Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingeniería de Sistemas Informáticos

Monitorización de sistemas IoT en un entorno DevOps

Proyecto Fin de Máster en Software de Sistemas Distribuidos y Empotrados

Curso académico 2022-2023

Tutor:
Jessica Díaz Fernández

Autor:
Alfonso Díez Ramírez

Índice

Contenido

Índice de figuras

Glosario de términos

A

API
: (Application Programming Interface) Conjunto de funcionalidades que ofrece como servicio para ser utilizado en otras aplicaciones · 18, 36

B

backend
: El backend es la parte de un sistema informático que se encarga de la lógica y el procesamiento de datos. · 28

C

CNCF
: La Cloud Native Computing Foundation (CNCF) es una organización sin ánimo de lucro que impulsa la adopción y desarrollo de tecnologías nativas de la nube, como contenedores y orquestadores, promoviendo estándares abiertos y fomentando la colaboración en la comunidad de computación en la nube. · 19

código abierto
: El código abierto se refiere a software cuyo código fuente es accesible y libremente disponible para que cualquier persona lo pueda ver, modificar y distribuir de manera abierta y colaborativa. · 1, 12, 15, 16, 19, 20, 23, 33

D

dispositivo
: Un dispositivo de IoT (Internet de las cosas) es un objeto físico con sensores y capacidades de conectividad que recopila y transmite datos a través de Internet. Estos dispositivos pueden interactuar con su entorno y realizar acciones automatizadas basadas en los datos recopilados. · 45, 46, 47

E

Edge
: Edge en IoT se refiere a la computación y procesamiento de datos en el extremo de la red, cerca de los dispositivos y sensores, en lugar de enviar todos los datos a la nube. Permite un procesamiento y análisis más rápido, reduciendo la latencia y optimizando el uso de ancho de banda. · VI, 4

ELK
: Elasticsearch, Logstash y Kibana, por sus siglas · VI, VII, 2, 3, 4, 6, 7, 15, 28, 29, 31, 32, 33

F

framework
: Un framework es una estructura de software que proporciona herramientas y bibliotecas predefinidas para facilitar el desarrollo de aplicaciones al ofrecer soluciones comunes y abstracciones reutilizables. · 33

frontal web
: Un frontal web, también conocido como frontend, es la parte de una aplicación web que se muestra y con la que los usuarios interactúan directamente. Incluye la interfaz de usuario, el diseño, la navegación y la interacción en el lado del cliente. · 36

I

IoT
: Internet of Things · 5, VI, VII, 1, 2, 3, 4, 7, 8, 9, 11, 14, 15, 20, 21, 23, 25, 26, 28, 29, 33

L

logs
: Los logs son registros o registros de eventos generados por sistemas informáticos, aplicaciones o dispositivos, que proporcionan información detallada sobre operaciones, errores, acciones y eventos relevantes para el monitoreo y solución de problemas. · VII, 2, 3, 8, 9, 15, 19, 22, 25, 26, 27, 30, 32, 33

M

máquina virtual
: Una máquina virtual es un entorno virtualizado que simula un sistema informático completo, incluyendo hardware y software, dentro de un sistema físico. Permite ejecutar múltiples sistemas operativos y aplicaciones de forma aislada y simultánea en un mismo equipo físico, lo que brinda flexibilidad, eficiencia y

seguridad en el despliegue de sistemas y aplicaciones. · 3, 6, 8, 9, 13, 14, 25, 27, 30, 36, 37

metodología
Una metodología es un enfoque estructurado y sistemático para abordar una tarea o proceso, que incluye principios, prácticas y técnicas específicas para lograr resultados consistentes y eficientes. · III, VI, 2, 5, 6, 11, 12

MQTT
Message Queing Telemetry Transport, Protocolo de comunicación máquina a máquina mediante el envío de mensajes. · 3, 4, 8, 23, 24, 25, 26, 27, 28, 29, 36

S

script
Un script es un conjunto de instrucciones o comandos escritos en un lenguaje de programación que se utilizan para automatizar tareas o realizar acciones específicas en un sistema o programa · 9, 36, 37, 39

servidor
Computadora que provee servicios a otros dispositivos en una red, como Internet. Responde a solicitudes y gestiona recursos. · 3, 22, 23, 24, 28, 36, 45

stack
conjunto de tecnologías, herramientas o componentes que se utilizan de manera conjunta para desarrollar o implementar una solución informática · VI, VII, 2, 3, 4, 15, 28, 31, 32

V

virtualización
La virtualización es una tecnología que permite crear y ejecutar instancias virtuales de sistemas operativos, servidores, redes y otros recursos informáticos. Permite maximizar la utilización de hardware, simplificar la administración y proporcionar mayor flexibilidad y escalabilidad en entornos de TI. · VI, 2, 3, 4, 12, 13

Resumen

El presente proyecto se enfoca en la aplicación de metodología DevOps en el ámbito de IoT mediante el uso de Vagrant en un entorno simulado. El objetivo principal es desplegar el stack ELK (Elasticsearch, Logstash y Kibana) y un clúster de servidores de K3S, cuyos pods emularán dispositivos del Edge. La implementación de este proyecto busca aprovechar la virtualización y el uso de contenedores para lograr una gestión eficiente y escalable de los entornos de desarrollo en el contexto de IoT.

El stack ELK es ampliamente reconocido por su capacidad para recopilar, procesar y visualizar grandes volúmenes de registros y datos. Al desplegar este stack en un entorno simulado de IoT mediante Vagrant, se brinda a los equipos de desarrollo y profesionales de IoT la oportunidad de explorar y evaluar su funcionalidad y rendimiento en un entorno controlado. Esto les permitirá adquirir una comprensión más profunda de las capacidades y limitaciones del stack ELK en el contexto de IoT, así como desarrollar estrategias efectivas para la gestión y análisis de datos a gran escala.

Por otro lado, el clúster de servidores de K3S es una distribución ligera de Kubernetes que facilita la creación y gestión de clústeres de contenedores. En este proyecto, el clúster de K3S se utilizará para simular dispositivos IoT en el entorno virtualizado. Al emular dispositivos a través del clúster de K3S, se podrá evaluar cómo interactúan y se comunican estos dispositivos en un entorno escalable y eficiente. Esta simulación permitirá a los equipos de desarrollo y profesionales de IoT analizar y optimizar el rendimiento de las aplicaciones y servicios IoT, así como explorar estrategias de escalabilidad y tolerancia a fallos.

El uso de Vagrant en este proyecto tiene como objetivo principal lograr una configuración consistente y reproducible de los entornos de desarrollo. Vagrant facilita la definición y compartición de archivos de configuración que describen el entorno deseado, asegurando que todos los miembros del equipo tengan una base común y evitando problemas causados por configuraciones inconsistentes. Esto ahorra tiempo y reduce la posibilidad de errores en la configuración de los entornos.

Además, la metodología DevOps se aplica de manera integral en este proyecto. DevOps se basa en la colaboración estrecha entre los equipos de desarrollo y operaciones para acelerar la entrega de software, mejorar la calidad y garantizar la estabilidad del sistema. En el contexto de IoT, la aplicación de DevOps es fundamental para gestionar la complejidad de los entornos distribuidos y heterogéneos. La virtualización proporcionada por Vagrant y la gestión de contenedores ofrecida por el clúster de K3S son pilares clave para la implementación exitosa de DevOps en IoT.

Abstract

This project focuses on the application of DevOps methodology in the Internet of Things (IoT) domain using Vagrant in a simulated environment. The main objective is to deploy the ELK stack (Elasticsearch, Logstash, and Kibana) and a K3S server cluster that will emulate devices. The implementation of this project aims to leverage virtualization and containerization to achieve efficient and scalable management of development environments in the context of IoT.

The ELK stack is widely recognized for its ability to collect, process, and visualize large volumes of logs and data. By deploying this stack in a simulated IoT environment using Vagrant, development teams and IoT professionals are provided with an opportunity to explore and evaluate its functionality and performance in a controlled environment. This will enable them to gain a deeper understanding of the capabilities and limitations of the ELK stack in the context of IoT and develop effective strategies for managing and analyzing large-scale data.

On the other hand, the K3S server cluster is a lightweight distribution of Kubernetes that facilitates the creation and management of container clusters. In this project, the K3S cluster will be used to simulate IoT devices in the virtualized environment. By emulating devices through the K3S cluster, it will be possible to assess how these devices interact and communicate in a scalable and efficient environment. This simulation will allow development teams and IoT professionals to analyze and optimize the performance of IoT applications and services, as well as explore scalability and fault tolerance strategies.

The use of Vagrant in this project aims to achieve consistent and reproducible configuration of development environments. Vagrant facilitates the definition and sharing of configuration files that describe the desired environment, ensuring that all team members have a common foundation and avoiding issues caused by inconsistent configurations. This saves time and reduces the likelihood of errors in environment configuration.

Furthermore, DevOps methodology is applied comprehensively in this project. DevOps is based on close collaboration between development and operations teams to accelerate software delivery, improve quality, and ensure system stability. In the context of IoT, the application of DevOps is crucial for managing the complexity of distributed and heterogeneous environments. The virtualization provided by Vagrant and container management offered by the K3S cluster are key pillars for the successful implementation of DevOps in IoT.

1. Introducción

1.1. Contexto

El contexto actual de desarrollo de software se caracteriza por la creciente demanda de implementaciones rápidas, estables y escalables, lo cual ha impulsado la adopción de metodologías y prácticas que permitan abordar estos desafíos. En este sentido, los entornos DevOps han surgido como una aproximación eficaz para mejorar las capacidades de desarrollo, integración y despliegue continuo de aplicaciones.

Adicionalmente, el Internet de las cosas (IoT) es una de las tecnologías de vanguardia que está transformando varios aspectos de nuestra vida cotidiana, así como también el mundo empresarial e industrial. A medida que avanzamos hacia un mundo más interconectado, el IoT continúa creciendo y evolucionando, con nuevas aplicaciones y desafíos emergentes. Esta creciente conectividad y la proliferación de dispositivos IoT plantea la necesidad de implementar un entorno DevOps para garantizar el desarrollo, la implementación y la operación efectiva de los sistemas IoT.

La implementación exitosa de un entorno DevOps requiere el cumplimiento de varios requisitos fundamentales. En primer lugar, se debe establecer una estrecha colaboración entre los equipos de desarrollo y operaciones, fomentando una comunicación fluida y una visión compartida del proyecto. Esta colaboración es esencial para garantizar que los requisitos y las necesidades del sistema IoT se comprendan y se traduzcan adecuadamente en el proceso de desarrollo.

Además, es esencial contar con herramientas y procesos automatizados que permitan la integración y entrega continua de software, agilizando el ciclo de vida del desarrollo y minimizando los tiempos de entrega. Esto es especialmente relevante en el contexto del IoT, donde las actualizaciones y mejoras de software deben ser implementadas de manera rápida y eficiente en un gran número de dispositivos distribuidos.

Entre los puntos fuertes de la aproximación DevOps se encuentra la mejora en la calidad del software, ya que la automatización de pruebas y la integración continua permiten identificar y corregir errores de manera temprana. Asimismo, la capacidad de respuesta ante cambios y la rápida adaptación a nuevas necesidades del negocio son características destacadas de estos entornos. Al tener una mayor visibilidad y control sobre el ciclo de vida de la aplicación, los equipos pueden tomar decisiones informadas y reducir el time to market.

Sin embargo, también existen algunos puntos débiles en esta aproximación. La complejidad y la curva de aprendizaje asociadas con la adopción de herramientas y prácticas DevOps pueden ser un desafío para algunos equipos. Además, la falta de una cultura colaborativa y resistencia al cambio pueden obstaculizar la implementación exitosa de estos entornos. Es fundamental fomentar una mentalidad de mejora continua y promover la colaboración entre los diferentes roles involucrados.

En este contexto, Vagrant ha demostrado ser una herramienta valiosa dentro de los entornos DevOps. Vagrant es una herramienta de código abierto que permite la creación y gestión de entornos de desarrollo virtualizados de manera reproducible. Al

proporcionar una configuración declarativa y fácil de usar, Vagrant permite a los equipos establecer entornos de desarrollo consistentes y portátiles, evitando problemas causados por diferencias en la configuración del entorno.

Además, Vagrant se integra perfectamente con otras herramientas y tecnologías utilizadas en los entornos DevOps, como Ansible o Docker. Esto permite la automatización de tareas de aprovisionamiento y configuración, así como la gestión de contenedores, facilitando la creación de entornos de desarrollo completos y listos para ser desplegados en cualquier infraestructura.

Por otro lado, la monitorización de logs se ha convertido en una práctica esencial. Los logs son registros que contienen información valiosa sobre el funcionamiento de un sistema, incluyendo errores, eventos y actividades relevantes. Los sistemas de monitorización de logs permiten recopilar, analizar y visualizar estos registros de manera centralizada, proporcionando una visión completa y en tiempo real del estado y rendimiento del sistema. Esto facilita la detección temprana de problemas, la toma de decisiones informadas y la optimización de la infraestructura. Algo que sirve de apoyo dentro de la metodología DevOps.

En conclusión, los entornos de trabajo con metodologías DevOps han surgido como una respuesta efectiva a los desafíos actuales en el desarrollo de software. Al cumplir con requisitos clave como la colaboración, la automatización y la entrega continua, los entornos DevOps mejoran las capacidades de desarrollo y permiten una mayor eficiencia en la entrega de software. Vagrant, como herramienta de virtualización, desempeña un papel importante en esta aproximación, proporcionando entornos de desarrollo consistentes y portátiles que facilitan la adopción de prácticas DevOps y la implementación exitosa de proyectos, adicionalmente, la monitorización de logs se ha vuelto fundamental para garantizar la fiabilidad y eficiencia de los sistemas en entornos empresariales.

1.2. Objetivo

Este proyecto tiene como objetivo aplicar metodología DevOps en el ámbito de Internet de las cosas (IoT) mediante la utilización de Vagrant en un entorno simulado. El despliegue del stack ELK y el clúster de servidores de K3S permitirá a los equipos de desarrollo y profesionales de IoT explorar y evaluar el rendimiento de estas tecnologías en un entorno controlado.

El resultado principal será un conjunto de scripts y piezas de código declarativo que permitirán desplegar y configurar un entorno de desarrollo cuyas características sean prácticamente idénticas a un entorno de producción con el objetivo de reducir el número de fallos introducidos por diferencias entre entornos, agilizar la velocidad de inserción de personal al equipo de desarrollo y reducir los tiempos de retraso asociados a cambios en la infraestructura o redes subyacentes, permitiendo que cualquier variación en estas se sustituya en el código y se aplique a todos los entornos de desarrollo en pocos minutos.

Otro de los objetivos es que la arquitectura de sistemas que despliegue estos scripts sea capaz de monitorizar un entorno IoT con miles de dispositivos, por lo que debe ser altamente escalable.

Para ello, el objetivo global es desplegar la siguiente arquitectura automáticamente:

Una de las partes a de la arquitectura objetivo es el stack ELK (Elasticsearch, Logstash y Kibana). Elasticsearch proporcionará una plataforma robusta para el almacenamiento y búsqueda de los registros generados por los dispositivos IoT simulados. Logstash será utilizado para la ingestión y procesamiento de estos registros, permitiendo su transformación y enriquecimiento antes de ser almacenados en Elasticsearch. Kibana se utilizará como herramienta de visualización y análisis para extraer información valiosa a partir de los datos recopilados. El despliegue exitoso del stack ELK en el entorno simulado será un hito clave en el proyecto.

Además, se pretende desplegar un cluster de K3S para simular los dispositivos IoT. K3S es una distribución ligera de Kubernetes que facilita la creación y gestión de contenedores. Los contenedores desplegados en este clúster representarán los dispositivos IoT, generando y enviando logs a través del protocolo MQTT. Para ello, se utilizará Mosquitto como servidor MQTT, que actuará como intermediario en la comunicación entre los dispositivos y el resto del sistema.

