ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

ANÁLISIS COMPARATIVO DE DOS PLATAFORMAS MIDDLEWARE OPEN SOURCE FUNDAMENTADO EN MÉTRICAS CUALITATIVAS BASADO EN EL FRAMEWORK SMI (SERVICE MEASUREMENT INDEX)

TRABAJO DE INTEGRACIÓN CURRICULAR PRESENTADO COMO REQUISITO PARA LA OBTENCIÓN DEL TÍTULO DE INGENIERA EN TECNOLOGÍAS DE LA INFORMACIÓN

VERÓNICA GABRIELA MALDONADO MORALES

veronica.maldonado@epn.edu.ec

DIRECTOR: Ph.D. ANA MARÍA ZAMBRANO VIZUETE

ana.zambrano@epn.educ.ec

DMQ, abril 2023

CERTIFICACIONES

Yo, VERÓNICA GABRIELA MALDONADO MORALES declaro que el Trabajo de Integración Curricular aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

VERONICA GABRIELA MALDONADO MORALES

Certifico que el presente trabajo de integración curricular fue desarrollado por VERÓNICA GABRIELA MALDONADO MORALES, bajo mi supervisión.

PhD. ANA MARÍA ZAMBRANO VIZUETE

DIRECTOR

DECLARACIÓN DE AUTORÍA

A través de la presente declaración, afirmamos que el trabajo de integración curricular aquí descrito, así como el (los) producto(s) resultante(s) del mismo, son públicos y estarán a disposición de la comunidad a través del repositorio institucional de la Escuela Politécnica Nacional; sin embargo, la titularidad de los derechos patrimoniales nos corresponde a los autores que hemos contribuido en el desarrollo del presente trabajo; observando para el efecto las disposiciones establecidas por el órgano competente en propiedad intelectual, la normativa interna y demás normas.

VERÓNICA GABRIELA MALDONADO MORALES

Ph.D. ANA MARÍA ZAMBRANO VIZUETE

DEDICATORIA

Este trabajo está dedicado a Rosaura, mi madre, el pilar fundamental de mi vida; si no fuera por su esfuerzo, dedicación, amor, desvelos, resiliencia y enseñanzas este logro no habría sido posible. Te amo mami.!

A mis hermanos Kléber, Vladimir y Diego, que me motivaron a seguir adelante, y en quienes siempre encuentro apoyo.

A Oswaldo mi padre (+) que a pesar de su ausencia y de lo mucho que me hace falta, su cariño y enseñanzas han sido la luz que ha guiado mi camino. De mi para ti hasta el cielo.!

Gabriela Maldonado

AGRADECIMIENTO

Agradezco a Dios por sanarme, por bendecir mi vida y la de los míos, por darme la fuerza para superar las dificultades y finalmente poder concluir con esta meta.

A mi familia, por haber creído en mí, dándome ejemplo de perseverancia y superación. Gracias por su paciencia y apoyo incondicional a lo largo de esta etapa de mi vida.

A Teresa mi tía (+) quien fue como mi segunda madre, gracias por el cuidado, amor y apoyo que me diste durante tus días de vida.

Un agradecimiento especial a mi Directora de Proyecto, Ph.D. Ana María Zambrano por su apoyo, comprensión, consejos, tiempo y sobre todo por no dejarme claudicar ante las adversidades.

Agradezco el haber coincidido en aquella etapa de mi vida con excelentes personas a los que hoy por hoy considero verdaderos amigos.

Gabriela Maldonado

ÍNDICE DE CONTENIDO

RESUMEN

El impacto de cada generación TIC (Tecnologías de la Información) en nuestras actividades y entornos es cada vez mayor. Según el *IoT Platforms Market*, el mercado de plataformas IoT está presente en varios sectores; sin embargo, aún está muy fragmentado [1]. Esto y un mercado en constante evolución con un gran abanico de posibilidades, puede incurrir en una difícil decisión al momento de elegir la solución que mejor responda a las necesidades de una organización.

Este trabajo pretende ser un estudio que aporte conocimiento técnico a través de un análisis comparativo de las plataformas *OpenIoT y DeviceHive,* bajo un marco de evaluación utilizando el *framework* SMI (*Service Measurement Index*) propuesto por CSMIC (*Cloud Service Measurement Initiative*) basado en métricas cualitativas que permitan identificar y facilitar un criterio en cuanto a la elección de las distintas herramientas tecnológicas. Además, de orientar a que estudiantes y docentes puedan profundizar en el mundo de IoT en proyectos futuros.

El Capítulo 1 aborda los conceptos fundamentales que giran en torno al mundo IoT, incluidas las tecnologías que le permiten funcionar óptimamente. Adicionalmente se exponen de los conceptos principales del SMI.

El Capítulo 2 presenta el estudio de cada uno de los componentes de las plataformas mencionadas, así como también las distintas herramientas que permiten su implementación. Adicionalmente se elabora un manual de usuario.

El Capítulo 3 muestra las capacidades y características primordiales de las plataformas mencionadas, seguido de un análisis comparativo mediante una valoración de métricas cualitativas, junto con sus respectivas conclusiones y recomendaciones.

PALABRAS CLAVE: Plataformas IoT, SMI, métricas cualitativas, análisis comparativo, *OpenIoT, DeviceHive*.

ABSTRACT

The impact of each generation ICT (Information CommunicationsTechnology) on our activities and environments is increasing. According to the IoT Platforms Market, the IoT platform market is present in several sectors; however, it is still very fragmented [1]. This and a constantly evolving market with a wide range of possibilities, can incur a difficult decision when choosing the solution that best responds to the needs of an organization.

This project intends to be a study that provides technical knowledge through a comparative analysis of the OpenIoT and DeviceHive platforms, under an evaluation framework using the SMI (Service Measurement Index) framework proposed by CSMIC (Cloud Service Measurement Initiative) based on qualitative metrics that allow identifying and facilitating a criterion regarding the choice of different technological tools. In addition, to guide students and teachers to deepen the world of IoT in future projects.

Chapter 1 addresses the fundamental concepts that revolve around the IoT world, including the technologies that enable it to function optimally. Additionally, the main concepts of the SMI are exposed.

Chapter 2 presents the study of each of the components of the aforementioned platforms, as well as the different tools that allow their implementation. Additionally, a user manual is prepared.

Chapter 3 shows the capabilities and main characteristics of the aforementioned platforms, followed by a comparative analysis through an assessment of qualitative metrics, together with their respective conclusions and recommendations.

KEYWORDS: IoT platforms, SMI, qualitative metric, comparative analysis, OpenIoT, DeviceHive.

1 INTRODUCCIÓN

El mundo Internet de las Cosas (IoT por sus siglas en inglés *Internet of Things*) continúa en expansión, y gestionar los despliegues de forma sencilla y eficiente para poder aprovechar los datos, se ha vuelto algo indispensable; y esto a su vez ha desencadenado en una gran variedad de plataformas IoT convirtiéndose en herramientas de interés para varios sectores. Este nuevo universo conectado presenta diferentes retos y uno de ellos es el saber elegir la solución que mejor se adapte a las necesidades de una organización. No obstante evaluar ese gran abanico de posibilidades implica un arduo trabajo no solo por la cantidad representativa de plataformas que oferta el mercado TI, sino también debido a las distintas herramientas y componentes en el que se desarrollan estás tecnologías.

El *framework* SMI (*Service Measurement Index)* propuesto por el *Cloud Services Measurement Initiative Consortium* (CSMIC) [2], permite hacer una comparación relativa significativa de los Servicios de TI sobre aquellos servicios que son funcionalmente similares. Por consiguiente, se plantea realizar un análisis comparativo entre dos plataformas IoT *OpenSource: OpenIoT y DeviceHive*, mediante un estudio específico basada en métricas cualitativas que permitirá evaluar dichas plataformas con el objetivo presentar una perspectiva y criterio de evaluación al momento de escoger un servicio IoT.

1.1 OBJETIVO GENERAL

El objetivo general de este Proyecto Curricular es analizar comparativamente plataformas *Middleware OpenSource* con fundamento en métricas cualitativas basado en el *framework* SMI (*Service Measurement Index).*

1.2 OBJETIVOS ESPECÍFICOS

Los objetivos específicos del Proyecto Curricular son:

1. Analizar el *framework* Índice de Medición del Servicio (SMI) detallando sus métricas cualitativas.
2. Analizar las características principales y específicas de las plataformas *OpenIoT y DeviceHive.*
3. Analizar los componentes indispensables para el despliegue de las plataformas *OpenIoT* y *DeviceHive*.
4. Analizar comparativamente las plataformas *OpenIoT y DeviceHive*, en base a las métricas cualitativas del *framework* SMI.

1.3 ALCANCE

Tomando en cuenta que, para crear un entorno inteligente se requiere de una arquitectura distribuida compleja, el Internet de las Cosas (IoT) utiliza un sistema denominado Middleware el cual se convierte en un componente necesario de esta tecnología y por ende de las plataformas IoT. Por tal motivo se abordará la importancia de este sistema analizando la Ingeniería de Software en IoT, seguido de un Estudio del Arte de las plataformas IoT.

Posteriormente se realizará un análisis comparativo entre las plataformas *OpenSource*: *OpenIoT y DeviceHive*. Además, se expondrán las características esenciales y ámbitos de aplicación, detallando y presentando un cuadro con propiedades y peculiaridades como: monitoreo, alertas, almacenamiento de datos, empresa de soporte, licencia, documentación, *committers*, entre otros. Se profundizará en la arquitectura, servicios, hardware y software idóneo y los componentes necesarios para el despliegue de las plataformas mencionadas.

Finalmente se presentará una tabla de las métricas cualitativas de acuerdo al *framework* Índice de Medición del Servicio (SMI) siendo en primera instancia obligatorio instalar los componentes de las plataformas seleccionadas para poder realizar este análisis comparativo. Los atributos cualitativos se medirán manejando una escala ordinal que se basa en un conjunto de calificaciones predefinidas, como excelente, alta, media, buena.

1.4 MARCO TEÓRICO

En el primer capítulo se reflejarán los conceptos básicos que rodean el IoT. Las significativas vulnerabilidades en este ambiente ponen en riesgo la seguridad y la integridad de la información y deben ser contrarrestadas, por lo tanto, el primer punto a abordar serán las bases sobre la Ingeniería de Software. El enfoque principal son las plataformas IoT, razón por la cual, se realizará un Análisis del Arte, y se puntualizará en el software y hardware requeridos por esta tecnología.

Para comprender el funcionamiento de estas plataformas, es necesario el estudio de algunas herramientas complementarias, ampliamente utilizadas en el mundo de IoT, las cuales posteriormente se irán detallando para finalmente realizar un análisis del *framework* SMI, mismo que permitirá realizar posteriormente el análisis comparativo entre dos plataformas IoT, las cuales se tomarán como muestra de estudio. Exponer el marco teórico relevante relacionado con el tema, incluyendo los argumentos que justifican la validez de lo realizado, con una revisión bibliográfica pertinente.

1.4.1 INGENIERÍA DE SOFTWARE PARA IOT

Esta nueva era, la de un mundo interconectado requiere nuevos enfoques para las técnicas estándar y una nueva generación de entornos de desarrollo de software para IoT. La mayoría de sistemas de misión crítica en IoT requieren técnicas de verificación y validación altamente escalables. Para superar estas dificultades, surgió por primera vez la denominada “Ingeniería de Software” después de dos conferencias patrocinadas por el Comité de Ciencia de la OTAN (Organización del Tratado del Atlántico Norte) celebrada en *Garmisch* (Alemania) en los años 1967 y 1968 [3].

La Ingeniería de Software tiene varias definiciones, sin embargo, estas coinciden en que la Ingeniería de Software es una disciplina de la ingeniería basada en la ciencia y las matemáticas, la cual no está limitada a simples procesos técnicos para el desarrollo de software, sino que además incluye teorías, métodos, ciclo de vida, costos y herramientas necesarias que proporcionan soluciones a los problemas de software.

De acuerdo al estudio “*Sobre los desafíos en ingeniería de sistemas de software IoT*” [4], un sistema de software de IoT debería considerar siete facetas diferentes que son: conectividad, cosas, comportamiento, inteligencia, dominio del problema, medio ambiente e interactividad; desafíos que permitirán abordar la mencionada multidisciplinariedad. Las etapas de la Ingeniería de Software, están directamente relacionadas una con la otra tal como se muestra en la **Figura 1.1**.

Figura 1.1. Ciclo de Vida de la Ingeniería de Software.

Estas etapas pretenden de una manera u otra, proporcionar una guía ordenada de los procesos facilitando y optimizando el desarrollo de software. Por lo tanto, es conveniente cumplir con las tareas de cada una de las etapas y no dar continuidad en el proceso si no se ha cumplido con la etapa anterior. El ciclo de vida de la Ingeniería de Software consta de numerosas tareas agrupadas en siete etapas descritas en la **Tabla 1.1.**

Tabla 1.1. Etapas de la Ingeniería de Software [4].

Análisis	Se investigará el problema, definiéndolo claramente y el procedimiento a aplicar, así como los elementos principales del producto.
Diseño	Se empleará la información recopilada en la etapa de análisis para crear modelos o rasgos precisos para productos o componentes del sistema.
Desarrollo	Los diseños creados son utilizados para desarrollar los elementos utilizados en el sistema.
Pruebas o Verificación	Garantiza que los elementos individuales que conforman un producto o sistema, cumplen con las especificaciones requeridas diseñadas durante la fase de diseño.
Implementación o Entrega	Sé distribuirá el producto hasta las manos del cliente.
Mantenimiento	Sé aplicarán las soluciones adecuadas a cualquier complicación del producto y este será re- liberado en su versión mejorada.
EOL (*End-of-Life*)	Todas las tareas se realizan para garantizar que los clientes y empleados tengan la certeza de que el producto ya no estará disponible, por lo tanto, no estará a la venta.

1.4.2 ESTADO DEL ARTE

El objetivo principal de este Proyecto Curricular es el análisis de plataformas IoT, las cuales son parte principal del mundo IoT. El papel clave de estas tecnologías, la amplia investigación y el desarrollo de productos en esta área, muestran la notabilidad de las plataformas IoT, y por consiguiente la importancia de presentar un Estado del Arte.

1.4.2.1 Definición de Plataforma IoT

Cada día son más los servicios y productos que ofrece el IoT por medio de las plataformas en la “nube” como en la “no nube”. Sin embargo, hoy por hoy no existe una definición única o estandarizada de las plataformas IoT. Por consiguiente, se presentará la definición de “Plataforma” desde dos perspectivas: Código Abierto e Investigación Científica.

a. Perspectiva Código Abierto

Una plataforma *Open Source*, también llamada de Código Abierto, es el término empleado a la plataforma distribuida bajo una licencia de libre acceso, donde los usuarios pueden utilizarla de forma independiente. Por lo tanto, una vez obtenida la plataforma, puede ser aprovechada, investigada, modificada y redistribuida libremente.

El concepto de “código abierto” se originó a partir del software libre, y lleva ese nombre por *Free Software Foundation*, fundada por *Richard Stallman* en 1984 [5]. Es necesario dejar en claro, que el concepto de libre no implica que sea gratuito, por lo contrario, significa que el usuario puede usarlo y adaptarlo a sus necesidades, siempre y cuando posea los conocimientos específicos y necesarios. Algunas de las características más famosas a los principios del software libre en plataformas *Open Source* son [5]:

- El programa se puede utilizar para cualquier propósito.
- Estudiar cómo funciona el programa y adaptarlo a las necesidades.
- Distribuir copias, mejorar el programa y publicar estas mejoras.

Un software es considerado *Open Source* siempre y cuando la licencia asegure los siguientes aspectos detallados en la **Tabla 1.2**.:

Tabla 1.2. Aspectos de *Open Source* [6].

