UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Securización depliegue IoT-Edge para entornos industriales

Trabajo Fin de Máster

Máster Universitario en Ciberseguridad y Ciberinteligencia

AUTOR/A: Mahedero Biot, Francisco

Tutor/a: Palau Salvador, Carlos Enrique

CURSO ACADÉMICO: 2021/2022

Resumen

El presente trabajo Fin de Máster expone una solución de securización de un nodo Edge motivado por el emergente marco para la adquisición, procesado y análisis en los sistemas de Internet de las cosas llamado Edge-Cloud computing. El Edge-Cloud computing consta de tres capas donde se dispone de la primera capa llamada Edge, la segunda fog computing y finalmente cloud computing.

Este trabajo se centra en la primera capa de este sistema jerárquico, es decir, el Edge computing, que ofrece en el análisis y procesamiento de datos, tiempos de respuesta mejorados y mayor disponibilidad de ancho de banda. Primeramente, se llevará a cabo un análisis del estado del arte de las tecnologías utilizadas en el paradigma del Edge-Cloud computing. En este análisis previo se estudiará cual es la mejor solución a nivel industrial tanto en seguridad como en software que ofrezca eficiencia y disponibilidad de servicios. Seguidamente, tras el análisis de la solución óptima a adoptar, se hará un estudio de que distribución de FIWARE es la más adecuada y que papel le corresponderá en la tesis. En este análisis de la herramienta, se hará hincapié en las características que nos ofrece FIWARE en su securización, tanto en el sistema de gestión de accesos e identidades como en la de las conexiones. No solo se hará el estudio previo de esta herramienta, sino también de todos los componentes como las plataformas de administración de cargas de trabajo y servicios como Kubernetes o los protocolos de conexión de los dispositivos industriales generadores de datos. Posteriormente, con las tecnologías ya definidas en la parte inicial del trabajo, se llevará a cabo de forma práctica el despliegue de la infraestructura, donde se priorizará la securización del primer nivel de la jerarquía, el nodo Edge.

Este entorno aplicará procesos de análisis y procesamiento de datos sobre información ficticia del entorno industrial. En este caso, se utilizarán datos ficticios extraídos de variadores de frecuencia mediante el protocolo MODBUS, que permitirá simular el funcionamiento de una planta industrial de la manera más realista posible. Se creará un entorno visual que permita la detección de errores en la planta de forma centralizada en Cloud. Finalmente se analizarán los resultados obtenidos y se plantearán líneas futuras de desarrollo de la tesis.

Resum

El present treball Fi de Màster exposa una solució de securització d'un node Edge motivat per l'emergent marc per a l'adquisició, processament i anàlisi en els sistemes d'Internet de les coses anomenat Edge-Cloud computing. El Edge-Cloud computing consta de tres capes on es disposa de la primera capa anomenada Edge, la segona fog computing i finalment cloud computing.

Aquest treball es centra en la primera capa d'aquest sistema jeràrquic, és a dir, l'Edge Computing, que ofereix en l'anàlisi i processament de dades, temps de resposta millorats i major disponibilitat d'amplada de banda. Primerament, es durà a terme una anàlisi de l'estat de l'art de les tecnologies utilitzades en el paradigma de l'Edge-Cloud Computing. En aquesta anàlisi previ s'estudiarà com és la millor solució a nivell industrial tant en seguretat com en programari que oferisca eficiència i disponibilitat de serveis. Seguidament, després de l'anàlisi de la solució òptima a adoptar, es farà un estudi de que distribució de FIWARE és la més adequada i que paper li correspondrà en la tesi. En aquesta anàlisi de l'eina, es posarà l'accent en les característiques que ens ofereix FIWARE en la seua securització, tant en el sistema de gestió d'accessos i identitats com en la de les connexions. No sols es farà l'estudi previ d'aquesta eina, sinó també de tots els components com les plataformes d'administració de càrregues de treball i serveis com Kubernetes o els protocols de connexió dels dispositius industrials generadors de dades. Posteriorment, amb les tecnologíes ja definides en la part inicial del treball, es durà a terme de manera pràctica el desplegament de la infraestructura, on es prioritzarà la securització del primer nivell de la jerarquía, el node Edge.

Aquest entorn aplicarà processos d'anàlisis i processament de dades sobre informació fictícia de l'entorn industrial. En aquest cas, s'utilitzaran dades fictícies extretes de variadors de freqüència mitjançant el protocol MODBUS, que permetrà simular el funcionament d'una planta industrial de la manera més realista possible. Es crearà un entorn visual que permeta la detecció d'errors en la planta de forma centralitzada en Cloud. Finalment s'analitzaran els resultats obtinguts i es plantejaran línies futures de desenvolupament de la tesi.

Abstract

This Master Thesis presents a solution for securing an Edge node motivated by the emerging framework for acquisition, processing and analysis in Internet of Things systems called Edge-Cloud computing. Edge-Cloud computing consists of three layers where we have the first layer called Edge, the second layer called fog computing and finally cloud computing.

This paper focuses on the first layer of this hierarchical system, i.e. edge computing, which offers in data analysis and processing, improved response times and higher bandwidth availability. First, a state-of-the-art analysis of the technologies used in the Edge-Cloud computing paradigm will be carried out. In this preliminary analysis, we will study which is the best solution at industrial level, both in terms of security and software that offers efficiency and availability of services. Then, after the analysis of the optimal solution to be adopted, a study will be made of which FIWARE distribution is the most suitable and what role it will play in the thesis. In this analysis of the tool, emphasis will be placed on the characteristics offered by FIWARE in its securisation, both in the access and identity management system and in that of the connections. Not only will a preliminary study of this tool be carried out, but also of all the components such as the workload management platforms and services such as Kubernetes or the connection protocols of the industrial data-generating devices. Subsequently, with the technologies already defined in the initial part of the work, the deployment of the infrastructure will be carried out in a practical way, where priority will be given to the securitisation of the first level of the hierarchy, the Edge node.

This environment will apply data analysis and processing processes on fictitious information from the industrial environment. In this case, fictitious data extracted from frequency inverters using the MODBUS protocol will be used, which will allow the operation of an industrial plant to be simulated as realistically as possible. A visual environment will be created that will allow the detection of errors in the plant in a centralised way in the Cloud. Finally, the results obtained will be analysed and future lines of development of the thesis will be proposed.

Índice general

Índice de figuras

Acrónimos

AMQP Advanced Message Queuing Protocol.

API Application Programming Interface.

BSON Binary JSON.

CIDR Classless Inter-Domain Routing.

CNI Container Network Interface.

CSR Certificate Signing Request.

DBMS Database Management System.

IoT Internet of Things.

IP Internet Protocol.

MQTT MQ Telemetry Transport.

OPC UA OPC Unified Architecture.

RBAC Role-based Access Control.

RTU Remote Terminal Unit.

SO Sistema Operativo.

TCP Transmission Control Protocol.

VPN Virtual private network.

Capítulo 1

Introducción

1.1. Motivación

Actualmente, existen un gran número de aplicaciones industriales que se resuelven mediante el uso de variadores de frecuencia. Estos se pueden definir como equipos industriales con variables tamaños y prestaciones que permiten la regulación de la velocidad de motores eléctricos que se encuentran conectados a diversos actuadores de todo tipo, así como compresores o bombas. En una planta industrial, el número de variadores puede llegar a ser de entre unas decenas a un millar.

En la industria 4.0, los procesos de producción se encuentran optimizados por el uso de IoT así como métodos de inteligencia artificial entre otras tecnologías.

Estas tecnologías ofrecen la oportunidad de la monitorización de estos procesos y de aportar seguridad mediante el aviso de cualquier avería que se haya podido producir sobre el equipo, como por ejemplo una parada de emergencia, lo que puede llevar a un impacto significativo para la planta a nivel económico.

El creciente uso del paradigma de computación distribuida llamado Edge-Cloud computing, el cual ofrece los beneficios que aporta la computación y almacenamiento de los datos pero en una ubicación más cercana de donde se generan, aporta la escalabilidad necesaria en entornos industriales debido al gran crecimiento de cantidad de datos y la necesidad de bajas latencias al utilizar actuadores.

Durante los últimos años las plataformas de edge industrial se han visto obstaculizadas por limitaciones en el escalado de las aplicaciones, así como en la administración de estas. *Kubernetes* ha surgido como solución, ya que permite un enfoque automatizado para escalar y administrar grandes cantidades de aplicaciones en contenedores en toda la infraestructura distribuida.

La seguridad en estos entornos adquiere gran importancia tanto por el uso de información crítica relacionada con el nivel de producción de la planta como por el acceso por medio de buses industriales que puede llegar a suponer el control de los variadores de forma remota. Los crecientes números de ataques realizados mundialmente hacen de la securización crucial para evitar la salida de datos sensibles al exterior de la plantas industriales y así impedir que se puedan llevar a cabo la inyección de comandos externos que supongan pérdidas económicas y de reputación o daños de cualquier especie. No solo la seguridad a nivel de red o de aplicación es crucial, la privacidad

de los datos de los usuarios con acceso a los sistemas contrae un papel importante para evitar posibles brechas de seguridad mediante ingeniería social. La aparición de *Cilium* aporta un enfoque en la seguridad de los servicios mediante el uso de *Kubernetes*, que no existía anteriormente al no basarse en tablas IP sinó en etiquetas. Su implantación en la industria manufacturera, por lo que reflejan las empresas que lo usan según la web de cilium, es aparentemente nula. En consecuencia, el objetivo de este proyecto es la creación de una arquitectura que sea autogestionada, escalable y segura para la industria con software innovador.

El proyecto está enfocado profesionalmente, ya que se han aplicado conocimientos previamente adquiridos en los proyectos VARIO-EDGE de nivel autonómico, y ASSIST-IoT de nivel europeo, proyectos desarrollados por el grupo de Sistemas y Aplicaciones de Tiempo Real Distribuido (SATRD) de la Universidad Politécnica de Valencia.

1.2. Objetivos

El objetivo de este trabajo fin de máster es la securización a diferentes niveles de un nodo Edge y de la transmisión de información tanto al propio dispositivo como la comunicación con el servidor Cloud, en el que también se incluye como subobjetivo la mejora de las arquitecturas actuales con tecnologías emergentes que permitan avanzar en la securización. El entorno industrial con el uso de variadores de frecuencia utilizados en la industria es un caso práctico y de uso actual en la industria 4.0, pero la securización del nodo Edge puede ser utilizado en cualquier entorno industrial diferente al propuesto en el proyecto. El objetivo principal puede ser desglosado en los diferentes subobjetivos:

- Diseño de la arquitectura de red y aplicación de las tecnologías emergentes que permiten su desarrollo.
- Instalación de una distribución del orquestador de contenedores (Kubernetes) liviana que sea soportada por el node Edge industrial utilizado.
- Estudio de la distribución del broker IoT a instalar y su posterior instalación.
- Securización de la lectura de los datos por medio de las soluciones que se encuentran presentes en el mercado con el software contenerizado.
- Securización de los datos de entorno en el despliegue de contenedores.
- Despliegue del sistema de autenticación en el nodo edge para el acceso a sus servicios
- Monitorización de los logs seleccionados y posterior visualización de estos.
- Instalación de un plugin de red con el fin de controlar las comunicaciones de red que se producen.
- Securizar las conexiones de red entre edge y cloud.
- Securización de las conexiones con el broker IoT mediante el uso de un proxy y de un sistema de autenticación.
- Desarrollo del dashboard encargado de la visualización de los errores en los variadores en funcionamiento.

Capítulo 2

Estado del arte

2.1. Hardware y software industrial

2.1.1. Variadores de frecuencia

Actualmente en la industria, los motores eléctricos se encargan del control de todo aquello que se necesita en las cadenas de producción. El variador de frecuencia se encarga de la regulación de la velocidad con el fin de que la electricidad sea ajustada adecuadamente a lo que la aplicación industrial demanda realmente.

Los variadores de frecuencia se usan con el fin de reducir la potencia de salida en una aplicación industrial como puede ser compresores o bombas, permitiendo así que estos funcionen a una velocidad exacta para su correcto funcionamiento. Podemos ver un ejemplo de variador de frecuencia en la figura 2.1.

Figura 2.1: Variador de frecuencia

El uso de estos equipos aportan ventajas como una productividad mejorada, incremento de la eficiencia energética, así como alargar la vida útil de los dispositivos conectados ya que se previene el deterioro y evita paradas que pueden llevar a pérdidas económicas [1].

El variador utilizado se comunica mediante el protocolo Modbus en el que se establece una comunicación entre cliente y servidor en los dispositivos. Los dos más utilizados actualmente son Modbus TCP/IP y el RTU. El protocolo Modbus RTU es el usado en las comunicaciones entre el variador y el nodo Edge en el proyecto.

2.1.2. Modbus RTU

Se puede definir Modbus RTU como un protocolo de comunicaciones el cual se basa en la arquitectura maestro/esclavo en la que puede existir de uno a varios esclavos que responden al maestro, encargado de la realización de las peticiones y único dispositivo con la posibilidad de iniciar la comunicación. Así pues, se puede decir que los esclavos son servidores y el maestro el cliente. En el caso de Modbus RTU la información dentro de la trama se encuentra codificada en binario. El formato de trama usado por Modbus RTU es el siguiente [2]:

- **Campo de dirección:** Este campo es usado para indicar en que dirección se encuentra el esclavo al que se le envía la petición, donde las direcciones varían de 0 a 247. El esclavo en la respuesta utiliza esta dirección para que el maestro pueda saber de donde proviene la respuesta.
- **Código de función:** Es el campo encargado de indicar cual es el código de operación que se le ha sido solicitado al cliente por parte del maestro, por ejemplo, leer un registro determinado. En el proyecto utilizamos el registro 3029 como ejemplo.
- **Campo de datos:** Contiene cual es la información necesaria para llevar a cabo una función
- **CRC:** Campo con el fin de verificar que la información entrante no contiene errores.

La figura 2.2 representa la trama Modbus RTU.

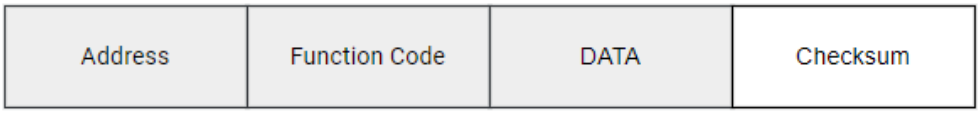

Figura 2.2: Trama Modbus RTU

Finalmente, la interfaz utilizada para trabajar con el protocolo Modbus RTU es el puerto serie RS-485.

Modbus RTU no dispone de seguridad como protocolo, es por ello que la securización del nodo edge es más importante aún.

2.1.3. Simatic IoT 2050

SIMATIC IOT2050 es una plataforma abierta y fiable para la recopilación, el procesamiento y la transmisión de datos en tiempo real en entornos de producción usada como nodo Edge. Es ideal para servir como puerta de enlace entre la nube o la empresa y la capa de TI de producción. La naturaleza abierta del sistema, que le permite soportar varios protocolos de comunicación, y la capacidad de programar en lenguajes de alto nivel, permite la creación de soluciones personalizadas [3]. La figura 2.3 muestra el dispositivo Simatic IOT2050.

Figura 2.3: Simatic IoT2050

La Simatic IoT 2050 ofrece:

- Diversas capacidades de programación en lenguajes de alto nivel.
- Implementación de soluciones de comunicación flexibles con diferentes protocolos, desde Modbus RTU y OPC UA hasta protocolos en la nube como MQTT/AMQP.
- Ejemplos de bibliotecas y aplicaciones de código abierto.

2.1.4. Node-Red

Node-RED es una herramienta de programación para conectar dispositivos, API y servicios en línea.

Node-RED proporciona un editor de secuencias basado en el navegador que facilita la unión de flujos utilizando múltiples nodos instalados en la paleta. Los flujos se pueden implementar en tiempo de ejecución con un sólo clic [4].

2.2. Virtualización

En esta sección, se incluye información relevante en el marco de desarrollo y despliegue de los componentes.

La gran ventaja de la virtualización reside en la exportación de software a cualquier elemento de procesamiento genérico, existiendo varias tendencias: máquinas virtuales, contenedores y unikernels. Las máquinas virtuales se instalan en un elemento de procesamiento gestionado por un hipervisor (p.ej., VMWare , Proxmox) o encima de un Sistema Operativo (SO), y es la opción más pesada ya que disponen de su propio SO; los contenedores, por otra parte, se instalan encima de un SO, y por el contrario, no requieren de uno propio debido a que virtualiza el subyacente, por lo que son más ligeros. Por último, los unikernels son como máquinas virtuales, pero con las mínimas funcionalidades de kernel requeridas para ejecutar la función (o funciones) que aloja, no con todo un SO. Son tan ligeros como los contenedores, más seguros que estos al no compartir SO subyacente con otros unikernels, pero más difíciles de diseñar y desarrollar, al tener que "reducir" el SO a las necesidades de la funcionalidad que implementa.

De entre las opciones disponibles, se define la virtualización basada en contenedores como el paradigma de diseño, desarrollo (o adaptación) y ejecución de los componentes funcionales de la arquitectura. De esta manera, cada componente se implementa de forma independiente, pudiendo ejecutarse todos ellos sobre un único SO. La tecnología concreta que se utiliza para el desarrollo de los componentes es Docker. Las imágenes de Docker (o Dockerfiles) son archivos que representan una aplicación con las librerías o dependencias que necesita para funcionar, así como su configuración por defecto y ficheros de propiedades. Dichas imágenes, una vez ejecutadas sobre un SO compatible con el motor de Docker, se denominan contenedores, que tienen las siguientes ventajas:

- Fácil configuración, ya que el usuario apenas debe configurar opciones antes de ser desplegado, siendo las opciones por defecto generalmente suficientes.
- Rápido despliegue y ligereza, al no necesitar contener un SO propio, pudiendo compartir el subyacente con otros.
- Portabilidad, ya que incluyen todas las dependencias y librerías necesarias para su funcionamiento, además de que Docker puede emplearse sobre múltiples SOs.
- Inmutabilidad, dado que el mismo código (librerías, funciones) se ejecutarán en cualquiera que sea el elemento de procesamiento.
- Soporte, al existir repositorios con imágenes open-source de múltiples tecnologías, así como una gran comunidad.
- Para cada componente, se desarrollará o adaptará un Dockerfile. Para integrar y desplegar los componentes, se recomiendan dos opciones, Docker compose y Kubernetes.

