UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

SISTEMA DE GESTIÓN DE CONTENEDORES BASADO EN TECNOLOGÍAS IoT

Rafael Vaño Garcia

Tutor: Carlos Enrique Palau Salvador

Trabajo Fin de Máster presentado en la Escuela Técnica Superior de Ingenieros de Telecomunicación de la Universitat Politècnica de València, para la obtención del Título de Máster en Ingeniería Telecomunicación

Curso 2017-18

Valencia, 1 de julio de 2018

Escuela Técnica Superior de Ingeniería de Telecomunicación
Universitat Politècnica de València
Edificio 4D. Camino de Vera, s/n, 46022 Valencia
Tel. +34 96 387 71 90, ext. 77190 • ext. 77199
Correo electrónico: poli (Consulta)
www.etsit.upv.es

VLC/ CAMPUS VALENCIA, INTERNATIONAL CAMPUS OF EXCELLENCE

EMAS Gestión ambiental verificada REG.NO. ES-CV- 000030

A mis padres por su inestimable apoyo en todo momento

A Carlos Palau por confiar en mí y darme la oportunidad de realizar este proyecto

A Benjamín y a Nacho por su ayuda y colaboración

Resumen

En este proyecto de carácter profesional se pretende realizar un sistema de gestión de las características de contenedores portuarios basado en la utilización del llamado Internet of Things. Se pretende avanzar hacia un modelo de contenedor inteligente, en el que cada contenedor tendrá una representación virtual en la nube y al mismo tiempo se generen eventos y alarmas en tiempo real con la información proporcionada por sensores instalados en el contenedor físico.

Para la realización de este trabajo se va a emplear la plataforma IoT FIWARE, financiada por la Comisión Europea. El componente principal que se va a utilizar es su context broker FIWARE Orion, necesario para gestionar los datos de los diferentes contenedores, y que se implementará en un entorno real.

El objetivo final de este proyecto es proporcionar una plataforma al usuario para que este tenga información en tiempo real sobre la mercancía que transporta un contenedor, y de esta manera acabar con la actual falta de información cuando un contenedor abandona un puerto.

Resum

En aquest projecte de caràcter professional es pretén realitzar un sistema de gestió de les característiques de contenidors portuaris basat en la utilització del nomenat Internet of Things. Es pretén avançar cap a un model de contenidor intel·ligent, en el que cada contenidor tindrà una representació virtual en el núvol, i al mateix temps es generen esdeveniments i alarmes en temps real amb la informació proporcionada per sensors instal·lats en el contenidor físic.

Per a la realització d'aquest treball es va a utilitzar la plataforma IoT FIWARE, que ha estat finançada per la Comissió Europea. El component principal que es va a utilitzar es el seu context broker FIWARE-Orion, necessari per a gestionar les dades dels diferents contenidors, y que s'utilitzarà en un entorn real.

L'objectiu final d'aquest projecte és proporcionar una plataforma a l'usuari per a què aquest tinga informació en temps reial sobre les mercaderies que transporta dins d'un contenidor, i d'aquesta manera acabar amb l'actual manca d'informació quan un contenidor abandona un port.

Abstract

In this professional project it is expected to realize a management system of the characteristics of cargo containers based on the use of the Internet of Things. The intention is to move towards an intelligent container model, in which each container will have a virtual representation in the cloud and, at the same time, the system will generate events and alarms in real time using the information provided by sensors located in the physical container.

To carry out this project, the IoT platform FIWARE will be used, which is financed by the European Commission. The main component that will be used is its context broker FIWARE Orion, which is required in order to manage the data of the different container. Orion will be implemented in a real environment.

The final goal of this project is to provide a platform for the user to have real time information about the merchandise that is being transported inside a container. By using this platform, the current lack of information when a container leaves a port will end.

Índice

Índice de figuras

Capítulo 1. Introducción

En el año 2017 un total de 8.400 millones de dispositivos se encontraban conectados a internet, lo que supone un aumento del 31.3% respecto al año anterior. Se estima que en el año 2020 un total de 20.415 millones de dispositivos estarán conectados a la red, lo que evidencia una tendencia a la conectividad de una gama cada vez más amplia gran de dispositivos, como electrodomésticos, automóviles, máquinas expendedoras, etc.

Además, se prevé que se producirá un aumento de las aplicaciones a medida para los sectores profesionales específicos, como los sensores de procesos para plantas de energía o las aplicaciones para casos de emergencias sanitarias que funcionen en tiempo real. El uso de dispositivos conectados en empresas en 2017 se sitúa en 1,6 millones de unidades, unas 300.000 unidades más que en el año 2016. Asimismo, el gasto total relacionado con la conectividad ha alcanzado los 256.340 millones de euros en 2017 [1].

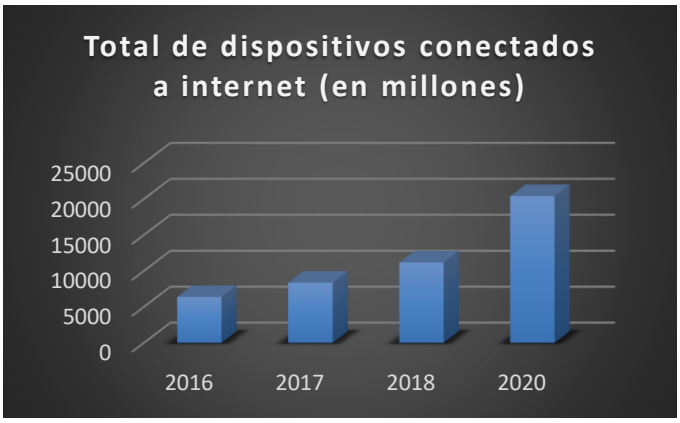

Figura 1. Evolución del número total de dispositivos conectados a internet

En este ámbito de conectividad, destaca el llamado *Internet of Things (*IoT*)*, concepto que se refiere a la conexión de elementos cotidianos a la red para tener un mayor control sobre ellos y avanzar hacia un tipo de dispositivos inteligentes. Unos claros ejemplos del *IoT* podría ser el control remoto de la temperatura en una habitación o del riego de un jardín.

Uno de los campos en que se ha empleado IoT con mayor interés y éxito, es en el dominio de aplicación de la logística y el transporte, debido a que se pretende aprovechar este mayor nivel de interconectividad de elementos para conseguir una monitorización segura, fiable y en tiempo real de los procesos del transporte de mercancías. Además, estos avances permitirán un mayor nivel de compenetración con los clientes de estos servicios, ya que podrán conocer detalles de sus productos transportados en tiempo real y sin la intermediación directa de la empresa de logística.

Es necesario destacar que el transporte marítimo representa el 90% del comercio mundial, o lo que es lo mismo, el 90% del comercio mundial se realiza a través de contenedores marítimos. Asimismo, el puerto de la ciudad de Valencia es el puerto con más tráfico de contenedores del Mediterráneo, ya que en 2017 alcanzó los 4.832.156 de contenedores (TEU) [2]. Estos datos hacen interesante la puesta en marcha de un proyecto de IoT para la gestión remota de las características de los contenedores, minimizando los posibles errores logísticos y aumentando el control sobre los productos transportados, especialmente para los contenedores refrigerados o *reefers*, ya que necesitan mantener los valores de ciertos parámetros (como la temperatura en su interior) entre unos umbrales.

Sistema de gestión de contenedores basado en tecnologías IoT

El trabajo que se va a realizar toma como objetivo la creación de un sistema de gestión de contenedores, mediante la utilización de las herramientas de la plataforma de IoT FIWARE (Orion, STH, Perseo, …) y la creación de sistemas ciberfísicos para avanzar hacia un tipo de contenedores inteligentes, teniendo cada contenedor físico su representación virtual (contenedores LXD) en un entorno cloud, donde se puedan consultar todos los datos almacenados en tiempo real relacionados con cada contenedor. Además, el sistema debe de ser fácilmente integrable en otros entornos como Smart Cities o Port Community Systems, contando con una API común para acceder a los recursos. Esta es una de las grandes diferencias con las soluciones actualmente disponibles. Adicionalmente, los datos se deberán de proporcionar a los usuarios en un formato fácilmente soportable para herramientas de Big Data Analytics.

Para la realización de este proyecto, ha sido necesaria la realización de un estudio del estado del arte actual de las tecnologías existentes, como los estándares de comunicaciones para redes del tipo LPWAN (Sigfox, NB-IoT, …), plataformas IoT existentes en el mercado (FIWARE, OpenIoT, …) y su adaptabilidad a la nube pública (Google, AWS, …), así como de los sensores que se podrían colocar en un contenedor o tecnologías ya existentes de monitorización de contenedores, siendo todas ellas tecnologías propietarias.

Finalmente, cabe destacar que este es un trabajo final de máster de orientación profesional, ya que se van a aplicar los conocimientos adquiridos durante la realización de una beca de colaboración de tipo B con el Grupo de Sistemas y Aplicaciones de Tiempo Real Distribuido de la Universitat Politècnica de València (SATRD-UPV), concretamente en el proyecto europeo de investigación INTER-IoT, a un proyecto de monitorización de contenedores portuarios basado en IoT llamado Cybercontainer. En este proyecto se cuenta con la colaboración de la empresa Infoport Valencia, que ha dado acceso a su sistema de gestión de transporte y logística para dar una mayor orientación práctica al trabajo.

Capítulo 2. Objetivos

En este capítulo se describen los objetivos que se intentan alcanzar en este Trabajo de Final de Máster.

- Introducirse a los proyectos tecnológicos profesionales.
- Realizar un estudio del estado del arte en los diferentes sensores físicos de monitorización de contenedores.
- Estudiar las diferentes soluciones de gestión de contenedores existentes en el mercado.
- Estudiar las tecnologías de comunicación más utilizadas en redes LPWAN
- Estudiar las diferentes plataformas de IoT existentes y seleccionar la más adecuada.
- Estudiar los brókeres de mensajería, los brokers de contexto y su aplicación para el intercambio de datos en un entorno IoT.
- Estudiar las características y funcionalidades de los sistemas ciberfísicos. Modelar y desarrollar el contenedor como CPS.
- Estudiar diferentes técnicas de virtualización: contenedores virtuales (Docker, LXD) para la posterior virtualización de un contenedor real.
- Desarrollar y utilizar una pasarela de comunicaciones entre los contenedores reales y virtuales.
- Instalar y configurar una plataforma FIWARE en un servidor Ubuntu real.
- Familiarizarse y realizar una gestión avanzada en entornos Linux: servidor Ubuntu, contenedores LXD y sistemas embebidos (Raspbian).
- Implementar y gestionar un entorno cloud real.
- Estudiar los siguientes componentes de la plataforma FIWARE: Orion (context bróker), STH-Comet (gestión de los datos históricos) y Perseo (CEP).
- Estudiar las bases de datos NoSQL, en concreto MongoDB, implementar la base de datos en el entorno de desarrollo y la crear diversas bases de datos MongoDB.
- Dominar el formato de intercambio de datos *JSON*, así como sus utilidades y realizar un tratamiento correcto y eficiente de estos archivos.
- Realizar un modelo de datos original y unificado para la gestión de los datos internamente y para el intercambio de datos entre los diferentes actores.
- Estudiar el lenguaje de programación Java y un entorno de desarrollo basado en este lenguaje.
- Desarrollar e implementar una API REST para que los actores externos puedan gestionar la información de los contenedores que se encuentren en la nube.
- Generar eventos de cada contenedor con una frecuencia de tiempo determinada que incluya los valores de sus sensores para la monitorización de la información.
- Generar alarmas de cada contenedor cuando se detecte que el valor de un determinado sensor haya superado unos umbrales.
- Utilizar y adaptar una gateway de comunicaciones para sistemas ciberfísicos.
- Preparar el sistema para su integración en otros entornos como Smart Cities o Port Community Systems.
- Realizar un demostrador o prototipo de laboratorio que permita comprobar los requisitos funcionales del sistema.
- Realizar una evaluación de las prestaciones del sistema desarrollado, así como un estudio de su escalabilidad.

Capítulo 3. Estado del arte

3.1 Sensores

Para medir las diferentes características de un contenedor de carga, es necesaria la colocación de varios sensores. Existen diversos tipos de sensores en el mercado, pero mediante la realización de un trabajo de investigación en el campo se han seleccionado los tipos de sensores más adecuados para la monitorización de las características de un contendor, de los que se hará una breve descripción y se mostrarán unos ejemplos de sensores reales.

3.1.1 Posición y desviación de la ruta

Uno de los aspectos más importantes en la monitorización de los contenedores de carga es sin duda el control de la posición exacta en tiempo real del contenedor. Esto permite al usuario saber dónde se encuentra en cada momento la mercancía y poder controlar el trayecto de dicho contenedor.

Otro aspecto importante que permite la monitorización de la posición del contenedor es el control de la ruta del buque que transporta el contenedor, ya que se puede detectar si ha habido una desviación en la ruta inicialmente prevista que pueda afectar al tiempo estimado de llegada al puerto (ETA).

El sistema más empleado para la monitorización de la posición es el GPS (Global Positioning System), ya que permite saber la posición de un objeto en la Tierra con una precisión de unos pocos metros mediante la comunicación con una red de 24 satélites utilizando el método de la trilateración [3]. Existen otros sistemas para la determinación de la posición, como el sistema de navegación europeo Galileo o el sistema ruso GLONASS, pero la tecnología GPS es la más utilizada actualmente. Otra posibilidad, sería la determinación de la posición mediante internet, pero es una opción menos precisa que la tecnología de determinación de la posición basada en constelaciones de satélites.

En resumen, la mejor opción es la colocación de un receptor GPS para la monitorización de la posición en los contenedores. Un módulo GPS válido podría ser el Adafruit Ultimate GPS HAT, que cuenta con una sensibilidad de -165 dBm, 66 canales, un consumo de 20mA, reloj en tiempo real (RTC) y conector para una antena externa. Este módulo GPS está diseñado para ser colocado en un sistema embebido, como una Raspberry Pi [4].

3.1.2 Temperatura

El control de la temperatura en el interior de un contenedor de carga es un aspecto muy importante, especialmente si el contenedor transporta mercancía sensible a los cambios de temperatura, como podrían ser alimentos o productos farmacéuticos.

Para la monitorización de la temperatura es necesaria la colocación de un sensor de temperatura que permita saber en cada momento la temperatura que hay en el interior de un contenedor. Actualmente, los tipos de sensores de temperatura más utilizados son el termoresistor, el sensor de unión PN y el termopar, siendo este último el más preciso.

Un termopar está formado por dos conductores diferentes que contactan en uno o más puntos, lo que produce una tensión cuando la temperatura en uno de estos puntos difiere de la temperatura de referencia en otras partes del circuito. Existen diferentes tipos de termopares, dependiendo de las combinaciones de metales que se hayan utilizado para su fabricación. El termopar más común es el de tipo K, debido a que presenta un amplio rango de temperaturas y un bajo coste [5]. El termopar de tipo K está formado un conductor positivo de níquel-cromo y un conductor negativo de níquel-aluminio. Según las tablas de referencia de un termopar tipo K, su rango de temperaturas abarca desde los -200 hasta los 1250ºC, suficiente para el rango de temperaturas que se van a medir dentro de un contenedor [6].

3.1.3 Humedad

Además de la temperatura, otro de los factores ambientales más influyente en el estado de la mercancía que transporta un contenedor es la humedad. El parámetro importante que medir es la humedad relativa en el interior, que mide la cantidad de agua en el aire del interior de contenedor en forma de vapor comparada con la máxima cantidad de agua que puede ser mantenida la temperatura en la que se ha hecho esta medición, dando como resultado un valor numérico en tanto por ciento.

Actualmente existen diversos tipos de sensores de humedad: mecánicos, por conductividad, por condensación, capacitivos, infrarrojos y de bloque de polímero resistivo. Los sensores capacitivos son los más utilizados, ya que son apropiados para ambientes con altas temperaturas y para aplicaciones que requieran un alto grado de sensibilidad a niveles bajos de humedad. Estos sensores se diseñan colocando material dieléctrico entre dos electrodos, de manera que se forma un pequeño capacitor. Las variaciones debidas a la presencia de vapor de agua en la constante dieléctrica del material dieléctrico producen una variación en el valor de la capacidad del capacitor.

El sensor SHT7x (RH/T) de Sensirion es un sensor capacitivo de tipo pin que mide la humedad relativa en el ambiente, con una temperatura de operación de -40 a 125ºC, un tiempo de respuesta de 8 segundos y una precisión con un error del ±3% en la medición de la humedad relativa [7].

3.1.4 Concentración de gases

El nivel de concentración de unos determinados gases, como podrían ser el oxígeno (O_2), el dióxido de carbono (CO_2) o el etileno (C_2H_4), puede tener un gran impacto en la mercancía del interior de un contenedor, especialmente si el contenedor transporta frutas y hortalizas. Estos alimentos siguen realizando el proceso de la respiración aún después de haber sido cortadas de la planta, consumiendo oxígeno y expulsando dióxido de carbono, esto implica que monitorizando los niveles de O_2 y CO_2 del interior del contenedor, se puede saber cómo va avanzando el proceso de maduración de los comestibles transportados.

La utilización de unos sensores capaces de medir los niveles de O_2 y CO_2 en el ambiente (en partes por millón -ppm-) son necesarios para realizar este tipo de monitorización. Otro aspecto interesante relacionado con este tipo de sensores es la posible automatización del contenedor para mantener unos niveles de O_2 y CO_2 óptimos para ralentizar el proceso de maduración de las frutas y hortalizas transportadas. Esto se conseguiría mediante la expulsión de CO_2 y la introducción de O_2 automáticamente en el contenedor.

3.1.5 Vibraciones

La detección de vibraciones fuertes es un aspecto importante en la monitorización de contenedores debido a que una vibración fuerte o la exposición prolongada a estas vibraciones puede afectar directamente a la integridad del contenido almacenado por el contenedor. Estos fenómenos se hacen especialmente sensibles si el contenido transportado son mercancías peligrosas.

Los sensores de vibraciones son acelerómetros piezoeléctricos que permiten trabajar en frecuencias altas, ya que en estas frecuencias es donde se producen las vibraciones. Los acelerómetros son unos dispositivos que permiten medir la aceleración o vibración de un objeto. La fuerza generada por la vibración hace que la masa comprima el material piezoeléctrico, lo que genera una carga eléctrica que es proporcional a la fuerza que se ha ejercido sobre el material [8]. La unidad de medida para las vibraciones son metros por segundo al cuadrado (m/s^2).

3.1.6 Impactos

Además de las vibraciones, otro factor para tener en cuenta para una mayor protección de la integridad de la mercancía transportada en un contenedor es el control de impactos. La colisión entre dos contenedores o con cualquier elemento puede ser potencialmente peligroso si se transportan mercancías peligrosas, por esto es conveniente registrar cualquier impacto para prevenirlos de manera más eficiente.

Los sensores de impacto más utilizados están formados por un circuito eléctrico simple con un conmutador, de esa forma, el circuito permanecerá abierto y solo se cerrará cuando se detecte un impacto, lo que producirá una señal eléctrica y se registrará la colisión.

Mediante el uso de unos simples sensores de impacto es posible detectar estos choques fácilmente para obtener un registro de colisiones durante un trayecto.

3.1.7 Proximidad

En el mismo orden que los sensores de vibraciones e impactos, en los contenedores que transportan mercancías peligrosas es conveniente la instalación de sensores de proximidad, pudiendo detectar la presencia de elementos en un cierto radio del contenedor y evitando posibles impactos con estos elementos, que bien podría darse el caso que fuera otro contenedor.

Un sensor de proximidad es un transductor que detecta objetos situados a una cierta distancia del transductor. Existen diversos tipos de sensores de proximidad: inductivos, capacitivos, interruptores de posición, fotoeléctricos y ultrasónicos.

Los sensores de proximidad ultrasónicos son una alternativa de bajo coste y bastante fiable, ya que pueden detectar objetos a una distancia de hasta 8 metros. Su funcionamiento es el siguiente: el sensor emite un sonido y mide el tiempo que tarda en regresar. Además, pueden detectar objetos de diferentes formas, superficies y materiales.

Un ejemplo es el sensor ultrasónico RS Pro, un sensor de bajo coste, fácil de colocar en un sistema embebido o una placa, y con un rango de detección que abarca desde los 0,2 hasta los 4 metros de distancia [9].

3.1.8 Apertura de puertas e integridad del precinto

Para garantizar la integridad de la mercancía transportada por un contenedor es crítico asegurar que no se ha manipulado la mercancía durante el transporte del contenedor. Esto se puede conseguir mediante la utilización de sensores para detectar la apertura de las puertas del contenedor, permitiendo saber los momentos exactos en que se ha abierto el contenedor.

Actualmente, existen en el mercado numerosos sensores magnéticos que detectan la apertura de una puerta. Su funcionamiento es simple, el sensor está formado por dos componentes y, cuando la puerta se encuentra cerrada, un imán crea un campo magnético sobre la otra parte del sensor. Este campo magnético desaparecerá cuando se abra la puerta.

Otra posible alternativa, sería la colocación de un sensor de luminosidad. Cuando un contenedor se encuentra cerrado, en su interior apenas hay luz, por lo que una apertura de sus puertas provocaría un cambio brusco de los valores de luminosidad (medida en luxes) que permitiría detectar la apertura de las puertas.

3.2 Tecnologías de comunicaciones para redes LPWAN

Una de las partes más importantes en el desarrollo de un sistema basado en el *Internet of Things* es la comunicación entre los diferentes dispositivos de la plataforma, especialmente la comunicación entre las redes de sensores de los contenedores y los dispositivos finales o *backends* que establecerán comunicación con la nube de la plataforma.

Para atender las necesidades de IoT se han desarrollado diversas tecnologías Low-Power Wide Area Networks (LPWANs), que se caracterizan por ofrecer conectividad inalámbrica de largo alcance a un gran número de dispositivos, menor consumo de energía y menor coste que las redes móviles, siendo su principal limitante el ancho de banda reducido.

3.2.1 LoRaWAN

LoRaWAN es un estándar de comunicaciones pensado para redes de área amplia de elementos con bajo consumo de potencia (LPWAN) desarrollado por la LoRaWAN Alliance, alianza que cuenta con miembros tan destacados en el ámbito de las comunicaciones como Cisco, Orange, IBM y ZTE. LoRaWAN define el protocolo de comunicaciones y la arquitectura del sistema para la red, mientras que la capa física LoRa es la que habilita el enlace de comunicaciones de largo alcance.

Esta especificación está dirigida claramente para su uso en el ámbito del *Internet of* Things, ya que está optimizado para sensores de bajo coste operados con batería y proporciona interoperabilidad entre diferentes dispositivos inteligentes sin la necesidad de realizar diseño e implementación de redes complejas, si no que da libertad al usuario para la realización de su propia red. Otras características importantes que ofrece LoRaWAN son una comunicación bidireccional segura, además de estar orientado a la movilidad de los dispositivos y a los servicios de localización.

➢ ***Arquitectura***

LoRaWAN utiliza una arquitectura de red en forma de estrella de estrellas en donde los gateways actúan como pasarelas transparentes entre los dispositivos finales de la red (los dispositivos inteligentes como máquinas de café o aperitivos, detectores de humo, …) y los servidores de red del backend. Los dispositivos finales se comunican directamente con los gateways mediante comunicaciones inalámbricas de un solo salto, utilizando la tecnología del espectro ensanchado para para evitar las interferencias entre las comunicaciones de diferentes dispositivos, además de utilizar un ancho de banda adaptativo (ABR) para aumentar la capacidad de los gateways.

La inteligencia y complejidad del sistema se encuentra en el servidor de red, encargado de filtrar los paquetes redundantes, realizar comprobaciones de seguridad y gestionar la velocidad de datos y la salida de RF para cada dispositivo final individualmente mediante un esquema de velocidad de datos adaptativa (ADR). Los gateways se comunican con los servidores de red utilizando comunicaciones IP estándar.

