UNIVERSIDAD DE LA REPÚBLICA URUGUAY

FACULTAD DE INGENIERÍA UDELAR

Campus Inteligente

Santiago Alles Conde

Proyecto de grado presentado a la Facultad de Ingeniería de la Universidad de la República
en cumplimiento parcial de los requerimientos para la obtención del título de Ingeniería en Computación.

Tutores:
Matias Richart
Eduardo Grampin

Tutores:
Federico Rodríguez
Gustavo Guimerans Martín Giachino

Montevideo, Uruguay Diciembre de 2022

Resumen

El objetivo general de este proyecto es definir las bases para el desarrollo del Campus Inteligente. Esto se refiere a definir las tecnologías y la arquitectura a utilizar en dicha iniciativa. El Campus Inteligente es un desarrollo de IoT (Internet of Things) que abarcaría los predios de las facultades de Ciencias Económicas y Administración; de Ingeniería y de Arquitectura de la Universidad de la República. A su vez se desea validar las decisiones tomadas mediante un prototipo de punta a punta.

Índice

1. Introducción

En este informe se presentará una propuesta para el desarrollo del **Campus Inteligente** en los predios de las facultades de Ciencias Económicas y Administración; de Ingeniería y de Arquitectura de la Universidad de la República.

El **Campus Inteligente** es una aplicación de tecnologías de IoT (Internet of Things), que constará en instalar sensores de diferentes tipos en las facultades, conectarlos mediante internet a un servidor y de esta manera disponibilizar los datos para ser consumidos por diferentes aplicaciones.

Empecemos hablando de la utilidad del internet, la cual va más allá de la comunicación o el entretenimiento. Ya que permite utilizar servicios como los pagos en línea o pedir entregas a domicilio, entre otros. Como consecuencia ha provocado que cambie la manera en que el mundo funciona. Esto ha convertido a el internet en una necesidad básica para millones de personas [9]. Además, hoy en día cada vez son más los dispositivos conectados a el internet. Se estimaba que para el año 2020 habría unos 50 billones de dispositivos conectados a el internet [38], pero en la actualidad existen aún más usos del internet entre los cuales se destaca el **IoT**.

En la actualidad existen muchas definiciones para el término IoT. Por ejemplo, según el *Institute of Electrical and Electronics Engineers (IEEE)* el IoT es la utilización de distintos protocolos de comunicación para conectar dispositivos físicos y virtuales de forma inequívoca. Esto genera una red configurable y accesible mediante el internet. [38].

Si bien hay diferencias entre las distintas definiciones existentes, tanto la presentada recientemente como las demás definiciones son amplias y siempre incluyen a dispositivos conectados entre sí mediante el internet. Cabe destacar que el hecho de que las definiciones sean amplias es un reflejo de la gran área de acción que posee el IoT. Como puede ser: casas inteligentes, ciudades inteligentes, entre otros.

1.1. Motivaciones

Se puede decir que hay tres grandes motivos detrás de la realización de este proyecto, las cuales son: el estado actual del IoT, los proyectos anteriores en el tema y el contexto mundial.

1.1.1. Estado del IoT

El IoT ha sido una gran innovación, la cual ha captado gran atención de la comunidad científica como menciona el artículo *Network optimizations in the Internet of Things: A review* [41]. Esto ha provocado que hayan muchos estudios dentro del área los cuales serán profundizados en la **Sección 3 Relevamiento de Arquitecturas y Tecnologías**. Además, el término sigue teniendo una gran relevancia como se puede ver en la Figura 1 que muestra las estadísticas del buscador Google [22]. Donde el 100 representa el momento en el cual se obtuvo el mayor número de búsquedas.

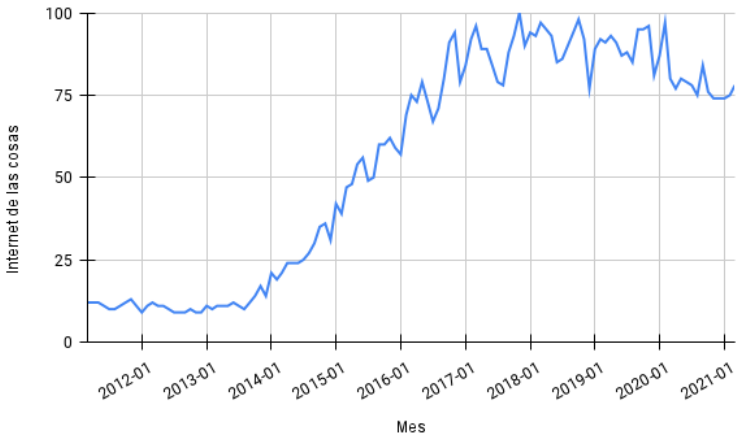

Figura 1: Interés en el Tema IoT en búsquedas de Google.

Por lo tanto una de las principales motivaciones para realizar este proyecto es relevar el estado del IoT en materia de protocolos de comunicación, plataformas de IoT y arquitecturas.

1.1.2. Esfuerzos anteriores

Otra motivación de este proyecto es el conectar los esfuerzos realizados en esta área en años anteriores. Lo que obviamente significa que este no es el primer esfuerzo en el campo del IoT realizado en la facultad.

Los esfuerzos anteriores serán presentados en la **Sección 2 Proyectos relacionados**. Estos proyectos desarrollaron dispositivos contadores de señales de Wi-Fi, evaluaciones de plataformas de IoT y un estudio de viabilidad del uso del IoT para localización de interiores.

1.1.3. Contexto mundial

El contexto mundial en el cual se realizó este proyecto también fue una motivación importante para el mismo. Ya que era muy peculiar, debido a que nos encontrábamos en la primera pandemia mundial debido al COVID-19.

Si bien la pandemia ya ha quedado en el pasado y se está volviendo a la normalidad. Hay que destacar que la pandemia generó cambios los cuales se prevén que continúen en el futuro. Como puede ser el preocuparse por el estado del aire en ambientes cerrados o evitar aglomeraciones, lo que motivó el desarrollo de este proyecto.

1.2. Objetivos

1.2.1. Objetivos Generales

Se marcan los siguientes objetivos generales para este proyecto :

- Definir y validar un diseño para el desarrollo del **Campus Inteligente** en la Facultad de Ingeniería, utilizando equipo de IoT.
- Definir las bases de cuáles serán las tecnologías a utilizar en este **Campus Inteligente**, haciendo especial énfasis en la plataforma de IoT.
- Presentar un prototipo de la solución para el **Campus Inteligente** que valide las decisiones tomadas.

- Evaluar la solución presentada mediante el prototipo.

1.2.2. Objetivos Específicos

Con el fin de completar los objetivos definidos anteriormente, se establecieron los siguientes objetivos específicos:

- Realizar un relevamiento del estado del IoT de posibles arquitecturas a utilizar para un desarrollo de IoT.
- Realizar un estudio del estado del arte de las plataformas open-source de IoT
- Implementar un prototipo de **Campus Inteligente** de punta a punta.
- Realizar una análisis del prototipo, donde se tendrán en cuenta aspectos como la escalabilidad y la seguridad.
- Realizar pruebas de rendimiento sobre el prototipo.

2. Proyectos relacionados

Como se mencionó en la sección **1.1 Motivaciones**, este no es el primer esfuerzo realizado por el grupo MINA de la Facultad de Ingeniería en el área de IoT. A continuación se presentan los tres proyectos anteriores y que este proyecto utiliza.

2.1. Proyecto: 'Relevamiento de arquitecturas para desarrollo de aplicaciones de Internet de las Cosas'

Este proyecto es el más antiguo de los tres que se tomaron como referencia para este proyecto. El mismo fue realizado en el 2018 por Alexis Arriola y Gabriela Wynants. Su proyecto se centró en un relevamiento de distintas plataformas de IoT [3]. Este proyecto fue fundamental para la realización de la presente tesis ya que dio las directrices de qué puntos observar a la hora de comparar las plataformas en la sección **3.3 Plataforma de IoT**.

Cabe mencionar que la tesis además del relevamiento de plataformas, realizó un

prototipo con dos de estas para compararlas y definir su rendimiento y experiencia de uso.

2.2. Proyecto: 'Aplicación de IoT con diversas tecnologías inalámbricas'

Esta es una tesis del 2019, la cual fue realizada por Federico Detta, la misma se trata del desarrollo de punta a punta de una aplicación sencilla de IoT. Con la que busca generar conocimiento en el uso de distintas tecnologías inalámbricas para la comunicación entre dispositivos de IoT así como en la recolección, almacenamiento y acceso a los datos. Se propone como caso de uso implementar una aplicación que, mediante el uso de sensores, permita contar la cantidad de individuos con dispositivos inalámbricos que se encuentran en un área determinada y disponibilizar estos datos [13].

De este proyecto se utiliza el caso especificado casi en su totalidad. Realizando modificaciones en como se disponibiliza los datos para que se integre de mejor manera con el **Campus Inteligente**.

2.3. Proyecto: 'Localización en interiores utilizando infraestructura de Internet de las Cosas'

Este proyecto es un sucesor directo del anterior, el cual toma lo desarrollado por Federico Detta y lo transforma en algo nuevo. Lo que Francisco Crizul y Gerardo Gomez realizaron en 2021 para este proyecto fue un sistema de localización en interiores [12]. Dicho sistema utiliza de base la aplicación de contar dispositivos de la sección anterior.

Lo que hace el sistema es medir las señales emitidas por las interfaces Wi-Fi y Bluetooth de los dispositivos que se encuentren al alcance y, utilizando la tecnología LoRa, se envían a un servidor para su procesamiento. El sistema posee dos mecanismos de localización existentes: *Fingerprinting* y *Trilateracion* a partir de la RSS. El RSS es un acrónimo de *Received Signal Strength* que significa "Fuerza de la señal recibida". Por lo general se utiliza al *Received Signal Strength Indicator*

para medir al RSS.

Además, este proyecto también cuenta con una aplicación para la visualización de los elementos localizados. Finalmente, se mide el desempeño del sistema y se evalúa la solución en un escenario de uso real.

Este proyecto fue utilizado para comprender mejor lo que implica un desarrollo de IoT pero su caso de uso no fue tenido en cuenta para esta primera versión del **Campus Inteligente**.

3. Relevamiento de Arquitecturas y Tecnologías

Con el fin de obtener el contexto del IoT en la actualidad, se realizó un relevamiento. En esta sección se presentan los resultados del mismo. Se abordarán las propuestas de arquitecturas y los protocolos de comunicación existentes para desarrollos de IoT. Así como las ofertas de plataformas para el IoT en la actualidad.

Es importante notar que un desarrollo de IoT consiste en distribuir sensores y actuadores en un área de interés formando una red. Esta red obtendrá información del estado de dicha área en tiempo real a través de los sensores. La información recolectada es enviada a través de internet a una plataforma de IoT que se encargará de procesar, guardar y disponibilizar la información. Esto no es una tarea trivial, ya que hay que proteger la red de fallos y ciberataques. Otra de las responsabilidades de la plataforma es activar los actuadores cuando es necesario, generando cambios en el área de interés.

Se destaca que la distribución de dispositivos para abarcar el área en cuestión no es algo trivial, ya que se busca cubrir el área al menor costo posible, mientras se conectan los dispositivos entre sí. Esto genera un problema de optimización que ha sido estudiado en proyectos como CSOCA: Chicken Swarm Optimization Based Clustering Algorithm for Wireless Sensor Networks [32] u Optimization approaches to sensor placement problems [36].

En ciertos casos también hay que prestar mucha atención al consumo de energía,

ya que el área donde se distribuyen los dispositivos puede ser extensa, bastante inhóspita y de difícil acceso, dando como resultado dificultades a la hora de realizar tareas como el cambio de batería de los dispositivos. Algunos artículos que profundizan en este tema son Low Power IoT Network Sensors Optimization for Smart Cities Applications [18] o Network optimizations in the Internet of Things: A review [41]

Para los intereses de este proyecto, el consumo de energía y la distribución de los sensores no forman parte de los principales desafíos a enfrentar, ya que los sensores se distribuirán en los salones y se da por hecho el acceso a corriente eléctrica y a Wi-Fi en todos los salones de la facultad.

Por otro lado, los desafíos que sí se deben enfrentar para este desarrollo son: la selección de la plataforma de IoT, definir qué protocolos de comunicación deben usarse y la definición de la arquitectura del sistema.

La selección de la plataforma de IoT no es algo trivial ya que la misma puede limitar las características que puede abarcar el **Campus Inteligente**. Por su parte, la arquitectura y los protocolos a utilizar inciden directamente sobre el rendimiento del sistema.

Otro desafío que se debe tener en cuenta es la seguridad del sistema ya que las aplicaciones de IoT manejan datos sensibles, los cuales deben ser protegidos y manipulados con responsabilidad.

3.1. Arquitecturas

A medida que se desarrollan aplicaciones de IoT, se han ido desarrollando distintas arquitecturas para cubrir los puntos débiles de las anteriores. A continuación se repasan las arquitecturas más usadas.

3.1.1. 3 capas

Esta arquitectura surgió en los inicios del IoT, compuesta por las siguientes capas:

- **Capa de percepción:** también conocida como capa de sensores, está conformada por los sensores y su responsabilidad es recolectar datos de interés, un ejemplo de esto puede ser una cámara o un termómetro. Además, esta capa puede contener actuadores, los cuales realizan una acción sobre el área de interés, un ejemplo de esto puede ser una cerradura o interruptor de luz inteligente.
- **Capa de red:** esta capa funciona como un puente entre las otras dos, también se le suele llamar capa de transmisión. Pueden utilizarse distintas tecnologías y protocolos, por ejemplo: Wi-Fi o *bluetooth*.
- **Capa de aplicación:** capa que se encarga de disponibilizar los servicios requeridos para que las aplicaciones funcionen. Los mismos pueden ser muy variados dependiendo de la necesidad de cada aplicación.

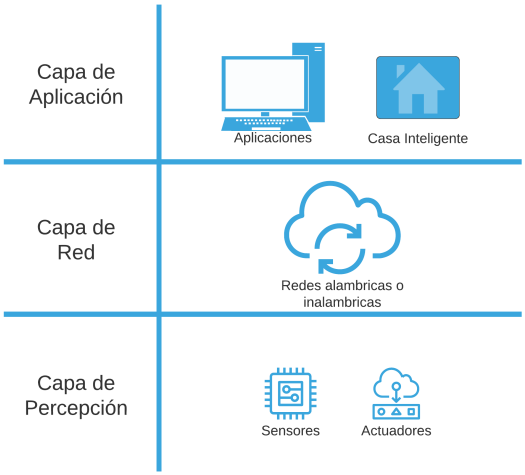

Figura 2: Esquema de la arquitectura de 3 capas.

Como se ilustra en la Figura 2, la principal ventaja de esta arquitectura es su sencillez y flexibilidad. Lo que permite aplicarla en múltiples escenarios, por ejemplo en una casa inteligente o en un sistema de cámaras de seguridad. Esto se debe a que captura la esencia del IoT.

Por otro lado, su mayor desventaja son las múltiples vulnerabilidades de seguridad que presenta. Dichas vulnerabilidades no se encuentran centralizadas en una

sola capa ya que cada una de ellas presentan problemas de seguridad [9].

La capa de percepción es vulnerable a distintos tipos de ataques como pueden ser: el *Eavesdroping*, *Replay Attack* o *Node capture*. *Eavesdroping* es un ataque en tiempo real el cual intercepta comunicaciones privadas como llamadas o mensajes telefónicos con el fin de robar información. Mientras que *Replay Attack* consta de obtener información auténtica de una comunicación entre un emisor y receptor, luego se utiliza la información del emisor para hacerse pasar por este último, logrando de esta manera autenticarse en el sistema. *Node capture* es cuando el atacante logra apoderarse de un nodo importante como puede ser un *gateway* y consigue acceso a la información que pasa por ese nodo.

Por su parte la capa de red también posee ciertas vulnerabilidades como son el *Denial of Services* o *Man in the middle*. El ataque *Denial of Services* consta de sobrecargar la capacidad de la infraestructura del sistema logrando de esa forma inhabilitarlo completamente. Por otro lado, *Man in the middle* es cuando el atacante logra interceptar y modificar una comunicación sin que los participantes de la misma se den cuenta.

Por último pero no menos importante, están las vulnerabilidades de la capa de aplicación, entre las cuales se encuentran *Cross Site Scripting* y *Malicius Code Attack*, entre otros. Ambos ataques generan cambios en el comportamiento esperado de la aplicación. *Cross Site Scripting* es la inyección de código desde el lado del cliente, por ejemplo usando JavaScript. Mientras que *Malicius Code Attack* es código presente en cualquier parte de la aplicación que genera efectos no deseados. Puede auto ejecutarse o requerir de acciones del usuario.

A su vez la sencillez que presenta a pesar de ser una de sus mayores ventajas puede ser un problema cuando se tienen escenarios más complejos como en desarrollos heterogéneos, este tipo de desarrollos usan múltiples tecnologías tanto en los sensores, como en la comunicación lo que puede llegar a sobrecargar a la capa de aplicación o a la capa de red con una nueva responsabilidad, como lo es el procesar los datos que llegan mediante distintos protocolos.

3.1.2. 4 capas

A medida que el desarrollo de aplicaciones de IoT avanzaba se deseaba mejorar la seguridad de los sistemas por lo que se propuso la arquitectura de 4 capas. Esta arquitectura consta de las 3 capas de la arquitectura anterior, pero se le agrega la capa de apoyo o de soporte.

La capa de soporte se encuentra entre la capa de red y la capa de aplicación, sus responsabilidades son confirmar que la información enviada proviene de usuarios auténticos y protege a los usuarios de amenazas. A pesar de lo mencionado anteriormente, hay un par de amenazas de seguridad para esta capa. Como: *Denial of Service* o *Malicious Insider Attack* [9].

Como ya se mencionó *Denial of Service* trata de inhabilitar completamente al sistema, por lo general sobrecargándolo.

Por su parte *Malicious Insider Attack* es un ataque difícil de prevenir ya que un usuario autorizado utiliza la información a la que tiene acceso para dañar al sistema.

En la Figura 3 se puede ver un esquema de esta arquitectura.

Cabe mencionar que otra responsabilidad de esta capa es el procesar los datos para que la capa de aplicación los pueda utilizar de forma sencilla. Algunos trabajos sugieren utilizar *Cloud Computing* para implementar esta capa [35].

3.1.3. 5 capas

Las arquitecturas de 4 capas juegan un rol importante en el desarrollo del IoT. Pero a pesar de eso dicha arquitectura posee ciertas vulnerabilidades como se mencionó en la sección anterior. Por todo esto surgió la arquitectura que explicaremos en esta sección.

Esta arquitectura tomó las tres capas que poseía la arquitectura de 3 capas y

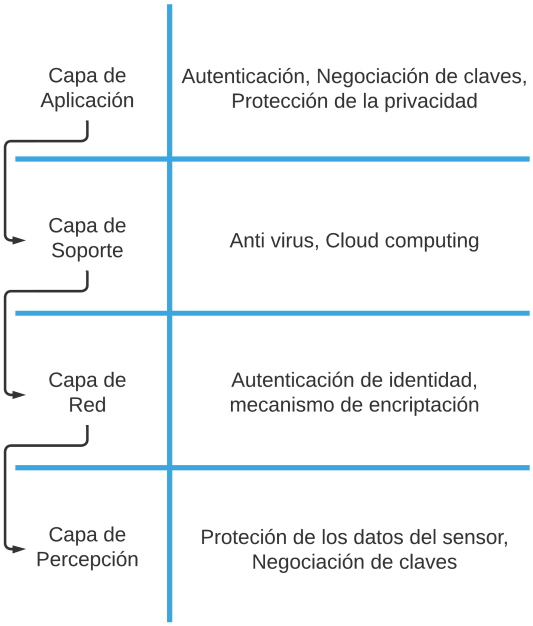

Figura 3: Esquema de la arquitectura de 4 capas.

le agrego dos más **Capa de procesamiento** y **Capa de negocios**, como se puede ver en la Figura 4

La **Capa de procesamiento** también llamada **Capa de middleware**, se encuentra entre la capa de Transporte y la de Aplicación. La responsabilidad de esta capa es recolectar los datos enviados desde la capa de transporte y procesarla, eliminando información que no sea útil y almacenando el resto.

Esto reduce la vulnerabilidad de las demás capas pero esta capa puede ser atacada mediante ataques de extenuación o *malware*.