En esta tesis se ha elegido Kubernetes como herramienta de despliegue de los servicios virtualizados debido a su facilidad en la automatización de despliegue de servicios.

2.2.1. Docker

Docker se define como un proyecto de código abierto que se encuentra basado en contenedores de Linux. Su uso proporciona en diversos sistemas operativos una adicional capa de abstracción y automatización de aplicaciones en diferentes sistemas operativos de forma virtualizada. Docker,

como motor de contenedores hace uso de de espacios de nombres para aportar la ventaja de que contenedores independientes puedan ejecutarse dentro de una instancia de Linux, característica que evita una sobrecarga en el inicio y manutención de máquina virtuales.

La arquitectura de docker se basa en una arquitectura cliente/servidor, en la que el cliente se comunica con el Docker ”daemon”, encargado de la creación, distribución y hacer correr los contenedores. Ambos, cliente y servidor pueden encontrarse en un mismo sistema o por el contrario, la conexión de un cliente remoto con el demonio de Docker. La comprensión del funcionamiento de Docker internamente pasa por conocer tres componentes:

- **Docker images:** Las imágenes de Docker son las plantillas sobre las que se crean los contenedores. Las modificaciones realizadas sobre el contenedor lanzado no se realizan sobre la imagen. Un término importante en las imágenes de Docker son los Dockerfiles. Mediante el Dockerfile se crea la descripción de la configuración que se encarga de la creación de una imagen.
- **Docker registries:** Registros de guardado de las imágenes de Docker en repositorios públicos o privados para el uso posterior de las imágenes.
- **Docker containers:** Plataformas aisladas creadas a partir de una imagen de Docker que se componen de todo lo necesario para el correcto funcionamiento de un servicio.

2.2.2. Kubernetes

Kubernetes (abreviado como k8s) es un orquestador de contenedores. Su introducción como tecnología adicional de virtualización responde a la posibilidad de aumentar la robustez del sistema en entornos de producción. Concretamente, la característica más interesante que puede proveer k8s como herramienta de empaquetado y despliegue reside en la posibilidad de realizar actualizaciones y reversiones de componentes sin interrupción de servicio, a parte de la posibilidad de automatización de despligues y monitorización continuaa de la salud de los contenedores. Kubernetes facilita las estrategias de despliegue ya que, para un determinado componente, tiene desacoplados el servicio (punto de entrada a un componente, con IP y puertos asociados) y la carga de trabajo o pods subyacentes. Los componentes de Kubernetes utilizados son los mostrados en la figura 2.4.

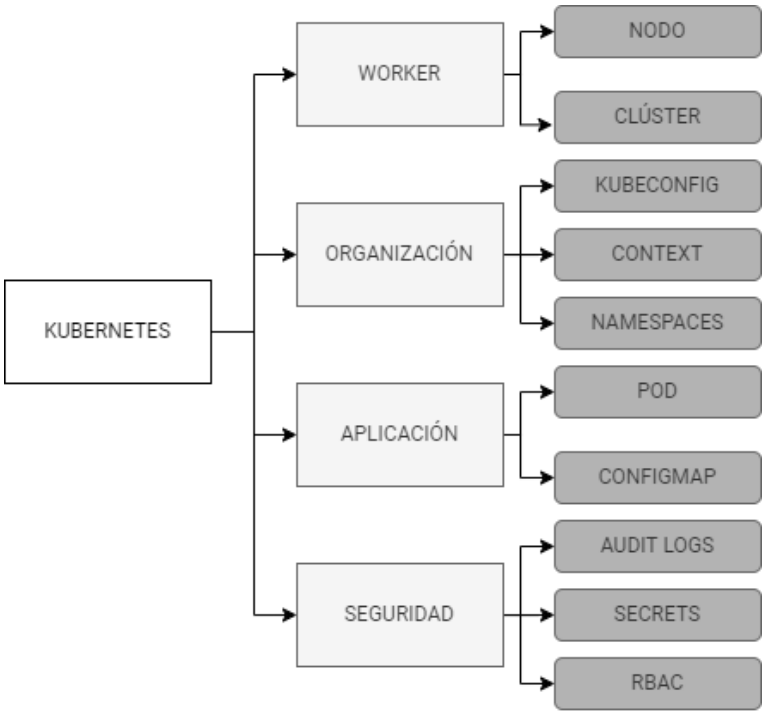

Figura 2.4: Distribución del estado del arte de Kubernetes

2.2.2.1. Nodo

Un nodo se define como una máquina de trabajo de Kubernetes. Un nodo puede ser tanto una máquina virtual como una máquina física [5]. Un componente máster gestiona cada nodo y a parte contiene los servicios necesarios para ejecutar pods.

2.2.2.2. Clúster

Clúster es el conjunto de nodos que ejecutan aplicaciones contenerizadas. Un clúster está formado por un plano de control y un nodo como mínimo.

El plano de control se encarga de que el clúster se mantenga en el estado deseado y de controlar las aplicaciones que se ejecutan y las imágenes de contenedores que se utilizan [6].

2.2.2.3. Kubeconfig

El archivo kubeconfig contiene la información necesaria para que la herramienta *kubectl* pueda conectarse con un clúster y así interactuar con la API de este. Los archivos *Kubeconfig* se utilizan con el fin de organizar toda la información sobre los clústeres, *namespaces* y usuarios.

A pesar de que *kubectl* por defecto busca el archivo llamada config en el directorio $HOME/.kube, también se puede especificar la ruta de este utilizando la variable de entorno KUBECONFIG [7].

2.2.2.4. Context

El *Context* en el archivo *kubeconfig* se usa con el fin de agrupar parámetros de acceso bajo un nombre [7]. Cada contexto se compone de tres componentes:

- Clúster
- Namespace
- Usuario

De esta manera se puede conmutar entre clústeres en diferentes ubicaciones desde una misma máquina.

2.2.2.5. Namespaces

Kubernetes facilita el soporte de varios clústeres virtuales, los cuales están respaldados por el mismo clúster físico. Los distintos clústeres virtuales se denominan espacios de nombres (*namespaces*) [8].

Los *namespaces* dividen los recursos de un clúster, siendo el nombre de los servicios desplegados en un clúster diferentes entre ellos pero no diferentes a los servicios desplegados en otro *namespace* del mismo clúster. En la creación de un clúster de Kubernetes, se crean tres espacios de nombres:

- **Default:** *Namespace* asignado a los servicios que no tienen uno concreto previamente asignado.
- **Kube-system:** Los servicios utilizados por Kubernetes para funcionar se encuentran desplegados en este espacio de nombre.
- **Kube-public:** Espacio de nombres creado de forma automática y puede ser leído por cualquier usuario. Este espacio de nombres se encuentra reservado para el uso interno del clúster, en el caso de que algunos servicios tengan que ser visibles en todo el clúster.

2.2.2.6. Pod

Un pod se define como el grupo de uno o más contenedores que utilizan almacenamiento y red compartidos y los cuales contienen detalles de como ejecutar los contenedores [9].

Los pods son la unidad más pequeña en las aplicaciones de Kubernetes. La representación de un conjunto de pods iguales se llama *Deployment*. Por otra parte un *DaemonSet* garantiza la ejecución de la copia de un Pod en todos los nodos.

2.2.2.7. ConfigMap

El *ConfigMap* es un objeto de Kubernetes que es usado para guardar variables no confidenciales con el formato clave-valor. Se pueden utilizar para pasar variables de entorno a los Pods. De esta forma es posible desacoplar los datos de entorno de la imagen de contenedor [10].

2.2.2.8. Audit Logs

La Auditoría de Kubernetes ofrece un conjunto de registros cronológicos relevantes para la seguridad que documentan la secuencia de acciones en un clúster. El clúster audita las actividades generadas por los usuarios, por las aplicaciones que utilizan la API de Kubernetes y por el propio plano de control [11].

Los Audit Logs permite a los administradores de clústeres responder a las siguientes preguntas:

- ¿Qué sucedió?
- ¿Cuándo ocurrió?
- ¿Quién lo inició?
- ¿Dónde se observó?
- ¿Desde dónde se inició?

Los registros de auditoría comienzan su ciclo de vida dentro del componente kube-apiserver . Cada solicitud en cada etapa de su ejecución genera un evento de auditoría, que luego se procesa previamente de acuerdo con una política determinada y se escribe en un backend. La política determina lo que se registra y los backends conservan los registros. Las implementaciones de back-end actuales incluyen archivos de registro y webhooks.

En el proyecto se ha optado por el uso de archivos de registros que conjuntamente con otras tecnologías ofrece la visualización de estos logs en el Cloud.

2.2.2.9. Secrets

Un *secret* es un objeto que contiene una pequeña cantidad de datos sensibles como una contraseña, un token o una clave. Esta información podría incluirse en una especificación Pod o en una imagen de contenedor. Utilizar un Secreto significa que no necesitas incluir datos confidenciales en el código de tu aplicación [12].

Dado que los Secretos pueden crearse independientemente de los Pods que los utilizan, hay menos riesgo de que el Secreto (y sus datos) queden expuestos durante el flujo de trabajo de creación, visualización y edición de Pods.

2.2.2.10. RBAC Authorization

El control de acceso basado en roles (RBAC) es un método encargado de regular el acceso a recursos informáticos o recursos de red según el rol de un usuario individual en una organización. La API de RBAC contiene cuatro tipos de objetos de Kubernetes: *Role, clústerRole, RoleBinding* y *clústerRoleBinding*.

- **Role**: Establece permisos dentro de un *namespace* particular.
- **clústerRole**: Establece permisos sin tener en cuenta los *namespaces*.

- **RoleBinding**: Otorga permisos definidos en un rol a un usuario o conjunto de usuarios.
- **clústerRoleBinding**: Otorga permisos definidos en un *clústerRole* a un usuario o conjunto de usuarios.

2.2.2.11. Orquestador de clústeres

El orquestador de clústeres tiene como objetivo implementar, monitorear y orquestar los recursos instanciados en cada uno de los clústeres de Kubernetes que se le agregan. Para cumplir con estos objetivos, el servicio utiliza 4 tecnologías diferentes, tales como: API REST, Prometheus, Mongo DB, mck8s y OSM. El orquestador de clústeres presenta las siguientes características principales:

Inteligencia de decisión: Proporciona inteligencia de decisión de Kubernetes accediendo a los servidores de métricas en los otros clústeres unidos para instanciar los servicios en el clúster seleccionado.

Control del ciclo de vida: El orquestador inteligente permite controlar el ciclo de vida de los habilitadores desde su implementación hasta su eliminación.

Ahorro de energía: Ahorra energía al ejecutar un trabajo cuando se requiere en lugar de tener un componente funcionando permanentemente sólo para una llamada API.

2.2.2.12. Distribuciones de Kubernetes

Existen múltiples distribuciones de Kubernetes: distribución principal de kubeadm, k0s, MicroK8s, Openshift, k3s etc. Para escoger entre las distintas distribuciones, hay que analizar las ventajas e inconvenientes de cada una de ellas (licencia, dificultad de despliegue, rendimiento, etc.). Para la ejecución de esta tesis, se valora:

- Que la licencia sea abierta y open-source.
- Que pueda instalarse en dispositivos con bajos recursos (p.ej., 1 GB RAM).

Tras valorar entre las múltiples opciones, se ha determinado que k3s es la opción más recomendable, para el nodo edge, siendo la distribución para cloud kubeadm como la elegida.

Una herramienta interesante es Helm, que permite (i) empaquetar todos los manifiestos de k8s de un determinado componente (o del sistema) en un único fichero comprimido, (ii) gestionar sus respectivos despliegues y (iii) manejar posibles actualizaciones y reversiones. Además, del mismo modo que existen repositorios públicos de imágenes de Docker, como DockerHub , existen repositorios abiertos de Helm, como Artifacthub.io .

En la figura 2.5 se muestra es el esquema de Kubernetes.

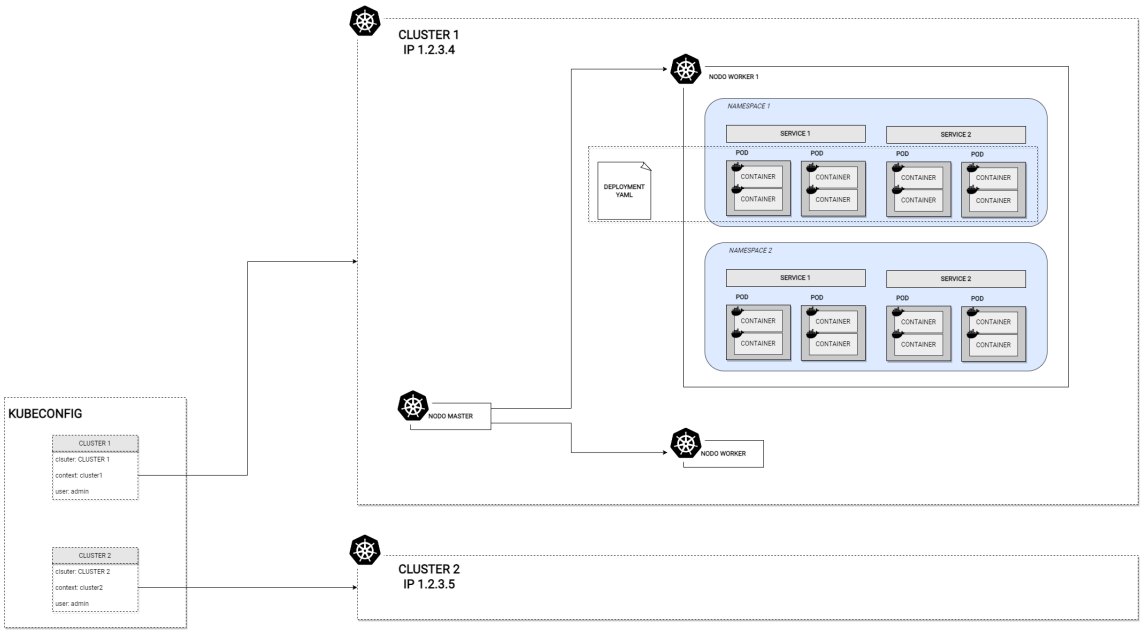

Figura 2.5: Esquema de Kubernetes

2.3. Container Network Interface

Kubernetes proporciona un plugin para la conectividad de red llamado Container Network Interface (CNI), que esencialmente construye un espacio de direcciones de red plano donde cada nodo posee un segmento de red individual (un bloque CIDR).Un plugin permite que cada pod que se ejecuta en un nodo físico tenga una dirección IP virtual única a nivel mundial y así hacer obsoleta la necesidad de traducción de direcciones de red para comunicaciones pod-to-pod.

Los CNI contemporáneos más populares incluyen Flannel, Calico y Cilium. Sin embargo en este proyecto se hace uso de Cilium por su posibilidad de conexiones entre clústeres y por tanto las políticas de red capaces de realizar entre estos.

2.3.1. Cilium

Cilium es un software de código abierto para asegurar de forma transparente la conectividad de red entre los servicios de aplicaciones desplegados mediante plataformas de gestión de contenedores de Linux como Docker y Kubernetes.

La base de Cilium es una nueva tecnología del núcleo de Linux llamada eBPF, que permite la inserción dinámica de una potente visibilidad de la seguridad y una lógica de control dentro del propio Linux. Dado que eBPF se ejecuta dentro del kernel de Linux, las políticas de seguridad de Cilium pueden aplicarse y actualizarse sin ningún cambio en el código de la aplicación o en la configuración del contenedor [13].

Las características principales de Cilium son:

- **Securizar APIs de forma transparente:** Capacidad para asegurar protocolos de aplicación modernos como REST/HTTP, gRPC y Kafka.
- **Comunicación segura de servicio a servicio basada en identidades:** Cilium asigna una identidad de seguridad a grupos de contenedores de aplicaciones que comparten políticas de seguridad idénticas. La identidad se asocia entonces a todos los paquetes de red emitidos por los contenedores de aplicaciones, lo que permite validar la identidad en el nodo receptor. De esta manera queda intacta la escalabilidad del sistema.
- **Acceso seguro a y desde servicios externos** La seguridad basada en etiquetas es la herramienta elegida para el control de acceso interno del clúster. Para asegurar el acceso a y desde los servicios externos, se admiten las políticas de seguridad tradicionales basadas en CIDR tanto para la entrada como para la salida. Esto permite limitar el acceso a y desde los contenedores de aplicaciones a determinados rangos de IP.
- **Balanceo de carga y simple Networking**
- **clústermesh**: Amplía la ruta de datos de red a través de múltiples clústeres. Permite que los puntos finales de todos los clústeres conectados se comuniquen al mismo tiempo que proporciona una aplicación completa de las políticas de red. El equilibrio de carga está disponible a través de las anotaciones de Kubernetes.

2.4. Plataforma IoT

Una plataforma IoT es el software que permite la conexión de sensores, actuadores y dispositivos centralizando la información y las comunicaciones con el fin de generar un valor adicional en un entorno digital.

Las plataformas IoT permiten que se produzca el intercambio de datos de forma segura a parte de la interpretación y procesamiento de estos para ser aplicados en procesos de producción. Los principales elementos en una plataforma IoT son: hardware de dispositivos y sensores, el software encargado de hacer un análisis de la información, conectividad en la red para hacer de manera sencilla la transmisión de datos y órdenes desde el hardware al cloud y la interfaz de usuario para facilitar las interacciones entre usuarios y sistemas IoT.

Actualmente el número de plataformas IoT asciende a un número muy elevado, por lo tanto, se hará hincapié en la plataforma FIWARE desplegada en el proyecto.

2.4.1. FIWARE

FIWARE es una plataforma Open source, respaldada por la Unión Europea en el programa Future Internet Public Private Partnership. FIWARE ofrece una arquitectura abierta conjuntamente con especificaciones que permite a desarrolladores, empresas y organizaciones elaborar su ecosistema IoT.

El acceso a la información se realiza mediante la API NGSI, vía por la cual se hace sencillo el procesamiento de cantidades grandes de datos. FIWARE permite mediante su gestión de datos, generar información de contexto, los cambios en el contexto haciendo uso de eventos y el procesamiento de cantidades grandes de información con técnicas de Big Data.

Los componentes principales de FIWARE utilizados en esta tesis son el broker IoT Orion, el Proxy Wilma y el gestor de identidades Keyrock.

2.4.1.1. FIWARE Orion Context Broker

FIWARE Orion Context Broker se encarga de la gestión de las comunicaciones entre dispositivos y aplicaciones con el fin de desacoplar las aplicaciones IoT de los dispositivos. La comunicación ente dispositivos y aplicaciones se realiza mediante el uso del protocolo NGSI, permitiendo la creación de eventos y la posterior notificación de su creación.