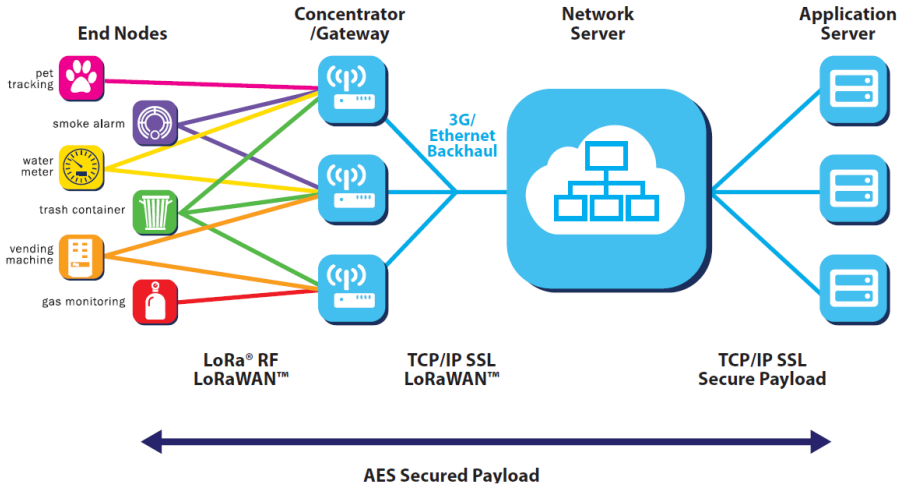

Figura 2. Arquitectura de red de LoRaWAN [11]

Cada gateway entregará el paquete recibido desde un nodo al servidor de red, transformándolo en protocolos de red de Backhaul (celular, Ethernet, satélite, Wifi, …). Además, el servidor es el responsable de los protocolos de capa superior intercambiando mensajes de señalización de control MAC / L2 / L3 con los nodos.

Una condición indispensable para que una red en forma de estrella de largo alcance sea viable, es que los gateways deben de tener una gran capacidad para recibir mensajes de un gran número de nodos. En LoRaWAN esto se consigue mediante la utilización de una tasa de datos adaptativa y mediante la utilización de un transceptor multicanal en el gateway, dando como resultado la correcta recepción de mensajes simultáneos en múltiples canales.

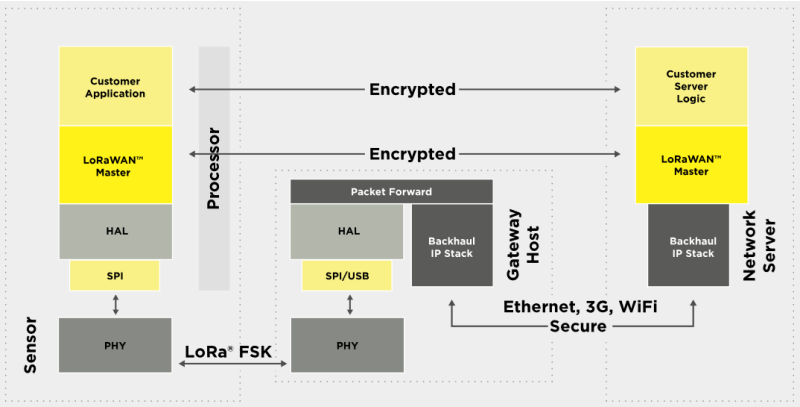

Figura 3. Pilas de protocolos de las comunicaciones en LoRaWAN [10]

Los servidores de red de la nube serán los encargados de comunicarse con la plataforma de *IoT* del usuario mediante comunicaciones TCP/IP seguras utilizando SSL [10] [11].

- ***Clases de dispositivos LoRaWAN***

Los dispositivos finales de la red no tienen las mismas características y no se van a utilizar con el mismo propósito ni para las mismas aplicaciones, por lo que LoRaWAN ha diseñado 3 diferentes clases de dispositivos que se van a describir en este apartado.

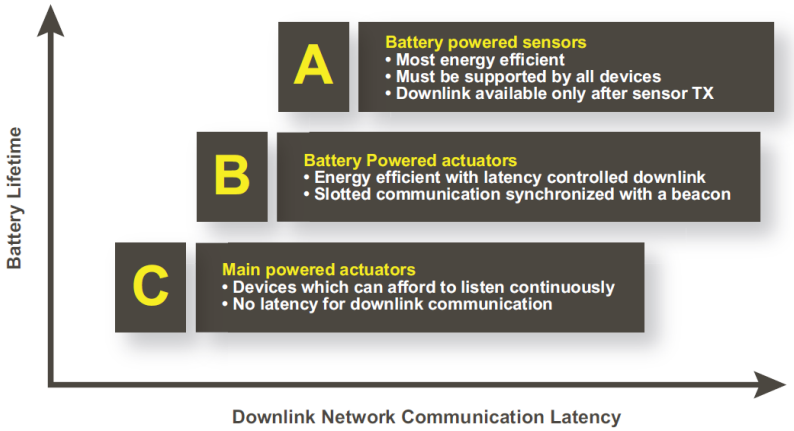

Figura 4. Clases de dispositivos de LoRaWAN [11]

- **Dispositivos de clase A:** los dispositivos finales de clase A permiten comunicaciones bidireccionales, donde una transmisión está seguida por dos pequeñas ventanas de recepción, la primera un segundo después de la transmisión y la segunda un segundo después de la primera ventana. La ranura de transmisión programada por el dispositivo final está basada en sus propias necesidades de comunicación, con una pequeña variación basada en una base de tiempo aleatoria (protocolo de tipo Aloha). Este tipo de operación de clase A es la de menor consumo de energía, porque solo espera una comunicación del servidor si el dispositivo ha transmitido antes.

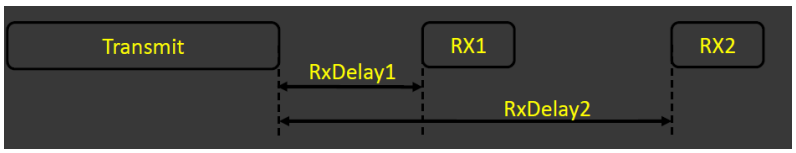

Figura 5. Ventanas de transmisión y recepción de un dispositivo de clase A [12]

- **Dispositivos de clase B:** permiten la recepción de datos sin la necesidad de transmitir con anterioridad, funcionalidad que se consigue mediante la apertura de ventanas de recepción extra en tiempos programados. Para que el dispositivo final abra su ventana de recepción en el tiempo programado, el dispositivo recibe una baliza sincronizada en el tiempo del Gateway, indicándole que abra su ventana de recepción. Finalmente, esto permite que el servidor sepa en qué momento está escuchando el dispositivo final. Estos dispositivos presentan un mayor consumo de energía que los dispositivos de clase A.

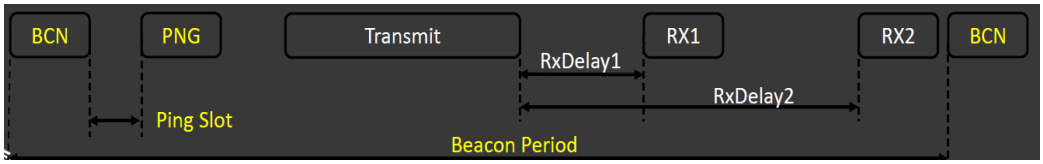

Figura 6. Ventanas de transmisión y recepción de un dispositivo de clase B [12]

- **Dispositivos de clase C:** los dispositivos de clase C se encuentran permanentemente escuchando, es decir, sus ventanas de recepción están siempre abiertas, excepto que el dispositivo esté transmitiendo. Estos dispositivos proporcionan los mejores tiempos de respuesta y capacidad de envío desde el servidor a los nodos, pero consumen mucha más energía en comparación a los dispositivos de clase A y B [11] [12].

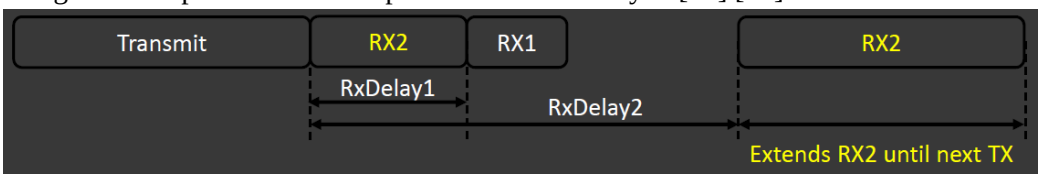

Figura 7. Ventanas de transmisión y recepción de un dispositivo de clase C [12]

➢ ***Canales de transmisión***

En LoRaWAN los nodos pueden trasmitir por cualquier canal disponible y utilizando cualquier tasa binaria, siempre que el nodo cumpla con las siguientes condiciones: debe cambiar de canal de forma pseudo-aleatoria en cada trasmisión y tiene que sobrepasar el máximo ciclo de trabajo ni el máximo tiempo de duración en la trasmisión, todo esto con relación a la sub-banda utilizada y a las regulaciones locales.

La especificación de LoRaWAN varía ligeramente entre regiones debido a las asignaciones regionales del espectro y los requerimientos regulatorios. En Europa y en Norteamérica, la especificación de LoRaWAN está definida, pero en otras zonas como en Asia aún se está definiendo.

En Europa, LoRaWAN define diez canales, ocho de los cuales son de tasa de datos múltiples desde 250 bps hasta 5.5 kbps, un solo canal de alta tasa de datos de 11kbps y un solo canal FSK a la velocidad de 50 kbps [11].

En Norteamérica, la banda ISM va desde los 902 hasta los 928 MHz, donde LoRaWAN define 64 canales de transmisión de 125 kHz desde los 902.3 a los 914.9 MHz, con incrementos de 200 Hz. Además, existen ocho canales de recepción de 500 kHz de ancho que abarcan desde los 923.3 MHz hasta los 927.5 MHz.

➢ ***Seguridad***

En los protocolos de comunicaciones es muy importante la seguridad, ya que los datos y la información que se comparten en las comunicaciones pueden ser sensibles. LoRaWAN utiliza dos capas de seguridad: una cada para la red y otra capa para la aplicación. La capa de seguridad de la red asegura la autenticidad del nodo en la red mientras que la capa de seguridad de la aplicación asegura que el operador de la red no tenga acceso a los datos del usuario de la aplicación. Para lograr estos niveles de seguridad se utiliza la encriptación AES con el intercambio de claves utilizando un identificador IEEE EUI64. Para que un dispositivo final pueda comunicarse con la red LoRaWAN es necesario que esté activado y se necesita la siguiente información [11] [12]:

- **Device address:** identificador de 32 bits, único en la red y presente en cada trama de datos. Se comparte entre el dispositivo final, el servidor de la red y el servidor de aplicaciones.
- **Network Session Key:** clave de 128 bits, única por cada dispositivo final y compartida entre el dispositivo final y el servidor de red. Proporciona seguridad en la comunicación entre el dispositivo final y el servidor de red.
- **Application Session Key:** clave de 128 bits, única por cada dispositivo final y compartida entre el dispositivo final y el servidor de aplicaciones. Se utiliza para encriptar y desencriptar mensajes de datos de las aplicaciones. Garantiza la seguridad extremo a extremo a nivel de aplicación.

3.2.2 Sigfox

La empresa francesa Sigfox ha desarrollado una tecnología destinada a su uso en el ámbito del *Internet of Things*, siendo su principal característica la comunicación entre diferentes dispositivos sin necesidad de establecer y mantener conexiones de red. Esto se consigue con una solución basada en software, donde la complejidad de la red y de la computación no se gestiona en los dispositivos, si no que se gestiona desde la nube, lo que supone un gran ahorro de energía y computación en los dispositivos.

El uso de la *Ultra Narrow Band* unido al intercambio de mensajes con tramas cortas permite a los dispositivos reducir drásticamente el consumo de energía y aumentar la vida de la batería, aspecto muy importante y a mejorar en los sistemas IoT.

➢ ***Arquitectura de red***

La arquitectura de red de Sigfox es del tipo estrella, donde se incluyen cuatro elementos:

- Los diferentes dispositivos u objetos que se quieren conectar. Necesitan incluir un módulo verificado por Sigfox para poder conectarse a la red.
- Las estaciones Sigfox.
- La nube de Sigfox: Sigfox CLOUD.
- Los servidores de red o plataforma del cliente.

Figura 8. Elementos de la arquitectura de Sigfox [13]

Los objetos emiten mensajes Sigfox de radio que son recibidos por las estaciones Sigfox que se encuentren dentro del rango. Todas las estaciones que reciban estos mensajes la transmitirán a la nube de Sigfox mediante un enlace punto a punto existente entre cada estación y la nube. Finalmente, la nube duplica estos mensajes e introduce un solo mensaje en la plataforma del cliente. Estos mensajes de transmisión o uplink pueden contener de 0 a 12 bytes de datos, utilizan modulación D-BPSK, su bitrate es de 100 o 600 bps dependiendo de la región de operación y una potencia de transmisión que puede alcanzar los 22 dBm.

La recepción de datos por los objetos está optimizada para minimizar el consumo de energía, por lo que no se pueden transmitir mensajes a los objetos en cualquier momento. El objeto debe de preguntar a la red para poder recibir un mensaje, transmitiendo un mensaje que contenga un *uplink request flag*. Este mensaje lo detectan todas las estaciones de la misma área, lo demodulan y lo transmiten a la nube. La nube transmite el mensaje a la plataforma del cliente, que detecta que no solo se está recibiendo un mensaje de uplink, sino que el objeto está preguntando si tiene que recibir un mensaje de downlink. En este momento, la plataforma del cliente decide si transmitir o no un mensaje de downlink, si decide transmitir un mensaje, lo envía a la nube y esta selecciona una estación para que lo transmita al objeto en cuestión. Estos mensajes de recepción de los objetos o downlink pueden contener 8 bytes de datos, utilizan modulación GFSK, su bitrate es de 600 bps y la sensibilidad mínima que debe de tener el receptor (en este caso el objeto) es de -132 dBm [13].

- ***Tecnología de radio***

Los dispositivos se comunican con las estaciones Sigfox mediante radiocomunicación, operando en la banda de frecuencia no comercial (ISM, de los 868 MHz a los 869 MHz y de los 902 a los 928 MHz, dependiendo de regiones), utilizando tecnología Ultra Narrow Band (UNB) combinada con las modulaciones D-BPSK y GFSK, lo que hace posible la existencia de más señales simultaneas en la banda de operación.

La modulación D-BPSK permite tener una alta eficiencia espectral, ya que posibilita coger un Hertz de la banda de operación para transmitir a un bit por segundo (1bps = 1Hz). El protocolo Sigfox utiliza modulación D-BPSK a una tasa de 100 o 600 bps (dependiendo de la región de operación) por tres razones principales: la modulación D-BPSK es fácil de implementar, una tasa binaria baja permite utilizar componentes de bajo coste y, finalmente, la estación base receptora tiene una alta sensibilidad, por lo que puede demodular señales muy cercanas al Noise Floor (NF). La sensibilidad de los receptores puede alcanzar los -134 dBm para una transmisión a 600 bps y los -142 dBm para una transmisión a 100 bps

Un dispositivo puede transmitir en cualquier lugar dentro de la banda de operación, ya que no existe ninguna sincronización entre el dispositivo y la estación base, y, por otra parte, una estación base es capaz de demodular cualquier señal de Sigfox dentro de la banda de operación. Por estas dos razones, no hay restricciones de precisión de baja frecuencia en los dispositivos.

El protocolo de Sigfox es un protocolo ligero con el objetivo de entregar mensajes de tamaño reducido. Un mensaje de uplink contiene como máximo 12 bytes de datos y un mensaje de downlink contiene solo 8 bytes de datos. Para un mensaje con 12 bytes de datos, una trama de Sigfox tendrá un tamaño de 26 bytes en total, lo que lo convierte en un protocolo con tramas de tamaño reducido [13].

Figura 9. Tamaño de una trama de Sigfox [13]

- ***Seguridad***

La seguridad es un aspecto muy importante en Sigfox y se aplica primero a los dispositivos. Existe un método de autenticación de extremo a extremo entre los dispositivos y la nube de Sigfox basado en una clave secreta. Esta clave secreta se almacena en una memoria no accesible asociada con el ID visible y específico en la memoria de solo lectura. La clave secreta se utiliza en los mensajes enviados por el dispositivo para crear una firma única para cada mensaje que autenticará al emisor. Esta firma incorpora un número de secuencia que es añadido a las tramas de radio para evitar su reemisión o reproducción.

El proceso de downlink proporciona una robustez adicional de seguridad, ya que los dispositivos eligen cuando quieren comunicarse y a qué frecuencia, lo que no permite que reciban mensajes malintencionados de un atacante.

Las estaciones bases están conectadas a la nube mediante un enlace punto a punto, utilizando una VPN encriptada mediante SSL, por lo que estas comunicaciones entre estos elementos son seguras, robustas y de confianza. Por otra parte, la nube de Sigfox está virtualizada en servidores privados alojados en data centers distribuidos en múltiples localizaciones físicas y, además, es segura, robusta y escalable. Finalmente, las plataformas de los usuarios se conectan a la nube mediante el uso de interfaces encriptados HTTPS para páginas web, APIs y callbacks.

Por último, cabe destacar que Sigfox tiene una alta resistencia a las interferencias, ya que implementa tres esquemas de diversidad para hacer las transmisiones robustas:

- **Diversidad temporal:** los datos se transmiten en tres tramas consecutivas de radio.
- **Diversidad de frecuencia:** cada una de estas tramas de radio se transmiten en un conjunto diferente de subportadora aleatoria.
- **Diversidad espacial:** cuando se transmiten estas tramas de radio no hay ningún enlace o camino establecido entre el dispositivo y la red, sino que existen varios caminos de propagación entre un dispositivo y múltiples estaciones base en el rango [13].

3.2.3 Comunicaciones machine-to-machine proporcionadas por los operadores móviles

El concepto de *machine-to-machine* (M2M) se refiere al intercambio de información entre dos máquinas remotas de una manera autónoma, fiable y eficiente. Con la popularización del *Internet of Things* y la necesidad de conectar diferentes dispositivos remotamente, se han adaptado diversas soluciones M2M para crear infraestructuras de *IoT*. Actualmente, los grandes operadores móviles, como Telefónica o Vodafone, ofrecen a sus clientes diversas soluciones M2M para sus clientes.

La idea principal es la interconexión de los diferentes dispositivos con la plataforma del cliente utilizando las infraestructuras de red (mayoritariamente la red LTE) de los operadores móviles, con conexiones directas de máquina a máquina, y sin necesidad de que los clientes tengan que desplegar y gestionar su propia red. Cada dispositivo necesita una tarjeta SIM M2M del operador para poder comunicarse en la red LTE [14].

Telefónica ofrece un servicio de M2M para las grandes empresas mediante el uso de una tarjeta SIM global M2M y una infraestructura M2M de conectividad móvil con datos, SMS y voz, utilizando su red de LTE ampliamente desplegada por todo el país. Los clientes pueden autogestionar sus comunicaciones desde una plataforma que ofrece servicios como visibilidad en tiempo real del estado y consumo de suscripciones, detección y corrección de problemas, alertas personalizadas o información sobre los dispositivos M2M conectados. Telefónica promociona este producto como parte del *IoT* y los beneficios que conlleva para las empresas. Vodafone también ofrece un producto muy similar al de Telefónica, ofertando tarjetas sim M2M para enviar datos a través de su propia red LTE.

La empresa Matona ha iniciado este año su andadura en España con el objetivo de hacerse el líder del mercado de las comunicaciones *IoT*, ofreciendo una plataforma de gestión a sus clientes totalmente orientada a las soluciones *IoT*. Con su tarjeta sim M2M multioperador los dispositivos se pueden conectar a la red en más de 180 países.

3.2.4 NarrowBand IoT

NarrowBand IoT (NB-IoT) es una tecnología de radio basada en comunicaciones celulares (LTE) para redes LPWAN estandarizada en 2016 por el 3rd Generation Partnership Project (3GPP) -asociación encargada del desarrollo de los estándares GSM, UMTS y LTE- en el marco de sus proyectos en el ámbito del *Internet of Things*. NB-IoT puede funcionar en bandas de frecuencia libres, en la banda de 200kHz actualmente no utilizada (antes ocupada por GSM) y en estaciones base de LTE, siendo necesario asignarles un bloque de recursos (RB) para las operaciones de NB-IoT en su banda de guarda. Además, permite conectar sensores directamente a las estaciones base.

NB-IoT destaca porque permite conectar una gran cantidad de dispositivos, concretamente más de 50000 por cada celda de NB-IoT, mientras minimiza el consumo de energía e incrementa el rango de cobertura en localizaciones que no están cubiertas por tecnologías de comunicaciones celulares convencionales.

Por otra parte, la mayoría de las características de LTE-Advanced como la agregación de portadoras, la conectividad dual o los servicios dispositivo a dispositivo no están soportadas por

NB-IoT. Asimismo, en esta tecnología tampoco está presente el concepto de calidad de servicio (QoS), ya que esta tecnología no está diseñada para entregar paquetes de datos sensibles al retardo.

Operadores móviles como Vodafone ofrecen un servicio de NarrowBand IoT parecido al servicio de M2M, utilizando su red de LTE ya desplegada y una tarjeta SIM de la operadora.

- ***Arquitectura de la red del núcleo***

Para entregar datos a una aplicación, se han diseñado dos optimizaciones para el Internet of Things celular (CIoT) en el Evolved Packet System (EPS): la optimización del plano de control (CP) y la optimización del plano de usuario (UP).

En la optimización del CP, los datos de uplink se transfieren desde el eNB (CIoT RAN) hasta el MME (Mobility Management Entity). En este momento, los datos se pueden transferir desde el *Serving Gateway* (SGW) hasta el *Packet Data Network Gateway* (PGW), o hasta el *Service Capability Exposure Function* (SCEF) que es el único destino posible para los paquetes no IP. Desde estos nodos se entregan finalmente al servidor de aplicaciones (CIoT Services). Los datos de downlink se transmiten por el mismo camino, pero en sentido contrario.

El SCEF es un nuevo nodo diseñado especialmente para los datos de tipo máquina. Se utiliza para entregar paquetes no IP por el plano de control, y proporciona un interfaz abstracto para los servicios de red, como la autenticación y la autorización.

Respecto a la optimización del UP, los datos se transfieren sobre portadoras de radio a través del SGW y el PGW hasta el servidor de aplicaciones. Este camino soporta paquetes IP y no IP.

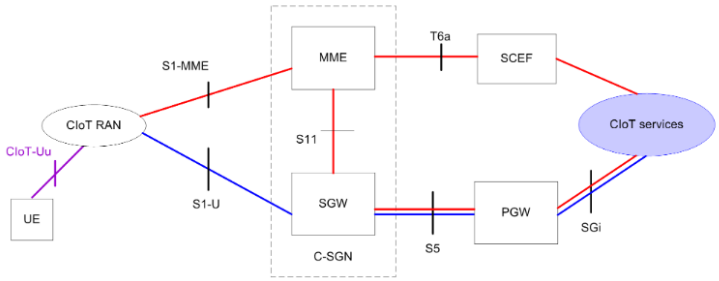

Figura 10. Arquitectura de red del núcleo de NB-IoT. En rojo se muestra el plano de control y en azul el plano del usuario [16]

- ***Arquitectura de la red de acceso***

La arquitectura de la red de acceso no presenta ninguna diferencia respecto a la arquitectura de LTE. Los eNBs (evolved Node B) están conectados al MME y el S-GW utilizando el interfaz S1. Aunque no hay definido ningún handover, sigue existiendo un interfaz X2 entre dos eNBs [16].

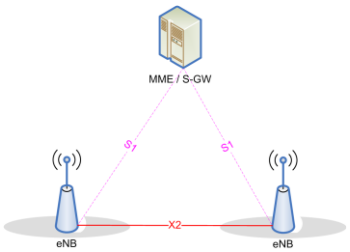

Figura 11. Arquitectura de la red de acceso de NB-IoT [16]

- ***Diseño de radio***

El diseño del interfaz de radio de NarrowBand IoT se deriva de LTE. La portadora de NB-IoT tiene un ancho de banda de 180 kHz con soporte para operar con multiportadora. En el enlace descendiente (downlink), se utiliza el acceso múltiple por división de frecuencia ortogonal (OFDMA) con un espaciado de subportadora de 15 kHz sobre 12 subportadoras con 14 símbolos utilizados para abarcar una subtrama de 1ms. En el enlace ascendente (uplink), se aplica el acceso múltiple por división de frecuencia de portadora única (SC-FDMA), con un espaciado de subportadora de 3.75 o 15 kHz.

NB-IoT define tres modos de operación para proporcionar flexibilidad:

- Autónoma: utilizando, por ejemplo, una o más portadoras GSM.
- Banda de guarda: utilizando los bloques de recursos (RB) no utilizados con una banda de guarda de portadora LTE.
- En banda: utilizando bloques de recursos con una portadora LTE.

Adicionalmente, NB-IoT utiliza el concepto de repeticiones y técnicas de combinación de la señal para mejorar la zona de cobertura. Para servir a los equipos de usuario (UE) en diferentes condiciones de cobertura que tienen diferentes rangos de pérdidas del camino, puede haber hasta tres configuraciones de mejora de cobertura (CE) en el acceso aleatorio con sus configuraciones específicas. Después de esto, el eNodeB selecciona la configuración de los recursos de radio, la modulación y el esquema de codificación (MCS) y las repeticiones dependiendo de la cobertura de los UEs [15].

