PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

FACULTAD DE CIENCIAS E INGENIERÍA

Evaluación y diseño de una Plataforma IoT con soporte de protocolo MQTT para un entorno de red empresarial

Tesis para obtener el Título Profesional de Ingeniero de las Telecomunicaciones

AUTOR:

VICMAR RAUL JIMENEZ CANAL

ASESOR:

JORGE BENAVIDES ASPIAZU

Lima, Diciembre, 2020

Resumen

La transformación digital ha traído consigo muchos beneficios y retos para las empresas; y pese a ello, aún se tiene muchos inconvenientes en adoptar un sistema centralizado para las tecnologías IoT.

Las plataformas IoT son sistemas solicitados por las empresas para unificar toda la labor de gestión y operativa relativa al IoT, sin embargo, las empresas para optar por un nuevo tipo de tecnología requieren de referencias de mercado como evaluaciones en ambientes de prueba, diseños validados o experiencias en la misma tecnología en otras empresas del mismo rubro. A pesar de esto, se carece de estas pruebas, evaluaciones y estándares en las plataformas IoT actuales del mercado por lo cual se dificulta su adopción.

Es así, como la presente tesis tiene como objetivo realizar la prueba, evaluación y diseño del rendimiento de una plataforma IoT empleando software de código libre. Esto se realizará mediante la implementación de todos los componentes en un centro de datos de prueba y como resultado, se podrá contrastar el beneficio de las distintas plataformas IoT en cada escenario establecido.

Dado que los escenarios propuestos son en base a plataformas IoT de terceros, se busca desarrollar un nuevo sistema que cubra las mismas capacidades, y por consiguiente, sea el inicio de futuros proyectos que se complementen con lo propuesto para las tecnologías IoT

Finalmente, obtenido toda la información y capacidades respecto a las plataformas, se realizó un análisis exhaustivo del porqué de cada parámetro recibido y como estos pueden beneficiar a los requerimientos empresariales de todo tamaño. Como resultado final, se propone un estudio de mercado en el cual la empresa analizara los costos de implementación para un tipo de escenario.

Indice

Contenido

Indice de figuras

Indice de tablas

INTRODUCCION

En la presente tesis se realizó la prueba, evaluación y diseño del rendimiento de una plataforma IoT empleando software de código libre. Estas plataformas son recientes en el mercado y debido a esto no se ha realizado evaluaciones de capacidades ni diferenciadores entre plataformas. Las evaluaciones son muy necesarias para que las empresas puedan adoptar nuevas tecnologías y escoger con base en diversos escenarios homologados. Por otro lado, la implementación de una plataforma IoT depende de muchos factores técnicos, por lo cual se propone un índice de rendimiento que calificará a estos y se obtendrá de manera teórica y práctica a través de los resultados obtenidos.

Para lograr los objetivos de esta tesis, se realizó un análisis del estado de las tecnologías actuales para establecer los parámetros a evaluar. Luego se elaboró un estudio para poder seleccionar las plataformas que serán evaluadas en el presente documento. Estas plataformas fueron implementadas en diversos servidores virtualizados mediante VMware con las capacidades asignadas para soportar las pruebas requeridas.

Posteriormente, se efectuaron las pruebas de rendimiento mediante el uso del software Jmeter a los parámetros establecidos de las plataformas IoT seleccionadas. Además, se creó una variable que evalúe el rendimiento de las plataformas comparándolas unas contra otras. Por último, se hizo el desarrollo y despliegue propio de una nueva plataforma IoT con la finalidad que sea usada en complemento a diversos proyectos e investigaciones de tecnología.

La solución obtenida y las pruebas realizadas demostraron que las plataformas evaluadas tienen un rendimiento muy variable y a partir de esto se puede escoger según requerimientos. Por otro lado, se busca que la plataforma desarrollada tenga el rendimiento más equilibrado y se pueda continuar su desarrollo en otros proyectos debido al uso de código libre.

1. SITUACIÓN ACTUAL DE LAS PLATAFORMAS IOT EN EL ENTORNO EMPRESARIAL

1.1 Identificación del ambiente actual

El internet de las cosas (IoT) no es algo de un futuro lejano sino una realidad que acompaña a las personas, compañías e industrias del mercado. El concepto de IoT lo experimentamos en diferentes dominios heterogéneos como un hogar inteligente, ciudades inteligentes, objetos de uso diario y sensores para la industria.[1]

Existe una gran heterogeneidad en el ambiente IoT debido a la presencia de distintos protocolos, fabricantes, desarrolladores, aplicaciones y servicios. Por ejemplo, en la figura 1 observamos que se requiere de muchos puntos intermedios (Middleware) para comunicar el dispositivo con aplicaciones y servicios. Además, se añade mayor complejidad debido a que no existe un

estándar y por ende los fabricantes tienen su propia manera de realizar la comunicación. Como resultado, las necesidades que son similares en concepto, pero distintas en ejecución e implementación se han distanciado más entre sí debido a que no existe una norma o arquitectura homologada. Además, los incentivos para ejecutar la interoperabilidad con terceros es nula dando como resultado un desafío mucho mayor a afrontar.[1]

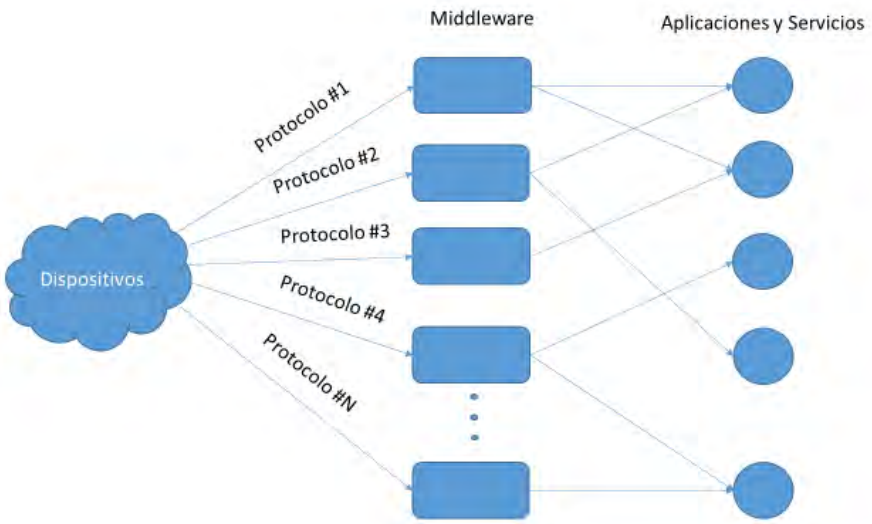

Figura 1: Arquitectura IoT de mensajería descentralizada

Fuente: Elaboración propia

El concepto clave para resolver esta descentralización y los desafíos de la heterogeneidad son las plataformas IoT, las cuales tienen un mercado en crecimiento del 33% para los siguientes 6 años. En el año 2021, se espera que el mercado de Plataformas IoT crezca a más de 1.6 Billones de dólares. [2]

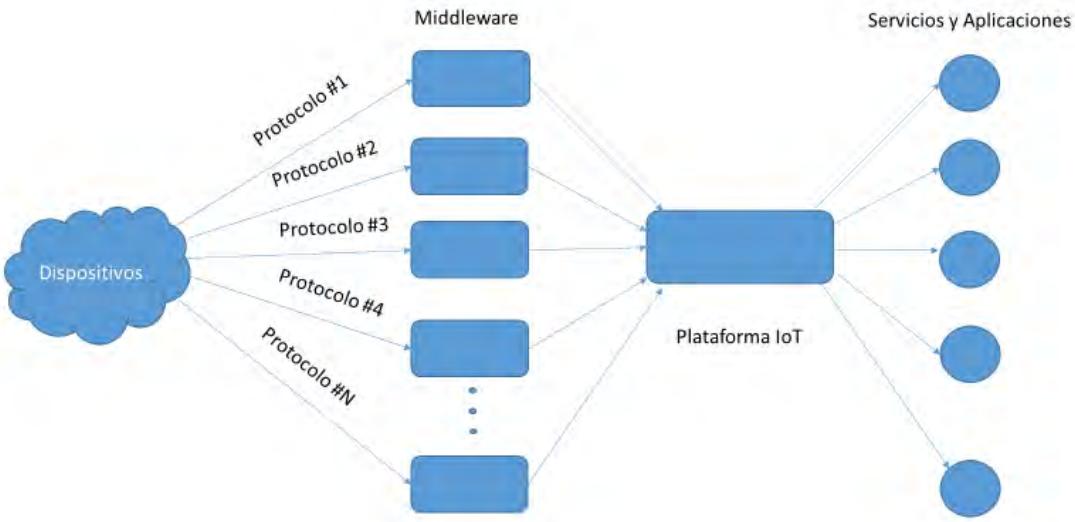

Figura 2: Arquitectura IoT con comunicación centralizada

Fuente: Elaboración propia

Estas plataformas IoT tienen como característica principal: [3]

- Módulos de gestión de información
- Módulos de gestión de dispositivos
- Módulos de interoperabilidad de protocolos
- Módulos de ejecución de tareas en grupo
- Módulos de analíticas
- Módulos de Comunicación centralizados.

Las características mencionadas de las plataformas IoT son la base para sobrellevar la interoperabilidad existente entre las diversas integraciones con dispositivos IoT. La gestión unificada y estructurada de los dispositivos IoT nos permitirá poseer mayores funciones que la suma de sus partes constituyentes por separado.[3]

1.2 Definición del problema

Las tecnologías IoT actuales poseen una gran heterogeneidad y poca interoperabilidad entre los componentes de hardware, software, redes y seguridad de los dispositivos. Esto da como resultado un ambiente de lenta adopción de la presente tecnología y gran reto de implementación, además, se añade una complejidad extra en países latinoamericanos debido a la brecha digital existente [4].

Las plataformas IoT actuales buscan cerrar esta brecha de interoperabilidad realizando una arquitectura unificada en todo su sistema compuesto. Esto permite realizar una gran variedad de acciones independientes al hardware que permiten agilizar la implementación, desarrollo, escalabilidad e identificación de errores [5]. Sin embargo, estas plataformas son recientes en el mercado y debido a esto no se ha realizado evaluaciones de capacidades ni diferenciadores entre plataformas las cuales son muy necesarias para que las empresas puedan adoptar tecnologías de una manera más pertinente. Por otro lado, la implementación de una plataforma IoT depende de muchos factores técnicos y comerciales, por ejemplo, el número de dispositivos a usar en el ambiente, proyección de crecimiento, protocolos de comunicación a usar, niveles de seguridad, base de datos, procesamiento de información y visualización de problemas de red [6]. A fin de que sea viable dar este siguiente salto tecnológico, se deberá poder evaluar y predecir el comportamiento de las plataformas emergentes del mercado el cual es el objetivo principal del presente estudio.

1.3 Hipótesis de la solución

En la presente tesis se evaluará el rendimiento de las plataformas disponibles en el mercado para poder crear un modelamiento base a partir de los datos obtenidos. Esto permitirá saber el rendimiento de una plataforma tercera respecto al promedio del mercado. Como propuesta adicional, a partir del

modelamiento realizado, se creará una nueva plataforma IoT de código libre para el desarrollo e investigación dentro del campus lo cual beneficiaria y originará futuras investigaciones.

Para realizar el diseño e implementación de una plataforma IoT se considerarán equipos que puedan actuar independiente al software y que cumplan con las funcionalidades básicas y protocolos requeridos en los ambientes propuestos. Estos dispositivos finales fueron simulados a través de generadores de carga. Sin embargo, estos estarán ligados con una plataforma IoT como un medio centralizado por el cual puedan ser gestionados y monitoreados de manera que actúen como sensor o actuador.

1.4 Alcance del proyecto

En la presente tesis se estableció un procedimiento para delimitar el rango de las pruebas a realizar. Estos procedimientos tienen como objetivo la prueba, evaluación y diseño de una plataforma IoT empleando software de código libre para resolver la problemática mencionada previamente.

La obtención de los datos implicó la implementación de varias plataformas IoT las cuales dependen de muchos factores técnicos, por lo cual se propone, además, un índice de rendimiento el cual se obtendrá de manera teórica y práctica a través de los resultados obtenidos con la finalidad de comparar cuantitativamente las plataformas. Finalmente, se diseñó una plataforma de código libre la cual estará bajo comparación con los parámetros obtenidos y así poder mejorar su rendimiento. Este diseño se limitó a la parte de red y lógica que incluye un Gateway de comunicación, una base de datos externa, automatización de tareas de recepción y envío de data. Este último diseño es el inicio de una plataforma de código libre que puede ser posteriormente complementada en futuros estudios.

2. ANÁLISIS DE REQUERIMIENTOS Y ARQUITECTURA PARA LA EVALUACIÓN Y DISEÑO DE UNA PLATAFORMA IOT

2.1 Componentes de las plataformas IoT y sus funcionalidades

Las plataformas IoT tienen requisitos funcionales básicos los cuales se diferencian uno de otros en sus capacidades y rendimiento a obtener frente a ciertos ambientes. Estas características básicas en las plataformas IoT son: [3]

- Registro y Gestión de dispositivos.
- Analíticas.
- Visualización.
- Protocolos de Seguridad.
- Interoperabilidad de protocolos de comunicación.

- Persistencia de información.
- Gestión de base de datos.

Estas características se consideran requerimientos funcionales básicos para una plataforma IoT y en algunos escenarios incluso indispensables. Por otro lado, en el diseño de las plataformas IoT se puede identificar componentes característicos que pueden ser representados por bloques, por ejemplo: [7]

- **Bloque de identificación**: Tiene como propósito la representación como objeto único a un dispositivo IoT mediante un ID único, por lo tanto, este puede ser diferenciado del resto.
- **Bloque de sensores**: Tiene como finalidad el poder ver las medidas del ambiente pertinente o poder realizar acciones con los actuadores.
- **Bloque de comunicación**: Se caracteriza por definir los protocolos de comunicación de los dispositivos IoT
- **Bloque de cómputo:** Tiene como responsabilidad de realizar el cómputo y procesamiento del IoT
- **Bloque de servicios**: Provee de todas las categorías de servicio provisionado por una plataforma IoT
- **Bloque semántico**: Es el encargado de saber cómo extraer información y cómo lidiar con esta de manera inteligente.

Sin embargo, para métodos teóricos se propone un nuevo modelo simplificado de cuatro bloques basado en lo propuesto según Hejazi [5] y Tsaryov [6] los cuales son:

- **Bloque de Sensores y Actuadores:** tiene como tarea la adquisición de data y actuar en caso sea necesario.
- **Bloque de Comunicación:** su tarea es que la data recolectada en el bloque anterior pueda ser enviada a través de la red de manera segura y de confianza
- **Bloque de Cómputo y almacenamiento:** La data recibida por parte de los sensores a través de la red es recolectada y procesada en este bloque
- **Bloque de Aplicaciones y Servicios:** luego del procesamiento, almacenamiento y análisis de data esta le muestra los resultados al usuario según peticiones solicitadas.

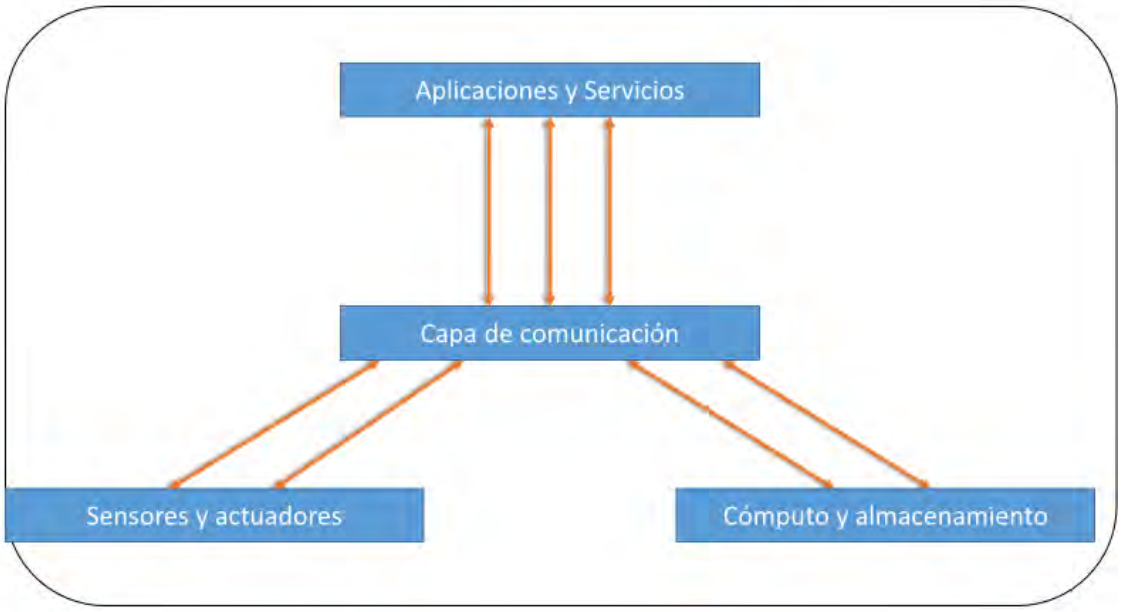

Figura 3: Diseño de componentes de una plataforma IoT

Fuente: Elaboración propia

Estos bloques de componentes permiten identificar la función específica que puede tener cada parte de la arquitectura en una plataforma IoT y de esta manera poder tener una implementación modular según requerimiento.

2.2 Tipos de arquitecturas IoT

A pesar que las plataformas IoT están ganando popularidad en el terreno tecnológico, aún no existe una clara definición de arquitecturas validadas para los componentes que la gobiernan o arquitecturas basadas en la aplicación [8]. En las arquitecturas basadas en la aplicación se puede encontrar dos tipos de servicio: las que están basadas por un único servicio o las que son multiservicio como se puede apreciar en la figura 4. Se denomina enfoque vertical a las arquitecturas basadas en un único servicio o aplicación los cuales son óptimos debido a que cada parte posee sus recursos propios dando ciertos beneficios en algunos ambientes, por otro lado, se denomina enfoque horizontal en la figura 4 a las arquitecturas multiservicio las cuales comparten bloques comunes entre los

servicios generando una reutilización que puede llevar a un óptimo consumo. Sin embargo, la elección entre uno y otro dependerá del requerimiento, además, esto afectara el diseño físico de la arquitectura a implementar [9].

