**Universidad Politécnica de Madrid**

**Escuela Técnica Superior de Ingeniería de Sistemas Informáticos**

Texto

Descripción generada automáticamente con confianza media

# 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**

## Resumen

El presente proyecto se enfoca en la aplicación de metodología DevOps en el ámbito de Internet de las cosas (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.

En resumen, 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.

## 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.

To sum up, this project aims to apply DevOps methodology in the Internet of Things (IoT) domain using Vagrant in a simulated environment. The deployment of the ELK stack and the K3S server cluster will enable development teams and IoT professionals to explore and evaluate the performance of these technologies in a controlled environment.

## Índice

Contenido

[Monitorización de sistemas IoT en un entorno DevOps 1](#_Toc136134049)

[Resumen 3](#_Toc136134050)

[Abstract 4](#_Toc136134051)

[Índice 5](#_Toc136134052)

[Introducción 6](#_Toc136134053)

[Contexto 6](#_Toc136134054)

[Objetivo 7](#_Toc136134055)

[Justificación 8](#_Toc136134056)

[Fundamentos teóricos: 9](#_Toc136134057)

[IoT 9](#_Toc136134058)

[DevOps 9](#_Toc136134059)

[Vagrant 10](#_Toc136134060)

[ELK Stack 12](#_Toc136134061)

[Kubernetes y K3S 13](#_Toc136134062)

[Mosquitto (MQTT) 15](#_Toc136134063)

[Metodología 17](#_Toc136134064)

[Configuración del entorno simulado: 17](#_Toc136134065)

[Integración de Mosquitto y Logstash: 17](#_Toc136134066)

[Pruebas y validación: 17](#_Toc136134067)

[Resultados Obtenidos 17](#_Toc136134068)

[Líneas de ampliación 17](#_Toc136134069)

[Ampliación funcional: 17](#_Toc136134070)

[Mejoras en la experiencia de usuario: 17](#_Toc136134071)

[Seguridad y privacidad: 17](#_Toc136134072)

[Anexos 17](#_Toc136134073)

[Código 17](#_Toc136134074)

[Referencias 17](#_Toc136134075)

[Glosario 17](#_Toc136134076)

## Introducción

### 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.

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. 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.

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 un entorno DevOps. 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.

### Objetivo

El principal objetivo es desarrollar un conjunto de scripts y piezas de código declarativo que permitan desplegar y configurar un entorno de desarrollo cuyas características sean prácticamente idénticas a un entorno de producción, para 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, al permitir 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 desplieguen 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.

### 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. Estos enfoques permiten una gestión eficiente de entornos de desarrollo, una mayor flexibilidad en la simulación y prueba de dispositivos del Edge de IoT, y una entrega continua y monitoreo en tiempo real. Al desacoplar la comunicación de los dispositivos utilizando Mosquitto y aprovechar la potencia del stack ELK, se logra una arquitectura escalable y modular que impulsa la eficiencia y el rendimiento

## Fundamentos teóricos:

### 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.

### DevOps

DevOps es una metodología y conjunto de prácticas que combina 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:

* **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.

### 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.

#### Ventajas de 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.

### 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.

* **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.
* **Logstash**: Es una herramienta de procesamiento y transformación de datos que se encarga de la ingesta de datos en tiempo real. Logstash permite recopilar logs y otros eventos de diferentes fuentes, como archivos de registro, bases de datos, sistemas de mensajería, entre otros. Luego, realiza una serie de transformaciones en los datos, como filtrado, enriquecimiento y formateo, para prepararlos antes de enviarlos a Elasticsearch.
* **Kibana**: Es una interfaz de usuario 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.

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.

### 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 incluyen:

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. Baja huella de memoria 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.

#### 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.

#### Principales comandos de K3s

[https://kubernetes.io/docs/reference/kubectl/docker-cli-to-kubectl/]

### 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:

1. 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.
2. 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.
3. 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:
   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.

1. 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. [http://www.steves-internet-guide.com/mosquitto-tls/]

## Metodología

### 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.

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 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.

### Planificación de las iteraciones

#### 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.

#### 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.

#### Iteración 3

Alcance:

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 maquinas virtuales definidas en el diseño, al menos 1 master y el numero 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 numero de nodos que hayamos establecido.

#### 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”

#### 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.

#### 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.

### Configuración del entorno simulado:

Descripción detallada de cómo se estableció el entorno simulado utilizando Vagrant, incluyendo la configuración de las máquinas virtuales, la conexión de Mosquitto y Logstash, y la implementación del stack ELK y el clúster de servidores de K3S.

### Integración de Mosquitto y Logstash:

Explicación de cómo se realizó la conexión entre Mosquitto y Logstash utilizando el plugin "logstash-input-mqtt" para la recepción de mensajes MQTT y su posterior procesamiento y envío a los destinos deseados.

### Pruebas y validación:

Descripción de las pruebas realizadas para verificar el funcionamiento correcto del entorno simulado, el procesamiento de los datos y la integración entre los componentes.

## Resultados Obtenidos

Presentación de los resultados obtenidos durante las pruebas y la validación, junto con un análisis de los datos recopilados. Se pueden incluir métricas de rendimiento, eficiencia y escalabilidad, así como ejemplos de visualización y análisis de datos realizados con el stack ELK.

## Líneas de ampliación

### Ampliación funcional:

Incluir AIOPS

### Mejoras en la experiencia de usuario:

Añadir un Dashboard a k3s

### Seguridad y privacidad:

Incluir ssl y verificación del mismo en todos los pasos del proceso de provisión

## Anexos

### Código

### Referencias

### Glosario