Para capturar y enviar los registros de MQTT a Logstash, se desplegará un agente Filebeat en la máquina virtual de MQTT. Filebeat es un componente de la suite de Elastic que permite recopilar, enviar y procesar registros de manera eficiente. El agente Filebeat estará configurado para leer los mensajes de los tópicos de MQTT y enviarlos a Logstash a través de una conexión TCP. Esto permitirá una integración fluida entre Mosquitto, Filebeat y Logstash, asegurando que los registros generados por los dispositivos IoT sean enviados y procesados de manera adecuada en el stack ELK.

En resumen, los objetivos principales de este proyecto de fin de máster son desplegar un entorno utilizando Vagrant que contenga el stack ELK, un cluster de K3S con contenedores que simulan ser dispositivos IoT y establecer una arquitectura que sirva de punto de unión entre ambas, permitiendo la recopilación, procesamiento y análisis de los registros generados por estos dispositivos. La integración de Mosquitto, Filebeat y Logstash asegurará la captura y transferencia eficiente de los logs, desacoplando lógica y temporalmente los sistemas, proporcionando una solución completa y escalable para la gestión de logs en un entorno de IoT simulado.

1.3. Justificación

La aplicación de enfoques como la virtualización, el uso de contenedores y la implementación de metodologías DevOps desempeñan un papel relevante en el campo de Internet de las cosas (IoT). En un entorno de IoT, donde la conectividad y la gestión de dispositivos distribuidos son fundamentales, estos enfoques proporcionan numerosos beneficios que impulsan la eficiencia y la escalabilidad del desarrollo de soluciones IoT.

La virtualización permite la creación de entornos aislados y reproducibles, lo que resulta especialmente valioso en el desarrollo de sistemas IoT. Mediante el uso de herramientas como Vagrant, es posible definir y compartir configuraciones que describen el entorno deseado, lo que garantiza una configuración consistente para todos los miembros del equipo. Esto reduce la posibilidad de errores y conflictos causados por configuraciones inconsistentes, agilizando así el proceso de desarrollo y facilitando la colaboración.

El uso de contenedores, como los proporcionados por tecnologías como Docker, también tiene un impacto significativo en el desarrollo de soluciones IoT. Los contenedores permiten la encapsulación de aplicaciones y sus dependencias en entidades ligeras y portátiles, lo que facilita su despliegue y ejecución en diferentes entornos. Al utilizar un cluster de k3s para desplegar contenedores que simulan dispositivos del Edge de IoT, se logra una mayor flexibilidad y escalabilidad en el manejo de estos dispositivos virtuales. Esto permite una prueba más eficiente y precisa de las soluciones IoT, así como la simulación de escenarios complejos y la evaluación de la interoperabilidad entre dispositivos.

La aplicación de metodologías DevOps en el contexto de IoT también es esencial para abordar los desafíos específicos de este campo. DevOps fomenta la colaboración estrecha entre los equipos de desarrollo y operaciones, lo que se traduce en una entrega más rápida y confiable de soluciones IoT. La automatización de los procesos de desarrollo, pruebas, implementación y monitoreo proporciona una mayor eficiencia y reduce los errores humanos. Además, la implementación de estrategias de entrega continua y monitoreo en tiempo real permite una mayor agilidad y capacidad de respuesta a los cambios en el entorno IoT.

La utilización de Mosquitto como un broker de mensajes MQTT para comunicar los dispositivos del Edge de IoT con el stack ELK ofrece una arquitectura flexible y desacoplada. Al separar la comunicación de los dispositivos y el procesamiento y almacenamiento de los datos en ELK, se logra una mayor escalabilidad y modularidad en el sistema. Esto permite la implementación de soluciones IoT a gran escala, así como la incorporación de nuevos dispositivos y servicios de forma independiente.

En conclusión, la combinación de la virtualización, el uso de contenedores y la aplicación de metodologías DevOps en el desarrollo de soluciones IoT ofrece numerosos beneficios que son de interés y que se abordarán en este proyecto.

2. Metodología

2.1. Descripción

Para el desarrollo del proyecto, se ha utilizada una metodología Iterativa incremental.

La metodología iterativa incremental es un enfoque de desarrollo de software que se basa en la creación de incrementos o versiones funcionales del producto a lo largo del tiempo. En lugar de desarrollar el software completo de una sola vez, se divide el proyecto en pequeñas iteraciones o ciclos de desarrollo.

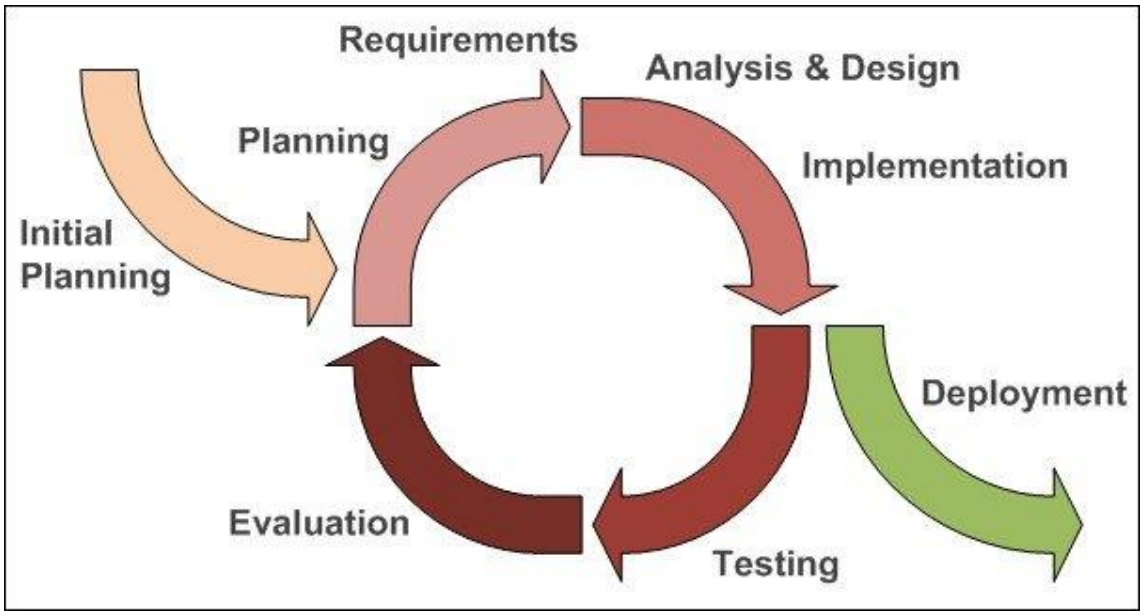


Ilustración 1 - Ilustración descriptiva de la metodología iterativa incremental [Fuente]

Cada iteración sigue un ciclo que incluye actividades como análisis de requisitos, diseño, implementación, prueba y evaluación. Durante cada iteración, se priorizan y desarrollan las características más importantes del sistema, lo que permite obtener resultados tangibles y funcionales en cada etapa.

A medida que se completa cada iteración, el producto se mejora y se agrega funcionalidad adicional, creando así una versión iterativa del software. Cada nueva versión se basa en las lecciones aprendidas de las iteraciones anteriores y se ajusta según los comentarios y requisitos del cliente o usuario final.

La metodología iterativa incremental tiene varios beneficios. En primer lugar, permite una mayor flexibilidad y adaptabilidad a medida que los requisitos y las necesidades del proyecto evolucionan con el tiempo. Al dividir el desarrollo en iteraciones, los cambios y ajustes pueden incorporar de manera más eficiente y rápida.

Además, al obtener resultados funcionales en cada iteración, los usuarios, en este caso los desarrolladores que utilizan el entorno, pueden comenzar a utilizar y evaluar el software en etapas tempranas del proceso de desarrollo. Esto facilita la identificación de problemas y permite realizar ajustes y mejoras antes de que se complete el producto final.

Otro beneficio clave es la mitigación de riesgos. Al desarrollar el software de manera incremental, se pueden identificar y abordar los riesgos más críticos y complejos desde el principio. Esto ayuda a minimizar los errores y los costos asociados con la corrección de problemas en etapas posteriores del proyecto.

Sin embargo, es importante destacar que la metodología iterativa incremental requiere una buena planificación y gestión de proyectos. Cada iteración debe tener objetivos claros y medibles, y los recursos y tiempos deben asignarse de manera adecuada para garantizar un progreso constante.

2.2. Planificación de las iteraciones

2.2.1. Iteración 1

El objetivo de esta Iteración es establecer los cimientos del proyecto y preparar el entorno necesario para su desarrollo completo. Para ello, realizaremos una serie de tareas clave que abarcarán desde la creación del proyecto y el repositorio de código hasta la instalación de las herramientas requeridas.

En primer lugar, nos enfocaremos en la creación del proyecto y estableceremos el repositorio de código correspondiente. Esto nos permitirá mantener un seguimiento organizado de las diferentes versiones y contribuciones al proyecto.

Además, desarrollaremos una plantilla para la documentación del proyecto, que nos ayudará a mantener una estructura clara y coherente en la documentación técnica y los recursos utilizados.

Otro aspecto importante de esta Iteración es la instalación de las herramientas necesarias. Nos aseguraremos de tener todas las dependencias y paquetes requeridos correctamente configurados en nuestro entorno de desarrollo. Esto incluirá la instalación y configuración de las herramientas principales que utilizaremos a lo largo del proyecto.

Además de estas tareas, también destinaremos tiempo a la formación interna en las tecnologías principales del proyecto, en particular, Vagrant. Esto nos permitirá familiarizarnos con su funcionamiento y aprovechar al máximo sus capacidades.

Previo a la entrega, realizaremos una fase de QA que garantice que la funcionalidad establecida para la entrega funciona correctamente.

Como resultado de esta Iteración, entregaremos un Vagrantfile inicial, integrado con Virtualbox, que nos permitirá generar una máquina virtual básica sin ningún tipo de provisión. Esta máquina virtual servirá como punto de partida para las próximas etapas del proyecto.

2.2.2. Iteración 2

Durante esta Iteración, nos enfocaremos en la implementación de la funcionalidad de ELK en el proyecto. Para lograrlo, seguiremos un conjunto de tareas clave. Primero, realizaremos una exhaustiva extracción de requisitos para comprender el alcance de la

monitorización necesaria en el proyecto. Luego, nos sumergiremos en la documentación de instalación de los tres productos principales: Elasticsearch, Kibana y Logstash.

Con base en nuestra comprensión de los requisitos y la documentación, procederemos al diseño de las piezas necesarias para la configuración de cada máquina. Esto incluirá la planificación de los recursos, la configuración de los parámetros y la integración adecuada de los componentes de ELK.

Una vez que hayamos definido el diseño, avanzaremos en la implementación práctica. Agregaremos las tres máquinas virtuales necesarias al código de la Vagrantfile, garantizando su correcta configuración y conexión en el entorno de desarrollo. También desarrollaremos los scripts de aprovisionamiento para cada una de las máquinas, asegurando una configuración coherente y eficiente.

Durante todo el proceso, evaluaremos continuamente nuestra implementación, buscando posibles mejoras y optimizaciones. Además, nos aseguraremos de seguir las mejores prácticas y estándares de seguridad para garantizar un entorno estable y protegido.

Previo a la entrega, realizaremos una fase de QA que garantice que la funcionalidad establecida para la entrega funciona correctamente; para ello, ejecutaremos el comando “vagrant up” y el resultado esperado es que en Virtualbox nos aparezcan 3 maquinas virtuales, 1 por cada producto, debemos cerciorarnos de que las 3 están correctamente interconectadas.

En resumen, esta Iteración se centrará en la integración exitosa de la funcionalidad de ELK en el proyecto. A través de una cuidadosa extracción de requisitos, la investigación exhaustiva y el diseño adecuado, así como la implementación práctica y la revisión continua, buscaremos asegurar que el resultado final cumpla con los estándares de calidad y eficiencia requeridos.

2.2.3. Iteración 3

En esta tercera iteración, continuaremos agregando funcionalidad al proyecto, centrándonos específicamente en la incorporación de K3S. Para ello, seguiremos un proceso similar al de la iteración anterior, enfocándonos en varias tareas clave.

En primer lugar, realizaremos una extracción de requisitos relacionados con la simulación de dispositivos IoT. Esto nos permitirá comprender en detalle las necesidades y funcionalidades requeridas para la implementación de los dispositivos en el entorno.

A continuación, nos sumergiremos en la lectura y estudio de la documentación necesaria para la instalación y creación de un cluster de K3S. Es fundamental comprender y dominar los conceptos y procedimientos necesarios para lograr una correcta implementación.

Posteriormente, procederemos al diseño de las piezas necesarias para la configuración del master y los nodos definidos en el diseño. Esto incluirá la definición de parámetros, configuraciones y características específicas para cada componente del cluster.

Una vez finalizado el diseño, incorporaremos las máquinas virtuales necesarias al código de la Vagrantfile.

Finalmente, desarrollaremos los scripts de provisión necesarios para automatizar la configuración de cada máquina virtual del cluster. Estos scripts garantizarán que las configuraciones y dependencias requeridas estén correctamente establecidas en cada nodo del cluster.

Para la fase de QA de esta iteración, deberemos cerciorarnos de que nos aparecen las máquinas virtuales definidas en el diseño, al menos 1 master y el número de nodos que determinemos. Para verificar que están correctamente interconectados, podemos ejecutar, una vez conectados por ssh al master, el comando kubectl get nodes y deberemos visualizar el número de nodos que hayamos establecido.

2.2.4. Iteración 4

En la cuarta iteración, nos enfocaremos en la incorporación de Mosquitto y Filebeat al proyecto, con el objetivo de establecer una comunicación efectiva entre los dispositivos desplegados en K3S y Logstash. Siguiendo un enfoque similar a las iteraciones anteriores, abordaremos diversas tareas clave.

Comenzaremos extrayendo los requisitos específicos relacionados con la comunicación entre los dispositivos simulados en K3S y Logstash. Esta fase nos permitirá comprender los protocolos y las funcionalidades necesarias para lograr una comunicación eficiente y confiable.

Tras ello, nos sumergiremos en la lectura y estudio de la documentación necesaria para la instalación y configuración de Mosquitto y Filebeat. Es fundamental comprender las opciones de configuración y consideraciones de seguridad relacionadas con estas herramientas para que la provisión se realice de forma efectiva.

Posteriormente, diseñaremos las piezas necesarias para la configuración de Mosquitto y Filebeat, teniendo en cuenta los requerimientos específicos de nuestro proyecto. Definiremos los parámetros de conexión, los temas (topics) de MQTT relevantes, las opciones de filtrado y transformación de los logs que serán enviados a Logstash.

Una vez finalizado el diseño, procederemos a añadir las máquinas virtuales necesarias al código de la Vagrantfile. Esto nos permitirá gestionar y controlar el despliegue y configuración de las máquinas virtuales que contendrán Mosquitto y Filebeat, asegurando un entorno de comunicación adecuado entre los dispositivos IoT y Logstash.

Por último, desarrollaremos los scripts de provisión necesarios para automatizar la instalación y configuración de Mosquitto y Filebeat en las máquinas virtuales correspondientes. Estos scripts se encargarán de garantizar que las dependencias, configuraciones y conexiones requeridas estén correctamente establecidas.

Para verificar que esta fase es correcta, deberemos cerciorarnos de que la comunicación hacia Mosquitto es correcta y que se permite la publicación y subscripción. Adicionalmente deberemos verificar que Filebeat es capaz de leer mensajes de

Mosquitto y de enviarlos a logstash. Para la primera parte, podemos desarrollar 2 script, uno de publicación y otro de subscripción, que ejecutaremos en la maquina host. Para la segunda parte, deberemos acudir a los logs de FIlebeat situados en la carpeta "/var/log/filebeat"

2.2.5. Iteración 5

En esta iteración, nuestro enfoque se centrará en el desarrollo de dos piezas de código adicionales que son esenciales para el funcionamiento del proyecto. Estas piezas tienen una dependencia específica y desempeñan un papel crucial en la comunicación y procesamiento de los datos.

La primera pieza de código a desarrollar será la simulación de los dispositivos IoT. Este código será responsable de generar datos simulados que representen el comportamiento de los dispositivos reales. Esto nos permitirá probar y validar el flujo de datos desde los dispositivos hasta Logstash a través de Filebeat. Además, definiremos un Dockerfile que permita la dockerización de este código para facilitar su despliegue en el entorno de K3S.

