ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

Facultad de Ingeniería en Electricidad y Computación

Sistema de monitoreo para clientes de proveedores de Internet utilizando contratos inteligentes

TRABAJO DE TITULACIÓN

Previo la obtención del Título de:

Magíster en Telecomunicaciones

Presentado por:

Harry Gustavo Mendoza Moreira
David Arturo Navarrete Mora

GUAYAQUIL - ECUADOR
Año: 2025

DEDICATORIA

Este trabajo de titulación va dedicado a mis padres, por su apoyo incondicional y motivación que han permitido poder completar cada paso de este proceso.

También se lo dedico a mis hermanos y amigos que con su aliento fueron un impulso extra para superar los retos que se presentaron en el camino.

Harry Gustavo Mendoza Moreira

Dedico este proyecto a mis padres, gracias a su esfuerzo, enseñanzas y apoyo han guiado mi camino en cada una de las etapas de mi vida. Me han servido de inspiración para alcanzar mis metas y superar cualquier desafío.

David Arturo Navarrete Mora

AGRADECIMIENTOS

Mi agradecimiento a cada uno de los docentes por impartirnos conocimientos valiosos. A mi compañero de este trabajo de titulación por su colaboración y trabajo en equipo.

Agradezco a mi familia por su apoyo y aliento, los cuales fueron fundamentales para el desarrollo de este trabajo.

Harry Gustavo Mendoza Moreira

Agradezco a mis padres por su motivación y sacrificio incesante. También mi agradecimiento a mis docentes por su valiosa guía durante el desarrollo de este proyecto. A mis compañeros, por su colaboración y apoyo, y a todos aquellos que aportaron de alguna forma a la realización de este trabajo

David Arturo Navarrete Mora

Declaración Expresa

Nosotros Harry Gustavo Mendoza Moreira y David Arturo Navarrete Mora acordamos y reconocemos que: La titularidad de los derechos patrimoniales de autor (derechos de autor) del proyecto de graduación corresponderá al autor o autores, sin perjuicio de lo cual la ESPOL recibe en este acto una licencia gratuita de plazo indefinido para el uso no comercial y comercial de la obra con facultad de sublicenciar, incluyendo la autorización para su divulgación, así como para la creación y uso de obras derivadas. En el caso de usos comerciales se respetará el porcentaje de participación en beneficios que corresponda a favor del autor o autores. El o los estudiantes deberán procurar en cualquier caso de cesión de sus derechos patrimoniales incluir una cláusula en la cesión que proteja la vigencia de la licencia aquí concedida a la ESPOL.

La titularidad total y exclusiva sobre los derechos patrimoniales de patente de invención, modelo de utilidad, diseño industrial, secreto industrial, secreto empresarial, derechos patrimoniales de autor sobre software o información no divulgada que corresponda o pueda corresponder respecto de cualquier investigación, desarrollo tecnológico o invención realizada por nosotros durante el desarrollo del proyecto de graduación, pertenecerán de forma total, exclusiva e indivisible a la ESPOL, sin perjuicio del porcentaje que me/nos corresponda de los beneficios económicos que la ESPOL reciba por la explotación de nuestra innovación, de ser el caso.

En los casos donde la Oficina de Transferencia de Resultados de Investigación (OTRI) de la ESPOL comunique a los autores que existe una innovación potencialmente patentable sobre los resultados del proyecto de graduación, no se realizará publicación o divulgación alguna, sin la autorización expresa y previa de la ESPOL.

Guayaquil, 11 febrero del 2025.

Firmado electrónicamente por:
HARRY GUSTAVO
MENDOZA MOREIRA

Ing. Harry Gustavo
Mendoza Moreira

Firmado electrónicamente por:
DAVID ARTURO
NAVARRETE MORA

Ing. David Arturo
Navarrete Mora

EVALUADORES

Firmado electrónicamente por:
RONALD RAUL CRIOLLO BONILLA

Msig. Ronald Raúl Criollo Bonilla

DIRECTOR

Firmado electrónicamente por:
FRANCISCO VICENTE NOVILLO PARALES

PhD. Francisco Vicente Novillo Parales

EVALUADOR

RESUMEN

El proyecto tiene como propósito diseñar un sistema de monitoreo para clientes de proveedores de Internet basado en contratos inteligentes asegurando el cumplimiento de los Acuerdos de Nivel de Servicio (SLA). Su finalidad radica en supervisar el servicio y aplicar compensaciones de forma automatizada, garantizando eficiencia y transparencia en la gestión del SLA. Surge dada la necesidad de mejorar la respuesta ante indisponibilidad del servicio e incrementar la confianza en proveedores y usuarios con tecnologías descentralizadas.

El desarrollo implicó la simulación mediante GNS3 de un entorno de red, recreando la conexión hasta el cliente final. El sistema de monitoreo se implementó con Zabbix, el cual, ante periodos de indisponibilidad, generaba alertas. Los datos recolectados antes de llegar a la red blockchain, simulada en Ganache, fueron procesados con scripts en Python. Mediante Truffle se desplegó el contrato inteligente, el cual automatizó las estipulaciones del SLA. Además, se desarrolló una interfaz gráfica mediante Node-RED, InfluxDB y Grafana para la presentación de las compensaciones y porcentajes de disponibilidad.

Las pruebas evidenciaron que el monitoreo de la disponibilidad del servicio es efectivo y que las compensaciones establecidas en el SLA fueron aplicadas automáticamente. Así mismo, la aplicación de tecnología Blockchain brindó seguridad e inmutabilidad y la interfaz gráfica proporcionó una forma amigable de observar la información en tiempo real.

Finalmente, la propuesta desarrollada optimiza considerablemente la gestión del SLA, garantizando transparencia en el cumplimiento de las cláusulas, mejora la respuesta ante incidentes y a su vez la satisfacción y confianza del usuario.

Palabras clave: contratos inteligentes, blockchain, SLA, inmutabilidad.

ABSTRACT

The project aims to design a monitoring system for Internet service provider customers based on smart contracts, ensuring compliance with Service Level Agreements (SLA). Its purpose is to supervise the service and apply compensations automatically, guaranteeing efficiency and transparency in SLA management. It arises from the need to improve response times to service unavailability and increase trust between providers and users through decentralized technologies.

The development involved simulating a network environment using GNS3, recreating the connection to the end customer. The monitoring system was implemented with Zabbix, which generated alerts during periods of unavailability. The collected data, before reaching the blockchain network simulated in Ganache, was processed using Python scripts. The smart contract was deployed with Truffle, automating the SLA stipulations. Additionally, a graphical interface was developed using Node-RED, InfluxDB, and Grafana to present compensations and availability percentages.

The tests showed that service availability monitoring is effective and that the compensations established in the SLA were applied automatically. Likewise, the use of blockchain technology provided security and immutability, while the graphical interface offered a user-friendly way to visualize real-time information.

Finally, the proposed system significantly optimizes SLA management, ensuring transparency in compliance with its clauses, improving incident response, and enhancing user satisfaction and trust.

Keywords: *smart contracts, blockchain, SLA, immutability.*

ÍNDICE GENERAL

ABREVIATURAS

ESPOL	Escuela Superior Politécnica del Litoral
SLA	Service Level Agreement
QoS	Quality of Service
CPE	Customer Premises Equipment
SNMP	Simpe Network Management Protocol
DLT	Distributed Ledger Technology
PoW	Proof of Work
PoS	Proof of Stake
IoT	Internet of Things
GNE3	Graphical Network Simulator 3
OSPF	Open Shortest Path First
ISP	Internet Service Provider

SIMBOLOGÍA

ms	milisegundo
%	porcentaje
ETH	Ether

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

CAPÍTULO 1

1. INTRODUCCIÓN

En el mercado de las telecomunicaciones, el cumplimiento de los Acuerdos de Nivel de Servicio (SLA siglas en inglés) se ha convertido en un factor clave para garantizar la satisfacción del cliente y mantener la confianza en los proveedores de Internet. Sin embargo, el cumplimiento de los tiempos del SLA a menudo presentan desafíos debido a la falta de automatización y la dependencia de procesos manuales. Esta situación puede generar inconformidad tanto a los usuarios finales como a la reputación de las empresas. Ante esta problemática, surge la necesidad de adoptar soluciones tecnológicas que permitan un control más preciso y en tiempo real sobre el cumplimiento de los SLA.

El presente proyecto consiste en la implementación de un sistema de monitoreo basado en contratos inteligentes, capaz de automatizar la supervisión del estado del servicio y garantizar la ejecución de medidas correctivas en caso de incumplimiento. A través de la integración de tecnologías avanzadas de blockchain y monitoreo de red, el sistema permitirá a los proveedores de telecomunicaciones detectar variaciones en la calidad del servicio y activar automáticamente los términos de los contratos acordados con sus clientes. Esta automatización no solo mejorará la eficiencia operativa, sino que también incrementará la transparencia y la seguridad en la gestión de los SLA.

Se espera que la adopción de contratos inteligentes contribuya a la optimización de recursos, reduciendo tiempos de respuesta ante incidentes y minimizando el uso de procesos manuales. A largo plazo, esta solución podría convertirse en un estándar en la industria, proporcionando una ventaja competitiva para las empresas que adopten esta tecnología, mientras mejoran la calidad del servicio y fortalecen la relación con sus clientes.

1.1. Descripción del problema

Las compañías de telecomunicaciones afrontan grandes retos cuando se trata de mantener la calidad de su servicio y los Acuerdos de Nivel de Servicio establecidos

con cada uno de sus clientes. En los SLAs se detallan responsabilidades contractuales tanto del proveedor como del usuario, entre las cuales se puede mencionar la disponibilidad del servicio, mantenimiento, soporte, penalizaciones, compensaciones, entre otros. (Contreras, 2024)

De este modo, cuando la disponibilidad del servicio resulta interrumpida y sobrepasa el tiempo considerado en el acuerdo, la compañía proveedora se ve obligada a efectuar las compensaciones correspondientes establecidas; lo que no solo afecta financieramente a la empresa, también ocasiona una disminución de la confianza de sus clientes actuales y de los potenciales clientes. Usualmente el cálculo de este proceso de incidentes y compensaciones es realizado de forma manual por el departamento de calidad, lo que puede llegar a crear inconformidad entre las partes, debido a errores humanos o mal interpretaciones de lo establecido.

Por otro lado, en los acuerdos de Calidad de Servicio (QoS por sus siglas en inglés) entre un proveedor y un cliente, es fundamental establecer parámetros clave como el ancho de banda, el cual define la capacidad máxima de transferencia de datos en la red, garantizando así que no se creen cuellos de botella. Otros de los parámetros es la latencia, que mide el tiempo que tarda un paquete en llegar a su destino, siendo crucial para aplicaciones en tiempo real, así como la pérdida de paquetes, que se refiere al porcentaje de datos que no llegan a su destino durante la transmisión, afectando la calidad del servicio. Así mismo, el jitter, que es la variación en el tiempo de llegada de los paquetes y puede impactar negativamente en servicios como la voz sobre IP; y la disponibilidad, que indica el porcentaje de tiempo en que el servicio está operativo y accesible, siendo esencial para asegurar una experiencia de usuario confiable y continua. (VAS Expert, 2023)

Una empresa que incumple con los SLA establecidos se ve perjudicada por múltiples factores, como pérdidas económicas significativas. Estudios demuestran que 78 % de los clientes consideran la calidad de servicio fundamental para aumentar su lealtad hacia el negocio. Además, cerca del 50% de clientes esperan una respuesta dentro de una hora ante inconvenientes y el 18% una respuesta inmediata. (Max, 2022)

En definitiva, estos factores generan un efecto adverso en las empresas de telecomunicaciones perjudicándolas financieramente y en su reputación.

1.2. Justificación del problema

En el panorama actual a nivel nacional, las compañías de telecomunicaciones enfrentan grandes dificultades para cumplir con los SLA, las cuales desencadenan una disminución en la confianza de sus clientes y perjudica sus ingresos. No obstante, la competencia en el mercado local obliga a estas empresas a mejorar constantemente sus servicios, buscando ofrecer la mejor calidad posible. En este contexto, resulta indispensable perfeccionar los sistemas de monitoreo y control para garantizar un servicio más eficiente y confiable. (SeguriLatam, 2022)

En este sentido, no cumplir con los SLA, como ya se mencionó, trae consigo efectos adversos para el proveedor del servicio. Estos pueden ir desde un daño a su reputación hasta sanciones y multas financieras. A pesar de esto, los mecanismos de monitoreo tradicional no resultan efectivos para asegurar la claridad e inmutabilidad de los datos para satisfacer a cabalidad lo establecido en los SLA. (Contreras, 2024)

De este modo, es esencial encontrar la solución al problema para conservar la competitividad y fidelidad de los usuarios. Con el cumplimiento de los SLA además de evitar sanciones y multas, se incrementa el compromiso y lealtad del usuario, elementos cruciales en un mercado cada vez más competitivo.

En función de lo mencionado, este proyecto plantea un sistema de monitoreo de clientes de compañías de telecomunicaciones vinculado con un contrato inteligente, para ayudar a satisfacer los SLA. Esto facilitará el seguimiento y supervisión de forma regular a los equipos de los usuarios con el objetivo de identificar problemas de forma anticipada, permitiendo respuestas más eficaces y rápidas ante estos inconvenientes, disminuyendo los tiempos de inoperatividad del servicio antes de que impacte negativamente en la experiencia del cliente. El sistema de notificación permitirá alertar al usuario y al proveedor de manera inmediata permitiendo una

gestión proactiva de incidencias y mejorando la capacidad de respuesta. El vínculo con el contrato inteligente asegurará que los tiempos de indisponibilidad o degradación del servicio sean respetados por las partes implicadas de forma imparcial.

A pesar de que existen otras alternativas de mecanismos de monitoreo y control, la seguridad en el cumplimiento del SLA a través de los contratos inteligentes, lo hace muy especial. Esta tecnología permite que los registros y la comunicación sean inalterables y pueden ser auditados con facilidad, mejorando las características de los sistemas convencionales. (Intelligence, 2024). Este tipo de contratos se desarrollan con la finalidad de ejecutar de forma automática los términos y condiciones de un acuerdo cuando especificaciones predefinidas no se han cumplido. Como se ejecutan en una Blockchain y son establecidos mediante un lenguaje de programación, estos contratos no necesitan de intermediarios, lo que garantiza que los acuerdos y transacciones se ejecuten de forma transparente y segura. (Jorge, 2020)

De esta forma, al integrar el contrato inteligente al sistema de monitoreo y control, se automatizarán varios procesos permitiendo que se cumplan las condiciones y compromisos entre los involucrados, sin necesidad de intervención humana. Esto a su vez garantizará el cumplimiento de los SLA al monitorear constantemente el estado del servicio en tiempo real, lo que le permitirá al contrato inteligente detectar cualquier incumplimiento de los SLA y responder de forma ágil, ejecutando las sanciones establecidas o notificando a los responsables. Con estas automatizaciones además de que los SLA se cumplan de forma exhaustiva, se disminuye los tiempos de respuesta y costos operativos asociados, e incrementa la eficiencia del sistema de monitoreo.

Desde el punto de vista económico, la reducción de ingresos ocasionados por descuentos en facturas a los clientes se transforma en un beneficio inmediato y a la vez aumenta la fidelidad de los usuarios y asegura ingresos a largo plazo. De esta manera, cumplir con los SLA permite al proveedor ser mucho más competitivo ante las demás compañías. En cuanto al ámbito social, la relación entre el cliente y

el proveedor se fortifica gracias a la confiabilidad del servicio, generando una buena reputación de la empresa hacia clientes potenciales. (Swain & Garza, 2022)

Por consiguiente, este proyecto no simplemente se justifica por motivos económicos y técnicos, sino también por su influencia positiva en la reputación de la compañía, las obligaciones regulatorias cumplidas y la satisfacción del usuario.

1.3. Objetivos

1.3.1. Objetivo general

Implementar un sistema de monitoreo que utilice contratos inteligentes que permitan a las empresas proveedoras del servicio de Internet la detección y resolución de interrupciones del servicio en tiempo real, garantizando el cumplimiento de los SLA.

1.3.2. Objetivos específicos

- Implementar el sistema de monitoreo que supervise el estado del servicio en tiempo real, facilitando la detección y resolución inmediata de cualquier variación de parámetros.
- Desarrollar un contrato inteligente que garantice en el sistema de monitoreo la transmisión segura de datos, facilitando la adecuada supervisión de la red y el cumplimiento de los acuerdos de servicio.
- Integrar el sistema de monitoreo con el Contrato Inteligente para que se automatice la verificación de los SLA, la gestión de interrupciones y la configuración de notificaciones de alerta al equipo técnico y al cliente.
- Ejecutar un plan de pruebas para la validación del correcto funcionamiento de la solución propuesta.

1.4. Propuesta de solución

Dado el crecimiento constante de las exigencias de los usuarios y la necesidad de mantener altos niveles de servicio en el sector de las telecomunicaciones, la propuesta de este proyecto se enfoca en la implementación de un sistema de monitoreo automatizado vinculado a contratos inteligentes para garantizar el cumplimiento de los Acuerdos de Nivel de Servicio (SLA). Esta solución se basa en el uso de tecnologías emergentes como contratos inteligentes para asegurar la transparencia, confiabilidad y automatización en la supervisión del servicio, permitiendo que cualquier falla o interrupción sea detectada y resuelta de manera rápida y eficiente.

El sistema propuesto operará a través de un sistema de monitoreo en tiempo real que recopilará continuamente datos sobre el estado de la red. En caso de que se detecten desviaciones en los parámetros establecidos en el SLA (como interrupciones del servicio), el sistema activará automáticamente un contrato inteligente preconfigurado. Este contrato inteligente garantizará que se ejecuten las acciones estipuladas, tales como penalizaciones o compensaciones automáticas al cliente afectado, también se generarán alertas inmediatas para el equipo técnico y al usuario. Además, las métricas de evaluación y la compensación serán presentadas por medio de una interfaz gráfica.

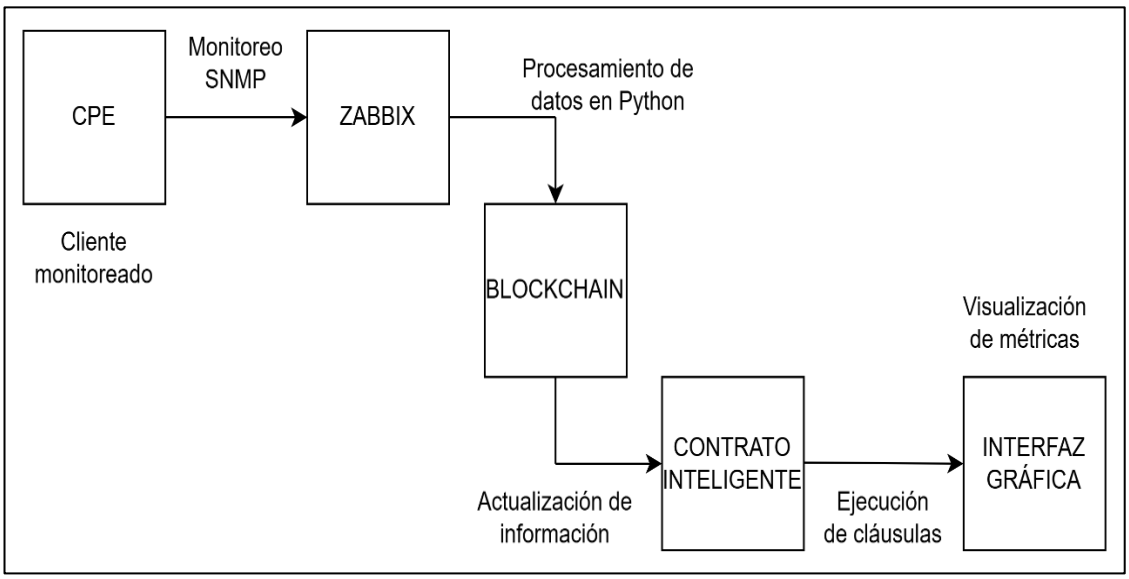


Figura 1.1 Diagrama de bloques de la solución

La metodología para implementar esta solución incluye varios pasos clave. En primer lugar, se realizará un análisis del sistema de monitoreo de la empresa de telecomunicaciones. Luego, se desarrollarán los contratos inteligentes, los cuales estarán programados para interactuar con los datos obtenidos por el sistema de monitoreo. Estos contratos no solo facilitarán la verificación automática del cumplimiento de los SLA, sino que también asegurarán la integridad de la información mediante la tecnología blockchain, como se refleja en la Figura 1.1. También, se configurarán alertas automáticas para que los equipos de soporte reciban notificaciones inmediatas en caso de incidentes, optimizando la respuesta y reduciendo el tiempo de inactividad. Finalmente, se desarrollará una interfaz gráfica para que las métricas principales sean visualizadas en un entorno amigable.

1.5. Alcance y consideraciones

El presente proyecto está enfocado a los proveedores de servicios, que ofrecen datos e Internet. Debido a que considera parámetros de calidad de servicio como tiempos de ping dentro del contexto de un servicio típico de Internet que utiliza como medio de transporte la fibra óptica. Además de estos aspectos esenciales para la calidad de la conexión el proyecto requiere la utilización de equipos compatibles con SNMP (Simple Network Management Protocol) en este caso CPE de capa 3 los cuales son comúnmente empleados en la prestación de servicios de Internet destinados a clientes corporativos. Las consideraciones técnicas y funcionales del proyecto se centran en garantizar la fiabilidad de los tiempos y cumplimientos del SLA.

Al considerar el monitoreo de otro tipo de servicio en lugar de uno de Internet, las modificaciones en el proyecto incluirían diferentes parámetros en el monitoreo e infraestructura que sea compatible con SNMP, la seguridad y privacidad de los datos a través de la tecnología blockchain seguiría siendo esencial, pero se centraría en proteger las comunicaciones y datos personales de los usuarios.

CAPÍTULO 2

2. MARCO TEÓRICO

En este apartado se abordarán conceptos teóricos esenciales en los que se fundamenta el desarrollo de este proyecto de titulación. Se analizarán los principios de la tecnología Blockchain, contratos inteligentes, y la gestión de los acuerdos de nivel de servicio (SLA). Asimismo, detallará la adopción de los sistemas de monitoreo actualizados, especialmente los que emplean la transmisión segura y eficiente de datos.

Este esquema facilitará la comprensión de la importancia de integrar estas tecnologías para satisfacer el cumplimiento de los SLA utilizando mecanismos automatizados.

2.1. Tecnología Blockchain

La tecnología Blockchain ha transformado la manera en que se gestionan y protegen los datos en entornos digitales, donde a pesar de ser la base de las criptomonedas, su potencial va más allá del ámbito financiero. Esta tecnología se fundamenta en un sistema de registro distribuido (DLT, por sus siglas en inglés Distributed Ledger Technology), en el cual los datos son almacenados a través de bloques encadenados en orden cronológico y de forma inmutable. De la misma forma, estos bloques a través de criptografía avanzada se encuentran protegidos de cualquier vulnerabilidad y solo pueden ser validados o agregados a la cadena por medio de algoritmos de consenso como Proof of Work (PoW) y Proof of Stake (PoS), donde el primero se basa en la resolución de problemas matemáticos complejos y el segundo se fundamenta en la cantidad de criptomonedas que los participantes poseen en la red. Estos algoritmos son empleados para validar las transacciones y asegurar la integridad de la cadena de bloques sin requerir un intermediario central. (Alfaro, 2021)

2.1.1. Características de redes blockchain

Las redes blockchain cuentan con características esenciales que la distinguen y ofrecen ventajas sobre otros tipos de tecnologías, como lo son la transparencia y seguridad, la irrevocabilidad, la inmutabilidad y la descentralización.

- Transparencia y Seguridad: En la red todos los nodos pueden acceder a una copia del libro mayor (ledger) asegurando la transparencia en el proceso. No obstante, mediante técnicas de cifrado los datos críticos se mantienen seguros, como datos personales o empresariales.
- Irrevocabilidad: Cuando los datos se encuentran dentro de la red, ya no pueden ser eliminados. De esta forma, al estar en la red, la información es distribuida para todos los nodos de la red.
- Inmutabilidad: Es una característica esencial de blockchain, dado que inmediatamente después que los datos son registrados en un bloque y son añadidos a la cadena, estos no pueden ser alterados sin que dicho cambio se detecte. Para conseguir esto, se emplean funciones hash criptográficas, de forma que la mínima alteración de la información repercutirá en todo el hash del bloque, así todo evento de cambio es detectado.
- Descentralización: Como se ha mencionado, Blockchain se caracteriza por compartir la información entre todos los nodos presentes en la red, en contraste con sistemas centralizados donde una unidad central gestiona los datos. Esto permite que ante un intento de alteración de los datos sea necesario cambiar todas las copias distribuidas en cada nodo, siendo una tarea muy complicada. **(Porxas & Conejero, 2018)**

2.1.2. Tipos de redes blockchain

Las características mencionadas en la sección 2.1.1. las poseen todas las redes blockchain. A pesar de esto, considerando otros criterios se pueden agrupar las redes blockchain en dos grupos: las redes públicas y las privadas.