- ***Eficiencia energética***

NB-IoT está diseñado para que los dispositivos tengan una larga vida útil, cuya batería pueda pasar de los 10 años de vida. Para alcanzar este fin, NB-IoT reutiliza los mecanismos de ahorro de energía de LTE, pero entendiendo los temporizadores involucrados para alcanzar una vida útil de la batería mayor. En LTE hay dos mecanismos claves para el ahorro de energía: Recepción Discontínua (DRX) y Modo de Ahorro de Energía (PSM). Ambos mecanismos varían la manera en la que el UE se comunica con la red. Esta comunicación requiere una Radio Resource Conection (RRC) establecida entre el UE y el eNB. Hay dos posibles estados: conectado y en reposo, asimismo, un UE en un estado de RRC conectado tiene una conexión RRC activa. El UE puede pasar de estar en modo RRC conectado a estar en modo RRC en reposo por diferentes causas, como la inactividad del UE o que este UE se desconecte. Para detectar inactividad en los UE, el eNB utiliza unos temporizadores de inactividad que se resetean después del envío de un paquete de datos [15].

3.3 Plataformas IoT

Actualmente, según el Internet of Things (IoT) Platform Market Report 2017-2022 [17] existen más de 360 plataformas de *IoT* en el mercado, por lo que es imposible describir cada una de estas plataformas. Por lo tanto, en este apartado se van a describir algunas de las plataformas *IoT* de código abierto más utilizadas, haciendo hincapié en FIWARE, ya que es la plataforma que se va a utilizar en este trabajo.

3.3.1 SOFIA2

SOFIA (Smart Objects For Intelligent Application) es una arquitectura de middleware que permite la interoperabilidad entre diversos dispositivos y sistemas desarrollada en un proyecto en el que participaron empresas como Nokia, Philips, Fiat, Acciona e Indra. SOFIA es una plataforma de código abierto, multiplataforma (Windows, Linux, Android, …), multilenguaje (Java, Javascript, Arduino, …) y tiene implementaciones para diversos protocolos de comunicación como TCP, HTTP (Rest y servicios web), Ajax Push, etc.

Más tarde, Indra mejoró este proyecto creando una plataforma llamada SOFIA2 que se focalizaba en el uso empresarial, añadiendo características como interfaces de Big Data, capacidades de integración con back-ends mediante protocolos estándar o interfaces REST para poder conectarse desde smartphones y otros dispositivos.

La arquitectura de SOFIA2 está formada por diversos elementos que se van a describir a continuación. El Smart Space es el entorno virtual donde diferentes aplicaciones interactúan con otras aplicaciones para proporcionar una funcionalidad compleja, y además se pueden comunicar con otros Smart Space mediante el establecimiento de relaciones de confianza. El núcleo de un Smart Space es el Semantic Broker Information (suele haber un SIB en cada Smart Space, pero puede tener varios para comunicar diferentes ontologías, que son descripciones semánticas de un conjunto de clases y atributos con el objetivo de representar objetos), que es el encargado de recibir, procesar y almacenar toda la información de las aplicaciones conectadas a la plataforma. Cada SIB está diseñado para una sola ontología, definida por una estructura JSON.

Por otra parte, el Knowledge Processor (KP) permite la conexión de cada aplicación con el Smart Space mediante el SIB, permitiendo la ejecución de operaciones con el SIB. Existen tres tipos de KP:

- Productor: introduce información en el SIB.
- Consumidor: extrae información del SIB.
- Prosumidor: introduce información en el SIB y la extrae de éste.

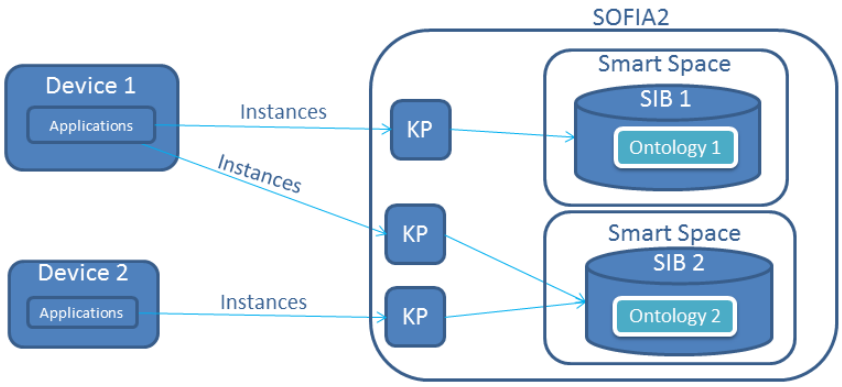

Figura 12. Conceptos principales de SOFIA2 [17]

Para establecer conexión desde un cliente se necesita un KP name, un identificador de KP y un token. El Smart Space Access Protocol (SSAP) es el lenguaje estándar que se utiliza en la comunicación entre los SIBs y los KPs, que a su vez utiliza un formato JSON.

Además, SOFIA2 incluye herramientas para el análisis del Big Data, como el almacenamiento de los datos en tiempo real y de un histórico de estos datos, una herramienta visual (SOFIA2

DATAFLOW) para el control de los datos que se introducen en la plataforma o herramientas de machine learning para el tratamiento de los datos.

3.3.2 Open IoT

Open IoT es una plataforma de middleware de código abierto para la integración de IoT y de la nube, siendo su principal objetivo permitir una configuración flexible y el despliegue de algoritmos para la recolección y el filtrado de datos transmitidos por dispositivos conectados a internet. La plataforma permite la integración de servicios de IoT dispersados administrativamente y geográficamente. También es necesario destacar que Open IoT permite proporcionar servicios de IoT de pago inmediato bajo demanda como "Sensing-as-a-Service" o sensorización como servicio.

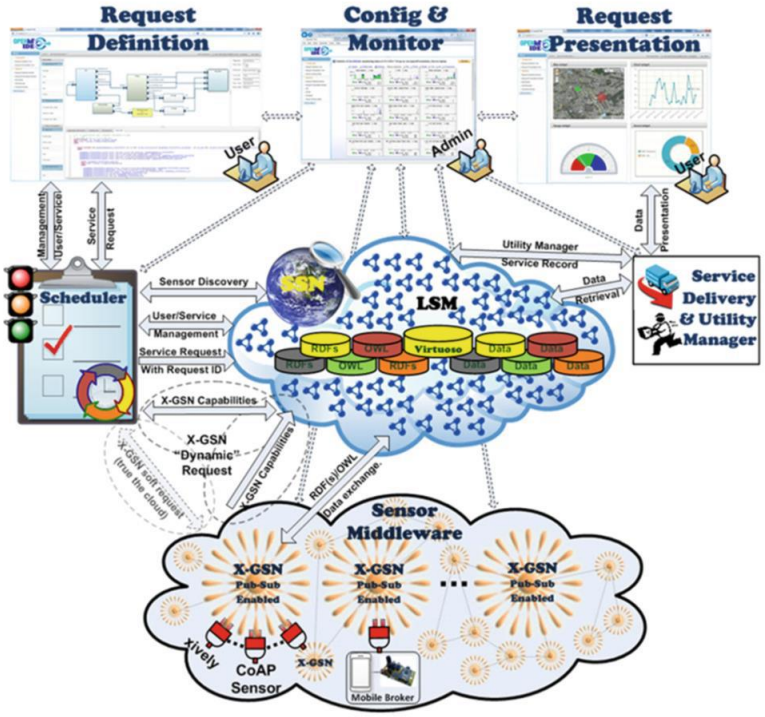

Figura 13. Componentes principales de la plataforma OpenIoT [17]

El Extended Global Sensor Network **(**X-GSN) actúa como un hub entre la plataforma IoT y los dispositivos físicos: recolecta, filtra y combina los flujos de datos de los dispositivos físicos y de los sensores virtuales. Estos datos se almacenan en una base de datos en la nube mediante una plataforma semántica de middleware, donde se utilizan datos anotados para almacenar los datos y los metadatos del sensor como triples en formato RDF. Además, se utiliza un clasificador (Scheduler) para asegurarse de que cada servicio demandado utilice los recursos que necesita.

El Service Delivery & Utility Manager tiene dos funcionalidades. Por una parte, combina los flujos de datos como lo indican los flujos de trabajo del servicio dentro del sistema OpenIoT para entregar el servicio demandado, haciendo uso de la información proporcionada por el Scheduler. Por otra parte, este componente actúa como servicio de medición que mantiene el seguimiento de medidas para cada servicio individual. Esta funcionalidad puede ayudar en los cobros por el uso del servicio y en la optimización de los recursos.

La plataforma cuenta con interfaces gráficas de usuario para realizar peticiones de servicio (Request Definition) y para visualizar los resultados de los servicios utilizados (Request Presentation).

3.3.3 UniversAAL IoT

UniversAAL IoT es una plataforma de código abierto que habilita la integración continua de dispositivos, servicios y aplicaciones a gran escala, implementando una interoperabilidad semántica para arquitecturas orientadas a servicios en el nivel de los protocolos de comunicaciones. El núcleo de la plataforma es un middleware que entiende los datos y la funcionalidad solo a través de la definición de ontologías. Este middleware es básicamente un facilitador del intercambio de mensajes entre componentes autónomos.

Uno de los procesos más importantes de UniversAAL IoT es el descubrimiento de la semántica. El primer nivel de descubrimiento está relacionado con el descubrimiento mutuo de las instancias de middleware en una subred, que está basado en el broadcast de una red estándar. Cuando se descubre un nodo, empieza la fase de emparejamiento para comprobar la compatibilidad y la autorización. Si se produce el emparejamiento, se establece una comunicación basada en el concepto de un *conector*, que habilita la creación de una red lógica llamada uSpace, cuya finalidad es la distribución de mensajes de una manera segura entre nodos de esa misma red.

Los mensajes se clasifican de acuerdo con unos patrones de comunicación, lo que se traduce en la existencia de diferentes *buses*, cada uno para atender un propósito específico. El segundo nivel de descubrimiento es el descubrimiento de la semántica, proporcionado por los buses de middleware. Los módulos que se conectan a los buses deben de describir semánticamente sus capacidades y requerimientos, y posteriormente registrarlos con los buses. Los buses se basan en un emparejamiento semántico para enrutar mensajes con contenido formulado semánticamente, dando como resultado el descubrimiento del proveedor (de funcionalidad) o del consumidor (de datos) adecuado.

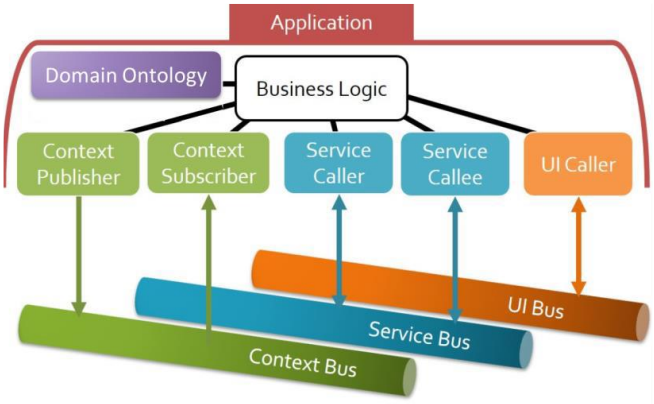

Figura 14. Esquema de funcionamiento de la plataforma UniversAAL IoT [17]

La abstracción del bus es el concepto principal detrás del paradigma de la interoperabilidad semántica tal como lo realiza UniversAAL.

3.3.4 FIWARE

FIWARE es una plataforma abierta de middleware para el Internet of Things, respaldada por la Comisión Europea dentro del Future Internet Public Private Partnership Programme. Esta plataforma proporciona especificaciones de API públicas y protocolos interoperables para la creación de nuevos servicios y aplicaciones de internet. FIWARE proporciona habilitadores genéricos (Generic Enablers -GE) que cubren diferentes áreas como el IoT, el tratamiento de datos, la seguridad, interfaz de usuario avanzada o la nube.

Esta plataforma permite la portabilidad de datos para diferentes aplicaciones y proporciona un conjunto de GEs que permite el acceso a información relevante a través de la API NGSI, lo que se traduce en un camino rápido y eficiente para recoger, publicar, intercambiar y procesar grandes cantidades de datos. La gestión de datos de FIWARE permite la generación de información de contexto, la gestión de los cambios del contexto a través de eventos y el procesado de grandes cantidades de información utilizando técnicas de Big Data.

Además, FIWARE proporciona soporte para una infraestructura de hosting en la nube a través de GEs que son accesibles a través de REST APIs. La arquitectura de la nube de FIWARE incluye diversos servicios, como servicios IaaS (Infrastructure as a Service): cloud computing, cloud networking, …; servicios de almacenamiento de datos y metadatos en la nube, y servicios de gestión y monitorización de aplicaciones y dispositivos.

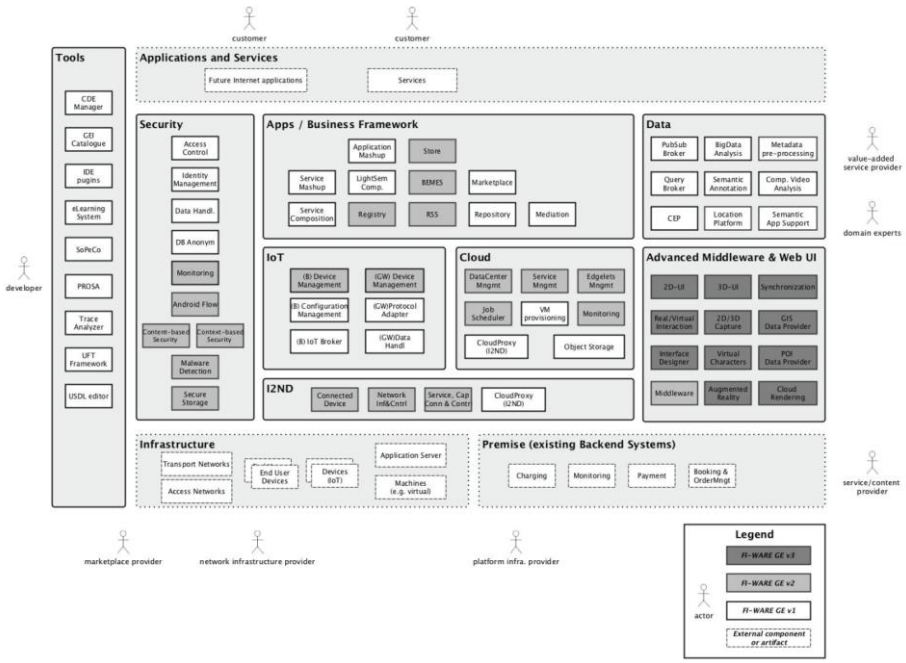

Figura 15. Arquitectura de FIWARE con los Generic Enablers [17]

En FIWARE, el GE de procesado de datos está dividido en dos bloques funcionales. Por una parte, el bloque de procesado por lotes proporciona infraestructuras bajo demanda que soportan software de análisis de Big Data, como clusters de Apache Hadoop, ya que en la arquitectura de FIWARE la infraestructura de procesado de datos es independiente del servicio de almacenamiento de datos. Por otra parte, el bloque de procesado de flujos proporciona una interfaz modular para el procesado de datos en tiempo real utilizando Apache Storm [17].

- ***FIWARE Orion Context Broker***

El Context Broker es el encargado de gestionar la comunicación entre aplicaciones y dispositivos o gateways con la finalidad de desacoplar las aplicaciones de IoT de las instalaciones de los dispositivos. El Context Broker se comunica con los dispositivos y las aplicaciones de IoT mediante el protocolo NGSI (actualmente utiliza la versión NGSIv2) y permite la generación y la recepción de notificaciones de eventos contextuales.

Los datos se almacenan en unidades llamadas entidades (entities). Cada entidad se identifica mediante un id único y pertenece a un tipo (type), al que pueden pertenecer varias entidades. Además, una entidad puede tener unos atributos (atributes), que describen información de la entidad, mediante un valor (value), un tipo de datos (type) y unos metadatos (metadata) opcionales a introducir por el usuario. Es necesario destacar que solo se almacena el último valor de cada atributo de las entidades. Estos datos se almacenan en una base de datos NoSQL del tipo MongoDB.

Las entidades se pueden clasificar mediante un sistema jerárquico, donde el nivel principal es un *Fiware-Service*, que puede contener niveles inferiores en forma de árbol, cuya ruta se indica con el llamado *Fiware-ServicePath*.

En Orion existen dos diferentes roles: productores y consumidores. Los productores se encargan de crear y actualizar los datos de las entidades, mientras que los consumidores se suscriben a Orion para que éste les notifique los datos de las entidades a las que se han suscrito previamente [19].

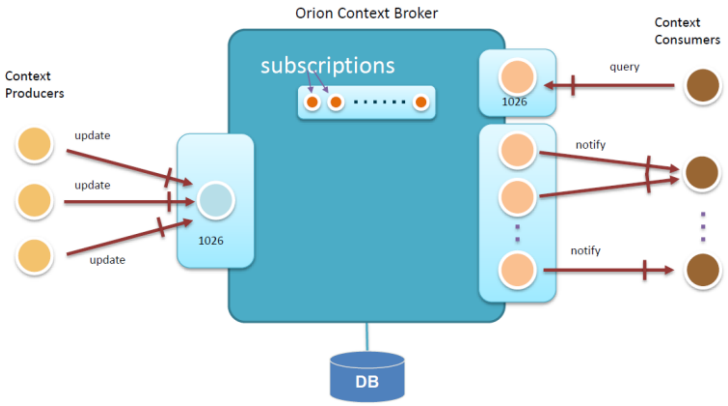

Figura 16. Diferentes roles en Orion Context Broker [19]

- ***FIWARE Short Time Historic - Comet***

El FIWARE STH se encarga de almacenar y gestionar la información histórica de las entidades previamente registradas por Orion en la base de datos. Es necesario porque en la base de datos de Orion solo se almacena el último valor de los atributos de cada entidad, por lo que no es posible consultar datos históricos mediante consultas a Orion.

Para poder almacenar los datos históricos de una entidad, el STH debe suscribirse a esa misma entidad de Orion. De esta forma, cuando se produce un cambio en los valores de los atributos de esta entidad, mediante la suscripción realizada por el STH se le notificarán los datos de dicha entidad y los almacenará en su propia base de datos de MongoDB.

Cuando un usuario quiere obtener una serie de datos históricos, existen diversos filtros (por tipo, durante un periodo de tiempo concreto, …) para que el usuario pueda gestionar de una mejor forma los datos que proporciona el STH [20].

- ***FIWARE Complex Event Processing (CEP): Perseo***

El CEP Perseo es un "motor de reglas distribuido" [21], que analiza los datos de las entidades del Orion Context Broker para comprobar si se han cumplidos unas reglas concretas, anteriormente indicadas por el usuario. Si Perseo detecta que un valor de un atributo de una entidad cumple cierta regla, generará un evento como el envío de un correo electrónico o el envío de una petición HTTP (GET, POST, …) a una url con un cuerpo de la petición indicado por el usuario.

El usuario debe de crear sus propias reglas mediante una petición POST, incluyendo en el cuerpo de la petición la regla en el lenguaje de reglas de Esper, que tiene un formato muy parecido al lenguaje de SQL [21].

3.4 MongoDB

MongoDB es una base de datos gratuita y de código abierto del tipo NoSQL, ya que está orientada a documentos. MongoDB almacena los datos en forma de documentos dinámicos con un formato similar a JSON llamado BSON (Binary JSON), lo que permite que la estructura de datos pueda variar con el tiempo y entre diferentes documentos.

Una única base de datos puede contener diversas colecciones, que son las que almacenan los documentos. Cabe destacar que cada documento tiene su propio identificador interno (campo "_id"), que es creado automáticamente cuando se crea el documento y no puede ser modificado.

MongoDB tiene una consola de comandos propia escrita en Javascript, lo que permite al usuario una fácil gestión de sus bases de datos mediante el uso de funciones de Javascript cuyo uso puede encontrar en la documentación oficial. Asimismo, existen librerías de clientes de MongoDB para diversos lenguajes de programación como Python, Java, C++, C# y Javascript [22].

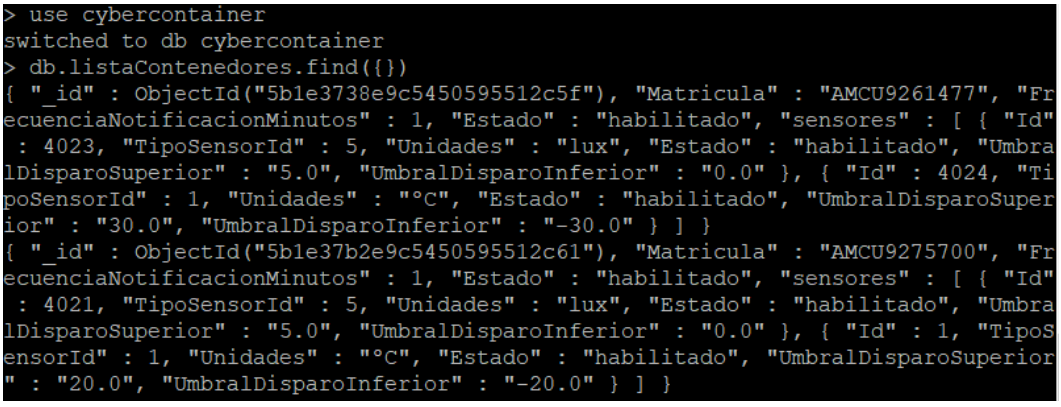

Figura 17. Ejemplo de una consulta y sus resultados en una consola de MongoDB

3.5 Sistemas ciberfísicos

Los sistemas ciberfísicos (cyber-physical systems – CPS) se obtienen a partir de integrar capacidades de computación, computación y procesos físicos en los objetos físicos del sistema, para avanzar hacia un tipo de objetos inteligentes. Están compuestos por diferentes partes que colaboran conjuntamente para alcanzar cierto comportamiento global como si de un solo sistema se tratara. Estas partes pueden ser tecnologías de comunicaciones, sistemas embebidos, sensores, actuadores o sistemas de software [23].

En un sistema ciberfísico, cada objeto físico tiene su propia representación virtual. Por esta razón, se suelen utilizar objetos físicos de tamaño reducido, con bajo consumo y baja capacidad de computación (como los sistemas embebidos), ya que se aprovecha la capacidad de computación del objeto virtual para realizar los procesos más pesados, que en un sistema normal deberían realizar los objetos físicos. Un ejemplo, serían los contenedores reales y virtuales de este proyecto.

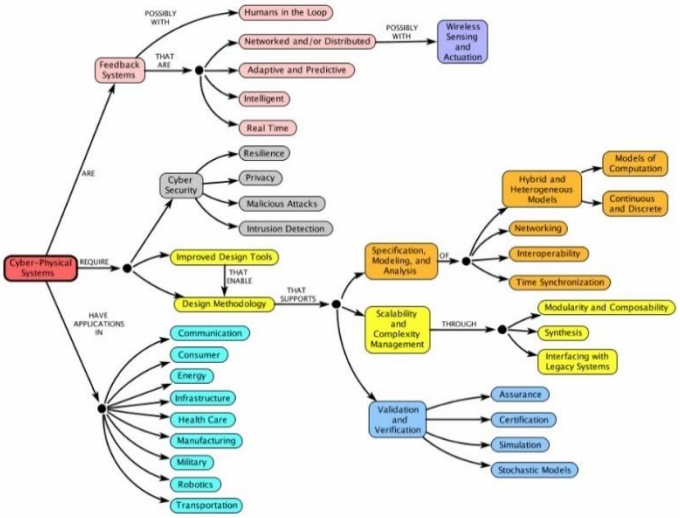

Figura 18. Mapa conceptual de los sistemas ciberfísicos [23]

3.6 Opciones para la virtualización del contenedor

Los contenedores son una tecnología que permite al usuario dividir una máquina para que pueda ejecutar más de una aplicación (en el caso de los contenedores de proceso, como Docker) o instancias del sistema operativo (en el caso de contenedores de máquinas, como LXD) en el mismo kernel y hardware de la máquina anfitriona. Además, esto se debe de hacer manteniendo el aislamiento entre estas cargas de trabajo [31].

Las máquinas virtuales (VMs) son una abstracción del hardware físico que convierten un servidor en múltiples servidores. El hipervisor es el encargado de permitir que múltiples máquinas virtuales se ejecuten en una sola máquina física. Cada VM incluye una copia completa de un sistema operativo, por lo que pueden ser costosas de arrancar [25].

