Aprendiendo Arduino

Aprendiendo a manejar Arduino en profundidad

MQTT

MQTT son las siglas de Message Queue Telemetry Transport y tras ellas se encuentra un protocolo ideado por IBM y liberado para que cualquiera podamos usarlo enfocado a la conectividad Machine-to-Machine (M2M).

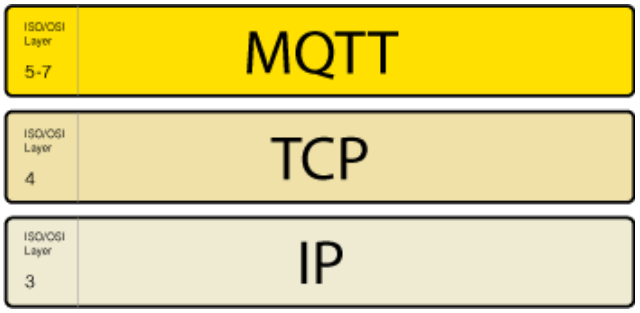
MQTT fue creado por el Dr. Andy Stanford-Clark de IBM y Arlen Nipper de Arcom — ahora Eurotech — en 1999 como una forma rentable y confiable de conectar los dispositivos de monitoreo utilizados en las industrias del petróleo y el gas a servidores empresariales remotos. Cuando se les planteó el reto de encontrar la manera de enviar los datos de los sensores de los oleoductos en el desierto a sistemas SCADA externos (control de supervisión y adquisición de datos), decidieron utilizar una topología de publicación/suscripción basada en TCP/IP que se basaría en los eventos para mantener bajos los costos de transmisión de los enlaces satelitales.

Aunque MQTTT todavía está estrechamente asociado con IBM, ahora es un protocolo abierto que es supervisado por la Organización para el Avance de los Estándares de Información Estructurada (OASIS).

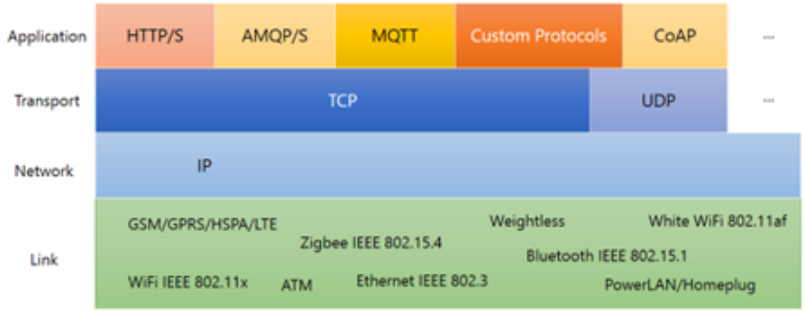
Web: http://mqtt.org/

Está enfocado al envío de datos en aplicaciones donde se requiere muy poco ancho de banda. Además, sus características le permiten presumir de tener un consumo realmente bajo así como precisar de muy pocos recursos para su funcionamiento.

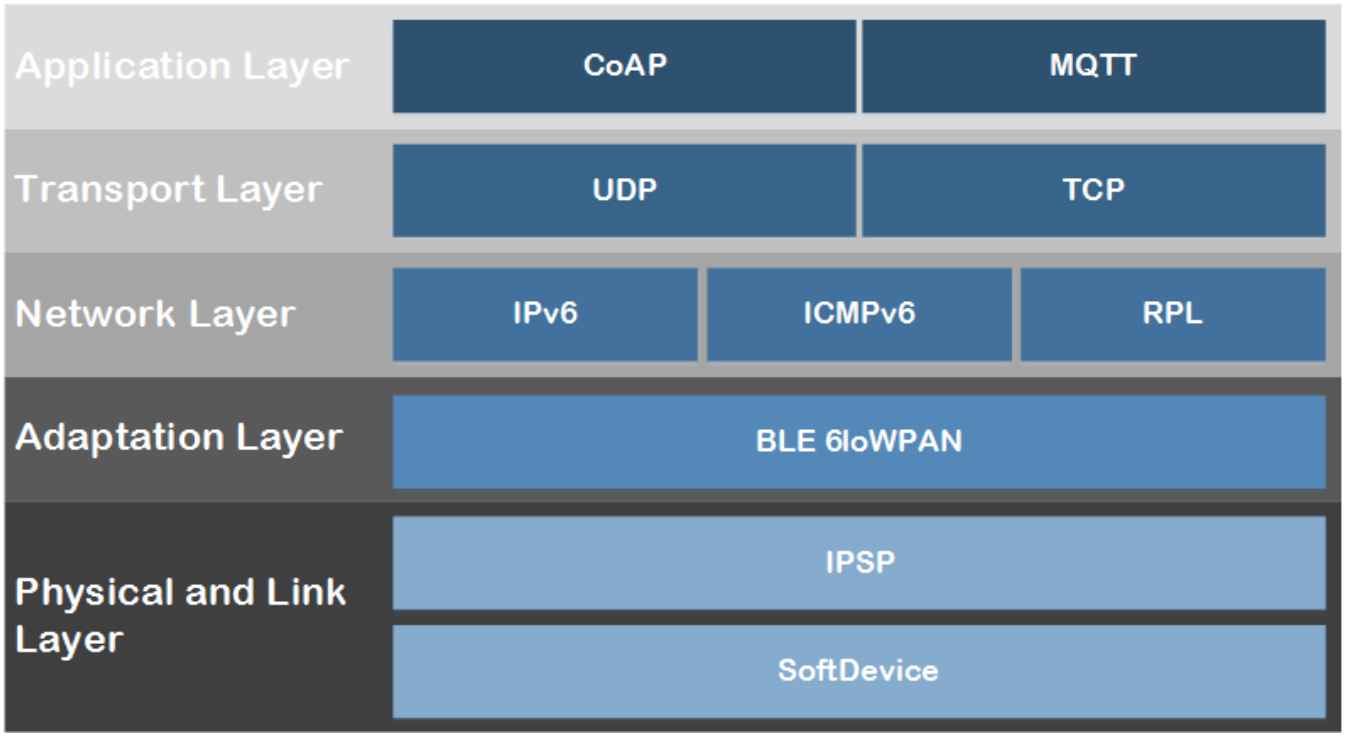
Estas características han hecho que rápidamente se convierta en un protocolo muy empleado en la comunicación de sensores y, consecuentemente, dentro del Internet de las Cosas.