Por su parte la **Capa de negocios** se encuentra luego de la capa de Aplicación, trabaja como administrador de todo el sistema. Lo que significa que se encarga de controlar las aplicaciones, ganancias y la privacidad de los usuarios. Además también se encarga de los flujos de datos.

Esta capa no es de las más vulnerables pero puede ser susceptible a ataques de

lógica de negocios o a ataques de día cero.

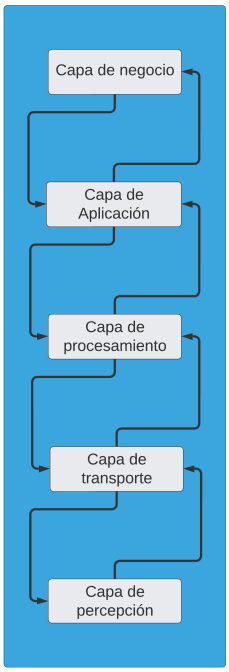

Figura 4: Esquema de la arquitectura de 5 capas.

3.1.4. 6 capas

A pesar de los esfuerzos en mejorar la arquitectura de las aplicaciones de IoT, las mismas siguen siendo vulnerables a ciertos ataques que pueden poner al sistema en riesgo. Debido a esto surgió otra arquitectura en capas para el IoT, esta vez de 6 capas. La mayor diferencia entre esta arquitectura y las demás radica en que no fue diseñada a partir de la arquitectura de 3 capas. Aunque sí se pueden apreciar algunas capas similares entre ambas arquitecturas.

Las capas de esta arquitectura son:

- **Capa de percepción:** también llamada *Capa Física* o *Capa de Sensores*. Como sus nombres indican esta capa está compuesta por los diferentes sensores de la aplicación de IoT. Los cuales se encargan de obtener información relevante para el sistema.

- **Capa de observación:** la mayor responsabilidad de esta capa es monitorizar / observar la información procedente de la *Capa de Percepción*, mientras se asegura que su origen sea legítimo y que se encuentre libre de amenazas.
- **Capa de procesamiento:** como se puede deducir por el nombre de esta capa, su principal trabajo es procesar los datos que recibe de la Capa de Observación, eliminando los datos innecesarios y almacenando el resto. Esta capa confía que los datos provenientes de la *Capa de Observación* están sin corromper.
- **Capa de seguridad:** esta capa está encargada de proteger al sistema de posibles ataques que vengan de la *Capa de Red*. También tiene la responsabilidad de asegurar los mensajes, por ejemplo cifrándolos, antes de enviarlos a la siguiente capa.
- **Capa de red:** recibe la información desde la *Capa de Seguridad*, la cual se encuentra cifrada para evitar vulnerabilidades y se encarga de enviarla a la siguiente capa ya sea mediante conexiones cableadas o redes inalámbricas. El medio utilizado dependerá de las necesidades de cada caso. A esta capa también se le conoce como *Capa de transición*
- **Capa de aplicación:** es la última capa de esta arquitectura, que se encarga de dar formato y presentar los datos. Además debe ofrecer múltiples aplicaciones a los usuarios, las cuales deben ofrecer un servicio en base a la información recolectada.

Como ya se mencionó esta arquitectura difiere de las demás ya que hace especial foco en la seguridad, de ahí que existan dos capas dedicadas a hacer más robusto el sistema, mientras que en las otras Arquitecturas siempre se hizo un foco funcional tratando de mejorar la seguridad a medida que se iteraba sobre las distintas arquitecturas. Un esquema de esta arquitectura se puede ver en la Figura 5

Si bien esto puede parecer un detalle menor, ya que las arquitecturas de 5 y 6 capas se parecen, el hecho de tener la seguridad del sistema presente desde el inicio hace que la arquitectura de 6 capas tenga una ventaja sobre las demás. Debido a esto se utilizara esta arquitectura como base para la solución del **Campus Inteligente**. Se profundizará en la arquitectura de la solución **4.2 Arquitectura**

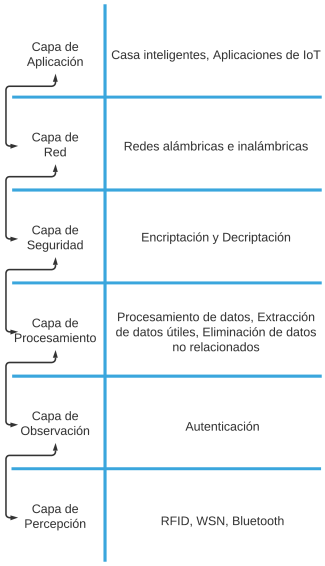

Figura 5: Esquema de la arquitectura de 6 capas.

3.2. Protocolos de comunicación

En esta sección se discutirán protocolos de comunicación que se utilizan en desarrollos de IoT, junto con breves descripciones de los puntos débiles de los mismos respecto a la seguridad. Cabe mencionar que puntos importantes a tener en cuenta son: el alcance, la capacidad y el consumo energético.

3.2.1. Bluetooth

El *bluetooth* es un protocolo de comunicación popular y muy utilizado en la actualidad. Este se puede encontrar en celulares, autos, auriculares, etc. El *bluetooth* permite realizar comunicaciones inalámbricas de corto alcance. Este protocolo está definido en el estándar IEEE 802.15.1. Este estándar ha sido mejorado a lo largo de los años, lo que ha provocado que mejore su alcance, capacidad y consumo de energía.

Además, hay que destacar que existen dos estándares dentro del *bluetooth*, el *Basic Rate (BR)* y el *Low Energy (LE)*. El BR es el estándar básico para el *bluetooth*, este tiene un alcance promedio de 10 metros, con un ancho de banda que puede alcanzar los 2.1 Mb/s. Por su parte el LE alcanza un ancho de banda de 2 Mb/s,

junto con un alcance similar al de BR. [8]. A pesar de que sus capacidades son muy similares, los mismos tienen diferencias en sus arquitecturas y el consumo de energía. Mientras que en el BR ambos dispositivos poseen los mismos permisos y el remitente de un mensaje tiene que recibir el permiso del receptor para enviar dicho mensaje. El LE utiliza una arquitectura maestro-esclavo, donde el maestro define cuando el esclavo debe enviarle los mensajes. Permitiendo así que el esclavo esté en reposo cuando no está enviando información lo que produce un menor consumo de energía [39]. Cabe destacar que existen dispositivos que logran aumentar el alcance de este protocolo aumentando la potencia de la señal, alcanzando unos 100 metros de alcance.

A pesar de ofrecer métodos de seguridad como lo son: el cifrado y el *handshake*. El *bluetooth* enfrenta algunos riesgos de seguridad como *bluejacking* y *bluesnarfing*. *Bluejacking* consta de enviar mensajes a un usuario, sin que el mismo se de cuenta, suele ser inofensivo ya que no afecta al usuario pero no deja de ser una vulnerabilidad. Por otro lado, *bluesnarfing* consta de robar información de un dispositivo lo que hace que sea más peligroso y dañino que el anterior.

3.2.2. ZigBee

ZigBee es un protocolo de comunicación el cual suele ser utilizado en redes PAN (Personal Area Network), la cual provee una red inalámbrica de corto alcance y bajo consumo de energía. La arquitectura de este protocolo es de cuatro capas: aplicación, red, *media access control* y física.

Este protocolo está basado en IEEE 802.15.4, alcanza un rango de unos 10 a 100 metros y tiene un ancho de banda de 250Kb/s [6]. Este protocolo puede ser utilizado en una gran variedad de escenarios, como pueden ser: casas inteligentes, cuidado de la salud, servicios de telecomunicaciones o automatización de edificios.

Otro punto importante es que la capa *Media Access Control* (MAC) es la encargada de la seguridad del protocolo, ofreciendo tres funciones de seguridad: *control de acceso*, *cifrado* y *control de integridad*.

Estas funciones de seguridad son proporcionadas a partir de una clave única para cada dispositivo de la red. Hay que destacar que la mayor debilidad de este protocolo se encuentra en que no posee métodos para cambiar dichas claves una vez asignadas.

3.2.3. LoRa

LoRa, que es una abreviatura de *Long Range*, es un protocolo de comunicación inalámbrica por radio frecuencia, que trabaja en 3 rangos de frecuencias ISM (*Industrial Scientific Medical*): 915MHz para América, 868MHz para Europa y 433MHz para Asia. Son bandas menores a las frecuencias de GHz en las que no se necesitan licencias para emitir. Como desventaja es que se tendrá interferencia, lo que podría generar que el mensaje no sea recibido [13].

Este protocolo es utilizado para la transmisión de datos en redes LPWAN (*Low Power Wide Area Networks*) entre dispositivos distribuidos. Existen dos tipos de dispositivos, los nodos o sensores que son los encargados de generar y transmitir los mensajes y los *gateways* quienes se encargan de recibir el mensaje y reenviarlo a la infraestructura.

Sus tasas de transferencia van de 0.3 kb/s a 50 kb/s y posee un rango de unos 2–5 kms [39]. La baja tasa de transferencia se debe a que está diseñado para tener un rango de algunos kilómetros en dispositivos a batería, a la vez que la batería tiene una vida útil de meses o años.

Este protocolo de comunicación define: la arquitectura de la red, la estructura de los paquetes de datos, como los paquetes son procesados en el servidor, como debe armarse el paquete, y como debe cifrarse.

Un dato no menor, es que en el 2015 se creó una asociación abierta sin fines de lucro llamada LoRa Alliance, con el objetivo de estandarizar redes Low Power Wide Area Network (LPWAN). Estas ofrecen una larga duración de batería, pudiendo enviar inalámbrico pequeñas cantidades de datos a largas distancias. Esto es importante ya que fomenta la estandarización de este tipo de redes bajo el pro-

tocolo LoRaWAN permitiendo que el IoT tenga y ofrezca un futuro sostenible [13].

LoRaWAN es un protocolo de red abierto que usa el protocolo propietario LoRa para la capa física. Fue diseñado para conectar inalámbricamente a internet dispositivos operados a batería.

3.2.4. Wi-Fi

El estándar IEEE 802.11 del *Institute of Electrical and Electronics Engineers*, conocido como Wi-Fi, opera en las bandas de radio Industriales, Científicas y Médicas (ISM), no reguladas y de uso libre. En las primeras versiones de este estándar, su alcance era de aproximadamente 100 m, lo que aumentó a cerca de 1 km en IEEE 802.11ah, diseñado principalmente para servicios de IoT [12]. Cabe destacar que el Wi-Fi tiene un ancho de banda entre 11 Mb/s y 300 Mb/s.

En la Figura 6 se puede ver la arquitectura de este protocolo. En la cual se puede notar que tanto los puntos de acceso (Access points AP) y los dispositivos están agrupados en Basic Service Set (BSS). Los cuales a su vez están conectados a un router, esto permite que los dispositivos de distintos BSS se comuniquen entre sí a la vez que tienen acceso a internet.

En cuanto a seguridad este protocolo tiene ciertas debilidades, como el hecho de que es susceptible a *eavesdropping* o *sniffing*, donde los mensajes pueden ser interceptados, modificados o borrados por completo. A pesar de esto, el Wi-Fi sigue evolucionando, lo que llevó al desarrollo de mecanismos de cifrado que mejoraran la seguridad del Wi-Fi. Actualmente, el Wi-Fi utiliza WPA que proviene de *Wi-Fi Protected Access* [5], es un protocolo de cifrado y actualmente ya existe WPA 2 y 3. Donde cada nueva versión es más robusta que la anterior.

3.2.5. Redes celulares

La conectividad de los celulares en la actualidad ofrece un gran campo de acción para el IoT, ya que los mismos cuentan con múltiples tipos de sensores y una buena capacidad de cómputo. Varios protocolos de comunicación han sido desarrollados

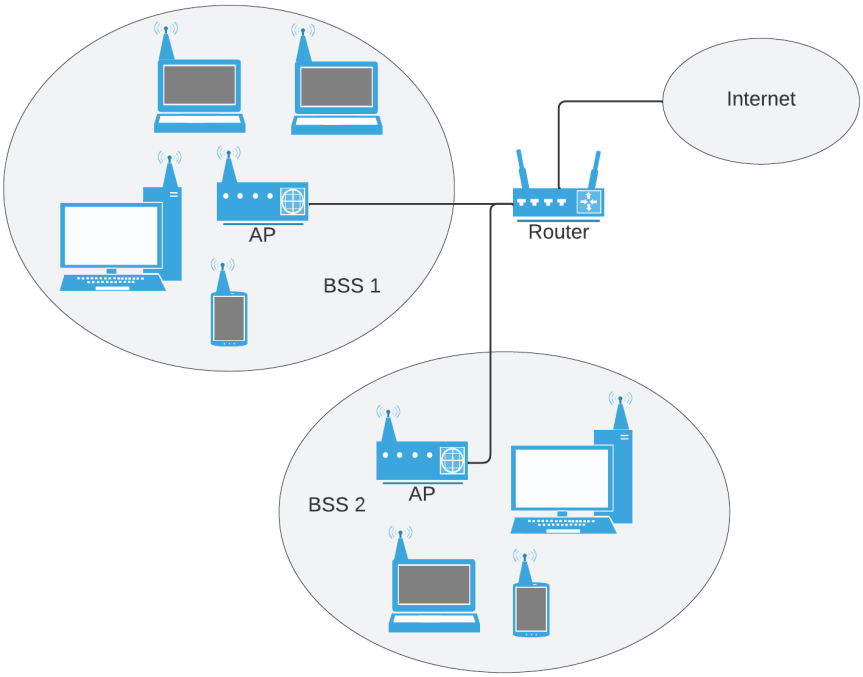

Figura 6: Arquitectura 802.11.

en este aspecto como 3G, 4G, LTE, LTE-M, NB-IoT y 5G. Para los intereses de este proyecto las últimas tres son las de mayor relevancia.

LTE-M y NB-IoT son protocolos para redes *low power wide area network* (LP-WAN por sus siglas) que utilizan bandas de radio para su comunicación. Fueron desarrollados por 3GPP (3rd Generation Partnership Project) en el 2016. LTE-M es una abreviatura de LTE-MTE que significa *Long Term Evolution-Machine Type Communication*. Este protocolo está diseñado para permitir conectarse a internet a un gran número de dispositivos y celulares, permitiendo comunicaciones M2M *Machine to Machine*. Por su parte NB-IoT es la abreviatura de *Narrowband Internet of Things* [1] [42]. Las mayores diferencias entre este protocolo y LTE-M son: el ancho de banda, mientras que el de NB-IoT es de 200kHz, el de LTE-M es significativamente mayor siendo de 1.4GHz; a su vez LTE-M posee velocidad máxima de transferencia de 1 Mb/s, que es cuatro veces mayor que la de NB-IoT a unos 250 kb/s. Sobre el consumo de energía o alcance de señal ambos protocolos son muy similares [21].

El protocolo 5G es el más actual de los mencionados, se estima estará disponible a su total capacidad para el 2025, a su vez se sabe que logra cumplir con gran parte de los desafíos de red que presenta el IoT como: altas velocidades de datos, alta escalabilidad, baja latencia, seguridad, larga vida de batería, entre otros. Sin embargo este protocolo abre la puerta a un nuevo conjunto de desafíos como privacidad o heterogeneidad, por mencionar algunos [9].

3.3. Plataforma de IoT

Otra de las tareas realizadas en este proyecto fue llevar a cabo un relevamiento de las plataformas de IoT disponibles en el mercado. Ya que una parte importante de un desarrollo de IoT, es la plataforma seleccionada para el mismo. Las mismas son plataformas cuya principal función consta de recibir los mensajes de los sensores y guardarlos en la base de datos, mientras que a su vez disponibiliza los *end points* para consumir los datos almacenados.

La plataforma debe constar con una serie de características funcionales para cumplir sus obligaciones, y dichas características son: seguridad, escalabilidad y flexibilidad. A continuación se describe qué se entiende por cada una de estas características en el contexto de una plataforma de IoT.

- **Seguridad:** qué tan seguras son las comunicaciones con la plataforma.
- **Escalabilidad:** qué tan bien maneja la plataforma el crecimiento horizontal de una solución de IoT (aumento en el número de sensores).
- **Flexibilidad:** qué tantos protocolos de comunicación acepta la plataforma y qué tan fácil es integrar diferentes tipos de sensores a la plataforma.

Además de las características funcionales recién mencionadas, hay características no funcionales que deben ser tomadas en cuenta. Las mismas son:

- **Comunidad:** En este punto se tiene en cuenta que tan activa y amplia es la comunidad que utiliza la plataforma.
- **Madurez:** Esta característica representa que tan afianzada está la plataforma y cómo ha evolucionado con el tiempo.

- **Documentación:** En este ítem se tiene en cuenta que tan completa, clara y bien ordenada es la documentación.
- **Usabilidad:** Para este punto se tiene en cuenta que tan intuitiva y sencilla de usar es la plataforma.

Estas características funcionales y no funcionales fueron definidas siguiendo lineamientos del proyecto de referencia *Relevamiento de arquitecturas para desarrollo de aplicaciones de Internet de las Cosas"* [3].

3.3.1. Importancia de las Plataformas de IoT

Los desarrollos de IoT suelen ser complejos, debido a que se utilizan distintos protocolos de comunicación, dispositivos y en algunos múltiples arquitecturas. Debido a esto se han desarrollado Plataformas de IoT que facilitan la integración de los distintos componentes.

En pocas palabras se puede decir que una Plataforma de IoT consiste de un *software* cuya principal función, como ya se mencionó, es conectar los diferentes dispositivos, y procesar, analizar y almacenar los datos que los mismos generan. En la Figura 7 se ve un esquema que muestra el lugar que suele ocupar la plataforma en un desarrollo de IoT [3].

Hay que destacar que las capacidades que una plataforma de IoT suele incluir para cumplir con su rol de facilitar el desarrollo de aplicaciones son:

- Facilidades para la personalización y/o creación de aplicaciones.
- Capacidad de procesar eventos (por ejemplo: flujo de eventos, agregación de datos, motores de reglas, etc).
- Capacidad de análisis y visualización de datos.
- Herramientas para ciberseguridad, como autenticación o cifrado.
- Capacidad de comunicación con múltiples dispositivos y protocolos de IoT.

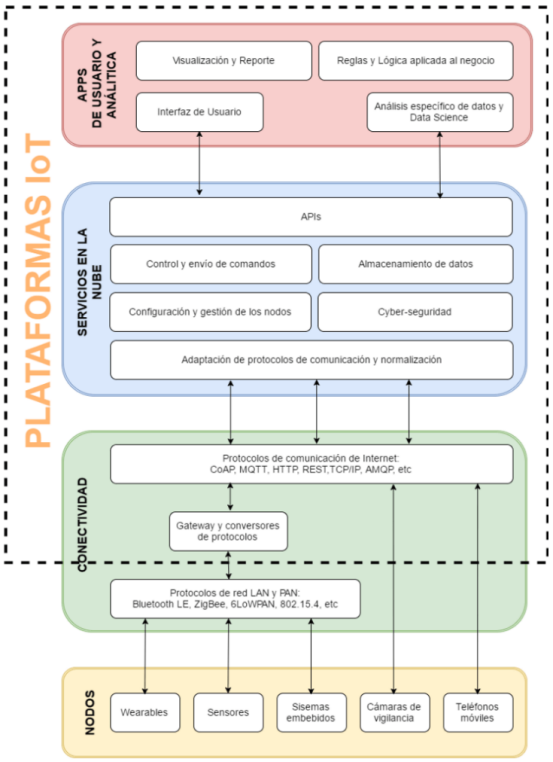

Figura 7: Ubicación de las plataformas IoT en la arquitectura de una solución IoT

3.3.2. Comparación

Para este documento se tuvieron en cuenta sólo plataformas de IoT con código abierto ya que se busca que el código quede como propiedad de la Facultad de Ingeniería. Esto provocó que se descartaran plataformas pagas como: Google Cloud's IoT platform, AWS IoT platform, Arduino Cloud IoT o Kaa IoT platform.

A pesar de ser una buena plataforma de IoT, se descartó la plataforma Fiware [19] debido a que al ser una plataforma muy grande y compleja no se adapta a nuestro escenario. Esta decisión fue tomada en base a lo comentado en la tesis Relevamiento de arquitecturas para desarrollo de aplicaciones de Internet de las Cosas [3].

Para seleccionar cuales serían las plataformas a tener en cuenta se utilizaron 4 fuentes de información que realizan un análisis de las plataformas [29] [37] [48]

[20], de las cuales obtuvimos tres plataformas candidatas para utilizarlas en el desarrollo del **Campus Inteligente**, las cuales son: Mainflux [26], OpenRemote [30] y ThingsBoard [45].