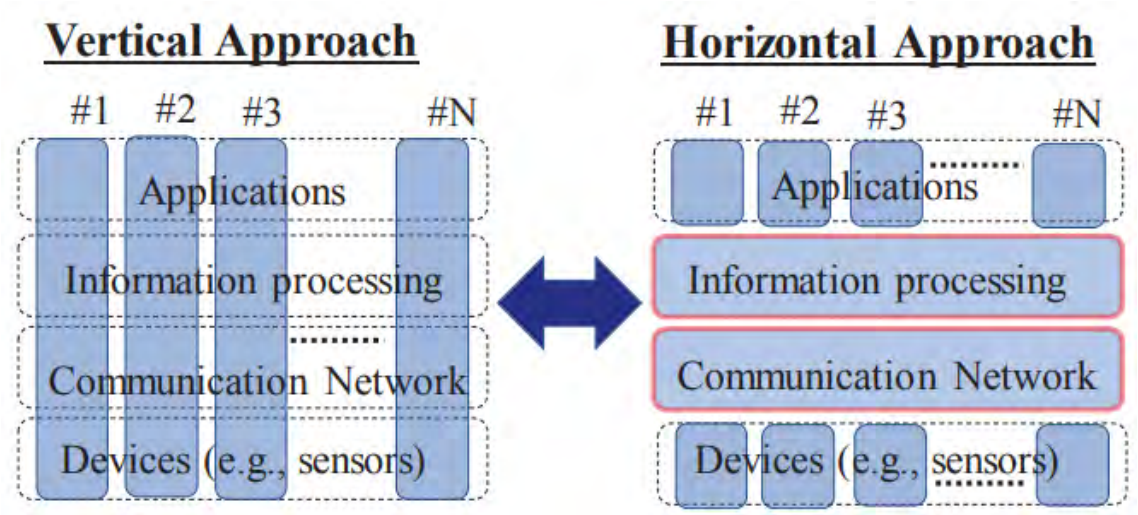

Figura 4: Arquitectura IoT vertical y horizontal

Fuente: [10]

2.3 Tipos de plataformas IoT

Actualmente, se encuentran distintos modelos de plataformas IoT en el mercado basadas en nube o despliegues locales lo cual da como resultado distintos tipos de análisis respecto cual es más beneficioso a cada aplicación. Por ejemplo, la plataforma que usa la marca LG está basada en la siguiente arquitectura multiservicio alojada en la nube de AWS IoT Core:

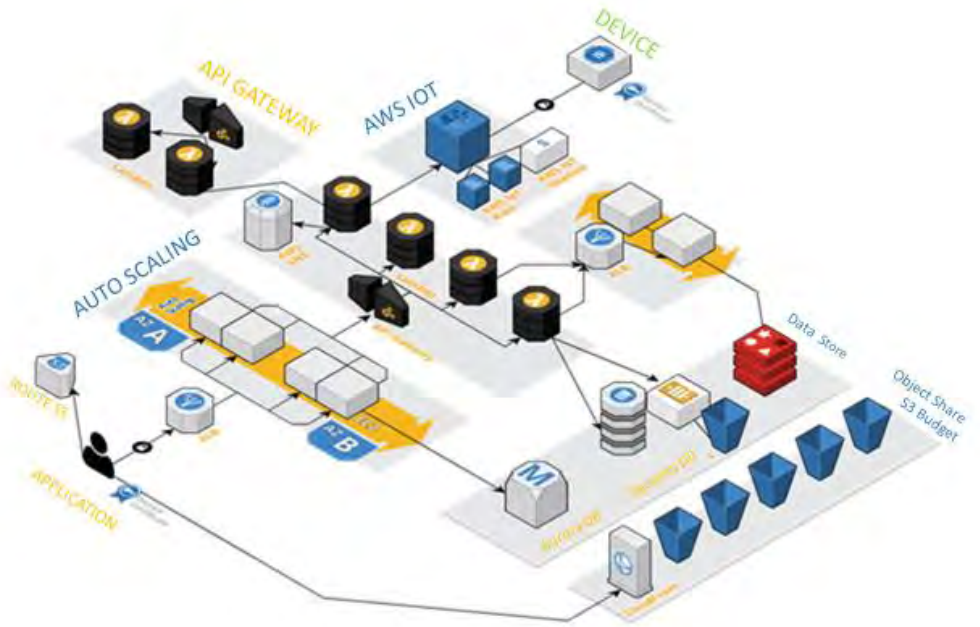

Figura 5: Arquitectura de LG basada en Cloud-AWS

Fuente: [11]

Podemos observar en la figura 5 que se mantienen los bloques de analíticas, procesamiento, almacenamiento, comunicación y bases de datos. Los bloques previamente mencionados se visualizan finalmente bajo la plataforma ThinQ de LG.

Por otro lado, a pesar que AWS IoT Core soporta distintos protocolos de comunicación como HTTP, Websocket y MQTT [12], LG se basó en el protocolo UDAP(Universal Discovery & Access Protocol) [13] el cual está basado en HTTP(HyperText Transfer Protocol) y esto permite la comunicación de las diversas gamas de producto que ofrece con su plataforma en la nube. Sin embargo, esto podría verse beneficiado en costos al usar un protocolo más ligero en ancho de banda y almacenamiento como MQTT [14] , pero el cambio llevaría a una reevaluación de los componentes en toda su gama de productos por lo cual habría nuevos retos económicos y técnicos a desarrollar a posteriori.

En segundo lugar, para plataformas de despliegue local usaremos de ejemplo a la plataforma de código abierto Mainflux. Respecto a esta plataforma podemos observar en la figura 6 que tiene distintos componentes en su arquitectura lógica

y física. Sin embargo, incluso a pesar de ser distinto se mantienen los bloques esenciales para un correcto diseño de una plataforma IoT.

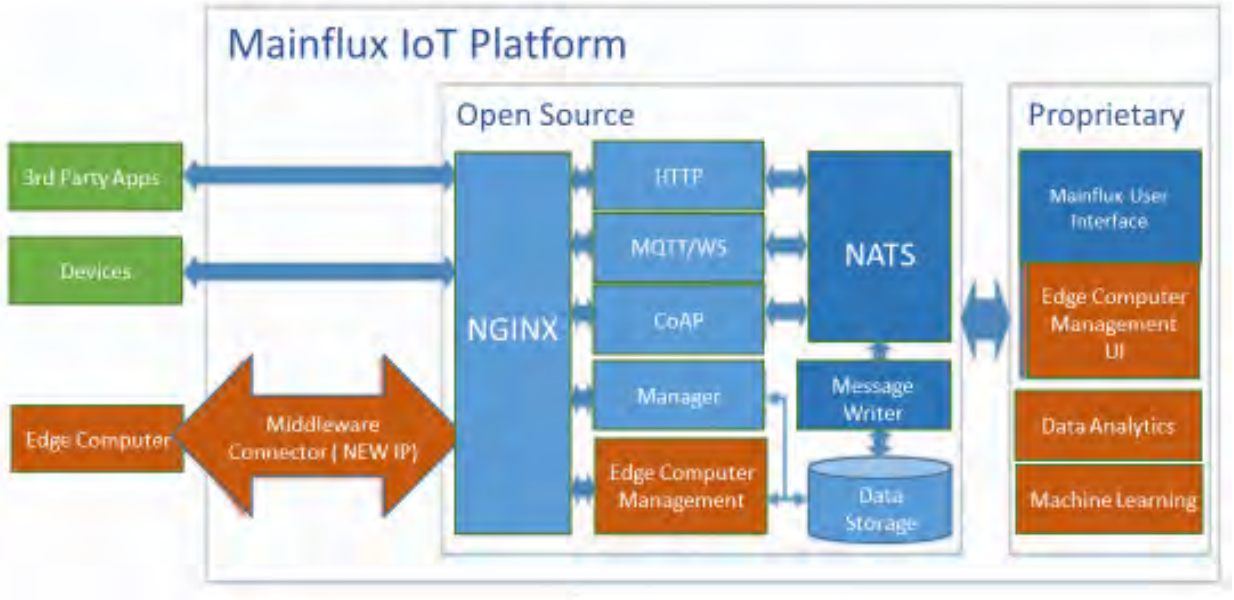

Figura 6: Componentes de plataforma IoT – Mainflux

Fuente: *[3]*

Estos bloques esenciales se mantienen como patrón común entre las distintas plataformas privadas o públicas en el mercado. Sin embargo, cada una tiene su manera de comunicar los bloques dentro de su arquitectura dando como resultado cierto nivel de complejidad al decidir cuál es la correcta para cierto tipo de servicio o aplicación.

2.4 Análisis para la selección de plataformas IoT

Como podemos observar en la figura 6, existen diversas características básicas que se analizaron en la elección de las plataformas IoT. Estas características básicas son parte del criterio de diseño y se consideran mínimas e indispensables en la mayoría de aplicaciones, sin embargo, entre ellas aun así existe una jerarquía de prioridades la cual se analizará a profundo en el presente estudio.

Entre las características basadas en [3] se ahondará en el presente capitulo en los protocolos de seguridad y comunicación debido a su criticidad y mayor jerarquía en la toma decisión de una plataforma. Por ejemplo, entre los protocolos de seguridad se puede optar por alguna que cumpla un estándar mínimo del mercado o requerido dentro de la arquitectura solicitada. Este tipo de decisiones conlleva una gran importancia al ejecutar el diseño, debido a que a partir de éste también se puede designar cierto nivel de servicio para aplicaciones de negocio y no solo influir en la arquitectura lógica y física del aplicativo como software en sí.

2.4.1 Análisis de seguridad

La seguridad se considera muy crítica en una arquitectura IoT debido a que incluye a las plataformas, dispositivos y canales de comunicación IoT. Actualmente, se aproxima que existen 7000 millones de dispositivos conectados a Internet en una variedad de gama en servicios , por ejemplo, industria, consumo masivo, infraestructura, logística, conectividad, etc. [15]. Sin embargo, muy a pesar del alentador número y el crecimiento esperado, todos y cada uno de estos dispositivos, incluso, la plataforma en sí, son puntos vulnerables a ataques. Estos ataques, en su mayoría maliciosos, buscan robar información, vulnerar la integridad de las personas o hacerlas partes de una botnet [16].

La vulneración de la seguridad en un ambiente IoT es un tema controversial y el cual es aprovechado debido a que no existe una arquitectura estandarizada. Por lo tanto, existen varios tipos de ataques penetrando por las diferentes capas definidas anteriormente en la figura 3. Por un lado, en el bloque de sensores, según lo afirmado en [17] [18] existen ataques de tipo:

Tabla 1: Tipos de ataque en el bloque de sensores

Tipo de Ataque	Descripción
Intrusión forzada de nodo	Cuando el atacante inserta un falso dispositivo entre los dispositivos actualmente colocados y logra obtener información.
Inyección de código malicioso	El atacante inyecta código malicioso para poder vulnerar los accesos provisto desde un dispositivo final.
Reemplazo de Hardware	El atacante realiza cambios a los componentes electrónicos del sensor o actuador de manera que puede generar una brecha para robar llaves de encriptación y comunicación

Fuente: Elaboración Propia

En el bloque de red y comunicaciones encontramos los siguientes ataques basados en [17] [19]:

Tabla 2: Tipos de ataque en el bloque de red

Tipo de Ataque	Descripción
Denegación de servicios (DoS)	Los ataques DoS lo utilizan para inutilizar el servicio de servidores o dispositivos.
Hombre en el medio (Man-in-Midlle)	A diferencia de un nodo falso o sobreescritura sobre un nodo, en este tipo de ataque se finge ser el server colector por lo cual se recibe la información sin ningún tipo de restricción.
Ataque al Gateway	El atacante tiene como objetivo restringir la conexión intermediara entre el servidor y el nodo mediante diferentes técnicas, dando como resultado la degradación del sistema.

Fuente: Elaboración Propia

En el bloque de procesamiento y almacenamiento encontramos los siguientes ataques basados en [17]:

Tabla 3: Tipos de ataque en el bloque de procesamiento y almacenamiento

Tipo de Ataque	Descripción
Recursos compartidos	Al acceder por puertos abiertos entre máquinas virtuales con recursos compartidos esto permitiría extracción de información confidencial
Ataque a máquinas virtuales	El ataque a una máquina virtual desde el mismo servidor físico con alguna vulnerabilidad en el arranque o en el código. Asimismo, insertar código en el arranque de la máquina virtual
Seguridad en la nube como infraestructura	La data almacenada y guardada como respaldo se vulnera en el proveedor mediante la ruptura de accesos o vulneración de la plataforma nube dando la disponibilidad de exfiltración de la información.

Fuente: Elaboración Propia

De los tipos de ataques mencionados previamente, hay una serie de medidas para proteger cada capa y así poder llegar a mitigar en su mayoría los ataques indebidos.

Tabla 4: Medidas de protección y prevención ante ataques

Medidas de Protección	Descripción
Uso de Firewall	Autenticación por dispositivo (AAA) y llaves de encriptación [20]
Arranque seguro	Arranque seguro desde los dispositivos IoT, servidores físicos y virtuales
Integridad de data	El uso de cheksum, bit de paridad, SHA-256, TLS/SSL y HTTPS
DNS seguro	El sistema de nombres de dominio (DNS) tiene alternativas seguras como por ejemplo TALOS, esto permitirá que la primera brecha de seguridad que son las peticiones este cubierta[21]
Suplantación de identidad	El correcto uso de protocolos de seguridad tanto física como lógica en los switches de acceso de toda la arquitectura

Fuente: Elaboración Propia

Como hemos observado previamente, existe una variedad inmensa de flancos débiles por parte de la arquitectura a nivel de seguridad, sin embargo, existen maneras de mitigar estas vulnerabilidades. Por otro lado, se debe considerar que al habilitar más funciones de seguridad al sensor o actuador este tendrá un mayor consumo energético y también de memoria, lo cual al día de hoy aún son grandes limitantes [21]. En las consideraciones a tomar para el diseño se debe tratar que el procesamiento, almacenamiento y comunicación de llaves de seguridad sean fuera del dispositivo para que este no sea sobrecargado en los pocos recursos que tiene al día de hoy.

2.4.2 Análisis de protocolos de comunicación

Los protocolos de comunicación son una característica crucial en las plataformas IoT. Según el uso de un protocolo se puede determinar el consumo en ancho de banda, energía y tipo de lógica en la implementación de la plataforma IoT. En el caso particular de los dispositivos IoT, sabemos que estos al día de hoy poseen recursos de cómputo y poder muy escasos. Por lo tanto, entre los protocolos a considerar se debe buscar los que sean más compatibles con lo descrito previamente. Es decir, de bajo consumo de energía y ancho de banda. Entre los protocolos de comunicación más conocidos y usados encontramos Constrained Application Protocol (CoAP) , HTTP, Message Queuing Telemetry Transport (MQTT),Extensible Messanging and Presence Protocol (XMPP) [14]. Debido a estas alternativas debemos analizar cuál es la más ventajosa para la comunicación entre dispositivos finales e intermediados.

2.4.2.1 HTTP

En primera lugar tenemos a HTTP el protocolo de comunicación de mayor uso en los últimos años [22]. El protocolo HTTP se basa en el uso de UDP/TCP en una lógica de cliente servidor para la transmisión de data. Sin embargo, cuando HTTP es usado en IoT surgen ciertos inconvenientes debido a la falta de autonomía y escasos recursos de cómputo en los sensores. En la figura 9 observamos que en ambientes IoT existe una mayor cantidad de paquetes, pero de tamaño reducido. Al existir un gran número de paquetes a enviar y tener una cabecera muy grande, una cabecera de 137 bytes [23], como lo tiene HTTP, provocaría un menor rendimiento y a su vez una pobre optimización de recursos al usarlos en dispositivos de recursos limitados como dispositivos finales.

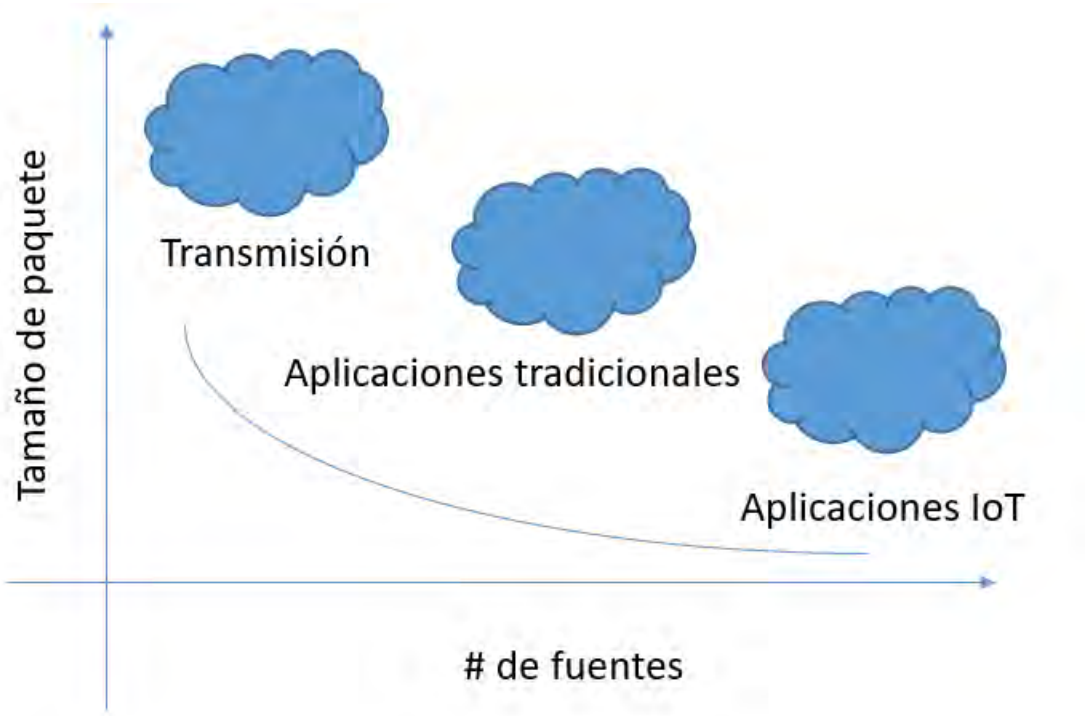

Figura 7: Tamaño de paquetes vs número de paquetes

Fuente: [10]

El protocolo HTTP no suele ser el más apropiado para este tipo de aplicación, como es señalado en [24]. HTTP es un protocolo centrado en documentos y no en datos por lo cual en ambientes basados en IoT no se puede aprovechar mejor su uso.

2.4.2.2 MQTT

En un ambiente donde los recursos son muy limitados como en IoT se necesita de todas las herramientas para aprovecharlo al máximo, por ello, para poder mejorar el uso de recursos es que existen los protocolos denominados ligeros. Estos protocolos se ejecutan sobre TCP/UDP, incluso, sobre HTTP. Por ejemplo, MQTT y CoAP son protocolos ligeros y debido a su tipo de patrón de

comunicación como observamos en figura 10 es que se puede obtener una mejor eficiencia que HTTP en ambientes IoT. Desde el año 2018 el modelo subscriptor/publicador que se observa en la figura 10 es el más usado según lo descrito en [23] desplazando a HTTP y CoAP. El beneficio que aportan estos tipos de protocolos es el uso de los recursos limitados de procesamiento, ancho de banda y energía en los dispositivos IoT, lo cual se demuestra ampliamente en diversos estudios [14], [23], [24].