Para el almacenamiento de datos se hace uso de las entidades. Las entidades están identificadas por un id único y que pertenece a un grupo llamada tipo. Por otra parte, cada entidad puede contener ciertos atributos que describen la entidad.Los datos son almacenados en una base de datos NoSQL llamada MondoDB. Es importante destacar que únicamente se almacena el último valor de cada atributo en una entidad [14].

Cabe destacar dos grupos con diferentes roles, los productores y los consumidores. Los productores generar datos y actualizan los datos de las entidades y por otra parte se encuentran los consumidores que realizan la subscripción a Orion para ser notificados de las actualizaciones de las entidades a los que se encuentren suscritos. En la figura 2.6 se muestra un esquema de los diferentes roles en Orion Context Broker.

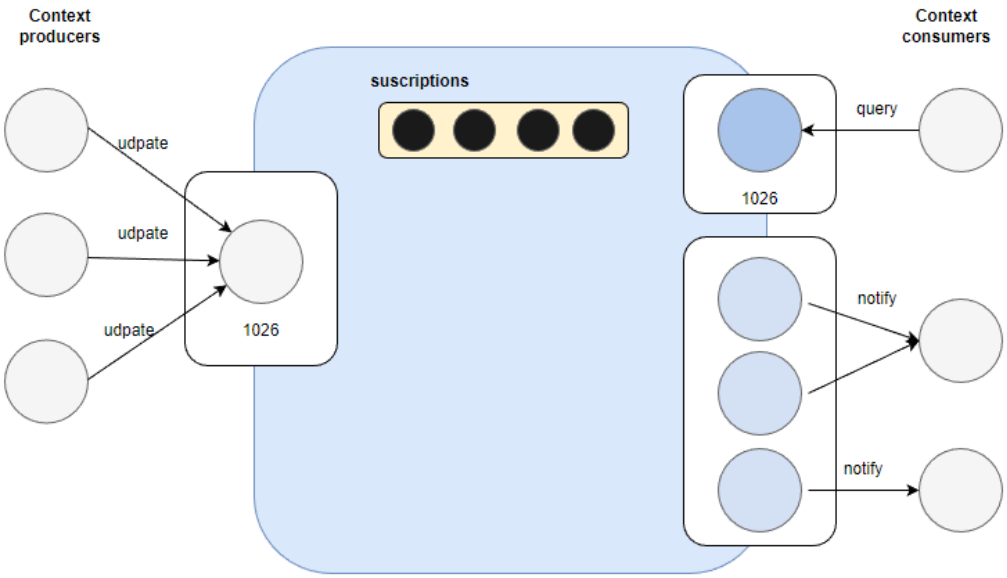

Figura 2.6: Roles en Orion Context Broker[14]

2.4.1.2. FIWARE PEP proxy

El PEP proxy es un componente del ecosistema FIWARE que aporta seguridad de autenticación y autorización a sus aplicaciones backend mediante el uso de peticiones al componente de gestión de identidades KeyRock. De esta forma, sólo los usuarios de FIWARE que tengan acceso podrán acceder al elemento que se encuentre securizado por el Proxy, como puede ser un agente http, orion o una aplicación web.[15]

El PEP proxy permite por otra parte la gestión de permisos y políticas específicas para sus recursos permitiendo diferentes niveles de acceso a sus usuarios.

2.4.1.3. Keyrock

Keyrock es el componente de FIWARE responsable de la gestión de identidades. El uso de Keyrock le permite conjuntamente con otros componentes como el nombrado PEP proxy, añadir seguridad de autenticación y autorización basada en OAuth2 a sus servicios y aplicaciones.[16]

Keyrock devuelve un token para acceder a los recursos sobre los que se tiene autorización como organización o como individuo. Los conceptos principales en el gestor de identidades Keyrock son:

- **Usuario**: Elemento con capacidad para la identificación con el sistema.
- **Rol en la organización**: Rol asignado sobre una organización con diferentes usuarios.
- **Organización**: Grupo de usuarios

- **Rol**: Conjunto de permisos atribuibles a una organización o usuario
- **Permiso**: Poder para consultar un recurso.
- **Aplicación**: Servicio securizado.

2.4.1.4. Agente IoT

Los agentes IoT son los componentes traductores de protocolos específicos IoT al protocolo de información NGSI, que es el modelo de intercambio de datos estándar de FIWARE. Este componente no es necesario si los dispositivos generadores de información lo usan de forma nativa.

Los agentes IoT pueden traducir diferentes formatos de mensaje a NGSI como por ejemplo JSON, LoRaWan o MQTT y incluso su programación con la librería PyNgsi para Python y traducir datos de archivos por ftp o excel.

2.5. Bases de datos

2.5.1. MongoDB

MongoDB es una base de datos OpenSource NoSQL, orientada a documentos. El almacenaje de información es realizado haciendo uso del formato BSON (Binary JSON), permitiendo así que la estructura de datos pueda cambiar en el tiempo y entre documentos [17].

La misma base de datos puede contener colecciones distintas, encargadas del almacenaje de documentos, donde cada documento dispone de su id interno, creado automáticamente en la creación del documento.

2.5.2. ElasticSearch

Elasticsearch es un motor de análisis y búsqueda distribuido, gratuito y de código abierto para todo tipo de datos, incluidos datos de texto, numéricos, con estructura y sin estructura. Elasticsearch se basa en Apache Lucene y se presentó en 2010 por Elasticsearch N.V [18].

Conocido por sus simples API REST, sistema distribuido, velocidad y escalabilidad, Elasticsearch es el componente central de Elastic Stack, un conjunto de herramientas gratuito y de código abierto para capturar, enriquecer, almacenar, analizar y visualizar datos. Conocido comúnmente como ELK Stack (para Elasticsearch, Logstash y Kibana (Dashboard)), Elastic Stack actualmente consta de un gran grupo de agentes que se llaman Beats, para enviar datos a Elasticsearch.

En la realización de este proyecto se hace uso del ELK Stack para la visualización de los Audit Logs.

2.5.3. MySQL

MySQL es un sistema de gestión de bases de datos (DBMS, Database Management System).Para agregar, acceder o procesar datos se requiere un sistema de control para manejarlos. Es por ello que surgió MySQL, con el propósito de actuar como un DBMS. Es un DBMS relacional [19].

Una base de datos relacional consiste en un conjunto de datos almacenados en diferentes tablas relacionadas.

Capítulo 3

Arquitectura inicial

En este capítulo, se aborda la arquitectura inicial la cual va a ser securizada posteriormente. Primero se describe la arquitectura física del proyecto, es decir, la estructura hardware utilizada en el prototipo. Seguidamente, se definen los componentes y sus flujos de ejecución en la arquitectura funcional, es decir, la lógica del trabajo, haciendo distinción entre cloud y edge.

El entorno de esta arquitectura se halla en una industria manufacturera en la que hay dos líneas de producción separadas. El ejemplo utilizado es una línea de bebida gaseosa y en la otra línea de producción la misma bebida pero sin azúcar añadido. El objetivo es que la información de errores en cada una de las fábricas divididas geográficamente se envíen a un servidor Cloud donde se monitorizan estos errores.

La arquitectura es una arquitectura reutilizable en cualquier ámbito, la diferencia radica en los procesamientos que se producen en el nodo edge y en Cloud. Esta arquitectura pretende ser desde su inicio una arquitectura segura a la hora de desplegar los servicios y de hacerlo en los nodos edge que pertocan y que se encuentran registrados en Cloud para evitar intrusiones de errores falsos.

3.1. Arquitectura física

La arquitectura física muestra la topología hardware en la que consiste el proyecto.

Esta arquitectura física está basada en el paradigma edge-cloud. Este paradigma aporta ciertas ventajas sobre aquel únicamente basado en cloud, como puede ser la ejecución de procesos de altos requisitos de latencia o en aquellas situaciones en las que sea necesario procesar datos de forma cercana geográficamente a donde se generan, para poder así reducir los tiempos de respuesta.

La sobrecarga de red se reduce al poder almacenar datos localmente, a la par que se produce un aumento de la seguridad. Mantener los datos en equipos del entorno industrial disminuye la probabilidad de ser objeto de ciberataques.

El paradigma basado únicamente en edge puede ser suficiente en algunos casos, pero en el caso que se tengan que ejecutar análisis de todo un entorno o de múltiples entornos industriales, se puede requerir de servidores de cierta potencia que en muchas ocasiones no se encuentran en las propias instalaciones de la empresa.

La vista física que se presenta para este proyecto se resumen en la **figura 3.1**. Consiste en una arquitectura edge-cloud, en la que los nodos edge están conectados a un determinado número de variadores de frecuencia, aunque en este proyecto únicamente se dispone de uno de ellos para las pruebas pertinentes. A su vez se conectarán a un servidor en la nube que no se encuentra fuera de planta de producción o empresa.

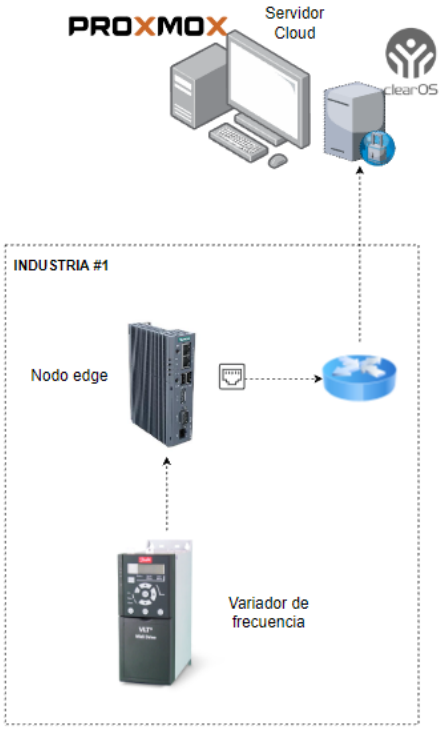

Figura 3.1: Arquitectura física

Por las características del variador de frecuencia obtenido, las comunicaciones se realizan mediante MODBUS RTU, protocolo principal que debe soportarse para las comunicaciones ente los variadores y los nodos edge. Para ello, el nodo edge debe tener disponible interfaces para la comunicación con los variadores, como es el puerto serie RS-485, interfaz habitual para trabajar con los protocolos Modbus RTU.

El hardware elegido es la gateway Simatic IoT 2050, que dispone de hardware y software necesario para poder introducir esta arquitectura en un entorno de producción.

El entorno seleccionado para el despliegue del servidor Cloud, donde se requiere mayor potencia computacional, es un máquina virtual en los servidores del SATRD.

La comunicación predeterminada entre el nodo edge y el servidor Cloud estará basada en TCP/IP, sobre túneles privados (VPN) sobre internet, ya que no se dispone de una red WAN privada.

3.2. Arquitectura funcional

La vista funcional tiene como objetivo describir las funcionalidades del sistema necesarias para abordar el proyecto. Esta vista muestra los componentes funcionales principales, sus responsabilidades e interacciones primarias. Esta sección detalla el carácter funcional de cada servicio desplegado y el flujo de ejecución diferenciado entre el edge y el cloud, así como la vista general del flujo de ambos componentes por separado.

3.2.1. Edge

El nodo edge está compuesto por cuatro componentes principales: Kubernetes, Node-red, API y la base de datos. Cada uno de estos componentes ofrecen una funcionalidad principal que es descrita a continuación:

- **Kubernetes:** La distribución elegida ha sido K3s, debido a sus características que lo hacen idóneo para dispositivos con bajos recursos, a parte de la licencia open-source que ofrece. En la instalación de k3s, es necesario utilizar este comando para que puedan funcionar todos los servicios [20].

```
curl -sfL https://get.k3s.io | INSTALL_K3S_VERSION=1.23.6+k3s1
    INSTALL_K3S_EXEC="server --kubelet-arg="feature-gates=DevicePlugins=
    true"" sh -s - --docker
```

  El uso de K3s permite instanciar los servicios en el hardware elegido de forma virtualizada y con la posibilidad de orquestar los recursos instanciados mediante la monitorización continua de la salud de los contenedores. Con el fin de separar las dos líneas de producción se ha decidido crear dos *namespaces*. Ambos contienen los mismos componentes pero con información distinta ya que se tratan de diferentes zonas de la empresa. Los servicios en un mismo *namespace* se encuentran aislados de los servicios del otro *namespace*, únicamente son visibles si se indica en el servicio al que se quiere conectar el *namespaces* en el que se encuentra.

- **Node-red:** Este componente es un orquestador de flujos de eventos. El orquestador permite recoger datos, procesarlos y transmitirlos. La ventaja de Node-Red es su facilidad de uso mediante una interfaz gráfica web que permite la creación de flujos de forma visual conectando cada uno de los módulos disponibles y seleccionados de la paleta.

  En la arquitectura funcional propuesta se ha hecho uso de tres nodos, tal y como muestra la figura 3.2:

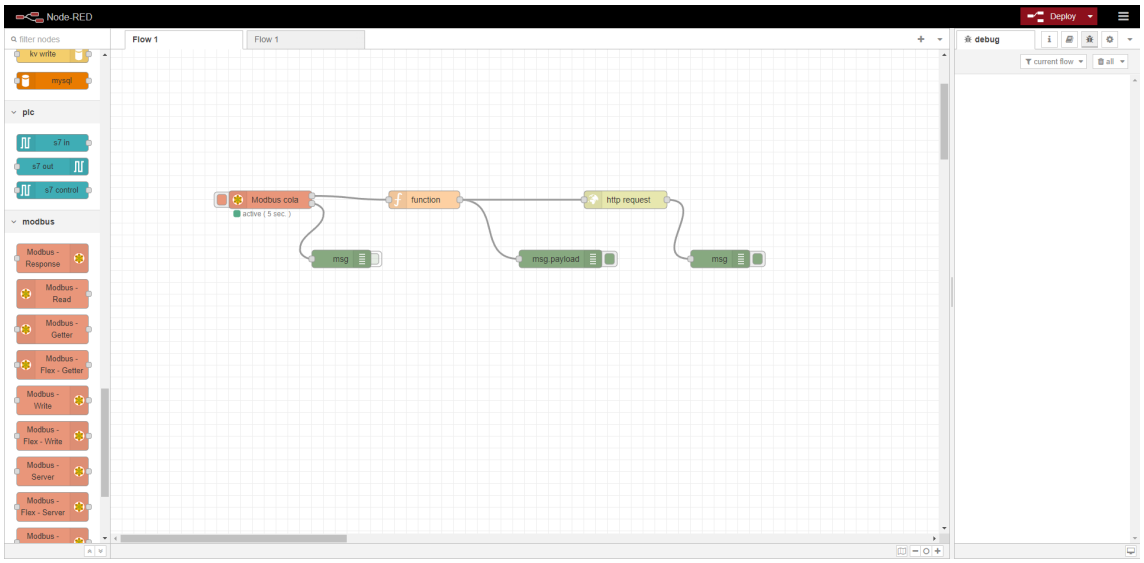

Figura 3.2: Flujo de Node-red

- **Modbus-Reader:** Nodo encargado de la lectura del puerto RS-485, mapeado a la interfaz ttyUSB0 de la gateway Simatic IoT 2050. La configuración del nodo de lectura está compuesto por la configuración del servidor, ajustado a los parámetros del variador de frecuencia y por la dirección y cantidad de bytes que se quieren leer. Cada una de las direcciones dispone de un parámetro distinto, en esta ocasión se utiliza la dirección 3029 que hace referencia a la máxima frecuencia a la que puede trabajar el equipo en cuestión.

  La generación de errores en un variador depende de ciertas variables que no se encuentran en el alcance de este proyecto, es por ello que no se utiliza una variable, sino una constante. En consecuencia, en el prototipo se programa una función que aleatoriza el valor de esta constante en el momento el nodo lector recibe un dato, para poder así generar errores.

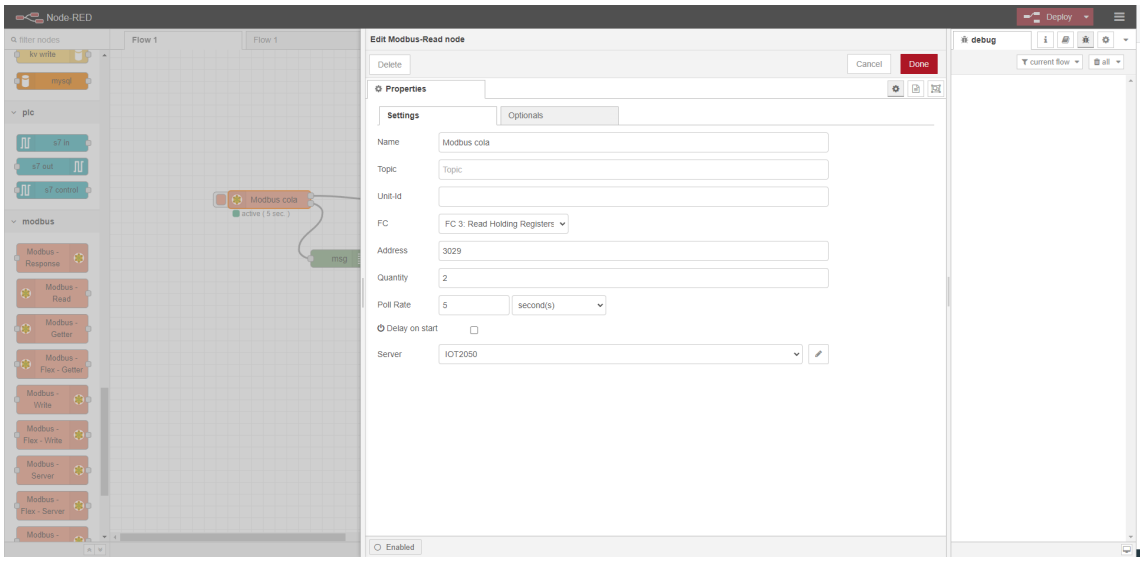

Figura 3.3: Ventana de edición de un Nodo Modbus-Read.

- **Nodo función:** Nodo responsable de aleatorizar el valor de la dirección 3029 para convertir esta constante en una variable y permitir así la generación de errores.