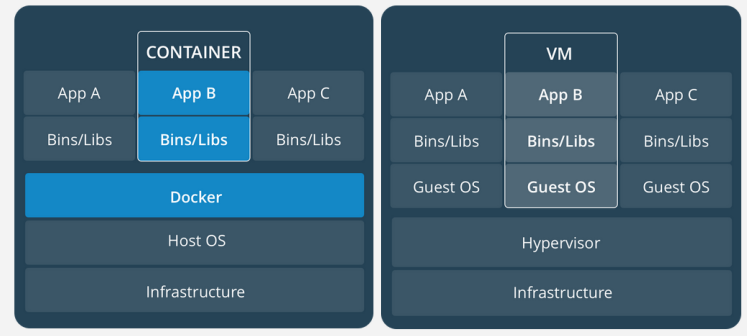

Figura 19. Comparación entre la estructura de un contenedor (izquierda) y de una máquina virtual (derecha) [25]

3.6.1 Docker

Docker es un proyecto de código abierto que permite al usuario automatizar el despliegue de aplicaciones dentro de contenedores de procesos. Docker permite que un pequeño subconjunto de un sistema de archivos (solo los archivos necesarios para que una aplicación se pueda ejecutar) se pueda integrar en un template, y, luego, hace posible que esto se implemente, repetidamente, en un contenedor ligero. La aplicación está empaquetada de manera que esté separada del sistema operativo en el que se va a ejecutar. Por ejemplo, un contenedor construido en RHEL también se puede ejecutar en Ubuntu.

Docker sigue el principio de la infraestructura inmutable, es decir, cada contenedor es igual que su template cuando se lanza, por lo que si se reinicia un contenedor se pierden todos los cambios que se han realizado. Además, Docker utiliza un archivo de configuración de texto sin formato, llamado *Dockerfile*, para describir los elementos del interior de un contenedor, y permite producir nuevos templates a partir de templates existentes y archivos nuevos.

Docker tiene una manera poco común de llevar a cabo las conexiones entre los contenedores. Cada contenedor no es una imagen del sistema operativo independiente en toda regla, por lo que no necesariamente tiene su propia dirección IP. Más bien, cada contenedor ejecuta uno o más procesos, cada uno de los cuales estará escuchando en un puerto. En el archivo *Dockerfile* se puede configurar cuál de estos puertos tiene que ser "expuesto", es decir, mapeado a un puerto del host mediante una traducción. Sin embargo, en versiones nuevas de Docker se puede asignar a cada contenedor una dirección IP propia y tener interfaces de red mapeadas a puentes o a redes superpuestas.

Cabe destacar que Docker utilizaba con anterioridad LXC para construir sus contenedores, pero actualmente utiliza *libcontainer*, lo que proporciona una funcionalidad similar [24] [25].

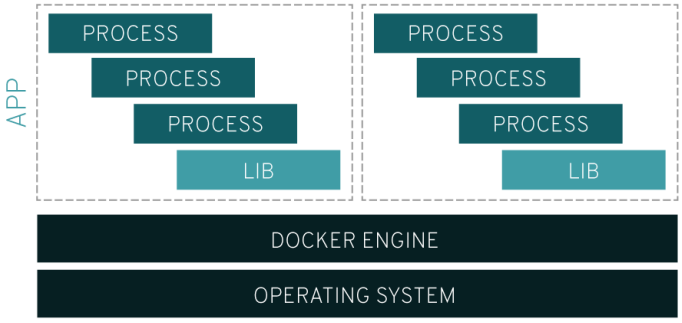

Figura 20. Contenedores Docker [24]

3.6.2 LXD

LXD es una herramienta de código abierto que proporciona un medio para gestionar contenedores de máquinas (machine containers). Ofrece al usuario una experiencia similar a las máquinas virtuales, pero utilizando en su lugar contenedores de Linux.

Es necesario destacar que LXD no es una reescritura de LXC, sino que se construye sobre LXC para proporcionar una mejor experiencia de usuario, y utiliza LXC a través de *liblxc*. En resumen, se trata de una alternativa a las herramientas y el sistema de templates de LXC con las características adicionales que provienen de ser controlables a través de la red.

LXD está formado por tres componentes:

- Un Daemon en todo el sistema, llamado *lxd*, que realiza todo el trabajo pesado. Proporciona una API REST para su utilización local, que también puede ser exportada si la red lo necesita. Además, este Daemon se comunica con *liblxc* para crear los propios contenedores.
- Una línea de comandos, llamada *lxc*, pero que se utiliza para manipular los contenedores LXD, no los LXC. Ésta se comunica con el Daemon mediante la API REST.
- Un plugin de OpenStack (Nova LXD) que permite a OpenStack utilizar LXD como hipervisor, por lo que OpenStack podrá crear instancias en LXD de la misma manera que normalmente crearía máquinas virtuales que se ejecutan en un hipervisor tradicional como KVM.

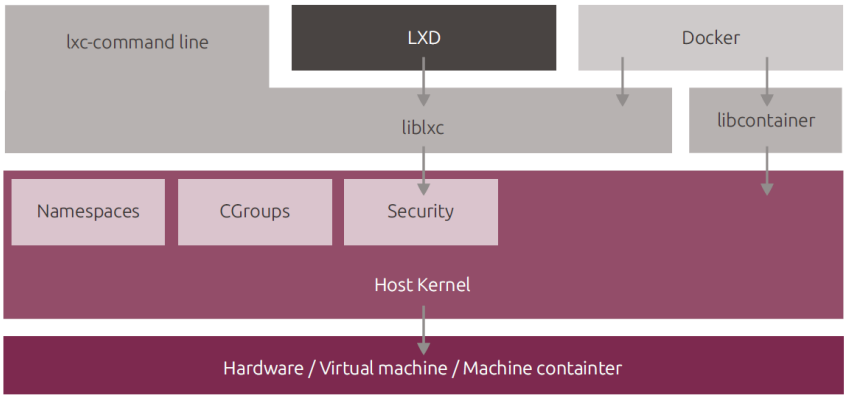

Figura 21. Contenedores LXD en Linux [27]

Las principales características de LXD se describen a continuación:

- Altamente escalable
- Seguridad: por defecto se crean contenedores sin privilegios, por lo que cualquier usuario (incluso los no root) pueden lanzar contendores. A partir de ahí, se proporcionan unos mecanismos de restricción para otros usuarios y aislarlos del resto del sistema.
- Intuitivo: proporciona una API clara y simple y una línea de comandos nítida.
- Basado en imagen, con una gran variedad de distribuciones Linux disponibles. Esto significa que cada contenedor tiene un sistema de archivos completo, a diferencia de los contenedores de proceso, que tienen una composición de templates.

- Compatible con la transferencia de contenedor e imagen de hosts cruzados (incluida la migración en vivo con CRIU).
- Control de recursos avanzado: cpu, memoria, utilización del disco, recursos del kernel, …
- Gestión de la red: creación y configuración de puentes, túneles de hosts cruzados, …
- Gestión del almacenamiento: soporte para múltiples backends de almacenamiento, grupos de almacenamiento y volúmenes de almacenamiento [25] [26].

3.7 Soluciones existentes de sistemas de tracking de contenedores

Actualmente, existen en el mercado diversas soluciones para monitorizar en tiempo real aspectos de contenedores de carga como su temperatura, su posición o la integridad de su contenido, por lo que se van a describir algunas de estas soluciones en este apartado. Cabe destacar que todas estas soluciones son propietarias, y por lo tanto solo funcionan con elementos de la misma empresa que ha desarrollado cada solución.

3.7.1 Remote Containers Management de Maerks Line

La solución Remote Containers Management (RCM) de la empresa danesa de envío marítimo de contenedores Maerks Line, permite monitorizar las condiciones interiores de contenedores refrigerados durante el trayecto del envío de la mercancía.

Los contenedores refrigerados incluyen una serie de sensores para monitorizar el estado de la mercancía en tiempo real, ya que miden los diferentes parámetros que son críticos para poder evaluar el estado de los productos refrigerados: la temperatura, la humedad y el nivel de CO_2 del interior del contenedor. Además, se pueden predefinir unos valores de umbral para estos parámetros con la finalidad de enviar notificaciones si los sensores detectan unos valores no óptimos de estos parámetros, para que el propietario de la mercancía pueda actuar y evitar la pérdida de calidad de ésta.

Otro aspecto importante de esta solución es que estos contenedores proporcionan información de su posición en tiempo real, ya que utilizan la tecnología GPS. Por lo tanto, el cliente puede conocer en cualquier momento la posición de su mercancía y exactamente cuándo llegará a su destino.

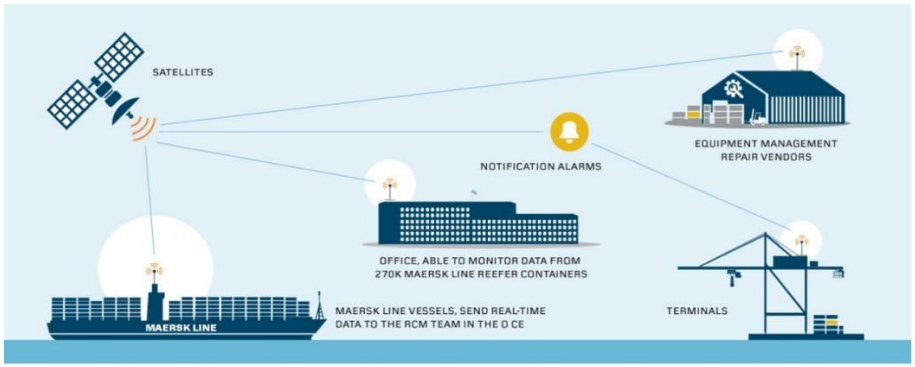

Figura 22. Esquema de funcionamiento de la solución de Maerks Line [28]

Los datos que recogen los sensores interiores de estos contenedores inteligentes se envían en tiempo real por comunicaciones de satélite. Los contenedores de un mismo buque transmiten estos datos a un gateway que se encuentra en el mismo buque, para que finalmente este gateway reenvíe los datos a un satélite. Seguidamente, el satélite se encargará de retransmitir estos datos a la plataforma cloud de Maerks Line. En las oficinas de Maerks Line se tiene capacidad para monitorizar datos de los más de 270000 contenedores de la empresa que se encuentren en una ruta [21].

3.7.2 Smart container de Loginno

La empresa israelí Loginno ha desarrollado un contenedor inteligente con el uso del IoT. Este contenedor permite la monitorización de sus características en tiempo real: mide la temperatura de su interior, proporciona información de su posición, es capaz de detectar los choques del contenedor y si se han abierto sus puertas durante el trayecto, lo que podría afectar a la integridad de la mercancía que transporta. Utilizando esta información, este contenedor inteligente proporciona notificaciones al cliente tanto si detecta irregularidades como si ha habido intrusiones (se ha abierto sin autorización), si se ha desviado de su ruta prevista o si ha habido un retraso respecto a su tiempo estimado de llegada.

Loginno proporciona una interfaz de usuario amigable para poder controlar todos los envíos de contenedores de un mismo cliente y recibir las notificaciones de los contenedores de manera correcta, asimismo, proporciona una app para smartphones para una mayor comodidad en la recepción de alertas y notificaciones. Otro aspecto importante, es que proporciona una API para poder utilizar estos datos en una plataforma propia del cliente [29].

3.7.3 Smart container de Traxens

La empresa francesa Traxens ha desarrollado una tecnología para crear contenedores inteligentes que permitan que el propietario del contenido transportado en el contenedor tenga una visibilidad en tiempo real de las condiciones interiores del contenedor y pueda recibir notificaciones y alertas de eventos clave cuando estos se produzcan. Además, proporciona una plataforma de big data para que el usuario tenga un mayor control sobre los datos de los contenedores.

La solución de contenedores inteligentes de la empresa francesa Tranxens está formada por tres elementos: Traxens-Box, Traxens-Net y Traxens-Hub [30].

- ***Traxens-Box***

El Traxens-Box es un dispositivo de monitorización que está permanentemente pegado a un contenedor y que se puede utilizar en todos los tipos de contenedores: refrigerados, tanques de líquido y secos o estándar. La funcionalidad de este dispositivo es la recolección de datos de posición GPS, temperatura, impactos, movimiento y vibración, y su posterior transmisión a un Traxens-Hub, que es la plataforma de datos de Traxens. Del mismo modo, también es posible conectar otros tipos de sensores (humedad, presión, …) a un Traxens-Box de manera inalámbrica conectándolos a la red de Traxens, la Traxens-Net.

Por otro lado, para el correcto funcionamiento de un Traxens-Box en un contenedor refrigerado, es necesario utilizar una unidad adicional llamada Traxens-Link y unos cables para conectar esta unidad al controlador del refrigerador. Además, este sistema hace posible el envío de datos al controlador del refrigerador para cambiar los parámetros de éste, pero siempre bajo un estricto nivel de seguridad [30].

- ***Traxens-Net***

La Traxens-Net es la red propia de Traxens: es una red de radio mallada diseñada específicamente para alcanzar un ratio óptimo entre el funcionamiento y el consumo de los dispositivos para el entorno único de los contenedores [30]. Sus principales características son las siguientes:

- Reparto de energía entre contenedores cercanos mediante la creación automática de clusters con la elección de un contenedor principal dependiendo de sus capacidades de comunicación y batería.
- Gran alcance, incluso en los barcos más grandes del mundo.
- Posibilidad de añadir sensores adicionales para medir un mayor número de parámetros, solo para un trayecto o permanentemente.
- Rendimiento óptimo en ambientes metálicos y húmedos.
- Rendimiento óptimo conforme a las normas locales de emisión de radio.

- ***Traxens-Hub***

La Traxens-Hub es la plataforma de Big Data que recolecta todos los datos de los TRAXEBS-BOXes y los prepara para mostrarlos al cliente de la manera más útil para éste, facilitando su comprensión y la toma de decisiones [30]. Sus principales características son las siguientes:

- Integración con la plataforma del cliente mediante servicios web o EDI (intercambio electrónico de datos).
- Acceso basado en web.
- Delegación de derecho de acceso a la plataforma, para permitir a los clientes o proveedores participar en la cadena de suministro.
- Envía notificaciones y alertas personalizadas a los usuarios.
- Herramientas de análisis de datos avanzadas.

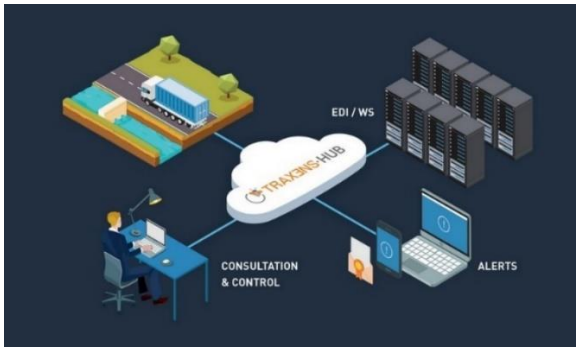

Figura 23. Arquitectua de la red de Traxens [30]

3.7.4 Cellotrack de Cellocator

Cellotrack es un producto diseñado por la división Cellocator de la empresa Pointer Telocation para el seguimiento avanzado de activos y la monitorización remota, que consta de un único dispositivo portable (incluye una batería de larga duración) que se coloca en el objeto que se quiera monitorizar, como podría ser un contenedor de carga. Este dispositivo está equipado con una carcasa impermeable de gran duración con un grado de protección IP67.

Este dispositivo cuenta con un acelerómetro 3D integrado para la detección de movimiento y vibraciones, un receptor GNSS que soporta GPS y GLONASS para determinar la posición y el seguimiento de una ruta de manera exacta, y, además, cuenta con 2 GPIOs configurables que se pueden utilizar para, entre otras funcionalidades, conectar diferentes tipos de sensores para medir los diferentes parámetros que el usuario quiera monitorizar.

Respecto a las comunicaciones de este dispositivo, CelloTrack dispositivo utiliza comunicaciones celulares 3G para comunicarse con la plataforma final del usuario y poder enviarle los datos que recoja el dispositivo durante el trayecto que se quiera monitorizar, por lo que es necesario el uso de una tarjeta sim de un operador móvil. Asimismo, el dispositivo transmite con una tasa de transmisión adaptativa, dependiendo del nivel de batería y del movimiento del dispositivo.

Por otro lado, otro producto de la misma empresa basado en el dispositivo CelloTrack es el llamado CelloTrack Container Lock. Este producto se coloca en la cerradura de los contenedores de carga y proporciona información sobre el contenedor al usuario, con las mismas funcionalidades que el CelloTrack. Además, el CelloTrack Container Lock contiene un sensor magnético que detecta la apertura de las puertas del contenedor, lo que permite al usuario tener un control estricto de todas las veces que se ha abierto el contenedor, es decir, un mayor control sobre la integridad del contenido transportado por el contenedor [31].

Capítulo 4. Arquitectura

En este apartado se pretende abordar la arquitectura del sistema que se va a desarrollar. En primer lugar, se describirá la arquitectura general del sistema, ya que aporta una visión global de los componentes del sistema. En segundo lugar, se profundizará en la arquitectura mediante el análisis de los componentes que conforman el servidor que se encontrará en la nube. Para finalizar, se mostrará y se justificará el modelo de datos utilizado, ya que es un componente clave cuando se pretende tratar una gran cantidad de información.

4.1 Arquitectura general del sistema

En la figura 24 se puede observar la arquitectura general del sistema final de gestión de contenedores.

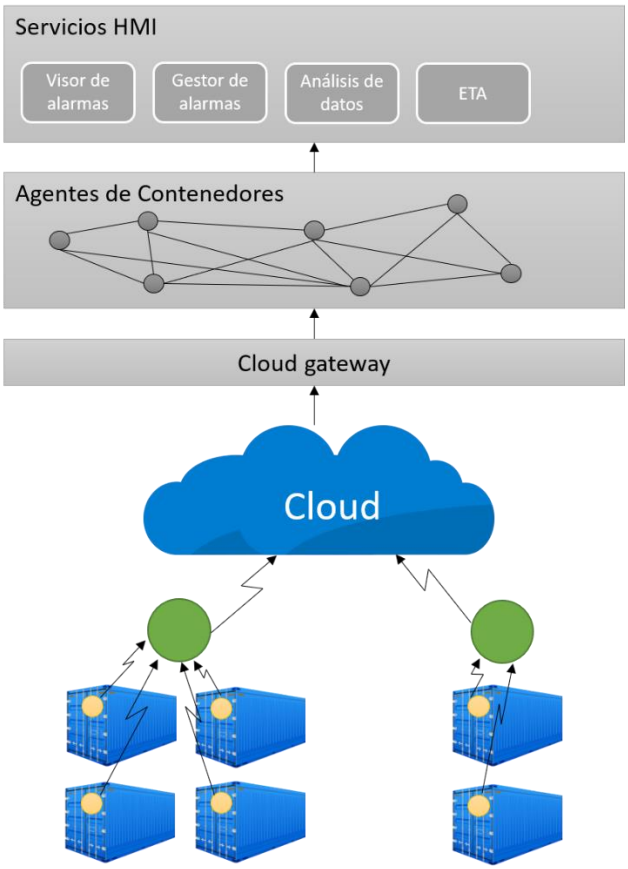

Figura 24. Arquitectura general del sistema

Cada contenedor cuenta con un sistema embebido (o dispositivo equivalente según la oferta disponible en el mercado), que tendrá la función de conectar, aglutinar y obtener la información de todos los sensores que se van a utilizar para monitorizar los diferentes parámetros del contenedor. Este sistema embebido se comunica con la nube de la plataforma mediante un gateway, que actúa como pasarela entre los contenedores de un mismo barco o zona y la nube de la plataforma. Existen diferentes tecnologías desarrolladas para establecer las comunicaciones en redes Low-Power Wide Area Networks (LPWANs), es decir, redes amplias con elementos de bajo consumo de potencia como son los sistemas embebidos. Estas tecnologías se han descrito con detalle en el apartado 3.2 de esta memoria.

Cada contenedor físico tiene su representación virtual en la plataforma en forma de contenedores virtuales LXD, que estarán gestionados por los agentes de contenedores, lo que convierte a este sistema en un sistema ciberfísico.

Finalmente, los datos correspondientes a cada contenedor o grupo de contenedores deben de ser fácilmente accesibles para los usuarios mediante una interfaz de usuario (HMI). Este no es el propósito último de este proyecto, sino que es proporcionar los datos con un formato estándar y fácilmente accesible para el usuario y que éste los pueda visualizar en su plataforma. En esta línea, el sistema cuenta con una API de gestión para que el usuario pueda gestionar sus contenedores en la plataforma, que se describirá posteriormente con más detalle.

4.2 Arquitectura interna del servidor

El servidor que se encuentra en la nube es el núcleo de todo el sistema, ya que es el encargado de alojar los diferentes componentes que tendrán como función recibir, almacenar y gestionar la información de todos los contenedores del sistema. La siguiente figura muestra un esquema con todos los componentes que se han instalado en el servidor.

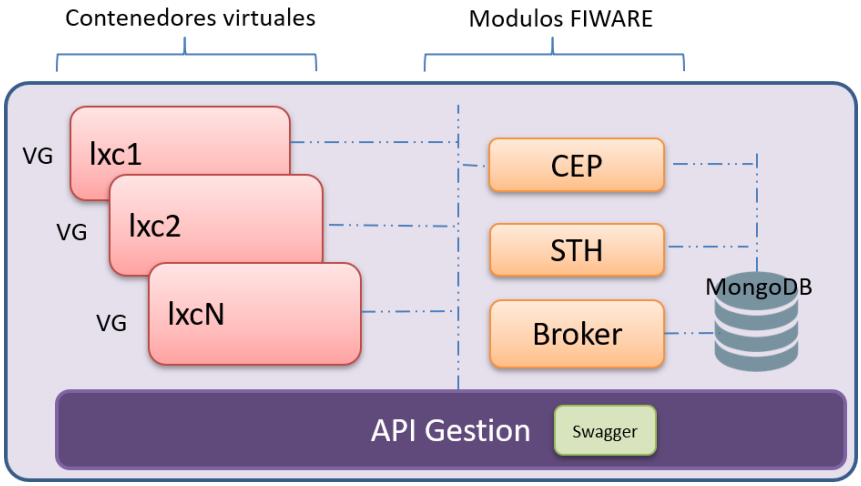

Figura 25. Arquitectura del servidor

Claramente se pueden observar cuatro partes diferenciadas:

- **Contenedores virtuales:** son los contenedores LXD que representan los contenedores físicos. Contienen la parte virtual de la gateway que se utilizará para comunicarse con los sistemas embebidos que se coloquen en los contenedores físicos y el gestor de eventos del contenedor.
- **Módulos FIWARE:** a este grupo pertenecen los módulos propios de la plataforma FIWARE (descritos en el apartado 3.3.4) que se utilizarán para el tratamiento de los datos. Estos módulos son Orion (context broker), STH (almacenamiento de datos históricos) y Perseo (se trata de un CEP, utilizado para gestionar las alarmas).
- **API de gestión:** se trata de una API REST cuya funcionalidad es la gestión de los contenedores de la plataforma, es decir, proporciona métodos para crear, eliminar, modificar y consultar los contenedores. Contiene una interfaz gráfica de Swagger.
- **Base de datos de MongoDB:** una base de datos del tipo MongoDB (NoSQL), con la que trabajarán los componentes de FIWARE, la API de gestión y los LXD.

4.3 Modelo de datos

FIWARE Orion utiliza una base de datos del tipo MongoDB para el tratamiento de los datos, por lo que se ha decido utilizar una base de datos de este tipo para todo el sistema, no siendo necesaria una base de datos del tipo SQL. Por lo tanto, el modelo de datos que se va a utilizar será con un formato JSON. A continuación, se expondrán el formato en JSON de un contenedor, un evento, una alarma y, finalmente, de la entidad de Orion que representará a un contenedor.

4.3.1 Contenedor

Cada contenedor se identifica mediante un identificador único para toda plataforma: su matrícula, ya que las matrículas son únicas para cada contenedor de carga. La matrícula tiene un formato específico: tres letras mayúsculas seguidas de una U, para finalizar con siete dígitos (por ejemplo, BMOU8710633). Además, cada contenedor cuenta con una frecuencia de notificación de eventos en minutos y su estado, que puede ser habilitado o deshabilitado.