Las redes públicas se caracterizan por carecer de jerarquía entre los nodos, esto permite que cualquiera acceda de manera que generen y validen bloques. Para tener seguridad en este tipo de bloques se emplea el algoritmo de consenso PoW, resolviendo problemas matemáticos complejos, para así realizar la validación de un bloque antes de integrarlo a la cadena. Por otro lado, en las redes privadas los actores que pueden cuentan con acceso y capacidad para sumar nuevos bloques

es muy limitada. De igual forma, no cualquier nodo puede realizar el proceso de validación de bloques, debido a que usualmente si existe jerarquía entre ellos. **(Porxas & Conejero, 2018)**

2.1.3. Funcionamiento

Esta tecnología tiene como fundamento la interconexión de nodos de manera que todos son capaces de procesar validar las transacciones en una red distribuida, permitiendo que no sea necesario depender de un actor central. En contraste una red centralizada que, si requiere de esta autoridad validadora central, para que se ejecuten las transacciones, tal como se aprecia en las Figuras 2.1 y 2.2

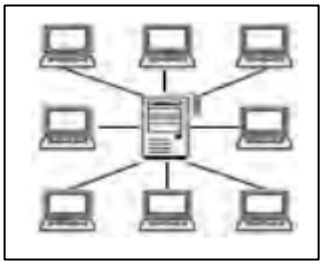


Figura 2.1 Red centralizada *(Porxas & Conejero, 2018)*

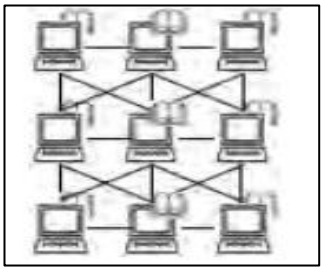


Figura 2.2 Red distribuida *(Porxas & Conejero, 2018)*

De forma general, para que cada transacción sea verificada, se emplea un algoritmo de consenso, una vez que pasa este proceso, la transacción es aprobada y seguidamente agregada a un bloque. Así, como su nombre lo indica, el bloque creado se enlaza con el anterior, generando una cadena constante y comprobable de sucesos, tal como se muestra en la Figura 2.3. Bajo este mecanismo, la tecnología blockchain es capaz de, aun contando con actores desconocidos, garantizar la seguridad e integridad de los datos. (Porxas & Conejero, 2018)

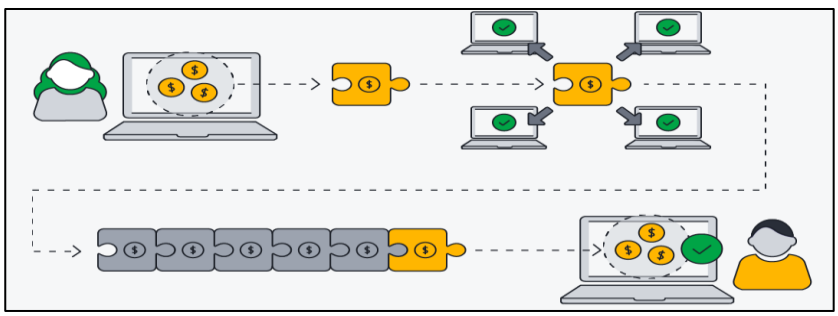


Figura 2.3 Funcionamiento de blockchain *(Freda, 2022)*

Un ejemplo de lo mencionado en el párrafo anterior se da en el proceso de validación en Bitcoin, la que se encuentro dentro de las aplicaciones más populares de esta tecnología. De esta forma, en intervalos de 10 minutos, las transacciones se congregan en un bloque, para posteriormente ser minado por nodos denominados “mineros”. Dichos nodos solucionan problemas criptográficos complejos, accediendo a una recompensa el que lo ejecuta en el menor tiempo posible, lo que también garantiza que el bloque sea integrado a la cadena. (Nakamoto, 2008)

2.1.4. Aplicaciones importantes

La tecnología blockchain tiene muchas aplicaciones en otras áreas, además de la financiera y criptomonedas. Ha permitido grandes avances en sectores como la logística, sistemas de energía y salud debido a su seguridad de la información y la trazabilidad en los procesos. Del mismo modo, tomando en cuenta lo mencionado en las secciones 2.1.2. y 2.1.3., esta tecnología permite grandes avances en sectores que demanden de almacenamiento de datos, donde requieran que la información y acceso a la misma sea compartido entre los distintos actores, y donde dichos actores sean desconocidos. (Porxas & Conejero, 2018)

Bajo este contexto son muchas las actividades que se pueden desarrollar en diferentes sectores, como es el caso de las criptomonedas. Asimismo, en el ámbito financiero, esta tecnología ha permitido implementar sistemas de pagos mucho más seguros y rápidos, debido a que no se requieren intermediarios, lo que reduce los gatos considerablemente. También, en las cadenas de suministro, la blockchain es capaz de rastrear de inicio a fin un objeto o producto, garantizando la legitimidad y transparencia de los datos. Asimismo, en la industria se plantea que un sistema de drones con la ayuda de tecnología blockchain podría disminuir el costo en trabajos repetitivos de almacén y a al mismo tiempo otorgar una trazabilidad. (Olaizola, 2020)

2.2. Contratos Inteligentes

Los contratos inteligentes (Smart Contracts, traducidos al inglés) han revolucionado la definición común de contratos, dado que hacen posible la ejecución automática de los mismas sin que se requiera de intermediarios humanos. La propuesta de contratos inteligentes surgió originalmente en 1994 por el informático y criptógrafo Nick Szabo estableciéndolos como un protocolo de transacción informático que lleva a cabo condiciones establecidas de un contrato. El propósito de esta propuesta era la de otorgar seguridad al cumplimiento del contrato, previniendo futuros conflictos y en consecuencia problemas legales. **(Fetsyak, 2020)**

Sin embargo, esta tesis no tuvo la repercusión esperada en aquella época, debido a ciertas limitaciones para su implementación. Entre dichas limitaciones se encontraba la imposibilidad de un código de programación capaz de gestionar y supervisar activos reales con el objetivo de cumplir los acuerdos. Por otro lado, existía una estricta regulación legal en cuanto al movimiento de activos. Asimismo, era complicado contar con código en el que las partes contratantes puedan confiar, garantizando que se ejecute íntegramente lo acordado y que sea inmune de manipulaciones. **(Fetsyak, 2020)**

De esta forma, con la aparición de blockchain y plataformas orientadas en la implementación de contratos inteligentes, fue como estos consiguieron un entorno ideal para su desarrollo y uso masivo. La blockchain solventa todos los impedimentos mencionados en el párrafo anterior, dado que permite controlar el activo mediante un código asociado a dicho activo. Por otro lado, en cuanto a la regulación de movimiento de activos, con la blockchain se desarrollan nuevos sistemas de pago. Del mismo modo, esta tecnología al contener al contrato, le proporciona la confianza dado el consenso que se realiza en toda la cadena de bloques, previniendo cualquier manipulación. **(Fetsyak, 2020)**

2.2.1. Definición de contratos inteligentes

A un contrato inteligente se lo puede considerar como un código o protocolo informático que es capaz de verificar y garantizar el cumplimiento de un contrato de manera automática. Se lo considera uno de los fundamentos de la blockchain

debido a su operación descentralizada y fidelidad de ejecución en su programación, sin riego de periodos de inactividad, fraude, injerencia de terceros o necesidad de intermediarios. **(Gómez, 2018)**

Gracias a estas funcionalidades, los contratos inteligentes ahorran tiempo y garantiza que las partes involucradas cumplan sus contratos en el mundo real sin requerir de la intervención o mediación de actores físicos. En contraste a lo que sucede con un contrato convencional, lo cuales demandan de intermediarios, los contratos inteligentes al no contar con esta necesidad, evita errores humanos en el proceso y en su defecto un mejor aprovechamiento del tiempo. **(Gómez, 2018)**

Este tipo de contratos opera siguiendo la lógica condicional “if-then”, siguiendo estas reglas lógicas como todo programa informático, sin embargo, estos contratos interactúan con activos reales. Por consiguiente, al cumplirse una condición preprogramada, contrato inteligente ejecuta la cláusula contractual preestablecida, sin necesidad de intervención humana. **(Gómez, 2018)**

Los contratos inteligentes son capaces de interactuar con otros contratos, ejecutar acciones, guardar datos y transmitir tokens.

2.2.2. Ventajas y desventajas de los contratos inteligentes

Cada día los contratos inteligentes son aplicados en más áreas comerciales debido a las ventajas que ofrecen para el desarrollo de las mismas. Entre estas se pueden destacar:

- Accesibilidad: Los contratos inteligentes son cada vez más requeridos por diferentes empresas con diversos fines.
- Transparencia y seguridad en la ejecución: Dado que estos contratos se ejecutan de forma automática, las cláusulas deben establecerse de forma precisa y clara, debido a que no se permiten interpretaciones al texto de la parte clausulada, por lo que deben ser traducidas al lenguaje de programación que lo ejecutará, impidiendo que existan cláusulas ambiguas.

- Autonomía y automatización: Después de establecerse el contrato inteligente, este opera de forma automática evitando la injerencia humana, lo que a su vez evita errores y riesgos operativos derivados de ejecuciones manuales. Además, reduce los tiempos de ejecución y costos operativos ante la ausencia de intermediarios.
- Inmutabilidad e irreversibilidad: Una vez desplegado el contrato, no existe probabilidad de alteración, ya que el contrato se establece en un bloque de información, para luego enlazarse a la cadena de bloques y ser custodiado por estos. **(González, 2019)**

Por otro lado, existen ciertas características que no terminan de convencer a ciertos sectores productivos, por lo que su implementación para ellos es incierta, entre estas se encuentran:

- Escasa regulación jurídica: En la mayoría de los países aún no existe una regulación específica y clara para este tipo de contratos.
- Errores de programación: Al ser irreversibles e inmutables, cualquier error en el establecimiento de las cláusulas o en la programación en general, puede desencadenar en grandes problemas económicos y legales, ya que es prácticamente imposible detener o alterar su ejecución.
- Altos costes: Actualmente su implementación es costosa, por lo que resulta rentable en empresas con un alto volumen de contratos parecidos. **(González, 2019)**

2.2.3. Usos y aplicaciones

Existe una gran variedad de áreas comerciales donde los contratos inteligentes han desarrollado aplicaciones. De forma general, en el sector financiero se los emplean para automatizar métodos de pagos y liquidaciones. En la industria inmobiliaria ayudan en la compra y venta segura de propiedades, eliminando la necesidad de participación de intermediarios. En el área de los seguros, se los suele usar para cuando suceden determinadas clases de siniestros y se debe

ejecutar de forma automática el pago de indemnizaciones. **(Porxas & Conejero, 2018)**

De igual forma, los contratos inteligentes se adaptan a muchos tipos de tecnologías, como el caso del Internet de las Cosas (IoT) donde se los emplea para la gestión de dispositivos. Un ejemplo de esto es que facilitan que los objetos y máquinas ejecuten acciones por sus dueños, de esta manera, en el caso de un automóvil, es capaz de agendar una revisión técnica al taller de forma automática cuando el kilometraje alcance un valor previamente determinado. **(Porxas & Conejero, 2018)**

2.3. Acuerdo de Nivel de Servicio (SLA)

Un acuerdo de Nivel de servicio, cuyas siglas son SLA, es un contrato formal realizado entre un proveedor de servicios y sus clientes. Este documento establece los servicios que el proveedor debe brindar a sus clientes de forma obligatoria y que parámetros de calidad debe cumplir.

2.3.1. Importancia de los SLAs

Los proveedores de servicios establecen SLAs para administrar las expectativas que tienen los clientes sobre el servicio brindado, así mismo establecer las circunstancias en las que no será responsable de interrupciones o problemas de rendimiento. Por otro lado, es útil para el cliente para poder realizar una comparación entre las ofertas de diferentes proveedores y los procedimientos para la resolución de inconvenientes como las notas de crédito en caso de fallas o incumplimientos. **(PINTO, 2016)**

Muchos proveedores establecen un SLA maestro para indicar los términos y condiciones que se aplicaran de forma general a los clientes. A menudo detalla específicamente los servicios y las métricas de desempeño que se aplicarán para evaluarlos. **(PINTO, 2016)**

2.3.2. Contenido SLA

Un SLA, en términos generales, suele incluir una declaración de objetivos, una enumeración de los servicios cubiertos por el acuerdo, así como una definición de las responsabilidades respectivas del proveedor y del cliente en el marco del SLA. En este contexto, el cliente tiene la responsabilidad de designar a un representante que esté disponible para colaborar en la resolución de problemas con el proveedor. A su vez, el proveedor debe garantizar el cumplimiento del nivel de servicio especificado en el acuerdo. El rendimiento del proveedor se evalúa a partir de diversas métricas clave, entre las que destacan el tiempo de respuesta y el tiempo de resolución, ya que estas métricas son fundamentales para medir la efectividad en la gestión de interrupciones en el servicio. **(PINTO, 2016)**

2.3.3. Métricas de rendimiento

Los SLA establecen las expectativas del cliente respecto al rendimiento y la calidad que debe brindar el proveedor del servicio mediante diversas métricas. Entre estas se incluyen la disponibilidad y el porcentaje de tiempo activo, o el tiempo durante el cual los servicios están operativos y accesibles para el cliente, comúnmente medidos y reportados de manera mensual. Asimismo, incluyen puntos de referencia de rendimiento específicos que permiten comparar periódicamente el rendimiento real con los estándares establecidos. Además, se contempla el tiempo de respuesta, que es el periodo que el proveedor tarda en atender una solicitud o problema del cliente; para esto, los proveedores de mayor tamaño suelen disponer de un servicio de atención al cliente encargado de gestionar dichas consultas. También se considera el tiempo de resolución, que representa el periodo necesario para resolver un problema una vez que ha sido registrado por el proveedor del servicio. Adicionalmente, se establece un cronograma de notificación previa para los cambios en la red que puedan afectar a los usuarios, así como las estadísticas de uso que se proporcionarán al cliente.

Este tipo de SLA puede detallar también parámetros de disponibilidad, rendimiento y otros factores relacionados con diversos componentes de la

infraestructura del cliente, tales como redes internas, servidores y componentes críticos, como fuentes de alimentación ininterrumpidas. **(PINTO, 2016)**

2.3.4. Repercusiones de romper el SLA

Un SLA, además de establecer métricas de rendimiento, puede contener un plan para gestionar el tiempo de inactividad y especificar cómo el proveedor compensará a los clientes en caso de no cumplir con los términos acordados. Los créditos de servicio suelen ser una medida de compensación común, en la cual el proveedor concede al cliente créditos basados en un cálculo determinado por el SLA. Por ejemplo, el proveedor puede otorgar créditos proporcionales al tiempo en que el rendimiento real no haya alcanzado el nivel garantizado. También es común que los proveedores establezcan un límite máximo en dólares para las sanciones, a fin de controlar su exposición financiera. Adicionalmente, el SLA incluirá una sección de exclusiones, donde se definen las circunstancias en las que las garantías y penalizaciones no aplicarán, tales como situaciones de desastres naturales o actos de terrorismo. Esta sección, frecuentemente denominada cláusula de fuerza mayor, tiene como objetivo eximir al proveedor de responsabilidad en situaciones fuera de su control. **(PINTO, 2016)**

2.4. Sistemas de monitoreo

Los sistemas convencionales de monitoreo de redes proveen de control y visibilidad sobre la infraestructura de servicios, sin embargo, en su gran mayoría requieren considerablemente de herramientas centralizadas y de intervenciones manuales, lo que a su vez no ofrece la seguridad en el manejo de datos como sucede con Blockchain (Rojas, 2019). Por estas razones, se realiza un análisis de sistemas complejos de monitoreo de clientes más populares como:

2.4.1. Zabbix

Zabbix es una solución de monitoreo de red altamente integrada que ofrece múltiples funciones en un solo sistema. Entre sus funciones se incluye la recopilación de datos a través de pruebas de ping y desempeño, soporte para

protocolo SNMP (trapping y polling), IPMI, JMX y monitoreo de VMware, además de permitir modificar los parámetros de monitoreo de forma personalizada. Las alertas son personalizables en cuanto a: escalado, destinatarios y tipos de medio de comunicación, además pueden incluir comandos remotos. Las capacidades gráficas de Zabbix permiten visualizar tráfico y monitoreos en tiempo real, el monitoreo web simula clics en sitios web para comprobar funcionalidad y tiempos de respuesta.

Zabbix presenta varias opciones de visualización, gráfica que combinan múltiples monitoreos en uno solo, mapas de red, pantallas personalizadas y diapositivas para una visión general de escritorio. Los datos históricos se almacenan en una base de datos con tiempo de almacenamiento configurable. La configuración no requiere muchos conocimientos y permite agregar múltiples hosts, los cuales son monitoreados una vez dados de alta en la base de datos, y la aplicación de templates a los dispositivos monitoreados. Los templates permiten agrupar chequeos y pueden heredar de otros templates, facilitando la configuración y escalabilidad del sistema.

La interfaz web de Zabbix, desarrollada en PHP, es accesible desde cualquier lugar y ofrece un log de auditoría de eventos. La API de Zabbix proporciona una interfaz programable para la manipulación de sus elementos y la integración con herramientas de terceros. El sistema de permisos asegura la autenticación de usuarios y permite limitar el acceso a determinadas vistas. Además, Zabbix está listo para monitorear ambientes complejos mediante proxies que permite un monitoreo remoto sencillo y eficiente. (3 Características de Zabbix, 2024)

2.4.2. Comparativa de sistemas de monitoreo

En la Tabla 2.1 se muestra una comparación de los aspectos más importantes entre Zabbix y otros sistemas de monitoreo como The Dude **(Mikrotik, s.f.)**, Ping Plotter **(wargaming, 2020)**, Cacti **(Genos, 2024)** y Nagios **(ionos, 2023)**. En esta comparativa se destacan aspectos como funcionalidades, ventajas y desventajas, entre otras, facilitando una mejor visión de las bondades de cada sistema.

Tabla 2.1 Comparativa de sistemas de monitoreo

Característica	**Zabbix** (3 Características de Zabbix, 2024)	**The Dude** (Mikrotik, s.f.)	**Ping Plotter** (wargaming, 2020)	**Cacti** (Genos, 2024)	**Nagios** (ionos, 2023)
Función principal	Monitoreo integral de redes, servicios y sistemas con personalización avanzada.	Gestión de redes con detección de dispositivos, mapeo y monitoreo básico de servicios.	Diagnóstico de red para identificar problemas de conectividad y velocidad.	Monitorización y análisis gráfico en tiempo real para redes y sistemas.	Supervisión de redes y servicios con sistema modular y código abierto.
Protocolos soportados	SNMP, IPMI, JMX, VMware, ICMP, DNS, TCP.	SNMP, ICMP, DNS, TCP.	Ping, traceroute, whois.	RRDtool para sondeos y gráficos.	Depende de plugins: SNMP, ICMP, TCP, entre otros.
Interfaz y visualización	Gráficos avanzados, mapas de red, pantallas personalizables, diapositivas y monitoreo en tiempo real.	Mapas de red automáticos y personalizables, gráficos básicos para enlaces.	Gráficos detallados sobre la ruta de los datos y los saltos de red.	Gráficos avanzados basados en RRDtool, con plantillas personalizables.	Interfaz web básica, plugins para visualización avanzada.
Facilidad de configuración	Fácil configuración con templates heredables y escalables.	Fácil instalación y configuración básica.	Configuración sencilla enfocada en diagnósticos.	Requiere conocimientos de RRDtool para configuración avanzada.	Configuración laboriosa, especialmente para usuarios con poca experiencia.
Alertas y notificaciones	Personalizables en escalado, destinatarios y tipos de medios. Incluyen comandos remotos.	Alertas básicas para dispositivos y enlaces.	Sin sistema de alertas avanzado; más orientado al análisis gráfico.	Notificaciones básicas para eventos y datos recopilados.	Configurable con contactos para notificaciones según incidentes.

Capacidades gráficas	Amplias opciones de gráficos y personalización, monitoreo web y datos históricos configurables.	Gráficas individuales para enlaces de red.	Visualización de problemas a través de gráficas de conectividad.	Gráficos avanzados y plantillas personalizables para datos detallados.	Requiere plugins para capacidades gráficas avanzadas.
Implementación	Requiere instalación en servidor; soporta proxies para monitoreo remoto eficiente.	Aplicación cliente-servidor; soporta servidor remoto y cliente local.	Software local; orientado a análisis puntual.	Requiere instalación en servidor y conocimientos para configuración.	Instalación en servidor externo, compatible con sistemas UNIX y Windows (vía VM).
Ventajas destacadas	Escalabilidad, integración con herramientas externas mediante API, autenticación de usuarios, versatilidad.	Facilidad de uso, detección automática de dispositivos, mapeo rápido.	Ideal para identificar problemas de conectividad y latencia de red.	Excelente para gráficos y reportes detallados; adecuada para redes de cualquier tamaño.	Amplia adaptabilidad gracias a su modularidad y más de 1000 plugins disponibles.
Limitaciones	Requiere recursos y conocimientos básicos para optimizar su configuración.	Funciones limitadas comparado con herramientas más avanzadas.	Enfocado solo en diagnóstico; no incluye monitoreo continuo ni alertas robustas.	Curva de aprendizaje en configuración avanzada.	Configuración compleja, falta de documentación en español.

CAPÍTULO 3

3. METODOLOGÍA Y DISEÑO DEL SISTEMA

El Capítulo 3 está enfocado en desarrollar la solución planteada para el problema descrito en la sección 1.4 de este documento. En este apartado se describe el proceso de desarrollo e implementación del sistema de monitoreo basado en contratos inteligentes, de forma que cada mecanismo funcione de la manera más adecuada y se integren en un solo sistema, satisfaciendo y cumpliendo todos los requerimientos establecidos previamente.

Con el objetivo de mejorar la comprensión y el desarrollo del proceso, el capítulo se divide en 3 fases principales. La primera fase comprende la implementación y configuración del sistema de monitoreo, de forma que se garantice la supervisión de los parámetros del servicio. La segunda fase consiste en el desarrollo del contrato inteligente, lo que agilizará la gestión automática de los Acuerdos de Nivel de Servicio (SLA). Por último, la tercera fase tiene la finalidad de integrar ambos sistemas de forma que interactúen entre sí de manera correcta y se asegura el cumplimiento las necesidades de la solución planteada. Por último, una fase adicional para el desarrollo de una interfaz gráfica para la presentación de las métricas consideradas.

3.1. Propuesta de solución implementada

Para el desarrollo de la solución del proyecto, primero se recreó el entorno a simular, de forma que sea lo más cercano posible a un entorno real. Para lograr esto se simuló una red mediante el simulador gráfico GNS3 con el objetivo de emular la operación de un cliente final y su conexión a Internet.

Una vez realizado el entorno, mediante el sistema de monitoreo Zabbix, se evalúa la operatividad del dispositivo del cliente, emitiendo alertas en casos de pérdidas de disponibilidad. Así, es posible evaluar la capacidad del sistema para satisfacer los requisitos establecidos previamente.

En el diagrama de bloques del proyecto, mostrado en la Figura 3.1, muestra cada una de las etapas, iniciando con un quipo CPE simulado, a través del protocolo

SNMP mediante Zabbix. Estos datos y alertas recopiladas son procesados por medio de scripts en Python, para interactuar con la red blockchain. Una vez en este punto, la billetera virtual permite ejecutar las transacciones y comunicarse con la red Blockchain, la cual es simulada por Ganache.

Esta conexión permite la comunicación con el contrato inteligente, el cual se despliega con la ayuda de Truffle y son programados en Solidity. Este contrato está diseñado de forma que gestione de manera automática las acciones estipuladas en el SLA como penalizaciones y compensaciones en caso de pérdidas de disponibilidad. Esta arquitectura garantiza seguridad y transparencia en el monitoreo del servicio.

Adicionalmente, luego de estas fases se desarrolló un interfaz para que el usuario visualice las métricas como el porcentaje de disponibilidad y las compensaciones aplicadas de ser el caso. Para esto, a través de Node-RED se utilizan los scripts de Python para el procesamiento de datos, antes mencionados, para así enviar los datos a InfluxDB y tener la base de datos de series temporales, para que, a través de esta, Grafana pueda presentar la información, tal como se aprecia en el diagrama de la Figura 3.1.

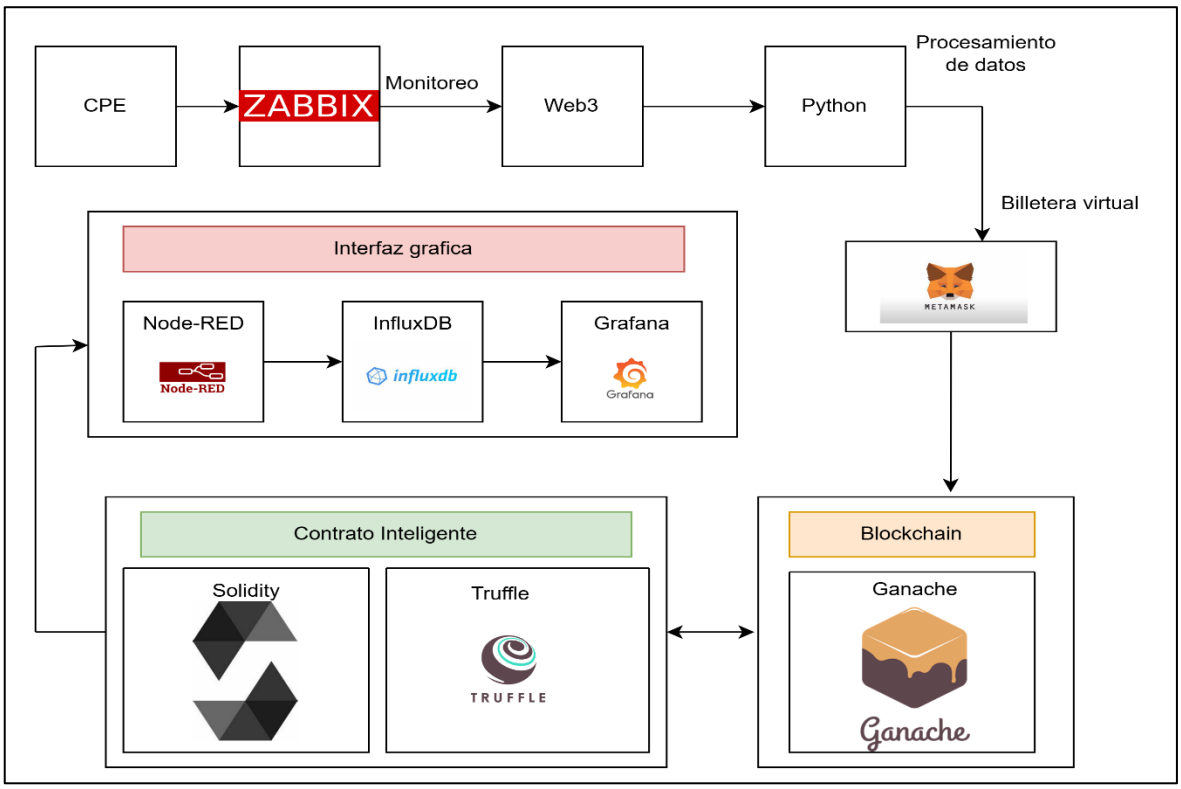


Figura 3.1 Diagrama de bloques de la solución

3.2. Componentes

La lista que se muestra a continuación cuenta con los componentes de software empleados en la solución, los que también se mencionan en la Figura 3.1. De esta forma se tiene lo siguiente:

- CPE (Customer Premises Equiment): CPE, siglas que por su traducción al español significan equipo en las instalaciones del cliente, hace referencia a los dispositivos instalados en la ubicación del cliente mediante el cual se conecta a los servicios de red o Internet. En este caso se utiliza un router que es simulado mediante el software GNS3.
- GNS3: Es un software empleado para diseñar, simular y probar redes de forma virtual.
- Zabbix: Plataforma de monitoreo y supervisión de red de código abierto.
- Web3: Es una librería con un conjunto de herramientas que permiten interactuar con blockchains, como las relacionadas con Ethereum, y contratos inteligentes.
- Python: Lenguaje de programación que sirve de “traductor” entre la plataforma de monitoreo y la red blockchain.
- Metamask: Billetera virtual, la cual es una extensión del navegador y permite interactuar de forma ágil al usuario con redes blockchain.
- Ganache: Suministra el entorno simulado de una red local Ethereum.
- Truffle: Es una herramienta que facilita el desarrollo y pruebas de contratos inteligentes.
- Solidity: Lenguaje de programación orientado para el desarrollo de contratos inteligentes.
- Node-RED: Herramienta de programación gráfica, se basa en flujos lo que facilita la automatización e integración de sistemas.
- InfluxDB: Se trata de una base de datos organizadas en función del tiempo, lo que la hace ideal para sistemas de monitoreo.

- Grafana: Plataforma que sirve para la presentación y visualización de datos e información en tiempo real.

3.3. Desarrollo de la propuesta

Como se lo mencionó en la introducción del Capítulo 3, el desarrollo del proyecto se lo dividió en 3 fases principales y la interfaz gráfica, las cuales se presentan a continuación:

3.3.1. Fase 1: Implementación del sistema de monitoreo

Como primera fase, se realizó la simulación de una red empleando el software de simulación gráfico, GNS3. Esto se realizó con el propósito de recrear el funcionamiento de un router de un cliente final y su conexión a Internet proporcionado por un ISP.

Con la simulación se diseñó un entorno de red realista mediante el cual es posible evaluar las interacciones y el comportamiento del sistema de monitoreo, lo que a su vez permite satisfacer los requisitos de vigilancia y supervisión establecidos en la propuesta. Adicionalmente, con el uso de dispositivos claves como routers, switches y dispositivos terminales, se pueden validar los procesos simulados e identificar posibles mejoras en la implementación.

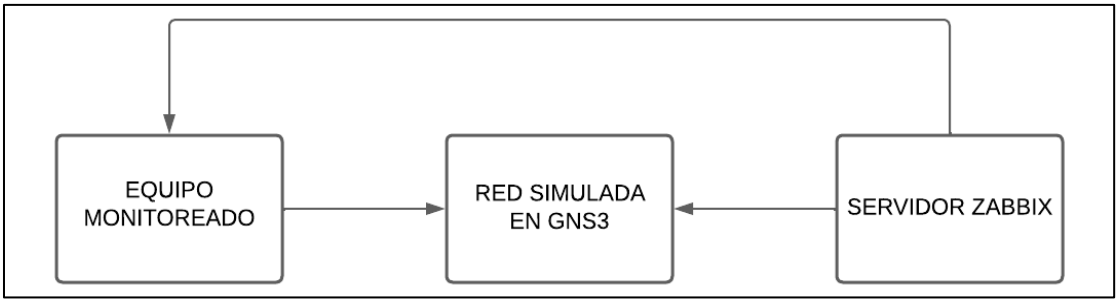


Figura 3.2 Fase 1

En la Figura 3.2 se ilustra el diagrama de bloques de la fase 1, en este diagrama se ejemplifica la conexión entre el equipo CPE simulado y el servidor de Zabbix. Con este esquema se muestra como el equipo del cliente envía datos de estado y rendimiento mediante el protocolo SNMP con destino al servidor Zabbix. Este

funciona como central de monitoreo, almacenando y analizando los datos en tiempo real, lo que a su vez más adelante permitirá detectar cualquier desviación de los parámetros establecidos en el SLA.

- **Red simulada**

La Figura 3.3 presente a continuación muestra el entorno de red simulado en GNS3, como se lo ha mencionado antes. En este esquema se estableció una estructura de conexiones que representa cómo los diferentes dispositivos de la red están organizados e interconectados.

