**PONTIFICIA UNIVERSIDAD**

**CATÓLICA DEL PERÚ**

**FACULTAD DE CIENCIAS E INGENIERÍA**


# Estudio comparativo de plataformas IoT para el control y monitoreo remoto de un equipo mecánico para el envío y recepción datos

**Trabajo de investigación para obtener el grado académico de bachiller en ciencias con mención en Ingeniería de las Telecomunicaciones**

**AUTOR:**

Jose Anthony Garcia Macavilca

**Asesor:**

César Stuardo Lucho Romero

Lima, julio, 2022

## RESUMEN

El internet de las cosas es la tecnología que permite conectar dispositivos a internet y las plataformas IoT las que permiten a un usuario final el monitoreo y control remoto de dichos dispositivos. Debido a esto, se propone la utilización de una plataforma IoT para monitorear y controlar remotamente los equipos mecánicos de la PUCP, como el banco de psicrometría ubicado en el Laboratorio de energía de la misma, para el desarrollo de laboratorios remotos. Por este motivo, el objetivo principal será el análisis y comparación de tres plataformas IoT que permitan el monitoreo y control remoto de un equipo mecánico: ThingSpeak, ThingsBoard y Kaa IoT. En la investigación, se evidencia que la plataforma que cumple con la mayoría de los requerimientos planteados para el desarrollo de un laboratorio remoto es Kaa IoT.



Palabras clave

Plataforma IoT, laboratorios remotos, Kaa IoT

# DEDICATORIA

A mi madre, Enedina, por ser la mujer que me inspira a seguir esforzándome.

# ÍNDICE GENERAL

# ÍNDICE DE TABLAS

# ÍNDICE DE FIGURAS

# INTRODUCCIÓN

Una tecnología que está tomando relevancia en la actualidad es el IoT, pues es la que permite que diferentes dispositivos puedan estar conectados a internet. Asimismo, las plataformas IoT son las que permiten a los usuarios finales poder interactuar con dichos dispositivos y, de esta manera, poder controlarlos y monitorearlos de manera remota.

Por otra parte, debido a la pandemia, los alumnos tuvieron que pasar de controlar los equipos mecánicos dentro de la PUCP a utilizar simuladores para el desarrollo de sus laboratorios. Sin embargo, las plataformas IoT podrían permitir a un alumno volver a controlar dichos equipos sin necesidad de estar presente en la PUCP, sino monitorearlos y controlarlos de manera remota desde cualquier lugar en donde se encuentren conectados a internet.

Por tanto, el objetivo principal de esta investigación será analizar y comparar algunas de las plataformas IoT que permitan el monitoreo y control remoto de un equipo mecánico. Así, en un futuro, se pueda utilizar una plataforma IoT para el desarrollo de un laboratorio de manera remota.

En el primer capítulo se desarrolla la problemática y justificación de la investigación, al igual que se definen los objetivos de esta. En el segundo se desarrolla una arquitectura simplificada de las plataformas IoT, los protocolos de comunicación IoT más utilizados y algunos puntos

importantes en la toma de decisión al escoger una plataforma IoT. Asimismo, se escogen las plataformas IoT a ser analizadas y comparadas. En el tercer capítulo, se realiza una comparación de las plataformas escogidas previamente basadas en los objetivos de la investigación. En el cuarto capítulo se desarrollan las recomendaciones de aplicabilidad de la presente investigación. Finalmente, se desarrollan las conclusiones y se muestran las referencias.

La presente investigación logra obtener, de las plataformas analizadas, a la más adecuada a ser utilizada para el desarrollo de un laboratorio remoto.

# Capítulo 1. Presentación del tema de investigación

## 1.1 Área de especialización en las telecomunicaciones

La presente investigación está enfocada en las áreas de sistemas flexibles y software para telecomunicaciones, e Internet de las cosas (IoT) dentro de la carrera de Ingeniería de las telecomunicaciones. Los conocimientos adquiridos de la primera área de especialización fueron enseñados en los cursos de Software para telecomunicaciones 1, actualmente llamado Ingeniería web para telecomunicaciones, y Gestión de servicios de TICS. En dichos cursos se realizaron proyectos de implementación de servicios Tics, lo cual permitieron adquirir conocimientos en el planteamiento y desarrollo de proyectos involucrados con las Tics. Asimismo, para la segunda área, los conocimientos adquiridos fueron enseñados en el curso electivo de Temas avanzados en servicios y aplicaciones. Los cursos antes mencionados ayudaron al desarrollo de la competencia de desarrollo e implementación de servicios y aplicaciones de telecomunicaciones y sistemas de distribución de contenidos sobre Internet, web móvil y video bajo demanda e Internet de las Cosas (IoT) que cumplan con criterios de seguridad, confiabilidad y bajo retardo.

## 1.2 Contextualización e identificación del problema

### 1.2.1 La aparición de COVID-19 y contexto actual

La aparición de COVID-19 paralizó al mundo entero. Negocios, empresas, colegios, universidades, entre otras entidades, tuvieron que parar en la realización de sus actividades debido a que el contexto obligaba al aislamiento total de sus colaboradores. Así, la Pontificia Universidad Católica del Perú tuvo que migrar del método de enseñanza tradicional, el presencial, al dictado de clases de forma no presencial, de manera remota. Enfocándonos en la enseñanza de las ingenierías, las cuales son carreras que demandan un contacto continuo con las experiencias prácticas, en un entorno no presencial, la utilización de simuladores toma un rol muy importante, ya que, en el caso que se trabaje con un equipo, estos buscan imitar el comportamiento que dichos equipos tendría en un entorno real. Así, alumnos de diferentes especialidades, pasaron de controlar un equipo físico dentro de la universidad a utilizar simuladores [1][2]. En la actualidad, la PUCP ha tomado el modelo de enseñanza semipresencialidad, por lo que se ha logrado retornar a los ambientes en donde es posible la interacción con los equipos. Sin embargo, no todos pueden realizar dichas experiencias debido a que no cumplen con todos los protocolos de bioseguridad necesarios para ingresar a la universidad, se encuentran fuera de Lima, etc, y continúan realizando las experiencias de manera remota utilizando los simuladores [3].

### 1.2.2 Internet de las cosas

El internet de las cosas o IoT, por sus siglas en inglés *Internet of things*, se refiere a la red colectiva de dispositivos conectados y a la tecnología que facilita la comunicación entre dichos dispositivos y la nube, o la comunicación entre esos mismos dispositivos. Gracias al desarrollo de *chips* de computadora de bajo costo y a las telecomunicaciones de alta velocidad,

actualmente se puede tener billones de dispositivos conectados a internet [4]. Así, es posible que los dispositivos puedan informar sus estados constantemente y que un usuario final pueda controlarlos de manera remota a través de una interfaz web o un aplicativo móvil, por ejemplo, considerando el estado actual de los dispositivos.

### 1.2.3 Banco de psicrometría

El banco de psicrometría es un módulo educativo que simula el proceso de acondicionamiento de aire y permite la compresión y análisis de los procesos psicrométricos por medio de experimentos [5]. La Pontificia Universidad Católica del Perú cuenta con un banco de Psicometría ubicado el Laboratorio de energía el cual es utilizado por aproximadamente 60 alumnos en el desarrollo de sus laboratorios, la parte práctica de sus estudios, en los cursos MEC245 (Laboratorio de Termodinámica General), MEC286 (Transferencia de calor) y MEC208 (Termodinámica 2). Sin embargo, por las características mecánicas de este equipo los alumnos deben estar presentes para poder interactuar con el mismo.

*Figura 1.1. Banco de psicrometría de la PUCP*
*Fuente: fotografía propia*

### 1.2.4 Problema identificado

Así, una vez explicado el contexto actual, el IoT y la situación actual dentro de la universidad se puede notar que es necesario que el banco de psicrometría pueda ser controlado de manera

remota por los alumnos que, por diferentes motivos, no pueden estar presente dentro de la universidad.

## 1.3 Objetivos, alcance y justificación

Para fines de la presente investigación, el banco de psicrometría mencionado en el punto 1.2.3 será modelado como una caja negra que constantemente está enviando y recibiendo *data*.

Así, el objetivo general de la presente investigación será el estudio y análisis de las plataformas IoT más adecuadas para controlar equipos mecánicos con cierta cantidad de canales en los cuales constantemente se está enviando y recibiendo *data*.

Específicamente, se buscará analizar si las plataformas cuentan con herramientas de visualización que permitan observar la data de diferentes maneras, si se permite el monitoreo y control remoto de un equipo mecánico, y si permiten una gestión usuarios a nivel de roles (como pueden ser alumno, jefe de práctica y administrador). De esta manera, se pueda lograr generar el ambiente del desarrollo de un laboratorio presencial con una fidelidad aceptable.

Con todo lo definido anteriormente se planea en dar respuesta a la siguiente pregunta:

- ¿Cuál de las plataformas IoT existentes es la más adecuada para controlar de forma remota un dispositivo mecánico con cierta cantidad de canales de transmisión y recepción de *data*?

Asimismo, el desarrollo de la investigación es importante debido a que permite una transformación de la enseñanza universitaria, ya que no solo permitiría el control de un equipo mecánico por parte de los alumnos, sino que permitiría una nueva manera de enseñanza en donde, por ejemplo, un profesor sería capaz de controlar un equipo remotamente desde un salón de clases y visualizar la data generada para su evaluación en tiempo real sin necesidad de dirigirse con todos sus alumnos hacia el lugar físico donde se encuentra dicho equipo, ya que el control total podría realizarse desde la nube a través de internet.

## 1.4 Tipo y enfoque de investigación

Para la presente investigación se utilizarán los tipos de investigación documental y de campo. Por tanto, se realizará un análisis exhaustivo de fuentes bibliográficas que permitan lograr el objetivo planteado en el punto 1.3. Asimismo, se realizará una visita de campo que consistirá en asistir al desarrollo de una experiencia práctica dentro del Laboratorio de energía de la PUCP para presenciar cómo se desarrollan actualmente dichas experiencias con el equipo antes mencionado. Así, se podrá obtener información acerca de la interacción, data generada, mediciones realizadas y las mejoras que se puede realizar a las experiencias actuales a través de una plataforma IoT.

# Capítulo 2. Investigación bibliográfica y visita de campo

El presente capítulo tiene como finalidad abordar la arquitectura de una plataforma IoT, ya que es fundamental su conocimiento para el desarrollo del tema de investigación; y se desarrollará los protocolos de comunicación más utilizados por las plataformas IoT. También, se describen algunos puntos claves al escoger una plataforma IoT. Asimismo, se presentan las plataformas a ser analizadas en la investigación. Por último, se describe la experiencia de un laboratorio presencial actual y se detectan condiciones adicionales a ser consideradas en el análisis propuesto.

## 2.1 Plataforma IoT

Sabemos que gracias a la tecnología IoT los dispositivos pueden estar conectados a internet, pero ¿cómo un usuario final podría interactuar con dichos dispositivos? De acuerdo a Mineraud [6], una plataforma IoT está definida como el *midleware* y la infraestructura que permite a los usuarios finales interactuar con los "objetos inteligentes", es decir a los dispositivos capaces de conectarse a la plataforma IoT. Asimismo, según Hejazi [7], las plataformas IoT son las encargadas de conectar a los elementos de un entorno IoT (los dispositivos, *gateways* y las

redes) con servicios y aplicaciones en la nube. Estos elementos pueden o no estar separados por grandes distancias, pero están controladas por una entidad centralizada de control, el cual es la unidad de procesamiento de una plataforma IoT. De ambas definiciones, se puede decir que una plataforma IoT permite la interacción de dispositivos y usuarios finales a través de aplicaciones en la nube. A continuación, se describe la arquitectura simplificada de una plataforma IoT, la cual consta de 4 bloques, los cuales son los siguientes: la capa física, *el IoT middleware*, capa de aplicación y una capa transversal de seguridad [8].

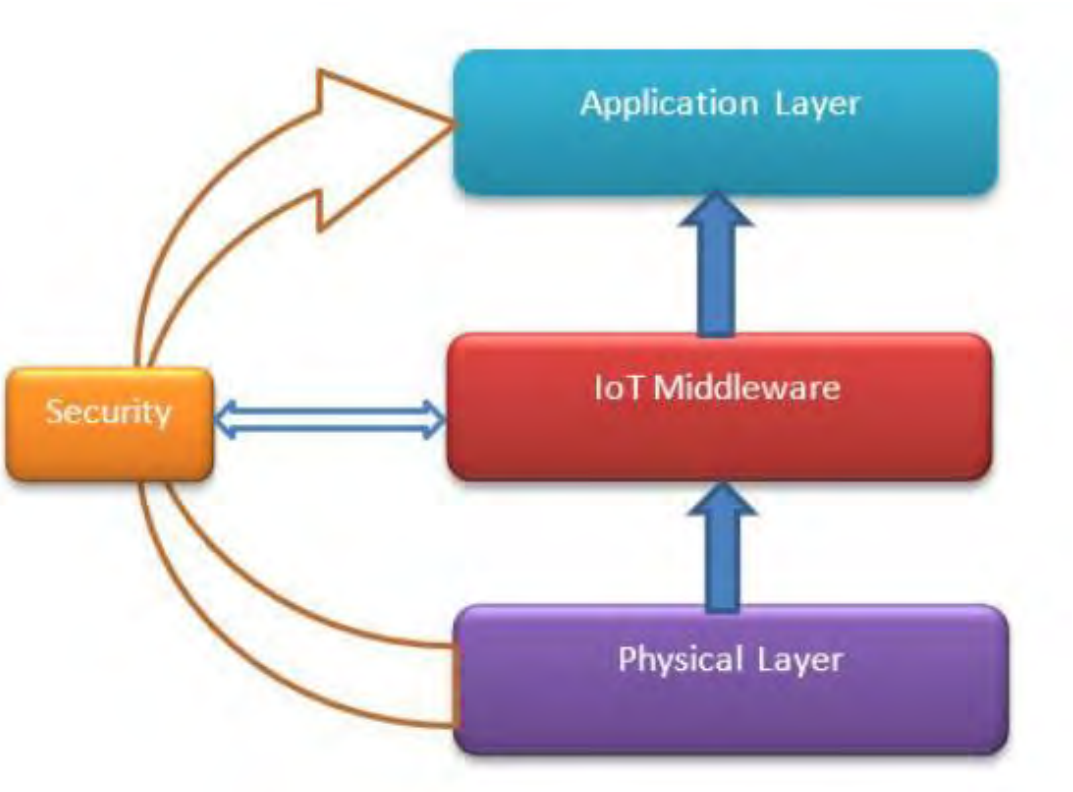


*Figura 2.1. Arquitectura simplificada de una plataforma IoT*
*Fuente:*[8]

### 2.1.1 Capa física

La capa física está conformada por el conjunto de dispositivos que adquieren data a través de sensores y que se comunican entre ellos a través de diferentes protocolos como por ejemplo Wifi, Bluetooth, Zigbee, Z-wave, LoRaWAN, etc [8]. Los dispositivos IoT pueden intercambiar *data* con otros dispositivos a los cuales se encuentran conectados y una aplicación, o también

recolectar *data* de otros dispositivos y procesarla localmente, o enviar dicha *data* a un servidor centralizado o a una aplicación *back-end* en la nube para su procesamiento [7]. Cabe mencionar que el uso de sensores en esta capa es muy importante. Un sensor es un componente de *hardware* usado para medir los parámetros del entorno físico del dispositivo y convertirlas en señales eléctricas [9]. Por ejemplo, se puede medir constantemente la temperatura y humedad de una habitación para que, posteriormente, se realice el procesamiento de dichos valores y se active el aire acondicionado cuando la temperatura baje de un cierto límite o la humedad supere cierto valor determinado. Asimismo, en esta capa se encuentran los *IoT gateways*. Los *IoT gateways* permiten que un dispositivo que utiliza protocolos de comunicación que no soporten el protocolo IP (como por ejemplo Zigbee o Z-wave) puedan comunicarse con la siguiente capa (el *IoT middleware*) [8]. Esto es posible debido a que el *IoT gateway* provee las funcionalidades y tecnología requeridas para "traducir" diferentes protocolos y reenviar la comunicación entre los dispositivos y otros sistemas [9].