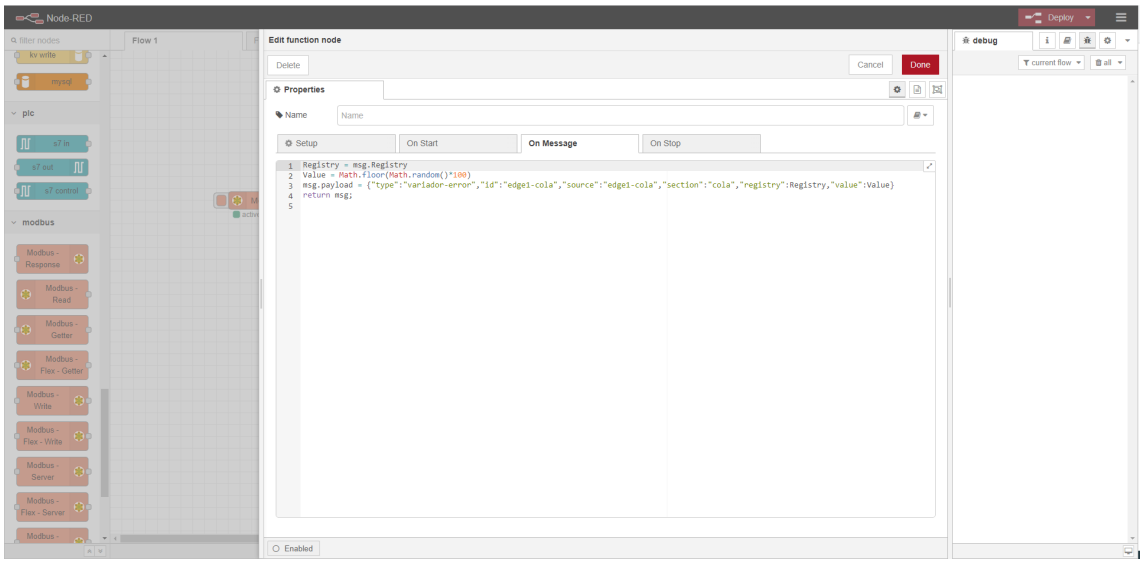

Figura 3.4: Ventana de edición de un Nodo Función.

- **Nodo httpRequest:** Llamada al servicio API del nodo Edge con el objeto de enviar la información recibida y que sea procesada como dato informativo o como un error.

- **API**: Una API se define como un conjuntos de definiciones y protocolos que son usados para diseñar e integrar el software de las aplicaciones. Permite la comunicación entre los componentes, así como la interacción del usuario con las funcionalidad subyacentes. La API diseñada se encarga de tres principales funciones:

- Procesamiento de la información obtenida por node-red.
- Posterior guardado en la base de datos de la información procesada.
- Envío de la información procesada a Cloud si de un error en el variador se trata.
- Comprobación de los registros de la base de datos.

La lógica programada es sencilla, si el valor de la dirección 3029 está por debajo de 50 se considera un error, se guarda en la base de datos local y se envía a Cloud para ser notificado. Si por el contrario se encuentra por encima del valor 50, no es considerado error, por lo tanto únicamente se guarda en la base de datos. Las llamadas a la API son dos:

- **GET /api**: Llamada a la base de datos para mostrarlos como respuesta a la llamada.
- **POST /api**: Procesado de los datos para comprobar si se trata de un error y posterior guardado en la base de datos y envío a Cloud si procede. El envío a Cloud, en la arquitectura inicial, no securizada, se realiza al agente IoT mediante una llamada http al puerto 8880 y dirección */upload*.

- **Base de datos:** Componente responsable de la persistencia de información local. La base es open-source y relacional, es decir, MySQL ya que los datos deben ser consistentes, sin posibilidad de error, por lo que manejar tablas será más conveniente que colecciones no relacionales.

El diagrama mostrado en la figura 3.5 indica el flujo de eventos que se ejecutan desde que un dato es generado hasta que es enviado en la arquitectura funcional del nodo Edge.

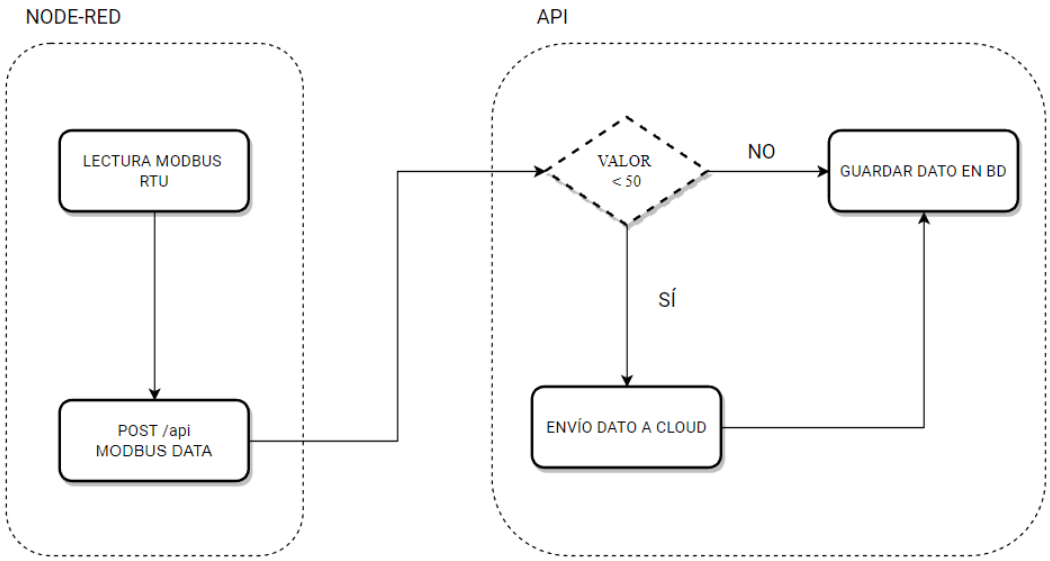

Figura 3.5: Diagrama de eventos en el nodo edge

Por otra parte el diagrama de la figura 3.6 resume como está compuesta la arquitectura a nivel de Kubernetes:

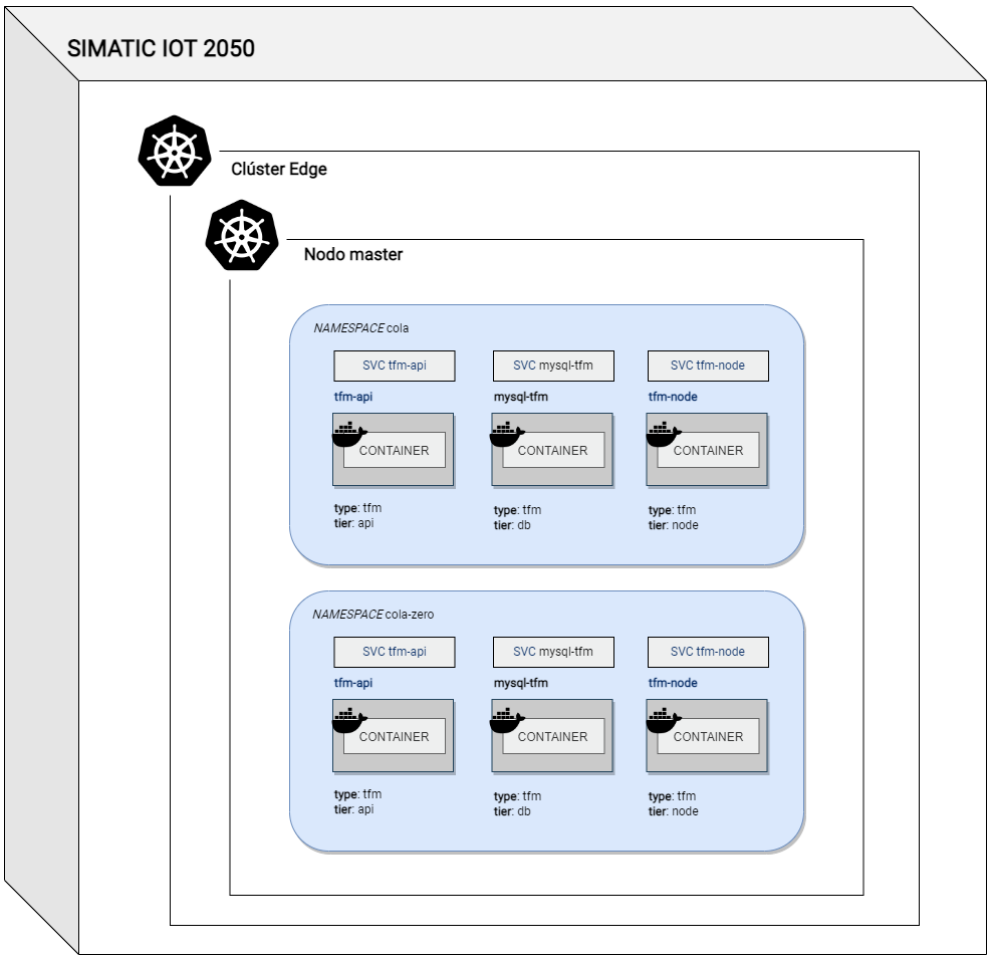

Figura 3.6: Arquitectura edge en Kubernetes

Finalmente, el diagrama de la figura 3.7 simplifica cada uno de los *namespaces* con el fin de dejar claro las interacciones de cada uno:

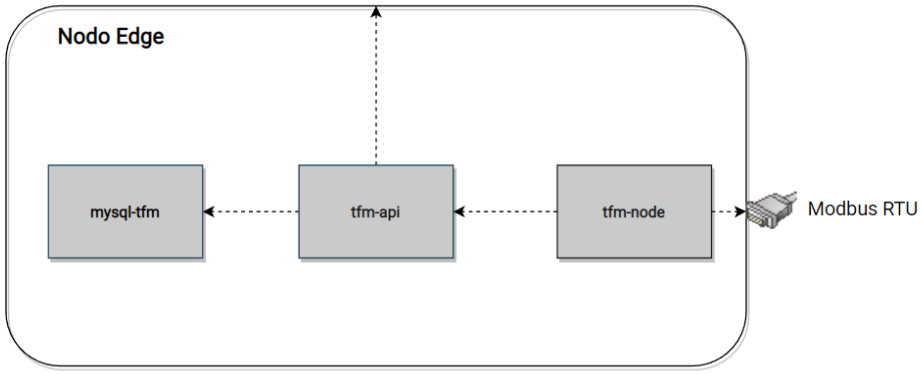

Figura 3.7: Bloques de servicios de la arquitectura edge

3.2.2. Cloud

El servidor Cloud se compone de tres componentes: el orquestador de clústeres, la plataforma IoT FIWARE y el Dashboard visualizador de errores. Estos componentes son responsables a nivel de arquitectura de ciertas funcionalidades que ayudan en la gestión de los nodo edge y de los datos que estos producen, sin embargo, tanto el orquestador de clústeres como FIWARE contienen componentes que ofrecen seguridad al proyecto y que serán explicados en el apartado de implementación de seguridad.

Así pues, las funcionalidades de estos tres componentes son las descritas a continuación:

- **Agente IoT FIWARE**: El agente IoT es la puerta de entrada para la conexión con el broker Orion. Ha sido programado haciendo uso de la librería PyNgsi de Python. La única funcionalidad del agente es la traducción del cuerpo de la llamada en formato JSON enviado por la API en el nodo Edge a NGSI. Una vez se ha realizado la conversión,los datos son enviados a Orion por el puerto 1026.
- **Context Broker Orion FIWARE:** El broker Orion se ocupa de la recepción de los datos y de la creación o actualización de una entidad. Es importante resaltar que únicamente se guarda el último valor recibido, por lo tanto, Orion al recibir un dato comprueba si este dato por su id y type ya están registrados o no. Si el dato se encuentra registrado se guarda si no se actualiza.

  El siguiente punto importante es la capacidad del broker Orion de generar alertas cuando el valor último registrado es diferente al último recibido. Estas alertas son recibidas por los consumidores (clientes) que realicen una petición de suscripción por tipo y id, también pudiendo elegir cual es el atributo que debe cambiar en la entidad para que se genere una alerta por parte de Orion.
- **Orquestador de clústeres:** El orquestador de clústeres juega un papel tanto en la arquitectura como en la seguridad posterior. Desde el punto de vista de la arquitectura inicial, ofrece un plano general de los servicios que han sido desplegados y monitoriza la salud de estos recursos. El orquestador está dividido en 3 pasos que ofrecen una centralización y control sobre los nodos edge y el propio clúster:

  1. **Agregación de clústeres**: Los clústeres son añadido al sistema del Orquestador de clústeres mediante su API expuesta. El sistema comprueba que este clúster contiene las características pertinentes para ser añadido y dependiendo del resultado lo habilita o no.
  2. **Repositorios de helm**: Helm permite instanciar los servicios de Kubernetes de forma empaquetada en vez de hacerlo por manifiestos. Para poder obtener los paquetes llamados *helm charts* con los servicios a desplegar, (un helm chart sería el broker Orion) es necesario añadir en el sistema un repositorio de helm, es decir, un repositorio de github con ciertas características necesarias para convertirse en un repositorio de helm.
  3. **Instanciación de servicios**: Con los clústeres añadidos y los respositorios de helm, se puede empezar a instanciar los servicios en los nodos edge que se deseen, es decir, desplegar los helm charts con los valores pertinentes.

- **Dashboard:** El dashboard se encarga de la visualización de los errores que ocurren en cualquiera de los nodo edge, haciendo distinción entre las diferentes líneas de producción que monitorizan cada uno de ellos. El dashboard contiene el logo de Coca-cola como ejemplo de planta real de producción de bebidas gaseosas. En el caso de este proyecto únicamente hay un nodo edge, pero son dos las líneas de producción. El dashboard se puede dividir en dos partes, el backend programado en Node JS y el frontend programado en Angular.

  Para que el dashboard pueda obtener las alertas se realiza la suscripción a Orion. La siguiente suscripción muestra que se realiza sobre cualquier id y tipo: variador-error, lo que significa que cualquier entidad que tenga como tipo variador-error será notificado al backend del dashboard.

```
 {
    "description": "Suscripcion",
    "subject": {
      "entities": [
        {
          "idPattern": ".*",
          "type":"variador-error"
        }
      ],
      "condition": {
        "attrs": [
          "valor"
        ]
      }
    },
    "notification": {
      "http": {
        "url": `${localIP}/api/v1/iot/fiware/callback`
      },
      "attrs": [
        "registro",
        "valor",
        "section"
      ],
      "attrsFormat": "legacy"
    },
    "expires": "2040-01-01T14:00:00.00Z",
    "throttling": 4
  }
}
```

  Por otra parte se indica que se quiere recibir la alerta únicamente si el atributo valor varía. Por consiguiente, cada vez que el valor registrado anteriormente sea distinto al recibido, será notificado al dashboard, mostrándose así por pantalla.

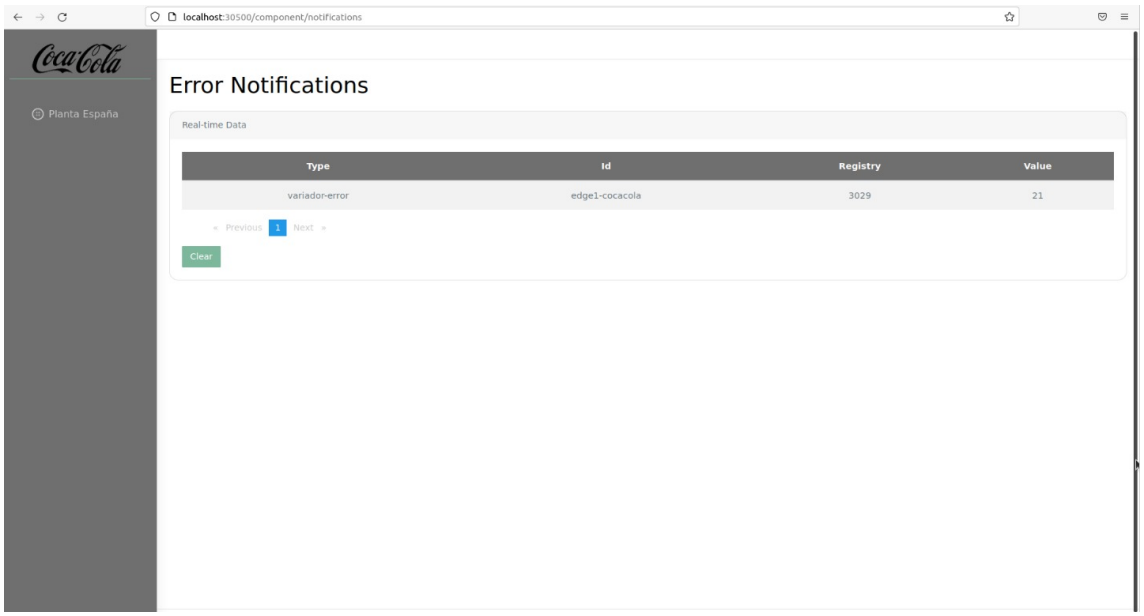

Figura 3.8: Dashboard

El diagrama de la figura 3.9 representa el flujo de eventos desde que el dato es recibido en Cloud hasta que es representado gráficamente en el dashboard.

