

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 (pertencia)
Jurado 2 (pertencia)
Jurado 3 (pertencia)

Ciudad de [lugar], [mes] de [año]

Resumen

El presente trabajo describe la implementación de un proyecto personal en el que se desarrolla una solución de Internet de las Cosas para un caso de uso de robot de exploración ambiental capaz de garantizar la inmutabilidad y transparencia de los datos. Para su implementación se utilizaron conceptos y herramientas tales como el desarrollo de sistemas embebidos, mensajería asíncrona, almacenamiento y procesamiento distribuido en la nube, entre otros.

Índice general

Resumen	I
1. Introducción general	1
1.1. Estado del arte	1
1.2. Motivación del trabajo	3
1.3. Alcance y objetivos	3
1.4. Requerimientos del producto	4
2. Introducción específica	7
2.1. Tecnologías de hardware y firmware utilizadas	7
2.1.1. Robot de exploración ambiental Espressif ESP32	7
2.1.2. Marco de trabajo ESP-IDF	7
2.1.3. Sistema operativo FreeRTOS	7
2.1.4. Testing unitario	8
2.2. Tecnologías backend utilizadas	8
2.2.1. Amazon Web Services	8
2.2.2. AWS App Runner	8
2.2.3. AWS Glue	8
2.2.4. AWS S3	9
2.2.5. AWS Athena	9
2.2.6. MQTT	9
2.2.7. Node.js	9
2.3. Tecnologías Blockchain utilizadas	9
2.3.1. Red Ethereum	9
2.3.2. Solidity	11
2.3.3. Biblioteca Web3.js	11
2.3.4. Ganache	11
2.3.5. Truffle	11
2.3.6. Alchemy	12
2.3.7. Etherscan	12
2.3.8. Metamask	12
2.4. Tecnologías de desarrollo utilizadas	12
2.4.1. Plataforma Docker	12
2.4.2. Plataforma de CI/CD	13
2.4.3. Visual Studio Code	13
2.4.4. Sistema operativo Ubuntu	13
3. Diseño e implementación	15
3.1. Análisis del software	15
4. Ensayos y resultados	17
4.1. Pruebas funcionales del hardware	17

5. Conclusiones	19
5.1. Conclusiones generales	19
5.2. Próximos pasos	19
Bibliografía	21

Índice de figuras

Índice de tablas

Capítulo 1

Introducción general

Esta sección 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

Las soluciones IoT (*Internet of Things* o Internet de las Cosas) se basan en la conexión de dispositivos físicos con sistemas software para recopilar, transmitir y analizar datos en *streaming* y de forma *batch*, mejorando 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 por red (Wi-Fi [1], 5G [5g], LoRaWAN [2]) para transmitirlos, una plataforma *back-end* formada por sistemas distribuidos en la nube para el almacenamiento, procesamiento y análisis de datos, y una interfaz de usuario para la visualización de resultados, pudiendo, en ocasiones, contar con sistemas de publicación y distribución de eventos en *streaming*.

Existen casos de uso de IoT en los cuales, debido a las condiciones del entorno y por motivos de capacidad, practicidad o seguridad para un humano, se utilizan robots exploradores como dispositivos físicos para la recopilación de datos. 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 varios trabajos, proyectos, e implementaciones comerciales de soluciones IoT utilizando robots para mejorar la seguridad, la eficiencia y la toma de decisiones basada en datos, como por ejemplo:

- El despliegue en Lotus Mountain [3] en Jilin, China para la seguridad en estaciones de esquí, implementado por perros robóticos equipados con sensores y tecnología de imágenes 3D que patrullan las pistas identificando peligros como desprendimientos y bloqueos en las rutas, mejorando así la seguridad de los esquiadores.

- El implementado por el Ayuntamiento de Bilbao [4] 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 [5] 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 [6], desarrollado por Boston Dynamics, un robot explorador cuadrupedo de propósito general capaz de explorar, almacenar y enviar información en tiempo real.
- BIKE [7], 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 [8], [9], 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 [10].
- El robot de exploración terrestre denominado Geobot [11] 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 [12] 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.