### 2.1.2 IoT Middleware

La capa *IoT Middleware*, también conocida como el *software back-end*, es el componente principal de una plataforma IoT [8]. Este componente es el encargado de recibir la *data* de los dispositivos conectados para procesarla y enviarla, por ejemplo, a una aplicación. De igual manera, puede procesar dicha *data* y, a través de la evaluación de reglas condición-acción, controlar dispositivos enviándoles comandos a ser ejecutados por sus respectivos actuadores (componentes de *hardware* que controlan o manipulan el entorno físico, pues convierten las señales eléctricas en acciones físicas que afectan a los dispositivos). Asimismo, un dispositivo puede conectarse a esta capa directamente si se cumple tres condiciones, las cuales son las siguientes:

- soportar una apropiada tecnología de comunicación (como el WiFi)

- utilizar un protocolo de transporte apropiado a la tecnología de comunicación utilizada (como HTTP o MQTT)
- usar un formato compatible de *payload* (como el JSON o XML)

Si las condiciones no se cumplen, y como se mencionó en el punto anterior, es necesario un *Gateway* para lograr la comunicación del dispositivo con el *IoT Middleware* [9].

### 2.1.3 Capa de aplicación

La capa de aplicación es la que permite al usuario final interactuar con la plataforma IoT. Esta capa es la que permite la visualización de data, control de dispositivos, etc [7][8]. Los servidores web presentan una gran relevancia en esta capa, ya que la gran mayoría de plataformas IoT tienen algún tipo de interfaz gráfica web, para que el usuario final interactúe con él, y le permita el control remoto de los dispositivos conectados al ambiente IoT a través de internet [7].

### 2.1.4 Capa de seguridad

El crecimiento exponencial que tiene el IoT significa también un incremento de los riesgos en la seguridad y privacidad [10]. Por ello, la capa de seguridad es un componente esencial que debe ser soportado por cada capa abordada anteriormente. Sin embargo, debido a que el costo de su despliegue en cada capa es costoso, las distintas compañías o proveedores de servicio en la nube solo brindan seguridad a algunas capas, lo que genera que la *data* de los dispositivos se encuentre expuesta a ataques como *Denial of Service* (DoS), *Man in the Middle* (MIM), etc [8].

## 2.2 Protocolos de la capa de aplicación IoT

En un entorno IoT, se encuentran interconectados una gran cantidad de dispositivos con sensores que generan cada segundo una alta cantidad de *data*. Así, la *data* generada resultante enviada entre dispositivos es inmensa [11]. Por este motivo, el análisis de cómo y qué tipo de protocolos son los más utilizados en un entorno IoT para la transmisión de tal cantidad de *data*

es muy importante. Existe una variedad grande de protocolos estandarizados para la capa de aplicación IoT, por lo que se revisarán los dos protocolos más populares: CoAP y MQTT [12].

### 2.2.1 Protocolo CoAP

El protocolo CoAP (por sus siglas en inglés *Constrained Application Protocol*) es un protocolo de transferencia web especializado para su uso en nodos y redes con "limitaciones", como puede ser alta cantidad de pérdidas, ancho de banda limitado o niveles de potencia de transmisión bajas. CoAP provee un modelo de interacción de *request*/*response* entre *endpoints* [13]. Es decir, se basa en un modelo cliente/servidor similar a HTTP donde el cliente realiza una solicitud al servidor y este le envía una respuesta. Asimismo, los clientes pueden realizar solicitudes utilizando métodos *GET*, *POST*, *PUT* y *DELETE*.

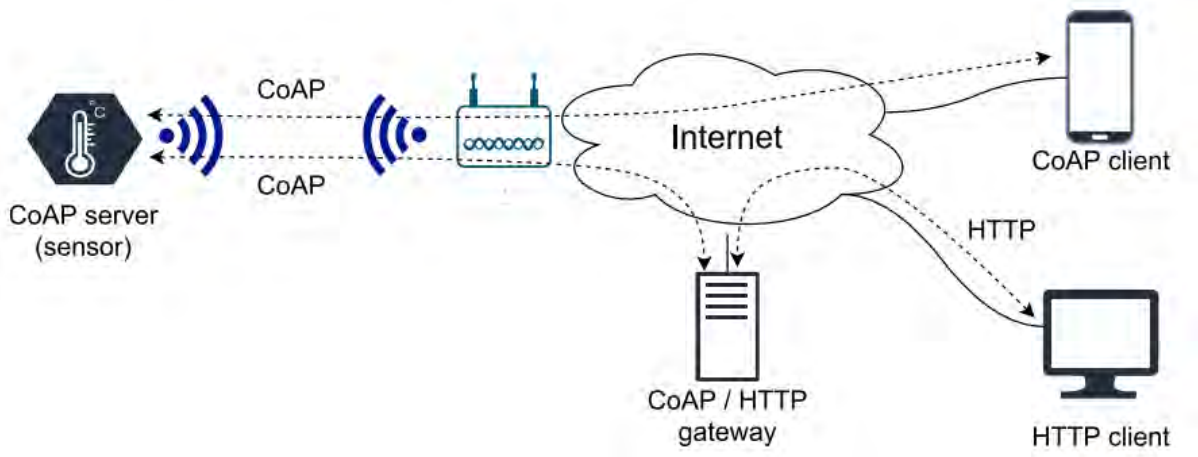


*Figura 2.2. Funcionamiento general de un sistema con CoAP*
*Fuente:* [12]

Con respecto a la confiabilidad del envío de mensajes, los *requets* o *responses* pueden estar marcados como "*confirmable*" o "*nonconfirmable*". Los primeros necesitan de un mensaje de *acknowledgement* del receptor para confirmar la recepción del mensaje, mientras que los segundos no necesitan de una respuesta por parte del receptor [14].

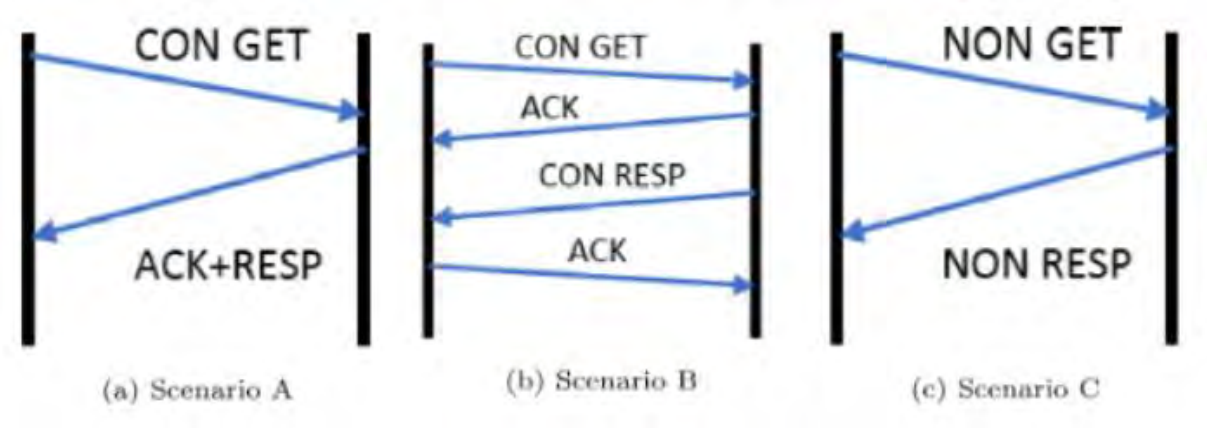


*Figura 2.3. Posibles escenarios de un mensaje GET CoAP*
*Fuente:* [12]

El protocolo CoAP está basado en el protocolo UDP (*User datagram protocol*) lo que le permite lograr el envío de información en situaciones, como en una red con mucha congestión, en donde otros protocolos basados en TCP fallarían en completar *handshake* [15].

### 2.2.2 Protocolo MQTT

MQTT es un protocolo de transporte de mensajería basado en una arquitectura de suscripción y publicación (*publish*/*subscribe*). Es un protocolo ligero, abierto, simple y diseñado para una implementación sencilla. Debido a estas características, se hace ideal su uso para comunicaciones *Machine-to-Machine* (M2M) e Internet de las cosas en donde el uso del ancho de banda requerido es importante [16]. En un entorno donde MQTT es utilizado, todos los sensores deben estar conectados a un servidor centralizado llamado *bróker*. Asimismo, los mensajes enviados y/o recibidos deben estar "etiquetados" con cierto *topic*. Al envío de mensaje se le llama "publicación" de mensaje y para que un dispositivo reciba cierto mensaje este debe estar "suscrito" al *topic* al cual el mensaje a recibir se encuentra etiquetado, de ahí que se conoce como una arquitectura basada en suscripción y publicación. Cada cliente puede suscribirse a múltiples *topics* y todos los clientes que se suscriben a cierto tópico reciben un mensaje cuando este es publicado con ese *topic* determinado [14].

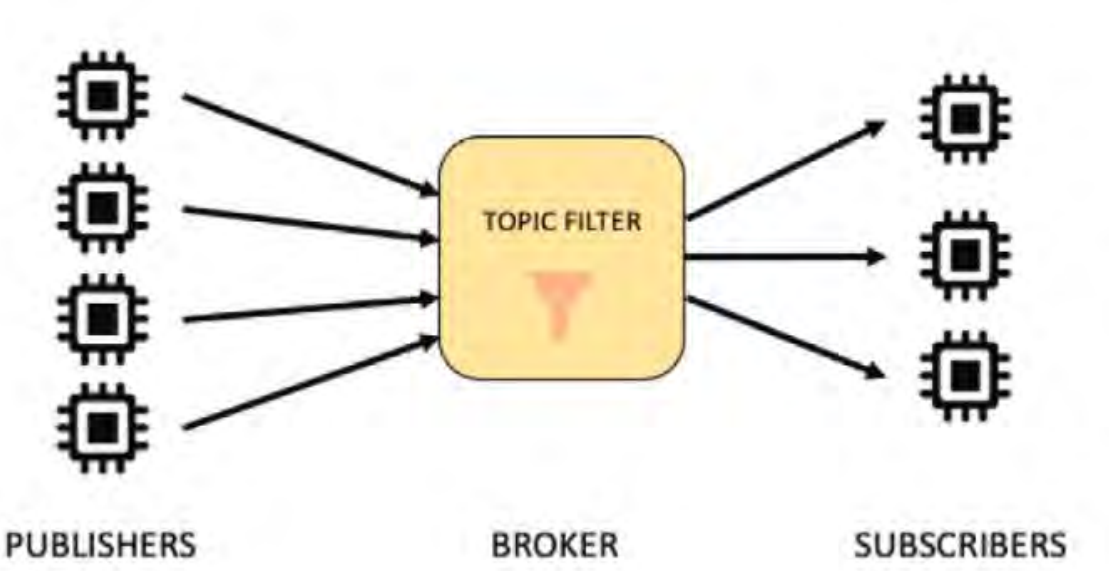


*Figura 2.4. Arquitectura de intercambio de mensajes MQTT*
*Fuente:*[12]

Por ejemplo, imaginemos que se tiene un sensor de temperatura en una habitación, un celular y un servidor *back-end*, que puede ser un servidor web, conectados a un broker MQTT (como se puede observar en la Figura 2.5). El celular y el servidor se encuentran "suscritos" al *topic* "temperatura". Así, el sensor al realizar una medición de temperatura en la habitación, lo "publica" con el *topic* "temperatura" hacia un *broker* MQTT. Cuando el *broker* recibe el mensaje publicado, este lo reenvía a todos los dispositivos que se encuentren suscritos al *topic* "temperatura", en este caso el celular y el servidor *back-end*.

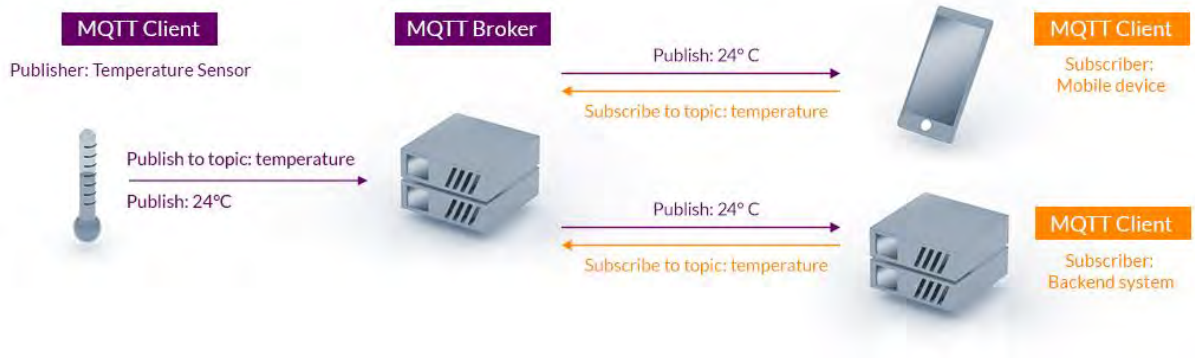


*Figura 2.5. Ejemplo de publicación y suscripción*
*Fuente:* [17]

El protocolo MQTT ofrece 3 niveles de QoS: QoS 0, QoS 1 y QoS 2 [16].

**a. QoS 0 (a lo mucho se entrega uno)**

El mensaje es enviado de acuerdo a las capacidades de la red. No se realizan retransmisiones o se recibe un mensaje de confirmación del receptor. Así, el mensaje enviado es entregado a lo mucho una vez [12].

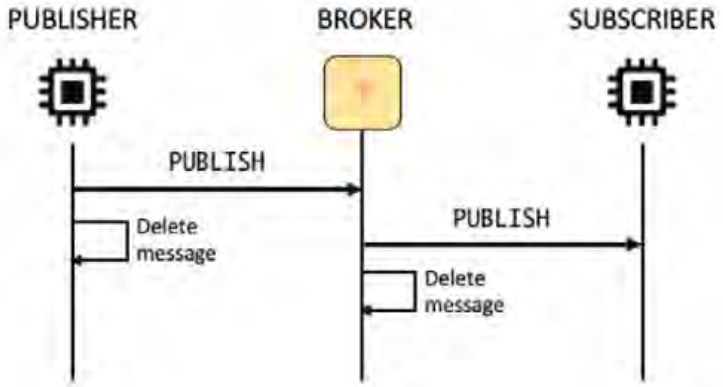


*Figura 2.6. QoS de nivel 0*
*Fuente:*[12]

**b. QoS 1 (al menos se entrega uno)**

El receptor debe responder como un mensaje de confirmación. Caso contrario, el mensaje es retransmitido. El mensaje puede enviarse varias veces, debido a las retransmisiones, pero se entrega al menos una vez [12].

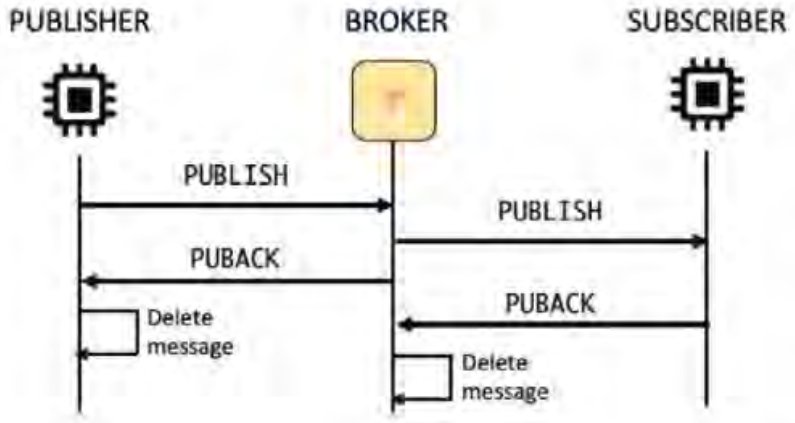


*Figura 2.7. QoS de nivel 1*
*Fuente:* [12]

**c. QoS 2 (exactamente uno entregado)**

Se realiza un *four-way handshake* que asegura que el mensaje es entregado exactamente una vez [12].

