

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

Monitorización de sistemas IoT en un entorno DevOps	5
Índice.....	VII
Índice de figuras.....	IX
Glosario de términos.....	X
Resumen	XII
Abstract	XIII
1. Introducción.....	1
1.1. Contexto	1
1.2. Objetivos	3
1.3. Metodología y Planificación	4
1.4. Estructura del documento.....	10
2. Fundamentos teóricos:.....	11
2.1. IoT.....	12
2.2. DevOps	14
2.3. Vagrant	16
2.4. ELK Stack.....	19
2.5. Kubernetes y K3S.....	24
2.6. Mosquitto (MQTT).....	27
2.7. Jenkins	30
3. Descripción de la Plataforma DevOps	33
3.1. Arquitectura de la solución	33
3.2. Decisiones de Diseño	36
3.3. Configuración del entorno	40
3.4. Pruebas y validación.....	45
4. Resultados & Discusión.....	48
5. Conclusiones y Líneas futuras.....	54
5.1. Conclusiones.....	54
5.2. Ampliación funcional.....	55
5.3. Seguridad y privacidad	55
6. Bibliografía	56
Anexos.....	60

I. Código	60
II. Archivos de configuración:.....	78

Índice de figuras

Ilustración 1 - Ilustración descriptiva de la metodología iterativa incremental [Fuente]	4
Ilustración 2 - Cloud Computing vs Edge computer de forma visual. [Fuente]	13
Ilustración 3 - Imagen que describe el ciclo de vida en un entorno DevOps donde se describen las etapas en las que intervienen ambos equipos [Fuente]	14
Ilustración 4 - Imagen representativa de las diferentes fases, análoga a la de DevOps, incluyendo la parte de Seguridad (SEC). [Fuente]	15
Ilustración 5 - Relación entre los componentes de Elasticsearch [Fuente]	21
Ilustración 6 - Landing page de Kibana tras realizar el login	22
Ilustración 7 - Consola dentro de Kibana, desde donde podríamos lanzar peticiones y visualizar los resultados	23
Ilustración 8 - Funcionamiento de un Broker MQTT	28
Ilustración 9 - Formato de un mensaje MQTT con lastWill [Fuente]	30
Ilustración 10 - Definición de la arquitectura de la Plataforma DevOps	34
Ilustración 11 - Dashboard de kibana con datos cargados por los dispositivos. Las tablas representan la temperatura, viento y humedad de Madrid por fecha, así como la relación entre sus medias	35
Ilustración 12 - Vista de ejecución del job configurado para el proyecto en Jenkins, podemos observar una ejecución manual y otra automática, diferenciadas por la detección de commits a la derecha de la fecha de ejecución	36
Ilustración 13 - Paso 1 de la configuración de Jenkins	41
Ilustración 14 - Paso 2 de la configuración de Jenkins	41
Ilustración 15 - Paso 3 de la configuración de Jenkins	42
Ilustración 16 - Paso 4 de la configuración de Jenkins	42
Ilustración 17 - Paso 5 de la configuración de Jenkins	43
Ilustración 18 - Resultado final de la configuración de credenciales en Jenkins	44
Ilustración 19 - Visualización de la pantalla de login del dashboard de kubernetes	44
Ilustración 20 - Visualización del dashboard de kubernetes de la sección de Deployments, con un deployment llamado devices con 3 pods	45
Ilustración 21 - Fecha antes de la ejecución del comando "vagrant up"	49
Ilustración 22 - Fecha tras la ejecución completa del comando vagrant up	50
Ilustración 23 - Evidencia del despliegue de las máquinas virtuales de la plataforma DevOps en VirtualBox	51
Ilustración 24 - Dashboard creado para el proyecto con los datos medidos por los dispositivos (media, máxima y mínima)	52
Ilustración 25 - Dashboard de kubernetes donde se puede visualizar el número de despliegues, los pods y replica sets	53

Glosario de términos

A

API

(Application Programming Interface) Conjunto de funcionalidades que ofrece como servicio para ser utilizado en otras aplicaciones · 22, 60

B

backend

El backend es la parte de un sistema informático que se encarga de la lógica y el procesamiento de datos. · 39

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. · 24

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. · 2, 16, 19, 21, 24, 25, 27, 55

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. · 73

E

ELK

Elasticsearch, Logstash y Kibana, por sus siglas · 6, 19, 20, 38, 40, 46, 47, 55

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. · 55

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. · 60

I

IoT

Internet of Things · 5, 1, 6, 7, 8, 12, 18, 19, 24, 25, 27, 28, 34, 37, 39, 40, 55

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. · 1, 7, 8, 19, 20, 23, 26, 27, 34, 35, 36, 38, 46, 47, 55

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. · 5, 7, 8, 17, 18, 34, 35, 37, 38, 46, 60, 61

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. · 4, 5, 14

MQTT

Message Queing Telemetry Transport, Protocolo de comunicación máquina a máquina mediante el envío de mensajes. · 7, 27, 28, 29, 34, 37, 38, 39, 40, 61

Responde a solicitudes y gestiona recursos. · 26, 27, 28, 29, 39, 61, 72

stack

conjunto de tecnologías, herramientas o componentes que se utilizan de manera conjunta para desarrollar o implementar una solución informática · 19, 20, 38, 46, 47

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 · 8, 60, 61, 65

servidor

Computadora que provee servicios a otros dispositivos en una red, como Internet.

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. · 16, 17

Resumen

El enfoque DevOps es crucial en el ámbito de IoT dada su capacidad para permitir ciclos de desarrollo rápidos, gestionar de manera eficiente la compleja infraestructura de IoT, garantizar la calidad del software, facilitar la escalabilidad y alta disponibilidad, así como promover la seguridad en las soluciones IoT.

Por otro lado, La monitorización de sistemas IoT emerge como un componente esencial para garantizar el funcionamiento eficiente y resiliente de las soluciones IoT. Al proporcionar una visión en tiempo real del rendimiento, estado y consumo de recursos de los dispositivos conectados, la monitorización facilita la detección temprana de fallos, la mejora continua y la toma de decisiones informadas respecto al entorno.

El presente Proyecto de Fin de Máster (PFM) se centra en la adopción de un enfoque DevOps en el contexto de IoT con el propósito de proporcionar capacidades de monitorización a este entorno y facilitar a los diversos equipos implicados el desarrollo, mantenimiento y evolución integral del sistema y los dispositivos.

Otro de los pilares de este PFM es la aplicación de un enfoque de Infraestructura como Código (IaC), tratando la configuración de las máquinas virtuales como un componente más del desarrollo, permitiendo trabajar en dicho código de forma distribuida, facilitando la reproducción del entorno, su evolución y mantenimiento. Este enfoque dota de mayor agilidad al equipo de desarrollo al poder replicar el entorno en sus propios equipos rápidamente garantizando que todos los miembros del equipo cuentan con entornos idénticos, minimizando los fallos.

Como parte del enfoque DevOps, se busca la mejora continua del software; para facilitar a los equipos de desarrollo esta labor, minimizando las tareas de carácter repetitivo, se han implementado capacidades de Entrega Continua (Continuos Deployment, CD). Para ello se hacen uso de herramientas de control de versiones, CI/CD y de contenedores.

El uso de contenedores para la simulación de dispositivos IoT en este PFM atiende a razones adicionales; dotan al entorno de portabilidad al no verse ligados al sistema operativo host, algo relevante en entornos heterogéneos como puede ser un entorno IoT y permiten a los equipos involucrados explorar estrategias de escalabilidad y tolerancia a fallos, así como investigar como interactúan y se comunican estos dispositivos en un entorno escalable y eficiente.

Dado el carácter escalable del entorno a simular, complementando la monitorización, este PFM plantea un método de ingesta de logs escalable que además sea tolerante a fallos mediante el uso de agentes y sistemas de colas con protocolos ligeros orientados a IoT que permitan desacoplar los componentes, facilitando la comunicación asíncrona entre ellos.

Por último, se aborda la simulación de dispositivos y la monitorización de los datos que estos generan mediante un dashboard con gráficas y la capacidad de filtrar en base a todos sus parámetros.

Abstract

The DevOps approach is crucial in the IoT domain given its ability to enable rapid development cycles, efficiently manage the complex IoT infrastructure, ensure software quality, facilitate scalability and high availability, and promote security in IoT solutions. On the other hand, IoT system monitoring emerges as an essential component to ensure efficient and resilient operation of IoT solutions. By providing real-time insights into the performance, status, and resource consumption of connected devices, monitoring facilitates early fault detection, continuous improvement, and informed decision-making regarding the environment.

This Master's Thesis project focuses on adopting a DevOps approach in the context of IoT with the aim of providing monitoring capabilities to this environment and facilitating the development, maintenance, and comprehensive evolution of the system and devices. The project also emphasizes the application of Infrastructure as Code (IaC), treating the configuration of virtual machines as an integral development component. This enables distributed work on the code, facilitating environment reproduction, evolution, and maintenance. This approach provides greater agility to the development team by quickly replicating the environment on their own machines, ensuring that all team members have identical environments and minimizing failures.

As part of the DevOps approach, continuous improvement of software is sought. To facilitate this for development teams and minimize repetitive tasks, Continuous Deployment (CD) capabilities have been implemented. This involves the use of version control tools, CI/CD, and containers.

The use of containers for simulating IoT devices in this project serves additional purposes. They provide portability to the environment by not being tied to the host operating system, which is relevant in heterogeneous environments such as IoT. Furthermore, they allow the involved teams to explore scalability and fault tolerance strategies, as well as investigate how these devices interact and communicate in a scalable and efficient environment.

Given the scalable nature of the simulated environment and complementing the monitoring aspect, this thesis proposes a scalable log ingestion method that is also fault tolerant. This is achieved by using agents and lightweight message queue systems with IoT-oriented protocols, enabling decoupling of components, and facilitating asynchronous communication among them.

Finally, the project addresses the simulation of devices and the monitoring of the data they generate through a dashboard with graphs and the ability to filter based on all their parameters.

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 enfoques y prácticas que permitan abordar estos desafíos [1]. En este sentido, los principios DevOps han surgido como una aproximación eficaz para mejorar las capacidades de desarrollo, integración y despliegue continuo de aplicaciones, siendo uno de los conjuntos de habilidades más demandados por las empresas [2].

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 [3]. 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 entornos o plataformas que soporten los principios DevOps para garantizar la implementación, la operación y monitorización efectiva de los sistemas IoT.

La implementación exitosa de la cultura DevOps requiere el cumplimiento de varios requisitos fundamentales. En primer lugar, la formación de los equipos debe ser multidisciplinar, donde los profesionales de diferentes áreas trabajen juntos de manera colaborativa, estableciendo mecanismos de comunicación efectivos entre los desarrolladores de software, las operaciones de TI, los expertos en calidad y pruebas, seguridad, y los equipos de soporte y operativa de negocio (áreas usuarias de los sistemas) [4]. Esta colaboración es esencial para garantizar que los requisitos y las necesidades del sistema IoT se comprendan y se traduzcan adecuadamente en los procesos de desarrollo, aseguramiento de la calidad, operación de sistemas y en los procesos de negocio.

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 [5].

Por otro lado, la monitorización en una aproximación DevOps presenta varios puntos fuertes. En primer lugar, mejora la calidad del software al identificar y corregir errores de manera temprana, en combinación con la automatización de pruebas y la integración continua. Además, esta cultura DevOps busca la capacidad de respuesta ante cambios y la rápida adaptación a nuevas necesidades del negocio. Al tener una plataforma monitorizada, los equipos obtienen mayor visibilidad y control sobre el ciclo de vida de la aplicación, lo que les permite tomar decisiones informadas y reducir el time to market.

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 donde se ejecutarán las aplicaciones [6].

Otra de las necesidades que surgen en el ámbito del desarrollo software actual es la de reducir la aparición de errores causados por diferencias entre entornos, tanto en la configuración del entorno de desarrollo de cada miembro del equipo, como la diferencia entre el entorno de desarrollo y los subsiguientes, por ejemplo, entorno de QA, entorno de preproducción y entorno de producción.

Para ello, Infrastructure as Code (IaC) ha demostrado ser un concepto útil para este propósito. Una de las tecnologías que pueden cubrir esta necesidad en el alcance de este proyecto, principalmente el de entornos de desarrollo virtualizados, portátiles y consistentes, es Vagrant.

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 [7].

Además, Vagrant se integra perfectamente con otras herramientas y tecnologías utilizadas en enfoques DevOps, como Ansible o Docker. Esto permite la automatización de tareas de provisión 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 último, Jenkins desempeña un papel crucial en un entorno DevOps como el que plantea este proyecto al proporcionar una plataforma sólida y flexible para la integración y entrega continuas de forma transversal a todos los entornos. Su capacidad para automatizar los procesos de pruebas y despliegue, junto con su amplia gama de plugins [8] y su integración con otras herramientas populares, lo convierten en una pieza fundamental en el ciclo de vida del desarrollo de software.

Jenkins facilita la colaboración entre los equipos de desarrollo y operaciones al permitir una implementación rápida y confiable de cambios en los sistemas, al tiempo que garantiza la estabilidad y calidad del software mediante pruebas automáticas.

Al proporcionar una visión clara del estado de los proyectos en sus dashboard de visualización de pipelines y una retroalimentación rápida sobre posibles problemas con las pruebas automáticas, Jenkins mejora la eficiencia de los equipos, reduce el tiempo de entrega y fomenta una mentalidad de mejora continua. En definitiva, Jenkins se ha convertido en una herramienta imprescindible para lograr los objetivos de un entorno DevOps al impulsar la automatización, la colaboración y la entrega continua de software de alta calidad [9].

En conclusión, la cultura DevOps ha surgido como una respuesta efectiva a los desafíos actuales en el desarrollo de software. Al promover la colaboración, la automatización y la entrega continua, las prácticas DevOps mejoran las capacidades de desarrollo y operación y permiten una mayor eficiencia en la entrega de software.

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 principios. Es fundamental fomentar una mentalidad de mejora continua y promover la colaboración entre los diferentes roles involucrados [10].

1.2. Objetivos

- **OBJ1:** Aplicar los principios DevOps en el ámbito del Internet de las cosas (IoT) a lo largo de todo el ciclo de vida del proyecto.
- **OBJ2:** Establecer una sólida monitorización de sistemas IoT como piedra angular del proyecto, permitiendo obtener una visión en tiempo real de los datos generados por los dispositivos.
- **OBJ3:** Implementar un proceso de entrega continua utilizando herramientas de Integración Continua/Despliegue Continuo (CI/CD) como Jenkins, para asegurar la eficiencia y agilidad en el despliegue de cambios y actualizaciones en los sistemas IoT.
- **OBJ4:** Establecer una herramienta de control de versiones como GitHub, garantizando la trazabilidad y colaboración efectiva entre los equipos de desarrollo y operaciones en el desarrollo de los diferentes componentes.
- **OBJ5:** Aplicar el enfoque de Infraestructura como Código (IaC) en el entorno de desarrollo mediante el uso de Vagrant, facilitando la reproducibilidad, evolución y mantenimiento del entorno de desarrollo de manera consistente y portátil.
- **OBJ6:** Implementar una solución de monitorización utilizando el stack ELK (Elasticsearch, Logstash y Kibana), permitiendo recopilar, analizar y visualizar los registros de los sistemas IoT, facilitando la detección temprana de problemas y toma de decisiones informadas.
- **OBJ7:** Diseñar una arquitectura de solución que sea escalable y resiliente, teniendo la posibilidad de tener alta disponibilidad en los componentes clave, de forma que se pueda escalar ante picos de carga y haya múltiples nodos para que no se sufra indisponibilidad ante errores.
- **OBJ8:** Implementar prácticas de seguridad sólidas en el desarrollo y operación de los sistemas IoT, garantizando la protección de los datos y la privacidad de los usuarios, así como la integridad y confidencialidad de la información transmitida.
- **OBJ9:** Establecer procesos y herramientas de automatización para la gestión de infraestructuras y despliegue de sistemas IoT, permitiendo la creación, configuración y actualización eficiente de los recursos necesarios, reduciendo el tiempo y los errores asociados a las tareas manuales.