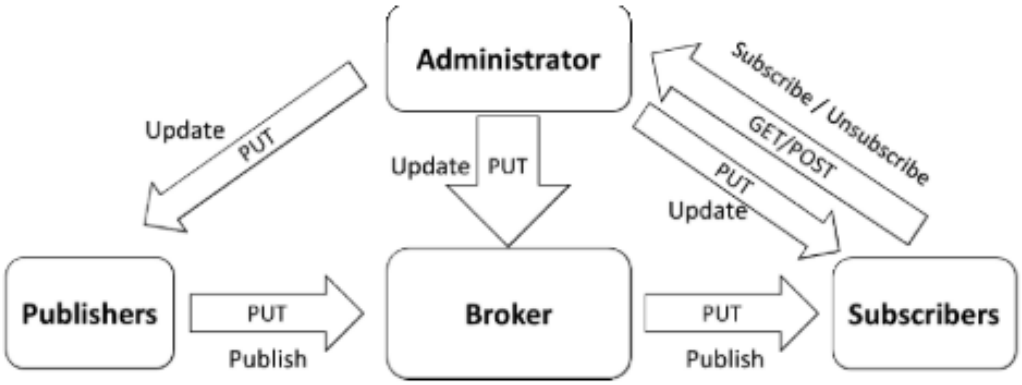

Figura 8: Modelo subscritor/publicador para MQTT

Fuente: [25]

Entre los protocolos ligeros, se prefiere el uso MQTT debido a su simplicidad y efectividad al estar basado en el modelo subscriptor/publicador, además, posee una cabecera de 2 bytes [24] , lo cual comparado contra HTTP es mucho menor que sus 137 bytes.

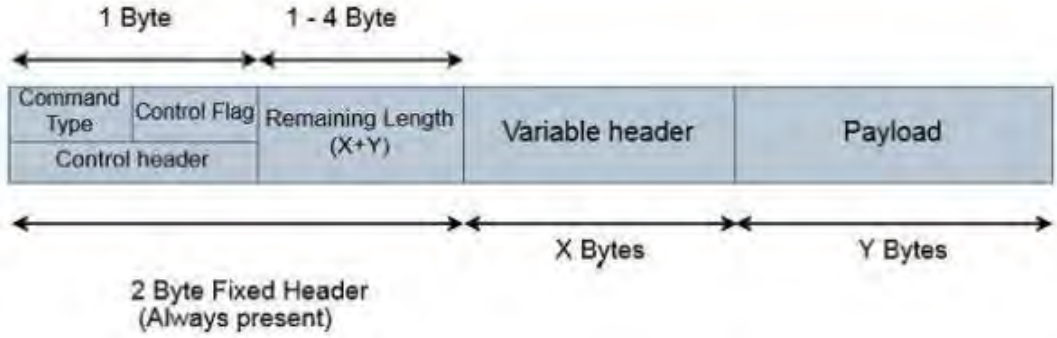

Figura 9: Segmento de paquete MQTT

Fuente: [26]

Como resultado, se obtiene un mejor rendimiento en ancho de banda y energía provistos por MQTT tanto por su patrón de comunicación como por el tamaño de cabecera [14].

Por otro lado, basados en [25][27] pudimos observar que CoAP tiene un mejor rendimiento tanto en RTT como en ancho de banda. Sin embargo, se sugiere el uso combinado de ambos protocolos MQTT y CoAP o el uso individual de MQTT para un mejor rendimiento. Esto es debido a que el modelo publicador/subscriptor es más adecuado para los ambientes IoT; específicamente se sugiere MQTT para su uso en sensores y CoAP se sugiere para comunicar dispositivos más robustos como los servidores debido a que pueden soportar sin ningún problema al protocolo HTTP [23].

Basado en las características previamente descritas, para el presente estudio se escogió el uso de MQTT en lugar de HTTP y CoAP debido a las características previamente mencionadas donde posee del mejor rendimiento en estos tipos de ambiente IoT.

2.5 Análisis y elección de plataformas actuales

El momento que una empresa decide adquirir software o hardware nuevo es todo un reto e incluso involucra un esfuerzo mayor si la tecnología aun esta en desarrollo. Para este caso, las diferentes plataformas IoT existentes en el mercado con diferentes características y funcionalidades especificas dificultan elegir o diferenciarlas entre sí debido a la poca madurez de la tecnología. Se puede considerar en la evaluación el tipo de hardware homologado, protocolos, prácticas de seguridad, visualización de data, integralidad con software de terceros, escalabilidad, estabilidad y el modelo de compra o soporte [5].

Se puede encontrar más de 450 plataformas IoT al día de hoy [28], todas estas alternativas se encuentran en constante cambio innovando funcionalidades, añadiendo nuevos conceptos en la búsqueda de ser la Plataforma líder del

mercado. Sin embargo, para poder evaluarlas se tiene que optar por ciertas características que se consideran el requerimiento básico de funcionalidad. Por ejemplo, para el presente estudio se realizó una comparativa entre plataformas con características de gestión de dispositivo, soporte de SDK, seguridad, protocolos de comunicación y recolección de data, analíticas, soporte de visualización y base de datos, esto se encontrará presente en el capítulo 3 del presente estudio.

Se realizó un estudio profundo respecto a 10 plataformas IoT como se puede ver en la tabla 5, evaluando dichas plataformas bajo requerimientos mínimos para una correcta funcionalidad basadas en [3]. Se ha tomado una muestra de 10 plataformas debido a la limitación de recursos, sin embargo, las plataformas escogidas nos proveerán de datos suficientes para saber su comportamiento a distintos niveles de cargas. Estos datos nos permitirán tener un concepto más preciso al evaluar las plataformas IoT para un diseño de red.

Tabla 5: Plataformas y servicios evaluados

IoT Software Platform	Device managment	Supported SDK	Security	Protocols for data collection	Analytics	DB
Kaa IoT Platform	Yes	Portable SDK REST API	Link Encryption (SSL), RSA key 2048 bits, AES key 256 bits	MQTT, CoAP, XMPP, TCP, HTTP	Real time IoT Data Analytics and Visualization with Kaa, Apache Cassandra and Apache Zappelin	MongoDB, Cassandra, Hadoop, Oracle NoSQL
SiteWhere	Yes	REST API	Link Encryption (SSL), Spring Security	MQTT, AMQP, Stomp, WebSockets, and direct socket connections	Real-time analytics (Apache Spark)	MongoDB, HBase , InfluxDB
ThingSpeak	No	REST and MQTT APIs	Basic Authentication	HTTP	MATLAB Analytics	MySQL
DeviceHive	No	REST AP, MQTT APIs	Basic Authentication using JSON Web Tokens (JWT)	REST API, WebSockets or MQTT	Real-time analytics (Apache Spark)	PostgreSQL ,SAP Hana DB
Zetta	No	REST APIs	Basic Authentication	HTTP	Using Splunk	Unknown
Distributed Services Architecture (DSA)	No	REST APIs	Basic Authentication	HTTP	No	ETSDB – Embedded Time Series
Thingsboard.io	Yes	REST APIs	Basic Authentication	MQTT, CoAP, and HTTP	Real time analytics(Apache Spark, Kafka)	Cassandra
Thinger.io	Yes	REST APIs	Link Encryption (SSL/TLS) and basic authentication	MQTT, CoAP, and HTTP	Yes	MongodB
WSo2	Yes	REST APIs	Link Encryption (SSL) and basic authentication	HTTP, WSO2 ESB, MQTT	Yes, WSO2 Data Analytics Server	Oracle, PostgreSQL, MySQL, or MS SQL
Mainflux	Yes	REST APIs	OAuth2.0, public key infrastructure (PKI) and client-side certificates	HTTP, MQTT, WebSocket, CoAP)	Yes (integrated) Platform not confirmed	Cassandra, MongoDB or InfluxDB or PostgreSQL

Fuente: Elaboración Propia

Entre las plataformas escogidas se ha enfatizado el uso de software de código libre. Se escogió esto debido a la maleabilidad e integralidad que los sistemas abiertos permiten al añadir, cambiar o mejorar características de la plataforma, lo cual es muy beneficioso para una tecnología que aún continúa desarrollándose. Por otro lado, muy a pesar de ser una inversión de tiempo y recursos la adopción de una plataforma abierta, permite una mayor aceleración para la adopción de la tecnología contribuyendo diferentes beneficios para la empresa [29].

Entre las plataformas escogidas se evaluarán los siguientes servicios:

Gestión de dispositivo: Provee especificaciones del dispositivo, grupo de dispositivos, reglas por grupo, administración de reglas y dispositivos en GUI.

Soporte de SDK: API, REST API, MQTT API

Seguridad: Soporte de protocolos de seguridad como SSL, RSA, AES, TLS o algún método de encriptación

Protocolos de comunicación: Soporte de protocolos de comunicación como MQTT, CoAP, XMPP, TCP, HTTP, AMQP.

Analíticas: análisis de data a tempo real o soporte de integración con terceros

Base de Datos: soporte de base datos No SQL, SQL propias o de terceros para el almacenamiento de la data [3]

A partir de estas características se escogió 4 de las 10 plataformas, como se puede observar en la tabla 6, a partir de esto se procederán a evaluar sus métricas de rendimiento en el capítulo 3 de la presenta tesis.

Tabla 6: Análisis de Plataformas elegidas

IoT Software Platform	Device managment	Supported SDK	Security	Protocols for data collection	Analytics	DB
Kaa IoT Platform	Yes	Portable SDK REST API	Link Encryption (SSL), RSA key 2048 bits, AES key 256 bits	MQTT, CoAP, XMPP, TCP, HTTP	Real time IoT Data Analytics and Visualization with Kaa, Apache Cassandra and Apache Zappelin	MongoDB, Cassandra, Hadoop, Oracle NoSQL
SiteWhere	Yes	REST API	Link Encryption (SSL), Spring Security	MQTT, AMQP, Stomp, WebSockets, and direct socket connections	Real-time analytics (Apache Spark)	MongoDB, HBase , InfluxDB
Thingsboard.io	Yes	REST APIs	Basic Authentication	MQTT, CoAP, and HTTP	Real time analytics(Apache Spark, Kafka)	Cassandra
Mainflux	Yes	REST APIs	OAuth2.0, public key infrastructure (PKI) and client-side certificates	HTTP, MQTT, WebSocket, CoAP)	Yes (integrated) Platform not confirmed	Cassandra, MongoDB or InfluxDB or PostgreSQL

Fuente: Elaboración propia

2.6 Mercado Actual

Según estadísticas se afirman que hay más de 30 plataformas IoT usadas por diferentes compañías, y el número sigue creciendo a medida que el tiempo pasa [9][10]. Las plataformas han sido desarrolladas por Startups o grandes empresas, por ejemplo, IBM. Todas estas empresas buscan ser el líder en ventas de plataformas IoT en el mercado [5][8]. Sin embargo, para este tipo de implementaciones se suele evaluar 2 opciones: plataformas propietarias y plataformas código libre. En el estudio se compara el software código libre versus las propietarias mediante las variables usadas en [29] : beneficio de las plataformas, cuotas entrantes, los beneficios de la aplicación, costo por usuario, costo de plataforma, máxima demanda, precio de aplicación, costo cruzado con efecto en red, función de demanda y plataforma con venta plana, como resultado,

se puede apreciar que hay mejores escenarios para la plataforma de código abierto dando un mejor rendimiento costo/beneficio/tiempo que su alternativa contraria la plataforma propietaria. Sin embargo, existe el caso contrario cumpliendo ciertas condiciones descritas en [29] como descuentos por volumen o algún tipo de arrendamiento a escala.

Lo que se busca en general para ambos casos, propietario o código libre, es siempre que las plataformas den un mayor ingreso que los costos globales requeridos por su mantenimiento. Un caso de análisis es la plataforma de LG. LG usa la marca ThinQ para categorizar a todos sus productos y servicios inteligentes, sin embargo, al ser este de un crecimiento no controlado la plataforma debe poder amoldarse a este cambio sin requerir tanto costo en despliegue y operaciones. Por lo tanto, realizó la migración de sus servidores a la nube lo cual según el reporte asignado en [11] se pudo observar que hubo un 80% de ahorro en costos de despliegues y esto permitió una mejora en el equipo de desarrollo debido a que se podían enfocar en programar la lógica de negocio para los servicios de LG según se menciona en [10]. Como resultado, hubo un crecimiento sugerido por el uso de la gestión inteligente de sus dispositivos en las diversas gamas dando a entender que el costo de la plataforma por dispositivo no excede al beneficio adquirido por dispositivo.

3. ANÁLISIS DE LAS PLATAFORMAS IOT

En las secciones previas se presentaron la problemática y los conceptos base para el análisis de rendimiento de las plataformas IoT. Según la cantidad de recursos físicos disponibles en el momento y las características designadas, se implementó, evaluó y analizo 4 de las 10 plataformas mencionadas en el capítulo previo.

La elección de estas 4 plataformas a desplegar y evaluar tuvo como alternativas de infraestructura servidores locales y servidores nube. Sin embargo, por costos y disponibilidad de recursos se escogió el despliegue local para evitar la saturación y sobrecostos por las pruebas de alto rendimiento hacia la nube, además, de posibles bloqueos de IP y retrasos en el RTT (round trip time). Se uso un switch Catalyst 3850, 2 switch Nexus 3048 y servidor UCS c220 M5 el cual se puede observar su diagrama físico, lógico y capacidades en la figura 10, 11 y tabla 7 respectivamente. Sobre esta infraestructura se realizó el despliegue y evaluación de los simuladores de carga y las 4 plataformas seleccionadas

previamente: Mainflux, Thingsboard, Kaa, Sitewhere.

Se observa en las siguientes imágenes los diagramas lógicos y físicos usados para el ambiente de prueba.

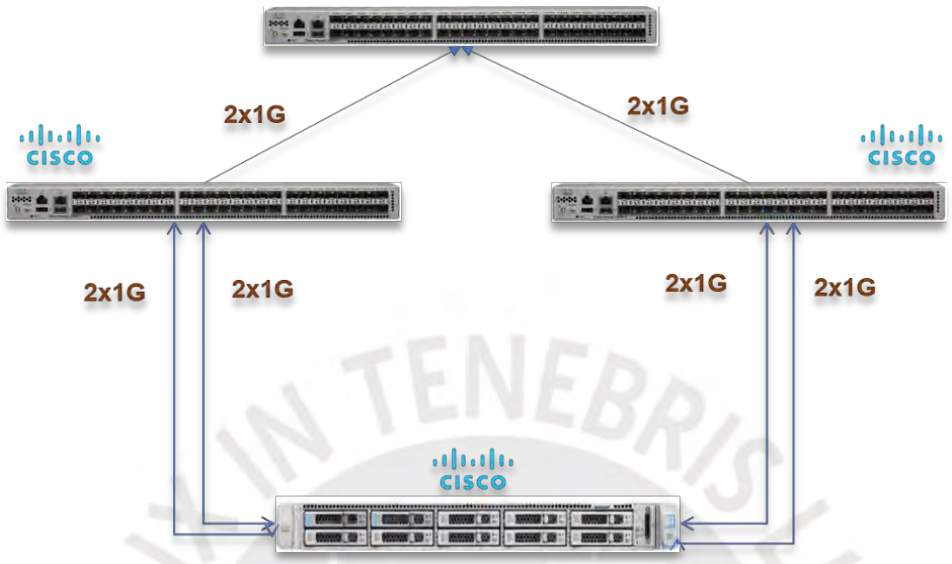

Figura 10: Modelo subscriptor/publicador para MQTT

Fuente: Elaboración Propia

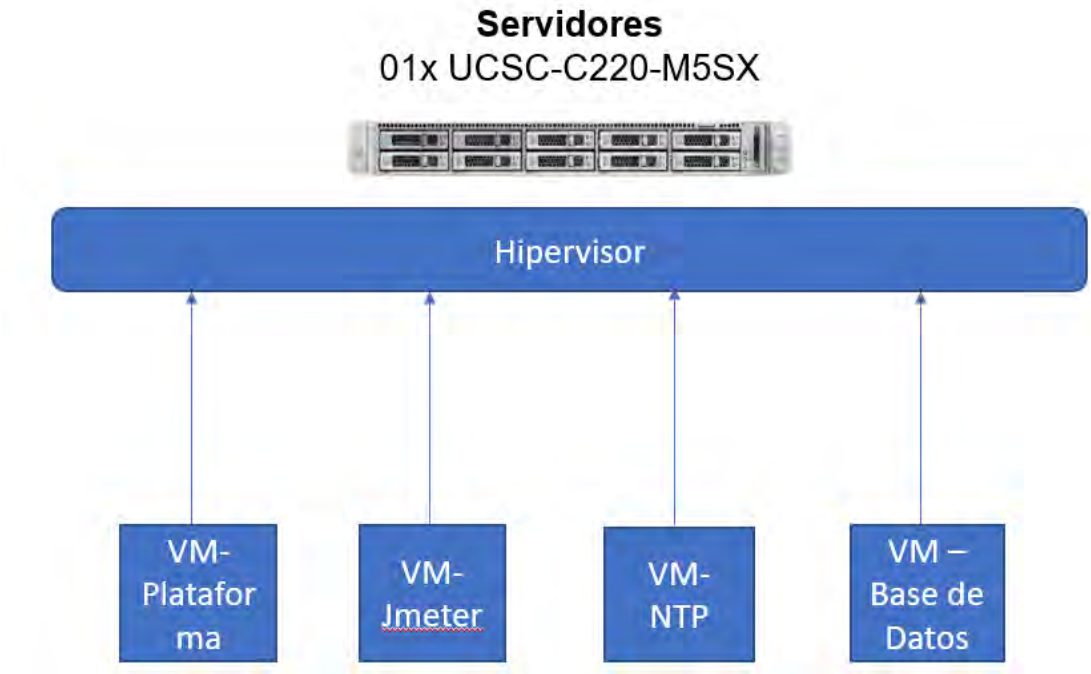

Figura 11: Modelo subscriptor/publicador para MQTT

Fuente: Elaboración Propia

Tabla 7: Recursos de cómputo locales

Servidor	UCS-C220-M5
CPU	1x Intel 4114 (8C, 2.2GHz,13,75M)
MEMORIA	128GB
HIPERVISOR	VMWARE 6.5
DISK	500 Gb SSD 6GB/s 2,5 “RAW

Fuente: Elaboración Propia

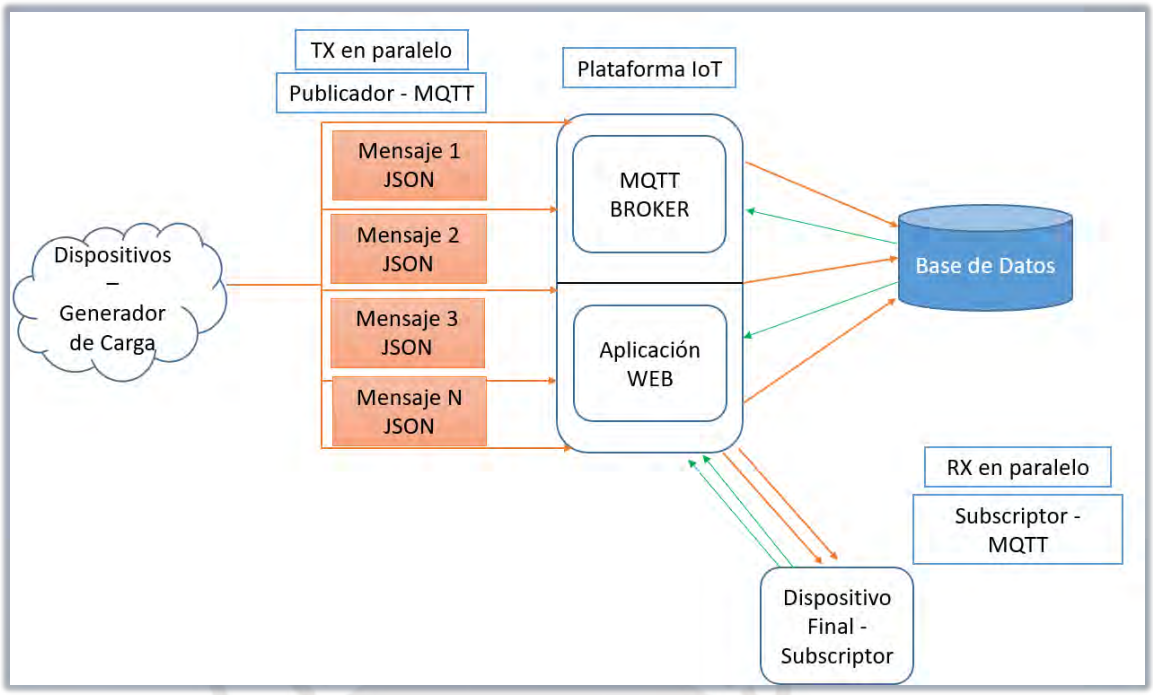

Figura 12: Diagrama lógico del ambiente de prueba

Fuente: Elaboración propia

3.1 Simuladores de carga

Para generar el tráfico proveniente de los dispositivos IoT se buscó software especializado de simulación de carga para tener distintos escenarios de prueba. Se busco software con la capacidad de simular la comunicación a través del protocolo MQTT y que este pueda enviar un paquete a una plataforma. Entre las plataformas que soportan MQTT encontramos a MqttBox, MqttFx, Lottus y Jmeter. El siguiente paso fue buscar que estos simuladores tengan la capacidad de procesar con paralelismo de multi hilo, como resultado, solo se optó por usar Jmeter debido a la flexibilidad de funciones al generar el estrés al servidor con cargas paralelas. El simulador Jmeter está basado sobre una arquitectura multi hilo por lo cual le permite realizar las tareas de paralelismo con los mensajes MQTT.

