UNIVERSIDAD DE LA REPÚBLICA
FACULTAD DE INGENIERÍA

Relevamiento de arquitecturas para desarrollo de aplicaciones de Internet de las Cosas

Autores:
Alexis ARRIOLA GARCIA
Gabriela WYNANTS LOMBARDINI

Supervisores:
Eduardo GRAMPIN
Lorena ETCHEVERRY

19 de julio de 2018

Resumen

El fenómeno IoT (Internet of Things) ha cobrado gran protagonismo en este último tiempo, y si bien hay mucha investigación al respecto es un tema que todavía presenta escasa madurez en algunos aspectos, como por ejemplo patrones de arquitectura o utilización de estándares en la implementación de una solución IoT. Esto lleva a que en algunos casos se cuente con información muy variada sobre un mismo tema o, en caso contrario, la información sobre un aspecto en particular sea muy escasa.

Reducir este problema fue unos de los objetivos que se tuvo presente durante la realización de este trabajo. Poder procesar de manera selectiva la información en aquellos aspectos en los que la misma era abundante, y realizar una investigación más a fondo en aquellos temas de los cuales se conocía poco.

En este trabajo se brinda información sobre el estado del arte de este paradigma, así como también sobre algunos de los protocolos más utilizados, los modelos de comunicación y las posibles arquitecturas utilizadas para el desarrollo de soluciones IoT. Además se realiza un relevamiento de algunas de las plataformas más populares en la actualidad, teniendo en cuenta plataformas comerciales y no comerciales, haciendo hincapié en dos de ellas: Fiware e Watson IoT (IBM).

Un objetivo importante que tuvo este trabajo fue la implementación de un prototipo, utilizando las dos plataformas mencionadas anteriormente. El prototipo busca: probar algunas de las características principales a la hora de desarrollar una solución de este tipo, tener una básica experiencia en la utilización de estas plataformas para realizar una comparación a grandes rasgos de las mismas, y poder destacar algunas consideraciones a tener en cuenta a la hora de llevar adelante un proyecto de IoT.

Al final del trabajo se describen las conclusiones obtenidas, y se mencionan algunos de los temas interesantes para realizar como trabajo futuro, entre ellos la investigación de aspectos de seguridad y el estudio del comportamiento de diferentes plataformas ante escenarios de estrés que se asemejen más a proyectos de gran escala en la vida real.

Indice

1 Introducción

El término Internet de las Cosas ha tomado mucha importancia últimamente, no obstante la idea de interconectar dispositivos para monitorearlos y controlarlos ya existe desde hace décadas. En los 70 ya existían sistemas muy básicos que utilizaban la línea telefónica para comunicarse entre sí, pero recién en la década del 90 apareció el primer dispositivo capaz de ser manejado a través de Internet. Se trataba de una tostadora que podía encenderse y apagarse mediante el uso de Internet.

En la actualidad se ha convertido en un concepto popular debido a los grandes avances que se han dado en áreas como:

- Conectividad: las conexiones son de bajo costo y de alta velocidad, lo que permite que esto no sea un problema a la hora de pensar en una solución IoT. Además se utiliza el protocolo IP, que gracias a su uso global cuenta con una definición precisa que permite su fácil incorporación en diferentes plataformas.
- Economía computacional: la industria está investigando y desarrollando sistemas con mayor potencia a un menor costo, y permitiendo que las tecnologías de comunicación se incorporen en dispositivos muy pequeños.
- Análisis de datos: el desarrollo de nuevos algoritmos y gran poder de cómputo, almacenamiento y servicios en la nube permiten más fácilmente la agregación, análisis y almacenamiento de los datos.

A pesar de esta popularidad no existe una única definición universalmente aceptada de lo que es IoT. Existen diferentes definiciones, todas con muchas similitudes entre sí, como por ejemplo, Internet Architecture Board describe en el RFC 7452 [1] que:

"El término "Internet de las Cosas" (IoT) denota una tendencia en la que un gran número de dispositivos embebidos emplean servicios de comunicación ofrecidos por los protocolos de Internet. Muchos de estos dispositivos, a menudo llamados "objetos inteligentes", no son operados directamente por seres humanos, sino que existen como componentes en edificios o vehículos, o se dispersan en el medio ambiente."

Por otro lado la siguiente definición publicada en la revista IEEE Communications Magazine vincula el IoT con los servicios en la nube:

"Internet de las cosas (IoT) es un marco en el que todas las cosas tienen una representación y una presencia en Internet. Más específicamente, Internet de las Cosas tiene como objetivo ofrecer nuevas aplicaciones y servicios que conectan los mundos físico y virtual, en el que las comunicaciones máquina a máquina (M2M) representan la comunicación de base que permite las interacciones entre las cosas y las aplicaciones en la nube."

A partir de todo esto se puede crear un concepto base de lo que es IoT: un paradigma que considera la presencia en el entorno de una variedad de dispositivos que a través de conexiones inalámbrica y/o cableadas son capaces de interactuar entre sí y

cooperar con otros dispositivos para crear nuevas aplicaciones o servicios y de esta manera alcanzar objetivos comunes.

La meta principal de IoT es permitir que las "cosas" se conecten en cualquier momento, en cualquier lugar con cualquier persona utilizando alguna vía/red, permitiendo que de esta manera ciudades, transportes, hogares y muchas otras áreas sean más inteligentes. Los dispositivos obtienen la "inteligencia" tomando y/o permitiendo decisiones relacionadas con el entorno gracias al hecho de que pueden comunicar información sobre sí mismos y pueden acceder a información que ha sido agregada por otros dispositivos.

1.1. Perspectivas futuras y motivación

Alrededor del año 2010, gracias al gran crecimiento en el uso de los teléfonos inteligentes y las tabletas, el número total de dispositivos conectados en el mundo ascendió a 12.5 mil millones, teniendo por primera vez en la historia un número mayor de dispositivos conectados que de personas en el mundo. De acuerdo a varios consultores y organizaciones (como McKinsey & Company, IHS, IDC, Bain & Company [2], entre otras) se estima que para el 2025 existirán alrededor de 75 mil millones de dispositivos conectados.

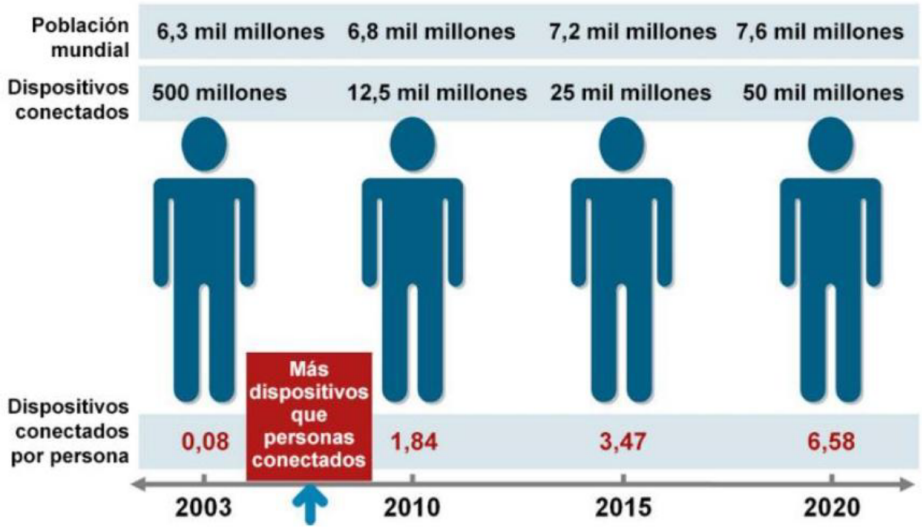


FIGURA 1.1: Evolución de población mundial y dispositivos conectados a Internet. Fuente: Cisco IBSG, Abril 2011

Existen varias razones que motivan el uso de soluciones de IoT. Por un lado hay un enorme número de dispositivos generando continuamente una gran cantidad de datos. Sin embargo, según McKinsey, la mayor parte de estos datos no están siendo utilizados actualmente, y aquellos datos que sí están siendo utilizados sirven principalmente para la detección de anomalías y control, y no para la optimización y predicción de eventos, lo cual proporcionaría un mayor valor. Esta es una razón entonces para el desarrollo de soluciones IoT que permitan la recolección y análisis de datos, para brindar a las empresas mayor productividad, reducción de costos, reducción de riesgos, entre otros.

Si bien actualmente hay un enorme número de dispositivos conectados a Internet, todavía hay algunos aspectos a mejorar para el desarrollo de soluciones IoT, como por

ejemplo, la transición de IPv4 a IPv6, la definición y uso de estándares y el desarrollo de hardware para la optimización del uso de energía por parte de los dispositivos.

1.2. Objetivos

A partir de las perspectivas e idea general establecida en la sección anterior decidimos plantearnos una serie de objetivos con el fin de conocer más a fondo las arquitecturas y plataformas disponibles para el desarrollo de aplicaciones de Internet de las Cosas.

En primer lugar decidimos investigar cinco de las plataformas más utilizadas y populares en la actualidad: Microsoft Azure, IBM Watson, Amazon IoT, Kaa y Fiware. La investigación se enfocará en algunas de las características fundamentales y más relevantes que ofrecen estas plataformas. Luego de hacer una investigación de cada una de estas plataformas se desea compararlas de forma de obtener ventajas y desventajas de cada una de ellas, que podrán tenerse en cuenta a la hora de elegir la plataforma a utilizar en un desarrollo IoT. Si es posible se espera poder realizar al menos un ejemplo sencillo utilizando cada plataforma, para entender cuales son los requerimientos mínimos que presenta cada una de ellas.

Para entender más a fondo algunas de estas plataformas y conocer algunos de sus componentes disponibles se realizará un prototipo utilizando dos plataformas: por un lado Fiware como representante de las plataformas open source, y por otro lado IBM Watson representando a las plataformas comerciales. Se buscará que la mayor parte del código realizado sea reautilizable en ambas plataformas.

Finalmente se realizará una comparación más a fondo de estas ultimas dos plataformas a partir de lo aprendido desarrollando el prototipo.

1.3. Modelos de comunicación IoT

En marzo de 2015, la IAB (Internet Architecture Board, organismo responsable de definir la arquitectura general de Internet marcando guías y orientaciones al IETF), publicó el RFC 7452 [1], el cual es una guía en la que se describen cuatro modelos de comunicación utilizados por los dispositivos en IoT. Estos cuatro modelos se detallaran a continuación:

1.3.1. Comunicación dispositivo a dispositivo (Device-to-Device)

El modelo de comunicación dispositivo a dispositivo representa dos o más dispositivos que se conectan directamente y se comunican entre sí. Estos dispositivos utilizan, frecuentemente, protocolos como Bluetooth, Z-Wave o ZigBee para establecer comunicaciones directas de dispositivo a dispositivo, como se muestra en la Figura 1.2.

Este modelo de comunicación se utiliza generalmente en sistemas que suelen utilizar pequeños paquetes de datos de información para comunicarse entre dispositivos con requisitos de velocidad de datos relativamente bajos. Un ejemplo de escenario con estas características son los sistemas de automatización del hogar.

Con este modelo la seguridad se ve simplificada principalmente porque se utiliza

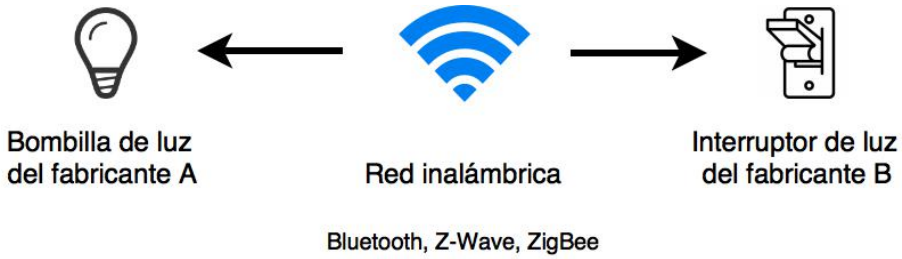


FIGURA 1.2: Modelo de comunicación dispositivo a dispositivo

una tecnología de radio de corto alcance, y porque hay una relación uno a uno entre los dispositivos.

Una pequeña desventaja que tiene este tipo de modelo es que, muchas veces, los dispositivos utilizan protocolos específicos por lo que diferentes dispositivos pueden no ser compatibles, obligando al usuario a seleccionar una familia de dispositivos que emplean un protocolo común. Por ejemplo, la familia de dispositivos que utiliza el protocolo Z-Wave no es nativamente compatible con la familia de dispositivos que utiliza ZigBee.

1.3.2. Comunicaciones de dispositivo a la nube (Device-to-Cloud)

En este modelo, el dispositivo se conecta directamente a un servicio de internet en la nube, utilizando generalmente los mecanismos de comunicación existentes, como las conexiones tradicionales de Ethernet o WiFi. La figura 1.3 es un ejemplo de esto.

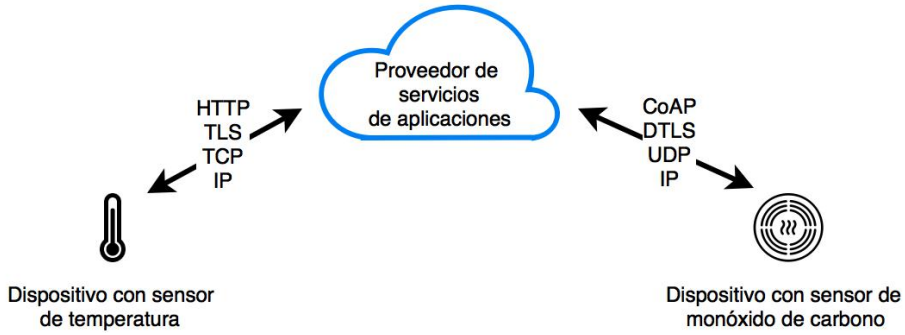


FIGURA 1.3: Modelo de comunicación dispositivo a la nube

La conectividad en la nube permite al usuario obtener acceso remoto a un dispositivo.

Un caso de uso de este modelo es el monitoreo remoto. A través de una cámara web se realiza el monitoreo y a su vez se guardan los datos en la nube para poder proporcionar acceso al usuario si éste no está presente en el mismo lugar donde se encuentra la cámara. Para este escenario es importante además contar con un buen ancho de banda como el proporcionado por WiFi o Ethernet.

Este modelo tiene como desventaja que al intentar integrar dispositivos de diferentes fabricantes pueden surgir problemas de interoperabilidad, ya que generalmente, el

dispositivo y el servicio en la nube son del mismo proveedor. Si entre el dispositivo y el servicio en la nube se utilizan protocolos de datos específicos, el usuario quedaría atado a un servicio en la nube específico, lo que impediría el uso de proveedores de servicios alternativos.

1.3.3. Comunicación dispositivo a gateway (Device-to-Gateway)

Este es uno de los modelos más usados en IoT y se da cuando los dispositivos se conectan a Internet a través de un dispositivo que hace las veces de gateway. Esto significa que hay un software de aplicación corriendo en un dispositivo de gateway local (como por ejemplo un smartphone), que actúa como intermediario entre el dispositivo y el servicio en la nube. Este gateway provee seguridad y otras funcionalidades como traducción de protocolos o de datos. Un diagrama de este modelo se puede ver la figura 1.4.

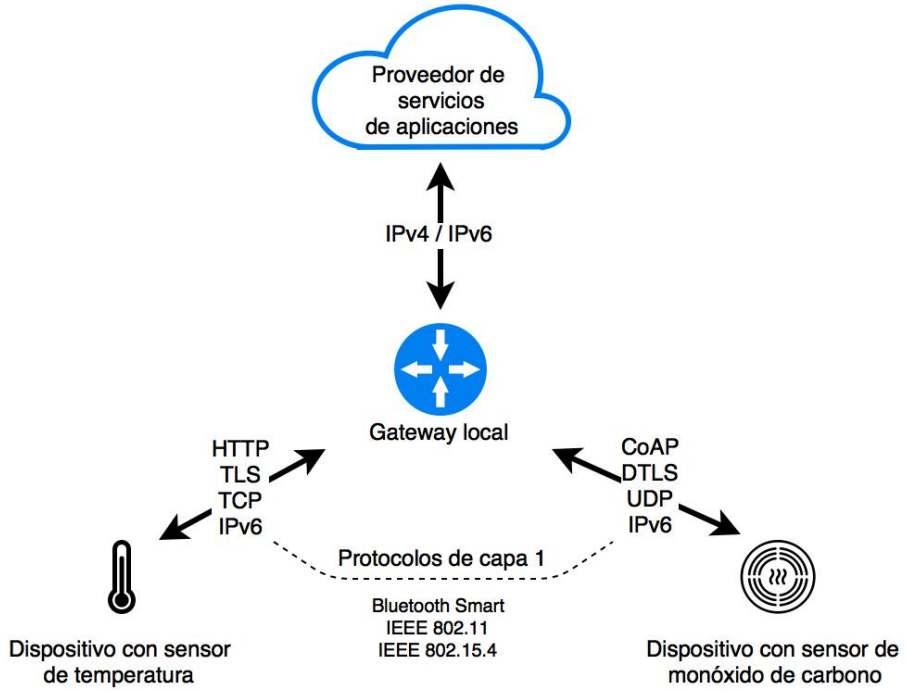


FIGURA 1.4: Modelo de comunicación dispositivo a gateway

La desventaja que presenta este modelo es la complejidad que representa programar un software de aplicación en el gateway, capaz de soportar varias tecnologías de comunicación a la vez y transformar sus diferentes formatos a un formato o protocolo entendible en Internet.

1.3.4. Comunicación de intercambio de datos a través del back-end (Back-End Data-Sharing)

Este modelo esencialmente extiende el modelo Device-to-Cloud. Los dispositivos de IoT y los datos de los sensores pueden ser accedidos por terceros autorizados. Esto significa que los datos generados por un dispositivo se pueden exportar y combinar con datos de otras fuentes y enviarlos a otros servicios para agregación y análisis. El objetivo principal de este modelo de arquitectura es conseguir interoperabilidad entre los diferentes servicios en la nube.

Un ejemplo de uso de este modelo es una aplicación de salud que recopila datos de varios dispositivos o aplicaciones como los pasos o distancia recorrida, las calorías quemadas, etc. y luego realiza un resumen y análisis de los datos obtenidos. Un diagrama de esta arquitectura se puede ver en la figura 1.5

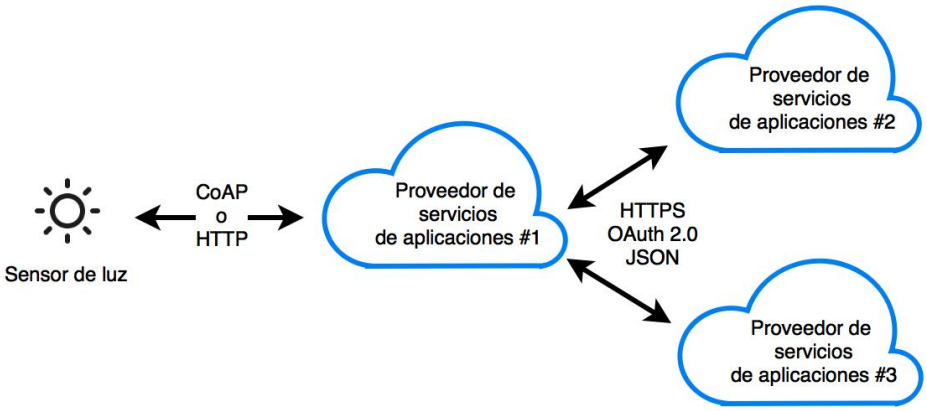


FIGURA 1.5: Modelo de comunicación de intercambio de datos a través del back-end

1.4. Protocolos

Existe una gran cantidad de opciones de conectividad y protocolos en torno a los sistemas de IoT. Es necesario elegir una buena combinación de dichos protocolos para lograr que los dispositivos interconectados se comuniquen de la manera más rápida y eficiente posible, sin requerir un uso elevado de los recursos.

Es posible hacer una categorización de los protocolos dependiendo de la capa en la que son usados. Se pueden distinguir tres capas principales: sesión o aplicación, red y enlace, además de las dos capas transversales: seguridad y gestión. A menudo es necesario que varios dispositivos se comuniquen entre sí y procesen información antes de que ésta llegue a Internet. Para lograr esta comunicación se utilizan los protocolos de la capa de enlace. Esta capa se encarga de conectar dos dispositivos, que podrían ser dos sensores o el sensor y el dispositivo que funciona como gateway. Por otro lado la capa de red cuenta con protocolos especializados para el encapsulamiento y enrutamiento de datos entre dispositivos. Por último, los protocolos de capa de sesión permiten la mensajería entre varios elementos del subsistema de comunicación. También se han desarrollado varios protocolos de seguridad y gestión, que aplican a todas las capas mencionadas anteriormente.

A continuación se mencionan algunas de las características principales de los protocolos más utilizados por las distintas capas, que se muestran en la figura 1.6.

- Capa de enlace
  - IEEE 802.11ah: Este protocolo (también conocido como HaLow) fue publicado en el 2017 como un ajuste del protocolo IEEE 802.11 (más conocido

FIGURA 1.6: Protocolos principales de IoT separados por capas

como WiFi) que busca, entre otras cosas, reducir el consumo de energía y mejorar el alcance y la robustez de la conectividad.

Frecuencia: bandas exentas de licencia por debajo de 1 GHz, excluyendo los espacios en blanco de TV.
Rango: hasta 1km
Ancho de banda: 150 Kbps a 346.666 Mbps

- Z-Wave: es un protocolo de baja potencia diseñado principalmente para la automatización del hogar y pequeños dominios comerciales. El protocolo es desarrollado por Sigma Designs Inc, e incluye una capa de seguridad. También está disponible la versión de código abierto del stack de protocolos Z-Wave, pero la misma no incluye la capa de seguridad. Opera sobre IEEE 802.15.4, y la topología de red que utiliza es de redes de mallas.

  Frecuencia: bandas por debajo de 1 GHz
  Rango: 30 a 100m
  Ancho de banda: 9.6, 40, 100 Kbps

- Bluetooth Low Energy: Bluetooth es un protocolo de comunicación de corto alcance que se utiliza tanto en capa física como en capa de enlace. El nuevo Bluetooth Low-Energy (BLE) fue diseñado para ofrecer un consumo de energía significativamente reducido en relación al clásico Bluetooth, ofreciendo un alcance similar. BLE no está realmente diseñado para la transferencia de archivos y es más adecuado para datos pequeños (chunks). Al estar ampliamente integrado en teléfonos inteligentes y muchos otros dispositivos presenta una importante ventaja frente a otras tecnologías de esta capa.

  El protocolo cuenta con dos modelos de comunicación: uno requiere que los dispositivos se emparejen para establecer comunicaciones bidireccionales, y el otro en cambio opera en modo “advertising” para la transmisión unidireccional de datos. El modo “advertising” permite que un dispositivo (por ejemplo un sensor) envíe periódicamente datos a otro dispositivo que esté escuchando, sin utilizar la energía necesaria para establecer y mantener un enlace completo bidireccional. Por otro lado, el emparejamiento de los dispositivos puede ser un inconveniente para soluciones IoT ya que generalmente esta configuración se debe llevar a cabo

en forma manual, y esto no escala si se quieren conectar un número importante de dispositivos.

Frecuencia: 2.4GHz (ISM)
Rango: 50-150m (Smart/BLE)
Ancho de banda: 1Mbps (Smart/BLE)

- ZigBee: Es una tecnología inalámbrica desarrollada como un estándar global abierto que satisface las necesidades de bajo costo y bajo consumo de energía. Opera sobre IEEE 802.15.4.

  Fue diseñado principalmente para domótica. Brinda compatibilidad con múltiples topologías de red, como redes punto a punto, punto a multipunto y redes de malla. Además soporta encriptación AES de 128 bits para conexiones seguras de datos, y cuenta con mecanismos de evitación de colisiones, reintentos y reconocimientos, lo cual lo convierte en un protocolo robusto.

  Frecuencia: 2.4 GHz, 915 MHz y 868 MHz.
  Rango: generalmente entre 50 y 100m, en las mejores condiciones puede llegar a 1000m
  Ancho de banda: 20, 40, 250 kbits/s

- Capa de red

  - 6LoWPAN: Un problema en las aplicaciones IoT es que las direcciones IPv6 son demasiado largas y no pueden encajar en la mayoría de las tramas de enlace de datos IoT, que son relativamente mucho más pequeñas. Por lo tanto, IETF (Internet Engineering Task Force) está desarrollando un conjunto de estándares para encapsular datagramas IPv6 en diferentes tramas de capa de enlace de datos para su uso en aplicaciones IoT.

    Uno de los protocolos más utilizados con este fin es el protocolo 6LoWPAN (IPv6 sobre Low Power Wireless Personal Area Network). El mismo encapsula encabezados largos IPv6 en paquetes pequeños IEEE802.15.4, que no pueden exceder los 128 bytes. De esta forma proporciona conectividad con otras redes IP e interoperabilidad. 6LoWPAN es un protocolo robusto y escalable. Soporta enrutamiento en malla. Se puede utilizar sobre múltiples plataformas de comunicación como por ejemplo Ethernet, WiFi y 802.15.4, entre otras.

  - RPL: es un protocolo de enrutamiento IPv6 de vector de distancia diseñado para redes de baja potencia y pérdidas (Low Power and Lossy Networks). Es capaz de construir rápidamente rutas de red, distribuir conocimientos de enrutamiento entre nodos y adaptar la topología de una manera muy eficiente. Soporta tres tipos de tráfico de datos: punto-a-punto, punto-a-multipunto y multipunto-a-punto, aunque está optimizado para el caso multipunto-a-punto.

  - Thread: Es un protocolo de red basado en IPv6, de documentación cerrada, pensado originalmente para dispositivos de domótica. Thread usa

6LoWPAN y encriptación AES. Provee comunicaciones dispositivo a dispositivo y dispositivo a la nube. Soporta redes de malla, lo que permite que los dispositivos intercambien mensajes sin tener un único punto de falla.

- Capa de aplicación/sesión
  - MQTT: El protocolo MQTT (Message Queue Telemetry Transport) fue introducido por IBM en 1999 y estandarizado por OASIS en 2013. Está diseñado para proporcionar conectividad integrada entre aplicaciones y middleware en un lado y redes y comunicaciones en el otro lado. Es liviano, abierto, simple y está diseñado para ser fácil de implementar. Sigue una arquitectura publish/subscriber.

    MQTT está diseñado para ser útil para muchos dispositivos pequeños y relativamente tontos que envían mensajes pequeños en redes de bajo ancho de banda. MQTT corre sobre los protocolos TCP/IP, o sobre otros protocolos de red que proporcionan conexiones bidireccionales ordenadas, sin pérdidas.

    Brinda tres tipos de entregas de mensajes:
    - "por lo menos uno": los que el mensaje se entregan al menos una vez, por lo que pueden llegar repetidos.
    - "a lo sumo uno": los mensajes se entregan de acuerdo con los mejores esfuerzos del entorno operativo, por lo que se pueden producir pérdidas de mensajes.
    - "exactamente uno": se asegura que el mensaje llegará exactamente una vez.

  - AMQP (Advanced Message Queuing Protocol): Es un protocolo abierto, orientado a mensajes, que se destaca por su interoperabilidad y fiabilidad. Corre sobre TCP/IP, y se basa en una arquitectura publish/subscriber. Al igual que MQTT está destinado a ser utilizado para permitir que los programas envíen y reciban mensajes de forma asíncrona independientemente de su elección de hardware, sistema operativo o lenguaje de programación.