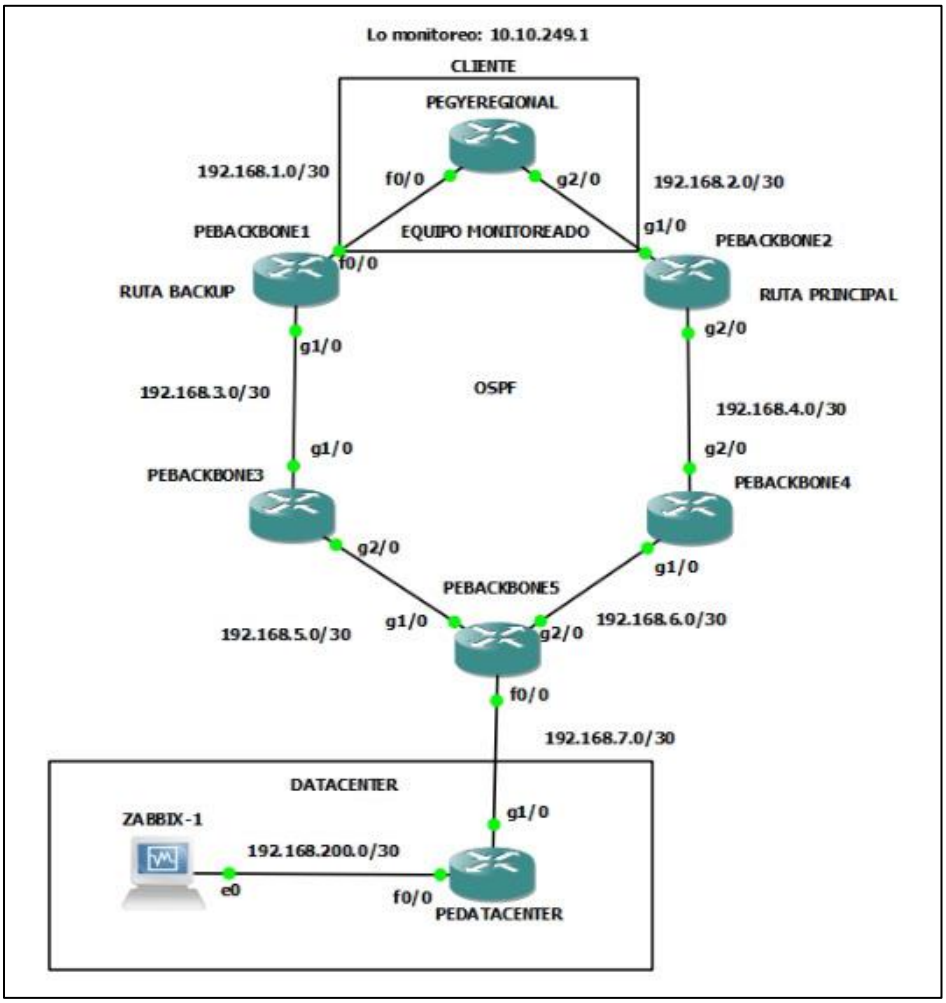


Figura 3.3 Red simulada

Este entorno simulado, detalla el proceso a través del cual la red se comunica y transmite datos al servidor de Zabbix. Esta configuración demuestra la comunicación e interacción entre los elementos de la red y el sistema de monitoreo, como en este caso en específico, el sistema de monitoreo Zabbix,

analiza el funcionamiento del router “PEGYEREGIONAL”, asegurando la gestión proactiva y la detección prematura de posibles problemas en la red.

En la topología de la red mostrada en la Figura 3.3, se destaca la presencia del protocolo Open Shortest Path First (OSPF) que calcula la ruta más corta entre dos nodos y sirve para interconectar redes de un proveedor de servicio. Así en este caso se logra el enrutamiento desde el dispositivo del cliente “PEGYEREGIONAL” hasta Zabbix para lograr realizar el monitoreo.

- **Vista global de Zabbix**

Una vez simulada la red, se procede con la configuración del sistema de Monitoreo. En la Figura 3.4 se ilustra el servidor de Zabbix, el cual es el receptor de las alarmas levantadas por el monitoreo del dispositivo, garantizando una vigilancia permanente y precisa del estado del equipo.

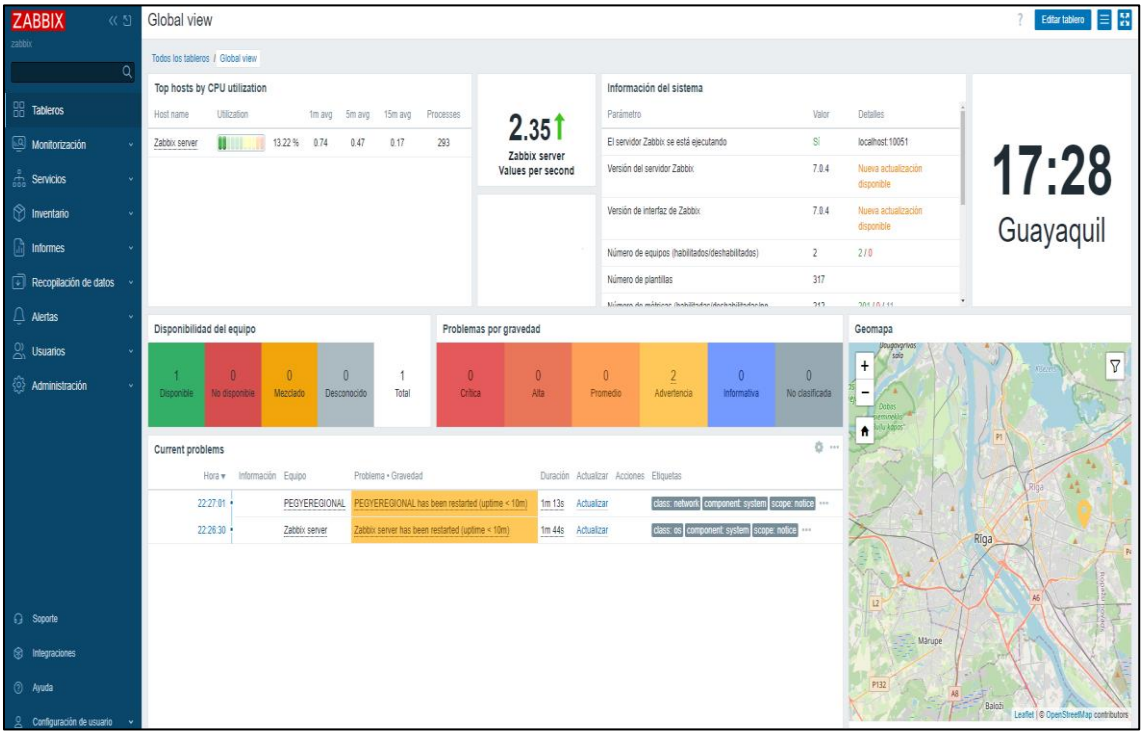


Figura 3.4 Vista global de Zabbix

El servidor de Zabbix analiza y suministra la información relevante referente al rendimiento y fallos posibles, generando alertas cuando se identifiquen situaciones críticas o anomalías. Con esta capacidad de monitoreo integral, se

logra una administración eficaz de la red y respuesta adecuado ante cualquier falla de operatividad que pueda surgir en el dispositivo.

- **Vista monitoreo equipo**

En la Figura 3.5 se evidencia una visión más completa del funcionamiento y disponibilidad del equipo de trabajo “PEGYEREGIONAL” a lo largo del tiempo. Esto a su vez, facilita un análisis exhaustivo de rendimiento de la red y del equipo.

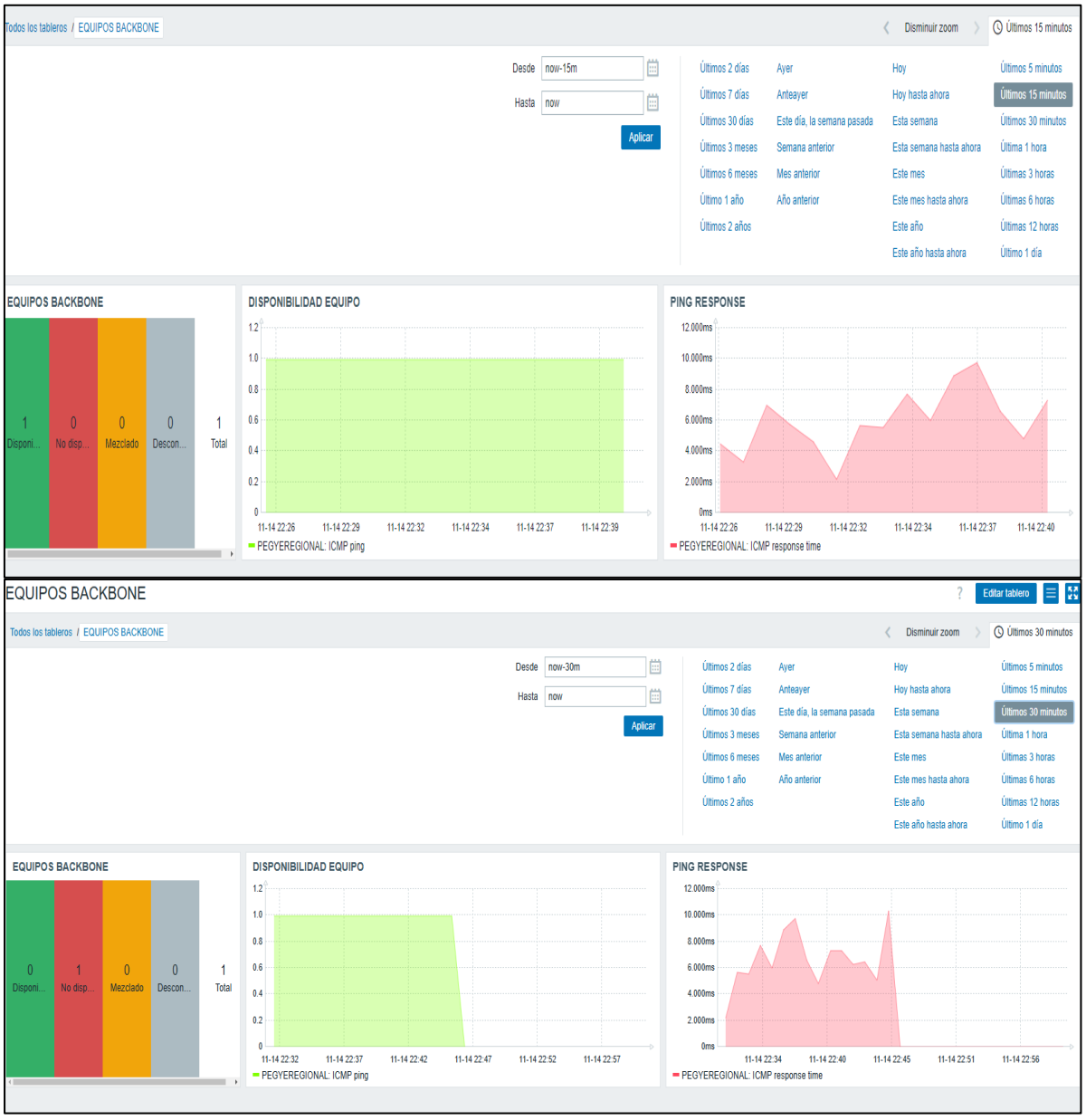


Figura 3.5 Dashboard de Zabbix

En esta Figura 3.5, se encuentran las gráficas que muestran la disponibilidad del equipo monitoreado. Del mismo modo muestra la respuesta ping, facilitando la

evaluación de la calidad de conexión y la detección de interrupciones en el servicio.

Estas gráficas proveen una representación visual clara de la continuidad operativa, facilitando la identificación de patrones y tendencias que podrían indicar problemas recurrentes o áreas que requieren atención para optimizar la estabilidad y eficiencia de la red.

- **Configuración de alertas**

La Figura 3.6 muestra la configuración detallada de las alertas en Zabbix. En estas se definen los parámetros específicos para la notificación de incidentes, en este caso la disponibilidad. Esta configuración abarca criterios como el tipo de problema, el nivel y comando identificado.

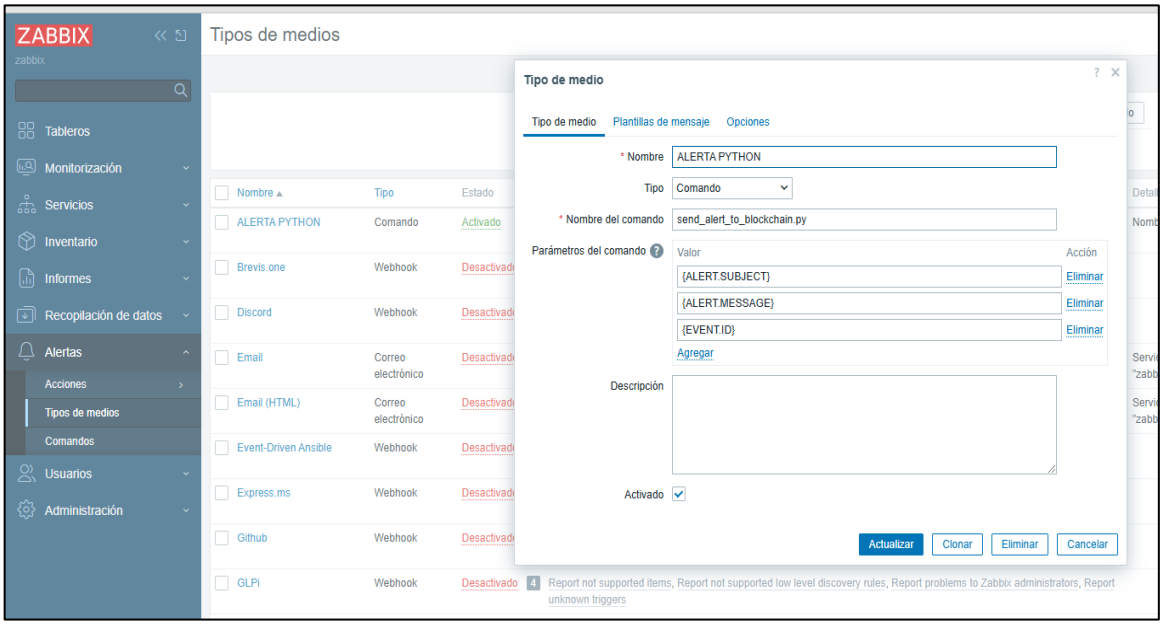


Figura 3.6 Alerta configurada

Por otro lado, la Figura 3.7, permite visualizar de forma general el sistema de monitoreo, proporcionando un resumen consolidado de los elementos vigilados y su estado actual. En esta vista general, se destacan métricas importantes como el historial de eventos recientes, disponibilidad y rendimiento. Con esta información, los administradores pueden ser capaces de detectar problemas potenciales y tendencias en los comportamientos.

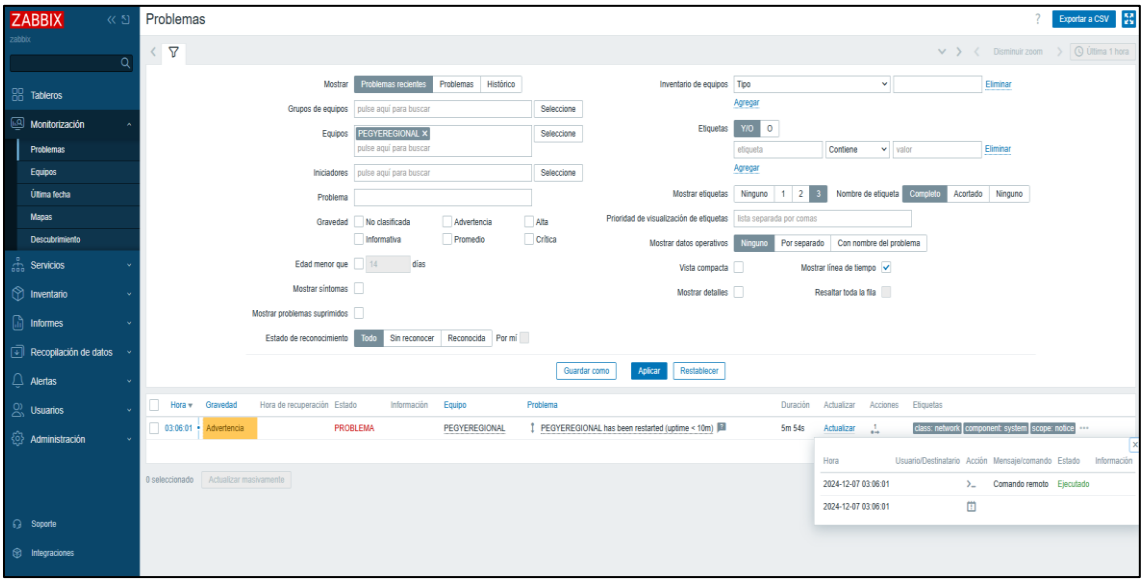


Figura 3.7 Configuración de alarmas

En la Figura 3.8 se visualizan los comandos empleados para el almacenamiento de las alertas en el sistema Zabbix., enfatizando la manera en que se configuran y gestionan los eventos suscitados.

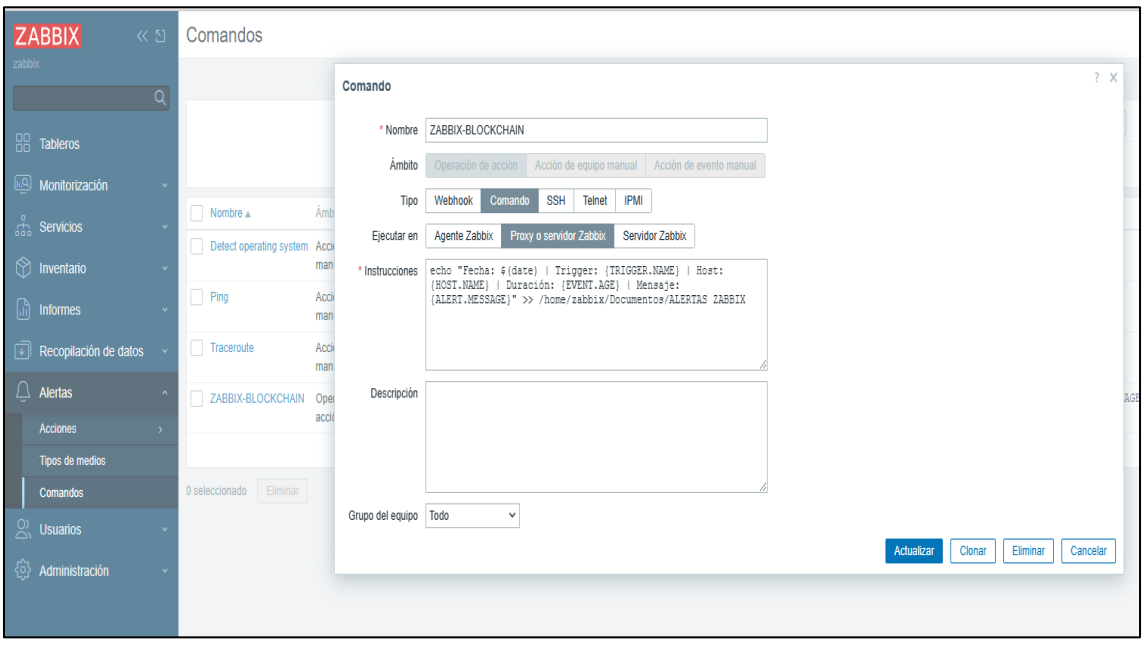


Figura 3.8 Comando de almacenamiento de alertas

Los comandos presentados en la Figura 3.8, son capaces de guardar información crítica referentes a las alertas, como el origen del incidente. Estos comandos permiten almacenar información crítica sobre las alertas, como su severidad, el

origen del incidente. Estos ayudan en la personalización del almacenamiento, ara conservar un historial organizado y accesible.

- **Registro de alertas**

La Figura 3.9 permite visualizar las alarmas registradas en el sistema Zabbix, se detalla en ellas, los eventos críticos identificados en el monitoreo. Con cada alerta se obtiene una categoría con información clave, como la respuesta de ping y el estado actual de resolución.

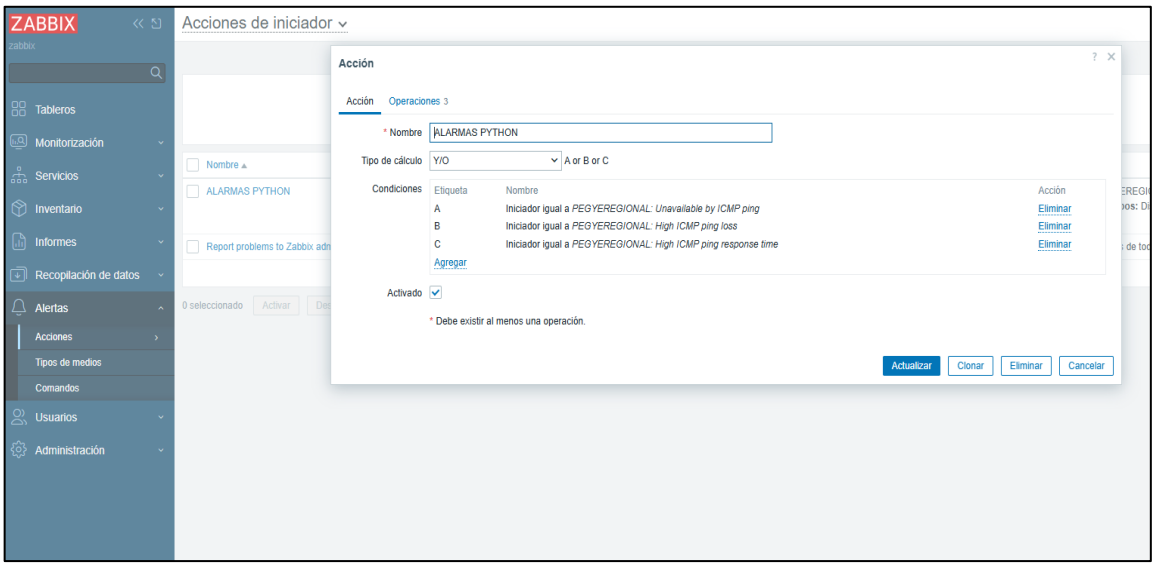


Figura 3.9 Registro de alertas

- **Reportes Log a “.txt”**

En las Figuras 3.10 y 3.11 se detalla como a través del sistema de monitoreo de Zabbix, se encuentra configurado un mecanismo para generar y reportar logs en formato de texto plano (.txt).

Figura 3.10 Reportes Log a “.txt”

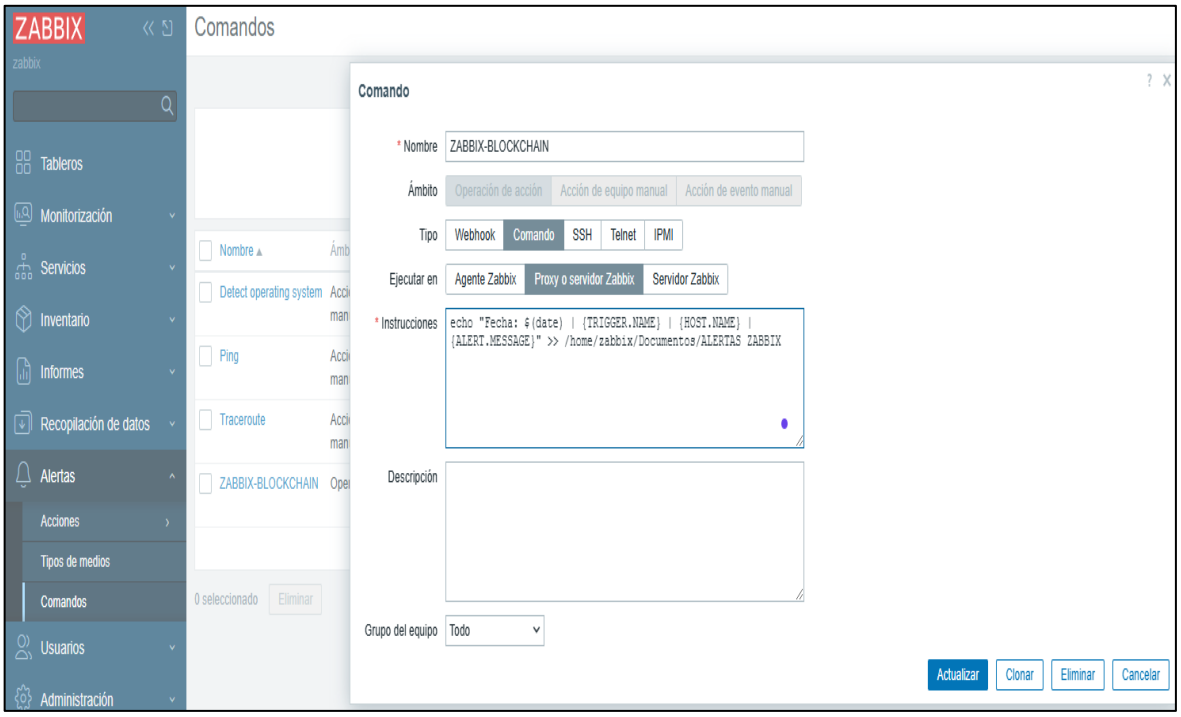


Figura 3.11 Reportes Log a ".txt" desde Zabbix

Los procesos mostrados en estas figuras permiten que las alertas y eventos monitoreados por Zabbix sean almacenados de manera estructurada en archivos, facilitando su análisis posterior. Estos archivos funcionan como un puente para integrar la información monitoreada con otros sistemas, como scripts de Python, haciendo posible una gestión automatizada de los datos y una mayor trazabilidad en el monitoreo.

- **Usuarios para envío de alertas**

En la Figura 3.12 se puede observar la configuración de usuarios en Zabbix para el envío de alertas.

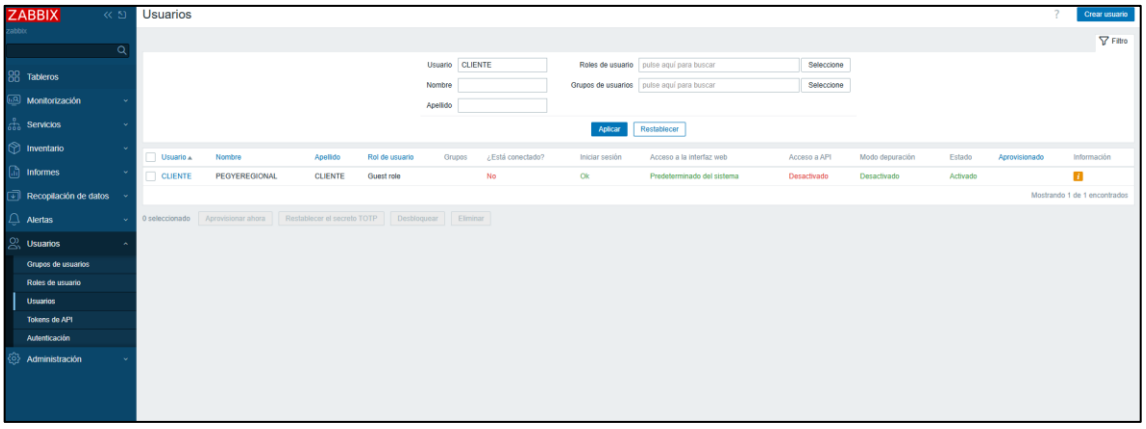


Figura 3.12 Usuarios para envío de alertas

De esta forma, se visualiza que se define el usuario “PEGYEREGIONAL” con privilegios de invitado. Este usuario se asocia con una dirección de correo electrónico, donde se le notificaran las alertas identificadas en el equipo monitoreado. Este mecanismo, facilita que las alarmas lleguen hasta el destinatario apropiado para su análisis y resolución.