Redistribución gratuita	Está prohibida la venta y distribución del software o parte de él. No se puede exigir el pago de regalías u otra tasa por su distribución.
Código fuente	El programa debe contener el código fuente, admitir la distribución de éste como de la forma compilada. De no ser así, ofertar su obtención por un costo moderado o a través de una descarga de Internet sin cargo.
Obras derivadas	No se puede impedir realizar modificaciones al programa y trabajos derivados y estas deben ser distribuidas bajo los mismos términos que el software original.
Integridad del código fuente del autor	Se puede impedir que el código fuente sea distribuido en forma modificada solamente si permite la distribución de "archivos parches", con el objetivo de modificar el programa durante el desarrollo.
No discriminación contra personas o grupos	Las condiciones de uso del programa no pueden discriminar a personas o grupos de personas.
No discriminación contra los campos de trabajo	No se puede negar el uso del programa a ninguna persona bajo ningún fin: personal, comercial, empresarial, militar, etc.
Distribución de licencia	Los derechos del programa deben ser aplicados a todos los que lo redistribuyan.
La licencia no debe ser específica para un producto	Los derechos otorgados a los usuarios de un programa no pueden depender de si el programa es parte de un paquete o versión de software en particular.
La licencia no debe restringir otro software	No debe limitarse el uso de otros programas distribuidos con el software licenciado.
La licencia debe ser tecnológicamente neutra	No se puede forzar el uso de una tecnología concreta.

b. Perspectiva de Investigación Científica

Definiciones con diferentes puntos de vista sobre las plataformas IoT han sido plasmadas en artículos científicos y son expuestas en la **Tabla 1.3**. La mayoría de estas definiciones hacen hincapié en la infraestructura y su papel principal como interfaz entre dispositivos y usuarios finales, responsable de gestionar y controlar dispositivos; mientras que, en algunas definiciones, la función predominante de las plataformas, es proveer un conjunto de herramientas de desarrollo e implementación de aplicaciones y otras tantas, han considerado el sistema operativo integrado, el *firmware* y la puerta de enlace en los dispositivos como una plataforma [7].

Tabla 1.3. Definiciones de Plataforma IoT [7].

	Definición
J. Mineraud et. al	Definen a una plataforma como un *Middleware* e infraestructura que proporciona conectividad entre el usuario final y las cosas inteligentes.
I. Ganchev et. al	Definen las principales características generales y esenciales de una plataforma IoT, como: monitorización, servicios conectividad ubicua, control y alerta, programación, envío, protocolos, programas, seguridad y mantenimiento.
B. Nakhuva et. al	Definen la plataforma IoT como una solución virtual en la nube que provee un conjunto completo de funciones independientes de la aplicación para el desarrollo, despliegue y mantenimiento de aplicaciones específicas de IoT. La solución traduce los datos en información útil, la analiza, toma decisiones, realiza mantenimiento predictivo, pago basado en el usuario y gestión de datos en tiempo real.
Abdollahei	Define la plataforma IoT como una interfaz entre los dispositivos y los usuarios finales, que provee automatización, monitorización y gestión de forma centralizada, utilizando diferentes tecnologías como la computación en la nube, análisis de big data y virtualización de la red.
J. Guth	Propone una arquitectura general de referencia para el ecosistema IoT cuya funcionalidad principal es la integración de IoT *Middleware*. El *Middleware* sirve como capa de integración para dispositivos y aplicaciones, y es responsable de la gestión de dispositivos/usuarios, y de procesar los datos.

De lo expuesto anteriormente podemos concluir que la Perspectiva de Código Abierto hace énfasis en lo indispensable y necesario que se vuelve el aprovechar el libre acceso a las plataformas IoT, ya que éstas son el soporte de esta nueva era tecnológica, mientras que la Perspectiva Científica se enfoca en la capacidad que tienen las plataformas IoT para proporcionar conexión e intercambio de información entre dispositivos, aplicaciones y usuarios finales por medio de componentes de software, ya sea a través de los recursos de la nube o no.

1.4.2.2 *Middleware*

El *Middleware* de integración de IoT es responsable de recibir datos de los dispositivos conectados para procesarlos, enviarlos a las aplicaciones conectadas y controlar los dispositivos en términos de envío de comandos que ejecutarán los actuadores respectivos. Un dispositivo se comunica directamente con el *Middleware* si admite una tecnología de comunicación adecuada, como WiFi, un protocolo de transporte como HTTP (*Hypertext Transfer Protocol*) o MQTT (*Message Queue Telemetry Transport*), y un formato de carga compatible, como JSON (*JavaScript Object Natation)* o XML (*Extensible Markup Language*). De otro modo, el dispositivo se comunicará con el *Middleware* mediante una puerta de enlace. Las razones mencionadas anteriormente generan la necesidad de varios componentes funcionales que el IoT *Middleware* debe soportar y se muestra en la **Figura 1.2** y se detallan a continuación en la **Tabla 1.4.**

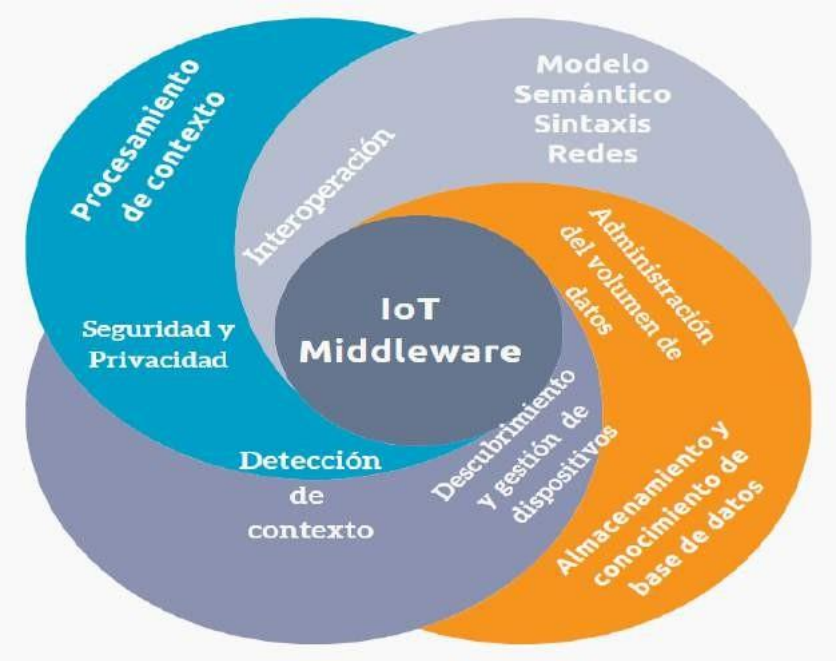

Figura 1.2. Componentes funcionales de IoT *Middleware* [8].

Tabla 1.4. Componentes Funcionales de *Middleware.*

Interoperación	Comparte información y la utiliza a través de diversos dominios de aplicaciones. Se clasifica en tres categorías: red, sintáctica y semántica. La Interoperación en Red define protocolos para el intercambio de información a través de diferentes redes de comunicación, sin importar el contenido de esta. La Sintáctica se ocupa del formato y la estructura de la codificación de la información que se intercambia entre sí, mientras que la Semántica define las reglas para entender el significado del contenido de información con un modelo semántico.
Detección de contexto	La detección del contexto recopila datos e identifica los factores que tienen un impacto significativo en la respuesta. El procesamiento del contexto extrae los datos del contexto, los procesa y toma una decisión.
Descubrimiento y gestión de dispositivos	Permite a cualquier dispositivo de la red detectar todos los dispositivos vecinos y notificar su presencia. Con módulos fiables, tolerantes a fallos, adaptables y óptimos para el consumo de recursos.
Seguridad y privacidad	La seguridad puede efectuarse de dos maneras: comunicación abstracta y segura de alto nivel entre pares, y por gestión segura de la topología, encargada de la autenticación de nuevos pares, de los permisos de acceso a la red y de la protección de la información de enrutamiento y de la información que se intercambia en la red.
Administrar el volumen de datos	Los retos en cuanto a la exposición de la cantidad de datos recogidos e intercambiados son: la consulta, la indexación, el modelado de procesos y la gestión de transacciones.

Por consiguiente, un *Middleware* actúa como capa de integración para una amplia variedad de sensores, actuadores, dispositivos y aplicaciones. Las aplicaciones de diversos dominios exigen una capa de abstracción / adaptación. El *Middleware* proporciona una API (Interfaz de Programación de Aplicaciones) para la comunicación de la capa física y los servicios requeridos para las aplicaciones, ocultando todos los detalles de la diversidad.

1.4.2.3 Arquitectura *Middleware*

El término *Middleware* ha ido evolucionando, y actualmente como plataforma IoT, es responsable de conectar la parte del *Front-End* de una aplicación con la parte *Back-End* y proporcionar de forma transparente las acciones de alto nivel dependientes de una capa de bajo nivel.

La falta de estandarización de ciertos términos IoT como: “cosas” y si “cosas” y “dispositivos” son iguales, no ha permitido aplicar una arquitectura general; no obstante, es necesaria una arquitectura de referencia que provea una base para comparar diversas plataformas IoT. Se ha tomado la arquitectura de referencia de IoT abstracta introducida por el Trabajo [9] basado en varias plataformas de IoT de última generación tanto de código abierto como propietarias; esta provee un punto de vista neutro de los elementos de las plataformas IoT y sus posibles conexiones. La arquitectura propuesta consta de cinco capas con diferentes componentes y su interrelación tal como se observa en la **Figura 1.3**. Cabe recalcar que los componentes se presentan sin cardinalidad y pueden omitirse.

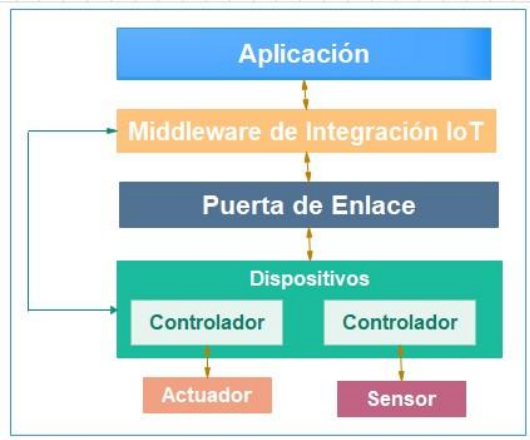

Figura 1.3. Arquitectura de referencia de IoT [9].

a. Actuador

Componente hardware que puede actuar mediante luz o sonido para manipular o controlar el entorno físico. Un Actuador recibe comandos de sus dispositivos conectados y convierte las señales eléctricas en alguna forma de acción física. Al igual que los sensores, los actuadores generalmente están conectados o incluso integrados en el dispositivo, y su conexión puede establecerse por cable o de forma inalámbrica.

b. Sensor

Está compuesto por un transductor, que permite detectar, medir o mostrar variaciones de una magnitud física concreta, transformarlas luego en señales eléctricas, analógicas

o digitales; y un acondicionador de señal, que amplifica dichas variaciones, convirtiéndolas a un formato fácil de comprender. Favorablemente han ido evolucionando en cuanto a tamaño, funcionalidad y en una reducción de costes. Sus desventajas continúan siendo el consumo de energía, la seguridad e interoperabilidad.

c. Dispositivo

Componente hardware, que se conecta a sensores y/o actuadores a través de cables o inalámbricamente, siendo la conexión entre el entorno físico y el mundo digital. Para procesar los datos de los sensores y controlar los actuadores, requiere de un software en forma de controladores.

d. Puerta de Enlace

Si un dispositivo no puede conectarse directamente a otros sistemas a través de un protocolo determinado o debido a limitaciones técnicas, lo hace por medio de una "puerta de enlace" denominado *Gateway*. Por lo tanto, una puerta de enlace es responsable de dar soporte a las tecnologías y protocolos de comunicación necesarios en ambas direcciones y de traducir los datos si es necesario.

1.4.3 PROTOCOLOS IOT

Los protocolos de comunicación cumplen un papel fundamental en el despliegue de las aplicaciones entre dispositivos IoT, ya que son la base para generar comunicación con el entorno mediante la información que reciben los sensores. Existe una variedad de protocolos empleados en IoT, sin embargo, en este Apartado se discutirán los más destacados, por lo cual se ha recolectado información acerca de estos protocolos y se ha realizado una comparativa enfocada en cuatro aspectos: modelo, utilidad, seguridad y accesibilidad; la cual es desplegada en la **Tabla 1.5.** y enmarcados en algunas de las categorías mostradas en la **Figura 1.4.**

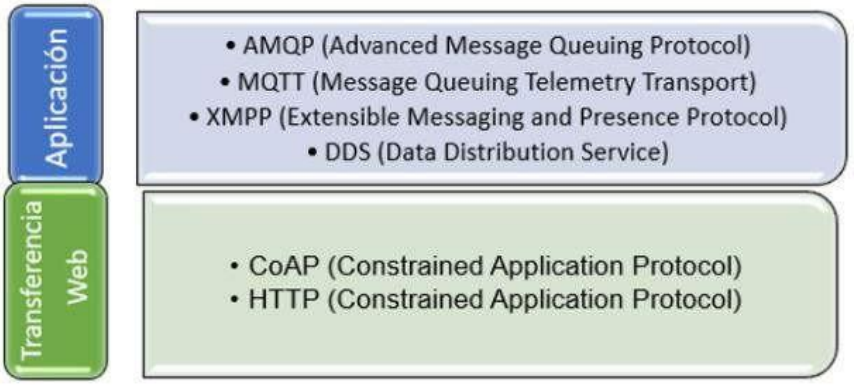

Figura 1.4. Protocolos Principales de IoT.

Tabla 1.5. Protocolos principales para IoT.

Protocolos	Modelo	Utilidad	Seguridad	Accesibilidad
HTTP	Cliente/servidor sin conexión.	De gran utilidad si se requiere enviar grandes cantidades de información.	HTTPS	Alta, debido a las innumerables herramientas de código abierto.
COAP (Protocolo de Aplicación Restringida)	Similar a HTTP, pero usa UDP/ multicast en lugar de TCP.	Frecuentemente, utilizado como tercer protocolo, luego de HTTP y MQTT.	DTLS	Opciones de software y hardware limitadas debido a su aceptación en el mercado.
MQTT	Publicar/ Suscribir	Carga MQTT específica para la aplicación. Diseñada para SCADA y redes remotas. Permite la comunicación con servidores.	SSL y TLS	Muchas plataformas IoT soportan HTTP y MQTT como los primeros dos protocolos de entrada de información.
XMPP (Protocolo Extensible de Mensajería y Comunicación de Presencia)	Petición/ Respuesta Publicar/ Suscribir	Diseñado originalmente para la mensajería en tiempo real, además es usado para: VOIP, vídeo, transferencia de archivos y juegos.	SASL Y TLS	Protocolo libre y abierto de la capa de aplicación. Usa datagramas en XML.
AMQP (Advanced Message Queuing Protocol)	Publicar/ Suscribir	Soporta transacciones, maneja un sistema de colas diseñado para la conexión entre servidores.	TLS	Altamente disponible para el intercambio de mensajes de misión crítica y empresarial entre diferentes plataformas.

1.4.4 HARDWARE Y SOFTWARE PARA IOT [10] [11]

El mundo IoT funciona con miles de millones de dispositivos interconectados, como ya se mencionó anteriormente, los mismos que deben cumplir con dos particularidades: ser de tamaño pequeño y de muy bajo consumo; para lo cual los componentes deben tener una menor potencia. Estas características son proporcionadas por los SoC (*System on Chip*). Un SoC es un circuito integrado que contiene todo lo necesario para el funcionamiento de un sistema: procesador, memoria RAM, almacenamiento, y controladores de E/S.

De acuerdo al *Open Source Hardware Association Open Hardware* o conocido también por sus siglas en inglés como OSHW *(Open Source Hardware),* es un término que permite denominar cualquier objeto tangible del mundo físico como un computador, un vehículo, etc., pero con la característica de que su principio de diseño tiene licencia para que cualquier persona pueda fabricar, modificar, distribuir y usar dicho objeto. Los términos de distribución de “Hardware de fuentes abiertas” deberán seguir los siguientes criterios detallados en la **Tabla 1.6.** [10].

Cabe recalcar que los criterios mencionados a continuación representan una manera de compartir información propuesta por los firmantes de OSHW, que busca el beneficio tanto de clientes como de colaboradores y desarrolladores.

Tabla 1.6. Aspectos de *Open Source Hardware.*

Documentación	Debe estar en ficheros de diseño en un formato que permita modificar y redistribuir los mismos. Si la documentación no está incluida, debe proporcionarse un medio para recopilar la información a un costo de reproducción razonable, preferiblemente una descarga libre de Internet.
Alcance	La documentación del hardware debe indicar claramente qué parte (si no toda) del diseño se publica bajo la licencia.
Programas Informáticos necesarios	El diseño puede requerir de un paquete informático como parte del mismo o para operar de forma correcta.
Obras derivadas	Se debe consentir modificaciones y obras derivadas, distribuidas bajo los mismos términos que la licencia de la obra original. Además, debe permitir la producción, distribución y uso de productos creados.
Libre distribución	No se limitará la venta o distribución de la documentación del proyecto. Además, no requiere el pago de regalías por estas ventas.
Atribución	La licencia puede requerir que documentos derivados y notificaciones de copyright asociadas con los dispositivos que atribuyan la autoría a la hora de distribuir ficheros de diseño.
No discriminación a personas o grupos	La licencia no puede discriminar ninguna persona o grupo de personas.
No discriminación a campos de aplicación	La licencia no puede limitar el uso del trabajo (incluyendo el objeto manufacturado) en áreas de aplicación específicas.
Distribución de licencia	Los derechos otorgados por la licencia se aplican a todos aquellos a los que sea redistribuido el trabajo sin la necesidad de ejecutar una licencia adicional.
La licencia no será específica a un producto	Los derechos otorgados por la licencia son independientes de si el trabajo licenciado es parte de un producto en particular. Todos deben tener los mismos derechos que la obra original.
La licencia no deberá restringir otro hardware o software	La licencia no debe imponer restricciones sobre lo que se agrega a la obra con el trabajo licenciado.
La licencia será neutra en término tecnológicos	Los términos no deben estar condicionados a una tecnología en particular, componente, material o estilo de interfaz o uso de la misma.