    AMQP está diseñado para soportar la mayor cantidad de escenarios de mensajería conocidos, incluyendo colas de mensajes clásicos, roundrobin, store-and-forward y combinaciones de los mismos.

  - CoAP: De la misma forma que HTTP, CoAP se basa en el exitoso modelo REST, en el que los servidores hacen que los recursos estén disponibles bajo una URL y los clientes acceden a estos recursos utilizando métodos como GET, PUT, POST y DELETE. A su vez, se define el mapeo de CoAP con HTTP, permitiendo que se construyan proxies que proporcionen acceso a los recursos de CoAP vía HTTP de una manera uniforme.

    CoAP puede cargar diferentes tipos de payloads, y puede identificar qué tipo de payload se está utilizando. Se integra con XML, JSON, CBOR, o cualquier otro formato de datos.

CoAP está diseñado para utilizar recursos mínimos, tanto en el dispositivo como en la red. En lugar de utilizar una pila de transporte compleja, corre sobre UDP/IP. Los mensajes enviados son pequeños, lo que genera poca o ninguna fragmentación en la capa de enlace.

1.5. Tamaño de mensajes y payload

Generalmente es posible utilizar diferentes formatos a la hora de enviar datos entre dispositivos. Es importante elegir los formatos adecuados, de forma que los datos sean fáciles de utilizar pero que también sean lo más óptimos posible en cuanto al tamaño del payload. Por ejemplo si un dispositivo detecta una temperatura de 20.635 ° C podría mandar este dato en un JSON de la siguiente forma:

{ “Count”: 1234, “Temperature”: 20.635 }

Este JSON ocupa 40 bytes. Para reducir su tamaño se pueden eliminar espacios, comprimir nombres, etc., logrando el siguiente mensaje que ocupa ahora 11 bytes:

{“t”:20.63}

Pero este mensaje se podría reducir más aún enviando la misma información pero en formato texto simplificado. En dicho caso el payload sería únicamente “20.63” y ocuparía 5 bytes. Si se quisiera los datos se podrían comprimir aún más si se mandara cómo binario; para este ejemplo ocuparía tan sólo 2 bytes y sería el entero con signo de 16 bits 0x080F.

No obstante, las optimizaciones antes descritas, son aplicadas al formato que se elige para enviar los mensajes en el payload. Luego de elegido el formato, se tiene que optar por el tipo de protocolo para enviar el mensaje teniendo en cuenta que este también influye en el tamaño del mensaje.

Por otro lado existen herramientas para comprimir el payload del mensaje que son soportadas por ciertas plataformas. En la referencia [3] muestra cómo utilizar Gzip[4] para comprimir y descomprimir los mensajes que son enviados y recibidos por Azure.

1.6. Paradigma Publish/Subscribe

En el paradigma publish/subscribe [5] existen dos actores principales: los suscriptores y los proveedores. Los suscriptores tienen la funcionalidad de suscribirse a tópicos o características de un evento para luego poder ser notificado ante cualquier publicación que coincida con los intereses a los que fueron suscriptos.

Un sistema básico de interacción publish/subscribe es el que se muestra en

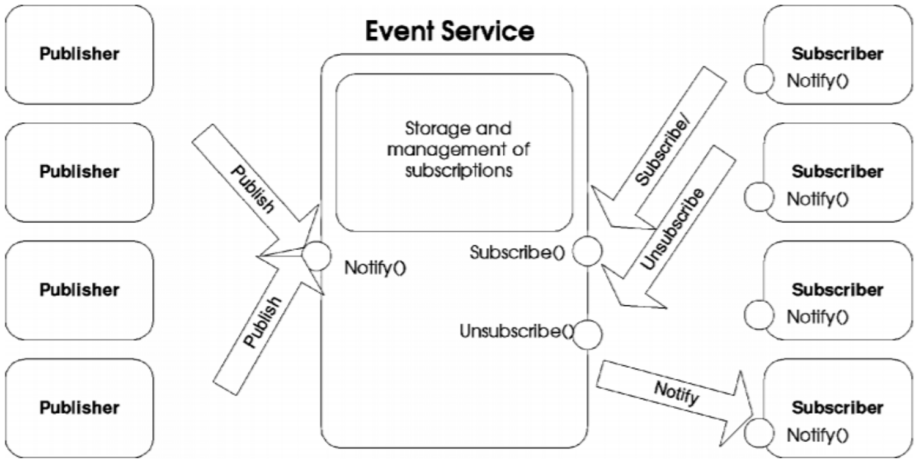


FIGURA 1.7: Diagrama del paradigma Publish/Subscribe. Fuente: The Many Faces of Publish/Subscribe [5]

la figura 1.7, en donde un servicio de notificación de eventos proporciona almacenamiento y gestión de las suscripciones así como una eficiente entrega de los eventos.

La fortaleza de este paradigma basado en tópicos se halla en que brinda un desacople en tiempo, espacio y sincronización entre publicadores y suscriptores. La eliminación de las dependencias entre los extremos aumenta la escalabilidad además de reducir la coordinación entre las entidades.

El desacople que proporciona este esquema de interacción puede ser descompuesto en tres dimensiones diferentes:

- Desacople del espacio: Los publicadores hacen las publicaciones en el servicio de evento, el cual se encarga de distribuir la información a los suscriptores. Los suscriptores reciben esta información, pero tanto los suscriptores como los publicadores no conocen cuántos hay de uno u otro ni la identidad de estos.
- Desacople del tiempo: Tanto los publicadores como los suscriptores no tienen por qué estar activos al mismo tiempo. Un publicador podría publicar mientras que los suscriptores no están activos; estos recibirán las publicaciones cuando se activen y perfectamente el publicador podría estar desactivado cuando esto suceda.
- Desacople de sincronización: Los suscriptores obtienen las publicaciones de manera asíncrona lo que les permite realizar cualquier otra operación de manera concurrente, mientras que los publicadores no se bloquean cuando están realizando publicaciones.

1.6.1. Tipos de Publish/Subscribe

Los suscriptores usualmente están interesados en algunos eventos en especial o en alguna caracteristicas o propiedad especial del evento pero no en todos los eventos. En base a estos requerimientos se pueden diferenciar tres esquemas que existen de Publish/Subscribe:

- Publish/Subscribe basado en tópico: este tipo de esquema extiende la noción de canales, los cuales son usados para agrupar la comunicación entre distintos usuarios, dando la posibilidad de caracterizar y clasificar el contenido del evento. Los usuarios pueden suscribirse y publicar en eventos basados en tópicos específicos, los cuales son identificados por una clave. La información del tópico viaja como parte del mensaje.

  Otra particularidad de ese modelo es que organiza los tópicos jerárquicamente por lo que permite hacer anidaciones de tópicos, por esto mismo cuando un suscriptor realiza una suscripción a un tópico automáticamente se suscribe a los sub-tópicos que lo conforman.

  Un tópico es un string que puede tener varios niveles de jerarquía, separados por "/". Por ejemplo el tópico *"case/living/temperatura"* representa la temperatura del living de la casa. Un cliente se podría suscribir a un tópico exacto o a un comodín como por ejemplo *"casa/+/temperatura"* y de esta forma recibiría todos los mensajes asociados a la temperatura de cualquier habitación de la casa. El signo de más (+) es un comodín de nivel único y sólo permite valores arbitrarios para una jerarquía. También existe un comodín multinivel (#) que permite suscribirse a más de un nivel, por ejemplo con el tópico *"casa/#"* el cliente se suscribe a todos los tópicos que comiencen por *"casa"*.
- Publish/Subscribe basado en contenido: la suscripción está basada en el contenido de los eventos, es decir en valores de los atributos o metadatos asociados a dichos eventos. Para suscribirse a dichos eventos se utilizan filtros formados por operadores de comparación básicos (=, >, <) y/o operadores lógicos (AND, OR, XOR, etc)
- Publish/Subscribe basado en tipo: los eventos se clasifican utilizando tipos de datos y los suscriptores se suscriben a los tipos de datos deseados. La ventaja que presentan respecto al esquema basado en tópicos es que no se necesita especificar toda la ruta del árbol jerárquico al momento de realizar una suscripción, y además este esquema permite realizar chequeos de tipos entre las interfaces en tiempo de compilación.

2 Plataformas

Actualmente el Internet de las cosas representa un mundo complejo e inmaduro, por lo que el uso de plataformas IoT que faciliten y guíen el desarrollo de soluciones IoT es imprescindible.

De forma general y simplificada se puede decir que una plataforma IoT es un software cuya principal función es conectar los diferentes dispositivos, y procesar, analizar y almacenar los datos que los mismos generan. En la figura 2.1 se puede ver que lugar ocupa una plataforma IoT en la arquitectura general de una solución IoT [6].

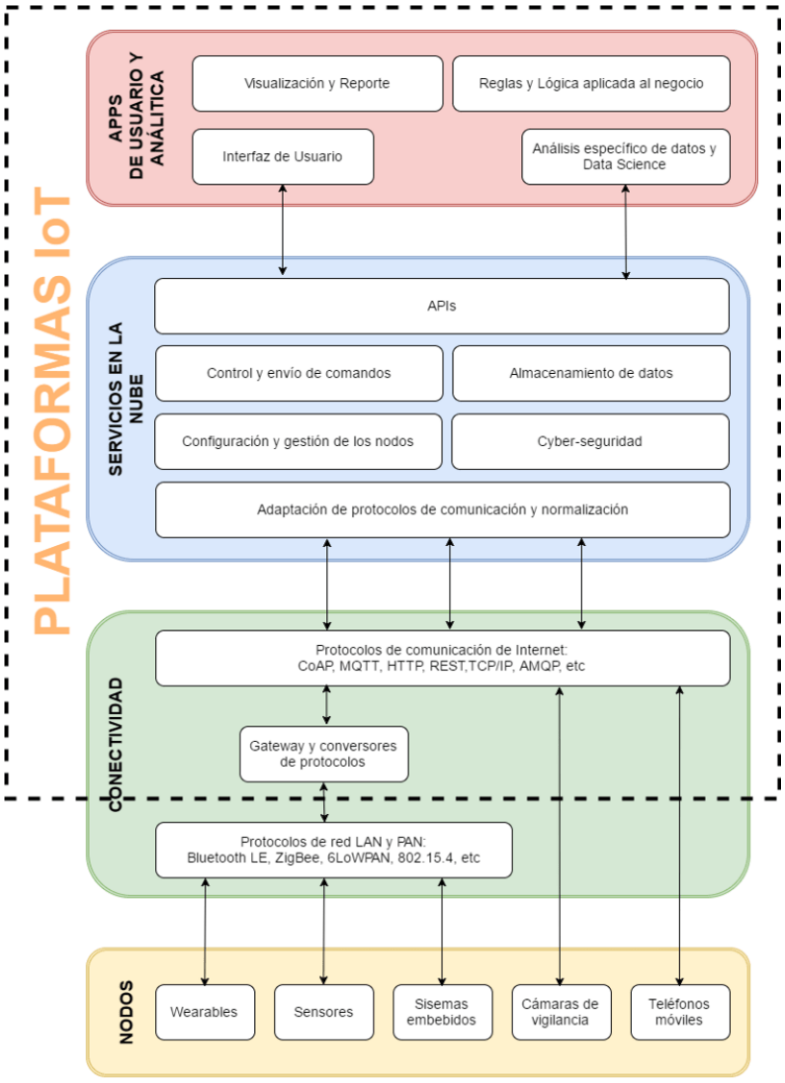


FIGURA 2.1: Ubicación de las plataformas IoT en la arquitectura de una solución IoT. Fuente: Para qué sirven las Plataformas IoT y cuáles son las más populares actualmente [6]

Las capacidades de una plataforma IoT generalmente incluyen:

- Personalización y construcción de aplicaciones (incluyendo SDK, IDE, App-Server, etc)
- Procesamiento de eventos: flujo de eventos y agregación de datos, streaming analytics, almacenamiento y gestión de la información. Toma de decisiones: motores de reglas, orquestación de flujos de trabajo y proceso de negocio (BPM)
- Análisis: Análisis y visualización de datos IOT (incluyendo cuadros de mando)
- Ciberseguridad: autenticación, encriptación, gestión de certificados, etc.
- Comunicaciones con dispositivos IoT (capa física como WIFI y capa de datos, como MQTT o HTTP)
- Interfaces de usuario, tanto para usuarios finales como para desarrolladores

En las siguientes secciones se detallarán algunas de las plataformas existentes hoy en día: Microsoft Azure, IBM Watson, Amazon IoT, Kaa y Fiware.

2.1. Azure

Azure IoT Hub es un servicio que permite comunicaciones bidireccionales confiables y seguras entre millones de dispositivos IoT y la plataforma [7]. Algunas de las características de Azure son:

- Provee múltiples opciones de comunicación dispositivo a dispositivo y dispositivo a la nube. Estas opciones incluyen mensajería unidireccional, transferencia de archivos y métodos de solicitud y respuesta.
- Provee enrutamiento de mensaje declarativo a otros servicios de Azure.
- Permite comunicaciones seguras y control de acceso usando claves de seguridad por dispositivo o certificados X.509.
- Proporciona una amplia supervisión de la conectividad de los dispositivos y los eventos de gestión de identidad de los mismos.
- Incluye librerías para las plataformas y lenguajes más populares.

2.1.1. Arquitectura

La figura 2.2 muestra, en alto nivel, como es la arquitectura de una solución IoT utilizando Azure [8], la cual cuenta con tres áreas principales: la conectividad de los dispositivos, el análisis y procesamiento de datos, y por último, el área de presentación. Básicamente lo que muestra el diagrama es como los datos de los dispositivos ingresan a la nube mediante el gateway; este deja a disposición los datos para que sean analizados y procesados para que luego puedan ser utilizados por la capa de presentación.

Se describen a continuación algunos de los componentes de esta arquitectura, exceptuando los componentes relacionados al almacenamiento y análisis de datos que se describirán en una sección aparte.

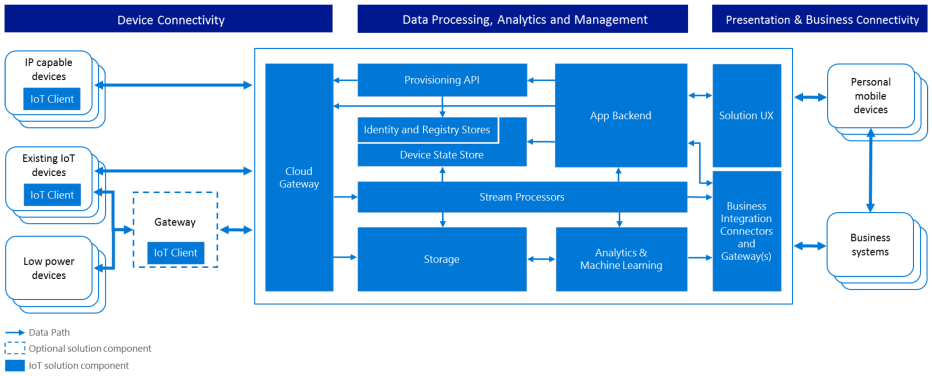


FIGURA 2.2: Arquitectura de Azure IoT. Fuente: Azure IoT Reference Architecture [8]

2.1.1.1. Cloud gateway

Un cloud gateway es la parte base de la arquitectura de la nube que permite la comunicación remota desde y hacia dispositivos o gateway locales. En general, gestiona todos los aspectos de la comunicación, incluida la gestión de conexión a nivel de protocolo de transporte, la seguridad del canal de comunicación, la autorización del dispositivo y la autorización hacia el sistema. También, recopila los datos utilizados para la facturación del servicio, diagnósticos y otras tareas de supervisión.

Algunos servicios que brinda Azure para la implementación de este componente son:

- Azure IoT Hub: Ocupa el lugar de la principal tecnología cloud gateway en Azure, el cual brinda un servicio de gran escala que permite una comunicación bidireccional segura desde diversos dispositivos, permite la conexión de grandes de cantidades de dispositivos y soporta el procesamiento de grandes volúmenes de datos hacia el back-end de la nube, así como comanda y controla el tráfico hacia los dispositivos. Azure IoT Hub provee soporte para los protocolos de comunicación AMQT 1.0, MQTT 3.1.1 y HTTP 1.1 sobre TLS.
- Azure Event Hubs: Es un servicio solo de procesamiento de datos a gran escala que recopila datos de telemetría de fuentes concurrentes a tasas de rendimiento muy altas. Puede ser usado en conjunto con IoT Hub, para flujos de telemetría secundaria (es decir, telemetría sin dispositivo), o para recopilar datos de otras fuentes del sistema (como datos meteorológicos o social stream). Event Hubs no ofrece la funcionalidad de identificar dispositivos ni capacidades de comando y control, por lo que debería usarse para flujos de datos adicionales pero no para la conexión principal de los dispositivos. Provee soporte para AMQP 1.0 y HTTPS.

Existe también la posibilidad de crear un cloud gateway personalizado que permita la traducción del protocolo y/o alguna forma de procesamiento personalizado antes de alcanzar la conexión final con la nube, como por ejemplo la transformación de mensajes. No obstante esto debe ser evaluado cuidadosamente ya que puede afectar la performance. Un framework que provee Azure para realizar esto es Azure IoT protocol gateway. Este framework de código abierto se utiliza para la traducción de protocolos. Facilita la comunicación bidireccional entre dispositivos y Azure IoT

Hub. Además permite procesamiento adicional como por ejemplo una autenticación personalizada, transformación de mensajes, comprimir/descomprimir y encriptación/desencriptación.

Por otra parte, Azure brinda un componente que puede ser usado en los dispositivos del cliente para simplificar la conectividad con Azure IoT Hub: Azure IoT device SDK. Las librerías que contiene permiten la conectividad de un gran número de dispositivos y gateway locales con Azure.

2.1.1.2. Solución UX (User eXperience)

Suele incluir un sitio web, pero también puede incluir servicios web y API con una interfaz gráfica de usuario en forma de una aplicación móvil o de escritorio. La seguridad es clave y la solución UX que proporciona control sobre el sistema y los dispositivos debe ser protegido adecuadamente con control de acceso diferenciado por los roles de usuarios y dependiendo de la autorización que tiene ese usuario.

2.1.1.3. App back-end

Implementa la lógica empresarial requerida de la solución. Implementa los modelos y abstracciones de objetos apropiados para dispositivos, grupos de dispositivos y relaciones entre dispositivos, reglas y acciones. También administra el acceso y las asociaciones entre dispositivos y usuarios.

Los servicios que brinda Azure para implementar el back-end son:

- Azure App Service: Es una plataforma con potentes capacidades para crear aplicaciones web y móviles para muchas plataformas y dispositivos móviles. Los servicios de Web Apps y Mobile Apps permite a los desarrolladores crear aplicaciones web y móviles utilizando distintos lenguajes como .NET, Java, NodeJS, PHP o Python. Además, Azure API Apps permite una fácil exposición y administración de APIs, a las que pueden acceder clientes móviles o web.
- Azure Notification Hubs: Permite el envío de notificaciones push a dispositivos móviles personales. Es compatible con las plataformas de iOS, Android, Windows y Kindle. Con una única llamada de la API, una notificación puede ser enviada a un usuario en particular o a un grupo de usuarios.
- Azure Active Directory: Las aplicaciones web y móviles se pueden integrar con Azure Active Directory (AAD) para la autenticación y el control de autorización. Las aplicaciones dependerán de la gestión de identidades de usuario en AAD y pueden proporcionar fácilmente control de acceso basado en roles para las funcionalidades de la aplicación. Los permisos y el control de acceso basado en roles se pueden gestionar como parte de una matriz de asociación entre identidades del dispositivo (mantenidas en el almacén de identidad del dispositivos) e identidades de usuarios administradas por el AAD. El diseño específico de esta matriz, la granularidad de los permisos y el nivel de control dependerán de los requisitos específicos de la solución.

2.1.2. Almacenamiento

2.1.2.1. Almacenamiento de identidad del dispositivo

Almacena y permite la validación criptográfica para la autentificación del dispositivo del cliente. No proporciona ningún recurso de indización o búsqueda más allá de la búsqueda directa por el identificador del dispositivo. El almacenamiento de la identidad y el registro del dispositivo están separados por razones de seguridad, las búsquedas en el registro no deben permitir la divulgación de material criptográfico.

El aprovisionamiento de dispositivos utiliza el almacén de identidad para crear identidades para nuevos dispositivos en el ámbito del sistema o para eliminar dispositivos del sistema. Los dispositivos también se pueden activar o desactivar. Cuando están inhabilitados, no tienen acceso al sistema, pero todas las reglas de acceso, claves y metadatos se mantienen.

Los cambios en el almacenamiento de identidad del dispositivo deben hacerse a través de la Provisioning API.

2.1.2.2. Almacenamiento del registro del dispositivos

Es una base de datos "índice" existente junto al almacén de identidad, que contiene datos de descubrimiento y referencia relacionados con dispositivos registrados. Mientras que el almacén de identidad sólo contiene atributos controlados por el sistema y material criptográfico que está inmediatamente disponible, el registro almacenará otra información de metadatos relacionada con el dispositivo.

El registro de dispositivos es un índice, mientras que el almacén de identidad representa la lista autorizada de identidades de dispositivo. El registro en el almacén de identidad determina si un dispositivo está o no activo en el sistema. Por razones de seguridad, el registro del dispositivo no debe almacenar ninguna clave u otra información criptográfica relacionada con el dispositivo.

La distinción entre los metadatos que describen el propio dispositivo y los datos operativos que reflejan el estado del dispositivo es importante porque afecta directamente la forma en que la información del registro se puede utilizar, almacenar en caché y distribuirse por todo el sistema. Los metadatos suelen ser datos de pocos cambios, mientras que se espera que los datos operativos cambien rápidamente.

Azure IoT Hub incluye un almacén de identidad de dispositivo integrado que es la autoridad para dispositivos registrados y proporciona credenciales de seguridad por dispositivo.

En el caso de que una solución específica requiere una implementación personalizada del almacén de identidad, este será un componente separado que almacenará todas las claves de seguridad necesarias para el dispositivo y tendrá potencialmente un estado "habilitado/inhabilitado". Hay que tener en cuenta que una implementación personalizada del almacén de identidad debe protegerse de forma adecuada, ya que almacena información de credenciales.

Si no se utiliza Azure IoT Hub, las implementaciones externas pueden realizarse

sobre Azure DocumentDB, Azure Tables, Azure SQL Database o soluciones de terceros.

- Azure DocumentDB: En Azure DocumentDB, cada dispositivo es representado por un documento. El identificador de dispositivo a nivel de sistema corresponde directamente al "id" del documento. Todas las demás propiedades se encuentran junto al "id" en el documento.
- Azure Tables: En Azure Tables, el almacén de identidad es mapeado a una tabla. Cada dispositivo es representado por una fila.
- SQL Database: El almacén de identidad también se asigna a una tabla y cada dispositivo está representado por una fila. El identificador de dispositivo a nivel de sistema se mantiene en una columna de clave principal de índice agrupado. Todas las demás propiedades se almacenan en columnas. Los datos complejos o los datos que requieren extensibilidad se pueden almacenar como JSON, si es necesario.
- Soluciones de terceros: También se pueden utilizar soluciones de terceros disponibles a través de Azure Marketplace o desplegadas directamente en nodos de procesamiento de Azure. Por ejemplo, en Cassandra, cada dispositivo puede representarse por una fila en una familia de columnas. El almacén será particionado e indexado para un acceso rápido según sea necesario.

2.1.2.3. Registro del dispositivo

La Provisioning API es la interfaz externa común para saber cómo se realizan los cambios en el almacén de identidad del dispositivo y en el registro del dispositivo.

Como alternativa a las técnicas tradicionales de programación, Azure API Apps puede ser usada para la implementación de la Provisioning API. API Apps proporciona una plataforma para crear, alojar y distribuir APIs en la nube. La interfaz de Provisioning debe implementarse como la API primaria sobre los almacenes de identidad y de registro y, opcionalmente, otros componentes de la solución interna si es necesario.

2.1.2.4. Almacenamiento del estado del dispositivo

Los datos operativos relacionados con los dispositivos residen en el almacén de estado del dispositivo. El almacén de estado del dispositivo está separado del almacén de registro del dispositivo. Aunque es a menudo deseable almacenar la información recibida desde el dispositivo, el almacén de estado es un elemento arquitectónico opcional. Una implementación sencilla puede incluir escribir el flujo de datos del dispositivo de una forma predeterminada, pero el almacén mismo y el flujo de datos a él pueden eliminarse o cambiarse.

La forma predeterminada para el almacén de estado de dispositivo es que conserva dos tipos de información. Una parte es el flujo completo de eventos entrantes del dispositivo. El otro es un registro de "últimos valores conocidos" que es una proyección de los últimos valores observados capturados desde el dispositivo.

Las opciones que se tienen en Azure para implementar esto son:

- Para una implementación mínima, los datos brutos entrantes o las proyecciones de mensajes se pueden almacenar en Azure Blobs o Tablas, lo que ayuda a optimizar el costo. Los mensajes entrantes se pueden reenviar "tal cual" o pueden modificarse a otra forma de salida. Los datos se pueden consumir directamente desde el almacén o pueden realizarse transformaciones adicionales utilizando Azure Data Factory 14, lo que permitirá la transformación secundaria, las agregaciones y el movimiento de datos según sea necesario para la solución.
- Además de los datos sin procesar, los últimos valores conocidos para un dispositivo se almacenan por separado como un registro que se sobrescribe constantemente. Este registro puede ser un objeto Blob o de tabla de Azure independiente, o puede agregarse como un documento independiente al registro del dispositivo para las capacidades de consulta rápida. Los valores agregados o calculados se pueden agregar a este o un registro separado también.

2.1.3. Análisis y procesamiento de datos

Azure soporta múltiples flujos de datos al mismo tiempo. Además, como la cloud gateway soporta múltiples clientes, los mismos datos pueden ser consumidos por diferentes procesadores de flujo para diferentes propósitos. Por ejemplo, un procesador de flujo puede actuar sólo para tipos especiales de eventos, mientras que otro podría realizar el procesamiento de eventos complejos en paralelo. Dichos procesadores pueden determinar la ruta de datos o realizar tareas complejas de procesamientos de eventos tales como agregación de datos, enriquecimiento de datos mediante correlación con datos de referencia, así como tareas analíticas tales como detección de umbrales o anomalías y generación de alertas.

También se pueden recolectar grandes cantidades de datos de telemetría sin procesar y cargarlos utilizando operaciones especiales para cargar grandes cantidades de datos. Esta es una alternativa viable para los datos de telemetría, cuando no hay necesidad de procesar registros individuales e intervenir inmediatamente sobre los datos. Puede cargarse y utilizarse como archivo o para análisis por lotes más adelante.

Además de procesar los datos del dispositivo, un procesador de flujo puede mantener la actualización de los "últimos valores conocidos" para los dispositivos. Los valores específicos agregados o precalculados también se pueden almacenar en el almacén de estado del dispositivo para facilitar el acceso por el back-end de la app si lo requiere la lógica de la solución. Los eventos entrantes también pueden ser enviados a módulos especializados para análisis avanzados y aprendizaje automático.

Los dispositivos suelen agregar el tráfico en múltiples flujos de datos utilizando discriminadores en el encabezado del protocolo utilizado por la aplicación (por ejemplo, propiedades de mensajes tales como "stream-id" o "subject") que permitirán el enrutamiento y procesamiento por el procesador de flujo apropiado.