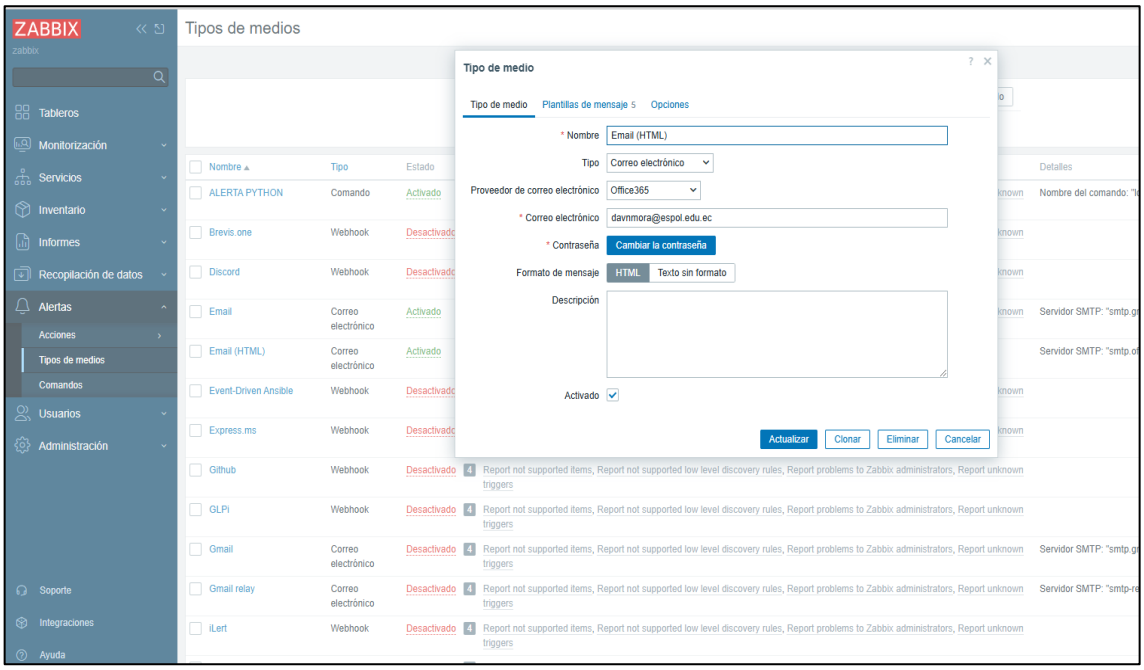


Figura 3.13 Configuración del servidor de correos para Zabbix

Por otro lado, se visualiza la Figura 3.13 donde se configura el servidor de correos que Zabbix utilizará para el envío de alertas. En esta configuración, se utiliza el servicio de Office365, usando un correo institucional de ESPOL como prueba.

Adicionalmente, se definen los parámetros correspondientes, como la dirección del servidor, el puerto, y las credenciales del correo. Esto hace posible que Zabbix envíe de forma automatizada las alertas al destinatario apropiado.

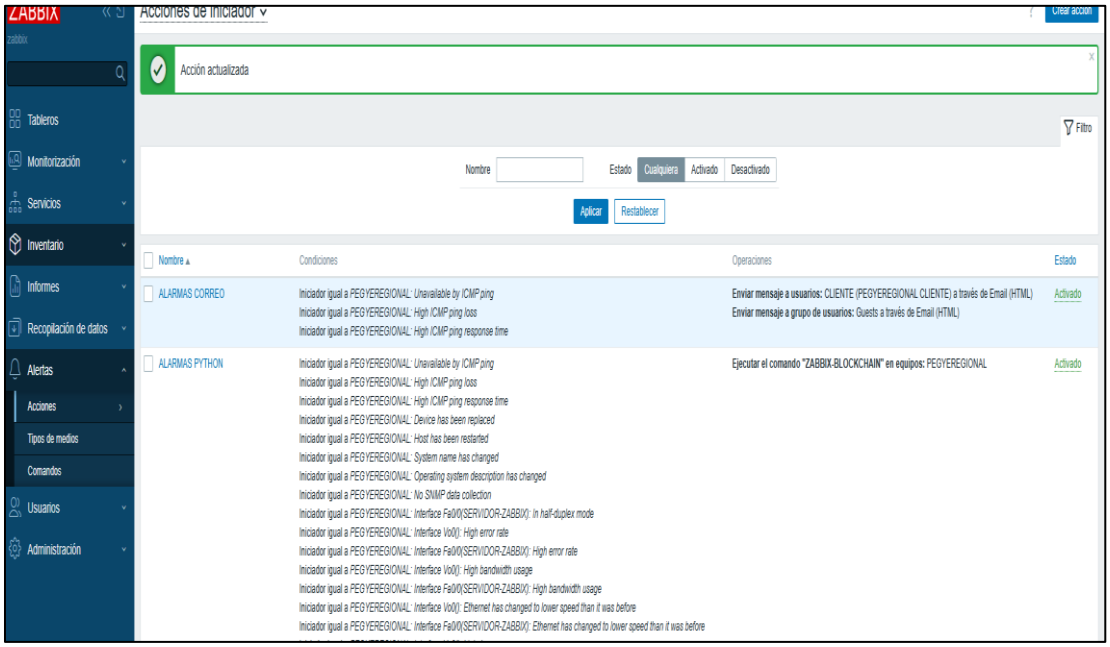


Figura 3.14 Configuración de notificación a correos

Para configurar las alarmas en Zabbix, como se muestra en la figura 3.14, se implementaron dos métodos de envío. El primero emplea un script en Python para registrar los eventos en la blockchain, procesando los datos de forma automática a partir de las alertas. El segundo método consiste en notificaciones por correo electrónico, configuradas con la regla Email (HTML), dirigidas al usuario PEGYEREGIONAL. Este enfoque asegura que las alertas se gestionen de manera efectiva.

3.3.2. Fase 2: Desarrollo del contrato inteligente

La fase 2 como su nombre lo indica, corresponde al desarrollo del contrato inteligente. Esta fase es esencial para garantizar la automatización y confiabilidad en la verificación del cumplimiento de los SLA a través de la tecnología blockchain. El contrato inteligente desempeña un papel vital dentro del sistema, dado que realiza acciones estipuladas de forma automática y periódica en caso de detectarse incumplimientos, informando sobre penalizaciones o compensaciones al cliente afectado de ser el caso.

- **Diseño del contrato inteligente**

En esta sección se muestra el desarrollo del contrato inteligente. El lenguaje de programación utilizado para la creación del contrato es Solidity, lenguaje que se caracteriza por su alto nivel de diseño en la elaboración de contratos inteligentes en la red. Solidity fundamenta en la sintaxis de lenguajes como Python y JavaScript, permitiendo establecer reglas y condiciones a cumplirse en el contrato, lo que a su vez facilita el cumplimiento automático de las cláusulas establecidas en el mismo.

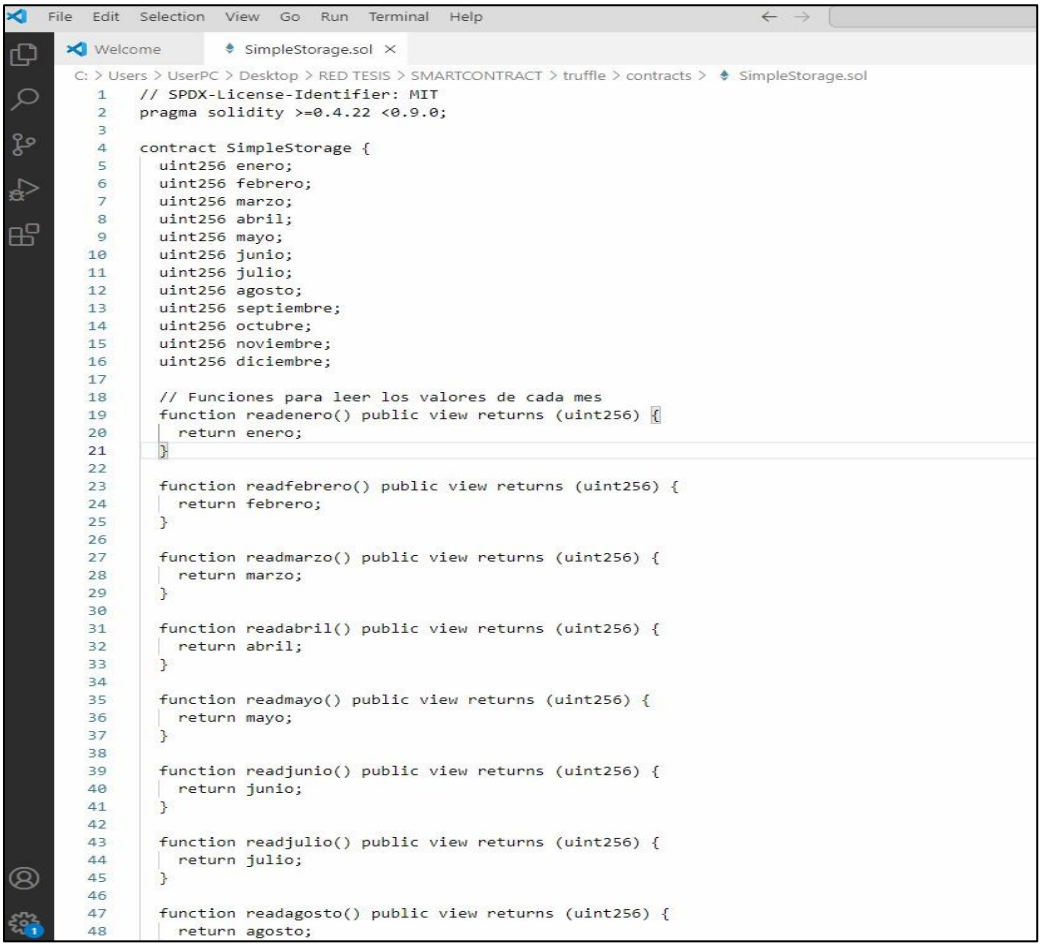
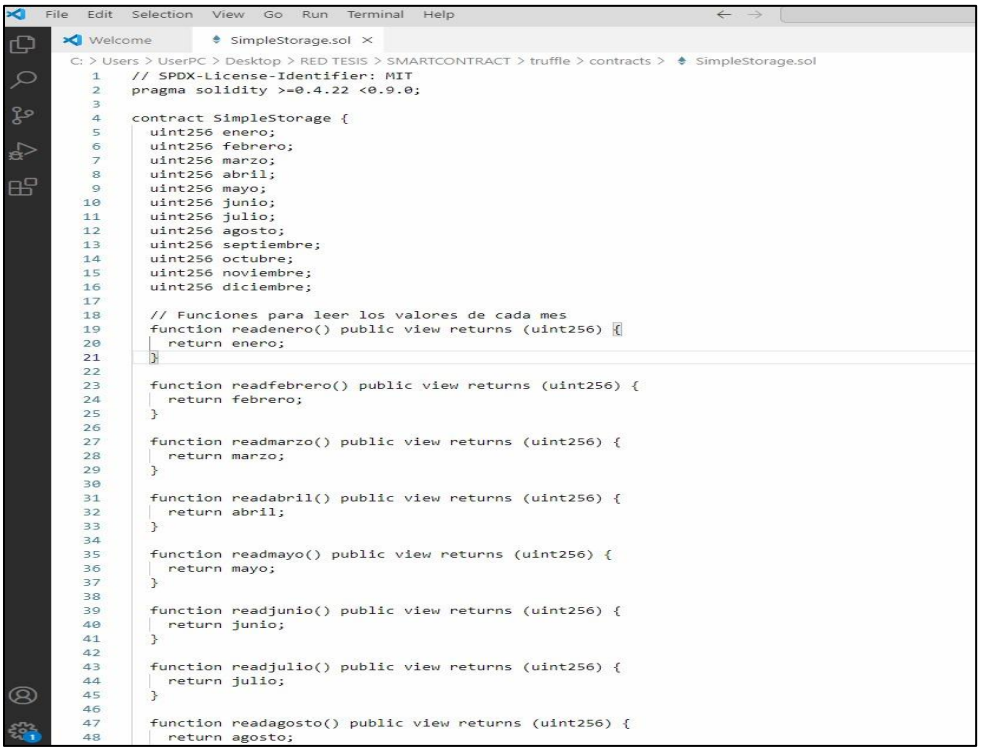


Figura 3.15 Contrato inteligente parte I

En esta primera parte del contrato inteligente, es capaz de leer cada uno de los valores de disponibilidad del servicio que serán almacenados de acuerdo con su mes en el contrato de forma recurrente al integrarse con los reportes de Zabbix. El código completo se lo puede visualizar en el apartado de anexos.



```
// SPDX-License-Identifier: MIT
pragma solidity >=0.4.22 <0.9.0;

contract SimpleStorage {
  uint256 enero;
  uint256 febrero;
  uint256 marzo;
  uint256 abril;
  uint256 mayo;
  uint256 junio;
  uint256 julio;
  uint256 agosto;
  uint256 septiembre;
  uint256 octubre;
  uint256 noviembre;
  uint256 diciembre;

  // Funciones para leer los valores de cada mes
  function readenero() public view returns (uint256) {
    return enero;
  }

  function readfebrero() public view returns (uint256) {
    return febrero;
  }

  function readmarzo() public view returns (uint256) {
    return marzo;
  }

  function readabril() public view returns (uint256) {
    return abril;
  }

  function readmayo() public view returns (uint256) {
    return mayo;
  }

  function readjunio() public view returns (uint256) {
    return junio;
  }

  function readjulio() public view returns (uint256) {
    return julio;
  }

  function readagosto() public view returns (uint256) {
    return agosto;
```

Figura 3.16 Contrato inteligente parte II

De forma similar a lo mostrado en la Figura 3.15 en este caso, se muestran las funciones con las que el contrato inteligente es capaz de almacenar los nuevos valores recibidos y organizarlos según su mes de ocurrencia.

3.3.3. Fase 3: Integración del sistema de monitoreo con el contrato inteligente

La fase 3 tiene como objetivo la integración del contrato inteligente con el sistema de monitoreo, de forma que se desarrolle una comunicación eficiente entre ambas partes que permite asegurar el cumplimiento de los SLA de forma automática. En la Figura 3.17 se puede visualizar un esquema de comunicación entre las partes.

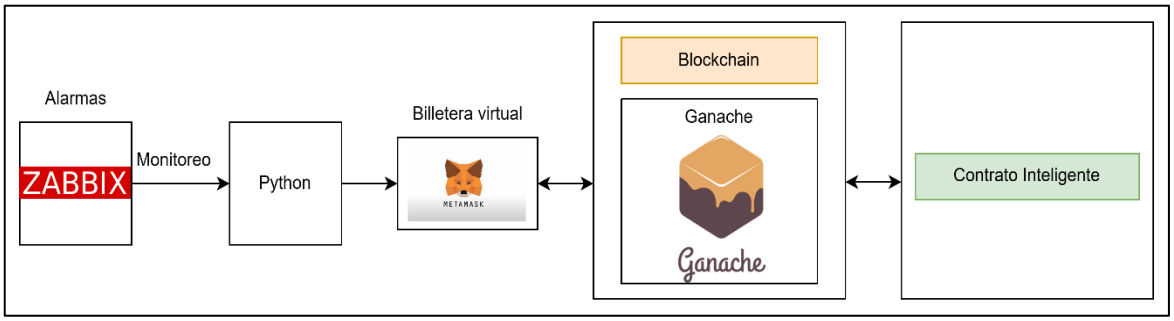


Figura 3.17 Fase 3: Comunicación entre Zabbix y la blockchain

La Figura 3.17, muestra el proceso de comunicación entre el sistema de monitoreo y la blockchain. Para esto Zabbix genera archivos de texto plano “.txt” los cuales contienen la información de los eventos ocurridos para posteriormente ser procesados en scripts en Python que funcionan como intermediarios. En estos scripts se procesan alarmas o métricas de desempeño. Estos datos son enviados son enviados a la blockchain a través de la interacción con la billetera digital Metamask y Ganache.

Una vez integrada, esta arquitectura permite es posible que los datos obtenidos mediante el sistema de monitoreo sean evaluados de forma automática según lo establecido en el contrato inteligente. En caso de existir incumplimientos en cuanto a la disponibilidad del servicio, se calculan y ejecutan las compensaciones o notificaciones de las partes implicadas mediante la ejecución automática del contrato inteligente.

- **Edición de los archivos de alerta**

Como primer paso para lograr que ambos sistemas se comuniquen entre sí, se realiza la descomposición del archivo de alertas monitoreadas proporcionado por Zabbix. Esto se lo ejecuta mediante el código escrito en Python que se muestra en la Figura 3.18 a continuación.

```python
from collections import defaultdict
import os


ruta_archivo = r"C:\Users\UserPC\Downloads\ALERTAS.txt"
ruta_txtsalida=r"C:\Users\UserPC\Downloads"

meses=["ene","feb","mar","abr","may","jun","jul","ago","sep","oct","nov","dic"]


def leer_alarmas(ruta_archivo):
    alarmas = []
    # Expresión regular para extraer los datos de cada alarma (si fuera necesario)
    try:
        with open(ruta_archivo, 'r', encoding='utf-8') as archivo:
            for linea in archivo:
                if linea.strip():
                    alarma2 = {}
                    segmentos = linea.split('|')
                    for segmento in segmentos:
                      clave, valor = segmento.split(':', 1)
                      clave = clave.strip().lower()
                      alarma2[clave] = valor.strip()
                    # Añadir la alarma si no es repetida
                    if alarma2 in alarmas:
                      print("Alarma repetida, no se enviará nuevamente.")
                    else:
                      alarmas.append(alarma2)
                else:
                    print("linea vacia")

        # Eliminar el archivo original después de leerlo
        if os.path.exists(ruta_archivo):
```

Figura 3.18 Código para división de archivos

En la Figura 3.18 se muestra la parte inicial del código que se utiliza para subdividir el archivo de alertas proporcionado por Zabbix. El archivo inicial cuenta con todas las alertas registradas, el propósito de este código es el de crear archivos de texto plano ".txt" para cada mes del año y así contar con la información ordenada y clasificada de manera más precisa para facilitar su procesamiento.

En la parte inicial del código se puede observar cómo se establece como ruta de entrada el archivo general de alarmas y de ruta de salida la misma carpeta para los nuevos archivos. Posteriormente, se procede con la manipulación de los datos, eliminado los innecesarios para quedarse solo con la información relevante, extrayendo parámetros clave como el nombre del *trigger* (alarma), el *hostname* del equipo afectado y el mensaje asociado. El código completo a detalle se lo puede observar en el anexo 2.

- **Despliegue de la blockchain**

Para continuar con el proceso, es necesario desplegar la blockchain local por medio de Ganache, instalándola en una laptop para que funcione como un nodo Ethereum y lograr simular transacciones con criptomonedas suministradas por Ganache en forma de direcciones que incluyen el número de cuenta y clave privada en formato hexadecimal.

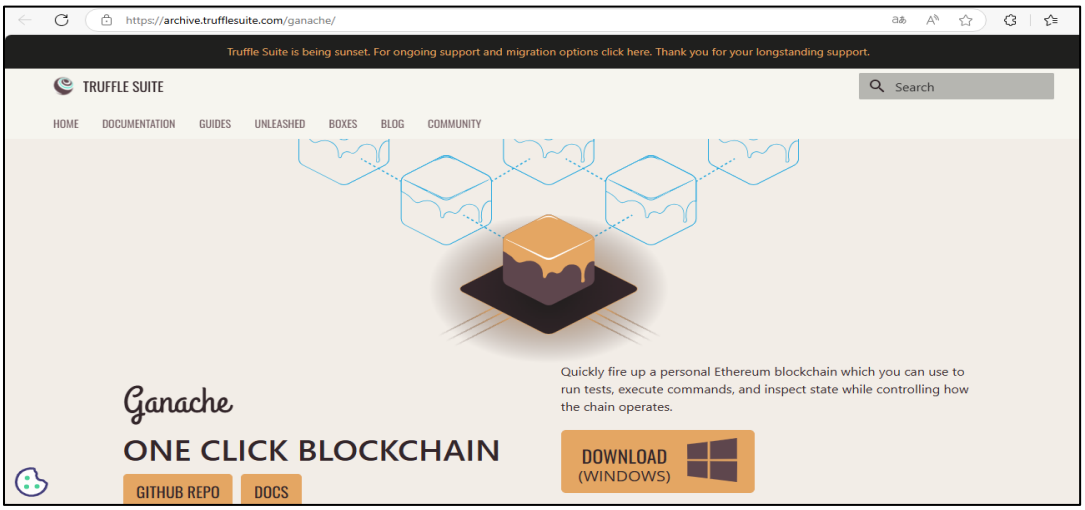


Figura 3.19 Instalación de Ganache

De esta manera se realiza la descarga e instalación de Ganache para a través del siguiente enlace: https://archive.trufflesuite.com/ganache/. Tal como se muestra en la Figura 3.19.

Luego de la instalación e inicio rápido de Ganache, se refleja en pantalla lo mostrado en la Figura 3.20. Aquí se observan las wallets disponibles asociadas sus respectivas direcciones o “address” y clave privada, junto con 100 ETH de prueba.

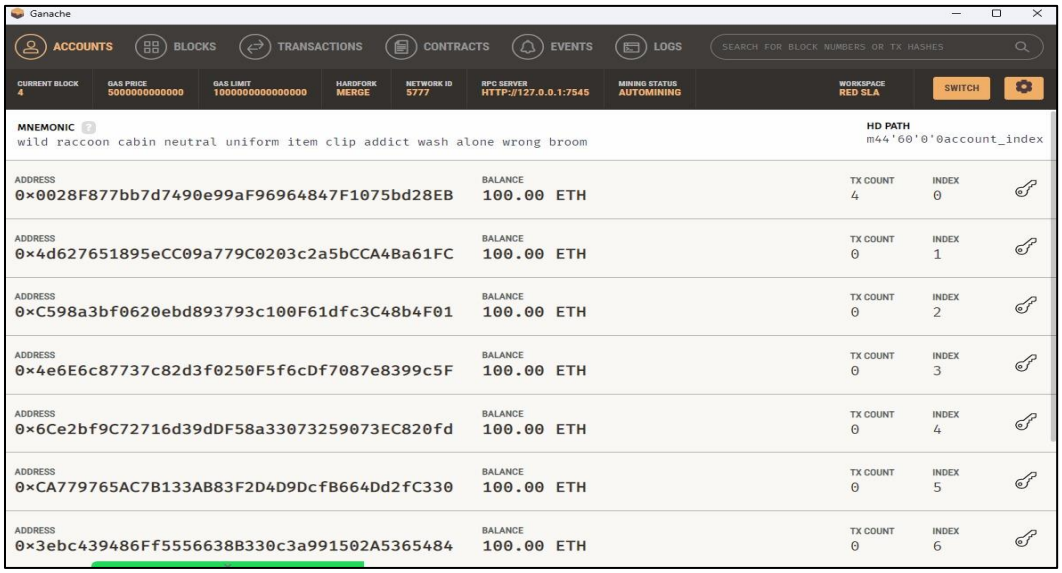


Figura 3.20 Wallets disponibles

Esto permite que usando una de estas direcciones se pueda transferir saldo a la billetera digital que se usará, en este caso Metamask. Con el uso de la billetera instalada como extensión del navegador, es posible interactuar con la blockchain de manera que a través del contrato inteligente las transacciones son enviadas y recibidas por Ganache.

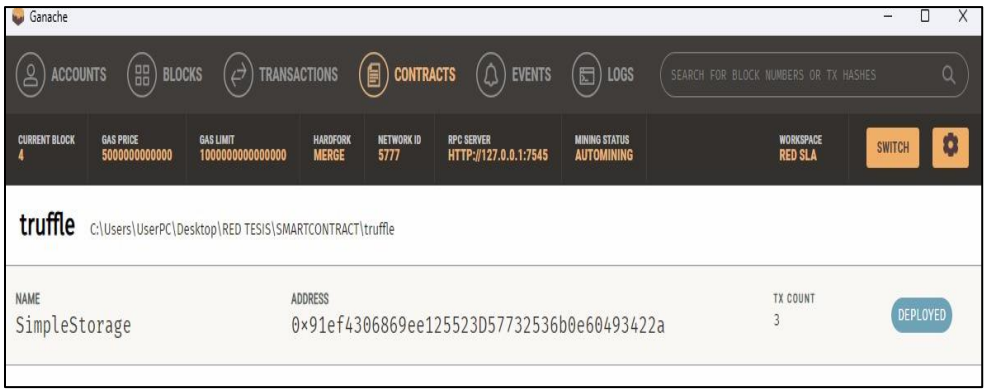


Figura 3.21 Despliegue del contrato inteligente

Una vez establecida la red blockchain se procede con el despliegue del contrato inteligente, para esto se hace uso de Truffle, que como se mencionó en el apartado 3.2, facilita el despliegue del contrato. En la Figura 3.21 se muestra como el contrato se ha desplegado exitosamente, existe comunicación y se le ha asignado la dirección correspondiente.

- **Instalación de la billetera digital**

Para instalar Metamask, en el navegador Google Chrome se descarga la extensión de esta desde el siguiente enlace: https://chromewebstore.google.com/detail/metamask/nkbihfbeogaeaoehlefnkodbefgpgknn?utm_source=ext_app_menu. Tal como se observa en la Figura 3.22.

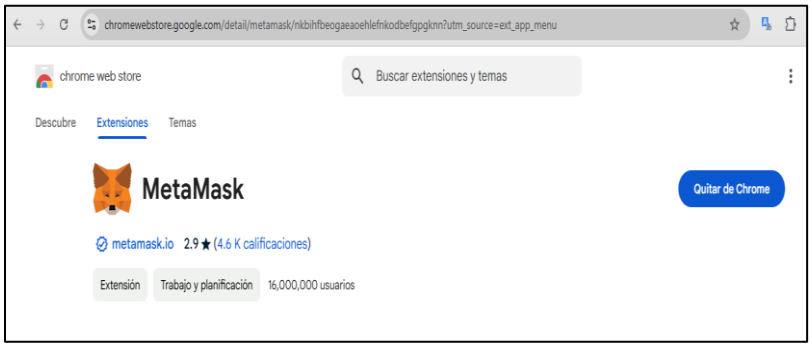


Figura 3.22 Instalación de Metamask

Una vez instala la billetera, es necesario crear una cuenta e iniciar sesión. Luego con los datos obtenidos en la sección anterior, suministrada por Ganache se procede a agregar saldo a la cuenta desde el menú del perfil y la opción de importar cuenta. Reflejando como se muestra en la Figura 3.23, que la cuenta posee una gran cantidad de ETH (moneda virtual) para poder realizar las transacciones.