- **OBJ10:** Promover la documentación exhaustiva y actualizada de los procesos, configuraciones y decisiones tomadas durante el desarrollo y operación de los sistemas IoT, facilitando la comprensión y el mantenimiento de estos a lo largo del tiempo, así como el intercambio de conocimientos entre los miembros del equipo y futuros usuarios o administradores del sistema.

1.3. Metodología y Planificación

Para el desarrollo del PFM, 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.

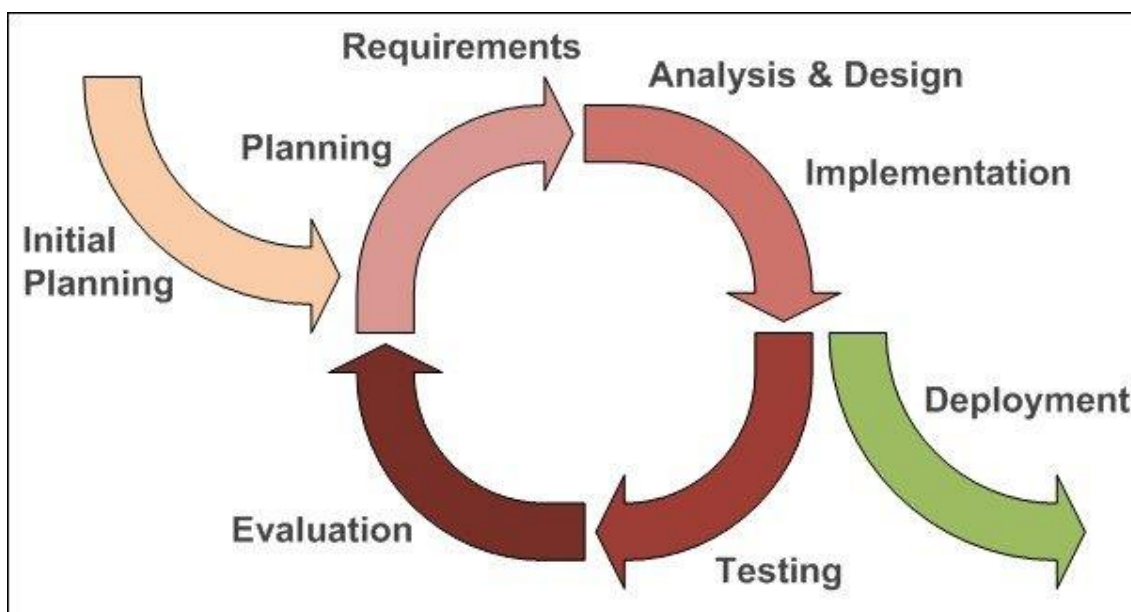


Ilustración 1 - Ilustración descriptiva de la metodología iterativa incremental [Fuente]

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.

1.3.1. 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 máquinas 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

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 `kubecttl get nodes` y deberemos visualizar el número 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

La iteración comienza con el diseño e implementación del dashboard de Kibana. En esta etapa, se define la estructura y el diseño del dashboard para visualizar los datos recopilados y analizados. Se consideran los requisitos y necesidades del equipo de desarrollo y operaciones para crear un dashboard intuitivo y eficiente.

A continuación, se procede al despliegue automático de Jenkins utilizando Vagrant. Se configuran los archivos de configuración necesarios y se automatiza el proceso de instalación y configuración de Jenkins en un entorno virtualizado. Esto permite tener un entorno de desarrollo consistente y reproducible, listo para su uso inmediato.

Una vez que Jenkins está en funcionamiento, se procede al desarrollo del pipeline en Jenkins. Se define un flujo de trabajo que incluye las etapas necesarias para desplegar los dispositivos automáticamente. Esto implica configurar la integración continua y las pruebas necesarias para garantizar que el código de los dispositivos se despliegue de manera confiable y eficiente en respuesta a los cambios realizados en el repositorio del proyecto.

Además de lo mencionado anteriormente, también se incluirá la instalación del dashboard de Kubernetes sobre el clúster K3S en la iteración de la metodología iterativa incremental.

Se procederá a la instalación y configuración del dashboard de Kubernetes, que permite visualizar y administrar los recursos del clúster de forma gráfica. Se utilizan los archivos de configuración y los comandos necesarios para desplegar el dashboard en el clúster K3S.

Con el dashboard de Kubernetes en funcionamiento, se logra una mayor visibilidad y control sobre el clúster K3S y los dispositivos IoT desplegados en él. Esto permite monitorear el estado del clúster, realizar acciones administrativas y facilitar la gestión de los dispositivos en el contexto del desarrollo de soluciones IoT.

Durante la iteración, se realizan pruebas y se ajustan los elementos desarrollados. Se recopilan comentarios y se realizan mejoras iterativas del proyecto, el despliegue automático de Jenkins y el pipeline de Jenkins. Al final de la iteración, se tiene un dashboard de Kibana funcional, un entorno Jenkins desplegado automáticamente, un dashboard de Kubernetes y un pipeline configurado que permite el despliegue automatizado de los dispositivos.

Iteración 7

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.

1.4. Estructura del documento

- El **capítulo 2** presenta los fundamentos teóricos de este PFM.
- El **capítulo 3** describe la contribución de este trabajo, la arquitectura de la solución, decisiones de diseño, configuraciones de la “plataforma DevOps” desarrollada, y pruebas y validación.
- El **capítulo 4** presenta los resultados y discusión.
- El **capítulo 5** presenta la conclusiones y líneas futuras.

2. Fundamentos teóricos:

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 [11].

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 [12].

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 [13].

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.

La adhesión de Jenkins al proyecto responde a la necesidad de implementación de estrategias de entrega continua, se utiliza dicha tecnología por su carácter abierto respecto a otras de la competencia como pueden ser; Azure DevOps, Github Actions, etc.

A continuación, veremos en detalle cada uno de estos conceptos y tecnologías.

2.1. 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 [11].

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 [12].

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 [13].

2.1.1. *Cloud Computing en el ámbito de IoT*

El Cloud computing en IoT es un modelo en el que los dispositivos IoT utilizan recursos y servicios en la nube para almacenar, procesar y analizar datos. En lugar de realizar todo el procesamiento localmente, los dispositivos IoT se conectan a la nube para acceder a una infraestructura escalable y flexible [14].

En este modelo, los datos generados por los dispositivos IoT se transmiten a la nube, donde se almacenan y procesan en servidores remotos. La nube ofrece una gran capacidad de almacenamiento y potencia de cómputo, permitiendo analizar grandes volúmenes de datos en tiempo real. Además, proporciona servicios adicionales como aprendizaje automático, análisis avanzado y herramientas de visualización.

El Cloud computing para IoT ofrece numerosas ventajas, como la capacidad de escalar rápidamente los recursos según las necesidades, reducir la carga de procesamiento en los dispositivos IoT, facilitar la implementación y actualización de aplicaciones, y permitir una colaboración y acceso remoto más eficientes. Sin embargo, también implica consideraciones de seguridad y privacidad de los datos, así como dependencia de la

conectividad a Internet. En general, el Cloud computing es un componente clave en la arquitectura de IoT, ya que permite aprovechar los beneficios de la nube para mejorar la capacidad de procesamiento y análisis de los dispositivos IoT.

2.1.2. Edge Computing

Edge computing es un modelo de computación distribuida en el cual el procesamiento de datos se realiza cerca de la fuente de generación, en los dispositivos o en puntos cercanos a ellos en la red. A diferencia del procesamiento tradicional en la nube, donde los datos se envían a servidores remotos para su procesamiento, el edge computing permite un procesamiento más rápido y una respuesta en tiempo real al minimizar la latencia y el ancho de banda utilizado [15].

En el edge computing, los dispositivos en el borde de la red, como sensores, cámaras y dispositivos IoT, pueden realizar tareas de procesamiento local, almacenamiento y análisis de datos. Esto permite una toma de decisiones más rápida y autónoma en tiempo real, lo que resulta especialmente beneficioso en aplicaciones de tiempo real, como la industria manufacturera, la salud y los vehículos autónomos.

Además, el edge computing también ayuda a reducir la dependencia de la conectividad constante a la nube y mejora la seguridad y privacidad de los datos al procesarlos localmente. Sin embargo, también implica desafíos en términos de administración, coordinación y seguridad de los dispositivos y sistemas distribuidos. En resumen, el edge computing es un enfoque que permite llevar el poder de cómputo y análisis de datos más cerca de la fuente de generación, lo que conduce a una mayor eficiencia, rendimiento y capacidad de respuesta en entornos distribuidos.

CLOUD COMPUTING VS. EDGE COMPUTING

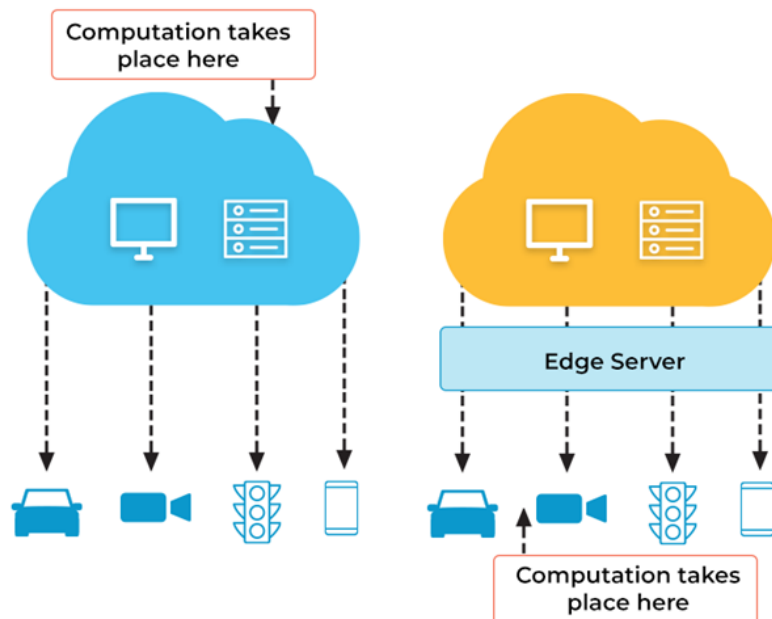


Ilustración 2 - Cloud Computing vs Edge computer de forma visual. [Fuente]

2.2. 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 [16].

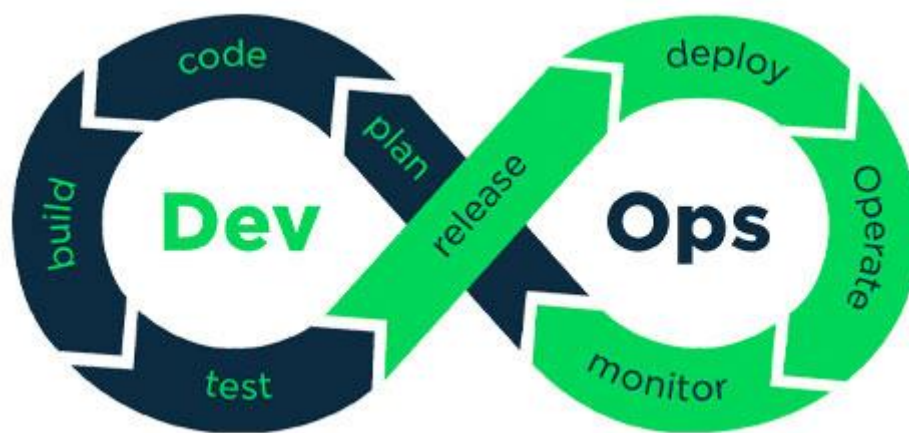


Ilustración 3 - Imagen que describe el ciclo de vida en un entorno DevOps donde se describen las etapas en las que intervienen ambos equipos [\[Fuente\]](#)

Los principios clave de DevOps incluyen [17] [18] [10]:

- **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 (IaC):** La Infraestructura como código (IaC) es una práctica en la que la configuración y administración de la infraestructura de TI se realiza a través de código, en lugar de hacerlo manualmente. En lugar de configurar y administrar servidores, redes y otros recursos de forma manual y ad hoc, se utiliza código para definir y automatizar la infraestructura.

Con la IaC, se utilizan herramientas y lenguajes de programación específicos para describir la infraestructura deseada. Esto incluye definir los recursos, sus propiedades y configuraciones, así como las relaciones entre ellos. Luego, el código se puede utilizar para crear, modificar y eliminar recursos de manera consistente y reproducible.

La IaC ofrece varios beneficios, como la capacidad de versionar y controlar los cambios de infraestructura, facilitar la colaboración entre equipos, permitir la creación de entornos de prueba y desarrollo rápidos y reproducibles, y mejorar la escalabilidad y la automatización. También ayuda a reducir errores humanos, mejora la eficiencia y promueve la estandarización en la gestión de la infraestructura.

- **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.

DevOps vs SecDevOps

SecDevOps, también conocido como DevSecOps, es un enfoque que integra la seguridad en todas las etapas del ciclo de vida de desarrollo y operación de software. Busca garantizar que la seguridad sea una prioridad desde el principio, en lugar de abordarla como una preocupación posterior. [19]

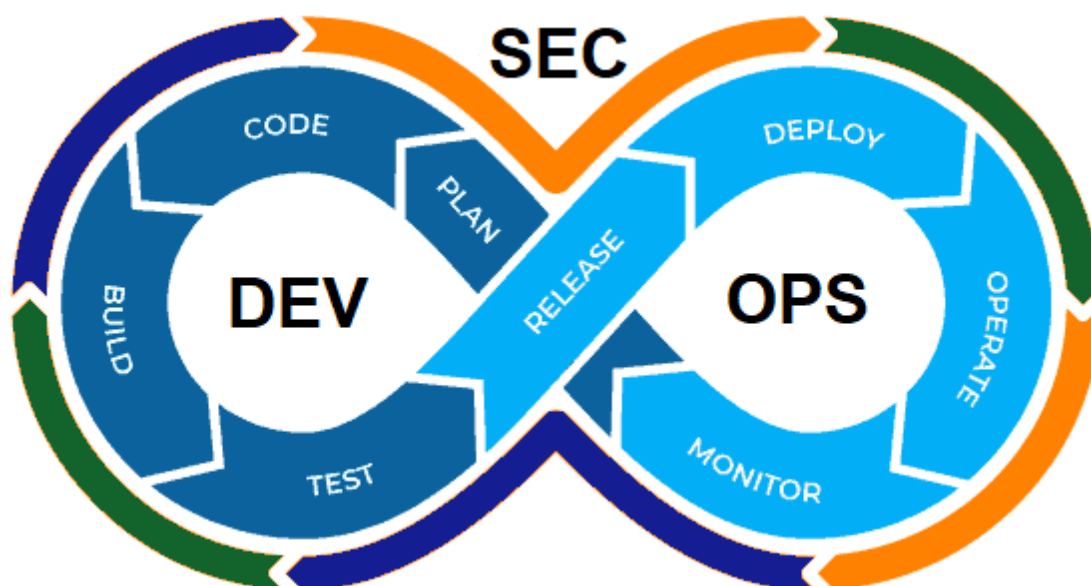


Ilustración 4 - Imagen representativa de las diferentes fases, análoga a la de DevOps, incluyendo la parte de Seguridad (SEC). [Fuente]

Este término comenzó a ganar popularidad en la industria del desarrollo software alrededor de 2012 [20], mientras que DevOps ya alcanzó una popularidad similar en 2007 [21]. SecDevOps es un término menos generalizado [22] aunque bastante interesante en algunos nichos de mercado del desarrollo software, sobre todo en aquellos cuyos sistemas requieren disponibilidad y manejan datos de carácter sensible o de alto riesgo. Veamos que los diferencia en algunos aspectos clave:

- Enfoque principal:
 - DevOps: Colaboración e integración entre equipos de desarrollo y operaciones para una entrega rápida y confiable de software.
 - SecDevOps: Integración proactiva y continua de la seguridad en todas las etapas del ciclo de vida de desarrollo y operación de software.
- Objetivo:
 - DevOps: Acelerar el ciclo de vida de desarrollo y despliegue de software, optimizando la eficiencia y la calidad.
 - SecDevOps: Garantizar que el software sea seguro y confiable, identificando y abordando las vulnerabilidades y riesgos de seguridad desde el principio.
- Enfoque de seguridad:
 - DevOps: La seguridad es un aspecto importante, pero no siempre se aborda de manera integral desde el inicio.
 - SecDevOps: La seguridad se integra en cada etapa del proceso de desarrollo y despliegue, trabajando estrechamente con los equipos de desarrollo y operaciones.
- Proceso de colaboración:
 - DevOps: Colaboración constante entre los equipos de desarrollo y operaciones para una entrega rápida y continua de software.
 - SecDevOps: Colaboración estrecha entre los equipos de desarrollo, operaciones y seguridad para garantizar que las prácticas de seguridad se implementen en todo momento.
- Resultado final:
 - DevOps: Entrega rápida y confiable de software, con una mayor eficiencia y calidad.
 - SecDevOps: Software seguro y confiable, con la integración de prácticas de seguridad desde el inicio.