MQTT es un protocolo pensado para IoT que está al mismo nivel que HTTP o CoAP:



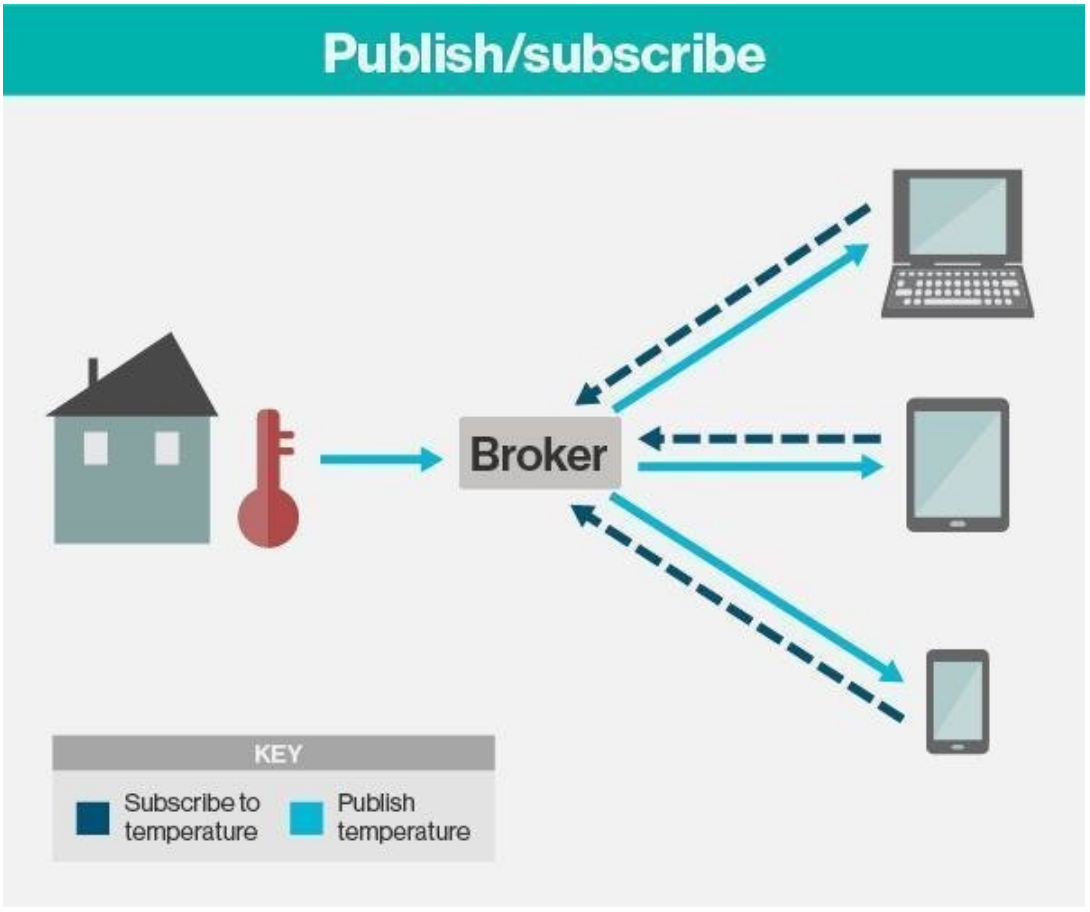
Comparativa MQTT y CoAP:



Un aspecto importante a tener en cuenta de los dispositivos IoT no es solamente el poder enviar datos al Cloud/Servidor, sino también el poder comunicarse con el dispositivo, en definitiva la bidireccionalidad. Este es uno de los beneficios de MQTT: es un modelo brokered, el cliente abre una conexión de salida al bróker, aunque el dispositivo esté actuando como Publisher o subscriber. **Esto normalmente evita los problemas con los firewalls porque funciona detrás de ellos o vía NAT.**

En el caso de que la comunicación principal se base en HTTP, **la solución tradicional para enviar información al dispositivo sería HTTP Polling. Esto es ineficiente y tiene un coste elevado en aspectos de tráfico y/o energía.** Una manera más novedosa de hacerlo sería con el protocolo WebSocket, que permite crear una conexión HTTP completa bidireccional. Esto actúa de canal socket (parecido al canal típico TCP) entre el servidor y el cliente. Una vez establecido, ya es trabajo del sistema escoger un protocolo para hacer un túnel sobre la conexión.