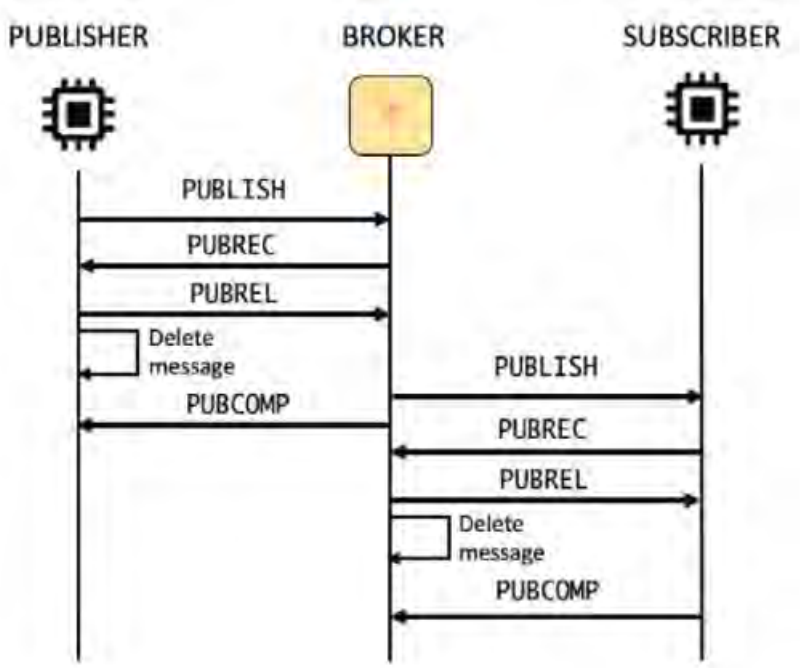


*Figura 2.8. QoS de nivel 2*
*Fuente:* [12]

Notamos que a diferencia del protocolo CoAP, MQTT tiene más niveles de QoS, lo que le permite tener transmisiones más confiables. Sin embargo, altos niveles de QoS generan latencia, debido al *handshake* para QoS 2, por ejemplo, y requerimiento de más ancho de banda, lo que resulta en más consumo energético [15].

### 2.2.3 CoAP vs MQTT

Como se mencionó anteriormente, ambos protocolos son muy populares en entornos IoT, pero ¿cuándo utilizar CoAP y cuándo utilizar MQTT? El utilizar CoAP o MQTT depende de qué tipo de aplicación de IoT se esté realizando. Para casos en donde los dispositivos necesiten comunicación entre ellos a través de una red del tipo WAN, como internet, MQTT es mejor debido a su arquitectura de "publicación/suscripción" con un "bróker" en medio de la

comunicación, pues permite una sencilla comunicación. Por otro lado, el hecho de que CoAP sea compatible con HTTP le permite ser la mejor opción cuando es un entorno basado en servicios web. Puede ser utilizado en caso se tenga poco ancho de banda y en redes locales, debido a que utiliza UDP, el cual soporta *multicast* y *broadcast* [7].

## 2.3 Factores importantes al escoger una plataforma IoT

Al 2021 se reportaron más de 600 plataformas IoT en el mercado [18]. Esta gran cantidad de soluciones dificulta la elección de una en particular para una empresa o para algún proyecto en específico. Sin embargo, se puede considerar algunos de los puntos más importantes a tener en cuenta al realizar la elección de una plataforma IoT apropiada [19].

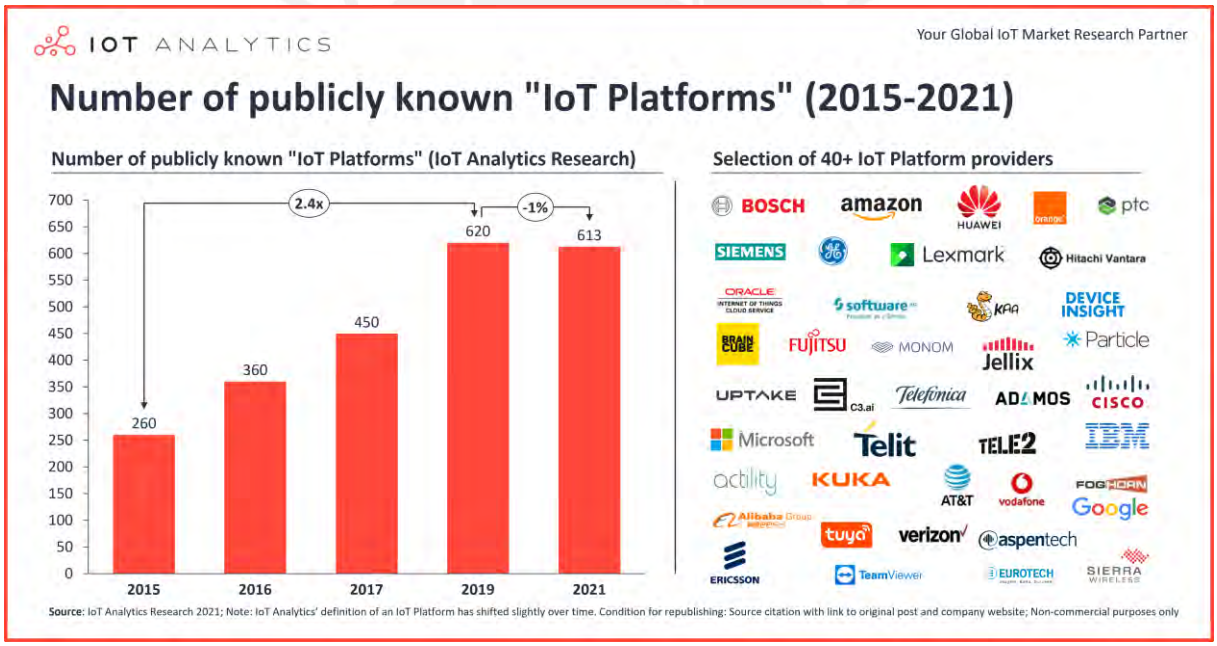


*Figura 2.9. Cantidad de plataformas IoT en el mercado al 2021*
*Fuente:* [18]

### 2.3.1 Estabilidad de la plataforma

En el mercado existe una gran cantidad de plataformas IoT y es probable que no todas las soluciones existentes estén libres de tener fallas. Por tal motivo, es muy importante que la probabilidad de que una plataforma se encuentre operativa sin interrupción por años, o por el

tiempo que se requiera, sea alta. Una buena práctica utilizada por empresas consumidoras de plataformas IoT es realizar consultas a pasados clientes de las plataformas a implementarse [7].

### 2.3.2 Escalabilidad

El concepto de escalabilidad connota la capacidad de un sistema para controlar un número cada vez mayor de elementos u objetos, procesar volúmenes crecientes de trabajo sin dificultad y/o tener potencial de ser ampliado para soportar dicho incremento de volumen de trabajo [20]. Así, una plataforma IoT debe ser escalable debido a la creciente cantidad de dispositivos que podrían estar conectados a este.

### 2.3.3 Seguridad

El aspecto de la seguridad es uno de los puntos más importantes de una plataforma IoT debido a que están comprometidos diferentes tipos y cantidades de dispositivos que podrían verse expuestos si no se tiene una buena capa de seguridad. Verificar los *feautures* de seguridad de una plataforma IoT debe ser una obligación. Por ejemplo, se debe brindar seguridad en la capa de transporte (TLS, por sus siglas en inglés *Transport Layer Security*) para comunicarse con los dispositivos y aplicaciones, o la autenticación de dispositivos y usuarios [21].

### 2.3.4 Análisis de datos y herramientas de visualización

Como se mencionó anteriormente, la cantidad de *data* en un entorno IoT es enorme. Por tal motivo, un factor muy importante de una plataforma IoT son las capacidades de agregar, analizar y visualizar *data* [19]. El tener el valor actual de un dispositivo específico, la tendencia de los valores generados de este o si dichos valores tienden a ser estacionarios brindan una perspectiva profunda del comportamiento de dicho dispositivo y permite tomar acciones correctas. Actualmente, existen varias soluciones especializadas en la visualización de *data*. Grafana es probablemente la más popular [22]. Grafana es un *framework* de visualización y

monitoreo de código abierto el cual puede estar conectado a múltiples fuentes de *data* y proveer visualizaciones atractivas. Las fuentes de *data* pueden ser bases de datos conocidas como MySQL, Oracle Database, PostgresSQL, etc [23].


*Figura 2.10. Ejemplo de dashboard generado con Grafana*
*Fuente:*[23]

### 2.3.5 Protocolos soportados

Como se mencionó anteriormente, existe una gran variedad de protocolos soportados por las plataformas IoT y dependiendo del uso que se pretenda dar a la plataforma, el protocolo a ser utilizado puede variar. Por lo tanto, la plataforma a ser escogida debe soportar el protocolo que se pretenda usar.

### 2.3.6 Performance de sistema

En una plataforma IoT, cuando un evento ocurre, una regla basada en eventos puede ser invocada. Debido a que una plataforma IoT controla una gran cantidad de dispositivos conectados, el performance es muy importante, ya que a medida que crece la cantidad de

dispositivos conectados, el tiempo promedio en analizar cada evento se incrementa y cada vez se necesita un mejor consumo de recursos [24].

## 2.4 Plataformas IoT existentes

Como se mencionó anteriormente, la alta cantidad de plataformas IoT existentes dificulta la toma de decisión al momento de seleccionar una plataforma IoT a ser utilizada. También, se mencionó alguno de los puntos a tener en consideración al momento de seleccionar una plataforma en particular. De dichos puntos, la presente investigación se enfocará en el protocolo soportado por la plataforma IoT y las herramientas de visualización de esta, por lo que estos puntos son el primer filtro para seleccionar las plataformas a ser analizadas y comparadas. Esto es debido a que la plataforma será utilizada por alumnos conectados a internet - por lo que se preferirán aquellas plataformas que soporten el protocolo MQTT, por lo discutido en 2.2.3. Asimismo, las herramientas de visualización son un punto importante, ya que el contar con herramientas de visualización que brinden el gráfico de la data de diferentes maneras permitirán al alumno un mejor análisis de esta. Como segundo filtro será la bibliografía revisada, ya que se escogerá las plataformas que más veces sean mencionadas por los autores revisados [25][8][24][26]. Así, las plataformas seleccionadas son ThingSpeak, ThingsBoard y Kaa, ya que cumplen tanto el primer y segundo filtro.

### 2.4.1 ThingsBoard

Thingsboard es una Plataforma IoT de código abierto que permite un rápido desarrollo, administración y escalamiento de proyectos IoT [27].

#### 2.4.1.1 Servicios de ThingsBoard

ThingsBoard fue diseñado para ser escalable horizontalmente, ser tolerante a fallas (puede ser implementado en un clúster de nodos donde cada nodo es idéntico, entonces no existe el

problema de “punto único de fallo”), ser robusto y ser eficiente (un solo nodo servidor puede manejar decenas de miles de dispositivos), y personalizable [28]. Entre los principales componentes e interfaces que estos servicios proporcionan se muestran en la Figura 2.11.

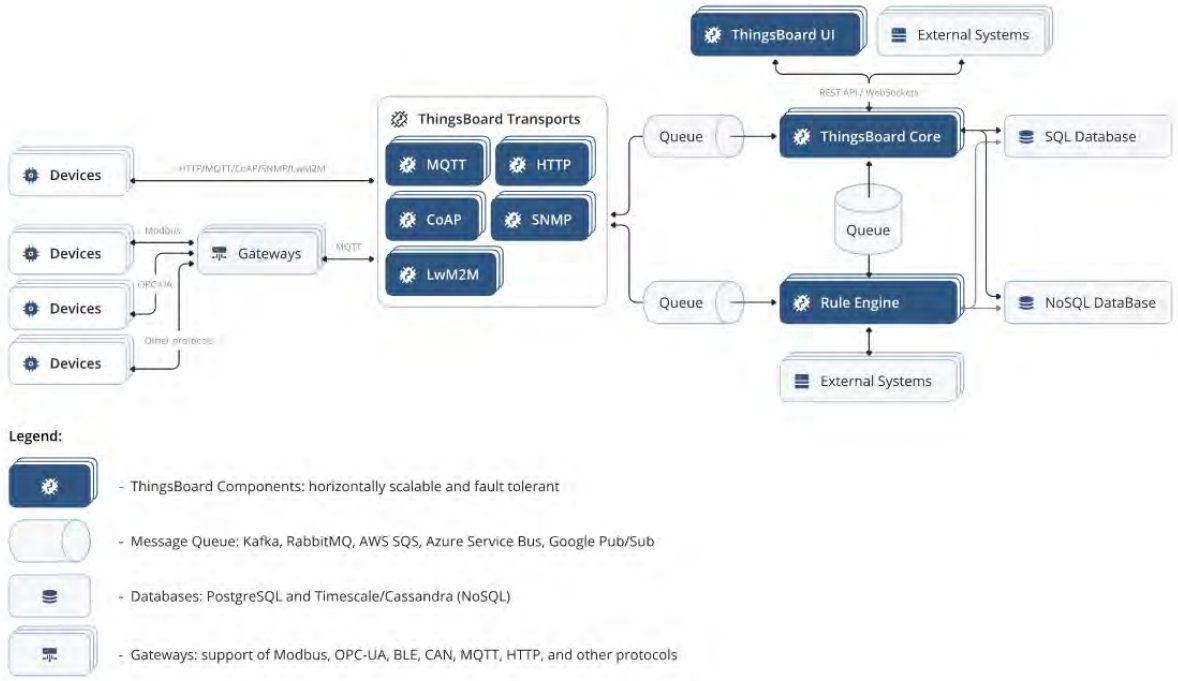


*Figura 2.11. Principales componentes e interfaces de los servicios de ThingsBoard*
*Fuente:*[28]

- **ThingsBoard Core:** Es el responsable de manejar las llamadas a las REST API y las suscripciones de WebSocket. Además, es el encargado de almacenar la información actualizada de las sesiones de los dispositivos activos y el monitoreo de estado de conectividad de los dispositivos.
- **ThingsBoard Rule Engine:** Es el corazón del sistema, ya que es el encargado del procesamiento de los mensajes entrantes.
- **ThingsBoard Web UI:** ThingsBoard provee un componente ligero escrito utilizando el *framework* Express.js que contiene la interfaz de usuario estática.

### 2.4.2 ThingSpeak

ThingSpeak es un servicio de plataforma de análisis IoT que permite agregar, visualizar y analizar *data* en tiempo real en la nube. Esta plataforma permite la visualización de la data enviada por los dispositivos hacia ThingSpeak y enviar alertas, bajo ciertas condiciones que el usuario ingrese, hacia plataformas web como Twitter. Una particularidad de esta plataforma es que es un producto de MathWorks, el cual también ofrece MATLAB, por lo que permite correr código MATLAB para realizar el preprocesamiento, visualización y análisis de la data [29]. MATLAB es una plataforma de programación diseñada por ingenieros y científicos el cual utiliza el lenguaje con el mismo nombre: MATLAB. Dicho lenguaje es un lenguaje basado en matrices, el cual es muy utilizado para operaciones matemáticas computacionales [30]. Algunos de los *features* de ThingSpeak son los siguientes:

- Se pueden utilizar protocolos como MQTT o HTTP para el envió de *data* de los dispositivos de manera sencilla.
- Visualización de la data adquirida en tiempo real.
- Mejorar el análisis de la data a través de la utilización del lenguaje MATLAB.
- Ejecutar el análisis IoT automáticamente basado en *schedules* o eventos.

#### 2.4.2.1 Canales en ThingSpeak

La plataforma ThingSpeak maneja el concepto de “canales”. ThingSpeak utiliza dichos canales para almacenar la data enviada por los dispositivos, los cuales realizan el envío a través del REST API o el MQTT API de la misma plataforma. Además, como ThingSpeak puede ejecutar código MATLAB, se puede enviar data a través de dicho lenguaje utilizando la función thingSpeakWrite() [29].