Los contenedores cuentan con sensores, por lo que en cada contenedor se incluye un array con todos los sensores del contenedor. Cada sensor se identifica con un id único. Sus características son: un identificador del tipo del sensor (temperatura, humedad, …), las unidades de sus mediciones, su estado actual y los umbrales superiores e inferiores para disparar una alarma si se sobrepasan estos valores en una medida.

```
{
  "Matricula": string,
  "FrecuenciaNotificacionMinutos": number,
  "Estado": string,
  "sensores": [
    {
      "Id": number,
      "TipoSensorId": number,
      "Unidades": string,
      "Estado": string,
      "UmbralDisparoSuperior": string,
      "UmbralDisparoInferior": string
    }
  ]
}
```

```
{
    "Matricula": "BMOU8710633",
    "FrecuenciaNotificacionMinutos": 2,
    "Estado": "habilitado",
    "sensores": [
        {
            "Id": 0,
            "TipoSensorId": 1,
            "Unidades": "lux",
            "Estado": "habilitado",
            "UmbralDisparoSuperior": 5,
            "UmbralDisparoInferior": 0
        },
        {
            "Id": 1,
            "TipoSensorId": 1,
            "Unidades": "ºC",
            "Estado": "habilitado",
            "UmbralDisparoSuperior": 5,
            "UmbralDisparoInferior": -1
        }
    ]
}
```

Figura 26. Esquema y ejemplo de un contenedor en JSON

4.3.2 Evento

Un evento es la información sobre un contenedor (valores de sus sensores, posición GPS, …) que se envía periódicamente a la plataforma del cliente. En cada evento se registra la fecha y la hora (en formato ISO 8601 yyyy-MM-ddThh:mm:ss.SSSZ) en la que se crea y se envía (FHEvento) además de las características del contenedor: matrícula e información de cada sensor, que incluye el id del sensor, el valor de la medida, su posición GPS (en formato GeoJSON) y la fecha y la hora en la que se ha registrado esta medida (FHRegistro). Finalmente, cabe destacar

que en los eventos no existe ningún identificador único, ya que es suficiente con la información que contienen (matrícula del contenedor e ids de sus sensores).

```
{
    "FHEvento": string,
    "EventosContenedor": [
        {
            "Matricula": string,
            "EventosSensor": [
                {
                    "ContenedorSensorId": number,
                    "Valor": string,
                    "PosicionGPS": {
                        "coordinates": [
                            number,
                            number
                        ],
                        "type": "Point"
                    },
                    "FHRegistro": string
                }
            ]
        }
    ]
}
```

```
{
    "FHEvento": "2018-06-11T12:34:00.799Z",
    "EventosContenedor": [
        {
            "Matricula": "BMOU8710633",
            "EventosSensor": [
                {
                    "ContenedorSensorId": 4019,
                    "Valor": "1",
                    "PosicionGPS": {
                        "coordinates": [
                            -0.308595436,
                            39.43556368
                        ],
                        "type": "Point"
                    },
                    "FHRegistro": "2018-06-11T10:30:34.00Z"
                },
                {
                    "ContenedorSensorId": 4020,
                    "Valor": "14",
                    "PosicionGPS": {
                        "coordinates": [
                            -0.308595436,
                            39.43556368
                        ],
                        "type": "Point"
                    },
                    "FHRegistro": "2018-06-11T10:33:59.00Z"
                }
            ]
        }
    ]
}
```

Figura 27. Esquema y ejemplo de un evento en JSON

4.3.3 Alarma

Una alarma es un tipo especial de evento que se envía a la plataforma del cliente cuando se detecta que un sensor ha obtenido un valor que se encuentra fuera de los límites superior e inferior que marcan los umbrales especificados en la definición del contenedor. Asimismo, cuando se detecta que el mismo sensor ha obtenido un valor que se encuentra dentro de los valores definidos por los umbrales, se envía otro mensaje de alarma para desactivar la alarma anteriormente disparada.

En cada alarma se registra la fecha y la hora (en formato ISO 8601 yyyy-MM-ddThh:mm:ss.SSSZ) en la que se crea y se envía (FHNotificacion), la fecha y la hora en la que se ha registrado la medida del sensor, la posición GPS (en formato GeoJSON), el id del sensor cuyo valor ha disparado una alarma y su valor. Además, dependiendo si el mensaje es de disparo o desactivación de la alarma, su estado será de *on* u *off*.

```
{
    "FHDisparoOnOff": string,
    "FHNotificacion": string,
    "PosicionGPS": {
        "type": "Point",
        "coordinates": [
            number,
            number
        ]
    },
    "ContenedorSensorId": number,
    "EstadoAlarma": string,
    "Valor": string
}
```

```
{
    "FHDisparoOnOff": "2018-05-15T07:12:33.001Z",
    "FHNotificacion": "2018-05-15T09:12:33.001Z",
    "PosicionGPS": {
        "type": "Point",
        "coordinates": [
            -0.3279804,
            39.4549639
        ]
    },
    "ContenedorSensorId": 1,
    "EstadoAlarma": "on",
    "Valor": "28.00"
}
```

Figura 28. Esquema y ejemplo de una alarma en JSON

4.3.4 Contenedor como entidad de FIWARE Orion

Los datos de los contenedores se deben de almacenar correctamente en la base de datos de FIWARE Orion para que éste los pueda procesar correctamente, por lo que es necesario crear una entidad que corresponda con cada contenedor. Estas entidades se almacenarán dentro del Fiware-Service llamado *cybercontainer* y con una ruta cuyo Fiware-ServicePath será */demo*.

Cada entidad tiene un identificador único, que en este caso es la matrícula del contenedor precedida de unos valores alfanuméricos (por ejemplo, gw-c4) necesarios para el correcto funcionamiento de la gateway que comunicará a los contenedores físicos con los contenedores virtuales. El funcionamiento de esta gateway se explicará en apartado 5.4.2 de esta memoria. Además, cada entidad pertenece a un tipo, que en este caso será su propia matrícula.

Los sensores del contenedor se incluyen en forma de atributos de la entidad. Cada atributo cuenta con un valor, un tipo de datos de este valor y unos metadatos. Los metadatos proporcionan la información de la fecha y hora de la última modificación del valor del sensor y la id del sensor (excepto el atributo de la posición GPS).


```
{
    "id": "gw-c4-BMOU8710633",
    "type": "BMOU8710633",
    "PosicionGPS": {
        "type": "geo:json",
        "value": {
            "coordinates": [
                -0.30346589,
                39.42984964
            ],
            "type": "Point"
        },
        "metadata": {
            "dateModified": {
                "type": "DateTime",
                "value": "2018-06-27T11:52:55.00Z"
            }
        }
    },
    "temperature": {
        "type": "float",
        "value": 9,
        "metadata": {
            "ContenedorSensorId": {
                "type": "integer",
                "value": 4020
            },
            "dateModified": {
                "type": "DateTime",
                "value": "2018-06-27T11:52:55.00Z"
            }
        }
    }
}
```


Figura 29. Ejemplo de un contenedor como entidad de Orion

Capítulo 5. Implementación

En este capítulo se pretende realizar una explicación detallada de cómo se ha desarrollado el sistema completo: herramientas utilizadas, elementos que se han desarrollado y la interacción entre los diferentes elementos del sistema. El capítulo se va a dividir en diversos subapartados que corresponden con los bloques que componen el sistema final, para así poder profundizar en cada uno de los bloques o partes que se han descrito en la arquitectura del capítulo anterior. Finalmente, se realizará un resumen que incluirá un esquema que pretende sintetizar todo lo explicado anteriormente.

5.1 Servidor y base de datos

Para construir el servidor en la nube del sistema, se ha utilizado una máquina virtual en la que se ha instalado como sistema operativo la distribución de Linux Ubuntu con la versión 16.04 LTS. Esta máquina virtual se ha creado dentro de una máquina física que actúa como servidor y se encuentra en el laboratorio del SATRD de la UPV, por lo que se le ha asignado la dirección IPv4 158.42.188.26 y el dominio cyberc.satrdlab.upv.es.

En el puerto 80 de esta máquina se ha instalado un servidor web Apache2 para, entre otras funciones, alojar una página web propia del proyecto, que se utilizará en la demo o prototipo que se describirá en el capítulo siguiente. Además, se ha instalado un Apache Tomcat en el puerto 8080, necesario para poder instalar el CEP y poder alojar la API de gestión.

Como se ha explicado anteriormente, se ha decidido utilizar una base de datos MongoDB. MongoDB es de código abierto, por lo que se ha descargado de su repositorio oficial [32] la versión Community Server 4.0.0. Posteriormente, se ha instalado en la máquina virtual en el puerto por defecto de MongoDB, el 27017, por lo que su url es http://cyberc.satrdlab.upv.es:27017 (no es accesible desde el exterior de la máquina virtual por cuestiones de seguridad).

5.2 Módulos FIWARE

El funcionamiento y las características de los módulos de FIWARE ya se han explicado en el apartado 3.3.4 de esta memoria, por lo que en esta sección no se pretende repetir esta explicación, sino que se va a describir como se han instalado estos módulos en la máquina virtual y como interactúan con los otros elementos.

5.2.1 FIWARE Orion

El context bróker de FIWARE se puede obtener de su repositorio oficial de GitHub [33]. Según su documentación oficial, solo se puede instalar en los sistemas operativos CentOS y RedHat, por lo que no se puede instalar directamente en nuestra máquina con un sistema operativo Ubuntu. Para poder instalar Orion en Ubuntu ha sido necesario realizar la instalación en un contenedor de Docker. En el archivo de configuración del contenedor de Docker, se ha indicado que el puerto de Orion sea su puerto por defecto, el 1026, y también se ha indicado que permita todos los orígenes en el CORS (Cross-Origin Resource Sharing). Por lo tanto, la url del context bróker es

http://cyberc.satrdlab.upv.es:1026/. Además, Orion necesita una base de datos MongoDB para funcionar, por lo que se ha indicado en sus ficheros de configuración que utilice la base de datos de la misma máquina a la que se puede acceder por el puerto 27017.

Para realizar una consulta de los contenedores registrados como entidades en Orion se debe de realizar una petición HTTP GET a la url http://cyberc.satrdlab.upv.es:1026/v2/entities, indicando las cabeceras siguientes:

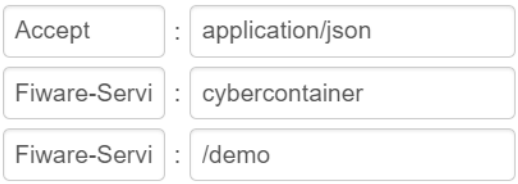

Figura 30. Cabeceras utilizadas en una petición HTTP GET a Orion

Para crear una entidad se tiene que realizar una petición HTTP POST a la url http://cyberc.satrdlab.upv.es:1026/v2/entities, con la entidad a crear con el formato correspondiente (figura 29) en el cuerpo de la petición e indicando las cabeceras siguientes:

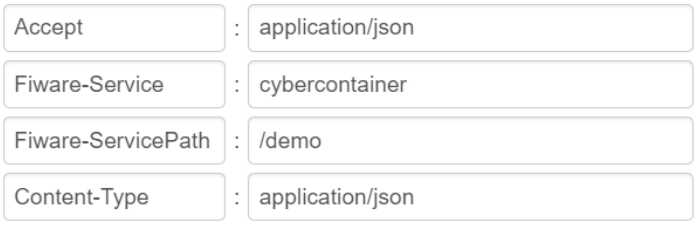

Figura 31. Cabeceras utilizadas en una petición HTTP POST y PATCH a Orion

Para modificar una entidad se tiene que realizar una petición HTTP PATCH a la url http://cyberc.satrdlab.upv.es:1026/v2/entities/{id-entidad}/attrs, incluyendo solo los atributos a modificar respetando el formato de los atributos de la entidad (figura 29) en el cuerpo de la petición e indicando las cabeceras que se muestran en la figura 31.

5.2.2 Fiware STH-Comet

El gestor de datos históricos de FIWARE (STH) llamado Comet se puede obtener de su repositorio oficial de GitHub [34]. Su instalación es sencilla y solo es necesario indicarle en sus ficheros de configuración que utilice la base de datos MongoDB (puerto 27017) para almacenar los datos históricos y su puerto, que en este caso será su puerto por defecto, el 8666, por lo que la url del STH es http://cyberc.satrdlab.upv.es:8666.

El STH necesita suscribirse previamente a cada entidad de Orion de la que se desee almacenar sus datos históricos. Para realizar una suscripción a Orion, se debe de realizar una petición HTTP POST a la url http://cyberc.satrdlab.upv.es:1026/v1/subscribeContext, cuyo cuerpo tiene que tener el formato de una suscripción de Orion, pero indicando la url del STH como url receptora. En la figura 32 se puede observar un ejemplo de suscripción del STH a Orion. En primer lugar, se incluyen las entidades, seguidas de los atributos a los que se quiere suscribir. En segundo lugar, se deben de indicar los campos de configuración de la suscripción, como la url receptora, la duración y las condiciones de notificación, es decir, las condiciones que se deben de cumplir para que el STH reciba notificaciones de Orion con los datos para su almacenamiento. En este caso,

se ha indicado que la condición sea *ONCHANGE*, es decir, que solo se almacenarán los valores de los atributos cuando el valor de estos cambie.

```
{
    "entities": [
        {
            "type": "AMCU9261477",
            "isPattern": "false",
            "id": "gw-c1-AMCU9261477"
        }
    ],
    "attributes": [
        "brightness",
        "temperature"
    ],
    "reference": "http://cyberc.satrdlab.upv.es:8666/notify",
    "duration": "PT24H",
    "notifyConditions": [
        {
            "type": "ONCHANGE",
            "condValues": [
                "brightness",
                "temperature"
            ]
        }
    ],
    "throttling": "PT30S"
}
```

Figura 32. Ejemplo de suscripción a Orion por parte del STH

Las consultas para obtener datos del STH se realizan para cada atributo de cada entidad de manera individual. Para obtener los datos de un atributo es necesario realizar una petición HTTP GET a la url http://cyberc.satrdlab.upv.es:8666/STH/v1/contextEntities/type/{entity-type}/id/{entity-id}/attributes/{entity-attribute}?{queryParam}, donde el parámetro *queryParam* es un parámetro para filtrar los resultados por diversos criterios, algunos de los cuales son: fecha concreta, periodo de tiempo, últimos N resultados, …

5.2.3 FIWARE Perseo

El CEP de FIWARE, llamado Perseo, es el encargado de gestionar las alarmas de los contenedores. Perseo está compuesto por dos componentes: el core o back-end y el front-end, que interaccionan entre sí, siendo necesario instalar ambos. Por una parte, el Perseo core se ha obtenido de su repositorio oficial de GitHub [36], y, posteriormente, se ha instalado como una aplicación en el Apache Tomcat que se encuentra en el puerto 8080 del servidor, siendo su url http://cyberc.satrdlab.upv.es:8080/perseo-core/. Por otra parte, el Perseo front-end también se ha obtenido de su repositorio oficial de GitHub [34]. Sin embargo, el front-end no se tiene que instalar en el Tomcat, por lo que se ha instalado en el puerto 9090 del servidor, siendo su url completa http://cyberc.satrdlab.upv.es:9090/.

Perseo necesita suscribirse previamente a cada entidad de Orion de la que se desee procesar sus datos para generar las alarmas. El proceso de suscripción es el mismo que se ha descrito en el apartado anterior (5.2.2) pero con una única diferencia: la url receptora (“reference”) tiene que ser la url específica de Perseo http://cyberc.satrdlab.upv.es:9090/notices, que pertenece a su front-end.

Las alarmas se disparan cuando un valor se encuentra fuera de los límites marcados por unos umbrales. Para que Perseo pueda generar alamas dependiendo de estos umbrales, es necesario especificar unas reglas (rules) en el lenguaje de reglas de Esper. Estas reglas se tienen que

introducir en Perseo mediante una petición HTTP POST a la url http://cyberc.satrdlab.upv.es:9090/rules, cuyo cuerpo tiene que tener el formato de mensaje de una regla de Perseo (figura 34).

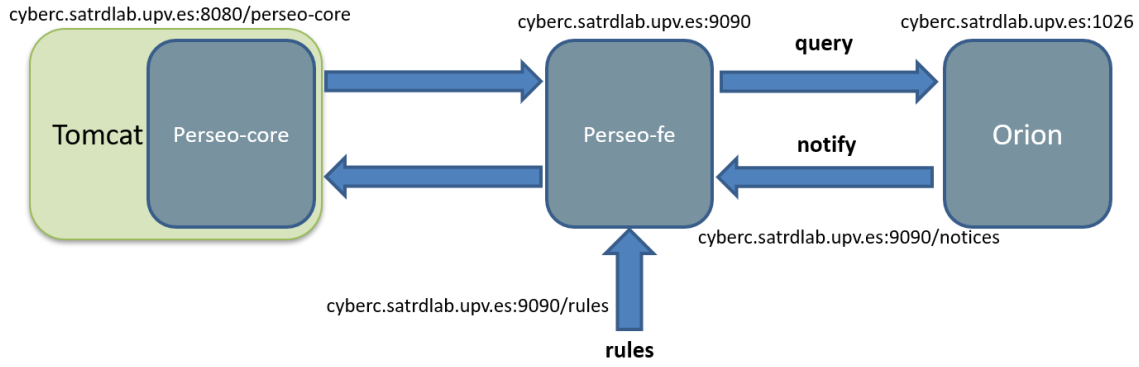

Figura 33. Funcionamiento de FIWARE Perseo

En la figura 34 se puede observar un ejemplo de mensaje de una regla de Perseo. En primer lugar, se indica el nombre de la regla y la regla en el lenguaje de procesado de eventos (EPL) Esper ("text"). Seguidamente, se indica la acción que se realizará cuando se cumpla esta regla, indicando que se realizará una petición HTTP POST a la url indicada ("url"), cuyo cuerpo ("template") está formado por texto en formato JSON con la estructura de una alarma (figura 28). De esta manera, al saltar una alarma se enviará una petición HTTP POST a la plataforma del cliente que incluya la alarma en su formato correspondiente.

```
{
   "name":"temperature_BMOU8710633_rule",
   "text":"select *,\"temperature_BMOU8710633_rule\" as ruleName, *, ev
       .temperature? as Temperature from pattern [every ev=iotEvent(cast
       (cast(temperature?,String),float)>3.5 and type=\"BMOU8710633\")]",
   "action":{
      "type": "post",
        "template": "{\"FHDisparoOnOff\": \"${dateModified}\"
            ,\"FHNotificacion\": \"${dateSend}\",\"PosicionGPS\": {\"type\":
            \"Point\",\"coordinates\": [ -0.3279804, 39.4549639 ]}
            ,\"ContenedorSensorId\": ${ContenedorSensorId},\"EstadoAlarma\":
            \"on\",\"Valor\": \"${temperature}\"}",
        "parameters": {
            "url": "http://demo.cybercontainer.es/cybercontainer/apicps
                /alarmas/"
        }
   }
}
```

Figura 34. Ejemplo de mensaje de una regla de Perseo

En la introducción de la regla en lenguaje Esper es donde se debe de personalizar la alarma para cada contenedor, indicando sus sensores específicos y los umbrales de cada sensor.

5.3 API de gestión

La API de gestión de contenedores se ha creado para gestionar los contenedores en la plataforma de una manera simple y unificada. Esta REST API se ha desarrollado en el lenguaje de programación Java utilizando principalmente la librería Jersey, que es una librería que simplifica el desarrollo de servicios web RESTful y sus clientes en Java. También se han utilizado las librerías de Java de MongoDB para interactuar con la base de datos y las librerías de Google para el tratamiento de objetos JSON, ya que esta API trabaja con JSON como formato de intercambio de datos. Es conveniente mencionar que se ha utilizado el IDE para desarrolladores de Java EE de Eclipse, concretamente la versión Eclipse Oxygen Jee.

Esta API está compuesta por 6 recursos diferentes:

- **Obtención de todos los contenedores:** método HTTP GET sin parámetros que devuelve un objeto JSON con todos los contenedores y sus características (con el formato de contenedor descrito en el apartado 4.3.1) que se encuentran registrados en el sistema. La url de este recurso es http://cyberc.satrdlab.upv.es:8080/cybercontainer/api/contenedores.
- **Obtención de un contenedor:** método HTTP GET cuyo parámetro es el identificador del contenedor, es decir, su matrícula. Devuelve un objeto JSON con el contenedor demandado. La url de este recurso es http://cyberc.satrdlab.upv.es:8080/cybercontainer/api/contenedores/{conetendorId}.
- **Alta de un contenedor:** método HTTP POST cuyo cuerpo debe de ser un objeto JSON que contenga el contenedor a introducir en el sistema. La url para acceder a este recurso es http://cyberc.satrdlab.upv.es:8080/cybercontainer/api/contenedores.
- **Modificación de contenedores:** método HTTP PUT sin parámetros cuyo cuerpo debe de ser un array JSON que incluya los parámetros de los contenedores que se quieran modificar. La url de este recurso es http://cyberc.satrdlab.upv.es:8080/cybercontainer/api/contenedores/.
- **Modificación de un contenedor:** método HTTP PUT, con el identificador del contenedor como parámetro, cuyo cuerpo debe de ser un objeto JSON que contenga las características del contenedor que se quieran modificar. La url de este recurso es http://cyberc.satrdlab.upv.es:8080/cybercontainer/api/contenedores/{conetendorId}.
- **Eliminación de un contenedor:** método HTTP DELETE cuyo parámetro es el identificador del contenedor que se quiera eliminar. La url de este recurso es http://cyberc.satrdlab.upv.es:8080/cybercontainer/api/contenedores/{conetendorId}

Los códigos de estado HTTP que se han implementado en esta API son los siguientes:

- **200:** operación realizada correctamente.
- **201:** contenedor creado correctamente.
- **400:** error en la petición. Ocurre porque se ha introducido un Id incorrecto en la url o un dato con un formato no válido en el cuerpo de la petición.
- **404:** contenedor no encontrado.
- **405:** método no permitido.
- **409:** el contenedor que se desea introducir ya existe.
- **500:** error interno del servidor.

Para mejorar la experiencia del usuario se ha generado una documentación de esta API utilizando el framework Swagger, por lo que se ha tenido que crear una definición de la API en el formato de Swagger, guardarla en un archivo con formato YAML (APIgestion.yaml) y adjuntarla a la API. De esta manera, se puede consultar en la url http://cyberc.satrdlab.upv.es:8080/cybercontainer/ la documentación de esta API, que incluye una interfaz gráfica para poder probar los recursos de esta API, así como el modelo de datos que se debe de introducir en cada método.

Figura 35. Métodos de la API de gestión vistos en Swagger

Finalmente, se va a realizar una explicación del funcionamiento interno de esta API. En primer lugar, es necesario aclarar que esta API trabaja con dos bases de datos y dos modelos de datos de un contenedor diferentes en MongoDB:

- Una base de datos llamada cybercontainer, que almacena una lista de los contenedores dados de alta en el sistema con el formato de un **contenedor en JSON** (apartado 4.3.1). Este modelo de datos ha sido creado para su utilización en la API y **no almacena valores** de los sensores de los contenedores.
- La base de datos de FIWARE Orion, que almacena las entidades con un formato de una **entidad de Orion** (4.3.4) y **almacena el último valor** obtenido por cada sensor.

La utilización de dos modelos de datos facilita el trabajo al usuario, ya que el formato un contenedor en JSON es más entendible o legible que el formato de las entidades de Orion. De esta manera, el proceso de conversión del formato de datos de la API al formato de los datos de JSON se realiza internamente y siendo totalmente transparente para el usuario.

Todos los procesos que se realizan en este servicio web, excepto los métodos GET, siguen un mismo patrón. Dependiendo del recurso de la API, en primer lugar, se realiza un procesado y una conversión de los datos recibidos. Seguidamente, se producen unas consultas y operaciones en FIWARE Orion, y si estas operaciones se realizan correctamente, posteriormente se realizan las correspondientes operaciones en la base de datos cybercontainer. En los métodos GET, no se realizan operaciones con Orion, si no que se efectúan directamente las consultas a la base de datos MongoDB.