El simulador Jmeter tiene como base lógica un archivo raíz que se le denomina el test plan. Este *test plan* es el que incluye toda la lógica y pasos a realizar dentro

del margen configurado. Dentro de esto podremos encontrar protocolos, tipo de respuestas, tiempos, bucles, tipo de mensajes, etc. [30]

Dentro del *test plan*, Jmeter nos permite generar grupos de hilos los cuales permiten realizar el paralelismo en él envió de mensajes. Este grupo de hilos permite generar una concurrencia simultanea tanto dentro de un mismo grupo de hilos o simultáneamente entre grupos de hilos [31]. Para el presente estudio se usó solo un grupo de hilos y dentro de éste una instancia para la transmisión de mensajes en paralelo. Configurando los parámetros, el grupo de hilos nos permite simular el número de mensajes y el contador de bucle son el número de instancias de mensajes en paralelo con un periodo definido de subida en 1 segundo, según indicado en la figura 13.

Dentro del grupo de hilos se puede definir qué acción realizar a partir de la concurrencia y la instancias, por ejemplo, la acción de simular el publicador de mensajes MQTT. Dentro del publicador MQTT encontramos la IP del bróker de la plataforma IoT, el *topic* al cual se publicará y el mensaje con su contenido. Cuando se ejecuta el Jmeter seguirá el flujo mostrado de la figura 12, en este proceso la plataforma IoT previamente se subscribe al mismo *topic* publicado debido al tipo de funcionamiento del patrón publicador/subscriptor.

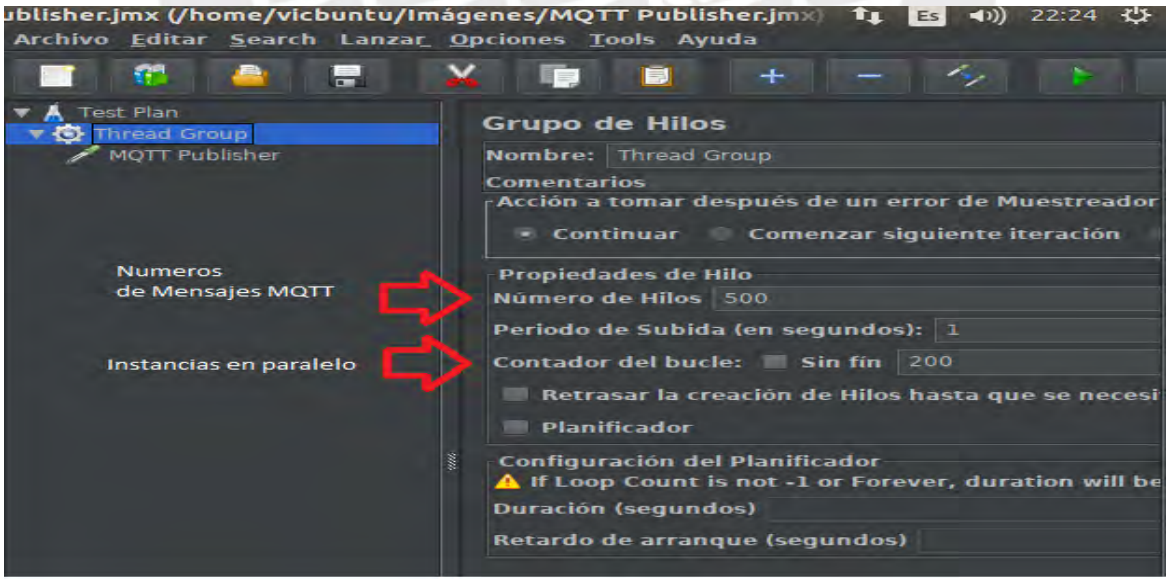

Figura 13: Interfaz gráfica Jmeter – Thread Group

Fuente: Elaboración propia

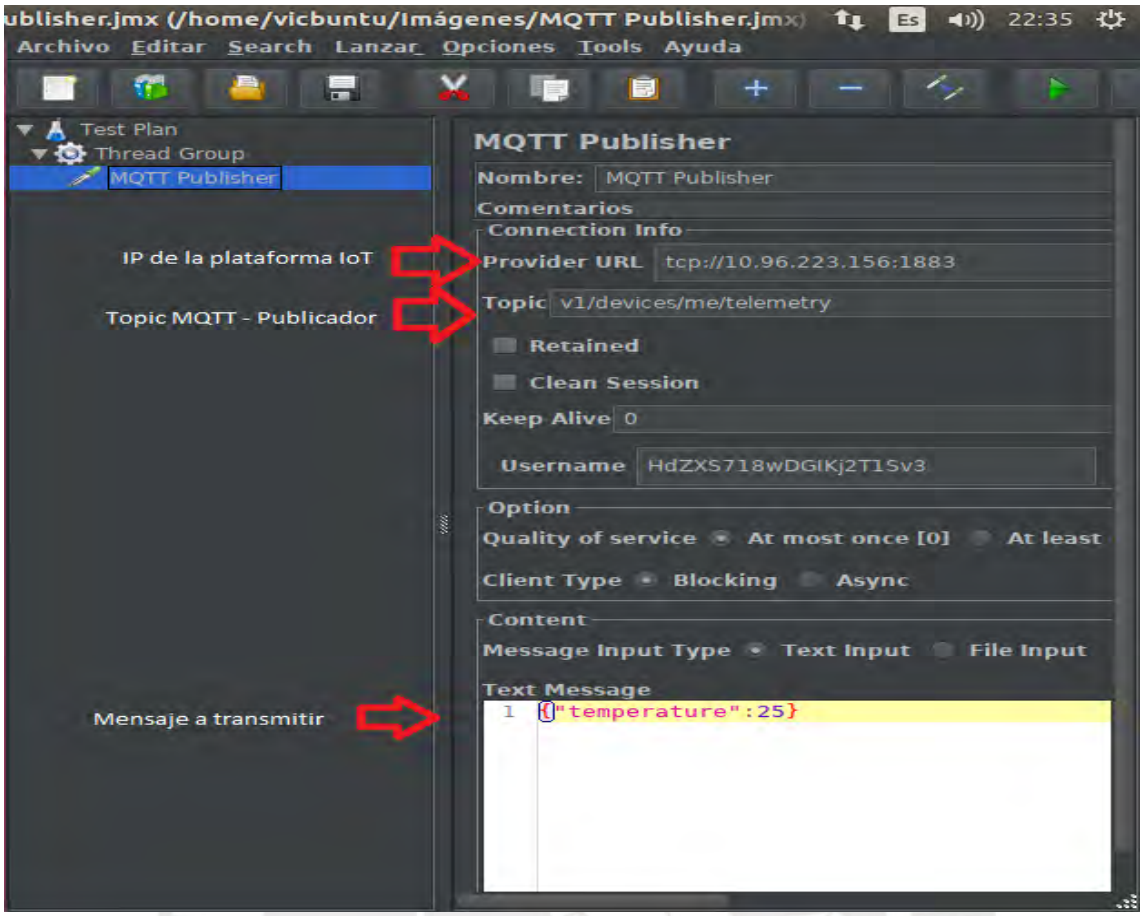

Figura 14: Interfaz gráfica Jmeter – Thread Group

Fuente: Elaboración propia

3.1.1 Simulación de carga con MQTT - Jmeter

En busca de simular diversos ambientes de prueba usando el Jmeter se decidió realizar las pruebas en un despliegue pequeño, mediano y grande. Para lograr este objetivo, se simularon 10, 100, 300 y 500 dispositivos virtuales con 2 tipos de concurrencias en simultaneo que son de 100 y 200 mensajes entregados por cada dispositivo, como resultado, se obtiene un espectro amplio de evaluación. El protocolo escogido para la evaluación de estas plataformas será MQTT con

QoS 0 debido al mejor rendimiento mostrado tanto en energía como en ancho de banda usado, lo cual es esencial en estos despliegues masivos [23].

Como se describió en el punto anterior, para poder generar esta carga se necesita de un publicador (Jmeter) y un subscriptor (Plataforma IoT). Se define el topic, la IP de envío, usuario y contraseña y el mensaje a enviar.

Después de identificados los valores, se procede a ejecutar dentro del sistema operativo un comando dentro de la plataforma como se puede observar en la figura 15. Este comando permite obtener el rendimiento computacional en vivo cuando la plataforma es sometida a la carga.

```
vicbuntu@vicbuntu:~$ dstat -ta --top-cpu --output test100-100-1.csv
```

Figura 15: Análisis de comportamiento – Simulación de carga

Fuente: Elaboración propia

```
----system---- ----total-cpu-usage---- -dsk/total- -net/total- ---paging-- ---system-- -most-expensive-
     time     |usr sys idl wai hiq siq| read  writ| recv  send|  in   out | int   csw |   cpu process
25-06 12:38:30| 59  12  28   0   0   0|   0     0 |   0     0 |   0     0 | 690  6403 |python3         51
25-06 12:38:31| 50   5  45   0   0   0|   0    16k|   0     0 |   0     0 | 555  1923 |python3         49
25-06 12:38:32| 50   6  44   1   0   0|  16k    0 |   0     0 |   0     0 | 603  1994 |python3         51
25-06 12:38:33| 63  10  26   0   0   1|   0     0 |   0     0 |   0     0 | 655  4960 |python3         50
```

Figura 16: Análisis de comportamiento – Inspección de rendimiento

Fuente: Elaboración propia

Finalmente, como observamos en figura 16 este comportamiento es visualizado y exportado como un CSV para obtener graficas resultantes de la prueba. El estrés generado por las cargas simuladas será evaluado bajo el criterio definido en el punto posterior.

3.2 Métricas de evaluación de las plataformas IoT

Las plataformas deberán asegurar una buena escalabilidad y rendimiento frente a diversos escenarios de estrés sometidos. El resultado obtenido nos dará un conocimiento previo del comportamiento de la plataforma frente a diferentes ambientes.

En primer lugar, definiremos una lista que involucra las características generales en un ambiente de cómputo usando de referencia a [30] y en base a las más críticas realizar la prueba de rendimiento.

Tabla 8: Identificación de variables de evaluación para las plataformas IoT

Métrica	Descripción
Sistema operativo	Las plataformas escogidas según su documentación están todas basadas en el sistema operativo Linux por lo cual para temas prácticos y de prueba se escogió Ubuntu 16.04
Versión de aplicaciones y bases de datos	Las bases de datos son donde se aloja la información usada de prueba. Las versiones de bases datos y aplicaciones se presentan en los anexos con los requerimientos oficiales dispuestos en cada plataforma. Las versiones son importantes para la compatibilidad entre partes de la plataforma.
Tipo de Hardware	Todas las plataformas fueron desplegadas bajo el mismo tipo de hardware físico mencionado previamente. No se optó por el despliegue nube debido a costos y rendimiento.
Protocolo de comunicación	Los protocolos de comunicación permiten el envío y recepción de la información la cual tiene como finalidad ser almacenada y usada como valor a un servicio.

Métrica	Descripción
Distribución de mensajes por segundo	Se analizará la cantidad de mensajes enviados, recibidos y guardados exitosamente en paralelo. Por lo cual se verá cuanto es la cantidad de perdidas también en los mensajes enviados.
Utilización promedio de CPU	Se define como el consumo a tiempo real del CPU en %. Debido al estrés al cual será sometido se espera obtener valores altos de utilización y también nos dará una referencia de cual plataforma usa de mejor manera el multi núcleo.
Ancho de banda	La cantidad de ancho de banda consumida en el transcurso de ejecución nos dará una noción practica a los límites de estrés en la capa de red.
Almacenamiento	La data alojada en la base de datos después de la transmisión de mensajes es esencial para dimensionar el consumo en disco. Además, evaluar algún algoritmo de deduplicación y compresiones nativas de las plataformas que nos brinde mejores resultados.

Fuente: Elaboración propia

Para poder realizar las pruebas de rendimiento en las plataformas, se evaluó el estrés generado por cargas simuladas de manera que se pueda obtener mensajes en órdenes superiores de 10^3 por segundo en paralelo. La evaluación de rendimiento esta medido bajo 4 pilares: CPU, ancho de banda, Cantidad de mensajes por segundo y almacenamiento como se recomienda en [32] . Estas variables son independientes entre sí, pero se correlacionan por lo cual se modelará la relación empírica entre estas.

3.2.1 Utilización de CPU

Las máquinas virtuales donde arrancan las plataformas IoT poseen una utilización promedio de CPU al estar en estado de reposo. Sin embargo, al aumentar la cantidad de mensajes de forma exponencial la CPU es una variable crítica en la estabilidad. Como resultado de la saturación podremos evaluar los limites en modos independientes (*stand-alone*), es decir no *clusterizadas*.

Por otro lado, podremos evaluar como manejan el proceso multi núcleo al recibir la inmensa cantidad de mensajes, este tipo de optimización en los procesos e instrucciones a nivel de arquitectura ayuda en el rendimiento general del sistema.

Figura 17: Curva de utilización de CPU multi núcleo

Fuente: Elaboración propia

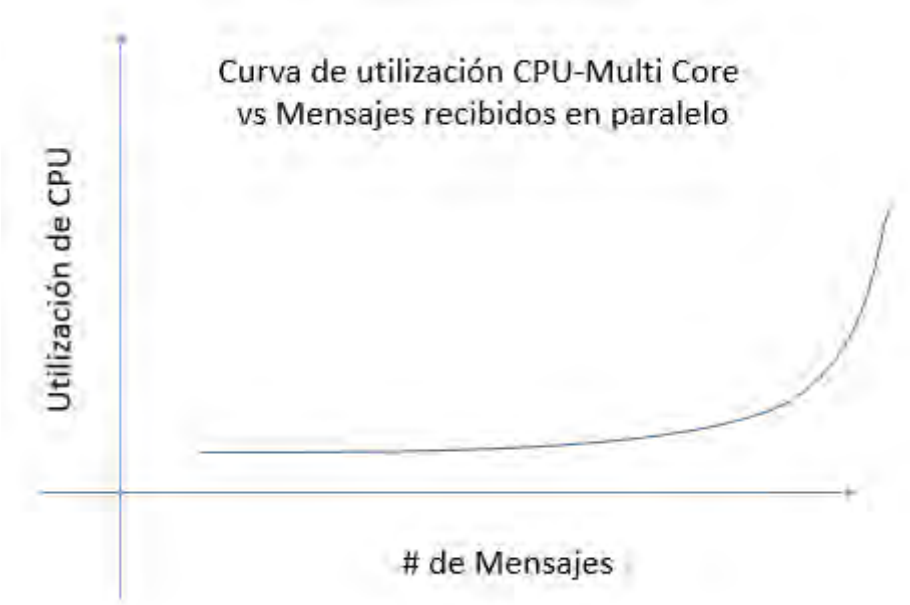

Figura 18: Curva de utilización de CPU multi núcleo

Fuente: Elaboración propia

3.2.2 Almacenamiento

El almacenamiento es la métrica basada en la recepción promedio de bytes por segundo que llega a ser guardada dentro de la base de datos de cada plataforma, sea esta local o externa. Esto no permite determinar la efectividad en el almacenamiento del envío total de mensajes, este parámetro debería escalar linealmente según la cantidad de mensaje, pero se corroborará con lo obtenido.

Como fue mencionado anteriormente, los mensajes serán enviados y recibidos mediante el protocolo MQTT, después, serán transmitidos a través del formato JSON para finalmente ser almacenados en las bases de datos.

3.2.3 Cantidad de mensajes

En la selección de protocolos de comunicación esto fue un factor crítico debido a la cabecera que puede ocupar un protocolo al comunicarse. En el presente estudio el protocolo MQTT facilita la no sobresaturación del almacenamiento.

Estos mensajes transmitidos a través del formato JSON al nivel escalado por segundo de ordenes de hasta 10^4 pps podría saturar un almacenamiento promedio muy rápidamente, además, como sabemos este factor es lineal con el ancho de banda por lo que la infraestructura de acceso de todos los dispositivos deberá ser de categoría alta. Por ejemplo, se considerara el peor escenario según lo observado en la tabla 9 en la cual un paquete está basado en el tamaño de [33], este paquete se multiplica por la cantidad de tiempo para poder ver el impacto en el almacenaje y lo critico de usar MQTT. Se plantea que cada mensaje tiene un tamaño total de 400 bytes que solo es alcanzable debido a que MQTT puede enviar varios mensajes sobre una misma conexión a contrario de HTTP que no lo puede hacer, como resultado, hay un mejor uso de ancho de banda porque no abrirá y cerrara sesiones que es lo que predominaría en el constante envió.