La segunda pieza de código se enfocará en el pipeline de Logstash. Este pipeline será responsable de recibir los logs enviados por Filebeat, aplicar filtros y transformaciones necesarias y finalmente enviarlos a Elasticsearch para su almacenamiento y análisis posterior. El desarrollo de este pipeline asegurará que los datos sean procesados y enriquecidos de manera adecuada antes de ser almacenados en Elasticsearch.

Durante la fase de aseguramiento de la calidad, llevaremos a cabo pruebas exhaustivas para garantizar el correcto funcionamiento de estas piezas de código. Realizaremos pruebas unitarias para verificar el comportamiento individual de cada componente y luego realizaremos pruebas integradas para evaluar su funcionamiento conjunto. Esto incluirá el despliegue de un pod en nuestro cluster de K3S y la verificación de que los logs sean recibidos correctamente en el tópico designado. También verificaremos los archivos ubicados en la máquina virtual de Logstash en la ruta "/var/log/logstash/", asegurándonos de que tanto la entrada (input) como la salida (output) del pipeline se conecten correctamente mediante la ausencia de mensajes de error en los logs.

En resumen, en esta iteración nos enfocaremos en el desarrollo del código de simulación de dispositivos y el pipeline de Logstash, así como en la dockerización de los dispositivos mediante un Dockerfile. Luego, realizaremos pruebas exhaustivas para asegurarnos de que estas piezas de código funcionen de manera adecuada, tanto individualmente como en conjunto. Este enfoque nos permitirá avanzar hacia una implementación sólida y confiable del proyecto.

2.2.6. Iteración 6

En esta última iteración, nos enfocaremos en dos actividades clave para finalizar el proyecto: pruebas End to End y documentación de la memoria del proyecto.

En primer lugar, llevaremos a cabo pruebas End to End para validar el funcionamiento completo de nuestro sistema. Para ello, utilizaremos el comando "vagrant up" para levantar todas las máquinas virtuales y asegurarnos de que estén correctamente configuradas. Además, desplegaremos al menos tres dispositivos simulados mediante el escalado de replicas en K3S. Esto nos permitirá probar la interacción entre los dispositivos, Mosquitto, Filebeat, Logstash, Elasticsearch y Kibana, y verificar que los datos se transmitan de manera adecuada a través del flujo completo.

Además, revisaremos la configuración de los índices en Elasticsearch, que definimos durante la creación del pipeline de Logstash. Realizaremos consultas a estos índices para verificar que la información se almacena correctamente y está disponible para su posterior análisis en Kibana.

Una vez finalizadas las pruebas, nos centraremos en la documentación de la memoria del proyecto. En esta fase, recopilaremos toda la información relevante sobre el proyecto, incluyendo la descripción detallada de la arquitectura, los componentes utilizados, las disquisiciones teóricas del planteamiento, las decisiones de diseño, las herramientas empleadas y los resultados obtenidos. También documentaremos los procedimientos de instalación y configuración del entorno de desarrollo, junto con cualquier problema o solución encontrada durante el proceso.

3. Fundamentos teóricos:

3.1. IoT

El IoT, o Internet de las cosas (en inglés, Internet of Things), es un concepto que se refiere a la interconexión de objetos físicos o dispositivos que están equipados con sensores, software y conectividad a Internet, lo que les permite recopilar y compartir datos. Estos dispositivos pueden abarcar una amplia variedad de objetos, desde electrodomésticos y dispositivos electrónicos personales hasta vehículos, maquinaria industrial y sensores incorporados en infraestructuras urbanas.

El objetivo principal del IoT es permitir la comunicación y la colaboración entre estos dispositivos, así como con sistemas y aplicaciones, con el fin de recopilar información en tiempo real, analizarla y utilizarla para tomar decisiones inteligentes, automatizar procesos y mejorar la eficiencia en diversos ámbitos, como el hogar inteligente, la salud, la agricultura, la industria, el transporte, etc.

La conexión de estos dispositivos a través de Internet permite el intercambio de datos de forma bidireccional, lo que significa que los dispositivos pueden enviar información y recibir instrucciones o comandos para llevar a cabo determinadas acciones. Esto crea un ecosistema digital en el que los objetos físicos se convierten en participantes activos en la recopilación, transmisión y procesamiento de datos, lo que brinda numerosas oportunidades y beneficios en términos de eficiencia, comodidad, seguridad, toma de decisiones informada y creación de nuevos casos de uso, repercutiendo positivamente en la experiencia de usuario.

Sin embargo, el IoT también plantea desafíos en términos de privacidad, seguridad y escalabilidad. Dado que los dispositivos IoT recopilan y transmiten datos sensibles, es fundamental garantizar la protección de la privacidad y la seguridad de la información. Además, la administración y el procesamiento de grandes volúmenes de datos generados por los dispositivos IoT requiere soluciones de almacenamiento y análisis eficientes y escalables. (1)

3.2. DevOps

DevOps es una metodología y conjunto de prácticas que combinan los aspectos del desarrollo de software (Dev) y las operaciones (Ops) en un enfoque integrado. Se basa en la colaboración estrecha y continua entre los equipos de desarrollo y operaciones con el objetivo de acelerar la entrega de software, mejorar la calidad y garantizar la estabilidad del sistema.

La filosofía detrás de DevOps es romper las barreras tradicionales entre los equipos de desarrollo y operaciones, fomentando la comunicación, la colaboración y la responsabilidad compartida. Se enfoca en automatizar los procesos, desde el desarrollo hasta la implementación y el monitoreo, para lograr una entrega continua y confiable de software.

Los principios clave de DevOps incluyen (2):

- **Colaboración**: Los equipos de desarrollo y operaciones trabajan juntos de manera estrecha y colaborativa, compartiendo conocimientos, objetivos y responsabilidades.
- **Automatización**: Se automatizan los procesos de desarrollo, pruebas, implementación y monitoreo para reducir los errores, mejorar la eficiencia y acelerar los tiempos de entrega.
- **Entrega continua**: Se busca entregar software en incrementos pequeños y frecuentes, lo que permite una retroalimentación rápida, una detección temprana de problemas y una mayor capacidad de respuesta a los cambios.
- **Infraestructura como código (IaaC)**: La infraestructura se trata como código, utilizando herramientas de gestión y aprovisionamiento automatizado para facilitar la configuración y despliegue de la infraestructura necesaria.
- **Monitorización y retroalimentación**: Se establecen mecanismos de monitoreo continuo del rendimiento y la calidad del software, lo que permite una retroalimentación rápida y la mejora continua del sistema.

Los beneficios de la implementación de DevOps incluyen una mayor agilidad en el desarrollo y despliegue de software, una mejor calidad del producto, una mayor eficiencia operativa y una mayor satisfacción del cliente.

En resumen, DevOps es una metodología que busca integrar los equipos de desarrollo y operaciones para lograr una entrega continua de software de alta calidad, a través de la colaboración, la automatización y una cultura de mejora continua.

3.3. Vagrant

Como comentamos anteriormente, Vagrant es una herramienta de código abierto diseñada para facilitar la creación y gestión de entornos de desarrollo portátiles y reproducibles. Proporciona una capa de abstracción sobre los sistemas de virtualización existentes, como VirtualBox, VMware o Hyper-V, permitiendo a los desarrolladores crear y configurar fácilmente máquinas virtuales con una configuración específica.

Con Vagrant, los desarrolladores pueden definir y describir el entorno de desarrollo deseado utilizando un archivo de configuración simple llamado "Vagrantfile". Este archivo especifica los detalles del sistema operativo, las configuraciones de red, los recursos de hardware y otros elementos necesarios para el entorno de desarrollo. Además, Vagrant permite la configuración y personalización del entorno utilizando scripts de provisión.

Una vez que el Vagrantfile y los scripts de provisión están configurados, los desarrolladores pueden utilizar comandos sencillos para crear y gestionar las máquinas virtuales. Vagrant se encarga automáticamente de la creación, configuración y aprovisionamiento de las máquinas virtuales, lo que facilita el proceso de configuración de entornos de desarrollo consistentes y reproducibles.

Los principales comandos de vagrant son:

- **Vagrant up**

El comando "vagrant up" se utiliza en Vagrant para crear y encender una máquina virtual según la configuración especificada en el Vagrantfile.

Cuando ejecutas "vagrant up" en el directorio de tu proyecto Vagrant, Vagrant lee el archivo de configuración Vagrantfile y procede a crear y configurar las máquinas virtuales de acuerdo con las especificaciones establecidas. Esto implica la descarga de la imagen base del sistema operativo, la creación de la máquina virtual en la plataforma de virtualización subyacente (como VirtualBox), y la aplicación de la configuración específica para cada máquina virtual.

El comando "vagrant up" también se encarga de realizar tareas como el aprovisionamiento automático, que puede incluir la instalación de software adicional, la configuración de la red, la creación de carpetas compartidas, entre otras acciones.

Una vez que el comando "vagrant up" se completa exitosamente, la máquina virtual estará en funcionamiento y lista para ser utilizada. Puedes acceder a ella mediante el comando "vagrant ssh" para ingresar a la máquina virtual y trabajar en ella.

- **Vagrant ssh**

El comando "vagrant ssh" se utiliza en Vagrant para acceder y conectarse a la máquina virtual creada con Vagrant.

Una vez que has ejecutado "vagrant up" y la máquina virtual está en funcionamiento, se puede utilizar el comando "vagrant ssh" desde el directorio del proyecto Vagrant para iniciar una sesión SSH en la máquina virtual.

Al ejecutar "vagrant ssh", Vagrant se encargará de establecer una conexión SSH con la máquina virtual y proporciona acceso a la línea de comandos de la máquina virtual directamente desde un terminal local. Esto permite interactuar con la máquina virtual como si estuvieras trabajando directamente en ella.

La conexión SSH establecida por Vagrant incluye la configuración necesaria para autenticarse automáticamente en la máquina virtual, por lo que no se requiere ningún tipo de contraseña o autenticación adicional.

Una vez que estés dentro de la sesión SSH de la máquina virtual, puedes ejecutar comandos y realizar tareas como lo harías en cualquier otra línea de comandos de un sistema operativo. Esto te permite instalar software, configurar el entorno de desarrollo y realizar otras operaciones necesarias dentro de la máquina virtual.

- Vagrant destroy

El comando "vagrant destroy" se utiliza en Vagrant para eliminar completamente una máquina virtual y todos sus recursos asociados.

Cuando se ejecuta "vagrant destroy", Vagrant detiene y apaga la máquina virtual, liberando todos los recursos utilizados por ella, como el espacio en disco y la memoria asignada. Además, se eliminan todos los archivos y configuraciones relacionados con la máquina virtual.

Este comando es útil cuando ya no se necesita la máquina virtual o se desea limpiar el entorno de desarrollo. Al ejecutar "vagrant destroy", se puede eliminar de manera segura la máquina virtual y comenzar desde cero, si es necesario.

Es importante tener en cuenta que el comando "vagrant destroy" es irreversible y eliminará permanentemente la máquina virtual. Por lo tanto, se recomienda utilizarlo con precaución y asegurarse de haber respaldado cualquier dato o configuración importante antes de ejecutarlo.

3.3.1. Ventajas del uso de Vagrant

Una de las ventajas clave de utilizar Vagrant en el desarrollo de IoT es la capacidad de reproducir entornos de manera consistente. Dado que el desarrollo de IoT a menudo involucra una combinación de hardware y software específicos, asegurar que todos los miembros del equipo tengan entornos idénticos puede ser un desafío. Sin embargo, con Vagrant, los desarrolladores pueden definir y compartir archivos de configuración que describen el entorno de desarrollo deseado. Esto garantiza que todos los desarrolladores tengan una base común y evita problemas causados por configuraciones inconsistentes, lo que ahorra tiempo y reduce la posibilidad de errores.

Además, Vagrant facilita la colaboración y el intercambio de proyectos de IoT. Al proporcionar una configuración fácil de seguir, los equipos pueden compartir sus proyectos con otros miembros o incluso con la comunidad en general. Esto fomenta la colaboración y acelera el proceso de desarrollo, ya que los desarrolladores pueden compartir rápidamente sus avances y permitir que otros los exploren y contribuyan. La capacidad de compartir entornos de desarrollo en forma de archivos Vagrant simplifica enormemente la configuración de nuevos miembros del equipo, permitiéndoles unirse rápidamente al proyecto sin problemas.

Otra ventaja significativa de utilizar Vagrant en el desarrollo de IoT es la capacidad de probar y depurar aplicaciones en diferentes entornos. En el desarrollo de IoT, es esencial garantizar que las aplicaciones funcionen correctamente en diversas configuraciones de hardware y software. Vagrant permite a los desarrolladores crear fácilmente múltiples máquinas virtuales con configuraciones específicas para simular diferentes entornos. Esto les permite probar y depurar sus aplicaciones en diferentes plataformas, sistemas operativos y configuraciones de red, lo que resulta en un producto final más robusto y compatible.

La automatización es otro beneficio clave que Vagrant aporta al desarrollo de IoT. Al aprovechar la funcionalidad de aprovisionamiento de Vagrant, los desarrolladores pueden automatizar la instalación y configuración de software y herramientas dentro de las máquinas virtuales. Esto ahorra tiempo y esfuerzo al eliminar la necesidad de configurar manualmente cada entorno de desarrollo individualmente. Además, la

automatización garantiza que los entornos sean consistentes y reproducibles, lo que reduce la posibilidad de errores humanos y facilita la escalabilidad a medida que el proyecto crece.

La seguridad también es una consideración crítica en el desarrollo de IoT, y Vagrant puede ayudar en este aspecto. Al utilizar máquinas virtuales aisladas, los desarrolladores pueden garantizar que sus proyectos se ejecuten en entornos seguros y controlados. Esto es especialmente relevante en el desarrollo de aplicaciones IoT, donde la seguridad y la privacidad de los datos son primordiales. Al utilizar Vagrant, los equipos de desarrollo pueden tener la tranquilidad de que sus proyectos se ejecutan en entornos virtuales protegidos, lo que minimiza el riesgo de filtraciones de datos o ataques malintencionados.

3.4. ELK Stack

El stack ELK, también conocido como stack Elastic, es una combinación de tres herramientas de código abierto ampliamente utilizadas en el análisis y visualización de datos: Elasticsearch, Logstash y Kibana. Cada una de estas herramientas cumple un papel importante dentro del stack y juntas ofrecen una solución integral para el procesamiento y análisis de logs y otros tipos de datos.

En conjunto, el stack ELK proporciona una solución completa para la recopilación, procesamiento, almacenamiento y visualización de datos en tiempo real. Su arquitectura distribuida y escalable permite manejar grandes volúmenes de datos de manera eficiente. Además, su flexibilidad y capacidad para integrarse con otras herramientas y sistemas lo convierten en una opción popular en entornos de análisis de logs y monitoreo de aplicaciones.

3.4.1. Elasticsearch

Es un motor de búsqueda y análisis, de tipo no SQL, distribuido y basado en Lucene. Actúa como core del stack ELK, proporcionando un almacenamiento altamente escalable y distribuido. Elasticsearch está diseñado para manejar grandes volúmenes de datos en tiempo real y ofrece una búsqueda y recuperación rápidas. Además, cuenta con capacidades de agregación y análisis avanzadas que permiten realizar consultas complejas sobre los datos indexados.

La arquitectura de Elasticsearch se compone de varios componentes clave que trabajan juntos para ofrecer una funcionalidad completa. A continuación, se detallan los componentes principales:

1. **Nodo**: Un nodo es una instancia de ejecución de Elasticsearch que forma parte de un clúster. Un clúster puede consistir en uno o más nodos y se utiliza para distribuir y replicar datos en el sistema.
2. **Índice**: Un índice es una colección lógica de documentos que tienen características similares. Los documentos dentro de un índice están

estructurados y se pueden buscar y analizar independientemente de otros índices en el clúster.

3. **Documento**: Un documento es la unidad básica de información en Elasticsearch. Está representado en formato JSON y se almacena en un índice. Los documentos son indexados y se pueden buscar y recuperar utilizando consultas.