Habiendo definido las plataformas más relevantes para el **Campus inteligente**, se continuó con la realización de una comparativa entre las distintas plataformas. Los puntos que se utilizaron para comparar las tres plataformas fueron los definidos anteriormente. Funcionales: seguridad, escalabilidad y flexibilidad. No funcionales: comunidad, madurez y documentación. Como se puede ver en la Figura 8.

	Funcionales			No Funcionales		
	Seguridad	Escalabilidad	Flexibilidad	Documentación	Comunidad	Madurez
Mainflux	Identificacon con tokens tanto para usaurios como para dispositivos	microervices, edge computing and cloud computing	HTTP, MQTT, WebSocket, CoAP	Posee una documentacion organizada y facil de entender	Posee preguntas en Stackoverflow pero no con demasiadas respuestas, ademas no encontre un chat de comunidad ni nada parecido, parece estar sentrado al soporte pago que ofrecen	La plataforma es mencionada por 2 de los 4 links tenidos en cuenta y su git posee 1700, y actividad reciente.
OpenRemote	- Ofrece distintos tipos de usuarios a la hora de utilizar la plataforma - No se encontro informacion sobre como asegurar las comunicaciones con los sensores	Solución Edge Gateway para conectar varias instancias con una instancia de administración central.	HTTP, KNX, LoRa, MQTT, Simulator, SNMP, Serial, TCP, UDP, Velbus, Websocket, Z-Wave	Gran catidad de documentacion a pesar de que no es facil de navegar o de ecntrar las cosas	Posee preguntas en Stackoverflow pero no con demasiadas respuestas, ademas no encontre un chat de comunidad ni nada parecido, parece estar sentrado al soporte pago que ofrecen	3 de los 4 links referencian a esta plataforma. Su git cuenta con 514 estrellas y cuenta con actividad reciente
ThingsBoard	Ofrece distintos tipos de usuarios. Identificacion de dispositivos: - Access Tokens - Basic MQTT credentials - X 509 certificates - Comunicaciones HTTPS y MQTT con TLS/SSL	Tiene dos arquitecturas posibles, una monolitica para pequeños usuarios y otra de microservicios para escenarios que requieran alta disponibilidad y una gran escalabilidad horizontal	HTTP, HTTPS, MQTT,CoAP	Documentacion clara, completa, facil de navegar y muy bien explicada	Posee un chat de comunidad activo a la fecha 01/03/2022, ademas de tener un hilo dedicado en stackoverflow con publicaciones recientes 01/03/2022	Se referencia a la plataforma en todos los links que se tuvieron en cuenta. Su repositorio de git posee actividad reciente, asi como 11000 estrellas.

Figura 8: Comparativas de plataformas de IoT.

Respecto a los aspectos funcionales de las plataformas, se destaca que ThingsBoard cuenta con mayores funcionalidades de ciberseguridad, ofreciendo administración de usuarios, utilización de tokens y distintas formas de asegurar las comunicaciones. Esto no quiere decir que las demás no ofrezcan seguridad. Por ejemplo Mainflux cuenta con identificación mediante tokens y OpenRemote con administración de usuarios.

En cuanto a la escalabilidad todas las plataformas elegidas cuentan con opciones para desarrollos a gran escala, por lo que no es un punto en el cual ninguna plataforma sobresalga.

Como último punto en cuanto a la funcionalidad de la plataforma tenemos la flexibilidad, en este punto se destaca OpenRemote, el cual cuenta con el mayor número de protocolos de comunicación soportados. También hay que destacar que ThingsBoard y Mainflux soportan un conjunto de protocolos muy similar.

En cuanto a los puntos no funcionales, ThingsBoard destaca por sobre las demás plataformas en todos los aspectos. Esto no quiere decir que Mainflux u OpenRemote posean malas características no funcionales, sino que ThingsBoard tiene mejores características. En cuanto a que OpenRemote y Mainflux tengan una comunidad menos desarrollada, se teoriza que se debe a que el enfoque de negocio de ambas plataformas está centrado en el soporte. Mientras que ThingsBoard posee un modelo de negocio *freemium*, con una versión gratuita y otra paga.

Luego de realizada la comparativa, se decidió no tener en cuenta a Mainflux para la comparativa práctica, ya que es la plataforma que apareció la menor cantidad de veces en las referencias tenidas en cuenta. A su vez su comunidad no es demasiado activa. Cabe mencionar que su flexibilidad no es mala pero tampoco era la mejor dentro de las plataformas comparadas. En los demás puntos ofrece buenas características pero no hace una diferencia significativa con el resto de las plataformas.

Como ya se mencionó, OpenRemote posee una gran flexibilidad, debido a la gran cantidad de protocolos de comunicación que soporta. Esto es algo muy beneficioso a la hora de conectar varios tipos de sensores con la plataforma. Por otro lado, la mayor desventaja de esta plataforma es que su documentación no es sencilla de navegar y eso no permitió encontrar demasiada información sobre algunos temas, como por ejemplo, como aseguran las comunicaciones con sensores. A pesar de esto se optó por instalar la plataforma para realizar una prueba práctica. Ya que se valora mucho el gran número de protocolos que acepta.

Por último pero no menos importante tenemos a ThingsBoard, la cual ofrece la comunidad más grande y activa; la mayor aceptación en GitHub, la documentación más completa y sencilla de navegar; junto con buenas herramientas para implementar seguridad. Su punto más débil es la flexibilidad ya que no soporta

tantos protocolos de comunicación como OpenRemote. Por eso se decidió hacer una prueba práctica para comparar ambas plataformas.

3.3.3. Pruebas Realizadas

Con el fin de seleccionar la plataforma que mejor se acomodara a los requerimientos del **Campus inteligente** se realizaron pruebas prácticas sobre las dos plataformas. Las pruebas constaron de dos partes:

1. Realizar la instalación de cada plataforma.
2. Realizar la conexión de un sensor a la misma utilizando el protocolo de comunicación HTTP.

Esto con el fin de comprobar su funcionamiento y la usabilidad de las plataformas.

El ambiente que se utilizó para las pruebas fue una máquina virtual creada con VirtualBox [31], con 3 procesadores, 3GB de RAM y con el sistema operativo Ubuntu 18 [47].

- **ThingsBoard:** Se logró instalar la plataforma de manera rápida y sin inconvenientes siguiendo las guías que ofrece la documentación de la plataforma [43] [44]. Cabe mencionar que se instaló la plataforma dentro de un contenedor de Docker [15]. La conexión de un sensor es sencilla y solo consta de crear una entidad 'dispositivo' en la plataforma, la cual tendrá un *access token* asignado. Luego solo resta enviar los datos al endpoint `THINGSBOARD_HOST_NAME/api/v1/ACCESS_TOKEN/telemetry` y con el header `Content-Type:application/json`. Para que la plataforma los procese y los almacene en la base de datos.
- **OpenRemote:** Se logró instalar la plataforma de manera satisfactoria, aunque el hecho de que la documentación no fuera tan clara como la de ThingsBoard, hizo que la instalación no fuera tan fluida como en el caso anterior. Se instaló la plataforma dentro de un contenedor de Docker. La conexión con el dispositivo no es tan sencilla ya que a diferencia de ThingsBoard donde acepta json sin importar su fromato siempre que este sea un json válido, OpenRemote precisa definir el tipo de entrada provocando que sea necesario realizar más configuraciones para su correcto funcionamiento.

3.3.4. Plataforma seleccionada

Basados en la comparativa realizada y en las pruebas prácticas, se define como la plataforma a utilizar en el prototipo del **Campus Inteligente** a **ThingsBoard**. También se recomienda su uso si algún día se lleva a cabo la implementación del **Campus Inteligente** a escala real.

Como ya se mencionó, el hecho de no soportar tantos protocolos de comunicación como lo hace OpenRemote era su mayor desventaja, pero la plataforma ofrece la posibilidad de desarrollar tu propio conector. El cual funciona como un puente entre la plataforma y el protocolo deseado a utilizar. A su vez se destaca que ThingsBoard fue mucho más intuitiva y sencilla de utilizar que OpenRemote durante la prueba práctica. Además, es la plataforma que ofrece mejores características no funcionales y en cuanto a las funcionales no tiene nada que envidiar a las otras.

ThingsBoard posee dos arquitecturas, una monolítica y otra de micro servicios. En la Figura 9 se puede ver la arquitectura de la plataforma en su versión monolítica. Mientras que en la Figura 10 se ve la arquitectura de micro servicios.

Según la documentación de la plataforma y el artículo *Performance Evaluation of Open Source IoT Platforms* [24], la arquitectura monolítica logra soportar de manera aceptable hasta 1000 dispositivos enviando datos en simultáneo. Esto nos indica que si el **Campus Inteligente** se encuentra por debajo de este número, se recomienda utilizar dicha arquitectura. Esta recomendación se basa en que es más sencilla de configurar, a la vez que cumple con los requerimientos del sistema. Por otro lado, si se espera extender al **Campus Inteligente**, será necesario el uso de la arquitectura de micro servicios ya que la arquitectura monolítica no escalaría de buena manera.

Por último se destaca que esta plataforma en cualquiera de sus arquitecturas ofrece un motor de reglas (*Rule Engine*). Es un sistema altamente configurable y personalizable, para el manejo y procesamiento de eventos. Este motor de reglas permite

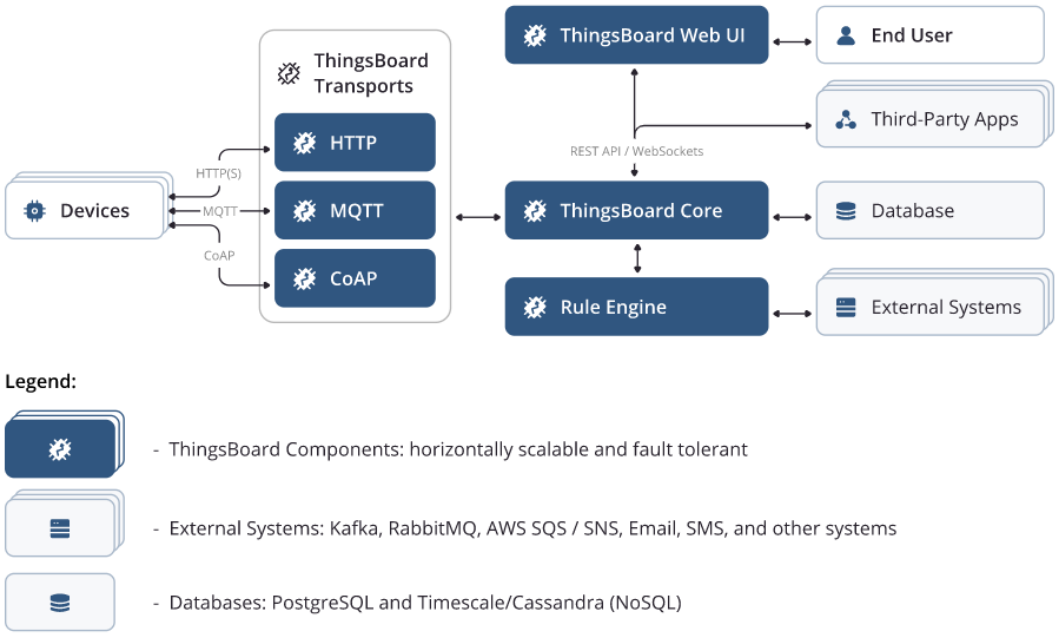

Figura 9: Diagrama de arquitectura monolítica de ThingsBoard.

filtrar, enriquecer y transformar los mensajes que llegan a la plataforma desde dispositivos de IoT. Además permite disparar acciones como puede ser notificaciones [45].

4. Campus inteligente

En esta sección se presentarán los casos de uso que se definieron para el **Campus Inteligente**. A su vez se describirá la arquitectura propuesta y la plataforma seleccionada para un posible despliegue del mismo. Se llama **Campus Inteligente** al desarrollo de IoT que se está proponiendo, el cual abarca desde los sensores a utilizar hasta las aplicaciones que consumen los datos obtenidos por los sensores.

4.1. Casos de uso

Se definieron cuatro casos de usos los cuales funcionan como núcleo del **Campus Inteligente**. Estos casos de uso definieron los requisitos que se deben cubrir en el diseño del **Campus Inteligente**. Cabe mencionar que de estos cuatro casos de

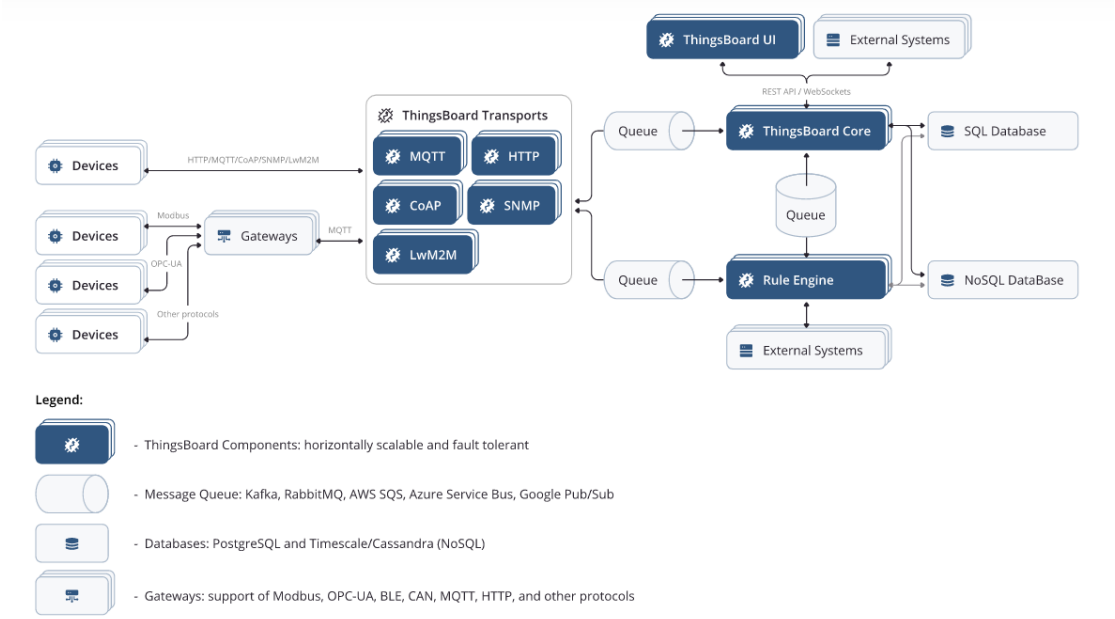

Figura 10: Diagrama de arquitectura de micro servicios de ThingsBoard.

uso se implementaron dos en el prototipo que será discutido en próximas secciones. Los casos de uso tenidos en cuenta son:

- **Contador de Personas:** en este caso de uso se busca estimar la cantidad de personas presentes en un salón.
- **Calidad del Aire:** como su nombre lo indica lo que se busca con este caso de uso es saber la calidad / estado del aire en un salón determinado.
- **Luces o computadoras encendidas en habitaciones vacías:** este caso de uso utilizaría al anterior: Contador de Personas, junto con sensores que informan del estado de los dispositivos en el salón, permitiendo que los mismos sean apagados de forma remota y automática cuando el salón está vacío.
- **Lugares en Estacionamiento:** dado que la facultad cuenta con más de un estacionamiento para los autos se consideró de valor poder saber en cuales de ellos hay lugares libres.

4.1.1. Contador de Personas

Este caso de uso fue uno de los implementados en el prototipo y fue el primero que se tuvo en cuenta debido a que es un caso de uso que fue trabajado con anterioridad. Como ya se comentó, esto tuvo lugar en la tesis *"Localización en interiores utilizando infraestructura de Internet de las Cosas"* [13].

Para este caso, se decidió que la cantidad de personas en un salón sería estimada mediante el conteo de dispositivos conectados a Wi-Fi, donde se asume que cada persona posee un solo dispositivo. El conteo se realiza mediante la recolección de MAC. Las direcciones MAC son identificadores para todos los dispositivos que utilizan la interfaz de red 802.11, las mismas son únicas globalmente y asignadas por el IEEE. Esta dirección consta de 6 bytes presentes en los mensajes de cada dispositivo. La implementación de este caso de uso en el prototipo será discutido en la sección **5.2 Contador de personas**

Este caso de uso al utilizar Wi-Fi para contar a las personas dentro del salón requiere de otra tecnología para los sensores, ya que de lo contrario también se detectarían los sensores y complicarían el conteo de dispositivos. A su vez se recomienda ser cuidadosos en el manejo de datos ya que se utilizan las direcciones MAC para identificar a los dispositivos. Las tecnologías que se recomiendan utilizar para este caso de uso es Wi-Fi y LoRa, para contar los dispositivos y enviar los datos a la plataforma respectivamente.

4.1.2. Calidad del aire

Con motivo de la pandemia este caso de uso fue propuesto para llevar control de la calidad del aire en espacios cerrados. Este caso de uso es importante ya que sabiendo la calidad del aire se puede actuar para evitar ambientes propensos a propagar enfermedades, como las variantes del COVID. A su vez también permitiría activar sistemas de ventilación o similares. Este fue el otro caso de uso implementado en el prototipo. Su implementación se discutirá en la sección **5.3 Calidad del aire**

Este caso de uso puede llegar a generar alguna problemática con el caso anterior, ya que en caso que se usen dispositivos que utilicen Wi-Fi podrían aparecer

en el conteo del caso anterior. Hay un proyecto que está trabajando en este caso de uso y utiliza Wi-Fi. Por lo tanto, una posible solución a esto sería ocultar de la red a los dispositivos que implementen este caso de uso, evitando así aparecer en el conteo del caso anterior. En caso de que esto no sea posible se debería ajustar el caso anterior para contemplar a estos dispositivos.

4.1.3. Luces o computadoras encendidas en salones vacíos

Este es uno de los casos de uso que quedaron fuera del prototipo. La idea del mismo es actuar sobre los salones apagando dispositivos electrónicos conectados a internet, logrando así evitar desperdiciar electricidad en salones vacíos.

Para una implementación de este sistema se requeriría del caso de uso Contador de Personas para lograr identificar cuándo un salón está vacío. Luego se requiere de dispositivos conectados a internet que puedan recibir una orden de forma remota. Una vez se tuvieran los requisitos anteriores con la correcta configuración de la plataforma de IoT se lograría implementar el caso de uso. Lo más interesante que aporta este caso de uso al **Campus Inteligente** es que haría uso de una funcionalidad de las plataformas de IoT que es activar actuadores de forma remota.

4.1.4. Lugares en estacionamiento

Otro de los casos que se discutió fue el de lugares en el estacionamiento. No se indagó demasiado en el mismo ya que se priorizaron al Contador de dispositivos y a la Calidad del Aire. Cabe mencionar que existen implementaciones que muestran los lugares libres para un estacionamiento, como se hace en los centros comerciales, aunque los mismos no envían esa información a una plataforma. Este sería el desafío a enfrentar en este caso. También se destaca que este caso de uso ayudaría a reducir las emisiones de gases de invernadero y el tráfico. Ya que el tiempo de búsqueda de un lugar para estacionar sería reducido [49].

A pesar de no haber profundizado demasiado en este caso de uso por lo explicado en el párrafo anterior, si se dedicó algo de tiempo para estudiar que implicaría el desarrollo de este caso de uso [28]. Primero hay que destacar que se requerirían

sensores capaces de detectar a un auto estacionado, por ejemplo sensores de ultrasonido. Los cuales junto a un microprocesador, enviarían mediante algún método de comunicación inalámbrica, como podría ser LoRa, la distancia a la que detectan un objeto. Se menciona LoRa, ya que la cantidad de información a enviar es reducida y posee un gran alcance y bajo costo. Los mensajes serían recibidos por la plataforma de IoT que se encargaría de procesarlos y almacenarlos. Por último una aplicación consumiría los datos de la plataforma de IoT e informaría al usuario de los lugares disponibles para estacionar.

4.2. Arquitectura

Con los casos de uso ya discutidos pasaremos a discutir la arquitectura propuesta para el **Campus Inteligente**. La arquitectura que se propone es una de seis capas basada en la arquitectura discutida en la sección **2.1.4 6 capas**. Esto ya que la misma ofrece buenas herramientas a la hora de implementar la seguridad ya que dos de las seis capas están diseñadas para eso, la capa de Observación y la de Seguridad. Además a pesar de ser la arquitectura con el mayor número de capas, no es tan compleja comparada con las demás. Por lo que se cree que es la mejor opción. En la Figura 11 podemos ver la propuesta de arquitectura junto con sus respectivas capas.