El Transporte de telemetría de cola de mensajes (MQTT) es un protocolo de código abierto que se desarrolló y optimizó para dispositivos restringidos y redes de bajo ancho de banda, alta latencia o poco confiables. Es un transporte de mensajería de publicación/suscripción que es extremadamente ligero e ideal para conectar dispositivos pequeños a redes con ancho de banda mínimo. El MQTT es eficiente en términos de ancho de banda, independiente de los datos y tiene reconocimiento de sesión continua, porque usa TCP. Tiene la finalidad de minimizar los requerimientos de recursos del dispositivo y, a la vez, tratar de asegurar la confiabilidad y cierto grado de seguridad de entrega con calidad del servicio.



El MQTT se orienta a grandes redes de dispositivos pequeños que necesitan la supervisión o el control de un servidor de back-end en Internet. No está diseñado para la transferencia de dispositivo a dispositivo. Tampoco está diseñado para realizar "multidifusión" de datos a muchos receptores. El MQTT es simple y ofrece pocas opciones de control. Las aplicaciones que usan MQTT, por lo general, son lentas en el sentido de que la definición de "tiempo real" en este caso se mide habitualmente en segundos.

Más información:

- https://en.wikipedia.org/wiki/MQTT
- Buena explicación de MQTT: https://www.artik.io/blog/2015/09/iot-101-networks/
- https://maxwellweru.com/communication-protocols-for-embedded-application-and-why-i-chose-mqtt-2/

MQTT también es un protocolo que está cobrando mucha importancia en la industria (IIoT). MQTT (Message Queuing Telemetry Transport, 'Cola de mensajes telemetría y transporte') es un protocolo publicar/suscribir diseñado para SCADA. Se centra en un mínimo encabezado (dos bytes de cabeza) y comunicaciones confiables. También es muy simple. Tal como HTTP, la carga MQTT es específica para la aplicación, y la mayoría de las implementaciones usan un formato JSON personalizado o binario.

MQTT en PLCs: https://www.youtube.com/watch?v=aX20J-sLyKU

Comparativa MQTT y Modbus: http://inubo.es/noticia/comparativa-entre-mqtt-y-modbus-como-protocolos-iot

MQTT es interesante usarlo cuando el ancho de banda bajo y no conozca su infraestructura. Asegúrese de que su proveedor tenga un broker MQTT a quien le pueda publicar información, y siempre asegure la comunicación con TLS (Transport Layer Security, 'seguridad en la capa de transporte').

Por ejemplo, MQTT sería una buena opción para monitorizar y controlar los paneles solares. MQTT es un protocolo de publicación/suscripción con brokers de mensajes centrales. Cada panel solar puede contener un nodo IoT que publique mensajes de tensión, corriente y temperatura.

MQTT está diseñado para minimizar el ancho de banda, lo que lo convierte en una buena opción para el monitoreo satelital de la línea de transmisión, pero hay una trampa. **La ausencia de metadatos en las cabeceras de los mensajes significa que la interpretación de los mensajes depende completamente del diseñador del sistema.**

Para compensar las redes poco fiables, MQTT soporta tres niveles de Calidad de Servicio (QoS):

- Disparar y olvidar (0) – Fire and Forget – At most once
- Al menos una vez (1) – At least once
- Exactamente una vez (2) – Exactly once

Si se solicita el nivel de calidad de servicio 1 ó 2, el protocolo gestiona la retransmisión de mensajes para garantizar la entrega. La calidad de servicio puede ser especificada por los clientes de publicación (cubre la transmisión del publicador al broker) y por los clientes suscriptores (cubre la transmisión de un broker a un suscriptor).

MQTT QoS 2 aumentará la latencia porque cada mensaje requiere dos handshake completos de ida y vuelta del remitente al receptor (cuatro en total del publicador al suscriptor).

En un patrón de publicación/suscripción es difícil saber la diferencia entre “Ha pasado mucho tiempo desde que supe de mi publicador” y “Mi publicador murió”. Ahí es donde entra en juego la Última Voluntad y el Testamento (LWT) de MQTT. Los clientes pueden publicar mensajes sobre temas específicos (por ejemplo, aisle15/rack20/panel5/FalloSensor) para que se entreguen si se desconectan sin enviar un mensaje de “desconexión”. Los mensajes se almacenan en el broker y se envían a cualquier persona que se haya suscrito al tema.