Azure Stream Analytics, Apache Storm implementado en Azure HDInsight, o procesadores de eventos personalizados pueden facilitar el flujo de datos desde el punto de ingreso en Azure IoT Hub o en Event Hubs. Estos procesadores, también se pueden utilizar como un motor de reglas, en el que se pueden definir, configurar y

activar o desactivar reglas para umbrales y límites.

Además de Azure Stream Analytics y Apache Storm, es posible construir una arquitectura Lambda en Azure utilizando frameworks como Apache Kafka, Apache Cassandra y Apache Spark para el procesamiento. El flujo entrante de datos (por ejemplo, telemetría del dispositivo) se encaminará a las diversas etapas de procesamiento a través de Kafka y se almacenará en Cassandra para poder ejecutar trabajos de lotes (trabajos Spark) sobre estos datos almacenados.

2.1.4. Modelo de negocio

Microsoft Azure ofrece cuatro posibilidades para poder usar IoT Hub [9]:

- Gratis, con una suscripción (requiere tarjeta de crédito o débito internacional) se puede probar de manera limitada el servicio de IoT Hub. La limitante está dada por la cantidad de mensajes que pueden ser transmitidos en el día; esta cantidad es de hasta 8.000 mensajes.
- S1, con este contrato puede transmitir hasta 400.000 mensajes por día entre todos los dispositivos conectados. Está pensado para soluciones de IoT que generan pequeñas cantidades de datos.
- S2, diseñado para las soluciones que generan grandes cantidades de datos, permitiendo que se envíen hasta 6 millones de mensajes por día.
- S3, similar al contrato anterior pero con esta cuenta puede enviar hasta 300 millones de mensajes por día entre todos los dispositivos conectados.

Además brinda planes de soportes flexibles desde 29 dólares por mes y garantiza una conectividad del 99.9 %.

En la tabla 2.1 se pueden ver los distintos planes con sus correspondientes precios en dólares.

Tipo de edición	Precio por unidad (al mes)	Número total de mensajes por día y por unidad	Tamaño del medidor de mensajes
Gratis	Gratis	8.000	0,5 KB
S1	USD 50	400.000	4 KB
S2	USD 500	6.000.000	4 KB
S3	USD 5000	30.000.000	4 KB

TABLA 2.1: Precios de Azure IoT

Cabe mencionar que Microsoft Azure cuentas con más servicios que pueden ser usados a la hora de diseñar una solución IoT, como por ejemplo, el uso de notificaciones, machine learning, análisis de datos y demás servicios que pueden ser útiles para el desarrollador, por lo que tiene a disposición una calculadora para poder hacer una aproximación más exacta de los gastos para la solución que se quiere desarrollar, teniendo en cuenta los servicios contratados.

2.2. IBM Watson

La plataforma IBM Watson [10] es la encargada de gestionar los dispositivos para una solución IoT, permitiendo registrar dispositivos para luego poder almacenar

y acceder a los datos de los mismos, proporcionando siempre una conexión segura desde el dispositivo a la plataforma y desde la plataforma al dispositivo, utilizando los protocolos MQTT y TLS.

Esta plataforma es un servicio disponible dentro de IBM Bluemix. Gracias a esto se puede integrar con otros servicios, como por ejemplo servicios para el análisis y almacenamiento de datos, desarrollo de app, etc, alojados en Bluemix.

2.2.1. Arquitectura

La arquitectura de la plataforma Watson IoT está diseñada para evitar el caso de que un dispositivo tome el lugar de otro, lo que mantiene la integridad de los datos del dispositivo. Los dispositivos se conectan a la plataforma utilizando una combinación de un identificador de cliente y un token de autenticación, que sólo el usuario conoce. Después de que se registran los dispositivos o se generan las claves API, el token de autenticación se guarda y se guarda en un hash para mantener la seguridad de las credenciales.

Se cuenta con un componente que permite definir políticas de seguridad en la plataforma. Esto permite por ejemplo, tener una política para garantizar que todos los puntos de autenticación entre la plataforma y los dispositivos se tienen que autenticar con tokens y se utiliza TLS para el transporte de los datos, y a todos los dispositivos que intenten conectarse a la plataforma y no tienen certificados o tokens válidos, se les niega el acceso.

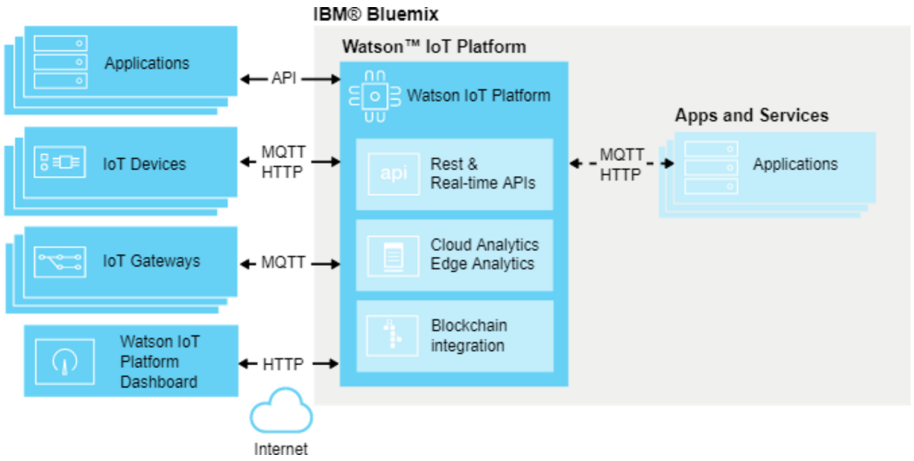


FIGURA 2.3: Arquitectura de una solución IoT de IBM. Fuente: Watson IoT [10]

2.2.2. Almacenamiento

Watson brinda un servicio que da la posibilidad de recuperar el último dato enviado por algún dispositivo. Esto funciona si el dispositivo está en línea o no, lo que le permite recuperar el estado del dispositivo independientemente de la ubicación física del dispositivo o el estado de uso. Los últimos datos de sucesos de un dispositivo se pueden recuperar para cualquier suceso específico que se ha producido hasta hace un máximo de 365 días.

El almacenamiento de los datos es esencial en cualquier solución IoT, ya que esto permite hacer un seguimiento de los cambios durante un periodo de tiempo determinado y utilizar luego estos datos con potentes herramientas de análisis. Para esto Bluemix tiene una variedad de servicios que simplifican esta tarea, por ejemplo IBM Cloudant NoSQL DB, el cual ofrece acceso a una capa de datos JSON de NoSQL completamente gestionadas que siempre están activos.

Otro servicio brindado por Bluemix en relación al almacenamiento es IBM Message Hub el cual proporciona un bus de mensajes escalable de alto rendimiento para el almacenamiento de datos históricos. Message Hub se crea en Apache Kafka, que es un sistema de mensajería de alto rendimiento de código abierto que proporciona una plataforma de baja latencia para manejar canales de información de datos en tiempo real.

2.2.3. Análisis de datos

Watson permite visualizar y mostrar los datos de los dispositivos en tiempo real mediante un panel de control. En el panel de control se muestran los datos del dispositivo en tiempo real, lo que permite realizar un seguimiento constante de los mismos y proporcionar una visión general y rápida de los datos, así como también una mejor comprensión de los mismos.

También se pueden hacer análisis más complejos, por ejemplo, se pueden especificar reglas que se basan en los datos que están brindando los dispositivos en tiempo real y que pueden activar alertas o acciones opcionales cuando se cumplen. Un ejemplo de esto puede ser crear una regla para asegurarse de que cuando se cae el dispositivo o cuando la temperatura del dispositivo se dispara, se envía una alerta al usuario y se envía un correo electrónico al administrador.

Por otro lado, Watson brinda los servicios necesarios para poder implementar todo este análisis de datos basado en reglas por fuera de la nube, osea, en los dispositivos gateway de la solución IoT. Al hacer todo el procesamiento, o por lo menos parte de él, en los dispositivos gateway, se reduce de forma notoria la cantidad de tráfico de datos del dispositivo a la nube.

2.2.3.1. IBM Streaming Analytics

IBM Streaming Analytics es una plataforma analítica avanzada que se puede utilizar para recopilar, analizar y correlacionar información en tiempo real desde distintos tipos de fuentes de datos, basada en InfoSphere Stream. Streaming Analytics permite ejecutar los trabajos creados utilizando InfoSphere Stream, y cuenta con un dashboard en el que se pueden visualizar los flujos de datos en tiempo real.

InfoSphere Stream Studio es una plataforma desarrollada para facilitar el análisis de datos de streaming. En vez de recopilar los datos, guardarlos y luego analizarlos, InfoSphere Streams aplica el análisis a los datos en movimiento. El análisis puede disparar eventos o alertas, de manera que se pueda reaccionar en tiempo real.

Podria decirse que InfoSphere Streams es una solución de Procesamiento Complejo de Eventos (CEP), sin embargo, InfoSphere Streams está diseñado para ser más escalable y soportar flujos de datos a un mayor ritmo que las soluciones tradicionales

de CEP.

InfoSphere Streams proporciona un lenguaje de programación (SPL) y un IDE, basado en Eclipse [11]. Una aplicación de Streams es un grafo dirigido, donde cada nodo consiste en un bloque como se muestra en la figura 2.4.

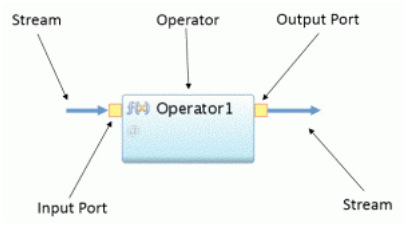


FIGURA 2.4: Componentes de un bloque básico de Infosphere Stream.
Fuente: Streams Quick Start Guide [12]

El componente principal de cada nodo es el operador. El flujo consiste en un stream de datos que llega al puerto de entrada de un operador. Cada operador puede tener 1 o más puertos de entrada, que son los encargados de consumir los datos. El operador procesa los datos en memoria y produce un nuevo stream de datos como salida, que se envía por el puerto de salida de dicho operador. Cada operador puede tener 1 o más puertos de salida.

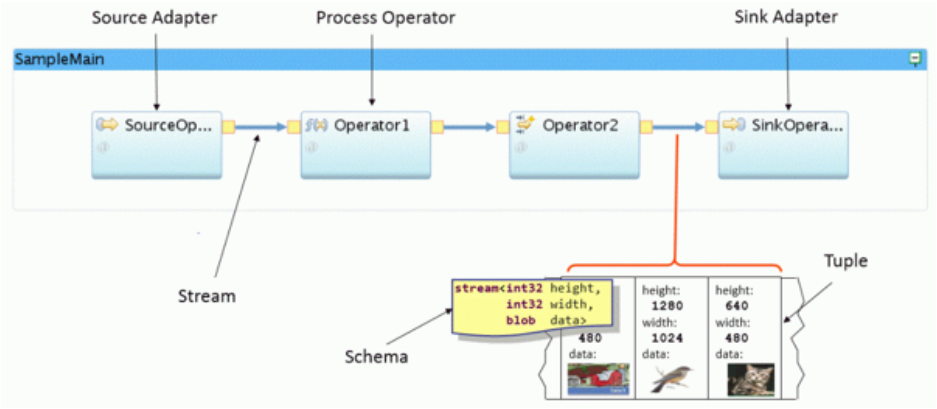


FIGURA 2.5: Ejemplo de una aplicación de Stream. Fuente: Streams Quick Start Guide [12]

A continuación se explican los diferentes componentes del flujo de ejemplo de la figura 2.5:

1. Source Adapter: se coloca al principio del grafo ya que es el encargado de leer datos de sistemas externos y producir un stream como salida
2. Stream: es una secuencia infinita de tuplas a ser procesadas
3. Tuple: es una lista estructurada de atributos
4. Schema: es una especificación de los tipos de datos y atributos de una tupla
5. Operator: procesa el stream de entrada y produce un nuevo stream. Puede operar en una ventana de datos (window). Un operador puede realizar diferentes tipos de procesamientos como por ejemplo filtrado, transformación, ordenamiento, etc.

6. Window: una secuencia finita de tuplas que el operador mantiene en memoria. Es posible definir ventanas de diferentes tipos, por ejemplo, por número de tuplas o por periodos. Es importante señalar que InfoSphere Streams mantiene en memoria RAM todos los datos necesarios, por lo tanto, las aplicaciones que analicen ventanas muy grandes requerirán mayor cantidad de memoria que aquellas aplicaciones que trabajen sobre ventanas pequeñas o tuplas individuales

7. Sink Adapter: se coloca al final de la aplicación ya que es el encargado de escribir datos en sistemas externos, y no produce un nuevo stream como salida

2.2.4. Modelo de negocio

IBM Bluemix permite elegir el modelo informático adecuado para su carga de trabajo y desplegar rápidamente más de 120 servicios, incluidos los servicios Watson. Permite la ejecución de aplicaciones en 49 centros de datos de IBM Cloud y tiene habilitada la facturación en 24 monedas.

El modelo de negocio consta de tres opciones: versión de prueba, adaptación progresiva y suscripción.

2.2.4.1. Versión de prueba

Permite el uso de Bluemix de forma gratuita durante 30 días. Esta versión incluye:

- Hasta 2 GB de memoria de contenedor y tiempo de ejecución
- Acceso a 10 servicios de IBM
- 1 certificado SSL

Una vez pasados los 30 días las aplicaciones de la cuenta se detienen. Para reiniciarlas es necesario proveer una tarjeta de crédito y elegir uno de los dos modelos de negocio disponibles: adaptación progresiva o suscripción. Estos planes incluyen el uso de algunos servicios, contenedores y tiempos de ejecución de forma gratuita; solo es necesario pagar por aquellos servicios, contenedores o tiempos de ejecución que no estén incluidos.

Si el pasaje a una cuenta paga no se realiza antes de los 60 días luego de que la versión gratuita haya expirado entonces las aplicaciones y configuraciones de servicios que se hayan hecho en los 30 días de prueba serán removidas.

También es posible solicitar un "código promocional" por ejemplo en caso que se quiera utilizar Bluemix en un ambiente de investigación para estudiantes o profesionales. Mediante estos códigos es posible agregar recursos o capacidades extras a la versión gratuita, como por ejemplo:

- Incrementar la cuota de memoria a un número de GB especificado por el código promocional
- Agregar una organización, con una cuota de memoria especificada por el código promocional

- Agregar un número ilimitado de organizaciones
- Cargar un número adicional de certificados SSL, según la especificación del código promocional
- Utilizar planes de servicio premium

2.2.4.2. Adaptación progresiva

Hay un número de servicios y tiempos de ejecución disponibles de forma gratuita; si se usa más de lo brindado gratuitamente entonces se cobrará por el uso extra que se dé.

Hay disponible un calculador de tarifas que permite simular el uso y obtener un costo aproximado de lo que se pagará en el mes de acuerdo a ese uso [13].

2.2.4.3. Suscripción

Se fija un precio mínimo por mes, personalizado de acuerdo al cliente y sus necesidades, y se recibe un descuento de suscripción que se aplica a ese cargo mínimo. El cliente también debe pagar por cualquier uso que exceda el monto mínimo de gasto fijado. Dentro de este modelo hay tres entornos de despliegue disponibles: público, dedicado o local.

- Suscripción Bluemix Público

  Permite crear aplicaciones en cuestión de minutos utilizando el tiempo de ejecución preferido, como Node.js, Java, PHP, Swift, Ruby y otros. Bluemix aloja más de 130 servicios exclusivos, entre los que se incluye IBM Watson. Las aplicaciones para principiantes están preconfiguradas y se pueden adaptar fácilmente para adaptarse a sus necesidades.

- Suscripción Bluemix Dedicado:

  Proporciona un entorno cloud privado, con hardware aislado físicamente en un centro de datos de IBM. Por cuestiones de seguridad, privacidad o rendimiento, muchas empresas necesitan este tipo de entorno.

  Bluemix Dedicado está diseñado para parecer un nodo más de la red local del cliente. Mediante tecnología VPN o Direct Link, el entorno Bluemix está conectado de manera segura a la red del cliente (su empresa).

  Bluemix Dedicado está disponible en más de 25 centros de datos IBM Cloud de todo el mundo. Esto significa que puede mantener sus datos en el país y colocar su informática y servicios donde sea mejor para sus usuarios.

  Bluemix Dedicado requiere un término mínimo de un año, e incluye:

  - Conectividad VPN a su infraestructura
  - Entorno totalmente redundante en un centro de datos de SoftLayer
  - Todos los tiempos de ejecución soportados (IBM Java Liberty, Node.js y runtime de código abierto incorporado)

- Todos los servicios dedicados que ha seleccionado y todos los servicios públicos de Bluemix
- Soporte Bluemix estándar

También se pueden pedir artículos/servicios opcionales como SoftLayer DirectLink o soporte premium.

Lo que paga cada mes durante ese período se basa en los servicios dedicados que desea, además de una cuenta de suscripción que le da acceso a todos los servicios públicos. El precio de Bluemix Dedicado se establece en base a la cantidad de capacidad asignada a su entorno.

Cada servicio disponible en dedicado también está disponible en público para que pueda probar y validar servicios antes de realizar la inversión para trasladarlos a su entorno dedicado.

- Suscripción Bluemix Local:

Bluemix Local se distribuye como servicio en el propio hardware del cliente o en una infraestructura convergente previamente integrada. El equipo de IBM mantiene una comunicación constante con el equipo de TI del cliente y colabora con el mismo en lo referido a actualizaciones, estado de la plataforma, parches y arreglos de seguridad.

Dado que Bluemix Local se despliega en el centro de datos del cliente, está muy cerca de sus sistemas existentes, por lo que puede mejorar el rendimiento de las aplicaciones híbridas.
En cloud público, dedicado y en local, la plataforma Bluemix central siempre es la misma. Esto significa que puede crear fácilmente aplicaciones y servicios en local y ampliarlos o migrarlos a una plataforma pública o dedicada, según el crecimiento de su estrategia o la evolución a lo largo del tiempo.

Bluemix Dedicado requiere un término mínimo de un año, e incluye:

- Una capacidad de entrega denominada Relay que permite a IBM conectarse a su implementación local y entregar actualizaciones de forma automática y consistente
- Todos los tiempos de ejecución soportados (IBM Java Liberty, Node.js y runtime de código abierto incorporado)
- Todos los servicios locales que ha seleccionado y acceso a todos los servicios públicos de Bluemix
- Soporte Bluemix estándar

Lo que el cliente paga cada mes durante ese período se basa en los servicios locales que desea, además de una cuenta de suscripción que le da acceso a todos los servicios públicos. Los cargos de uso de los servicios en Bluemix Public se calculan en base al acuerdo de cuenta de suscripción.

2.2.5. Tópicos

El broker MQTT de IBM utiliza tópicos para rutear los mensajes que en él se publican [14].

Existen varias formas de definir los tópicos, dependiendo del contexto en el que se esté. Si se trata de aplicaciones existen dos casos:

- Para suscribirse a eventos utilizan los tópicos de la siguiente forma: iot-2/type/{device type}/id/{device id}/evt/{event type}/fmt/{format type}
- Para publicar comandos utilizan los tópicos de la siguiente forma: iot-2/type/{device type}/id/{device id}/cmd/{command type}/fmt/{format type}

Si se trata de dispositivos existen también dos casos:

- Para publicar eventos utilizan tópicos de la siguiente forma: iot-2/evt/{event type}/fmt/{format type}
- Para suscribirse a comandos utilizan tópicos de la siguiente forma: iot-2/cmd/{command type}/fmt/{format type}

Un dispositivo puede enviar mensajes a diferentes tópicos, pero en general no es conveniente enviar muchos mensajes a partir de un único censo de datos ya que se genera más overhead en la red y se consume la cuota (cantidad de mensajes por la que se paga) innecesariamente.

La lógica de separación en tópicos forma parte de la aplicación cliente y no de la plataforma. Luego de que los mensajes son separados en tópicos se pueden aplicar reglas [15] dentro de la plataforma para tomar medidas a partir de esos datos (por ejemplo enviar notificaciones) y actuar en tiempo real, es decir antes de que los datos sean almacenados en la base de datos.

2.2.6. Demo

IBM posee algunos planes para que estudiantes puedan tener una licencia gratuita, durante 6 meses, para poder utilizar los servicios de Bluemix (en particular Watson) con fines de investigación. Gracias a esto se pudo acceder a la plataforma y de esta manera, mediante tutoriales que se encontraban en la documentación de Watson, realizar algunas pruebas básicas para luego implementar una simple aplicación que se podría usar en un entorno IoT.

El tutorial [16] explica paso a paso desde cómo registrar un dispositivo en Watson hasta como hacer una simple interfaz web que muestra la temperatura de un sensor y apague o prenda las luces de un dispositivo. Para realizar esto, se brindaban una serie de librerías y el código necesario, para simular en una computadora que se contaba con una placa Arduino, la cual contenía una luz led y un sensor de temperatura.

Siguiendo los pasos del tutorial se procedió a realizar el registro de esta placa de Arduino simulada en la computadora. También se brindaba el código necesario para crear los mensajes que serían enviados a la plataforma. Luego de realizar ciertos ajustes en el código brindado, se pudo ejecutar este sin problemas. Al ingresar a la plataforma se observó que efectivamente que existía una comunicación entre la plataforma y el dispositivo ya que se observó que estaban llegando los mensajes.

El tutorial finaliza con la implementación de una aplicación sencilla de control remoto para interactuar a través de la plataforma con el dispositivo. Para crear esta

aplicación se utilizó Node-RED y para el almacenamiento de datos se creó una base MongoDB. Esta aplicación además de recibir los datos de temperatura del sensor, daba la posibilidad de encender o apagar la luz led.

En las figuras 2.6, 2.7 y 2.8 se muestran una series de capturas en donde se puede ver los mensajes recibidos y enviados para apagar la luz led. En la figura 2.6 se muestra que la luz led se encuentra apagada ("led" : 0). En la figura 2.7 se muestra el mensaje que es enviado desde la plataforma para encender la luz (Message : true). Por último, en la figura 2.8 se muestra que al enviar los mensajes la luz ya se encuentra encendida ("led" : 1).

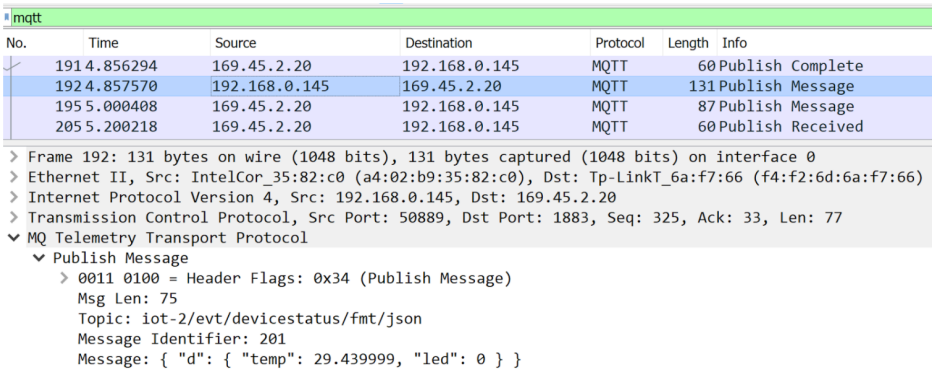


FIGURA 2.6: Captura del primer mensaje. Aplicación IBM Watson

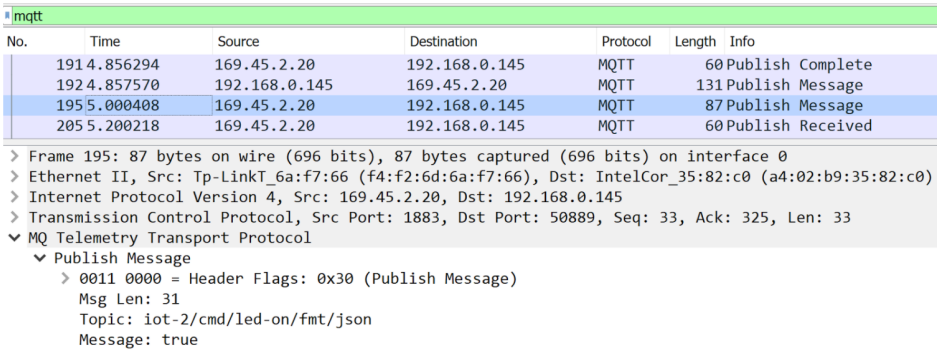


FIGURA 2.7: Captura del segundo mensaje. Aplicación IBM Watson

2.2.7. Tamaño de mensajes y payload

Los formatos posibles para el payload son [17]:

- JSON: es el formato estándar para Watson IoT. El esquema del mismo siempre debe tener como raíz un campo "d" ({"d": {...}}).
- Texto
- Binario

El tamaño máximo del payload soportado por Watson IoT es de 131072 bytes. Los mensajes que superen este tamaño serán rechazados por la plataforma, y el cliente que lo envió será desconectado de la misma.

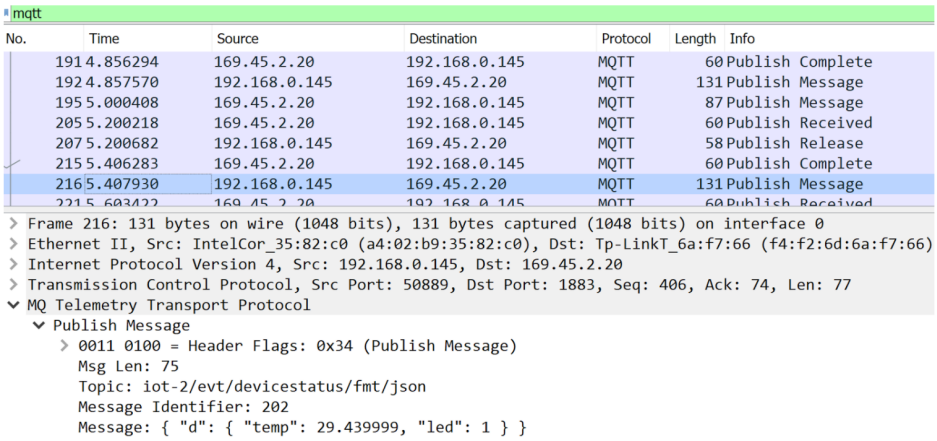


FIGURA 2.8: Captura del tercer mensaje. Aplicación IBM Watson

Analizaremos a continuación el caso concreto de un mensaje enviado durante la demo realizada anteriormente. Observamos que el tamaño total del mensaje enviado fue de 131 bytes. El mismo está formado por:

- 14 bytes de la cabecera de Ethernet
- 20 bytes de la cabecera de IP
- 20 bytes de la cabecera de TCP
- 2 bytes de la cabecera de MQTT
- 75 bytes de payload

De todo esto el único tamaño que se podría reducir, en caso de que fuera posible, es el tamaño del payload.

2.3. AWS IoT

AWS IoT [18] es una plataforma en la nube, desarrollada por Amazon, que permite a los dispositivos conectados interactuar con facilidad y seguridad con aplicaciones en la nube y con otros dispositivos.

AWS IoT ofrece autenticación y cifrado integral en todos los puntos de conexión, a fin de que los datos nunca se intercambien entre dispositivos y AWS IoT sin una identidad probada. Asimismo, permite proteger el acceso a dispositivos y aplicaciones mediante políticas con permisos granulares.

AWS IoT almacena el último estado de un dispositivo para que pueda leerse o definirse en cualquier momento, haciendo que el dispositivo aparezca en las aplicaciones como si estuviera conectado en todo momento. Esto supone que una aplicación puede leer el estado de un dispositivo incluso si está desconectado; también permite establecer el estado de un dispositivo e implementarlo cuando se vuelve a conectar.