4.2.1. Capa de percepción

La capa de percepción es la primera capa y está compuesta por los distintos sensores del **Campus Inteligente**, los cuales podrán tener distintos protocolos de comunicación. Se espera que sin importar el protocolo de comunicación de cada sensor, exista un *gateway* que se logre comunicar mediante algún protocolo IP. Esto para que sea posible la comunicación con la siguiente capa, la capa de observación.

En el diagrama de la Figura 11 se representó esta realidad mediante sensores que son capaces de enviar la información directamente a un *router* de internet, como con sensores que no son capaces de tal cosa, como es el caso de los sensores

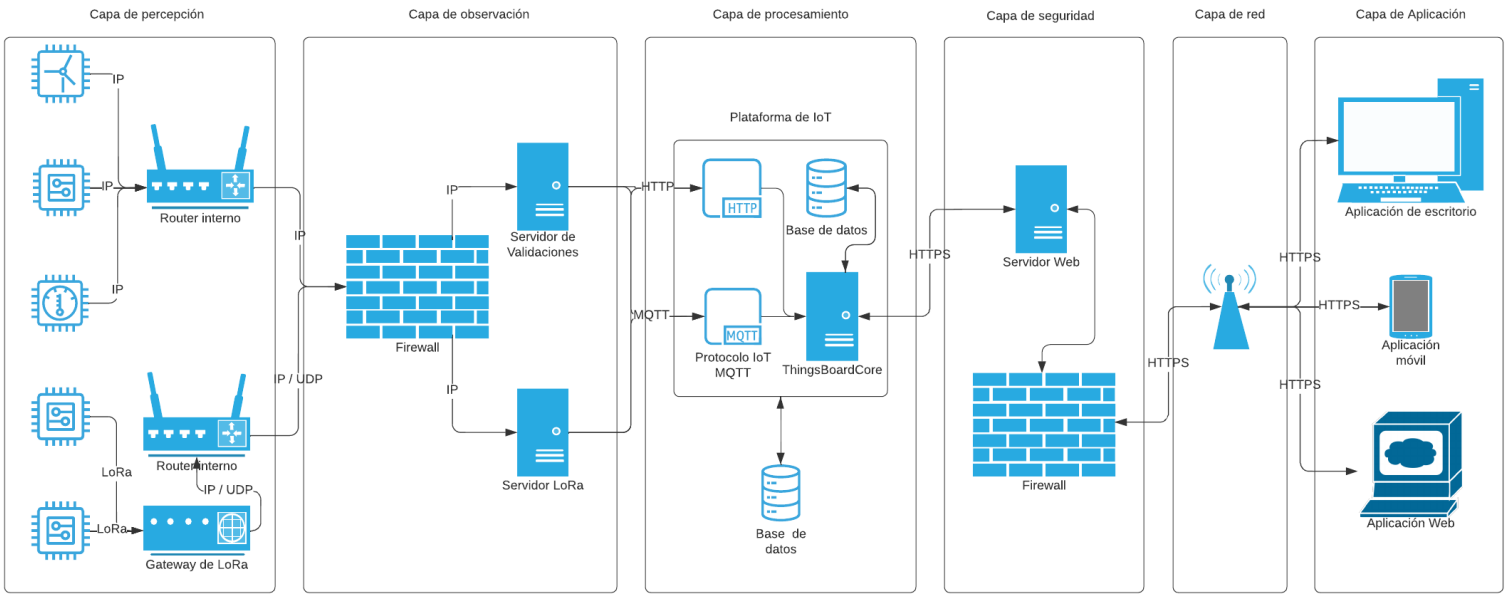

Figura 11: Arquitectura del Campus Inteligente.

LoRa que requieren de su propio *gateway* para lograr enviar su información a la siguiente capa.

4.2.2. Capa de observación

Esta es la segunda capa y su objetivo es validar que la información sea legítima y que provenga de sensores válidos. Para cumplir con esta tarea se propone utilizar servidores que realicen la autenticación de los sensores. Un ejemplo de esto es el caso de los sensores LoRa, los cuales tienen dos tipos de autenticación con el servidor LoRa correspondiente. Las cuales son activación por personalización (ABP - Activation by Personalization), o activación por aire (OTAA - Over-the-Air Activation). La primera consta de configurar las claves manualmente para el sensor y el servidor. Por el otro lado la segunda opción las claves se calculan dinámicamente.

Esto se representó como dos servidores separados los cuales realizan la autenticación de distintos tipos de sensores, uno para el caso LoRa y otro para los demás sensores. Esta generalización se debe a que esos casos de uso no estaban tan desarrollados como el otro.

A su vez esta capa cuenta con un *firewall* que se encarga de filtrar los mensajes maliciosos que puedan ser enviados desde direcciones o puertos desconocidos. A su vez se podría incluir un filtrado por contenido para evitar mensajes que no tuvieran el formato deseado. Luego, para los mensajes que cumplan con la forma aceptada por los servidores, son enviados a los mismos y estos se encargan de la autenticación y de pasar los datos a la siguiente capa.

Se destaca que la plataforma de IoT ofrece varios métodos de autenticación, como por ejemplo mediante tokens de acceso, pero algunos protocolos como por ejemplo LoRa requiere de una autenticación previa realizada en un servidor LoRa. Este último tipo de autenticación es el que se llevará a cabo en esta capa.

4.2.3. Capa de Procesamiento

Como su nombre lo indica esta capa se encarga del procesamiento de los datos que le llegan desde la capa de observación. Además, esta capa está encargada de disponibilizar los datos obtenidos para ser consumidos por las aplicaciones.

Como se ve en la Figura 11, la Capa de Procesamiento está formada por la plataforma de IoT, las bases de datos para almacenar la información y por los *end points* para recibir y disponibilizar dicha información.

4.2.4. Capa de Seguridad

La responsabilidad de esta capa es proteger al sistema de amenazas que provengan de la capa 5: Capa de Red. Para lograr este objetivo se propone la utilización de un servidor *proxy* y un *firewall*.

Los servidores que funcionan como *proxy* deben utilizar cache, de esta manera se evitaría recargar a la capa de procesamiento. A la vez que se controlan las conexiones con dicha capa al tener la capa de seguridad como intermediario.

Por su parte el *firewall* se encargará de filtrar las amenazas provenientes de la capa de red y de esta forma se mejora aún más la seguridad y el control sobre las

conexiones que llegan a la capa de procesamiento.

4.2.5. Capa de Red

Esta capa está encargada de comunicar la capa de seguridad y la de aplicación. Permitiendo que la información esté disponible para los usuarios.

Esta capa no requiere de una implementación o despliegue ya que se utilizaría el internet para conectar las distintas capas. Particularmente se utilizaría el protocolo HTTPS.

Cabe mencionar que puede que se utilice internet para conectar las capas de Percepción con la de Observación, esto puede ocurrir en el caso que hayan sensores que no sean parte de la red interna del **Campus Inteligente**.

4.2.6. Capa de Aplicación

Esta capa está conformada por las distintas aplicaciones que se implementen para utilizar la información que ofrece el **Campus Inteligente**. Dichas aplicaciones pueden funcionar en distintas tecnologías como: páginas web, aplicaciones de escritorio para computadoras o aplicaciones para celulares.

5. Prototipo

La realización de un prototipo de punta a punta es uno de los principales objetivos de esta tesis, ya que el mismo será utilizado para validar las decisiones de diseño tomadas. Además ofrece un estimativo del esfuerzo requerido a la hora de desplegar el **Campus Inteligente**. Cabe destacar que debido a poseer un tiempo finito para la realización de este proyecto y a las prioridades definidas, los esfuerzos en el desarrollo de medidas de seguridad no fueron completos. Todo lo que no se logró cubrir de forma completa será abordado en la sección **9. Trabajo futuro**.

En la Figura 12 podemos ver la arquitectura seguida en el prototipo. Como se aprecia en la imagen la arquitectura seguida para el prototipo es una simplificación de la arquitectura de la sección **4.2. Arquitectura**. La mayor diferencia es

la falta de *firewalls* y que en el caso de uso Calidad del Aire no se usa servidor en la capa de observación.

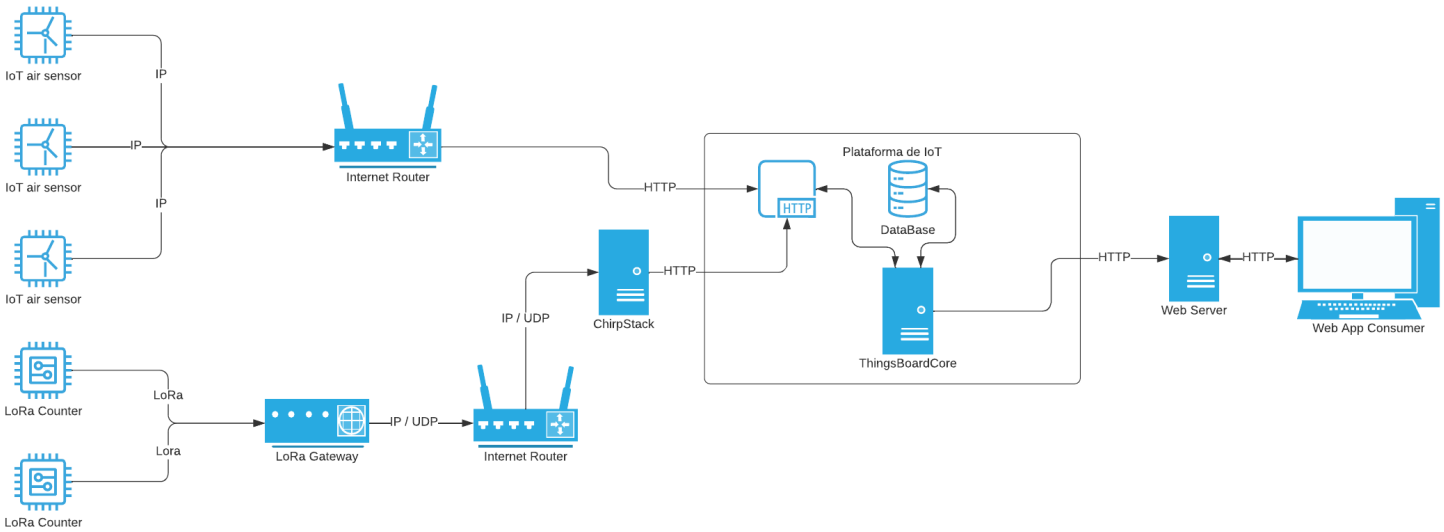

Figura 12: Arquitectura del prototipo.

5.1. Tecnologías y Hardware

Las tecnologías utilizadas en el Prototipo fueron: LoRa, Wi-Fi, HTTP, la plataforma de IoT ThingsBoard, el servidor de LoRa ChirpStack [11], React Web [27] para el desarrollo de dos aplicaciones web y un servidor *Proxy* utilizando http-proxy-middleware [10].

LoRa y el servidor ChirpStack se utilizaron para el caso de uso Contador de personas. El Wi-Fi y el protocolo HTTP se utilizó para comunicar a los distintos actores que forman parte del prototipo. Además se desarrollaron dos aplicaciones React / Javascript. La primera que se desarrolló fue un simulador de sensores del estado del

aire, debido a que el caso de uso Calidad del Aire se estaba desarrollando mientras se realizaba este proyecto. Por otro lado, la otra aplicación es una página web que consume datos de la plataforma de IoT, la cual sería la base para la aplicación web en un desarrollo real.

En cuanto al Hardware utilizado para el prototipo fue:

- Una computadora Dell, con un Intel core i5 de octava generación, con Windows 10 de 64 bits, 8GB de ram y con una terjeta de video NVIDIA GeForce GTX 1050 Ti.
- Dos placas Sparkfun ESP32 LoRa 1-Channel Gateway [40].
- Un *gateway* Dragino [16].

5.2. Contador de personas

Como se mencionó con anterioridad, el conteo de personas será estimado mediante la cantidad de dispositivos conectados a Wi-Fi, que se encuentren en el salón o área delimitada, donde se estima que cada persona posee uno y solo un dispositivo conectado a Wi-Fi.

Para realizar el conteo de dispositivos se utiliza una placa Sparkfun ESP32 LoRa 1-Channel Gateway [40], la cual recolecta las MAC de los dispositivos conectados a Wi-Fi mediante el código desarrollado en la tesis [13]. Las MAC recolectadas son enviadas al *gateway* Dragino [16]. El cual se configuró para reenviar los mensajes al servidor LoRa ChirpStack [11]. El servidor auténtica a los dispositivos mediante ABP, pero tanto el servidor como las placas tienen la capacidad de utilizar OTAA.

La comunicación entre la placa y el *gateway* es mediante LoRa. Mientras que entre el *gateway* y el servidor se utilizan mensajes IP/UDP, los cuales encapsulan al mensaje original de la placa Sparkfun ESP32 LoRa 1-Channel Gateway. Luego los mensajes son decodificados y enviados mediante el protocolo HTTP por el servidor LoRa ChirpStack hacia la plataforma de IoT.

A partir de este punto se realizaron dos implementaciones para comprobar la

flexibilidad del prototipo. La primera aproximación fue dejar cierto cómputo del caso de uso en los usuarios finales, llevando a una solución descentralizada. Luego se optó por desarrollar una solución centralizada, con el fin de proteger los datos manejados por el sistema. En las siguientes secciones se discutirán cada una de ellas con sus ventajas y sus contras.

5.2.1. Solución descentralizada

Como se comentó, esta fue la primera aproximación a la hora de completar el Contador de personas. Lo que se hizo fue que la aplicación web del **Campus Inteligente** realizara el conteo de las MAC. Para esto la aplicación solicita todas las MAC recolectadas en los últimos 5 minutos en un salón particular. Luego las cuenta, seguido de esto muestra el valor al usuario.

En la Figura 13 se puede ver que una vez recibidos los datos de la plataforma, se chequea que los mismos sean del tipo esperado con un **if**. Luego se crea una estructura **map** llamada **macsMap**, luego se itera sobre las mac recibidas y se obtiene solo la mac (ya que se recibe la mac junto la intensidad con la que se captó separadas por un '|') realizando un **get** de dicha MAC sobre **macsMap** y con un if se confirma si la mac ya fue contada con anterioridad. En caso de no haber sido contada la misma se agrega al map y se sigue con la siguiente. Finalmente se guarda el tamaño del map como resultado del conteo.

Se destaca que el uso de la estructura **map** se debe a que facilita el control

```
if (!key.includes("Air")) {
  const macsMap = new Map();

  state.data_data.forEach(item => {
    const mac = item.value.split("|")[0];

    const isCounted = macsMap.get(mac);

    if (isCounted === undefined) macsMap.set(mac, mac);
  });

  value.data = macsMap.size;
```

Figura 13: Código de conteo de dispositivos.

de si una MAC ya fue contada o no, simplificando el algoritmo de conteo.

La principal ventaja que presenta esta implementación es que el cómputo requerido para contar las MAC quedan a cargo de las aplicaciones finales, en lugar de quedar como una responsabilidad de la plataforma de IoT. Pero esto viene dado por un alto costo ya que las MAC son enviadas hacia la aplicación para su conteo lo que no representa un manejo responsable de esa información sensible. Otro punto negativo es la carga que puede representar esta solución para la capa de red, ya que se envían mayores cantidades de datos y esto puede generar una mayor latencia para el usuario.

5.2.2. Solución centralizada

Luego de ver las contras de la solución descentralizada se decidió implementar una solución que lograra un mejor desempeño en esas áreas. El camino que se siguió fue hacer que la plataforma cuente las MAC recolectadas y las guarde en la base de datos. Evitando de esta manera que las MAC salgan del ecosistema del **Campus Inteligente**.

Para el conteo de las MAC se utiliza el motor de reglas que ofrece ThingsBoard, donde se creó una regla para cada nuevo mensaje que provenga de las placas Sparkfun ESP32 LoRa 1-Channel Gateway. En la Figura 14 se puede apreciar la regla creada, la misma es disparada en el momento que llega un nuevo mensaje. Cabe mencionar que se configuró la plataforma para que solo los mensajes correspondientes a este caso de uso disparen la regla.

Cuando la plataforma recibe un mensaje que corresponde con los tratados por la regla, la plataforma lo envía a la misma, esto se aprecia en el nodo input. El siguiente nodo es uno de filtrado, el cual dependiendo del tipo del mensaje realiza diferentes acciones. Si es un pedido a una RPC al dispositivo, el pedido es enviado a otro nodo, el cual realiza el pedido. Si es un pedido a una RPC desde el dispositivo se guarda mediante un nodo log. Misma acción se sigue cuando el tipo del mensaje es diferente a los tipos definidos. Si es del tipo **post attributes**, tipo que utilizan los dispositivos para enviar atributos a ser guardados en la plataforma, los

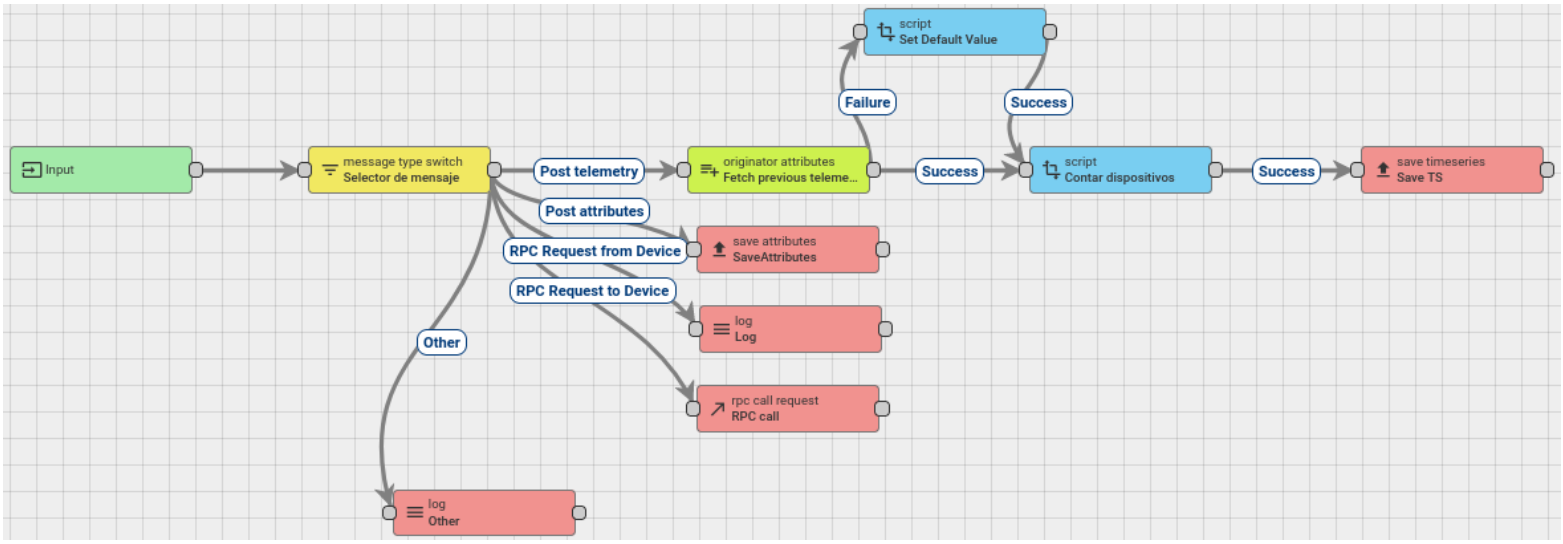

Figura 14: Regla agregada para el conteo de las MAC.

atributos son guardados en la base de datos. Todas estas acciones son iguales en la regla por defecto que ThingsBoard tiene configurada desde la instalación. Pero las modificaciones se realizaron sobre el flujo de mensajes *post telemetry*. Flujo que notifica a la plataforma que se ha realizado una nueva medición y que hay datos para ser procesados y guardados.