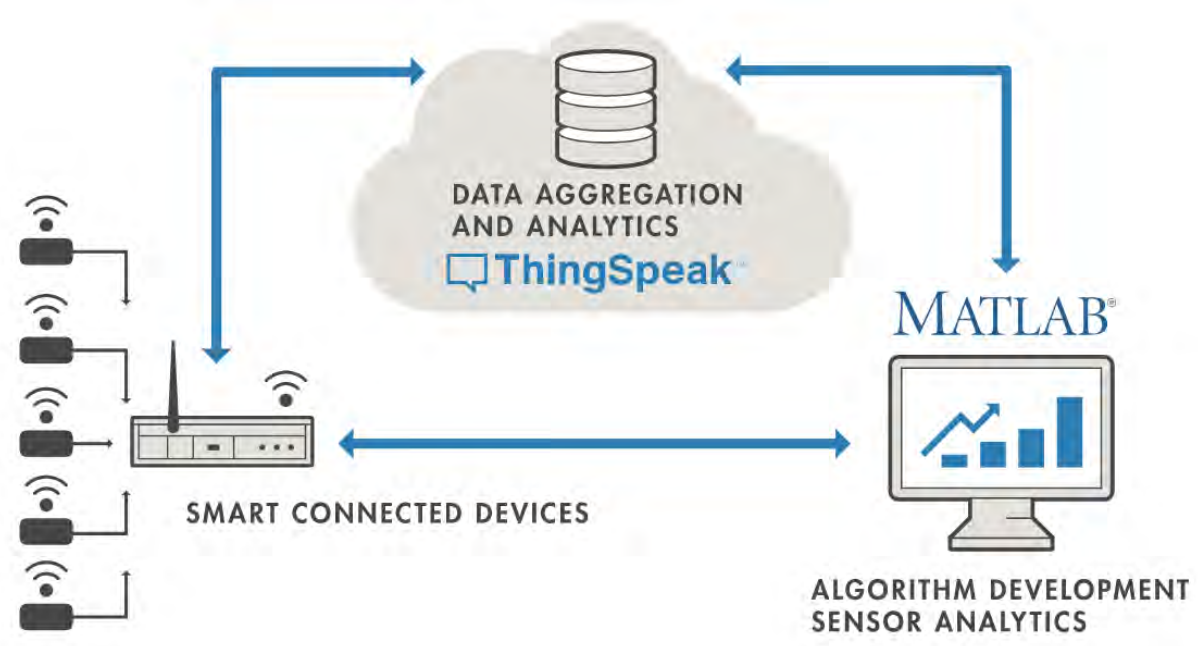


*Figura 2.12. Sistema IoT según ThingSpeak*
*Fuente:*[31]

En la Figura 2.12 se muestra un sistema IoT a alto nivel en donde se puede observar cómo interactúa ThingSpeak con los dispositivos que envían la data. A la izquierda se muestra los dispositivos que se conectan a la plataforma a través de un *Gateway IoT*, en el caso de que no puedan conectarse directamente a la plataforma por medio de MQTT, por ejemplo, (como se explicó en el punto 2.1.2) y en el medio se encuentra la plataforma ThingSpeak encargada de recibir la data y analizarla. Además, a la derecha se muestra el caso en donde la data puede ser obtenida de la plataforma hacia una computadora con MATLAB para permitir a un ingeniero o científico la elaboración de algoritmos que se podrían ejecutar eventualmente en la nube o en los mismos dispositivos [31].

### 2.4.3 Kaa

Kaa es una Plataforma IoT *end-to-end* nativa en la nube aplicable a cualquier escala de proyectos IoT empresariales. Esta plataforma tiene un gran rango de *features* que permiten a los desarrolladores implementar aplicaciones para productos inteligentes [32].

### 2.4.3.1 Arquitectura: basada en microservicios

La arquitectura de Kaa está basada en el enfoque de microservicios en su totalidad. Para conseguir que la plataforma esté basada en microservicios, Kaa utiliza 3 técnicas. En primer lugar, los protocolos de comunicación HTTP y NATS son usados para transportar mensajes, y JSON y Avro para codificar dichos mensajes entre todos los microservicios. En segundo lugar, los microservicios están distribuidos como imágenes Docker. Por último, Kaa utiliza sistemas de orquestación de contenedores, de los cuales Kubernetes, es el preferido por Kaa [33].

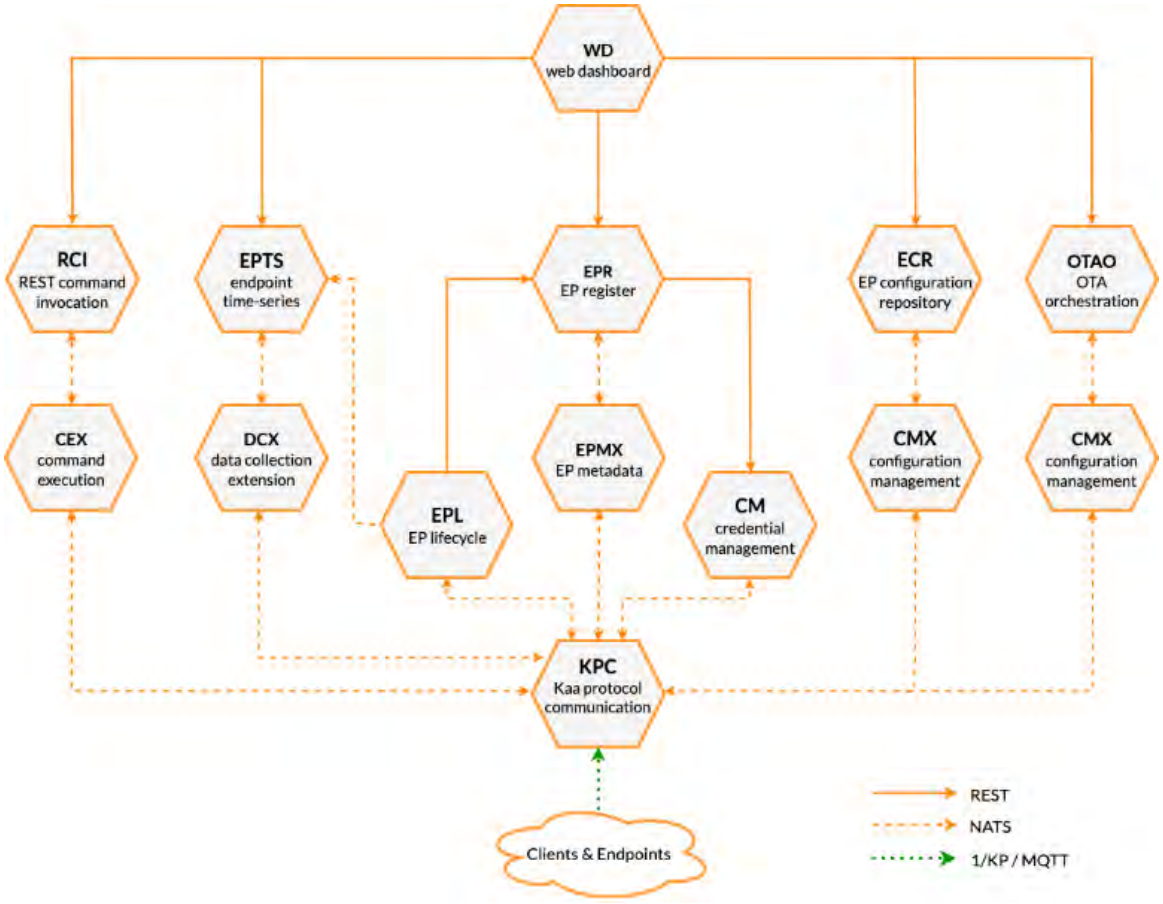


*Figura 2.13. Arquitectura basada en microservicios de Kaa*
*Fuente:* [33]

### 2.4.3.2 Conceptos utilizados en Kaa

A continuación, se explicarán algunas entidades y conceptos que utiliza Kaa.

- **Endpoint**: El *endpoint* hace referencia a cualquier dispositivo que se desea administrar con la plataforma Kaa. Toda la data entrante en Kaa está asociada con *endpoints* [34].

- **Client**: Para permitir que los dispositivos (los *endpoints*) puedan enviar *data* hacia Kaa, es necesario un "cliente". Un cliente Kaa puede ser cualquier aplicación en *software* que soporte un protocolo IoT para intercambiar *data* con el servidor Kaa. Dicho protocolo puede ser MQTT, CoAP, etc [34].

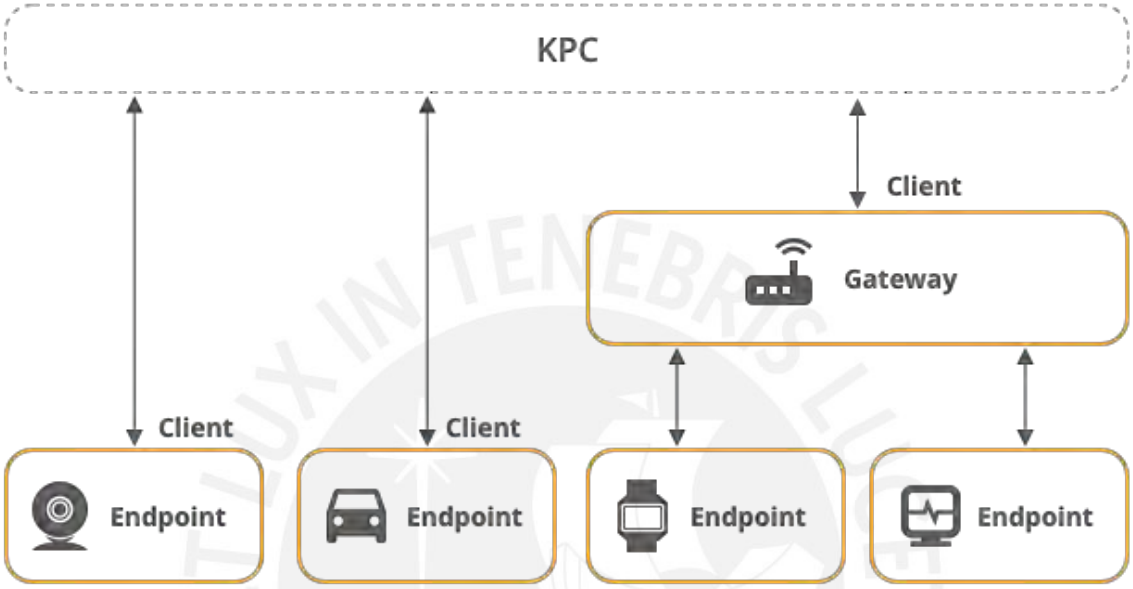


*Figura 2.14. Endpoint y Clients en Kaa*
*Fuente:*[34]

- **Aplicaciones y versiones de aplicaciones**: Kaa utiliza el concepto de "aplicación" para permitir la coexistencia de diferentes tipos de dispositivos simultáneamente en una misma solución. Una aplicación puede verse como un contenedor donde se coloca la configuración del sistema que depende del tipo de dispositivo. Por ejemplo, en una casa inteligente que tenga refrigeradoras y cafeteras, se pueden crear dos aplicaciones, una para las refrigeradoras y otra para las cafeteras. Así, todas las refrigeradoras son representadas por un conjunto de *endpoints* en una aplicación y las cafeteras pertenecen a otra aplicación. Otro concepto manejado por Kaa son las versiones de aplicaciones. Un dispositivo puede evolucionar constantemente, se le agrega o se le quitan *features*. En Kaa, esto se representa con versiones dentro de una aplicación. Se puede tener

múltiples versiones dentro de una aplicación, lo que permite que se puedan agregar nuevas funcionalidades a los *endpoints* sin afectar a los ya existentes [34].

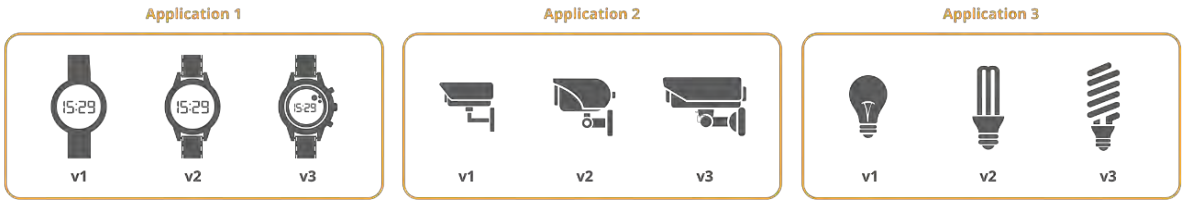


*Figura 2.15. Aplicaciones y versiones de aplicaciones*
*Fuente:*[34]

## 2.5 Visita realizada a un laboratorio presencial

Como se mencionó en 1.2.1, actualmente la PUCP ha tomado el modelo de enseñanza semipresencial, el cual ha permitido que los alumnos puedan volver a realizar los laboratorios presencialmente. Se realizó una visita a uno de los laboratorios presenciales con la finalidad de identificar otros puntos a analizar planteados en 1.3 y qué valor agregado tendría el utilizar una plataforma IoT respecto a controlar el equipo manualmente de manera presencial. El laboratorio tiene una duración de 4 horas y el desarrollo se separa en dos partes.

1. En la primera parte, el jefe del laboratorio (el encargado de supervisar y apoyar a los alumnos en el desarrollo de sus experiencias), explica lo que se realizará en la sesión.
2. En la segunda parte, se realizan las experiencias las cuales se basan en realizar mediciones manuales a las características del aire en diferentes procesos psicrométricos. Para causar el cambio de las características de aire, el alumno debe presionar *switches* que encienden, por ejemplo, resistencias que permiten que el aire dentro del equipo aumente su temperatura. Para esto, el dispositivo cuenta con un tablero el cual se puede observar en la Figura 2.16.


*Figura 2.16. Tablero de control del equipo a ser controlado remotamente*
*Fuente: fotografía propia*

Las mediciones antes mencionadas se realizan cada cierto intervalo de tiempo; es decir, no existe una recolección continua de la data generada por el equipo. Así, se pueden obtener dos puntos extras a los planteados en 1.3 a ser evaluados también en el siguiente capítulo.

- El alumno tiene un acceso limitado en tiempo del control del equipo. Así, la plataforma IoT escogida debe permitir controlar el tiempo que el usuario tiene acceso al control remoto del dispositivo mecánico.
- El control remoto del dispositivo debe apuntar, principalmente, a activar y desactivar *switches* como se mostró en la Figura 2.16.

Por último, se identifica que al contar con una plataforma IoT al cual se encuentre el dispositivo mecánico conectado permitiría un monitoreo continuo de la *data* generada por el equipo, ya que este, a través de sensores, se lo enviaría a la plataforma. Así, se mejora la experiencia del alumno, ya que no solo no necesitaría realizar mediciones manualmente, sino que, además, podría observar *data* que actualmente se considera, por llamarlo de alguna manera, “perdida”, debido a los intervalos de tiempo en lo que no se realiza una medición.

# Capítulo 3. Análisis comparativo de plataformas IoT

En el capítulo 2 se presentaron las tres plataformas IoT a ser analizadas. En el presente capítulo se realizará un estudio comparativo entre dichas plataformas enfocándonos en 3 aspectos principales: las herramientas de visualización que ofrece, monitoreo y control remoto de los dispositivos conectados a la plataforma y el nivel de gestión de usuarios que ofrece la plataforma. Para realizar la prueba de funcionamiento del monitoreo y control remoto de los dispositivos desde la plataforma, se simulará el envío de data del equipo mecánico a través de un programa escrito en Python y, específicamente, se utilizará la librería paho-mqtt la cual permite la conexión a un bróker MQTT, publicación de mensajes y suscripción a tópicos para la recepción de mensajes [35].

## 3.1 ThingSpeak

### 3.1.1 Análisis de data y herramientas de visualización en ThingSpeak

Es posible realizar un análisis de datos en ThingSpeak, ya que cuenta con una opción llamada "MATLAB Analysis" en donde es posible obtener la data de diferentes canales y procesar dicha

data, según sea requerido, utilizando lenguaje MATLAB. Por ejemplo, dado un sensor que envíe valores de temperatura y humedad a cierto canal, con MATLAB Analysis se podría calcular el valor del punto de rocío y agregarlo al canal para su visualización. Además, es posible utilizar la App de visualizaciones de MATLAB desde ThingSpeak con la cual se puede visualizar y explorar la data recolectada o la data analizada utilizando visualizaciones del tipo *area plot*, *line plot* o cualquier otro tipo de gráfico posible disponible en la galería de ploteo de MATLAB [36]. Además, es posible la asignación de un URL público al gráfico generado que permita su visualización por cualquier persona que tenga dicha URL o embeberlo en páginas webs.