Actualmente, AWS IoT se encuentra disponible en las siguientes regiones de AWS:

- EE.UU. Este (Norte de Virginia)

- EE.UU. Este (Ohio)
- EE.UU. Oeste (Oregón)
- UE (Irlanda)
- UE (Frankfurt)
- UE (Londres)
- Asia Pacífico (Sídney)
- Asia Pacífico (Seúl)
- Asia Pacífico (Tokio)
- Asia Pacífico (Singapur)

De todas formas se puede usar AWS IoT con independencia de la región geográfica en la que se encuentre, siempre y cuando se tenga acceso a una de las regiones de AWS anteriores.

2.3.1. Arquitectura y diseño lógico

AWS IoT está formado por los siguientes componentes:

- SDK para dispositivos: Permite conectar, autenticar e intercambiar mensajes con AWS IoT mediante protocolos MQTT, HTTP o WebSockets. El SDK para dispositivos admite C, JavaScript y Arduino. También es posible desarrollar y utilizar un SDK propio.
- Autenticación y autorización: AWS IoT ofrece autenticación mutua y cifrado en todos los puntos de conexión para que los datos nunca se intercambien entre dispositivos y AWS IoT sin una identidad probada. AWS IoT admite el método de autenticación de AWS (denominado "SigV4"), así como una autenticación basada en el certificado X.509. Las conexiones que utilizan HTTP pueden hacer uso de alguno de estos métodos, las que utilizan MQTT usan una autenticación basada en certificado y, por último, las conexiones con WebSockets pueden utilizar SigV4. Con AWS IoT se pueden usar certificados generados por AWS IoT y certificados firmados por la entidad de certificación (CA) de su elección.
- Gateway para dispositivos: Permite intercambiar mensajes utilizando un modelo de publicación o suscripción, lo que permite las comunicaciones individuales o múltiples (un dispositivo conectado difunde datos a varios suscriptores sobre un tema concreto). Este gateway admite protocolos MQTT, WebSockets y HTTP 1.1.
- Registro: El registro asigna una identidad única a cada uno de los dispositivos, de manera uniforme e independiente al tipo de dispositivo o a la forma en la que se conecta. Admite metadatos que describen las capacidades de un dispositivo, por ejemplo, si un sensor informa sobre la temperatura y si los datos son Fahrenheit o Celsius.

  El registro permite almacenar metadatos acerca de los dispositivos sin un costo extra. Además, los metadatos del registro no caducan, siempre que obtenga acceso o actualice su entrada en el registro al menos una vez cada 7 años.

- "Sombra" del dispositivo: Es una versión persistente, virtual o "sombra" de cada uno de los dispositivos, que incluye el último estado del dispositivo, de forma que las aplicaciones u otros dispositivos puedan leer mensajes e interactuar con él. Además de persistir el último estado, también es posible persistir el estado que se desea en el futuro para cada uno de los dispositivos, incluso aquellos que estén desconectados. AWS IoT comparará la diferencia entre el último estado registrado y el deseado y ordenará al dispositivo que la compense.

  Las sombras del dispositivo permiten almacenar el estado de los dispositivos hasta un año de forma gratuita. Las mismas persisten ilimitadamente si se las actualiza al menos una vez al año. De lo contrario, caducan.

- Motor de reglas: El motor de reglas evalúa los mensajes de entrada publicados en AWS IoT, los transforma y los entrega a otro dispositivo o servicio en la nube, en función de las reglas empresariales que defina. Una regla se puede aplicar a datos de uno o de varios dispositivos y puede tomar una o varias acciones en paralelo.

  El motor de reglas también puede direccionar mensajes a puntos de enlace de AWS, incluidos AWS Lambda, Amazon Kinesis, Amazon S3, Amazon Machine Learning, Amazon DynamoDB, Amazon CloudWatch y Amazon Elasticsearch Service con la integración incorporada de Kibana.

  Las reglas se pueden crear en la consola de administración o escribirlas utilizando una sintaxis similar a SQL. Las mismas pueden tener comportamientos diferentes según el contenido del mensaje. Por ejemplo, si la lectura de una temperatura supera un determinado límite, se podría activar una regla que transmitiera datos a AWS Lambda. También se pueden crear reglas para que tengan en cuenta otros datos en la nube, como los datos de otros dispositivos. Por ejemplo, se podría indicar que se realizara una acción si esta temperatura superara en un 15 % la media de los otros 5 dispositivos.

  El motor de reglas proporciona decenas de funciones disponibles que se pueden utilizar para transformar los datos y, además, es posible crear una cantidad ilimitada de ellas mediante AWS Lambda. Las reglas también pueden activar la ejecución de código Java, Node.js o Python en AWS Lambda, lo que le da la máxima flexibilidad y la posibilidad de procesar datos de dispositivos.

AWS IoT se integra directamente con otros servicios que brinda Amazon como por ejemplo Amazon S3 (para almacenamiento escalable en la nube), Amazon DynamoDB (provee el manejo de bases de datos no SQL), Amazon Kinesis (para el procesamiento en tiempo real de los datos de streaming a gran escala), AWS Lambda, Amazon Simple Notification Service y Amazon Simple Queue Service.

Es posible hacer que las aplicaciones se comuniquen directamente con los dispositivos conectados usando el gateway para dispositivos o el motor de reglas en AWS IoT, de forma de no tener que usar el registro ni las sombras de los dispositivos. Sin embargo, se recomienda su uso porque ofrecen una experiencia de desarrollo y administración más rica y estructurada.

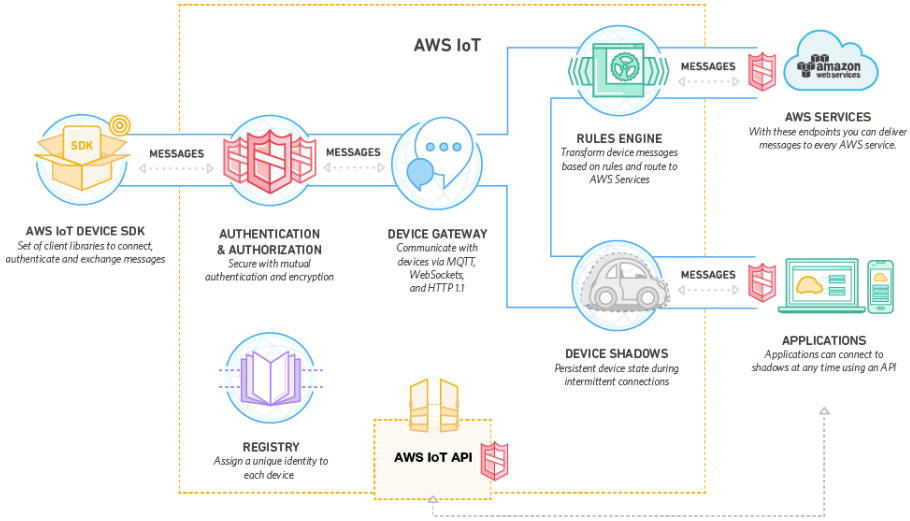


FIGURA 2.9: Arquitectura de AWS IoT. Fuente: How the AWS IoT Platform Works [19]

2.3.2. Almacenamiento y análisis de datos

Con AWS IoT es posible filtrar, transformar y utilizar datos de dispositivos sobre la marcha. AWS IoT facilita la utilización de servicios de AWS como AWS Lambda, Amazon Kinesis, Amazon S3, Amazon Machine Learning y Amazon DynamoDB, Amazon CloudWatch y Amazon Elasticsearch Service para lograr aplicaciones de IoT incluso más potentes.

2.3.3. Protocolos

AWS IoT es compatible con HTTP, WebSockets y MQTT, y además admite otros protocolos personalizados y propios del sector, y los dispositivos pueden comunicarse entre sí aunque utilicen protocolos distintos.

2.3.4. Lenguaje de aplicación

Actualmente, AWS ofrece los SDK para dispositivos de AWS IoT para los lenguajes C y Node.js, así como para la plataforma Arduino Yún. Además, varios fabricantes de hardware se han asociado a AWS para poner los SDK para dispositivos de AWS IoT disponibles en sus respectivas plataformas, e incluso como los SDK para dispositivos de AWS IoT son de fuente abierta, pueden ser migrados a los lenguajes y las plataformas de hardware que se desee si no son compatibles.

2.3.5. Modelo de negocio

Los precios dependen del número de mensajes publicados en AWS IoT (costo de publicación) y el número de mensajes enviados por AWS IoT a dispositivos o aplicaciones (costo de envío). La capa gratuita de AWS IoT permite comenzar con 250.000 mensajes gratuitos (publicados o enviados) al mes durante 12 meses. En la tabla 2.2 se pueden ver los precios por región:

Si se trata de grandes volúmenes de mensajería pueden existir precios personalizados.

Región	Precio
EEUU	5 USD por millón de mensajes
UE	5 USD por millón de mensajes
Asia Pacífico (Sídney, Seúl)	6 USD por millón de mensajes
Asia Pacífico (Tokio, Singapur)	8 USD por millón de mensajes

TABLA 2.2: Precios de AWS IoT por región

Un mensaje publicado o entregado consiste en un bloque de datos de 512 bytes. Aunque el mensaje sea de menor también se cobrar como un mensaje de 512 bytes. De la misma forma si se quiere enviar un mensaje de 900 bytes el mismo se contará como dos mensajes.

2.3.5.1. Medición de mensaje MQTT

MQTT Connect	Se miden por el tamaño del mensaje, incluido el tamaño del tema "Will" y la carga del mensaje "Will".
MQTT PubAck (recibido del dispositivo)	Se mide como un único mensaje de 512 bytes.
MQTT Ping	Incluye la solicitud y la respuesta Ping. Ambas se miden como un único mensaje de 512 bytes.
Suscripción MQTT	Se mide por el tamaño de la lista de temas enviada en el mensaje de suscripción.
Publicación MQTT (entrante)	Se mide el tamaño de la carga y el tema en bytes
Publicación MQTT (saliente)	Se mide el tamaño de la carga y el tema en bytes

TABLA 2.3: Medición de mensaje MQTT para AWS IoT

Los siguientes mensajes MQTT quedan excluidos de la medición:

- MQTT Disconnect
- MQTT ConnAck
- MQTT PubAck (enviado por el servicio)
- MQTT SubAck
- MQTT Unsubscribe

Los mensajes MQTT transferidos dentro de una conexión WebSocket se miden como mensajes MQTT dentro de una conexión TLS mutuamente autenticada.

2.3.5.2. Medición de solicitudes y respuestas HTTP

En la tabla 2.4 se describe la medición de solicitudes y respuestas HTTP para AWS IoT.

Solicitudes	Las solicitudes HTTP REST API se miden según el número de mensajes de 512 bytes necesarios para codificar el cuerpo de la solicitud HTTP.
Respuestas	Las respuestas HTTP REST API se miden según el número de mensajes de 512 bytes necesarios para codificar el cuerpo de la respuesta HTTP.
Respuestas negativas	En el caso de respuestas HTTP con códigos de respuesta HTTP en el intervalo 4xx y 5xx y que contiene un cuerpo, AWS IoT mide el número de mensajes de 512 bytes necesarios para codificar el cuerpo de la respuesta.

TABLA 2.4: Medición de solicitudes y respuestas HTTP para AWS IoT

2.3.6. Tópicos

El broker de AWS utiliza tópicos para rutear los mensajes publicados hacia los suscriptores. Los tópicos son definidos por el usuario, y pueden tener varios niveles de jerarquía que se crean utilizando "/" para separarlos. Por ejemplo un tópico válido es *"sensor/temp/habitacion1"*.

En AWS los tópicos están aislados para cada cuenta y región [20]. Esto significa que el tópico *"sensor/temp/habitacion1"* de una cuenta AWS es independiente de *"sensor/temp/habitacion1"* de otra cuenta AWS. A su vez el tópico *"sensor/temp/habitacion1"* de una cuenta en la región us-east-1 es independiente del tópico *"sensor/temp/habitacion1"* de la misma cuenta pero en la región us-east-2. Esto se debe a que AWS no permite enviar ni recibir mensajes a través de distintas cuentas o regiones.

Cualquier tópico que empiece con $ se considera reservado y no es posible utilizarlo para suscribirse o publicar, excepto una lista predefinida de ellos que se puede encontrar en la documentación de AWS [21].

2.4. Kaa

Kaa [22] es una plataforma open-source de middleware multipropósito para Internet of Things (IoT), que permite construir soluciones completas de IoT de extremo a extremo, aplicaciones conectadas y productos inteligentes.

Kaa presenta varias ventajas, que hacen que el desarrollo de IoT sea rápido y fácil. En primer lugar, Kaa es independiente del hardware. Esto significa que no se requieren modificaciones para la ejecución en diferentes dispositivos, generando así un alto nivel de compatibilidad con prácticamente cualquier tipo de dispositivos conectados, sensores y gateways. Además cuenta con opciones ilimitadas para protocolos de conectividad e integración con análisis.

Entre las capacidades clave que brinda Kaa para las soluciones IoT se destacan:

- Administrar un número ilimitado de dispositivos conectados
- Configurar la interoperabilidad entre dispositivos
- Realizar pruebas de servicio A/B
- Realizar el aprovisionamiento y la configuración de dispositivos remotos

- Realizar monitoreo de dispositivos en tiempo real
- Distribuir actualizaciones de firmware por aire
- Crear servicios en la nube para productos inteligentes
- Recopilar y analizar datos
- Analizar el comportamiento del usuario y entregar notificaciones específicas

2.4.1. Arquitectura y diseño lógico

La plataforma está compuesta por:

- Servidores Kaa: son el back-end de la plataforma. Exponen interfaces de integración y se encargan de administrar tenants (entidades de negocio que utilizan una instancia de Kaa para ejecutar de forma independiente aplicaciones Kaa), aplicaciones, usuarios y dispositivos.
- Extensiones Kaa: módulos de software independientes que implementan un conjunto aislado de funciones destinadas a ampliar las capacidades de comunicación entre el servidor Kaa y los clientes Kaa.
- Endpoint SDKs: biblioteca utilizada para la comunicación, serialización de datos, persistencia y otras funciones realizadas entre un endpoint (entidad cliente administrada por una instancia de Kaa, como por ejemplo dispositivos, sensores, teléfonos móviles, etc) y un servidor Kaa. Los SDK están diseñados para facilitar la creación de aplicaciones cliente que se ejecuten en varios dispositivos conectados.

Los nodos de servidores se interconectan formando clusters. Cada cluster está asociado a una instancia de Kaa particular, y requieren instancias de bases de datos NoSQL y SQL para almacenar datos y metadatos de los endpoints. En la figura 2.10 se muestra el diagrama de la arquitectura.

Cada nodo ejecuta una combinación de servicios de Control, Operaciones y Bootstrap.

2.4.2. Almacenamiento y análisis de datos

Kaa utiliza dos tipos de bases de datos: SQL y NoSQL.

2.4.2.1. Base de datos SQL

La base de datos SQL se utiliza para almacenar metadatos de tenants, aplicaciones, grupos de endpoints y otros. Estos datos tienen la característica de no crecer a medida que aumenta el número de endpoints registrados, y gracias a eso son adecuados para almacenarlos en un base de datos relacional.

Kaa soporta MariaDB y PostgreSQL como bases de datos SQL.

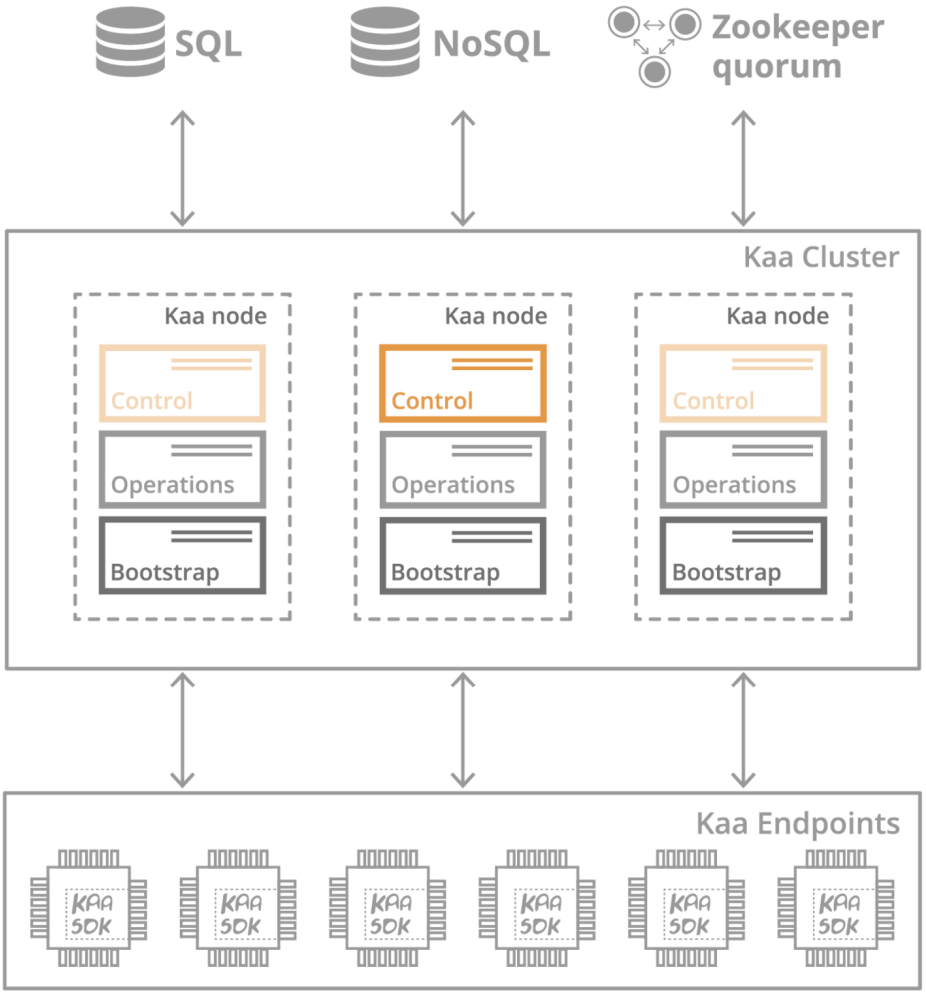


FIGURA 2.10: Arquitectura de Kaa. Fuente: Kaa Architecture Overview [23]

2.4.2.2. Base de datos NoSQL

La base de datos NoSQL se utiliza para guardar datos relacionados a los endpoints, como por ejemplo los datos generados por los mismos. Estos datos crecen linealmente con el aumento del número de endpoints, y es por esto que las bases de datos no relacionales son las más adecuadas para almacenarlos.

Kaa soporta Apache Cassandra y MongoDB como bases de datos NoSQL.

2.4.3. Disponibilidad y escalabilidad

Los clusters de Kaa escalan horizontal y linealmente. Kaa es además una plataforma elásticamente escalable, lo que significa que cuando la carga aumenta la plataforma escala agregando más recursos, y cuando la carga disminuye se remueven los recursos innecesarios. Este tipo de escalabilidad se utiliza mayormente en entornos de computación en la nube, ya que generalmente se paga por uso y de esta forma se evita pagar por recursos que no se están usando activamente.

Otro aspecto importante de Kaa es que su arquitectura no presenta un único punto de falla. Los servicios de Operaciones y Bootstap son idénticos en todos los nodos

del cluster, y funcionan en modo activo-activo en HA (High Availability), lo que permite lograr el balanceo de carga. Hay dos métodos de equilibrio de carga que se utilizan dependiendo del creador de las solicitudes para el cluster: Kaa endpoint SDK o REST API. A su vez uno de los nodos del clúster contiene un servicio de Control activo. En caso de que el nodo falle, se activa un servicio de Control de otro nodo, que hasta el momento estaba en inactivo.

2.4.4. Transporte

Kaa está diseñado para soportar virtualmente cualquier protocolo de transporte de datos.

Kaa permite utilizar diferentes protocolos de transporte, como por ejemplo HTTP y MQTT. Se pueden definir diferentes canales para distintas acciones realizadas sobre un mismo endpoint, y cada canal soporta un protocolo de transporte específico. Por ejemplo: se puede definir un canal para enviar notificaciones y otro para enviar datos de configuración y perfil. En el canal de notificaciones se pueden enviar mensajes SMS y en el canal de configuración se puede utilizar el protocolo TCP. Cada canal es responsable de la codificación, compresión y entrega de datos.

Kaa provee implementaciones de canales de transporte por defecto, pero además permite crear implementaciones personalizadas de canales de transporte para cualquier servicio.

Para cada canal existen tres modos de transferencia de datos:

- Upstream: los datos se envían desde el endpoint al servidor
- Downstream: los datos se envían desde el servidor al endpoint
- Bidireccional: los datos viajan en ambos sentidos

En la tabla 2.5 se describen los canales por defecto ofrecidos por Kaa.

2.4.5. Lenguajes de aplicación

Las aplicaciones cliente pueden escribirse utilizando diferentes lenguajes de programación: C, C++, Java y Objective-C.

2.4.6. Modelo de negocio

La plataforma Kaa IoT está licenciada bajo Apache Software License 2.0, tanto el servidor como los componentes cliente. Kaa es 100 % libre, y se puede utilizar para crear soluciones de código abierto o propietarias (productos comerciales).

2.4.7. Notificaciones y tópicos

El subsistema de notificación de Kaa permite la entrega de mensajes desde el servidor Kaa a los endpoints. La estructura de los datos transportados por las notificaciones está definida por el esquema de notificación, que se configura en el servidor Kaa y se integra en los endpoints de Kaa.

Canal	Servidor destino	Servicios	Modos de transferencia soportados	Protocolo
Default bootstrap	Bootstrap	Bootstrap	Bidireccional	HTTP 1.1
Default operation long poll	Operations	Todos menos Bootstrap	Bidireccional para perfiles, configuración, notificaciones y asociación de usuarios. Downstream para eventos y logs	HTTP 1.1 long poll
Default operation HTTP	Operations	Eventos y logging	Upstream para eventos y logs	HTTP 1.1
Default KaaTCP channel	Operations	Todos menos Bootstrap	Bidireccional	MQTT v3.1

TABLA 2.5: Canales de transporte ofrecidos por defecto por Kaa

En Kaa las notificaciones están organizadas en tópicos. Cada tópico puede estar asociado a uno o varios endpoints groups. Para suscribirse a una notificación específica, los endpoints deben pertenecer a uno o más endpoints groups asociados con el tópico de notificación correspondiente.

Los tópicos pueden ser obligatorios u opcionales. Las notificaciones de tópicos obligatorios se entregan de manera forzada. Los tópicos opcionales requieren suscripción. Es responsabilidad del código del cliente agregar oyentes de notificación y suscribirse a los tópicos opcionales. Las notificaciones y tópicos se pueden administrar mediante la interfaz Admin UI o mediante la API REST.

2.4.8. Demo

Ejecutamos una aplicación de Kaa Data Collection en Java. La misma consistió en la recolección de registros de endpoints. Los registros enviados desde los endpoints se envían a los Servicios de Operaciones donde son persistidos para su posterior procesamiento, o bien se envían para el análisis inmediato del flujo de datos. La aplicación muestra cómo se pueden registrar las lecturas de un sensor de temperatura en una base de datos y configurar el período de muestreo del sensor. En este caso los registros de las temperaturas son simuladas por software, es decir que no se obtienen realmente de un sensor.

Para poder ejecutar esta demo tuvimos que descargar la Sandbox del sitio oficial de Kaa. Descargamos el proyecto Java de la aplicación de demostración y lo ejecutamos. En la consola van apareciendo las temperaturas recolectadas. Los datos se almacenan en la base de datos MongoDB de la Sandbox usando el MongoDB log appender. Para acceder a los datos recopilados se consulta una instancia de la base de datos MongoDB en el Sandbox corriendo el siguiente comando en la consola de Kaa Sandbox:

$ mongo kaa

db.logs_$your_application_token$.find()

Para saber el token de la aplicación (your_application_token) vamos a la web de Kaa Sandbox =>Administration UI, y utilizamos las credenciales devuser con password devuser123. Luego vamos a Applications =>Data collection demo y ahí podemos encontrar el token de la aplicación.

Interceptamos con Wireshark los mensajes que se enviaban desde la aplicación a la Sandbox, pero no pudimos ver el contenido de los mismos. En la implementación de transporte por defecto la comunicación de datos entre el SDK de Kaa y el servidor de Kaa está asegurada mediante cifrado híbrido con RSA y AES; creemos que es por esto que no fuimos capaces de ver lo mensajes, y no encontramos la forma de cambiar esa configuración por defecto.

2.5. Fiware

Fiware [24] es una plataforma abierta y estándar, impulsada por la Unión Europea, que proporciona un entorno basado en OpenStack y un conjunto de API (llamados Generic Enablers - GE) que facilitan la conexión a IoT, procesan y analizan grandes volúmenes de datos en tiempo real o incorporan funciones avanzadas para la interacción del usuario.

FIWARE propone e introduce la idea de tener un estándar único para recopilar, gestionar, publicar e informar sobre los cambios en la información que está siendo recolectada por la plataforma. Hasta el momento no existe un estándar a nivel mundial para poder manipular toda esta información de contexto que se recolecta. La ausencia de estándares provoca, por ejemplo, que una solución IoT diseñada para una ciudad no pueda ser aplicada en otra sin tener que realizar esfuerzos de adaptación que seguramente incrementarán significativamente los costos del proyecto. Este estándar se llama NGSI; en un apartado más adelante se encuentra una descripción más detallada del mismo.

2.5.1. Arquitectura

Los componentes principales que forman la arquitectura de FIWARE son:

- IDAS: Es una implementación del Backend Device Management GE. Está formado por los IoT Agents, que son los encargados de traducir los protocolos específicos de IoT de los dispositivos (Ultralight2.0, MQTT, LWM2M/CoAP, etc) al protocolo de información de contexto NGSI. No se necesita este componente si los dispositivos o gateways admiten de forma nativa la API NGSI.
  Al usar los agentes, los dispositivos se representan en FIWARE como entidades NGSI en un ContextBroker.

- Orion Context Broker: Es una implementación del Publish/Subscribe Context Broker GE, que provee las interfaces NGSI9 y NGSI10. Mantiene representaciones virtuales de los dispositivos físicos. Tiene la capacidad de mediar entre los sensores y los consumidores de los datos que éstos generan.