En situaciones en las que es necesario explorar y monitorear un área ambientalmente sensible, como una reserva natural o un sitio afectado por un desastre ecológico y el objetivo es recopilar datos críticos, tales como niveles de contaminación, temperatura, humedad, y calidad del aire, es de gran importancia almacenar estas mediciones de una manera en la que se pueda asegurar la integridad y transparencia de los datos, como por ejemplo, en una cadena de bloques (blockchain).

Una arquitectura blockchain [13] se basa en el agrupamiento de transacciones que luego de ser procesadas, son almacenadas en bloques encadenados de forma 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*

[14]. 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* en un caso de uso interactivo desde afuera de la red, se realiza a través de otro componente conocido como dApps (*decentralized applications*) [15] que haciendo uso de ciertas bibliotecas invocan a estos componentes para almacenar y obtener datos en y desde el *distributed ledger*.

La utilización de blockchain en arquitecturas IoT tiene varias ventajas, entre las que se pueden mencionar las vinculadas a las capacidades de descentralización, cifrado, seguridad y consenso que ayudan a aumentar la trazabilidad y transparencia de las operaciones, y la automatización de operaciones disparados por eventos utilizando Smart Contracts. En la industria se pueden encontrar

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 de Internet de las Cosas.

Se pretendió implementar una arquitectura robusta y flexible que pueda ser extrapolada a casos de uso en la industria en los que sea necesaria la integración de un sistema embebido con una red blockchain para el almacenamiento transparente e inmutable de datos sensibles.

Por otra parte, se buscó desarrollar un producto cuya implementación pueda contribuir a aumentar el conocimiento público y el estado del arte de proyectos de código abierto para soluciones IoT integradas a blockchain en Argentina.

1.3. Alcance y objetivos

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

- La publicación del endpoint MQTT [16] 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 *back-end* vía MQTT.
- La arquitectura e implementación de los sistemas *back-end* y el modelo de datos necesario para el almacenamiento de las mediciones enviadas por el robot.
- La arquitectura, implementación y despliegue de la dApp y *smart contracts* 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 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 plataforma 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 plataforma debe poder procesar las mediciones de parámetros ambientales enviadas por el robot para generar métricas de valor para el usuario de negocio.
- f) La plataforma debe brindar dos *front-end* con interfaz web:
 - 1) El *front-end* para el usuario de negocio.
 - 2) El *front-end* para el usuario administrador.
- g) El *front-end* 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 *front-end* 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 plataforma debe contar con al menos un *back-end* de procesamiento y acceso a datos operacionales para la lógica de negocio.

- b) La plataforma debe contar con al menos un *back-end* de acceso, procesamiento, almacenamiento de datos analíticos para la generación de métricas.
 - c) El envío de los valores ambientales censados al *back-end* 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. Requerimiento de testing
- a) Se debe incluir tests de unitarios de componentes.
 - b) Se debe 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 agregado 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

Esta sección presenta una breve introducción técnica a las herramientas hardware y software utilizadas en el trabajo.

2.1. Tecnologías de hardware y firmware utilizadas

2.1.1. Robot de exploración ambiental Espressif ESP32

ESP32 [17] 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 cores 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 varios periféricos. En la siguiente imagen (??) se puede apreciar la placa ESP32-WROOM-32D [ESP32_wroom_32d_datasheet] utilizada para el desarrollo del presente trabajo.

2.1.2. Marco de trabajo ESP-IDF

Espressif Systems proporciona recursos básicos de hardware y software para ayudar a los desarrolladores de aplicaciones a realizar sus ideas utilizando el hardware de la serie ESP32. El framework de software de Espressif está destinado al desarrollo de aplicaciones de IoT (Internet de las cosas) con Wi-Fi, Bluetooth, administración de energía y varias otras características del sistema. Sus componentes son:

1. Toolchain, utilizado para compilar el código para ESP32.
2. Build tools, que provee utilidades como CMake [18] y Ninja [19] para construir la aplicación completa para ESP32.
3. ESP-IDF [20], que brinda la API de desarrollo para ESP32 y scripts para ejecutar Toolchain.

Además de las herramientas mencionadas se utilizó el conjunto de bibliotecas y drivers provistos por el proyecto ESP-IDF-Lib [21] basados en el framework ESP-IDF.