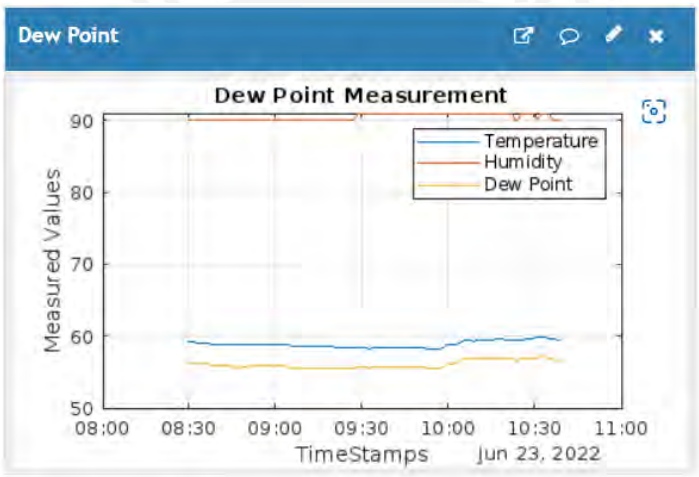


*Figura 3.1. Gráfica generada en ThingSpeak*
*Fuente: elaboración propia*

### 3.1.2 Monitoreo y control remoto de dispositivos en ThingSpeak

Como se mencionó en el punto 2.4.2.1 ThingSpeak posee el concepto de canales en donde almacena la data recibida. Dicho canal puede tener un máximo de 8 *fields*. Es decir, si representamos un canal como una entidad, se puede tener hasta 8 valores que caracterizan a dicha entidad. Asimismo, y de manera sencilla, se puede asociar dicho canal a un "dispositivo MQTT" el cual al ser creado brinda las credenciales necesarias para el envío de *data* a través del protocolo MQTT (client ID, username, and password).

### 3.1.3 Prueba de monitoreo y control remoto de dispositivos en ThingSpeak

Se creó un canal llamado “My Machine Channel” y debido, a que solo permite 8 atributos a poder ser manejado por un canal, solo se envió valores a 8 tópicos a través del protocolo MQTT. En la Figura 3.2 podemos notar que los valores llegan correctamente y, por ello, se visualizan en ThinSpeak, lo que permite el monitoreo del estado del dispositivo en tiempo real. El código utilizado se encuentra en el Anexo 1.

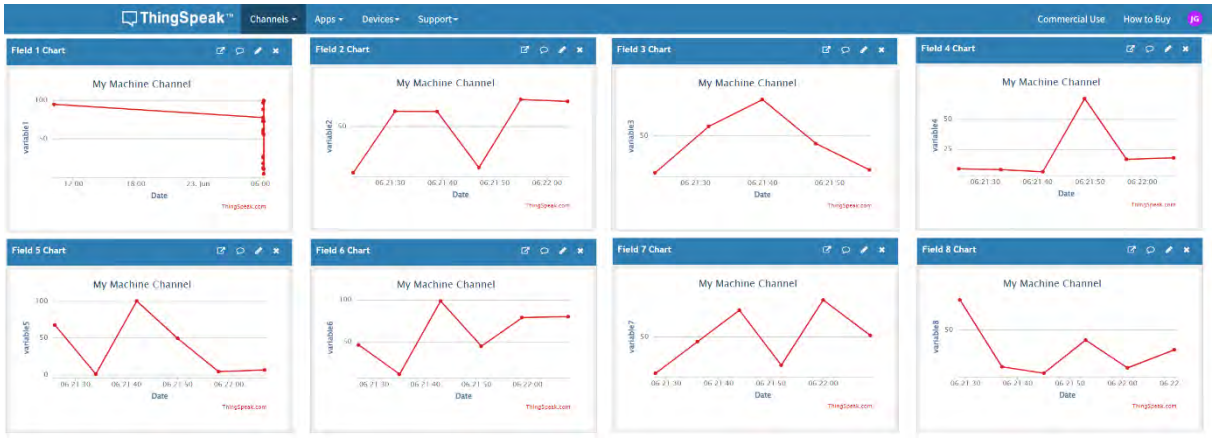


*Figura 3.2. Gráficas generadas de los 8 valores enviados*
*Fuente: elaboración propia*

Sin embargo, ThingSpeak no permite directamente controlar un equipo. Es decir, no permite de manera visual una herramienta, por ejemplo, un botón, que permita el control sobre un equipo. La funcionalidad se podría realizar suscribiendo el dispositivo a un canal y que el alumno a través de un cliente MQTT, como MQQTX para desktop [37], envíe un valor al canal. Como el dispositivo está suscrito al MQTT bróker de ThingSpeak, recibiría dicho valor y realizaría una acción, como el apagarse o encenderse. Sin embargo, esto no es conveniente, pues se busca controlar remotamente el equipo directamente desde la plataforma IoT.

### 3.1.4 Gestión de usuarios en ThingSpeak

ThingSpeak no tiene una funcionalidad propia que permita una gestión de usuarios.

## 3.2 ThingsBoard

### 3.2.1 Herramientas de visualización en ThingsBoard

ThingsBoard permite múltiples herramientas de visualización utilizando los *ThingsBoard* widgets. Los *widgets* de ThingsBoard no solo permiten la visualización de data, sino que cada *widget* brinda al usuario otras funcionalidades como el control remoto de dispositivos, administración de alarmas o mostrar contenido estático como un contenido en HTML [38]. Existen 5 tipos de *widgets* en ThingsBoard, los cuales son los siguientes:

- ***Latest values*:** Permite visualizar los últimos valores de un atributo específico de un dispositivo.

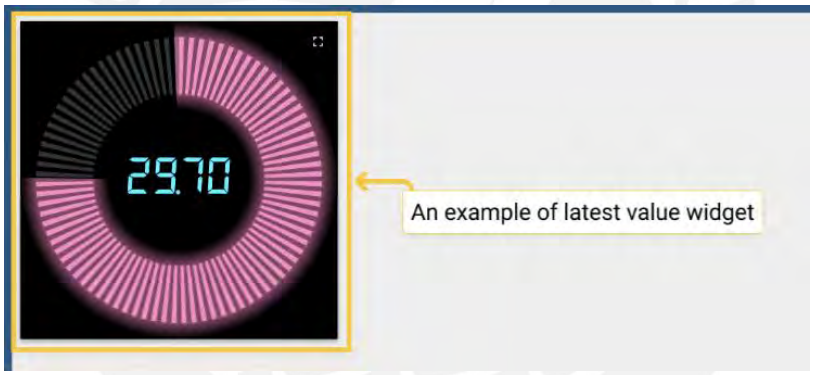


*Figura 3.3. Widget del tipo latest values*
*Fuente:* [38]

- ***Time series***: Permite la visualización de valores dentro de un periodo de tiempo seleccionado.


*Figura 3.4. Widget del tipo time series*
*Fuente:*[38]

- ***RPC* (*Control widget*):** Permite el envío de comandos RPC hacia los dispositivos, lo que permite controlarlos remotamente desde ThingsBoard.

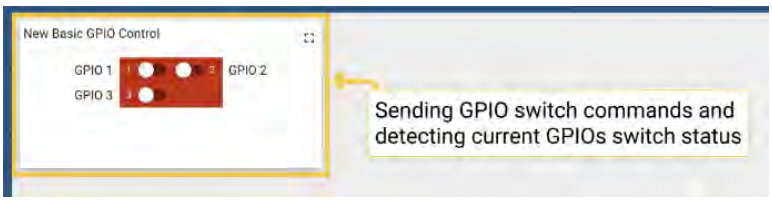


*Figura 3.5. Widget del tipo control widget*
*Fuente:*[38]

- ***Alarm Widget*:** Permite la visualización de alarmas, las cuales pueden ser configuradas para activarse dadas ciertas reglas.

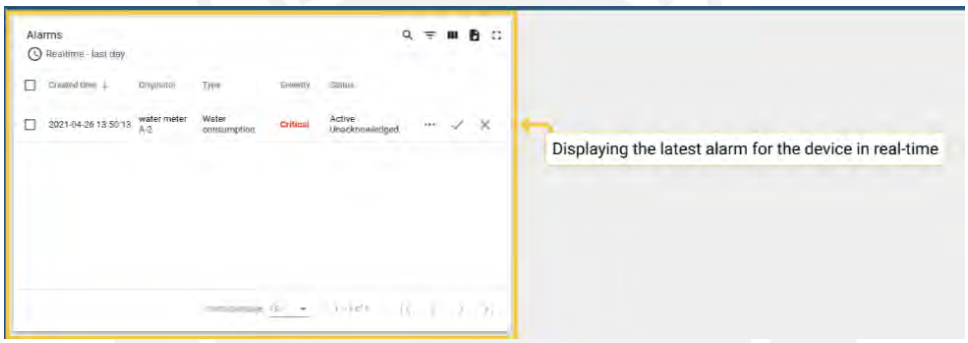


*Figura 3.6. Widget del tipo alarm*
*Fuente:*[38]

- ***Static***: Permite visualizar un contenido HTML personalizable.

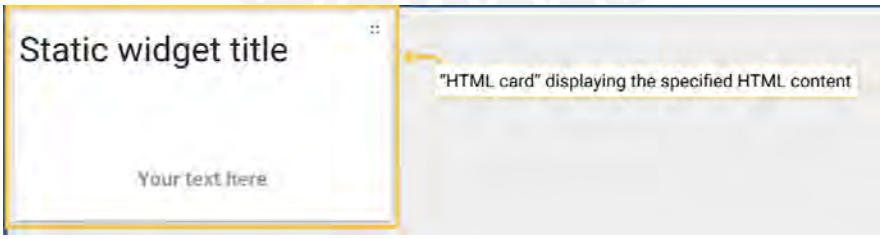


*Figura 3.7. Widget del tipo static*
*Fuente:*[38]

### 3.2.2 Monitoreo y control remoto de dispositivos en ThingsBoard

ThinsgBoard permite conectar dispositivos a través de diferentes maneras una de ellas es a través del protocolo MQTT.

### 3.2.3 Prueba de monitoreo y control remoto de dispositivos en ThingsBoard

Para realizar la prueba de control remoto, en la plataforma ThingsBoard se creó el dispositivo "My machine" y se envió data con valores de las variables variableX (para $X \geq 1$ $y$ $X \leq 20$). El resultado del recibimiento de data se puede observar al generar un *dashboard* del tipo *Time Series* asociado al dispositivo anteriormente creado. El archivo utilizado se encuentra en el Anexo 2.

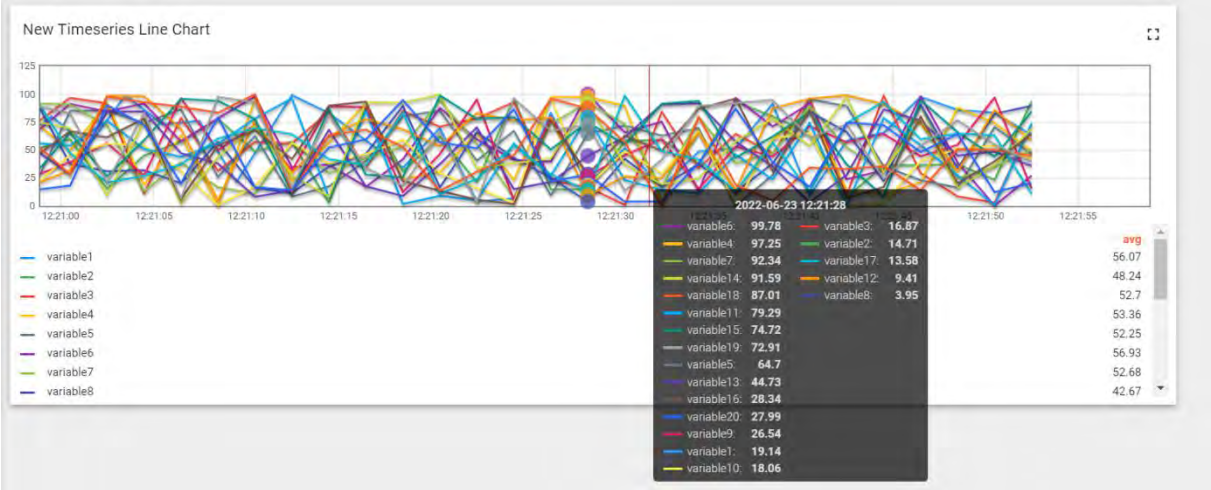


*Figura 3.8. Dashboard del tipo time series generado*
*Fuente: Elaboración propia*

Se puede observar que la plataforma recibe exitosamente la data enviada. Asimismo, para comprobar el control remoto, se agregó un Switch control, el cual es un *widget* del tipo control. Además, se creó un archivo Python que simula el estado de switch de un equipo, con la finalidad de comprobar el correcto control de dicho "switch" desde la plataforma. Se comprueba el funcionamiento en la Figura 3.9 y el archivo utilizado se encuentra en el Anexo 3


*Figura 3.9. Ejemplo de control remoto de un switch con un switch control*
*Fuente: elaboración propia*

### 3.2.4 Gestión de usuarios en ThingsBoard

Uno de los *features* más importante de Thingsboard es que brinda la posibilidad de la creación de *customers*. Se puede entender un *customer* como un conjunto de usuarios al cual puede ser asignado a un dispositivo y, por ende, a un *dashboard* asociado a este dispositivo. Así, se permite la visualización de dicho *dashboard* solo por los usuarios perteneciente a un *customer* en específico [39].

### 3.2.5 Prueba de gestión de usuarios en ThingsBoard

Se crea un *dashboard* adicional llamado “My Machine Dashboard – Alumnos” el cual será asignado a un *customer* llamado “Alumno”. Dicho *dashboard* solo tendrá un gráfico que mostrará la evolución solo de las variables 1 y 2 de “My Machine”. Además, se crea un usuario llamado “Alumno 1” perteneciente a “Alumno”. Así, al ingresar con las credenciales de

Alumno 1 se ve que solo se puede ver el gráfico “Varibale 1 vs. Variable 2”. Así, se comprueba que ThingsBoard permite un grado de gestión de usuarios, ya que los alumnos creados dentro del *customer* “Alumno” solo podrán ver el *dashboard* que se les fue asignado.