  El Context Broker almacena los últimos datos generados por los dispositivos en MongoDB. Si se quiere almacenar el histórico de los datos generados por los dispositivos es necesario utilizar algún GE que lo permita y conectarlo con

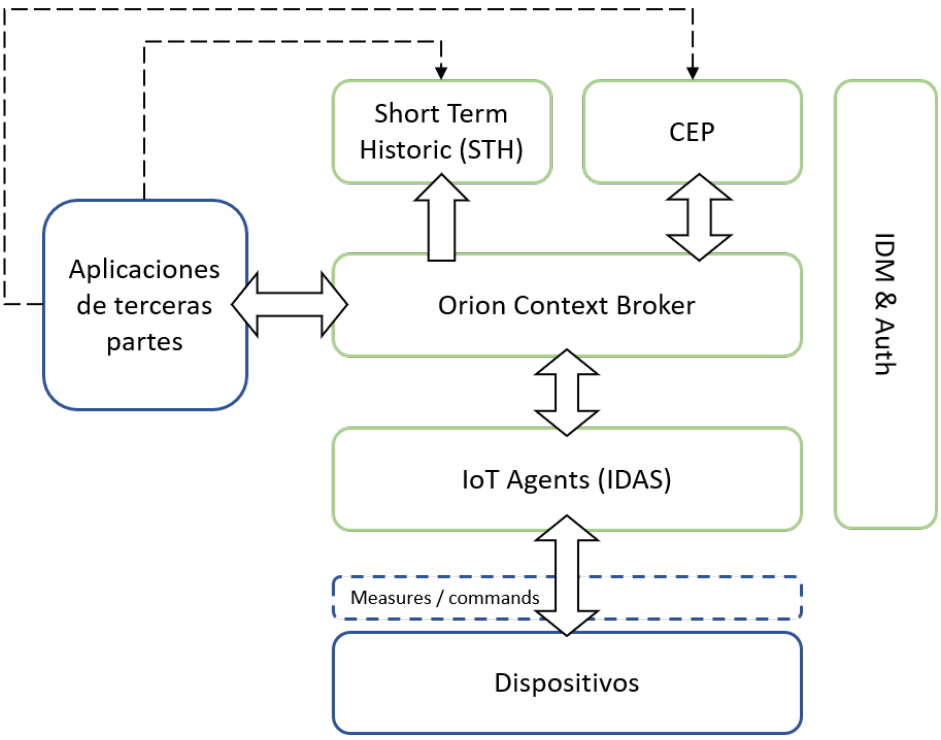


FIGURA 2.11: Arquitectura de Fiware

el Context Broker. Por ejemplo, es posible conectar el Context Broker con Cygnus que le permite conectarse con algún medio de almacenamiento específico como HDFS, CKAN o MySQL.

- IDM & Auth: Brindan los mecanismos para garantizar la confiabilidad, seguridad y privacidad en la entrega y uso de servicios.
- Short Term Historic (STH): Este componente se encarga de proporcionar información agregada de series de tiempo sobre la evolución en el tiempo de los valores de los atributos de una entidad. La utilidad de estos componentes es brindar mayor facilidad y simplicidad para el manejo de información agregada que la que ofrece una herramienta analítica de big data.
- Complex Event Processing (CEP): Analiza datos de eventos en tiempo real. Pretende analizar y reaccionar a los eventos generando una respuesta inmediata y desencadenando acciones de acuerdo a las condiciones cambiantes.

Un dato a tener en cuenta es que no es necesario el registro de dispositivos específicos, a menos que se necesite modificar algunos parámetros específicos del dispositivo (por ejemplo, zona horaria, asignación de atributos a Context Broker) o se desee configurar comandos de dispositivo. No obstante, a partir de la versión 2 del Orion Context Broker (NGSI10), es necesario crear las entidades antes de poder actualizar el valor de sus atributos.

2.5.2. NGSI (Next Generation Service Interface)

La API NGSI de Fiware define:

- Un modelo de datos para la información de contexto, basado en un modelo de información simple que utiliza la noción de entidades de contexto

- Una interfaz de datos de contexto para intercambiar información mediante operaciones de consulta, suscripción y actualización
- Una interfaz de disponibilidad de contexto para intercambiar información sobre cómo obtener información de contexto (separar las dos interfaces está actualmente en discusión)
- Un conjunto de roles típicos desempeñados por componentes compatibles con NGSI

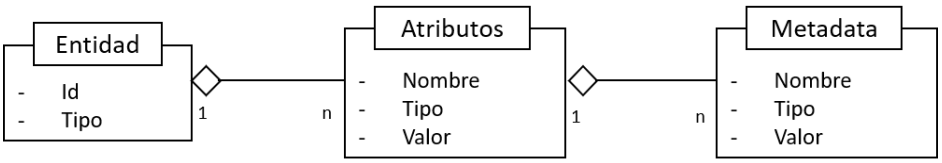


FIGURA 2.12: Modelo de datos de NGSI

Los elementos principales del modelo de datos de NGSI son:

- Entidad: es la representación virtual de todo tipo de objeto físico o lógico (por ejemplo sensor, persona, tabla, habitación, etc). Cada entidad tiene un id y un tipo. Los tipos de las entidades son tipos semánticos, que intentan describir el tipo de cosa que representa la entidad.

  La representación de una entidad en NGSI 10 v2 es mediante un JSON con la siguiente estructura:

```
{
  "id": "entityID",
  "type": "entityType",
  "attr_1": <val_1>,
  "attr_2": <val_2>,
  ...
  "attr_N": <val_N>
}
```

- Atributos: son propiedades de las entidades. Los atributos tiene un nombre, un tipo, un valor y metadata opcional. Por ejemplo la temperatura corporal de una persona se puede representar por un atributo de nombre “body_temperature”, del tipo “temperature”, con valor “37.5” y como metadata puede tener el momento en el que se tomo la temperatura, la unidad de medida, etc.
- Metadata: la metadata se puede usar en varios lugares en NGSI; uno de ellos es como campo opcional dentro de los atributos. Cada objeto metadata tiene un nombre, un tipo y un valor. Se debe tener en cuenta que en NGSI no está previsto que los metadatos puedan contener metadatos anidados.

2.5.3. Publish/Subscribe Context Broker

El Context Broker GE es un Publish/Subscriber que permite la publicación de información de contexto por parte de los Context Producer para que esté disponible para su consumo por otras entidades, conocidas como Context Consumers. Las

aplicaciones y otros GEs en FIWARE pueden cumplir el rol de Context Producers, Context Consumers o ambos.

El Publish/Subscribe Context Broker GE soporta dos tipos de comunicación: push y pull. Esto significa que los Context Producers pueden insertar información de contexto en el broker cuando la información esté disponible o cuando el Context Producer lo desee. A su vez el broker puede pedir información de contexto a los Context Producers si estos proveen la habilidad de ser consultados (en este caso se los conoce como Context Providers). De manera similar, los Context Consumers pueden pedir información de contexto al broker cuando lo deseen (on-request mode), asi como tambien el broker puede pushear información de contexto a los Context Consumers interesados (subscription mode).

El Publish/Subscribe Context Broker cuenta con las siguientes operaciones principales, dependiendo de la interfaz NGSI que esté implementando [25]:

- NGSI-10 v1:
  - /updateContext
  - /queryContext
  - /subscribeContext
  - /unsubscribeContext
  - /updateContextSubscription
  - /notifyContext
- NGSI-10 v2:
  - /entities
  - /entities/<entityId>/attrs
  - /subscriptions

A partir de la versión 2 de NGSI10 es necesario registrar todas las entidades involucradas para luego poder modificar sus atributos asociados. Esto se hace mediante un POST al endpoint */entities*. Luego, para modificar uno o varios atributos asociados a un entidad se realiza un POST al endpoint */entities/<entityId>/attrs*.

En la versión 2 la creación y borrado de suscripciones se realizan a través del endpoint */subscriptions*. Una de las modificaciones que hubo con respecto a las suscripciones es que a partir de esta versión no se puede especificar el tipo de condición bajo la cual se recibirá una notificación por parte del Context Broker como en la versión 1 [26], sino que por defecto envía las notificaciones ante cualquier cambio en los atributos especificados.

2.5.3.1. Principales componentes dentro del modelo

- Publish/Subscribe Context Broker: como ya se mencionó anteriormente es el principal componente de la arquitectura. Tiene como principal función, controlar el flujo de datos (contexto) entre las entidades de la arquitectura.
- Context Producer: es el componente encargado de generar los datos de contexto a ser transferidos al Context Broker.

- Context Provider: es un tipo especial de Context Provider que proporciona información de contexto a demanda en modo síncrono, esto significa que el Context Broker o inclusive un Context Consumer puede invocar el Context Provider con el fin de adquirir información de contexto. Para esto el Context Provider debe registrar su disponibilidad enviando mensajes apropiados al Context Broker y a su vez debe exponer interfaces para proporcionar la información de contexto.

- Context Consumer: podría decirse que es la entidad interesada en la información de contexto, por ejemplo una aplicación, puede obtener la información enviando una petición al Context Broker o enviando directamente una petición al Context Provider. Otra forma que tiene el Context Consumer de obtener la información es mediante la suscripción a actualizaciones sobre la información de contexto. El Context Consumer registra, junto con la suscripción, una operación callback o una url a un servicio que será utilizada por el Context Broker para notificar al Context Consumer sobre las actualizaciones. Comúnmente se utiliza como referencia una url que brinde un servicio REST.

La figura 2.13 muestra los principales componentes de esta arquitectura del Context Broker y cómo interactúan entre sí.

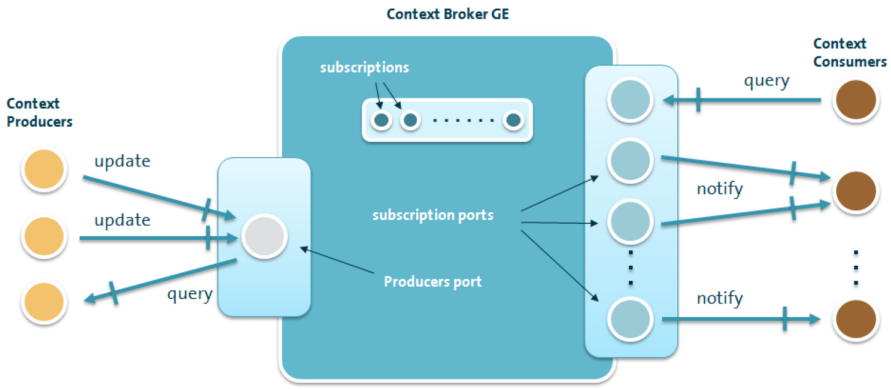


FIGURA 2.13: Arquitectura Publish/Subscribe Fiware. Fuente: Context Broker - Architecture Description [27]

2.5.4. Almacenamiento y análisis de datos

Existen varios backends donde crear y mantener conjuntos de datos históricos con respecto a la información de contexto manejada en tiempo real. Algunos de ellos son:

- HDFS: el sistema de archivos distribuidos de Hadoop

- MicroStrategy: el componente de Business Intelligence. Maneja una base de datos MySQL privada por servicio FIWARE.

- CKAN: la plataforma de datos abiertos de FIWARE. En base a las capacidades de almacenamiento, los datos se pueden cargar en FileStore (un gestor de archivos simple) o en el DataStore, que se basa en PostgreSQL. CKAN no provee ninguna herramienta analítica.

- STH Comet: una base de datos histórica a corto plazo construida sobre MongoDB, para el consumo privado de datos. Permite además realizar agregación de los datos.

También es posible procesar con Big Data GEs grandes conjuntos de datos, ya sea por sí mismos o en combinación con información de contexto.

FIWARE permite realizar consultas y actualizar los datos almacenados, suscribirse a cambios en los datos, y realizar consultas geolocalizadas.

FIWARE también permite el procesamiento en tiempo real de flujos de datos multimedia, como videos, mensajería instantánea multimedia, imágenes, audios, entre otros. Para lograr esto se puede utilizar el Real-time Multimedia Stream processing GE.

2.5.5. Cygnus

El GE Orion Context Broker únicamente almacena el último valor de cada atributo para cada entidad. Para almacenar de manera histórica estos datos existe el GE Cygnus [28], que es un conector que permite almacenar los datos de contexto seleccionados provenientes del Context Broker en un almacenamiento basado en HDFS, MongoDB o MySQL, entre otros.

La conexión del Context Broker con Cygnus es sencilla, ya que aprovecha la funcionalidad de suscripción del broker, por lo que basta únicamente con realizar una suscripción al mismo pasándole como url de referencia la dirección donde corre Cygnus, y especificando los atributos y entidades que se desean persistir.

Internamente, Cygnus está basado en Apache Flume [29], tecnología desarrollada para el diseño y la ejecución de recopilación de datos y agentes de persistencia. La forma más simple de usar Cygnus consiste en utilizar flujos básicos de source - channel - sink como se describe en la documentación de Apache Flume. El ***source*** (HttpSource) recibe los mensajes del Orion Context Broker y los acumula en un ***channel*** (MemoryChannel) luego de transformarlos en eventos Flume. Por último el ***sink*** se encarga de consumir los eventos que están en el channel y transformarlos en datos apropiados para persistirlos en la plataforma de almacenamiento externa elegida.

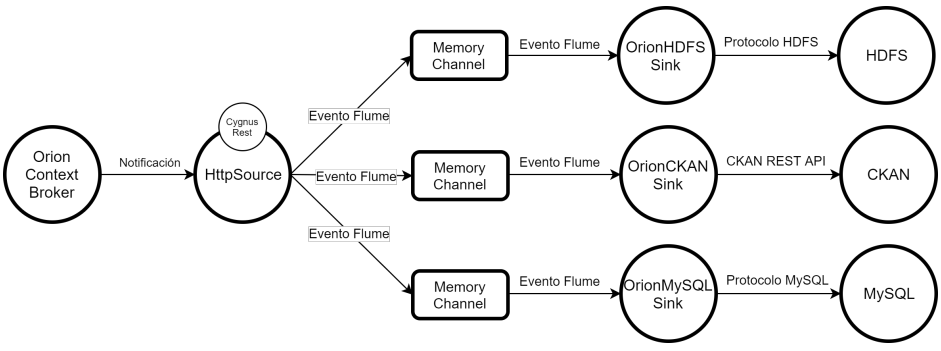


FIGURA 2.14: Ejemplo de arquitectura básica de Cygnus

La figura 2.14 muestra una simple configuración de un agente Cygnus. No obstante, existen variedad de configuraciones distintas para poder mejorar y sacar el mejor rendimiento posible a Cygnus. Por ejemplo, se pueden tener múltiples *sinks* consumiendo de un único *channel*, está configuración puede agregar un overhead ya que los *sinks* tienen que "competir" por el *channel*. También se pueden tener múltiples *channels* y múltiples *sinks*; la dificultad de tener ésto es que se necesita de algún tipo de selector para para poder distribuir los eventos en los distintos *channels*. En la documentación de la configuración de Cygnus [30] se muestran ejemplos de estas y de otras configuraciones de un nivel más avanzado que se pueden utilizar en Cygnus, que dependen del problema que se quiere solucionar.

2.5.6. Complex Event Processing (CEP)

El CEP es un GE encargado de analizar los flujos de datos en el tiempo, genera información inmediata y permite una respuesta instantánea a las condiciones cambiantes. CEP reacciona a situaciones en lugar de eventos únicos. Una situación se define como una condición que se basa en una serie de eventos que se han producido a lo largo de una ventana de tiempo dinámica llamada contexto de procesamiento.

IBM Proactive Technology Online, conocido también con el nombre de Proton [31], es una implementación de FIWARE CEP GE. Es una plataforma web integrada escalable para permitir el desarrollo, implementación y mantenimiento de aplicaciones basadas en eventos. Esta plataforma está implementada en Java y para poder ser desplegada utiliza Apache Tomcat. Una de las grandes ventajas que tiene esta tecnología es que permite la construcción de aplicaciones CEP a través de una interfaz web.

Con este GE se pueden definir patrones sobre eventos específicos [32], por ejemplo aquellos que ocurren dentro de una ventana de tiempo. Estos patrones se definen, como ya se mencionó anteriormente, usando una interfaz web sin la necesidad de escribir ningún código. Esto genera la ventaja de que se hace más fácil escribir la lógica de procesamiento de eventos, mantenerla y cambiarla a lo largo del tiempo, ya que no se necesita de ningún conocimiento de programación en ningún lenguaje específico. Ejemplos de patrones son:

- Sequence: los eventos deben ocurrir en un orden específico para que se detecte el patrón.
- Aggregate: utiliza funciones de agregación (promedio, suma, resta, etc) sobre un conjunto de eventos.
- Absent: detecta la ausencia de un determinado evento dentro de una ventana de tiempo definida.
- All: todos los eventos especificados deben llegar para que el patrón coincida.

A su vez cada patrón está asociado con un contexto de procesamiento de eventos. Éste agrupa instancias de eventos para que puedan procesarse de forma relacionada. Asigna a cada instancia de evento una o más particiones de contexto. El contexto puede ser de procesamiento temporal, de procesamiento en segmentación o un contexto compuesto por los dos tipos anteriores.

- Temporal: define una ventana de tiempo. Si un patrón está asociado a un contexto de este tipo, se procesan sólo los eventos que lleguen dentro de esa ventana de tiempo.
- Segmentación: define un criterio de coincidencia basado en alguno de los atributos del evento. Por ejemplo, separa los eventos según un ID.

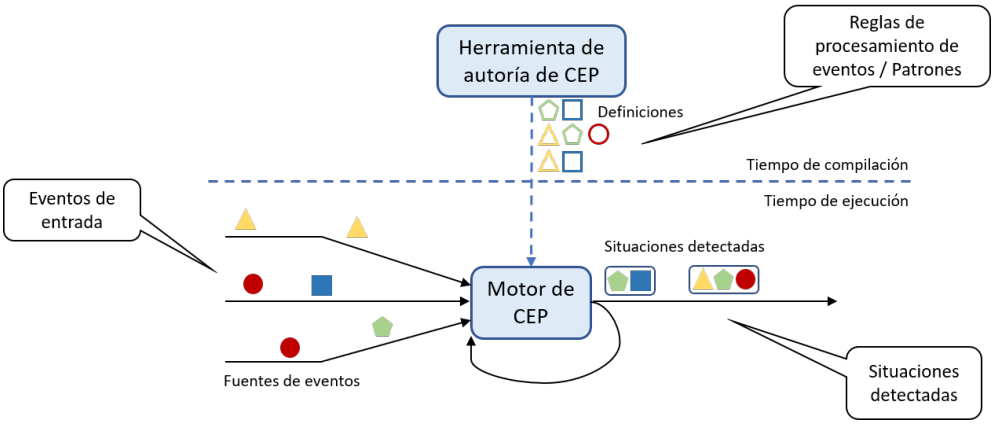


FIGURA 2.15: Arquitectura de CEP a alto nivel

2.5.7. Protocolos

FIWARE admite los protocolos de transporte estándar (TCP, UDP) así como los túneles cifrados (TLS, DTLS). Además soporta HTTP, MQTT y CoAP. Si el dispositivo utiliza un protocolo diferente será necesario realizar una traducción específica entre el protocolo del dispositivo y el modelo estándar de FIWARE (NGSI). Para esto se ofrecen diferentes frameworks de desarrollo de IoT Agents.

Los protocolos utilizados definir el formato del payload de los mensajes son Ultralight 2.0 y JSON.

2.5.8. Tópicos

Las suscripciones a ciertos eventos permiten, como se comentó anteriormente, que los consumidores reciban únicamente los datos que sean interesantes para ellos. Para suscribirse a estos datos o eventos es necesario realizar un POST al endpoint */subscriptions* de NGSI-10 v2 [33]. En la solicitud se especifican varios datos, entre ellos :

- los tipos de entidades cuyos eventos se desean escuchar
- los atributos cuyos valores se desean escuchar
- una url de referencia utilizada como callback una vez que ocurre el evento
- la duración de la suscripción
- los valores de la condición.

Si esta solicitud es exitosa la respuesta devuelve, entre otros datos, un subscriptionId que se utilizará para saber a qué suscripción corresponde el mensaje enviado

por el Context Broker cuando éste notifique a los consumidores, además de usarse para desuscribirse en caso de ser necesario.

Un ejemplo del payload que se envía para realizar una suscripción es el siguiente:

```
{
  "description": "Descripcion de la suscripcion",
  "subject": {
    "entities": [
      {
        "id": "Habitacion1",
        "type": "Habitacion"
      }
    ],
    "condition": {
      "attrs": [“presion”]
    }
  },
  "notification": {
    "http": {
      "url": "http://localhost:1028/accumulate"
    },
    "attrs": [“temperatura”]
  },
  "expires": "2016-04-05T14:00:00.00Z",
  "throttling": 5
}
```

El campo *expires* (equivalente a *duration* en NGSI 10 v1) se especifica según la ISO 8601 e indica el tiempo durante el cual la suscripción estará activa, luego de este tiempo la suscripción será ignorada. Si no se especifica la suscripción será permanente. Por otro lado el campo *throttling* es usado para especificar el intervalo de tiempo entre cada notificación. En el ejemplo se usa un intervalo de 5 segundos, o sea que si se envía una notificación, la siguiente se enviará a los 5 segundos, si es que todavía se sigue cumpliendo la condición.

Es importante observar que la lista de atributos que se envían dentro de *condition* no tiene por qué coincidir con los atributos que se desean escuchar. En este ejemplo el Context Broker notificará a los consumidores correspondientes cuando haya un cambio en la presión, pero en la notificación sólo viajará el valor de la temperatura y no el de la presión, ya que este es el único atributo que se especifica en *attributes*.

2.5.9. Demo

Realizamos un tutorial que obtuvimos del canal de Youtube de Mooc INFOTEC [34], en el que llevamos a cabo los siguientes pasos:

1. Descargamos el Context Broker Orion de la página oficial de Fiware
2. Levantamos ese broker en una máquina virtual.En la configuración de red de la VM cambiamos NAT por Adaptador sólo anfitrión (para poder acceder desde la máquina host)

3. También levantamos una VM de Ubuntu con la misma configuración de red

4. Escribimos el código del tutorial (python) y lo ejecutamos en la máquina con Ubuntu. Este código se comunica con el Context Broker Orion.

El ejemplo simula la obtención de datos meteorológicos, los inserta en el Context Broker y luego los consulta, edita y elimina.

Utilizando Wireshark pudimos capturar el mensaje que se enviaba al Context Broker. El detalle de la captura se puede ver en la figura 2.16

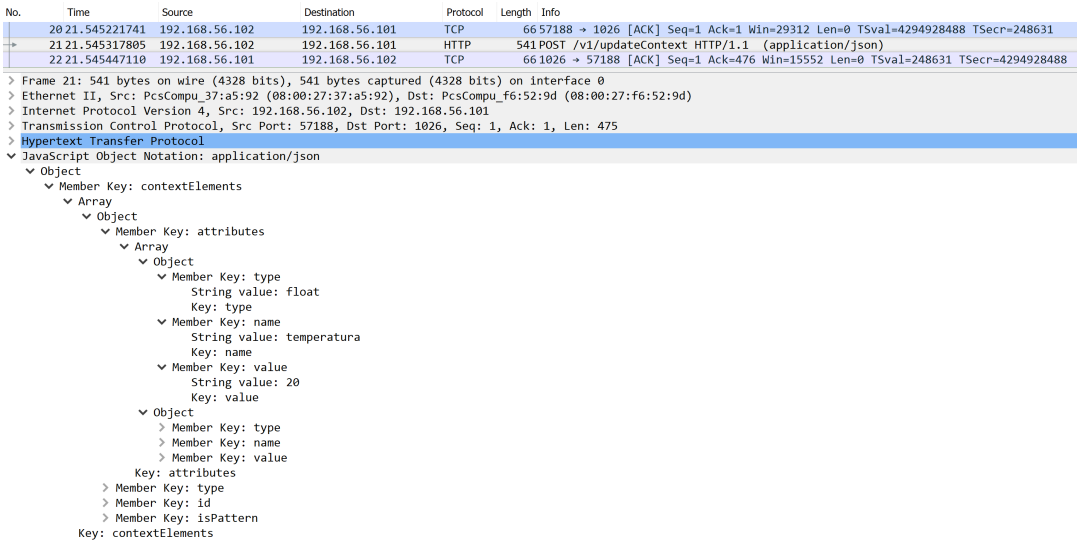


FIGURA 2.16: Captura de mensajes Fiware

A partir del código generado por este tutorial hicimos las modificaciones necesarias para hacer una prueba de la suscripción a eventos. Para esto levantamos en *http://<ip_local>:<puerto>/accumulate* un servicio REST que funciona como acumulador, y que será utilizado como callback de la suscripción. Esta url se pasa en el campo reference de la solicitud de suscripción.

Las figuras 2.17 y 2.18 muestran las capturas de los mensajes obtenidas a través de Wireshark para la solicitud de suscripción por parte del Context Consumer, y notificación por un evento de cambio por parte del Context Broker hacia el acumulador, respectivamente.

2.5.10. Tamaño de mensajes y payload

Si observamos el mensaje enviado al broker que se muestra en la figura 2.16 podemos ver que el tamaño total del mismo es de 541 bytes, que se reparten de la siguiente manera:

- 14 bytes de la cabecera de Ethernet
- 20 bytes de la cabecera IP
- 32 bytes de la cabecera TCP
- 232 bytes de la cabecera HTTP

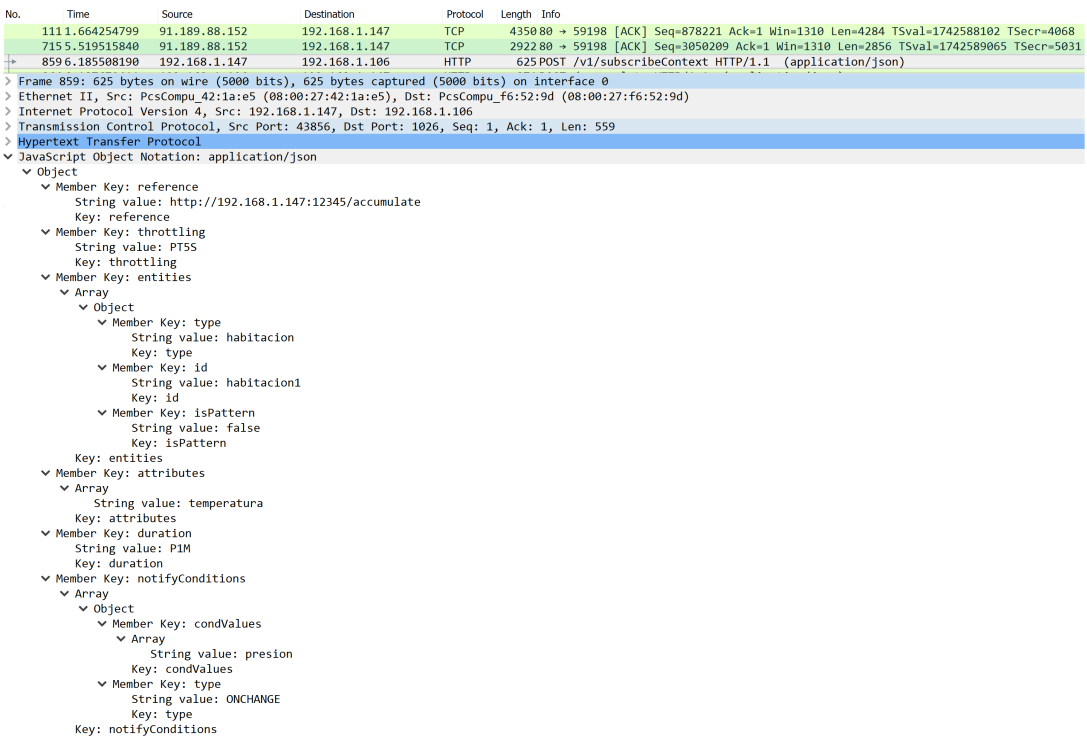


FIGURA 2.17: Solicitud de suscripción realizada al Context Broker

- 243 bytes de payload

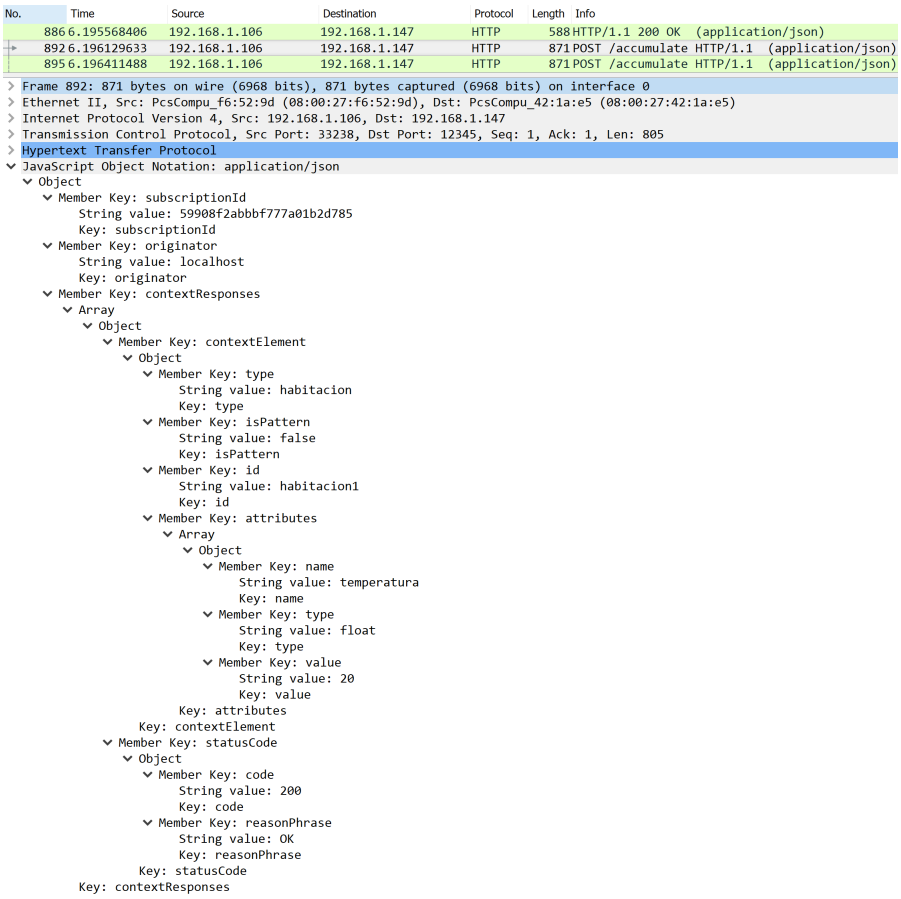


FIGURA 2.18: Notificación enviada por Context Broker al Context Consumer a partir de un evento de cambio en la temperatura

El overhead que tiene el mensaje es más del doble del tamaño total del payload del mismo. La mayor parte de este overhead está dado por los cabezales de HTTP. Este overhead no se puede evitar debido a que el Context Broker se implementa como una API REST y por lo tanto es obligatorio el uso de HTTP como protocolo de comunicación con el mismo.