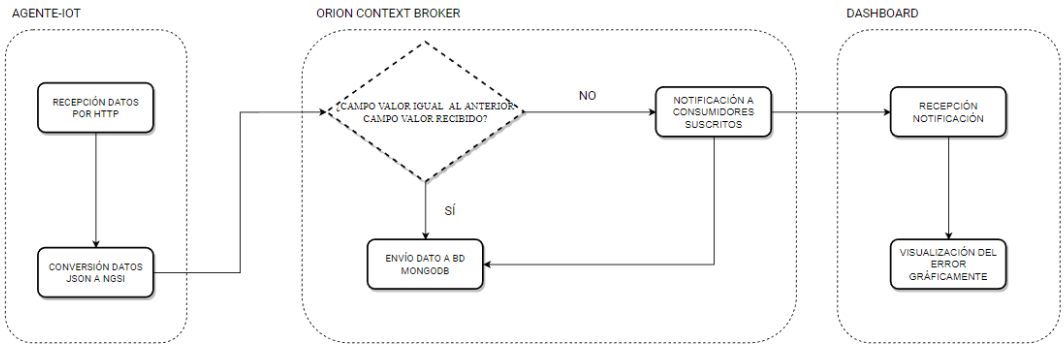

Figura 3.9: Diagrama de eventos en cloud

La figura 3.10 muestra la arquitectura de Kubernetes en Cloud:

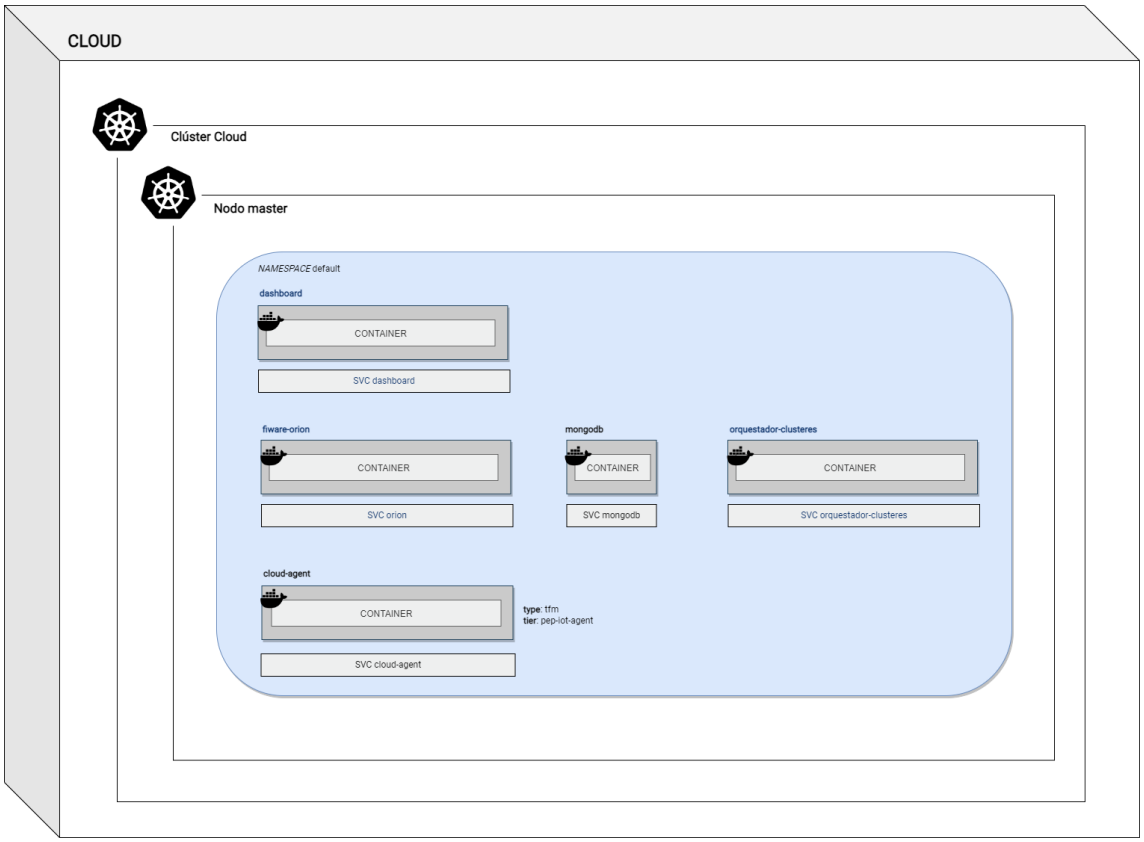

Figura 3.10: Arquitectura cloud en Kubernetes

Finalmente, en la figura 3.11 se muestra un resumen de las interacciones de los servicios que conforman el Cloud de forma resumida:

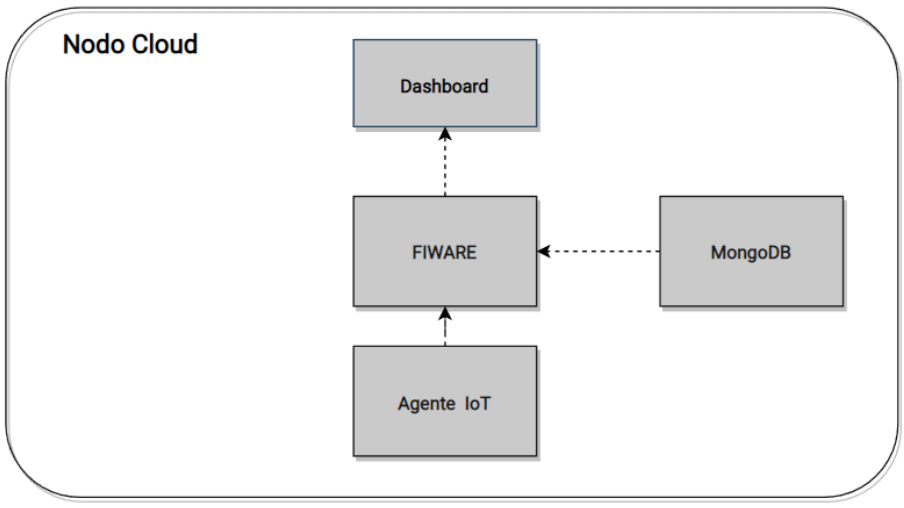

Figura 3.11: Bloques de servicios en la arquitectura cloud

Capítulo 4

Securización de la arquitectura

Este capítulo desarrolla el despliegue de seguridad realizado en la tesis. La implementación de la seguridad es realizada sobre diferentes puntos de la arquitectura inicial. El capítulo se divide en diferentes secciones, una por cada tecnología, y subsecciones, que ahondan en los componentes que aportan seguridad a la infraestructura inicial, anteriormente descrita. Finalmente, se presenta la arquitectura securizada final. Cabe tener en cuenta el contexto en el que se realiza, donde se dispone de un nodo edge para dos líneas de producción (una bebida gaseosa y otra bebida gaseosa sin azúcar) que envían los datos desde los variadores de frecuencia encargados de la regulación de las máquinas envasadoras.

4.1. Kubernetes

Esta sección muestra la seguridad que ofrecen los componentes de Kubernetes y como pueden ser utilizados en la securización de un nodo edge en el ámbito industrial. Así pues, también se explica como se ha realizado la instalación de cada uno de ellos.

4.1.1. Namespaces

Los *namespaces* a parte de la función que realizan de división de recursos en un mismo clúster físico, también proporcionan la posibilidad de crear políticas y reglas de acceso para los diferentes usuarios que quieran acceder a ellos.

El uso de *namespaces* en la tesis es idóneo, ya que permite la división de los recursos de la arquitectura funcional entre las líneas de producción, obteniendo de esta forma un *namespace* para la bebida gaseosa y otro para la misma sin azúcar, permitiendo así realizar ajustes en los componentes internos (Node-red, API y BD) y asignar usuarios y roles a cada *namespace*.

La creación de un *namespace* se puede realizar de distintas formas, la usada en la tesis es mediante la línea de comandos y la herramienta *kubectl*. Los *namespaces* creados son cola y cola-zero de la siguiente manera:

```
kubectl create ns cola
kubectl create ns cola-zero
```

4.1.2. Services

Los servicios en Kubernetes pueden ser de diferentes tipos, pero para el proyecto se utilizan dos principalmente: NodePort y clústerIP. NodePort permite exponer un puerto al exterior del clúster mientras que clústerIP únicamente lo expone a la red interna del clúster y por lo tanto solo es visible para los servicios internos a este.

Es el tipo *clústerIP* el que ofrece seguridad en la tesis, ya que al no estar expuesto al exterior del clúster solo aquellos servicios en la misma red son capaces de interactuar entre sí, siendo el administrador el único usuario con la capacidad de abrir un puerto con el fin de hacer accesible un dashboard o algún servicio para el desarrollo de una aplicación fuera del clúster.

Los servicios que se disponen en el nodo edge securizado son:

- mysql-tfm
- tfm-api
- tfm-node

Estos servicios están duplicados, es decir, los nombres son los mismos en el *namespace* cola y en el cola-zero, pero no se pueden comunicar entre sí al menos que se especifique el *namespace*.

4.1.3. Acceso a puerto hardware desde un contenedor

El acceso directo a interfaces de hardware del host donde se encuentra un contenedor es esencial en IoT. A pesar de la existencia de una vía para realizar este acceso, no es segura, ya que se trata de dar permisos de administrador sobre el host al contenedor. Así pues, se ha implementado una solución que cubría esta necesidad llamada smarter-device-manager.

El smarter-device-manager trata de un contenedor que una vez implementado empieza a leer en el directorio /dev y dependiendo de los dispositivos que se hayan indicado para ser mapeados del host a Kubernetes los identifica para que sean exportados.

En el caso de la tesis, la gateway Simatic IoT 2050 accede al puerto modbus por la interfaz */dev/ttyUSB0*, la cual será mapeada dentro del contenedor como *smarter-devices/ttyUSB0*.

Mediante la aplicación del siguiente *ConfigMap*, se indica las interfaces a mapear.

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: ConfigMap
metadata:
  name: smarter-device-manager-rpi #Nombre del recurso ConfigMap.
  namespace: default #Namespace donde se lanza el recurso.
data:
  conf.yaml: |
    - devicematch:  ^ttyUSB[0-9]*$
      nummaxdevices: 10
EOF
```

Para que se aplique el mapeo en la distribución k3s, se debe crear el *DaemonSet*, de la siguiente manera:

```
cat <<EOF | kubectl apply -f -
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: smarter-device-manager
  namespace: default
  labels:
    name: smarter-device-manager #Etiqueta de identifiación de recursos relevantes
    role: agent #Etiqueta de identifiación de recursos relevantes
spec:
  selector: #Identificar conjunto de recursos con el mismo label
    matchLabels:
      name: smarter-device-manager
  updateStrategy:
    type: RollingUpdate
  template:
    metadata:
      labels:
        name: smarter-device-manager
      annotations:
        node.Kubernetes.io/bootstrap-checkpoint: "true"
    spec:
      nodeSelector:
        smarter-device-manager : enabled #Lanzar DaemonSet en el que el nodo tenga el selector indicado.
      priorityClassName: "system-node-critical"
      hostname: smarter-device-management
      hostNetwork: true
      dnsPolicy: clústerFirstWithHostNet
      containers:
      - name: smarter-device-manager #Nombre contenedor.
        image: registry.gitlab.com/arm-research/smarter/smarter-device-manager:v1.20.7 #Nombre de la imagen.
        imagePullPolicy: IfNotPresent
        securityContext:
          allowPrivilegeEscalation: false
          capabilities:
            drop: ["ALL"]
        resources: # Límites de recursos y petición de uso de recursos.
```

```
            limits:
              cpu: 100m
              memory: 15Mi
            requests:
              cpu: 10m
              memory: 15Mi
          volumeMounts: # Montaje de carpeta dentro de un contenedor
            - name: device-plugin
              mountPath: /var/lib/kubelet/device-plugins
            - name: dev-dir
              mountPath: /dev
            - name: config
              mountPath: /root/config
            - name: sys-dir
              mountPath: /sys
        volumes: #Mapeado del volumeMount en una ruta en la máquina host
          - name: device-plugin
            hostPath:
              path: /var/lib/kubelet/device-plugins
          - name: dev-dir
            hostPath:
              path: /dev
          - name: sys-dir
            hostPath:
              path: /sys
          - name: config
            configMap:
               name: smarter-device-manager-rpi
        terminationGracePeriodSeconds: 30
EOF
```

Finalmente con el siguiente comando se implementa un pod de administrador de dispositivos en un nodo:

```
kubectl label node <nodo> smarter-device-manager=enabled
```

Para finalizar, se debe hacer uso de este recurso accesible en Kubernetes desde el manifiesto del *deployment* que quiera acceder al puerto [21]. En el caso de la tesis, es node-red el encargado de leer los datos que proceden del variador de frecuencia, por tanto se debe insertar en el *deployment* de node-red:

```
  resources:
    limits:
      smarter-devices/ttyUSB0: 2
      memory: 512Mi
    requests:
      smarter-devices/ttyUSB0: 2
      cpu: 10m
      memory: 50Mi
```

4.1.4. Secrets

El uso de los *secrets* se ha basado en la declaración de constantes sensibles como son contraseñas o nombres de usuario entre otros. La creación de este recurso de Kubernetes permite que se pasen estas variables codificadas en base64 y que estas constantes no se vean incluidas en la declaración de deployment o de un Pod.

La declaración de *secrets* aportan control sobre como se usan estas constantes confidenciales y de esta manera se reduce la posibilidad de que se vea exfiltrado un dato sensible.

Aquí puede surgir la duda de porque se han utilizado Secrets y no ConfigMap, la diferencia es que al aplicar ambos y describir su contenido mediante el comando *describe*, en el *ConfigMap* se aprecian todas las claves y valores mientras que en *Secrets* esto no ocurre.

Un ejemplo de *Secret* utilizado es el siguiente (la compartición de este secret no conlleva ningún riesgo de seguridad al tratarse de una simple demostración):

```
apiVersion: v1
kind: Secret
metadata:
  name: api-secrets
type: Opaque # Default tipo de Secret
data:
  MYSQL_USER: dXNlcg==
  MYSQL_PASSWORD: cGFzc3dvcmQ=
```

4.1.5. Audit Logs

4.1.5.1. Implementación

La implementación de los Audit Logs se ha realizado mediante una *Audit policy* que define cuales son los eventos que deben de ser guardados. Estos eventos pueden ser guardados en un fichero de logs o pueden ser enviados mediante un webhook a un servicio web.

Se ha utilizado la opción de guardado de los logs en un fichero. Ha sido esta la elección debido a la facilidad que aporta esta opción para mostrar los logs en un dashboard.

Primero cabe definir cuales son los eventos a registrar. Los recursos que se han monitorizado han sido los pods, dividiéndose en dos políticas:

- **RequestResponse**: Registrar metadatos de eventos (*timestamp*, usuario, verbo etc.), cuerpos de solicitud y respuesta.
- **Metadata**:Registrar los metadatos de la solicitud pero no la respuesta o el cuerpo de la solicitud.

De esta manera, utilizando ambos niveles, en el nivel *RequestResponse* se registran los pods en general, por lo tanto se obtendrá cualquier evento que los envuelva a ellos y en el nivel *Metadata* se registran los metadatos de los logs de los pods y el status.

El manifiesto queda de la siguiente manera:

```
apiVersion: audit.k8s.io/v1
kind: Policy

omitStages:
  - "RequestReceived"

rules:
  - level: RequestResponse
    resources:
    - group: ""
      resources: ["pods"]
  - level: Metadata
    resources:
    - group: ""
      resources: ["pods/log", "pods/status"]
    omitStages:
      - "RequestReceived"
```

Este manifiesto se guarda en un archivo, el cual es llamado en la tesis audit-policy.yaml y está ubicado en la ruta /etc/Kubernetes/audit. Con el fin de hacer efectivo este manifiesto se siguen los siguientes pasos:

1. Creación del archivo donde se guardarán los logs

```
sudo mkdir /var/log/Kubernetes/auditlog/
cd /var/log/Kubernetes/auditlog/
sudo touch audit.log
```

2. Editar el archivo de configuración *kube-apiserver*:

```
sudo nano /etc/Kubernetes/manifests/kube-apiserver.yaml
```

3. Actualizar la sección de *volumeMounts*:

```
volumeMounts:
- mountPath: /etc/Kubernetes/audit/audit-policy.yaml
    name: audit
    readOnly: true
- mountPath: /var/log/Kubernetes/auditlog/audit.log
    name: audit-log
    readOnly: false
```

4. Actualizar la sección *volumes*:

```
volumes:

- hostPath:
    path: /etc/Kubernetes/audit/audit-policy.yaml
    type: File
  name: audit
- hostPath:
    path: /var/log/Kubernetes/auditlog/audit.log
    type: FileOrCreate
  name: audit-log
```

5. Finalmente, se actualiza la sección de comandos del archivo para que se habilite la auditoría en el clúster

```
 --audit-policy-file=/etc/Kubernetes/audit/audit-policy.yaml
 --audit-log-path=/var/log/Kubernetes/auditlog/audit.log
```

Para que los cambios sean efectivos se mueve el archivo kube-apiserver.yaml fuera de la carpeta y se vuelve a mover otra vez dentro para que el pod kube-apiserver se reinicie y cargue la nueva configuración [22].

```
mv /etc/Kubernetes/manifests/kube-apiserver.yaml .
mv kube-apiserver.yaml /etc/Kubernetes/manifests/
```

4.1.5.2. Visualización de logs

El registro de eventos realizado hasta el momento se encuentra en un archivo de logs en formato json. En la tesis se ha utilizado FileBeat, ElasticSearch y Kibana con el fin de poder visualizarlos desde nuestro servidor Cloud, ya que el despliegue de estos servicios en el Edge suponen una carga computacional demasiado grande para un dispositivo diseñado para cargas livianas.

El flujo de eventos se resumen en los siguientes pasos, tal y como muestra la figura 4.1:

- Registro de un nuevo log en el archivo audit.log.
- FileBeat registra un cambio en el archivo y envía el nuevo log a ElasticSearch situado en el Cloud junto con Kibana.

- Kibana obtiene los datos de ElasticSearch.

Figura 4.1: Diagrama de eventos en la visualización de logs

Los tres elementos se instalan mediante el uso del Orquestador de clústeres haciendo uso de helm charts [23].

El helm chart y nombre del helm chart utilizados en cada uno de los casos son los siguientes:

- **ElasticSearch y Kibana (cloud)**: https://charts.bitnami.com/bitnami siendo el nombre elasticsearch y kibana respectivamente.
- **FileBeat (edge)**: https://helm.elastic.co siendo el nombre del chart filebeat. A diferencia de ElasticSearch y Kibana, al helm chart de FileBeat se le modifican *values* para indicar en que dirección se encuentra el ElasticSearch donde guardar los datos enviados. Los *values* son los siguientes:

```
daemonset:
  extraEnvs: []
filebeatConfig:
    filebeat.yml: |
      filebeat.inputs:
      - type: filestream
        id: my-filestream-id
        paths:
          - /var/log/Kubernetes/audit/audit.log
        parsers:
          - ndjson:
              keys_under_root: true
      output.elasticsearch:
        hosts: ["ElasticHost:9200"]
        protocol: http
tolerations:
- operator: "Exists"
```

4.1.6. Usuarios

La creación de usuarios en el proyecto se hace tanto a nivel de Kubernetes como a nivel de sistema operativo, de esta manera permite que un usuario al acceder a su sesión solo disponga del usuario de Kubernetes que se le ha sido asignado. Los usuarios que se han creado son:

- Encargado de la línea de distribución de la cola.

- Encargado de la línea de distribución de la cola-zero.
- Encargado de planta.

Por otra parte el administrador de Kubernetes ya tiene el usuario creado al crear un clúster, por lo tanto este no será necesario generarlo desde cero.Un usuario de Kubernetes inicialmente no dispone de ningún permiso, exceptuando el administrador, el cual puede realizar cualquier acción.

La creación de cada uno de estos usuarios es idéntico uno al otro, lo único que cambia son los roles que se explican en la subsección 2.2.2.10. Es por ello, que se detalla la creación de solo uno de ellos siendo el elegido el Encargado de la línea de distribución de cola *(Encargado cola)*.

Para la creación de un usuario, se debe de poseer un certificado generado por Kubernetes y luego que este certificado sea aprobado por la API de Kubernetes [24]. Los pasos a seguir son:

1. **Crear una clave privada**: Se genera una clave privada y un *Certificate Signing Request* CSR. Cabe destacar que el *CN* debe ser el nombre del usuario. Los comandos son:

```
openssl genrsa -out encargadocola.key 2048
openssl req -new -key encargadocola.key -out encargadocola.csr
```

2. **Crear un *CertificateSigningRequest***: La petición de firma del certificado permite que Kubernetes valide la clave del usuario creado anteriormente. En este apartado se utiliza el archivo CSR generado, pero debe de estar codificado en base64 en el apartado *request*. El siguiente *CertificateSigningRequest* es el utilizado para el usuario *Encargado cola*:

```
cat <<EOF | kubectl apply -f -
apiVersion: certificates.k8s.io/v1
kind: CertificateSigningRequest
metadata:
  name: encargadocola
spec:
  request: LS0tLS1CRUdJTiBDRVJUS ...
  signerName: Kubernetes.io/kube-apiserver-client
  expirationSeconds: 86400  # Expira en un día el certificado.
  usages:
  - client auth
EOF
```

3. **Aprobar el *CertificateSigningRequest***: Para que el usuario sea válido, el administrador debe aprobar el CSR. Se consigue con el comando:

```
kubectl certificate approve encargadocola
```

4. **Obtener el certificado generado del CSR**: El último paso es la exportación del certificado a un archivo.