4. **Shard**: Un shard es una porción de un índice. Los índices de Elasticsearch se dividen en múltiples shards para distribuir y paralelizar los datos en el clúster. Cada shard es una instancia independiente de Lucene que contiene una parte de los datos del índice.

5. **Réplica**: Una réplica es una copia de un shard. Las réplicas se utilizan para mejorar la disponibilidad y la tolerancia a fallos. Elasticsearch distribuye las réplicas en diferentes nodos dentro del clúster para garantizar la redundancia de los datos.

6. **Cluster**: Un clúster de Elasticsearch está compuesto por uno o más nodos que trabajan juntos para almacenar y procesar datos. Los nodos en un clúster se comunican y comparten datos entre sí para lograr una alta disponibilidad y una mejora en el rendimiento.

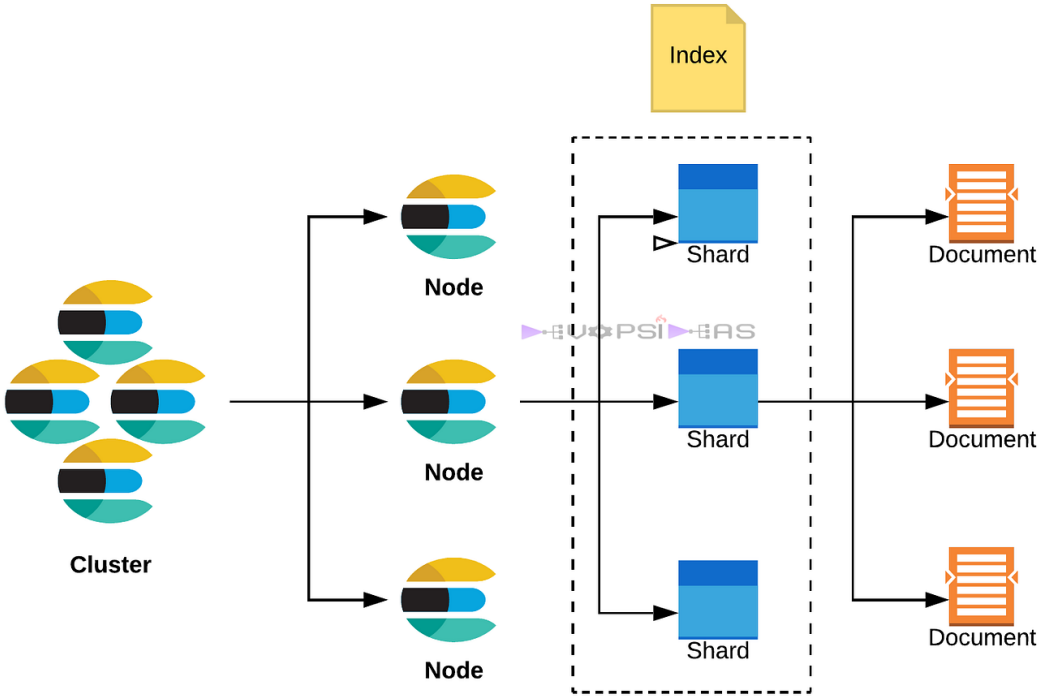


Ilustración 2 - Relación entre los componentes de Elasticsearch

3.4.2. Logstash

Logstash es una herramienta de procesamiento y transporte de datos de código abierto, desarrollada por Elastic. Se utiliza para recopilar, transformar y enviar datos de diferentes fuentes a Elasticsearch u otros destinos, como bases de datos o sistemas de almacenamiento.

La arquitectura de Logstash permite una gran flexibilidad y escalabilidad. Puedes ejecutar múltiples instancias de Logstash en paralelo para procesar grandes volúmenes de datos y distribuir la carga de trabajo. Esta se basa en pipelines compuestos por diferentes etapas de procesamiento.

Estas etapas son:

- **Input**: Esta etapa se encarga de la adquisición de datos desde diversas fuentes. Logstash proporciona una amplia gama de plugins de entrada que permiten la lectura de archivos de registro, eventos de red, bases de datos, servicios web, colas de mensajes, entre otros. Cada plugin de entrada define cómo se lee y se estructura la entrada de datos, así como la configuración específica del plugin.
- **Filter**: En esta etapa, Logstash realiza transformaciones y enriquecimientos de los datos recibidos. Los plugins de filtro se utilizan para analizar, filtrar, modificar y enriquecer los eventos de datos. Algunas de las operaciones comunes realizadas en esta etapa incluyen la eliminación de campos no deseados, el cambio de formatos de fecha, la normalización de datos y la extracción de información adicional.
- **Output**: En esta etapa, los datos procesados se envían a un destino específico. Logstash proporciona una variedad de plugins de salida que permiten el envío de datos a Elasticsearch, bases de datos como MySQL o PostgreSQL, servicios web, colas de mensajes, archivos de registro, entre otros. Cada plugin de salida define cómo se envían los datos y cómo se estructura la salida.

3.4.3. Kibana

Es una interfaz web que se utiliza para visualizar y explorar los datos almacenados en Elasticsearch. Kibana ofrece una amplia gama de herramientas de visualización, como gráficos, tablas, mapas y paneles de control interactivos, que permiten a los usuarios analizar y comprender los datos de manera intuitiva. Además, ofrece capacidades de búsqueda y consultas avanzadas, lo que facilita la exploración de datos y la detección de patrones y tendencias.

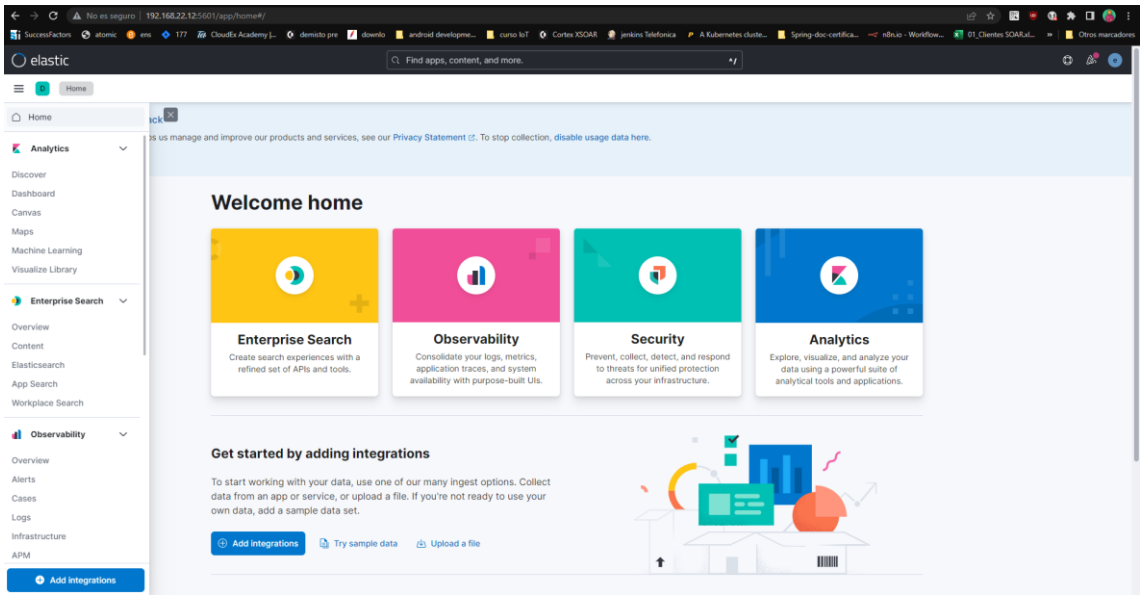


Ilustración 3 - Landing page de Kibana tras realizar el login

Las principales funcionalidades nativas de Kibana son:

1. Búsqueda y consulta de datos: Kibana permite realizar búsquedas avanzadas y consultas en los datos almacenados en Elasticsearch. Utilizando su lenguaje de consulta, es posible filtrar y obtener resultados específicos basados en diferentes criterios, como palabras clave, rangos de fechas, campos específicos, entre otros. Para ello, podemos hacer uso de la API o de la sección de Consola, dentro de las herramientas de desarrollador.

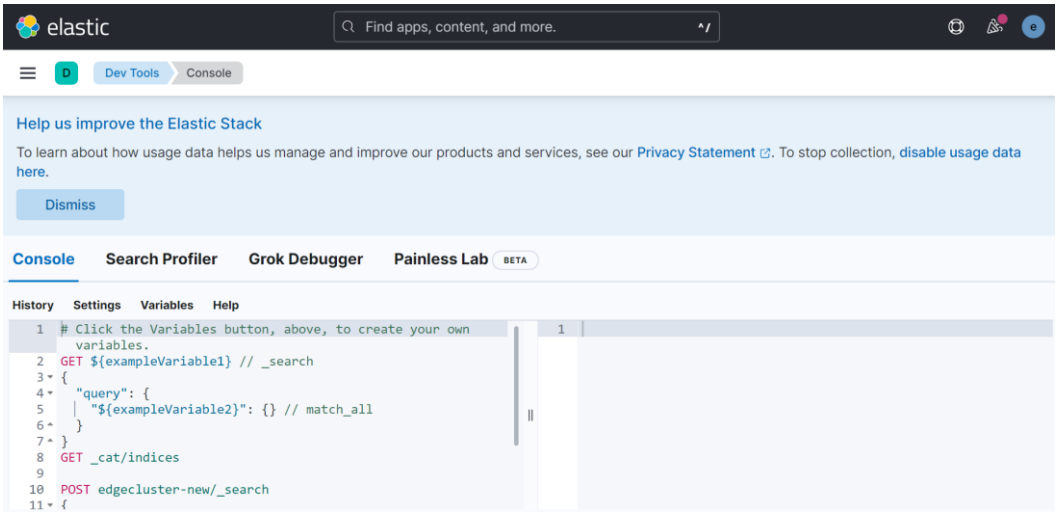


Ilustración 4 - Consola dentro de Kibana, desde donde podríamos lanzar peticiones y visualizar los resultados

2. Visualización de datos: Kibana ofrece una amplia gama de opciones de visualización para representar los datos de manera clara y significativa. Entre las opciones disponibles se incluyen gráficos de barras, gráficos circulares, gráficos de dispersión, mapas, diagramas de dispersión, tablas, métricas y más. Estas

visualizaciones se pueden personalizar y configurar según las necesidades específicas del usuario.

3. Dashboards interactivos: nos permite crear paneles interactivos que contienen múltiples visualizaciones y tablas. Estos paneles, conocidos como dashboards, permiten organizar y presentar los datos de manera coherente en un solo lugar. Los usuarios pueden personalizar los dashboards según sus preferencias, agregar filtros y actualizar en tiempo real para tener una visión general de los datos.

4. Métricas y monitoreo en tiempo real: Kibana proporciona herramientas para el monitoreo en tiempo real de datos y métricas. Los usuarios pueden crear visualizaciones y alertas en tiempo real para supervisar el estado de los sistemas y servicios. Esto es especialmente útil en entornos de monitoreo de salud de infraestructuras, donde se necesita estar al tanto de las métricas clave y los eventos importantes en tiempo real.

5. Exploración y análisis de logs: en Kibana se incluyen funciones específicas para la exploración y análisis de logs. Permite importar y visualizar logs almacenados en Elasticsearch, realizar búsquedas y filtrar registros según diferentes criterios, aplicar análisis de texto, crear gráficos y tablas para representar patrones y tendencias en los logs, y realizar correlación de eventos.

6. Seguridad y gestión de usuarios: Kibana proporciona opciones de seguridad y gestión de usuarios para controlar el acceso a las diferentes funcionalidades de la plataforma. Permite configurar roles y permisos, autenticación de usuarios, integración con proveedores de identidad externos y cifrado de comunicaciones.

3.5. Kubernetes y K3S

Kubernetes es una plataforma de código abierto para la orquestación y gestión de contenedores. Fue desarrollada originalmente por Google y posteriormente donada a la Cloud Native Computing Foundation (CNCF). Kubernetes proporciona un entorno de ejecución para contenedores, como Docker, y facilita la administración y escalado de aplicaciones en contenedores en entornos de producción.

La principal función de Kubernetes es automatizar y coordinar la distribución, escalado y gestión de aplicaciones empaquetadas en contenedores a través de múltiples nodos de un clúster. Proporciona un conjunto de características y componentes que permiten una orquestación eficiente de los contenedores, asegurando que las aplicaciones se ejecuten de manera confiable y escalable.

Algunas características clave de Kubernetes son:

1. **Escalado automático**: Kubernetes permite escalar automáticamente las aplicaciones en función de la demanda, aumentando o disminuyendo el número de réplicas de los contenedores según las métricas de rendimiento establecidas.

2. **Alta disponibilidad**: Kubernetes garantiza que las aplicaciones estén siempre disponibles, incluso en caso de fallos en los nodos del clúster. Si un nodo falla, Kubernetes reprograma automáticamente los contenedores en otros nodos saludables.
3. **Despliegue declarativo**: Se define el estado deseado de la aplicación y Kubernetes se encarga de mantener el estado actual en línea con el estado deseado, gestionando las actualizaciones y las versiones de las aplicaciones de forma transparente.
4. **Gestión del almacenamiento**: Kubernetes ofrece una gestión flexible y automatizada del almacenamiento persistente para los contenedores, permitiendo que los datos se mantengan de manera segura y accesible.
5. **Networking**: Kubernetes proporciona una infraestructura de red virtual que permite la comunicación entre los contenedores y los servicios dentro del clúster, facilitando la conectividad y el enrutamiento.
6. **Integración con herramientas y servicios**: Kubernetes se integra con una amplia gama de herramientas y servicios, como sistemas de registro, monitoreo y balanceo de carga, lo que permite construir soluciones completas y escalables.

K3s, por otro lado, es una distribución ligera de Kubernetes diseñada para entornos con recursos limitados, como dispositivos IoT, sistemas embebidos o entornos de prueba y desarrollo. Fue desarrollado por Rancher Labs como una alternativa simplificada y optimizada de Kubernetes.

Kubernetes es una plataforma de código abierto ampliamente utilizada para orquestar y gestionar contenedores en entornos de producción. Sin embargo, la implementación y administración de un clúster de Kubernetes completo puede ser compleja y requerir recursos significativos.

K3s aborda esta problemática al ofrecer una versión más ligera y simplificada de Kubernetes. Proporciona todas las funcionalidades clave de Kubernetes, como la programación de contenedores, el escalado automático, la tolerancia a fallos y la administración centralizada, pero con un enfoque más ligero y fácil de implementar.

Algunas de las características destacadas de K3s incluyen:

1. **Tamaño reducido**: K3s tiene un tamaño de distribución mucho más pequeño en comparación con Kubernetes estándar, lo que facilita su implementación en dispositivos con recursos limitados.
2. **Bajo consumo de memoria RAM y CPU**: K3S está optimizado para utilizar menos recursos de memoria y CPU, lo que lo hace adecuado para entornos con restricciones de recursos.

3. **Fácil instalación y administración**: K3S simplifica el proceso de instalación y configuración de un clúster de Kubernetes. Utiliza un agente ligero que se ejecuta en cada nodo del clúster, lo que facilita su administración y mantenimiento.
4. **Integración con herramientas de contenedores**: K3s es compatible con las herramientas y tecnologías de contenedores estándar, como Docker, lo que permite ejecutar y administrar contenedores de manera eficiente.
5. **Seguridad**: K3s incluye características de seguridad integradas, como la autenticación y el cifrado del tráfico, para garantizar la protección de los recursos del clúster.

La simplicidad y eficiencia de K3s lo hacen especialmente adecuado para entornos de IoT, donde se requiere una orquestación ligera y eficiente de contenedores. Permite desplegar y administrar fácilmente clústeres de contenedores en dispositivos IoT y entornos con recursos limitados, brindando las ventajas de Kubernetes sin comprometer el rendimiento ni la funcionalidad.

3.5.1. Principales términos relacionados con Kubernetes

Nodo (Node): Un nodo es una máquina física o virtual que forma parte del clúster de Kubernetes. Cada nodo tiene capacidad para ejecutar contenedores y se encarga de recibir y ejecutar las tareas asignadas por el planificador de Kubernetes.