La implementación de un sistema de IoT demanda a más de un robusto componente de hardware para interactuar y transmitir información, un componente de software apropiado para administrar la información generada y actuar sobre el hardware, siendo necesario definir qué es software. De acuerdo a la Real Academia Española, "*es un conjunto de programas, instrucciones y reglas informáticas que permiten ejecutar distintas tareas en un dispositivo electrónico"* [12].

1.4.5 ONTOLOGÍA SSN (*SEMANTIC SENSOR NETWORK*)

OWL *(Web Ontology Language)* es uno de los instrumentos utilizados por la Web Semántica[1], desarrollado para disponer de un término que precise ontologías Web

[1] Web Semántica gestiona una estructuración común por medio de herramientas como por ejemplo OWL, RDF y SPARQL para permitir que los usuarios de Internet procesen, compartan y envíen información de forma inmediata y fácil.

mediante la especificación descriptiva de propiedades y clases. En la **Tabla 1.7.** se detallan brevemente estos factores [13].

Tabla 1.7. Propiedades y Clases de Ontologías Web.

Clases o conceptos	Ideas elementales que necesitan precisarse, como: métodos, tipos de objetos, procesos de inferencia, etc.
Instancias o individuales	Utilizadas para representar un objeto particular de una clase o concepto.
Relaciones o propiedades	Crea interacciones y asociaciones entre conceptos e instancias del dominio. Como algunas de las subclases que están relacionadas.

1.4.5.1 Origen de la Ontología SSN

La alta demanda de sistemas de sensores virtuales y físicos, capaces de recolectar, examinar y procesar gran cantidad de información en tiempo real y no real de varios tipos de fuentes, incluidos datos estructurados y no estructurados, han propiciado el diseño de la ontología SSN-XG (*Semantic Sensor Network Ontology*), la cual fue impulsada por el OGC (*Open Geospatial Consortium*), la Universidad Wright State y CSIRO (*Commonwealth Scientific and Industrial Research Organization)* Agencia Nacional Australiana para la ciencia como foro para la instauración de la ontología OWL.

La ontología SSN está diseñada bajo una arquitectura modular (horizontal y vertical) para extender naturalmente los conceptos principales a partir de un vocablo mínimo común, denominado SOSA. En la **Figura 1.5** se esquematizan los diferentes módulos clasificados en normativos y no normativos de este diseño.

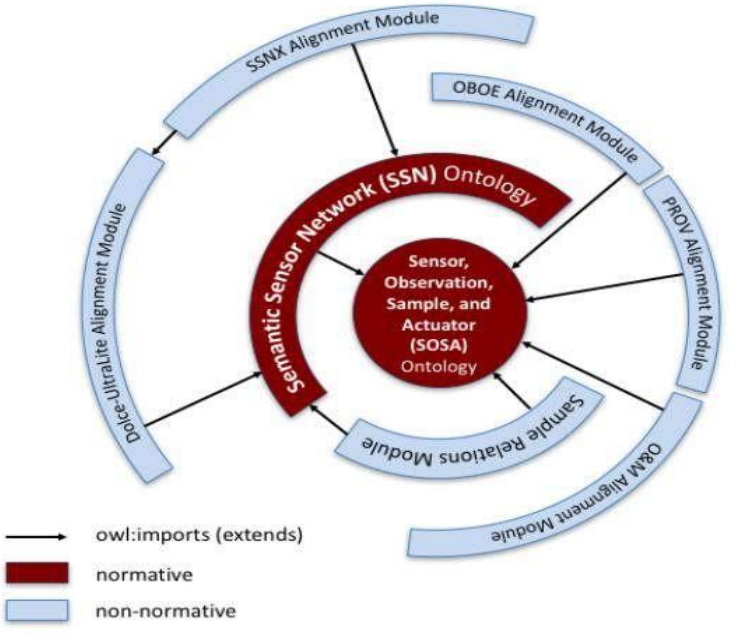

Figura 1.5. Diseño modular de la Ontología SSN [15].

A continuación, se detalla brevemente los conceptos básicos de SOSA [14]:

- ***Sensors* (Sensores):** Dispositivos que ponderan una o más magnitudes físicas.
- ***Observation* (Observaciones):** Valor real o aproximado de una cantidad física.
- ***Sample* (Muestras):** Ejemplos característicos de una magnitud en particular.
- ***Actuators* (Actuadores):** Regulan una magnitud física a partir de directrices.

1.4.5.2 Desarrollo de la Ontología SSN

La ontología SSN, se compone de 10 módulos que están estructurados conceptualmente pero no organizados físicamente. Describe 41 conceptos y 39 características del objeto, que de manera directa heredan de 11 conceptos DUL (*Descriptions and Situations*)-*UltraLite*) y 14 características del objeto de DUL. Especifica la exactitud, los sensores y las capacidades de los mismos, al igual que investigaciones y procedimientos usados para la detección. También integra conceptos de alcance y viabilidad [16].

La ontología DUL es una versión abreviada de la composición de dos ontologías de alto nivel o esenciales: DOLCE[2] (*Descriptive Ontology for Linguistic and Cognitive Engineering*) dando como resultado una ontología disponible en varios módulos, ligera y aplicable tanto para contextos sociales como para contextos físicos. Además, facilita la interoperabilidad de diferentes ontologías de nivel intermedio y bajo. La ontología se puede ver desde cuatro puntos de vista primordiales expuestos en la **Tabla 1.8.** [16].

Tabla 1.8. Puntos primordiales de la Ontología SSN.

Perspectiva del Sensor	Se centra en qué sentidos, cómo percibir, y que detecta.
Perspectiva de Observación	Se centra en los metadatos asociados con los datos de observación.
Perspectiva del Sistema	Se enfoca en los sensores y sistemas de monitorización.
Atributos y Perspectiva de la Propiedad	Se enfoca en lo que es observable sobre los elementos o atributos que detectan un atributo específico.

Hoy por hoy se confirma que la ontología *SSN* ofrece varias ventajas al mundo de la Web y son el resultado del trabajo y la investigación realizada por las organizaciones involucradas, incluyendo distintos tipos de herramientas, conceptos y métodos de otras tecnologías en *SSN* y gracias a su conocimiento de arquitecturas modulares, es posible implementar sistemas escalables, robustos y fáciles de configurar.

[2] DOLCE es el primer módulo de la biblioteca de ontologías fundacionales *WonderWeb*, apunta a capturar las categorías ontológicas subyacentes al lenguaje natural y al sentido común humano.

1.4.6 ÍNDICE DE MEDICIÓN SERVICIO (SMI)

Para realizar una evaluación de los servicios en la nube y seleccionar el más apropiado, y a su vez resolver varios problemas a los que se enfrentan los usuarios y proveedores en el entorno de nube, se propone un método de innovación mediante parámetros *Service Measurementindex* (SMI) diseñados e implementados por *Cloud Service Measurement Index Consortium* (CSMIC).

Scott Feuless miembro del CSMIC [2], fue el autor principal de las métricas, y el pionero en la idea de utilizar un marco coherente y estándar para guiar la evaluación de los servicios en la nube. El SMI es un marco de aplicación que define una técnica para el cálculo de un índice relativo, que permite comparar servicios de TI entre sí o para rastrear servicios a lo largo del tiempo. Los atributos SMI están diseñados sobre la base de las normas de la Organización Internacional de Normalización del Consorcio (ISO). Consiste en un conjunto de indicadores clave de rendimiento *Key Performance Indicator* (KPI) relevantes para el negocio que proveen un método estandarizado para medir y cotejar un servicio empresarial.

1.4.6.1 Principios de Operación

Una comparación relativa significativa de los Servicios de TI puede ser posible, pero solamente, de aquellos servicios que son funcionalmente similares. Esto implica que se debe utilizar una taxonomía estándar de los Servicios de TI para evitar que cada usuario por separado del Índice de Gestión de Servicios defina las funciones que está buscando.

El SMI se visualiza actualmente como un marco jerárquico. El nivel superior divide el espacio de medición en 7 categorías. Cada categoría se clarifica aún más con 3 o más atributos de son una combinación de medidas cualitativas y cuantitativas. Luego, dentro de cada atributo, se definirá un KPI que describirá los datos que se recopilarán para cada medida o métrica. Proporciona un enfoque holístico de QoS (*Quality of Service*) que necesitan los clientes para seleccionar un proveedor de servicios en la nube basado en: Responsabilidad, Agilidad, Garantía de servicio, Costo, Desempeño, Seguridad y Privacidad, y Usabilidad, tal como se muestra en la **Figura 1.6.** Algunos de estos KPI serán específicos del servicio, mientras que otros se aplicarán a todos los servicios.

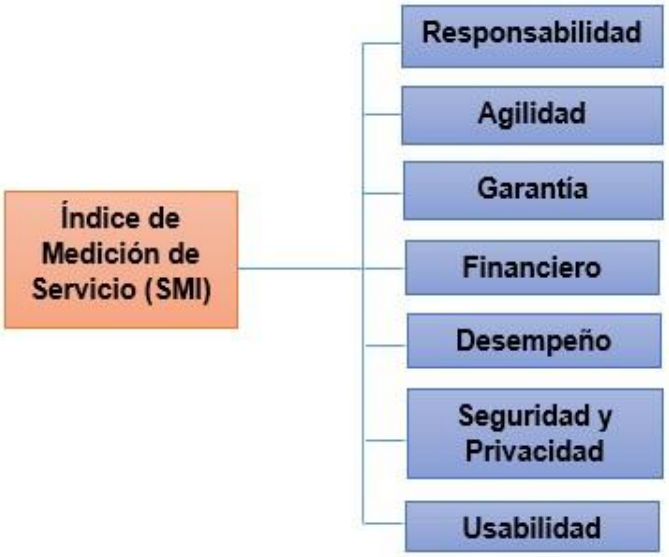

Figura 1.6. Categorías del Índice de Medición del Servicio – SMI [2].

1.4.6.2 Métricas Cualitativas

Como ya se había mencionado anteriormente, cada categoría tiene diversos atributos, tanto de naturaleza cuantitativa como cualitativa. Sin embargo, este estudio se centrará en los atributos cualitativos como la facilidad de uso, la flexibilidad, la instalación, la seguridad, entre otros; los mismos que se evaluarán mediante una escala ordinal fundamentada en un conjunto de calificaciones predefinidas como: excelente, alta, media y buena.

Las métricas cualitativas son difíciles de traducir en números, ya que depende la manera en la que se esté analizando. Existe un abanico de opciones de plataformas de *Middleware* IoT disponibles; por lo tanto, las métricas cualitativas permitirán filtrar el *Middleware*. En la **Tabla 1.9.** se presenta la clasificación de estas métricas, las cuales serán detalladas posteriormente.

Tabla 1.9. Clasificación de las métricas cualitativas según SMI.

Métricas cualitativas	Categoría SIM	Definición
Instalación	*Usability* (Usabilidad)	Complejidad en la instalación del Middleware.
Aprendizaje	*Usability* (Usabilidad)	Facilidad de aprendizaje.
Usabilidad	*Usability* (Usabilidad)	Facilidad de uso.
Seguridad y Privacidad	*Security and Privacy* (Seguridad y Privacidad)	Medida en que un servicio puede satisfacer los requisitos de seguridad del usuario en términos de control de acceso, privacidad, datos, infraestructura, etc.
Flexibilidad	*Agility* (Agilidad)	Calificación de la capacidad de agregar o eliminar funciones predefinidas de un servicio para adaptarse a las preferencias de los usuarios.

a. Instalación

La Instalación se define como el proceso a seguir para implementar la plataforma. Su documentación debe ser de fácil acceso en algunos repositorios de software públicos (*github*), que debe contener características técnicas, versión, sistema operativo recomendado, procesador, memoria mínima requerida, espacio necesario en el disco duro y si hay un contenedor como *Docker* (proyecto de código abierto que optimiza el despliegue y virtualización de aplicaciones dentro de contenedores de software) o una versión de máquina virtual que pueda ejecutarse localmente. Los indicadores se dividirán en Fácil, Medio y Complejo, los usuarios son responsables de la evaluación.

b. Aprendizaje

CSMIC SMI define el Aprendizaje como "*el esfuerzo requerido por los usuarios para aprender a usar el servicio"* [2]. Para que los servicios basados en la nube sean tan atractivos para los clientes como los servicios que no lo son, los esfuerzos para aprender a usar el servicio deben poseer un umbral más bajo y tener un tiempo y experiencia mínimos relacionados con la disposición para usar el servicio.

c. Usabilidad

De acuerdo a la norma ISO 25000 [17] "*Usabilidad es la capacidad del producto software para ser entendido, aprendido, usado y resultar atractivo para el usuario, cuando se usa bajo determinadas condiciones"*. Esta característica se subdivide a su vez en capacidades del producto y son expuestas en la **Tabla 1.10**.

La Usabilidad juega un papel importante en el uso rápido de los servicios en la nube. Cuanto más fácil sea usar y aprender acerca de un servicio en la nube, más rápido una organización puede pasar a servicios en la nube.

Tabla 1.10. Capacidades de un producto.

Capacidad	Descripción
Para reconocer su adecuación	Permite que el usuario determine si el software es adecuado para sus necesidades.
De aprendizaje	Ayuda al usuario a comprender su aplicación.
Para ser usado	Ayuda al usuario a manejarlo y controlarlo con destreza.
De protección contra errores de usuario	Capacidad del sistema que protege a los usuarios de hacer errores.
De estética de la interfaz de usuario	Capacidad de la interfaz de usuario de agradar y satisfacer la interacción con el usuario.
Accesibilidad	Admite el uso de un producto por usuarios con características y discapacidades particulares.

d. Seguridad y Privacidad

La preocupación de casi todas las organizaciones es la protección de datos y la privacidad. Controlar el alojamiento de datos en otras organizaciones es un problema clave y requiere que los proveedores de la nube adopten políticas de seguridad estrictas. La seguridad y la privacidad también son de naturaleza multidimensional y contienen muchos atributos, como la privacidad, la pérdida de datos y la integridad.

Las plataformas *Middleware* IoT, cuentan con características similares, y la comparación cualitativa tradicional no facilita la elección de un determinado *Middleware*. Las métricas cualitativas basadas en SMI estudiadas en este documento tienen como objetivo mitigar la subjetividad, tanto como ayudar a los usuarios a vislumbrar la opción más acorde al momento de optar por un *Middleware* de IoT para una solución en particular.

2 METODOLOGÍA

La continua evolución de las TICs requiere la conectividad entre objetos comunes y plantea un gran desafío tecnológico debido a los inconvenientes a la hora de realizar esta tarea, y es así como surgen las plataformas IoT. Por lo tanto, en este Capítulo se presentará el estudio a nivel explicativo trabajando en detalle con las plataformas *Middelware: OpenIoT y DeviceHive* para probar la efectividad de las técnicas y herramientas que pueden usarse para resolver problemas o evaluar proyectos futuros que permitirán realizar el análisis cualitativo basado en SMI.

2.1 *MIDDLEWARE OPEN SOURCE*

Tiempo atrás la industria de las plataformas IoT era prácticamente inexistente y, si bien es cierto el panorama ha cambiado, este sigue siendo complejo. Hoy por hoy se cuenta con proveedores de plataformas IoT para proyectos y tecnologías de toda índole: gratis o de pago. El propósito primordial del *Middleware* es ser tolerante a fallas y altamente disponible. A continuación, se exponen las características más representativas del *Middleware Open Source*.

- Garantiza una integración perfecta de aplicaciones y sistemas, desarrollados e implementados por diferentes organizaciones y proveedores sin restricción alguna.
- La migración a cualquier nueva plataforma o sistema es perfecta. Admite los principales protocolos de comunicación: *MQTT, CoAP, HTTP*, *WebSockets*[3], etc.
- Admite APIs[4] abiertas, modelos de implementación de la nube altamente disponibles y modelo de datos de información distribuido y extensible que proporcionan una alta escalabilidad.