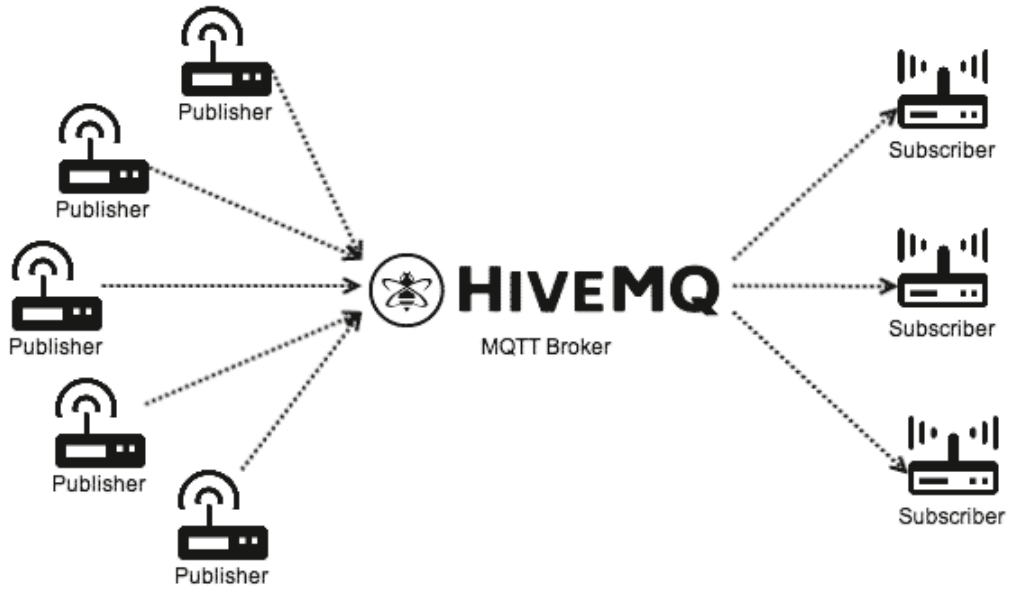
MQTT de un vistazo

- Ancho de banda muy bajo
- TCP/IP
- Publicar/suscribir transferencia de mensajes
- Topología de muchos a muchos a través de un broker central
- Sin metadatos
- Tres niveles de QoS
- Última Voluntad y Testamento revela nodos desconectados

Las ventajas de usar el protocolo MQTT son:

- Es asíncrono con diferentes niveles múltiples de calidad del servicio, lo que resulta ser importante en los casos donde las conexiones de Internet son poco confiables.
- Envía mensajes cortos que se vuelven adecuados para las situaciones de un bajo ancho de banda.
- No se requiere de mucho software para implementar a un cliente, lo cual lo vuelve fantástico para los dispositivos como Arduino con una memoria limitada.
- Podemos cifrar los datos enviados y usar usuario y password para proteger nuestros envíos.

Si quisiera grabar en una BBDD con MQTT, un suscriptor a una serie de topics se encarga de grabar los datos cada vez que cambia un valor o cada cierto tiempo, por ejemplo con un script de python o ejecutando Node-RED en una máquina virtual o en el propio servidor (o Raspberry Pi) donde corre el broker (Mosquitto):



NodeRed no es más que un software que se instala en un nodo aunque se instale en el mismo servidor que el broker.

Cinco cosas a saber de
MQTT: https://www.ibm.com/developerworks/community/blogs/5things/entry/5_things_to_know_about_mqtt_the_protocol_for_internet_of_things?lang=en

Buen artículo sobre MQTT: https://internetofthingsagenda.techtarget.com/definition/MQTT-MQ-Telemetry-Transport

Recursos MQTT:

- Web: http://mqtt.org/
- MQTT specificaction: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html
- Documentación MQTT: https://mqtt.org/documentation
- Software MQTT: https://github.com/mqtt/mqtt.github.io/wiki/software?id=software
- Wiki MQTT: https://github.com/mqtt/mqtt.github.io/wiki
- Tutorial de Adafuit muy completo: https://learn.adafruit.com/mqtt-adafruit-io-and-you?view=all
- http://blog.whatsbee.net/?incsub_wiki=objetos-conectados/introduccion-a-mqtt

Para ampliar información de MQTT en Arduino y Raspberry Pi:

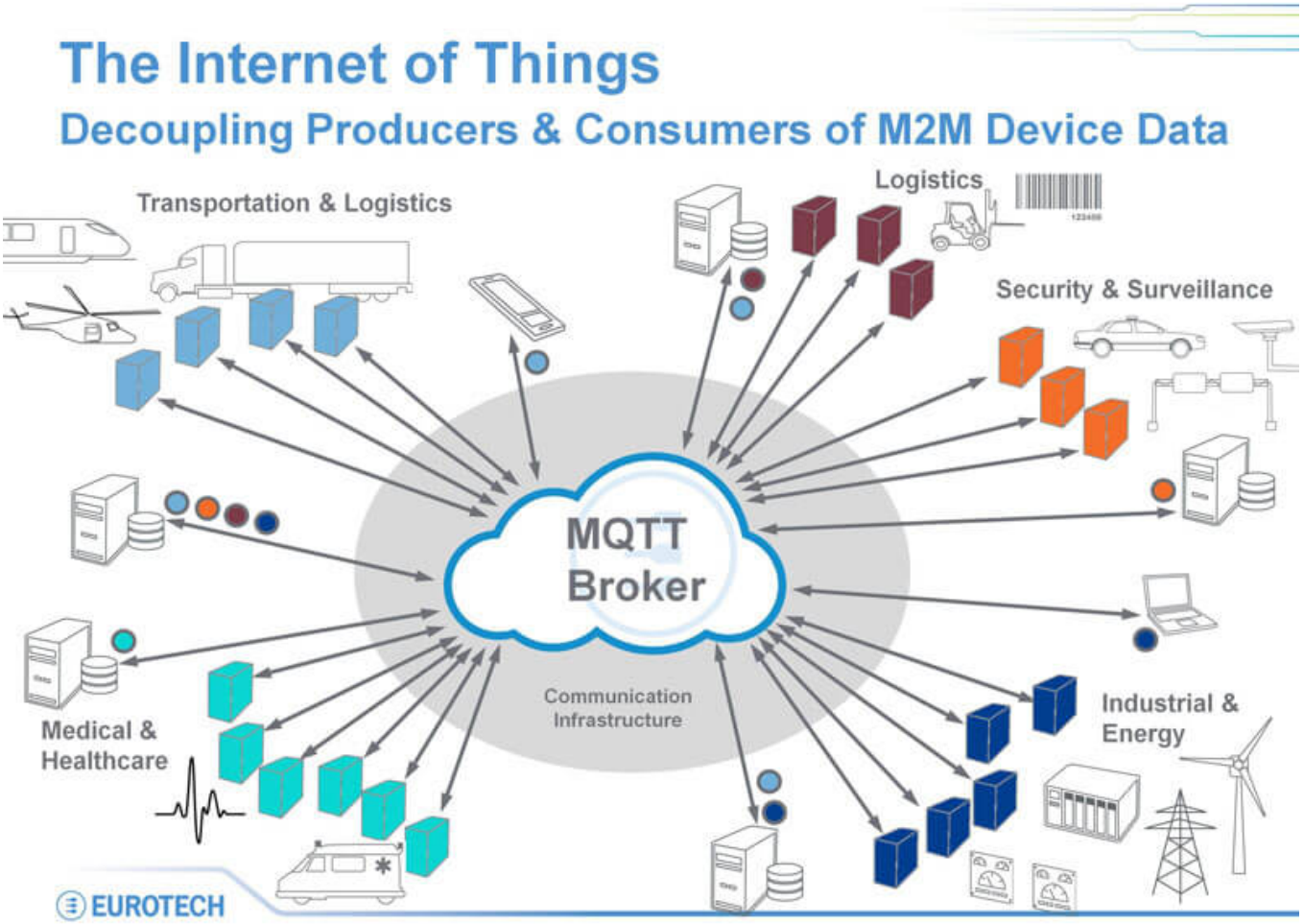
- https://www.baldengineer.com/mqtt-introduction.html
- https://www.baldengineer.com/mqtt-tutorial.html
- https://www.baldengineer.com/multiple-mqtt-topics-pubsubclient.html