En resumen, mientras que DevOps se enfoca en la colaboración y entrega rápida de software, SecDevOps amplía ese enfoque al integrar la seguridad desde el principio, asegurando que el software sea seguro y confiable en cada etapa del ciclo de vida de desarrollo y operación.

2.3. Vagrant

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. [7]

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 se ejecuta "vagrant up" en el directorio del 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.

El comando Vagrant up también se puede utilizar seguido del nombre concreto del recurso a instanciar dentro de nuestro Vagrantfile para hacer provisiones parciales de nuestro entorno con el objetivo, por ejemplo, de hacer pruebas aisladas de los sistemas que utilizaremos [23].

- **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 [24].

- **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 [25].

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.

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. [26]

2.4. 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 [27].

2.4.1. *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 [28].

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 [29]:

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.

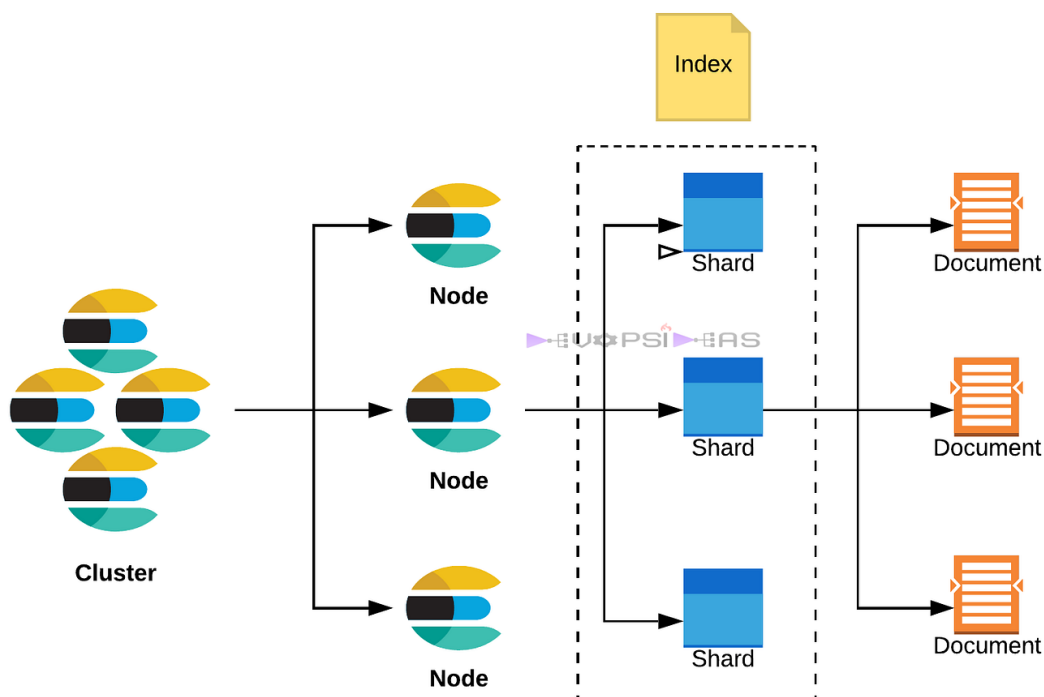


Ilustración 5 - Relación entre los componentes de Elasticsearch [\[Fuente\]](#)

2.4.2. 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 [30]:

- **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.

2.4.3. 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 [31].

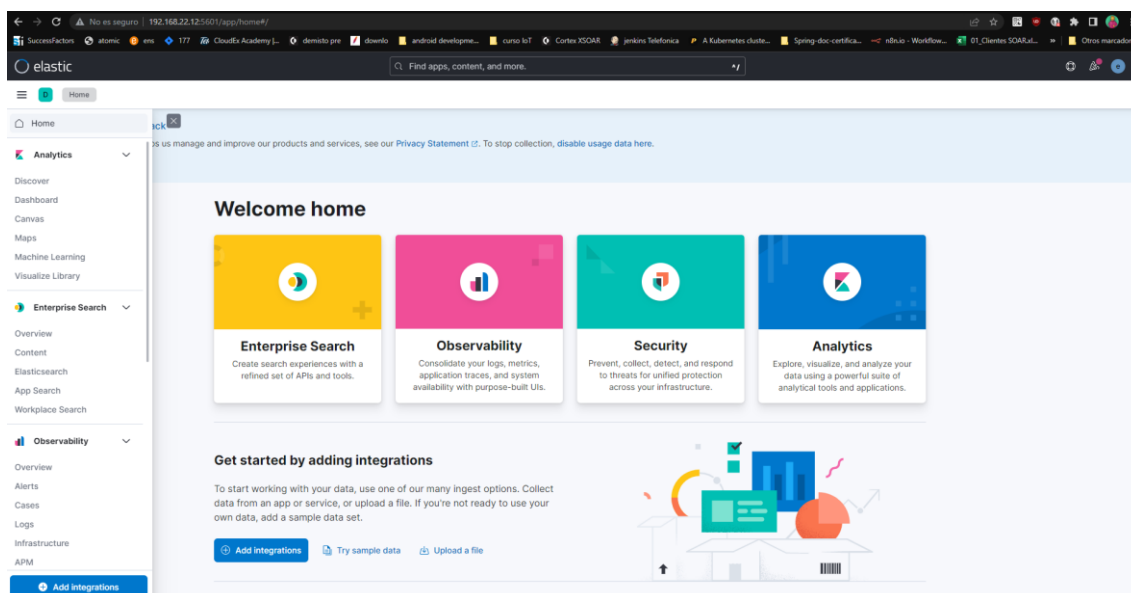


Ilustración 6 - Landing page de Kibana tras realizar el login

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.

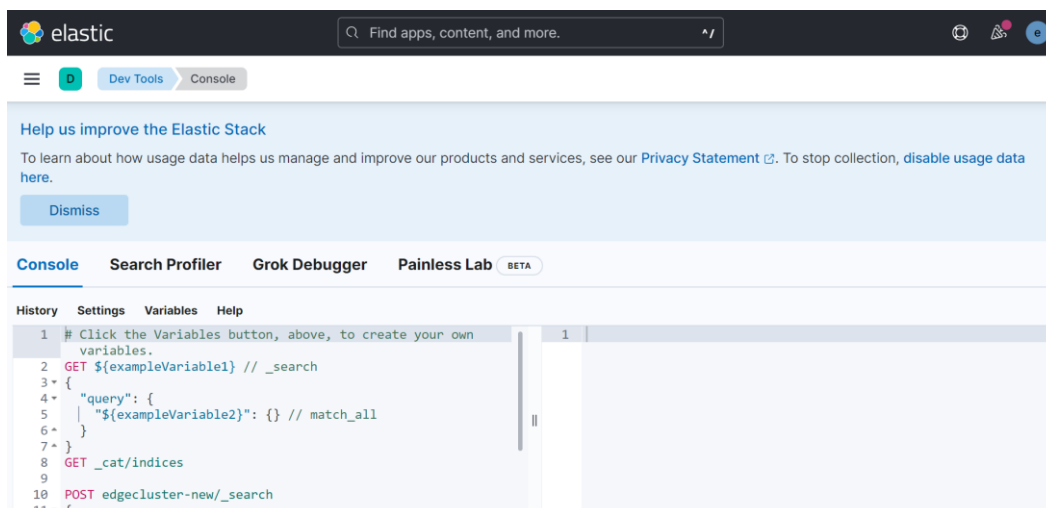


Ilustración 7 - Consola dentro de Kibana, desde donde podríamos lanzar peticiones y visualizar los resultados

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.

2.5. 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 [32]:

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 [33]:

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.

2.5.1. Principales términos relacionados con Kubernetes

Los principales términos y componentes de Kubernetes son [32]:

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.

2.5.2. Principales comandos de K3s

Los principales comandos de K3S usados en este proyecto son [34]:

- 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 [35]:

- `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 despliegue.

2.6. 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) [36].

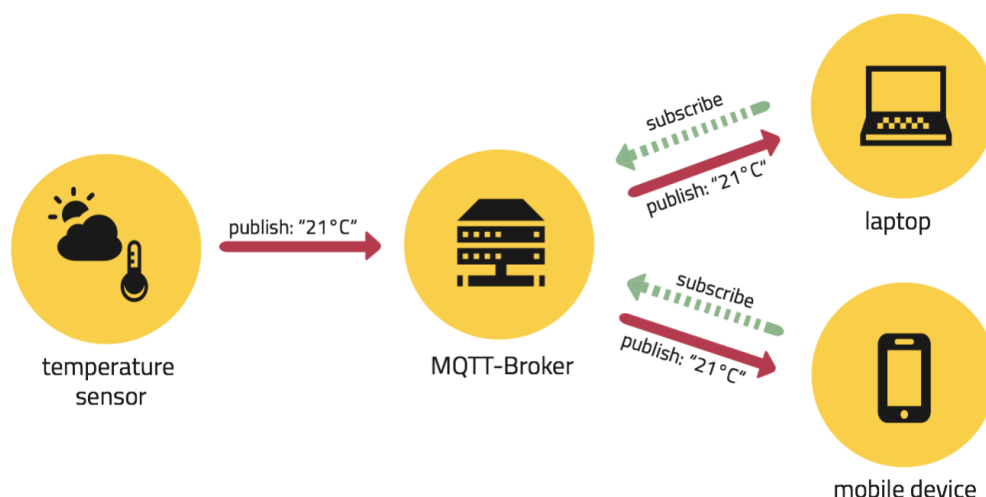


Ilustración 8 - Funcionamiento de un Broker MQTT

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 [37]:
 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 [38].

2.6.1. Last Will en MQTT

El "last will" (última voluntad) es un mecanismo en el protocolo MQTT (Message Queuing Telemetry Transport) que permite a un cliente especificar un mensaje que se enviará automáticamente a otros clientes suscritos cuando el cliente emisor se desconecte inesperadamente. Esta característica es especialmente útil en escenarios donde es importante notificar a los demás clientes sobre la desconexión de un cliente en particular [39].

El funcionamiento del "last will" en MQTT se puede describir de la siguiente manera:

- 1) El cliente que se conecta al broker MQTT tiene la opción de especificar un mensaje de "last will". Este mensaje puede contener información relevante, como el motivo de la desconexión o cualquier otra información que se considere importante transmitir a los demás clientes.

- 2) Cuando el cliente establece una conexión con el broker MQTT, incluye el mensaje de "last will" y el topic donde se publicará este mensaje, junto con otros detalles de la conexión, como el tema (topic) al que se suscribirá y los mensajes que publicará.
- 3) El broker MQTT almacena el mensaje de "last will" junto con la información de conexión del cliente.
- 4) Si el cliente se desconecta de manera inesperada, ya sea debido a un cierre de sesión abrupto o una pérdida de conectividad, el broker MQTT detecta la desconexión y busca el mensaje de "last will" asociado con ese cliente.
- 5) Una vez que se encuentra el mensaje de "last will", el broker MQTT lo publica automáticamente en el tema especificado por el cliente durante la conexión.
- 6) Todos los clientes suscritos a ese tema recibirán el mensaje de "last will" enviado por el broker MQTT. Esto les permite tomar medidas en consecuencia o realizar cualquier acción necesaria en respuesta a la desconexión del cliente.

MQTT-Packet:	
CONNECT	
contains:	Example
clientId	"client-1"
cleanSession	true
username (optional)	"hans"
password (optional)	"letmein"
lastWillTopic (optional)	"/hans/will"
lastWillQos (optional)	2
lastWillMessage (optional)	"unexpected exit"
lastWillRetain (optional)	false
keepAlive	60

Ilustración 9 - Formato de un mensaje MQTT con lastWill [\[Fuente\]](#)

Es importante tener en cuenta que el mensaje de "last will" se envía solo en el caso de una desconexión no planificada del cliente. Si el cliente se desconecta de manera adecuada y envía un mensaje de desconexión explícito, el mecanismo de "last will" no se activará.

2.7. Jenkins

Jenkins es una herramienta de código abierto utilizada para la automatización de la integración y entrega continuas (CI/CD) en el desarrollo de software. Proporciona un entorno de automatización flexible y extensible que permite a los equipos de desarrollo y operaciones acelerar y mejorar el ciclo de vida de desarrollo de aplicaciones [40].

A nivel teórico, es importante comprender los siguientes aspectos clave sobre Jenkins:

1. Integración continua (CI): Jenkins facilita la integración continua al permitir la construcción, prueba y análisis automatizados del código fuente de una aplicación cada vez que se realizan cambios en el repositorio. Esto garantiza que el código nuevo y existente se integre sin problemas y se detecten rápidamente los problemas de compatibilidad y errores.
2. Entrega continua (CD): Jenkins también permite la entrega continua automatizada, lo que implica llevar el código de una aplicación a través de diversas etapas de pruebas y despliegue hasta su lanzamiento final en producción. Esto se logra mediante la configuración de pipelines (flujos de trabajo) que definen las acciones y etapas necesarias para compilar, probar, empaquetar y desplegar la aplicación de manera controlada.
3. Automatización de tareas: Jenkins ofrece una amplia gama de plugins y funcionalidades que permiten la automatización de diversas tareas en el proceso de desarrollo de software. Esto incluye la compilación del código fuente, la ejecución de pruebas automatizadas, la generación de informes, la notificación de resultados, la implementación en diferentes entornos, la gestión de dependencias y muchas otras actividades relacionadas con el desarrollo y despliegue de aplicaciones.
4. Extensibilidad y flexibilidad: Jenkins es altamente extensible y personalizable, lo que significa que se pueden agregar plugins y configuraciones específicas para adaptarse a los requisitos y necesidades del proyecto. Esto permite integrar Jenkins con diferentes herramientas y servicios, como sistemas de control de versiones, sistemas de gestión de incidencias, servicios de notificación, herramientas de pruebas automatizadas, servicios de nube, entre otros.
5. Interfaz gráfica y visualización: Jenkins ofrece una interfaz gráfica de usuario (GUI) intuitiva que facilita la configuración y gestión de los flujos de trabajo de CI/CD. Además, proporciona informes y visualizaciones claras sobre el estado y los resultados de las ejecuciones de los pipelines, lo que permite un monitoreo y seguimiento eficiente del proceso de desarrollo.
6. Comunidad activa y soporte: Jenkins cuenta con una gran comunidad de desarrolladores y usuarios que contribuyen con la mejora continua de la herramienta y brindan soporte en forma de documentación, tutoriales y foros de discusión. Esto garantiza un entorno de apoyo sólido y recursos disponibles para resolver problemas y compartir buenas prácticas.