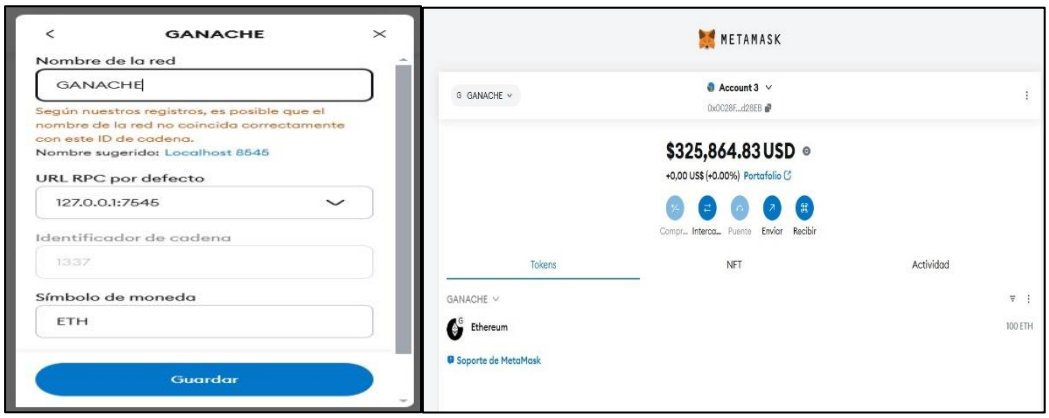
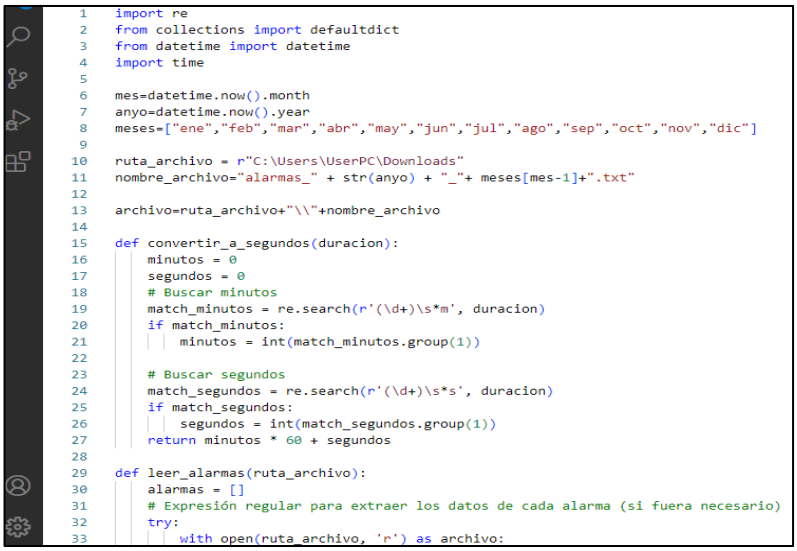


Figura 3.23 Cuenta suministrada desde Ganache

- **Lectura y envío de alarmas**

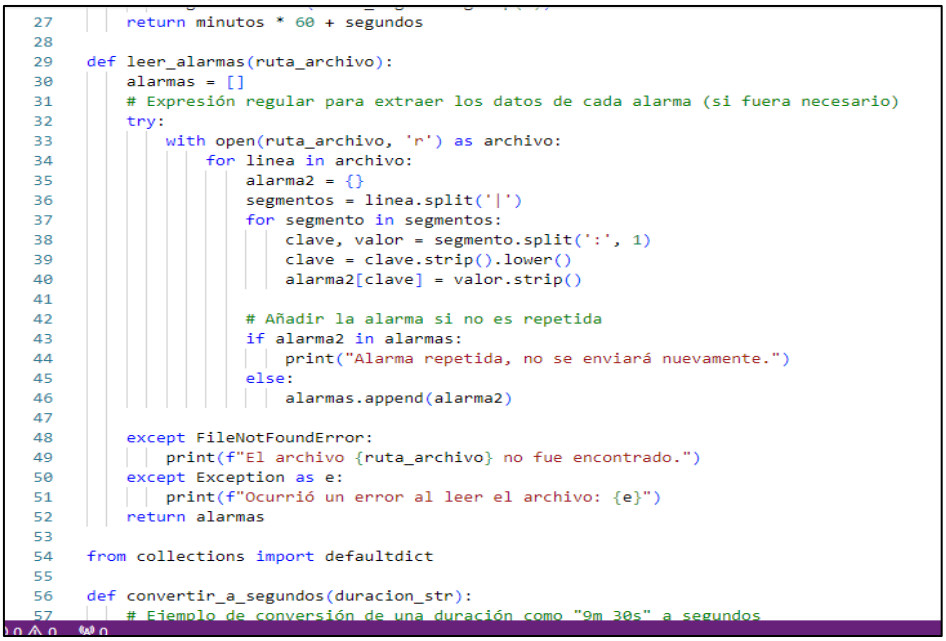
Para la lectura y envío de alertas desde el servidor de Zabbix hasta el contrato inteligente también se hace un script de Python. El cual se muestra como referencia en la Figura 3.24 y se encuentra completo a detalle en el anexo 3.



```python
import re
from collections import defaultdict
from datetime import datetime
import time

mes=datetime.now().month
anyo=datetime.now().year
meses=["ene","feb","mar","abr","may","jun","jul","ago","sep","oct","nov","dic"]

ruta_archivo = r"C:\Users\UserPC\Downloads"
nombre_archivo="alarmas_" + str(anyo) + "_"+ meses[mes-1]+".txt"

archivo=ruta_archivo+"\\"+nombre_archivo

def convertir_a_segundos(duracion):
    minutos = 0
    segundos = 0
    # Buscar minutos
    match_minutos = re.search(r'(\d+)\s*m', duracion)
    if match_minutos:
        minutos = int(match_minutos.group(1))

    # Buscar segundos
    match_segundos = re.search(r'(\d+)\s*s', duracion)
    if match_segundos:
        segundos = int(match_segundos.group(1))
    return minutos * 60 + segundos

def leer_alarmas(ruta_archivo):
    alarmas = []
    # Expresión regular para extraer los datos de cada alarma (si fuera necesario)
    try:
        with open(ruta_archivo,
```

Figura 3.24 Código de lectura y envío de alarmas

El código de la Figura 3.24 se lo desarrollo con dos objetivos, la primera etapa busca leer las alarmas registradas en el equipo monitoreo y la segunda etapa tiene como propósito interactuar con la red blockchain y enviar dichos datos al contrato inteligente.



```python
    return minutos * 60 + segundos

def leer_alarmas(ruta_archivo):
    alarmas = []
    # Expresión regular para extraer los datos de cada alarma (si fuera necesario)
    try:
        with open(ruta_archivo, 'r') as archivo:
            for linea in archivo:
                alarma2 = {}
                segmentos = linea.split('|')
                for segmento in segmentos:
                    clave, valor = segmento.split(':', 1)
                    clave = clave.strip().lower()
                    alarma2[clave] = valor.strip()

                # Añadir la alarma si no es repetida
                if alarma2 in alarmas:
                    print("Alarma repetida, no se enviará nuevamente.")
                else:
                    alarmas.append(alarma2)

    except FileNotFoundError:
        print(f"El archivo {ruta_archivo} no fue encontrado.")
    except Exception as e:
        print(f"Ocurrió un error al leer el archivo: {e}")
    return alarmas

from collections import defaultdict

def convertir_a_segundos(duracion_str):
    # Ejemplo de conversión de una duración como "9m 30s" a segundos
```

Figura 3.25 Código de lectura de alarmas

En esta primera parte se puede evidenciar como se ha programado el script para que se ejecute la lectura de los archivos de alarmas clasificados por meses. Con esto se desglosa los datos y se extrae la información más relevante como el tiempo de inactividad y la fecha.

Por otro lado, el segundo propósito del código se encuentra en la Figura 3.26 a continuación.

```
# Conecta a la blockchain local (por ejemplo, Ganache)
ganache_url = "http://127.0.0.1:7545"  # Cambia esto si usas otra red
web3 = Web3(Web3.HTTPProvider(ganache_url))

# Verifica la conexión
if not web3.is_connected():
    print("No se pudo conectar a la blockchain")
    exit()

# Dirección del contrato desplegado (actualízala según tu caso)
contract_address = "0x91ef4306869ee125523D57732536b0e60493422a"

# ABI del contrato (puedes copiarlo desde la compilación en Truffle o Remix)
contract_abi = [
  {
    "inputs": [
      {
        "internalType": "string",
        "name": "_mes",
        "type": "string"
      },
      {
        "internalType": "uint256",
        "name": "_valor1",
        "type": "uint256"
      },
      {
        "internalType": "uint256",
        "name": "_valor2",
        "type": "uint256"
```

Figura 3.26 Código de envío de alarmas

En este segundo parte del código se interactúa con el contrato inteligente gracias a los procesos previamente realizados. Se puede observar que se define la URL suministrada por Ganache para acceder a la red blockchain. Además, se establece la dirección del contrato inteligente proporcionada por Ganache. Con estos datos y gracias a Ganache, es posible que se ejecuten las transacciones en la red y que el contrato inteligente pueda recibir valores.

- **Lectura del contrato inteligente**

Para la lectura del contrato inteligente se empleó el código de la Figura 3.27. En primera instancia el código verifica si los datos ingresados para la lectura de valores son los correctos.

```python
def leer_valores_del_mes(mes):
    try:
        # Verificar que el mes sea una cadena de texto válida
        if not isinstance(mes, str):
            raise ValueError("El mes debe ser una cadena de texto."

        # Seleccionar la función de lectura según el mes
        if mes.lower() == "enero":
            value = contract.functions.readenero().call()
        elif mes.lower() == "febrero":
            value = contract.functions.readfebrero().call()
        elif mes.lower() == "marzo":
            value = contract.functions.readmarzo().call()
        elif mes.lower() == "abril":
            value = contract.functions.readabril().call()
        elif mes.lower() == "mayo":
            value = contract.functions.readmayo().call()
        elif mes.lower() == "junio":
            value = contract.functions.readjunio().call()
        elif mes.lower() == "julio":
            value = contract.functions.readjulio().call()
        elif mes.lower() == "agosto":
            value = contract.functions.readagosto().call()
        elif mes.lower() == "septiembre":
            value = contract.functions.readseptiembre().call()
        elif mes.lower() == "octubre":
            value = contract.functions.readoctubre().call()
        elif mes.lower() == "noviembre":
            value = contract.functions.readnoviembre().call()
        elif mes.lower() == "diciembre":
            value = contract.functions.readdiciembre().call()
        else:
            raise ValueError("Mes no válido.")
```

Figura 3.27 Código de lectura del contrato

Posteriormente, el código procede a la lectura de los valores almacenados por mes. En esta información se presentará el tiempo que el servicio no ha estado disponible.

- **Iteración de códigos**

Como se explicó previamente, en el servidor de Zabbix se encuentran ejecutando scripts de Python que sirven para registrar las alertas, separarlas en archivos distintos por mes de ocurrencia y para enviar a la blockchain.

```
zabbix@zabbix-VirtualBox:~$ crontab -l

59 23 28-31 * *  ["$(date +\%d -d tomorrow)" == "01" ] && /usr/bin/python3.12 /h
ome/zabbix/Documentos/SEPARAR ALARMAS MENSUALES.py
0 0 1 * * /usr/bin/python3.12 /home/zabbix/Documentos/ENVIAR ALERTAS A BLOCKCHAI
N.py
```

Figura 3.28 Programación para iteración de códigos

Es por esto, que en la Figura 3.28, se muestra la programación en el servidor mencionado, donde se establece que el programa de separación de alarmas por meses se ejecute el último día del mes, para que de esta manera todos los meses exista un archivo con alarmas exclusivas del mes correspondiente. De forma similar, el programa de envío de alertas se ejecuta al inicio de cada mes para que la información de las alarmas registradas el mes anterior sean cargadas a la blockchain para el procedimiento correspondiente.

3.3.4. Interfaz gráfica

Una vez logrado el funcionamiento del sistema como tal, es necesario presentar valores de disponibilidad y compensaciones, a través de un entorno amigable para los usuarios. Para esto se realiza una interfaz gráfica de manera que los datos sean visualizados de forma sencilla y entendibles.

- **Desarrollo de la interfaz**

Para desarrollar el entorno visual para el usuario lo primero que se utilizó fue Node-RED que es una herramienta de programación gráfica basada en flujos, lo que permite una fácil programación e integración con otros servicios sin necesidad de instalar sistemas más complejos. La información para su instalación se la puede encontrar en su página web oficial (https://nodered.org/docs/) como se muestra en la Figura 3.29.

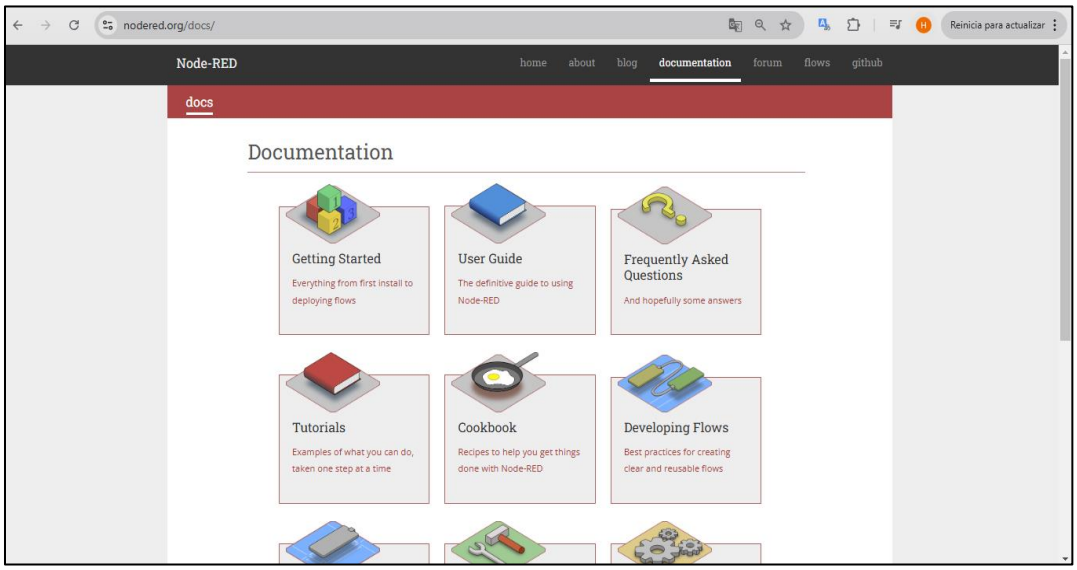


Figura 3.29 Instalación de Node-RED

- **Flujo desarrollado**

El objetivo de la interfaz gráfica es que el usuario pueda acceder a la información de la disponibilidad de su servicio contratado, así como a las posibles compensaciones en su factura mensual debido a pérdidas del servicio durante el mes.

Para lograr este propósito se desarrolló la topología o flujo en Node-RED que se muestra a continuación en la Figura 3.30.

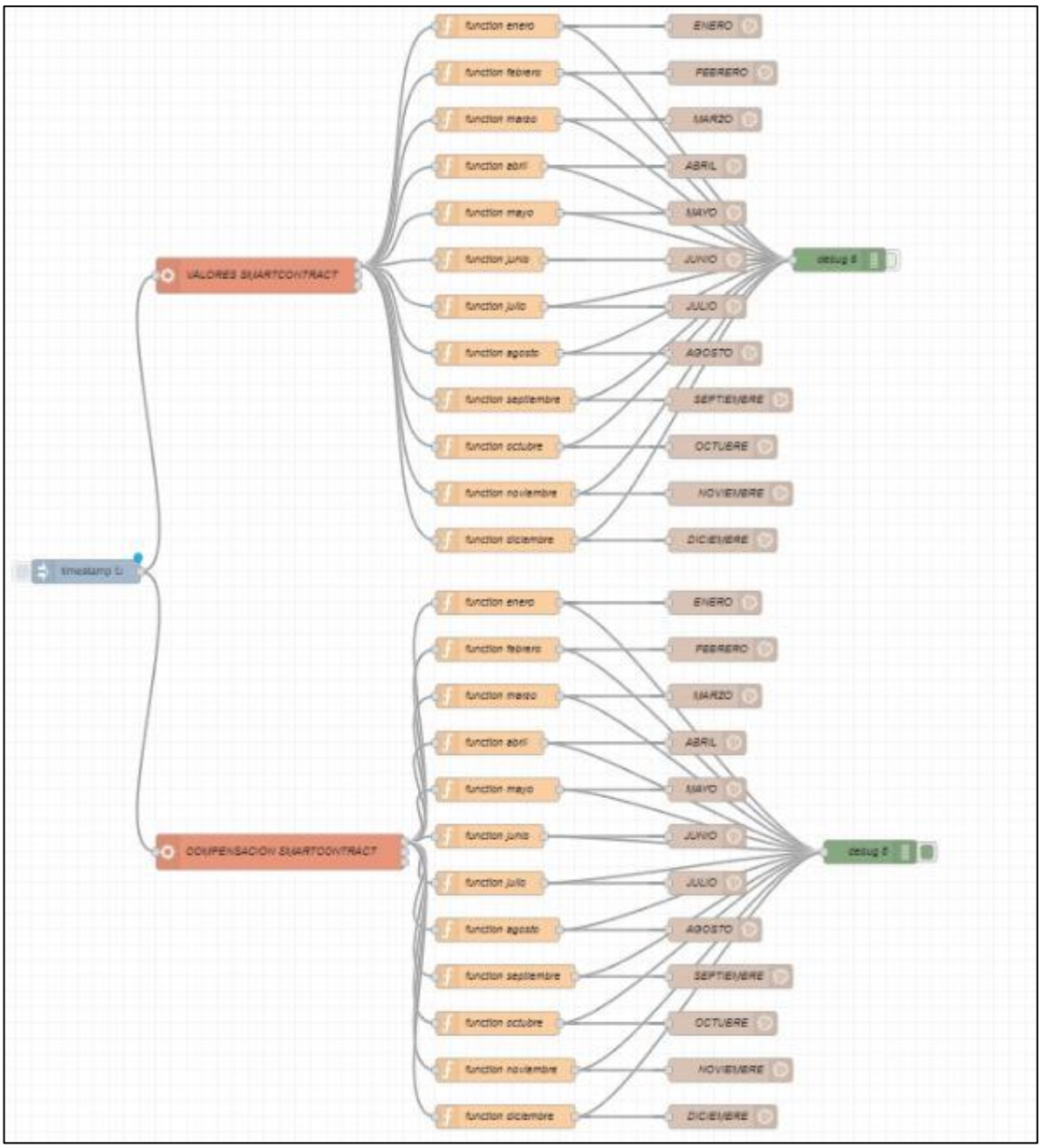


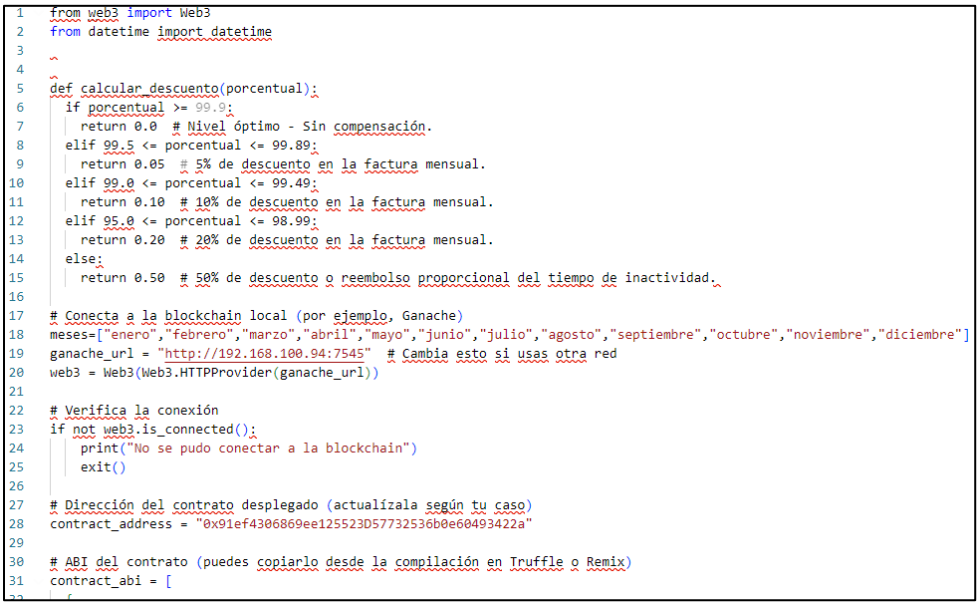
Figura 3.30 Flujo de Node-RED

En la Figura 3.30 se puede apreciar que se ha distribuido a topología en dos partes principales. La primera parte, el nodo “VALORES SMARTCONTRACT”, ejecutando el código en Python del anexo 3, busca leer y obtener los valores registrados en el contrato inteligente, de esta manera es posible conocer el tiempo que no ha estado disponible el servicio para el usuario gracias al monitoreo constante a través de Zabbix, para finalmente mostrarlo por pantalla al usuario cuando lo requiera. Posteriormente, las funciones de Node-RED en el flujo etiquetadas para cada uno de los meses utilizan un código desarrollado en Java, el cual se muestra a continuación en la Figura 3.31 y con una visión completa en el anexo 5. Con este código se logra realizar las modificaciones de formato necesarias para enviar la información a la base de datos de influx DB.

```
// Supongamos que msg.payload contiene el resultado del exec
let execOutput = msg.payload;

// Verificamos si es un string
if (typeof execOutput !== 'string') {
    return { payload: "El resultado del exec no es un string válido" };
}

// Reemplazamos las comillas simples por comillas dobles para hacer un JSON válido
execOutput = execOutput.replace(/'/g, '"');

// Intentamos convertir el string a un objeto JSON
let data;
try {
    data = JSON.parse(execOutput); // Ahora debería ser un objeto válido
} catch (error) {
    return { payload: "Error al parsear el resultado del exec: " + error.message };
}

// Creamos un array para almacenar todos los puntos de datos
let points = [];

// Recorremos todos los meses del objeto

let value = data["enero"];

// Aseguramos que 'tags' contenga solo un valor simple y no un objeto complejo
let point = {
porcentaje_de_disponibilidad : value,       // 'fields' con valor simple
};

// Añadimos el punto al array
points.push(point);
```

Figura 3.31 Código de las funciones de Node-RED

La segunda parte del flujo de la Figura 3.30, el nodo “COMPENSACION SMARTCONTRACT” se enfoca en el cálculo de las compensaciones del usuario para mostrar estos datos a través del Dashboard y que el usuario pueda acceder en cualquier momento de forma intuitiva a esta información. Esto también se lo realiza con la ayuda del código mostrado a continuación en la Figura 3.32 y detallado en su totalidad en el anexo 6.



```
from web3 import Web3
from datetime import datetime


def calcular_descuento(porcentual):
    if porcentual >= 99.9:
        return 0.0  # Nivel óptimo - Sin compensación.
    elif 99.5 <= porcentual <= 99.89:
        return 0.05  # 5% de descuento en la factura mensual.
    elif 99.0 <= porcentual <= 99.49:
        return 0.10  # 10% de descuento en la factura mensual.
    elif 95.0 <= porcentual <= 98.99:
        return 0.20  # 20% de descuento en la factura mensual.
    else:
        return 0.50  # 50% de descuento o reembolso proporcional del tiempo de inactividad.

# Conecta a la blockchain local (por ejemplo, Ganache)
meses=["enero","febrero","marzo","abril","mayo","junio","julio","agosto","septiembre","octubre","noviembre","diciembre"]
ganache_url = "http://192.168.100.94:7545"  # Cambia esto si usas otra red
web3 = Web3(Web3.HTTPProvider(ganache_url))

# Verifica la conexión
if not web3.is_connected():
    print("No se pudo conectar a la blockchain")
    exit()

# Dirección del contrato desplegado (actualízala según tu caso)
contract_address = "0x91ef4306869ee125523D57732536b0e60493422a"

# ABI del contrato (puedes copiarlo desde la compilación en Truffle o Remix)
contract_abi = [
```

Figura 3.32 Código de las compensaciones a los usuarios

Como se observa en la muestra del código, la primera instancia entre las líneas 5 y 15, detalla los umbrales de disponibilidad para ejecutar las compensaciones. Estos valores y su manera de aplicación se detallan de forma extendida en el capítulo 4 correspondiente a la sección de resultados. De igual forma, una vez procesada la información, es enviada a la base de datos.

Posteriormente, la plataforma Grafana utiliza la base de datos en InfluxDB para mostrar las métricas como el porcentaje de disponibilidad y la compensación aplicable según dicho porcentaje. Todo esto a través de un entorno visual, agradable y comprensible para el usuario.

- **Iteración de cálculos de compensaciones**

Las compensaciones debido a la falta de disponibilidad se deben actualizar de forma diaria, por este motivo, adicional a las configuraciones mencionadas previamente, se programó en Node-RED que el código para el cálculo de compensaciones sea ejecutado de forma automática diariamente a las 12h00, tal como se muestra en la Figura 3.33.

De esta forma, se garantiza que la información esté lo más actualizada posible para que el usuario pueda acceder a ella en el momento que desee.

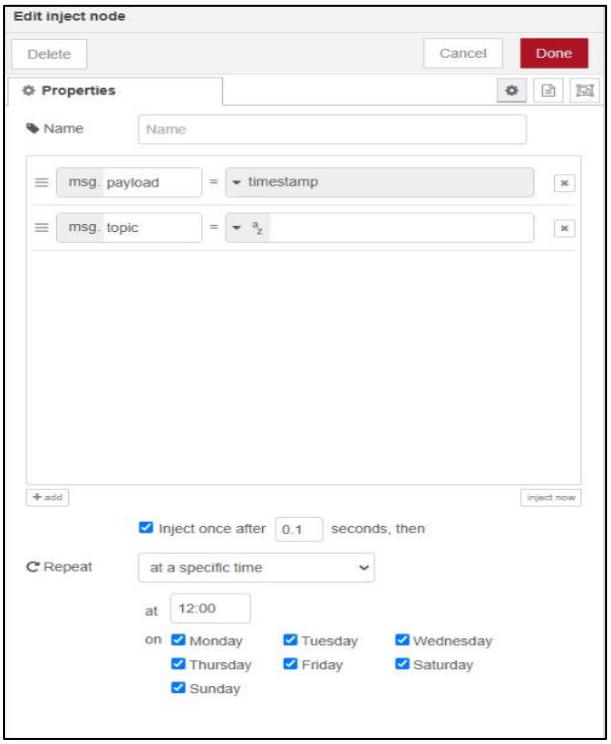


Figura 3.33 Iteración del código de cálculo de compensaciones.

CAPÍTULO 4

4. Resultados y análisis

Este capítulo comprende las pruebas de funcionamiento ejecutadas y los resultados obtenidos, enfatizando la viabilidad y eficiencia de la solución planteada. Entre estas pruebas se destacan la simulación de la red monitoreada por medio de Zabbix, así como la generación y el registro de alertas. También se presenta el procesamiento y envío de las alarmas hacia la blockchain, junto con el despliegue del contrato inteligente. Adicionalmente, se muestra la operatividad de la interfaz gráfica para que los usuarios puedan visualizar el rendimiento de su servicio y sus compensaciones de ser el caso.

De esta forma, se analiza el rendimiento del sistema en las diferentes fases implementadas, resaltando los aciertos y áreas de oportunidad de mejora.

4.1. Prueba de funcionamiento del sistema

Este aparatado comprende la explicación de la prueba de funcionamiento ejecutada para el todo el sistema desarrollado, entiendo el entorno en el que se ejecutó y cómo se desarrollaron las pruebas.

4.1.1. Simulación de la red

Para lograr simular el equipo del cliente monitoreado, se implementó una red en GNS3 como se muestra en la Figura 3.3, la topología permite la simulación de una red que cumpla las funciones de una red de un proveedor de Internet. En esta red se utiliza el protocolo de enrutamiento dinámico OSPF, dada sus facilidades de configuración y escalabilidad hasta medianas empresas. En esta red se monitoreará el funcionamiento de un equipo, el cual cumplirá el papel del dispositivo final de un usuario y se lo denomina PEGYEREGIONAL.

Figura 4.1. Equipo monitoreado

Como se aprecia en la Figura 4.1, el equipo monitoreado es un router, al cual se le asignó la loopback de monitoreo 10.10.249.1, la cual se verificará su comunicación con el servidor.

- **Comunicación con el datacenter**

Como se mencionó, para el enrutamiento dinámico de la red se utilizó el protocolo OSPF, de esta forma se puede evidenciar en la Figura 4.2 la tabla de enrutamiento del datacenter o en su defecto del dispositivo denominado “PEDATACENTER”, que se lo puede visualizar en la parte inferior de la topología de la Figura 3.3.

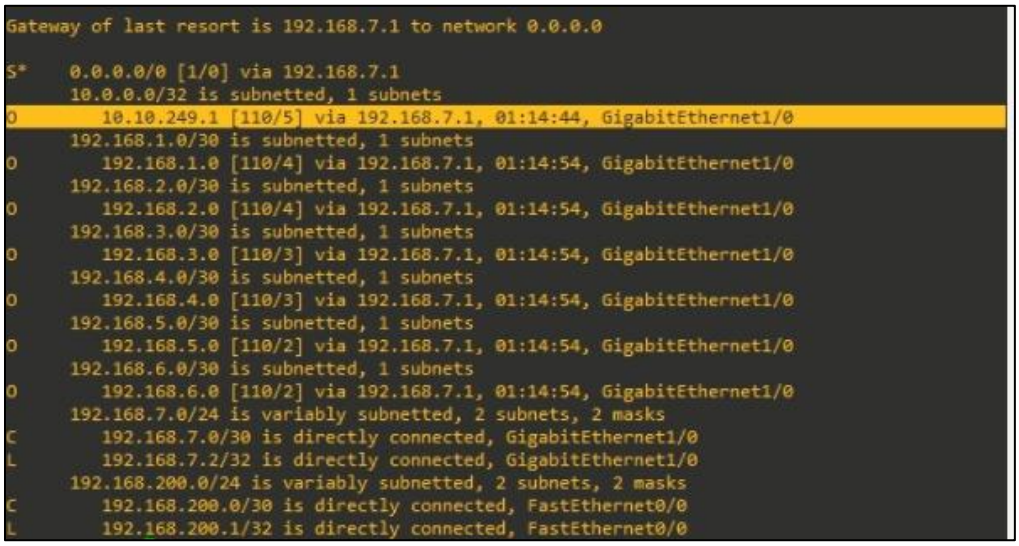


Figura 4.2. Tabla de enrutamiento del data center

En esta tabla se puede observar como el servidor ha aprendido la ruta hasta la loopback de monitoreo 10.10.249.1 y por lo cual la comunicación entre ambos es factible.