A continuación, se expone de manera detallada las plataformas *OpenIoT* y *DeviceHive*, las cuales han sido escogidas para el análisis cualitativo mediante el SMI.

2.1.1 OPENIOT

OpenIoT es una plataforma *Open Source* que se lanzó bajo la licencia LGPL V3.0. Acepta comunicaciones REST y GSN (*Global Sensor Network*) con su servidor. Conecta todos los sensores (para *OpenIoT* un sensor es todo componente que provee observaciones) por

[3] WebSocket, tecnología que permite conexiones de manera persistente entre el navegador del usuario y el servidor.
[4] API, conjunto de definiciones y protocolos que permiten la comunicación entre dos aplicaciones de software a través de un conjunto de reglas.

medio de tecnología en la nube basada en el modelo de *Cloud Computing*. La infraestructura versátil de *OpenIoT* crea un puente entre la semántica y la informática de datos mediante SSN [18] y junto con el manejo de sensores móviles y parámetros asociados de QoS constituyen su principal factor diferenciador; dado que provee la base para el desarrollo y la detección de nuevas aplicaciones IoT a gran escala.

A pesar de que *OpenIoT* es un proyecto fascinante y de haber recibido el premio *Black Duck* por ser uno de los diez principales proyectos de código abierto [18], no ha recibido actualizaciones de su repositorio de *github* desde agosto de 2015. A continuación, se detallan las herramientas tecnológicas que permiten el funcionamiento de la plataforma *OpenIoT*.

2.1.1.1 Maven

Maven hoy por hoy es un proyecto de nivel superior de la *Apache Software Foundation*, ampliamente utilizado por *OpenIoT* para el desarrollo y gestión de sus proyectos Java. Facilita una solución interoperable a nivel de sistema operativo y configuraciones descriptivas (gracias a HTML). Maven se basa en el concepto de POM *(Proyect Object Model)* que genera documentación de empaquetado, aprueba los test, y prepara las *builds* (proceso de convertir archivos de código fuente en artefactos de software independientes y/o versión de un programa).

Es primordial conocer que un artefacto es un componente de software que puede ser incluido como dependencia en un proyecto**.** Generalmente se trata de un archivo JAR**.** o WAR. Los artefactos pueden tener dependencias entre sí, por lo tanto, al incluir un artefacto en un proyecto, también se obtendrá sus dependencias.

Por otro lado, se declara un identificador único del proyecto/artefacto, a partir de la unión de tres identificadores y estos se detallan en la **Tabla 2.1.**

Tabla 2.1. Componentes para identificar un artefacto.

groupId	Simboliza la organización autora o propietaria del artefacto. Por ejemplo: *com.ibm, org.jboss, org.apache, etc.*, por lo que este id por sí solo no puede identificar un artefacto.
artifactId	Es el nombre del proyecto/artefacto actual. Por ejemplo: *http-client, hibernate, tomcat, commons-collections*, etc.
versión	Identifica el número de versión del artefacto.

a. Modelo de Objeto del Proyecto (*POM – PROJECT OBJECT MODEL*) [19]

Cada proyecto Maven consta de un archivo denominado *pom.xml*, que como ya se mencionó anteriormente permite encontrar información sobre todos los

atributos del proyecto. En la **Tabla 2.2** se detallan las características que admite el POM.

Tabla 2.2. Características de la existencia de POM.

Reutilización universal de la lógica de construcción	Los *plugins* funcionan dentro del POM, son independientes de la ubicación del proyecto y son reutilizables.
Portabilidad e integración de herramientas	Los IDEs tienen un lugar común para encontrar toda la información acerca del proyecto.
Búsqueda y filtrado sencillo de artefactos del proyecto	Permite buscar e indexar elementos en un repositorio utilizando la información almacenada en el POM.

b. Manejo de Dependencias

El motivo primordial para declarar una dependencia es que el archivo ***pom.xml*** provee los requerimientos previos indispensables para compilar el proyecto. Maven funciona para dependencias transitivas. Esto significa que, si el proyecto "A" depende del proyecto "B", Maven descargará de forma automática el JAR junto con todas las dependencias de ese proyecto, lo cual simplifica este proceso.

c. Manejo del Ciclo de Vida del Artefacto de *Build* (*Goals*)

Maven se basa en una arquitectura de *plugins* extensible, formada por pequeños módulos del mismo nombre que la arquitectura que provee funcionalidad en la forma de invocación y se denominan *goals*. Maven se define como una serie de diferentes fases de tres ciclos de *build* de software [20]. El ciclo establecido posee las etapas detalladas en la **Tabla 2.3.**

Tabla 2.3. Etapas del ciclo de Maven.

Validación *(validate)*	Valida que el proyecto sea correcto.
Compilación *(compile)*	Compila el código.
Test *(test)*	Mediante las pruebas unitarias se prueba el código fuente.
Empaquetar *(package)*	Empaquetar el código compilado y lo transforma en algún formato tipo .jar o .war.
Pruebas de integración *(integration-test)*	Procesa y despliega el código en algún entorno que permita ejecutar las pruebas de integración.
Verificar *(verify)*	El código empaquetado es válido y cumple los criterios de calidad.
Instalar *(install)*	El código empaquetado en el repositorio local de Maven puede ser usado como dependencia de otros proyectos.
Desplegar *(deploy)*	Despliega un paquete en el repositorio central disponible para el equipo de desarrollo.

Como se ha mostrado, Maven es un sistema “todo en uno” que proporciona agilidad de proyectos de software (*builds*) y una interfaz común para el desarrollo independiente, con dependencias mejoradas entre módulos y otras versiones de librerías. Por todo esto, Maven es considerado un marco estructural modular simple, ágil y extensible que simplifica el desarrollo de proyectos de software y se utiliza para nuevas tecnologías, como en el caso de la plataforma *OpenIoT*.

2.1.1.2 *Virtuoso Universal Server*

Virtuoso es un software diseñado por *OpenLink* en colaboración con Kingsley UYI Idehen y Orri Erling [21] arquitecto gestor de software, que lo determinan como una mezcla innovadora de *Middleware* y motor de base de datos. Su funcionalidad adicional es la implementación de *SPARQL [21]*. La edición de código abierto se denominada como *OpenLink Virtuoso y* no incluye un motor de base de datos virtual ni replicación de datos.

a. Arquitectura de Virtuoso [21]

El diseño de Virtuoso tiene un “Servidor Universal” que facilita el aprovechamiento del soporte de múltiples procesadores e hilos del sistema operativo. Múltiples hilos pueden operar en un solo índice en forma de árbol con poca interferencia con otros hilos. Existe un caché que comparte páginas de la base de datos entre diferentes hilos. El almacenamiento en caché se graba en el disco a través de un proceso en segundo plano. El diagrama de la arquitectura de un servidor híbrido único de Virtuoso se muestra en la **Figura 2.1**, la cual provee la funcionalidad de un servidor tradicional separado en un solo producto [21]. Virtuoso admite múltiples bases de datos que incorporan *Progress, Oracle* y demás motores de bases de datos compatibles con *ODBC* (*Open DataBase Connectivity*) y *JDBC* (*Java™ Database Connectivity*).

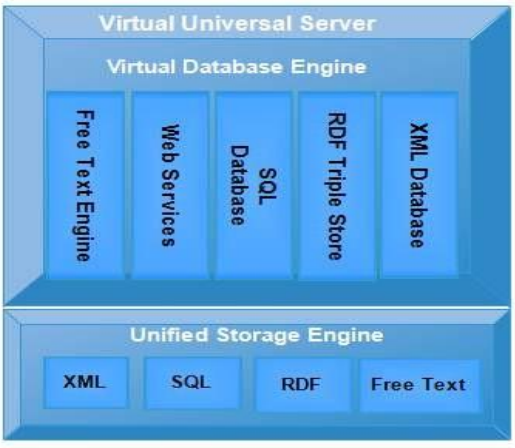

Figura 2.1. Arquitectura de Virtuoso.

Virtuoso puede almacenar, indexar y buscar datos XML. Dado que combina *XPath* (*XML Path Language*), *XSLT* (*Extensible Transform Sheet Language*), *XQuery* y consultas de bases de datos SQL, no hay comparación entre todas las bases de datos relacionales habilitadas para XML [21]. En **Tabla 2.4**, se detallan las herramientas mencionadas.

Tabla 2.4. Herramientas de Virtuoso.

XPath	Es una forma estándar de extraer datos de un árbol XML. De forma predeterminada, todos los estándares XML tienen un *XPATH* integrado. *XPATH* de Virtuoso se proporciona en forma de funciones SQL y dos predicados SQL especiales.
XSLT	Es el procesador integrado en Virtuoso. Recomendado para convertir datos en datos XML u otros datos no XML.
XQUERY	Intérprete incorporado de Virtuoso. Realiza las mismas tareas que XSLT conjuntamente consulta mecanismos de mapeo SQL y XML. Se invoca a *XQuery* desde SQL para resolver datos XML en un mejor lenguaje. No obstante, no se utiliza para filtrar eficazmente los datos XML almacenados.

b. Almacenamiento Físico

Virtuoso brinda un modelo objeto-relacional extensible que unifica la flexibilidad del acceso relacional a través de la herencia, la recopilación de datos en tiempo de ejecución y el acceso basado en identidad. Para organizar la información, Virtuoso utiliza el almacenamiento universal. Las tablas se almacenan en un conjunto específico de archivos. El tamaño máximo de cada conjunto de archivos es de 32 TB para páginas de 4 GB x 8 KB.

En resumen, *Virtuoso Universal Server* puede trabajar con datos vinculados y funciones de SSN ofreciendo un abanico de proyectos de diferentes campos que se pueden compartir en la Web a través de la base de datos en RDF sin restricciones a un formato determinado. Se puede convertir un formato en particular a un formato de interés, además se puede procesar consultas en otros lenguajes relacionales y todas las funciones que Virtuoso maneja en un "Servidor Universal", razones por las cuales Virtuoso es una herramienta convincente para generar plataformas IoT.

2.1.1.3 Servidor de Aplicaciones *JBoss Server 7.1.1 Final*

Un Servidor de Aplicaciones es una implementación de la especificación J2EE (*Java™ 2 Platform, Enterprise Edition*). Las aplicaciones desarrolladas bajo este estándar se pueden efectuar en cualquier Servidor Web o de Aplicaciones que cumpla con los estándares. Un Servidor de Aplicaciones está en el corazón de los sistemas distribuidos a gran escala, permite la creación y la implementación de software apoyado en componentes. Un sistema distribuido permite optimizar tres aspectos esenciales en una aplicación y son expuestos en la **Tabla 2.5** [22].

Tabla 2.5. Aspectos fundamentales de *JBoss Server.*

Alta Disponibilidad	Para que el sistema funcione las 24 horas del día, los 365 días del año requiere el uso de técnicas de equilibrio de carga y recuperación ante fallos (*failover*).
Escalabilidad	A medida que aumenta la carga de trabajo (número de solicitudes), aumentan las capacidades del sistema. Debido a la capacidad limitada de recursos de cada sistema, solo puede manejar un número limitado de solicitudes.
Mantenimiento	Flexibilidad para actualizaciones, depuración y mantenimiento del sistema. La solución es construir su lógica de negocio con unidades modulares reutilizables.

Un Servidor de Aplicaciones provee una estructura en tres capas, tal como se puede mirar en la **Figura 2.2**. Este tipo de servidores permiten estructurar el sistema de forma más eficiente; gracias a la capa intermedia, los procesos pueden ser gestionados de forma separada a la Capa de Datos y la Capa Cliente, simplificando la gestión e integrando múltiples fuentes. El *kernel* en tiempo de ejecución también admite la implementación y administración de aplicaciones comerciales de alto rendimiento. La arquitectura de *JBoss* está orientada a servicios y, gracias a las licencias de código abierto, se puede descargar, usar y distribuir sin restricciones, lo que la convierte en la plataforma de *Middleware* más utilizada. Las ventajas de *JBoss* son [22]:

- Flexibilidad consistente, incrustable y orientado a arquitectura de servicios.
- Servicios del *Middleware* para cualquier objeto de Java.
- Soporte completo para JMX[5] (*Java Management Extensions).*

La plataforma *OpenIoT* utiliza el Servidor de Aplicaciones *JBoss AS 7.1.1 Final* que es una actualización de *JBoss AS 7.1.0 Final* que se apoya en la arquitectura muy ligera de *JBoss* AS 7. *JBoss AS7.1* promueve la seguridad y ofrece un alto grado de capacidad de gestión y de agrupación en clústeres.

Como se mencionó anteriormente, *JBoss* es uno de los Servidores de Aplicaciones más populares en el mundo de IoT en términos de seguridad, rendimiento, escalabilidad, capacidad de mantenimiento y compatibilidad global [22]. Proporciona funcionalidad y reduce la complejidad del desarrollo de aplicaciones. De hecho, *JBoss* es de fácil acceso gracias a una licencia de código abierto, pero puede ser un poco difícil de implementar dependiendo de su sistema operativo y la versión de su servidor; sin embargo, esto no influyó en la concepción de que *JBoss Server* sea considerado una herramienta primordial para la creación de nuevas aplicaciones a partir del desarrollo del *Web World*.

[5] JMX, tecnología que define una arquitectura de gestión, API, patrones de diseño, y los servicios de monitoreo y administración de aplicaciones basadas en Java.)

Figura 2.2. Arquitectura de un Servidor de Aplicaciones.

2.1.1.4 Arquitectura de *OpenIoT*

La arquitectura *OpenIoT* es una particularización de *European Research Cluster on the Internet of Things* (IERC). Está compuesta por siete módulos pertenecientes a diferentes planos lógicos, expuestos en la **Figura 2.3**.

Estos planos son: Utilidad/Aplicación (*Utility/Application),* Virtualizado *(Virtualized) y* Físico *(Physical)* [23]:

a. Plano de Utilidad/Aplicación (*Utility/Application Plane*)

Este plano está compuesto por tres componentes principales que trabajan con la interfaz Web 2.0, y con los cuales es posible acceder a la solicitudes, configuración, monitoreo e interfaz gráfica de *OpenIoT.* A continuación, se exponen estos componentes a detalle:

- **Definición de Solicitudes (*Request Definition):*** Enruta las solicitudes de servicio. Incluye un conjunto de servicios para detallar y desarrollar las peticiones anteriores, y al mismo tiempo enviarlas al *Global Scheduler.*
- **Presentación de Solicitudes (*Request Presentation*):** Selecciona *mashups* de una librería para descubrir servicios y notifica directamente al SD&UM. Además, es responsable de recuperar los datos relevantes de los sensores seleccionados.

- **Configuración y Monitoreo (*Configuration and Monitoring*):** Administra y configura las capacidades de los sensores y los servicios implementados dentro de *OpenIoT*. Los usuarios pueden monitorear los diversos módulos implementados.

b. Plano Virtualizado (*Virtualized Plane*)

Este plano permite procesar las solicitudes realizadas en el Plano de Aplicación, así como almacenar, realizar consultas y monitorear procesos, gracias a los componentes detallados a continuación.

Tabla 2.6. Componentes del Plano Virtualizado [23].

Programador (Scheduler)	Resuelve todas las solicitudes de *Definition Request* y asegura el acceso oportuno a los recursos requeridos. Permite descubrir los sensores y flujos de datos relacionados para configurar el servicio, gestionar, seleccionar y activar recursos.
Almacenamiento de datos en la nube (Linked Stream Middleware Light, LSM-Light)	LSM fue rediseñado para las capacidades de datos *push-pull*; una interfaz en la nube actúa como una DB almacenando los metadatos necesarios para el desempeño de *OpenIoT* (datos funcionales).
Administrador y Prestación de Servicios (Service Delivery Manager & Utility SD & UM)	Por un lado, combina los flujos de datos representados por el servicio mientras trabaja en el sistema para prestar el servicio de envío de solicitudes. Por otro lado, facilita la medición, conserva el seguimiento de las métricas de cada usuario para cada servicio y facilita funciones como la contabilidad, facturación y la optimización de recursos para que sean compatibles con los paquetes de servicios públicos.

c. Plano Físico (*Physical Plane*)

En este plano se lleva a cabo el proceso de los sensores físicos o virtuales a través del *Sensor Middleware* (*Extended global Sensor Network, X-GSN*) que de forma semántica; filtra, recopila, integra y registra secuencias de datos desde dispositivos físicos o sensores virtuales. Trabaja como un eje entre el mundo físico y *OpenIoT*. Puede ser implementado sobre la base de uno o más nodos, pertenecientes a diversas entidades administradoras.