Para los mensajes del tipo *post telemetry*, primero se invoca a un nodo que trae la última entrada de la base de datos guardada para los datos de este caso de uso, para esto se especifican ciertos campos, en este caso son *'deviceCounter'* y *'deviceList'*, mientras que *diviceCounter* lleva el total de dispositivos contados sin repetir, *deviceList* es una lista de las MAC contadas. Cuando la última entrada en la base de datos no posee alguno de los campos buscados, el nodo falla, por eso se invoca a un nodo script el cual enriquece dicha entrada con valores por defecto para el caso de uso ($deviceCounter = 0$, $deviceList = $ ''). Luego es invocado el segundo nodo script el cual revisa la lista para asegurarse que la MAC presente en el mensaje sea nueva y en dicho caso aumenta *deviceCounter* en uno y agrega la MAC junto con su *time stamp* a la lista. En caso de que la MAC ya esté en la lista lo que hace es actualizar el *time stamp* de la misma. Pero esto no es lo único que hace este nodo, ya que mientras recorre la lista para asegurarse que la MAC sea nueva, va comprobando que las MAC guardadas en la lista no tengan más de cinco minutos

sin actualizar. En caso de que esto suceda, elimina la MAC de la lista y resta uno al contador. Por último, tenemos el nodo que guarda el mensaje enriquecido en la base de datos donde puede ser consumido tanto por la plataforma de IoT como por aplicaciones de terceros, como la aplicación desarrollada **Campus Inteligente**.

Esta solución evita que las MAC recolectadas sean enviadas a los usuarios finales, así como también disminuye la carga sobre la capa de red dado que solo envía el valor numérico de la cantidad de personas. El precio de esto es que se genera una carga mayor para la plataforma de IoT cosa que puede ser perjudicial a la hora de escalar la solución, pero teniendo en cuenta que ThingsBoard ofrece una arquitectura monolítica y una de micro servicios esto no debería ser un problema, ya que en el peor de los casos la arquitectura de micro servicios debería de ser capaz de manejarlo.

Se quiere destacar que otro factor que incentivó a implementar esta otra solución fue probar las capacidades del motor de reglas que ofrece ThingsBoard, prueba que fue más que satisfactoria ya que se pudo cumplir con las expectativas de este caso de uso.

Para finalizar hay que mencionar que esta solución permite llevar un registro del número de personas en el salón. Por lo que facilita el realizar gráficas con las personas en un salón algo que sería costoso con la otra solución.

5.3. Calidad del Aire

Este caso de uso fue desarrollado de forma paralela al prototipo, en un proyecto del taller de sistemas ciber físicos, por lo tanto para el prototipo se decidió simular el caso de uso. Se sabe que los sensores se comunican utilizando HTTP e IP sobre alguna tecnología inalámbrica. Basados en esto se simulan los sensores de aire mediante una aplicación web desarrollada con React, la cual mediante mensajes HTTP se comunica con la plataforma de IoT.

Para simular las lecturas de los sensores se obtuvo un conjunto de datos real de *'GAMS Indoor Air Quality Dataset'* [25], con los siguientes datos:

- Dióxido de carbono (CO2).
- Humedad.
- PM 10: partículas suspendidas en el aire con un diámetro de menos de 10 µm.
- PM 2.5: partículas suspendidas en el aire con un diámetro de menos de 2.5 µm.
- Temperatura en grados celsius.
- Compuestos orgánicos volátiles.

Esta aplicación simula el funcionamiento de tres sensores de aires, los cuales fueron configurados en la plataforma. Luego se utilizaron los tokens para cada sensor desde la aplicación con el fin de que la plataforma identificara a los mensajes como provenientes de los sensores.

La aplicación permite configurar el intervalo entre mensajes, así como iniciar o detener el envío de mensajes. Esto es posible utilizando el input y los dos botones, que se pueden apreciar en la Figura 15 donde el input representa el intervalo en segundos, el botón 'Iniciar' empieza el envío de mensajes y el botón 'Detener' lo termina.

Cuando se inicia el envío de mensajes, la aplicación lee una entrada del conjunto de datos y lo envía a la plataforma de IoT como si fuera un sensor. Luego espera el intervalo indicado, para hacer lo mismo pero esta vez simulando a otro sensor. El intervalo por defecto es de 5 segundos. La aplicación continúa este ciclo de enviar el mensaje, esperar y cambiar de sensor hasta que se hace clic en el botón de 'Detener'. La política que sigue para cambiar de sensores es Round-Robin.

Para enviar lo datos se utiliza la siguiente url:

```
1 'http://<IP plataforma>:<puerto plataforma>/api/v1/<token sensor>/
    telemetry'
```

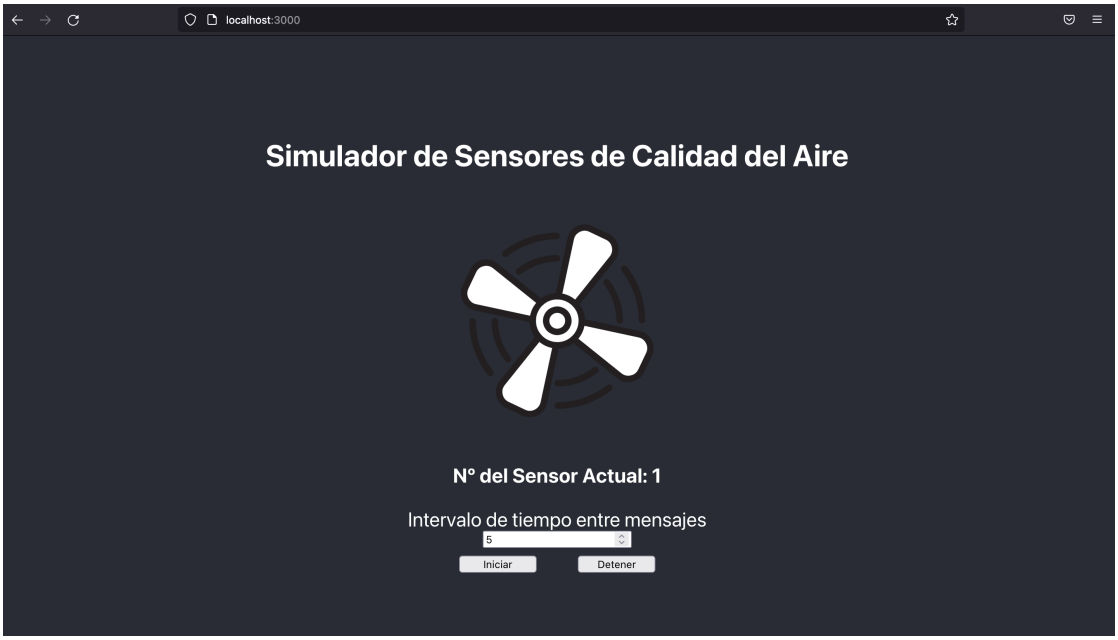

Figura 15: Simulador de sensores de calidad de aire.

Una vez que los datos son recibidos por la plataforma de IoT, la misma los guarda en la base de datos, quedando disponibles para ser utilizados por la aplicación web. Esta aplicación sera explicada en la próxima sección.

5.4. Aplicación Web

Para completar el desarrollo de punta a punta, se implementó una aplicación web utilizando React con la cual se consumió tanto la API Rest como algunos de los web sockets que brinda ThingsBoard.

La aplicación inicia realizando la autenticación con la plataforma. Para esto se usa uno de los usuarios que vienen por defecto en la misma luego de su instalación.

- **Usuario:** tenant@thingsboard.org
- **Contraseña:** tenant

La solicitud realizada fue:

```
fetch(‘/api/auth/login‘, {
      method: "POST",
      headers: {
        ’Content-Type’: ’application/json’
      },
      body: ’{"username":"tenant@thingsboard.org",
              "password":"tenant"}’
    })
```

La respuesta obtenida de esa solicitud es de la siguiente forma:

```
{token: <Token>, refreshToken: <RefreshToken>}
```

El token es guardado en el estado de la aplicación y es utilizado para las próximas solicitudes que se le hagan a la plataforma de IoT.

Una vez completada la autenticación, la aplicación pide todos los sensores presentes en el prototipo. La solicitud utilizada es de la siguiente forma:

```
fetch(‘/api/tenant/devices?pageSize=15&page=0‘, {
      method: "GET",
      headers: {
        ’Content-Type’: ’application/json’,
        ’x-authorization’: ‘Bearer ${this.token}‘

      }})
```

Una vez obtenidos los sensores los mismos son agrupados por salones. Esto es posible ya que a la hora de agregar un sensor al sistema el mismo es llamado con el tipo de sensor y con el número / nombre del salón que pertenece. Por ejemplo: un sensor de aire en el salón 307 sería ingresado al sistema bajo el nombre *Air-307*.

A continuación, la aplicación solicita las conexiones mediante web socket necesarias. Para esto se utiliza el siguiente código.

```
const webSocket = new WebSocket("ws:localhost:3001/api/ws/plugins/
    telemetry?token=" + token);

    webSocket.onopen = () => {
      const object = {
        tsSubCmds: [
          {
```

```
            entityType: "DEVICE",
            entityId: entityId,
            scope: "LATEST_TELEMETRY",
            cmdId: 10,
            keys: "temperature,co2,humidity"
          }
        ],
        historyCmds: [],
        attrSubCmds: []
      };
```

En el prototipo las conexiones mediante web sockets se realizaron para los sensores de aire con el fin de mostrar en tiempo real la temperatura, la humedad y la concentración de CO2. En la Figura 16 podemos ver la aplicación del **Campus Inteligente** con los tres salones que se configuraron para el prototipo.

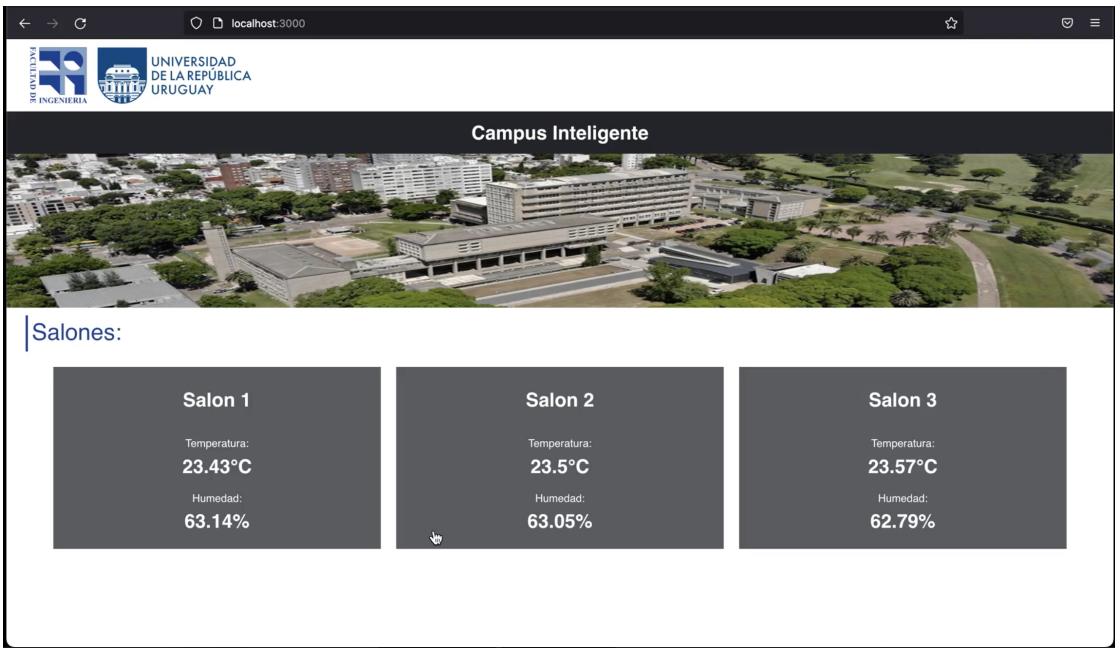

Figura 16: Aplicación web Campus Inteligente

Dentro de la aplicación, se puede hacer clic en uno de los salones para obtener más información sobre el mismo. Esto se puede apreciar en la Figura 17.
A su vez dentro de esta pantalla se puede hacer clic en la sección de la cantidad de personas en el salón, lo que desplegará el modal que vemos en la Figura 18.

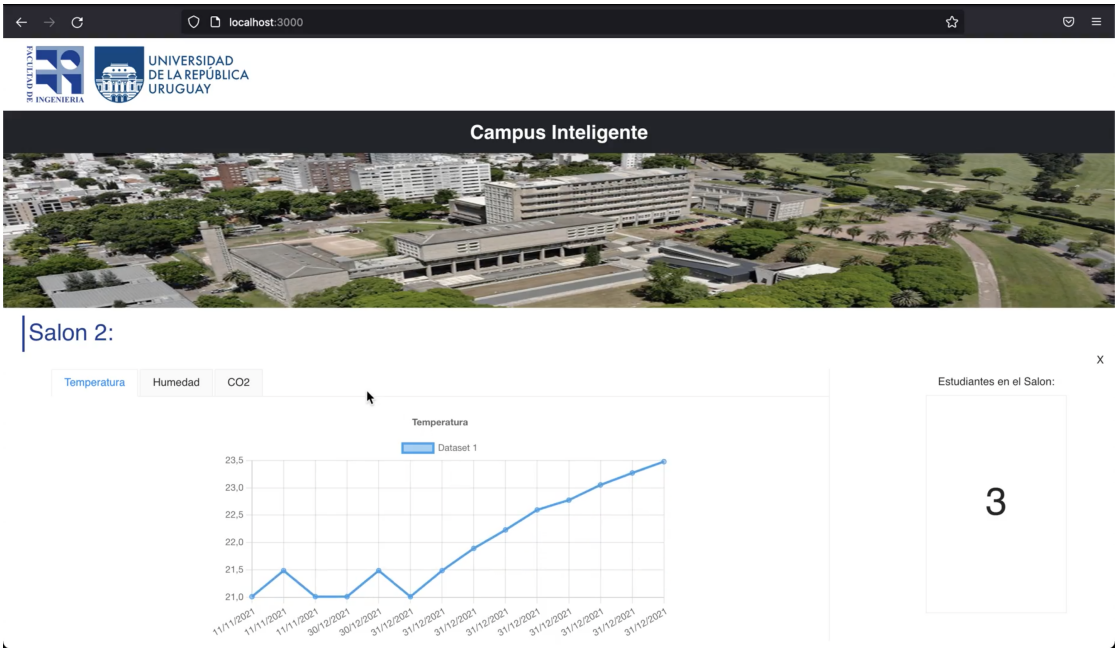

Figura 17: Aplicación web Campus Inteligente Información de salón

Para finalizar se destaca que en varios *end points* utilizados en el prototipo contaban con soporte para paginación. Aunque este no fue utilizado en la realización del prototipo, ya que se contaba con solo cinco sensores. A pesar de esto, se recomienda el uso de paginación cuando se realice la implementación real.

También se destaca que no se realizaron esfuerzos en cuanto a la aseguración de las comunicaciones, ya que se utilizó HTTP y no se usó cifrado para los contenidos o contraseñas, medidas que se recomiendan a la hora de pasar del prototipo al producto final y que la plataforma ThingsBoard soporta.

5.5. Integración con la plataforma

A la hora de integrar un dispositivo, como un sensor, se sugiere seguir la guía de la plataforma ThingsBoard [43]. En la misma se explica cómo crear dentro de la plataforma ThingsBoard una entidad Device. Hay que destacar que a esto se le llama *aprovisionamiento de dispositivos* (device provisioning) y hay varias formas

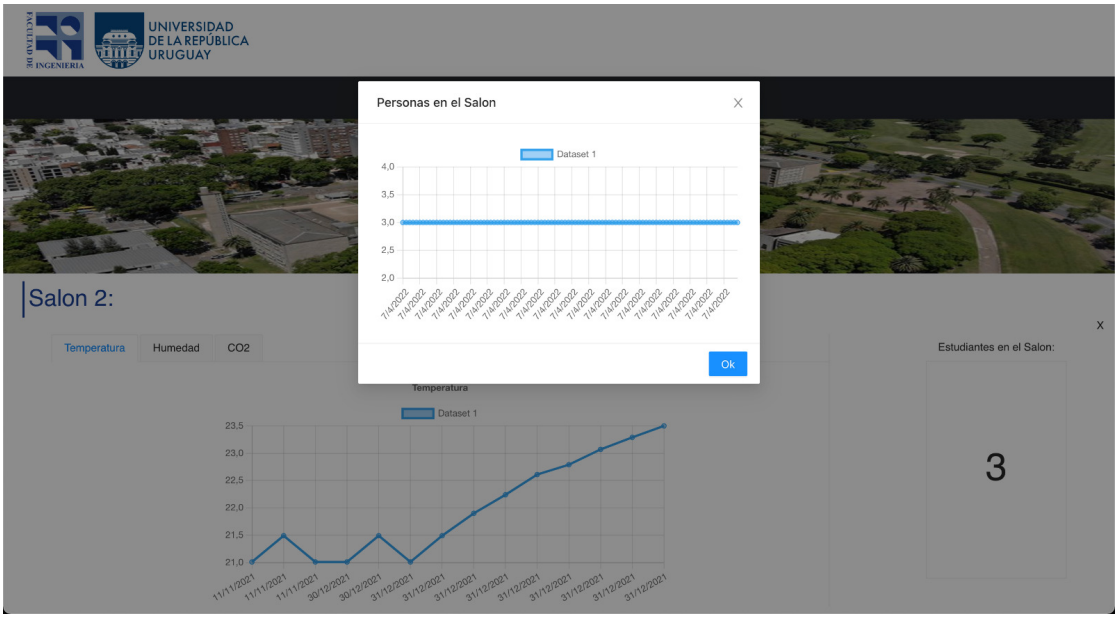

Figura 18: Aplicación web Campus Inteligente Información de personas en el salón

de realizarlo: manual, por rest API o por firmware. Para este proyecto se utilizó la forma manual. La cual luego de provisto el dispositivo se utilizan las credenciales generadas por la plataforma para configurar al dispositivo. Una vez configurado el dispositivo se puede comunicar con la plataforma con un mensaje similar al siguiente ejemplo:

```
curl -v -X POST -d "{\"temperature\": 25}" $THINGSBOARD_HOST_NAME/
    api/v1/$ACCESS_TOKEN/telemetry --header "Content-Type:
    application/json"
```

Una observación no menor es que para este proyecto todos los tipos de sensores fueron conectados mediante HTTP, pero la plataforma ofrece otros protocolos compatibles los cuales ya fueron mencionados. Otro punto no menor es que se pueden definir perfiles para los dispositivos los cuales permiten definir reglas del motor de reglas para procesar los mensajes o especificar las propiedades que se recibirán. En caso de usar el perfil *default* todas las propiedades enviadas en formato json serán guardadas en un objeto por la plataforma y quedarán disponibles para su consumo.

6. Escalabilidad y Rendimiento

En esta sección se abarcará otro de los objetivos de este proyecto que es el análisis del prototipo y su rendimiento.

Para eso se investigó el rendimiento de la plataforma de IoT utilizando papers que abordaron el tema. Además se realizaron pruebas de rendimiento tanto en el uso de CPU de la plataforma, como en la latencia de las solicitudes.

6.1. Estudio: Performance Evaluation of Open Source IoT Platforms

Cuando se escala una aplicación de IoT, el rendimiento de la plataforma que se utiliza es uno de los principales desafíos. Esto se debe a que la misma es la que se encarga de recopilar y procesar los datos de los diferentes sensores. Entonces al aumentar el tamaño de la aplicación, el número de sensores aumenta, lo que genera una mayor carga de trabajo sobre la plataforma lo que puede afectar el rendimiento de esta.

Debido a lo mencionado anteriormente, se decidió investigar el rendimiento de la solución monolítica de la plataforma de ThingsBoard. Para esto se busco algún artículo relacionado al tema, siendo seleccionado el artículo: *Performance Evaluation of Open Source IoT Platforms* [24].

En dicho artículo, se realiza una comparativa entre dos plataformas *open source* ThingsBoard y SiteWhere. Dicha comparativa constó de enviarles *requests* a las plataformas desde múltiples dispositivos virtuales. Los cuales están conectados directamente a la misma. El experimento se realizó con distintas cantidades de dispositivos conectados, siendo estas: 10, 100, 200, 500, 800, 1000. Este experimento fue realizado una vez para cada uno de los principales protocolos soportados por la plataforma, HTTP y MQTT.

La máquina en la cual se corrió la plataforma ThingsBoard con su arquitectura monolítica, fue una Lenovo IdeaPad 310, con un Intel Core I7-7500U 2.7 GHz,

8 GB de ram y con el sistema operativo Ubuntu 16.04.

Los resultados que nos interesan del artículo son los rendimientos de la plataforma ThingsBoard, los cuales fueron de 696.3 msg/sec para 1000 dispositivos conectados por HTTP y 1928 msg/sec para MQTT.