4.1.2. Sistema de monitoreo

La implementación del sistema de monitoreo, como se lo explicó en las secciones 3.1 y 3.3.1, se lo llevó a cabo mediante Zabbix. Este servidor se encuentra alojado en una máquina virtual en el cual también se ejecutan varias funciones y en GNS3 se encuentra mediante una conexión física al dispositivo “PEDATACENTER” como se puede observar en la Figura 3.3.

- **Tablero de monitoreo**

En la figura a continuación se muestra la vista principal de los equipos monitoreados por Zabbix.

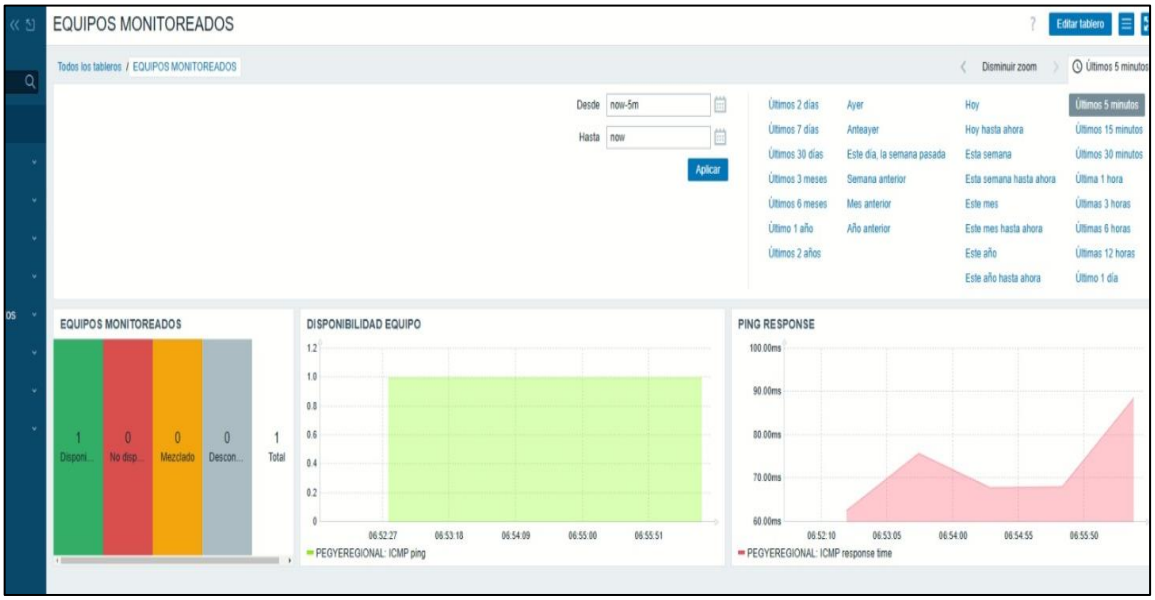


Figura 4.3. Tablero de monitoreo de Zabbix

En esta figura se muestra un dashboard que para fines prácticos y de desarrollo de este proyecto, se está monitoreando un dispositivo como antes se mencionó. Además, Zabbix proporciona gráficas referentes a la disponibilidad de dicho equipo y su respuesta ping.

- **Alarmas**

El sistema de monitoreo, como se detalla en la sección 3.3.1, se ha configurado para que evalúe las pérdidas de disponibilidad o de respuesta de comunicación entre el equipo monitoreado y el servidor. De esta forma al registrarse cada una de estas eventualidades el sistema de monitoreo la registrara en un archivo de texto plano (.txt).

- **Creación de archivo ".txt"**

La Figura 4.4 muestra todas las alarmas que ha sido confirguradas y las cuales se envía a Python para así se registradas en un archivo de texto (.txt)

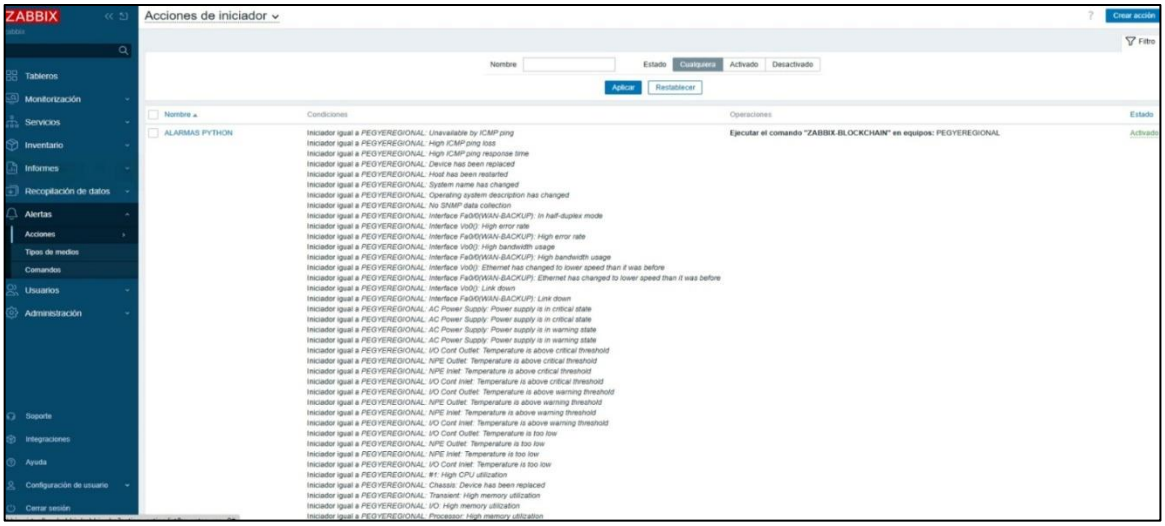


Figura 4.4. Configuración de Zabbix para crear el archivo .txt de las alarmas

- **Archivo general de alarma**

Una vez ejecutado los comandos y detectadas las alarmas, en este caso por pérdidas de comunicación con el dispositivo monitoreo, Zabbix como se explica en la sección anterior, crea un archivo con el detalle de estas alarmas. En este archivo se encuentran registradas todas las alarmas junto con información como la fecha y hora a la que ocurrieron, el tipo de alerta, entre otros detalles.

- **División de archivos de alarma**

Para llevar un control más organizada y fácil de gestionar, se decidió dividir los archivos de alarma por mes de ocurrencia, para esto se empleó un script de Python (revisar anexo 2) que realiza esta función y se puede apreciar en la Figura 4.5 a continuación.

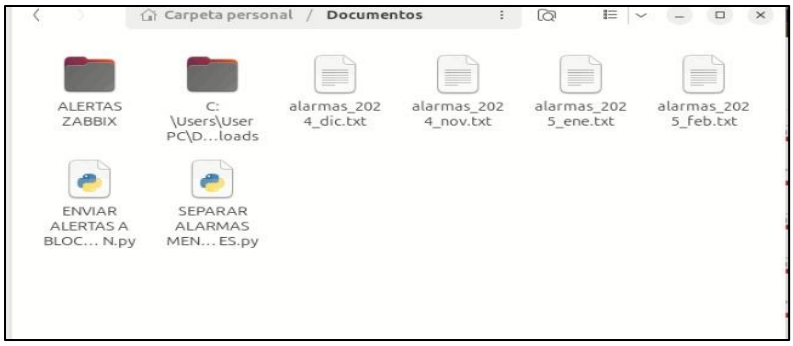


Figura 4.5. Archivos de alarmas y programas de Python

Una vez ejecutado este código, como se aprecia en la Figura 4.5, el archivo de alarmas general fue borrado y reemplazado por archivos de alarmas individuales de cada mes.

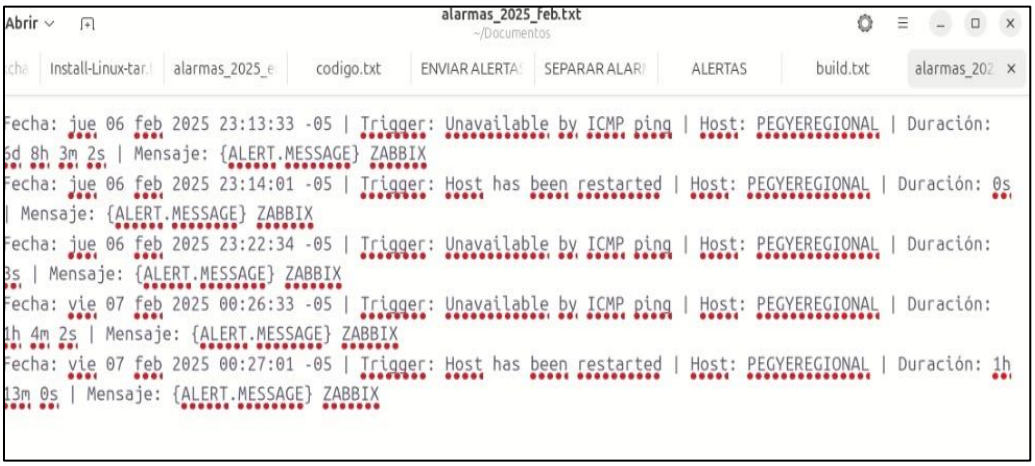


Figura 4.6. Archivo de alarmas de febrero

Al ingresar a uno de estos archivos, como en el caso de la Figura 4.6, se puede apreciar solo la información de las alarmas de dicho mes, en esta prueba se observa el mes de febrero, registrado los periodos en los que no hubo respuesta ping del dispositivo monitoreado.

- **Alerta vía correo electrónico**

Adicionalmente, al registro de alarmas en los archivos de texto, otra de las configuraciones de Zabbix, permite que el usuario y equipo de soporte sea notificado por medio de correo electrónico, indicando el tipo de problema y la fecha en el que sucede.

- **Envío de alertas a la blockchain**

Una vez en este que se ha creado cada uno de los archivos individuales de cada mes, se procede a leer estos archivos y enviarlos a la blockchain ejecutando el otro programa de Python mostrado en la Figura 4.5, denominado “ENVIAR ALERTAS A LA BLOCKCHAIN” (revisar anexo 3)

Figura 4.7. Ejecución del programa de envío de alertas

En la Figura 4.7 se visualiza la ejecución del programa mencionado, para esta prueba se analizó el caso del mes de febrero, mostrando que se han registrado 5 alarmas y un periodo de 427 minutos pérdida de disponibilidad del servicio. Además, se muestra el hash de la transacción. Este hash funciona como una especia de huella digital para cada transacción para que puedan ser identificada y sin riesgo de que se pueda alterar a información.

Cabe recalcar que para que se puede llevar a cabo esta transacción, el contrato inteligente debe estar establecido previamente y se lo explica en la sección 4.1.3.

- **Actualización de los archivos de alarma**

Para que los archivos de alarma se actualizados con la nueva información cada mes y así mismo estos datos sean enviados a la blockchain, se programó el servidor de Zabbix mediante Crontab, el cual en sistemas Linux es capaz de ejecutar programas o scripts en horarios predefinidos. De esta forma, se configuró para que el programa de división de alarmas se ejecuta cada fin de mes y el de envío de datos a la red blockchain se ejecute el primero de cada mes, permitiendo así que la información esté actualizada.

4.1.3. Establecimiento del contrato inteligente

Para desplegar el contrato se utiliza primero Ganache para simular la red Blockchain. Así se obtienen los recursos necesarios para el establecimiento del contrato, como la dirección de este y las wallest disponibles.

- **Dirección del contrato inteligente**

Tal como se explicó antes a través de Ganache y Truffle se despliega el contrato inteligente, proporcionando su dirección, como se refleja en la Figura 4.8

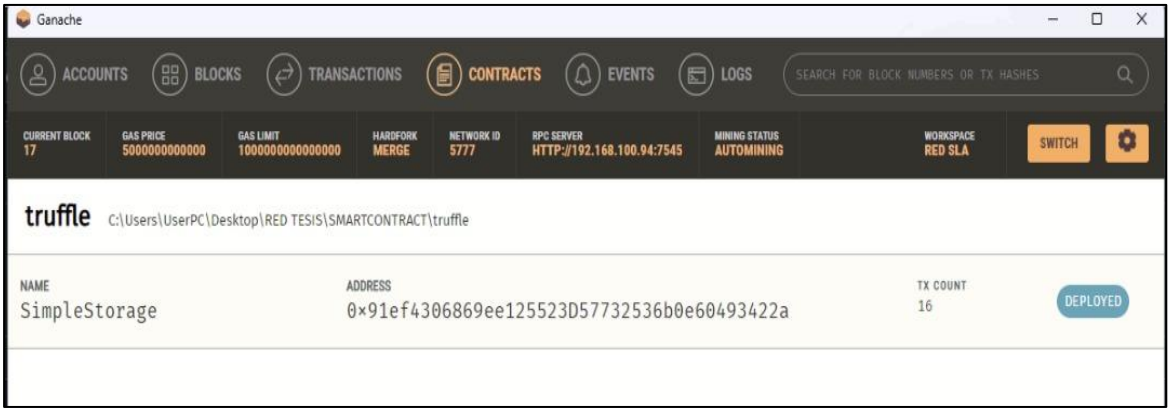


Figura 4.8. Contrato inteligente operativo

El contrato se lo ha desarrollado en Solidity y tiene una estructura simple con los umbrales de porcentaje de disponibilidad y su correspondiente compensación y otra para registrar los valores de cada mes cuando el servicio no ha estado disponible. Esto se encuentra de forma más detallada en la sección de análisis resultados.

- **Transacciones**

Una vez definida la estructura para el intercambio de información se ejecutan transacciones. Estas transacciones para poder efectuarse consumen gas que es el costo de envío de la información a través de la red blockchain. Para esto Ganache suministra wallets con el gas o saldo suficiente para el envío de la información y registro de los tiempos de indisponibilidad en el contrato inteligente.

En la Figura 4.9 se observa el registro de las tracciones ejecutadas donde muestra la dirección de origen de la información y el gas consumido en dicha transacción.

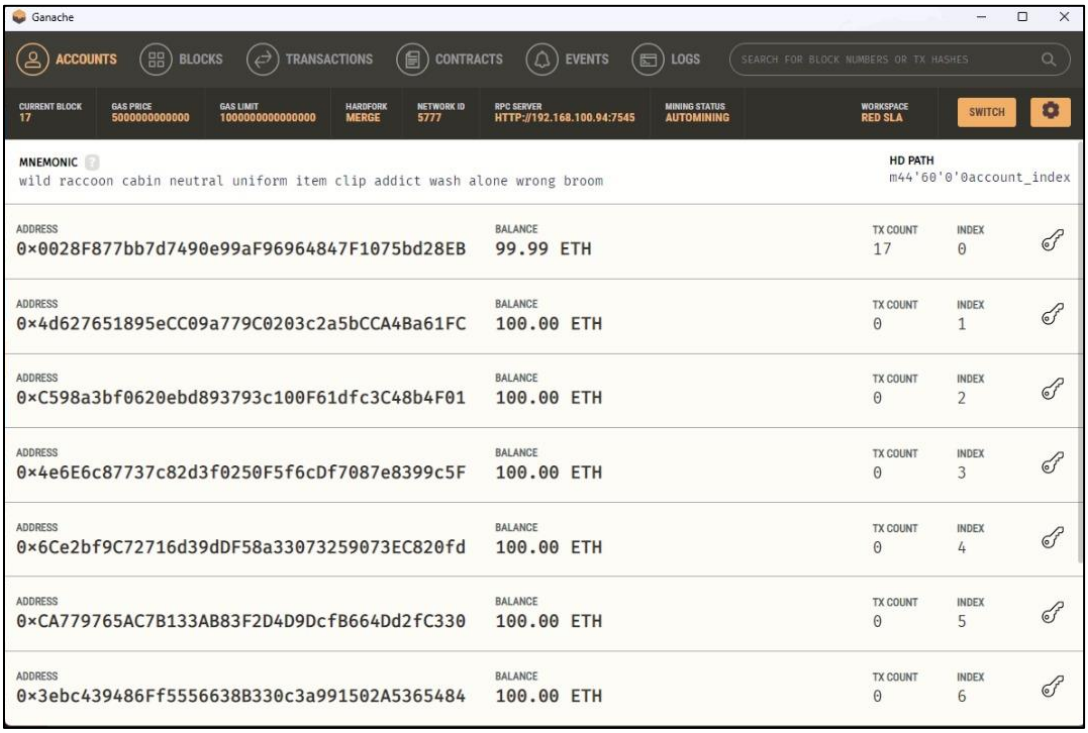


Figura 4.9. Transacciones

4.1.4. Operación de la interfaz gráfica

Para el desarrollo de esta interfaz se utilizó Node-RED, InfluxDB y Grafana para así poder presentar la información al usuario en un entorno más amigable.

- **Flujo de Node-RED**

En Node-RED, se desarrolló el flujo de la Figura 3.30, con este flujo se realizan dos funciones. La primera lee los valores de lo periodos en lo que no ha estado disponible el servicio y los adapta para almacenarlos en una base de datos. La segunda función se centra en que de acuerdo con estos porcentajes de indisponibilidad realizar los cálculos de las posibles compensaciones aplicables según sea el caso y enviar estos valores a la base de datos.

- **Base de datos**

La información procesada por Node-RED llega a la base de datos en InflurxDB, dado que permite gestionar y almacenar los datos con marcas de tiempo, ideal para el propósito del proyecto.

Una vez registrados los valores, esta base de datos será la fuente de información de grafana.

- **Grafana**

Con la información proporcionada a través de la base de datos antes mencionada, Grafana muestra un dashboard amigable para el usuario con los indicadores de porcentaje de disponibilidad y la compensación aplicable según corresponde, tal como se muestra en la Figura 4.10 y analizado más a detalle en la sección de resultados posteriormente.

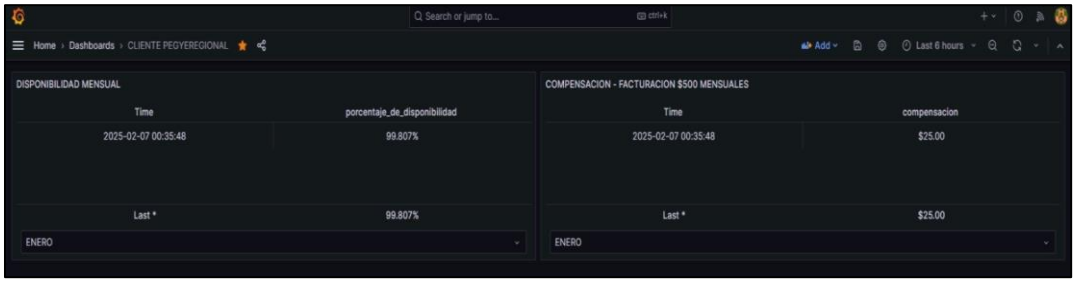


Figura 4.10. Dashboard de Grafana

4.2. Análisis de resultados

Esta sección muestra los resultados y el análisis de estos de una forma más detallada. Se detallan los resultados de cada una de las fases de la propuesta de solución.

4.2.1. Funcionamiento del sistema de monitoreo

Este apartado consiste en el análisis de las pruebas de funcionamiento ejecutadas de la fase 1 que comprende la implementación del sistema de monitoreo.

- **Reporte de incidencias**

La primera fase de pruebas comprende la evaluación del funcionamiento del sistema de monitoreo con respecto la detección y registro de las alarmas ocurridas en el equipo evaluado.

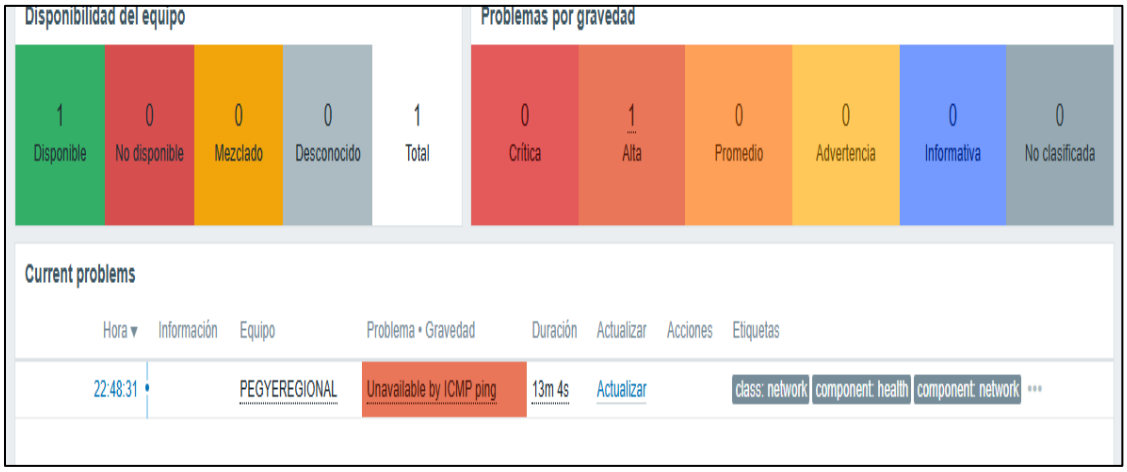


Figura 4.11 Problemas detectados

En la Figura 4.11 se muestra la operación del sistema de monitoreo, donde ha detectado una alarma, reportando la pérdida de disponibilidad en el equipo simulado del cliente, demostrando su correcta operación.

Mediante Zabbix, el sistema identifica de forma automática que el servicio se ha interrumpido, a través de la compilación de datos en tiempo real por medio del protocolo SNMP. Así, esta eventualidad se puede visualizar en la interfaz gráfica de Zabbix, por medio de alertas detalladas que muestran el tipo de problema. Lo que a su vez permite validad la eficiencia del monitoreo y permite una respuesta ágil de otros actores involucrados como el equipo de soporte técnico, garantizando que la pérdida de disponibilidad se atendida de forma eficaz.

- **LOGS**

Continuando con el proceso de la sección anterior 1, al registrar y detectar una alarma el sistema de monitoreo procede a grabarlas mismas en archivos de texto (.txt), como se logra observar en la Figura 4.12.

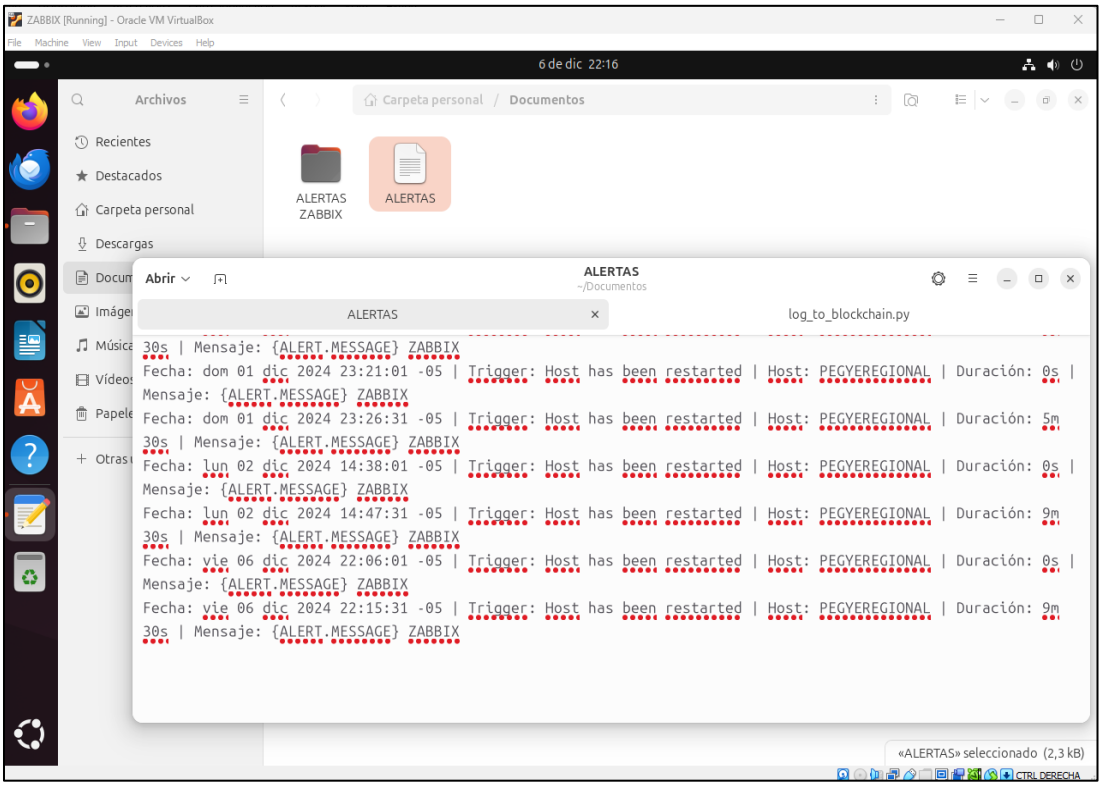


Figura 4.12 Alarmas almacenadas

Esta figura presenta como las alertas generadas debido a la pérdida de disponibilidad en el servicio, han sido detectadas y grabadas en el archivo de texto. Cada una de las alertas se registra con información importante del evento, como la hora y fecha de la interrupción, facilitando crear un historial ordenado y preciso de las alarmas identificadas.

Además, el formato de registro Este formato de registro agiliza la consulta y procesamiento automatizado de los datos dado su estructura, esto a su vez garantiza el seguimiento y trazabilidad de las eventualidades en la red.

- **Notificación por correo electrónico**

Además de registrar las alertas en archivos de texto, el sistema de monitoreo también se encarga de notificar, por medio de correo electrónico, a los clientes y personal técnico ante la pérdida de la disponibilidad del servicio, como se muestra en la Figura 4.13 a continuación.

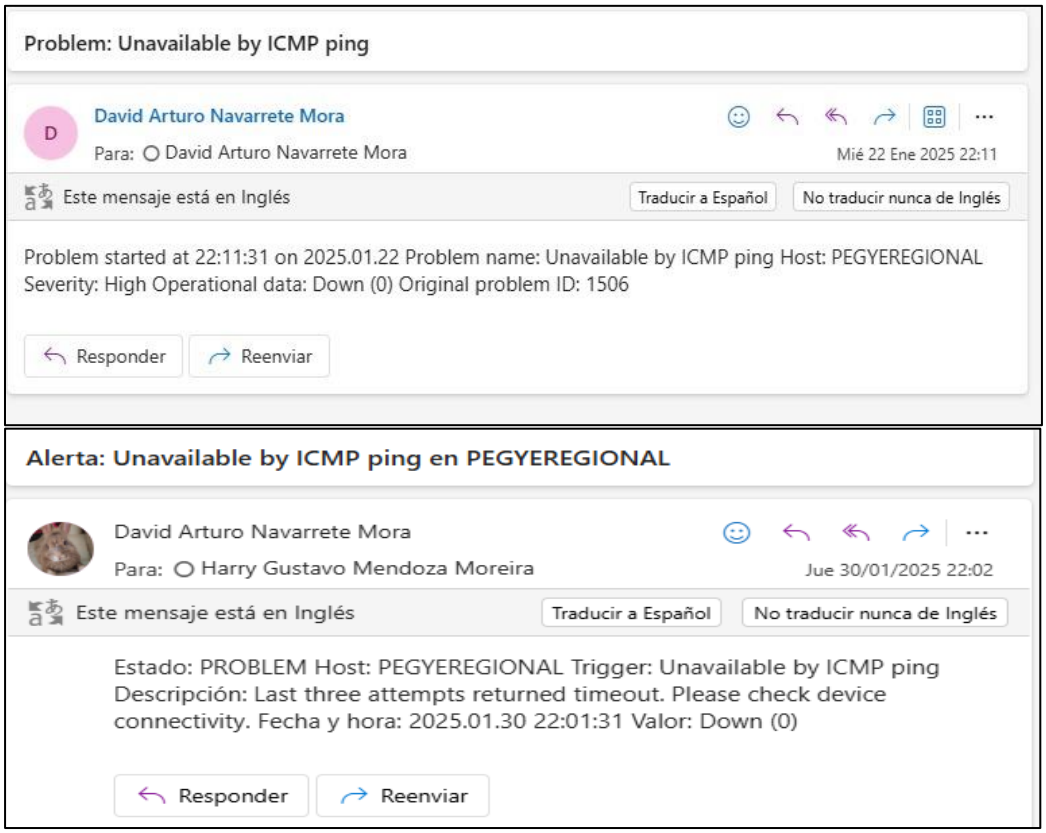


Figura 4.13 Notificación por correo electrónico

En esta Figura 4.13 se aprecia que el usuario y personal técnico han sido informados de la eventualidad registrada por medio del correo electrónico enlazado. Entre los detalles de la alarma, se especifica la pérdida de disponibilidad del servicio, así como la fecha y hora a la que aconteció.