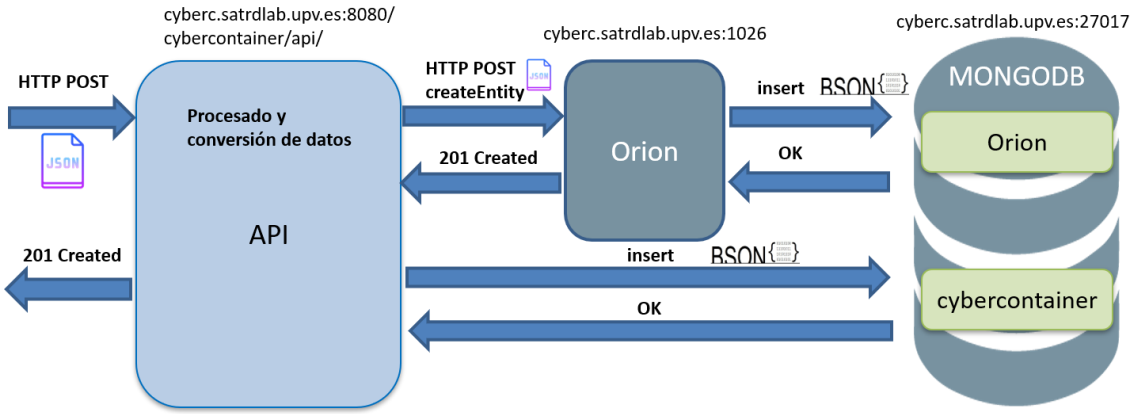

Figura 36. Proceso de alta de un contenedor

En la figura 36 se puede observar un esquema que resume el proceso de alta de un contenedor. Cuando se envía una petición POST para el alta de un contenedor, se procesa el JSON recibido y se crea un nuevo JSON con el formato de una entidad de Orion. Previamente, se comprueba si existe una entidad en Orion con la matrícula especificada, y si no existe esta matrícula, se procede a continuar con el proceso de alta. Seguidamente, se envía una petición HTTP POST a Orion para crear esta nueva entidad en el context broker. Orion insertará esta entidad en su base de datos correspondiente de MongoDB, convirtiendo la entidad que representa el contenedor a un formato BSON para poder introducirlo mediante un comando *insert* a la base de datos MongoDB. Si este proceso se ha producido correctamente, se insertará en la base de datos llamada cybercontainer el contenedor recibido originalmente en la petición POST a la API.

```
// 1. Connect to MongoDB instance running on localhost
MongoClient mongoClient = new MongoClient();

// Access database named 'cybercontainer'
MongoDatabase database = mongoClient.getDatabase("cybercontainer");

// Access collection named 'listaContenedores'
MongoCollection<Document> collection = database.getCollection("listaContenedores");
List<Document> results = collection.find().into(new ArrayList<Document>());
List<Container> containersList = new ArrayList<>();

for(Document d : results) {
    List<Sensor> sensoresContenedor = new ArrayList<>();
    Container container = new Container();
    container.setMatricula(d.getString("Matricula"));
    container.setEstado(d.getString("Estado"));
    container.setFrecuenciaNotificacionMinutos(d.getInteger("FrecuenciaNotificacionMinutos"));
    List<Document> sensores = (List<Document>) d.get("sensores");
    for(Document s : sensores) {
        Sensor sensor = new Sensor();
        sensor.setId(s.getInteger("Id"));
        sensor.setTipoSensorId(s.getInteger("TipoSensorId"));
        sensor.setUnidades(s.getString("Unidades"));
        sensor.setEstado(s.getString("Estado"));
        sensor.setUmbralDisparoSuperior(s.getDouble("UmbralDisparoSuperior"));
        sensor.setUmbralDisparoInferior(s.getDouble("UmbralDisparoInferior"));
        sensoresContenedor.add(sensor);
    }
    container.setSensores(sensoresContenedor);
    containersList.add(container);
}

Container [] containers = containersList.toArray(new Container[]{});
mongoClient.close();
return JSend.success(containers, 200);
```

Figura 37. Código de la obtención de un contenedor almacenado en MongoDB en Java

```
String Matricula = containerJson.getMatricula();
String Estado = containerJson.getEstado();
int FrecuenciaNotificacionMinutos = containerJson.getFrecuenciaNotificacionMinutos();
if(FrecuenciaNotificacionMinutos <= 0) FrecuenciaNotificacionMinutos = 5;
List<Sensor> sensores = containerJson.getSensores();

String matriculaPattern = "([A-Z]){3}U(\\d){7}"; //formato correcto matrícula del contenedor
if(Pattern.matches(matriculaPattern, Matricula)) {

    MongoClient mongoClient = MongoDB.getMongoClient("localhost");
    MongoDatabase database = mongoClient.getDatabase("cybercontainer");
    MongoCollection<Document> collection = database.getCollection("listaContenedores");

    BasicDBObject matriculaQuery = new BasicDBObject();
    matriculaQuery.put("Matricula", containerJson.getMatricula());

    List<Document> results = collection.find(matriculaQuery).into(new ArrayList<Document>());

    if(results.isEmpty()) {//si no existe
        if(containerJson.hasNulls()) {//si hay campos vacíos en el JSON
            mongoClient.close();
            return JSend.error("Dato incorrecto", 400);
        }else {
            Document container = new Document()
                    .append("Matricula", Matricula)
                    .append("FrecuenciaNotificacionMinutos", FrecuenciaNotificacionMinutos)
                    .append("Estado", Estado)
                    .append("sensores", sensores);
            collection.insertOne(container);

            mongoClient.close();
            return JSend.success("Contenedor creado", 201);
```

Figura 38. Código de la inserción de un contenedor en MongoDB en Java

5.4 Contenedores LXD y gateway de comunicaciones

5.4.1 Contenedores LXD

Para la representación virtual de los contenedores se ha decidido utilizar la tecnología LXD. Para crear un contenedor LXD, se debe de indicar la imagen de la distribución de Linux con la que se quiere cargar el contenedor virtual. Se ha decidido utilizar Ubuntu 16.0 LTS Server para que tenga concordancia con el sistema operativo que utiliza el servidor. Para crear un contenedor LXD se ha utilizado el comando siguiente: `lxc launch ubuntu: {nombre del contenedor}`. A cada LXD se le asigna una dirección IPv4 diferente del rango 10.0.10.0/24 para este proyecto, por lo que no están dentro de la misma subred que el servidor (su IP es 158.42.188.26), pero son accesibles desde el servidor y desde el exterior a través del servidor. Todos los contenedores se crean en el mismo servidor.

```
cyber@cyberc:~$ lxc launch ubuntu: BMOU8710633
Creating BMOU8710633
Starting BMOU8710633
```

Figura 39. Creación de un contenedor LXD

Mediante el comando `lxc list` de LXD se obtiene una lista con todos los contenedores que contiene el sistema, mostrando su estado, su dirección IP, su tipo y el número de snapshots (instantánea del estado del contenedor en un momento determinado, a modo de copia de seguridad) que se han generado.

```
cyber@cyberc:~$ lxc list
+------+---------+--------------------+------+------------+-----------+
| NAME |  STATE  |        IPV4        | IPV6 |    TYPE    | SNAPSHOTS |
+------+---------+--------------------+------+------------+-----------+
| api  | RUNNING | 10.0.10.108 (eth0) |      | PERSISTENT | 0         |
+------+---------+--------------------+------+------------+-----------+
| lxc1 | RUNNING | 10.0.10.18 (eth0)  |      | PERSISTENT | 0         |
+------+---------+--------------------+------+------------+-----------+
| lxc2 | RUNNING | 10.0.10.160 (eth0) |      | PERSISTENT | 0         |
+------+---------+--------------------+------+------------+-----------+
```

Figura 40. Lista de contenedores LXD

5.4.2 Gateway de comunicaciones

En este sistema ciberfísico, el sistema embebido que se encuentra en el contenedor de carga físico se tiene que comunicar con el contenedor virtual LXD que se encuentra en el servidor de la nube. Para llevar a cabo esta comunicación, se va a emplear una gateway que se ha desarrollado dentro del proyecto europeo de investigación INTER-IoT, en el que se ha participado mediante una beca de colaboración tipo B.

Esta gateway ha sido desarrollado en el lenguaje de programación Java utilizando Apache Maven para crear y gestionar el proyecto de Java. La gateway está compuesta por dos partes, que cuentan con su propia consola para poder ejecutar sus propios comandos.

- ***Physical gateway***

Esta parte se debe de incluir en el sistema embebido que se encuentra en el contenedor físico y actúa como la parte cliente de la comunicación. Se encarga de conectar y gestionar los sensores conectados físicamente al sistema embebido mediante la utilización de unos controladores específicos para cada tipo de sensor (Panstamp, Arduino, Coap y simuladores). En primer lugar, la gateway física tiene que establecer una conexión con la parte virtual de la gateway. Una vez se ha establecido la conexión correctamente, ya se puede proceder a conectar sensores o dispositivos a la gateway. También se puede comprobar la lista de sensores que se han conectado a la gateway virtual, y si se encuentran activados.

Cuando se conecta un nuevo sensor (device), se añade como un nuevo atributo a la entidad del contenedor de Orion mediante un *update* de esta entidad. Posteriormente, la parte física de la gateway empieza a recibir datos del sensor, que son enviados a la parte virtual de la gateway mediante el uso del protocolo WebSocket (RFC 6455).

"El protocolo WebSocket permite llevar a cabo comunicaciones bidireccionales entre un cliente que ejecute un código que no es de confianza en un entorno controlado y un servidor que ha optado por realizar comunicaciones en ese código" [37]. Este protocolo se diseñó para solucionar problemas en las limitaciones de conexiones TCP sobre puertos diferentes al 80, ya que este protocolo permite multiplexar diferentes servicios WebSocket sobre un único puerto TCP. De esta manera, con la apertura de un único puerto TCP en nuestro servidor, las múltiples gateways se podrían comunicar a través de este único puerto.

La parte física de la gateway cuenta con un archivo de configuración donde se indica el host, que en este caso corresponderá con la dirección IP del contenedor virtual LXD, y el puerto de la parte virtual de la gateway, siendo el puerto 8829 el elegido (puerto que pertenece al rango de

puertos 8809-8872 no asignados según la IANA). Además, se debe de indicar el tipo de sensores que se van a conectar para activar los controladores correspondientes.

```
eu.inter-iot.gateway.connector.host: 158.42.188.26/lxc1
eu.inter-iot.gateway.connector.port: 8829

extension.eu.inter-iot.gateway.device-controller.arduino-uno.enabled: false
extension.eu.inter-iot.gateway.device-controller.panstamp.enabled: false
extension.eu.inter-iot.gateway.device-controller.simulator.enabled: true
extension.eu.inter-iot.gateway.protocol-module.coap.enabled: false
extension.eu.inter-iot.gateway.an-module.serial.enabled: false
extension.eu.inter-iot.gateway.console.enabled: true
```

Figura 41. Fichero de configuración de la physical gateway

- ***Virtual gateway***

Esta parte se debe de incluir en el contenedor virtual LXD y actúa como la parte servidor de la comunicación. Se encarga de recibir, gestionar y enviar a Orion los datos que recibe de la parte física de la gateway, además de gestionar la conexión con la gateway física y sus sensores.

La gateway física se mantiene a la espera de conexiones por parte de la gateway física. Cuando se conecta un sensor a la gateway física, ésta empieza a recibir los valores de las medidas del sensor, que serán enviados a la gateway virtual. Seguidamente, la gateway virtual procesa el dato recibido y crea un JSON con el formato del contenedor de una entidad de Orion (apartado 4.3.4). Finalmente, este JSON se envía como cuerpo de una petición HTTP PATCH a Orion, actualizando el atributo correspondiente con el valor recibido del sensor. En la consola de la gateway virtual se puede observar la respuesta de Orion a la petición de *update* enviada.

La parte física de la gateway cuenta con un archivo de configuración donde se indican los parámetros de configuración de Orion, es decir, el host y el puerto de Orion, además del Fiware-Service y el Fiware-ServicePath.

```
eu.inter-iot.gateway.mw-module.orion.fiware-service: cybercontainer
eu.inter-iot.gateway.mw-module.orion.fiware-servicepath: /demo
eu.inter-iot.gateway.mw-module.orion.protocol: http
eu.inter-iot.gateway.mw-module.orion.host: 158.42.188.26
eu.inter-iot.gateway.mw-module.orion.port: 1026
eu.inter-iot.gateway.mw-module.orion.notify.ip: 10.0.10.18
eu.inter-iot.gateway.mw-module.orion.notify.protocol: http
eu.inter-iot.gateway.mw-module.orion.notify.port: 8080
eu.inter-iot.gateway.mw-module.orion.gateway.uuid: c1
```

Figura 42. Fichero de configuración de la virtual gateway

- ***Comunicaciones entre la virtual gateway y la physical gateway***

A modo de resumen, la figura 43 muestra un esquema de las comunicaciones entre el gateway físico y el gateway virtual. Por último, es necesario destacar que para poder utilizar el protocolo WebSocket, ha sido necesario permitir expresamente en el servidor Apache (en este caso actúa como proxy) las comunicaciones con este protocolo en el puerto 8829. Además, se ha tenido que configurar un proxy para que se pueda acceder desde el exterior a las direcciones IP de los contenedores virtuales. Esto se ha conseguido modificando sus propiedades en su archivo de configuración.

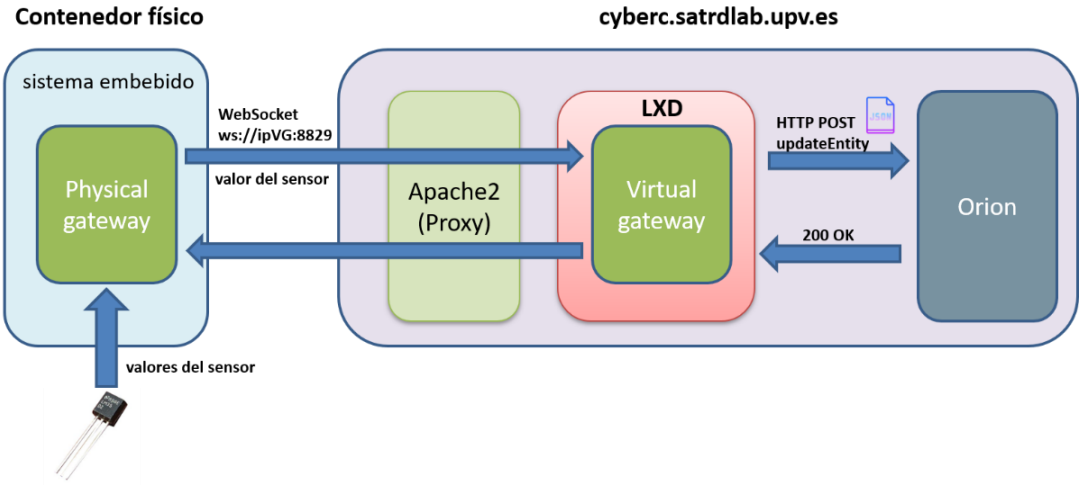

Figura 43. Comunicaciones entre el gateway físico y el virtual

5.5 Gestor de eventos

Para monitorizar los valores de los sensores que tiene instalados cada contenedor, es necesario generar unos eventos de cada contenedor (su formato de datos se ha explicado en el apartado 4.3.2) cada cierto tiempo, para almacenarlos en la misma plataforma o enviarlos a la plataforma del cliente. Se ha desarrollado un programa en Java llamado Gestor de eventos, que se ejecutará en cada contenedor virtual (LXD) para aprovechar la capacidad de almacenamiento y de computación de los micro contenedores virtuales. Este programa tiene como función crear los eventos cada cierto tiempo del contenedor en el que se ejecute. Es conveniente apuntar que se ha utilizado el IDE para desarrolladores de Java EE de Eclipse, concretamente la versión Eclipse Oxygen Jee. En este apartado se va a explicar el funcionamiento del Gestor de eventos.

El Gestor de eventos cuenta con un archivo de configuración llamado *configuration.properties*. En este archivo se le indica al programa la matrícula del contenedor, la url de la API de gestión de contenedores, de la plataforma del cliente y de Orion, además de los Fiware-Service y Fiware-ServicePath a los que se van a realizar las consultas para obtener información de las entidades.

```
matricula: BMOU9226583
api-gestion-endpoint: http://cyberc.satrdlab.upv.es:8080/cybercontainer/api/contenedores/
api-orion-endpoint: http://cyberc.satrdlab.upv.es:1026
api-IPV-endpoint: http://demo.cybercontainer.es/cybercontainer/apicps/eventos/
Fiware-Service: cybercontainer
Fiware-ServicePath: /demo
```

Figura 44. Ejemplo del fichero de configuración del Gestor de eventos

Cuando el programa se inicia, su primera tarea consiste en obtener los datos del fichero de configuración y guardarlos en variables. Seguidamente, obtiene la frecuencia de notificación en minutos del contenedor mediante la realización de una petición HTTP GET a la API de gestión, introduciendo como parámetro la matrícula obtenida del fichero de configuración (por ejemplo

http://cyberc.satrdlab.upv.es:8080/cybercontainer/api/contenedores/BMOU8710633). Una vez ha obtenido estos datos, el programa crea un objeto de la clase *ScheduledExecutorService* indicando la tarea a realizar periódicamente y la frecuencia de tiempo en que tendrá que realizarla, que será la obtenida de la API de gestión.

```
//ScheduledExecutorService
scheduler = Executors.newSingleThreadScheduledExecutor();
Runnable task = new Main();
int initialDelay = 0;
FrecuenciaNotificacionMinutos = Api.getFrecuenciaNotificacion(apiGestionEndpoint, matricula, defaultFrecuenciaNotificacionMinutos);

//Inicia el servicio cada X minutos
scheduler.scheduleAtFixedRate(task, initialDelay, FrecuenciaNotificacionMinutos,
        TimeUnit.MINUTES);
```

Figura 45. Código de un ExecutorService en Java

La tarea que tiene que realizar periódicamente el ExecutorService se ha programado dentro del método *Run* de la clase Main. En primer lugar, se realiza una consulta a Fiware Orion mediante una petición HTTP POST a la url de Orion e indicando expresamente que se quiere obtener la entidad correspondiente al contenedor (la matrícula se indica mediante un filtro en la url) y sus campos dateModified y ContenedorSensorId de los metadatos de la entidad (por ejemplo http://cyberc.satrdlab.upv.es:1026/v2/entities?type=BMOU8710633&metadata=dateModified,ContenedorSensorId). Esta petición tendrá como respuesta la entidad en formato JSON (apartado 4.3.4) que corresponda con el contenedor cuya matrícula sea la anteriormente introducida.

```
public static JsonArray getOrionEntityJson(String apiOrionEndpoint, String fiwareService, String fiwareServicePath, String matricula) {
    String response="";
    JsonArray orionResponseJson = new JsonArray();

    try {
        URL url = new URL(apiOrionEndpoint+"/v2/entities?type="+matricula+"&metadata=dateModified,ContenedorSensorId");
        HttpURLConnection con = (HttpURLConnection) url.openConnection();
        con.setRequestProperty("Accept", "application/json");
        con.setRequestProperty("Fiware-Service", fiwareService);
        con.setRequestProperty("Fiware-ServicePath", fiwareServicePath);

        // Reading the result
        BufferedReader in = new BufferedReader(new InputStreamReader(
                con.getInputStream()));

        String linea;
        while ((linea = in.readLine()) != null) {
            response += linea;
        }
        orionResponseJson = new JsonParser().parse(response).getAsJsonArray();
        logger.info("Respuesta del servidor: " + con.getResponseCode() + "  " + con.getResponseMessage());
        con.disconnect();
    } catch (Exception e) {
        //e.printStackTrace();
        logger.error(e.getMessage());
    }

    return orionResponseJson;
}
```

Figura 46. Código de la obtención de una entidad de Orion en Java

En segundo lugar, se "parsea" (analiza) la entidad obtenida con formato JSON utilizando la librería Gson de Google para extraer los datos necesarios para poder construir un evento con formato JSON igual al que se ha descrito en el apartado 4.3.2. Los datos que se extraen de la entidad son: posición GPS (latitud y longitud) y de cada sensor del contenedor (atributo de la entidad) su valor, el id del sensor (ContenedorSensorId) y su fecha de modificación (dateModified). Para este proceso se han creado unos objetos de Java llamados *Evento* y *EventoSensor*, utilizados para representar el formato en JSON de un Evento en un objeto de Java.

```java
//Root JSON object
JsonObject root =  orionResponse.get(0).getAsJsonObject();

//Obtener informacion de la Posicion GPS
double lon = root.get("PosicionGPS").getAsJsonObject().get("value").getAsJsonObject().get("coordinates")
        .getAsJsonArray().get(0).getAsDouble();
double lat = root.get("PosicionGPS").getAsJsonObject().get("value").getAsJsonObject().get("coordinates")
        .getAsJsonArray().get(1).getAsDouble();

//Obtener datos de los sensores
Set<String> keys = root.keySet();
String[] elements = keys.toArray(new String[] {});

for(int i=0; i<elements.length; i++) {
    if(!elements[i].equals("id") && !elements[i].equals("type") && !elements[i].equals("type") && !elements[i].equals("PosicionGPS")) {
        EventoSensor eventoSensor = new EventoSensor();

        int ContenedorSensorId = root.get(elements[i]).getAsJsonObject().get("metadata").getAsJsonObject().get("ContenedorSensorId")
                .getAsJsonObject().get("value").getAsInt();
        String FHRegistro = root.get(elements[i]).getAsJsonObject().get("metadata").getAsJsonObject().get("dateModified")
                .getAsJsonObject().get("value").getAsString();
        String Valor = root.get(elements[i]).getAsJsonObject().get("value").getAsString();

        eventoSensor.setValor(Valor);
        eventoSensor.setContenedorSensorId(ContenedorSensorId);
        eventoSensor.setFHRegistro(FHRegistro);
        eventoSensor.setLat(lat);
        eventoSensor.setLon(lon);

        eventosSensoresList.add(eventoSensor);
    }
}

evento.setEventosSensor(eventosSensoresList.toArray(new EventoSensor[] {}));
```

Figura 47. Código del paseo de una entidad de Orion en Java

Una vez se han obtenido los datos necesarios para la creación de un evento en JSON, se procede a su construcción mediante el uso de la librería Gson. Finalmente, se envía este evento a la API del cliente como el cuerpo de una petición HTTP POST a la url especificada en el archivo de configuración del programa. Toda la información que se genera durante la ejecución del programa (la traza de ejecución) se almacena en un fichero de registro, generándose un fichero nuevo cada 24 horas que contiene la traza de ejecución del programa durante ese día. Para implementar esta funcionalidad se ha utilizado la librería log4j de Apache.

```java
//Construccion JSON del Evento
JSONObject eventoJson = new JSONObject();
JSONArray EventosContenedores = new JSONArray();
JSONObject eventosContenedor = new JSONObject();
JSONArray EventosSensorJson = new JSONArray();

//Fecha del evento
Format formatter = new SimpleDateFormat("yyyy-MM-dd'T'hh:mm:ss.SSS'Z'");
String FHEvento = formatter.format(new Date());
evento.setFHEvento(FHEvento);

//Elementos globales: FHEvento y Matricula
eventoJson.put("FHEvento", evento.getFHEvento());
eventosContenedor.put("Matricula", matricula);

//Elementos de cada sensor
for(int i=0; i<evento.getEventosSensor().length; i++) {
    JSONObject eventoSensorJson = new JSONObject();

    //Introudccion de la Posicion GPS
    JSONObject PosicionGPS = new JSONObject();
    JSONArray coordinates = new JSONArray();
    coordinates.add(0, evento.getEventosSensor()[i].getLon());
    coordinates.add(1, evento.getEventosSensor()[i].getLat());
    PosicionGPS.put("type", "Point");
    PosicionGPS.put("coordinates", coordinates);

    //Introduccion de valores
    eventoSensorJson.put("FHRegistro", evento.getEventosSensor()[i].getFHRegistro());
    eventoSensorJson.put("PosicionGPS", PosicionGPS);
    eventoSensorJson.put("ContenedorSensorId", evento.getEventosSensor()[i].getContenedorSensorId());
    eventoSensorJson.put("Valor", evento.getEventosSensor()[i].getValor());
    EventosSensorJson.add(eventoSensorJson);
}

eventosContenedor.put("EventosSensor", EventosSensorJson);
EventosContenedores.add(eventosContenedor);
eventoJson.put("EventosContenedor", EventosContenedores);
```

Figura 48. Creación de un evento con formato JSON en Java

```
11/jun/2018 12:34:00.712 [main]- INFO  - Main Cargando configuración del fichero: configuration.p
roperties
11/jun/2018 12:34:00.716 [main]- INFO  - Api Buscando la frecuencia de notificación para el conte
nedor con matrícula: BMOU8710633
11/jun/2018 12:34:00.791 [main]- INFO  - Api 200 OK - Petición correcta a la API de gestión: matr
ícula correcta
11/jun/2018 12:34:00.793 [pool-2-thread-1]- INFO  - Main Cargando configuración del fichero: conf
iguration.properties
11/jun/2018 12:34:00.793 [pool-2-thread-1]- INFO  - Main Iniciando el gestor de eventos del conte
nedor con matrícula: BMOU8710633
11/jun/2018 12:34:00.793 [pool-2-thread-1]- INFO  - Main Obteniendo los datos de FIWARE-Orion del
 contenedor con matrícula: BMOU8710633
11/jun/2018 12:34:00.795 [pool-2-thread-1]- INFO  - Api Respuesta del servidor: 200  OK
11/jun/2018 12:34:00.799 [pool-2-thread-1]- INFO  - Main Enviando evento a la API de Infoport...
11/jun/2018 12:34:00.800 [pool-2-thread-1]- INFO  - Main Evento a enviar:
{"FHEvento":"2018-06-11T12:34:00.799Z","EventosContenedor":[{"Matricula":"BMOU8710633","EventosSe
nsor":[{"ContenedorSensorId":4019,"Valor":"1","PosicionGPS":{"coordinates":[-0.308595436,39.43556
368],"type":"Point"},"FHRegistro":"2018-06-11T10:30:34.00Z"},{"ContenedorSensorId":4020,"Valor":"
14","PosicionGPS":{"coordinates":[-0.308595436,39.43556368],"type":"Point"},"FHRegistro":"2018-06
-11T10:33:59.00Z"}]}]}
11/jun/2018 12:34:00.930 [pool-2-thread-1]- INFO  - Api Respuesta del servidor: 201  Created
11/jun/2018 12:34:00.931 [pool-2-thread-1]- INFO  - Main 2018-06-11T12:34:00.931+0200
```

Figura 49. Ejemplo de la traza de ejecución del gestor de eventos

A modo de resumen, en la figura 50 se puede observar el flujograma del Gestor de eventos, mostrándose en una sola imagen todo el proceso descrito anteriormente.