Se destaca de las conclusiones del artículo que ThingsBoard logra mejor rendimiento en el protocolo HTTP, mientras que SiteWhere logra mejor rendimiento en MQTT, pero con una mayor tasa de errores que la presentada por ThingsBoard.

A su vez el rendimiento de la plataforma ThingsBoard en HTTP con su REST API se mantiene considerablemente constante en cuanto al tiempo de respuesta promedio al variar la cantidad de dispositivos conectados. Mientras que el mejor rendimiento de mensajes por segundo lo logra al alcanzar los 800 dispositivos conectados y se mantiene cuando se conectan 1000 dispositivos.

6.2. Pruebas de Rendimiento

Esta sección explicará en qué consistieron las pruebas de rendimiento realizadas sobre el prototipo. Las mismas se dividen en dos tipos de pruebas, las pruebas de carga para ver la velocidad de respuesta del prototipo y las pruebas de uso de recursos para ver el rendimiento de la plataforma de IoT.

Cabe destacar que se evaluó el rendimiento utilizando el caso de uso Contador de personas ya que es el caso de uso implementado completamente con dos soluciones.

6.2.1. Pruebas de carga

Como ya se mencionó una de las pruebas que se realizaron fueron pruebas de carga, para las cuales se utilizó JMeter. Que es una aplicación *open source* desarrollada en su totalidad en Java, diseñada para realizar pruebas de cargas y medir rendimientos.

Las pruebas consistieron en realizar la autenticación con la aplicación y acto se-

guido se pedían los datos para el caso de uso. Para esto se implementaron dos pruebas una para cada implementación del caso de uso como se puede apreciar en la Figura 19. Además en dicha imagen podemos apreciar el usuario definido para la prueba junto con la URL utilizada. Además se aprecia la utilización de un *JSON Extractor* para obtener el token del usuario y el *HTTP Heders Manager* que se lo agrega a la siguiente solicitud. Otra cosa que se logra ver en la imagen es que ambas pruebas son muy similares, empezando las dos con la autenticación y la utilización del *JSON Extractor* y el *HTTP Heders Manager*. Pero la segunda solicitud cambia un poco de un test al otro, esto se ve reflejado en la URL de cada solicitud.

Para el caso descentralizado tenemos:

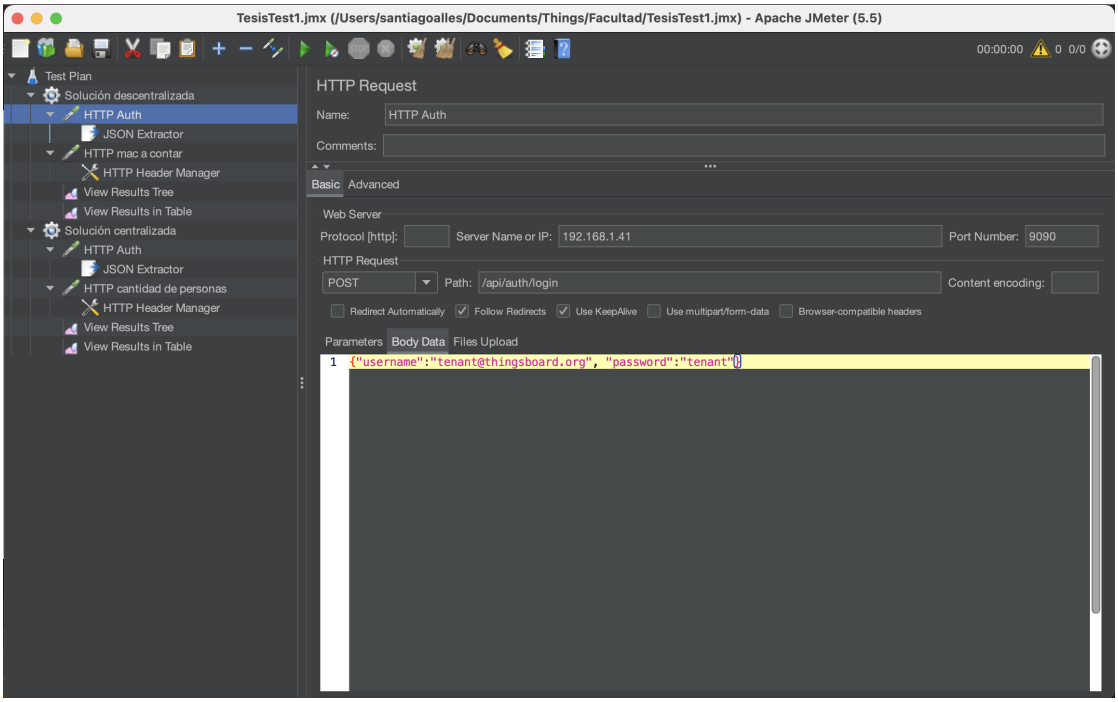

Figura 19: Pruebas desarrolladas en JMeter

```
1 /api/plugins/telemetry/DEVICE/bda7f5b0-ca6d-11ec-9286-bdaafd47cd18
    /values/timeseries?endTs=1664055422159&keys=data_data&startTs
    =1664047200000
```

Mientras que para el caso centralizado es:

```
1 /api/plugins/telemetry/DEVICE/bda7f5b0-ca6d-11ec-9286-bdaafd47cd18
    /values/timeseries?keys=deviceCounter
```

La diferencia entre ambas URL radica en que la solicitud del caso descentralizado solicita el registro de MAC con fecha anterior a *endTs* y posterior a *startTs*. Mietras que la del caso centralizado solicita solo el último valor de la propiedad *deviceCounter*. Por más detalles de la implementación utilizada en JMeter se recomienda recurrir al archivo TesisTest1.jmx en el repositorio de GitLab [4].

Se destaca que para esta prueba se utilizó otra computadora la cual corría el conjunto de pruebas en JMeter. La computadora era una MacBook Pro con un procesador Quad-Core Intel Core I5, 16 GB de RAM Y con un sistema operativo macOS Big Sur versión 11.5.2. Además hay que mencionar que la conexión entre las máquinas fue realizada mediante Wi-Fi.

Para ambas pruebas se simularon dos escenarios, todo utilizando el ambiente mencionado anteriormente. Un escenario simulaba tener 20 alumnos en el salón y el otro 100. Cada prueba se realizó cuatro veces por escenario con 100, 200, 400 y 800 solicitudes respectivamente, las cuales simulan a usuarios concurrentes del **Campus Inteligente**.

Los resultados para las pruebas en el escenario de 20 personas en el salón los podemos ver en la Figura 20; en la cual se graficó el promedio de tiempo de respuesta de las solicitudes.

Como se aprecia en la gráfica la solución centralizada posee un mejor tiempo de respuesta que su contraparte para todas las pruebas. Además el tamaño de mensaje recibido desde la plataforma es menor siendo de 474 bytes, contra los 1217 de la solución descentralizada. Esto se hace más notorio cuando comparamos los tamaños en las pruebas de 100 alumnos en el salón, donde la solución centralizada sólo aumenta en un mega bit su tamaño pasando a ser de 475, mientras que la solución descentralizada pasa a ser de 5622.

A pesar de lo mencionado anteriormente los resultados en los tiempos de respuesta no cambiaron de manera significativa, como se puede apreciar en la Figura

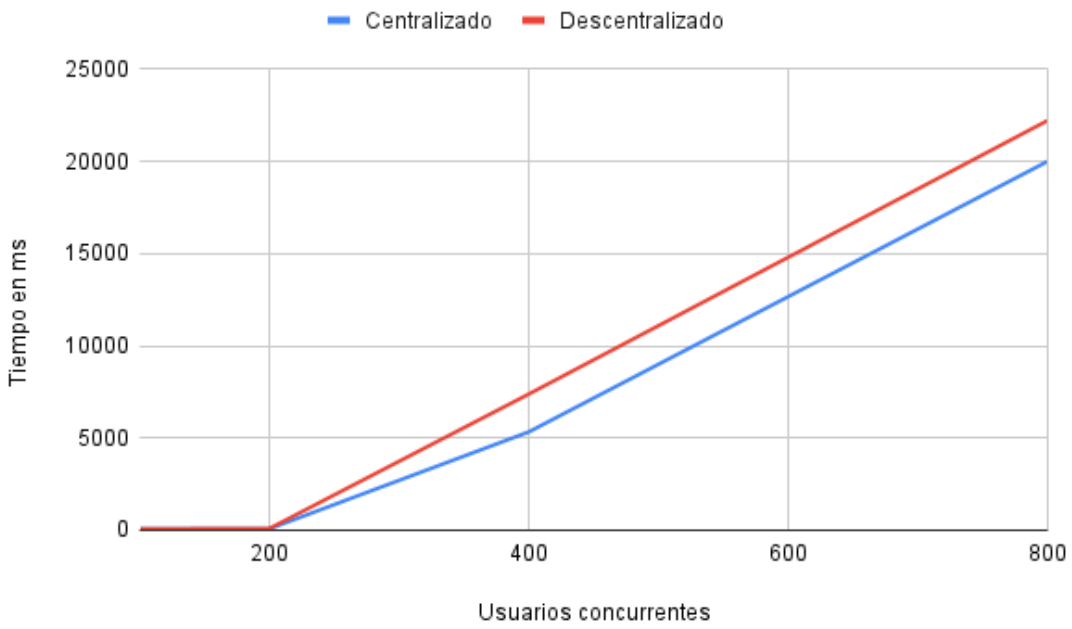

Figura 20: Promedio de tiempo de respuesta con 20 alumnos simulados en el salón

21.

Con los resultados de estas pruebas podemos concluir que la solución centralizada posee un mejor tiempo de respuesta que la descentralizada a la vez que los mensajes enviados son de menor tamaño.

6.2.2. Uso de CPU y Memoria

Para realizar esta prueba se desarrolló un *script* utilizando Python [34]. El mismo simula de 1 a 100 sensores y permite configurar cuantas rondas de mensajes enviará cada sensor, así como el número de mensajes que se enviará en cada ronda. Además de que permite definir un intervalo de espera entre cada ronda de mensajes. El script puede ser encontrado en el repositorio de GitLab [4].

El motivo por el que se desarrolló este *script* fue para obtener valores más cercanos a los que se verían en un desarrollo real ya que con los 2 sensores disponibles no

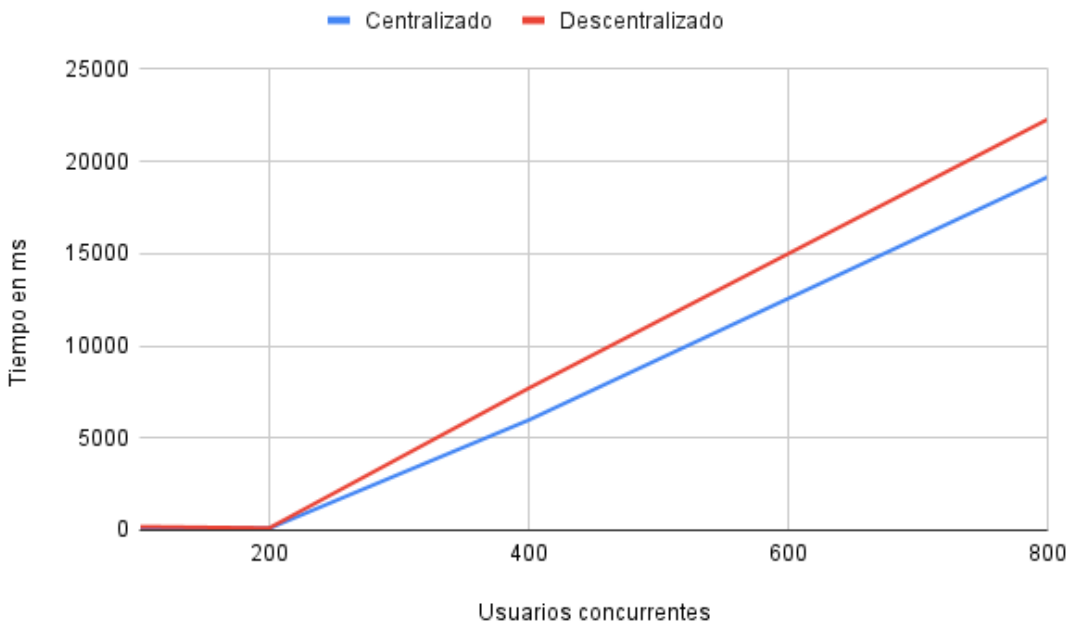

Figura 21: Promedio de tiempo de respuesta con 100 alumnos simulados en el salón

sería suficiente.

El máximo número de dispositivos que se pueden simular con el *script* es de 100 sensores, ya que cada sensor que se simula hay que configurarlo a mano en la plataforma de IoT. El simular más sensores requeriría de cierto esfuerzo y se consideró que 100 dispositivos simulados aportarían suficiente información para las pruebas.

La prueba consistió de correr el *script* simulando los 100, durante 10 rondas, enviando un mensaje por ronda y esperando un segundo entre rondas. Esta prueba se realizó sobre las dos implementaciones del caso de uso del prototipo.

Para ver los resultados de dichas pruebas se utilizó el comando de Docker:

```
1 docker stats
```

Este comando retorna un flujo de datos en vivo, de todos los contenedores activos en Docker [14]. La información que muestra está dividida en los siguientes campos:

- *CONTAINER ID*: identificador del contenedor.

- *NAME*: nombre del contenedor.
- *CPU %*: porcentaje de CPU del anfitrión utilizado por el contenedor.
- *MEM %*: porcentaje de memoria del anfitrión utilizado por el contenedor.
- *MEM USAGE / LIMIT*: total de memoria utilizado por el contenedor y la cantidad total que está habilitado a utilizar.
- *NET I/O*: cantidad de datos que el contenedor ha enviado y recibido de la interfaz de red
- *BLOCK I/O*: cantidad de datos que el contenedor ha escrito y leído de dispositivos de almacenamiento del anfitrión.
- *PIDs*: número de procesos o hilos que el contenedor ha creado.

En la Figura 22 se pueden apreciar los resultados de la prueba al correrla sobre la solución descentralizada, donde se ve un gran consumo de la CPU, lo cual no se considera un problema ya que la prueba fue realizada en la máquina virtual utilizada para el prototipo y los recursos de la misma eran escasos. Se destaca que el uso de memoria no fue excesivo y que los procesos iniciados fueron 287. El resto de los datos no ofrecen información relevante, ya que las pruebas fueron realizadas una detrás de otra sobre la misma máquina solo cambiando la configuración de la plataforma de IoT.

Por otro lado, en la Figura 23 se aprecian los resultados de la prueba centralizada de los cuales se destacan la cantidad de procesos iniciados siendo 349 y que el consumo de CPU fue aún mayor.

Estos resultados demuestran que la plataforma requiere de más recursos a la hora de procesar las MAC para el caso de uso Contador de personas.

Una vez comparadas las soluciones se decidió ver cómo evolucionaba el rendimiento del caso centralizado a medida que la cantidad de sensores. El motivo por el cual esta prueba fue realizada solo sobre el caso centralizado es porque se cree que el mismo es el más apto para usar en el **Campus Inteligente**.

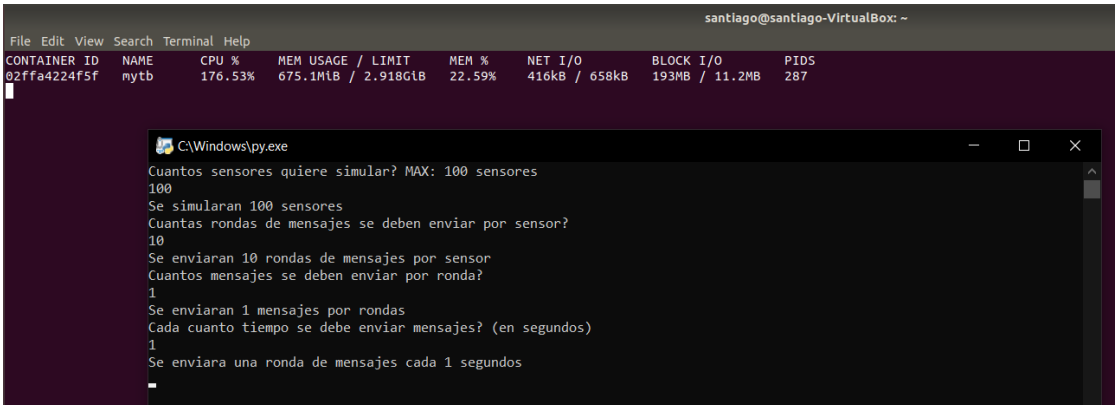

Figura 22: Resultados de la prueba sobre la solución descentralizada

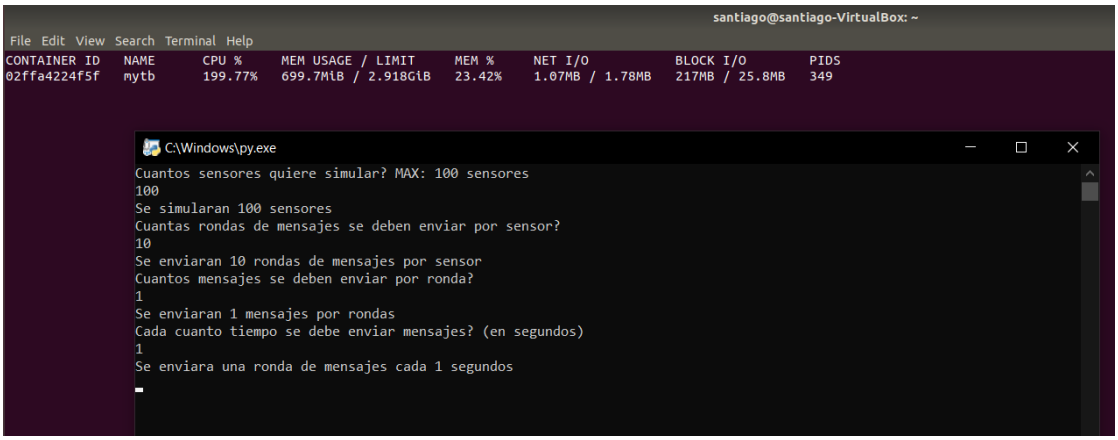

Figura 23: Resultados de la prueba sobre la solución centralizada

La prueba constó de la utilización del *script* en cuatro tandas las cuales simulaban 25, 50, 75 y 100 dispositivos cada una. Luego, utilizando el mismo comando que para las pruebas anteriores se obtuvieron los valores que se aprecian en la Figura 24 y la Figura 25.

En la Figura 24 se logra apreciar que el consumo de CPU crece de una manera aproximada a la forma lineal. Esto es algo destacable ya que el aumento de dispositivos no dispara de manera descontrolada el consumo de recursos y esto es muy bueno para el **Campus Inteligente**.

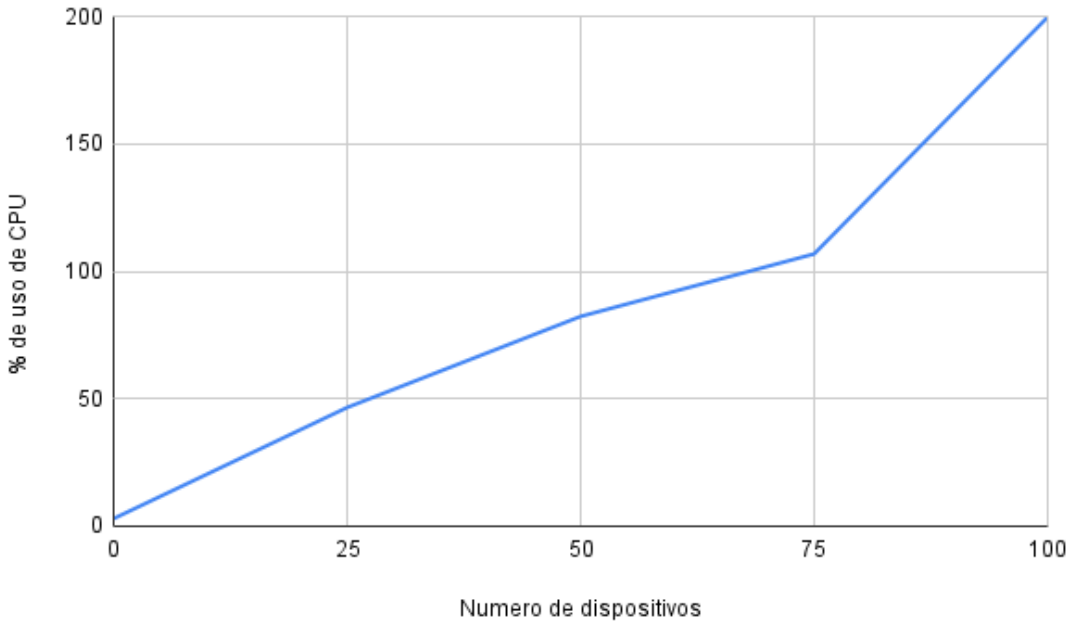

Figura 24: Consumo de CPU al aumentar los dispositivos conectados

Por su parte la Figura 25 nos permite ver como evoluciona el consumo de RAM a medida que aumentan los dispositivos conectados. El mismo aumenta muy lentamente manteniéndose entre 24 % y 26 % en todas las pruebas realizadas. También una característica muy beneficiosa para el *Campus Inteligente*.

