazon Web Service. *AWS CloudWatch*. URL: https://aws.amazon.com/cloudwatch/.

[46] Node.js. *Node.js*. URL: https://nodejs.org/en.
[47] swagger.io. *Open API Specification*. URL: https://swagger.io/specification/.
[48] soliditylang.org. *Solidity programming language*. URL: https://soliditylang.org/.
[49] Ethereum.org. *Welcome to Ethereum*. URL: https://ethereum.org/.
[50] Ethereum.org. *Ethereum Virtual Machine (EVM)*. URL: https://ethereum.org/en/developers/docs/evm/.
[51] Truffle Suite. *Truffle | What is Truffle?* URL: https://etherscan.io/.
[52] Truffle Suite. *Truffle | What is Truffle?* URL: https://sepolia.etherscan.io/.
[53] Truffle Suite. *Truffle | What is Truffle?* URL: https://holesky.etherscan.io/.
[54] Soliditylang.org. *Contract ABI Specification*. URL: https://docs.soliditylang.org/en/latest/abi-spec.html.
[55] Web3.js. *Web3.js - Ethereum JavaScript API*. URL: https://web3js.readthedocs.io/en/v1.10.0/.
[56] Wikipedia. *Proof of Stake*. URL: https://en.wikipedia.org/wiki/Proof_of_stake.
[57] Truffle Suite. *Ganache | One click blockchain*. URL: https://archive.trufflesuite.com/ganache/.
[58] Truffle Suite. *Truffle | What is Truffle?* URL: https://archive.trufflesuite.com/docs/truffle/.
[59] Alchemy. *Alchemy | The most reliable way to build web3 apps*. URL: https://www.alchemy.com/.
[60] Etherscan. *Etherscan*. URL: https://etherscan.io/.
[61] Metamask. *Metamask | Your home in Web3*. URL: https://metamask.io/.
[62] Github. *Github*. URL: https://github.com/.
[63] Amazon Web Services. *AWS CodePipeline*. URL: https://aws.amazon.com/codepipeline/.
[64] Amazon Web Services. *AWS Elastic Container Registry*. URL: https://aws.amazon.com/ecr/.
[65] Visualstudio. *Visualstudio Code*. URL: https://code.visualstudio.com/.
[66] Ubuntu. *Ubuntu*. URL: https://ubuntu.com/.
[67] Microsoft. *Microsoft Azure*. URL: https://azure.microsoft.com/.
[68] Microsoft. *Microsoft Fabric | Fundamentals*. URL: https://learn.microsoft.com/en-us/fabric/.
[69] IBM Documentation. *Simple Network Time Protocol*. URL: https://www.ibm.com/docs/en/i/7.4.0?topic=services-simple-network-time-protocol.
[70] Google. *Google Cloud Web3*. URL: https://cloud.google.com/application/web3/faucet.