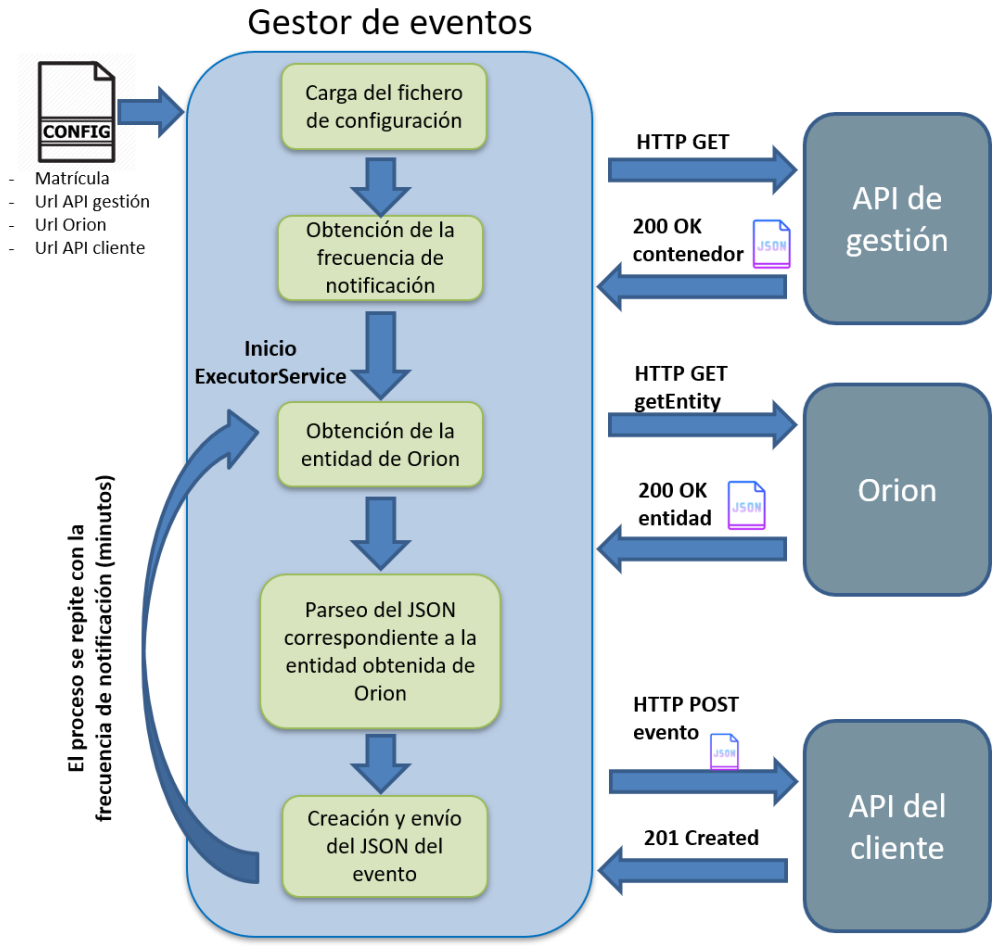

Figura 50. Flujograma del Gestor de eventos

5.6 Resumen/esquema completo de la implementación

En este último apartado del capítulo 5 de la memoria, se va a realizar un resumen de toda la implementación que se ha descrito en el capítulo. En la figura 51 se puede observar un esquema que incluye la implementación completa del sistema, con todos los componentes que se han utilizado y la interacción entre ellos. Además, se puede observar la arquitectura final de la máquina virtual Ubuntu que actúa como servidor en la nube, con los puertos utilizados y los componentes que se han instalado.

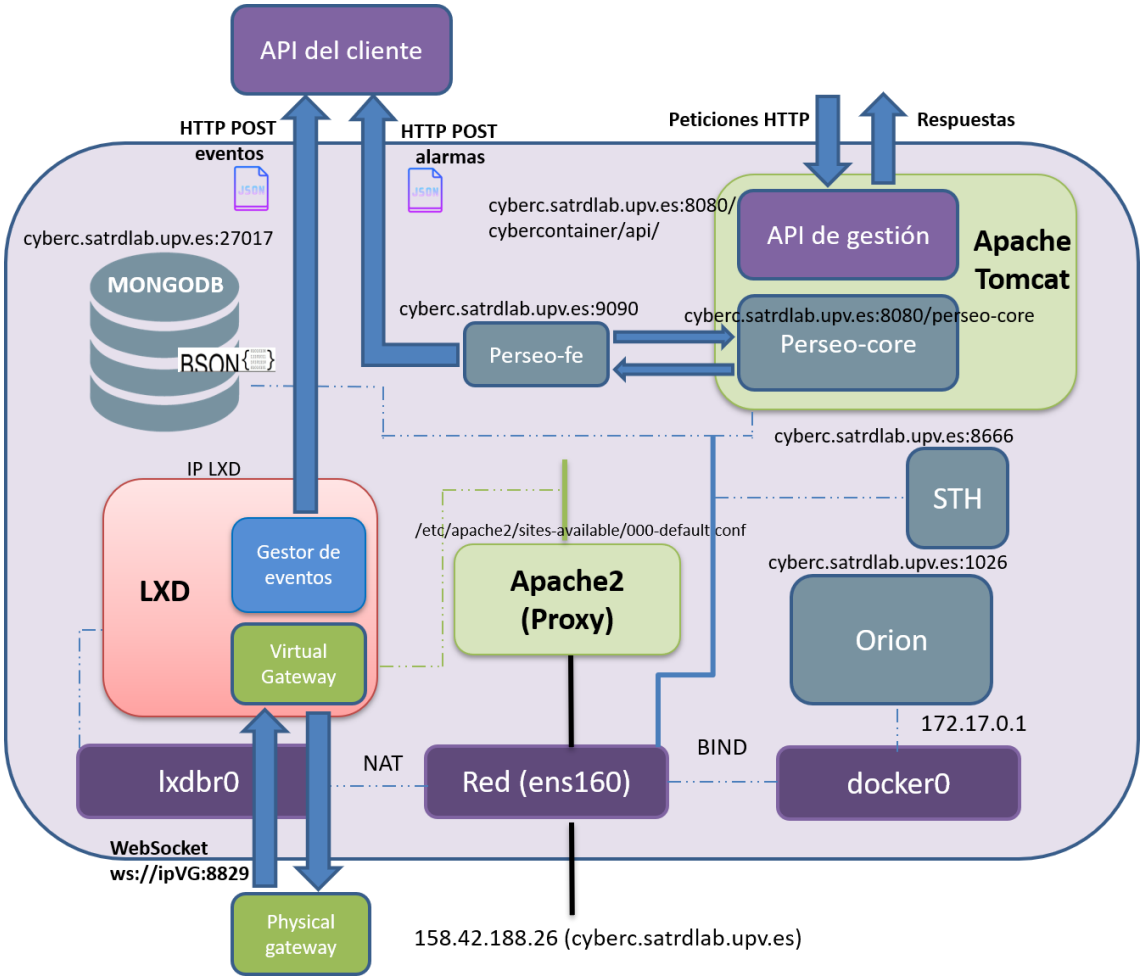

Figura 51. Esquema de resumen de la implementación del sistema

Capítulo 6. Prototipo

Una vez se ha implementado completamente todo el sistema, el siguiente paso es realizar un prototipo o una demo de laboratorio para comprobar que el sistema funciona correctamente, ya que no se disponen de contenedores de carga reales para monitorizarlos. Como se ha mencionado en la introducción, en este prototipo el cliente será Infoport Valencia, empresa pública que ha dado acceso a su tecnología para la realización de pruebas, por lo que se utilizará su propia API para enviar los eventos y las alarmas de los contenedores.

6.1 Contenedores

En esta demo se han utilizado cuatro contenedores con matrículas reales que se muestran en la tabla 1.

Matrícula	Tipo de contenedor	Operación	Sensores
AMCU9261477	45R1	Almacenado en puerto	Posición Luminosidad Temperatura
AMCU9275700	2232	Almacenado en puerto	Posición Luminosidad Temperatura
BMOU9226583	4532	Trayecto terrestre	Posición Luminosidad Temperatura
BMOU8710633	22R1	Trayecto marítimo	Posición Luminosidad Temperatura

Tabla 1. Contenedores que se utilizan en la demo

Los dos primeros contenedores son contenedores con una posición fija, representando contenedores almacenados en el puerto de Valencia; mientras que los otros dos contenedores son contenedores que se encuentran en tránsito. El contenedor con matrícula BMOU9226583 representa un contenedor en un trayecto terrestre (por ejemplo, transportado por un camión) y el contenedor con matrícula BMOU8710633 representa un contenedor en un trayecto marítimo.

6.2 Montaje

Para simular un entorno con contenedores reales, se ha realizado un montaje con los siguientes elementos:

- **2 Raspberry Pi**: una Raspberry Pi es un sistema embebido de bajo coste que se ha utilizado para simular un contenedor físico. El sistema operativo que se ha instalado en las Raspberrys es Raspbian, una distribución de Linux especialmente desarrollada para la Raspberry Pi. A la Raspberry se conectan los diferentes sensores y, en su

interior, se ejecuta la parte física de la gateway. Los contenedores simulados por las Raspberrys son los dos primeros de la tabla 1 (AMCU9261477 y AMCU9275700). Asimismo, los sensores de estos contenedores son simulados.

- **Un PC**: la utilización de un pc es necesaria para controlar la ejecución de las diferentes partes de la demo (gateways, simuladores, Orion, …) y comprobar que los contenedores se monitorizan adecuadamente. Además, en este pc se ejecutan dos simuladores correspondientes a los dos últimos contenedores de la tabla 1 (BMOU9226583 y BMOU8710633), ya que no se dispone de más Raspberry Pis.
- **Un router 4G**: este router es el encargado de proporcionar conexión a internet a las dos Raspbery Pis y al pc, para poder enviar los datos a la plataforma que se encuentra en la nube.
- **El servidor**: obviamente, se necesita el servidor cuya implementación se ha descrito en el capítulo anterior para probar el funcionamiento de la plataforma.

Figura 52. Montaje: router 4G y Raspberry Pis

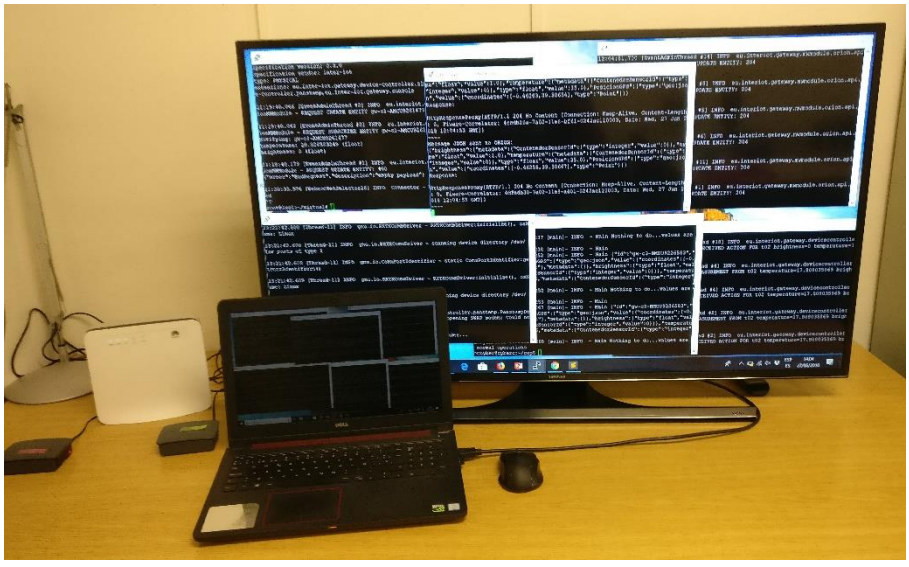

Figura 53. Montaje completo del prototipo

6.3 Gestión de los contenedores utilizando la API de gestión

Una vez realizado el montaje completo del sistema, el siguiente paso es dar de alta los contenedores en el sistema. En este apartado, se va a comprobar el funcionamiento de la API de gestión mediante el uso de su interfaz gráfica de Swagger, dando de alta un contenedor en el sistema, obteniendo sus datos y, finalmente, eliminándolo del sistema. El contenedor que se describe en esta prueba es el de matrícula BMOU8710633, con una frecuencia de notificación de 2 minutos, estado habilitado y con dos sensores: uno de luminosidad y otro de temperatura, con sus correspondientes umbrales (el sensor de posición se incluye por defecto en todos los contenedores). No obstante, el proceso es el mismo para todos los contenedores.

El primer paso es añadir el contenedor al sistema mediante el método HTTP POST, añadiendo el JSON del contenedor como cuerpo de la petición.

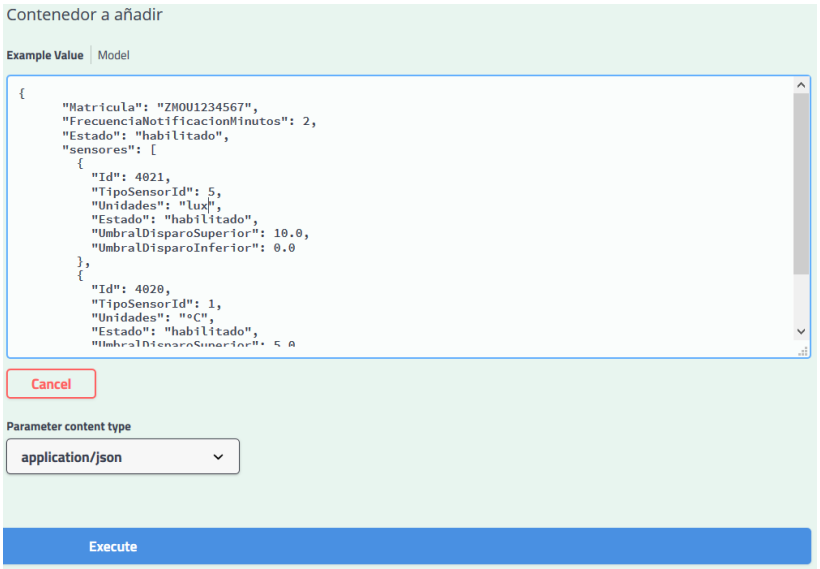

Figura 54. Creación de un contenedor utilizando el Swagger de la API de gestión

Como la respuesta de la petición ha sido un 201 Created, el contenedor se ha añadido correctamente al sistema. Para comprobar que se ha añadido correctamente, se ha realizado una petición a la API con el método HTTP GET para que devuelva la información sobre el contenedor que se ha creado.

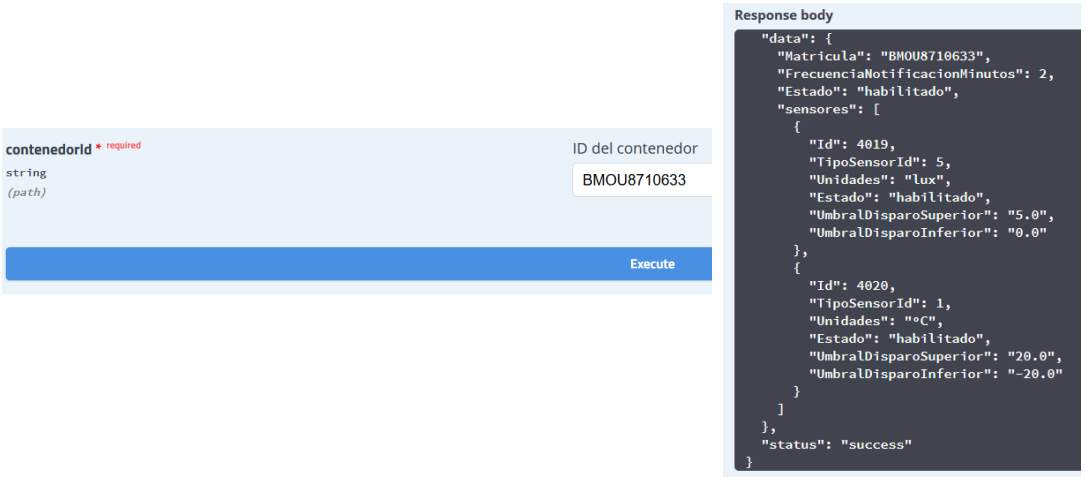

Figura 55. Obtención de un contenedor utilizando el Swagger de la API de gestión

Además, también se ha comprobado que el contenedor se ha creado como una nueva entidad de Orion con su modelo de datos correspondiente. Esto se comprueba mediante una petición HTTP GET a Orion, indicando que se quiere obtener expresamente el contenedor que se ha dado de alta.

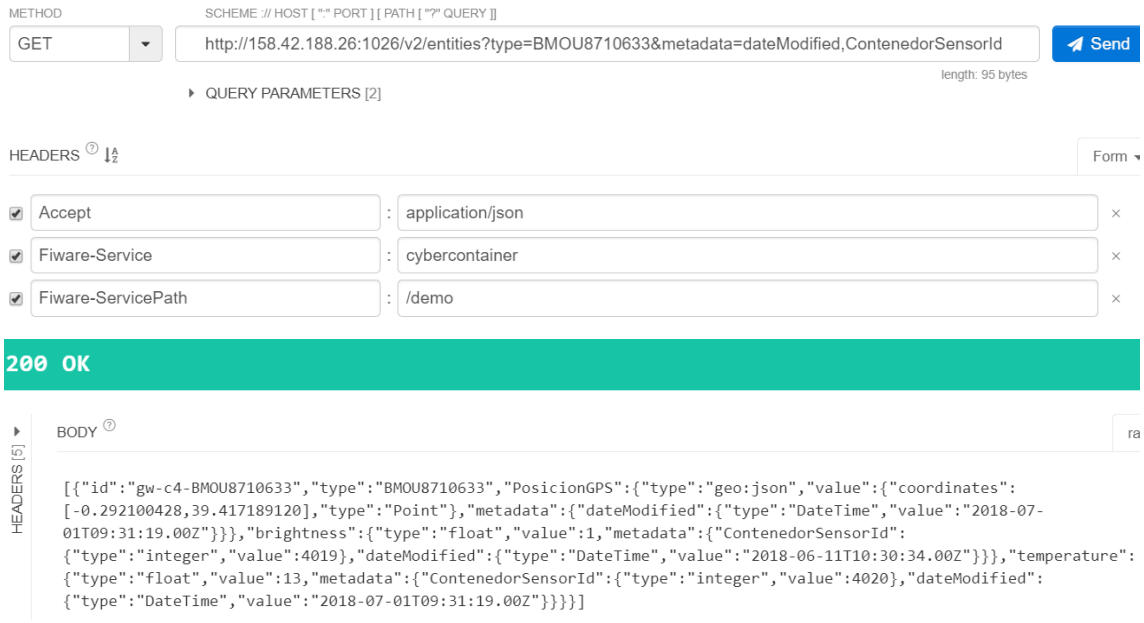

Figura 56. Obtención de los datos de un contenedor registrados en FIWARE Orion

Cuando ya no se desee monitorizar un contenedor, éste se debe de eliminar del sistema realizando una petición HTTP DELETE con la matrícula del contenedor. Para realizar esta comprobación, se va a eliminar el contenedor con matrícula ZMOU1234567, ya que el contenedor con matrícula BMOU8710633 se necesita para la realización de este demostrador.

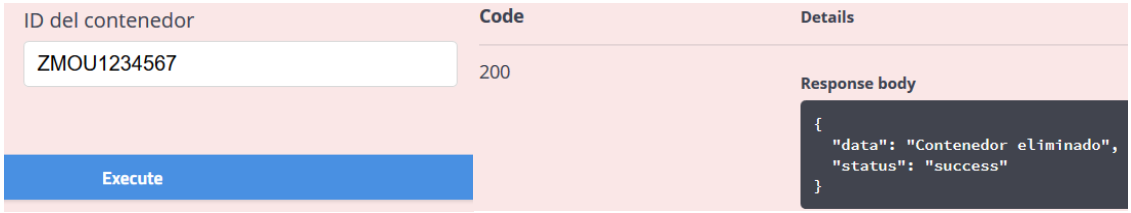

Figura 57. Eliminación de un contenedor utilizando el Swagger de la API de gestión

En la figura 58 se puede comprobar que el contenedor con matrícula ZMOU1234567 no se encuentra en el sistema.

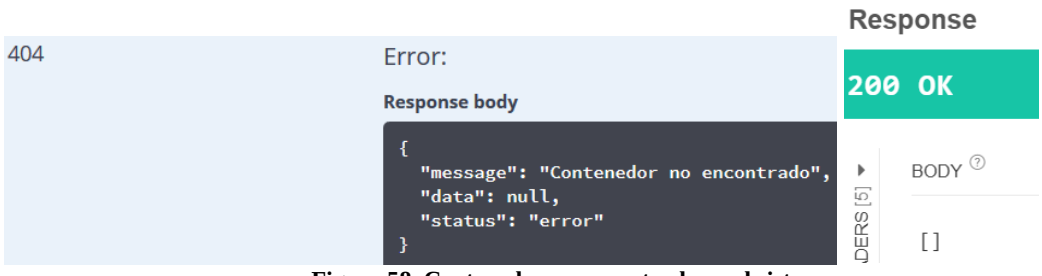

Figura 58. Contenedor no encontrado en el sistema

6.4 Gateway de comunicaciones

En este punto, los contenedores han sido dados de alta en el sistema, por lo que el siguiente paso es preparar el sistema para que los sensores empiecen a enviar datos a la plataforma. Como se ha expuesto anteriormente, solo se disponen de dos Raspberry Pis, que simulan los contenedores con matrículas AMCU9261477 y AMCU9275700, respectivamente. Por lo tanto, solo se va a emplear contenedores LXD (con direcciones IP 10.0.10.18 y 10.0.10.160, respectivamente) y la gateway de comunicaciones para estos dos contenedores. El contenedor que se describe en este apartado es el AMCU9261477, ya que el proceso es el mismo para los dos contenedores.

En primer lugar, se ha iniciado la gateway virtual en el contenedor virtual LXD1 mediante el comando `bash bin/run` con la configuración adecuada (figura 42), indicando la url de Orion del servidor en la nube). Una vez iniciada, se mantiene a la espera de nuevas conexiones por parte de la gateway física.

En segundo lugar, se ha iniciado la gateway física en la Raspberry Pi mediante el comando `bash bin/run` con la configuración adecuada (figura 41), indicando la dirección IP del LXD1 (158.42.188.26/lxc1, ya que se debe de acceder a través del proxy), el puerto 8829 para la utilización del protocolo WebSocket y que se va a utilizar el controlador para simuladores de sensores.

El siguiente paso es conectar los dispositivos a la gateway física, que en este caso se trata de un simulador de un sensor de temperatura y un simulador de un sensor de luminosidad (no se ha incluido un simulador de un receptor GPS al tener estos contenedores una posición fija). Previamente a su conexión, se deben de configurar estos simuladores. Se ha creado en la gateway física un dispositivo llamado AMCU9261477 y con un identificador t01, que incluye un sensor simulado de temperatura y de luminosidad. Para conectar este dispositivo, se ha ejecutado el comando `device connect t01` en la consola de la gateway física.