Buenos artículos de MQTT en español:

- https://ricveal.com/blog/primeros-pasos-mqtt/
- https://ricveal.com/blog/uso-mqtt/
- https://ricveal.com/blog/arduino-mqtt/
- M2M: https://ricveal.com/blog/3-preguntas-m2m/

Arquitectura MQTT

MQTT (Message Queue Telemetry Transport), un protocolo usado para la comunicación machine-to-machine (M2M) en el “Internet of Things”. Este protocolo está orientado a la comunicación de sensores, debido a que consume muy poco ancho de banda y puede ser utilizado en la mayoría de los dispositivos empotrados con pocos recursos (CPU, RAM, …).



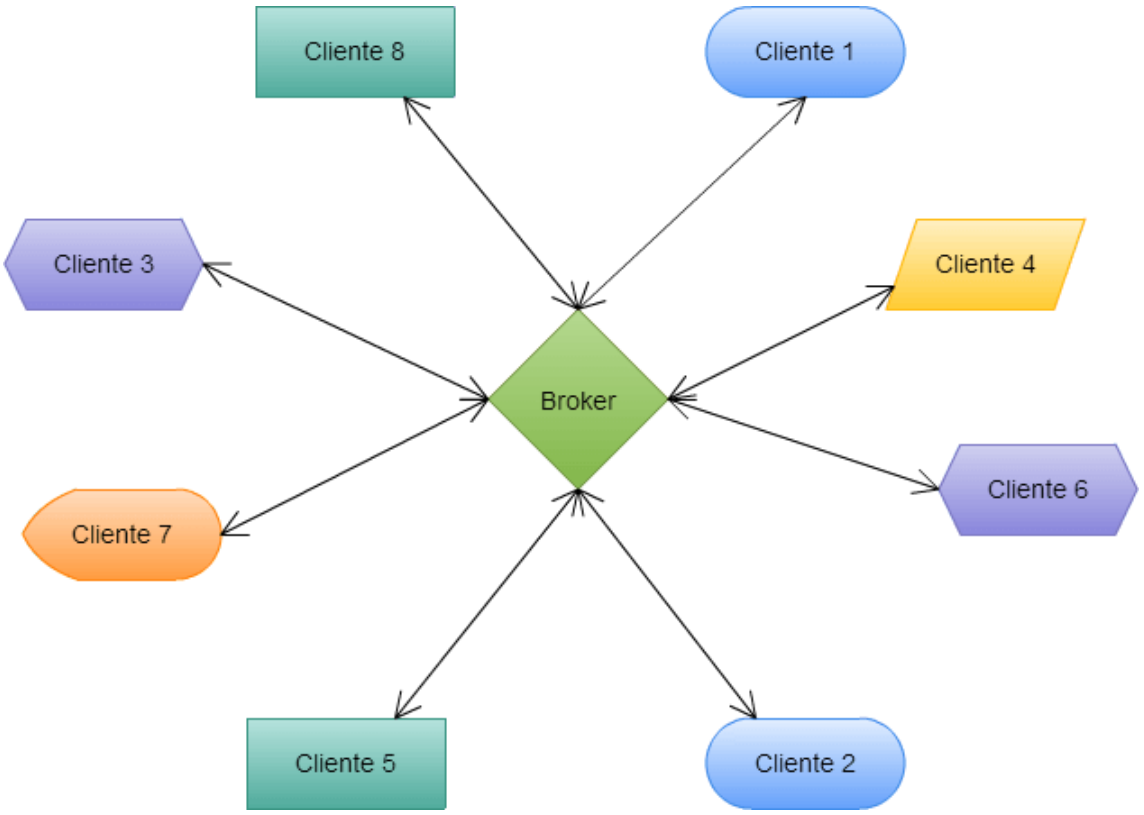
Un ejemplo de uso de este protocolo es la aplicación de Facebook Messenger tanto para android y Iphone. La arquitectura de MQTT sigue una topología de estrella, con un nodo central que hace de servidor o “broker”. **El broker es el encargado de gestionar la red y de transmitir los mensajes, para mantener activo el canal, los clientes mandan periódicamente un paquete (PINGREQ) y esperan la respuesta del broker (PINGRESP).** La comunicación puede ser cifrada entre otras muchas opciones.