Por otro lado, el contenido del payload se podría reducir (por ejemplo, utilizando nombres de atributos más cortos, o utilizando el formato establecido por el protocolo Ultralight 2.0). Esta reducción se podría aplicar pero sólo hasta cierto punto, ya que el mensaje debe seguir la estructura del modelo NGSI, y por lo tanto hay partes del payload que no es posible modificar.

Implementamos las mismas funcionalidades del tutorial para crear y modificar entidades pero utilizando la versión 2 del broker y observamos que los tamaños de los mensajes son significativamente menores. A partir de la figura 2.19 podemos ver que el tamaño total del mensaje enviado para la modificación de una entidad es de 349 bytes, que se reparten de la siguiente manera:

- 14 bytes de la cabecera de Ethernet
- 20 bytes de la cabecera IP
- 32 bytes de la cabecera TCP
- 232 bytes de la cabecera HTTP

- 49 bytes de payload

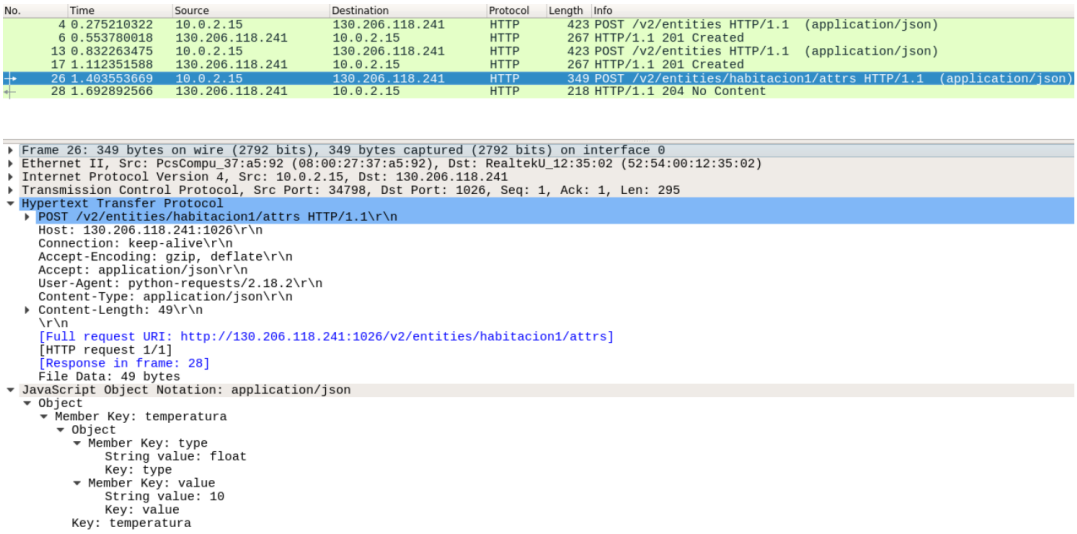


FIGURA 2.19: Notificación enviada por Context Broker al Context Consumer a partir de un evento de cambio en la temperatura, utilizando la versión 2 del broker y NGSI10

2.6. Resumen

En la tabla 2.6 se muestran de forma resumida algunas de las características principales de cada plataforma descritas a lo largo de este capítulo. A partir de la misma se puede obtener un panorama general de cada una de las plataformas.

Los aspectos que se comparan en la tabla (autenticación, modelo de negocio, precio, almacenamiento, etc) son los que a nuestro criterio son los más relevantes a la hora de generar una idea a grandes rasgos de cada plataforma. Otros aspectos importantes a tener en cuenta para obtener una comparación más profunda pueden ser el modo de despliegue (cloud, on premise), compatibilidad con dispositivos certificada, herramientas adicionales, especialización de la plataforma, entre otras.

Cabe destacar que, si bien las plataformas brindan herramientas orientadas al análisis de datos históricos, la información recabada sobre análisis de datos que se muestra en la tabla hace referencia únicamente a algunas de las herramientas disponibles para el análisis de datos en tiempo real. Esta decisión se tomó debido a que en el prototipo (capítulo 3) solo se utilizan herramientas de análisis de datos en tiempo real, por lo que la investigación de las herramientas orientadas a otros tipo de análisis de datos no fue abordada.

	Azure	IBM Watson	Amazon	KAA	FIWARE
Protocolos	HTTP, AMQP, MQTT y protocolos a medida usando Protocol Gateway	HTTP, MQTT	HTTP, WebSockets y MQTT, además de protocolos personalizados. Los dispositivos pueden comunicarse entre sí aunque utilicen protocolos distintos	HTTP y MQTT, aunque puede soportar virtualmente cualquier protocolo de transporte de datos	HTTP (API NGSI)
SDK/Lenguaje	.Net y UWP, Java, C, NodeJS	C#, C, Python, Java, NodeJS	C, Node.js, Arduino Yún. Aunque como los SDK son open source, pueden ser migrados a los lenguajes y las plataformas de hardware	C, C++, Java y Objective-C	Soporta cualquier lenguaje que permita realizar pedidos HTTP
Modelo de negocio	Comercial	Comercial	Comercial	Open source	Open source
Precio	Paga por tráfico de datos, almacenamiento y otros servicios contratados	Pago por número de dispositivos, tráfico de datos y almacenamiento	Paga por tráfico y publicación de datos	Gratuita	Gratuita
Autenticación	Por dispositivos con SAS token	Por dispositivo, con token	Certificado X.509 con autenticación cliente, IAM Service, Cognito Service	Utiliza encriptación híbrida basado en RSA y AES. Durante el inicio de una nueva sesión, cada endpoint envía una clave de sesión cifrada y firmada digitalmente. Para autenticar un endpoint y validar su solicitud, el nodo Kaa valida la firma digital de la clave de sesión	Por defecto no tiene, pero se pueden utilizar otros GE para manejar aspectos de autenticación
Seguridad	TLS (Server Authentication)	Por defecto TLS, pero se pueden utilizar otros servicios brindado por IBM Cloud referentes a seguridad.	TLS (Mutual Authentication)	En la implementación de transporte predeterminada, la comunicación de datos entre el SDK y el servidor Kaa se asegura usando cifrado híbrido con RSA y AES. Además dicha implementación es personalizable.	Por defecto no tiene, pero se pueden utilizar otros GE para manejar aspectos de seguridad.
Almacenamiento	Azure Blob Storage. Puede almacenar grandes cantidades de datos de objetos no estructurados, como datos de texto o binarios	Cloudant NoSQL DB (versión gratuita). No es posible almacenar en más de una BD. No hay control sobre la conexión con Watson IoT	AWS Lambda, Amazon S3, Amazon Machine Learning y Amazon DynamoDB, Amazon CloudWatch y Amazon Elasticsearch Service	Bases de datos SQL (MariaDB y PostgreSQL) y NoSQL (Apache Cassandra y MongoDB)	HDFS, MySQL, CKAN, MongoDB, Kafka, DynamoDB, PostgreSQL y Carto. Posibilidad de almacenamiento simultáneo en más de una BD. Control sobre la conexión con Orion Broker
Análisis de datos	Azure Stream Analytics, un motor de procesamiento de eventos para configurar cálculos analíticos en tiempo real sobre datos destreaming. Los datos pueden proceder de dispositivos, sensores, sitios web, fuentes de redes sociales, aplicaciones, sistemas de infraestructura, etc	IBM Streaming Analytics basado en InfoSphere Stream. Muy potente, usado en otros contextos fuera de IoT. Implementación mediante una interfaz gráfica y por línea de código. Posibilidad de realizar un "debug" de lo implementado. Curva de aprendizaje elevada	Amazon Kinesis Data Analytics. Procesa de manera sencilla datos de streaming en tiempo real con SQL estándar sin tener que aprender a usar lenguajes de programación ni marcos de procesamiento nuevos	Utiliza Apache Flume (Hadoop, Spark, Storm, etc), por loque es capaz de transferir los datos que llegan a la plataforma a cualquier sistema de Stream Analytics que soporte Apache Flume.	Complex Event Processing (CEP) basado Proton. Comunicación mediante estandar NGSI. Orientado al uso con Orion Broker. Implementación mediante interfaz gráfica. No se puede realizar "debug" de lo configurado. Curva de aprendizaje baja

TABLA 2.6: Principales características de las plataformas estudiadas

3 Prototipo

El propósito de este capítulo es describir y explicar el prototipo que se desarrolló para este proyecto. Lo que se buscó con este prototipo es la implementación de una posible solución IoT para dos plataformas distintas: IBM Watson y FIWARE, y a partir de esto poder realizar una comparación de ambas, teniendo en cuenta facilidades y dificultades que se presenten a la hora de implementar una solución para un mismo problema en ambas plataforma.

3.1. Problema planteado

El problema está relacionado con el control de la calidad del aire en diferentes puntos del país. A través de la página de la Dirección Nacional de Medio Ambiente (DINAMA) [35] se obtuvieron datos reales de sensores que están midiendo la calidad del aire en distintos puntos del país.

A partir de estos datos se implementa una solución IoT. Se desarrolla un componente que simula ser los sensores, enviando los datos a las distintas plataformas. A medida que estos flujos de datos llegan se realiza un análisis en tiempo real para determinar si es necesario emitir una alerta o no en caso de que algún valor excediera los límites establecidos [36]. Estas alertas, al igual que los datos que llegan a las plataformas, son mostrados en tiempo real por un componente web. Tanto los datos que llegan como las alertas emitidas, son almacenados por la plataforma en una base de datos.

Los valores de concentración máxima en el aire permitida para cada contaminante se obtuvieron de los estándares elaborados en el marco del grupo GESTA Aire [36], y se pueden ver en la tabla 3.1. Para simplificar el desarrollo de la solución y las pruebas se utilizó una escala en la que 1 hora en la realidad planteada corresponde a 30 segundos en la solución diseñada, tanto como para mandar los mensajes como para calcular el tamaño de las ventanas. Como los valores de los datos de la DINAMA corresponden a un dato por hora se agregó una espera de 30 segundos entre el envío de un mensaje y otro.

Cabe mencionar que algunos de los datos provenientes de la DINAMA tuvieron que ser modificados (aumentando su valor) para que excedieran los límites y de esta manera poder ver el funcionamiento de las alertas. Con el objetivo de obtener al menos una alerta por contaminante todos los archivos de datos fueron modificados. Teniendo en cuenta los valores límite por contaminante en el periodo de tiempo indicado se decidió calcular un promedio de los valores que llegaban a la plataforma en una ventana correspondiente a dicho periodo.

A modo de ejemplo consideremos la secuencia de mensajes de la figura 3.1, en donde cada número representa un mensaje recibido. Supongamos que cada mensaje llega

	Período real	Período escalado	Concentración límite ($\mu g/m^3$)
Monóxido de carbono (CO)	8 horas	4 minutos	10000
Dióxido de nitrógeno (NO_2)	1 hora	30 segundos	200
Ozono (O_3)	8 horas	4 minutos	100
Dióxido de azufre (SO_2)	24 horas	12 minutos	125
PM10	24 horas	12 minutos	100
Azufres reducidos totales	1 hora	30 segundos	15

TABLA 3.1: Concentraciónes límites por contaminante

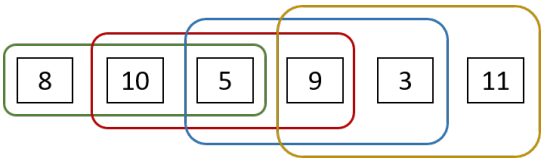


FIGURA 3.1: Ejemplo de ventanas utilizadas

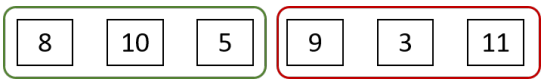


FIGURA 3.2: Ejemplo de ventanas no superpuestas

a la plataforma cada una hora, y que las ventanas de tiempo son de 3 horas, por lo tanto en el caso normal cada ventana incluirá 3 mensajes. En la figura se pueden ver las diferentes ventanas en diferentes colores. Tal como se ve en la figura cada vez que llega un nuevo mensaje se crea una nueva ventana. De esta forma se controla que el promedio de los valores incluidos en una ventana no supere el límite indicado. Si las ventanas se crearán al término de la ventana anterior como se muestra en la figura 3.2 se perderían valores promedios y por lo tanto la solución no sería correcta. En la solución planteada el tamaño de la ventana para cada contaminante se corresponde con el valor "Periodo escalado" de la tabla 3.1.

En las siguientes secciones se describe la solución planteada para cada plataforma.

En ambos casos, las distintas mediciones de los sensores y las alertas que se generan a partir de cada plataforma se visualizan mediante una aplicación web que llamamos AppIoT. La misma está implementada en Angular.

3.2. Fiware

Los GE de Fiware utilizados (Orion Context Broker, CEP y Cygnus con su respectiva base de datos) se encuentran hosteados en la nube de Fiware, por lo que para poder utilizarlos primero es necesario el registro en FI-LAB [37]. Luego de realizado el registro en necesario hacer una serie de pasos antes de poder lanzar una instancia de algún componente:

1. Crear una clave privada (.pem) que luego se asociará a una instancia y se utilizará para establecer una conexión externa con dicha instancia (por ejemplo conectarse por ssh).

2. Crear un grupo de seguridad, configurando los puertos que utilizará la instancia.

3. Crear una IP pública en donde estará hosteada la instancia.

Una vez que se tiene todo se procede a crear las instancias del Orion Context Broken y del CEP, asociandoles la clave, el grupo de seguridad y la IP pública previamente creadas. Vale aclarar que Fiware Lab brinda una única IP pública, por lo que solamente se podrá tener desplegada una instancia por cuenta. No es necesario crear una instancia de Cygnus ya que este componente viene incluido en la instancia del Context Broker así como también una base de datos MongoDB.

Para el correcto funcionamiento y comunicación de los distintos GE fue necesario utilizar la última versión de cada uno de ellos. Para actualizar el Context Broker corrimos desde la terminal el comando *yum install contextBroker* obteniendo la versión 1.7.0. De manera similar se realiza la actualización de Cygnus utilizando el comando *yum install cygnus*. Por último para actualizar el CEP es necesario descargar los .war actualizados del repositorio [38], actualizarlos en la carpeta de Tomcat y reiniciarlo para ver los cambios.

3.2.1. Diseño

En la figura 3.3 se muestra un diseño del prototipo implementado para la plataforma Fiware, en el cual se puede apreciar cómo se comunican los Generic Enablers descritos anteriormente en la sección 2.5.

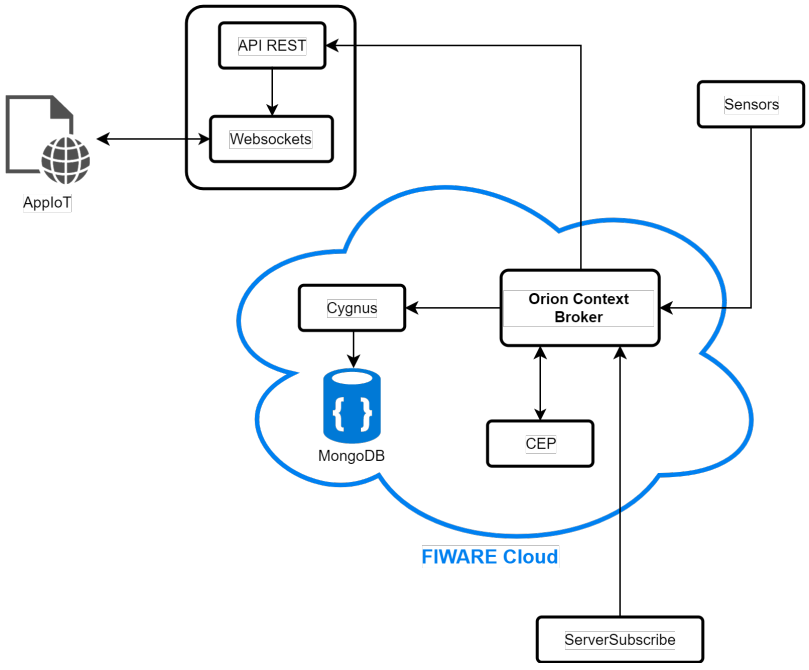


FIGURA 3.3: Arquitectura del prototipo implementada utilizando Fiware

En el caso de esta plataforma fue necesario la implementación de un servidor REST, que se encarga de recibir los mensajes y alertas enviados por el Context Broker y luego realizar un broadcast vía websockets a todas las conexiones abiertas que tenga el mismo. El cliente web es el que mantiene una conexión de websockets con nuestro servidor, y de esta forma recibe en tiempo real todos los mensajes enviados por el Context Broker para mostrarlos en pantalla de forma amigable al usuario.

El componente Sensors es el encargado de simular los sensores y de enviar los datos

al Context Broker. Está implementado en Java y toma los datos a enviar de archivos CSV que fueron descargados del sitio web de la DINAMA. Existe un CSV con los datos del aire para cada planta. Se definió que cada planta representará una entidad, y que cada uno de los contaminantes de los cuales se tienen datos se corresponderá con un atributo de dicha entidad. Por lo tanto por cada fila del CSV se enviará un POST al endpoint *<ip>:<puerto>/v2/entities/<entity_ID>/attrs* con todos los valores de los atributos de dicha entidad. El componente utiliza hilos que se encargan de enviar los datos. Se crea un hilo por cada sensor simulado; de esta forma el envío de datos se asemeja más a la realidad, en donde los mensajes de los sensores llegarán intercalados a la plataforma.

El componente ServerSubscribe se utiliza para crear las suscripciones a los datos que genera cada sensor, pasándole la url al servicio REST al que se notificará cuando ocurran cambios; en nuestro caso es la direccion donde esta corriendo el servidor REST. Para esto se envía un POST al endpoint *<ip>:<puerto>/v2/subscribe* como se mencionó anteriormente. Idealmente estas suscripciones las debería hacer el cliente web, que es el que realmente desea suscribirse a dichos datos, pero por problemas de CORS fue necesario crear un componente aparte para dicho fin. Dada la existencia de este componente, se decidió utilizarlo también para crear las suscripciones del CEP y Cygnus al Context Broker.

El flujo de funcionamiento del prototipo entonces es el siguiente:

1. Se crean las suscripciones correspondientes al cliente web, el CEP y Cygnus.
2. Se crea una entidad por cada sensor y luego cada uno comienza a enviar secuencialmente datos de sus atributos al Context Broker.
3. Una vez que los datos llegan al Context Broker el mismo los reenvía a todos los suscriptores que corresponda:
   a) El cliente web recibe esos datos por websockets y los muestra en pantalla.
   b) El CEP recibe los datos y en base al contexto y los patrones definidos envía alertas al Contex Broker cuando corresponda. Una vez que el Context Broker las recibe las vuelve a reenviar a todos sus suscriptores.
   c) Cygnus recibe todos los datos y los almacena en la base de datos previamente configurada.

3.2.2. Cygnus

Para configurar este componente se utilizan dos archivos ubicados en */usr/cygnus/conf*: agent_iot.conf y cygnus_instance_iot.conf. El nombre de estos archivos tiene que seguir el formato agent_<ID>.conf y cygnus_instance_<ID>.conf respectivamente. Esto permite que puedan existir varias instancias de Cygnus y cada par de archivos configura cada instancia identificada por un ID.

Una observación importante es que actualmente la última versión del Context Broker todavía no es compatible con la última versión de Cygnus, por lo que a la hora de realizar la suscripción de Cygnus al broker es necesario agregar el campo *"attrsFormat"* con valor *"legacy"* que permite que los mensajes que se envíen a la url de referencia especificada respeten el formato de la versión anterior del broker.

3.2.2.1. Configuración

En el archivo agent_iot.conf se configuran los componentes de la arquitectura de Cygnus (source, channel y sink). Como ya se mencionó anteriormente se utiliza una base MongoDB para almacenar los datos, en la figura 3.4 se muestra el diagrama de la configuración para el prototipo implementado.

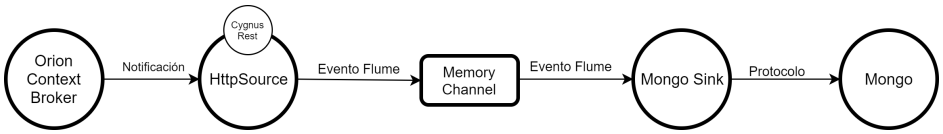


FIGURA 3.4: Arquitectura de Cygnus implementada en el prototipo

Estos tres componentes se definen en el archivo agent_iot.conf a través de los campos:

```
cygnusagent.sources = http-source
cygnusagent.sinks = mongo-sink
cygnusagent.channels = mongo-channel
```

El nombre de estos no tiene porque ser el descrito arriba sino que puede usarse cualquiera, siempre que se mantenga la misma coherencia de nombres en todo el archivo.
Otras líneas importantes de este archivo son:

```
cygnusagent.sources.http-source.port = 5050
cygnusagent.sources.http-source.handler.notification_target = /notify
```

De esta forma se define el puerto y la ruta en donde estará escuchando Cygnus para recibir los datos mediante métodos POST y guardarlos en la base. Esta dirección es la que es usada por el ServerSuscribe para suscribir a Cygnus a los cambios en el Context Broker. Las siguientes líneas son utilizadas para especificarle al sink cual es el channel que debe usar y la dirección de la base de datos.

```
cygnusagent.sinks.mongo-sink.channel = mongo-channel
cygnusagent.sinks.mongo-sink.mongo_hosts = localhost:27017
```

En cuanto al channel, las siguientes líneas son usadas para indicar la capacidad en bytes del mismo y la cantidad máxima de bytes que puede transferir por transacción.

```
cygnusagent.channels.mongo-channel.capacity = 1000
cygnusagent.channels.mongo-channel.transactionCapacity = 100
```

Por último, para poder levantar Cygnus se ejecuta el siguiente comando en consola:

```
/usr/cygnus/bin/cygnus-flume-ng agent --conf /usr/cygnus/conf/ -f /usr/
    cygnus/conf/agent_iot.conf -n cygnusagent -Dflume.root.logger=INFO,
    console
```

3.2.3. CEP

Para configurar el CEP se utiliza la interfaz gráfica como se mencionó anteriormente. La url para acceder a dicha interfaz es *http://<IP_Instancia_CEP>:8080/AuthoringTool*

Un proyecto CEP está construido a partir de las definiciones de los eventos, de los productores y consumidores, de los contextos y de la definición de EPAs (Event Processing Agents). El conjunto de estas definiciones crea lo que se conoce como una aplicación EPN (Event Processing Network). Para el caso del prototipo planteado no fue necesario la definición de productores, ya que estos son útiles en los casos en que se desea que la propia aplicación EPN sea la que busque los datos, por ejemplo, leer los datos de un archivo.

En una primera instancia se proceden a definir los eventos de entrada y salida que tendrá la aplicación. En la figura 3.5 se muestra la definición del único evento de entrada utilizado y en la figura 3.6 la definición del evento de salida.

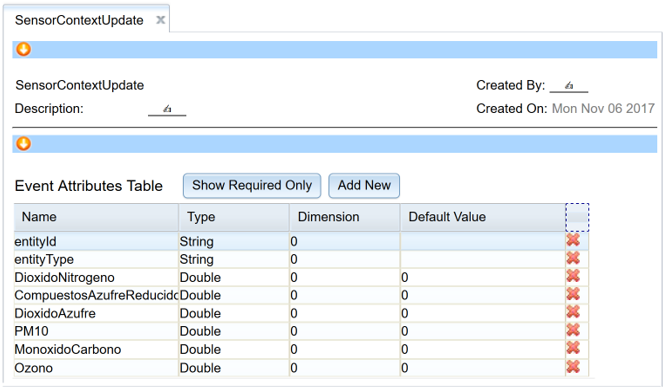


FIGURA 3.5: Evento de entrada SensorContextUpdate

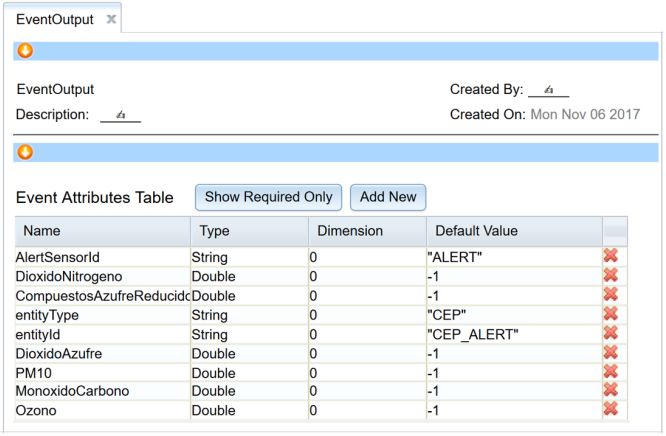


FIGURA 3.6: Evento de salida EventOutput

Para que la comunicación entre el Context Broker y el CEP sea efectiva, a la hora de definir el evento de entrada se tienen que tener ciertas consideraciones. Obligatoriamente el evento de entrada tiene que tener dos atributos de nombre entityId y entityType. Además el nombre del evento tiene que ser <entityType>ContextUpdate.

Los otros atributos que están definidos se corresponden con los mismos nombres de los contaminantes que son enviados en las notificaciones del Context Broker.