Pod: Un pod es la unidad básica de despliegue en Kubernetes. Representa un grupo de uno o más contenedores que comparten recursos y se ejecutan en el mismo nodo. Los pods son efímeros y pueden ser creados, escalados y eliminados fácilmente.

Replication Controller: El controlador de replicación es responsable de garantizar que un número específico de réplicas de un pod esté siempre en ejecución. Si el número de réplicas cae por debajo de lo especificado, el controlador de replicación crea nuevos pods para mantener el estado deseado.

Servicio (Service): Un servicio es un objeto de Kubernetes que define una política de red estable para acceder a un conjunto de pods. Proporciona una dirección IP virtual y un nombre de DNS para permitir la comunicación con los pods a través de un conjunto de reglas de enrutamiento.

Volumen (Volume): Un volumen es un recurso de almacenamiento persistente que puede ser montado en uno o más pods. Los volúmenes permiten que los datos sean compartidos y sobrevivan a la vida útil de los pods, lo que es útil para aplicaciones que requieren almacenamiento persistente.

Namespaces: Los namespaces son una forma de dividir el clúster en múltiples particiones lógicas o entornos virtuales. Permiten organizar y separar los recursos en grupos lógicos, lo que facilita la gestión y el aislamiento de las aplicaciones.

Planificador (Scheduler): El planificador es el componente de Kubernetes encargado de asignar pods a los nodos disponibles en función de los requisitos de recursos, las políticas

de afinidad y las restricciones definidas. El planificador distribuye y balancea la carga de trabajo de manera eficiente en el clúster.

3.5.2. Principales comandos de K3s

- k3s server: Inicia el servidor K3s, que es el componente principal del clúster. Este comando se ejecuta en el nodo maestro.
- k3s agent: Inicia un agente K3s, que se une al clúster y ejecuta las tareas asignadas por el servidor. Este comando se ejecuta en los nodos de trabajo.
- k3s crictl: el comando k3s crictl proporciona una forma de ejecutar comandos directamente en el runtime CRI en lugar de usar las herramientas de administración de Kubernetes normales. Esto permite a los administradores de clústeres inspeccionar y depurar contenedores directamente en tiempo de ejecución. Por ejemplo, “k3s crictl logs <container-id>” muestra los output logs de un contenedor específico.
- k3s check-config: Comprueba la configuración del clúster K3s y muestra un resumen de su estado actual.
- Kubectl o k3s kubectl: K3s utiliza el mismo comando kubectl que Kubernetes para interactuar con el clúster. Con este comando, puedes administrar y controlar los recursos del clúster, como pods, servicios, volúmenes, etc. en caso de ejecutar K3S kubectl obtendremos prácticamente el mismo resultado pero el comando se habrá ejecutado de forma optimizada para el cluster K3S.
  Entre estos comandos, encontramos (3):
  - kubectl get: Obtiene información sobre los recursos del clúster. Por ejemplo, puedes ejecutar kubectl get pods para obtener una lista de los pods en el clúster.
  - kubectl create: Crea un nuevo recurso en el clúster. Puedes usar comandos como kubectl create deployment o kubectl create service para crear implementaciones de aplicaciones o servicios.
  - kubectl describe: Proporciona detalles de un recurso específico. Por ejemplo, puedes usar kubectl describe pod <nombre-del-pod> para obtener información detallada sobre un pod en particular.
  - kubectl apply: Aplica la configuración definida en un archivo YAML o JSON al clúster. Esto permite crear o actualizar recursos de manera declarativa.
  - kubectl delete: Elimina un recurso específico del clúster. Por ejemplo, puedes usar kubectl delete pod <nombre-del-pod> para eliminar un pod.
  - kubectl exec: Ejecuta un comando dentro de un contenedor en un pod en ejecución. Esto te permite interactuar con el entorno del contenedor, por ejemplo, ejecutar una shell o ejecutar comandos específicos.
  - kubectl logs: Muestra los registros (logs) de un pod específico. Puedes utilizar opciones adicionales, como --follow, para seguir en tiempo real los registros del pod.
  - kubectl scale: Escala el número de réplicas de un recurso, como un deployment. Por ejemplo, kubectl scale deployment <nombre-del-deployment> --replicas=3 aumentaría el número de réplicas a 3.

- kubectl expose: Expone un servicio mediante un balanceador de carga o un tipo de servicio específico. Por ejemplo, kubectl expose deployment <nombre-del-deployment> --type=LoadBalancer crea un servicio accesible desde el exterior a través de un balanceador de carga.
- kubectl rollout: Gestiona las actualizaciones de implementaciones en el clúster. Se pueden utilizar comandos como kubectl rollout status, kubectl rollout pause, kubectl rollout resume para administrar el proceso de depliegue.

3.6. Mosquitto (MQTT)

Mosquitto es un broker o servidor de mensajería MQTT (Message Queuing Telemetry Transport) de código abierto. MQTT es un protocolo de comunicación ligero y eficiente diseñado específicamente para la comunicación entre dispositivos de IoT (Internet de las cosas).

Mosquitto proporciona un servidor que permite a los dispositivos IoT enviar y recibir mensajes a través del protocolo MQTT. Actúa como intermediario entre los dispositivos y las aplicaciones que desean recibir los datos generados por esos dispositivos.

Algunas características y funciones clave de Mosquitto son:

I. **Conexión y suscripción**: Los dispositivos MQTT pueden conectarse a Mosquitto y suscribirse a diferentes temas (topics) para recibir mensajes. Esto permite una comunicación eficiente y escalable entre los dispositivos y las aplicaciones.

II. **Publicación de mensajes**: Los dispositivos pueden publicar mensajes en Mosquitto, especificando el tema al que pertenecen. Estos mensajes pueden contener información como datos de sensores, eventos o cualquier otro tipo de información relevante para el sistema IoT.

III. **Calidad de servicio (Quality of Service, QoS)**: Mosquitto ofrece diferentes niveles de QoS para garantizar la entrega confiable de los mensajes. Esto permite adaptar la calidad de servicio a las necesidades específicas de la aplicación, asegurando que los mensajes se entreguen correctamente. Estos son (4):

    1. QoS 0 (At most once): En este nivel, el mensaje se entrega como máximo una vez, sin confirmación ni reintentos. El remitente envía el mensaje una vez y no espera ninguna respuesta de confirmación del receptor. Esto proporciona la entrega más rápida y eficiente, pero no garantiza la entrega del mensaje. Si el receptor no está disponible o no puede recibir el mensaje en ese momento, se perderá sin ninguna notificación. Este nivel se utiliza en aplicaciones donde la pérdida ocasional de mensajes no es crítica, como actualizaciones de estado o datos que no necesitan ser recibidos de manera confiable.

2. QoS 1 (At least once): En este nivel, el mensaje se entrega al menos una vez, con confirmación de recepción. El remitente envía el mensaje al receptor y espera una confirmación de entrega. Si el receptor no puede confirmar la recepción, el remitente reenvía el mensaje hasta que se reciba la confirmación. Esto garantiza que el mensaje se entregue al menos una vez, pero puede haber duplicación de mensajes en caso de reintentos. Este nivel se utiliza en aplicaciones donde la entrega confiable de mensajes es importante, pero la duplicación ocasional de mensajes no causa problemas graves, como actualizaciones de estado críticas o comandos que deben ejecutarse al menos una vez.

3. QoS 2 (Exactly once): En este nivel, el mensaje se entrega exactamente una vez, sin duplicación ni pérdida. El remitente y el receptor realizan un proceso de intercambio de mensajes para garantizar la entrega exacta y única. El remitente envía el mensaje y espera una confirmación de recepción. Luego, el receptor envía una confirmación de recepción y el remitente envía una confirmación final. Este proceso de intercambio garantiza que el mensaje se entregue exactamente una vez, sin duplicación. Este nivel se utiliza en aplicaciones donde la entrega precisa y sin duplicaciones de los mensajes es crítica, como transacciones financieras o comandos que deben ejecutarse exactamente una vez.

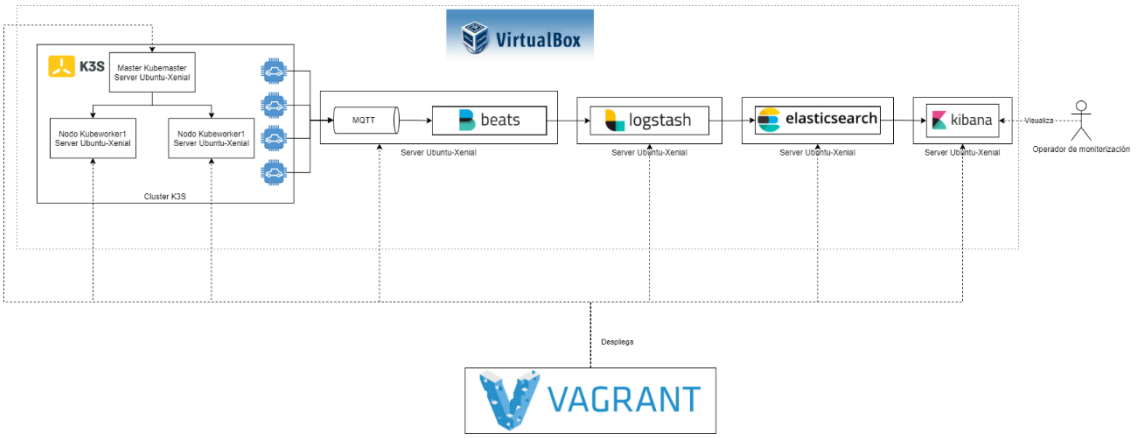
Es importante tener en cuenta que los niveles de QoS más altos (QoS 1 y QoS 2) introducen un mayor consumo de ancho de banda y latencia debido a las confirmaciones y reintentos involucrados. Por lo tanto, es recomendable seleccionar el nivel de QoS adecuado en función de los requisitos de la aplicación y las restricciones de rendimiento de la red.

IV. Autenticación y seguridad: Mosquitto admite diferentes métodos de autenticación para garantizar la seguridad en la comunicación entre dispositivos y el servidor MQTT. También es posible configurar el cifrado de extremo a extremo para proteger la confidencialidad de los mensajes transmitidos. (5)

4. Resultados y Discusión

4.1. Arquitectura de la solución

La arquitectura completa de la solución es la descrita en el siguiente diagrama:



La arquitectura de sistemas compuesta por estas 7 máquinas virtuales se organiza de la siguiente manera:

- Cluster K3S: Consiste en un master y 2 nodos donde se despliegan los pods que simulan dispositivos del Internet de las cosas (IoT). El cluster K3S utiliza la tecnología de Kubernetes para gestionar la orquestación y escalado de los pods de manera eficiente.
- Máquina virtual con un broker MQTT: Esta máquina virtual aloja un broker MQTT que actúa como intermediario para recibir los logs generados por los dispositivos simulados. El broker MQTT proporciona una comunicación bidireccional y ligera entre los dispositivos y las demás herramientas de la arquitectura. Esta misma máquina virtual contiene un agente de Filebeat; su función es recopilar los logs generados por el broker MQTT y enviarlos a Logstash para su procesamiento y filtrado posterior.
- Máquina virtual con Logstash: Aquí se despliega Logstash, una herramienta de procesamiento y transformación de logs. Logstash recibe los logs enviados por Filebeat y los procesa según las configuraciones definidas, como filtros y transformaciones. Su objetivo es enriquecer y estructurar los datos antes de enviarlos al siguiente componente.
- Máquina virtual con Elasticsearch: Esta máquina virtual aloja Elasticsearch, un motor de búsqueda y almacenamiento de datos distribuido. Elasticsearch se encarga de recibir los logs procesados por Logstash y almacenarlos de forma eficiente para su posterior consulta y análisis. Proporciona un almacenamiento escalable y de alto rendimiento.
- Máquina virtual con Kibana: Aquí se encuentra Kibana, una plataforma de visualización de datos y análisis. Kibana permite a los usuarios y desarrolladores explorar, buscar y visualizar los logs almacenados en Elasticsearch. Proporciona una interfaz gráfica intuitiva para crear paneles de control, gráficos y tablas personalizadas.

Cada componente desempeña un papel específico en la arquitectura, trabajando en conjunto para recibir, procesar, almacenar y visualizar los logs generados por los dispositivos IoT simulados. Esta arquitectura permite una comunicación eficiente, un

procesamiento de logs escalable y una visualización interactiva de los datos para un análisis efectivo.

El despliegue de esta arquitectura se orquesta mediante Vagrant y un único Vagrantfile, con el objetivo de simplificar su uso.

4.2. Decisiones de Diseño

4.2.1. Utilización de ficheros compartidos para albergar contraseñas generadas durante la instalación

Una de las decisiones que se ha tomado durante todo el proyecto es la de utilizar ficheros en carpetas compartidas entre máquinas virtuales para albergar los ficheros que debían ser provisionados y para albergar aquellas contraseñas que se generaban durante la instalación de componentes y que debían ser accesibles por otros componentes o por el usuario final.

Esta decisión se toma para conseguir la automatización y configuración completa de forma automática de todos los componentes y sus intercomunicaciones

4.2.2. Utilización de Python para los dispositivos

Se decide utilizar python atendiendo a algunas ventajas que presenta en entornos agiles y para proyectos de media-baja complejidad.

En primer lugar, Python es conocido por su facilidad de uso y su sintaxis clara, lo que permite a los desarrolladores expresar de manera eficiente la lógica de simulación. Además, Python cuenta con una amplia comunidad y numerosas bibliotecas especializadas en IoT, lo que facilita el desarrollo en este entorno y la integración con otros componentes del ecosistema IoT.

Python es un lenguaje versátil y portátil, lo que significa que el código de los dispositivos simulados escrito una vez puede ejecutarse en diferentes plataformas y arquitecturas. Asimismo, Python se integra de manera fluida con k3s, lo que permite desplegar fácilmente las simulaciones como contenedores y aprovechar las capacidades de orquestación y gestión de Kubernetes.

Por último, Python ofrece amplio soporte para los protocolos comunes utilizados en IoT, facilitando la comunicación y la interacción de los dispositivos simulados con otros componentes de la infraestructura, como en este caso, MQTT. En resumen, Python proporciona un entorno de desarrollo eficiente y flexible para simular dispositivos IoT, aprovechando las ventajas de k3s y garantizando la interoperabilidad con otros sistemas y protocolos utilizados en el campo del IoT y es por todas estas razones que se ha decido su utilización en este proyecto.

4.2.3. Desarrollo de un solo Vagranfile

La creación de un solo Vagrantfile en lugar de varios es una decisión que se ha tomado atentiendo a varias ventajas, principalmente en términos de facilidad de uso por parte de los desarrolladores y reutilización del código en futuros proyectos.

Facilidad de uso: Al tener un solo Vagrantfile, los desarrolladores tienen una única ubicación donde pueden configurar y gestionar todas las máquinas virtuales necesarias para el proyecto. Esto simplifica la administración y el despliegue, ya que no se requiere realizar un seguimiento de múltiples archivos de configuración. Además, la sintaxis y estructura del Vagrantfile se mantienen coherentes y consistentes en todo el proyecto, lo que facilita su comprensión y mantenimiento.

Delimitación clara de secciones: El Vagrantfile permite delimitar claramente las secciones de configuración para cada máquina virtual. Esto significa que se pueden especificar de manera ordenada y estructurada los detalles de cada máquina, como su nombre, configuración de red, asignación de recursos y provisionamiento. Al tener estas secciones bien definidas en un solo archivo, es más fácil realizar ajustes y modificaciones sin tener que navegar entre varios archivos de configuración dispersos.

Reutilización de código: Al contar con un solo Vagrantfile, es posible reutilizar secciones de configuración en futuros proyectos. Por ejemplo, si se ha definido una sección de configuración específica para el despliegue de una máquina virtual en el entorno de desarrollo, se puede utilizar esa misma sección en otros proyectos sin necesidad de partir el archivo en varios ficheros. Esto ahorra tiempo y esfuerzo, ya que no es necesario volver a escribir o copiar y pegar código repetitivo.