En esta forma de comunicación se desacoplan los clientes que publican (**Publisher**) de los que consumen los datos (**Suscribers**). Eso significa que los clientes no se conocen entre ellos unos publican la información y otros simplemente la consumen, simplemente todos tienen que conocer al message broker.

El desacoplamiento se produce en tres dimensiones:

- En el espacio: El publicador y el suscriptor no tienen porqué conocerse.
- En el tiempo: El publicador y el suscriptor no tienen porqué estar conectados en el mismo momento.
- En la sincronización: las operaciones en cualquiera de los dos componentes no quedan interrumpidas mientras se publican o se reciben mensajes.

Es precisamente el broker el elemento encargado de gestionar la red y transmitir los mensajes.



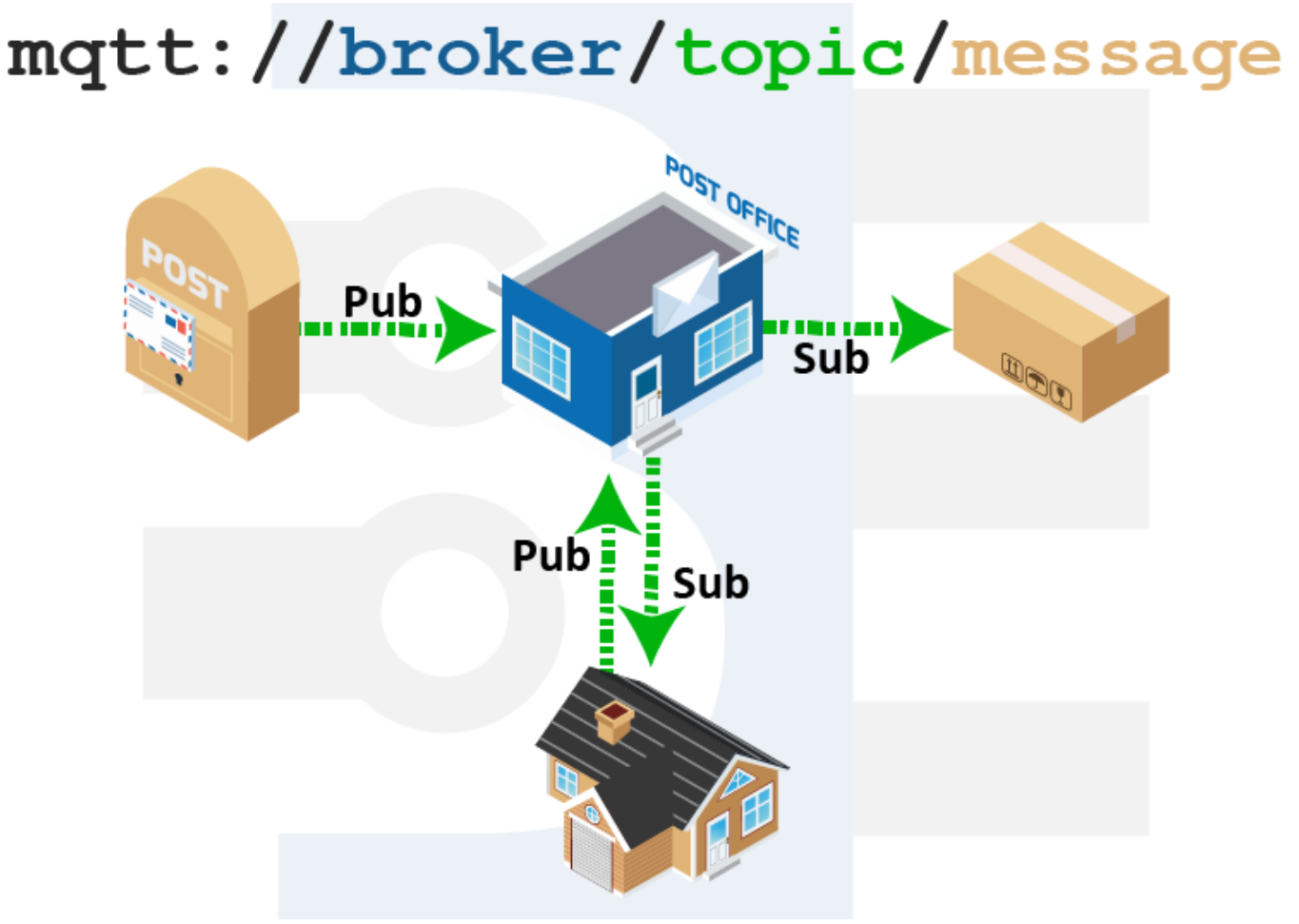
Una característica interesante es la capacidad de MQTT para establecer comunicaciones cifradas lo que aporta a nuestra red una capa extra de seguridad.