En la tabla 9 observamos que en una semana 1000 pps podría llegar a ocupar 4TB sin considerar deduplicación y compresión. Por otro lado, con las características habilitadas se puede llegar a un consumo de 2TB por semana. Estos valores son muy altos a pesar de la reducción de data y ancho de banda por lo cual siempre va a ser un reto el poder mantener la data fría y la data caliente, el uso de *tiering* o factor de réplica sería recomendable para este tipo de soluciones.

Tabla 9: Almacenamiento vs Tiempo

	1 paquete x seg	1000 paquetes x seg	1 paquete x seg	1000 paquetes x seg
Tiempo	Tamaño ocupado en TB	Tamaño ocupado en TB	Tamaño ocupado en TB (Deduplicacion + comprensión 1.5:1.25)	Tamaño ocupado en TB (Deduplicacion + comprensión 1.5:1.25)
1 segundo	0.0000004	0.0004	2.13333E-07	0.000213333
1 hora	0.000024	0.024	0.0000128	0.0128
24 horas	0.000576	0.576	0.0003072	0.3072
1 semana	0.004032	4.032	0.0021504	2.1504
1 mes	0.016128	16.128	0.0086016	8.6016
1 año	0.193536	193.536	0.1032192	103.2192
5años	0.96768	967.68	0.516096	516.096

Fuente: Elaboración Propia

4. PRUEBAS Y RESULTADOS

En el presente capitulo se detalla las pruebas realizadas a las distintas plataformas IoT para determinar su rendimiento en distintos ambientes de estrés según los escenarios planteados anteriormente. Las pruebas realizadas muestran comportamientos diversos los cuales se explicarán en las secciones siguientes.

4.1 Prueba de concepto

En las siguientes secciones vamos a observar el procedimiento de despliegue y datos obtenidos para realizar la comparación entre las plataformas. Para el despliegue de estas plataformas, pudimos observar que se requerían de distintos prerrequisitos en el sistema operativo, por ejemplo, desde la base de datos hasta la integración con microservicios. Se explicará brevemente en cada plataforma la serie de prerrequisitos de la arquitectura debido a que influye en el camino de transmisión y recepción de la información.

Las máquinas virtuales ejecutadas sobre el servidor visto en la tabla 7 poseen los siguientes recursos en general.

Tabla 10: Característica Máquina Virtual

Sistema Operativo	Ubuntu 16.04
vCPU	4 vCPU
MEMORIA	16GB
HIPERVISOR	VMWARE 6.5
DISK	100GB – Provisionamiento delgado

Fuente: Elaboración Propia

4.1.1 Despliegue de la plataforma Thingsboard

La plataforma Thingsboard posee distintos sistemas operativos compatibles para su despliegue, por ejemplo, Windows, CentOS, Ubuntu. Para estos escenarios de prueba se realizó sobre Ubuntu 16.04 TLS por practicidad y mejor rendimiento sobre este sistema operativo.

Thingsboard posee distintos tipos de compatibilidad como SQL, NOSQL y base de datos hibridas. En el presente caso, se usó PostgreSQL más Cassandra debido a la sugerencia de la marca para escenarios mayor a 5000 mensajes por segundo [34].

La configuración del PostgreSQL fue de un despliegue estándar en el cual se inicializa el usuario y las conexiones necesarias para su correcto funcionamiento, véase la figura 19.

```
# instalacion :
sudo apt install -y wget

# Importacion del repositorio de llaves:
wget --quiet -O - https://www.postgresql.org/media/keys/ACCC4CF8.asc | sudo apt-key add -

# Añadir repositorio al sistema:
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

# Instalar e inicializar sistema:
sudo apt update
sudo apt -y install postgresql-12
sudo service postgresql start

# creacion de usuario/contraseña
sudo su - postgres
psql
\password
\q

# Conexion a la base de datos
psql -U postgres -d postgres -h 127.0.0.1 -W
CREATE DATABASE thingsboard;
\q
```

Figura 19: Configuración Básica de PostgreSQL

Fuente: Elaboración propia

Por otro lado, para el encolamiento en los mensajes recibidos y almacenados en el sistema se usará el de la memoria que se encuentra por defecto, sin embargo, se recomienda el uso de RabbitMQ o Kafka debido a su solución de grado empresarial.

Como resultado de la configuración, se puede tener acceso al portal y su correcto funcionamiento para los ambientes de prueba, como puede verse en la figura 20. Dentro del portal de la plataforma observamos sus funcionalidades principales que son gestión de dispositivos por reglas, segmentación por grupos, creación de apartados personalizados, búsqueda de errores, gestión de dispositivos y clientes.

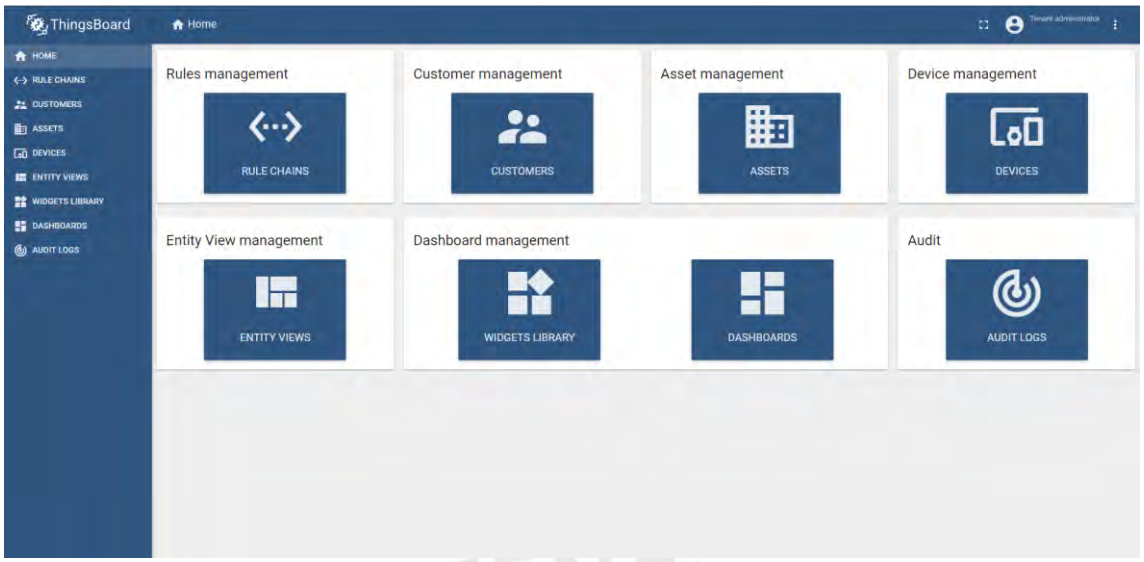

Figura 20: Plataforma Thingsboard – Panel principal

Fuente: Elaboración propia

Para realizar la prueba de rendimiento, primero se tiene que crear el dispositivo de manera lógica de manera que la plataforma sabe de qué dispositivo se suscribirá una información en específico.

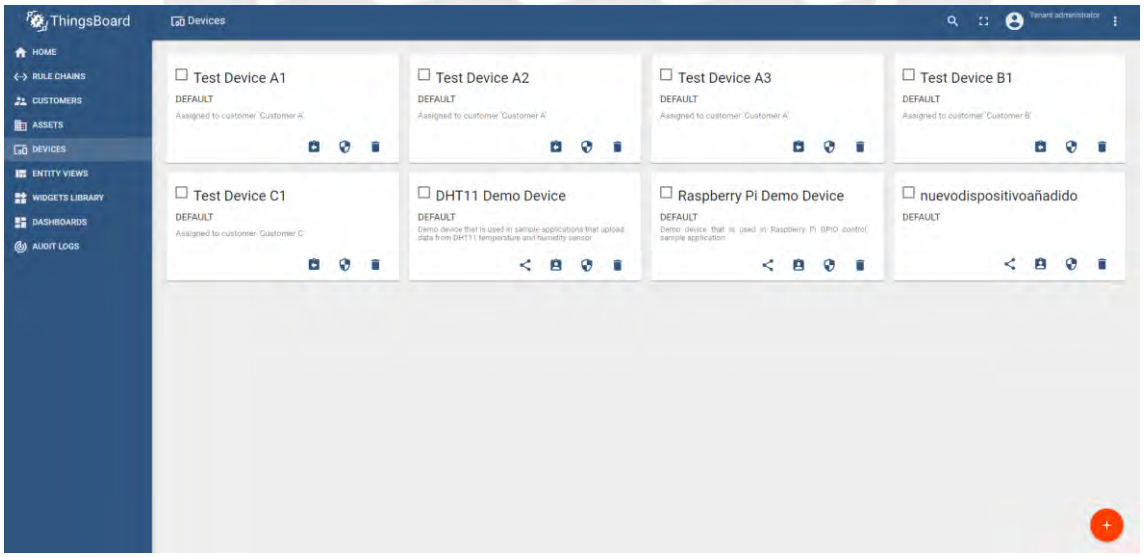

Figura 21: Plataforma Thingsboard – Panel de dispositivos

Fuente: Elaboración propia

En segundo lugar, se realizó el registro de tokens como un agregado de seguridad como se evaluó en capítulos anteriores, véase la figura 22 que presenta un token por dispositivo y un token de acceso el cual deberá ser usado por el simulador de carga Jmeter proveyéndole los accesos necesarios para emitir la carga de trabajo, como se puede apreciar en la figura 23.

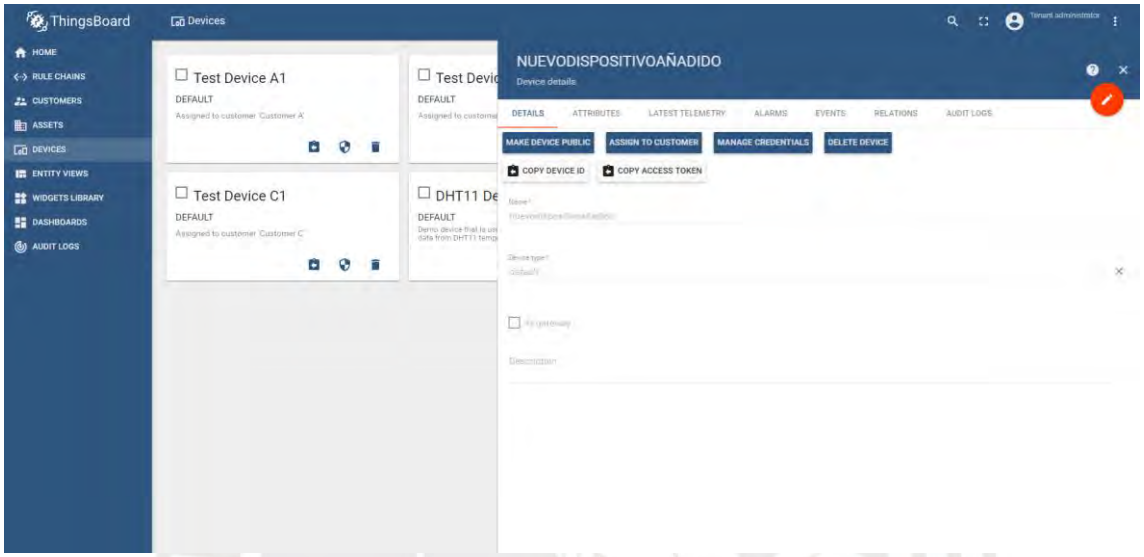

Figura 22: Plataforma Thingsboard – Gestión de dispositivo

Fuente: Elaboración propia

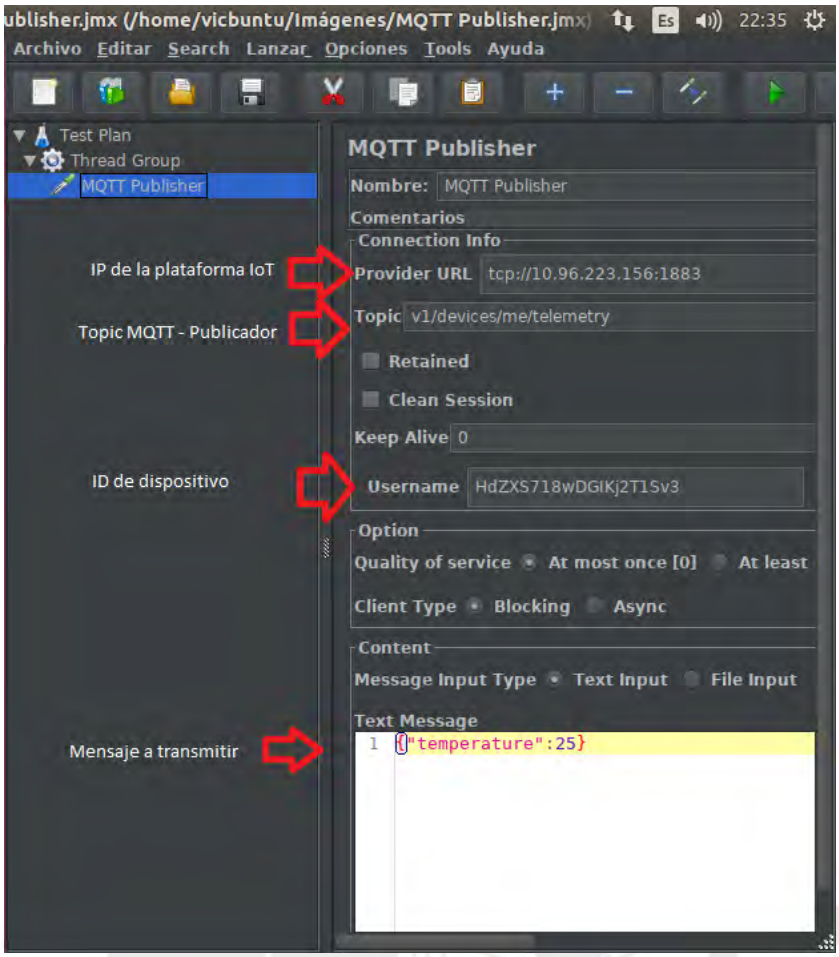

Figura 23: Jmeter – Simulación de carga a plataforma Thingsboard

Fuente: Elaboración propia

Finalmente, como se puede apreciar en las figuras 24 y 25 se recibió un mensaje con la asignatura temperatura y un valor designado. Este valor llego en el mismo tiempo, sin embargo, toda la carga de mensajes no llego al mismo tiempo. El factor determinante para que no llegue al mismo tiempo es el encolamiento debido a la cantidad de mensajes y cómo éste es recibido bajo las capacidades del sistema.

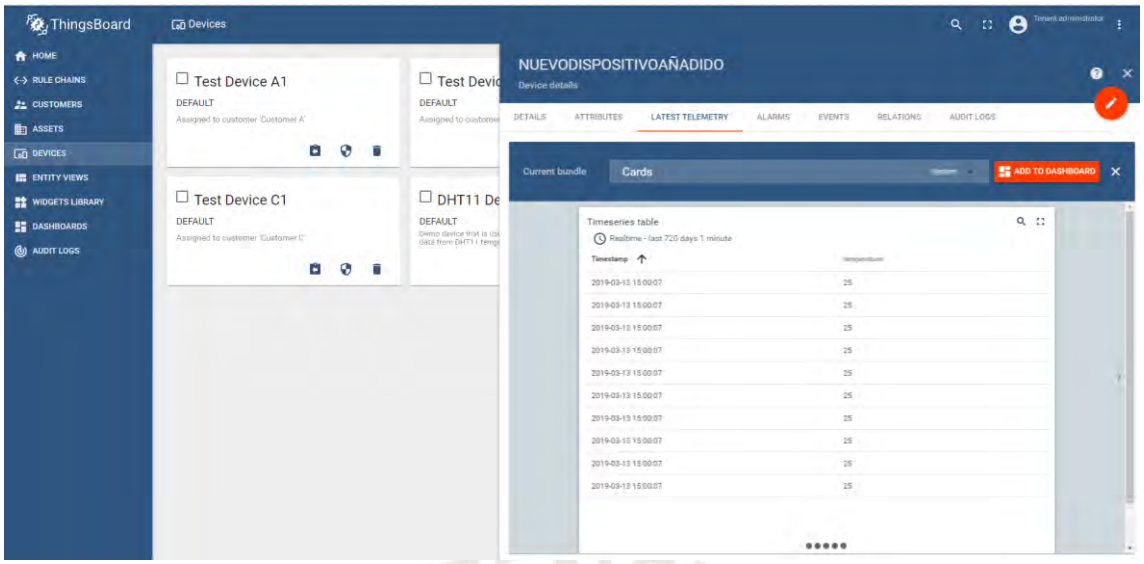

Figura 24: Plataforma Thingsboard – Registro de telemetría de dispositivos

Fuente: Elaboración propia

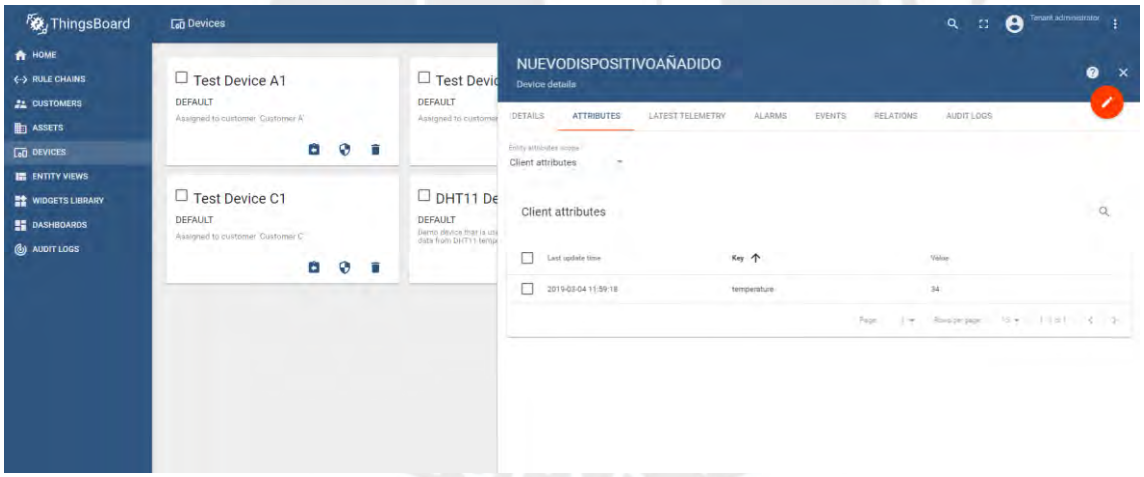

Figura 25: Plataforma Thingsboard – Telemetría de dispositivos

Fuente: Elaboración propia

Para el presente caso, para evaluar la cantidad de recursos se ejecutaron los comandos que se observan en la figura 26. El comando *dstat –top -cpu* nos permitió analizar en tiempo real el consumo de ancho de banda, CPU y uso de disco a tiempo real para los distintos escenarios de prueba. Por otro lado, toda

esa información fue exportada en formato CSV para poder analizar y realizar graficas comparativas de lo obtenido.

Para asegurar una evaluación imparcial se realizaron tres pruebas por cada escenario de prueba, es decir, tres pruebas para 100/200 instancias de dispositivos en una concurrencia con la cantidad de mensajes de 10/100/300/500 por instancia.