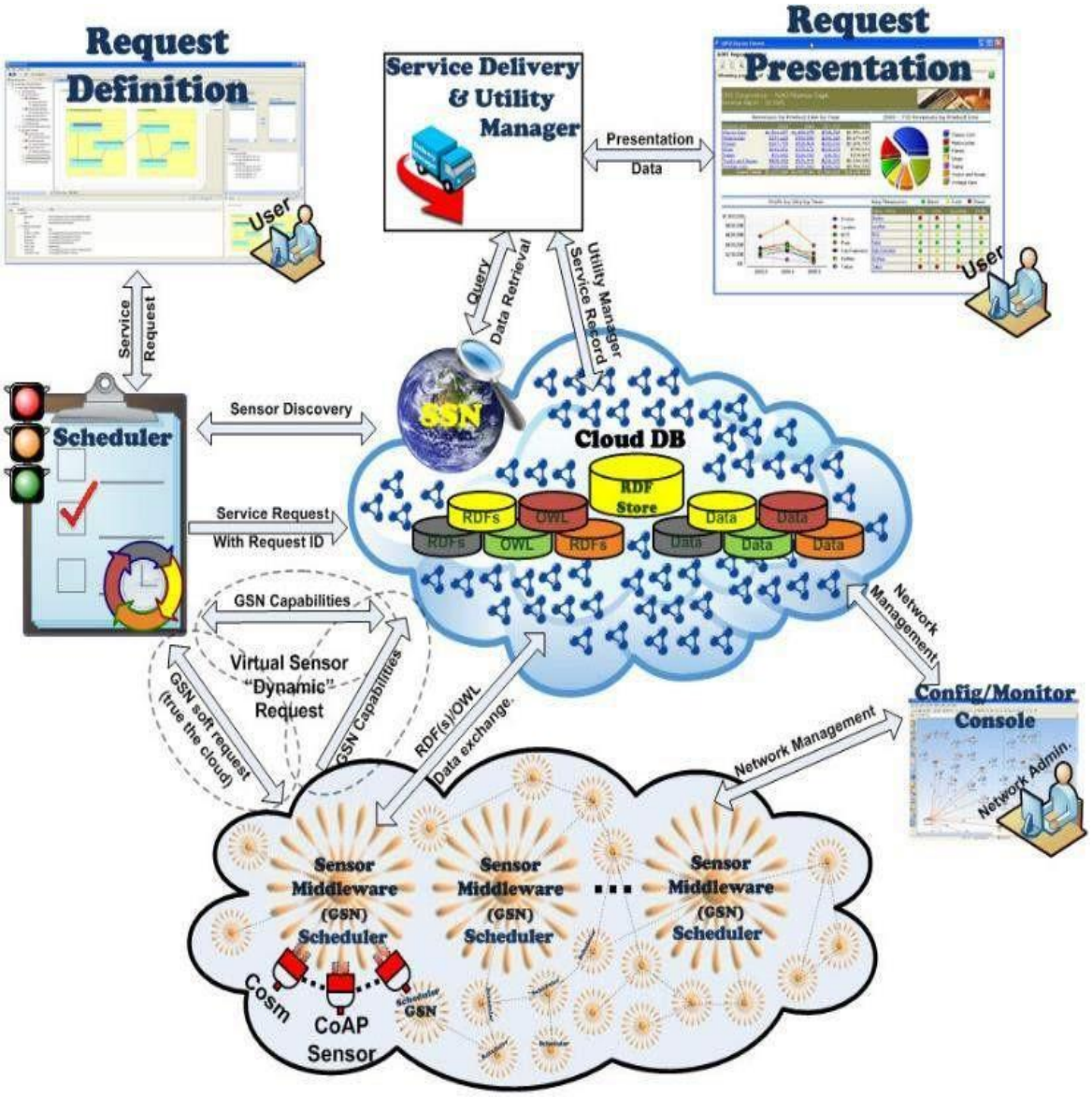

Figura 2.3. Arquitectura *OpenIoT* [23].

Una vez descritos los tres planos y las distintas tareas que realizan y hacen posible el funcionamiento de *OpenIoT*, es necesario detallar los componentes esenciales de esta plataforma.

2.1.1.5 Programador Global (*Global Scheduler*)

El *Global Scheduler* es un componente de software que controla cómo los servicios de IoT acceden a los diferentes recursos administrados por la plataforma. Cada solicitud de servicio se puede analizar y se conecta con el resto de la plataforma por medio de DB (Base de Datos de la Nube). Seguidamente, se muestra una síntesis de las funciones y servicios admitidos por el *Scheduler* [23]:

- **Descubrimiento de Recursos (*Resource Discovery*):** Manifiesta la disponibilidad del sensor virtual basado en la entidad "*availableSensors*".
- **Estado de Servicio Registrado (*Registered Service Status*):** El usuario puede verificar el estado de un servicio en particular mediante el *ServiceID*, el cual es verificado por la entidad "*serviceStatus"* y envía toda la información al usuario.
- **Obtener Servicio (*Get Service*):** Provee la reseña de un servicio registrado por medio de la entidad "*serviceDescription*".
- **Obtener los Servicios Disponibles (*Get the Available Services*):** Los usuarios pueden recopilar una lista de servicios registrados asociados con un usuario determinado en la entidad "*serviceDescription*" especificando el ID de usuario.
- **Gestión del Servicio de Usuario (*Service User Management*):** Permite gestionar el ciclo de vida de los servicios.
- **Obtener Usuario (*Get User*):** Proporciona información del usuario y las condiciones para la puesta en marcha del filtrado de acceso y control de datos.
- **Recursos de Actualización del Servicio (Service Update Resources):** Verifica el servicio activado en la entidad "*serviceDescription*". Comprueba que el sensor portátil cumpla con los requisitos. De lo contrario, el sensor se eliminará de la entidad "*sensorServiceRelation*" y se buscará un nuevo sensor, si se encuentra, se registrará con la identidad en cuestión. De lo contrario, este campo se actualizará con "true" en la entidad "*serviceStatus*".

2.1.1.6 Plataforma De Datos LSM [23]

La arquitectura *OpenIoT* permite que la pieza clave llamada *Link Sensor Middleware (LSM)* procese y entregue los datos del sensor. En el contexto del flujo de datos del sensor en *OpenIoT, LSM* trata el sensor virtual como una fuente de datos de entrada. La representación de los datos está vinculada a los datos de los sensores virtuales. Hay dos formas esenciales de importar y enviar datos a *LSM* en función de *"pull"* y *"push".* Ambos procedimientos se usan en *OpenIoT* según el caso de uso real. Con el mecanismo *"pull" LSM* indaga periódicamente datos similares a un flujo de datos, en tanto que en el mecanismo "*push*", admite a la fuente de datos enviar dichos datos activamente al LSM.

2.1.1.7 *X-GSN [23]*

XGSN es un sistema *Open Source* fundamentado en metadatos relacionados con sensores virtuales, representación semántica de observaciones de sensores y extensiones de *SSN*. Diseñado como una plataforma altamente escalable que puede manejar la recuperación de datos en múltiples dispositivos y protocolos aprovechando las capacidades existentes del *Middleware* global de la red de sensores *GSN*. *XGSN* también gestiona el proceso de anotación para recibir observaciones del sensor y generar secuencias *RDF* que se envían al *Middleware* del sensor habilitado para la nube correspondiente. Existen dos tipos esenciales de anotaciones semánticas agregadas a *XGSN* expuestas en la **Tabla 2.7.**

Tabla 2.7. Anotaciones semánticas de XGSN.

Anotaciones	Descripción
De metadatos, relacionados con sensores, dispositivos de detección y sus capacidades	Estas son habitualmente enlazadas a los sensores virtuales declarados en una instancia XGSN. Concretamente, especifica un dispositivo que genera datos para un sensor virtual específico: tipo de observación que produce, ubicación, tipo de fuente, quien es el responsable, la organización, etc.
Relacionadas con las observaciones o mediciones que producen continuamente los sensores	Contiene información semántica para interpretar el tiempo y el contexto de las observaciones, valores únicos y más. Estas anotaciones semánticas particulares de la representación del sensor virtual proveen las tareas de descubrimiento e investigación en entornos IoT.

Concluyendo, los beneficios que proporciona el enfoque de *XGSN* hacen de este una herramienta de gran utilidad digna de ser incorporada a la plataforma *OpenIoT* de forma amplia, facilitando un sistema sumamente flexible y extensible que permite administrar el ciclo de vida de los datos del sensor desde la recolección de datos hasta la publicación en el contexto de la Web Semántica de las cosas.

2.1.1.8 *CUPUS* (*CloUd-based Publish/Subscribe*) [23]

CUPUS es un sistema de publicación/suscripción basado en contenido, es decir, admite suscripciones booleanas sin estado y sobre ventanas deslizantes que son *stateful* (cortafuegos de estado). La nube *CUPUS* ofrece notificaciones *push* en la nube a destinos ampliamente distribuidos, como dispositivos móviles, prácticamente en tiempo real.

CUPUS consta de un *broker* en la nube y entidades externas como *brokers* móviles, editores o suscriptores. El motor *CPSP* (*OpenIoT Cloud Broker*), admite todos los mensajes de las entidades de suscriptores, editores o *brokers* móviles, apoyándose en las entradas de usuario o *ICO (Sensor & ICO Internet-Connected Objects Selection)*. Luego de gestionar el mensaje, el *broker* de la nube, remite un aviso o suscripción a la entidad usuario final pudiendo ser un editor, suscriptor o un mediador móvil.

El componente autónomo y parte del *Middleware CUPUS* es denominado *QoS Manager.* Posee tres funcionalidades primordiales desplegadas en la **Tabla 2.8**.

Tabla 2.8. Funcionalidades Primordiales de *Qos Manager.*

Funcionalidades	Primordiales de *QoS Manager*
Supervisión y administración de suscripción de *QoS*	El administrador *QoS* agrega varios registros de datos de sensores (consultas) para definir los requisitos generales de la aplicación asociados con la recopilación de datos de sensores.
Supervisión y administración de publicaciones *QoS*	Los administradores de *QoS* revelan datos de los sensores monitoreados, administran la recopilación de datos de los sensores, optimizan el ancho de banda y el consumo de energía y responden a los requisitos de la aplicación.
Recolección de lecturas de todos los sensores	*QoS* recopila medidas en todos los sensores y utiliza *XGSN* para enviarlas a la base de datos *Cloud* de *OpenIoT*.

2.1.1.9 Aplicaciones

OpenIoT es adecuada para diversos campos interrelacionados de la ciencia y la tecnología, que incluyen [23]:

- Recopilar y procesar datos de casi cualquier sensor del mundo, incluidos algoritmos de procesamiento de sensores, algoritmos de procesamiento de redes sociales, dispositivos físicos, y más.
- Proveer una configuración de servicios IoT que incluya datos de varios sensores.
- Computación en la nube, incluidos los sistemas de privacidad y seguridad basados en servicios públicos. Así como la optimización de los recursos de infraestructura en la nube y *Middleware* de *OpenIoT.*
- Facilitar la integración del sensor con un esfuerzo mínimo para implementar el controlador de acceso apropiado.
- Visualizar datos de IoT apoyados en combinaciones apropiadas (tablas, gráficos, mapas, etc.).

Como se puede ver la plataforma *OpenIoT*, combina una serie de tecnologías que facilitan y optimizan el desarrollo de nuevos proyectos orientados al mundo IoT, respondiendo con eficiencia las continuas exigencias de las TICs. A continuación, se detallan los proyectos más relevantes en los que se ha aplicado la plataforma *OpenIoT* en la **Tabla 2.9.**

Tabla 2.9. Aplicaciones de *OpenIoT.*

Campus Smart [24]	El proyecto *Campus Smart* es esencialmente una aplicación del modelo de servicios de localización. Los estudiantes del campus de la Universidad KIT (Universidad Tecnológica de Karlsruhe) desempeñan el papel empresarial de usuarios finales, la universidad como proveedor de soluciones *Campus Guide* y como proveedor de instalaciones e infraestructura de equipos de comunicación en el campus universitario. Ete proyecto aclara el potencial de *OpenIoT* en la creación de aplicaciones integradas en la entrada del sensor "conectado a Internet" de las aplicaciones de redes sociales. Un escenario colaborativo llamado "*Smart Meetings*" representa la conexión entre lo virtual y lo físico en una sola aplicación, con interacción mejorada con elementos del lugar de trabajo.
OpenIoT con Qowisio [25]	Qowisio es una aplicación que facilita el ingreso rápido y fácil a la próxima generación de IoT. Permite crear una cuenta de usuario y escanear un código QR para asociarlo con un objeto vinculado. Durante cada escaneo, el módulo de aplicación completa la aplicación móvil y muestra los datos del objeto en tiempo real. Por ejemplo, controla la temperatura en la habitación de los niños. Simplemente se debe descargar e instalar la aplicación por medio de *google play store* o *apple store* y a continuación crear una cuenta e instalar el objeto conectado para luego escanear el código QR de este. Finalmente se desplegará una interfaz gráfica que permite acceder a los datos del objeto vinculado.
***CSIRO Phenonet* [26]**	El proyecto *CSIRO Phenonet* expone una red de sensores que recopila información sobre el campo de cultivos preliminares. La información recopilada de los sensores abarca la temperatura del suelo, la humedad, las previsiones meteorológicas, la temperatura / contaminación y demás parámetros ambientales. El potencial de recaudar esta información y proporcionar retroalimentación en tiempo real es un mecanismo valioso para científicos y agricultores.

2.1.2 *DEVICEHIVE*

DeviceHive es otra plataforma *Open Source* IoT publicada con la licencia Apache 2.0. Puede implementarse mediante *Docker* y *Kubernetes*. Permite conectarse a cualquier dispositivo por medio de *API REST*, *WebSockets* o *MQTT*.

DeviceHive ofrece la creación de prototipos hasta soluciones empresariales. Permite pensar en el desarrollo de negocios en lugar de en las formalidades técnicas. Cuenta con un kit de herramientas modular y fácil de usar, el cual permite interactuar con la nube *DeviceHive* y los principales protocolos utilizados en IoT. Es un servicio que se ejecuta en Ubuntu *Snappy Core*[6] para IoT. Esto significa que puede interactuar con este servicio utilizando un idioma de elección, siempre y cuando tenga bibliotecas de enlace para *DBus* (Bus de mensajes estándar de Linux).

A continuación, se listan las principales características de este *Middleware* [27]:

[6] Snappy Ubuntu Core es nuevo sistema operativo para la nube y los dispositivos. Ofrece una mayor seguridad, actualizaciones confiables y el enorme ecosistema de Ubuntu.

- Agnóstico de la industria respaldada por expertos de *DataArt* compañía global de Ingeniería de Software.
- Fuente abierta, transparente y libre. Gratis para uso comercial con enfoques modernos de seguridad y tolerante a fallos.
- Admite nubes públicas y privadas e implementación híbrida se integra fácilmente con otros productos, servicios y soluciones empresariales.
- Conecta cualquier dispositivo utilizando *REST API, WebSockets o MQTT*. *Node.js, Python* y *firmware* personalizado.
- Enfoque de arquitectura orientada a servicios basada en contenedores con escalabilidad lineal, gestionada por *Kubernetes* para cargas de producción.

DeviceHive posee una amplia gama de opciones de integración. En la **Figura 2.4** se esquematiza los elementos primordiales en este *Middleware* acompañado de una breve explicación de algunos de ellos.