2.1.3. Sistema operativo FreeRTOS

FreeRTOS (Free Real-Time Operating System) [22] es un sistema operativo en tiempo real (RTOS) de código abierto diseñado para microcontroladores y sistemas embebidos. Es ligero, eficiente y está optimizado para ejecutar tareas con

bajas latencias, siendo ideal para dispositivos con recursos limitados. FreeRTOS permite la gestión de tareas concurrentes, sincronización de eventos y comunicación entre tareas, todo en un entorno de ejecución determinista.

2.1.4. Testing unitario

Con el fin de maximizar la calidad durante el proceso de desarrollo del producto se implementaron test unitarios para todos los servicios del robot y del joystick. El conjunto de herramientas utilizadas para tal fin fueron:

- Ceedling [**SoftwareTool_Ceedling**]: herramienta de orquestación de tests unitarios, inyección de objetos mocks.
- CMock [**SoftwareTool_CMock**]: framework de mock objects para sistemas embebidos.
- Unity [**SoftwareTool_Unity**]: framework de unit testing para sistemas embebidos.
- Gcov [**SoftwareTool_CeedlingGCov**]: plugin the ceedling para evaluar y reportar la cobertura.

En los capítulos siguientes se describe la configuración de dichas herramientas y se presentan los resultados tras su ejecución.

2.2. Tecnologías backend utilizadas

2.2.1. Amazon Web Services

Amazon Web Services (AWS) 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 App Runner

AWS App Runner 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.

2.2.3. AWS Glue

AWS Glue es un servicio totalmente administrado de AWS diseñado para facilitar la extracción, transformación y carga de datos (ETL) en la nube que 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. Resulta útil para proyectos de Big Data y análisis de datos ya que brinda un servicio de catálogo de datos, asistencia para la generación de código ETL y un servicio de ejecución de trabajos de procesamiento paralelo.

2.2.4. AWS S3

AWS S3 (Simple Storage Service) es un servicio de almacenamiento de objetos proporcionado por AWS totalmente administrado. Está diseñado para almacenar y recuperar cualquier cantidad de datos desde cualquier lugar, de forma segura, escalable y económica. Es ideal para almacenamiento de datos no estructurado de grandes volúmenes, sitios web estáticos, archivos multimedia, copias de seguridad, etc.

2.2.5. AWS Athena

AWS Athena es un servicio de análisis de datos serverless proporcionado por AWS que permite consultar directamente datos almacenados en Amazon S3 y los esquemas definidos en Glue, utilizando SQL estándar sin necesidad de configurar ni administrar servidores. Es ideal para analizar grandes volúmenes de datos de forma rápida y económica.

2.2.6. MQTT

MQTT (Message Queuing Telemetry Transport) es un protocolo de comunicación asincrónico, ligero y orientado a mensajes, diseñado específicamente para dispositivos con recursos limitados y redes de baja ancho de banda. Es ampliamente utilizado en casos de uso IoT para la transmisión de datos en tiempo real entre dispositivos, sensores y aplicaciones. Utiliza un modelo de comunicación basado en publish/subscribe y una estructura de datos basada en topics a los cuales los componentes clientes se conectan a un servicio broker para publicar o recibir notificaciones.

2.2.7. Node.js

Node.js es un entorno de ejecución de código abierto, construido sobre el motor de JavaScript V8 de Google Chrome. Aunque debido a su diseño puede ser utilizado para desarrollar aplicaciones backend de propósito general que requieran escalabilidad y rendimiento, es utilizado principalmente como servidor web. Para su funcionamiento utiliza un modelo single-thread de event loop con I/O no bloqueante, por lo que gestiona un bucle de eventos encolados y los procesa invocando sus callbacks de forma asincrónica sin realizar bloqueo de entradas y salidas en los puertos de comunicaciones, permitiendo atención de múltiples solicitudes y paralelismo de tareas.

2.3. Tecnologías Blockchain utilizadas

2.3.1. Red Ethereum

Ethereum es una red blockchain pública diseñada para el procesamiento de transacciones de forma descentralizada con almacenamiento distribuido, inmutable y de acceso libre (*permissionless*). La red se encuentra formada por los nodos de procesamiento, también denominados validadores, que tienen como función procesar transacciones y como otras redes blockchain, utiliza una estructura de datos basada en una cadena de bloques, en los cuales se van agrupando las transacciones validadas.