En resumen, la creación de un solo Vagrantfile en lugar de varios proporciona facilidad de uso para los desarrolladores al tener una única ubicación para gestionar las máquinas virtuales del proyecto. Además, las secciones bien delimitadas permiten una configuración ordenada y la reutilización de código en futuros proyectos, lo que ahorra tiempo y promueve la consistencia en el desarrollo.

4.2.4. Inclusión de Filebeat en la arquitectura

La inclusión de Filebeat en el proyecto para enviar los mensajes de MQTT a Logstash se fundamenta en varias razones importantes que mejoran la eficiencia y la mantenibilidad del proyecto.

En primer lugar, contar con un agente como Filebeat instalado en la máquina virtual de MQTT brinda una mejora significativa en la eficiencia del proceso de envío de mensajes. Filebeat está diseñado específicamente para la recolección y envío eficiente de logs y datos, lo que minimiza el consumo de recursos y la latencia en la transmisión de información. Al utilizar Filebeat, aseguramos que los mensajes de MQTT se envíen de manera rápida y eficiente a Logstash, optimizando el rendimiento del sistema en general.

Además, la elección de Filebeat se basa en su soporte oficial del input plugin de MQTT. Filebeat cuenta con una amplia comunidad de usuarios y un respaldo oficial, lo que garantiza la disponibilidad de actualizaciones, mejoras y correcciones de errores en el input plugin de MQTT. Esto facilita la integración y la mantenibilidad del proyecto, ya que podemos confiar en la estabilidad y compatibilidad de Filebeat en relación con el protocolo MQTT.

Por otro lado, se ha optado por enviar los mensajes a Logstash en lugar de hacerlo directamente a Elasticsearch por objetivos académicos y para incluir todas las piezas del stack ELK en el proyecto. Esta elección nos permite comprender y utilizar todas las herramientas y componentes del stack en su totalidad, lo que nos brinda un conocimiento más completo de su funcionamiento y nos permite aprovechar todas las capacidades y funcionalidades que ofrecen.

Además, al enviar los mensajes a Logstash antes de almacenarlos en Elasticsearch, tenemos la oportunidad de realizar filtrados y mapeos avanzados de los datos. Logstash proporciona una amplia gama de plugins y opciones de configuración que nos permiten transformar y enriquecer los datos antes de su indexación en Elasticsearch. Esto nos brinda flexibilidad para adaptar los datos a nuestras necesidades específicas, realizar análisis más avanzados y garantizar una estructura coherente en los registros almacenados.

4.2.5. Inclusión de Mosquitto

La inclusión de Mosquitto en la arquitectura se justifica por diversas motivaciones que buscan mejorar el rendimiento y la flexibilidad del sistema, aprovechar la ligereza del protocolo MQTT y aplicar los conocimientos adquiridos en asignaturas del master.

Una de las principales motivaciones para incluir Mosquitto es desacoplar los dispositivos de las herramientas de monitorización. Utilizando MQTT los dispositivos pueden enviar y recibir mensajes sin estar directamente vinculados a las herramientas de monitorización. Esto proporciona una arquitectura más flexible y escalable, ya que los dispositivos pueden comunicarse de manera independiente con el broker, y las herramientas de monitorización pueden suscribirse a los temas relevantes para recibir los datos necesarios. Además, al garantizar alta disponibilidad y disaster recovery en el servidor MQTT, se asegura que no haya pérdida de información y se restablezca el funcionamiento completo en caso de una interrupción del servicio.

Además, el protocolo MQTT se caracteriza por su ligereza y bajo consumo de recursos, lo que lo hace ideal para entornos IoT con dispositivos con recursos limitados. Comparado con otras tecnologías de mensajería como ActiveMQ y Kafka, MQTT tiene una sobrecarga mucho menor debido a su diseño simple y eficiente. Esto significa que los dispositivos pueden enviar y recibir mensajes de manera más rápida y con un menor consumo de ancho de banda y energía.

En términos de beneficios, MQTT destaca por su capacidad de proporcionar una comunicación confiable y eficiente en entornos IoT. Algunos de los beneficios de utilizar MQTT en entornos IoT son:

- Ligereza y eficiencia: MQTT está diseñado para ser liviano y eficiente, lo que lo hace adecuado para dispositivos con recursos limitados. Permite una comunicación rápida y confiable con un bajo consumo de ancho de banda y energía.
- Baja latencia: MQTT proporciona una baja latencia de comunicación, lo que permite una respuesta rápida entre los dispositivos y los servicios de backend.

Esto es especialmente importante en entornos en tiempo real y aplicaciones que requieren una comunicación ágil.

- Escalabilidad: MQTT es altamente escalable y puede manejar una gran cantidad de dispositivos conectados simultáneamente. Esto es esencial en entornos IoT donde se espera una gran cantidad de dispositivos interconectados.
- Suscripción selectiva: MQTT permite a los clientes suscribirse selectivamente a los temas de interés, lo que significa que los dispositivos y las herramientas de monitorización pueden recibir solo los datos relevantes para ellos. Esto reduce la carga de tráfico de red y facilita el procesamiento de datos.
- Conexión persistente: MQTT admite conexiones persistentes, lo que garantiza que los dispositivos puedan recuperarse automáticamente de las interrupciones de conexión sin perder mensajes. Esto garantiza la confiabilidad en la comunicación en entornos inestables o con conectividad intermitente.

En resumen, la inclusión de Mosquitto en la arquitectura aprovecha los beneficios del protocolo MQTT, como la capacidad de desacoplar los dispositivos de las herramientas de monitorización, la ligereza del protocolo y la aplicación de conocimientos adquiridos en asignaturas del master. Comparado con otras tecnologías de mensajería, MQTT destaca por su eficiencia, baja latencia, escalabilidad y capacidad de suscripción selectiva, lo que lo convierte en una elección favorable para entornos IoT y en especial para este proyecto.

4.3. Configuración del entorno

La configuración del entorno es la principal funcionalidad resultante del proyecto, para replicar el entorno en el PC de un nuevo desarrollador que se una al grupo de trabajo, se pueden seguir los siguientes pasos:

1- Instala Vagrant: Ve al sitio web oficial de Vagrant, descarga el instalador y sigue los pasos para completar la instalación, aceptando los términos de uso.
2- Clona el repositorio: Abre una linea de comandos (CMD o Powershell) y ejecuta el siguiente comando para clonar el repositorio del proyecto de Vagrant: git clone https://github.com/alfonsodiezramirez-upm/vagrant-ELK-edgeCluster.git
3- Verifica VirtualBox: Asegúrate de tener instalado VirtualBox. Si no lo tienes, descárgalo desde su página oficial y ejecuta el archivo de instalación.
4- Navega al directorio del repositorio: Abre una ventana de CMD o Powershell y ve a la ubicación donde clonaste el repositorio, donde se encuentra el archivo Vagrantfile.
5- Inicia las máquinas virtuales: Ejecuta el comando "vagrant up" y espera unos minutos. Verás cómo las máquinas virtuales del proyecto se añaden en VirtualBox.

Con estos pasos, tendremos instalado Vagrant, clonado el repositorio y puesto en marcha las máquinas virtuales del proyecto.

4.4. Pruebas y validación

En cada iteración se ha planificado una fase de QA, en esta sección, se agregan todas las pruebas de diferente tipo que se han realizado a lo largo del proyecto. Para ello nos apoyaremos en el lenguaje Gherkin que se utiliza en herramientas como cucumber:

4.4.1. Pruebas unitarias:

Feature	Todos los sistemas se inician correctamente
	Como miembro del equipo de desarrollo Quiero asegurarme de que todos los sistemas se inician correctamente Para garantizar el funcionamiento adecuado del entorno de desarrollo
Scenario	Iniciar correctamente Elasticsearch
	Given que el sistema Elasticsearch está configurado correctamente **When** se inicia el sistema Elasticsearch **Then** Todos los componentes del sistema se inician correctamente; Podemos conectarnos al sistema accediendo a la ip de la maquina en el puerto 9200 de un navegador de la máquina host; Podemos hacer login mediante la contraseña generada; Disponemos de un token en el fichero situado en ./shared/token;
Scenario	Iniciar correctamente Kibana
	Given que el sistema Kibana está configurado correctamente **When** se inicia el sistema Kibana **Then** Todos los componentes del sistema se inician correctamente; Podemos conectarnos al sistema accediendo a la ip de la maquina en el puerto 5601 de un navegador de la máquina host;
Scenario	Iniciar correctamente Logstash
	Given que el sistema Logstash está configurado correctamente **When** se inicia el sistema Logstash **Then** Todos los componentes del sistema se inician correctamente; Podemos a la máquina virtual y no observamos errores en los logs de systemctl;
Scenario	Iniciar correctamente el nodo master de K3S
	Given que el nodo master de K3S está configurado correctamente **When** se inicia el nodo master de K3S **Then** el master arranca correctamente, generando el fichero ./shared/tokenk3s; Podemos ejecutar comandos de kubectl si nos conectamos por ssh a la máquina virtual;
Scenario	Iniciar correctamente el nodo “esclavo” de K3S
	Given que el nodo “esclavo” de K3S está configurado correctamente **When** se inicia el nodo “esclavo” de K3S **Then** el nodo arranca correctamente;
Scenario	Iniciar correctamente Mosquitto
	Given que Mosquitto está configurado correctamente **When** se inicia Mosquitto **Then** Mosquitto se inicia correctamente; ejecutando en la máquina host los scripts ./DevicesSimulation/consumidorPrueba.py y

	./DevicesSimulation/app.py en python3 vemos que el mensaje generado en app.py es recibido por consumidorPrueba;
Scenario	Iniciar correctamente los dispositivos simulados en el clúster de K3S
	Given que el clúster K3S está configurado correctamente **When** se inician los dispositivos simulados en el clúster de K3S **Then** todos los dispositivos simulados se inician correctamente en el clúster de K3S

4.4.2. Pruebas Integradas

Feature:	Conformación del cluster de K3S
	Como desarrollador Quiero que los nodos del cluster de K3S se interconecten formando un cluser gestionado por el nodo master Para que mi entorno se parezca al de producción lo máximo posible
Scenario	Iniciar correctamente el clúster K3S
	Given que el clúster K3S está configurado correctamente **When** se inicia el clúster K3S **Then** todos los nodos del clúster K3S se inician correctamente

Feature:	Pruebas de integración del K3S y Mosquitto
	Como miembro del equipo de desarrollo **Quiero** asegurarme de que el stack ELK y el clúster K3S funcionen correctamente juntos **Para** garantizar la correcta integración de las tecnologías en el entorno de desarrollo
Scenario	Enviar y procesar datos correctamente desde los dispositivos simulados en K3S
	Given que el clúster K3S y Mosquitto están configurados y conectados correctamente y se generan datos de prueba desde los dispositivos simulados **When** los datos se envían a Mosquitto **Then** el mensaje se recibe en el tópico dispositivos/{idGenerado} y un suscriptor lo puede recibir;

Feature:	Pruebas de integración entre Mosquitto y Filebeat
	Como miembro del equipo de desarrollo **Quiero** asegurarme de que Mosquitto y Filebeat se integren correctamente **Para** garantizar la correcta transmisión y procesamiento de datos desde Mosquitto a Filebeat
Scenario	Enviar y procesar datos correctamente desde los dispositivos simulados en K3S
	Given que el clúster K3S y Mosquitto están configurados y conectados correctamente y se generan datos de prueba desde los dispositivos simulados **When** los datos se envían a Mosquitto

	Then el mensaje se recibe en el tópico dispositivos/{idGenerado} y un suscriptor lo puede recibir;

4.4.3. Pruebas End to End

Feature:	Pruebas de flujo completo
	Como miembro del equipo de desarrollo Quiero asegurarme de que todos los componentes funcionen correctamente juntos Para garantizar la correcta integración de las tecnologías en el entorno de desarrollo
Scenario	Enviar y visualizar logs correctamente en Kibana
	Given que el stack ELK, Mosquitto, Filebeats y el clúster K3S están configurados y conectados correctamente y se generan logs de prueba **When** los logs se envían desde los dispositivos simulados a Mosquitto **Then** los logs se visualizan correctamente en Kibana

5. Líneas futuras

5.1. Ampliación funcional

5.1.1. Implementación de AIOPS en el análisis de logs

Se plantea como línea de ampliación en el aspecto de ampliación funcional, la implementación de AIOPS en el proyecto.

La implementación de AIOPS (Inteligencia Artificial para Operaciones de IT) en un proyecto de IoT que utiliza ELK para la monitorización brinda una gran ventaja en términos de aprovechamiento de las posibilidades de IA que ofrece este framework.

ELK permite la recopilación, análisis y visualización de datos en tiempo real. Al combinar AIOPS con ELK, se potencia la capacidad de procesamiento y análisis de datos en un entorno de IoT, permitiendo la detección temprana de anomalías, la optimización de recursos, el mantenimiento predictivo y la toma de decisiones basada en información en tiempo real. La IA aplicada a través de ELK puede identificar patrones, correlaciones y tendencias ocultas en los datos generados por los dispositivos IoT, proporcionando información valiosa para mejorar la eficiencia, la seguridad y la experiencia del usuario.

5.2. Mejoras en la experiencia de usuario

5.2.1. Despliegue de dashboard para interacción con K3S

Para mejora la experiencia de usuario de los desarrolladores de nuestro entorno virtual, una de las posibles mejoras es la instalación y despliegue del dashboard de kubernetes sobre nuestro cluster de K3S.

Para ello podemos instalar el dashboard de kubernetes u otro de los muchos que se ofertan en el mercado, tanto de código abierto como privativos, como son; Portainer, Rancher o Kubeapps, entre otros.

5.3. Seguridad y privacidad

Una posible línea de ampliación para mejorar la seguridad de los sistemas en nuestra arquitectura consiste en implementar certificados SSL/TLS válidos y confiables en todos los componentes que actualmente carecen de ellos o que utilizan certificados autofirmados. Esta mejora garantizaría una comunicación segura y encriptada entre los diferentes elementos de la arquitectura, protegiendo los datos en tránsito y reduciendo el riesgo de ataques de tipo “Man in the middle” o suplantación de identidad. Al adoptar certificados SSL/TLS emitidos por una autoridad de certificación reconocida, se fortalecerá la confianza y la integridad de los sistemas, brindando una capa adicional de seguridad en toda la infraestructura.

Esto es algo que se debe haber contemplado sí o sí en el entorno productivo, por lo que adoptar esta mejora, hará que nuestro entorno de desarrollo y productivo sean más similares. Reduciendo los posibles errores, como hemos comentado durante este trabajo.

Para adoptar este enfoque desde un punto de vista opensource (Código abierto), podemos apoyarnos en el certbot del proyecto Let’s Encrypt. Certbot es una herramienta de software de código abierto utilizada para la gestión automatizada de certificados

SSL/TLS. Su principal objetivo es facilitar el proceso de obtención, renovación e instalación de certificados SSL/TLS válidos y confiables en servidores web. Certbot es desarrollado por la Electronic Frontier Foundation (EFF) y es parte del proyecto Let's Encrypt, que es una autoridad de certificación sin ánimo de lucro que ofrece certificados SSL/TLS gratuitos.

6. Bibliografía

1. **Emnify.** Emnify.com. [En línea] 10 de Enero de 2023. [Citado el: 03 de Junio de 2023.] https://www.emnify.com/blog/iot-challenges-2023.

2. **Atlassian.** atlassian.com. [En línea] https://www.atlassian.com/es/devops/what-is-devops#:~:text=La%20premisa%20clave%20de%20DevOps,ciclo%20de%20desarrollo%20e%20implementaci%C3%B3n..

3. **Kubernetes.** kubernetes.io. [En línea] https://kubernetes.io/docs/reference/kubectl/.

4. **The HiveMQ Team.** hivemq.com. [En línea] 16 de Febrero de 2015. [Citado el: 03 de Junio de 2023.] https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels/.

5. **Mosquitto.** mosquitto.org. [En línea] [Citado el: 03 de Junio de 2023.] https://mosquitto.org/man/mosquitto-tls-7.html.