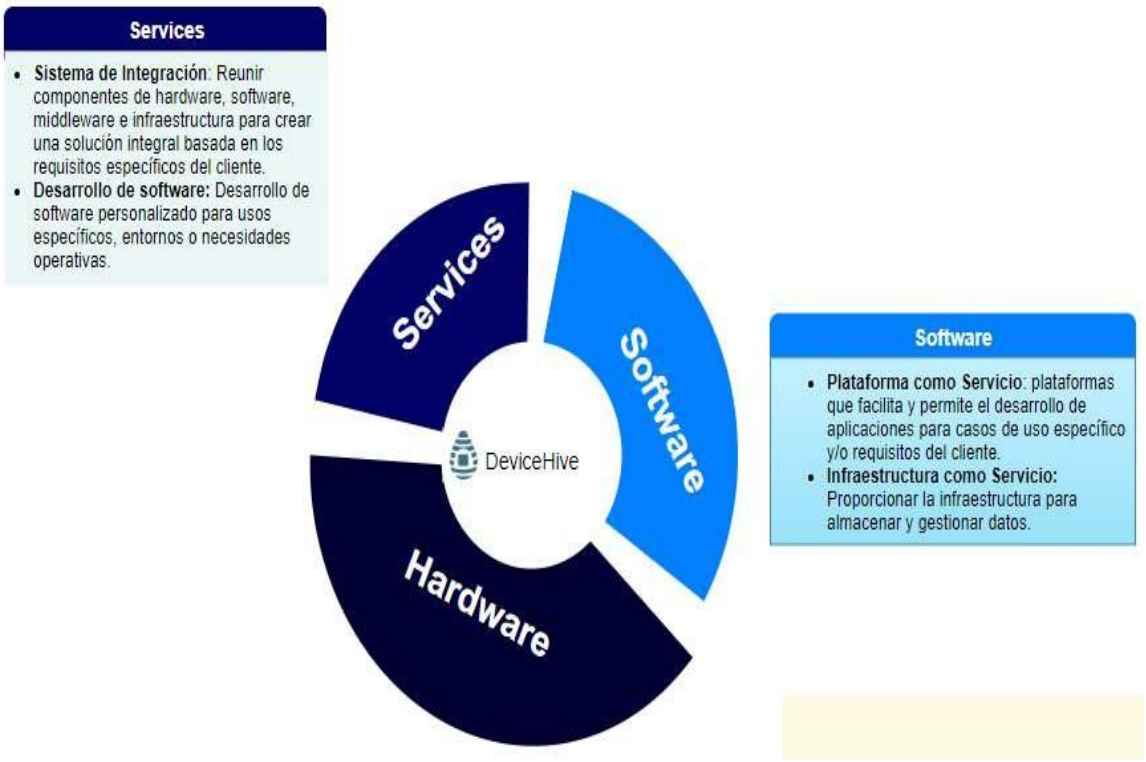

Figura 2.4. Elementos primordiales en *DeviceHive.*

2.1.2.1 Arquitectura [27]

DeviceHive se basa en microservicios altamente escalables y disponibles. Los datos de series temporales (datos de sensores) son siempre un desafío, *DeviceHive* permite, no solo escuchar cientos de dispositivos al mismo tiempo, sino también escalar a la cantidad requerida de instancias para garantizar la seguridad y disponibilidad de los datos. A continuación, en la **Figura 2.5** se expone la arquitectura de los microservicios de *DeviceHive* y se detalla brevemente cada uno de estos.

a. PostgreSQL

Está ligado con el Servicio *Backend de DeviceHive.* Requiere almacenamiento RDBMS para mantener toda la metainformación (datos fríos en términos de arquitectura Lambda[7]). Incluidos datos que no son de series temporales tales como: datos sobre dispositivos, usuarios, y datos de configuración, es decir, es la base de datos *backend*.

b. Hazelcast IMDG

Ocasionalmente es preciso almacenar datos de series de tiempo por un largo período; generalmente trabajos analíticos. Al mismo tiempo, sigue siendo necesario el acceso inmediato a los datos. Para estos fines, se utiliza el microservicio *Hazelcast* IMDG que es una red de datos en memoria agrupada (*clustered in-memory data grid*). Todas las notificaciones se guardan en caché distribuido para un acceso más rápido y se eliminan en dos minutos de forma predeterminada (el límite de tiempo es programable).

c. Bus de mensajes (*Kafka*)

Kafka se encarga de la comunicación entre servicios y equilibra la carga entre ellos. Es un sistema de mensajería rápida, distribuida y tolerante a fallos. Cubre todas las necesidades del cliente. Actualmente se utiliza *Websocket Kafka Proxy [27]*, microservicio escrito en Node.js. Provee más flexibilidad sin comprometerse con ninguna implementación de bus de mensajes. Todos los servicios implementan la API del cliente y no dependen de ningún método o configuración específicos de Kafka.

d. Servicio *Frontend DeviceHive*

Servicio que proporciona *API RESTful* y *Websocket*. Responsable de las verificaciones primarias, enviando solicitudes al servicio *Backend* y entregando respuesta de forma asincrónica. También procesa algunas llamadas ligeras a DB.

[7] La arquitectura Lambda es un conjunto de principios para sistemas Big Data en tiempo real. Permite lecturas de baja latencia y actualizaciones de forma linealmente escalable y tolerante a fallos.

e. Servicio de Autenticación *DeviceHive*

Toda la autenticación en *DeviceHive* se realiza mediante tokens *JWT* (*JSON Web Tokens*). Contienen toda la información sobre privilegios de usuario, dispositivos disponibles, redes o tipos de dispositivos. El servicio *DeviceHive Auth* provee una *API RESTful* para generar, validar y actualizar estos tokens.

f. Servicio de *backend de DeviceHive*

Responsable de almacenar datos en *Hazelcast*[8], gestionar suscripciones y recuperar datos por solicitud de otros servicios (*Hazelcast* o DB). No tiene API de acceso público, toda la comunicación con ella se realiza a través de *Message Bus (Kafka).*

g. Servicio de Complemento *DeviceHive* (opcional)

DeviceHive crea pequeñas aplicaciones denominadas complementos que hacen su propia lógica comercial. El Servicio de Complemento *DeviceHive* administra estas aplicaciones y requiere un proxy W*ebsocket* más el de conectividad de complementos. *DeviceHive* facilita plantillas de complementos en los lenguajes: *Node.js, Python* y Java como asistencia al usuario al momento de crear sus propios complementos.

Para que un usuario registre su propio complemento, debe definir algunos parámetros de consulta (dispositivos, redes, etc.) y recibirá la dirección del proxy W*ebsocket* con los *tokens* JWT requeridos. Si la autenticación fue exitosa, todas las notificaciones solicitadas se enviarán a la sesión de W*ebsocket* desde la nube *DeviceHive*.

h. Complemento MQTT (opcional)

El complemento *DeviceHive MQTT* es la capa de transporte entre los clientes *MQTT* y el servidor *DeviceHive*. El corredor utiliza sesiones de *WebSocket* para comunicarse con *DeviceHive Server* y el servidor *Redis*[9] (*Remote Dictionary Server*) para la funcionalidad de persistencia. *DeviceHive* admite el protocolo de comunicación *MQTT* mediante el uso de nombres específicos para enviar datos a la nube *DeviceHive*. En caso de utilizar diferentes convenciones de nomenclatura, el *plugin* se utilizará como un *bróker MQTT* ordinario.

[8] Hazelcast, sistema de almacenamiento distribuido *Open Source* permite administrar datos y distribuir el procesamiento usando el almacenamiento en memoria y la ejecución paralela.

[9] Redis, tipo de servidor de memoria rápida para datos, provee una base de datos en memoria como de clave – valor (key-value store). Funciona como memoria caché.

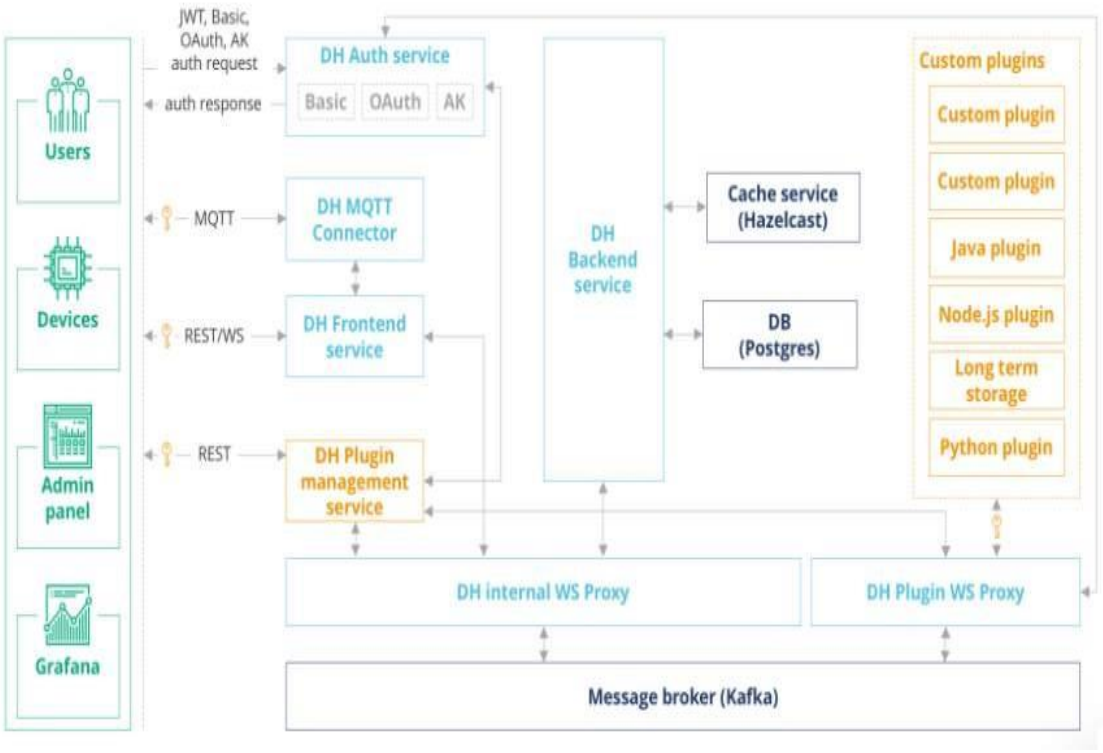

Figura 2.5. Arquitectura de microservicios *DeviceHive* [27].

2.1.1.2 Servicio de Administración de Complementos [27]

El Servicio de Administración de Complementos es un microservicio adicional de *DeviceHive*, que permite gestionar las suscripciones de notificaciones o comandos del dispositivo. Para registrar, eliminar y actualizar complementos, este servicio se integra con *Swagger*[10]. El complemento admite varias combinaciones de suscripciones: por redes, dispositivos o tipos de dispositivos. A continuación, se detallan cada una de estas combinaciones:

a. Servicio de Administración de Complementos

Microservicio que provee información sobre complementos después del registro, además, permite actualizar y eliminar el complemento.

b. Servicio de Autenticación

Microservicio que crea y actualiza los tokens JWT del usuario y del complemento.

[10] Swagger es una serie de reglas, especificaciones y herramientas que permiten documentar APIs. Permite crear documentos realmente útiles para las personas que lo necesitan y para que todos lo comprendan.

c. *WebSocket Kafka Proxy*

Servicio *Node.js* incluye algunas de las funciones esenciales del intermediario de mensajes, permite comunicarse con *Apache Kafka* a través de *WebSockets*. La instancia interna establece la comunicación entre los microservicios, la instancia del complemento permite a los usuarios trabajar con los complementos creados.

d. Kafka TOPIC_N

Utilizado por usuarios dedicados con mensajes de suscripción (comandos o notificaciones). Adicionalmente *DeviceHive* cuenta con diferentes plantillas para los Servicios de Administración.

2.1.2.3 Seguridad y Componente *Cassandra*

Para *DeviceHive* es importante mantener las cosas seguras y protegidas. La autenticación está protegida por JWT. Los beneficios claves de este enfoque son [27]:

- **Tokens:** No tienen estado y son autónomos, no es necesario almacenarlos en el servidor, son compactos y fáciles de transmitir.
- **Portabilidad:** Un token se puede utilizar con varios *backends*.
- **Descentralizado:** Se puede generar en el servidor independiente.
- **Fecha y hora de vencimiento:** Son configurables.
- **Conectividad:** Utiliza conectividad TLS (*Transport Layer Security*) para la comunicación de dispositivos y aplicaciones.
- **Acceso:** El acceso para usuarios, redes y dispositivos se basa en roles.
- **Conexión:** Ofrece conexión segura con HTTPS (proporciona una guía de configuración) y *WebSocket* .

En cuanto al complemento ***Cassandra*****,** se conoce que es un complemento que permite almacenar comandos y notificaciones obtenidas por medio de la plataforma *DeviceHive*, tal como se observa en la **Figura 2.6**.

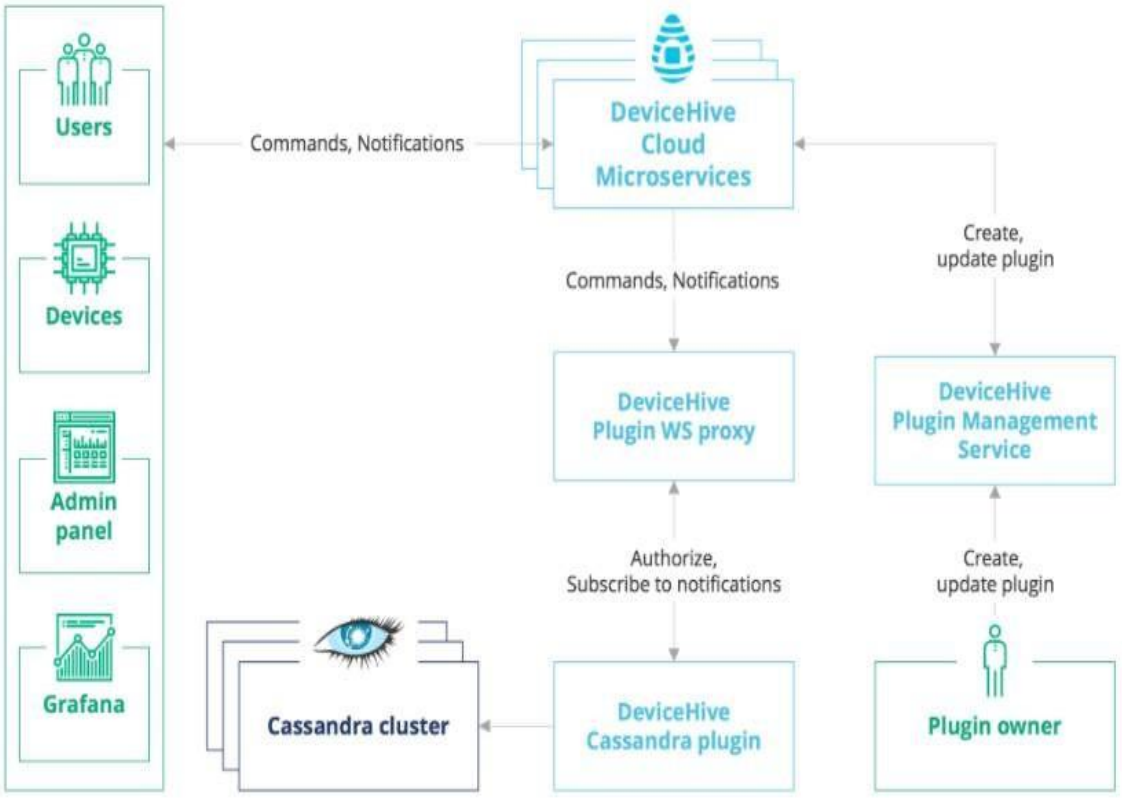

Figura 2.6. Complementos *Cassandra* [27].

La aplicación consta de dos partes: el servicio de creación de esquemas y el complemento definido en el archivo *docker-compose*. Al iniciar el servicio, el proceso de creación de esquemas se ejecutará, primero para crear tablas y esquemas UDT (*User Defined Types*) desde JSON y el complemento verificará el estado del proceso utilizando un intervalo predefinido y un número de verificaciones. El servicio de creación de esquemas siempre debe funcionar solo en un nodo para evitar la modificación simultánea del esquema que causa excepciones en ***Cassandra*****,** sin embargo, es posible escalar el Servicio de Complementos tanto como se necesite.

Al iniciar el tiempo de ejecución, tanto el complemento como el servicio de creación de esquemas compararán la tabla existente y los esquemas UDT. En caso de una discrepancia o una falta de coincidencia de columna / campo, una falta de coincidencia de claves primarias y de agrupación, o una falta de coincidencia de orden, la aplicación fallará.

En caso de que ya exista un UDT o una tabla, se le notificará al usuario. El mensaje que llega, puede ser un comando, actualización de comando o notificación, dependiendo del tipo de datos que se insertarán en el grupo de tablas apropiado. Además, los datos se filtrarán para que coincidan con el esquema de tabla descrito.