En el caso del evento de salida también se deben definir los dos atributos entityId y entityType. Esto se debe a que este evento es enviado al Context Broker para que sea persistido por Cygnus y que a su vez sea el broker quien notifique los clientes sobre esto, ya que al producirse un evento de salida quiere decir que se produjo una alerta en algunos de los contaminantes. Por defecto, como se muestra en la figura, los valores de los contaminantes es -1 y solo el contaminante que produjo la alerta es quien va a tener un valor positivo (valor que hizo saltar la alerta). También existe una campo AlertSensorId que se usa par especificar el ID del sensor que produjo la alerta. Estos valores son configurados en el EPA, descrito más adelante.

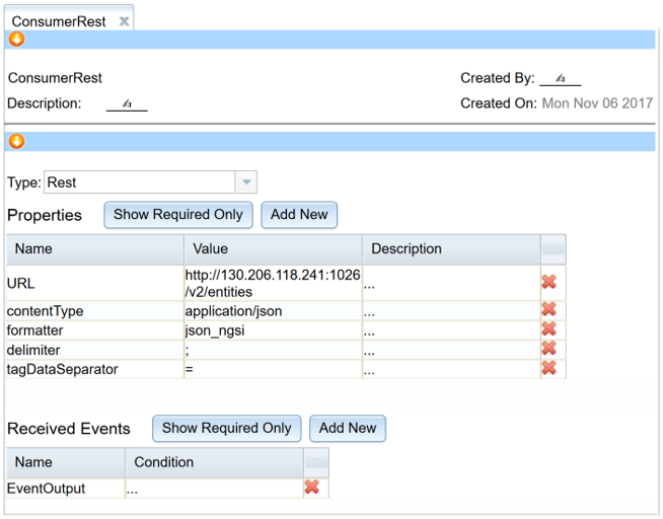


FIGURA 3.7: Consumidor ConsumerRest

La figura 3.7 muestra la definición del consumidor, en este caso es del tipo REST pero también podría ser un archivo en donde se guardan los eventos de salida asociados. La URL se corresponde con la URL del Context Broker utilizada para actualizar los valores de las entidades. El consumidor tiene asociado un evento de salida, por lo que, cada vez que se cree un evento de salida de ese tipo, esté se le enviará al consumidor definido.

La definición del contexto es compuesta, ya que se utiliza una definición de contexto temporal y otra de contexto segmentado. Para el contexto temporal se define una ventana de tiempo de la duración indicada en la tabla 3.1, que se crea con la llegada de cada nuevo mensaje, desde que se inicia la aplicación hasta que termine. Para la solución planteada se crearon tres contextos temporales: Alert1HourInterval, Alert8HourInterval y Alert24HourInterval. En la figura 3.8 se muestra la configuración de Alert1HourInterval; los demás son análogos cambiando el valor del campo "Relative Time Terminator".

La definición del contexto segmentado se hace por el atributo entityID del evento de entrada. En conclusión, el contexto compuesto es una ventana de tiempo y los eventos de entrada que caen en esta ventana son agrupados por el ID del sensor.

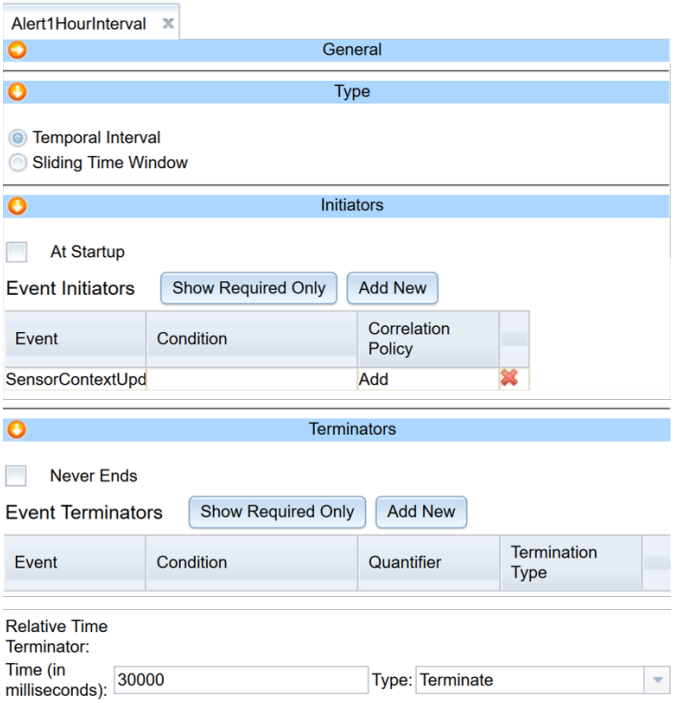


FIGURA 3.8: Alert1HourInterval

Por último se definen los EPAs, que son los que hacen uso de todas las definiciones de los componentes anteriores. En este caso se creó un EPA por cada contaminante y cada uno es del tipo Aggregate. En la figura 3.9 se muestra esta definición, para un EPA definido, así como también la especificación del contexto, descrito anteriormente.

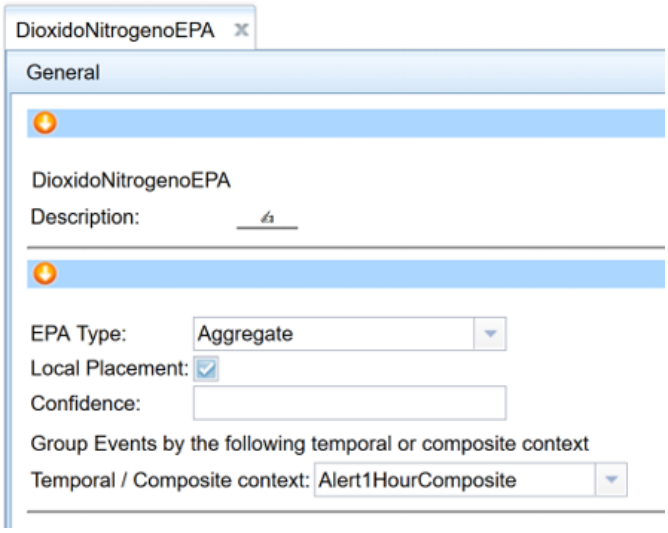


FIGURA 3.9: Evento de entrada DioxidoNitrogenoEPA - Sección General

En la figura 3.10 se muestra la definición de los eventos de entrada que serán tenidos en cuenta por el EPA, en este caso SensorContextUpdate. También se definen las variables computadas que se utilizaran; en este caso se quiere saber el promedio de los valores del contaminante por lo que se define una variable *Promedio* del tipo

Average en la que se guardará el promedio para el atributo *DioxidoNitrogeno*.

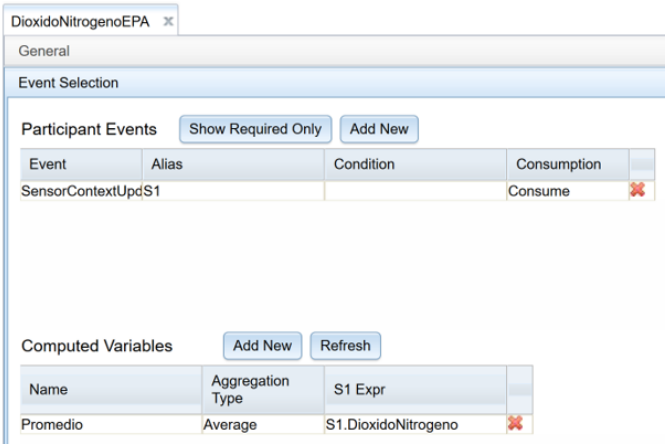


FIGURA 3.10: Evento de entrada DioxidoNitrogenoEPA - Sección EventSelection

Posteriormente, en la sección *condition* dentro del EPA (figura 3.11), se define la condición que se tiene que cumplir para que se ejecute el evento de salida. Para este caso se define que la variable *Promedio*, definida en el paso anterior, sea mayor que un cierto valor establecido y que esta condición sea evaluada al final de la ventana del contexto temporal (Evaluation policy: deferred).

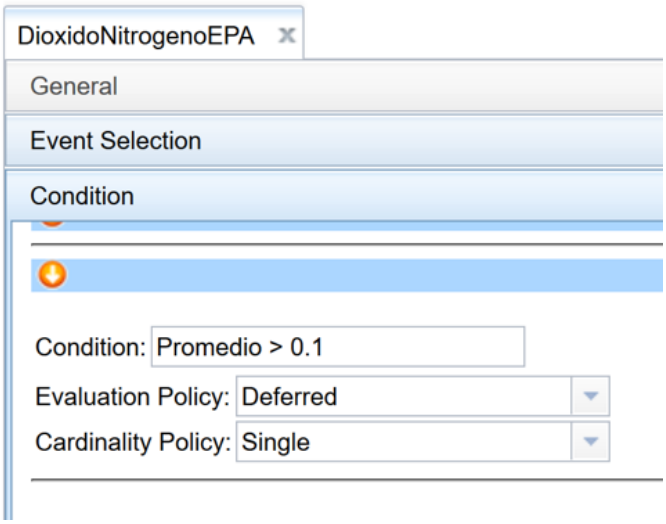


FIGURA 3.11: Evento de entrada DioxidoNitrogenoEPA - Sección Condition

Por último, se configura la sección Derivation (figura 3.12), en la que se establecen los eventos de salida que se ejecutarán y los valores que estos tendrán. En el atributo correspondiente al contaminante para el cual se está definiendo el EPA, se le asigna la variable Promedio, y además en el atributo AlertSensorId se le asigna el ID del sensor para el cual se realiza la alerta, que se obtiene del contexto que se definió.

Luego de haber configurado todos los patrones y diferentes aspectos del CEP se procede a exportar dicho proyecto para desplegar el CEP. Para esto se hace click en

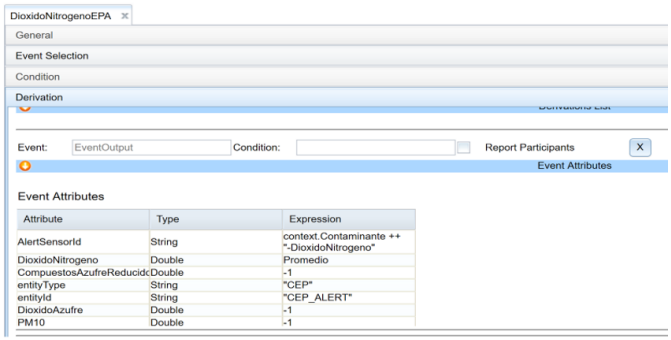


FIGURA 3.12: Evento de entrada DioxidoNitrogenoEPA - Sección Derivation

Save and Export y luego *Export to external Repository*. A continuación se le asigna un nombre y una url en donde desplegarlo. De esta forma se almacena la definición del proyecto en el repositorio externo, el cual es accesible mediante un GET [39] a

```
http://<IP_Instancia_CEP>:8080/ProtonOnWebServerAdmin/resources/
    definitions
```

En la respuesta de dicho request se encuentra la url de la definición del proyecto (URL_DefPrototipo). A continuación se hace un PUT a

```
http://<IP_Instancia_CEP>:8080/ProtonOnWebServerAdmin/resources/instances/
    ProtonOnWebServer
```

con el siguiente payload:

```
{
  "action": "ChangeDefinitions",
  "definitions-url": <URL_DefPrototipo>
}
```

Por último, para levantar el CEP es necesario realizar un PUT a

```
http://<IP\_Instancia\_CEP>:8080/ProtonOnWebServerAdmin/resources/
    instances/ProtonOnWebServer
```

enviando como payload:

```
{
  "action": "ChangeState",
  "state": "start"
}
```

Para pararlo tambien se debe realizar un PUT al mismo endpoint en el payload:

```
{
  "action": "ChangeState",
  "state": "stop"
}
```

3.3. IBM

3.3.1. Diseño

Para simular los sensores y enviar los mensajes desde los mismos se utiliza, al igual que en el caso de Fiware, la aplicación Sensors. En este caso los mensajes viajan utilizando el protocolo MQTT, por lo que es necesario crear una conexión MQTT

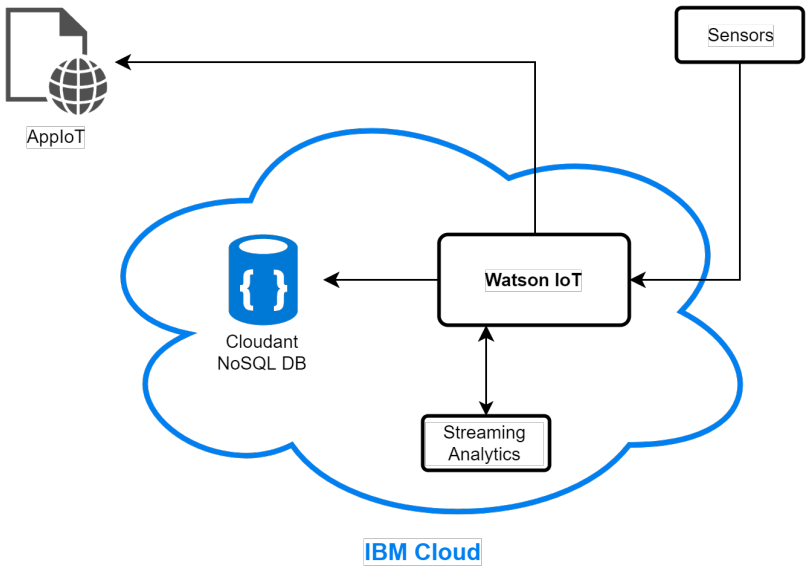


FIGURA 3.13: Arquitectura del prototipo implementada utilizando IBM

entre Sensors e Watson IoT, por cada sensor. Utilizando esta conexión y por medio de una suscripción a un tópico específico, se envian los mensajes a Watson. El tópico en el que se publican los mensajes es ïot-2/evt/devicestatus/fmt/jsonz en él se definen el tipo de evento y el formato del mensaje.

Cada mensaje que se envía tiene la siguiente estructura:

```
{
  "d": {
    <CONTAMINANTE>: <VALOR>,
    <CONTAMINANTE>: <VALOR>,
    ...
    <CONTAMINANTE>: <VALOR>
  }
}
```

En el caso de IBM la aplicación web AppIoT recibe los mensajes de Watson IoT utilizando la libreria iot-nodejs [40], recomendada en la documentación de Watson. Dentro de los módulos que ofrece esta librería utilizamos el módulo Application-Client que se encarga de conectar aplicaciones a la plataforma Watson IoT. En primer lugar se crea una instancia de Application Client pasandole al constructor un JSON con, al menos, los siguientes campos:

1. org: el ID de la organización
2. id: el ID de la aplicación dentro de la organización
3. auth-key: API key
4. auth-token: API key token
5. domain: (Optional) es la URL de la plataforma Watson. Por defecto el valor es el servidor de produccion de Watson IoT ïnternetofthings.ibmcloud.com".

En nuestro caso creamos la instancia de la siguiente forma:

```
const appClient = new Client.IotfApplication({
  'org': 'ita3c6',
  'id': 'AppIoT',
  'domain': 'internetofthings.ibmcloud.com',
  'auth-token': 'sa2uvkI&QHLK+YQd@U',
  'auth-key': 'a-ita3c6-1cyq1gc2tt'
});
```

Luego para conectarse y suscribirse a los eventos de los dispositivos ejecutamos el siguiente código:

```
appClient.connect();
appClient.on('connect', () => {
  appClient.subscribeToDeviceEvents();
});
```

Los mensajes se envían desde Sensors y llegan a Watson IoT. Desde aqui se envian mediante la conexión establecida al cliente web AppIoT, y además se almacena en la base de datos Cloudant NoSQL DB. A su vez son analizados por la aplicación de Streaming Analytics, que es la encargada de enviar un nuevo mensaje a Watson en el momento en que se dispare una alerta. Esta alertar generada también se almacena en la base de datos Cloudant y se envía al cliente web AppIoT.

3.3.2. Watson IoT

Para utilizar Watson IoT en primer lugar es necesario crear una cuenta en IBM Cloud. A pesar de que la plataforma es paga, pudimos obtener una licencia gratuita por 6 meses utilizando nuestro correo electrónico estudiantil.

Luego de crear la cuenta es necesario lanzar el servicio de Watson IoT. Para esto hay que dirigirse al dashboard de IBM Cloud y hacer click en “Create resource” y a continuación elegir “Internet of Things Platform” (Watson). Una vez creado el servicio, desde el dashboard de IBM Cloud hacer click en el servicio recien creado y a continuación lanzar el mismo haciendo click en “Launch”. Esto nos redirige al dashboard de Watson IoT.

Para registrar los diferentes dispositivos es necesario seguir los pasos indicados en la documentación [41]. Una vez que se registra cada dispositivo se muestran algunos datos que son claves para que el dispositivo se pueda conectar con Watson (ver figura 3.14), por lo que es necesario guardarlos ya que hay algunos datos que luego no se pueden volver a ver. En caso de que estos datos se pierdan es necesario registrar nuevamente el dispositivo.

Para conectar una aplicación externa a Watson IoT, como es el caso de la aplicación web AppIoT y la aplicación de Streaming Analytics, es necesario generar una API key y un token. Para generarlos se deben llevar a cabo los pasos descriptos en la documentación [42]. Al igual que los datos mostrados luego del registro de los dispositivos, estos datos se deben guardar ya que una vez terminado el proceso de generación de las API keys y tokens no es posible consultar sus valores, por lo que en caso de que se pierdan es necesario volver a generarlos.

Organization ID	ita3c6
Device Type	Sensor
Device ID	Sensor01
Authentication Method	use-token-auth
Authentication Token	K!*SDsT!LdiyB)v&oE

FIGURA 3.14: Datos del registro de un dispositivo en Watson IoT

Para almacenar el histórico de los mensajes que pasan por Watson se utilizó la base de datos Cloudant NoSQL. Se eligió esta base de datos ya que esa la que estaba disponible de forma gratuita. Para crear dicha base se siguieron los pasos indicados en la documentación correspondiente [43].

3.3.3. Streaming Analytics

Para crear una instancia de Streaming Analytics [44] se selecciona desde el dashboard de IBM Cloud la opción "Streaming Analytics". Una vez creado se lanza haciendo click en "Launch", lo que redirige a la consola de Stream, en donde se pueden enviar y administrar los trabajos (jobs).

Para poder utilizar InfoSphere Streams Studio hay dos opciones [45]:

1. descargar y levantar una maquina virtual con todo lo necesario, o
2. instalar cada componente por separado en un sistema operativo linux.

Por simpleza se eligió utilizar la máquina virtual.

Para poder utilizar el proyecto realizado es necesario exportarlo como .sab (Streaming Application Bundle). Este archivo se debe subir a la consola de Stream como un trabajo nuevo, y luego de esto el mismo queda corriendo siempre y cuando la instancia este ejecutandose.

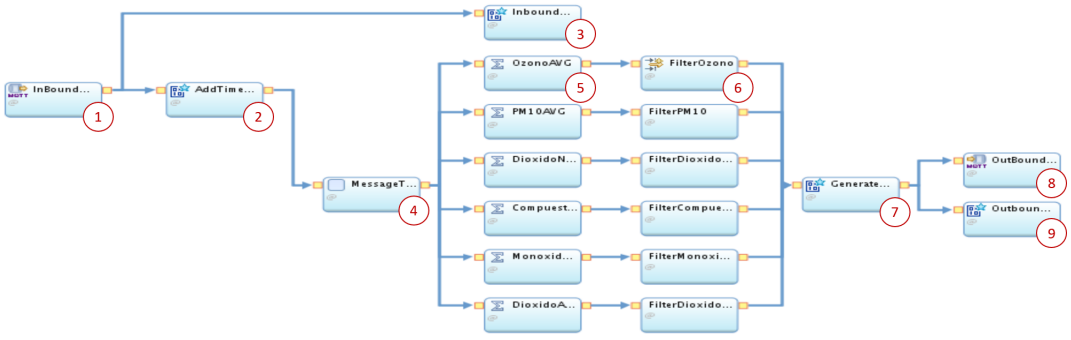


FIGURA 3.15: Diagrama de la solución realizada con Infosphere Stream Studio

En la figura 3.15 se puede observar el diagrama de la aplicación que se utiliza para el procesamiento de eventos complejos relacionados con la detección de alertas que se utilizó en el prototipo. A continuación se explicará cada uno de los operadores utilizados.

El componente marcado con el número 1 corresponde con el operador InBoundMessage. Se trata de un Source Adapter, encargado de suscribirse a todos los tópicos de Watson IoT en los que los sensores publican datos para poder recibirlos. Para poder hacer esto se utiliza la API key y el token creados inicialmente. El operador marcado con el número 2 es InBoundMessageDbg cuya función es escribir logs para poder verlos luego en la consola de Stream. El operador 3 es del tipo Aggregate, y se encarga de agregar un campo timestamp a cada tupla indicando una marca temporal de llegada de la tupla a la plataforma. La misma se utilizará para el manejo de las ventanas. El operador número 4 se corresponde con MessageToTuple, que se encarga de transformar el JSON que recibe como entrada en una tupla. El número 5 es un operador del tipo Aggregate. Se crea un operador de este tipo por cada contaminante y cada uno de ellos está configurado para crear ventanas de tiempo del periodo especificado en la tabla 3.1, particionadas por el tipo de dispositivo. Una vez culminado el tiempo de la ventana se calcula el promedio de los valores que llegaron en la misma para dicho contaminante. El operador 6 es del tipo Filter, y se encarga de dejar pasar solo aquellas tuplas cuyo valor promedio sea mayor que un valor límite establecido. Las salidas de estos filtros son las entradas del operador GeneneratedMessage (número 7) que se encarga de generar el mensaje que se va a enviar al exterior. El operador OutBoundMessage (número 8) envía este mensaje al tópico "iot-2/type/AlertDevice/id/AlertDevice01/evt/alert/fmt/json" de Watson. Para esto también utiliza la API key y token creados al principio. El operador OutBoundMessageDbg (número 9) se utiliza como debugger.

3.4. Comparación de plataformas utilizadas

A continuación se comparan, según nuestra opinión y experiencia, algunas de las características fundamentales de las dos plataformas más utilizadas a lo largo de esta investigación: IBM y Fiware. Ambas presentan ventajas y desventajas, pero según nuestro criterio no hay una mejor que la otra en términos generales, ya que depende del contexto en el que se deseen utilizar. En la tabla 3.2 se muestran de forma resumida los principales aspectos que se comparan a lo largo de esta sección.

En primer lugar podemos comparar el protocolo de comunicación que utiliza cada plataforma. Como ya se mencionó en capítulos anteriores todos los componentes pertenecientes a Fiware solo se comunican mediante HTTP usando el estándar NGSI. Por el otro lado, IBM permite la comunicación con Watson IoT mediante el protocolo MQTT o HTTP, según prefiera el usuario. Como ya se demostró en las demos, los tamaños de los mensajes MQTT son significativamente menores que los mensajes HTTP, esto significa que al utilizar mensajes MQTT la red no se sobrecarga tanto como al usar HTTP. Cabe mencionar que para poder utilizar MQTT con Fiware es necesario la utilización de un GE que haga de "traductor" entre un protocolo y otro, lo que ocasiona que se aumente el tiempo de procesamiento por cada mensaje.

En cuanto a la configuración de la base de datos, en el caso de IBM mediante unos simples pasos se puede lograr la conexión entre Watson IoT y el motor de base de datos deseado. En Fiware no es tan sencillo, primero hay que configurar el GE Cygnus para que haga de "puente" entre Orion Context Broker y el motor de base de datos deseado. Esta configuración puede no ser tan sencilla, no obstante una vez lograda permite un mayor control, por parte del usuario, sobre los flujos de datos que

	Watson IoT	Fiware
Protocolos	HTTP, MQTT	HTTP (API NGSI)
SDK/Lenguaje	C#, C, Python, Java, NodeJS	Soporta cualquier lenguaje que permita realizar pedidos HTTP
Seguridad	Por defecto TLS, pero se pueden utilizar otros servicios brindado por IBM Cloud referentes a seguridad.	Por defecto no tiene, pero se pueden utilizar otros GE para manejar aspectos de seguridad.
Autenticación	Por dispositivo, con token	Por defecto no tiene, pero se pueden utilizar otros GE para manejar aspectos de autenticación.
Precio	Pago por número de dispositivos, tráfico de datos y almacenamiento	Gratuito
Almacenamiento	Cloudant NoSQL DB(versión gratuita). No es posible almacenar en más de una BD. No hay control sobre la conexión con Watson IoT.	HDFS, MySQL, CKAN, MongoDB, Kafka, DynamoDB, PostgreSQL y Carto. Posibilidad de almacenamiento simultáneo en más de una BD. Control sobre la conexión con Orion Broker
Análisis de datos	IBM Streaming Analytics basado en InfoSphere Stream. Muy potente, usado en otros contextos fuera de IoT. Implementación mediante una interfaz gráfica y por línea de código. Posibilidad de realizar un "debug" de lo implementado. Curva de aprendizaje elevada.	Complex Event Processing (CEP) basado Proton. Comunicación mediante estandar NGSI. Orientado al uso con Orion Broker. Implementación mediante interfaz gráfica. No se puede realizar "debug de lo configurado. Curva de aprendizaje baja.

TABLA 3.2: Comparación de plataformas IBM y Fiware

son guardados. Permite guardar simultáneamente en distintos motores de base de datos y dependiendo de la configuración elegida se puede llegar a lograr una mejor performance al momento de guardar los datos. Esto no es posible en IBM.

El otro componente importante a comparar es el relacionado con el análisis de datos, más específicamente con el procesamiento de eventos complejos. En el caso de Fiware, este componente (CEP) es bastante sencillo. La configuración se hace mediante una interfaz gráfica sin la necesidad de escribir ninguna línea de código. Esto es una gran ventaja ya que no necesita que el usuario requiera de algún conocimiento previo sobre algún lenguaje. En el caso de IBM la utilización de InfoSphere Stream Studio requirió una curva de aprendizaje bastante elevada en un principio, la implementación se puede hacer también mediante una interfaz gráfica y/o por un editor de código. Cabe mencionar que esta herramienta es muy potente y no solo es usada en relación a IoT sino que es muy usada en otros áreas (Big Data, Social streaming, Commercial Stream Analytics, etc). Además, tiene la ventaja respecto al CEP en Fiware, que permite que se pueda ejecutar la aplicación de Streaming Analytics en modo debug, lo que permite una mejor comprensión de la dirección que toman los flujos de datos dentro de la aplicación, permitiendo encontrar rápidamente donde es que está el error si es que lo hay. Esto es de una gran ayuda cuando se esta enfrente del procesamiento de un evento complejo que maneja muchos flujos. En Fiware esto no es posible por lo que si algo falla se debe revisar toda la configuración y los logs de la aplicación hasta encontrar donde esta el error, pudiendo insumir un tiempo considerable.

Otro punto a mencionar es que IBM tiene a disposición distintas librerías para diversos lenguajes, tanto para el desarrollo a nivel del dispositivo como para el desarrollo de aplicaciones, lo cual es una ayuda importante para el desarrollador. Por ejemplo, a nivel de aplicación web, IBM proporciona una librería que permite rápidamente, en pocas líneas de código, la conexión y suscripción a los tópicos de Watson IoT y de esta manera obtener los datos en tiempo real. En el caso de Fiware es necesario la

implementación de un servidor intermedio entre la aplicación web y el Orion Context Broker, para poder brindar los datos por medio de web sockets, lo que conlleva mayor tiempo de implementación y de recursos.