A nivel práctico, Jenkins se instala en un servidor y se configura para trabajar con los proyectos de software que se desean automatizar. Aquí hay una descripción paso a paso de cómo utilizar Jenkins.

Una vez instalado y configurado (instalados los plugins necesarios y las dependencias a nivel de sistema operativo para la funcionalidad deseada), el funcionamiento de Jenkins se basa en la ejecución de Jobs, también conocidos como pipelines.

Estos Jobs se ejecutan automáticamente o mediante interacción manual y su objetivo puede ser muy variado, aunque sus principales funciones suelen ser la compilación de

un proyecto, la ejecución de pruebas, la generación de informes o el despliegue de una aplicación, entre otras.

Los triggers automáticos de estos Jobs determinan cuándo se debe ejecutar un trabajo. Pueden ser desencadenados por eventos como cambios en el repositorio de código fuente, programados para ejecutarse en un momento específico o disparados manualmente.

Por último, Jenkins ofrece una amplia gama de características que facilitan la visualización y el seguimiento de la ejecución de los jobs. Una de las características clave es la vista basada en stages, que proporciona una representación gráfica del flujo de trabajo del pipeline. Esta vista muestra los diferentes pasos del pipeline, también conocidos como stages, y permite ver el progreso de cada uno de ellos. Esto es especialmente útil cuando se trabaja con pipelines complejos que implican múltiples etapas y tareas.

Además de la vista basada en stages, Jenkins permite acceder a los logs de la ejecución del pipeline. Los logs proporcionan información detallada sobre cada paso del pipeline, incluyendo cualquier mensaje de salida, advertencia o error que se haya producido. Estos logs son extremadamente útiles para el seguimiento y la depuración, ya que permiten identificar rápidamente cualquier problema o anomalía en el proceso de automatización.

Jenkins también ofrece la posibilidad de generar informes y métricas sobre la ejecución de los jobs. Estos informes pueden incluir datos como la duración de la ejecución, el número de compilaciones exitosas o fallidas, las pruebas unitarias ejecutadas y sus resultados, entre otros.

3. Descripción de la Plataforma DevOps

3.1. Arquitectura de la solución

La arquitectura de sistemas (ver Ilustración 10) está compuesta por diferentes sistemas que, en conjunto, satisfacen las necesidades establecidas en los objetivos del PFM.

Los componentes son:

- Plataforma de Integración y Entrega Continua (CI&CD) que automatiza la construcción (build), testing, y despliegue de módulos software de sistemas IoT-Edge y que es independiente al entorno al utilizarse de forma cruzada a todos los que se utilicen.
- Entorno de desarrollo donde se desplegarán los módulos software de la capa Edge del sistema IoT-Edge y los dispositivos simulados. Se compone de:
 - **Plataforma de orquestación de contenedores:** Añade portabilidad, escalabilidad y tolerancia a fallos. En este caso la tecnología elegida es K3S. Este componente no existe en el entorno de producción al utilizarse en entornos previos para simular dispositivos y elementos de red.
 - **Sistema de mensajería basado en colas con protocolo MQTT:** Desacopla la generación de logs de la ingesta y su procesamiento. La tecnología para este sistema es Mosquitto.
 - **Plataforma de Monitorización Continua:** Plataforma que permite la monitorización de los dispositivos en tiempo real, permitiendo visualizar los datos y detectar anomalías que posibiliten la mejora continua. Se compone de los siguientes componentes:
 - **Agente ligero de recolección de eventos:** añade la funcionalidad necesaria para consumir los mensajes de las colas y que se estos lleguen a la plataforma de procesamiento e ingesta de los eventos. Aprovechamos en este caso otro componente del stack ELK; Beats.
 - **Plataforma de procesamiento e ingesta de eventos:** Posibilita la ingesta en la plataforma de indexación de los logs recolectados por los agentes mediante pipelines que, adicionalmente, se encargan de enriquecer y adaptar los logs a un formato que facilite su tratamiento. En este caso se utiliza el componente del stack ELK: Logstash.
 - **Plataforma de indexación de logs:** Aporta persistencia a los logs y un motor de búsqueda eficaz, adicionalmente, cuenta con la capacidad de desplegarse en un cluster, lo que aporta escalabilidad y tolerancia a fallos. Se utiliza el componente del stack ELK: Elasticsearch.
 - **Plataforma de visualización de datos:** Complementa las capacidades de monitorización del resto de la arquitectura posibilitando la visualización por parte de los usuarios, tanto de forma libre, mediante búsquedas, como mediante dashboards. En

este caso la tecnología utilizada es el componente del stack ELK: Kibana.

La instalación y configuración de las plataformas de CI&CD y monitorización junto con la creación y configuración del entorno de desarrollo y que en este PFM se ha denominado **Plataforma DevOps** está automatizada mediante la tecnología Vagrant.

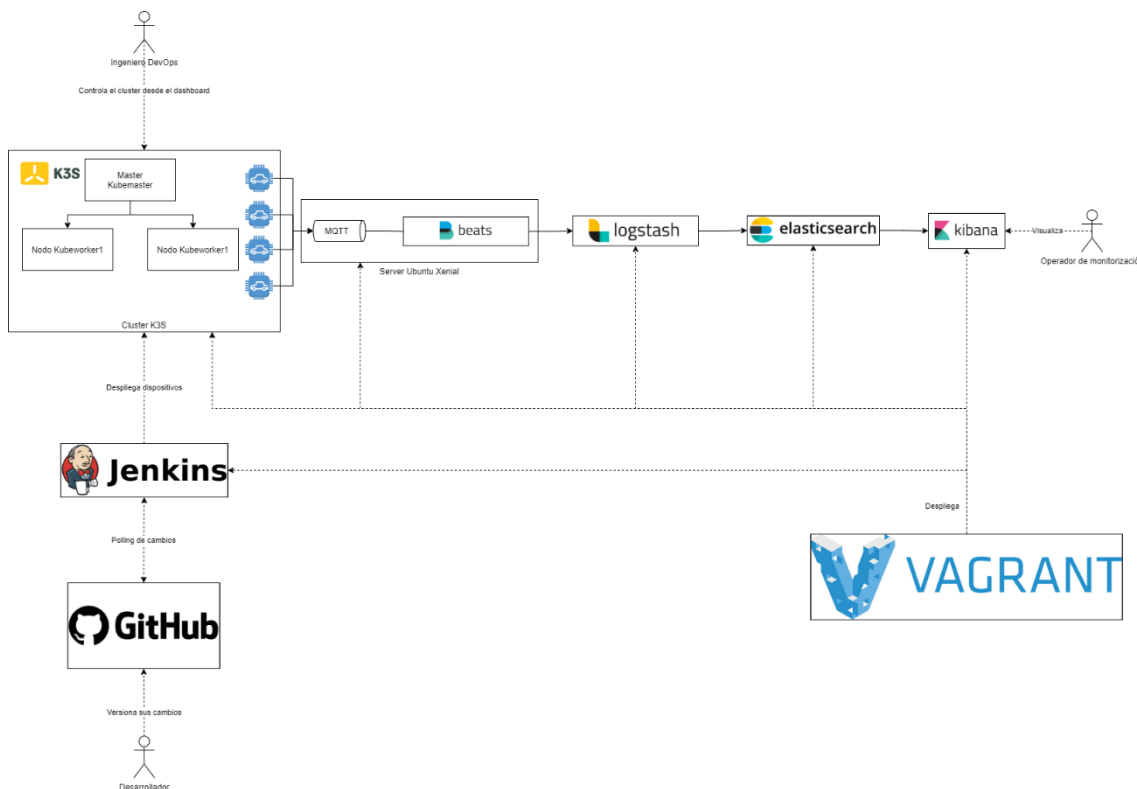


Ilustración 10 - Definición de la arquitectura de la Plataforma DevOps

El despliegue de toda esta Plataforma DevOps se realiza en siete máquinas virtuales:

- **Cluster K3S:** Consiste en un master y 2 nodos (3 máquinas virtuales) donde se despliegan los pods que simulan dispositivos Edge de un sistema “tipo” IoT-Edge. El cluster K3S utiliza la tecnología de Kubernetes para gestionar la orquestación y escalado de los pods de manera eficiente. Dentro del cluster también se hace la instalación del dashboard de kubernetes para facilitar la gestión del cluster por parte de los ingenieros DevOps del proyecto.
- **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. Aquí se contará dentro de la arquitectura de un dashboard de visualización de datos relevantes de los dispositivos (estos datos se detallan en el apartado 4):

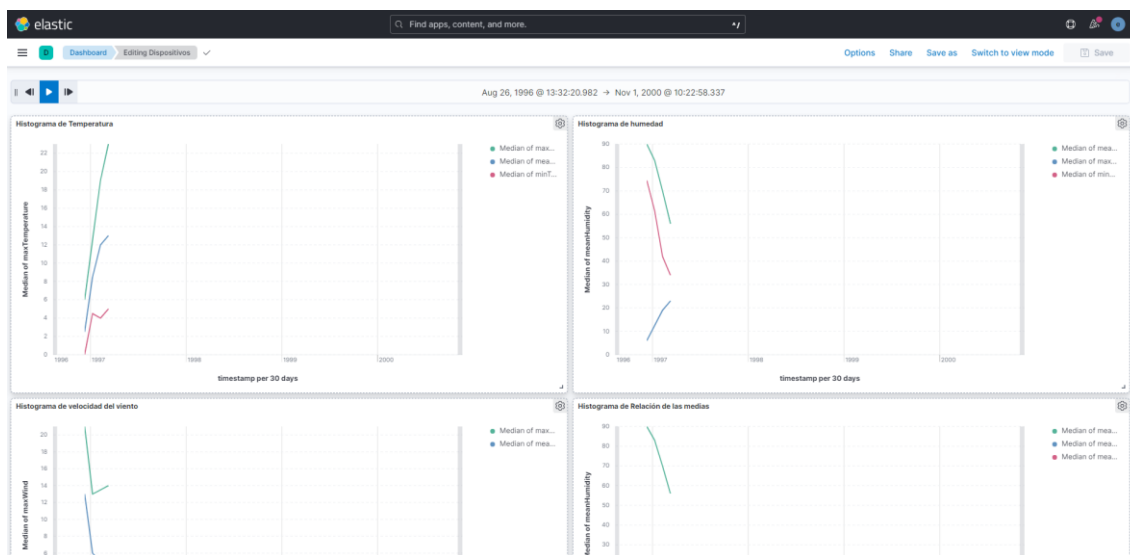


Ilustración 11 - Dashboard de kibana con datos cargados por los dispositivos. Las tablas representan la temperatura, viento y humedad de Madrid por fecha, así como la relación entre sus medias.

- **Máquina virtual con Jenkins:** En esta máquina se despliega Jenkins, su objetivo es el de desplegar los dispositivos automáticamente cuando se realice algún cambio en el repositorio de Github del proyecto por parte de los desarrolladores. Para el correcto funcionamiento de esta pieza dentro de la arquitectura debemos realizar algunos pasos de configuración de carácter manual que se detallan en el punto 3.3.1.

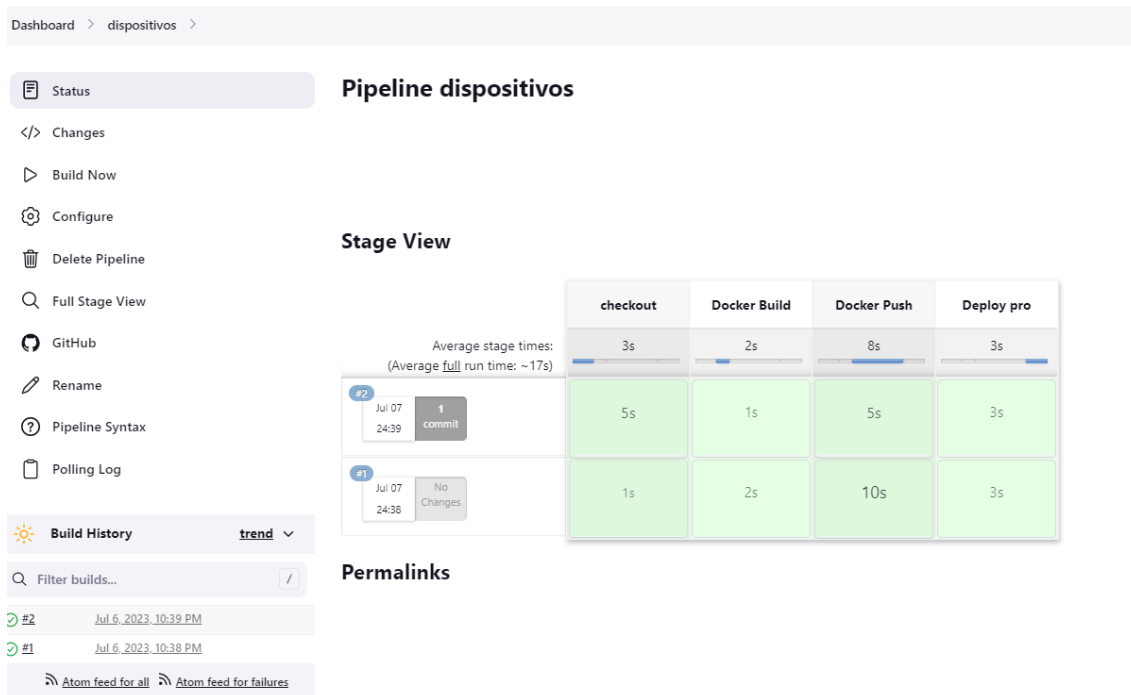


Ilustración 12 - Vista de ejecución del job configurado para el proyecto en Jenkins, podemos observar una ejecución manual y otra automática, diferenciadas por la detección de commits a la derecha de la fecha de ejecución.

Cada componente desempeña un papel específico en la arquitectura, trabajando en conjunto para desplegar la infraestructura necesaria; ejecutar una simulación de los dispositivos distribuida en un cluster; redespargar el software de los dispositivos ante cambios; recibir, procesar, almacenar y visualizar los logs generados por los dispositivos edge 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.

3.2. Decisiones de Diseño

3.2.1. 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

3.2.2. Utilización de Python para los dispositivos

Se decide utilizar python atendiendo a algunas ventajas que presenta en entornos ágiles 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 decidido su utilización en este proyecto.

3.2.3. Desarrollo de un solo Vagrantfile

La creación de un solo Vagrantfile en lugar de varios es una decisión que se ha tomado atendiendo 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.

3.2.4. 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.

3.2.5. *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.