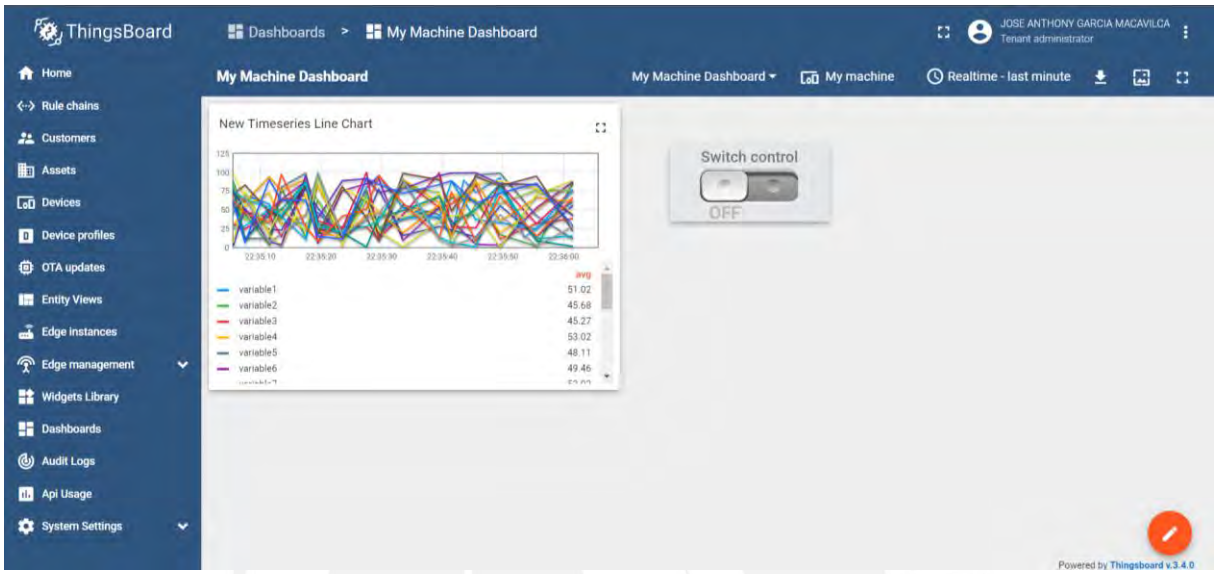


*Figura 3.10. Dashboard observado por el administrador*
*Fuente: elaboración propia*

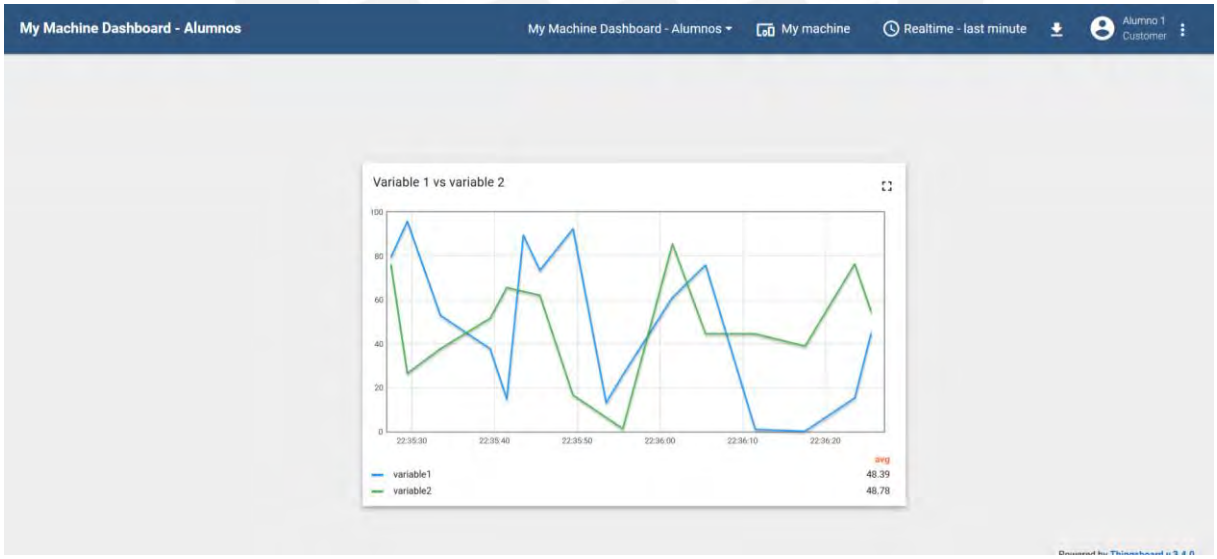


*Figura 3.11. Dashboard observado por el Alumno 1*
*Fuente: elaboración propia*

Sin embargo, se notó que un dispositivo solo puede asignarse a un *customer*, por lo que no se puede tener más de un rol al cual asignar diferentes tipos de *dashboards*. Una manera seria crear

dos dispositivos y enviar data a ambos dispositivos. Así, un dispositivo sería asignado a alumnos y el otro a jefes de práctica. No obstante, esto sería ineficiente, ya que se tendría que enviar la data repetida a dos dispositivos en ThingsBoard, lo cual no es lo adecuado. Además, el acceso de un usuario al sistema no se puede limitar a un cierto tiempo limitado. Para lograrlo se debería realizar dicha funcionalidad de forma manual.

## 3.3 Kaa IoT

### 3.3.1 Herramientas de visualización en Kaa IoT

El *feature* de visualización de Kaa es implementado por el microservicio llamado *Web Dashboard* el cual provee una interfaz de usuario para la interacción con otros servicios. El principal componente de la interfaz de usuario es un *dashboard*. Un *dashboard* es una página en donde un *widget* puede ser ubicado. Un widget es el componente de la interfaz de usuario que le permite realizar ciertas acciones o acceder a *data* de los dispositivos [40]. Kaa cuenta con una colección de diferentes tipos de *widgets*. Por ejemplo, se tiene el *multi series chart* que es utilizado para mostrar series de datos en el tiempo de uno o varios *endpoints*. También, se tiene los *widgets* de ejecución de comando que permiten enviar mensajes hacia los dispositivos.

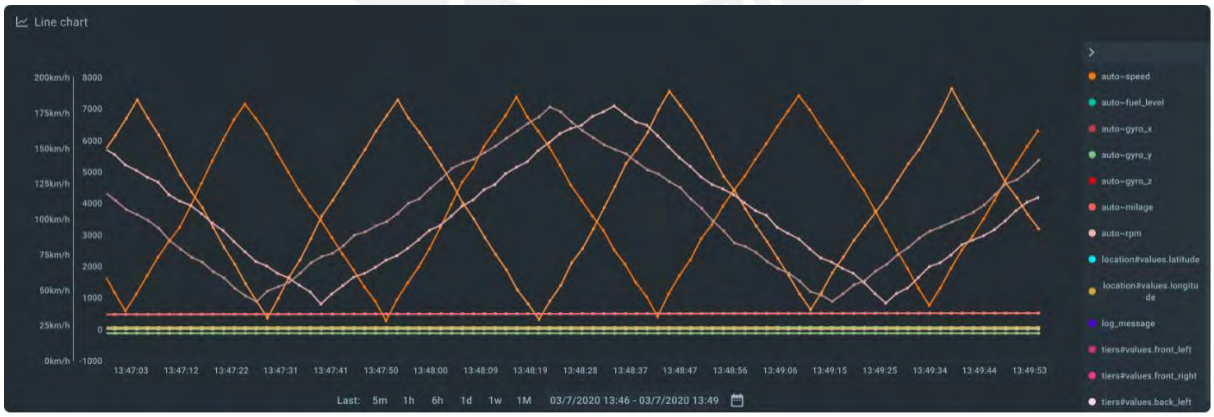


*Figura 3.12. Ejemplo de multi series chart*
*Fuente:* [41]

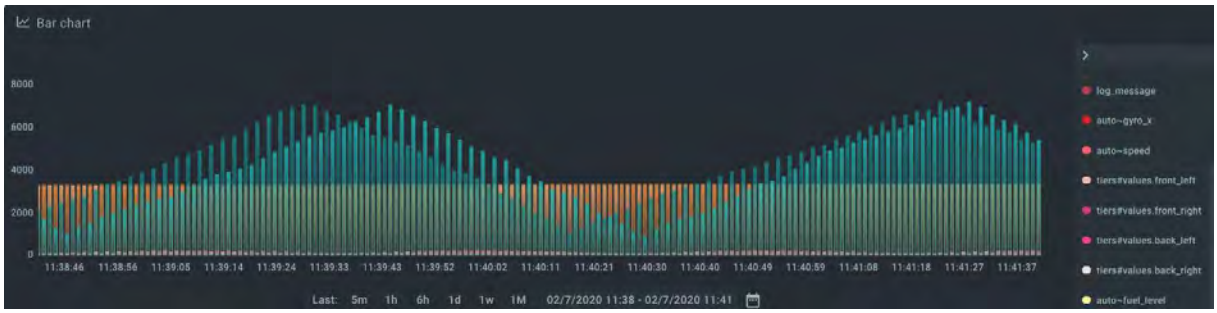


*Figura 3.13. Ejemplo de multi series chart en modo bar*
*Fuente:*[41]


*Figura 3.14. Widget del tipo ejecución de comando*
*Fuente:*[41]

### 3.3.2 Monitoreo y control remoto de dispositivos en Kaa

Kaa permite el envío de *data* a través de los protocolos MQTT y HTTP, lo cual permite el monitoreo del estado de un dispositivo agregado a Kaa [42]. Para el control de un dispositivo, Kaa provee un *feauture* llamado "invocación de comando" el cual permite el envío de mensajes hacia los dispositivos conectados para que estos reaccionen a dichos mensajes. Por ejemplo, se puede hacer que se encienda o apague un foco remotamente. Para esto Kaa presenta dos conceptos: comando y tipo de comando. El comando es un mensaje corto que se envía a los dispositivos conectados y el tipo de comando representa el tipo de comando que se requiere ser ejecutado por el dispositivo (Por ejemplo, "reiniciar", "encender-luz" o "cerrar-puerta") [43].

### 3.3.3 Prueba de monitoreo y control remoto de dispositivos en Kaa

Se crea una aplicación llamada “MyMachineApp” y una versión de dicha aplicación. A dicha aplicación se le agrega un *endpoint*, el cual tiene un token asociado que será usado para el envío de *data* a través del protocolo MQTT. Se enviará valores de variablesX (para $X \geq 1$ $y$ $X \leq 20$). El funcionamiento se comprueba en la Figura 3.15 y el archivo utilizado se encuentra en el Anexo 4.


*Figura 3.15. Recibimiento de las variables enviadas*
*Fuente: elaboración propia*

Para el control remoto del dispositivo, se agregó un *widget* que permita invocar un comando el cual al ser desactivado causa que la variable2 tome valores de 0 y cuando se activa tome valores diferentes a cero. Se comprueba el funcionamiento en las figuras Figura 3.16 y Figura 3.17.

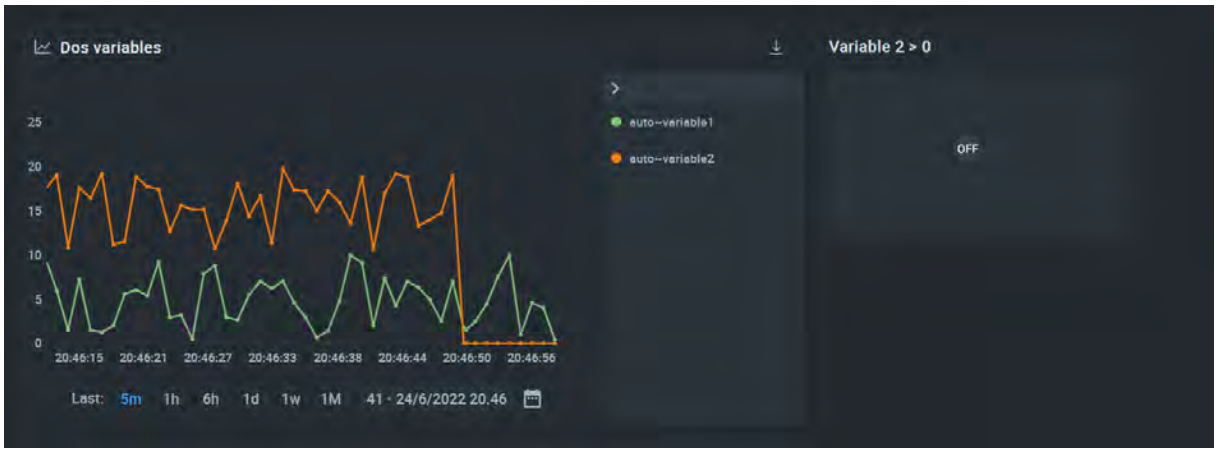


*Figura 3.16. Ejemplo de control de dispositivo con "variable2>0" desactivado*
*Fuente: elaboración propia*

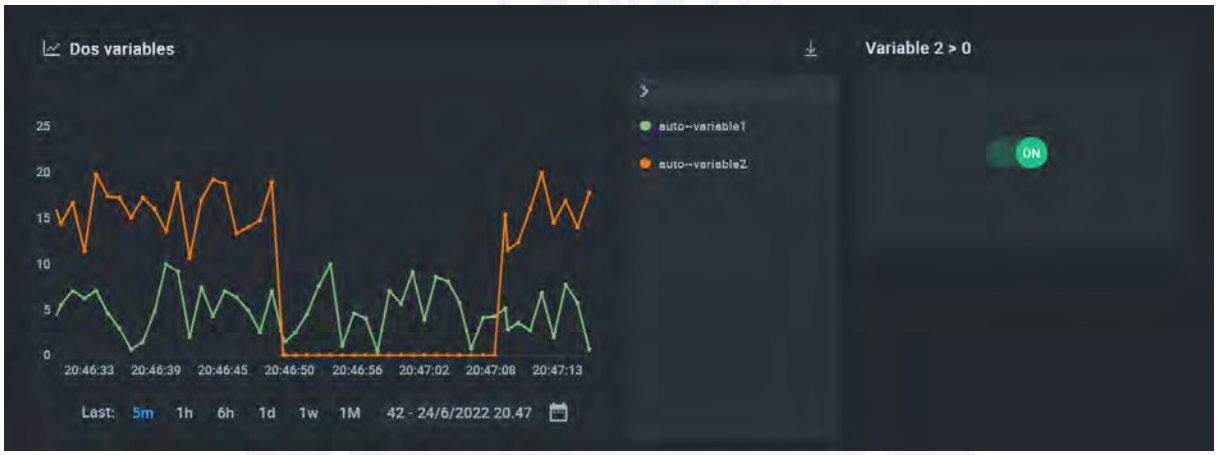


*Figura 3.17. Ejemplo de control de dispositivo con "variable2>0" activado*
*Fuente: elaboración propia*

### 3.3.4 Gestión de usuarios en Kaa

Kaa permite añadir usuarios, asignarles permisos y restringir sus accesos a uno o más recursos del tipo *application* o *endpoint*. Para conseguir estas funcionalidades, Kaa trabaja con Keycloak, el cual es un administrador de identidad y acceso de código abierto [44]. Así, la gestión de usuarios se puede lograr a dos niveles. En primer lugar, se puede restringir el acceso por tipo de recurso. Es decir, se podría asignar a un usuario solo la visualización de *dashboards*, lo que le permitiría tener acceso a todos los *dashboards* existentes. En segundo lugar, se puede

tener un nivel más elevado de restricción utilizando políticas y permisos según la fuente (*application*, *endpoint* o *dashboard*) y que acción puede realizar (lectura y/o escritura) [45].

### 3.3.5 Prueba de gestión de usuario en Kaa

Se crearon los usuarios alumno1 y jp1 a los cuales se les crearon sus respectivas políticas llamadas “alumno-permit-policy” y “jp-permit-policy”, respectivamente. Para permitir que alumno1 vea un *dashboard* diferente que jp1 (el cual tendrá el botón de control creado en 3.3.3), se le asignó la lectura de la aplicación y del *endpoint* asociados a MyMachine, así como también de su *dashboard* correspondiente (el que no tiene el *widget* de control). Por otro lado, al jp se le asignó permisos de lectura de la aplicación y *endpoint* asociados a MyMachine, así como también de su *dashboard* correspondiente (el que tiene el *widget* de control).