Ethereum tiene como *token* el Ether cuyo símbolo es ETH y tiene varios usos, pudiendo ser utilizado como criptomoneda de cambio y ahorro entre los usuarios finales de la red, pero también para pagar el gas (costo de ejecución de transacciones y smart contracts) y los fees a los validadores.

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) y opera en ranuras de tiempo llamadas slots, con un bloque propuesto aproximadamente cada 12 segundos. El primer paso es la recolección de transacciones enviadas por los usuarios a la red (por ejemplo, para transferir dinero o ejecutar contratos) y su almacenamiento en un *mempool* temporal. Luego, un validador es seleccionado de forma aleatoria para proponer el siguiente bloque, seleccionando del *mempool* aquellas transacciones con tarifas de gas mas altas para maximizar su recompensa. El validador seleccionado incluye: las transacciones válidas, estado actualizado del sistema, hash del bloque anterior y los datos adicionales como la firma del bloque. Como resultado, este bloque es propuesto al resto de la red. Un comité de validadores, seleccionado de forma aleatoria, revisa el bloque verificando que las transacciones sean válidas, el bloque no esté duplicado o malicioso y el bloque sea coherente con el estado de la blockchain. Si el bloque es válido, los validadores emiten un voto (*attestation*) que confirma su aprobación. Si más de 2/3 de los validadores en el comité atestiguan el bloque, se considera finalizado y 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 seguridad, integridad y consenso de la red, aplica una técnica llamada *slashing* para la penalización por acciones malisiosas o incorrectas durante el procesamiento. 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 *stak* si proponen múltiples bloques en un mismo slot, o votan de manera inconsistente (por ejemplo, intentando atacar la red), o están inactivos durante largos periodos de tiempo.

Antes de la versión 2.0 de Ethereum, se utilizaba otro protocolo de consenso llamado PoW (Proof-of-Work), en el cual los nodos validadores desempeñaban el rol de mineros que competían por la generación del bloque, recompensando al que lo lograba generar y desaprovechando los recursos de cómputo utilizados por los que no lo lograron. El protocolo PoS en la version actual de Ethereum tiene varias ventajas con respecto a PoW consumiendo un 99,9% de energía, aumenta la escalabilidad con técnicas de *sharding*, y reduce las barreras de entrada al no requerir disponer de un hardware costoso para poder participar del proceso de validación.

intro a la evm

Se diferencia de otras blockchains, como Bitcoin, porque no es solo un libro mayor digital, sino también una plataforma programable en la cual utilizando un SDK se pueden contruir programas denominados Smart Contracts que se despliegan y ejecutan en la red.

Los Smart Contracts (o contratos inteligentes) son programas informáticos autónomos que se ejecutan en redes blockchain como Ethereum, Solana o Binance

Smart Chain. Están diseñados para automatizar, verificar y hacer cumplir acuerdos sin necesidad de intermediarios. Funcionan bajo el principio de "Si sucede X, entonces ejecutar Y", permitiendo transacciones y acciones de forma transparente, segura e inmutable. Una vez desplegados en la blockchain, no se pueden modificar, y todas las transacciones quedan registradas públicamente.

El ciclo de vida de los Smart Contracts comienza con su desarrollo utilizando alguno de los lenguajes de programación y SDK soportado, como por ejemplo Solidity y Truffle. Una vez desarrollado, el contrato es desplegado en la red de blockchain, siendo este proceso de despliegue una transacción inmutable para la red. Al estar desplegado y disponible puede ser invocado o ejecutado automáticamente cuando se cumplen ciertas condiciones, generando nuevas transacciones inmutables, procesadas por la red y disponibles para ser consultadas.

Ethereum tiene varias redes diseñadas para distintos propósitos, desde la Mainnet (red principal) hasta múltiples testnets para pruebas de desarrollo. Estas redes permiten a desarrolladores y usuarios probar contratos inteligentes y aplicaciones descentralizadas (dApps) sin arriesgar fondos reales. Cada red tiene sus propias características y configuración de parámetros como el consenso, gas fees y emisión de bloques.