4.2.2. Despliegue del contrato inteligente

La sección 4.2.2 muestra que el contrato inteligente ha sido desplegado, lográndose observar cómo gracias al entorno simulado de la blockchain a través de Ganache, el contrato se encuentra operativo, lo que se puede verificar en la Figura 4.14 gracias a las transacciones que se muestran en ella, indicando que el contrato ya está operativo.

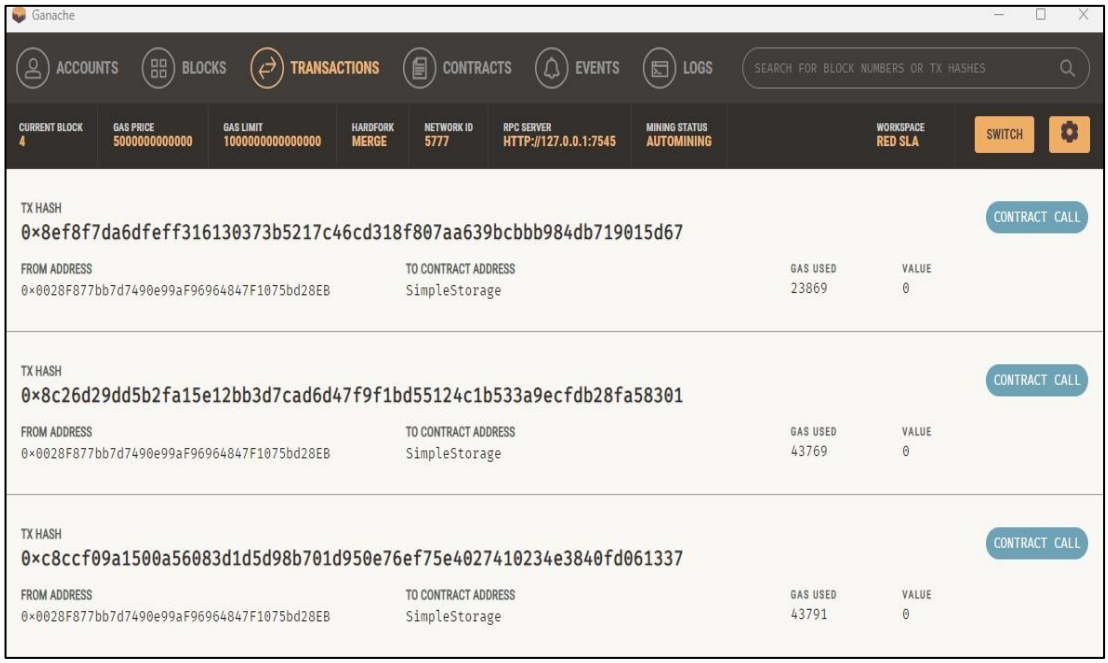


Figura 4.14 Contrato inteligente desplegado

4.2.3. Funcionamiento de la integración del sistema de monitoreo con el contrato inteligente.

Esta sección comprende los resultados de las pruebas de funcionamiento y los resultados de la fase 3. Aquí se evalúa el proceso de comunicación entre Zabbix y la blockchain, así como la respuesta del contrato inteligente ante las eventualidades registradas.

Tal como se explica en las secciones 3.3.3, para lograr esta integración, el sistema de monitoreo clasifica y agrupa las alarmas registradas, por mes de ocurrencia para posteriormente ser enviadas a la blockchain y el contrato inteligente. Finalmente, después de todo este proceso, el usuario mediante Grafana logra acceder a la visualización del desempeño del servicio y verificar si posea alguna compensación por pérdida de disponibilidad. De esta forma se muestran a continuación los resultados de estas etapas.

- **Clasificación de Alertas.**

Para llevar un registro ordenado de las alarmas registradas, se decidió agruparlas de acuerdo con el mes en el que sucedían. Esto se lo realiza a través del servidor

de Zabbix. De esta manera como se menciona en la sección 3.3.3 en el apartado de edición de los archivos de alarma, se emplea el script de Python cuyo código se encuentra en el anexo 2.

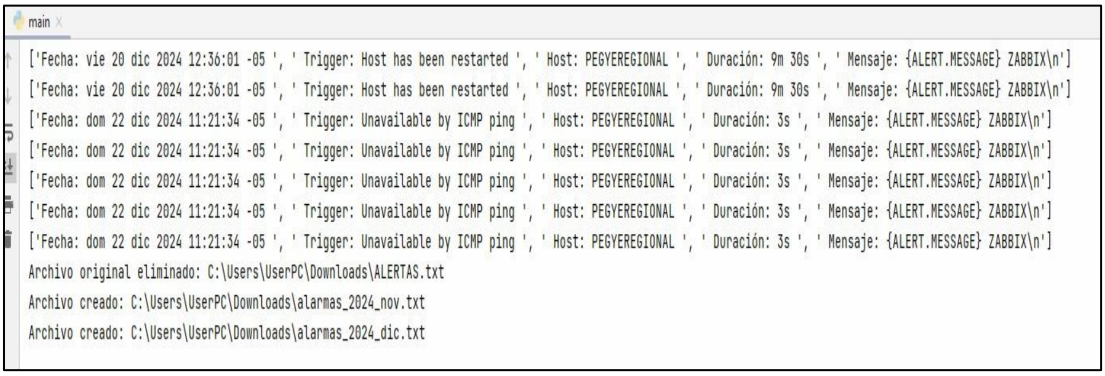


Figura 4.15 División de archivos de alarma

Lo mencionado en el párrafo anterior se puede observar en la Figura 4.15, donde se muestra a través del terminal, el resultado de la ejecución del código para la clasificación de alertas por meses.

En este caso se muestra que se han creado dos archivos de alarma correspondientes a lo registrado durante los meses de diciembre y enero. Cumpliendo así con el propósito de tener historiales ordenados para facilitar el procesamiento de la información.

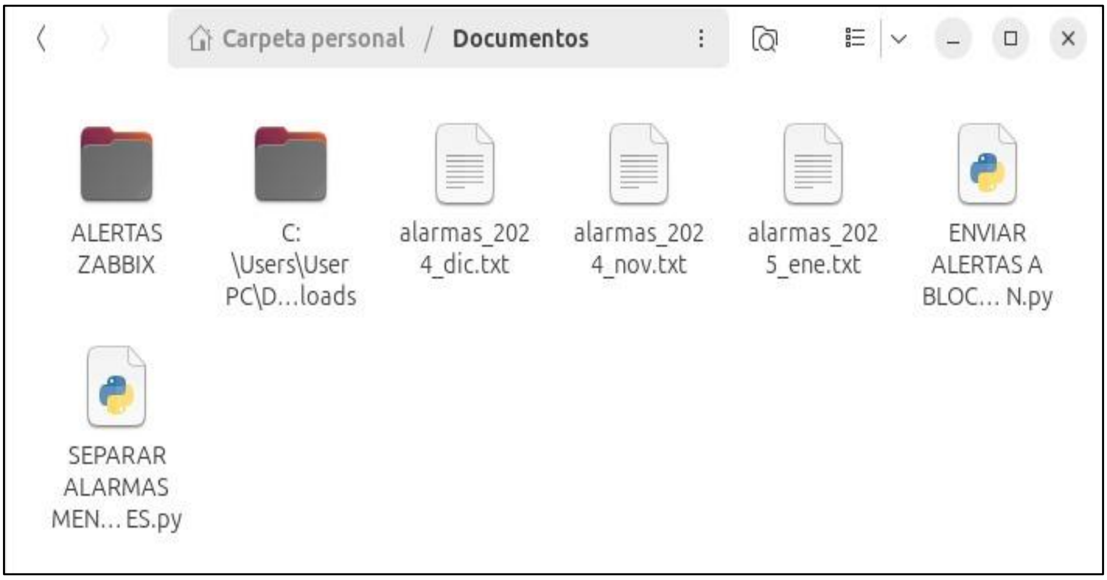


Figura 4.16 Archivos de alarmas creados

Al igual que con la figura anterior, en esta Figura 4.16 se evidencia que en el servidor de Zabbix se crean un archivo distinto por cada mes en el que ha ocurrido una alerta por pérdida de disponibilidad. En este caso se muestran los archivos para las alarmas de los meses de diciembre, noviembre y enero que han sido creados luego de ejecutar el código del anexo 2.

- **Registro de alertas.**

Posteriormente de contar con los archivos de alertas de cada mes, se envía esta información al contrato inteligente para determinar si deben existir compensaciones o no para los usuarios. Esto se lo lleva a cabo con la ejecución, en el servidor de Zabbix, del código del anexo 3 cuyo resultado se muestra en la Figura 4.17.

Figura 4.17 Lectura de alarmas y registro en el contrato inteligente

Al ejecutarse el código del anexo 3, se leen las alarmas ocurridas en el mes, determinando el tiempo de inactividad para posteriormente enviar esta información hasta el contrato inteligente como se lo explica en los apartados de la sección 3.3.3 y como se muestra en la Figura 4.17. Aquí se observa el tiempo total de inactividad del mes (305 minutos en el mes de enero) y cómo se ha realizado la transacción por la blockchain.

- **Transacciones ejecutas**

Para efectuar las acciones descritas en el apartado anterior y que los valores de tiempo de interrupción del servicio sean registrados en el contrato es necesario

que se realicen transacciones en la blockchain con monedas virtuales. Para esto se emplea la cuenta proveída por Ganache que simula el entorno de la blockchain.

Figura 4.18 Transacciones ejecutadas

En la Figura 4.18, se logra observar cómo las transacciones con la criptomoneda Ethereum, están siendo efectuadas por lo que nuestra cuenta en la billetera digital disminuye.

4.2.4. Presentación de la interfaz gráfica

En el capítulo 3, en la sección 3.3.4 se detalla que para la presentación de los datos al usuario se realiza un dashboard con programación a través de Node-RED, influx DB y Grafana. En esta interfaz se podrán observar los porcentajes de disponibilidad del servicio, así como si existe alguna compensación para el usuario en base a los porcentajes antes mencionados y la fecha y hora de la última actualización ejecutada.

Con el dashboard el usuario es capaz de seleccionar el mes que desee para conocer los valores mencionados.

- **Cálculo de compensaciones**

Antes de mostrar la información a través del dashboard, es importante detallar la forma como se calculan las compensaciones. Como se lo explica en el capítulo 2, los SLA son contratos entre el proveedor del servicio y el cliente, en este contrato se detallan las responsabilidades del proveedor y del usuario, referente principalmente a la calidad del servicio y las normas o estándares que debe

cumplir, así como las acciones a ejecutar en caso de que se incumplan las condiciones estipuladas.

Para calcular las compensaciones se hará uso de la Tabla 4.1 mostrada a continuación:

Tabla 4.1 Compensaciones

Disponibilidad mensual	Tiempo máximo de inactividad admitido	Compensación
99.9% o más (Nivel óptimo)	≤ 43.2 minutos	Sin compensación.
99.5% - 99.89%	43.2 min – 3.6 horas	5% de descuento en la factura mensual.
99.0% - 99.49%	3.6 – 7.2 horas	10% de descuento en la factura mensual.
95.0% - 98.99%	7.2 – 36 horas	20% de descuento en la factura mensual.
Menos de 95.0%	Más de 36 horas	50% de descuento o reembolso proporcional del tiempo de inactividad.

En esta Tabla 4.1 se pueden visualizar los umbrales de tiempo de disponibilidad del servicio mensualmente junto con su respectiva compensación aplicable. Cabe destacar que los valores en la tabla se los definió de esta manera con fines prácticos para el desarrollo del proyecto. Dichos valores presentes en la tabla pueden variar de acuerdo con las políticas propias de cada proveedor del servicio de Internet o a las normas que regulan su operación.

Adicionalmente para fines del desarrollo de la propuesta, se definió que el cliente monitoreado paga por el servicio un valor de $500,00 mensuales. A partir de este valor, se podrá ver más adelante una prueba de compensación aplicada por la falta de disponibilidad.

- **Paneles del porcentaje de disponibilidad y compensaciones**

Al acceder al dashboard de Grafana en primera instancia, se puede observar el panel que permite seleccionar el mes al cual se desea realizar la consulta.

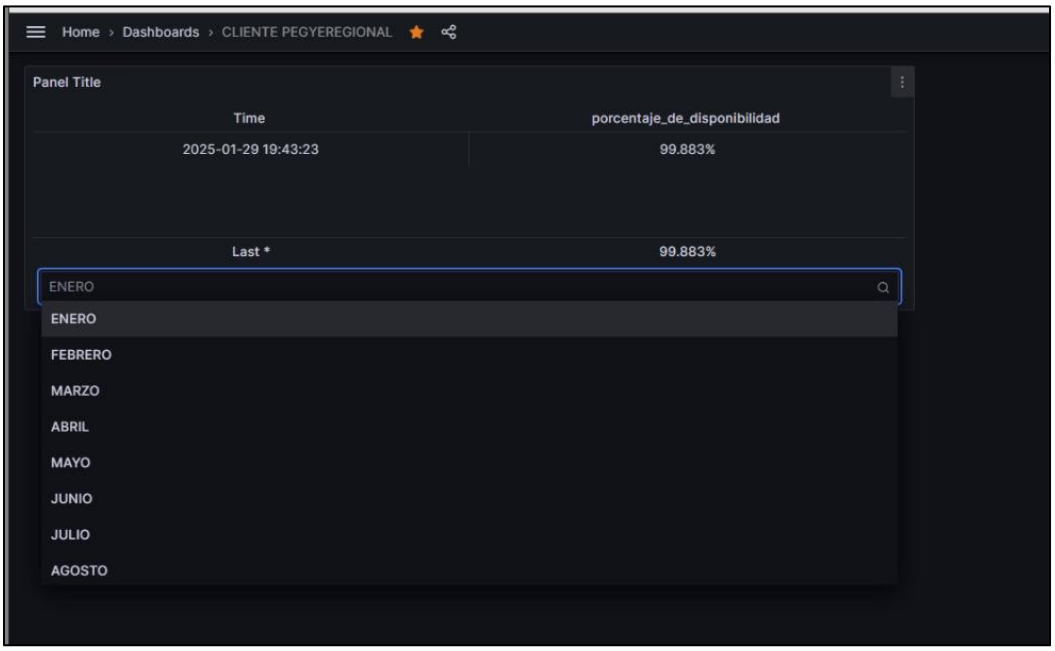


Figura 4.19 Panel de porcentaje de disponibilidad

En la Figura 4.19 se visualiza lo antes mencionado. Se cuenta con una lista desplegada para seleccionar el mes que se desea realizar la consulta del porcentaje de tiempo que el servicio ha estado disponible. De esta forma el sistema permite acceder a la información de forma fácil e intuitiva.

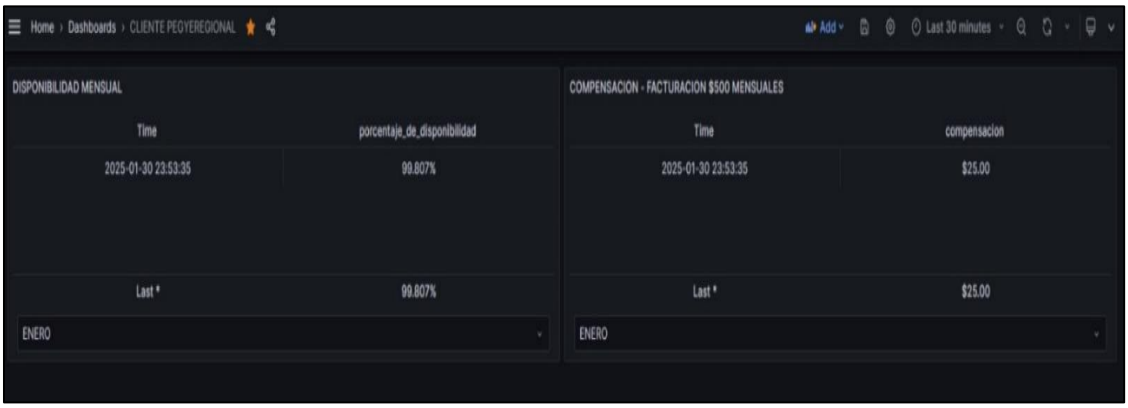


Figura 4.20 Dashboard de Grafana

La Figura 4.20 muestra la visión completa del Dasboard donde se presentan los dos paneles, el del porcentaje de disponibilidad, el de la compensación aplicable y la fecha y hora de la última actualización de la base de datos.

En este caso se simuló que el mes consulado es el de Enero, donde se registró un porcentaje de disponibilidad de 99,807%, por lo que según la Tabla 4.1 le corresponde una compensación del 5% al costo mensual del servicio. De esta manera el usuario deberá recibir una compensación de $25, dado los $500 mensuales que normalmente pagaría por el servicio contratado.

De esta forma, el contrato inteligente permite ejecutar estas de forma automatizada estas compensaciones de forma segura y transparente, permitiendo un trato justo al usuario y ahorrando gastos adicionales al proveedor por la actuación de terceros actores en este proceso.

CAPÍTULO 5

5. Conclusiones y recomendaciones

Luego de ejecutar la solución propuesta de un sistema de monitoreo basado en contratos inteligentes, demostrando su funcionamiento mediante el desarrollo de cada una de sus fases y las respectivas pruebas de funcionamiento, en este capítulo se detallan las conclusiones y recomendaciones del trabajo realizado.

5.1. Conclusiones

- Se logró a implementar un entorno de red a través de GNS3, consiguiendo simular la conexión entre un cliente final hasta su servicio de Internet de forma muy precisa. Gracias a esto se pudo desarrollar pruebas confiables y así valorar la eficiencia del sistema de monitoreo, permitiendo supervisar el trabajo del equipo del cliente en tiempo real y generando alertas inmediatas frente a pérdidas de disponibilidad. Con estas herramientas del sistema es posible la detección temprana de problemas de operatividad y la respuesta y solución rápida de los mismos.

- Con la implementación de la blockchain y el contrato inteligente, gracias a la ayuda de herramientas como Ganache, Truffle y Solidity, se consiguió que dicho contrato tenga la capacidad de ejecutar las compensaciones de forma automática en base a los términos establecidos en el SLA. Gracias a esto se prescinde de la necesidad de terceros para realizar estas acciones de forma manual, permitiendo disminuir los tiempos de respuesta y asegurar la transparencia en el cumplimiento de los acuerdos y las compensaciones.

- El desarrollo de la interfaz gráfica otorga al usuario una herramienta intuitiva para visualizar en cualquier momento los porcentajes de disponibilidad y las compensaciones a las que puede aplicar dichos porcentajes. Con esto se aumenta la satisfacción del cliente y la transparencia en el cumplimiento de las compensaciones.

- El esquema de la arquitectura centrada en Blockchain asegura que los datos recopilados y las transacciones efectuadas en el contrato inteligente se caractericen por ser inmutables y seguras, protegiendo los datos de cualquier

alteración y aumentando la confianza de la gestión del SLA y el sistema de monitoreo.

- Con las pruebas realizadas y la simulación de estos entornos se evidenció que la propuesta planteada es factible para ser implementada por proveedores de Internet en un entorno real, agilizando el cumplimiento de las estipulaciones del SLA y disminuyendo los tiempos de respuesta ante eventualidades.

5.2. Recomendaciones

- Se sugiere simular entorno de redes más complejas con la finalidad de ejecutar pruebas con un mayor número de dispositivos para medir el rendimiento del sistema de monitoreo ante escenarios con un mayor flujo de datos y de variaciones de disponibilidad del servicio.
- Se aconseja revisar posibles mejoras en el contrato inteligente en cuanto al consumo de gas para mejorar su eficiencia e investigar alternativas de escalabilidad en entornos de redes blockchain más sofisticadas.
- Considerando el auge de la inteligencia artificial (IA), se propone integrar algoritmos de IA que faciliten el análisis de patrones en la información registrada por el sistema de monitoreo Zabbix con el objetivo de prever las alarmas antes de que sucedan.
- Por último, es importante pulir la interfaz gráfica para que el usuario cuente con mecanismos más fáciles de acceder e interactivos, implementando incluso reportes automatizados.

REFERENCIAS

3 Características de Zabbix. (2024). (Zabbix) Recuperado el 5 de julio de 2024, de https://www.zabbix.com/documentation/4.0/es/manual/introduction/features#:~:text=Zabbix%20es%20una%20soluci%C3%B3n%20de,funcionalidades%20en%20un%20solo%20paquete.&text=se%20pueden%20definir%20umbrales%20muy,base%20de%20datos%20previamente%20recopilados.

Alfaro, J. (22 de Abril de 2021). *BlockChain-As-A-Service: Análisis y validación.* Obtenido de Universitat Politècnia de Cantalunya: https://upcommons.upc.edu/bitstream/handle/2117/348786/155954.pdf?sequence=1

Contreras, R. (10 de Abril de 2024). *SLA en contratos TIC: guía para una perfecta ejecución*. Obtenido de Digital360 Iberia: https://www.computing.es/mundo-digital/slas-y-penalizaciones-en-contratos-de-servicios-tic/

Fetsyak, I. (Diciembre de 2020). *CONTRATOS INTELIGENTES: ANÁLISIS JURÍDICO DESDE EL MARCO LEGAL ESPAÑOL*. Obtenido de Universidad de la Rioja: http://www.Dialnet-ContratosInteligentes-7814692.pdf

Freda, A. (11 de Noviembre de 2022). *¿Qué es la tecnología de cadena de bloques?* Obtenido de AVG: https://www.avg.com/es/signal/what-is-blockchain

Genos. (2024). *Monitorización Cacti by Genos*. Recuperado el 5 de julio de 2024, de https://genos.es/cacti-soporte/

Gómez, I. (Septiembre de 2018). *BLOCKCHAIN. LA REVOLUCIÓN DE LA INDUSTRIA.* Obtenido de Escola Tècnica Superior d'Enginyeria Industrial de Barcelona: https://upcommons.upc.edu/bitstream/handle/2117/122913/in-s-tfg-bc.definitivo-jf.pdf?sequence=1

González, T. (Junio de 2019). *LOS CONTRATOS INTELIGENTES: UN NUEVO RETO PARA EL DERECHO INTERNACIONAL .* Obtenido de Universidad Pontificia Comillas: https://repositorio.comillas.edu/xmlui/bitstream/handle/11531/29632/tfg%20final%20final%20final.pdf?sequence=1

Intelligence, M. (2024). *Blockchain en el tamaño del mercado de telecomunicaciones y análisis de participación tendencias de crecimiento y pronósticos (2024-2029)*. Obtenido de Mordor Intelligence: https://www.mordorintelligence.com/es/industry-reports/blockchain-in-telecom-market

ionos, e. e. (10 de diciembre de 2023). *Nagios*. Recuperado el 5 de julio de 2024, de https://www.ionos.com/es-us/digitalguide/servidores/herramientas/nagios-todos-los-procesos-de-red-a-tu-alcance/

Jorge, P. (30 de Marzo de 2020). *Blockchain y contratos inteligentes: aproximación a sus problemáticas y retos jurídicos*. Obtenido de Universidad Externado de Colombia: https://www.redalyc.org/journal/4175/417564980007/html/

Max, D. (10 de Septiembre de 2022). *50 Customer Service Statistics You Need To Know (Updated For 2023)*. Obtenido de https://www.netomi.com/customer-service-statistics

Mikrotik. (s.f.). *The Dude*. Recuperado el 2024, de https://mikrotik.com/thedude

Nakamoto, S. (2008). *Bitcoin: A Peer-to-Peer Electronic Cash System.* Obtenido de Bitcoin.org: https://bitcoin.org/bitcoin.pdf

Olaizola, I. (Abril de 2020). *Estado del arte de la aplicación de la tecnología Blockchain en la Cadena de Suministro.* Obtenido de Escola Tècnica Superior d'Enginyeria Industrial de Barcelona:

https://upcommons.upc.edu/bitstream/handle/2117/189124/tfm-v015-final.pdf?sequence=1

PINTO, M. R. (2016). *GESTION DE MANTENIMIENTO DE HARDWARE.*

Porxas, N., & Conejero, M. (2018). *Tecnología blockchain: funcionamiento, aplicaciones y retos jurídicos relacionados.* Obtenido de Dialnet: https://www.uria.com/documentos/publicaciones/5799/documento/art02.pdf?id=7875

Rojas, S. (Junio de 2019). *Implicaciones del uso de blockchain aplicado en contratos inteligentes en procesos de contratación pública en México.* Obtenido de Centro de Investigación y Docencia Económicas, S.A.: https://repositorio-digital.cide.edu/bitstream/handle/11651/3626/164633.pdf

SeguriLatam. (6 de Noviembre de 2022). *¿Qué es un SLA y cómo cumplir un acuerdo de nivel de servicio?* Obtenido de SeguriLatam: https://www.segurilatam.com/actualidad/que-es-un-sla-y-como-cumplir-un-acuerdo-de-nivel-de-servicio_20211106.html

Swain, A. K., & Garza, V. R. (23 de Marzo de 2022). *Factores clave para lograr acuerdos de nivel de servicio (SLA) para la resolución de incidentes de tecnología de la información (TI)*. Obtenido de SpringerLink: https://link.springer.com/article/10.1007/s10796-022-10266-5

VAS Expert. (23 de Enero de 2023). *Cómo utilizar QoS para garantizar la calidad del acceso a Internet*. Obtenido de VAS Expert: https://vasexperts.com/es/blog/quality-of-service/how-to-use-qos-to-ensure-the-internet-access-quality/

wargaming. (6 de febrero de 2020). *¿Qué es PingPlotter? ¿Cómo utilizo PingPlotter?* Obtenido de https://na.wargaming.net/support/es_mx/products/wowp/article/19117/

ANEXOS

Anexo 1: Código del Contrato Inteligente

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.4.22 <0.9.0;

contract SimpleStorage {
  uint256 enero;
  uint256 febrero;
  uint256 marzo;
  uint256 abril;
  uint256 mayo;
  uint256 junio;
  uint256 julio;
  uint256 agosto;
  uint256 septiembre;
  uint256 octubre;
  uint256 noviembre;
  uint256 diciembre;

  // Funciones para leer los valores de cada mes
  function readenero() public view returns (uint256) {
    return enero;
  }

  function readfebrero() public view returns (uint256) {
    return febrero;
  }

  function readmarzo() public view returns (uint256) {
    return marzo;
  }

  function readabril() public view returns (uint256) {
    return abril;
  }

  function readmayo() public view returns (uint256) {
    return mayo;
  }

  function readjunio() public view returns (uint256) {
    return junio;
  }

  function readjulio() public view returns (uint256) {
    return julio;
  }
```

```
function readagosto() public view returns (uint256) {
  return agosto;
}

function readseptiembre() public view returns (uint256) {
  return septiembre;
}

function readoctubre() public view returns (uint256) {
  return octubre;
}

function readnoviembre() public view returns (uint256) {
  return noviembre;
}

function readdiciembre() public view returns (uint256) {
  return diciembre;
}

// Funciones para escribir valores en cada mes
function writeenero(uint256 newenero) public {
  enero = newenero;
}

function writefebrero(uint256 newfebrero) public {
  febrero = newfebrero;
}

function writemarzo(uint256 newmarzo) public {
  marzo = newmarzo;
}

function writeabril(uint256 newabril) public {
  abril = newabril;
}

function writemayo(uint256 newmayo) public {
  mayo = newmayo;
}

function writejunio(uint256 newjunio) public {
  junio = newjunio;
}

function writejulio(uint256 newjulio) public {
  julio = newjulio;
}
```

```
  function writeagosto(uint256 newagosto) public {
    agosto = newagosto;
  }

  function writeseptiembre(uint256 newseptiembre) public {
    septiembre = newseptiembre;
  }

  function writeoctubre(uint256 newoctubre) public {
    octubre = newoctubre;
  }

  function writenoviembre(uint256 newnoviembre) public {
    noviembre = newnoviembre;
  }

  function writediciembre(uint256 newdiciembre) public {
    diciembre = newdiciembre;
  }
}
```

Anexo 2: Código de división de archivos de alerta

```
from collections import defaultdict
import os

ruta_archivo = r"C:\Users\UserPC\Downloads\ALERTAS.txt"
ruta_txtsalida=r"C:\Users\UserPC\Downloads"

meses=["ene","feb","mar","abr","may","jun","jul","ago","sep","oct","nov","dic"]

def leer_alarmas(ruta_archivo):
    alarmas = []
    # Expresión regular para extraer los datos de cada alarma (si fuera necesario)
    try:
        with open(ruta_archivo, 'r', encoding='utf-8') as archivo:
            for linea in archivo:
                if linea.strip():
                    alarma2 = {}
                    segmentos = linea.split('|')
                    for segmento in segmentos:
                        clave, valor = segmento.split(':', 1)
                        clave = clave.strip().lower()
                        alarma2[clave] = valor.strip()
                # Añadir la alarma si no es repetida
                    if alarma2 in alarmas:
                        print("Alarma repetida, no se enviará nuevamente.")
                    else:
```