```
kubectl get csr myuser -o jsonpath='{.status.certificate}'| base64 -d >
    encargadocola.crt
```

Para finalizar la creación del usuario, falta crear un usuario en el sistema operativo Debian, donde es guardado el nuevo kubeconfig. Para la creación del usuario es necesario dos pasos:

```
sudo useradd -m encargadocola
sudo passwd encargadocola
```

Para finalizar la creación de un usuario completo, es decir, crear un kubeconfig del nuevo usuario con sus permisos para ser exportado a un usuario del SO, se le tiene que asignar roles.

4.1.7. Roles

Los usuarios creados deben de poseer permisos distintos con el fin de que únicamente les sea posible gestionar ciertos recursos. Se aplican dos tipos de roles, los roles por *namespace* y los roles por clúster.

Los encargados tienen aplicados roles por *namespace*, así pues solo se les permite la gestión de recursos en su propia línea de envasamiento. Por tanto, el encargado de la línea de cola solo tiene acceso a los recursos en el *namespace* cola y el encargado de la línea de cola-zero solo tiene acceso a los recursos en el *namespace* cola-zero.

El encargado de encargado de cola va a tener acceso solo a listar y eliminar pods en el *namespace* cola:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: cola
  name: encargadocolarole
rules:
- apiGroups: [""]
  resources: ["pods","configmaps","services"]
  verbs: ["get", "watch", "list", "delete"]
```

Así pues, el rol se asigna a un usuario utilizando un recurso de Kubernetes llamado *RoleBinding*, donde se define cual es el role a aplicar y a qué usuario:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
   name: role-encargado
   namespace: cola
subjects:
- kind: User
  name: encargadocola
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: encargadocolarole
  apiGroup: rbac.authorization.k8s.io
```

Por otra parte, el encargado de planta dispone del rol de clúster para poder así acceder a los recursos

del clúster que se hayan definido. Este usuario tiene acceso a los recursos y verbos asignados en el *clústerRole*:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: clústerRole
metadata:
  name: encargadoplanta
rules:
- apiGroups: [""]
  resources: ["secrets","configmaps","services"]
  verbs: ["get", "watch", "list"]
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["create", "watch", "list","get","edit"]
- apiGroups: [""]
  resources: ["nodes"]
  verbs: ["watch", "list","get","edit"]
- apiGroups: [""]
  resources: ["validatingwebhookconfigurations"]
  verbs: ["create", "watch", "list","get","edit"]
```

Igual que se ha realizado anteriormente, para que sea efectivo ese *clústerRole* se debe asignar al usuario pertinente, es decir, al encargado de planta [25].

```
apiVersion: rbac.authorization.k8s.io/v1
kind: clústerRoleBinding
metadata:
   name: clúster-role-planta
subjects:
- kind: User
  name: encargadoplanta
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: clústerRole
  name: encargadoplanta
  apiGroup: rbac.authorization.k8s.io
```

4.1.8. Creación de un archivo kubeconfig

El paso final es la creación de un nuevo archivo Kubeconfig para el usuario del SO creado.

Para su creación se ejecuta el siguiente comando:

```
kubectl config set-credentials myuser --client-key=encargadocola.key --client-
    certificate=encargadocola.crt --embed-certs=true
```

Luego se añade el contexto:

```
kubectl config set-context encargadocola --clúster=Kubernetes --user=
    encargadocola
```

Una vez se tiene el kubeconfig realizado, se debe de copiar y pegar en otro archivo los campos relacionados con el usuario creado, es decir, el contexto y el usuario, y el clúster será también copiado aunque solo se dispondrá de uno. Este archivo se copia en el nuevo usuario creado en el SO y así cada uno de los encargados tendrá los accesos restringidos sin posibilidad de conmutar al usuario de administración.

4.2. FIWARE

Los componentes de FIWARE utilizados en la securización de la arquitectura son el PEP proxy y el gestor de identidades Keyrock ubicados en Cloud, encargados de la securización de la plataforma IoT FIWARE.

El elemento securizado es el agente IoT. Las comunicaciones de la API del nodo Edge con el agente están interceptadas por el PEP proxy que a su vez consulta en Keyrock si el nodo que solicita comunicarse con el agente IoT está autenticado.

Se pueden diferenciar dos líneas de defensa, siendo la primera el proxy que recibe las peticiones al agente IoT no siendo este accesible de forma directa sin pasar por el proxy y por otra parte se encuentra la capa encargada de la autenticación que define que usuarios tienen acceso al agente.

En esta sección se definen los pasos para la configuración de ambos componentes y como securizan las interacciones con el nodo edge.

4.2.1. Keyrock

En Keyrock, del gestor de identidad de FIWARE cabe tener en cuenta diferentes definiciones que se hacen uso en la tesis:

- **Usuario**: El usuario es el nodo edge, el cual es capaz de identificarse en el sistema mediante el uso de usuario/contraseña.
- **Permiso**: Autorización para la consulta/modificación de un recurso. En la arquitectura, el permiso que se otorga es la capacidad de realizar una llamada POST para enviar datos al agente.
- **Rol**: El rol está compuesto de un solo permiso nombrado en el anterior punto.
- **Aplicación**: La aplicación a securizar, es decir, el agente IoT.

Se debe tener en cuenta que Oauth2 se basa en *tokens*, es decir, cada nodo se autentica usando un usuario y una contraseña y *Keyrock* le devuelve un *token* que le permite al nodo acceder a las aplicaciones que tiene permisos.

Para realizar cualquier llamada al agente IoT, es necesario el uso del *token* en el mensaje. Las comunicaciones con keyrock son HTTPS para que la comunicación se vea encriptada y de esta manera minimizar el riesgo de que el token pueda llegar a ser interceptado.

Primero, se describe la instalación de *Keyrock*. Aunque existen helm charts de *keyrock*, ninguno contiene en *values* todas las variables que interesan en el desarrollo de la tesis. Es por ello que se crea un nuevo helm chart a partir de un docker-compose con la finalidad de tener todo el control sobre el *Keyrock* que se despliega en al arquitectura securizada. Su despliegue se realiza con el uso del orquestador de clústeres, haciendo uso de un repositorio de helm propio.

Las variables sensibles en el despliegue de *Keyrock* están declaradas en el *Secrets*.

```
1 apiVersion: v1
```

```
kind: Secret
metadata:
  name: keyrock-secrets
type: Opaque
data:
  MYSQL_USER: dXN1cg==
  MYSQL_PASSWORD: cGFzc3dvcmQ=
  MYSQL_ROOT_PASSWORD: cGFzc3dvcmQ=
  IDM_DB_PASS: cGFzc3dvcmQ=
  IDM_DB_USER: cm9vdA==
  IDM_ADMIN_EMAIL: YWRtaW5AdGZtLmNvbQ==
  IDM_ADMIN_PASS: dGZtcGFzcw==
  IDM_ADMIN_USER: YWRtaW4=
```

Por otra parte, entre las variables que no son sensibles, se puede destacar **IDM_HTTPS_ENABLED** que permite que las llamadas a *Keyrock* estén securizadas. Con el objetivo de securizar estás conexiones mediante TLS, se crean los certificados autofirmados que posteriormente son ubicados en */media/certs/keyrock*, de donde la instanciación del gestor de identidades obtiene los certificados. La generación de los certificados se consigue con los comandos:

```
openssl genrsa -out idm-2018-key.pem 2048
openssl req -new -sha256 -key idm-2018-key.pem -out idm-2018-csr.pem
openssl x509 -req -in idm-2018-csr.pem -signkey idm-2018-key.pem -out idm-2018-
    cert.pem
```

Para hacer accesible el servicio de *Keyrock* desde el exterior del clúster, se declara el tipo de *Service* de Kubernetes como NodePort mientras se realizan las configuraciones pertinentes.

La configuración de los usuarios y permisos pueden llevarse a cabo mediante el uso de la interfaz física o con llamadas a la API. En esta tesis, la configuración se ha realizado con la interfaz física. Los pasos a realizar para crear un usuario y darle permisos son los siguientes[16]:

1. **Creación del usuario**: En la creación de un usuario se accede a la interfaz web la cual se accede por el navegador en la dirección *localhost:NodePort* y se loguea con el email y contraseña (**IDM_ADMIN_EMAIL** y **IDM_ADMIN_PASS**) indicados en el despliegue de *Keyrock* mediante helm chart. La creación del usuario se realiza en la pestaña del menú *Usuarios* ->Crear usuario, donde se rellena un formulario.

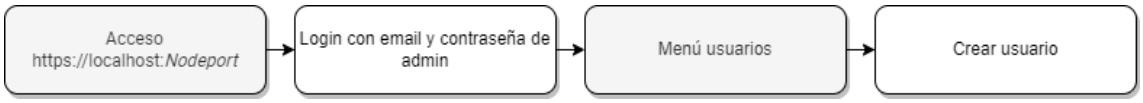

Figura 4.2: Flujo de eventos para la creación de un usuario

2. **Creación de aplicaciones**:

   a) Acceso al menú de Aplicaciones y registro de una aplicación.

   b) Se rellenan todos los campos, aunque los relativos a *URL* no son críticos debido a que se utilizan únicamente cuando hay que redireccionar a una ruta a los usuarios tras un login o logout.

c) En la pestaña de gestionar roles al darle a siguiente en el anterior paso se guarda la configuración tal y como la sugiere *Keyrock* de momento.

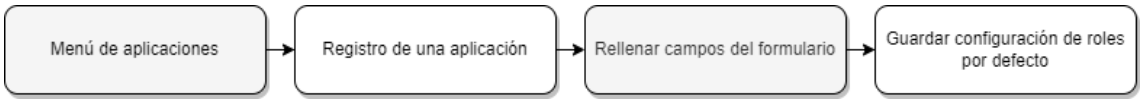

Figura 4.3: Flujo de eventos para la creación de una aplicación

3. **Creación de roles y permisos**:

   a) Los roles y permisos de una aplicación se gestionan accediendo primeramente a ella.

   b) La opción a seleccionar es *Gestión de roles*.

   c) Se añade un nuevo rol pulsando +, que será llamado *Edge*.

   d) Tras la creación del nuevo rol, se añaden permisos apretando el símbolo +. Para que la API del nodo tenga acceso al agente, el endpoint a indicar es /upload.

   e) Se selecciona el permiso creado para asignarlo al rol Edge.

   f) En la pantalla de la aplicación, se le asigna el rol al usuario creado con la opción Autorizar.

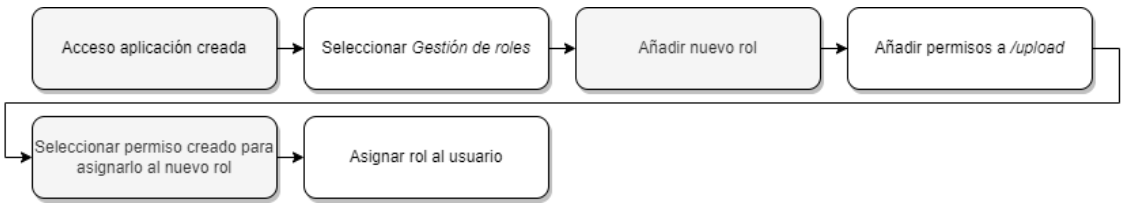

Figura 4.4: Flujo de eventos para la creación y asignación de roles

4.2.2. PEP Wilma

El PEP Wilma hace función de proxy tal y como se explica en el apartado 2.4.1.2. El esquema es el mostrado en la figura 4.5.

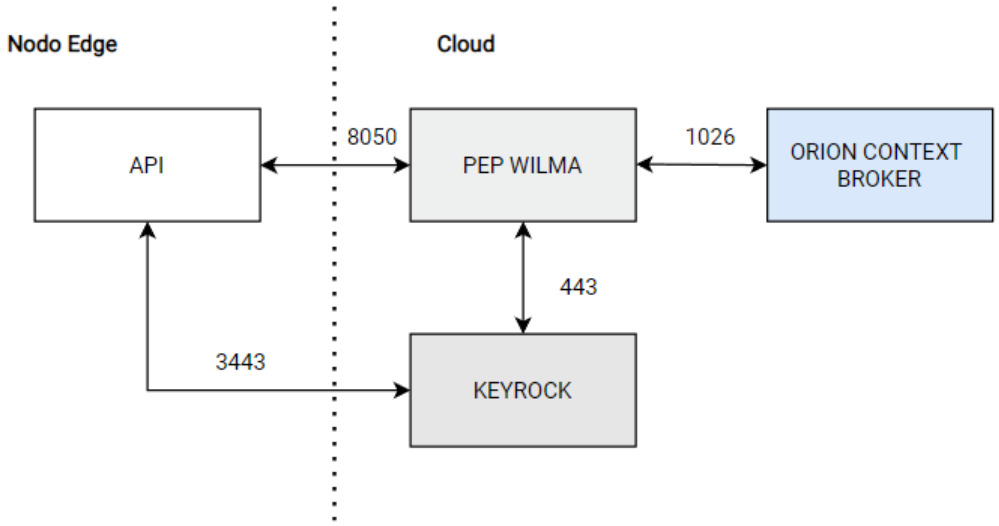

Figura 4.5: Esquema de securización PEP Wilma

Con anterioridad a la configuración del proxy es necesario identificarlo en *Keyrock* y asociarlo a la aplicación registrada anteriormente. Para ello se accede a la aplicación y se registra un nuevo PEP proxy. Una vez registrado, se devuelven 3 valores: Application Id, nombre del PEP proxy y contraseña del PEP proxy.

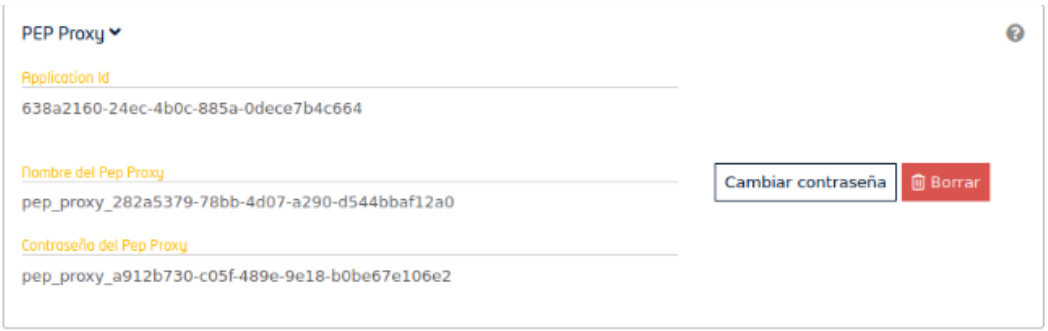

Figura 4.6: Keyrock valores para el registro de PEP Wilma

Para su despliegue, debido a la inexistencia de ningún helm chart se ha creado uno a partir de un docker-compose igual que en *Keyrock*. También es necesario la creación de certificados para que las conexiones sean seguras [15]. Se consiguen los certificados autofirmados con los siguientes comandos:

```
openssl genrsa -out key.key 2048
openssl req -new -sha256 -key key.key -out csr.crt
openssl x509 -req -in csr.crt -signkey key.key -out cert.crt
```

Estos certificados luego son ubicados en */media/certs/* de donde la instancia del *PEP proxy* los obtiene.

Su despliegue se realiza haciendo uso del orquestador de clústeres, que al igual que *Keyrock*, el helm chart se encuentra en un repositorio propio. Las variables a destacar son:

- **PEP_PASSWORD**: Contraseña del PEP proxy
- **PEP_PROXY_USERNAME**: Nombre del PEP proxy
- **PEP_PROXY_APP_ID**: Application Id
- **PEP_PROXY_APP_PORT**: 8880 (Puerto del Agente IoT)
- **PEP_PROXY_HTTPS_ENABLED**: true (Habilitar HTTPS)
- **PEP_PROXY_APP_HOST**: cloud-agent (*Service* del Agente IoT)
- **PEP_PROXY_IDM_SSL_ENABLED**:true (Conexión HTTPS con *Keyrock*)

4.3. Cilium

Las características utilizadas en Cilium para la securización de las conexiones entre Edge y Cloud son *clústermesh* junto con encriptación de las conexiones mediante TLS y las *Network policies*.