La elección de esta concurrencia y separación se basó en [32] y se adaptó a lo planteado, sin embargo, para posteriores estudios se buscará un análisis granular para realizar una curva ascendente desde 1 mensaje hasta 500 mensajes con sus respectivas instancias. Esto se plantea debido a que se busca una mayor precisión y lograr visualizar el punto de quiebre en los modelos Standalone de estos servicios.

```
vicbuntu@vicbuntu:~$ dstat -ta --top-cpu --output test500-200-3.csv
```

Figura 26: Plataforma Thingsboard – monitoreo de recursos

Fuente: Elaboración propia

Finalmente, la data obtenida fue evaluada en secciones posteriores para observar la comparación frente a las otras plataformas bajo el mismo escenario de estrés.

4.1.2 Despliegue de la plataforma SiteWhere

La plataforma SiteWhere versión 2.1 está basada en microservicios. Esto nos brindó la posibilidad de un despliegue en la nube de AWS y otro en infraestructura local, como se explicó previamente, se optó por la solución de infraestructura local con base Linux para su implementación. Para su correcto despliegue se usó como base de infraestructura a Kubernetes el cual ejecutara los microservicios en los cuales está basado SiteWhere.

Por otro lado, como otros requisitos base, se usó Helm el cual agiliza el proceso de configuración de estos contenedores. Además, para el control, conexión y monitoreo se usa Istio versión 1.6 el cual añade las virtudes a la plataforma, véase la figura 27 para las configuraciones pertinentes. En la figura 27 observamos los comandos usados para la instalación del Kubernetes, Istio y el repositorio de SiteWhere.

```
# Instalacion Kubernetes
sudo apt-get update && sudo apt-get install -y apt-transport-https
curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -
echo "deb https://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee -a /etc/apt/sources.list.d/kubernetes.list
sudo apt-get update
sudo apt-get install -y kubectl

# Instalacion Istio

curl -L https://istio.io/downloadIstio | sh -
cd istio-1.6.5
export PATH=$PWD/bin:$PATH
istioctl install --set profile=demo
kubectl label namespace default istio-injection=enabled
namespace/default labeled
kubectl label namespace default istio-injection=enabled

# Instalacion Sitewhere desde el repositorio de Sitewhere

helm repo add sitewhere https://sitewhere.io/helm-charts
helm repo update
helm install --name sitewhere sitewhere/sitewhere
```

Figura 27 Plataforma SiteWhere – Configuración de Plataforma

Fuente: Elaboración propia

Los pasos previos ejecutados nos brindaron acceso a la plataforma para poder realizar las configuraciones pertinentes, además, se agregó los dispositivos mediante el uso de la plataforma como se observa en la figura 29.

Luego de agregado el dispositivo, se procedió con el registro del token ID para añadir una capa de seguridad de accesos a la información brindada por los dispositivos.

Figura 28 Plataforma SiteWhere – Dashboard principal

Fuente: Elaboración propia

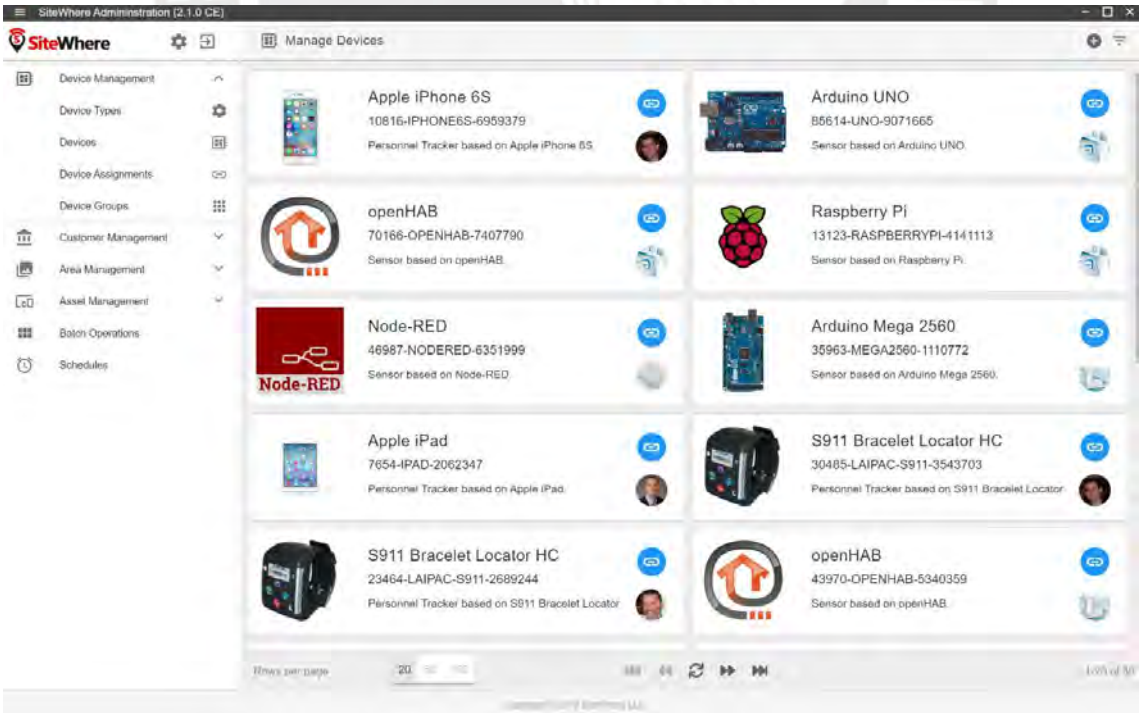

Figura 29 Plataforma SiteWhere – Configuración de Plataforma

Fuente: [35]

Finalmente, cuando la plataforma estuvo configurada se procedió a realizar la prueba de rendimiento y la captura de información similar al proceso anterior como se observa en la figura 23 y 26, los resultados obtenidos se encuentran en formato CSV para su posterior análisis.

4.1.3 Despliegue de la plataforma Mainflux

La plataforma Mainflux posee todas sus características como microservicios bajo el sistema de Docker, el cual se puede obtener del repositorio de GIT de la marca propietaria. En el caso de la base de datos se usará la de PostgreSQL lo cual se empleó la misma configuración que la figura 19.

El servicio se ejecutó dentro del contenedor como se observa en la figura 31, en esta figura vemos que existen procesos como usuarios y dispositivos por lo cual se valida su correcto despliegue. Como resultado, se obtiene acceso a la plataforma en su version 0.7.0 . Mainflux esta basado en mosquitto como broker MQTT por lo que hace una entrada eficiente de informacion bajo estre protocolo.

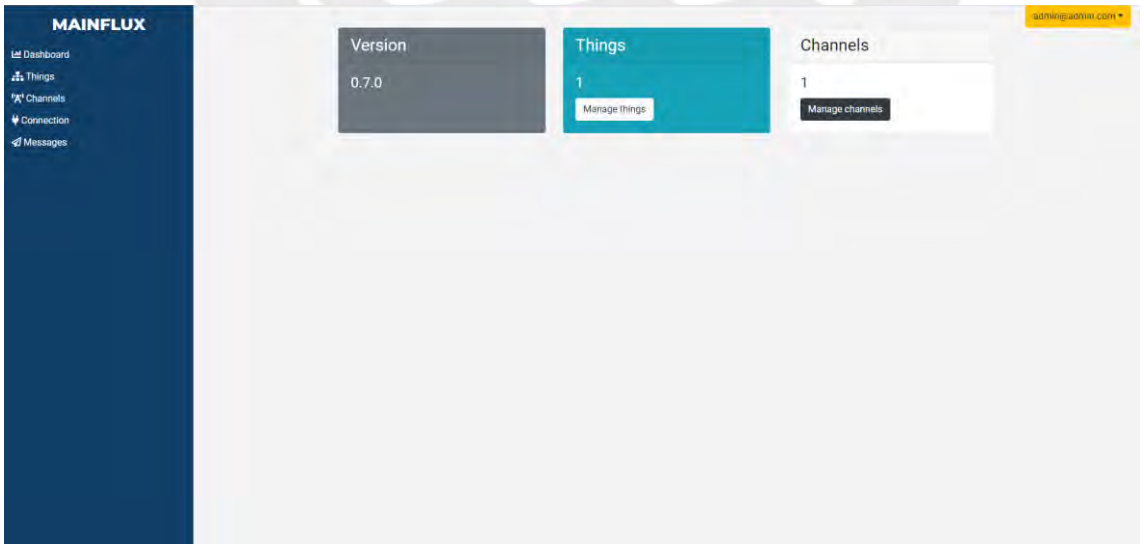

Figura 30 Plataforma Mainflux – Pestaña Principal de Mainflux

Fuente: Elaboración propia

Figura 31 Plataforma Mainflux – Ejecución de Servicio Mainflux

Fuente: Elaboración propia

Se realizó el procedimiento de registrar un dispositivo de manera que reciba los mensajes publicados por el simulador de carga Jmeter, este proceso se realiza en la figura 32. El dispositivo registrado posee el token ID y el key ID para asegurar una mayor protección para enviar su información.

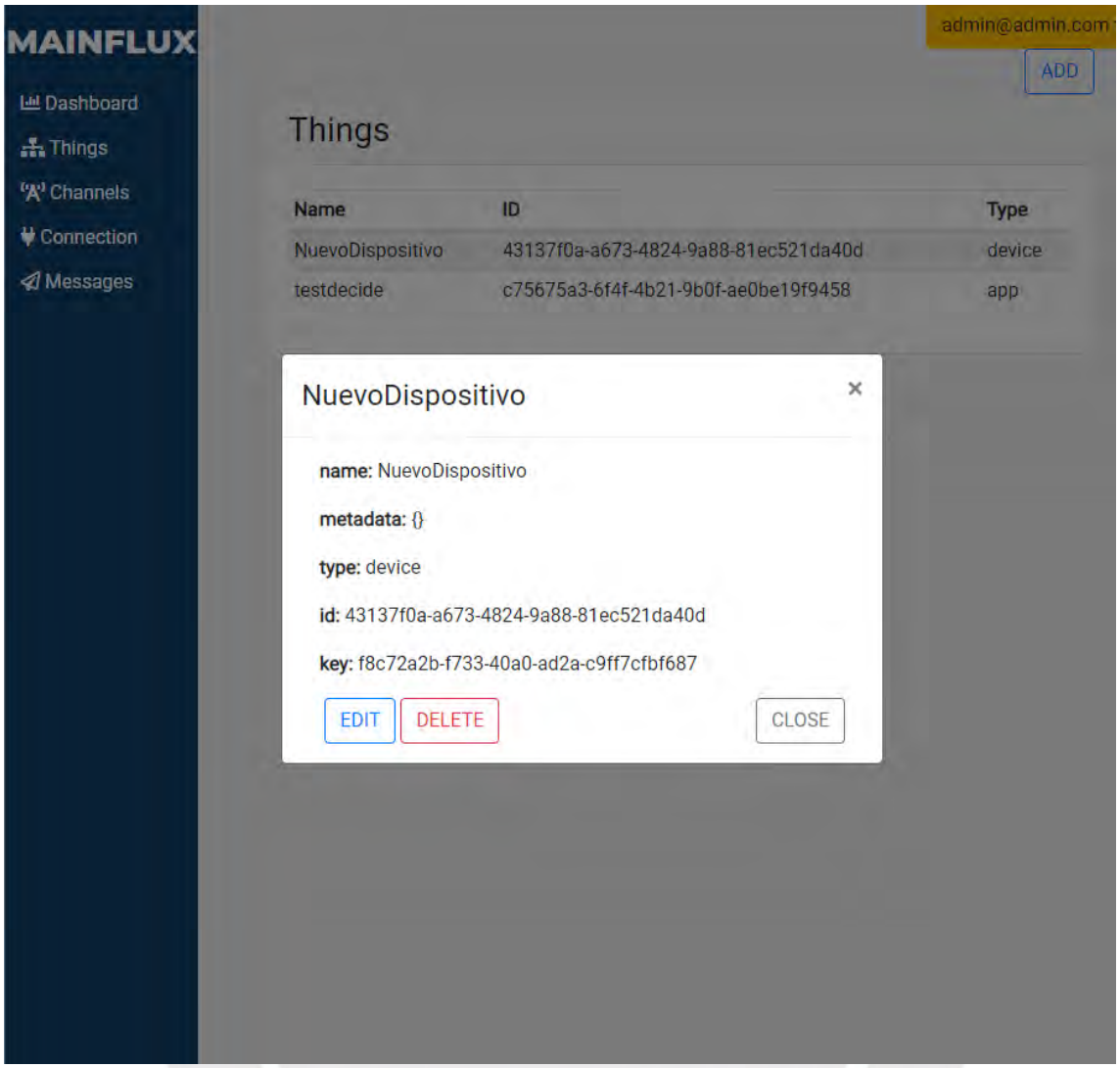

Figura 32 Plataforma Mainflux – Configuración de Dispositivo Nuevo

Fuente: Elaboración propia

Finalmente, luego del registro del dispositivo, se agregó el topic al cual escuchar como se observa en la figura 33. Posteriormente, después de la correcta configuración de la plataforma, se procedió a realizar la simulación y captura de carga para los diversos ambientes mencionados con la finalidad de recolectar las métricas en formato CSV como se vio en la figura 26 para un posterior análisis.

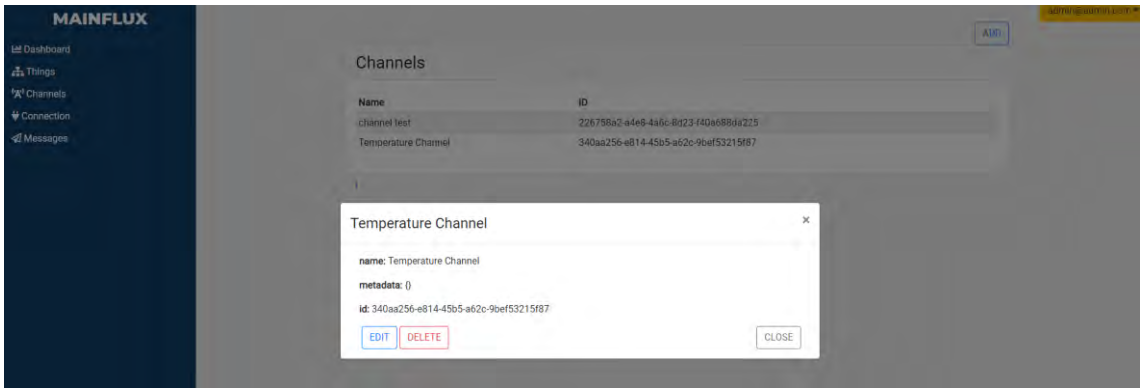

Figura 33 Plataforma Mainflux – Configuración de Topic

Fuente: Elaboración propia

4.1.4 Despliegue de la plataforma Kaa

La plataforma Kaa es compatible con diversos sistemas operativos como CentOS, Red Hat Enterprise, Oracle Linux y Ubuntu para su despliegue. Por otro lado, esta plataforma provee de un pequeño ambiente de prueba de su sistema en un *sandbox* basado en la nube. Sin embargo, para el presente estudio se consideró el Ubuntu 16.04 TLS para hegemonizar las pruebas.

Kaa requiero de software de terceros para su funcionalidad completa, por ejemplo, Oracle JDK8, PostgreSQL 9.4, MariaDB 5.5 y Zookeeper 3.4.5, los requerimientos se encuentran en [36]. Se empleo el usó PostgreSQL debido al recurso previamente configurado para el resto de sistemas. Sin embargo, aparte de requerir a PostgreSQL, se requiere una base de datos NoSQL por lo que se despliega MongoDB.

Se observa en la figura 34 la configuración de los servicios de terceros requeridos en el sistema como el JDK, base de datos, Zookeeper y el despliegue de la plataforma como servicio, con esto validamos el correcto despliegue y funcionamiento de la plataforma.

```
#Configuracion del JDK8
sudo apt-get install wget ca-certificates curl
sudo add-apt-repository ppa:webupd8team/java
sudo apt-get update
sudo apt-get install oracle-java8-installer

#Creacion de base de Datos Kaa
CREATE DATABASE "kaa"
   WITH OWNER "postgres"
   ENCODING 'UTF8'
   LC_COLLATE = 'en_US.UTF-8'
   LC_CTYPE = 'en_US.UTF-8'
   TEMPLATE template0;

#Instalacion del Zookeeper
sudo apt-get install zookeeper
sudo /usr/share/zookeeper/bin/zkServer.sh start
netstat -ntlp | grep 2181

#Instalacion e ejecucion de MongoDB
sudo apt-key adv --keyserver keyserver.ubuntu.com --recv 7F0CEB10
echo 'deb http://downloads-distro.mongodb.org/repo/ubuntu-upstart dist 10gen' | sudo tee /etc/apt/sources.list.d/mongodb.list
sudo apt-get update
sudo apt-get install -y mongodb-org=2.6.9 mongodb-org-server=2.6.9 mongodb-org-shell=2.6.9 mongodb-org-mongos=2.6.9 mongodb-org-tools=2.6.9
sudo service mongod start

#Instalacion de la Platarforma
tar -xvf kaa-deb-*.tar.gz
sudo dpkg -i kaa-node-*.deb
sudo service kaa-node start
```

Figura 34 Plataforma Kaa – Configuración de la Plataforma

Fuente: Elaboración propia

Como se observa en la figura 35, las configuraciones previas nos brindan acceso a la plataforma para poder realizar las pruebas pertinentes, por otro lado, se agregó los dispositivos mediante el uso de la plataforma el cual incluye su token ID como se observa en la figura 36.