*Figura 3.18. Lista de usuarios en la plataforma*
*Fuente: elaboración propia*

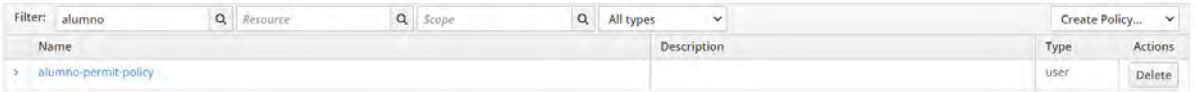


*Figura 3.19. Políticas asociadas al alumno1*
*Fuente: elaboración propia*

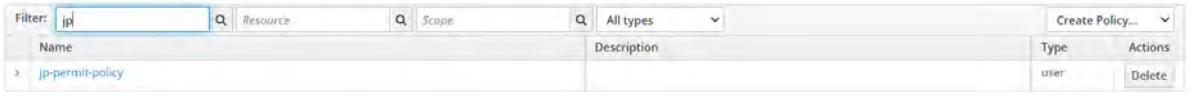


*Figura 3.20. Políticas asociadas al jp1*
*Fuente: elaboración propia*

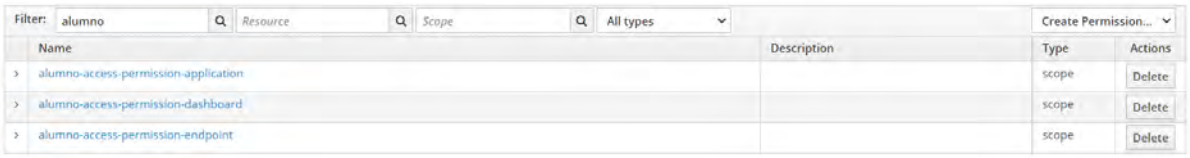


*Figura 3.21. Permisos asociados a alumno-permit-policy*
*Fuente: elaboración propia*

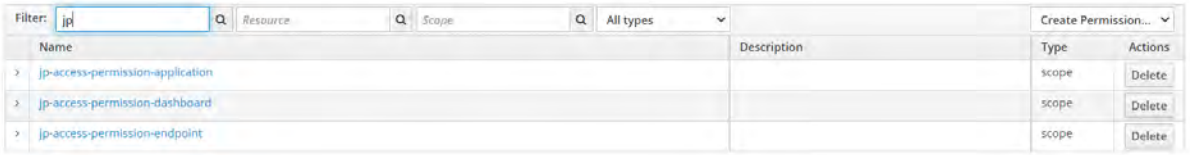


*Figura 3.22. Permisos asociados a jp-permit-policy*
*Fuente: elaboración propia*

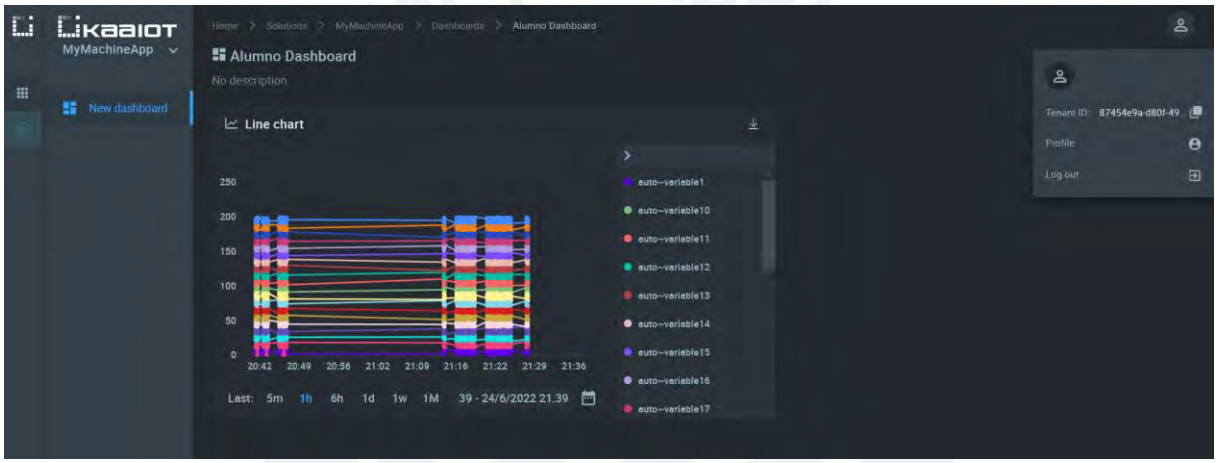


*Figura 3.23. Vista del usuario alumno1*
*Fuente: elaboración propia*

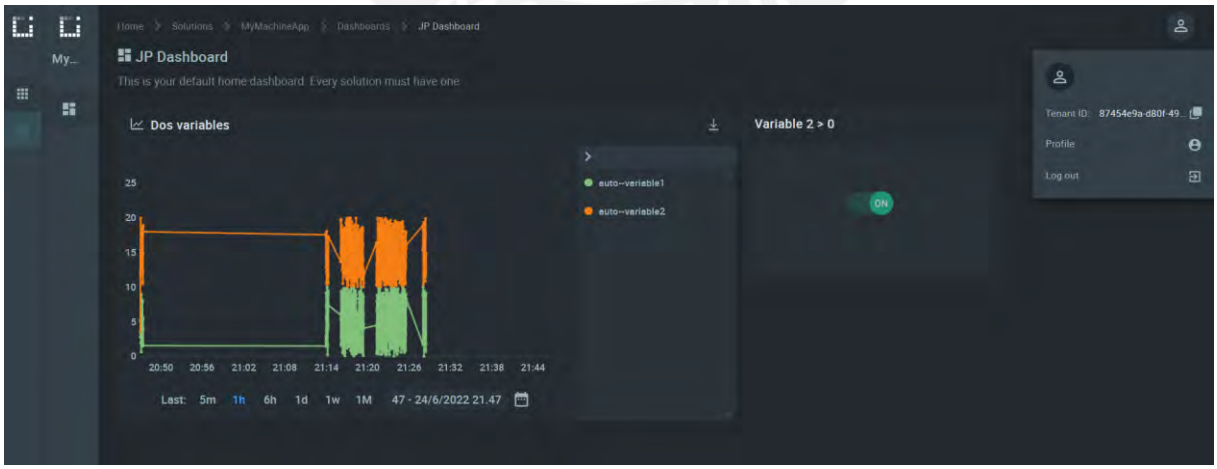


*Figura 3.24. Vista del usuario jp1*
*Fuente: elaboración propia*

Notamos que es posible un nivel de gestión de roles separado en alumno, jp y administrador (el creador de los *dasboards*, el cual tiene control de toda la plataforma). Sin embargo, el acceso de un usuario al sistema no se puede limitar a un cierto tiempo limitado. Para lograrlo se debería realizar dicha funcionalidad de forma manual, activando y desactivando permisos.

## 3.4 Análisis comparativo de herramientas de visualización

En los ítems anteriores se presentó las herramientas de visualización que ofrecen las plataformas analizadas. Se observa que las 3 plataformas analizadas ofrecen un catálogo amplio al momento de visualizar la data recibida por los dispositivos y, por lo tanto, más de una manera de representar dicha *data*. Las tres plataformas cuentan con gráficos lineales, lineales múltiples (más de una variable en una gráfica), gráficos de barras, etc. Sin embargo, sólo ThingsBoard y Kaa tienen maneras gráficas de representar un *switch*.

*Tabla 3.1. Tabla comparativa de herramientas de visualización de las plataformas analizadas*
*Fuente: Elaboración propia*

| | ThingSpeak | ThingsBoard | KaaIoT |
|---|---|---|---|
| ¿Permite la visualización de la data en diferentes tipos de gráficas? | Sí | Sí | Sí |
| ¿Permite la representación gráfica de un *switch*? | No | Sí | Sí |

## 3.5 Análisis comparativo de monitoreo y control remoto de dispositivo

En los ítems anteriores se presentó la manera en la que las plataformas analizadas permitían el monitoreo y el control remoto de los dispositivos conectados a ellas. Se observa que las 3 plataformas permiten recibir data de los dispositivos, utilizando diferentes protocolos (en las pruebas realizadas se prefirió utilizar MQTT), y que, gracias a las herramientas de visualización de dichas plataformas, permiten un monitoreo continuo de los equipos. Sin embargo, ThingSpeak tenía la desventaja de que los dispositivos, a nivel de plataforma, solo podían enviar

8 variables diferentes. Por otro lado, se observa que ThingsBoard y Kaa permiten herramientas de control remoto de dispositivos. Es decir, tienen alguna manera de envío de comandos a los dispositivos y que estos reaccionen a ellos. Sin embargo, ThingSpeak no cuenta con dicha funcionalidad, aunque se podría buscar una manera de conseguirlo, como se explicó anteriormente, pero no es lo adecuado para fines de la presente investigación.

*Tabla 3.2. Tabla comparativa del monitoreo y control remoto de dispositivo*
*Fuente: Elaboración propia*

| | ThingSpeak | ThingsBoard | KaaIoT |
|---|---|---|---|
| ¿Permite el monitoreo continuo de los dispositivos conectados a la plataforma? | Sí, pero con limitaciones. | Sí | Sí |
| ¿Permite el control remoto de dispositivos desde la propia plataforma? | No | Sí | Sí |

## 3.6 Análisis comparativo de gestión de usuarios

En los ítems anteriores se presentó la manera en la que las plataformas analizadas permitían el control de usuarios a través de roles, si estas tenían implementada esta funcionalidad. Se observa que ThingSpeak no permite una gestión de usuarios a nivel de roles. Por otro lado, ThingsBoard y Kaa, sí. De estas dos últimas plataformas, se observa que ThingsBoard no permite tener más un solo rol adicional diferente al de administrador y que Kaa permite un nivel superior de gestión de usuarios, ya que permite una asignación de los recursos al cual un usuario puede acceder. Asimismo, se halló que ninguna de estas dos últimas plataformas permite el acceso limitado en tiempo de los recursos asignados a los usuarios.

*Tabla 3.3. Tabla comparativa de gestión de usuarios*
*Fuente: Elaboración propia*

| | ThingSpeak | ThingsBoard | KaaIoT |
|---|---|---|---|
| ¿Permite la gestión de usuarios a nivel | No | Sí, pero con limitaciones en la cantidad de roles. | Sí |

| | | | |
|---|---|---|---|
| de roles (alumno, jp y administración)? | | | |
| ¿Permite el acceso limitado por tiempo? | No | No | No |

# Capítulo 4. Recomendaciones de aplicabilidad

En el capítulo anterior se realizó el análisis de tres plataformas IoT basados en ciertas funcionalidades que estas deben tener para poder ser aplicados en el desarrollo de un laboratorio remotamente. Sin embargo, se notó que a medida que los requerimientos eran más específicos, como el tener una gestión de usuarios a nivel de roles o limitar el acceso a la plataforma por tiempo, no todas las plataformas analizadas cumplen con los requerimientos planteados. Así, existe una relación inversamente proporcional entre requerimientos soportados por una plataforma y la probabilidad de encontrar una plataforma que cumpla con todos esos requerimientos. Así, mientras más requerimientos se tengan, menor es la probabilidad de encontrar una plataforma que cumpla con todos los requerimientos planteados.

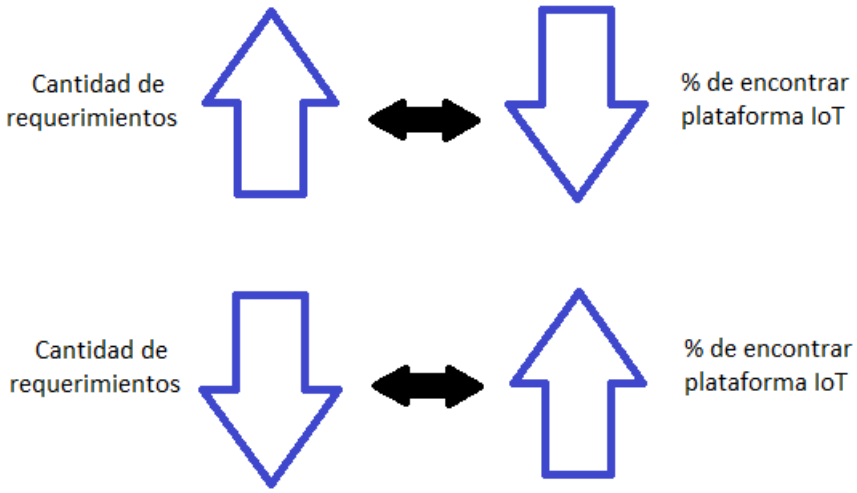


*Figura 4.1. Relación cantidad de requerimientos-probabilidad de encontrar plataforma IoT*
*Fuente: elaboración propia*

La presente investigación puede ser aplicada al momento de tomar una decisión de la utilización de una plataforma IoT en el desarrollo de un laboratorio remoto o, según aumenten los requerimientos, plantear el diseño e implementación de una plataforma IoT propia basada en los elementos comunes de las plataformas IoT analizadas en la presente investigación. De esta manera poder cumplir todos los requerimientos a ser cumplidos por la plataforma. El desarrollo de la investigación ha permitido identificar tres elementos comunes dentro de las plataformas analizadas las cuales son las siguientes:

- El tipo de protocolo de comunicación utilizado para la interacción plataforma-dispositivo.
- El componente encargado de recibir los mensajes de los dispositivos conectados
- La interfaz de usuario final que permite la interacción usuario-dispositivo.

Un caso similar se plantea en [46] en donde se realizó el diseño de una plataforma IoT para el control de variables físicas con propósitos educativos en donde se consiguió obtener información en tiempo real de variables físicas de forma remota.

# CONCLUSIONES

1. Se realizó el estudio y análisis de las plataformas IoT más adecuadas para el control remoto de un dispositivo mecánico con cierta cantidad de canales de transmisión y recepción de *data*.
2. Por lo desarrollado en 3.4, se encuentra que todas las plataformas analizadas cuentan con herramientas de visualización que permiten observar la *data* a través de diferentes maneras, como gráficos de líneas y gráficos de barras.
3. Después de las pruebas realizadas en 3.1.3, 3.2.3 y 3.3.3, se obtiene que todas las plataformas analizadas permiten el monitoreo remoto de los canales de un equipo mecánico, pero no todas las plataformas permiten el control remoto del mismo, pues no brindan una herramienta propia que permita el envío de comandos hacia un dispositivo conectado.
4. Como se puede verificar en las pruebas realizadas en 3.1.4, 3.2.5 y 3.3.5, no todas las plataformas permiten una gestión de usuarios a nivel de roles, ya que, como en el caso de ThingSpeak, no tienen dicha funcionalidad implementada o la creación de roles es limitado en cantidad, como en el caso de ThingsBoard.
5. De las tres plataformas analizadas, se encuentra que la plataforma más adecuada para el control remoto de un dispositivo mecánico con cierta cantidad de canales de transmisión y recepción de *data* es Kaa IoT.

# BIBLIOGRAFÍA

[1] "Laboratorios virtuales en Ingeniería PUCP - PuntoEdu PUCP." https://puntoedu.pucp.edu.pe/vida-estudiantil/laboratorios-virtuales-en-ingenieria-pucp-se-desarrollan-con-exito/ (accessed Jun. 15, 2022).

[2] "La transformación digital en Ingeniería PUCP y sus 8 procesos clave liderados por el Ing. Genghis Ríos - Departamento de Ingeniería." https://departamento.pucp.edu.pe/ingenieria/2020/04/30/transformacion-digital-en-departamento-de-ingenieria/ (accessed Jun. 15, 2022).

[3] "Preguntas frecuentes sobre la semipresencialidad del ciclo 2022-1 - Facultad de Ciencias Sociales." https://facultad.pucp.edu.pe/ciencias-sociales/notas-de-prensa/preguntas-frecuentes-sobre-la-semipresencialidad-del-ciclo-2022-1/ (accessed Jun. 15, 2022).

[4] "What is IoT? - Internet of Things Beginner's Guide - AWS." https://aws.amazon.com/what-is/iot/ (accessed Jun. 09, 2022).

[5] L. K. B. Félix, "DISEÑO E IMPLEMENTACIÓN DE UN CONTROLADOR PARA LA FASE DE CALENTAMIENTO Y ENFRIAMIENTO SENSIBLE DEL BANCO DE PSICROMETRÍA," PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ, 2010.

[6] J. Mineraud, O. Mazhelis, X. Su, and S. Tarkoma, "A gap analysis of Internet-of-Things platforms," *Comput. Commun.*, vol. 89–90, pp. 5–16, 2016, doi: 10.1016/j.comcom.2016.03.015.

[7] H. Hejazi, H. Rajab, T. Cinkler, and L. Lengyel, "Survey of platforms for massive IoT," in *2018 IEEE International Conference on Future IoT Technologies, Future IoT 2018*, 2018, vol. 2018-Janua, pp. 1–8. doi: 10.1109/FIOT.2018.8325598.

[8] O. Toutsop, K. Kornegay, and E. Smith, "A Comparative Analyses of Current IoT Middleware Platforms," in *Proceedings - 2021 International Conference on Future*

*Internet of Things and Cloud, FiCloud 2021*, 2021, pp. 413–420. doi: 10.1109/FiCloud49777.2021.00067.

[9] J. Guth, U. Breitenbucher, M. Falkenthal, F. Leymann, and L. Reinfurt, "Comparison of IoT platform architectures: A field study based on a reference architecture," in *2016 Cloudification of the Internet of Things, CIoT 2016*, 2016, pp. 1–6. doi: 10.1109/CIOT.2016.7872918.

[10] M. Abomhara and G. M. Køien, "Cyber security and the internet of things: Vulnerabilities, threats, intruders and attacks," *J. Cyber Secur. Mobil.*, vol. 4, no. 1, pp. 65–88, 2015, doi: 10.13052/jcsm2245-1439.414.

[11] S. Khare and M. Totaro, "Big Data in IoT," *2019 10th Int. Conf. Comput. Commun. Netw. Technol. ICCCNT 2019*, pp. 6–12, 2019, doi: 10.1109/ICCCNT45670.2019.8944495.