En cuanto a la documentación de cada plataforma, en el caso de IBM se cuenta con una amplia documentación sobre cada uno de sus componentes. Además cuenta con diversos ejemplos de uso y tutoriales para cada uno de ellos y con distintos niveles de dificultad. También rápidamente se puede encontrar en la web ejemplos de código de otros usuarios que utilizan la plataforma. Por otra parte, con Fiware nos dió la sensación de que no es así, la documentación no siempre está actualizada, los ejemplos que presentan son muy básicos y no hay muchos ejemplos de otros usuarios en la web.

Respecto al modelo de negocio Fiware presenta una ventaja importante ya que es gratuito y open source.

En cuanto al modelo de datos, Fiware presenta la ventaja de contar con un estándar para el envío de datos entre sus componentes. De esta forma se facilita el reuso de componentes entre diferentes aplicaciones que usen dicho estándar. Por ejemplo si se quisieran reutilizar componentes o compartir datos entre dos soluciones IoT de Fiware las modificaciones que deberían hacerse probablemente serían mínimas, mientras que si los componentes a reutilizar y/o los datos a compartir pertenecen a soluciones IoT de IBM las modificaciones necesarias podrían no ser tan sencillas ya que IBM no cuenta con un estándar para el manejo de datos.

4 Pautas para el diseño de una solución IoT

A pesar de la gran cantidad de avances e investigaciones que se han llevado a cabo en esta área, aún no existen patrones o pautas estándares para llevar a cabo el desarrollo de soluciones IoT. A pesar de esto se pueden sacar algunas conclusiones en base a los patrones que más se repiten en diferentes soluciones desarrolladas e investigadas en diferentes artículos académicos[46][47][48].

Generalmente, en IoT, se hace uso de sensores y/o dispositivos con limitados recursos de almacenamiento y procesamiento los cuales pueden presentar problemas de confiabilidad, rendimiento, seguridad y privacidad. Por otro lado se tiene "la nube" (Cloud en inglés) la cual, virtualmente, posee recursos ilimitados en cuanto al almacenamiento y al poder de procesamiento, además es un paradigma bastante estudiado por lo que se puede decir que tiene algún grado de madurez y corre con la ventaja que tiene resuelto algunas de las preocupaciones en IoT (seguridad, procesamiento, confiabilidad, etc)[47]. Debido a estas características es que la sugerencia principal a la hora de diseñar soluciones IoT es utilizar Cloud Computing, conocido comúnmente como CloudIoT. Esto permite el acceso a recursos compartidos y configurables, ayudando entre otras cosas a que la solución sea fácilmente escalable. Sin embargo la mayoría de los ejemplos que utilizan Cloud Computing con IoT implementan soluciones sencillas o simples pruebas de concepto de las arquitecturas planteadas, lo que da la idea de que aún no existen soluciones de arquitecturas formalmente validadas con sólidas evidencias de eficiencia y efectividad. En la mayoría de los casos la arquitectura predominante para la transmisión de datos en la nube según los artículos analizados es REST.

Según el estudio *"Integration of Cloud Computing and Internet of Things: a Survey"*[47] existen áreas de aplicación que hacen uso de este paradigma para facilitar la creación de una solución IoT. Algunas de estas son:

- *Salud:* La utilización de CloudIoT representa una solución prometedora para administrar datos de sensores de salud de manera eficiente y permite abstraer detalles técnicos, eliminando la necesidad de experiencia o control sobre la infraestructura tecnológica.
- *Smart home:* posiblemente el ámbito doméstico es donde más existe una interacción del usuario con el entorno. CloudIoT puede permitir la interacción directa del usuario con sensores y/o actuadores, además, brinda la posibilidad de crear una interconexión entre los dispositivos del hogar, un control de estos dispositivos de manera remota y la automatización de estos.
- *Control energético:* CloudIoT hace posible analizar y procesar grandes cantidades de datos e información procedentes de diferentes dispositivos ubicados en

distintas zonas geográficas, con el fin de implementar un control inteligentes de estos dispositivos.

Además de que todavía no existen patrones de diseño definidos para desarrollar una solución IoT, dicho paradigma cuenta con varios desafíos por resolver. Por ejemplo, uno de los desafíos que presenta utilizar Cloud Computing con IoT es que si todos los datos generados por los distintos dispositivos IoT se transmiten por la red se generaría una congestión de la misma y una latencia en la comunicaciones que puede resultar intolerable para aplicaciones en tiempo real. Es por esto que es necesario proponer estrategias para minimizar la cantidad de recursos consumidos por aplicaciones IoT y decidir cuales son realmente los datos necesarios para evitar la congestión de la red. Para esto se pueden utilizar por ejemplo gateways inteligentes que realicen un preprocesamiento de los datos almacenando temporalmente los mismos de forma local al dispositivo antes de enviarlos a la nube.

Otro problema que presenta este paradigma es la estandarización de datos y servicios. La información producida por los dispositivos IoT no siguen ningún estándar por lo que utilizan diferentes formatos, unidades, etc. Manejar este tema es importante en el contexto de CloudIoT para permitir que las aplicaciones saquen un mayor provecho de los datos brindados por los dispositivos IoT y de la escalabilidad y disponibilidad de los servicios en la nube.

Otro tema muy importante a tener en cuenta es la seguridad y privacidad de los datos. Al utilizar CloudIoT los datos se almacenan en la nube, que puede ser gestionada por algún tercero, por lo que la externalización de estos datos puede traer una serie de problemas relacionados con la privacidad y acceso a los datos, ya que pueden ser manejados por terceros o utilizados indebidamente. Por lo que es necesario investigar los mecanismos destinados a proteger los datos obtenidos de los dispositivos de IoT y garantizar la seguridad al almacenarlos en la nube.

Además de la adopción de ciertos patrones para el desarrollo de soluciones IoT, es necesario analizar algunos puntos importantes del problema que se desea resolver para desarrollar la solución que mejor se adecue a la realidad. A continuación se describen algunos de estos puntos, algunos de los cuales son útiles para definir la plataforma a utilizar y otros son independientes de la misma.

En primer lugar se debe tener claro cuál es el problema que se desea resolver y qué datos se necesitan analizar. De esta forma se definen qué dispositivos se utilizarán como fuentes de datos (por ejemplo sensores), y al conocer qué es lo que se desea analizar es posible enviar y recibir solo los datos necesarios para dicho análisis, con el fin de no sobrecargar la red con datos inútiles. Esta consideración es independiente de la plataforma IoT que se utilice finalmente para resolver el problema planteado.

Una vez que se conoce que dispositivos se van a utilizar como fuentes de datos, un punto importante a tener en cuenta a la hora de elegir qué plataforma utilizar para una solución IoT es conocer la ubicación geográfica de los mismos, su interoperabilidad y qué protocolos soportan. A partir de esto se puede definir si será necesario utilizar gateways para conectar los dispositivos con la plataforma utilizada o si los dispositivos se podrán conectar directamente a la misma, ya que no todas las plataformas soportan los mismos protocolos y no todos los protocolos permiten la comunicación entre ellos.

Por otro lado, algo fundamental para la elección de la plataforma a utilizar es conocer la cantidad aproximada de mensajes que se van a enviar y recibir y el tamaño de los mismos. Hay plataformas pagas que basan su precio en la cantidad y/o tamaño de mensajes enviados y recibidos. Esto no sería un problema si se utiliza una plataforma open source. No obstante en el diseño de cualquier solución IoT es conveniente sobrecargar lo menos posible la red, sin importar qué plataforma específica se utilice.

También hay que tener en cuenta qué tan escalable se quiera que sea la solución, ya que si en un futuro se quiere agregar algún nuevo componente a la plataforma podría no ser algo trivial, o si el volumen de datos es muchos más grande que el esperado se podrían obtener resultados inesperados. En plataformas pagas tal vez se tenga que adaptar el nuevo componente a los servicios que ésta ofrece, mientras que en una open source el agregado de un nuevo componente puede ser modelado e implementado de la forma que se desee. Por otro lado, sería muy deseable hacer un estimativo del volumen de datos que podría recibir la solución y que tan importante es la velocidad de acceso a estos datos, ya que esto influye notoriamente en el sistema de base de datos que se desea usar, además hay que considerar que éste sea compatible con la plataforma elegida.

La seguridad es un tema muy importante a tener en cuenta. Una falla de seguridad podría significar que se tenga acceso a información sensible así como también podría ocasionar que se comprometa el funcionamiento de dispositivos críticos. En cuanto a la seguridad de datos, existen regulaciones a nivel de leyes que pueden prohibir que estos datos salgan del país, por lo que una solución IoT utilizando almacenamiento en servidores que no estén en el país no es posible. No obstante existen plataformas que permiten obtener una versión "in-house" para la instalación local de la misma.

También es importante tener en cuenta la madurez de la plataforma elegida. Debido a que IoT es un tema en pleno auge existe un gran número de plataformas disponibles para el desarrollo de soluciones IoT, pero muchas de ellas están todavía "inmaduras", es decir que no cuentan con suficiente documentación, ejemplos, ni soporte técnico, lo que dificulta el desarrollo de la solución. A su vez, es deseable contar un equipo de desarrollo capacitado para la implementación de la solución IoT. De todas formas hay que tener en cuenta que hay una curva de aprendizaje importante, y que debido a esto generalmente los proyectos de IoT llevan más tiempo que lo que se planifica inicialmente. Dependiendo del tamaño y requisitos de la solución a desarrollar se necesitarán personas con diferentes capacidades dentro del mismo equipo de desarrollo.

A medida que se vayan tomando algunas de estas decisiones es conveniente realizar pruebas piloto o prototipos para asegurarse que todo siga dentro de los parámetros establecidos, o para ir cambiando alguna de las decisiones tomadas. Las pruebas pueden ser por ejemplo de la red de sensores o de diferentes aplicaciones o componentes utilizados, entre otras.

5 Conclusiones

A lo largo de esta investigación se constató la importancia que está teniendo actualmente el Internet de las Cosas y la inmadurez existente en algunos aspectos claves del desarrollo de soluciones IoT. Uno de los aspectos todavía inmaduros en esta área es una arquitectura estándar para el desarrollo de soluciones. No obstante, en estos últimos años se ha visto una tendencia hacia el uso de el paradigma Cloud IoT, como se menciona en el capítulo 5, por lo que se cree que éste puede ser el paradigma predominante para el futuro inmediato.

Dado que IoT está en pleno auge están apareciendo continuamente nuevas plataformas en el mercado, por lo que existe una fuerte competencia entre ellas para lograr el liderazgo. Esto se debe en parte a que se estima que en el futuro cercano el mercado de IoT se incrementará considerablemente, por lo que invertir en el desarrollo de plataformas IoT podría dar muy buenos resultados. Esto hace que a la hora de tener que elegir una plataforma IoT para una solución particular se tenga un gran abanico de opciones dificultando la tarea de elección.

Con el objetivo de reducir en algún grado esta dificultad se investigaron algunas de las plataformas más populares en la actualidad, tanto open source como comerciales. En especial se hizo hincapié en dos de ellas: IBM y Fiware, con las cuales se desarrolló un prototipo para probar algunas de sus características y lograr hacer una comparación más profunda de las mismas. Gracias al desarrollo del prototipo se conocieron algunas ventajas y puntos débiles de ambas plataformas. Además se pudo probar la interacción de las plataformas con otros componentes, por ejemplo aquellos encargados de la identificación de eventos complejos y almacenamiento histórico de los datos.

Gracias a la implementación del prototipo se pudo observar que las plataformas comerciales generalmente son más “user-friendly” que las open source. Al menos para las dos plataformas utilizadas en el prototipo también se pudo observar que el tiempo del proyecto se reduce en el caso de utilizar plataformas comerciales, ya que éstas generalmente brindan librerías para que el desarrollador trabaje más cómodamente, y facilitan la comunicación de los diferentes componentes de la plataforma.

Por otra parte las plataformas open source le dan al desarrollador más libertad y control sobre la solución que está implementando, permitiéndole además, en caso que lo quisiera, realizar una personalización de la plataforma según los requisitos necesarios. Se debe tener en cuenta que para poder llevar a cabo dichas personalizaciones o correcciones de defectos en la plataforma podría ser de gran ayuda contar con personal capacitado.

6 Trabajo futuro

Como se mencionó a lo largo del trabajo los aspectos de seguridad tienen una importancia muy grande a la hora de desarrollar soluciones IoT, ya que se maneja una gran variedad y cantidad de datos que podrían ser sensibles o privados. Este tema no se investigó en este trabajo, pero sería interesante abordarlo en el futuro.

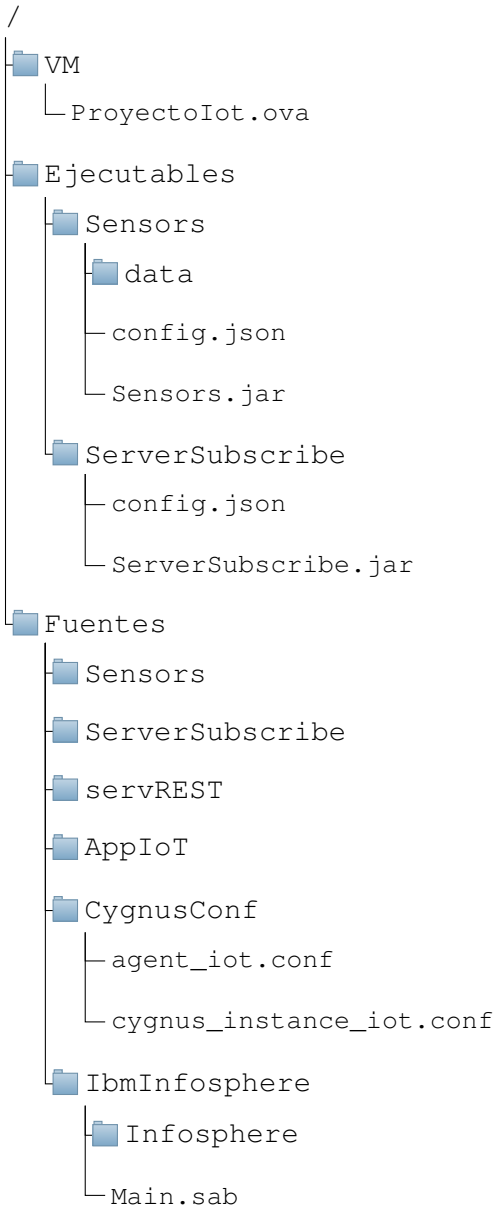
Dado que gran parte de las soluciones IoT están pensadas para manejar grandes volúmenes de datos, otro análisis interesante a realizar en un futuro es el comportamiento de las plataformas ante eventos de estrés de las mismas, como por ejemplo escenarios con un gran número de dispositivos conectados y enviado datos, desde diferentes ubicaciones geográficas, y utilizando una gran cantidad de tópicos y/o enviando una gran cantidad de datos a un mismo tópico. A su vez, analizar el comportamiento de la red cuando se envían estas grandes cantidades de datos, ya que podría ocurrir una degradación de la misma y esto podría influir en la performance de la solución IoT.

Por último, otro tema interesante a investigar son las herramientas disponibles en el mercado para realizar tanto machine learning como reportes y demás análisis y agregaciones de los datos obtenidos mediante IoT.

7 Anexo

Junto a este informe se entrega un archivo *prototipo.zip* que contiene el código fuente y una máquina virtual necesarias para correr el prototipo realizado en este trabajo. Para el caso de Fiware se utilizaron todos los componentes de forma local dentro de la VM.

Luego de descomprimir el archivo *prototipo.zip* se encuentra la siguiente estructura de directorios y archivos:



En los archivos de configuración (config.json) dentro de la carpeta Ejecutables se pueden cambiar datos como por ejemplo direcciones IP, puertos, credenciales, etc.

Para correr el prototipo correctamente se deben seguir los siguientes pasos en orden:

1. Levantar la máquina virtual configurando en la sección Red que esté conectado a Adaptador puente. Al hacer esto empiezan a correr diferentes procesos:
   a) ContextBroker (<IP_VM>:1026)
   b) CEP (localhost:8080/AuthoringTool/Main.html)
   c) servREST (localhost:9090/api/fiwareReference)
   d) AppIoT (<IP_VM>:9000/ibm y <IP_VM>:9000/fiware)
2. Dentro de la máquina virtual:
   a) Para ingresar, se puede utilizar el usuario *root* y contraseña *fiware*
   b) En caso de que se quiera probar IBM con otras credenciales ejecutar el comando

   ```
   nano /usr/share/nginx/html/AppIoT/assets/config.json
   ```

   y modificar los campos correspondientes con IBM.
   c) En caso de que se quiera probar Fiware ejecutar el comando

   ```
   nano /usr/share/nginx/html/AppIoT/assets/config.json
   ```

   y modificar el campo *fiware.websocketUrl* cambiando solamente la IP por <IP_VM>y guardar el archivo.
   d) Si se quiere probar Fiware y tener un almacenamiento histórico de los datos se debe activar el componente Cygnus. Para esto se debe ejecutar el comando

   ```
   ./../home/startCygnus.sh start.
   ```
3. Abrir en un navegador la aplicación web AppIoT (<IP_VM>:9000/ibm si se quiere probar IBM o <IP_VM>:9000/fiware si se quiere probar Fiware).
4. En caso que se quiera probar Fiware. Modificar en el archivo */Ejecutables/ServerSubscribe/config.json* solamente las IP de los campos *fiware.urlSubcription* y fiware.entityCreate por <IP_VM>
5. En caso que se quiera probar Fiware. Ejecutar *ServerSubscribe.jar* mediante el comando

   ```
   java –jar ServerSubscribe.jar
   ```

   Esperar a que termine la ejecución.
6. En caso de querer probar el procesamiento de eventos complejos con IBM es necesario crear una cuenta en IBM Cloud, crear un servicio Streaming Analytics y levantar el archivo */Fuentes/IbmInfosphere/Main.sab* como un trabajo. No fue posible dejar este servicio corriendo porque la cuenta gratuita solo permite 50 horas de ejecución gratuitas mensuales.

7. Modificar el campo *platform* en el archivo */Ejecutables/Sensors/config.json* para elegir la plataforma que se quiera probar (elegir entre *IBM* o *FIWARE*). Además, si se elige Fiware, modificar la IP del campo *fiware.url* por <IP_VM>. Si se quieren cambiar las credenciales utilizadas para IBM modificar los campos dentro del objeto *ibm* que se encuentra en dicho archivo de configuración.

8. Ejecutar *Sensors.jar* mediante el comando

   ```
   java –jar Sensors.jar
   ```

9. En la aplicación web comenzarán a llegar los mensajes y alertas.

Bibliografía

[1] RFC 7452. *Architectural Considerations in Smart Object Networking*. Última visita: junio 2018. URL: https://tools.ietf.org/html/rfc7452.

[2] Bain & Company. *Defining the Battlegrounds of the Internet of Things*. Última visita: junio 2018. URL: http://www.bain.com/publications/articles/defining-the-battlegrounds-of-the-internet-of-things.aspx#en2.

[3] *Compressing and optimizing Azure IoT Messages and network traffic*. Última visita: junio 2018. URL: https://kevinsaye.wordpress.com/2017/09/22/compressing-and-optimizing-azure-iot-messages-and-network-traffic/.

[4] *GZIP*. Última visita: junio 2018. URL: http://www.gzip.org/.

[5] P.Th. Eugster, P. Felber, R. Guerraoui, and A.M. Kermarrec. *The Many Faces of Publish/Subscribe*. Última visita: junio 2018. URL: https://www.cs.ru.nl/~pieter/oss/manyfaces.pdf.

[6] Para qué sirven las Plataformas IoT y cuáles son las más populares actualmente. Última visita: junio 2018. URL: http://www.ermesh.com/plataformas-iot-internet-de-las-cosas/.

[7] Azure IoT Hub Documentation. Última visita: junio 2018. URL: https://docs.microsoft.com/en-us/azure/iot-hub/.

[8] Azure IoT. Última visita: junio 2018. URL: https://azure.microsoft.com/en-au/updates/microsoft-azure-iot-reference-architecture-available/.

[9] Azure IoT. *Precios IoT Hub*. Última visita: junio 2018. URL: https://azure.microsoft.com/es-es/pricing/details/iot-hub/.

[10] IBM Bluemix Docs - Internet of Things Platform. *About Watson IoT Platform*. Última visita: junio 2018. URL: https://console.bluemix.net/docs/services/IoT/iotplatform_overview.html#about_iotplatform.

[11] *Introducción a IBM InfoSphere Streams y SPL*. Última visita: junio 2018. URL: https://www.ibm.com/developerworks/ssa/data/library/techarticle/introduccion-streams/index.html.

[12] *Streams Quick Start Guide*. Última visita: junio 2018. URL: https://developer.ibm.com/streamsdev/docs/streams-quick-start-guide/.

[13] IBM Bluemix. *Hoja de precios*. Última visita: junio 2018. URL: https://console.bluemix.net/?direct=classic/&cm_mc_uid=33451399742514887185837&cm_mc_sid_50200000=1494764670#/pricing/cloudOEPaneId=pricing&paneId=pricingSheet.

[14] IBM. *Explore MQTT and the Internet of Things service on IBM Bluemix*. Última visita: junio 2018. URL: https://www.ibm.com/developerworks/cloud/library/cl-mqtt-bluemix-iot-node-red-app/index.html.

[15] IBM. *Real Time Data Analysis Using IBM Watson IoT Platform Analytics*. Última visita: junio 2018. URL: https://developer.ibm.com/recipes/tutorials/real-time-data-analysis-using-ibm-watson-iot-platform-analytics/.

[16] IBM Bluemix. *Tutorial IBM Bluemix IoT*. Última visita: junio 2018. URL: https://www.ibm.com/developerworks/community/blogs/941f1004-4e3d-4a4b-87ed-30d8045fde4e/entry/ibm_bluemix_tutorial_-_connecting_a_device_using_internet_of_things_of_bluemix_v1.1?lang=en.

[17] IBM. *Mensajería de MQTT*. Última visita: junio 2018. URL: https://console.bluemix.net/docs/services/IoT/reference/mqtt/index.html#ref-mqtt.

[18] Amazon. *AWS IoT*. Última visita: junio 2018. URL: https://aws.amazon.com/es/iot-platform/.

[19] *How the AWS IoT Platform Works*. Última visita: junio 2018. URL: https://www.amazonaws.cn/en/iot-platform/how-it-works/.

[20] Amazon. *Message Broker for AWS IoT*. Última visita: junio 2018. URL: http://docs.aws.amazon.com/es_es/iot/latest/developerguide/iot-message-broker.html.

[21] Amazon. *AWS Topics*. Última visita: junio 2018. URL: http://docs.aws.amazon.com/es_es/iot/latest/developerguide/topics.html.

[22] Kaa Project. Última visita: junio 2018. URL: https://www.kaaproject.org/.

[23] *Kaa - Architecture overview*. Última visita: junio 2018. URL: https://kaaproject.github.io/kaa/docs/v0.10.0/Architecture-overview/.

[24] Fiware. Última visita: junio 2018. URL: https://www.fiware.org/.

[25] *FIWARE-NGSI v2 Specification*. Última visita: junio 2018. URL: https://orioncontextbroker docs.apiary.io/#reference.

[26] *Orion Context Broker NGSI API v1 Specification*. Última visita: junio 2018. URL: http://fiware.github.io/context.Orion/api/v1/.

[27] *ContextBroker - Architecture description*. Última visita: junio 2018. URL: https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/FIWARE.ArchitectureDescription.Data.ContextBroker.

[28] Fiware. *Cygnus*. Última visita: junio 2018. URL: http://fiware-cygnus.readthedocs.io/en/latest/index.html.

[29] Apache. *Flume 1.8.0 Developer Guide*. Última visita: junio 2018. URL: http://flume.apache.org/FlumeDeveloperGuide.html.

[30] Fiware. *Cygnus configuration examples*. Última visita: junio 2018. URL: http://fiware-cygnus.readthedocs.io/en/latest/cygnus-ngsi/installation_and_administration_guide/configuration_examples/index.html#section2.

[31] *Complex Event Processing (CEP) - Proactive Technology Online*. Última visita: junio 2018. URL: http://proactive-technology-online.readthedocs.io/en/latest/ProtonUserGuide_FI_WARE5_4_1/index.html.

[32] Fiware - CEP. *Real-time processing of context events*. Última visita: junio 2018. URL: http://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/FIWARE.OpenSpecification.Data.CEP.

[33] Fiware. *NGSI APIv1 Walkthrough*. Última visita: junio 2018. URL: https://fiware-orion.readthedocs.io/en/master/user/walkthrough_apiv2/index.html.

[34] Mooc Infotec. *Internet de las Cosas*. Última visita: junio 2018. URL: https://www.youtube.com/playlist?list=PL6CfSCeX-2gNwNcJYsDrAMhRof6npjg_B.

[35] MVOTMA. *Calidad del aire - Observatorio ambiental*. Última visita: junio 2018. URL: https://www.dinama.gub.uy/oan/?datos-abiertos=estaciones-de-aire.

[36] *Propuesta estándares de emisiones de fuentes móviles año 2015 - Grupo Gesta Aire*. Última visita: junio 2018. URL: http://www.mvotma.gub.uy/portal/cotama/item/10009010-propuesta-estandares-de-emisiones-de-fuentes-moviles-ano-2015-grupo-gesta-aire.html.

[37] *FIWARE Lab*. Última visita: junio 2018. URL: https://account.lab.fiware.org/.

[38] *Proton repository*. Última visita: junio 2018. URL: https://github.com/ishkin/Proton.

[39] *FIWARE-CEP v5.4.1 Specification*. Última visita: junio 2018. URL: http://htmlpreview.github.io/?https://github.com/ishkin/Proton/blob/master/documentation/apiary/CEP-apiary-blueprint.html.

[40] *Client libraries and samples for connecting to IBM Watson IoT using nodejs*. Última visita: junio 2018. URL: https://github.com/ibm-watson-iot/iot-nodejs.

[41] *How to Register Devices in IBM Watson IoT Platform*. Última visita: junio 2018. URL: https://developer.ibm.com/recipes/tutorials/how-to-register-devices-in-ibm-iot-foundation/.

[42] *How to Register Devices in IBM Watson IoT Platform - Generate API Keys*. Última visita: junio 2018. URL: https://developer.ibm.com/recipes/tutorials/how-to-register-devices-in-ibm-iot-foundation/#r_step5.

[43] *Configure Cloudant NoSQL DB as Historian Data Storage for IBM Watson IoT*. Última visita: junio 2018. URL: https://developer.ibm.com/recipes/tutorials/cloudant-nosql-db-as-historian-data-storage-for-ibm-watson-iot-parti/.

[44] *Integrate IBM Streaming Analytics Service with Watson IoT Platform*. Última visita: junio 2018. URL: https://developer.ibm.com/recipes/tutorials/integrate-ibm-streaming-analytics-service-with-watson-iot-platform/?cm_mc_uid=85545017034415007484211&cm_mc_sid_50200000=1512825799.

[45] *Introduction to Streams v4.2 Quick Start Edition (QSE) Edit me*. Última visita: junio 2018. URL: http://ibmstreams.github.io/streamsx.documentation/docs/4.2/qse-intro/.

[46] *E. Cavalcante et al., On the interplay of Internet of Things and Cloud Computing: A systematic mapping study, Computer Communications (2016)*. Última visita: junio 2018. URL: http://dx.doi.org/10.1016/j.comcom.2016.03.012.

[47] *Integration of cloud computing and Internet of Things: A survey*. Última visita: junio 2018. URL: http://dx.doi.org/10.1016/j.future.2015.09.021.

[48] *The Emerging Internet of Things Marketplace From an Industrial Perspective: A Survey*. Última visita: junio 2018. URL: http://ieeexplore.ieee.org/document/7004800/.