2.1.2.4 Aplicaciones [27]

DeviceHive es participe en algunas industrias las cuales se detallan a continuación en la **Tabla 2.10.**

Tabla 2.10. Aplicaciones de *DeviceHive.*

Automotor	*DeviceHive* permite digitalizar servicios, recopilar y analizar datos de vehículos, conectarlos a la nube o integrarlos con entornos inteligentes. Fácil de integrar con proveedores de datos: mapas, servicios de atascos, estacionamiento inteligente, API de taxis, etc. Utiliza algoritmos de aprendizaje automático para crear las mejores soluciones de asistencia a la conducción en tiempo real. Aborda la conectividad y recopila datos de OBD II (*On Board Diagnostics* II), CAN (*Campus Area Network*), sensores o cualquier otro dispositivo inteligente instalado en un automóvil. Entre los principales beneficios están: • Monitoreo, mantenimiento de vehículos y análisis de hábitos de conducción y seguros inteligentes. • Integración de entorno inteligente y rastreo inteligente de flotas y mercancías.
Energía y Servicios Públicos	*DeviceHive* ofrece una visión holística de la tecnología para empresas de servicios públicos. Provee información valiosa sobre datos para empresas y clientes, permite un uso eficaz de los recursos y una adaptación más rápida y flexible de nuevas tecnologías que responden a la demanda del mercado. La arquitectura de microservicio se integra fácilmente con cualquier infraestructura o solución y ofrece los siguientes beneficios: • Ambientes inteligentes y gestión de activos. Energía verde y automatización. • Mantenimiento predictivo, monitoreo de condiciones y optimización del rendimiento. • Habilitación de la nube e integración de hogares inteligentes gestionando y analizando los datos para informar al cliente.
Ambientes Inteligentes	*DeviceHive* integra a la perfección los componentes, actuando como un sistema nervioso central para entornos inteligentes como ciudades, oficinas, edificios, hogares, fábricas y otros tipos de infraestructura. Provee los medios y las herramientas para analizar datos, aprendizaje automático, inteligencia artificial, gestionar dispositivos y servicios de acuerdo con información valiosa. Entre los beneficios principales están: • Gestión de activos y automatización de servicios urbanos. • Transporte, monitoreo de tráfico, balanceo y logística. • Edificios, ascensores y estacionamientos inteligentes con sistemas de seguridad y protección. • Energía renovable, monitoreo ambiental y previsiones basadas en datos.
Salud y Ciencias Biológicas	En el campo de la salud y las ciencias biológicas *DeviceHive* proporciona algunos beneficios que se enuncian a continuación: • Prototipos rápidos y administración de dispositivos médicos. • Entorno hospitalario inteligente. • Automatización y mantenimiento predictivo. • Análisis y gestión de datos.
Viajes / Hospitalidad	*DeviceHive* permite optimizar recursos y servicios. Automatizar y gestionar operaciones de forma eficaz. Aplica automáticamente las preferencias de invitados en todos los lugares visitados. Se integra sin problemas y de forma segura con un hotel, lo que permite controlar cada función y hacerla inteligente. A continuación, se enuncian los principales beneficios: • Automatización y control de entorno inteligente. • Gestión de dispositivos, medios, entretenimiento y personalización. • Salas inteligentes, optimización de recursos y mantenimiento predictivo.

En el Anexo I se presenta las pautas necesarias para la configuración e instalación de las distintas herramientas que requieren las plataformas *OpenIoT y DeviceHive*, así como los aciertos y desaciertos durante este proceso. Estas herramientas pueden ser implementadas en el sistema operativo *Windows o Linux*.

Para cualquiera de los dos escenarios en los que se pretenda implementar la plataforma se requiere descargar los paquetes necesarios para cada una de las herramientas mencionadas en el Capítulo anterior, sin embargo, se ha escogido a Linux como sistema operativo para ambas plataformas debido a que *DeviceHive* requiere de gran capacidad de memoria y ocasiona que el sistema se ralentice y no permita concluir con la instalación de todos los componentes y mucho menos es posible garantizar su rendimiento.

Una vez que se logró corregir los errores que se presentaron en la instalación de las distintas herramientas que hacen posible el funcionamiento de las plataformas *OpenIoT y DeviceHive*, en el siguiente Apartado se procederá a realizar análisis comparativo de dichas plataformas fundamentado en métricas cualitativas basado en el *framework* SMI.

3 RESULTADOS, CONCLUSIONES Y RECOMENDACIONES

Una vez determinado el comportamiento en la instalación de las dos plataformas *OpenSource* escogidas: *OpenIoT y DeviceHive*; se valorará, evaluará y comparará el desempeño de las mismas; utilizando el *framework* SMI definido en el **Capítulo 2**, para lo cual se considerará los atributos cualitativos de estas dos plataformas.

Como ya se había mencionado anteriormente los atributos cualitativos se medirán mediante una escala ordinal que consta de un conjunto de calificaciones predefinidas, como: buena, alta, media, regular, excelente, etc.

3.1 MÉTRICAS UTILIZADAS

En este Proyecto de Titulación se ha realizado una investigación de las plataformas *Middleware* IoT que podrían usarse para resolver problemas o emprender proyectos a nivel individual o empresarial. Una vez finalizada la investigación y la instalación de las plataformas *OpenIoT* y *DeviceHive* se evaluará el desempeño de las misma basado en atributos cualitativos utilizando una escala ordinal considerando la valoración expuesta en la **Tabla 3.1.**

Tabla 3.1. Escala de Valoración.

Escala de Valoración	
EXCELENTE	96-100%
ALTA	90-95%
MEDIA	80-89%
BUENA	≥70%

Adicionalmente se detallará propiedades y características como: monitoreo, alertas, almacenamiento de datos, análisis en tiempo real, registro y administración de dispositivos, interfaz, protocolo, empresa de soporte, licencia, fecha de inicio, documentación, *committers* y código fuente.

- **Administración de Plataforma:** Tanto *OpenIoT* como *DeviceHive* cuentan con herramientas que permiten una mejora continua en cuanto a sus procesos internos se refiere, lo cual asegura el funcionamiento de cada uno de los componentes y sistemas de estas plataformas. En el caso de *DeviceHive* cuenta con su propia consola de administración.

- **Alertas- Registro (log):** En el caso de *OpenIoT* no se cuenta con un componente o servicio en particular dedicado a realizar está función, mientras que en *DeviceHive* flujo de datos disponible de los topics de Kafka contiene notificaciones de dispositivos; sus componentes trabajan con *plugins* que se realizan las funciones de alertas.

- **Almacenamiento de datos:** *OpenIoT* garantiza el almacenamiento de grandes flujos de datos provenientes del middleware del sensor con *LSM-Light* y una capa de almacenamiento administra el flujo de datos permanente o transitorio de los dispositivos, mientras que en *DeviceHive* el componente *PostgreSQL* es la base de datos de *back-end* quien almacena todos los datos de dispositivos, usuarios, redes y ajustes de configuración.

- **Análisis en tiempo real:** Para el caso de *DeviceHive* la agregación y el análisis de datos recibidos de los sensores sobre la marcha permite crear un sistema de monitoreo en tiempo real mediante *Cassandra y Kafka*, mientras que en *OpenIoT* la nube CUPUS distribuye notificaciones automáticas desde la nube a destinos ampliamente distribuidos casi en tiempo real.

- **Código fuente:** En este caso tanto *OpenIoT* como *DeviceHive* proveen su código fuente en el repositorio de *github* disponible para todos los usuarios.

- ***Commiters*:** Se encuentran disponibles en el repositorio de *github* de cada plataforma, *DeviceHive* cuenta con un número mayor de *commiters* con respecto a *OpenIoT* que posee un número menor, muy posiblemente debido a las desactualizaciones de esta plataforma.
- **Documentación:** En cuanto a la documentación *DeviceHive* provee en su página oficial toda la información necesaria para su implementación, mientras tanto *OpenIoT* debido a desactualizaciones no cuenta con una página oficial, sin embargo, la documentación está disponible en el repositorio de *github*.
- **Empresa de soporte:** Tanto *OpenIoT* como *DeviceHive* son autónomas.
- **Entorno:** Tanto *OpenIoT* como *DeviceHive* trabajan en el entorno de Java, adicionalmente *DeviceHive* trabaja también con Go.
- **Fecha de Inicio:** De acuerdo a la documentación ofrecida tanto en la página oficial de *DeviceHive* como en el repositorio de *github* para el caso de *OpenIo*T la fecha de inicio fue en el año 2012.
- **Interfaz:** *DeviceHive* trabaja con *REST, API, MQTT*, mientras que *OpenIoT* solo lo hace con *REST*.
- **Licencia:** La información en cuanto a la licencia de cada una de las plataformas está disponible en la documentación disponible al usuario.
- **Monitoreo**: Para *OpenIoT* la monitorización es una de las principales funcionalidades mediante *JavaMelody* proporciona información de monitoreo para todo el entorno del host, como para cada componente individual de la plataforma, mientras que *DeviceHive* proporciona herramientas de monitoreo que no requieren conectar inicialmente *hardware* real a la plataforma para monitorear la conectividad de los dispositivos como es el caso de *Hazelcast IMDG*.
- **Protocolo:** *DeviceHive* permite los protocolos *MQTT* y *WebSocket* API. Adicionalmente parara todos los servicios *RESTful*, proporciona la herramienta *Swagger* API que permite probar la instalación y otras capacidades. Por su parte la plataforma *OpenIoT* no cuenta con algún protocolo en particular, como ya se ha mencionado en el Capítulo 2 *OpenIoT* trabaja en conjunto con otras herramientas que a su vez utilizan otras tecnologías que permitirán la comunicación con los protocolos correspondientes.

- **Registro y administración de dispositivos:** Como ya se había mencionado en el Capítulo 2, *DeviceHive* se basa en microservicios escalables, y es independiente del hardware y de la nube con API de administración de dispositivos en diferentes protocolos, por su parte *OpenIoT* mediante las funciones y servicios de su componente *Global Scheduler* se encarga de la administración y registro de dispositivos y servicios.

- **Seguridad:** *DeviceHive* proporciona seguridad en cuanto a autenticación por medio de tokens, conectividad TLS para la comunicación de dispositivos y aplicaciones, y conexión HTTPS y *WebSocket*, mientras que *OpenIoT* maneja la seguridad bajo tres módulos: Servidor de seguridad denominado CAS para autenticación y autorización, Cliente de seguridad para aplicaciones web que actúan directamente con el usuario y Consola de administración de seguridad manejado con el módulo *security managment.*

Una vez detalladas las mencionadas propiedades tanto de *OpenIoT* como de *DeviceHive* a continuación, en la **Tabla 3.2.** se muestra un resumen de estas capacidades y características.

Tabla 3.2. Capacidades y características de *OpenIoT y DeviceHive.*

	DeviceHive	***OpenIoT***
Administración de Plataforma	Si	Si
Alertas- Registro (log)	Vía *plugins*	No especificado
Almacenamiento de datos	Sí	Sí
Análisis en tiempo real	Sí	Sí
Código fuente	https://github.com/devicehive	https://github.com/OpenIotOrg
Commiters	Aprox 33	12
Documentación	https://devicehive.com	https://github.com/OpenIotOrg/openiot/wiki/Documentation
Empresa de soporte	*DeviceHive*	*OpenIoT*
Entorno	Java, Go	Java
Fecha de Inicio	Aprox 2012	2012
Interfaz	*REST, API, MQTT*	*REST*
Licencia	Apache 2.0	LGPL V3.0
Monitoreo	Sí	Sí
Protocolo	*WebSockets MQTT*	No especificado
Registro y administración de dispositivos	Si	Sí
Seguridad	Sí	Sí

3.2 VALORACIÓN DE MÉTRICAS CUALITATIVAS

Si bien es cierto las métricas cualitativas son métricas difíciles de traducir en números y el utilizar una comparación cualitativa tradicional está sujeta a la subjetividad y no ayudaría a

elegir adecuadamente un *Middleware,* el análisis comparativo propuesto ofrece una mirada global de las plataformas de estudio gracias a las métricas cualitativas correctas que maneja SMI para recopilar (y posteriormente medir) la información adecuada. Es por ello que en este Proyecto se plantea utilizar SMI con el objetivo de reducir la subjetividad y filtrar el *Middleware* más óptimo para una solución determinada.

Los aspectos importantes de la comparativa cualitativa tomando como referencia la **Tabla. 1.10 del Capítulo 1 Apartado 1.4.6.2** según SMI son las siguientes: Usabilidad, Instalación, Aprendizaje, Seguridad y Flexibilidad. A continuación, se expone la comparativa entre las plataformas *OpenIoT y DeviceHive* de acuerdo los aspectos mencionados.

3.2.1 ANÁLISIS DE USABILIDAD

Al hablar de usabilidad de acuerdo a lo especificado en la norma ISO 25000 [17], las métricas de usabilidad se analizan desde dos perspectivas: desde el punto de vista del software y desde el punto de vista de "uso". En cuanto al primer punto *OpenIoT* no cuenta con actualizaciones, la última actualización fue realizada aproximadamente en 2015; *DeviceHive* a pesar de que la última actualización de su página oficial fue en 2018, sus repositorios se mantienen activos hasta aproximadamente el 2020; desde el punto de vista de uso, al no contar con actualizaciones el número de usuarios que utilizan *OpenIoT* ha disminuido, contrarrestado con el número de usuarios de *DeviceHive* que va en aumento. Otras métricas de usabilidad son la "facilidad para aprender" a usar *Middleware*, la eficiencia y la satisfacción del usuario, donde *OpenIoT* ofrece una mayor eficiencia con una interfaz de usuario bastante intuitiva con un mínimo esfuerzo, permitiendo la visualización de información correspondiente a los servicios de IoT en diferentes formatos como mapas, gráficos de diferentes tipos o paneles. Además, proporciona la capacidad de desarrollar aplicaciones prácticamente sencillas sin necesidad de programación, lo que agiliza la creación de dichas aplicaciones. Por el contario de *DeviceHive,* en la cual se debe trabajar en su mayor parte con códigos para obtener resultados y no incluye paneles de visualización, pero permite enviar los datos a otras aplicaciones para proporcionar los resultados de forma gráfica, en este sentido, la satisfacción del usuario que proporciona *OpenIoT* es mayor.

3.2.2 ANÁLISIS DE INSTALACIÓN

Con respecto a la instalación, *OpenIoT* proporciona la documentación necesaria y de fácil acceso en el repositorio de *github.* Ofrece la opción de instalar la plataforma en Windows o Linux, inicialmente se optó por realizarlo en Windows, sin embargo durante el proceso se

presentaron inconvenientes debido a paquetes que no eran compatibles y a pesar de no presentarse errores al final de la instalación no se conseguía el despliegue de la plataforma; por lo tanto, se inició la instalación en Linux, específicamente en Ubuntu en el cual se logró superar los inconvenientes y se logró el despliegue de la plataforma, no obstante, debido a desactualizaciones de la misma no se pudo explorar otras funcionalidades.

DeviceHive permite innumerables opciones de implementación y es adecuado para todo tipo de organizaciones, ya sea en una empresa que apenas está iniciando o una gran empresa con trayectoria. Esto incluye implementar *Docker Compose, Kubernetes* para facilitar el desarrollo de nubes públicas, privadas o híbridas. Adicionalmente ofrece la opción de *Playground* e instalación manual. Se presentan varios servicios de *DeviceHive* con *Docker Compose: DeviceHive Frontend Service, DeviceHive Backend Service, DeviceHive Authentication Service, DeviceHive Management Console, DeviceHive WebSocket Proxy, DeviceHive Proxy Nginx, Kafka, PostgreSQL y Hazelcast IMDG*. Toda esta información se encuentra disponible en la página oficial de *DeviceHive [27]*.

DockerCompose que fue la opción elegida debido a que es la opción más rápida y reúne todos los contenedores que requiere esta plataforma, así como la compatibilidad necesaria para obtener el despliegue de la misma.

3.2.3 ANÁLISIS DE APRENDIZAJE

En cuanto a la definición de aprendizaje en CSMIC SMI, *OpenIoT* le permite al usuario aprender a usar el servicio sin mucho esfuerzo en poco tiempo, brindándole una interfaz gráfica muy completa donde puede administrar los dispositivos conectados, con una serie de variables como temperatura, humedad; así como, también permite que el usuario pueda desarrollar aplicaciones sin programación, mientras que para *DeviceHive* el esfuerzo y tiempo de aprendizaje que se le debe dedicar es mayor en comparación con *OpenIoT, ya que DeviceHive* no cuenta con paneles de visualización y requiere que los datos sean enviados a otras aplicaciones para presentar y representar la información. Adicionalmente *DeviceHive* permite realizar cualquier cambio para beneficio propio del cliente, sin embargo, no es la manera más fácil, esto requiere el uso de plantillas de complementos de *DeviceHive* disponible para *Java, Node.js y Python,* lo cual implica contar o adquirir conocimiento extra sobre estas aplicaciones alternas. Cabe recalcar que tanto *OpenIoT* como *DeviceHive* cuentan con una interfaz de usuario en idioma inglés.

3.2.4 ANÁLISIS DE SEGURIDAD

Con respecto a las características de seguridad descritas en las métricas cualitativas se puede detallar que e*l módulo OpenIoT Security and Privacy* cuenta con una consola de administración para administrar servicios, usuarios y permisos. De manera predeterminada, esta consola es implementada en una instancia local del Servidor de Aplicaciones *JBoss* con un certificado SSL para autenticar y autorizar el acceso a sus recursos. Adapta un servidor CAS habilitado para OAuth2.0 (versión 3.5.2) para autenticación y autorización. Su configuración permite almacenar datos en un servidor *LSM-Light*. La configuración de CAS puede ser modificada, utilizando el método de superposición de *Maven2 war*.