Cilium debe aplicarse en ambos de los clústeres, es decir, en el nodo Edge y en Cloud. La instalación de Cilium se ha llevado a cabo con Cilium CLI. Para la instalación del CLI se siguen los siguientes pasos:

```
curl -L --remote-name-all https://github.com/cilium/cilium-cli/releases/latest/
    download/cilium-linux-amd64.tar.gz{,.sha256sum}
sha256sum --check cilium-linux-amd64.tar.gz.sha256sum
sudo tar xzvfC cilium-linux-amd64.tar.gz /usr/local/bin
rm cilium-linux-amd64.tar.gz{,.sha256sum}
```

4.3.1. clústermesh y Transparent TLS Encryption

La encriptación se realiza directamente en la instalación de Cilium con Cilium CLI. En el caso de la tesis se hace uso de *Wireguard* [26]. El flag es:

```
--encryption wireguard
```

Seguidamente es necesario para la configuración de clústermesh utilizar en cada uno de los clústeres un nombre y un ID distintos a nivel de Kubernetes. Los flags son para edge:

```
--clúster-name edge --clúster-id 1
```

Finalmente se instala cilium con los flags indicados en cada clúster de forma manual:

```
cilium install --encryption wireguard --clúster-name cloud --clúster-id 1
cilium install --encryption wireguard --clúster-name edge --clúster-id 2
```

Para que la configuración de clústermesh sea más ágil se trabaja con contextos de Kubernetes. El *Kubeconfig* es el archivo a modificar siendo cualquiera de los dos válido, es decir, el clúster edge o el cloud. De esta forma es posible realizar las instalaciones desde una misma máquina conmutando entre clústeres.

El archivo *Kubeconfig* tiene la siguiente forma:

```
apiVersion: v1
clústers:
- clúster:
    certificate-authority: edge-ca-file
    server: https://1.2.3.4
  name: edge
- clúster:
    certificate-authority: cloud-ca-file
    server: https://1.2.3.5
```

```
  name: cloud
contexts:
- context:
    clúster: edge
    user: edge
  name: edge
- context:
    clúster: cloud
    user: cloud
  name: cloud
current-context: ""
kind: Config
preferences: {}
users:
- name: edge
  user:
    client-certificate: edge-cert-file
    client-key: edge-key-file
- name: cloud
  user:
    client-certificate: cloud-cert-file
    client-key: cloud-key-file
```

Primero se habilita el clústermesh en ambos contextos (edge y cloud):

```
cilium clústermesh enable --context edge --service-type NodePort
cilium clústermesh enable --context cloud --service-type NodePort
```

Una vez se tiene el clústermesh activado, se deben conectar los dos clústeres. Con Cilium CLI solo se debe de hacer en una dirección, es decir, cloud con edge o edge con cloud [27].

```
cilium clústermesh connect --context $clúster1 --destination-context $clúster2
```

Se puede comprobar la conexión con el comando:

```
cilium clústermesh status --context $clúster1 --wait
```

4.3.2. Network policies

Las *Network policies* en el proyecto permiten que únicamente las APIs situadas en los dos *namespaces* cola y cola-zero puedan conectarse con con el PEP proxy al puerto 8050, así se puede evitar que cualquiera que quiera inyectar información falsa sobre errores no lo pueda hacer ya que está denegado el acceso.

Para identificar que la información proviene de la API se hace uso de los labels que la identifican en Kubernetes. En el deployment de la API en el nodo edge se declaran dos labels:

- **type**: tfm
- **tier**: api

De esta forma se identifica de donde provienen los datos y con el label **k8s:io.Kubernetes.pod.namespace** se indica el *namespace* permitido. También por medio de labels se indica cual es el label al que puede acceder [28]. El manifiesto del network policy tiene la siguiente estructura:

```
apiVersion: "cilium.io/v2"
kind: CiliumNetworkPolicy
metadata:
  name: "clústermesh-ingress-l4-policy"
  # description: "demo: allow only employee to access protected-db"
spec:
  endpointSelector:
    matchLabels:
      io.kompose.service: pep-iot-agent
  ingress:
  - toPorts:
    - ports:
      - port: "8050"
        protocol: TCP
    fromEndpoints:
      - matchLabels:
          type: tfm
          tier: api
          k8s:io.Kubernetes.pod.namespace: cola
      - matchLabels:
          type: tfm
          tier: api
          k8s:io.Kubernetes.pod.namespace: cola-zero
```

Capítulo 5

Prototipo

Tras finalizar la especificación y diseño de una nueva arquitectura y la implementación de la seguridad con todos los archivos configurados para el correcto funcionamiento del sistema, se realiza un prototipo con el fin de comprobar que el sistema funciona como se espera en un entorno real. Los datos son enviados desde el variador de frecuencia hasta cloud pasando por el edge. En esta sección se muestra únicamente las evidencias de la seguridad implementada, ya que el montaje de la arquitectura inicial se indica en el capítulo 3.

Toda la información para el despliegue del prototipo se encuentra alojado en el repositorio **FMBIoT**

5.1. Montaje

La simulación del entorno de pruebas se realiza sobre los elementos nombrados en la arquitectura física:

- **Variador de frecuencia**: Dispositivo encargado de la generación de datos.
- **Simatic IoT 2050**: Nodo edge con todos los servicios virtualizados.
- **Router**: Dispositivo que proporciona conexión a internet al nodo edge.
- **Servidor virtualizado**: Servidor que alberga todos los servicios que tienen una alta carga computacional y que deben ser centralizados.

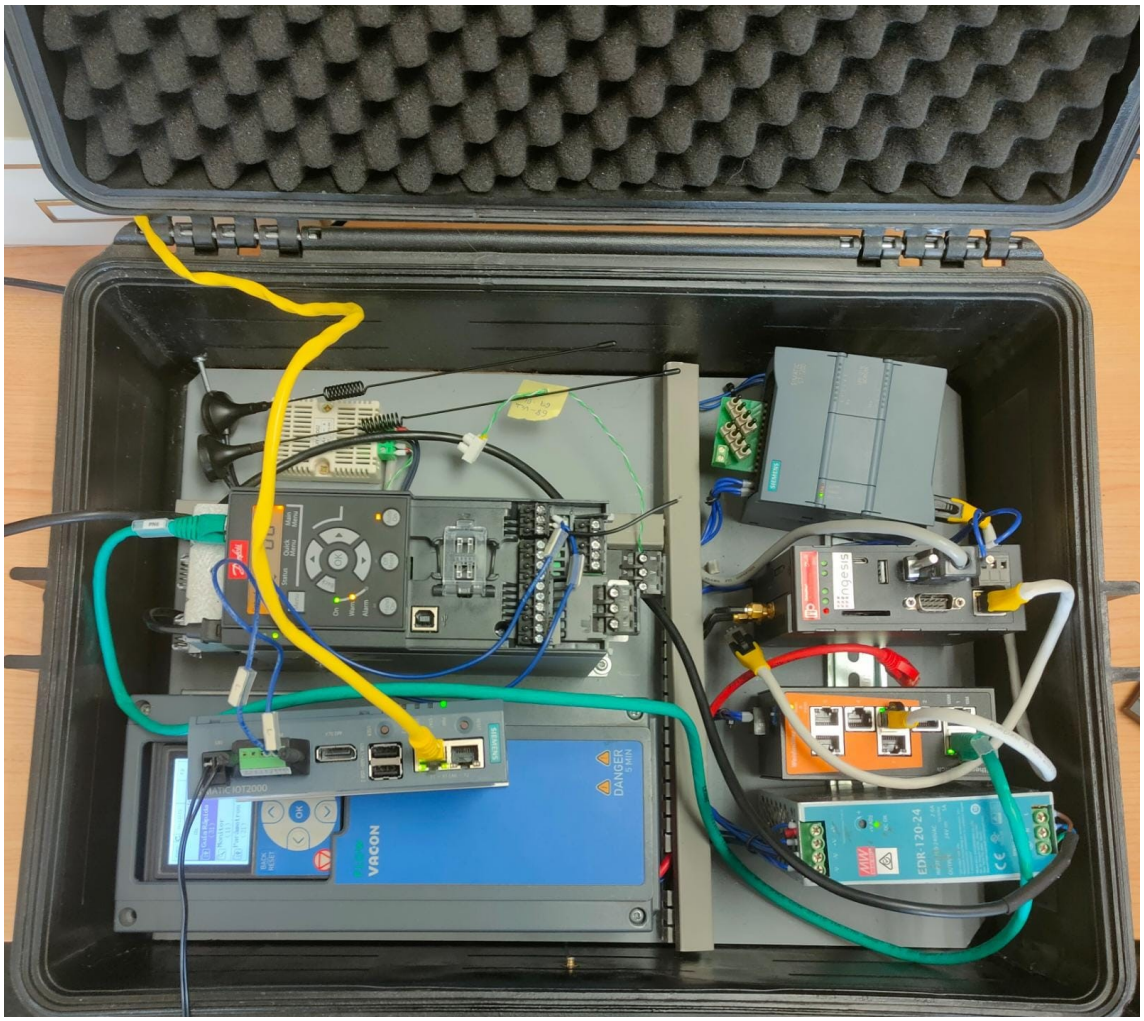

Figura 5.1: Montaje de la tesis

5.2. Servicios

Los servicios pueden ser divididos en aquellos que forman parte del Cloud y del Edge y aquellos que forman parte de ambos. Cada subsección muestra imágenes de la implementación en el prototipo de la securización.

5.2.1. Edge

Los contenedores que están en Edge son aquellos que conforman la arquitectura. El único que aporta seguridad es el contenedor **smart-devices** para la conexión segura entre el contenedor de node-red y el hardware fisico de la simatic iot 2050.

Así pues, estos son los contenedores que conforman los servicios del nodo edge tal y como muestra la figura 5.2

```
k8s-machine4@k8s-machine4:~/tfm/EDGE/kubernetes/Logs$ kubectl get po -n cola
NAME                          READY   STATUS    RESTARTS   AGE
mysql-tfm-0                   1/1     Running   0          6d2h
tfm-api-78d88ff747-qm87r      1/1     Running   0          5d9h
tfm-node-85f8585b9d-r6st4     1/1     Running   0          6d2h
k8s-machine4@k8s-machine4:~/tfm/EDGE/kubernetes/Logs$ kubectl get po -n default
NAME                          READY   STATUS    RESTARTS   AGE
smarter-device-manager-kq9wz  1/1     Running   0          5m15s
k8s-machine4@k8s-machine4:~/tfm/EDGE/kubernetes/Logs$ kubectl get po -n kube-system
NAME                                  READY   STATUS    RESTARTS      AGE
cilium-56b6f                          1/1     Running   0             6d6h
cilium-operator-5d67fc458d-jqcws      1/1     Running   0             6d6h
clustermesh-apiserver-b784f98cb-tdwdv 2/2     Running   0             6d6h
coredns-64897985d-gnd5f               1/1     Running   0             25d
coredns-64897985d-rrrn8               1/1     Running   0             25d
etcd-k8s-machine4                     1/1     Running   276           25d
filebeat-filebeat-bw9zc               1/1     Running   1 (40s ago)   46s
kube-apiserver-k8s-machine4           1/1     Running   1 (8d ago)    8d
kube-controller-manager-k8s-machine4  1/1     Running   5 (8d ago)    25d
kube-proxy-z9bwq                      1/1     Running   0             25d
kube-scheduler-k8s-machine4           1/1     Running   5 (8d ago)    25d
```

Figura 5.2: Servicios en el nodo Edge

Por otra parte la implementación de cada una de las partes de seguridad del nodo edge son probadas para verificar que funcionan correctamente:

- **Usuario y roles**: Para la comprobación del funcionamiento se crea encargado cola como usuario tanto en Kubernetes como en el SO y seguidamente se aplican el rol configurado para él. Tal como se puede comprobar en la figura 5.3 no puede acceder a los recursos que no se encuentren en el *namespace* cola y únicamente a los que se indican en el rol.

```
$ kubectl get po -n cola-zero
Error from server (Forbidden): pods is forbidden: User "encargadocola" cannot list resource "pods" in API group "" in the namespace "cola-zero"
$ kubectl get po -n cola
NAME                          READY   STATUS    RESTARTS   AGE
mysql-tfm-0                   1/1     Running   0          7d1h
tfm-api-78d88ff747-qm87r      1/1     Running   0          6d8h
tfm-node-85f8585b9d-r6st4     1/1     Running   0          7d2h
$ kubectl get configmap -n cola
NAME               DATA   AGE
kube-root-ca.crt   1      7d2h
$ kubectl get deployment -n cola
Error from server (Forbidden): deployments.apps is forbidden: User "encargadocola" cannot list resource "deployments" in API group "apps" in the namespace "cola"
```

Figura 5.3: Acceso usuario encargado cola a recursos

- **Audit Logs**: El contenedor encargado del envío a Elasticsearch en cloud es *Filebeat*.
- **Services**: Los services que conforman el nodo edge son uno por cada uno de los pods instanciados.
- **Namespaces**: Dos *namespaces* creados, cada uno para una línea de producción.

```
k8s-machine4@k8s-machine4:~/tfm/EDGE/kubernetes/Logs$ kubectl get ns
NAME              STATUS   AGE
cola              Active   6d3h
cola-zero         Active   6d3h
default           Active   25d
kube-node-lease   Active   25d
kube-public       Active   25d
kube-system       Active   25d
```

Figura 5.4: Namespaces en el nodo edge

5.2.2. Cloud

Los servicios Cloud están compuestos por todos aquellos servicios que necesitan más recursos para funcionar. Los servicios Cloud que aportan seguridad son los siguientes:

- **ElasticSearch y Kibana**: Forman parte del componente de visualización de logs. Reciben los datos del nodo edge mediante el uso del contenedor de Filebeat. La visualización de los logs se realiza en la interfaz gráfica de Kibana.

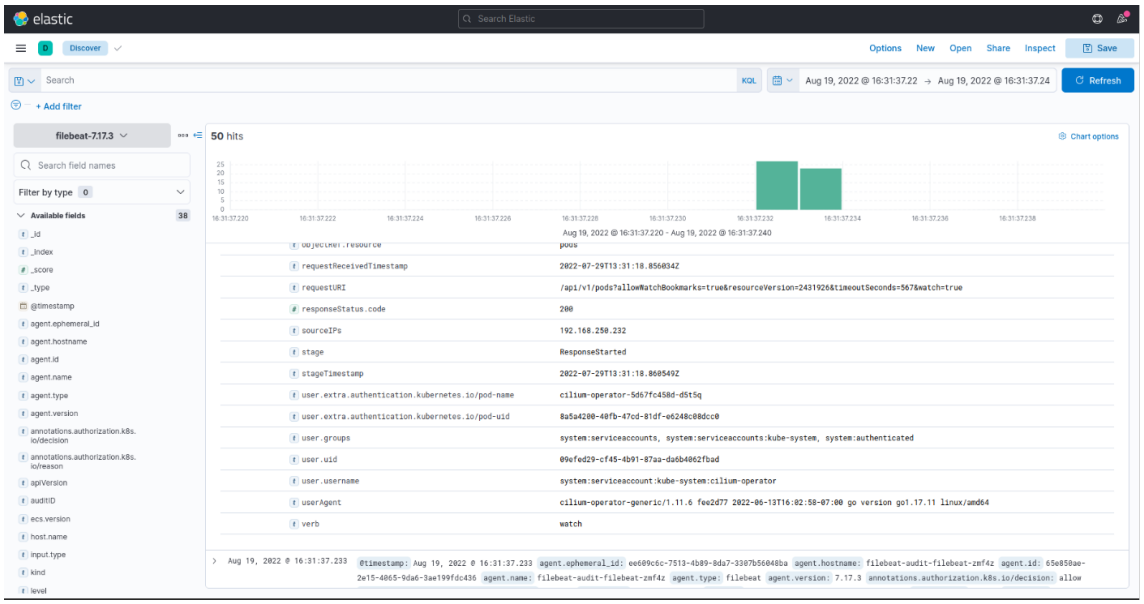

Figura 5.5: Kibana dashboard

En la figura 5.5, los logs guardados disponen del verbo que se ha realizado en este caso *watch*, del usuario y del *timestamp* en el que se realiza la acción.

- **Keyrock**: Gestor de identidades.
- **PEP proxy**: Proxy verificador de la identidad del cliente que quiere acceder a la aplicación securizada.