6.2.3. Conclusiones de las pruebas

Ambas soluciones han mostrado un comportamiento similar al esperado, donde la solución centralizada ofrece mejores tiempos de respuesta y menor tamaño de mensajes, pero con la contrapartida de que requiere de más recursos en la plataforma de IoT para lograr procesar los datos.

En resumen se considera que la solución centralizada es la más apta para utilizar en el desarrollo real del **Campus Inteligente**, no porque sea estrictamente mejor que la otra en rendimiento, si no debido al mejor manejo de la información que realiza.

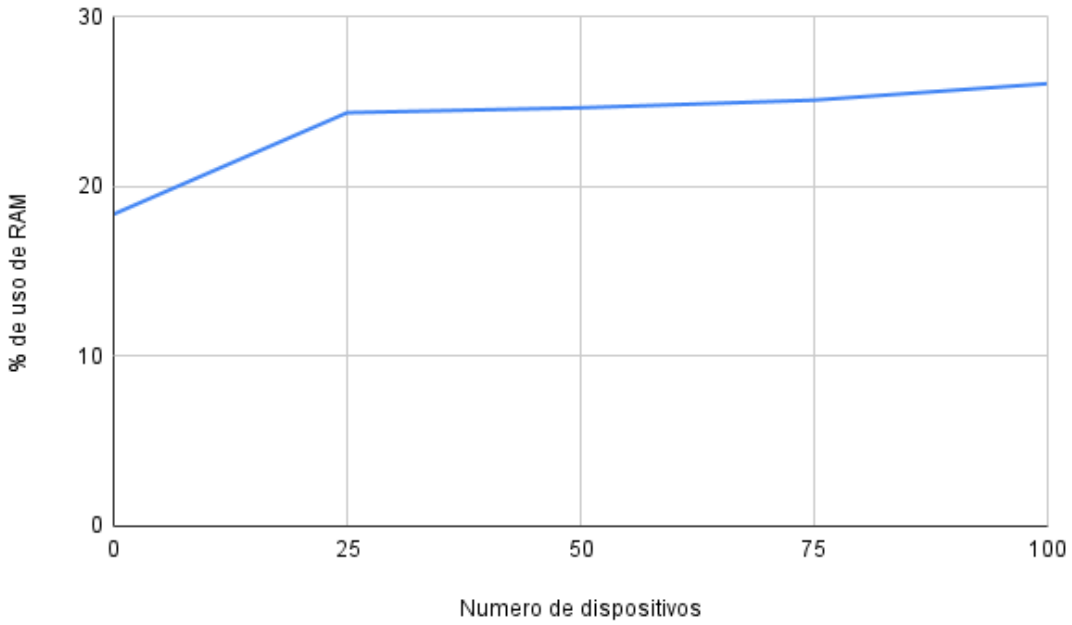

Figura 25: Consumo de RAM al aumentar los dispositivos conectados

7. Despliegue Campus Inteligente

En esta sección se discutirá qué hardware se requerirá a la hora de realizar el despliegue del **Campus Inteligente**. Como ya se ha mencionado se tendrá en cuenta al predio comprendido por las facultades de Ingeniería, Ciencias económicas y de administración; y de Arquitectura. Todas ellas de la Universidad de la República en Montevideo, Uruguay, abarcando:

- Edificio de la Facultad de Arquitectura, con 26 salones y 2 salas de máquinas
- Edificio principal de la Facultad de Ingeniería con 40 salones y 4 salas de máquinas
- Edificio principal de la Facultad de ciencias económicas y de administración con 23 salones
- Aulario del Faro con 15 salones
- Aulario área social y artística con 32 salones

En estos edificios tenemos un total de 142 salones. Estos datos fueron obtenidos de las páginas web de las facultades [2][7][23]. Esto indica que se requerirían al menos 142 sensores de cada tipo para realizar un despliegue total del **Campus Inteligente**, esto se debe a que se colocaría un conjunto de sensores por cada salón.

En cuanto a los *gateways* necesarios, para el Wi-Fi se debería relevar las capacidades de conexión existente en los campus de las facultades para luego realizar un plan que permita obtener conexión Wi-Fi en todos los salones. Por otro lado, para definir cuántos *gateways* LoRa se requieren se busco el alcance de los *gateways* utilizado para el prototipo. Según LoRa Developer Portal [33], el rango en zonas urbana de LoRa es de 5 kilómetros. Con esto un solo *gateway* podría cubrir toda la zona de interés. A pesar de esto con el fin de evitar pérdida de información por interferencias, se recomienda utilizar tres *gateways* para cubrir las zonas de interés, además se sugiere ubicar cada *gateway* en uno de los edificios principales de cada facultad.

Para los servidores se recomienda utilizar dos máquinas dedicadas, una para el servidor LoRa y otra para la plataforma de IoT. Además, se recomienda ubicarlos en el edificio principal de la Facultad de Ingeniería, si bien cualquiera de los tres edificios principales sería una buena opción dada la experiencia de la Facultad de Ingeniería se considera que la es la mejor opción para abarcar los servidores y administrarlos.

A su vez se recomienda que la conexión entre las capas de Observación, Procesamiento y Seguridad sea en una red local, para fortalecer la seguridad. Según lo visto en la sección de **6.1 Estudio: Performance Evaluation of Open Source IoT Platforms** la capa de Procesamiento con un Intel Core I7-7500U 2.7 GHz, 8 GB de RAM y con el sistema operativo Ubuntu 18.04, se cubrirían los requerimientos mínimos para dicha capa. Por lo tanto el servidor de dicha capa debe tener al menos esas características. Por otro lado, para los demás servidores de la capa de Observación y el *firewall*, pueden ubicarse todos en una misma máquina. Por último para el servidor *Proxy* y el *firewall* de la capa de Seguridad no se específica el hardware recomendado ya que el mismo dependerá de los usuarios a utilizar y las funciones que finalmente se le asignen al *proxy*. Por ejemplo podrían tener un

solo *firewall* independiente y más de un *proxy* para balancear la carga.

8. Conclusiones

Se logró presentar un diseño para el **Campus Inteligente**. Para ello se llevó a cabo un relevamiento de arquitecturas y protocolos, mediante el cual se decidió utilizar una arquitectura de seis capas, ya que presenta una mayor robustez en cuanto a la seguridad además de definirse que protocolos de comunicación utilizar: LoRa, Wi-Fi e internet (HTTP y IP/UDP).

Otra de las tareas que se realizó fue la selección de plataforma la cual a partir de una comparación se eligió ThingsBoard como la plataforma a utilizar.

Luego se implementó un prototipo de la arquitectura propuesta, el cual demostró la viabilidad de la misma. Otro punto que le da valor al prototipo es que ofrece la capacidad de evaluar distintas soluciones en un entorno controlado. Esto se demostró en la sección **6.2 Pruebas de Rendimiento**, donde dos soluciones para el mismo caso de uso fueron probadas, permitiendo así seleccionar la que mejor se adapta a las necesidades del **Campus Inteligente**.

Con lo mencionado anteriormente se considera que los objetivos planteados han sido cumplidos satisfactoriamente. A pesar de esto aún hay áreas que se beneficiarían de mayores esfuerzos. Dichas áreas serán mencionadas en la sección **9. Trabajo Futuro**

9. Trabajo futuro

En las siguientes secciones se describirán las áreas donde se puede profundizar con el fin de mejorar al **Campus Inteligente**.

9.1. Seguridad

Como se ha mencionado a lo largo del proyecto la seguridad es importante en los desarrollos de IoT. Y a pesar de que en el prototipo se realizaron algunos esfuerzos como identificar a los mensajes de los dispositivos con tokens o autenticar a los usuarios de la página web. Quedan esfuerzos a implementar para mejorar este aspecto. Como pueden ser configurar que las comunicaciones se realicen mediante HTTPS o implementar los *Firewalls*.

9.2. Enviar acciones a dispositivos

Otra área en la cual se puede seguir profundizando es en el envío de datos/acciones a los dispositivos desde la plataforma. Con el fin de activar un actuador y obtener un cambio en el ambiente. Esto se sabe que es posible con la plataforma seleccionada ya que hay guías sobre eso. Por ejemplo: de como prender y apagar un led desde la plataforma [46] o de como hacer invocaciones RPC (*Remote Procedural Calls*).

9.3. Montevideo Inteligente

Por último pero no menos importante se podría trabajar en adaptar el prototipo para un escenario más grande y complejo, como podría ser Montevideo Inteligente. Para esto habría que tener en cuenta que los números de los sensores crecerían de gran manera. Ya no sería posible desarrollar una solución con la arquitectura monolítica de ThingsBoard. Por lo que instalar y configurar la arquitectura de micro servicios sería necesario.

También hay que destacar que algunos desafíos no tenidos en cuenta ganarán relevancia como la ubicación de los dispositivos para lograr abarcar la mayor área con el menor costo y el consumo de energía.

Referencias

[1] 3gpp. *Standards for the IoT*. URL: https://www.3gpp.org/news-events/1805-iot_r14. (accessed: 08.09.2022).

[2] Facultad de ciencias económicas y de administración. *Ubicación locaciones de cursos FCEA*. URL: https://fcea.udelar.edu.uy/images/micrositios/comunicacion/Novedades/2022/Salones_2022.pdf. (accessed: 01.10.2022).

[3] Gabriela Wynants Lombardini Alexis Arriola Garcia. *Relevamiento de arquitecturas para desarrollo de aplicaciones de Internet de las Cosas*. URL: https://www.colibri.udelar.edu.uy/jspui/handle/20.500.12008/19035. (accessed: 10.12.2022).

[4] Santiago Alles. *campus-inteligente*. URL: https://gitlab.fing.edu.uy/santiago.alles.conde/campus-inteligente/-/tree/main/. (accessed: 11.10.2022).

[5] Wi-Fi Alliance. *Wi-Fi Alliance*. URL: https://www.wi-fi.org. (accessed: 05.09.2022).

[6] ZigBee Alliance. *ZigBee Specification FAQ*. URL: https://web.archive.org/web/20130627172453/http://www.zigbee.org/Specifications/ZigBee/FAQ.aspx. (accessed: 02.09.2022).

[7] Facultad de arquitectura. *Facultad de arquitectura servicios*. URL: http://www.fadu.edu.uy/patio/wp-content/uploads/downloads/2012/06/plano_-farq_DIC2011_web.pdf. (accessed: 01.10.2022).

[8] Bluetooth. *Bluetooth® Technology Website » Feed*. URL: https://www.bluetooth.com/specifications/specs/core-specification-5-3/. (accessed: 01.09.2022).

[9] Muhammad Burhan y col. «IoT elements, layered architectures and security issues: A comprehensive survey». En: *Sensors (Switzerland)* 18.9 (sep. de 2018). ISSN: 14248220. DOI: 10.3390/s18092796.

[10] Steven Chim. *http-proxy-middleware*. URL: https://www.npmjs.com/package/http-proxy-middleware. (accessed: 21.09.2022).

[11] ChirpStack. *Quickstart Debian or Ubuntu - ChirpStack open-source LoRaWAN Network Server*. URL: https://www.chirpstack.io/project/guides/debian-ubuntu/. (accessed: 09.02.2022).

[12] Francisco Crizul, G Gerardo e Ing Eduardo Gramp. *Localización en interiores utilizando infraestructura de Internet de las Cosas*. URL: https://www.colibri.udelar.edu.uy/jspui/handle/20.500.12008/27962. (accessed: 10.12.2022).

[13] Federico Detta. *Aplicación de IoT con diversas tecnologías inalámbricas*. URL: https://www.colibri.udelar.edu.uy/jspui/handle/20.500.12008/23742. (accessed: 10.12.2022).

[14] Docker. *docker stats — Docker Documentation*. URL: https://docs.docker.com/engine/reference/commandline/stats/. (accessed: 28.09.2022).

[15] Docker. *Empowering App Development for Developers — Docker*. URL: https://www.docker.com/. (accessed: 03.03.2022).

[16] Dragino. *LPS8 Indoor LoRaWAN Gateway*. URL: https://www.dragino.com/products/lora-lorawan-gateway/item/148-lps8.html. (accessed: 09.02.2022).

[17] Dragino. *User Manual*. URL: https://www.dragino.com/downloads/downloads/LoRa_Gateway/LPS8/LPS8_LoRaWAN_Gateway_User_Manual_v1.3.2.pdf. (accessed: 04.11.2022).

[18] Vitor Fialho y Fernando Fortes. «Low Power IoT Network Sensors Optimization for Smart Cities Applications». En: *SEST 2019 - 2nd International Conference on Smart Energy Systems and Technologies* (2019), págs. 10-15. DOI: 10.1109/SEST.2019.8849071.

[19] Fiware. *Fiware*. URL: https://www.fiware.org. (accessed: 10.12.2022).

[20] geekflare. *12 plataformas y herramientas de Internet de las cosas (IoT) de código abierto*. URL: https://geekflare.com/es/iot-platform-tools/. (accessed: 28.02.2022).

[21] Edgar Gomez. *¿Cuál es la diferencia entre LTE-M y NB-IoT?* URL: https://es.linkedin.com/pulse/cu%C3%A1l-es-la-diferencia-entre-lte-m-y-nb-iot-edgar-gomez. (accessed: 08.09.2022).

[22] Google. *Google Trends*. URL: https://trends.google.com/trends/?geo=UY. (accessed: 26.01.2022).

[23] Facultad de Ingeniería. *Cifras — Facultad de Ingeniería*. URL: https://www.fing.edu.uy/es/plandeobras/el-edificio/cifras. (accessed: 04.04.2022).

[24] Ahmed A. Ismail, Haitham S. Hamza y Amira M. Kotb. «Performance Evaluation of Open Source IoT Platforms». En: *2018 IEEE Global Conference on Internet of Things, GCIoT 2018* (2019). DOI: 10.1109/GCIoT.2018.8620130.

[25] Jerry Liu. *gams-dataset*. URL: https://github.com/twairball/gams-dataset. (accessed: 07.03.2022).

[26] Mainflux. *Mainflux*. URL: https://docs.mainflux.io. (accessed: 01.03.2022).

[27] Inc Meta Platforms. *React – Una biblioteca de JavaScript para construir interfaces de usuario*. URL: https://es.reactjs.org. (accessed: 05.04.2022).

[28] MobiDev. *IoT Smart Parking System - Smart Parking Solution based on IoT Sensors and AWS IoT*. URL: https://www.youtube.com/watch?v=-9s9QkpRzWs. (accessed: 19.09.2022).

[29] Openremote. *Choosing among the best Open Source IoT platforms in the market — OpenRemote*. URL: https://openremote.io/making-sense-of-open-source-iot-platforms-to-avoid-pitfalls/. (accessed: 28.02.2022).

[30] Openremote. *Openremote*. URL: https://github.com/openremote/openremote/wiki. (accessed: 01.03.2022).

[31] Oracle. *Oracle VM VirtualBox*. URL: https://www.virtualbox.org. (accessed: 03.03.2022).

[32] Walid Osamy, Ahmed A. El-Sawy y Ahmed Salim. «CSOCA: Chicken Swarm Optimization Based Clustering Algorithm for Wireless Sensor Networks». En: *IEEE Access* 8 (2020), págs. 60676-60688. ISSN: 21693536. DOI: 10.1109/ACCESS.2020.2983483.

[33] LoRa — Developer Portal. *LoRa and LoRaWAN: Technical overview — DEVELOPER PORTAL*. URL: https://lora-developers.semtech.com/documentation/tech-papers-and-guides/lora-and-lorawan/. (accessed: 04.04.2022).

[34] Python. *Welcome to Python.org*. URL: https://www.python.org. (accessed: 27.09.2022).

[35] Tie Qiu y col. «How can heterogeneous internet of things build our future: A survey». En: *IEEE Communications Surveys and Tutorials* 20.3 (jul. de 2018), págs. 2011-2027. ISSN: 1553877X. DOI: 10.1109/COMST.2018.2803740.

[36] D Ramsden. «Optimization approaches to sensor placement problems». En: 2009.August (2009). URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.159.3832&rep=rep1&type=pdf.

[37] Record.evolution. *5 Open-Source IoT Platforms: Making Innovation Count — Record Evolution*. URL: https://www.record-evolution.de/en/blog/open-source-iot-platforms-making-innovation-count/. (accessed: 28.02.2022).

[38] Ola Salman y col. *IoT survey: An SDN and fog computing perspective*. Oct. de 2018. DOI: 10.1016/j.comnet.2018.07.020.

[39] Pallavi Sethi y Smruti R. Sarangi. *Internet of Things: Architectures, Protocols, and Applications*. 2017. DOI: 10.1155/2017/9324035.

[40] Sparkfun. *ESP32 LoRa 1-Channel Gateway - SPX-14893 - SparkFun Electronics*. URL: https://www.sparkfun.com/products/retired/14893. (accessed: 09.02.2022).

[41] N. N. Srinidhi, S. M. Dilip Kumar y K. R. Venugopal. *Network optimizations in the Internet of Things: A review*. Feb. de 2019. DOI: 10.1016/j.jestch.2018.09.003.

[42] Accent Systems. *Diferencias entre NB-IOT y LTE-M*. URL: https://accent-systems.com/es/diferencias-nb-iot-lte-m/. (accessed: 08.09.2022).

[43] ThingsBoard. *Getting Started with ThingsBoard — ThingsBoard Community Edition*. URL: https://thingsboard.io/docs/getting-started-guides/helloworld/. (accessed: 03.03.2022).

[44] ThingsBoard. *Installing ThingsBoard CE on Ubuntu Server — ThingsBoard Community Edition*. URL: https://thingsboard.io/docs/user-guide/install/ubuntu/. (accessed: 03.03.2022).

[45] ThingsBoard. *ThingsBoard Documentation — ThingsBoard Community Edition*. URL: https://thingsboard.io/docs/. (accessed: 01.03.2022).

[46] ThingsBoard. *ThingsBoard: ESP8266 GPIO control over MQTT using Thingsboard*. URL: https://blog.thingsboard.io/2017/01/esp8266-gpio-control-over-mqtt-using.html. (accessed: 05.10.2022).

[47] Ubuntu. *Ubuntu Releases*. URL: https://releases.ubuntu.com. (accessed: 03.03.2022).

[48] Vedcraft. *Emerging Open Source IoT Platforms You Should Not Miss - Vedcraft*. URL: https://vedcraft.com/tech-trends/emerging-opensource-iot-platforms/. (accessed: 28.02.2022).

[49] vemcogroup. *How IoT Is Creating a Totally New Smart Parking Concept*. URL: https://vemcogroup.com/blog/how-iot-is-creating-a-totally-new-smart-parking-concept. (accessed: 19.09.2022).

10. Apéndices

Junto con el informe se entregan dos archivos *.rar*. En el archivo *Maquina para pruebas.rar* contiene la máquina virtual utilizada para las pruebas de rendimiento, la cual cuenta con la plataforma ThingsBoard instalada en un contenedor de Docker. Dicha plataforma tiene configurada la regla de conteo de dispositivos y los 100 dispositivos utilizados por el script del archivo script.py. Esta máquina puede ser descargada desde el link:

`https://mega.nz/file/M0lSTbzY#aAwvfn2LeOcBJCg7OPNUmgMsOTD7HSUEhk\_r7oZHp-Y`

Por su parte el archivo *Prototipo Campus Inteligente.rar* contiene la máquina virtual que contiene el servidor ChirpStack, la plataforma de IoT ThingsBoard en un contenedor de Docker y el código necesario para formatear las placa Sparkfun ESP32 LoRa 1-Channel Gateway. La máquina puede ser descargada desde el link:

`https://mega.nz/file/zsxkjRZR#mAV7nqCir1tmSSY1psG9MHc39S53ux72aEoksl8Ct0A`

A la hora de importar las máquina virtuales se sugiere seguir la siguiente guía:

`https://www.youtube.com/watch?v=5mzk1uiQvXs`

Además en el repositorio de GitLab *campus-inteligente* [4] se pueden encontrar el resto de los códigos desarrollados: las dos aplicaciones web para el campus inteligente, el simulador de sensores de aire, el script para las pruebas de rendimiento y la prueba de JMeter.