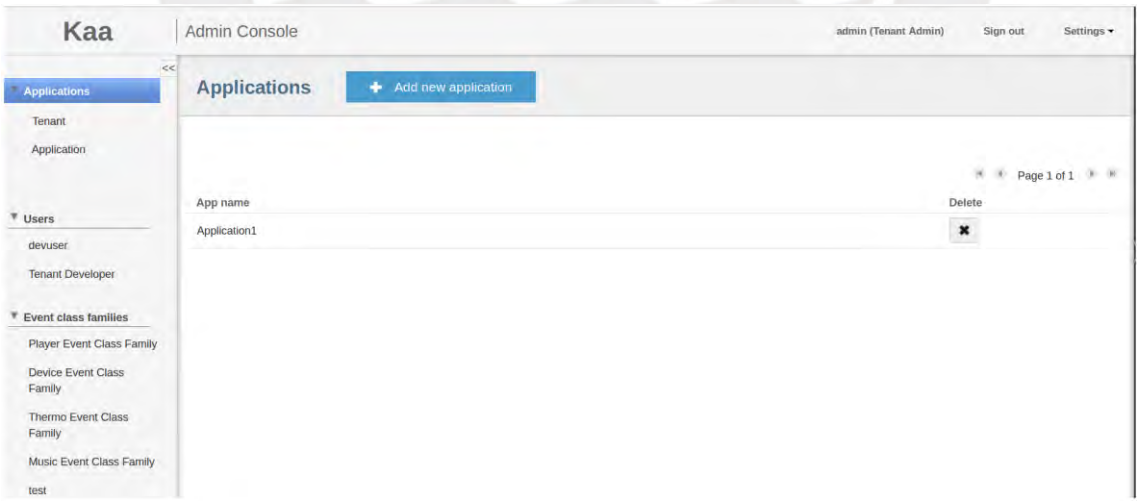

Figura 35 Plataforma Kaa – Pestaña principal – Versión 0.9

Fuente: Elaboración propia

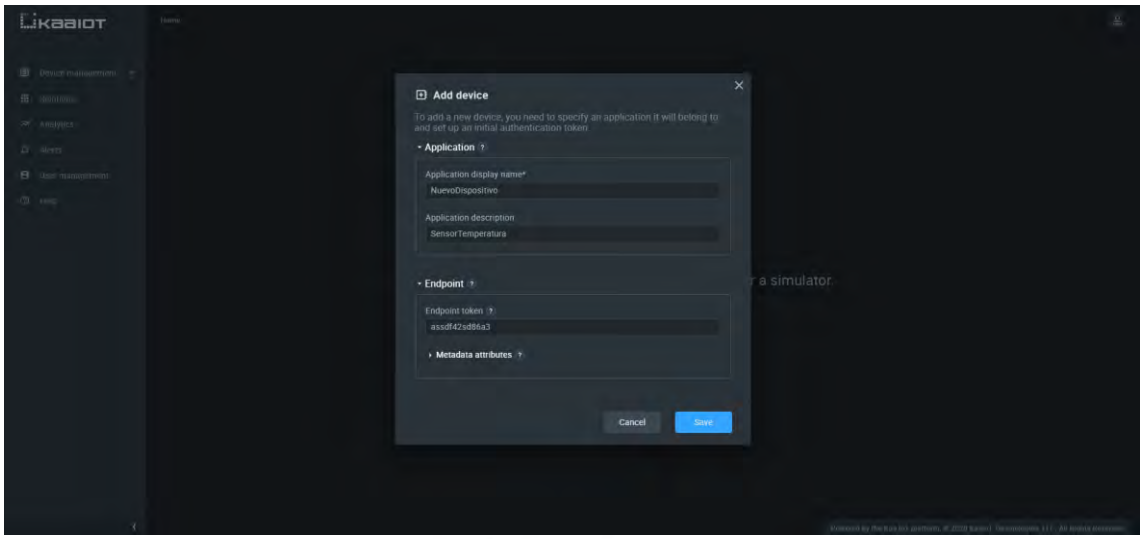

Figura 36 Plataforma Kaa – Configuración de dispositivo – Versión Actual

Fuente: Elaboración propia

Se realizó el procedimiento de registrar el dispositivo nuevo a un nuevo topic. El topic será generado a través de Jmeter según los escenarios planteados en el presente estudio.

Por último, se procedió a realizar la prueba de rendimiento y la captura de información como se observó previamente en las figuras 23 y 26, los resultados obtenidos se exportaron en formato CSV facilitando el posterior análisis de las métricas obtenidas.

4.2 Análisis de resultados

4.2.1 Análisis de resultados – Mensajes

Las plataformas evaluadas, debido a los distintos escenarios simulados (pequeños, medianos y grandes), permitieron observar distintos tipos de comportamiento en los parámetros evaluados. En la figura 37 y 38 existen resultados asimétricos e interesantes en los mensajes enviados, por ejemplo, a

una concurrencia de 100 instancia por 10 mensajes por segundo la plataforma SiteWhere tuvo el mejor rendimiento entre las 4 plataformas evaluadas. La plataforma SiteWhere tuvo una capacidad de recibir y almacenar 534 mensajes de los 1000 efectuados, es decir, 53% de los mensajes enviados fueron capturados considerando un QoS 0 en MQTT. Sin embargo, se observó un comportamiento irregular en los escenarios mayores, por ejemplo, a 100 mensajes por 100 instancias el de mayor rendimiento fue la plataforma Kaa recibiendo y almacenando 1666 mensajes de los 10000 totales, por otro lado, no era lo esperado que la plataforma KAA tuviese un patrón de caída tan abrupto en los escenarios mayores llegando a estar ultimo dentro del test.

Por otro lado, para una concurrencia de 100 y 200 instancias en 300 y 500 mensajes por dispositivo observamos como líder absoluto en ambas categorías a la plataforma Thingsboard con 1857 y 2864 mensajes respectivamente.

Finalmente, se observa un gran porcentaje de caída de paquetes lo cual se esperaba debido a que es un escenario de nodos únicos y sin optimización de distribución de paquetes. Se podría mejorar el porcentaje de paquetes caídos cambiando el QoS en el protocolo MQTT, pero se tendría que evaluar el impacto en ancho de banda requerido para la retransmisión de estos paquetes.

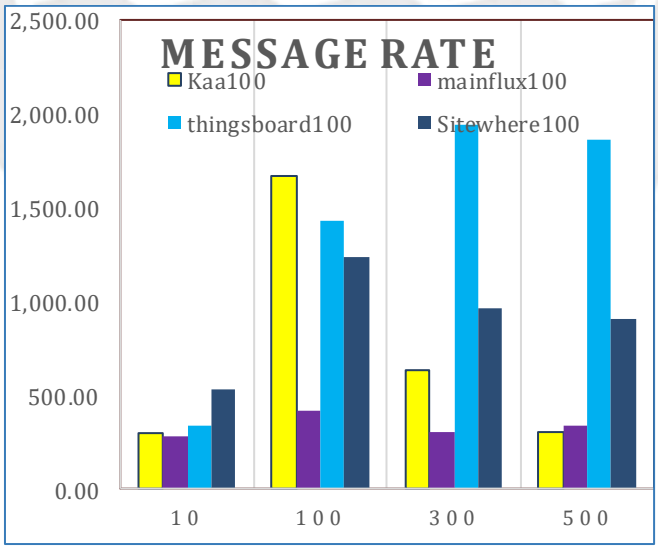

Figura 37 Troughput de mensajes para una concurrencia de 100 instancias en paralelo

Fuente: Elaboración propia

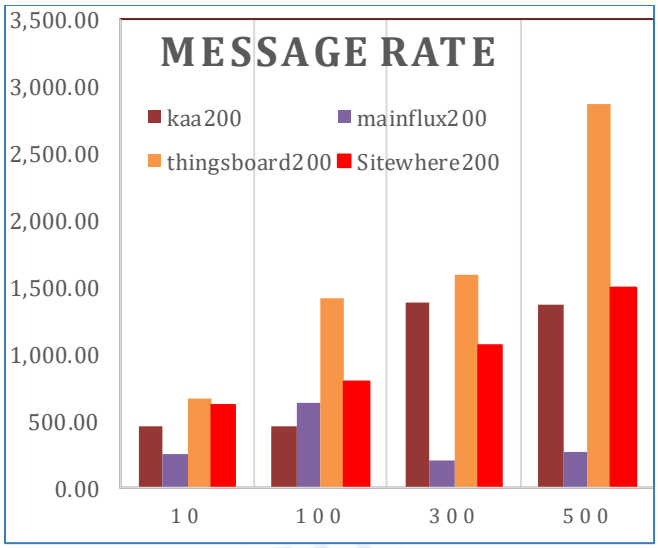

Figura 38 Troughput de mensajes para una concurrencia de 200 instancias en paralelo

Fuente: Elaboración propia

4.2.2 Análisis de resultados – CPU

Lo esperado en el consumo de CPU es que este sea de un valor alto debido a la saturación misma del sistema. Se observó en los datos obtenidos que no hay un correcto uso de los múltiples núcleos de los servidores, esto es debido por el mismo código de configuración de las plataformas, por lo cual se suele saturar en gran medida. Se aseguró que las máquinas virtuales sean las únicas ejecutadas en su debido momento para que no exista procesos fuera del ciclo de reloj que obligue el retraso de ejecución de los núcleos virtuales. En la tasa de 100 concurrencias por dispositivo apreciamos que el de mayor rendimiento es la plataforma Kaa, con un valor de 15% de utilización de CPU, que a su vez se mantuvo como la de menor consumo de CPU en los 4 escenarios incluso en el escenario de 200 concurrencias en paralelo. Por otro lado, vemos que la plataforma con más consumo de CPU fue Thingsboard, que a su vez fue el de mayor tasa de recepción y almacenaje de mensajes de manera exitosa. Sin embargo, el consumo de 85% de CPU es excesivo para la cantidad de paquetes procesados pudiendo provocar errores y saturación al mismo servidor.

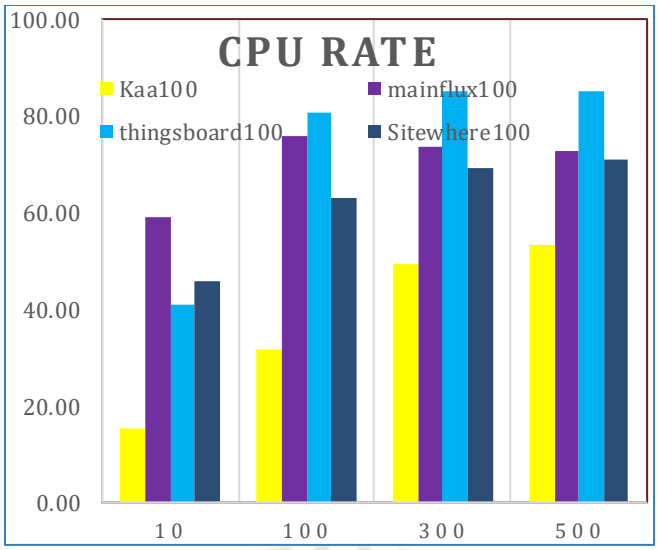

Figura 39 Consumo de CPU para una concurrencia de 200 instancias en paralelo

Fuente: Elaboración propia

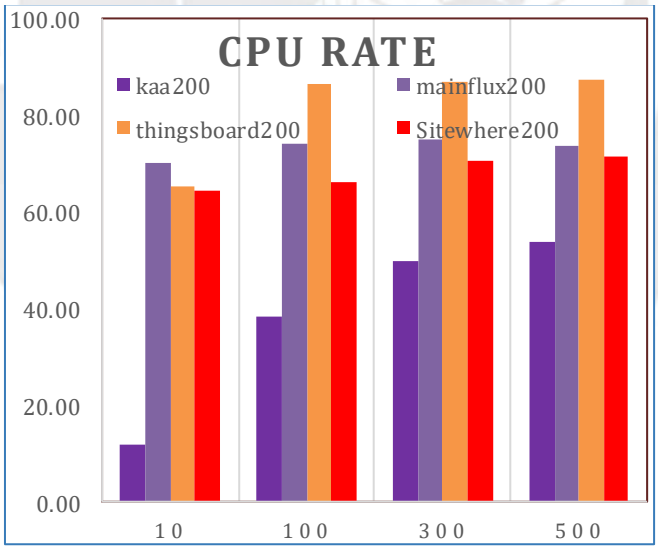

Figura 40 Consumo porcentual de CPU para una concurrencia de 200 instancias en paralelo

Fuente: Elaboración propia

4.2.3 Análisis de resultados – Data almacenada y ancho de banda

Para la reducción de la data a almacenar se empleó el protocolo MQTT debido a su ligero requerimiento, sin embargo, a la cantidad de mensajes enviados por segundo en los diversos ambientes se espera observar cierta saturación en canales de acceso.

La data recolectada que se muestra en la figura 41 permite observar que existe un flujo muy pequeño de información en la plataforma Kaa para ambos escenarios de concurrencia paralela de 100 y 200. Esto se podría justificar debido a que la plataforma cuenta con deduplicación y compresión por lo cual, en un ambiente simulado de data correlacionada, esta tiende a reducir bastante su consumo en el almacenamiento. Por otro lado, la plataforma Mainflux fue la de mayor tamaño con 439 Megabytes/s a una concurrencia de 200 instancias con 500 mensajes, lo cual es bastante alto. Sin embargo, es optimizable mediante compresión y deduplicación por hardware o software de terceros, pero en un ambiente tradicional esta cantidad de flujo de información saturaría los almacenamientos monolíticos no preparados para estas cargas

En segundo lugar, a una concurrencia de 100 instancias por 500 mensajes observamos resultados muy similares entre las plataformas de Thingsboard y SiteWhere. Sin embargo, nuevamente Mainflux acabo ocupando más espacio de lo obtenido con las otras plataformas a pesar de no haber sido la que recibió más mensajes a 100 instancias paralelas.

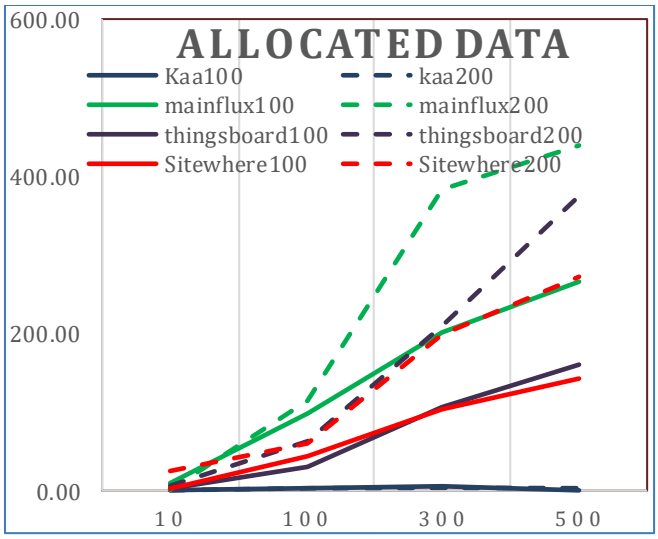

Figura 41 Data almacenada para una concurrencia de 100 y 200 instancias en paralelo

Fuente: Elaboración propia

Por último, tenemos el análisis para el ancho de banda done se obtuvieron resultados favorables para algunas y no tanto para otras plataformas, por ejemplo, Thingsboard tiene el menor flujo consumido en 0.13 Megabytes/s para 10 mensajes a una concurrencia de 100 instancias paralelas, por otro lado, Mainflux en el mismo escenario a 500 dispositivos tiene un flujo de 1.8 Gigabytes/s lo cual para una red no preparada se estima que será un pésimo escenario de despliegue. Este abrupto cambio que hay entre las plataformas es debido a la forma que usan el Gateway de comunicación y el procesamiento del mismo paquete en sí.

En el segundo caso, el flujo a 200 instancias por 500 mensajes, observamos que Mainflux recibe una carga de 3.1 Gigabytes/s lo cual es un valor critico a tomar en cuenta para el diseño de la red de acceso, distribución y core debido a que un número alto de paquetería acumulada estaría como cola de tráfico.

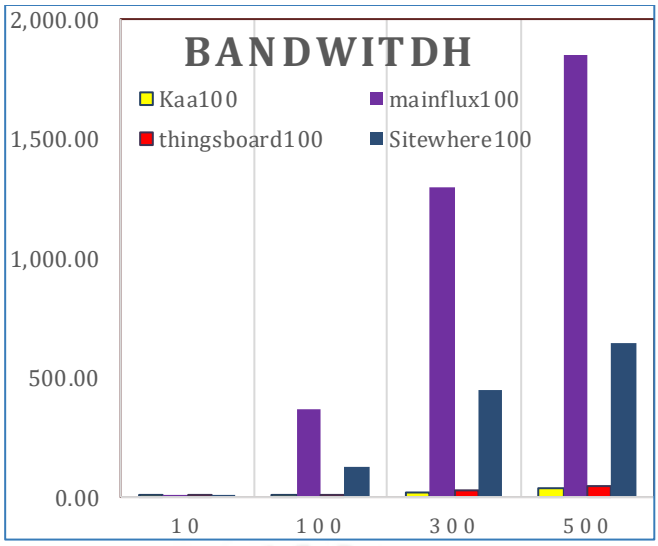

Figura 42 Ancho de banda consumido para una concurrencia de 100 y 200 instancias en paralelo - MBps

Fuente: Elaboración propia

4.3 Valor estimado para obtener el rendimiento de las plataformas IoT

Las plataformas IoT al ser sistemas computacionales, virtualizados o ejecutados como servicio único en un servidor, van a depender de los recursos computacionales y de los factores externos que implica a la red. Una alternativa para estimar el comportamiento de las plataformas es usando el patrón estadístico de las evaluaciones previas, por lo cual, se asignó pesos a las 4 variables, como se observa en la figura 43. Estos pesos fueron asignados según valores predecibles de las plataformas según los valores recolectado, además, de consideraciones de prioridad por escalabilidad, criticidad y sostenibilidad económica. Por lo tanto, se obtuvo un índice por el cual se podría medir a estas plataformas siempre y cuando se replique la misma prueba o el fabricante desee dar valores de escenarios de prueba con la plataforma a distintas cargas. Este criterio no es inamovible y puede adaptarse según requerimientos y criterios del usuario debido a que los valores asignados se fundamentaron para proporcionar una idea básica del posible rendimiento de la plataforma frente a otras.

La variable Pr en la figura 43 es el rendimiento esperado de una plataforma al

instante analizado bajo las 4 variables propuestas, además, se sabe que la sumatoria de los pesos será igual a 1 lo que define al final la distribución de pesos. Por otro lado, las plataformas que obtengan un mayor valor de Pr serán las que obtienen un mejor rendimiento bajo los escenarios propuestos.

$$Pr = \sum r = \Gamma_1 x_1 + \Gamma_2 x_2 + \Gamma_3 x_3 + \cdots + \Gamma_N x_N$$

Figura 43 Ecuación de suma ponderada con pesos

Fuente: Elaboración propia

Tabla 11 Pesos distribuidos para las 4 variables de evaluación

	rx
Troughput de mensajes	0.4
Ancho de Banda	0.2
Disco	0.1
CPU	0.3

Fuente: Elaboración propia

Como resultado, aplicando estos valores distribuidos para evaluar las 4 plataformas previamente desplegadas se obtuvo lo siguiente en el escenario de 100 instancias paralelas con 10 y 100 mensajes por segundo:

Tabla 12 Resultado de valores para las plataformas evaluadas

	KAA	MAINFLUX	THINGSBOARD	Sitewhere
10	144.03	133.74	153.9894199	230.8646423
100	687.071	182.119	581.3169917	509.6357648

Fuente: Elaboración propia

Según lo obtenido para la ecuación, observamos que SiteWhere demostró un mejor rendimiento para 10 mensajes con una concurrencia de 100 instancias en paralelo. Para el caso de 100 mensajes con concurrencia de 100 instancias obtuvimos como líder a la plataforma Kaa lo cual se puede apreciar de los resultados visualizados en las gráficas previas.

Finalmente, se realizó el diseño de red de una plataforma IoT con la finalidad de poder someterla bajo el mismo índice de evaluación, buscando además, dar inicio a próximos estudios para el diseño completo de la plataforma y que esta pueda ser usadas como propiedad de la universidad y diferentes tesis de la presente casa de estudios.