3.2.6. *Trigger del pipeline de Jenkins mediante SCM polling*

Como parte de la configuración del pipeline de Jenkins se establece que el trigger para dicho job va a ser de tipo SCM polling, esto implica que, acorde al cron que configuremos, Jenkins realizará peticiones al proyecto git que hemos configurado para verificar si hay algún cambio desde la última ejecución del pipeline, ejecutándolo en caso afirmativo.

Esta decisión se toma debido a la dificultad que supondría hacerlo mediante webhooks al no tener asignada una IP pública para la máquina virtual.

El funcionamiento de trigger mediante webhook suele simplificar mucho la carga en el servidor de Jenkins al ser un agente externo quien activa ante cambios la ejecución del pipeline. Se suele configurar de forma que se active cuando se realice una reléase en github y se suele diferenciar entre los diferentes entornos en base a la tag asociada a la reléase, siendo, por ejemplo, “v.1.0.0” un paso a entorno de preproducción y la tag “v1.0.0-RELEASE” un pase a producción.

3.3. 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 línea 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.

3.3.1. Configuración de Jenkins

Crearemos el pipeline siguiendo los siguientes pasos:

1. Tras logearnos en la web de Jenkins, accederemos a la sección de gestión de jenkins (manage Jenkins), a la sección de plugins, e instalaremos el plugin “SSH Agent”.

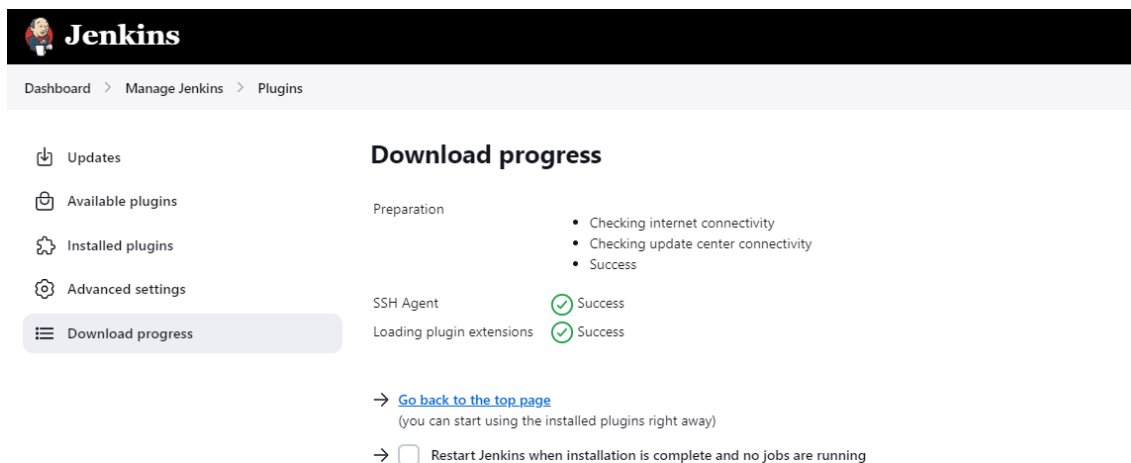


Ilustración 13 - Paso 1 de la configuración de Jenkins

2. Tras ello, volvemos a la pantalla principal y clickamos en nuevo item:

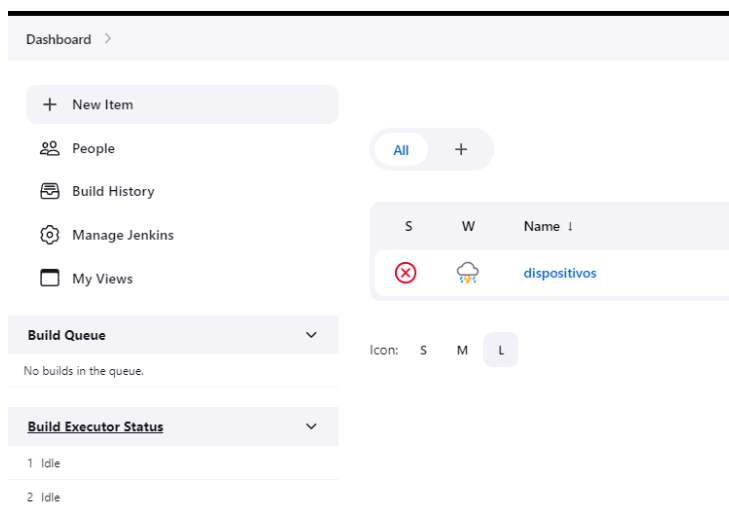


Ilustración 14 - Paso 2 de la configuración de Jenkins

3. Le damos un nombre y seleccionamos la opción Pipeline

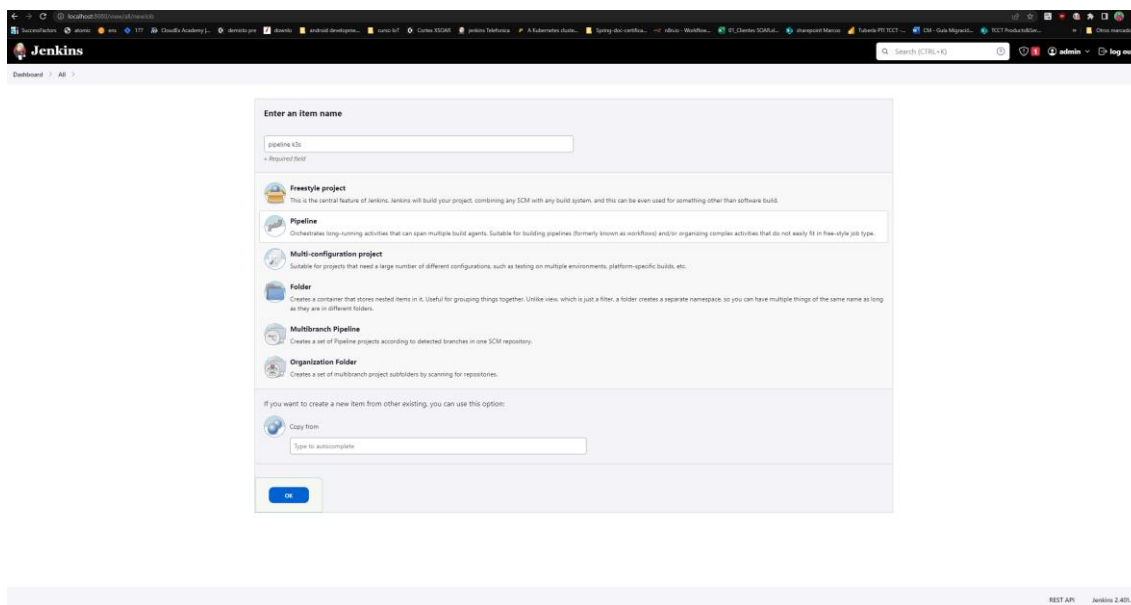


Ilustración 15 - Paso 3 de la configuración de Jenkins

4. Configuramos el GitHub Project y el Build Trigger Poll SCM con un cron de ejecución de cada 5 minutos (aunque se puede poner menos tiempo o configurar una ventana diaria para mantener los dispositivos actualizados, en este caso ponemos 5 minutos para realizar pruebas con sencillez)

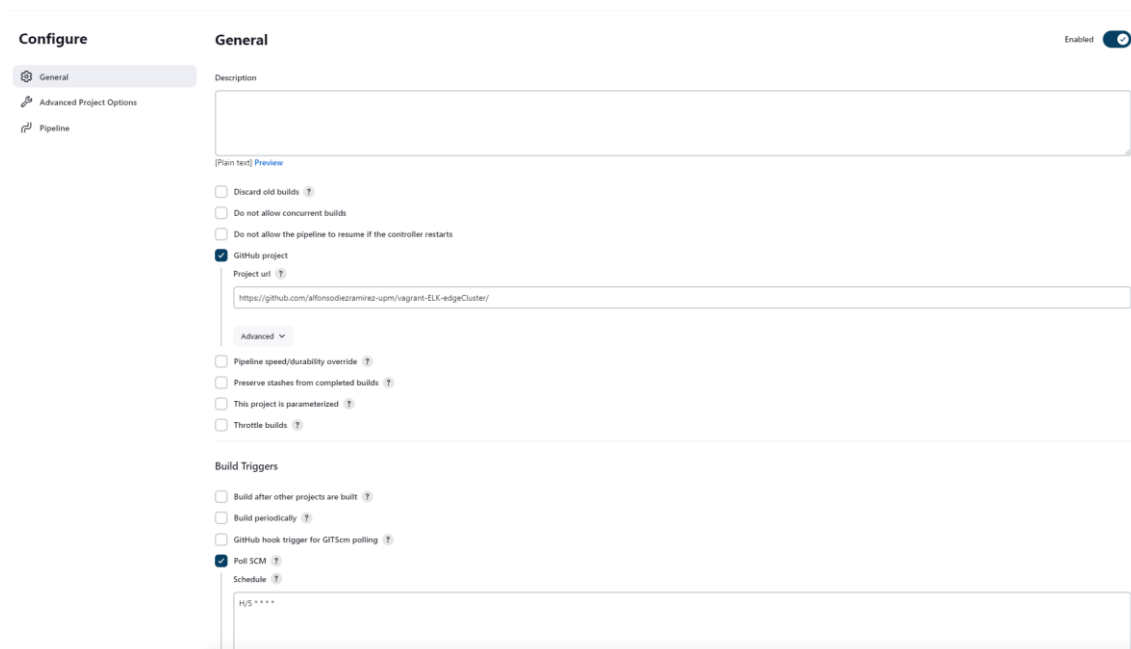


Ilustración 16 - Paso 4 de la configuración de Jenkins

5. Por último, copiamos el contenido del fichero del proyecto “pipeline-jenkins.txt” en la sección pipeline, indicando que se trata de un pipeline script

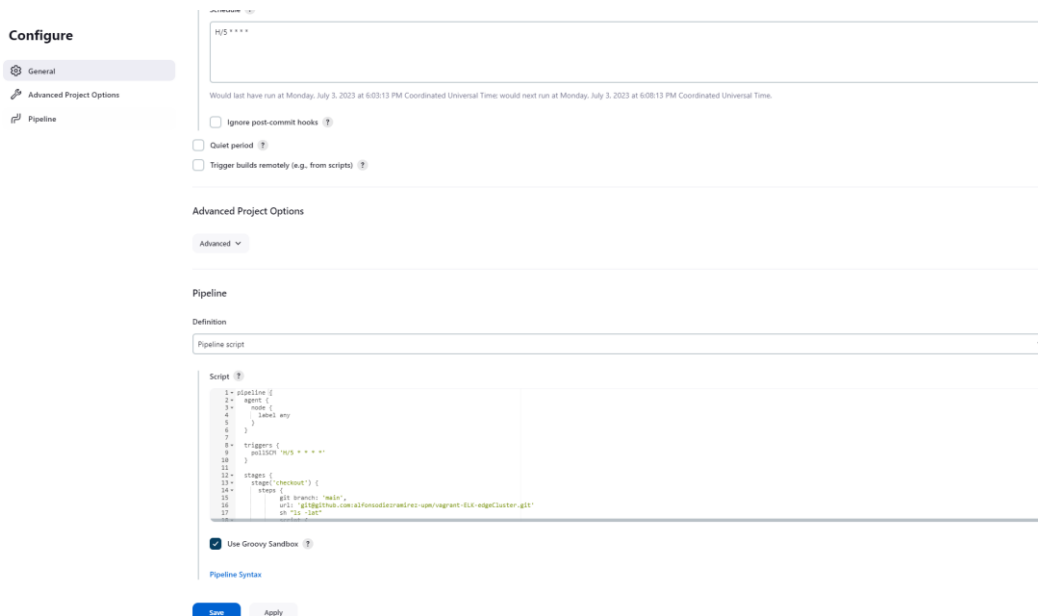


Ilustración 17 - Paso 5 de la configuración de Jenkins

A continuación, configuramos las credenciales necesarias, estas son las siguientes:

Kubemaster; Para conectarnos por ssh a la maquina master del cluster de k3s, configuramos unas credenciales que permitan tener el certificado y el usuario que utilizaremos.

1. Accedemos a la sección Manage Jenkins > Credentials y creamos una nueva credencial global
2. Seleccionamos el tipo "SSH Username with private key"
3. Escribimos el id "kubemaster" para que coincida con lo que el pipeline buscará
4. Escribimos el username "vagrant"
5. En la sección Private Key, clickamos sobre "Enter directly" y añadimos la private key de vagrant. Obtenemos la private key de vagrant que se encuentra en la ruta relativa del proyecto `./vagrant/machines/kubemaster/virtualbox/private_key`. Este fichero se actualiza si ejecutamos `vagrant up` sobre todo el proyecto, por lo que es recomendable revisarla y sustituirla periódicamente.

dockerhub; Para hacer push al registro de dockerhub donde se almacenan las imágenes.

1. Accedemos a la sección Manage Jenkins > Credentials y creamos una nueva credencial global
2. Seleccionamos el tipo "username with password"
3. Escribimos el id "dockerhub" para que coincida con lo que el pipeline buscará
4. Escribimos el username "alfonsoiot"
5. En la sección password, escribimos la contraseña para dockerhub asociada a la cuenta.
6. *Es importante matizar que se puede sustituir esta credencial por cualquier otra cuenta de dockerhub pero que esta deberá ser adaptada también en el pipeline.

El resultado final de las credenciales debe ser similar al siguiente:

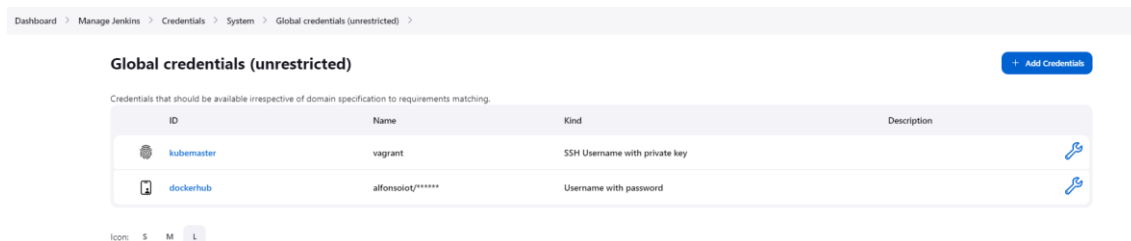


Ilustración 18 - Resultado final de la configuración de credenciales en Jenkins

3.3.2. Acceso al Dashboard de Kubernetes para K3S

El dashboard de kubernetes se instala automáticamente mediante el script de provisión de kubemaster, sin embargo, dado que el servicio que expone reserva un puerto aleatorio en el rango 30000-32000, deberemos acudir a un fichero compartido para obtener la url para el acceso.

La ruta donde se vuelca esta url es “./shared/K3SDashboard/dashboardURL”.