6. **Steve.** steves-internet-guide. [En línea] 14 de Febrero de 2023. http://www.steves-internet-guide.com/mosquitto-tls/.

Anexos

I. Código

Vagrantfile

El código del vagrantfile realiza las siguientes acciones:

1. Se define una máquina virtual con nombre "elastic" que se configura con la imagen base "ubuntu/xenial64". Se asigna una dirección IP privada de "192.168.22.11". También se configura el forward de puertos desde el puerto 9200 de la máquina host al puerto 9200 de la máquina virtual para que Kibana u otros usuarios vía API puedan conectarse. Se sincroniza la carpeta local "./shared/" con la ruta "/sharedData" dentro de la máquina virtual que utilizaremos para volcar el token de conexión de Kibana y la contraseña del usuario “elastic”. Finalmente, se provisiona la máquina virtual ejecutando un script de shell llamado "ElasticsearchProvision.sh" que se encargará de la instalación y configuración de Elasticsearch.

2. Se define otra máquina virtual llamada "kibana" con configuraciones similares a la máquina "elastic", pero con la siguiente dirección IP diferente del rango ("192.168.22.12"). También se configura el reenvío del puerto 5601 del host al puerto 5601 de la máquina virtual para permitir exponer el frontal web de kibana a través de dicho puerto. Al igual que antes, se sincroniza una carpeta local y se provisiona la máquina virtual con un script de shell llamado "KibanaProvision.sh" que se encarga de la instalación y configuración de kibana.

3. Se define la máquina virtual "logstash". Esta vez se especifican algunas configuraciones adicionales dentro del bloque **logstash.vm.provider "virtualbox"** para ajustar la memoria asignada y los recursos de CPU con el objetivo de que, en caso de que quisiéramos instalar algún plugin pudiéramos, dado que con menos recursos no es posible ejecutar el comando “bin/logstash-plugin install <plugin_name>”. La máquina virtual se configura con la dirección IP "192.168.22.13" y se reenvían los puertos desde el 5044 del host al puerto 5044 de la máquina virtual para la recepción de eventos de logstash. También se sincroniza una carpeta local y se provisiona la máquina virtual con un script de shell llamado "logstashProvision.sh".

4. Se define la máquina "mosquitto" con la dirección IP "192.168.22.20". Se reenvían los puertos desde el puerto 1883 de la máquina host al puerto 1883 de la máquina virtual donde se encuentra escuchando el servidor MQTT, con el objetivo de que los dispositivos puedan enviar mensajes y suscribirse a tópicos. Se sincroniza una carpeta local y se provisiona la máquina virtual con dos scripts de shell, "mosquittoProvision.sh" y "beatstologstashProvision.sh" que instalaran y configurara, mosquitto y filebeats respectivamente.

5. Se define una máquina virtual "kubemaster" que se utilizará como nodo maestro para Kubernetes (K3S). Se asigna una dirección IP privada de "192.168.22.30". Se reenvían los puertos desde el puerto 6443 y el puerto 8001 del huésped a los

mismos puertos de la máquina virtual. Se sincroniza una carpeta local y se provisiona la máquina virtual con un script de shell llamado "masterNode.sh".

6. Se definen dos máquinas virtuales adicionales llamadas "kubeworker1" y "kubeworker2". Ambas máquinas virtuales se configuran de manera similar a "kubemaster" pero con diferentes direcciones IP ("192.168.22.31" y "192.168.22.32" respectivamente). También se sincronizan carpetas locales y se provisionan las máquinas virtuales con un script de shell llamado "workerNode.sh".

En resumen, este código de Vagrant configura varias máquinas virtuales con diferentes roles y configuraciones de red, sincronización de carpetas y provisión de scripts de shell. Se puede utilizar para crear un entorno de desarrollo o despliegue de aplicaciones que incluya Elasticsearch, Kibana, Logstash, Mosquitto y un clúster Kubernetes utilizando K3S.

```
Vagrant.configure("2") do |config|
    #Configuramos la MV de Elastic:
    config.vm.define "elastic" do |elastic|
        elastic.vm.box = "ubuntu/xenial64"
        elastic.vm.network "private_network", ip: "192.168.22.11"
        elastic.vm.network :forwarded_port, guest: 9200, host: 9200
        elastic.vm.synced_folder "./shared/", "/sharedData"
        elastic.vm.provision "shell", privileged: false,  path:
"ElasticsearchProvision.sh"
    end
    #Configuramos la MV de Kibana:
    config.vm.define "kibana" do |kibana|
        kibana.vm.box = "ubuntu/xenial64"
        kibana.vm.network "private_network", ip: "192.168.22.12"
        kibana.vm.network :forwarded_port, guest: 5601, host: 5601
        kibana.vm.synced_folder "./shared/", "/sharedData"
        kibana.vm.provision "shell",  path: "KibanaProvision.sh"
    end
    #Configuramos la MV de Logstash:
    config.vm.define "logstash" do |logstash|
        logstash.vm.box = "ubuntu/xenial64"
        logstash.vm.provider "virtualbox" do |vb|
            vb.memory = "2048"
            vb.cpus = 2
        end
        logstash.vm.network "private_network", ip: "192.168.22.13"
        logstash.vm.network :forwarded_port, guest: 5044, host: 5044
        logstash.vm.synced_folder "./shared/", "/sharedData"
        logstash.vm.provision "shell", privileged: false, path:
"logstashProvision.sh"
    end
    #Configuramos la MV de Mosquitto y como agente ligero Filebeat:
    config.vm.define "mosquitto" do |mosquitto|
        mosquitto.vm.box = "ubuntu/xenial64"
        mosquitto.vm.network "private_network", ip: "192.168.22.20"
        mosquitto.vm.network :forwarded_port, guest: 1883, host: 1883
        mosquitto.vm.synced_folder "./shared/", "/sharedData"
        mosquitto.vm.provision "shell", path: "mosquittoProvision.sh"
        mosquitto.vm.provision "shell", path: "beatstologstashProvision.sh"
    end
    #Configuramos la MV de K3S que hará de master:
    config.vm.define "kubemaster" do |kubemaster|
        kubemaster.vm.box = "ubuntu/xenial64"
        kubemaster.vm.network "forwarded_port", guest: 6443, host: 6443
        kubemaster.vm.network "forwarded_port", guest: 8001, host: 8001
        kubemaster.vm.network "private_network", ip: "192.168.22.30"
```

```
        kubemaster.vm.synced_folder "./shared/", "/sharedData"
        kubemaster.vm.provision "shell", privileged: false, path:
"masterNode.sh"
    end
    #Configuramos la MV de K3S que hará de primer worker:
    config.vm.define "kubeworker1" do |kubeworker1|
        kubeworker1.vm.box = "ubuntu/xenial64"
        kubeworker1.vm.hostname = "node1"
        kubeworker1.vm.network "private_network", ip: "192.168.22.31"
        kubeworker1.vm.synced_folder "./shared/", "/sharedData"
        kubeworker1.vm.provision "shell", privileged: false, path:
"workerNode.sh"
    end
    #Configuramos la MV de K3S que hará de segundo worker:
    config.vm.define "kubeworker2" do |kubeworker2|
        kubeworker2.vm.box = "ubuntu/xenial64"
        kubeworker2.vm.hostname = "node2"
        kubeworker2.vm.network "private_network", ip: "192.168.22.32"
        kubeworker2.vm.synced_folder "./shared/", "/sharedData"
        kubeworker2.vm.provision "shell", privileged: false, path:
"workerNode.sh"
    end
end
```

ElasticsearchProvision.sh

El código de provisión de elasticsearch realiza las siguientes acciones:

1. Actualiza la lista de paquetes disponibles utilizando apt-get update.
2. Instala el paquete openjdk-8-jre evitando que solicite confirmación (-y, que escribe en el stdin y constantemente durante la ejecución del comando al que acompaña) utilizando apt-get install.
3. Descargamos la clave GPG del repositorio de Elasticsearch y se guarda en /usr/share/keyrings/elasticsearch-keyring.gpg.
4. Instala el paquete apt-transport-https para permitir el uso de repositorios HTTPS en APT.
5. Agrega el repositorio de Elasticsearch (https://artifacts.elastic.co/packages/8.x/apt) al archivo /etc/apt/sources.list.d/elastic-8.x.list, utilizando la clave GPG almacenada previamente.
6. Actualiza la lista de paquetes nuevamente para incluir el repositorio de Elasticsearch recién agregado.
7. Instala la versión 8.5.3 de Elasticsearch utilizando apt-get install.
8. Recarga el daemon de systemd para que reconozca los cambios realizados en los archivos de configuración del servicio.
9. Habilita el servicio de Elasticsearch para que se inicie automáticamente al arrancar el sistema, utilizando systemctl enable.
10. Inicia el servicio de Elasticsearch utilizando systemctl start.
11. Ejecuta el comando elasticsearch-reset-password para restablecer la contraseña del usuario "elastic". La contraseña generada se guarda en la variable fullPass.
12. Extrae los últimos 20 caracteres de la variable fullPass y los asigna a la variable pass dado que el resultado del comando bin/elasticsearch-reset-password

devuelve “Changed password for user [#user#] PASSWORD #user# =” antes de la password, que es de 20 caracteres por defecto.

13. Guarda la contraseña en un archivo llamado "admin" en la carpeta compartida "/sharedData".
14. Ejecuta el comando elasticsearch-service-tokens create para crear un token de servicio para el usuario "elastic" y el rol "kibana". El token generado se guarda en la variable fullToken. Es importante no ejecutar este comando como sudo o en un script privilegiado, puesto que daría conflicto de permisos al arrancar Elastic posteriormente.
15. Extrae los últimos 64 caracteres de la variable fullToken y los asigna a la variable token, análogamente al caso de la contraseña, el comando devuelve un texto previo al token.
16. Guarda el token en el archivo "token" en la carpeta compartida "/sharedData".
17. Cambia los permisos del directorio /etc/elasticsearch/service_tokens para permitir que el usuario elasticsearch tenga acceso de lectura y escritura al archivo de tokens.

```
sudo apt-get update
sudo apt-get install -y openjdk-8-jre
wget -qO - https://artifacts.elastic.co/GPG-KEY-elasticsearch | sudo gpg --
dearmor -o /usr/share/keyrings/elasticsearch-keyring.gpg
sudo apt-get install apt-transport-https
echo "deb [signed-by=/usr/share/keyrings/elasticsearch-keyring.gpg]
https://artifacts.elastic.co/packages/8.x/apt stable main" | sudo tee
/etc/apt/sources.list.d/elastic-8.x.list
sudo apt-get update && sudo apt-get install elasticsearch=8.5.3 -y
sudo /bin/systemctl daemon-reload
sudo /bin/systemctl enable elasticsearch.service
sudo systemctl start elasticsearch.service
fullPass=$(yes | sudo /usr/share/elasticsearch/bin/elasticsearch-reset-
password -u elastic)
pass=${fullPass: -20}
#Utilizamos para albergar la contraseña y el token un fichero en una synced
folder para poder usarlo en otra Máquina virtual.
echo $pass>/sharedData/admin
#Es importante no ejecutar este comando como root dado que si ejecutamos la
creación del token como tal, el fichero service_tokens no cuenta con
permisos de lectura para el usuario elasticsearch y por tanto no levanta el
servicio, como alternativa si se muestra el error
/etc/default/elasticsearch: permission denied, podemos ejecutarlo como root
y cambiar los permisos de service_tokens más adelante.
fullToken=$(sudo /usr/share/elasticsearch/bin/elasticsearch-service-tokens
create elastic/kibana kibana)
token=${fullToken: -64}
echo $token>/sharedData/token
sudo chmod -R 660 /etc/elasticsearch/service_tokens
```

KibanaProvision.sh

Este código realiza una serie de pasos para instalar y configurar Kibana en un sistema Linux. A continuación, se explica qué hace cada línea:

1. Descarga e importa la clave de firma GPG de ElasticSearch. El comando wget se utiliza para descargar el contenido del URL https://artifacts.elastic.co/GPG-KEY-elasticsearch. Luego, el comando gpg se utiliza con la opción --dearmor para convertir la clave en un formato que se pueda utilizar para autenticar paquetes.

Finalmente, la salida se redirige al archivo /usr/share/keyrings/elasticsearch-keyring.gpg.

2. Actualiza los repositorios de paquetes utilizando el comando apt-get update. Esto asegura que la información más reciente sobre los paquetes esté disponible en el sistema.

3. Instala el paquete apt-transport-https utilizando el comando apt-get install. Este paquete permite que el sistema utilice el protocolo HTTPS para descargar paquetes.

4. Agrega la fuente de paquetes de ElasticSearch al archivo /etc/apt/sources.list.d/elastic-8.x.list. El comando echo se utiliza para agregar una línea de texto al archivo. La línea de texto especifica la ubicación de la fuente de paquetes (https://artifacts.elastic.co/packages/8.x/apt), el archivo de clave (/usr/share/keyrings/elasticsearch-keyring.gpg) y la rama principal (stable main).

5. Actualiza los repositorios de paquetes nuevamente para que el sistema tenga en cuenta la nueva fuente de paquetes.

6. Instala Kibana 8.5.3 utilizando el comando apt-get install kibana=8.5.3 -y. El parámetro -y se utiliza para confirmar automáticamente cualquier solicitud de confirmación durante la instalación.

7. Lee el contenido del archivo /sharedData/token y asigna su valor a la variable token.

8. Agrega varias configuraciones al archivo de configuración de Kibana (/etc/kibana/kibana.yml). Utilizando el comando sudo echo, se agregan las siguientes líneas de configuración:

    - server.host: "192.168.22.12": Especifica la dirección IP en la que Kibana escuchará las conexiones entrantes.
    - server.port: 5601: Especifica el puerto en el que Kibana estará disponible.
    - elasticsearch.hosts: "https://192.168.22.11:9200": Especifica la dirección IP y el puerto de ElasticSearch al que Kibana se conectará.
    - elasticsearch.serviceAccountToken: "$token": Especifica el token de la cuenta de servicio para autenticarse con ElasticSearch.
    - elasticsearch.ssl.verificationMode: "none": Desactiva la verificación del certificado SSL al conectarse a ElasticSearch.

9. Recarga los archivos de configuración del sistema utilizando systemctl daemon-reload. Esto asegura que los cambios realizados en los archivos de configuración anteriores tengan efecto.

10. Habilita el servicio de Kibana para que se inicie automáticamente al arrancar el sistema utilizando systemctl enable kibana.service.

11. Inicia el servicio de Kibana utilizando systemctl start kibana.service.

```
wget -qO - https://artifacts.elastic.co/GPG-KEY-elasticsearch | sudo
gpg --dearmor -o /usr/share/keyrings/elasticsearch-keyring.gpg
sudo apt-get update
sudo apt-get install apt-transport-https
echo "deb [signed-by=/usr/share/keyrings/elasticsearch-keyring.gpg]
https://artifacts.elastic.co/packages/8.x/apt stable main" | sudo
tee /etc/apt/sources.list.d/elastic-8.x.list
sudo apt-get update && sudo apt-get install kibana=8.5.3 -y
token=$(cat /sharedData/token)
sudo echo "server.host: \"192.168.22.12\"">/etc/kibana/kibana.yml
sudo echo "server.port: 5601">>/etc/kibana/kibana.yml
sudo echo "elasticsearch.hosts:
\"https://192.168.22.11:9200\"">>/etc/kibana/kibana.yml
sudo echo "elasticsearch.serviceAccountToken:
\"$token\"">>/etc/kibana/kibana.yml
sudo echo "elasticsearch.ssl.verificationMode:
\"none\"">>/etc/kibana/kibana.yml
sudo /bin/systemctl daemon-reload
sudo /bin/systemctl enable kibana.service
sudo systemctl start kibana.service
```

logstashProvision.sh

Este cñodigo descarga Logstash mediante la realización de los siguientes pasos:

1. Descarga la clave de firma GPG de ElasticSearch utilizando el comando wget. El parámetro -qO - indica que se debe mostrar la salida en la salida estándar sin guardarla en un archivo. La salida se pasa al comando apt-key add - para agregar la clave a la lista de claves confiables.