Por último, se destaca que a la hora de poner en marcha la máquina virtual *Prototipo Campus Inteligente* o cualquiera de los códigos entregados se debe corroborar que las direcciones IP que se utilizan sean correctas, de lo contrario la comunicación no será efectiva. El caso más complejo de configurar se encuentra en ChirpStack donde hay que acceder a los archivos de configuración para modificar las IP y luego reiniciar los servicios. Las rutas a los archivos y comandos para iniciar los servicios se encuentran en la guía de instalación de ChirpStack.

A continuación se presentan guías de las distintas instalaciones realizadas a lo largo de este proyecto. Se recomienda no utilizar las contraseñas presentes en estas guías si no que se recomienda utilizar contraseñas seguras.

10.1. Instalación y configuración de VirtualBox

La aplicación utilizada para la creación de las máquinas virtuales fue VirtualBox [31]. La cual se instaló sobre una computadora con un I5 de octava generación, con 8 GB de RAM y Windows 10.

La versión instalada fue la 6.1.22 y la versión de Ubuntu que se utilizó fue la 18.04.5.

Los pasos seguidos para realizar la instalación fueron:

1. Descargar el instalador de VirtualBox la página web:

   `https://www.virtualbox.org/wiki/Downloads`

2. Una vez descargado el instalador ejecutarlo.

3. Navegar en el instalador:

   - Clic en el botón *Siguiente* en la página de bienvenida.
   - Definir ruta para la instalación y hacer clic en el botón *Siguiente*.
   - Seleccionar si se desea: crear un icono en el escritorio, crear un icono en la barra de inicio rápido, *'Register file associations"* y/o *Create start menú entries*. Luego hacer clic en el botón *Siguiente*.
   - Hacer clic en el botón *Yes* en la advertencia de desconexión de red.
   - Hacer clic en el botón *Install*.
   - Dar permiso al instalador para hacer cambios en el dispositivo y esperar que finalice la institución.
   - Hacer clic en el botón *Finish* una vez la instalación haya terminado.

4. Abrir la aplicación para corroborar que la instalación haya sido exitosa.

Una vez instalado VirtualBox se debe crear o importar una máquina virtual. A continuación se enumeran los pasos necesarios para crear una máquina virtual:

1. Descargar la imagen del sistema operativo deseado, en nuestro caso Ubuntu 18.04

2. Abrir la aplicación VirtualBox.
3. Hacer clic en el botón *'Nueva'*
4. Seguir los pasos en el modal. No se recomienda utilizar el modo experto a no ser que ya haya utilizado VirtualBox con anterioridad.
5. Completar los campos *Nombre, Carpeta de máquina y versión*, que serán el nombre que se le asignará a la máquina, donde se guardará la misma y la versión del sistema operativo que se usará respectivamente. Luego hacer clic en *'Siguiente'*.
6. Seleccionar el tamaño de RAM deseado y hacer clic en *'Siguiente'*.
7. Seleccionar la opción *'Crear un disco virtual ahora'* y hacer clic en crear.
8. Seleccionar la opción VDI y clic en *'Siguiente'*
9. Seleccionar la opción *'Reservado dinámicamente'* y hacer clic en *'Siguiente'*
10. Definir el tamaño máximo que puede adquirir el disco duro y hacer clic en el botón de crear.
11. Una vez finaliza la creación de la máquina, hacer clic derecho sobre la misma y luego clic sobre la opción *Configuración...'*.
12. Ir a la pestaña de almacenamiento, en controlador IDE seleccionar donde dice vacío. Luego hacer clic en el icono de un disco azul, que se encuentra al final del campo *'Unidad Óptica'*.
13. Hacer clic en *'Seleccionar un archivo de disco'*
14. Seleccionar la imagen del sistema operativo previamente descargada y darle abrir.
15. Ir a la pestaña de Red y seleccionar el adaptador adaptador puente o *Bridged Adapter*.
16. Luego hacer clic en aceptar e iniciar la máquina virtual.

17. Una vez iniciada la máquina virtual seguir los pasos para instalar el sistema operativo.

18. Una vez instalado el sistema operativo se deben agregar las *Guest Additions* que es un paquete de software que añade funcionalidades y mejora el rendimiento de la máquina virtual.

19. Para eso hay que ir a la pestaña de dispositivos en la parte superior de la ventana.

20. Seleccionar la opción *Insertar imagen de CD de las 'Guest Additions'* del menú.

21. Ejecutar el CD en la máquina virtual.

22. Una vez finalizada la ejecución reiniciar la máquina virtual.

23. Con eso tenemos la máquina virtual lista para ser utilizada.

10.2. Instalación de ChirpStack

Para la instalación de ChirpStack se siguieron los siguientes pasos que derivaron de las guías presentes en la página de ChirpStack [11].

1. Primero se instalaron las dependencias necesarias utilizando el siguiente comando:

```
sudo apt install mosquitto mosquitto-clients redis-server
   redis-tools postgresql
```

2. Luego debemos hacer la configuración de PostgreSQL para ello accedemos usando:

```
sudo -u postgres psql
```

3. Una vez dentro de PostgreSQL corremos los siguientes comandos para crear los roles y las bases de datos que se necesitaran para el funcionamiento de ChirpStack:

```
create role chirpstack_as with login password 'dbpassword';
create role chirpstack_ns with login password 'dbpassword';
create database chirpstack_as with owner chirpstack_as;
create database chirpstack_ns with owner chirpstack_ns;
\c chirpstack_as
create extension pg_trgm;
create extension hstore;
\q
```

4. A continuación se configura el repositorio de software de ChirpStack usando los siguientes comandos:

```
sudo apt install apt-transport-https dirmngr
sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys
    1CE2AFD36DBCCA00
sudo echo "deb https://artifacts.chirpstack.io/packages/3.x/
   deb stable main" | sudo tee /etc/apt/sources.list.d/
   chirpstack.list
sudo apt update
```

5. Una vez terminada la configuración previa empezamos a instalar los componentes de ChirpStack. Primero instalamos el *Gateway Bridge* con el siguiente comando:

```
sudo apt install chirpstack-gateway-bridge
```

6. Una vez finalizada la instalación, hay que correr el siguiente comando para iniciar el servicio.

```
sudo systemctl start chirpstack-gateway-bridge
```

7. Luego debemos intalar el *Network Server* utilizando el siguiente comando:

```
sudo apt install chirpstack-network-server
```

8. Cuando la instalación finalice configurar el archivo:

```
/etc/chirpstack-gateway-bridge/chirpstack-gateway-bridge.toml
```

   En el cual debemos colocar la siguiente línea para configurar la contraseña de la base de datos:

```
dsn="postgres://chirpstack_ns:dbpassword@localhost/
    chirpstack_ns?sslmode=disable"
```

9. Además debemos configurar la banda para que sea compatible con la que usaremos en los sensores: en nuestro caso utilizamos:

```
[network_server.band]
name="AU915"

[network\_server.network\_settings]
enabled\_uplink\_channels=[0, 1, 2, 3, 4, 5, 6, 7, 64]
```

10. Luego iniciamos el *gateway* con el siguiente comando:

```
sudo systemctl start chirpstack-network-server
```

11. A continuación se debe instalar el *Application Server* usando el comando:

```
sudo apt install chirpstack-application-server
```

12. Una vez instalado el servidor de aplicación, editar el archivo:

```
/etc/chirpstack-application-server/chirpstack-application-
    server.toml
```

13. Editar el archivo utilizando los siguientes valores:

```
[general]
log_level=4

[postgresql]
dsn="postgres://chirpstack_as:dbpassword@localhost/
    chirpstack_as?sslmode=disable"

[application_server.external_api]
jwt_secret="verysecret"
```

14. Por último debemos iniciar el servidor de aplicación con el comando:

```
sudo systemctl start chirpstack-application-server
```

    Con eso tenemos al ecosistema de ChirpStack corriendo en nuestra máquina.

10.2.1. Configuración de una aplicación

En esta sección vamos a explicar como configurar una aplicación en ChirpStack una vez ya está instalado:

1. Primero debemos acceder a la aplicación con nuestras credenciales.
2. Ahora debemos iniciar asociando un servidor de red. Para eso, se debe hacer clic en la opción *Network-Server* en el menú de la izquierda.
3. Hacer clic en *ADD*.
4. Completar los campos con el nombre y la dirección del servidor que se desea agregar.
5. Luego hacemos clic en la opción *Gateway-profiles* en el menú de la izquierda
6. Hacemos clic en el botón *CREATE*

7. Completar los campos con el nombre deseado para nuestro perfil, así como el periodo de tiempo cada el cual se estarán enviando datos, los canales de LoRa soportados y el network server a utilizar. Por ejemplo:

    - *Name*: Dragino Profile.
    - *Stats Interval*: 30 segundos.
    - *Enabled Channels*: 0,1,2

8. Hacer clic en *ADD EXTRA CHANNEL*.

9. Completar los campos, por ejemplo con los siguientes valores:

    - *Modulation*: LoRa
    - *Bandwidth*:125 kHz
    - *Frequency*: 916800000
    - *Spreading factors*:10

10. Con todos los campos ya completados hacemos clic en *CREATE GATEWAY-PROFILE*.

11. A continuación, hacemos clic en *Gateways* en el menú de la izquierda y luego clic en *CREATE*

12. Completar los campos requeridos. A contención un ejemplo:

    - *Name*: Mi *gateway*.
    - *Description*: Descripción de Mi *gateway*.
    - *Gateway ID*: a840411eb7204150.
    - *Network Server*: Selecciona el servidor de red deseado.
    - *Gateway Profile*: Seleccionamos el perfil de *gateway* creado.

13. Con los campos completos hacemos clic en *CREATE GATEWAY*.

14. A continuación se debe crear una aplicación, pero antes de eso es necesario crear un perfil de servicio.

15. Hacer clic en *Service profiles* en el menú de la izquierda seguido de un clic en *CREATE*.

16. Completar el nombre del servicio y el servidor de red que utilizará y hacemos clic en *CREATE SERVICE-PROFILE*.

17. Ahora vamos a *Applications* en el menú de la izquierda seguido de un clic en *CREATE*

18. Completar los campos con el nombre de la aplicación creada, una descripción y el *service profile* a utilizar. Seguido hacemos clic en *CREATE APPLICATION*.

19. Antes de crear el dispositivo que enviará los datos debemos crearle un *Device Profile*.

20. Ir al menú de la izquierda y hacer clic en *Device profiles*, luego hacer clic en *CREATE*.

21. Completar los campos de forma que quede algo similar a lo siguiente

    - *Name*: Mi device profile.
    - *Network Server*: mismo que se utilizó al crear los componentes anteriores
    - *LoRaWAN MAC version*: elegir la que utilices en este proyecto se utilizó la versión 1.0.1.
    - *LoRaWAN regional parameters revision*: en este proyecto se utilizó el valor **A**.

22. Luego hacer clic en la pestaña superior *JOIN (OTAA / ABP)* y completar la *Factory preset frequency* con la frecuencia que se este utilizando en este proyecto fue 916800000.

23. Clic en *CREATE DEVICE-PROFILE*

24. Acceder al perfil recién creado y actualizarlo para que en el campo *ADR algorithm* utilice *Default ADR algorithm*.

25. Ahora solo falta crear el dispositivo. Con todos los componentes ya creados, seleccionamos la aplicación, en la tabla de aplicaciones que se muestra cuando se hace clic en el botón *Applications* del menú de la izquierda.

26. Dentro de la sección de *Devices* hacemos clic en *CREATE*.

27. Completar los campos para obtener algo similar a lo siguiente:

    - *Name*: Mi device.
    - *Description*: Descripción de Mi device.
    - *Device EUI*: Completar con el EUI del dispositivo.
    - *Device profile*: Seleccionar el perfil creado.

28. Con todos los campos ya completados solo falta hacer clic en *CREATE DEVICE*

Con eso ya queda configurada la aplicación con todos los componentes necesarios.

10.2.2. Integración con ThingsBoard

Como ThingsBoard no soporta LoRa se debió utilizar ChirpStack como mediador entre los sensores y la plataforma. Debido a esto se integró ChirpStack con ThingsBoard. En esta sección describiremos los pasos a seguir para realizar dicha integración.

1. Dentro de la aplicación web de ChirpStack hacer clic en el botón *Applications* del menú de la izquierda.

2. Seleccionar la aplicación que se desea integrar con ThingsBoard.

3. De las pestañas superiores seleccionar la que dice *Integrations*.

4. Hacer clic en *ADD* en la tarjeta de ThingsBoard.

5. Colocar la dirección del servidor de things board y hacer clic en *ADD INTEGRATION*.

6. Clic en la pestaña *DEVICES*.

7. Para cada dispositivo que se desee integrar con ThingsBoard seguir los siguientes pasos:

   a) Clic en el dispositivo para ver su información.

   b) Clic en la pestaña *Configuration*.

   c) Dentro de esa pestaña hacer clic en la sub-pestaña *VARIABLES*.

   d) Clic en *ADD VARIABLES*.

   e) Completar el nombre de la variable con 'ThingsBoardAccessToken' y el valor con el *access token* de la representación del dispositivo en ThingsBoard.

8. Una vez configurados todos los dispositivos la integración ya se encuentra finalizada.

10.3. Configuración del Dragino

Para lograr configurar el *gateway* Dragino utilizamos la conexión mediante Ethernet, esto fue logrado configurando el puerto Ethernet con los siguientes valores:

- IP: 172.31.255.253
- Netmask: 255.255.255.252

Luego simplemente accediendo a la URL: *http://172.31.255.253:8000* somos capaces de comunicarnos con el gateway y modificar sus configuraciones.

Por más información se recomienda visitar el Manual del Usuario [17].

10.4. Instalación de ThingsBoard

En esta sección explicaremos cómo instalar ThingsBoard. Para esto hay dos maneras de hacerlo: utilizando Docker o no utilizándolo. Se utilizó como base la documentación de la página de ThingsBoard [43][44].

A continuación se explicarán la manera que se utilizó para este proyecto, la cual fue utilizando Docker ya que es la manera más sencilla y rápida de tener una versión de ThingsBoard corriendo localmente. Recordar que se utilizó Ubuntu 18.4 como sistema operativo.

10.4.1. Instalación con Docker

El pre-requisito para poder seguir esta opción de instalación es tener Docker instalado en la máquina que se desea instalar ThingsBoard.

Para configurar un contenedor de Docker es sencillo solo hay que ejecutar la siguiente línea de comando:

```
mkdir -p ~/.mytb-data && sudo chown -R 799:799 ~/.mytb-data
mkdir -p ~/.mytb-logs && sudo chown -R 799:799 ~/.mytb-logs
docker run -it -p 9090:9090 -p 7070:7070 -p 1883:1883 -p
    5683-5688:5683-5688/udp -v ~/.mytb-data:/data \
-v ~/.mytb-logs:/var/log/thingsboard --name mytb --restart always
    thingsboard/tb-postgres
```

Una vez finalizada la ejecución de dicho comando tendremos a ThingsBoard disponible en la URL: *http://localhost:8080*

10.4.2. Instalación sin Docker

La instalación sin Docker de ThingsBoard es un poco más compleja que la anterior ya que hay que configurar la base de datos y el servicio de encolamiento de mensajes (*queue service*).

Para instalarla de esta manera recomendamos seguir la guía que se encuentra en el enlace [44]. A pesar de lo mencionado anteriormente a continuación se presentan los pasos seguidos para su instalación:

1. Primero se instaló Java 11 (OpenJDK) con los siguientes comandos:

```
sudo apt update
sudo apt install openjdk-11-jdk
```

2. Luego debemos descargar los paquetes de instalación, utilizando el comando:

```
wget https://github.com/thingsboard/thingsboard/releases/
   download/v3.4.1/thingsboard-3.4.1.deb
```

3. Usando el siguiente comando instalamos ThingsBoard como un servicio:

```
sudo dpkg -i thingsboard-3.4.1.deb
```

4. Luego hay que instalar la base de datos principal para ThingsBoard, en nuestro caso seleccionamos PostgreSQL. Para la instalación se siguieron los siguientes pasos:

```
sudo apt install -y wget
wget --quiet -O - https://www.postgresql.org/media/keys/
   ACCC4CF8.asc | sudo apt-key add -RELEASE=\$(lsb\_release -
   cs)
echo "deb http://apt.postgresql.org/pub/repos/apt/ \${RELEASE
   }"-pgdg main | sudo tee /etc/apt/sources.list.d/pgdg.list
sudo apt update
sudo apt -y install postgresql-12
sudo service postgresql sta
```

5. Despues de instalada la base de datos se debe crear un usuario para ello se utilizaron los siguientes comandos:

```
sudo su - postgres
psql
\password
\q
```

6. Luego creamos una tabla siguiendo los siguientes pasos:

```
psql -U postgres -d postgres -h 127.0.0.1 -W
CREATE DATABASE thingsboard;
\q
```

7. Después de creada la tabla debemos configurarla en ThingsBoard, entramos en el archivo de configuraciones usando la siguiente comando:

```
sudo nano /etc/thingsboard/conf/thingsboard.conf
```

8. Dentro del archivo debemos agregar las siguientes línea s:

```
# DB Configuration
export DATABASE_TS_TYPE=sql
export SPRING_DATASOURCE_URL=jdbc:postgresql://localhost
   :5432/thingsboard
export SPRING_DATASOURCE_USERNAME=postgres
export SPRING_DATASOURCE_PASSWORD=
   PutYourPostgreSQLPasswordHere
# Specify partitioning size for timestamp key-value storage.
   Allowed values: DAYS, MONTHS, YEARS, INDEFINITE.
export SQL_POSTGRES_TS_KV_PARTITIONING=MONTHS
```

9. Luego pasamos a configurar el sistema de encolamiento de mensaje, para nuestra utilización usamos la opción en memoria. Para ello debemos editar el siguiente archivo:

```
sudo nano /etc/thingsboard/conf/thingsboard.conf
```

   Agregando la siguiente línea :

```
export JAVA\_OPTS="\$JAVA\_OPTS -Xms256M -Xmx256M"
```

10. Una vez realizadas las configuraciones debemos correr con el siguiente script:

```
sudo /usr/share/thingsboard/bin/install/install.sh --loadDemo
```

11. Por último debemos iniciar la instancia de ThingsBoard con el siguiente comando:

```
sudo service thingsboard start
```

10.4.3. Configurar un dispositivo

A la hora de configurar un dispositivo en ThingsBoard lo primero que hay que hacer es crear un *Device profile*, el cual se utiliza para definir como la plataforma debe procesar los mensajes que provengan de ese tipo de dispositivos. Es decir que

es una manera de crear tipos para los dispositivos. Luego se debe crear una instancia de un dispositivo, al cual se le asigna el *Device profile* creado y por último se debe integrar el dispositivo con la instancia de la plataforma.

A continuación los pasos para realizar las tareas mencionadas anteriormente:

1. Iniciar sesión en la plataforma.
2. Ir a la pestaña *Device profiles* en el menú de la izquierda.
3. Hacer clic en el icono +.
4. Seleccionar la opción *Create new device profile*
5. Completar los campos *Name* y *Rule Chain*, con el nombre y la regla de procesado deseados para el tipo de dispositivo que este creando.
6. Hacer clic en el botón *Add*.
7. Ir a la pestaña *Devices* en el menú de la izquierda.
8. Hacer clic en el icono +.
9. Seleccionar la opción *Add new device*.
10. Completar el nombre deseado y seleccionar el *Device profile* deseado.
11. Hacer clic en *Add*.
12. En la tabla donde se muestran todos los dispositivos existentes seleccionar el recientemente creado.
13. En el panel donde nos muestran los detalles del dispositivo hacer clic en el botón *Copy access token* y configurar los mensajes que se envíen a la plataforma para utilizar dicho token.

Notar que el último paso depende de la tecnología a utilizar. Por ejemplo en http se debe enviar el *access token* en la URL.