Con la utilización del comando `device list` se puede obtener un listado de los dispositivos conectados a la gateway con sus propiedades.

```
gateway> device list
UUID                         NAME                         CONTROLLER
STATE
------------------------     ------------------------     ------------------------
------------------------
t01                          AMCU9261477                  simulator
DISCONNECTED
```

Figura 59. Lista de dispositivos conectados a la gateway física

Cuando el dispositivo se conecta correctamente a la gateway física, ésta empieza recibir los datos del dispositivo, mostrando un mensaje en su consola que incluye el nombre y el valor enviado por el dispositivo. Seguidamente, la gateway física empieza a enviar estos datos a la gateway virtual, que procesará estos datos para posteriormente enviarlos a Orion. Cuando se realiza el envío de datos a Orion, en la consola de la gateway virtual se muestra el código de respuesta de las peticiones HTTP realizadas a Orion. En la figura 60 se pueden observar las consolas de las dos partes de la gateway durante el proceso de envío de datos por parte del dispositivo a la plataforma.

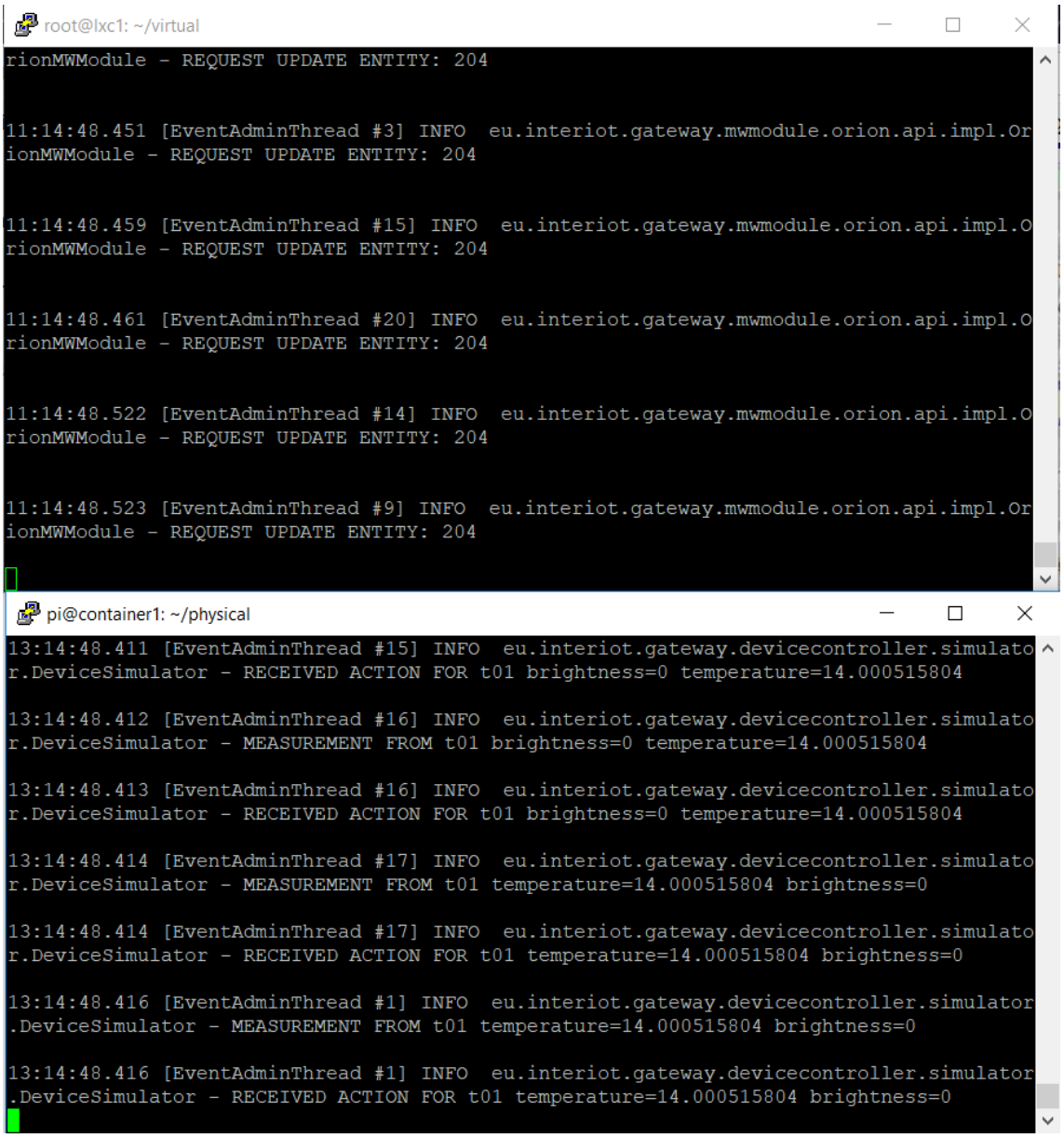

Figura 60. Consolas de las partes virtual y física de la gateway durante el envío de datos

Finalmente, es conveniente mencionar que cuando la gateway virtual se desconecta de la gateway física, en la consola de la gateway virtual se muestra un mensaje avisando de esta desconexión.

Figura 61. Mensaje de aviso de desconexión de la gateway física

6.5 Simuladores

Los contenedores con matrículas BMOU9226583 y BMOU8710633, no van a tener sus contenedores virtuales LXD correspondientes ni van a utilizar la gateway para el envío de datos, ya que no se dispone de más Raspberry Pis.

Se ha desarrollado un programa en el lenguaje de programación Java para que actúe como simulador de un sensor de temperatura, luminosidad y posición GPS. El funcionamiento de este programa consiste en la creación de valores aleatorios dentro de un rango determinado para emular los valores reales que mediría un sensor real. Para la emulación de la posición GPS, el programa utiliza archivos que contienen una traza real de un trayecto con formato GPS eXchange Format (GPX). Cada un determinado tiempo en segundos (en esta demostración cada 5 segundos), el programa envía los valores de los simuladores a Orion mediante un *updateEntity* (petición HTTP PATCH).

Los parámetros de configuración de este programa se introducen a través de la línea de comandos cuando se ejecuta, introduciéndose, entre otros parámetros, la matrícula del contenedor. Además, toda la información que se genera durante la ejecución del programa (la traza de ejecución) se almacena en un fichero de registro, generándose un fichero nuevo cada 24 horas. Para implementar esta funcionalidad se ha utilizado la librería log4j de Apache.

En el caso del contenedor BMOU9226583, se ha utilizado un fichero GPX (TrayectoTerrestre.gpx) que contiene un trayecto por carretera que empieza en el Polígono Industrial Fuente del Jarro y acaba en la terminal de contenedores del puerto de Valencia.

En el caso del contenedor BMOU8710633, se ha utilizado un fichero GPX (TrayectoMaritimo.gpx) que contiene un trayecto marítimo que empieza en el puerto de Valencia y acaba en el puerto de Ibiza.

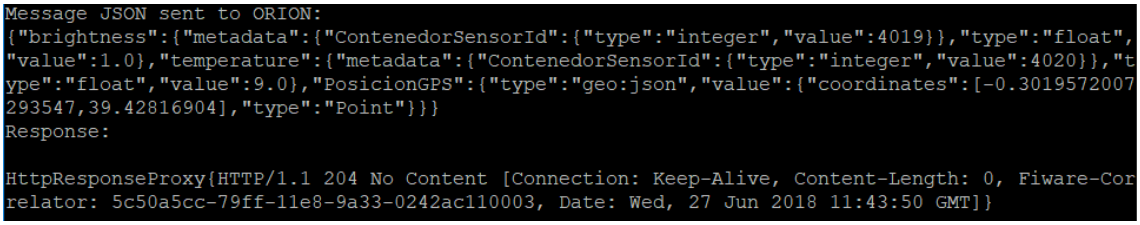

Figura 62. Ejemplo de la traza de ejecución del simulador

6.6 Monitorización en una interfaz gráfica

Llegados a este punto, los contenedores han sido dados de alta en el sistema y se envían sus valores a la plataforma para su tratamiento. El gestor de eventos y FIWARE Perseo empiezan a generar eventos y alarmas, respectivamente, cuando se empiezan a recibir los datos de los contenedores. Se puede afirmar que el prototipo montado tiene una funcionalidad completa, por lo que en este punto es necesario comprobar que la información introducida y generada por la plataforma es correcta.

Para realizar esta comprobación, se ha desarrollado una pequeña aplicación web (desarrollada con los lenguajes HTML, Javascript y CSS) alojada en el servidor web Apache, a la que se puede acceder mediante la url http://cyberc.satrdlab.upv.es/demo_pos_cyber2.html. En esta aplicación web, se puede seleccionar uno de los cuatro contenedores que se han utilizado en este demostrador. Del contenedor seleccionado se muestran los valores de la posición (latitud y longitud) y de la temperatura en tiempo real, obtenidos directamente de Orion. Además, se muestra la posición en tiempo real mediante un marcador sobre un mapa del tipo OpenStreetMap, que es de código abierto.

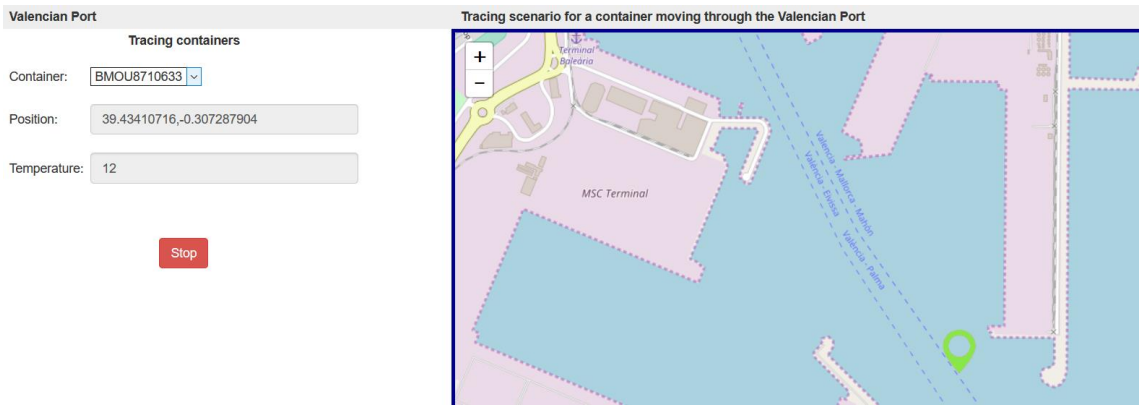

Figura 63. Aplicación web de monitorización de contenedores

Por otra parte, cuando el valor de la temperatura supere uno de los umbrales definidos en la definición del contenedor, se mostrará un aviso de que se ha generado una alarma.

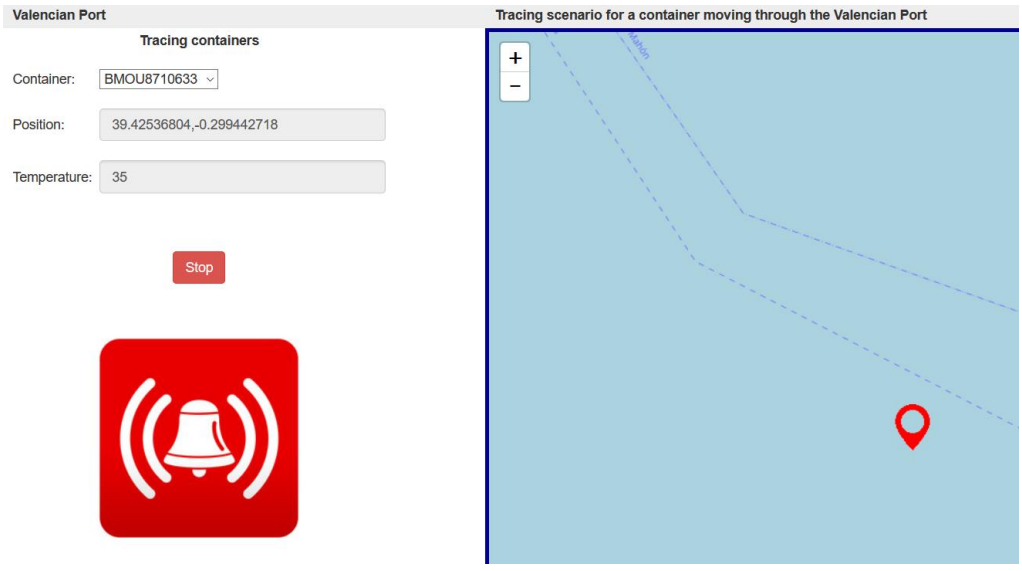

Figura 64. Alarma en la aplicación web de monitorización de contenedores

6.7 Visionado de los datos en la plataforma del cliente

En el apartado anterior se ha podido comprobar la validez de los datos que contiene y genera la plataforma mediante su visualización en una interfaz gráfica generada por una aplicación web, que obtiene los datos de Orion en tiempo real. No obstante, no se han comprobado los eventos y las alarmas generadas por la plataforma.

Como se ha mencionado anteriormente, estos eventos y alarmar se generan para enviarlos mediante peticiones HTTP POST a las plataformas propias de los clientes. En esta demostración el cliente se trata de Infoport Valencia, por lo que la plataforma ha sido configurada para enviar los eventos y las alarmas a la API de la plataforma de Infoport, cuya url no se puede incluir en esta memoria por razones de seguridad y confidencialidad. Sin embargo, se va a comprobar si los eventos y las alarmas han sido generadas correctamente mediante la visualización de los eventos y alarmas en la interfaz gráfica de la plataforma propia de Infoport.

En las figuras 65 y 66 se puede observar que los eventos y las alarmas se reciben correctamente en la plataforma del cliente.

Monitorización

Matrícula	Variable	Valor	Unidad	Fecha	Latitud	Longitud
AMCU9261477	Temperatura	17.114232583	Grados Celsius	12/06/2018	39.457832	-0.320699
AMCU9261477	Luminosidad	0	Lux	11/06/2018	39.457832	-0.320699
AMCU9275700	Temperatura	15.258159107	Grados Celsius	12/06/2018	39.457832	-0.320699
AMCU9275700	Luminosidad	0	Lux	11/06/2018	39.457832	-0.320699
BMOU8710633	Temperatura	10	Grados Celsius	12/06/2018	39.42839312	-0.30215835
BMOU8710633	Luminosidad	1	Lux	11/06/2018	39.42839312	-0.30215835
BMOU9226583	Temperatura	6	Grados Celsius	12/06/2018	39.4514703	-0.31366074
BMOU9226583	Luminosidad	1	Lux	11/06/2018	39.4514703	-0.31366074

Figura 65. Visualización de eventos en la plataforma de Infoport

Alarmas

Descripción	Contenedor	Tipo Sensor	Valor	Unidad	Fecha	Latitud	Longitud
Umbral Superado	BMOU8710633	Temperatura	7	Grados Celsius	2018-06-29T18:45...	39.4549639	-0.3279804

Figura 66. Visualización de alarmas en la plataforma de Infoport

Capítulo 7. Conclusiones y líneas futuras

7.1 Conclusiones

En este trabajo final de máster de carácter profesional se desarrollado una plataforma de gestión de contenedores basada en sistemas ciberfísicos y tecnologías IoT. Se ha avanzado hacia un modelo de contenedor inteligente, ya que se han modelado los contenedores como sistemas ciberfísicos, teniendo cada contenedor de carga físico su propia representación virtual como contenedor de Linux LXD.

Para poder llevar a cabo este proyecto, previamente se ha tenido que realizar un estudio exhaustivo del estado del arte relacionado con este proyecto. Se han estudiado los tipos de sensores aplicables a un contenedor que se pueden encontrar en el mercado, las soluciones existentes de tracking de contenedores para observar las características que se pueden implementar y aspectos que se pueden mejorar, las tecnologías de comunicaciones en redes LPWAN, las opciones de virtualización para crear los contenedores físicos y las plataformas existentes de IoT en el mercado. Este estudio ha sido interesante desde el punto de vista de un estudiante, ya que proporciona una base en el campo tan interesante y novedoso del Internet of Things y de los sistemas ciberfísicos. También es interesante destacar que la realización de este estudio ha ocupado una parte importante del tiempo de desarrollo de este proyecto.

Profundizando en la plataforma de IoT FIWARE, gracias al previo estudio de demás plataformas unido a que es un estándar financiado por la Comisión Europea, se ha utilizado su Context Broker en detrimento de otras opciones como Rabbit MQ o Apache Kakfa. Además, para mantener una cierta coherencia en el sistema, también se ha utilizado su gestor de datos históricos (STH) y su CEP (Perseo).

La creación de una máquina virtual basada en Linux desde cero para que funcione en una máquina que actúa como servidor real es una tarea muy interesante desde el punto de vista profesional, ya que en el ámbito de los sistemas basados en cloud se debe de tener conocimientos de gestión de servidores. Además, la mayoría de los servidores utilizan distribuciones de Linux como sistema operativo.

Una parte muy importante en el desarrollo de este sistema ha sido la creación de un modelo de datos que actúe como un estándar en todos los procesos de intercambio de datos de la plataforma. En la creación del modelo de datos, se ha intentado mantener un equilibrio entre su legibilidad por parte del usuario (en otras palabras, que sea fácil de leer y entender) y su compatibilidad entre los diferentes elementos de la plataforma.

La mayor parte del tiempo en este proyecto se ha utilizado en la implementación del sistema, ya que la plataforma está integrada por elementos muy diferentes entre sí, pero que a su vez tienen que comunicarse entre ellos.

Es necesario destacar que se han cumplido todos los objetivos previstos inicialmente y que el sistema funciona correctamente, pero aún está en una fase de desarrollo, ya que hay aspectos que mejorar y nuevas funcionalidades que se pueden implementar.

Por último, personalmente me gustaría destacar que ha sido muy interesante realizar un trabajo final de máster de carácter profesional, ya que el resultado final es un producto que funciona realmente a diferencia de un producto teórico que se tendría que implementar en un futuro. Asimismo, la realización de este proyecto me ha supuesto realizar un proceso de aprendizaje que

se ha iniciado con un estudio teórico, para finalmente aplicar los conocimientos adquiridos en la implementación de un sistema real.

7.2 Líneas futuras

La realización de un proyecto tecnológico abre un abanico infinito de propuestas y mejoras, por la razón de que la tecnología, y más aún la basada en IoT, se encuentra en un proceso continuo de mejora y desarrollo. Además, en este caso en concreto cuando los usuarios utilicen la plataforma proporcionarán un *feedback* al desarrollador con la propuesta de posibles mejoras y la detección de algunos errores.

Una de las líneas futuras de desarrollo enmarcadas en este proyecto, es el desarrollo de una plataforma propia en la que se puedan visualizar los datos generados por el sistema de gestión de contenedores, ya que en este proyecto se ha optado por enviar los datos directamente a la plataforma propia del cliente. Esta plataforma recibiría los eventos y las alarmas generadas en el sistema y las mostraría en una interfaz de usuario amigable para cualquier usuario. Una solución sería utilizar el framework de desarrollo web Django.

Esta plataforma está preparada para utilizar solamente sensores, por lo que no hay una comunicación bidireccional entre la parte física y la virtual. Por lo tanto, se podría actualizar la plataforma para la inclusión de actuadores, que reciban mensajes de la plataforma para actuar en consecuencia. Esta funcionalidad es interesante, ya que las plataformas IoT actuales están avanzando hacia la utilización de un modelo de sensor/actuador.

Como la plataforma está enfocada a su uso en contenedores de carga marítimos, convendría preparar la plataforma para su integración en plataformas propias de los puertos o en plataformas de Smart Cities. Un claro ejemplo sería la integración de esta plataforma con el Port Community System que utiliza el puerto de Valencia, una tecnología que gestiona el flujo de información del puerto de Valencia (salidas, llegadas, gestión de mercancías, …).

Finalmente, un aspecto importante a mejorar es la seguridad del sistema. En FIWARE existe la posibilidad de utilizar tokens de autentificación en las peticiones a su API.

Glosario de términos

API	Application Programming Interface
CEP	Complex Event Processing
CORS	Cross-Origin Resource Sharing
CPS	Cyber Physical System
EPL	Event Processing Language
GPS	Global Positioning System
HMI	Human Machine Interface
HTTP	Hypertext Transfer Protocol
HTML	HyperText Markup Language
IANA	Internet Assigned Numbers Authority
IDE	Integrated Development Environment
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IoT	Internet of Things
ISO	International Organization for Standardization
JSON	JavaScript Object Notation
LPWAN	Low-Power Wide-Area Network
REST	Representational State Transfer
RFC	Request for Comments
SATDR	Grupo de Sistemas y Aplicaciones de Tiempo Real Distribuidos
STH	Short Time Historic

Bibliografía

[1] https://blogthinkbig.com/8-400-millones-de-dispositivos-estaran-conectados-a-internet-a-finales-de-2017

[2] Autoridad Portuaria de Valencia, "Boletín estadístico APV Diciembre de 2017", https://www.valenciaport.com/wp-content/uploads/Bolet%C3%ADn-Estad%C3%ADstico-Diciembre-2017.pdf [Online].

[3] Gobierno de los Estados Unidos, *Información oficial relativa al Sistema de Posicionamiento Global,* https://www.gps.gov/ [Online]

[4] Adafruit Inudstries, *Adafruit Ultimate GPS HAT for Raspberry Pi* https://www.adafruit.com/product/2324 [Online]

[5] Omega engineering, *¿Qué es un sensor termopar?* https://es.omega.com/prodinfo/termopares.html

[6] Tablas de referencia del termopar tipo K: https://es.omega.com/temperature/pdf/Type_K_Thermocouple_Reference_Table.pdf

[7] Sensirion sensors, *Digital Humidity Sensor SHT7x*, https://www.sensirion.com/en/environmental-sensors/humidity-sensors/pintype-digital-humidity-sensors/ [Online]

[8] Omega engineering, *¿Qué es un acelerómetro?* https://es.omega.com/prodinfo/acelerometro.html

[9] Rs store, *Sensor ultrasónico RS Pro,* https://es.rs-online.com/web/p/sensores-de-proximidad-ultrasonicos/2370799/

[10] LoRa Alliance, *What is the LoRaWAN Specification?* https://www.lora-alliance.org/about-lorawan [Online]

[11] LoRa Alliance, *A technical overview of LoRa and LoRaWAN,* November 2015

[12] LoRa Alliance, *LoRaWAN 101A Technical Introduction*, 2015

[13] Sigfox, *Sigfox technology overview* https://www.sigfox.com/en/sigfox-iot-technology-overview [Online]

[14] Carles Antón-Haro y Mischa Doler, *Machine-to-machine (M2M) Communications*, 2015

[15] Pilar Andrés-Maldonado y PabloAmeigeras, *NarrowBand IoT Data Transmission Procedures for Massive Machine Type Communications*¸ Universidad de Granada

[16] Rohde&Schwarz, *NarrowBand Internet of Things Whitepaper*, https://www.rohde-schwarz.com/es/aplicaciones/internet-de-las-cosas-en-banda-estrecha-white-paper_230854-314242.html [Online]

[17] Activage Project, *Report on IoT European Platforms*, September 2017

[18] INTER-IoT, *Methods for Interoperability and Integration v.1*, December 2016

[19] Documentación oficial de FIWARE Orion: https://fiware-orion.readthedocs.io

[20] Documentación oficial de FIWARE STH Comet https://fiware-sth-comet.readthedocs.io

[21] Documentación oficial de FIWARE Perseo (CEP) http://fiware-iot-stack.readthedocs.io/en/latest/cep/index.html

[22] MongoDB Inc, *What is MongoDB?* https://www.mongodb.com/what-is-mongodb [Online]

[23] Universidad de Berkeley, *Cyber-physical Systems* https://ptolemy.berkeley.edu/projects/cps/ [Online]

[24] Docker Inc., *What is Docker*, https://www.docker.com/what-docker [Online]

[25] Docker Inc., *What is a container*, https://www.docker.com/what-container [Online]

[26] Canonical Group, *LXD Introduction,* https://linuxcontainers.org/lxd/# [Online]

[27] Canonical Group, *The no-nonsense way to accelerate your business with containers*, February 2017

[28] Maersk Line, *What is Remote Container Management,* https://www.maerskline.com/shipping/remote-container-management [Online]

[29] Loginno, *Smart container* https://loginno.com/ [Online]

[30] Traxens, *TRAXENS Technology*, http://www.traxens.com/en/technology [Online]

[31] Pointer Telocation LTD, *Cellocator*, https://www.cellocator.com/products/cellotrack/ [Online]

[32] https://www.mongodb.com/download-center

[33] https://github.com/telefonicaid/fiware-orion

[34] https://github.com/telefonicaid/fiware-sth-comet

[35] https://github.com/telefonicaid/perseo-fe

[36] https://github.com/telefonicaid/perseo-core

[37] IETF, *RFC 6455: The WebSocket Protocol*, https://tools.ietf.org/html/rfc6455 [Online]