```
                alarmas.append(alarma2)
            else:
                print("linea vacia")

        # Eliminar el archivo original después de leerlo
        if os.path.exists(ruta_archivo):
            os.remove(ruta_archivo)
            print(f"Archivo original eliminado: {ruta_archivo}")
        else:
            print(f"El archivo {ruta_archivo} no existe.")

    except FileNotFoundError:
        print(f"El archivo {ruta_archivo} no fue encontrado.")

    return alarmas

def organizar_alarmas_por_ano_y_mes(alarmas):
    # Crear una estructura para agrupar alarmas por año y mes
    alarmas_organizadas = defaultdict(lambda: defaultdict(list))

    for alarma in alarmas:
        # Extraer la fecha
        fecha_str = alarma["fecha"]
        fecha_partes = fecha_str.split()  # Dividir la fecha en partes
        ano = fecha_partes[-3]  # Año está en la tercera posición desde el final
        mes = fecha_partes[-4]  # Mes está en la cuarta posición desde el final

        # Agregar la alarma al diccionario organizado
        alarmas_organizadas[ano][mes].append(alarma)

    # Convertir a un diccionario normal (opcional)
    return {ano: dict(meses) for ano, meses in alarmas_organizadas.items()}

def guardar_alarmas_en_txt(alarmas_organizadas, directorio_salida):
    # Crear el directorio de salida si no existe
    if not os.path.exists(directorio_salida):
        os.makedirs(directorio_salida)

    for ano, meses in alarmas_organizadas.items():
        for mes, alarmas in meses.items():
            # Crear el nombre del archivo basado en el año y el mes
            nombre_archivo = os.path.join(directorio_salida, f"alarmas_{ano}_{mes}.txt")
            with open(nombre_archivo, "w") as archivo:
                for alarma in alarmas:
                    linea = ( f"Fecha: {alarma['fecha']} | " f"Trigger: {alarma['trigger']} | " f"Host:
{alarma['host']} | " f"Duración: {alarma['duración']} | "
                    f"Mensaje: {alarma['mensaje']}\n")
                    archivo.write(linea)

            print(f"Archivo creado: {nombre_archivo}")
```

```
diccionario=(organizar_alarmas_por_ano_y_mes(leer_alarmas(ruta_archivo)))
guardar_alarmas_en_txt(diccionario,ruta_txtsalida)
```

Anexo 3: Código de lectura y envío de alertas al contrato inteligente

```
import re
from collections import defaultdict
from datetime import datetime
import time

mes=datetime.now().month
anyo=datetime.now().year
meses=["ene","feb","mar","abr","may","jun","jul","ago","sep","oct","nov","dic"]

ruta_archivo = r"C:\Users\UserPC\Downloads"
nombre_archivo="alarmas_" + str(anyo) + "_"+ meses[mes-1]+".txt"

archivo=ruta_archivo+"\\"+nombre_archivo

def convertir_a_segundos(duracion):
    minutos = 0
    segundos = 0
    # Buscar minutos
    match_minutos = re.search(r'(\d+)\s*m', duracion)
    if match_minutos:
        minutos = int(match_minutos.group(1))

    # Buscar segundos
    match_segundos = re.search(r'(\d+)\s*s', duracion)
    if match_segundos:
        segundos = int(match_segundos.group(1))
    return minutos * 60 + segundos
```

```
def leer_alarmas(ruta_archivo):
    alarmas = []
    # Expresión regular para extraer los datos de cada alarma (si fuera necesario)
    try:
        with open(ruta_archivo, 'r') as archivo:
            for linea in archivo:
                alarma2 = {}
                segmentos = linea.split('|')
                for segmento in segmentos:
                    clave, valor = segmento.split(':', 1)
                    clave = clave.strip().lower()
                    alarma2[clave] = valor.strip()

                # Añadir la alarma si no es repetida
                if alarma2 in alarmas:
                    print("Alarma repetida, no se enviará nuevamente.")
                else:
                    alarmas.append(alarma2)

    except FileNotFoundError:
        print(f"El archivo {ruta_archivo} no fue encontrado.")
    except Exception as e:
        print(f"Ocurrió un error al leer el archivo: {e}")
    return alarmas

from collections import defaultdict

def convertir_a_segundos(duracion_str):
    # Ejemplo de conversión de una duración como "9m 30s" a segundos
    partes = duracion_str.split()
    total_segundos = 0
    for parte in partes:
        if 'm' in parte:
```

```
            total_segundos += int(parte.replace('m', '')) * 60
        elif 's' in parte:
            total_segundos += int(parte.replace('s', ''))
    return total_segundos

def procesar_alarmas(alarmas, parametro):
    # Crear un defaultdict para agrupar el tiempo por mes
    tiempo_por_mes = defaultdict(int)

    for alarma in alarmas:
        if alarma['trigger'] == parametro:
            # Extraer el mes
            fecha_str = alarma["fecha"]
            fecha_partes = fecha_str.split()
            mes = fecha_partes[-4]  # El mes está en la cuarta posición desde el final

            # Convertir la duración a segundos
            duracion_str = alarma["duración"]
            duracion_segundos = convertir_a_segundos(duracion_str)

            # Sumar el tiempo al mes correspondiente
            tiempo_por_mes[mes] += duracion_segundos

    return dict(tiempo_por_mes)  # Convertir a un diccionario normal para el resultado final

alarmas_leidas = leer_alarmas(archivo)
if alarmas_leidas:
    print(f"Se detectaron {len(alarmas_leidas)} alarmas:")
    dic_alarmas=procesar_alarmas(alarmas_leidas, "Unavailable by ICMP ping")
    print(dic_alarmas)



from web3 import Web3
```

```
# Conecta a la blockchain local (por ejemplo, Ganache)
ganache_url = "http://127.0.0.1:7545"  # Cambia esto si usas otra red
web3 = Web3(Web3.HTTPProvider(ganache_url))

# Verifica la conexión
if not web3.is_connected():
    print("No se pudo conectar a la blockchain")
    exit()

# Dirección del contrato desplegado (actualízala según tu caso)
contract_address = "0x91ef4306869ee125523D57732536b0e60493422a"

# ABI del contrato (puedes copiarlo desde la compilación en Truffle o Remix)
contract_abi = [
  {
    "inputs": [
      {
        "internalType": "string",
        "name": "_mes",
        "type": "string"
      },
      {
        "internalType": "uint256",
        "name": "_valor1",
        "type": "uint256"
      },
      {
        "internalType": "uint256",
        "name": "_valor2",
        "type": "uint256"
      }
    ],
```

```
  "name": "actualizarMes",
  "outputs": [],
  "stateMutability": "nonpayable",
  "type": "function"
},
{
  "inputs": [],
  "name": "readenero",
  "outputs": [
    {
      "internalType": "uint256",
      "name": "",
      "type": "uint256"
    }
  ],
  "stateMutability": "view",
  "type": "function"
},
{
  "inputs": [],
  "name": "readfebrero",
  "outputs": [
    {
      "internalType": "uint256",
      "name": "",
      "type": "uint256"
    }
  ],
  "stateMutability": "view",
  "type": "function"
},
{
  "inputs": [],
```

```
  "name": "readmarzo",
  "outputs": [
    {
      "internalType": "uint256",
      "name": "",
      "type": "uint256"
    }
  ],
  "stateMutability": "view",
  "type": "function"
},
{
  "inputs": [],
  "name": "readabril",
  "outputs": [
    {
      "internalType": "uint256",
      "name": "",
      "type": "uint256"
    }
  ],
  "stateMutability": "view",
  "type": "function"
},
{
  "inputs": [],
  "name": "readmayo",
  "outputs": [
    {
      "internalType": "uint256",
      "name": "",
      "type": "uint256"
    }
```

```
  ],
  "stateMutability": "view",
  "type": "function"
},
{
  "inputs": [],
  "name": "readjunio",
  "outputs": [
    {
      "internalType": "uint256",
      "name": "",
      "type": "uint256"
    }
  ],
  "stateMutability": "view",
  "type": "function"
},
{
  "inputs": [],
  "name": "readjulio",
  "outputs": [
    {
      "internalType": "uint256",
      "name": "",
      "type": "uint256"
    }
  ],
  "stateMutability": "view",
  "type": "function"
},
{
  "inputs": [],
  "name": "readagosto",
```

```
  "outputs": [
    {
      "internalType": "uint256",
      "name": "",
      "type": "uint256"
    }
  ],
  "stateMutability": "view",
  "type": "function"
},
{
  "inputs": [],
  "name": "readseptiembre",
  "outputs": [
    {
      "internalType": "uint256",
      "name": "",
      "type": "uint256"
    }
  ],
  "stateMutability": "view",
  "type": "function"
},
{
  "inputs": [],
  "name": "readoctubre",
  "outputs": [
    {
      "internalType": "uint256",
      "name": "",
      "type": "uint256"
    }
  ],
```

```
  "stateMutability": "view",
  "type": "function"
},
{
  "inputs": [],
  "name": "readnoviembre",
  "outputs": [
    {
      "internalType": "uint256",
      "name": "",
      "type": "uint256"
    }
  ],
  "stateMutability": "view",
  "type": "function"
},
{
  "inputs": [],
  "name": "readdiciembre",
  "outputs": [
    {
      "internalType": "uint256",
      "name": "",
      "type": "uint256"
    }
  ],
  "stateMutability": "view",
  "type": "function"
},
{
  "inputs": [
    {
      "internalType": "uint256",
```

```
      "name": "_valor1",
      "type": "uint256"
    }
  ],
  "name": "writeenero",
  "outputs": [],
  "stateMutability": "nonpayable",
  "type": "function"
},
{
  "inputs": [
    {
      "internalType": "uint256",
      "name": "newenero",
      "type": "uint256"
    }
  ],
  "name": "writefebrero",
  "outputs": [],
  "stateMutability": "nonpayable",
  "type": "function"
},
{
  "inputs": [
    {
      "internalType": "uint256",
      "name": "newmarzo",
      "type": "uint256"
    }
  ],
  "name": "writemarzo",
  "outputs": [],
  "stateMutability": "nonpayable",
```

```
  "type": "function"
},
{
  "inputs": [
    {
      "internalType": "uint256",
      "name": "newabril",
      "type": "uint256"
    }
  ],
  "name": "writeabril",
  "outputs": [],
  "stateMutability": "nonpayable",
  "type": "function"
},
{
  "inputs": [
    {
      "internalType": "uint256",
      "name": "newmayo",
      "type": "uint256"
    }
  ],
  "name": "writemayo",
  "outputs": [],
  "stateMutability": "nonpayable",
  "type": "function"
},
{
  "inputs": [
    {
      "internalType": "uint256",
      "name": "newjunio",
```

```
      "type": "uint256"
    }
  ],
  "name": "writejunio",
  "outputs": [],
  "stateMutability": "nonpayable",
  "type": "function"
},
{
  "inputs": [
    {
      "internalType": "uint256",
      "name": "newjulio",
      "type": "uint256"
    }
  ],
  "name": "writejulio",
  "outputs": [],
  "stateMutability": "nonpayable",
  "type": "function"
},
{
  "inputs": [
    {
      "internalType": "uint256",
      "name": "newagosto",
      "type": "uint256"
    }
  ],
  "name": "writeagosto",
  "outputs": [],
  "stateMutability": "nonpayable",
  "type": "function"
```

```
},
{
  "inputs": [
    {
      "internalType": "uint256",
      "name": "newseptiembre",
      "type": "uint256"
    }
  ],
  "name": "writeseptiembre",
  "outputs": [],
  "stateMutability": "nonpayable",
  "type": "function"
},
{
  "inputs": [
    {
      "internalType": "uint256",
      "name": "newoctubre",
      "type": "uint256"
    }
  ],
  "name": "writeoctubre",
  "outputs": [],
  "stateMutability": "nonpayable",
  "type": "function"
},
{
  "inputs": [
    {
      "internalType": "uint256",
      "name": "newnoviembre",
      "type": "uint256"
```

```
      }
    ],
    "name": "writenoviembre",
    "outputs": [],
    "stateMutability": "nonpayable",
    "type": "function"
  },
  {
    "inputs": [
      {
        "internalType": "uint256",
        "name": "newdiciembre",
        "type": "uint256"
      }
    ],
    "name": "writediciembre",
    "outputs": [],
    "stateMutability": "nonpayable",
    "type": "function"
  }
]

# Instancia del contrato
contract = web3.eth.contract(address=contract_address, abi=contract_abi)
# Dirección y clave privada de una cuenta de Ganache (usa tu propia cuenta)
account_address = "0x0028F877bb7d7490e99aF96964847F1075bd28EB"
private_key = "0xa70892520a886baf62f80d225d004a853124719a9321294a01166a41d42c816a"


# Asumiendo que ya tienes una instancia de Web3 (web3) configurada y el contrato cargado
def actualizar_valores_del_mes(mes, valor1):
    try:
```

```
# Seleccionar la función de escritura según el mes
if mes.lower() == "ene":
    func = contract.functions.writeenero(valor1)
elif mes.lower() == "feb":
    func = contract.functions.writefebrero(valor1)
elif mes.lower() == "mar":
    func = contract.functions.writemarzo(valor1)
elif mes.lower() == "abr":
    func = contract.functions.writeabril(valor1)
elif mes.lower() == "may":
    func = contract.functions.writemayo(valor1)
elif mes.lower() == "jun":
    func = contract.functions.writejunio(valor1)
elif mes.lower() == "jul":
    func = contract.functions.writejulio(valor1)
elif mes.lower() == "ago":
    func = contract.functions.writeagosto(valor1)
elif mes.lower() == "sep":
    func = contract.functions.writeseptiembre(valor1)
elif mes.lower() == "oct":
    func = contract.functions.writeoctubre(valor1)
elif mes.lower() == "nov":
    func = contract.functions.writenoviembre(valor1)
elif mes.lower() == "dic":
    func = contract.functions.writediciembre(valor1)
else:
    raise ValueError("Mes no válido.")

# Construir la transacción para llamar a la función seleccionada
transaction = func.build_transaction({
    "chainId": 1337,  # Chain ID de Ganache (ajustar según tu red)
    "gas": 200000,  # Límite de gas
    "gasPrice": web3.to_wei("10", "gwei"),  # Precio del gas
```

```
        "nonce": web3.eth.get_transaction_count(account_address),  # Nonce de la cuenta
    })

    # Firmar la transacción
    signed_txn = web3.eth.account.sign_transaction(transaction, private_key)

    # Enviar la transacción
    tx_hash = web3.eth.send_raw_transaction(signed_txn.raw_transaction)
    print(f"Transacción enviada con hash: {tx_hash.hex()}")

    # Esperar a que la transacción sea confirmada
    tx_receipt = web3.eth.wait_for_transaction_receipt(tx_hash)
    print(f"Transacción confirmada: {tx_receipt}")

  except ValueError as ve:
    print(f"Error de validación: {ve}")
  except Exception as e:
    print(f"Error al interactuar con el contrato: {e}")

actualizar_valores_del_mes(meses[mes-1],dic_alarmas[meses[mes-1]])
```

Anexo 4: Código de lectura del contrato inteligente

```
def leer_valores_del_mes(mes):
  try:
    # Verificar que el mes sea una cadena de texto válida
    if not isinstance(mes, str):
      raise ValueError("El mes debe ser una cadena de texto.")

    # Seleccionar la función de lectura según el mes
    if mes.lower() == "enero":
      value = contract.functions.readenero().call()
    elif mes.lower() == "febrero":
```

```
            value = contract.functions.readfebrero().call()
        elif mes.lower() == "marzo":
            value = contract.functions.readmarzo().call()
        elif mes.lower() == "abril":
            value = contract.functions.readabril().call()
        elif mes.lower() == "mayo":
            value = contract.functions.readmayo().call()
        elif mes.lower() == "junio":
            value = contract.functions.readjunio().call()
        elif mes.lower() == "julio":
            value = contract.functions.readjulio().call()
        elif mes.lower() == "agosto":
            value = contract.functions.readagosto().call()
        elif mes.lower() == "septiembre":
            value = contract.functions.readseptiembre().call()
        elif mes.lower() == "octubre":
            value = contract.functions.readoctubre().call()
        elif mes.lower() == "noviembre":
            value = contract.functions.readnoviembre().call()
        elif mes.lower() == "diciembre":
            value = contract.functions.readdiciembre().call()
        else:
            raise ValueError("Mes no válido.")

        print(f"Valor de {mes}: {value}")
    except ValueError as ve:
        print(f"Error de validación: {ve}")
    except Exception as e:
        print(f"Error al leer los valores del contrato: {e}")

# Llamar a la función para leer el valor
leer_valores_del_mes("enero")
```

Anexo 5: Código de las funciones de Node-RED

```
// Supongamos que msg.payload contiene el resultado del exec
let execOutput = msg.payload;

// Verificamos si es un string
if (typeof execOutput !== 'string') {
   return { payload: "El resultado del exec no es un string válido" };
}

// Reemplazamos las comillas simples por comillas dobles para hacer un JSON válido
execOutput = execOutput.replace(/'/g, "");

// Intentamos convertir el string a un objeto JSON
let data;
try {
   data = JSON.parse(execOutput); // Ahora debería ser un objeto válido
} catch (error) {
   return { payload: "Error al parsear el resultado del exec: " + error.message };
}

// Creamos un array para almacenar todos los puntos de datos
let points = [];

// Recorremos todos los meses del objeto

let value = data["enero"];

// Aseguramos que 'tags' contenga solo un valor simple y no un objeto complejo
let point = {
porcentaje_de_disponibilidad : value,       // 'fields' con valor simple
};
```

```
// Añadimos el punto al array
points.push(point);

// Asignamos el array de puntos como msg.payload
msg.payload = points;

return msg;
```

Anexo 6: Código para el cálculo de las compensaciones por falta de disponibilidad del servicio.

```
from web3 import Web3
from datetime import datetime

def calcular_descuento(porcentual):
  if porcentual >= 99.9:
    return 0.0  # Nivel óptimo - Sin compensación.
  elif 99.5 <= porcentual <= 99.89:
    return 0.05  # 5% de descuento en la factura mensual.
  elif 99.0 <= porcentual <= 99.49:
    return 0.10  # 10% de descuento en la factura mensual.
  elif 95.0 <= porcentual <= 98.99:
    return 0.20  # 20% de descuento en la factura mensual.
  else:
    return 0.50  # 50% de descuento o reembolso proporcional del tiempo de inactividad.

# Conecta a la blockchain local (por ejemplo, Ganache)
meses=["enero","febrero","marzo","abril","mayo","junio","julio","agosto","septiembre","octubre
","noviembre","diciembre"]
ganache_url = "http://192.168.100.94:7545"  # Cambia esto si usas otra red
web3 = Web3(Web3.HTTPProvider(ganache_url))
```

```
# Verifica la conexión
if not web3.is_connected():
    print("No se pudo conectar a la blockchain")
    exit()

# Dirección del contrato desplegado (actualízala según tu caso)
contract_address = "0x91ef4306869ee125523D57732536b0e60493422a"

# ABI del contrato (puedes copiarlo desde la compilación en Truffle o Remix)
contract_abi = [
  {
    "inputs": [
      {
        "internalType": "string",
        "name": "_mes",
        "type": "string"
      },
      {
        "internalType": "uint256",
        "name": "_valor1",
        "type": "uint256"
      },
      {
        "internalType": "uint256",
        "name": "_valor2",
        "type": "uint256"
      }
    ],
    "name": "actualizarMes",
    "outputs": [],
    "stateMutability": "nonpayable",
    "type": "function"
  },
```

```
{
  "inputs": [],
  "name": "readenero",
  "outputs": [
    {
      "internalType": "uint256",
      "name": "",
      "type": "uint256"
    }
  ],
  "stateMutability": "view",
  "type": "function"
},
{
  "inputs": [],
  "name": "readfebrero",
  "outputs": [
    {
      "internalType": "uint256",
      "name": "",
      "type": "uint256"
    }
  ],
  "stateMutability": "view",
  "type": "function"
},
{
  "inputs": [],
  "name": "readmarzo",
  "outputs": [
    {
      "internalType": "uint256",
      "name": "",
```

```
      "type": "uint256"
    }
  ],
  "stateMutability": "view",
  "type": "function"
},
{
  "inputs": [],
  "name": "readabril",
  "outputs": [
    {
      "internalType": "uint256",
      "name": "",
      "type": "uint256"
    }
  ],
  "stateMutability": "view",
  "type": "function"
},
{
  "inputs": [],
  "name": "readmayo",
  "outputs": [
    {
      "internalType": "uint256",
      "name": "",
      "type": "uint256"
    }
  ],
  "stateMutability": "view",
  "type": "function"
},
{
```

```
  "inputs": [],
  "name": "readjunio",
  "outputs": [
    {
      "internalType": "uint256",
      "name": "",
      "type": "uint256"
    }
  ],
  "stateMutability": "view",
  "type": "function"
},
{
  "inputs": [],
  "name": "readjulio",
  "outputs": [
    {
      "internalType": "uint256",
      "name": "",
      "type": "uint256"
    }
  ],
  "stateMutability": "view",
  "type": "function"
},
{
  "inputs": [],
  "name": "readagosto",
  "outputs": [
    {
      "internalType": "uint256",
      "name": "",
      "type": "uint256"
```

```
    }
  ],
  "stateMutability": "view",
  "type": "function"
},
{
  "inputs": [],
  "name": "readseptiembre",
  "outputs": [
    {
      "internalType": "uint256",
      "name": "",
      "type": "uint256"
    }
  ],
  "stateMutability": "view",
  "type": "function"
},
{
  "inputs": [],
  "name": "readoctubre",
  "outputs": [
    {
      "internalType": "uint256",
      "name": "",
      "type": "uint256"
    }
  ],
  "stateMutability": "view",
  "type": "function"
},
{
  "inputs": [],
```

```
  "name": "readnoviembre",
  "outputs": [
    {
      "internalType": "uint256",
      "name": "",
      "type": "uint256"
    }
  ],
  "stateMutability": "view",
  "type": "function"
},
{
  "inputs": [],
  "name": "readdiciembre",
  "outputs": [
    {
      "internalType": "uint256",
      "name": "",
      "type": "uint256"
    }
  ],
  "stateMutability": "view",
  "type": "function"
},
{
  "inputs": [
    {
      "internalType": "uint256",
      "name": "_valor1",
      "type": "uint256"
    }
  ],
  "name": "writeenero",
```

```
  "outputs": [],
  "stateMutability": "nonpayable",
  "type": "function"
},
{
  "inputs": [
    {
      "internalType": "uint256",
      "name": "newenero",
      "type": "uint256"
    }
  ],
  "name": "writefebrero",
  "outputs": [],
  "stateMutability": "nonpayable",
  "type": "function"
},
{
  "inputs": [
    {
      "internalType": "uint256",
      "name": "newmarzo",
      "type": "uint256"
    }
  ],
  "name": "writemarzo",
  "outputs": [],
  "stateMutability": "nonpayable",
  "type": "function"
},
{
  "inputs": [
    {
```

```
      "internalType": "uint256",
      "name": "newabril",
      "type": "uint256"
    }
  ],
  "name": "writeabril",
  "outputs": [],
  "stateMutability": "nonpayable",
  "type": "function"
},
{
  "inputs": [
    {
      "internalType": "uint256",
      "name": "newmayo",
      "type": "uint256"
    }
  ],
  "name": "writemayo",
  "outputs": [],
  "stateMutability": "nonpayable",
  "type": "function"
},
{
  "inputs": [
    {
      "internalType": "uint256",
      "name": "newjunio",
      "type": "uint256"
    }
  ],
  "name": "writejunio",
  "outputs": [],
```

```
  "stateMutability": "nonpayable",
  "type": "function"
},
{
  "inputs": [
    {
      "internalType": "uint256",
      "name": "newjulio",
      "type": "uint256"
    }
  ],
  "name": "writejulio",
  "outputs": [],
  "stateMutability": "nonpayable",
  "type": "function"
},
{
  "inputs": [
    {
      "internalType": "uint256",
      "name": "newagosto",
      "type": "uint256"
    }
  ],
  "name": "writeagosto",
  "outputs": [],
  "stateMutability": "nonpayable",
  "type": "function"
},
{
  "inputs": [
    {
      "internalType": "uint256",
```

```
      "name": "newseptiembre",
      "type": "uint256"
    }
  ],
  "name": "writeseptiembre",
  "outputs": [],
  "stateMutability": "nonpayable",
  "type": "function"
},
{
  "inputs": [
    {
      "internalType": "uint256",
      "name": "newoctubre",
      "type": "uint256"
    }
  ],
  "name": "writeoctubre",
  "outputs": [],
  "stateMutability": "nonpayable",
  "type": "function"
},
{
  "inputs": [
    {
      "internalType": "uint256",
      "name": "newnoviembre",
      "type": "uint256"
    }
  ],
  "name": "writenoviembre",
  "outputs": [],
  "stateMutability": "nonpayable",
```

```
    "type": "function"
  },
  {
    "inputs": [
      {
        "internalType": "uint256",
        "name": "newdiciembre",
        "type": "uint256"
      }
    ],
    "name": "writediciembre",
    "outputs": [],
    "stateMutability": "nonpayable",
    "type": "function"
  }
]

# Instancia del contrato
contract = web3.eth.contract(address=contract_address, abi=contract_abi)
# Dirección y clave privada de una cuenta de Ganache (usa tu propia cuenta)
account_address = "0x0028F877bb7d7490e99aF96964847F1075bd28EB"
private_key = "0xa70892520a886baf62f80d225d004a853124719a9321294a01166a41d42c816a"


def segundos_por_mes(anio):
  # Días en cada mes en un año no bisiesto
  dias_por_mes = {
    "enero": 31, "febrero": 28, "marzo": 31, "abril": 30,
    "mayo": 31, "junio": 30, "julio": 31, "agosto": 31,
    "septiembre": 30, "octubre": 31, "noviembre": 30, "diciembre": 31
  }

  # Comprobar si el año es bisiesto
```

```
  if (anio % 4 == 0 and anio % 100 != 0) or (anio % 400 == 0):
    dias_por_mes["febrero"] = 29

  # Calcular segundos para cada mes
  segundos_por_mes = {mes: dias * 24 * 60 * 60 for mes, dias in dias_por_mes.items()}
  return segundos_por_mes

def leer_valores_del_mes(mes):
    try:
        # Verificar que el mes sea una cadena de texto válida
        if not isinstance(mes, str):
            raise ValueError("El mes debe ser una cadena de texto.")

        # Seleccionar la función de lectura según el mes
        if mes.lower() == "enero":
            value = contract.functions.readenero().call()
        elif mes.lower() == "febrero":
            value = contract.functions.readfebrero().call()
        elif mes.lower() == "marzo":
            value = contract.functions.readmarzo().call()
        elif mes.lower() == "abril":
            value = contract.functions.readabril().call()
        elif mes.lower() == "mayo":
            value = contract.functions.readmayo().call()
        elif mes.lower() == "junio":
            value = contract.functions.readjunio().call()
        elif mes.lower() == "julio":
            value = contract.functions.readjulio().call()
        elif mes.lower() == "agosto":
            value = contract.functions.readagosto().call()
        elif mes.lower() == "septiembre":
            value = contract.functions.readseptiembre().call()
        elif mes.lower() == "octubre":
```

```
            value = contract.functions.readoctubre().call()
        elif mes.lower() == "noviembre":
            value = contract.functions.readnoviembre().call()
        elif mes.lower() == "diciembre":
            value = contract.functions.readdiciembre().call()
        else:
            raise ValueError("Mes no válido.")

    except ValueError as ve:
        print(f"Error de validación: {ve}")
    except Exception as e:
        print(f"Error al leer los valores del contrato: {e}")
    return value

# Llamar a la función para leer el valor
disponibilidad={}
for mes in meses:
  tiempo_mensual=segundos_por_mes(datetime.now().year)
  porcentual=round(100-leer_valores_del_mes(mes)*100/tiempo_mensual[mes],3)
  valor=calcular_descuento(porcentual)*500
  disponibilidad[mes]=valor
print(disponibilidad)
```