Se considero lo siguiente para su diseño:

- Sistema operativo base se usará Linux (Ubuntu 16.04) debido a su mejor adaptabilidad, rendimiento y uso de librerías.
- Base de datos NoSQL MongoDB debido al buen desempeño, escalabilidad y adaptabilidad para estos ambientes.
- El Gateway de comunicación estaba basado en Eclipse Mosquitto el cual es un software de código libre capaz de soportar el protocolo MQTT con su modelo de publicador/subscriptor.

Después de someter la plataforma a las mismas cargas en los diversos escenarios planteados anteriormente, se obtuvo como resultado lo siguiente:

Tabla 13 Capacidades y cantidades requeridas

	KAA	MAINFLUX	THINGSBOARD	Sitewhere	Nueva Plataforma
10	144.03	133.74	153.9894199	230.8646423	167.856
100	687.071	182.119	581.3169917	509.6357648	433.629

Fuente: Elaboración propia

Podemos observar de la tabla 13 que el rendimiento de la plataforma diseñada tiene un comportamiento superior al promedio de las 4 plataformas, sin embargo, se espera que al activar distintas funcionalidades extras el rendimiento se vea afectado.

Por lo tanto, con el uso del índice podemos tener un valor estimado del comportamiento de las plataformas y ser una guía práctica de como estas podrían optimizarse para un mejor rendimiento.

4.4 Evaluación económica

Como hemos observado de los resultados previos, los costos de inversión para el despliegue de plataformas IoT debe considerar una adquisición vertical a la infraestructura, es decir, desde el computo hasta la red de acceso y distribución. Se evaluará el diseño para una mediana empresa que se aproxima un uso de 500 dispositivos finales. El diseño y costeo se realizará para un entorno local y nube para el soporte de una plataforma IoT promedio. Este proyecto se considerará que es una renovación tecnológica separada a una infraestructura existente dentro de una compañía.

El equipamiento necesario para la alta disponibilidad y el diseño sugerido para

el soporte de una plataforma IoT son los siguientes:

Consideraciones para esta evaluación:

- Se optará para las mejores prácticas para preservar la alta disponibilidad y continuidad de negocio.
- El escenario busca soportar 500 dispositivos Wi-Fi o Beacons.
- La compañía posee recursos de infraestructura previamente instalados.
- Los sensores y dispositivos finales no se considerarán dentro del costeo debido a que no acarrean un costo mayor dentro del arrendamiento operativo a 5 años del proyecto.

Tabla 14 Capacidades y cantidades requeridas

	Escenario 500 Dispositivos	Capacidades
Access Points	21	Access Point para interiores, soporte de Beacon y controlador nube
Switches de Acceso para los Access Points	2	Switch de 24 puertos 1G cobre + 4 puertos de subida de 10G SFP+
Switch para servidor	2	Switch de 24 subida de 10G SFP+
Almacenamiento	1	Conexión FC 4 puertos 8G + 10 Discos de 5x1.92TB SDD
Servidor	2	Soporte de hipervisor VMware, 2 puertos FC 8G, 2 puertos SFP+ 10G, 2xCPU Intel 4214, 1 x SD 32GB para arranque Interno y 96GB de RAM

Fuente: Elaboración propia

La cantidad de access points están sugeridas bajo la norma ANSI/TIA-4966 [37], la norma sugiere el uso de 21 Access Point para la cantidad de 500 personas, el cual asumiremos como un dispositivo como persona para una situación práctica. Los switches de acceso serán de 24 puertos de 1G cobre más 4 puertos de 10G SFP+ como subida, se considera 2 de estos switches para mantener la alta disponibilidad.

En primer lugar, tenemos al servidor que requerirá de switches dedicados para no saturar el tráfico en los switches de acceso. Estos switches para los servidores deberán ser de 24 puertos de 10G SFP+ y se considerara la cantidad de 2 para mantener la alta disponibilidad. En segundo lugar, necesitaremos 2 servidores que mantengan la alta disponibilidad del aplicativo, por lo cual, se aconseja que sean de la misma capacidad como se especifica en la tabla 14 y a su vez mantenga una conexión redundante hacia el almacenaje con las capacidades indicadas que están basadas en los valores obtenidos en la tabla 9.

Los costos reflejados en la tabla 15 es para un proyecto con 5 años de alcance, por lo cual, se adquiere el soporte de los equipos para la cantidad de años asignada.

Tabla 15 Costo de inversión en infraestructura local para el proyecto a 5 años

	Descripción	Precio Unitario	Cantidad	Precio Total
Switch de acceso	Switch Catalyst 9200L-E	$ 2600	2	$ 5200
Switch datacenter	Nexus 3524-x	$ 5350	2	$ 10700
Servidor	UCS-5180 + 2xUCSB200M5	$ 22000	1	$ 22000
Access Points	Meraki MR-33	$ 300	21	$ 6300
Storage	Flash system 5010E Storwize	$ 12000	1	$ 12000
Implementación	Servicios de implementación y soporte más horas de soporte y mantenimiento	$ 5000	1	$ 5000
PRECIO VALOR TOTAL (No incluye IGV)				$ 61,200.00

Fuente: Elaboración propia

En la tabla 16, se costeó el escenario de nube publica con instancias dedicadas con proyección y soporte a 5 años bajo el servicio de AWS IoT Core. Para realizar el costeo en este servicio se debe delimitar ciertos parámetros básicos como disponibilidad, concurrencia, peso, número de instancias y registros que deban hacer los dispositivos para mantener el plano de control de manera correcta y funcional.

Tabla 16 Costo de inversión en infraestructura nube para el proyecto a 5 años

	Descripción	Precio Unitario	Cantidad	Precio Total
Infraestructura Cloud	500 dispositivos - 75% de disponibilidad al día - 1 mensaje por minuto – 2 registros por día por dispositivo	$ 69,235.00	1	$ 69,235.00
Switch de acceso	Switch Catalyst 9200L-E	$ 2600.00	2	$ 5200.00
Access Points	Meraki MR-33	$ 300.00	21	$ 6300.00
Implementación	Servicios de implementación más horas de soporte y mantenimiento	$ 5000.00	1	$ 5000.00
PRECIO VALOR TOTAL (No incluye IGV)				$ 85,735.00

Fuente: Elaboración propia

Ambos proyectos de infraestructura en un arrendamiento a 5 años tendrían un costo mensual de $ 1485.8 y $ 2081.44 respectivamente bajo una tasa de interés de 2.428%, lo cual nos da a notar en el caso de una empresa mediana le es más costo-efectivo la adquisición del entorno local. Sin embargo, considerar que los costos brindados son referenciales y la arquitectura sujeta a cambios. Además, el retorno de inversión dependerá de los casos de uso que se vaya a dar a la plataforma, a pesar de esto, siempre se buscará la viabilidad económica del proyecto y que el costo no exceda las ganancias previstas por el uso de este.

Las plataformas IoT no solo buscan mejorar el retorno de inversión y optimizar procesos. Al estar ligado a la transformación digital esto permite que la empresa pueda usar la información de sus procesos mucho más allá de la utilidad alguna vez vista. Por ejemplo, en azucareras y plantas industriales en general, buscan optimizar gastos de petróleo y sus derivados. Estos gastos se optimizan mediante análisis de la información del comportamiento recibido a través de sensores. Por lo cual, las empresas están muy interesadas en estos impactos debido a que hoy en día son evaluadas bajo métricas de cuan eco amigables son frente al entorno que lidia con sus procesos brindándoles ciertos beneficios.

5. CONCLUSIONES

- De los resultados obtenidos se evidencia que los comportamientos de las plataformas a distintos escenarios están lejos de una linealidad y que el presente estudio hizo evidente que se requerirán entornos de red bastante robustos para poder solventar estas cargas. Por lo tanto, el análisis de los datos mediante el benchmarking nos ayuda a visualizar las mejores prácticas a distintos escenarios y a su vez conocer de antemano el rendimiento esperado a tales ambientes dando así una herramienta a las empresas para que puedan evaluar esta nueva tecnología.

- El análisis realizado de la data obtenida del uso promedio de CPU evidencia una falla critica en estas plataformas al no realizar un correcto uso de los múltiples núcleos. El encolamiento era mayor dentro de los procesos de reloj a pesar de asignarle una cantidad mayor de CPU virtual como parte de pruebas extras. Por lo cual, se recomienda que futuras plataformas o nuevas versiones

de las existentes posean un buen uso del procesamiento de la información debido a que es crítico para establecer todo el proceso de comunicación y poseer una mayor capacidad de mensajes concurrentes.

- De las plataformas evaluadas se observa que soportan cargas de 1000 a 10,000 dispositivos de una manera correcta. Estas presentan un pico de consumo de 80% de utilización del CPU lo cual es un valor limite que se establece en las buenas prácticas.

- Se esperaba una alta eficiencia para la transmisión de mensajes debido al uso de MQTT como protocolo de comunicación, sin embargo, se observó una eficiencia del 53% para 1000 mensajes reduciendo dramáticamente hasta 16% en 10000 mensajes y en posterior se mantuvo una eficiencia menor al 1%. Sin embargo, se pudo aumentar la eficiencia realizando un cambio en el QoS del protocolo MQTT, pero se añadía una carga aún mayor al procesamiento en el sistema por lo cual no se siguió este camino.

- El uso del índice de evaluación nos permitió tener un valor estimado del comportamiento de las plataformas frente a otras, por lo cual, esto ayudara a próximos desarrollos a que tengan una herramienta de comparación para optimizar procesos y código como se observó con los valores obtenidos del diseño de la nueva plataforma.

6. RECOMENDACIONES

- Dentro de los ambientes propuestos se observa que el consumo de almacenamiento evaluado a 1, 3 o 5 años es muy grande, por lo cual, se sugiere el uso de compresión y deduplicación integrada en el almacenaje o integración con software de terceros, esto ayudara en el espacio efectivo usado y evitar la saturación de discos debido a que la información recibida por los dispositivos es muy parecida por lo cual se vería muy beneficiada con la compresión y deduplicación.

- Como diseño de la arquitectura de la plataforma IoT, se recomienda que los middlewares sean el único centralizador de mensajes para distintos tipos de protocolos como HTTP, MQTT, AMQP o CoAP debido a que cada dispositivo y plataforma tienen un tipo de estructura y sintaxis propio para realizar él envió de la información. Esto se podrá analizar en próximos estudios buscando un escenario modular a nivel de lenguaje de programación, protocolo y dispositivos mejorando así la escalabilidad y la integración del sistema.

- Complementario al tipo de evaluación que se realizó en el presente estudio, se recomienda una evaluación granular desde 1 mensaje hasta 100,000 mensajes entrantes por segundo de manera que se pueda visualizar y evaluar en que momentos existen quiebres en la linealidad de la data obtenida. Esto no se pudo realizar en la presente tesis debido a las limitantes de software como generador de carga.

Bibliografía:

[1] T. Jell, S. Ag, A. Bröring, and S. Ag, “BIG IoT – Interconnecting IoT Platforms from different domains The Problem of Missing IoT Interoperability,” pp. 98–101, 2017.

[2] IoT Analytics, “IoT platforms. The central backbone for the internet of things.,” no. November, pp. 1–24, 2015, [Online]. Available: http://www.iot-analytics.com/wp/wp-content/uploads/2016/01/White-paper-IoT-platforms-The-central-backbone-for-the-Internet-of-Things-Nov-2015-vfi5.pdf.

[3] E. Varga, “Unified IoT Platform Architecture,” pp. 6–13, 2018.

[4] “Intel estima que el mercado de chips para IoT será muy fragmentado Negocios | El Comercio Perú.” https://elcomercio.pe/economia/negocios/intel-estima-mercado-chips-iot-sera-fragmentado-noticia-644005-noticia/ (accessed Jun. 11, 2020).

[5] H. Hejazi, H. Rajab, T. Cinkler, and L. Lengyel, “Survey of platforms for massive IoT,” *2018 IEEE Int. Conf. Futur. IoT Technol. Futur. IoT 2018*, vol. 2018-Janua, pp. 1–8, 2018, doi: 10.1109/FIOT.2018.8325598.

[6] L. Nikityuk and R. Tsaryov, “Optimization of the Process of Selecting of the IoT-Platform for the Specific Technical Solution IoT-Sphere,” *2018 Int. Sci. Conf. Probl. Infocommunications Sci. Technol. PIC S T 2018 - Proc.*, pp. 401–405, 2019, doi: 10.1109/INFOCOMMST.2018.8632088.

[7] B. T. Mi, X. Liang, and S. Sen Zhang, “A Survey on Social Internet of Things,” *Jisuanji Xuebao/Chinese J. Comput.*, vol. 41, no. 7, pp. 1448–1475, 2018, doi: 10.11897/SP.J.1016.2018.01448.

[8] J. Guth, U. Breitenbucher, M. Falkenthal, F. Leymann, and L. Reinfurt, “Comparison of IoT platform architectures: A field study based on a reference architecture,” *2016 Cloudification Internet Things, CIoT 2016*, pp. 1–6, 2017, doi: 10.1109/CIOT.2016.7872918.

[9] J. B. Aspiazu and H. E. Hernandez-Figueroa, “Middleware Design for Application Integration in IoT Networks,” *Proc. - 2017 Int. Conf. Comput. Sci. Comput. Intell. CSCI 2017*, no. 1, pp. 1326–1331, 2018, doi: 10.1109/CSCI.2017.231.

[10] T. Yokotani, “Requirements on the IoT communication platform and its standardization,” *2017 Proc. Japan-Africa Conf. Electron. Commun. Comput. JAC-ECC 2017*, vol. 2018-Janua, pp. 1–4, 2018, doi: 10.1109/JEC-ECC.2017.8305765.

[11] “AWS Case Study: LG Electronics.” https://aws.amazon.com/solutions/case-studies/lg-electronics/ (accessed Jun. 12, 2020).

[12] “AWS IoT Core Overview - Amazon Web Services.” https://aws.amazon.com/iot-core/ (accessed Jun. 13, 2020).

[13] U. Discovery *et al.*, “UDAP Specifications (For Second Screen TV and Companion Apps),” vol. 0, pp. 1–77, 2013.

[14] T. Yokotani and Y. Sasaki, “Comparison with HTTP and MQTT on required network resources for IoT,” *ICCEREC 2016 - Int. Conf. Control. Electron. Renew. Energy, Commun. 2016, Conf. Proc.*, pp. 1–6, 2017, doi: 10.1109/ICCEREC.2016.7814989.

[15] “How Many IoT Devices Are There in 2020? More than Ever!” https://techjury.net/blog/how-many-iot-devices-are-there/#gref (accessed Jun. 21, 2020).

[16] “IoT: Usability Dream or Privacy Nightmare? | AT&T Cybersecurity.” https://cybersecurity.att.com/blogs/security-essentials/iot-usability-dream-or-privacy-nightmare (accessed Jun. 21, 2020).

[17] T. Aziz and E. Haq, “Security Challenges Facing IoT Layers and its Protective Measures,” *Int. J. Comput. Appl.*, vol. 179, no. 27, pp. 31–35, 2018, doi: 10.5120/ijca2018916607.

[18] K. Angrishi, “Turning Internet of Things(IoT) into Internet of Vulnerabilities (IoV) : IoT Botnets,” pp. 1–17, 2017, [Online]. Available: http://arxiv.org/abs/1702.03681.

[19] N. Vidgren, K. Haataja, J. L. Patiño-Andres, J. J. Ramírez-Sanchis, and P. Toivanen, “Security threats in ZigBee-enabled systems: Vulnerability evaluation, practical experiments, countermeasures, and lessons learned,” *Proc. Annu. Hawaii Int. Conf. Syst. Sci.*, pp. 5132–5138, 2013, doi: 10.1109/HICSS.2013.475.

[20] K. K. Kolluru, C. Paniagua, J. Van Deventer, J. Eliasson, J. Delsing, and R. J. Delong, “An AAA solution for securing industrial IoT devices using

next generation access control," *Proc. - 2018 IEEE Ind. Cyber-Physical Syst. ICPS 2018*, pp. 737–742, 2018, doi: 10.1109/ICPHYS.2018.8390799.

[21] C. J. Fèret, "'Talos.,'" *Notes Queries*, vol. s8-X, no. 255, p. 397, 1896, doi: 10.1093/nq/s8-X.255.397-c.

[22] "An overview of HTTP - HTTP | MDN." https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview (accessed Jun. 21, 2020).

[23] C. C. Goh, E. Kanagaraj, L. M. Kamarudin, A. Zakaria, H. Nishizaki, and X. Mao, "IV-AQMS: HTTP and MQTT protocol from a realistic testbed," *2019 IEEE Int. Conf. Sensors Nanotechnology, SENSORS NANO 2019*, pp. 19–22, 2019, doi: 10.1109/SENSORSNANO44414.2019.8940094.

[24] "MQTT vs. HTTP: which one is the best for IoT? - MQTT Buddy - Medium." https://medium.com/mqtt-buddy/mqtt-vs-http-which-one-is-the-best-for-iot-c868169b3105 (accessed Jun. 21, 2020).

[25] J. Huang, P. Tsai, and I. Liao, "Fog Computing Environment," pp. 198–203, 2017.

[26] "MQTT Packet Format with examples - OpenLabPro.com." https://openlabpro.com/guide/mqtt-packet-format/ (accessed Jun. 25, 2020).

[27] H. W. Van Der Westhuizen and G. P. Hancke, "Practical comparison between COAP and MQTT - Sensor to server level," *2018 Wirel. Adv. WiAd 2018*, 2018, doi: 10.1109/WIAD.2018.8588443.

[28] I. T. Europe, "Welcome to Connected Industry – Day 1 Today ' s ' Connected Industry ' lineup," no. June, 2018.

[29] N. Economides and E. Katsamakas, "Two-sided competition of proprietary vs. open source technology platforms and the implications for the software industry," *Manage. Sci.*, vol. 52, no. 7, pp. 1057–1071, 2006, doi: 10.1287/mnsc.1060.0549.

[30] V. Araujo Soto, "Performance evaluation of scalable and distributed IoT platforms for Smart Regions," p. 103, 2017.

[31] "Apache JMeter - Apache JMeter™." https://jmeter.apache.org/ (accessed Jun. 25, 2020).

[32] 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.

[33] “HTTP vs. MQTT: A tale of two IoT protocols | Google Cloud Blog.” https://cloud.google.com/blog/products/iot-devices/http-vs-mqtt-a-tale-of-two-iot-protocols (accessed Jun. 25, 2020).

[34] “ThingsBoard IoT Platform deployment scenarios | ThingsBoard.” https://thingsboard.io/docs/reference/iot-platform-deployment-scenarios/ (accessed Jun. 18, 2020).

[35] “GitHub - sitewhere/sitewhere: .” https://github.com/sitewhere/sitewhere (accessed Jul. 17, 2020).

[36] “System installation · Kaa.” https://kaaproject.github.io/kaa/docs/v0.10.0/Administration-guide/System-installation/ (accessed Jun. 18, 2020).

[37] “ANSI/TIA-4966: Telecommunications Infrastructure Standard for Educational Facilities | Standards Informant » Standards Informant.” https://blog.siemon.com/standards/ansitia-4966-telecommunications-infrastructure-standard-for-educational-facilities (accessed Jul. 21, 2020).