En la figura 5.6 se muestran los servicios activos en Cloud.

```
fmbiot@fmbiot:~$ kubectl get po
NAME                                    READY   STATUS    RESTARTS      AGE
cloud-agent-57484cd8f4-8nb6x            1/1     Running   0             45h
dashboard-c9cf786fc-8hst8               1/1     Running   0             45h
elastic-elasticsearch-coordinating-0    1/1     Running   0             45h
elastic-elasticsearch-coordinating-1    1/1     Running   0             45h
elastic-elasticsearch-data-0            1/1     Running   0             45h
elastic-elasticsearch-data-1            1/1     Running   0             45h
elastic-elasticsearch-ingest-0          1/1     Running   0             45h
elastic-elasticsearch-ingest-1          1/1     Running   0             45h
elastic-elasticsearch-master-0          1/1     Running   0             45h
elastic-elasticsearch-master-1          1/1     Running   0             45h
fiware-keyrock-7ddcd47798-t28zq         1/1     Running   0             45h
kibana-kibana-54fffcc67d-vdj6z          1/1     Running   0             45h
mysql-keyrock-0                         1/1     Running   0             45h
orion-84b67d8bb8-6726n                  1/1     Running   0             45h
orion-mongo-866fccd689-j4glg            1/1     Running   2 (45h ago)   45h
pep-iot-agent-66b749df4c-q5x5s          1/1     Running   0             40h
```

Figura 5.6: Servicios en Cloud

5.2.3. Cloud y Edge

Los contenedores que comparten ambos son los de Cilium, ubicados en el *namespace* Kube-system. En la figura 5.7 se muestra como tanto desde el nodo edge como desde el nodo Cloud se es capaz de ver que están conectados los clústeres entre sí.

```
# ^cilium node list
Name                IPv4 Address      Endpoint CIDR   IPv6 Address   Endpoint CIDR
cloud/fmbiot        192.168.250.168   10.0.0.0/24
edge/k8s-machine4   192.168.250.232   10.0.0.0/24
```

Figura 5.7: Captura de los nodos de cilium

Por otra parte también se es capaz de la verificación del funcionamiento de la encriptación con Wireguard de los datos cuando las comunicaciones se producen entre estos dos clústeres tal como muestra la figura 5.8

```
root@k8s-machine4:/home/cilium# tcpdump -n -i cilium_wg0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on cilium_wg0, link-type RAW (Raw IP), capture size 262144 bytes
08:46:12.231904 IP 10.0.0.205.60640 > 10.0.0.143.3443: Flags [S], seq 423755345, win 64860, options [mss 1380,sackOK,TS val 1340328231 ecr 0,nop,wscale 7], length 0
08:46:12.233278 IP 10.0.0.143.3443 > 10.0.0.205.60640: Flags [S.], seq 3381935077, ack 423755346, win 64582, options [mss 1330,sackOK,TS val 1690561682 ecr 1340328231,nop,wscale 7], length 0
08:46:12.233342 IP 10.0.0.205.60640 > 10.0.0.143.3443: Flags [.], ack 1, win 507, options [nop,nop,TS val 1340328233 ecr 1690561682], length 0
08:46:12.233444 IP 10.0.0.205.60640 > 10.0.0.143.3443: Flags [P.], seq 1:352, ack 1, win 507, options [nop,nop,TS val 1340328233 ecr 1690561682], length 351
08:46:12.233504 IP 10.0.0.205.60640 > 10.0.0.143.3443: Flags [P.], seq 352:412, ack 1, win 507, options [nop,nop,TS val 1340328233 ecr 1690561682], length 60
08:46:12.233710 IP 10.0.0.143.3443 > 10.0.0.205.60640: Flags [.], ack 352, win 502, options [nop,nop,TS val 1690561682 ecr 1340328233], length 0
08:46:12.233744 IP 10.0.0.143.3443 > 10.0.0.205.60640: Flags [.], ack 412, win 502, options [nop,nop,TS val 1690561682 ecr 1340328233], length 0
08:46:12.236416 IP 10.0.0.143.3443 > 10.0.0.205.60640: Flags [P.], seq 1:335, ack 412, win 502, options [nop,nop,TS val 1690561685 ecr 1340328233], length 334
08:46:12.236464 IP 10.0.0.205.60640 > 10.0.0.143.3443: Flags [.], ack 335, win 505, options [nop,nop,TS val 1340328236 ecr 1690561685], length 0
08:46:12.236989 IP 10.0.0.205.60640 > 10.0.0.143.3443: Flags [F.], seq 412, ack 335, win 505, options [nop,nop,TS val 1340328236 ecr 1690561685], length 0
08:46:12.240098 IP 10.0.0.143.3443 > 10.0.0.205.60640: Flags [F.], seq 335, ack 413, win 502, options [nop,nop,TS val 1690561686 ecr 1340328236], length 0
08:46:12.240143 IP 10.0.0.205.60640 > 10.0.0.143.3443: Flags [.], ack 336, win 505, options [nop,nop,TS val 1340328239 ecr 1690561686], length 0
08:47:02.894849 IP 10.0.0.205.60642 > 10.0.0.143.3443: Flags [S], seq 3740853312, win 64860, options [mss 1380,sackOK,TS val 1340378894 ecr 0,nop,wscale 7], length 0
08:47:02.896966 IP 10.0.0.143.3443 > 10.0.0.205.60642: Flags [S.], seq 1813941120, ack 3740853313, win 64582, options [mss 1330,sackOK,TS val 1690612345 ecr 1340378894,nop,wscale 7], length 0
```

Figura 5.8: Captura de la encriptación datos

Finalmente, la arquitectura final securizada se muestra en la figura 5.9.

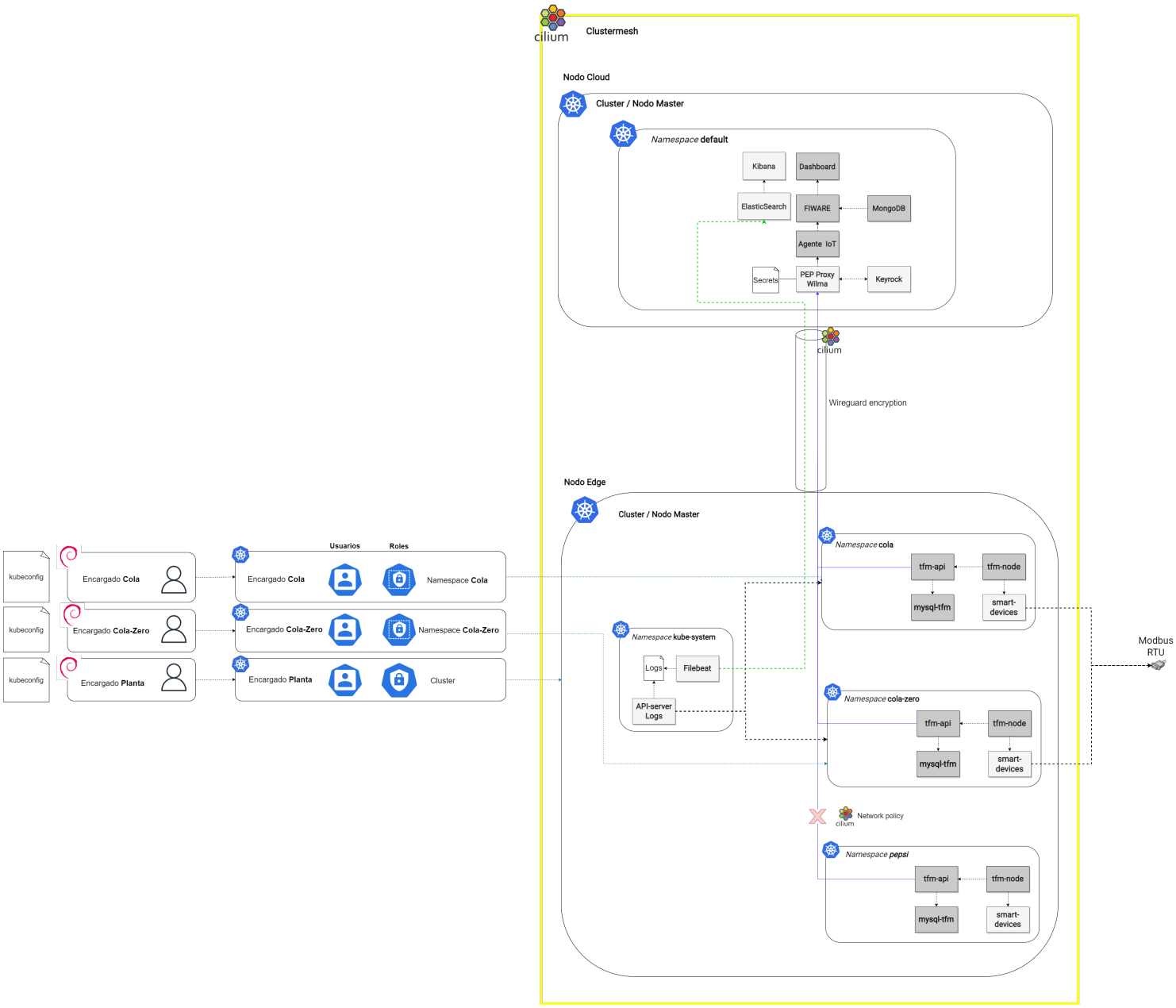

Figura 5.9: Arquitectura final

Capítulo 6

Conclusiones y líneas futuras

6.1. Conclusiones

En esta tesis se ha realizado la securización de un nodo edge en un entorno industrial haciendo uso de nuevas tecnologías emergentes sobre una arquitectura inicial que pretende ser segura desde su desarrollo con el uso de diferentes tecnologías como FIWARE o el orquestador de clústeres.

Con el fin de realizar este proyecto previamente se tuvo que realizar un estudio sobre los sistemas de virtualización que existen y cual es el más óptimo para el despliegue de servicios, siendo este Kubernetes debido a la robustez que aporta ante fallos en los servicios. Por otra parte, el hecho de la virtualización de todos los servicios hacía surgir nuevos problemas como el acceso de estos al hardware físico del nodo. Por ello se han investigado los diferentes sensores aplicables en un entorno real para probar el acceso de los servicios virtualizados a las interfaces físicas y como hacerlo de forma segura. Kubernetes aporta una gran diversidad de recursos para la securización de los despliegues en la plataforma, de los cuales se han seleccionado aquellos que más aplicación pueden llegar a tener en un entorno industrial. También es importante la distribución de Kubernetes, pues habiendo diferentes distribuciones que nos permite su uso en hardware con pocos recursos, k3s ha sido elegida como la opción óptima por ser una distribución liviana y con menor consumo de recursos.

FIWARE dispone de diferentes context brokers al igual que distintas distribuciones pensadas tanto para Cloud como Edge. Finalmente, debido a la facilidad que aporta el uso del Context Broker Orion junto con las opciones de securización que ofrece FIWARE, ha sido la opción seleccionada.

Cilium como CNI supone una tecnología novedosa que se encuentra en maduración y que ofrece la posibilidad de la conexión de clústeres como si se tratase de uno, permitiendo también así la encriptación de los datos entre los diferentes clústeres. Esta parte ha supuesto un avance importante en la facilidad de securización de conexiones entre los servicios virtualizados en el proyecto.

El trabajo se ha enfocado principalmente al desarrollo del montaje del sistema, tanto en la realización de una nueva arquitectura funcional inicial que sea segura desde su desarrollo como su securización. A pesar de que el sistema cumple con los requisitos iniciales, cabe destacar que hay aspectos a mejorar en la implementación y en las funcionalidades que ofrece.

Esta tesis aporta un avance a nivel de seguridad industrial con el uso de tecnologías de libre uso, las cuales ofrecen una gama muy amplía de opciones para aumentar la seguridad en todos los niveles y evitar vulnerabilidades que puedan acarrear una pérdida económica o de reputación empresarial. Por otra parte, permite que las empresas pequeñas que no pueden disponer de un gran presupuesto en seguridad sean capaces de mitigar posibles ataques con las opciones de seguridad implementadas y las demás disponibles, lo cual también aporta un crecimiento económico a la empresa por la fiabilidad que ofrece al cliente, así como evitar pérdidas económicas por ataques.

Para finalizar, destacar que este proyecto ha supuesto la aplicación de todos los conocimientos relacionados con la línea de investigación del proyecto **ASSIST-IoT** y **VARIO-EDGE**. Esto ha permitido visualizar de forma más sencilla cómo los diferentes componentes pueden ser utilizados conjuntamente entre sí.

6.2. Líneas futuras

El desarrollo de un proyecto tecnológico en un entorno como el industrial ofrece una gran variedad de propuestas que permiten mejorarlo, ya que se encuentra en un proceso continuo de desarrollo. Las líneas futuras pueden ser más exactas cuando se produce cierto *feedback* del cliente para poder así desarrollar mejoras.

Como línea futura de desarrollo se propone el uso de inteligencia artificial. El uso de la inteligencia artificial en el proyecto puede ser usado en dos ámbitos. El primer ámbito es en la detección de errores de los variadores de frecuencia para así realizar de forma preventiva un mantenimiento de estos y evitar fallos posteriores. Por otra parte, el uso de IA puede encontrarse en el escalado de servicios en el caso de que haya una gran demanda de datos en algún momento del día. De esta manera se realiza este escalado de forma preventiva para poder ofrecer el servicio sin interrupciones.

La segunda línea se encuentra relacionada con la posibilidad de modificación de los flujos de node-red en cada nodo edge para así poder realizarlo de forma centralizada, sin la necesidad de abrir y acceder a la interfaz de node-red. Todo este proceso puede ser realizado mediante el uso de la API de node-red y el dashboard en Cloud.

Finalmente, la tercera línea es el orquestador de clústeres. Aunque su mejora está en desarrollo, su estado no está listo para ser aplicado en producción. El orquestador de clústeres supone un elemento clave en la nueva arquitectura que se pretende realizar, lo que permite que únicamente aquellos nodos que hayan sido previamente subscritos al orquestador puedan ser funcionales en la arquitectura. Es por ello que convendría probar todo el despliegue en un entorno real donde el número de variadores de frecuencia y nodos edge fuese mayor.

Referencias

[1] ABB. *Qué es un variador de frecuencia: Definición, cómo funciona, características y ventajas*. [En línea]. Disponible en: https://new.abb.com/drives/es/que-es-un-variador. 2022.

[2] Juan José Rabadán Barastegu. «Diseño y desarrollo de una red MODBUS RTU basada en Arduino». Disponible en:https://idus.us.es/bitstream/handle/11441/69399/TFG_Juan%20Jos%C3%A9%20Rabadan%20Barastegui.pdf?sequence=1&isAllowed=y. Tesis de mtría. Escuela Técnica Superior de Ingeniería, Universidad de Sevilla, 2017.

[3] Grup Carol. *SIEMENS IOT2050: La puerta de enlace inteligente para soluciones industriales de IoT*. [En línea]. Disponible en: https://www.grupcarol.com/siemens-iot2050-la-puerta-de-enlace-inteligente-para-soluciones-industriales-de-iot/. 2020.

[4] Node-RED. *Node-RED*. [En línea]. Disponible en: https://nodered.org/. 2021.

[5] Kubernetes. *Nodos*. [En línea]. Disponible en: https://kubernetes.io/es/docs/concepts/architecture/nodes/. 2022.

[6] RedHat. *¿Qué es un clúster de Kubernetes?* [En línea]. Disponible en: https://www.redhat.com/es/topics/containers/what-is-a-kubernetes-cluster. 2020.

[7] Kubernetes. *Organizar el acceso a los clústeres utilizando archivos kubeconfig*. [En línea]. Disponible en: https://kubernetes.io/es/docs/concepts/configuration/organize-cluster-access-kubeconfig/. 2020.

[8] Kubernetes. *Namespaces*. [En línea]. Disponible en: https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/. 2022.

[9] Kubernetes. *Pods*. [En línea]. Disponible en: https://kubernetes.io/docs/concepts/workloads/pods/. 2022.

[10] Kubernetes. *ConfigMaps*. [En línea]. Disponible en: https://kubernetes.io/docs/concepts/configuration/configmap/. 2022.

[11] Kubernetes. *Auditing*. [En línea]. Disponible en: https://kubernetes.io/docs/tasks/debug/debug-cluster/audit/. 2022.

[12] Kubernetes. *Secrets*. [En línea]. Disponible en: https://kubernetes.io/docs/concepts/configuration/secret/. 2022.

[13] Cilium. *Introduction to Cilium & Hubble*. [En línea]. Disponible en: https://docs.cilium.io/en/stable/intro/. 2022.

[14] FIWARE. *Welcome To Orion Context Broker*. [En línea]. Disponible en: https://fiware-orion.readthedocs.io/en/master/. 2022.

[15] FIWARE. *PEP Proxy - Wilma*. [En línea]. Disponible en: https://fiware-pep-proxy.readthedocs.io/en/latest/. 2021.

[16] FIWARE. *User guide*. [En línea]. Disponible en: https://fiware-idm.readthedocs.io/en/latest/user_and_programmers_guide/user_guide/index.html. 2021.

[17] MongoDB. *What is a Document Database?*. [En línea]. Disponible en: https://www.mongodb.com/document-databases?lang=es-es. 2012.

[18] elastic. *¿Qué es Elasticsearch?*. [En línea]. Disponible en: https://www.elastic.co/es/what-is/elasticsearch. 2020.

[19] Gustavo B. *¿Qué es MySQL? Explicación detallada para principiantes*. [En línea]. Disponible en: https://www.hostinger.es/tutoriales/que-es-mysql. 2022.

[20] Rancher. *Installation Options*. [En línea]. Disponible en: https://rancher.com/docs/k3s/latest/en/installation/install-options/. 2021.

[21] Alexandre Ferreira. *smarter-device-manager*. [En línea]. Disponible en: https://gitlab.com/arm-research/smarter/smarter-device-manager. 2022.

[22] Daniel Olaogun. *Kubernetes Audit Logs | Use Cases & Best Practices*. [En línea]. Disponible en: https://www.containiq.com/post/kubernetes-audit-logs. 2022.

[23] Aaron Pejakovic. *Kubernetes — Audit logging with the elastic stack*. [En línea]. Disponible en: https://medium.com/elmo-software/kubernetes-audit-logging-with-the-elastic-stack-843b3f70683f. 2022.

[24] Kubernetes. *Certificate Signing Requests*. [En línea]. Disponible en: https://kubernetes.io/docs/reference/access-authn-authz/certificate-signing-requests/#normal-user/. 2021.

[25] Kubernetes. *Using RBAC Authorization*. [En línea]. Disponible en: https://kubernetes.io/docs/reference/access-authn-authz/rbac/#role-and-clusterrole. 2022.

[26] Cilium. *WireGuard Transparent Encryption*. [En línea]. Disponible en: https://docs.cilium.io/en/v1.10/gettingstarted/encryption-wireguard/. 2022.

[27] Cilium. *Setting up Cluster Mesh*. [En línea]. Disponible en: https://docs.cilium.io/en/v1.12/gettingstarted/clustermesh/clustermesh/#gs-clustermesh. 2022.

[28] Cilium. *Layer 3 Examples*. [En línea]. Disponible en: https://docs.cilium.io/en/v1.9/policy/language/. 2022.

APÉNDICE A

OBJETIVOS DE DESARROLLO SOSTENIBLE

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenible	**Alto**	**Medio**	**Bajo**	**No procede**
ODS 1. **Fin de la pobreza.**				X
ODS 2. **Hambre cero.**				X
ODS 3. **Salud y bienestar.**				X
ODS 4. **Educación de calidad.**				X
ODS 5. **Igualdad de género.**				X
ODS 6. **Agua limpia y saneamiento.**				X
ODS 7. **Energía asequible y no contaminante.**				X
ODS 8. **Trabajo decente y crecimiento económico.**		X		
ODS 9. **Industria, innovación e infraestructuras.**	X			
ODS 10. **Reducción de las desigualdades.**				X
ODS 11. **Ciudades y comunidades sostenibles.**				X
ODS 12. **Producción y consumo responsables.**				X
ODS 13. **Acción por el clima.**				X
ODS 14. **Vida submarina.**				X
ODS 15. **Vida de ecosistemas terrestres.**				X
ODS 16. **Paz, justicia e instituciones sólidas.**				X
ODS 17. **Alianzas para lograr objetivos.**				X

Reflexión sobre la relación del TFG/TFM con los ODS y con el/los ODS más relacionados.

La industria se encuentra en constante adaptación a las nuevas tecnologías con el fin de reducir costes, aumentar la productividad y sobretodo ahora, reducir la posibilidad de un ataque informático en sus instalaciones que puedan llevar a parar la producción, lo cual significa pérdidas económicas y de reputación. La tesis está alineada con la innovación y seguridad en la industria (ODS 9), ofreciendo así soluciones de costes reducidos que pueden evitar situaciones no deseadas. La seguridad en la industria, cada vez más importante por el uso de IoT en ella, no solo reporta avances en la confidencialidad de los datos ante un atacante externo por ejemplo, sino que es decisivo en el crecimiento económico (ODS 8) de la empresa, ya que una empresa segura es fiable y ofrece confianza al cliente, lo cual favorece este crecimiento económico.