2.3.2. Solidity

Solidity [23] es un lenguaje de programación de alto nivel, orientado a contratos inteligentes, específicamente diseñado para funcionar en la Ethereum Virtual Machine (EVM). 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. crear y desplegar contratos inteligentes (smart contracts) en Ethereum y otras blockchains compatibles con EVM, como Binance Smart Chain y Polygon. Estos contratos inteligentes automatizan y aseguran transacciones sin necesidad de intermediarios, siendo usados en:

2.3.3. Biblioteca Web3.js

blablabla

2.3.4. Ganache

Ganache [24] 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).

2.3.5. Truffle

Truffle [25] es un framework de desarrollo para Ethereum y otras blockchains compatibles con EVM (Ethereum Virtual Machine). 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.

2.3.6. Alchemy

Como se mencionó mas arriba, la dApp, implementada como un servicio Node.js, es la responsable de invocar al Smart Contract desplegado en Ethereum utilizando un endpoint RPC publicado por cualquier nodo de la red. Debido a que los nodos públicos de la red pueden resultar limitados por motivos de seguridad, rendimiento y confiabilidad, resulta una mejor alternativa levantar un nodo EVM administrado o consumir esto como un servicio de un proveedor como Alchemy. Alchemy [26] es una plataforma de desarrollo blockchain que proporciona herramientas e infraestructura para crear y gestionar dApps en Ethereum y otras redes compatibles con EVM. Es conocida como el "AWS de Blockchain" debido a que 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.

2.3.7. Etherscan

Etherscan [27] 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).

2.3.8. Metamask

MetaMask [28] 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 tu 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.

2.4. Tecnologías de desarrollo utilizadas

2.4.1. Plataforma Docker

Docker [29] es un proyecto de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores de software, proporcionando una capa adicional de abstracción y automatización de virtualización de aplicaciones en múltiples sistemas operativos. Docker utiliza características de aislamiento de recursos del kernel Linux, tales como cgroups y espacios de nombres (namespaces) para

permitir que contenedores livianos independientes se ejecuten en paralelo de manera aislada evitando la sobrecarga de iniciar y mantener máquinas virtuales.

2.4.2. 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 [30]: servicio de repositorio y control de versiones de código fuente.
- AWS CodePipeline [31]: servicio de compilación, empaquetado y ejecución *builds*.
- AWS Elastic Container Registry [32]: 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 código compilado y se versiona en Artifact Registry.

2.4.3. Visual Studio Code

Visual Studio Code [33] 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.

2.4.4. Sistema operativo Ubuntu

Ubuntu [34] 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

El sistema cuenta con una arquitectura robusta y flexible en la que se integra el dispositivo robótico de exploración ambiental [35] desarrollado en el marco de la Carrera de Especialización de Sistemas Embebidos, con un sistema *back-end* desplegado en la nube pública [36], y una red Blockchain [13] a fin de poder asegurar la inmutabilidad y transparencia de las lecturas ambientales.

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.

3.1. Análisis del software

La idea de esta sección es resaltar los problemas encontrados, los criterios utilizados y la justificación de las decisiones que se hayan tomado.

Se puede agregar código o pseudocódigo dentro de un entorno `lstlisting` con el siguiente código:

```
\begin{lstlisting}[caption= "un epígrafe descriptivo"]
  las líneas de código irían aquí...
\end{lstlisting}
```

A modo de ejemplo:

```
1 #define MAX_SENSOR_NUMBER 3
2 #define MAX_ALARM_NUMBER 6
3 #define MAX_ACTUATOR_NUMBER 6
4
5 uint32_t sensorValue[MAX_SENSOR_NUMBER];
6 FunctionalState alarmControl[MAX_ALARM_NUMBER]; //ENABLE or DISABLE
7 state_t alarmState[MAX_ALARM_NUMBER]; //ON or OFF
8 state_t actuatorState[MAX_ACTUATOR_NUMBER]; //ON or OFF
9
10 void vControl() {
11
12     initGlobalVariables();
13
14     period = 500 ms;
15
16     while(1) {
17
18         ticks = xTaskGetTickCount();
19
20         updateSensors();
```

```
21  
22     updateAlarms () ;  
23  
24     controlActuators () ;  
25  
26     vTaskDelayUntil(&ticks , period) ;  
27 }  
28 }
```

CÓDIGO 3.1. Pseudocódigo del lazo principal de control.