Una vez accedamos a dicha url, nos encontraremos con la pantalla de login del dashboard, para acceder deberemos utilizar un token que se genera durante la ejecución del script de provisión y que también se alberga en el sistema de ficheros compartidos, en la ruta “./shared/K3SDashboard/admin-user_token”, de la siguiente forma:

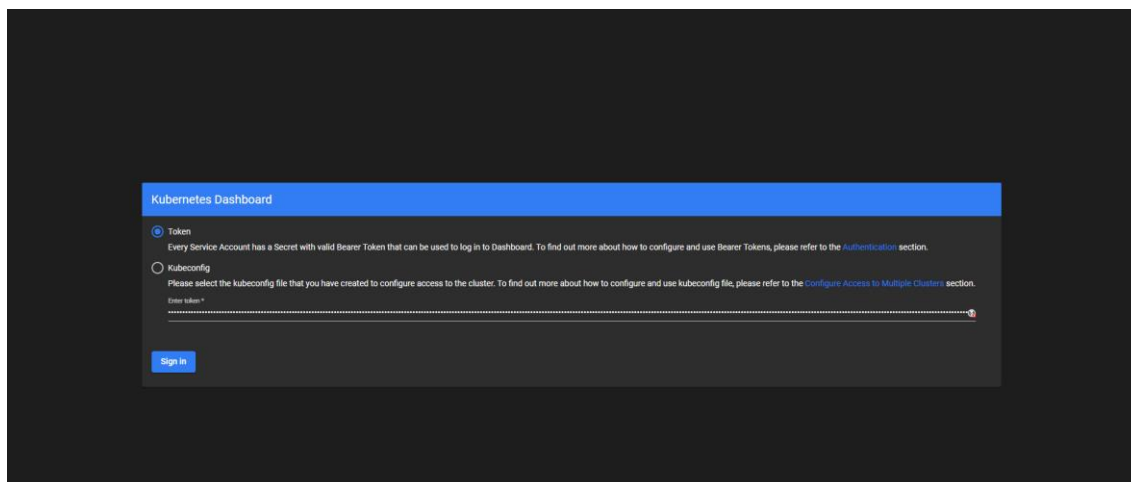


Ilustración 19 - Visualización de la pantalla de login del dashboard de kubernetes

Una vez dentro nos encontraremos con el dashboard, donde podremos interactuar con nuestros pods, deployments y demás elementos de kubernetes de manera más sencilla y visual que utilizando la línea de comandos.

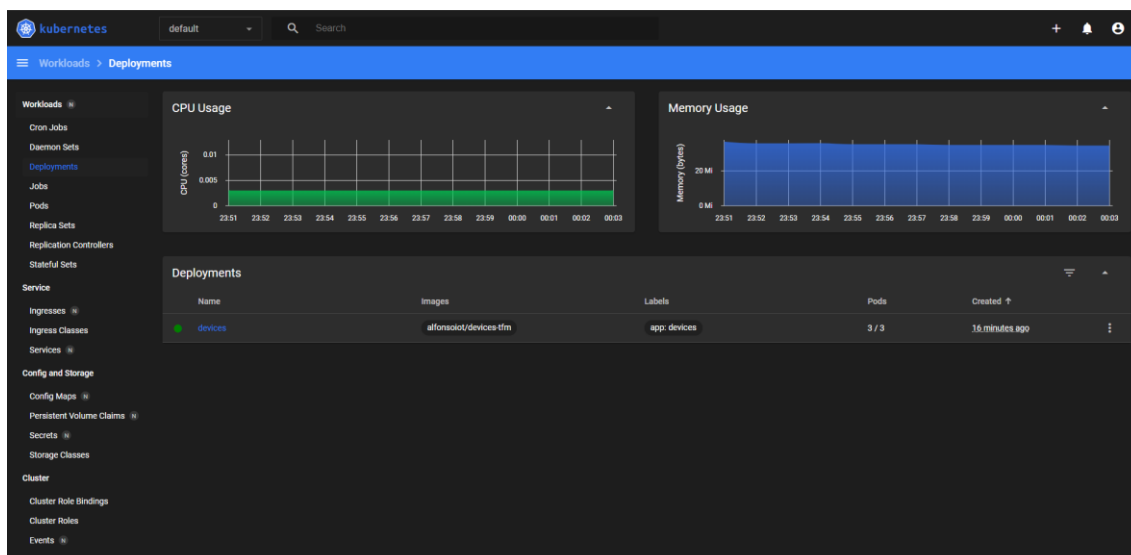


Ilustración 20 - Visualización del dashboard de kubernetes de la sección de Deployments, con un deployment llamado devices con 3 pods.

3.4. Pruebas y validación

En cada iteración definida en el apartado 1.4 como parte de la metodología y planificación del proyecto 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:

3.4.1. 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
Scenari	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;
Scenari	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;
Scenari	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	

3.4.2. Pruebas Integradas

Feature:	Conformación del cluster de K3S
Como desarrollador Quiero que los nodos del cluster de K3S se interconecten formando un cluster 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
<p>Given que el clúster K3S y Mosquitto están configurados y conectados correctamente y se generan datos de prueba desde los dispositivos simulados</p> <p>When los datos se envían a Mosquitto</p> <p>Then el mensaje se recibe en el tópico dispositivos/{idGenerado} y un suscriptor lo puede recibir;</p>	

Feature:	Pruebas de integración entre Mosquitto y Filebeat
<p>Como miembro del equipo de desarrollo</p> <p>Quiero asegurarme de que Mosquitto y Filebeat se integren correctamente</p> <p>Para garantizar la correcta transmisión y procesamiento de datos desde Mosquitto a Filebeat</p>	
Scenario	Enviar y procesar datos correctamente desde los dispositivos simulados en K3S
<p>Given que el clúster K3S y Mosquitto están configurados y conectados correctamente y se generan datos de prueba desde los dispositivos simulados</p> <p>When los datos se envían a Mosquitto</p> <p>Then el mensaje se recibe en el tópico dispositivos/{idGenerado} y un suscriptor lo puede recibir;</p>	

3.4.3. Pruebas End to End

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

4. Resultados & Discusión

Fruto de este PFM se han obtenido los siguientes resultados:

- **Implantación exitosa de cultura DevOps**

Durante el desarrollo del proyecto, se logró implementar de manera exitosa un enfoque DevOps en un proyecto de IoT centrado en la monitorización de sistemas IoT. Se establecieron los procesos y las herramientas necesarias para ello y los mecanismos para facilitar el trabajo de los diferentes equipos aplicando los principios culturales de DevOps.

Discusión:

La implementación exitosa del enfoque DevOps en el proyecto de IoT ha demostrado ser altamente beneficioso. Se observó una mejora significativa en la calidad del software, con una detección temprana de errores y una mayor eficiencia en el proceso de desarrollo. La integración continua y la monitorización del entorno permitieron identificar y corregir problemas de manera ágil, lo que resultó en una reducción en el número de errores encontrados en las diferentes etapas del proyecto.

Además, la adopción de DevOps facilita la colaboración y la comunicación entre los diferentes equipos hipotéticamente involucrados en el proyecto, dado que se trata de un proyecto de carácter unipersonal se han establecido los mecanismos. La formación de equipos multidisciplinares permite un intercambio de conocimientos más fluido y una toma de decisiones conjunta basada en una visión compartida del proyecto. Esto contribuye a una mayor eficiencia operativa y a una mejor comprensión de los requisitos y necesidades del sistema IoT.

En términos de entrega continua, el uso de Jenkins como herramienta de CI/CD agilizó el ciclo de vida del desarrollo y permitió una entrega rápida y confiable de cambios en el sistema. Esto ha sido fundamental para depurar la solución de una forma ágil e implementar los cambios que han ido surgiendo como resultado de obtener una visión más precisa de las posibilidades de las diferentes tecnologías.

En resumen, la implementación del enfoque DevOps en el proyecto de IoT ha demostrado ser altamente exitosa, mejorando la calidad del software, facilitando la colaboración entre equipos y permitiendo una entrega continua y eficiente. Estos resultados respaldan la importancia y los beneficios de adoptar los principios DevOps en el ámbito de IoT y brindan una base sólida para futuras investigaciones y aplicaciones prácticas en este campo.

- **Despliegue de dispositivos más rápido y eficiente que de forma manual**

Como parte del desarrollo del proyecto, se logró un despliegue más rápido y eficiente de dispositivos en el cluster de k3s mediante el uso de Jenkins en comparación con el despliegue manual. El proceso automatizado implementado en Jenkins permitió una configuración y despliegue más ágil de los dispositivos

IoT en el cluster, reduciendo significativamente el tiempo requerido y minimizando la posibilidad de errores humanos.

Discusión:

La automatización del despliegue de dispositivos en el cluster de k3s a través de Jenkins ha demostrado mejorar en términos de eficiencia en los tiempos de entrega. El proceso manual requeriría un tiempo considerable para configurar y desplegar cada dispositivo de forma individual, lo que podría ser propenso a errores y retrasos. Sin embargo, la implementación de Jenkins permite la creación de un flujo de trabajo automatizado y estandarizado, donde los dispositivos se configuran y despliegan de manera rápida y homogénea.

Además, el uso de Jenkins facilitó la gestión y monitorización del proceso de despliegue. Se establecieron registros del proceso que permitieron un seguimiento y análisis continuo de las tareas realizadas. Esto ha proporcionado durante el proyecto una mayor visibilidad y control sobre el estado del despliegue, detectando posibles problemas de forma anticipada.

- **Despliegue de Infraestructura**

Se logró el despliegue de la infraestructura utilizando Vagrant, permitiendo la creación y configuración eficiente de máquinas virtuales en el entorno de desarrollo. Mediante el uso de un archivo de configuración declarativo y sendos archivos de provisión, se consiguió definir y reproducir rápidamente el entorno de trabajo necesario para nuestro proyecto.

Un ejemplo de tiempo de despliegue es el siguiente:

```
PS C:\Users\alfon\Desktop\TFM\VagrantProject\vagrant_getting_started\vagrant-ELK-edgeCluster> Get-Date
miércoles, 12 de julio de 2023 18:07:35

PS C:\Users\alfon\Desktop\TFM\VagrantProject\vagrant_getting_started\vagrant-ELK-edgeCluster> vagrant up
Bringing machine 'jenkins' up with 'virtualbox' provider...
Bringing machine 'elastic' up with 'virtualbox' provider...
Bringing machine 'kibana' up with 'virtualbox' provider...
Bringing machine 'logstash' up with 'virtualbox' provider...
Bringing machine 'mosquitto' up with 'virtualbox' provider...
Bringing machine 'kubemaster' up with 'virtualbox' provider...
Bringing machine 'kubeworker1' up with 'virtualbox' provider...
Bringing machine 'kubeworker2' up with 'virtualbox' provider...
==> jenkins: Importing base box 'ubuntu/focal64'...
```

Ilustración 21 - Fecha antes de la ejecución del comando "vagrant up"

```
kubeworker2: [INFO] Creating killall script /usr/local/bin/k3s-killall.sh
kubeworker2: [INFO] Creating uninstall script /usr/local/bin/k3s-agent-uninstall.sh
kubeworker2: [INFO] env: Creating environment file /etc/systemd/system/k3s-agent.service.env
kubeworker2: [INFO] systemd: Creating service file /etc/systemd/system/k3s-agent.service
kubeworker2: [INFO] systemd: Enabling k3s-agent unit
kubeworker2: Created symlink from /etc/systemd/system/multi-user.target.wants/k3s-agent.service to /etc/systemd
kubeworker2: [INFO] systemd: Starting k3s-agent
PS C:\Users\alfon\Desktop\TFM\VagrantProject\vagrant_getting_started\vagrant-ELK-edgeCluster> Get-Date

miércoles, 12 de julio de 2023 18:25:03

PS C:\Users\alfon\Desktop\TFM\VagrantProject\vagrant_getting_started\vagrant-ELK-edgeCluster>
```

Ilustración 22 - Fecha tras la ejecución completa del comando vagrant up

La ejecución completa del VagrantFile en el ejemplo ha tardado 17 minutos y 28 segundos, algo que, de tener que configurarse manualmente, incluso siguiendo un procedimiento muy detallado y simplificado, habría llevado mucho más tiempo y supuesto la posibilidad de múltiples errores manuales.

Por tanto, la automatización proporcionada por Vagrant ha agilizado significativamente el proceso de creación de las máquinas virtuales, minimizando los errores humanos y reduciendo el tiempo necesario para tener un entorno de desarrollo listo para trabajar.

Discusión:

La adopción de Vagrant para el despliegue de infraestructura ofreció varias ventajas notables. En primer lugar, la configuración declarativa permitió una fácil reproducción del entorno en diferentes máquinas, asegurando que todos los miembros del equipo trabajaran en un entorno idéntico. Esto garantiza una mayor coherencia en el desarrollo y facilita la colaboración entre los miembros del hipotético equipo.

Además, la facilidad de uso de Vagrant simplifica el proceso de configuración y gestión de las máquinas virtuales. La creación de scripts de provisión permitió instalar y configurar las dependencias necesarias de manera automatizada, evitando la configuración manual repetitiva y minimizando los errores humanos. Para ilustrar el despliegue de infraestructura mediante Vagrant, se adjunta una captura de pantalla que muestra el entorno de desarrollo con las máquinas virtuales desplegadas. Esta visualización proporciona una representación clara y tangible del resultado obtenido, demostrando la eficacia y la viabilidad del enfoque utilizado.

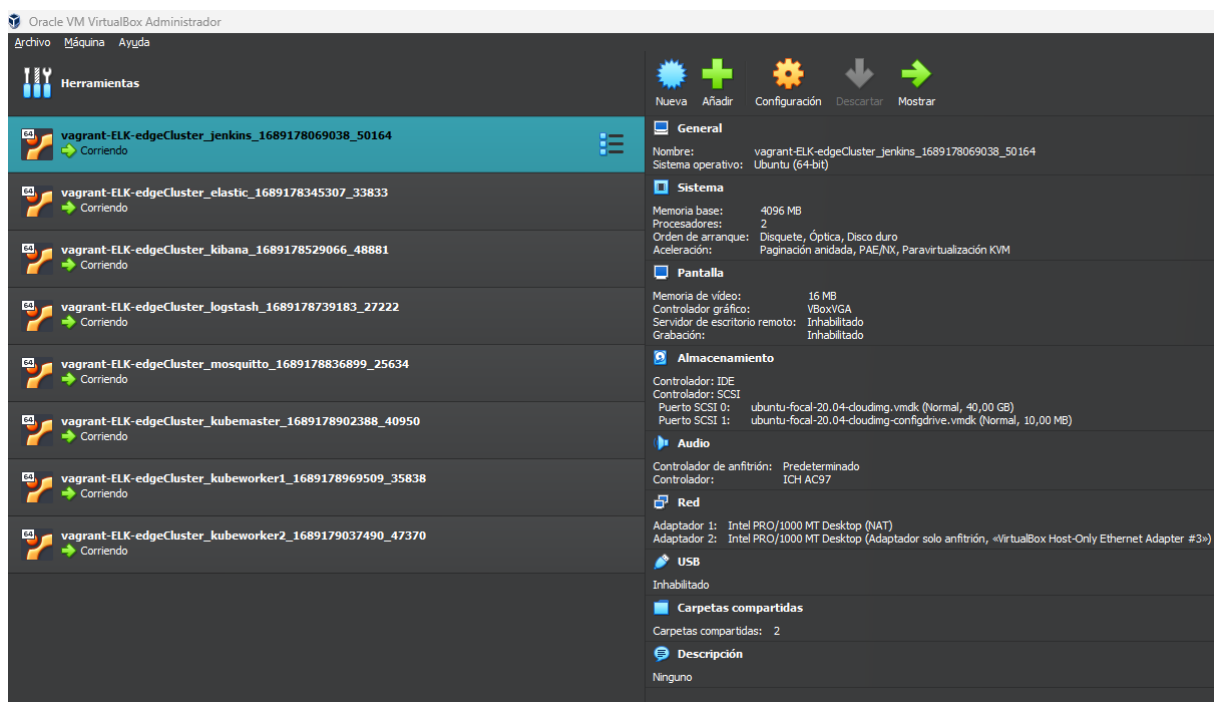


Ilustración 23 - Evidencia del despliegue de las máquinas virtuales de la plataforma DevOps en VirtualBox

- **Monitorización de los dispositivos en Kibana**

La monitorización de los dispositivos se realizó con éxito mediante la configuración del index pattern y los dashboard de visualización en Kibana, se consiguió recopilar, analizar y visualizar los datos generados por los dispositivos IoT de manera centralizada y accesible.