2. Instala el paquete apt-transport-https utilizando el comando apt-get install. Este paquete permite que el sistema utilice el protocolo HTTPS para descargar paquetes.

3. Agrega la fuente de paquetes de Logstash al archivo /etc/apt/sources.list.d/elastic-8.x.list. El comando echo se utiliza para agregar una línea de texto al archivo. La línea de texto especifica la ubicación de la fuente de paquetes (https://artifacts.elastic.co/packages/8.x/apt), y la rama principal (stable main).

4. Actualiza los repositorios de paquetes utilizando el comando apt-get update. Esto asegura que la información más reciente sobre los paquetes esté disponible en el sistema.

5. Instala Logstash 8.5.3 utilizando el comando apt-get install logstash=1:8.5.3-1 -y. El parámetro -y se utiliza para confirmar automáticamente cualquier solicitud de confirmación durante la instalación.

6. Lee el contenido del archivo /sharedData/admin y asigna su valor a la variable pass.

7. Copia el archivo /sharedData/pipelines.yml al directorio /etc/logstash/. Esto se realiza utilizando el comando sudo cp, que copia el archivo manteniendo los mismos permisos y propietarios.
8. Copia el pipeline /sharedData/beatstoelastic.config al directorio /etc/logstash/.
9. Utiliza el comando sed para reemplazar la cadena #PASSWORD# por el valor de la variable pass en el pipeline situado en el archivo /etc/logstash/beatstoelastic.config.
10. Cambia el propietario del directorio /usr/share/logstash al usuario y grupo logstash.logstash utilizando el comando sudo chown.
11. Concede permisos de escritura y ejecución al directorio /usr/share/logstash/data utilizando el comando sudo chmod 777.
12. Recarga los archivos de configuración del sistema utilizando systemctl daemon-reload. Esto asegura que los cambios realizados en los archivos de configuración tengan efecto.
13. Habilita el servicio de Logstash para que se inicie automáticamente al arrancar el sistema utilizando systemctl enable logstash.service.
14. Inicia el servicio de Logstash utilizando systemctl start logstash.service.

```
wget -qO - https://artifacts.elastic.co/GPG-KEY-elasticsearch | sudo
apt-key add -
sudo apt-get install apt-transport-https
echo "deb https://artifacts.elastic.co/packages/8.x/apt stable main"
| sudo tee -a /etc/apt/sources.list.d/elastic-8.x.list
sudo apt-get update && sudo apt-get install logstash=1:8.5.3-1 -y
pass=$(cat /sharedData/admin)
sudo cp /sharedData/pipelines.yml /etc/logstash/pipelines.yml
sudo cp /sharedData/beatstoelastic.config
/etc/logstash/beatstoelastic.config
sudo sed -i "s/#PASSWORD#/$pass/g"
/etc/logstash/beatstoelastic.config
sudo chown -R logstash.logstash /usr/share/logstash
sudo chmod 777 /usr/share/logstash/data
sudo /bin/systemctl daemon-reload
sudo /bin/systemctl enable logstash.service
#instalamos el plugin de mqtt, se descarta por decisión de diseño
#sudo /usr/share/logstash/bin/logstash-plugin install logstash-
input-paho-mqtt
sudo systemctl start logstash.service
```

mosquittoProvision.sh

Para la instalación de Mosquitto, hacemos uso del siguiente script. El script realiza los siguientes pasos:

1. Actualiza los repositorios de paquetes utilizando el comando apt update. Esto asegura que la información más reciente sobre los paquetes esté disponible en el sistema.
2. Instala la versión 11 del entorno de ejecución Java (JRE) de OpenJDK utilizando el comando apt install openjdk-11-jre -y. El parámetro -y se utiliza para confirmar automáticamente cualquier solicitud de confirmación durante la instalación.
3. Instala el paquete mosquitto utilizando el comando apt-get install mosquitto -y. Este paquete es el broker MQTT que se instalará en el sistema.
4. Instala la utilidad mosquitto-clients utilizando el comando apt-get install mosquitto-clients -y. Esta utilidad proporciona herramientas de línea de comandos para interactuar con el broker MQTT.
5. Recarga los archivos de configuración del sistema utilizando systemctl daemon-reload. Esto asegura que los cambios realizados en los archivos de configuración tengan efecto.
6. Habilita el servicio de Mosquitto para que se inicie automáticamente al arrancar el sistema utilizando systemctl enable mosquitto.service.
7. Inicia el servicio de Mosquitto utilizando systemctl start mosquitto.service.

```
sudo apt update
sudo apt install openjdk-11-jre -y
sudo apt-get install mosquitto -y
sudo apt-get install mosquitto-clients -y
sudo /bin/systemctl daemon-reload
sudo systemctl enable mosquitto.service
sudo systemctl start mosquitto.service
```

beatstologstashProvision.sh

Para la instalación de Filebeat en el mismo server/maquina virtual que mosquitto, usamos este script. El código realiza las siguientes acciones:

1. Descarga la clave de firma GPG de ElasticSearch utilizando el comando wget. El parámetro -qO - indica que se debe mostrar la salida en la salida estándar sin guardarla en un archivo. La salida se pasa al comando apt-key add - para agregar la clave a la lista de claves confiables del sistema.
2. Instala el paquete apt-transport-https utilizando el comando apt-get install. Este paquete permite que el sistema utilice el protocolo HTTPS para descargar paquetes.
3. Agrega la fuente de paquetes de Filebeat al archivo /etc/apt/sources.list.d/elastic-8.x.list. El comando echo se utiliza para agregar una línea de texto al archivo. La línea de texto especifica la ubicación de la fuente de paquetes (https://artifacts.elastic.co/packages/8.x/apt), y la rama principal (stable main).

4. Actualiza los repositorios de paquetes utilizando el comando apt-get update. Esto asegura que la información más reciente sobre los paquetes esté disponible en el sistema.
5. Instala una versión específica de Filebeat (8.5.3) utilizando el comando apt-get install filebeat=8.5.3. Los paquetes necesarios se descargan e instalan en el sistema.
6. Copia el archivo /sharedData/filebeat.yml al directorio /etc/filebeat/. Esto se realiza utilizando el comando sudo cp, que copia el archivo manteniendo los mismos permisos y propietarios.
7. Lee el contenido del archivo /sharedData/admin y asigna su valor a la variable pass.
8. Utiliza el comando sed para reemplazar la cadena #PASSWORD# por el valor de la variable pass en el archivo /etc/filebeat/filebeat.yml.
9. Recarga los archivos de configuración del sistema utilizando systemctl daemon-reload. Esto asegura que los cambios realizados en los archivos de configuración tengan efecto.
10. Habilita el servicio de Filebeat para que se inicie automáticamente al arrancar el sistema utilizando systemctl enable filebeat.
11. Inicia el servicio de Filebeat utilizando systemctl start filebeat.

```
wget -qO - https://artifacts.elastic.co/GPG-KEY-elasticsearch | sudo
apt-key add -
sudo apt-get install apt-transport-https
echo "deb https://artifacts.elastic.co/packages/8.x/apt stable main"
| sudo tee -a /etc/apt/sources.list.d/elastic-8.x.list
sudo apt-get update && sudo apt-get install filebeat=8.5.3
sudo cp /sharedData/filebeat.yml /etc/filebeat/filebeat.yml
pass=$(cat /sharedData/admin)
sudo sed -i "s/#PASSWORD#/$pass/g" /etc/filebeat/filebeat.yml
sudo systemctl daemon-reload
sudo systemctl enable filebeat
sudo systemctl start filebeat
```

masterNode.sh

Para el despliegue de K3S y los dispositivos simulados:

1. Genera un token aleatorio para usar con K3s. Esto se realiza utilizando el comando openssl rand -base64 12. El token generado se guarda en la variable k3stoken.
2. Guarda el valor de k3stoken en un archivo llamado k3stoken ubicado en la carpeta /sharedData/. Esto se realiza mediante el comando echo $k3stoken > /sharedData/k3stoken.

3. Descarga e instala K3s en la máquina utilizando el comando curl. La URL https://get.k3s.io se utiliza para obtener el script de instalación de K3s. El script se ejecuta con varios parámetros, incluyendo K3S_TOKEN que se establece con el valor de k3stoken, K3S_KUBECONFIG_MODE que se establece en "644" para establecer los permisos correctos en el archivo de configuración, y otros parámetros como --cluster-init, --node-name, --with-node-id, --tls-san y --node-ip que configuran diferentes aspectos del servidor K3s.
4. Se despliegan una imagen de los dispositivos en el clúster Kubernetes utilizando el comando kubectl create deployment. El nombre de la imagen de la aplicación se establece en alfonsoiot/devices-tfm y se asigna el nombre de devices al despliegue.
5. Se escala el número de réplicas del despliegue a 3 utilizando el comando kubectl scale. Esto significa que se crearán tres réplicas de la aplicación en el clúster.

```
k3stoken=$(openssl rand -base64 12)
echo $k3stoken>/sharedData/k3stoken
curl -sfL https://get.k3s.io | K3S_TOKEN=$k3stoken
K3S_KUBECONFIG_MODE="644" sh -s - server --cluster-init --node-name
master --with-node-id --tls-san "192.168.22.30" --node-ip
192.168.22.30
#Despliegue de imagen de prueba del entorno, con 3 réplicas:
kubectl create deployment --image=alfonsoiot/devices-tfm devices
kubectl scale --replicas=3 deployment/devices
```

workerNode.sh

Análogamente, desplegamos cada nodo de forma similar, cambiando los parámetros de la instalación para incluir el nodo maestro en el parámetro K3S_URL.

```
k3stoken=$(cat /sharedData/k3stoken)
curl -sfL https://get.k3s.io | K3S_URL=https://192.168.22.30:6443
K3S_TOKEN="$k3stoken" sh -
```

App.py

El dispositivo simulado cuenta con el siguiente código para su ejecución. El código hace lo siguiente:

1. Importa los módulos necesarios: **json**, **random**, **time**, **paho.mqtt.client** y **uuid**.
2. Generamos un ID único basado en UUID para identificar el dispositivo.
3. Configuramos los parámetros del cliente MQTT, como la dirección del broker, el puerto y el topic al que se enviarán los mensajes. El topic se genera concatenando "dispositivos/" con el ID del dispositivo.
4. Definimos la función de callback **on_connect** que se ejecuta cuando el cliente se conecta al broker MQTT. En este caso, simplemente muestra un mensaje de que se ha conectado al broker y se suscribe al topic.

5. Define una función de callback **on_publish** que se ejecuta después de publicar un mensaje MQTT. En este caso, muestra un mensaje de que el mensaje se ha publicado con éxito.
6. Crea una instancia del cliente MQTT.
7. Configura los callbacks del cliente MQTT.
8. Inicia un bucle infinito para enviar mensajes de forma continua.
9. Genera un diccionario que simula los datos de un dispositivo. Incluye campos como el ID del dispositivo, la temperatura (generada aleatoriamente), el estado de habilitación (generado aleatoriamente) y la marca de tiempo actual.
10. Convierte el diccionario en una cadena JSON utilizando **json.dumps()**.
11. Conecta al cliente MQTT al broker con un tiempo de conexión de 60 segundos.
12. Publica el mensaje JSON en el topic MQTT utilizando **client.publish()**.
13. Imprime el mensaje JSON y el topic en la consola para verificar el envío.
14. Espera 60 segundos antes de enviar el siguiente mensaje para evitar sobrecargar el broker.

```
import json
import random
import time
import paho.mqtt.client as mqtt
import uuid
from datetime import datetime

#Generamos un id único, basado en uuid
deviceId = uuid.uuid4()
# Configuración del cliente MQTT
broker = "192.168.22.20"  # Dirección del broker MQTT
port = 1883  # Puerto del broker MQTT
topic = "dispositivos/"+str(deviceId)  # Topic MQTT al que se
enviará el mensaje, generamos una jerarquia para mejorar el
tratamiento

# Callback que se ejecuta al conectarse al broker MQTT
def on_connect(client, userdata, flags, rc):
    print("Conectado al broker MQTT")
    # Suscribirse al topic después de la conexión
    client.subscribe(topic)

# Callback que se ejecuta al publicar un mensaje MQTT
def on_publish(client, userdata, mid):
    print("Mensaje publicado con éxito")

# Creamos la instancia de MQTT
client = mqtt.Client()

# Configuramos los callbacks
client.on_connect = on_connect
```

```
client.on_publish = on_publish
#Bucle infinito para enviar los mensajes de forma ilimitada
while True:
    # Generar un diccionario con múltiples campos que simulen ser un
dispositivo
    now = datetime.now()
    message = {
        "deviceId": str(deviceId),
        "temperature": random.randint(1, 100),
        "isEnable": random.uniform(0, 1),
        "timestamp": now.strftime("%Y-%m-%dT%H:%M:%S.%fZ")
    }
    # Convertir el diccionario a formato JSON
    message_json = json.dumps(message)
    # Conectamos al broker MQTT con tiempo de conexión de 30
segundos
    client.connect(broker, port, keepalive=60)

    # Publicamos el mensaje en el topic MQTT
    client.publish(topic, message_json)
    print(message_json + " on topic "+ topic)
    # Esperamos 60 segundos para no bombardear al broker
    time.sleep(60)
```

Dockerfile

El dockerfile se encarga de que cuando ejecutemos el comando “Docker build -t”, se construya correctamente la imagen, mediante la descripción de una serie de comandos, en este caso:

1. FROM python:3.9: Esta instrucción establece la imagen base a utilizar, en este caso, la imagen oficial de Python versión 3.9. Esta imagen proporciona un entorno preconfigurado con Python instalado.
2. WORKDIR /usr/app: Esta instrucción establece el directorio de trabajo dentro del contenedor donde se copiarán los archivos de la aplicación y donde se ejecutarán los comandos. En este caso, el directorio de trabajo se establece en /usr/app.
3. COPY . .: Esta instrucción copia todos los archivos y directorios del contexto de construcción actual (el directorio donde se encuentra el Dockerfile) al directorio de trabajo del contenedor. Esto incluye el código fuente de la aplicación y el archivo requirements.txt.
4. RUN pip3 install -r requirements.txt: Esta instrucción ejecuta, dentro del contenedor, el comando pip3 install para instalar las dependencias de la aplicación especificadas en el archivo requirements.txt. Este fichero contiene las siguientes dependencias:
    1. paho-mqtt
    2. uuid
    3. datetime

5. CMD ["python", "app.py"]: Esta instrucción especifica el comando predeterminado que se ejecutará cuando se inicie un contenedor basado en esta imagen. En este caso, se ejecutará el archivo app.py utilizando el intérprete de Python. Esto asume que app.py es el punto de entrada principal de la aplicación.

```
FROM python:3.9
WORKDIR /usr/app
COPY . .
RUN pip3 install -r requirements.txt
CMD [ "python", "app.py" ]
```

II. Archivos de configuración:

Pipelines.yml

```
- pipeline.id: beatstoelastic
  path.config: "/etc/logstash/beatstoelastic.config"
  pipeline.workers: 3
```

Filebeat.yml

```
filebeat.inputs:
- type: mqtt
  hosts:
    - 'tcp://127.0.0.1:1883'
  topics:
    - dispositivos/#
  qos: 2
  client_id: filebeat_mqtt
output.logstash:
  hosts: ["192.168.22.13:5044"]
  ttl: 120
  pipelining: 0
logging.level: info
logging.to_files: true
logging.files:
  path: /var/log/filebeat
  name: filebeat
  keepfiles: 7
  permissions: 0640
```

beatstoelastic.config

```
input {
    beats {
        client_inactivity_timeout => 3000
        port => 5044
        ssl  => false
    }
}
filter {
    mutate {
        add_field => {
```

```
            "nuevo_campo1" => "prueba"
            "nuevo_campo2" => "prueba2"
          }
  }
}
output {
  elasticsearch {
    hosts => ["https://192.168.22.11:9200"]
    index => "edgecluster-new"
    ssl_certificate_verification => false
    user => "elastic"
    password => "#PASSWORD#"
 }
  file {
      path => "/var/log/logstash/output.log"
  }
}
```