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 pudo 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. *LoraWAN*. URL: <https://es.wikipedia.org/wiki/LoRaWAN>.
- [3] 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.
- [4] 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/>.
- [5] Los40.com | Dani Cabezas. *Así se gestiona un “tecnobosque”*. URL: <https://los40.com/2024/12/10/asi-se-gestiona-un-tecnobosque/>.
- [6] Boston Dynamics. *Spot*. URL: <https://www.bostondynamics.com/products/spot>.
- [7] Waygate Technologies. *BIKE - An advanced crawler robot for remote visual inspection*. URL: <https://www.bakerhughes.com/waygate-technologies/robotic-inspection/bike>.
- [8] Latam Mining. *Robots y minería: Gobierno argentino quiere implementarlos*. URL: <https://www.latam-mining.com/robots-y-mineria-gobierno-argentino-quiere-implementarlos/>.
- [9] 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>.
- [10] Universidad Nacional de San Juan. *Robots en la minería*. URL: http://www.unsj.edu.ar/home/noticias_detalle/4810/1.
- [11] Ing. Nelson Dario García Hurtado e Ing. Melvin Andrés González Pino. *Robot de exploración terrestre Geobot*. URL: https://www.unipamplona.edu.co/unipamplona/portallG/home_40/recursos/01_general/revista_1/09102011/v01_09.pdf.
- [12] 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.
- [13] Wikipedia. *Blockchain*. URL: <https://en.wikipedia.org/wiki/Blockchain>.
- [14] Wikipedia. *Smart Contracts*. URL: https://en.wikipedia.org/wiki/Smart_contract.
- [15] Wikipedia. *DApp - Decentralized Application*. URL: https://en.wikipedia.org/wiki/Decentralized_application.
- [16] OASIS. *MQTT Protocol Specification*. URL: <https://mqtt.org/mqtt-specification/>.
- [17] Espressif. *ESP32*. URL: <https://www.espressif.com/en/products/socs/esp32>.
- [18] CMake. *CMake*. URL: <https://cmake.org/>.

- [19] CMake. *Ninja*. URL: <https://cmake.org/cmake/help/latest/generator/Ninja.html>.
- [20] Espressif. *ESP-IDF Programming Guide | Get Started*. URL: <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/get-started/>.
- [21] Readthedocs by Ruslan V. Uss. *ESP-IDF Components library*. URL: <https://esp-idf-lib.readthedocs.io/en/latest/>.
- [22] FreeRTOS. *FreeRTOS | Real-time operating system for microcontrollers and small microprocessors*. URL: <https://www.freertos.org/>.
- [23] soliditylang.org. *Solidity programming language*. URL: <https://soliditylang.org/>.
- [24] Truffle Suite. *Ganache | One click blockchain*. URL: <https://archive.trufflesuite.com/ganache/>.
- [25] Truffle Suite. *Truffle | What is Truffle?* URL: <https://archive.trufflesuite.com/docs/truffle/>.
- [26] Alchemy. *Alchemy | The most reliable way to build web3 apps*. URL: <https://www.alchemy.com/>.
- [27] Etherscan. *Etherscan*. URL: <https://etherscan.io/>.
- [28] Metamask. *Metamask | Your home in Web3*. URL: <https://metamask.io/>.
- [29] Docker. *Docker*. URL: <https://docker.com/>.
- [30] Github. *Github*. URL: <https://github.com/>.
- [31] Amazon Web Services. *AWS CodePipeline*. URL: <https://aws.amazon.com/codepipeline/>.
- [32] Amazon Web Services. *AWS Elastic Container Registry*. URL: <https://aws.amazon.com/ecr/>.
- [33] Visualstudio. *Visualstudio Code*. URL: <https://code.visualstudio.com/>.
- [34] Ubuntu. *Ubuntu*. URL: <https://ubuntu.com/>.
- [35] 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>.
- [36] Amazon Web Services. *¿Qué es una nube pública?* URL: <https://aws.amazon.com/es/what-is/public-cloud/>.