Discusión:

La monitorización de los dispositivos en Kibana brinda valiosos resultados para nuestro proyecto. En primer lugar, nos permite detectar y solucionar de manera temprana posibles fallos o problemas en los dispositivos. Los datos monitorizados, como la temperatura, la humedad y la velocidad del viento, fueron recopilados mediante el muestreo de datos reales y representados visualmente en Kibana, lo que nos permite identificar patrones, anomalías y tendencias.

Además, la capacidad de visualización de Kibana permite crear paneles personalizados con gráficas y estadísticas basadas en los datos monitorizados. Esto facilita la comprensión y el análisis de los datos generados por los dispositivos IoT, lo que a su vez nos brinda una base sólida para la toma de decisiones informadas respecto al entorno.

Para complementar la información, se adjunta una captura de pantalla que muestra un panel de visualización en Kibana con los datos monitorizados de temperatura, humedad y velocidad del viento. Estos datos, representados en forma de gráficas y métricas, nos brindan una visión clara y detallada del comportamiento de los dispositivos IoT en tiempo real.



Ilustración 24 - Dashboard creado para el proyecto con los datos medidos por los dispositivos (media, máxima y mínima)

- **Monitorización del cluster de K3S**

La monitorización del cluster de K3S mediante el dashboard de Kubernetes proporciona una visión completa y detallada del estado y rendimiento del cluster. Mediante la visualización de métricas y registros en el dashboard, se ha podido supervisar la salud de los nodos, la utilización de recursos y el rendimiento de las aplicaciones desplegadas en el cluster.

Discusión:

La monitorización del cluster de K3S mediante el dashboard de Kubernetes proporciona una experiencia de usuario mejorada en comparación con el uso de la consola de comandos. El dashboard brinda una interfaz visual intuitiva y amigable que facilita la supervisión y gestión del cluster, sin requerir conocimientos profundos de línea de comandos. Esto permitió a los usuarios, incluso aquellos con menos experiencia técnica, acceder y comprender rápidamente la información esencial del cluster.

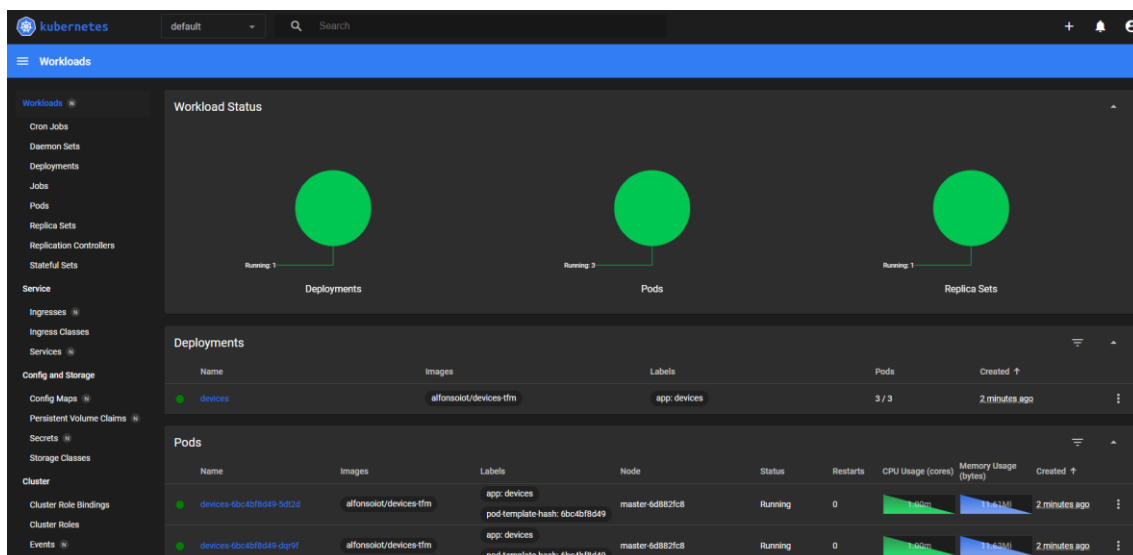


Ilustración 25 - Dashboard de kubernetes donde se puede visualizar el número de despliegues, los pods y replica sets

Además, el dashboard ofrece la capacidad de monitorizar los recursos del cluster de forma más sencilla. Podemos visualizar métricas en tiempo real, como la utilización de CPU, la memoria y el almacenamiento, lo que nos permitió identificar rápidamente posibles cuellos de botella y realizar ajustes oportunos para optimizar el rendimiento del cluster.

Otra ventaja significativa es la capacidad de escalar un despliegue en el cluster de manera rápida a través del dashboard. permite visualizar el estado actual de los nodos y los recursos disponibles, lo que nos permite tomar decisiones informadas sobre la escalabilidad del cluster. Gracias a esto, se puede decidir si escalar vertical u horizontalmente según las necesidades del proyecto actual y también estar preparados para futuros despliegues en proyectos evolutivos.

5. Conclusiones y Líneas futuras

5.1. Conclusiones

En este proyecto, se han logrado cumplir los siguientes objetivos establecidos originalmente:

- **OBJ1:** Se han aplicado los principios DevOps en el ámbito del IoT a lo largo de todo el ciclo de vida del proyecto, posibilitando mediante la aplicación cultural y las herramientas utilizadas, una colaboración efectiva entre los hipotéticos equipos de desarrollo y operaciones, representados por el autor de este PFM.
- **OBJ2:** Se ha establecido una sólida monitorización de sistemas IoT, permitiendo obtener una visión en tiempo real de los datos generados por los dispositivos y de su infraestructura.
- **OBJ3:** Se implementó un proceso de entrega continua utilizando herramientas de CI/CD como Jenkins, asegurando la eficiencia y agilidad en el despliegue de cambios.
- **OBJ4:** Se ha utilizado GitHub como herramienta de control de versiones, garantizando la trazabilidad y colaboración efectiva entre los equipos.
- **OBJ5:** Se ha aplicado el enfoque de IaC en el entorno de desarrollo mediante Vagrant, asegurando la consistencia y portabilidad del entorno.
- **OBJ6:** Se ha implementado una solución de monitorización utilizando el stack ELK, permitiendo el análisis y visualización de los registros de los sistemas IoT.
- **OBJ7:** Se ha diseñado una arquitectura escalable y resiliente, que permite escalar ante picos de carga, aunque no de forma automática, y garantizar alta disponibilidad en los puntos críticos de la misma.
- **OBJ9:** Se establecieron procesos y herramientas de automatización para la gestión de infraestructuras y despliegue, reduciendo el tiempo y los errores asociados a tareas manuales.
- **OBJ10:** Se promovió la documentación exhaustiva y actualizada, facilitando el mantenimiento y el intercambio de conocimientos.

Sin embargo, los siguientes objetivos no se han alcanzado en su completitud:

- **OBJ8:** Se ha implementado utilizando prácticas de desarrollo seguro, pero no se han garantizado fehacientemente las características de seguridad necesarias en un entorno como este al no asegurar el acceso a recursos mediante TLS y certificados firmados por entidades confiables que permitan que la comunicación sea confiable.

Pese a que no se han cumplido los objetivos en su completitud, sí que es importante destacar que la plataforma DevOps se caracteriza por estar completamente automatizada en todos las líneas planteadas originalmente y concluir que esta automatización de procesos y tareas ha agilizado el desarrollo y despliegue de los sistemas IoT, reduciendo los tiempos de entrega y minimizando errores, lo que supone un gran hito de cara a futuros proyectos de este ámbito que se basen en las líneas desarrolladas en este PFM o para futuras ampliaciones del mismo.

5.2. Ampliación funcional

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.

5.3. Seguridad y privacidad

Implementación de certificados válidos y confiables en todos los componentes

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 auto firmados. 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] R. A. J. P. a. J. G. Jessica Díaz, «DevOps in practice: an exploratory case study,» vol. 1, 2018.
- [2] L. S. Vailshery, «Statista.com,» 28 Abril 2023. [En línea]. Available: <https://www.statista.com/topics/9369/devops/#topicOverview>.
- [3] L. S. Vailshery, «Statista.com,» 28 Abril 2023. [En línea]. Available: <https://www.statista.com/topics/2637/internet-of-things/#topicOverview>.
- [4] J. & F. D. Humble, Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation, Addison-Wesley Professional, 2010.
- [5] L. K. P. O. M. Lwakatare, «Relationship of DevOps to Agile, Lean and Continuous Deployment.,» 2016.
- [6] A. H.-L. E. C. a. A. L. A. Miransky, «Operational-Log Analysis for Big Data Systems: Challenges and Solutions,» vol. 33, 2016.
- [7] J. Palat, «Introducing vagrant,» 2012.
- [8] jenkins, «plugins.jenkins.io,» [En línea]. Available: <https://plugins.jenkins.io/>.
- [9] J. F. Smart, Jenkins: The Definitive Guide, O'Reilly, 2011.
- [10] G. D. P. W. a. J. H. E. Kim, The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations, 2016.
- [11] L. I. A. & M. G. Atzori, «The internet of things: A survey,» *Computer networks*, vol. 54, pp. 2787-2805, 2010.
- [12] J. B. R. M. S. & P. M. Gubbi, «Internet of Things (IoT): A vision, architectural elements, and future directions,» *Future generation computer systems*, vol. 29, pp. 1645-1660, 2013.
- [13] Emnify, «Emnify.com,» 10 Enero 2023. [En línea]. Available: <https://www.emnify.com/blog/iot-challenges-2023>. [Último acceso: 03 Junio 2023].
- [14] I. K. A. A. A. a. E. -N. H. M. Aazam, «Cloud of Things: Integrating Internet of Things and cloud computing and the issues involved,» de *Proceedings of 2014 11th International Bhurban Conference on Applied Sciences & Technology (IBCAST)*, Islamabad, Pakistan, 2014.

- [15] J. D. a. J. E. P. R. López-Viana, «Continuous Deployment in IoT Edge Computing : A GitOps implementation,» de *Iberian Conference on Information Systems and Technologies (CISTI)*, Madrid, Spain, 2022.
- [16] G. G. J. H. a. N. S. C. Ebert, «DevOps,» vol. 33, nº 3, 2016.
- [17] Atlassian, «atlassian.com,» [En línea]. Available: <https://www.atlassian.com/es/devops/what-is-devops#:~:text=La%20premisa%20clave%20de%20DevOps,ciclo%20de%20desarrollo%20e%20implementaci%C3%B3n..>
- [18] M. Virmani, «Understanding DevOps & bridging the gap from continuous integration to continuous delivery,» Galicia, Spain, 2015.
- [19] L. b. O. a. A. K. V. Mohan, «BP: Security Concerns and Best Practices for Automation of Software Deployment Processes: An Industrial Case Study,» 2018.
- [20] Google, «Trends,» [En línea]. Available: <https://trends.google.es/trends/explore?cat=174&date=all&q=SecDevOps&hl=es>.
- [21] Google, «Trends,» [En línea]. Available: <https://trends.google.es/trends/explore?cat=174&date=all&q=DevOps&hl=es>.
- [22] Google, «Trends,» [En línea]. Available: <https://trends.google.es/trends/explore?cat=174&date=all&q=SecDevOps,DevOps&hl=es>.
- [23] Hashicorp, «developer.hashicorp,» [En línea]. Available: <https://developer.hashicorp.com/vagrant/docs/cli/up>.
- [24] Hashicorp, «developer.hashicorp.com,» [En línea]. Available: <https://developer.hashicorp.com/vagrant/docs/cli/ssh>.
- [25] Hashicorp, «developer.hashicorp.com,» [En línea]. Available: <https://developer.hashicorp.com/vagrant/docs/cli/destroy>.
- [26] M. Hashimoto, Vagrant Up and Running, O'Reilly, 2013.
- [27] S. D. a. K. Swaminathan, «Efficient surveillance and monitoring using the ELK stack for IoT powered Smart Buildings,» de *2nd International Conference on Inventive Systems and Control (ICISC)*, Coimbatore, India, 2018.
- [28] Elasticsearch, «elastic.co,» [En línea]. Available: <https://www.elastic.co/guide/en/elasticsearch/reference/8.8/elasticsearch-intro.html>.

- [29] GeekCulture, «medium.com,» 23 Diciembre 2022. [En línea]. Available: <https://medium.com/geekculture/elasticsearch-architecture-1f40b93da719>.
- [30] Elasticsearch, «elastic.co,» [En línea]. Available: <https://www.elastic.co/guide/en/logstash/current/execution-model.html>.
- [31] Elasticsearch, «elastic.co,» [En línea]. Available: <https://www.elastic.co/guide/en/kibana/master/introduction.html>.
- [32] kubernetes, «kubernetes.io,» [En línea]. Available: <https://kubernetes.io/docs/concepts/overview/>.
- [33] k3s, «docs.k3s.io,» [En línea]. Available: <https://docs.k3s.io/>.
- [34] k3s, «docs.k3s.io,» [En línea]. Available: <https://docs.k3s.io/cli>.
- [35] Kubernetes, «kubernetes.io,» [En línea]. Available: <https://kubernetes.io/docs/reference/kubectl/>.
- [36] mosquito, «mosquitto.org,» [En línea]. Available: <https://mosquitto.org/man/mosquitto-8.html>.
- [37] The HiveMQ Team, «hivemq.com,» 16 Febrero 2015. [En línea]. Available: <https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels/>. [Último acceso: 03 Junio 2023].
- [38] Mosquitto, «mosquitto.org,» [En línea]. Available: <https://mosquitto.org/man/mosquitto-tls-7.html>. [Último acceso: 03 Junio 2023].
- [39] HiveMQ, «HiveMQ.com,» 9 Marzo 2015. [En línea]. Available: <https://www.hivemq.com/blog/mqtt-essentials-part-9-last-will-and-testament/>. [Último acceso: 04 Junio 2023].
- [40] Jenkins, «jenkins.io,» [En línea]. Available: <https://www.jenkins.io/doc/>.
- [41] «<https://docs.k3s.io/installation/kube-dashboard>,» [En línea].
- [42] Steve, «steves-internet-guide,» 14 Febrero 2023. [En línea]. Available: <http://www.steves-internet-guide.com/mosquitto-tls/>.
- [43] W. d. D. V. P. a. A. P. Alessio Botta, «Integration of Cloud computing and Internet of Things: A survey,» *Future Generation Computer Systems*, vol. 56, nº 0167-739X, pp. 684-700, 2016.
- [44] N. M. Ishtiaq Ali, «Virtual Machines and Networks - Installation, Performance Study, Advantages and Virtualization Options,» *International Journal of Network Security and its Applications*, vol. 3, nº arXiv:1105.0061, pp. 1-15, 2011.

- [45] J. R. J. F. S. H. A. N. S. a. Y. C. M. Alam, «Orchestration of Microservices for IoT Using Docker and Edge Computing,» *IEEE Communications Magazine*, vol. 56, nº 9, pp. 118-123, 2018.