[12] V. Seoane, C. Garcia-Rubio, F. Almenares, and C. Campo, "Performance evaluation of CoAP and MQTT with security support for IoT environments," *Comput. Networks*, vol. 197, no. April, p. 108338, 2021, doi: 10.1016/j.comnet.2021.108338.

[13] I. E. T. F. IETF, "RFC 7252 - The Constrained Application Protocol - CoAP RFC 7252.pdf," *Request for Comments: 7252*. 2014. Accessed: Jun. 13, 2022. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc7252#section-2

[14] T. Jaffey, "MQTT and CoAP, IoT Protocols | The Eclipse Foundation," 2014. https://www.eclipse.org/community/eclipse_newsletter/2014/february/article2.php (accessed Jun. 13, 2022).

[15] S. Hamdani and H. Sbeyti, "A comparative study of COAP and MQTT communication protocols," *7th Int. Symp. Digit. Forensics Secur. ISDFS 2019*, pp. 1–5, 2019, doi: 10.1109/ISDFS.2019.8757486.

[16] mqtt-v5.0, "MQTT Version 5.0," *Oasis-Open*, no. March, p. 137, Mar. 2019, Accessed:

Jun. 13, 2022. [Online]. Available: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html

[17] MQTT, "MQTT: The Standard for IoT Messaging," *Mqtt.Org*, 2021. https://mqtt.org/ (accessed Jun. 13, 2022).

[18] IoT Analytics, "IoT Platform Companies Landscape 2021/2022," 2021. https://iot-analytics.com/iot-platform-companies-landscape/ (accessed Jun. 29, 2022).

[19] M. Ullah and K. Smolander, "Highlighting the key factors of an IoT platform," *2019 42nd Int. Conv. Inf. Commun. Technol. Electron. Microelectron. MIPRO 2019 - Proc.*, pp. 901–906, 2019, doi: 10.23919/MIPRO.2019.8756748.

[20] A. B. Bondi, "Characteristics of scalability and their impact on performance," *Proc. Second Int. Work. Softw. Perform. WOSP 2000*, pp. 195–203, 2000, doi: 10.1145/350391.350432.

[21] I. Ilunin, "How to Choose Your IoT Platform - Should You Go Open-Source?," *IoT All*, no. di, p. 1, 2017, Accessed: Jun. 20, 2022. [Online]. Available: https://medium.com/iotforall/how-to-choose-your-iot-platform-should-you-go-open-source-23148a0809f3

[22] L. Bulej, T. Bures, P. Hnetynka, V. Camra, P. Siegl, and M. Topfer, "IVIS: Highly customizable framework for visualization and processing of IoT data," *Proc. - 46th Euromicro Conf. Softw. Eng. Adv. Appl. SEAA 2020*, pp. 585–588, 2020, doi: 10.1109/SEAA51224.2020.00095.

[23] Grafana Labs, "Grafana Features | Grafana Labs," 2020. https://grafana.com/grafana/ (accessed Jun. 20, 2022).

[24] A. A. Ismail, H. S. Hamza, and A. M. Kotb, "Performance Evaluation of Open Source IoT Platforms," *2018 IEEE Glob. Conf. Internet Things, GCIoT 2018*, pp. 1–5, 2019, doi: 10.1109/GCIoT.2018.8620130.

[25] P. P. Ray, "A survey of IoT cloud platforms," *Futur. Comput. Informatics J.*, vol. 1, no. 1–2, pp. 35–46, 2016, doi: 10.1016/j.fcij.2017.02.001.

[26] S. Gupta, A. Gupta, and Y. Hasija, "Transforming IoT in aquaculture: A cloud solution," in *AI, Edge and IoT-based Smart Agriculture*, Academic Press, 2022, pp. 517–531. doi: 10.1016/b978-0-12-823694-9.00020-7.

[27] "ThingsBoard - Open-source IoT Platform." https://thingsboard.io/ (accessed Jun. 25, 2022).

[28] "ThingsBoard architecture | ThingsBoard Community Edition." https://thingsboard.io/docs/reference/ (accessed Jun. 25, 2022).

[29] M. Chwalisz, "ThingSpeak Documentation," *Mathworks*, pp. 1–2, 2019, Accessed: Jun. 22, 2022. [Online]. Available: https://www.mathworks.com/help/thingspeak/

[30] MathWorks, "What Is MATLAB? - MATLAB & Simulink," 2018. https://www.mathworks.com/discovery/what-is-matlab.html (accessed Jun. 22, 2022).

[31] ThingSpeak, "Learn More About ThingSpeak," *Thingspeak.com*, 2019. https://thingspeak.com/pages/learn_more (accessed Jun. 22, 2022).

[32] Kaa, "Enterprise IoT Platform with Free Plan | Kaa," 2021. https://www.kaaiot.com/ (accessed Jun. 25, 2022).

[33] "KAA: Architecture overview." https://docs.kaaiot.io/KAA/docs/current/Architecture-overview/ (accessed Jun. 25, 2022).

[34] "KAA: Kaa concepts." https://docs.kaaiot.io/KAA/docs/current/Kaa-concepts/ (accessed Jun. 25, 2022).

[35] Roger Light, "paho-mqtt · PyPI," 2020. https://pypi.org/project/paho-mqtt/#client (accessed Jun. 23, 2022).

[36] "MATLAB Plot Gallery." https://www.mathworks.com/products/matlab/plot-gallery.html# (accessed Jun. 23, 2022).

[37] “Introduction - MQTT X Documentation.” https://mqttx.app/docs (accessed Jun. 25, 2022).

[38] “Widgets Library | ThingsBoard Community Edition.” https://thingsboard.io/docs/user-guide/ui/widget-library/ (accessed Jun. 25, 2022).

[39] “Devices | ThingsBoard Community Edition.” https://thingsboard.io/docs/user-guide/ui/customers/ (accessed Jun. 25, 2022).

[40] “KAA: WD.” https://docs.kaaiot.io/KAA/docs/v1.3.0/Features/Visualization/WD/#endpoint-management (accessed Jun. 24, 2022).

[41] “KAA: Widgets.” https://docs.kaaiot.io/KAA/docs/current/Features/Visualization/WD/Widgets/#command-execution (accessed Jun. 29, 2022).

[42] “KAA: Connecting your first device.” https://docs.kaaiot.io/KAA/docs/v1.3.0/Tutorials/getting-started/connecting-your-first-device/ (accessed Jun. 24, 2022).

[43] “KAA: Sending commands to device.” https://docs.kaaiot.io/KAA/docs/v1.3.0/Tutorials/getting-started/sending-commands-to-device/ (accessed Jun. 24, 2022).

[44] “Keycloak.” https://www.keycloak.org/ (accessed Jun. 24, 2022).

[45] “KAA: User management.” https://docs.kaaiot.io/KAA/docs/v1.3.0/Tutorials/getting-started/user-management/ (accessed Jun. 24, 2022).

[46] R. E. Q. Padilla and C. L. C. Lobos, “IoT Platform for the Control and Monitoring of Physical Variables with Open Hardware Technology,” *2019 IEEE 39th Cent. Am. Panama Conv. CONCAPAN 2019*, vol. 2019-Novem, pp. 0–4, 2019, doi: 10.1109/CONCAPANXXXIX47272.2019.8977015.

# ANEXOS

**Anexo 1. Código utilizado para la Prueba de monitoreo y control remoto de dispositivos en ThingSpeak**

```
import random
import time
from tracemalloc import start
import paho.mqtt.client as mqtt_client


def connect_mqtt(broker, port, client_id, username, password):
    def on_connect(client, userdata, flags, rc):
        if rc == 0:
            print("Connected to MQTT Broker!")
        else:
            print("Failed to connect, return code %d\n", rc)

    client = mqtt_client.Client(client_id)
    client.username_pw_set(username, password)
    client.on_connect = on_connect
    client.connect(broker, port)
    return client


if __name__ == '__main__':

    broker = 'mqtt3.thingspeak.com'
    port = 1883
```

```
client_id = 'ACc6Iy0JAigXHhsZKhkNJgE'
username = 'ACc6Iy0JAigXHhsZKhkNJgE'
password = 'wt4UFJbNbw4yCgYzdTC+kSXN'

# connect to mqtt broker:
client = connect_mqtt(broker, port, client_id, username, password)
client.loop_start()

# publish a message:
channel = "1777730"
while True:
    for field in range(8):
        time.sleep(1)
        topic = f"channels/{channel}/publish/fields/field{field+1}"
        msg = round(random.random()*100, 2)
        # print(f"Send `{msg}` to topic `{topic}`")
        result = client.publish(topic, msg)
        # result: [0, 1]
        status = result[0]
        if status == 0:
            print(f"Send `{msg}` to topic `{topic}`")
        else:
            print(f"Failed to send message to topic {topic}")
```

**Anexo 2. Código 1 utilizado para la Prueba de monitoreo y control remoto de dispositivos en ThingsBoard**

```
import random
import time
import paho.mqtt.client as mqtt_client
import json


def connect_mqtt(broker, port, username):
    def on_connect(client, userdata, flags, rc):
        if rc == 0:
            print("Connected to MQTT Broker!")
        else:
            print("Failed to connect, return code %d\n", rc)

    client = mqtt_client.Client()
    client.username_pw_set(username)
    client.on_connect = on_connect
    client.connect(broker, port)
    return client
```

```
if __name__ == '__main__':

    broker = 'demo.thingsboard.io'
    port = 1883

    username = 'y855KRplsvlhqnrsAqjc' #MyMachine access token

    # connect to mqtt broker:
    client = connect_mqtt(broker, port, username)
    client.loop_start()

    # publish a message:
    topic = "v1/devices/me/telemetry"
    while True:
        time.sleep(2)
        for i in range(20):
            random_value = round(random.random()*100, 2)
            msg = {f"variable{i+1}":random_value}
            result = client.publish(topic, json.dumps(msg))
            # result: [0, 1]
            status = result[0]
            if status == 0:
                print(f"Send `{msg}` to topic `{topic}`")
            else:
                print(f"Failed to send message to topic {topic}")
```

**Anexo 3. Código 2 utilizado para la Prueba de monitoreo y control remoto de dispositivos en ThingsBoard**

```
import json
import paho.mqtt.client as mqtt

switch = {"enabled": False}

def on_connect(client, userdata, flags, rc):
    if rc == 0:
        print("Connected to MQTT Broker!")
        client.subscribe('v1/devices/me/rpc/request/+')
        print("Subscribe to 'v1/devices/me/rpc/request/+'")
        print("##### ESTADO INICIAL DEL SWITCH ######")
        print(switch)
    else:
        print("Failed to connect, return code %d\n", rc)


def on_message(client, userdata, msg):
    # print('Topic: ' + msg.topic + '\nMessage: ' + str(msg.payload))
    data = json.loads(msg.payload)
```

```
    if data["method"] == "setValue":
        switch["enabled"] = data["params"]
    print("##### SWITCH ACTUALIZADO ######")
    print(switch)

# start the client instance
client = mqtt.Client()

client.on_connect = on_connect
client.on_message = on_message

username = "y855KRplsvlhqnrsAqjc"
client.username_pw_set(username)

broker = 'demo.thingsboard.io'
port = 1883
client.connect(broker,port)


client.loop_forever()
```

**Anexo 4. Código utilizado para la Prueba de monitoreo y control remoto de dispositivos en Kaa**

```
# Simple MQTT-based command execution client for the Kaa IoT platform.
# Handles "reboot" and "zero" command types.
# See https://docs.kaaiot.io/KAA/docs/current/Tutorials/sending-commands-to-device/

import json
import paho.mqtt.client as mqtt
import random
import signal
import string
import time

KPC_HOST = "mqtt.cloud.kaaiot.com"  # Kaa Cloud plain MQTT host
KPC_PORT = 1883  # Kaa Cloud plain MQTT port

ENDPOINT_TOKEN = "JYnieNRXVg"       # Paste your endpoint token
APPLICATION_VERSION = "caqrje8n7nk35lgkub0g-v1"  # Paste your application version

class DataCollectionClient:

    def __init__(self, client):
        self.client = client
        self.data_collection_topic                                          =
f'kp1/{APPLICATION_VERSION}/dcx/{ENDPOINT_TOKEN}/json/32'
```

```
        command_reboot_topic =
f'kp1/{APPLICATION_VERSION}/cex/{ENDPOINT_TOKEN}/command/reboot/status'
        self.client.message_callback_add(command_reboot_topic,
self.handle_reboot_command)
        self.command_reboot_result_topik =
f'kp1/{APPLICATION_VERSION}/cex/{ENDPOINT_TOKEN}/result/reboot'

        command_zero_topic =
f'kp1/{APPLICATION_VERSION}/cex/{ENDPOINT_TOKEN}/command/zero/status'
        self.client.message_callback_add(command_zero_topic, self.handle_zero_command)
        self.command_zero_result_topik =
f'kp1/{APPLICATION_VERSION}/cex/{ENDPOINT_TOKEN}/result/zero'

        self.SET_ZERO = False

    def connect_to_server(self):
        print(f'Connecting to Kaa server at {KPC_HOST}:{KPC_PORT} using application
version {APPLICATION_VERSION} and endpoint token {ENDPOINT_TOKEN}')
        self.client.connect(KPC_HOST, KPC_PORT, 60)
        print('Successfully connected')

    def disconnect_from_server(self):
        print(f'Disconnecting from Kaa server at {KPC_HOST}:{KPC_PORT}...')
        self.client.loop_stop()
        self.client.disconnect()
        print('Successfully disconnected')

    def handle_reboot_command(self, client, userdata, message):
        print(f'<--- Received "reboot" command on topic {message.topic} \nRebooting...')
        command_result = self.compose_command_result_payload(message)
        print(f'command result {command_result}')
        client.publish(topic=self.command_reboot_result_topik, payload=command_result)
        # With below approach we don't receive the command confirmation on the server side.
        # self.client.disconnect()
        # time.sleep(5)  # Simulate the reboot
        # self.connect_to_server()

    def handle_zero_command(self, client, userdata, message):
        print(f'<--- Received "zero" command on topic {message.topic} \nSending zero
values...')
        command_result = self.compose_command_result_payload(message)
        self.SET_ZERO = not self.SET_ZERO
        client.publish(topic=self.data_collection_topic, payload=self.compose_data_sample())
        client.publish(topic=self.command_zero_result_topik, payload=command_result)

    def compose_command_result_payload(self, message):
        command_payload = json.loads(str(message.payload.decode("utf-8")))
        print(f'command payload: {command_payload}')
        command_result_list = []
        for command in command_payload:
```

```
            commandResult = {"id": command['id'], "statusCode": 200, "reasonPhrase": "OK",
"payload": "Success"}
            command_result_list.append(commandResult)
        return json.dumps(
            command_result_list
        )

    def compose_data_sample(self):
        data = {}
        for i in range(20):
            if i == 1 and self.SET_ZERO:
                value = 0
            else:
                value = round(random.random()*10+10*i, 2)

            data[f"variable{i+1}"] = value
        return json.dumps(data)


def on_message(client, userdata, message):
    print(f'Message received: topic {message.topic}\nbody
{str(message.payload.decode("utf-8"))}')


def main():
    # Initiate server connection
    client = mqtt.Client(client_id=''.join(random.choice(string.ascii_uppercase + string.digits)
for _ in range(6)))

    data_collection_client = DataCollectionClient(client)
    data_collection_client.connect_to_server()

    client.on_message = on_message

    # Start the loop
    client.loop_start()

    # Send data samples in loop
    listener = SignalListener()
    while listener.keepRunning:

        payload = data_collection_client.compose_data_sample()

        result =
data_collection_client.client.publish(topic=data_collection_client.data_collection_topic,
payload=payload)
        if result.rc != 0:
            print('Server connection lost, attempting to reconnect')
            data_collection_client.connect_to_server()
        else:
```

```
        print(f'--> Sent message on topic "{data_collection_client.data_collection_topic}":\n{payload}')

        time.sleep(1)

    data_collection_client.disconnect_from_server()


class SignalListener:
    keepRunning = True

    def __init__(self):
        signal.signal(signal.SIGINT, self.stop)
        signal.signal(signal.SIGTERM, self.stop)

    def stop(self, signum, frame):
        print('Shutting down...')
        self.keepRunning = False

if __name__ == '__main__':
    main()
```