Adicionalmente *OpenIoT* proporciona seguridad a través del módulo *security-server que* depende de los componentes *utils.commons* y *lsm-light.client, d*el módulo *security-management* que funciona en conjunto con la consola de administración y del módulo *security-client* que proporciona utilidades de autenticación y control de acceso. Puede ser utilizado en aplicaciones web que proporcionan una interfaz de usuario e interactúan directamente con los usuarios. Además, puede ser utilizado en proveedores de servicios intermedios que interactúan con otros componentes a través de *WebServices* también conocidas como REST.

Por su parte *DeviceHive* proporciona autenticación protegida por JSON Web Tokens (JWT). Lo beneficioso es que el token no tiene estado y es autónomo, no es necesario almacenarlo en el lado del servidor; puede ser usado con múltiples *backends.* Además, se reduce el tiempo de cómputo y de red. El mecanismo de autenticación mediante JSON Web Tokens trabaja mediante métodos de acceso, creación, actualización, complemento/creación (crea tokens de acceso y actualización de JWT para el complemento) y complemento/autenticación (autentica un complemento y la carga útil del complemento JWT) de tokens.

Adicionalmente *DeviceHive* proporciona conectividad TLS para la comunicación de dispositivos y aplicaciones y el acceso para usuarios, redes y dispositivos está basado en roles con una conexión segura HTTPS y *WebSocket*.

3.2.5 ANÁLISIS DE FLEXIBILIDAD

Por último, la flexibilidad para agregar o eliminar funciones predefinidas de un servicio tanto *OpenIoT* como *DeviceHive* proporcionan una capacidad de adaptación en cuanto a las preferencias de los usuarios. La infraestructura de *OpenIoT* proporciona una configuración

flexible y la creación de algoritmos que permiten recoger y filtrar los diferentes flujos de datos procedentes de los objetos conectados. Además de su versatilidad para anotar semánticamente datos de cualquier dispositivo de E/S disponible. *OpenIoT* también tiene una función especial que puede ayudar fácilmente a vincular conjuntos de datos entre sí, según sus similitudes. Por lo tanto, es posible manejar dinámicamente un flujo de datos específico que también es compatible con sensores móviles, incluso sin interfaces extrañas.

DeviceHive provee una configuración flexible a través de su arquitectura basada en microservicios que a más de permitir agregar o eliminar funciones, permite comunicarse con casi todos los dispositivos que admitan los protocolos ya mencionados, solo con implementar una lógica ordinaria. Los dispositivos compatibles con *Python, Node.js* o Java, como placas Linux, dispositivos *Android Things*, etc., se pueden conectar fácilmente simplemente instalando la biblioteca cliente de *DeviceHive*.

3.3 VALORACIÓN SMI

En base al análisis detallado y la escala ordinal anteriormente mencionados, se expone un mapa de calor que se puede observar en la **Tabla 3.3** junto con gráfico de barras en la **Figura 3.1** que representan el conjunto de calificaciones para *OpenIoT* y *DeviceHive* en cuanto a las métricas cualitativas.

Tabla 3.3. Mapa de Calor Métricas Cualitativas *OpenIoT – DeviceHive.*

PLATAFORMAS	MÉTRICAS CUALITATIVAS				
	Usabilidad	**Instalación**	**Aprendizaje**	**Seguridad**	**Flexibilidad**
OpenIoT	Media	Buena	Excelente	Media	Alta
DeviceHive	Alta	Alta	Media	Excelente	Alta

Figura 3.1. Calificación Métricas Cualitativas de *OpenIoT y DeviceHive.*

Finalmente fue posible ofrecer una perspectiva holística de las plataformas IoT haciendo énfasis en *OpenIoT y DeviceHive* a través de la instalación de dichas plataformas lo que implica un arduo trabajo, no solo por el conocimiento básico que requiere, sino también debido a las distintas herramientas y componentes que hacen posible el desarrollo y funcionamiento de estas tecnologías.

Basándonos en los resultados obtenidos en cuanto a las métricas cualitativas y capacidades se puede concluir que si bien es cierto la calificación de ambas plataformas de acuerdo a la escala utilizada no difiere significativamente; *DeviceHive* se encuentra en nivel superior con respecto a *OpenIoT*, especialmente por la falta de actualizaciones lo que conlleva a problemas insuperables en su implementación o en el desenvolvimiento de alguno de sus componentes, incluso podrían presentarse vulnerabilidades en la seguridad del sistema.

No obstante, la plataforma *OpenIoT* podría ser la opción más viable en el ámbito educativo que como se mencionó en el Capítulo 2, es una de las aplicaciones más representativas de esta plataforma. Además, con respecto a la métrica cualitativa de Aprendizaje, *OpenIoT* ofrece una amigable interfaz gráfica que puede ser de mucha ayuda a usuarios que apenas están incursionando en este mundo de interconexión y permitir que los conocimientos sean perfeccionados para futuras aplicaciones. Por su parte *DeviceHive* podría ser la mejor opción para aplicaciones en ambientes inteligentes orientado especialmente al sector automotriz, servicios públicos, empresarial y otros tipos de infraestructura, ya que son los sectores en los cuales esta plataforma ofrece mayores beneficios como se expuso en el **Apartado 2.1.2.4**, así como también la alta Flexibilidad para incorporar nuevas funciones y conectarse con dispositivos compatibles, además de su excelente Seguridad gracias al mecanismo de autenticación que maneja y una conexión segura.

3.4 CONCLUSIONES

Del análisis realizado en este Proyecto de Integración Curricular, así como de la información y evidencia proporcionada anteriormente, se pueden extraer las siguientes conclusiones:

- El objetivo general de este Proyecto de Integración Curricular es analizar comparativamente plataformas Middleware fundamentado en métricas cualitativas bajo el *framework* SMI, con el fin de probar la efectividad de las mismas, el cual fue cumplido mediante un estudio de tipo explicativo y descriptivo.

- El estudio realizado y la implementación de *OpenIoT* y *DeviceHive*, se logró realizar el análisis comparativo dejando de lado la subjetividad y proporcionando una guía al usuario al momento de elegir una plataforma como herramienta de trabajo y acorde a sus requerimientos.
- El implementar una plataforma, cualquiera que esta sea, se requiere de herramientas adicionales que permitan su funcionamiento, lo cual implica conocimientos previos, o a su vez nuevos y por ende involucra un arduo trabajo investigativo que permita profundizar sobre esta temática y contar con personal capacitado para las diferentes funciones que requiere la implementación de una plataforma IoT.
- El *framework* SMI fue primordial en este Proyecto de Integración Curricular, al ser la herramienta que permitió evaluar las plataformas *OpenIoT y DeviceHive* y realizar una comparativa entre ellas mediante un método estandarizado.
- Si bien es cierto en la comparativa realizada ninguna de las dos plataformas posee una baja calificación de acuerdo a los resultados obtenidos en base a SMI, sin embargo, *OpenIoT* no cuenta con actualizaciones y es algo que no puede pasar desapercibido, ya que esto dificulta trabajar con la misma, razón por lo cual se podría decir que bajo este escenario el Middleware más idóneo es *DeviceHive* por sus múltiples servicios y prestaciones con un alto margen de seguridad y disponibilidad.
- Es de gran ayuda el tener acceso a plataformas *Middleware Open Source* mediante repositorios en su mayoría ubicados en GitHub, lo cual permite que usuarios inexpertos conozcan y aprendan de estas herramientas, y que a su vez usuarios con mayor experiencia y de conocimientos sólidos diseñen nuevos proyectos o realicen mejoras en colaboración con la comunidad IoT.
- Las métricas cualitativas no pueden expresarse numéricamente; sin embargo, esto no implica que sean menos importantes o insignificativas. Al ser cualificadas mediante SMI dejan de ser un conjunto de características comunes y corrientes y evolucionan en un conjunto de atributos capaces de reducir la subjetividad al momento de realizar una elección.

3.5 RECOMENDACIONES

Una vez concluido el presente estudio, y en base a los resultados obtenidos, se pone a consideración las siguientes recomendaciones:

- Se recomienda indagar sobre otras áreas del IoT como: en el sector financiero, educativo, empresarial y proponer la implementación y uso de plataformas IoT en el país, ya que son tecnologías que facilitan el enlace entre dispositivos, potenciando el acceso a datos, así como la optimización en sus procesos, pero lamentablemente muy poco utilizadas en Ecuador.
- Como se mencionó en el desarrollo de este Proyecto de Integración Curricular, un análisis de plataformas IoT requiere no solo del estudio de estás como tal, sino también del estudio y conocimiento de herramientas adicionales que permiten el funcionamiento de las mismas, las cuales están sujetas a modificaciones, actualizaciones e incluso eliminación de ciertos componentes y que de no ser consideradas pueden dificultar o incluso impedir la culminación del análisis. Por ello es recomendable revisar toda la documentación y considerar todos los posibles inconvenientes que pueden presentarse a futuro en el momento de escoger una determinada arquitectura.
- Es recomendable indagar y evaluar otras plataformas IoT bajo el margen SMI realizando la medición de servicios utilizando métricas cuantitativas como tiempo de conexión, latencia, error de perdida de paquetes y disponibilidad.
- Se recomienda contar con un dispositivo que tenga buenas prestaciones, ya que la implementación de las plataformas consume gran cantidad de memoria y produce que el computador se ralentice lo cual dificulta explorar los servicios y módulos que ofrecen las plataformas.
- Se debería fomentar el estudio del mundo IoT a nivel educativo por medio de seminarios que permitan adquirir los conocimientos necesarios para crear e implementar proyectos en este ámbito tecnológico.

4 REFERENCIAS BIBLIOGRÁFICAS

[1] IoT Platforms Market Report (2018-2023). [Online]. Available: https://iot-analytics.com/product/iot-platforms-market-report-2018-23/.

[2] CSMIC, “Service Measurement Index Framework Version 2.1,” 2014. [Online]. Available: http://csmic.org/downloads/SMI_Overview_TwoPointOne.pdf.

[3] W. S. Mumphrey, «Capítulo 7. Ingeniería del Software», p. 145.

[4] «Ingeniería de software: Qué es, objetivos, características y más», Carreras Universitarias, oct. 20, 2019. https://micarrerauniversitaria.com/c-ingenieria/ingenieria-de-software/. [Accedido: mar-19-2021].

[5] «Todo sobre Plataformas Open Source», studylib.es. https://studylib.es/doc/7646824/todo-sobre-plataformas-open-source. [Accedido: 17-abr-2021].

[6] «The Open Source Definition | Open Source Initiative». Available: https://opensource.org/docs/osd.

[7] «Understanding IoT Platforms: Towards a comprehensive definition and main characteristic description | Request PDF», ResearchGate. Available: https://www.researchgate.net/publication/334566308_Understanding_IoT_Platforms_Towards_a_comprehensive_definition_and_main_characteristic_description.

[8] «(PDF) Role Of Middleware For Internet Of Things: A Study», ResearchGate, doi: 10.5121/ijcses.2011.2307.

[9] CIoT16_Comparison of IoT Platform Architectures - A Field Study based on a Reference Architecture.pdf (uni-stuttgart.de)

[10] «Open Source Hardware Association», Open Source Hardware Association. Available: https://www.oshwa.org/

[11] «The Open Source Definition | Open Source Initiative». Available: https://opensource.org/docs/osd.

[12] (Real Academia Española, “software” en Diccionario de la lengua española, España, 2016)

[13] “Guía Breve de Web Semántica”. [Online]. Available: https://www.w3c.es/Divulgacion/GuiasBreves/WebSemantica. [Accedido: 30-oct-2020].

[14] “Vista General del Lenguaje de Ontologías Web (OWL)”. [Online]. Available: https://www.w3.org/2007/09/OWL-Overview-es.html#s1.2. [Accedido: 01-nov-2020].

[15] “Vocabulario para redes de sensores: Ontología SSN del W3C | datos.gob.es”. [Online]. Available en: http://datos.gob.es/es/noticia/vocabulario-para-redes-de-sensores-ontologia-ssn-del-w3c. [Accedido: 26-oct-2020].

[16] M. Compton, P. Barnaghi, L. Bermudez, R. García-Castro, O. Corcho, S.Cox, J. Graybel, M. Hauswirth, C. Henson, A. Herzog, V. Huang, K. Janowicz, D. Kelsey, D. Le Phuoe, L. Lefort, M. Leggieri, H. Neuhaus, A. Nikolov, K. Page, A. Passant, A. Sheth, K. Taylor “The SSN Ontology of the W3C Semantic Sensor Network Incubator Group”, Published by Elsevier B.V All rights reserved, doi: 10.1016/j.websem.2012.05.003.

[17] «NORMAS ISO 25000». https://iso25000.com/index.php/normas-iso-25000 [Accedido: 24-oct-2022].

[18] «OpenIoT: Enabling the Convergence of IoT and Cloud Computing», Open Source For You, mar. 03, 2019. https://www.opensourceforu.com/2019/03/openiot-enabling-the-convergence-of-iot-and-cloud-computing/. [Accedido: jul-20-2020].

[19] «Introducción a Maven», uqbar-wiki. wiki.uqbar.org/wiki/articles/maven.html. [Accedido: may-16-2021].

[20] l30n4rd0, « (1) New Message! » https://engine-ering.org/2018/09/16/introduccion-a-maven/. [Accedido: may-02-2021].

[21] Plataforma de Datos Virtuoso Arquitectura, Tecnologias y Caso de Estudio.pdf https://riull.ull.es/xmlui/bitstream/handle/915/1165/Plataforma%20de%20Datos%20Virtuoso%20Arquitectura,%20Tecnologias%20y%20Caso%20de%20Estudio.pdf?sequence=1

[22] «Servidores de aplicaciones». http://www.jtech.ua.es/j2ee/2003-2004/abierto-j2ee-2003-2004/sa/sesion1-apuntes.htm. [Accedido: 28-febr-2018].

[23] «Documentation · OpenIotOrg/openiot Wiki · GitHub». https://github.com/OpenIotOrg/openiot/wiki/Documentation. [Accedido: 27-oct- 2022].

[24] «Smart City / Smart Campus – OpenIoT». http://www.openiot.eu/?page_id=52. [Accedido: 28-febr-2018].

[25] A. S. Author, «OpenIoT by Qowisio», AppAdvice. https://appadvice.com/app/openiot/1115129508. [Accedido: 27-oct-2022].

[26] OpenIoT - Phenonet, sensorised farming (CSIRO). [Online Video]. Available: https://www.youtube.com/watch?v=9mbjVll79rM

[27] «DeviceHive - Open Source IoT Data Platform with the wide range of integration options. » https://devicehive.com/.

[28] Semantic Sensor Network Ontology. (2021, 22 November) [Online]. Available: https://www.w3.org/TR/vocab-ssn/.

[29] J. Siegel y J. Perdue, «Cloud Services Measures for Global Use: The Service Measurement Index (SMI)», en 2012 Annual SRII Global Conference, jul. 2012, pp. 411-415, doi: 10.1109/SRII.2012.51

[30] H. Morán, K. Huertas, "Análisis Comparativo De Plataformas Cloud Con Soporte Orientado A Servicios De Internet De Las Cosas", Trabajo de Investigación Tecnológica, Facultad de Ingeniería de Sistemas, U. Católica de Colombia, Bogotá DC, Colombia, 2017.

[31] J. Morenos. "Estudio de las plataformas software existentes para la Internet de las cosas", Memoria proyecto fin de carrera, Dpto. Ingeniería Técnica de Telecomunicaciones, UOC, Catalunya, España, 2015.

[32] M. Caponi & S. Pío Alvarez. (2018, enero). Tutorial_Configuración_SSL_servidores_Tomcat_JBoss_v1.5.odt. [Online]. 9-18. Aviable: https://www.agesic.gub.uy/innovaportal/file/3868/1/tutorial_configuracion_ssl_servidores_tomcat_jboss_v1.5.pdf.

[33] «JBoss Application Server Downloads - JBoss Community». https://jbossas.jboss.org/downloads. [Accedido: 27-oct-2022].

[34] D. Quintão, shiro-faces: JSF 2 TagLib for Apache Shiro. This taglib reimplements all original JSP tags as their Facelets equivalent, so they can be used in JSF projects. 2017. [Online]. Available: https://github.com/deluan/shiro-faces.

5 ANEXOS

El desarrollo del presente Trabajo de Integración Curricular se complementa con el siguiente anexo:

ANEXO I. Manual de Usuario para la instalación de las plataformas *OpenIoT y DeviceHive.*