Anexos

I. Código

El código se puede localizar íntegramente junto con el resto de elementos del PFM en el siguiente repositorio público de GitHub: [Repositorio vagrant-ELK-edgeCluster](#)

Vagrantfile

El código del vagrantfile realiza las siguientes acciones:

1. Se define una máquina virtual para Jenkins con la imagen base "ubuntu/focal64", equivalente a Ubuntu:20.03LTD, esto se debe a que la versión de Ubuntu 14 que se utiliza en el resto de MV no está soportada por Jenkins y la 18.03 se encuentra en end of life. Se le asigna la dirección ip 192.168.22.40 y se hace forward de los puertos 2376, 8080 y 50000, aunque solo sería estrictamente necesario el 8080. Se sincroniza la carpeta local "./shared/" con la ruta "/sharedData" dentro de la máquina virtual que utilizaremos para volcar el token del usuario admin de Jenkins que nos será solicitado durante el primer arranque y hasta que creamos un usuario nominal. Por último se ejecuta el script de provisión "jenkins2.sh" que se encarga de la instalación, configuración básica y obtención del token.
2. 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.
3. 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.
4. 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".

5. 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.
6. 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".
7. 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|
  config.ssh.forward_agent = true
  config.vm.define "jenkins" do |jenkins|
    jenkins.vm.box = "ubuntu/focal64"
    jenkins.vm.hostname = "jenkins"
    jenkins.vm.provider "virtualbox" do |jen|
      jen.gui = false
      jen.cpus = 2
      jen.memory = "4096"
    end
    jenkins.vm.network "private_network", ip: "192.168.22.40"
    jenkins.vm.network "forwarded_port", guest: 2376, host: 2376
    jenkins.vm.network "forwarded_port", guest: 8080, host: 8080
    jenkins.vm.network "forwarded_port", guest: 50000, host:
50000
    jenkins.vm.synced_folder "./shared/", "/sharedData"
    jenkins.vm.provision "shell", path: "jenkins2.sh"
  end
  #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
```

Jenkins2.sh

1. Se descarga y agrega la clave de apt de Jenkins al archivo de claves /usr/share/keyrings/jenkins-keyring.asc y se agrega la entrada del repositorio de Jenkins al archivo /etc/apt/sources.list.d/jenkins.list.
2. Se actualiza la lista de paquetes disponibles en los repositorios configurados mediante apt-get update.
3. Se instala el paquete default-jre y default-jdk para tener la versión predeterminada de Java en el sistema, siendo este un requisito para la ejecución de Jenkins.
4. Se instala el paquete git para el control de versiones.
5. Se instala el paquete git-ftp para la sincronización de repositorios Git con servidores FTP.
6. Se instala Docker siguiendo los siguientes pasos:
 - Se configuran los repositorios y las claves de apt para Docker.
 - Se actualiza la lista de paquetes disponibles.
 - Instalamos el paquete docker-ce, docker-ce-cli y containerd.io.
7. Se instala y configura Jenkins de la siguiente forma:
 - Se instala el paquete jenkins en el sistema.
 - Se agrega el usuario jenkins al grupo docker para permitir el acceso a Docker sin ser root.
 - Se inicia el servicio de Jenkins.
 - Se espera 1 minuto a que el servicio inicie correctamente y se genere el archivo initialAdminPassword (se debe tener en cuenta que en portátiles con menor rendimiento quizás se deba esperar más u obtener esta password accediendo a la maquina por ssh y consultando el fichero.)

8. Se copia el archivo initialAdminPassword generado por Jenkins en la ubicación /var/lib/jenkins/secrets/ a la carpeta compartida /sharedData/jenkinsadmin.
9. Se reinicia el servicio de Jenkins para asegurarnos de que el grupo de Docker cuenta con el usuario de Jenkins dado que en ocasiones solo con iniciarlo después de su adhesión no es suficiente y da error de permisos al tratar de acceder al Daemon.

```
#!/bin/bash
echo "añadiendo apt-keys de jenkins"
curl -fsSL https://pkg.jenkins.io/debian/jenkins.io-2023.key | sudo tee \
/usr/share/keyrings/jenkins-keyring.asc > /dev/null
echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \
https://pkg.jenkins.io/debian binary/ | sudo tee \
/etc/apt/sources.list.d/jenkins.list > /dev/null

echo "actualizando apt-get"
sudo apt-get -qq update

echo "Instalando default-java"
sudo apt-get -y install default-jre > /dev/null 2>&1
sudo apt-get -y install default-jdk > /dev/null 2>&1

echo "Instalando git"
sudo apt-get -y install git > /dev/null 2>&1

echo "Instalando git-ftp"
sudo apt-get -y install git-ftp > /dev/null 2>&1

echo "instalando docker"
sudo apt-get update
sudo apt-get install ca-certificates curl gnupg
sudo install -m 0755 -d /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg -
dearmor -o /etc/apt/keyrings/docker.gpg
sudo chmod a+r /etc/apt/keyrings/docker.gpg
echo \
"deb [arch="$(dpkg --print-architecture)" signed-
by=/etc/apt/keyrings/docker.gpg]
https://download.docker.com/linux/ubuntu \
"$(. /etc/os-release && echo "$VERSION_CODENAME)" stable" | \
sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd.io -y >
/dev/null 2>&1

echo "Instalando jenkins"
sudo apt-get -y install jenkins --allow-unauthenticated > /dev/null
2>&1
sudo usermod -aG docker jenkins
sudo service jenkins start

sleep 1m
```

```
cp /var/lib/jenkins/secrets/initialAdminPassword  
/sharedData/jenkinsadmin  
sudo service jenkins restart
```

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`.
12. Esperamos 120 segundos para garantizar que el servicio de kibana se inicie completamente
13. Lanzamos una petición POST utilizando `curl` para cargar el dashboard previamente desarrollado y que se encuentra en la ruta compartida `"/sharedData/kibana/dashboardanddependencies.json"`


```
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/elasticsearch-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

sleep 120

#Cargamos el dashboard y sus dependencias mediante curl, para
disponer de ello desde el arranque, si lo cambiamos, deberemos
exportar el dashboard utilizando el curl curl GET
192.168.22.12:5601/api/kibana/dashboards/export?dashboard=15393120-
1ab0-11ee-85ef-d37137d9859b
admin=$(cat /sharedData/admin)
curl --location --request POST
'192.168.22.12:5601/api/kibana/dashboards/import?force=True' --
header 'kbn-xsrf: reporting' --user "elastic:$admin" --header
'Content-Type: application/json' -d
@/sharedData/kibana/dashboardanddependencies.json
```

logstashProvision.sh

Este código 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/elasticsearch-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.
6. Despliegue del dashboard de kubernetes [41]
 1. Se establece la URL base del repositorio de Kubernetes Dashboard en GitHub.
 2. Se obtiene la última versión de Kubernetes Dashboard utilizando la URL anterior.
 3. Se crean los recursos necesarios para implementar Kubernetes Dashboard utilizando el archivo YAML recomendado para la versión obtenida anteriormente.
 4. Se crean los usuarios y roles necesarios para el acceso de administrador en Kubernetes Dashboard, utilizando los archivos YAML disponibles en la carpeta compartida.
 5. Se crea un token de acceso para el usuario administrador de Kubernetes Dashboard y se guarda en el archivo `"admin-user_token"` en la carpeta compartida.
 6. Se modifica el tipo de servicio de Kubernetes Dashboard de `"ClusterIP"` a `"LoadBalancer"`, lo que permite acceder al panel de control desde fuera del clúster.

- Se obtiene el puerto en el que se ha expuesto el load balancer para indicárselo mediante un archivo de configuración al usuario y que pueda acceder al dashboard.

```
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 imagenes de prueba del entorno, con 3 replicas:
kubectl create deployment --image=alfonsoiot/devices-tfm devices
kubectl scale --replicas=3 deployment/devices

GITHUB_URL=https://github.com/kubernetes/dashboard/releases
VERSION_KUBE_DASHBOARD=$(curl -w '%{url_effective}' -I -L -s -S
${GITHUB_URL}/v2.7.0 -o /dev/null | sed -e 's|.*/||')
sudo k3s kubectl create -f
https://raw.githubusercontent.com/kubernetes/dashboard/${VERSION_KUB
E_DASHBOARD}/aio/deploy/recommended.yaml
sudo k3s kubectl create -f /sharedData/K3SDashboard/dashboard.admin-
user.yml -f /sharedData/K3SDashboard/dashboard.admin-user-role.yml

sudo k3s kubectl -n kubernetes-dashboard create token admin-user >
/sharedData/K3SDashboard/admin-user_token

kubectl patch svc kubernetes-dashboard -n kubernetes-dashboard --
type='json' -p
' [{"op": "replace", "path": "/spec/type", "value": "LoadBalancer"}] '

PORT=$(kubectl get svc -n kubernetes-dashboard -o go-
template='{{range .items}}{{range .spec.ports}}{{if
.nodePort}}{{.nodePort}}{{"\n"}}{{end}}{{end}}{{end}}')
URL="https://192.168.22.30:${PORT}/#/login"
echo "${URL}">/sharedData/K3SDashboard/dashboardURL
```

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:

- Importa los módulos necesarios: **json**, **random**, **time**, **paho.mqtt.client** y **uuid**.
- 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. Bucle para leer y enviar datos desde un archivo CSV:
 1. Abre el archivo CSV llamado "weather_madrid_LEMD_1997_2015.csv" en modo de lectura.
 2. Recorre cada fila del archivo utilizando un objeto lector de CSV.
 3. Modifica aleatoriamente algunos valores de temperatura, humedad y viento.
 4. Crea un mensaje en formato JSON con los datos modificados y otros valores estáticos como el ID del dispositivo y la marca de tiempo.
 5. Conecta al broker MQTT especificado.
 6. Publica el mensaje JSON en el tema MQTT.
 7. Imprime el mensaje JSON y el topic en la consola.
 8. Se espera 10 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
import csv

#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 jerarquía 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 recorriendo el fichero para leer los datos del csv,
variandolos y enviandolos a MQTT
while True:
    with open("./weather_madrid_LEMD_1997_2015.csv", "r") as
csv_file:
        reader = csv.reader(csv_file)
        for row in reader:
            datetime_object = datetime.strptime(row[0], '%Y-%m-%d')
            tempModification = random.randint(-5, 5)
            humModification = random.randint(-10, 10)
            windModification = random.randint(-10, 10)
            message = {
                "deviceId": str(deviceId),
                "maxTemperature": int(row[1])+tempModification,
                "meanTemperature":int(row[2])+tempModification,
                "minTemperature":int(row[3])+tempModification,

"maxHumidity":max(0,min(100,int(row[7])+humModification)),
"meanHumidity":max(0,min(100,int(row[8])+humModification)),
"minHumidity":max(0,min(100,int(row[9])+humModification)),
                "maxWind":max(0,int(row[16])+windModification),
                "meanWind":max(0,int(row[17])+windModification),
                "isEnabled": 1,
                "timestamp": datetime_object.strftime("%Y-%m-
%dT%H:%M:%SZ")
            }

            # 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 10 segundos para no bombardear al broker
            time.sleep(10)

```

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" ]
```

Pipeline-jenkins.txt

1. La etapa 'checkout' realiza la clonación de un repositorio de Github y muestra una lista de archivos con el comando ls -lat.
2. La etapa 'Docker Build' realiza la construcción de una imagen de Docker utilizando el directorio ./DevicesSimulation, donde se encuentra el Dockerfile de la app de Python que simula los dispositivos.

3. La etapa 'Docker Push' realiza el inicio de sesión en un registro de Docker utilizando las credenciales proporcionadas y luego sube la imagen construida al repositorio en dockerhub.
4. La etapa 'Deploy pro' realiza el despliegue de la imagen de Docker en un clúster Kubernetes utilizando SSH. Elimina el despliegue existente, crea un nuevo despliegue y escala el número de réplicas a 3.

```
#!/groovy
pipeline {
  agent none
  stages {
    stage('checkout') {
      agent any
      steps {
        git branch: 'main',
        url: 'https://github.com/alfonsodiezramirez-upm/vagrant-ELK-edgeCluster.git'
        sh "ls -lat"
      }
    }
    stage('Docker Build') {
      agent any
      steps {
        sh "cd ./DevicesSimulation"
        sh "docker build -t alfonsoiot/devices-tfm:latest ./DevicesSimulation"
      }
    }
    stage('Docker Push') {
      agent any
      steps {
        withCredentials([usernamePassword(credentialsId: 'dockerhub',
passwordVariable: 'dockerHubPassword', usernameVariable: 'dockerHubUser')]) {
          sh "docker login -u ${env.dockerHubUser} -p ${env.dockerHubPassword}"
          sh "docker push alfonsoiot/devices-tfm:latest"
        }
      }
    }
    stage('Deploy pro') {
      agent any
      steps {
        sshagent(credentials : ['kubemaster']) {
          echo "deploying"
          withCredentials([sshUserPrivateKey(credentialsId: 'kubemaster',
keyFileVariable: 'identity', passphraseVariable: '', usernameVariable: 'userName')]) {
            sh "ssh-keygen -f
'/var/lib/jenkins/.ssh/known_hosts' -R '192.168.22.30' || true"
```

```
sh "ssh ${env.userName}@192.168.22.30
-o StrictHostKeyChecking=no 'kubectl delete deployments devices' || true"

sh "ssh ${env.userName}@192.168.22.30
-o StrictHostKeyChecking=no 'kubectl create deployment --image=alfonsoiot/devices-
tfm:latest devices'"

sh "ssh ${env.userName}@192.168.22.30
-o StrictHostKeyChecking=no 'kubectl scale --replicas=3 deployment/devices'"
    }
  }
}
}
}
```

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

1. Input (Entrada):

- Se utiliza el plugin "beats" para recibir datos desde un cliente Beats a través de peticiones al puerto 5044.

- `client_inactivity_timeout => 3000`: Configura el tiempo de inactividad en milisegundos después del cual se cerrará una conexión si no hay actividad.
- `port => 5044`: Especifica el puerto en el que Logstash escucha los eventos entrantes.
- `ssl => false`: Indica que la comunicación con el cliente Beats no utiliza SSL.

2. Filter (Filtro):

- Utiliza el plugin "json" para analizar el campo "message" y convertirlo en una estructura JSON. Los campos parseados se almacenan en la raíz del documento.
- `source => "message"`: Especifica el campo que se utilizará como origen para el análisis JSON.
- Utiliza el plugin "mutate" para realizar modificaciones en los campos del evento.
- `remove_field => ["message", "agent", "ecs"]`: Elimina los campos "message", "agent" y "ecs" del evento.

3. Output (Salida):

- Se utiliza el plugin "elasticsearch" para enviar los eventos procesados a un clúster de Elasticsearch.
- `hosts => ["https://192.168.22.11:9200"]`: se especifica las direcciones IP y puertos de los nodos de Elasticsearch a los que se enviara el documento fruto del pipeline.
- `index => "edgecluster-new"`: Especifica el índice de Elasticsearch en el que se almacenarán los eventos.
- `ssl_certificate_verification => false`: Desactiva la verificación del certificado SSL al conectar con Elasticsearch.
- `user => "elastic" y password => "#PASSWORD#"`: Especifica las credenciales de autenticación para conectar con Elasticsearch, dicho valor se sustituye por la password real durante el proceso de provisión.
- Utiliza el plugin "file" para escribir los eventos en un archivo de registro local.
- `path => "/var/log/logstash/output.log"`: Especifica la ruta del archivo de registro donde se guardarán los eventos.

```
input {  
  beats {
```

```
    client_inactivity_timeout => 3000
    port => 5044
    ssl => false
  }
}
filter {
  json {
    source => "message"
  }
  mutate {
    remove_field => [ "message", "agent", "ecs" ]
  }
}
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"
  }
}
```

Dashboard.admin-user.role.yml

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: admin-user
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- kind: ServiceAccount
  name: admin-user
  namespace: kubernetes-dashboard
```

Dashboard.admin-user.yml

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: admin-user
  namespace: kubernetes-dashboard
```