La comunicación se basa en unos “topics” (temas), que el cliente que publica el mensaje crea y los nodos que deseen recibirlo deben subscribirse a él. La comunicación puede ser de uno a uno, o de uno a muchos.

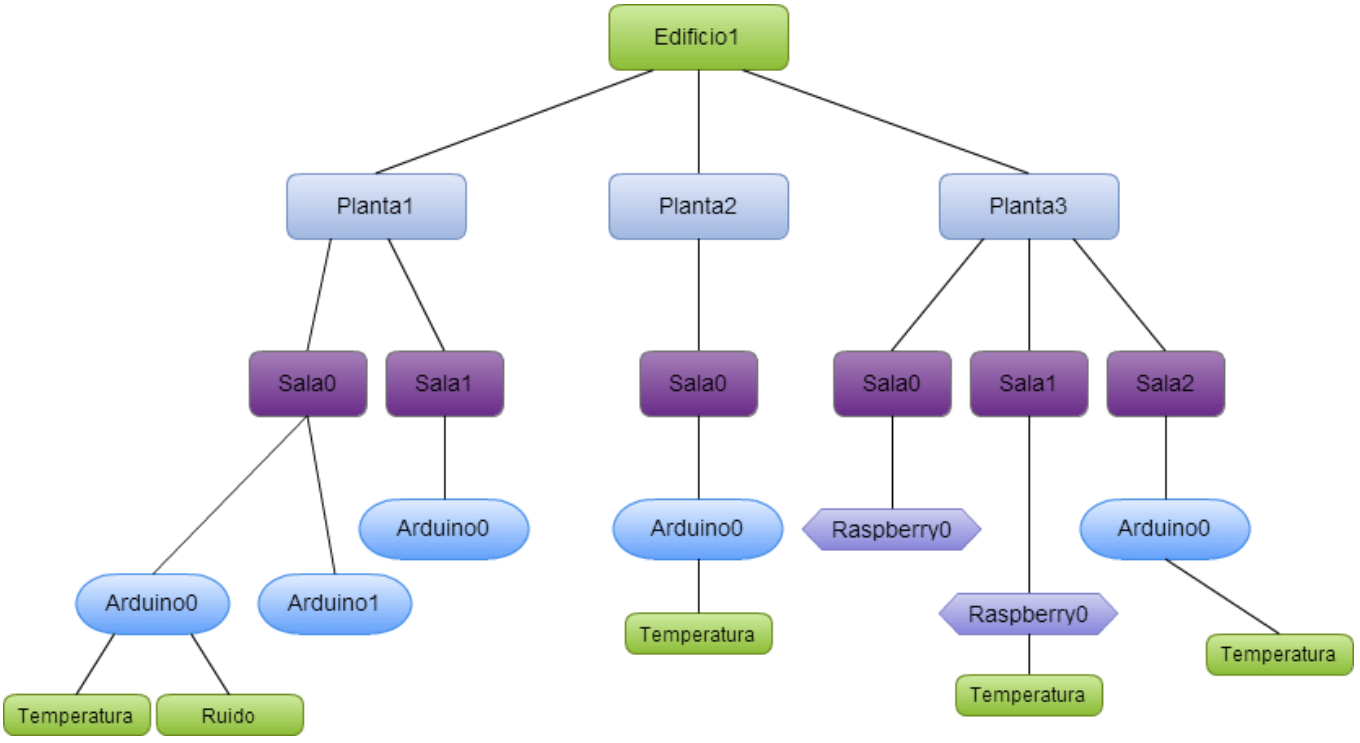
Dentro de la arquitectura de MQTT, es muy importante el concepto “topic” o “tema” en español ya que a través de estos “topics” se articula la comunicación puesto que emisores y receptores deben estar suscritos a un “topic” común para poder entablar la comunicación. Este concepto es prácticamente el mismo que se emplea en colas, donde existen un publicadores (que publican o emiten información) y unos suscriptores (que reciben dicha información) siempre que ambas partes estén suscritas a la misma cola.

Este tipo de arquitecturas, lleva asociada otra interesante característica: la comunicación puede ser de uno a uno o de uno a muchos.

Un “topic” se representa mediante una cadena y tiene una estructura jerárquica. Cada jerarquía se separa con ‘/’.



Por ejemplo, "**edificio1/planta5/sala1/raspberry2/temperatura**" o "**/edificio3/planta0/sala3/arduino4/ruido**". De esta forma se pueden crear jerarquías de clientes que publican y reciben datos, como podemos ver en la imagen:

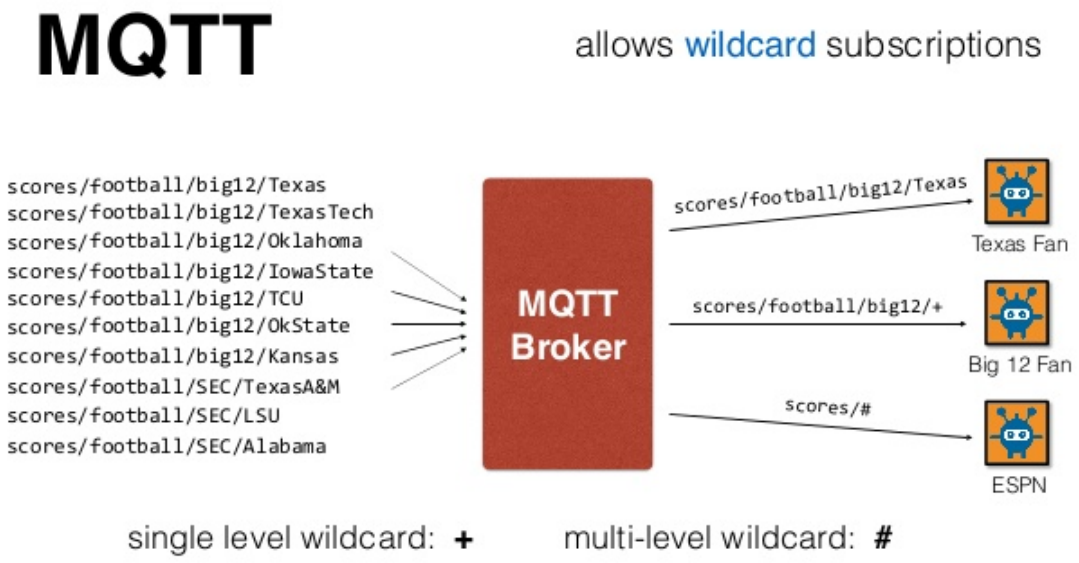


De esta forma un nodo puede subscribirse a un "topic" concreto ("edificio1/planta2/sala0/arduino0/temperatura") o a varios ("edificio1/planta2/#").

Existen dos comodines en MQTT para suscribirse a varios topics:

- Multi-level Wildcard: # (se suscribe a todos los hijos bajo esa cola)
- Single Level Wildcard: + (se suscribe solo a ese nivel)

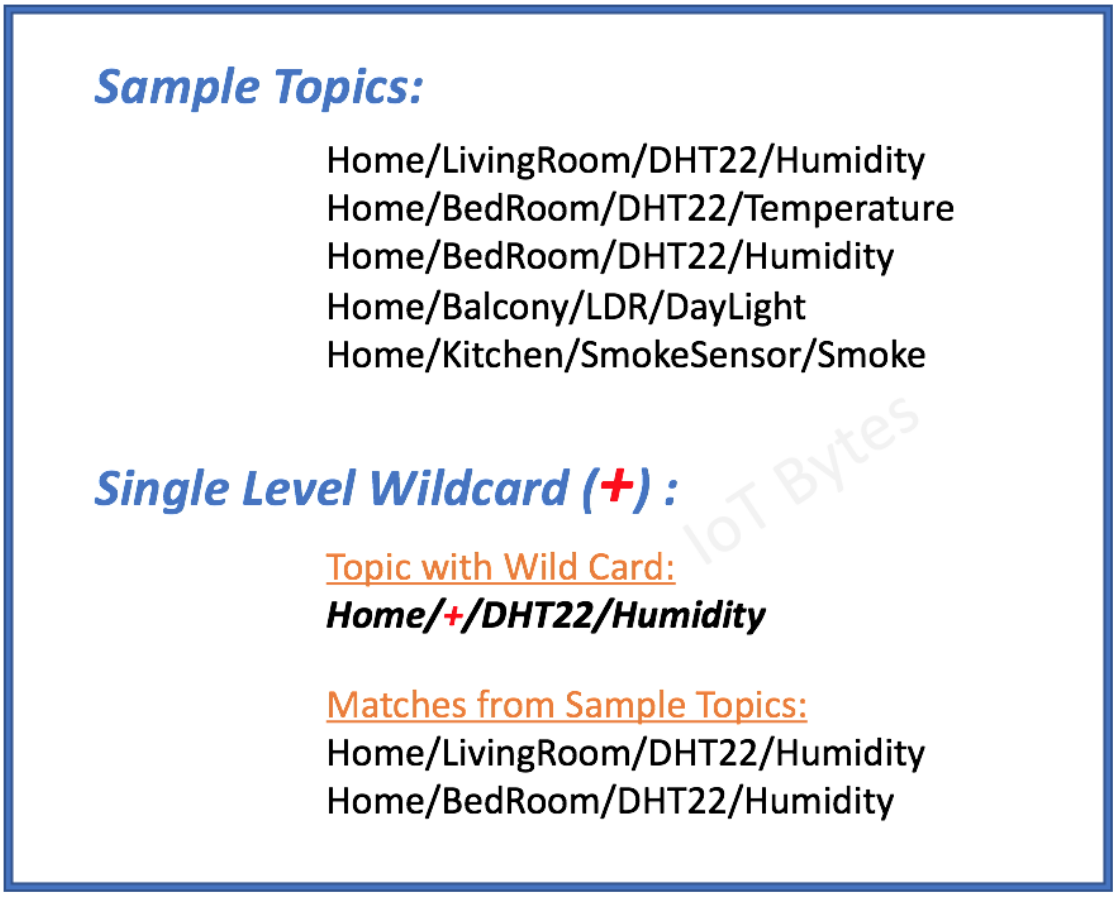
Un carácter de tema especial, el carácter dólar ($), excluye un tema de cualquier suscripción de root wildcard. Normalmente, el $ se utiliza para transportar mensajes específicos del servidor o del sistema.



Ejemplos de Topics MQTT Válidos:

- casa/prueba/topic
- casa/+/topic
- casa/#
- casa/+/+
- +/#
- #

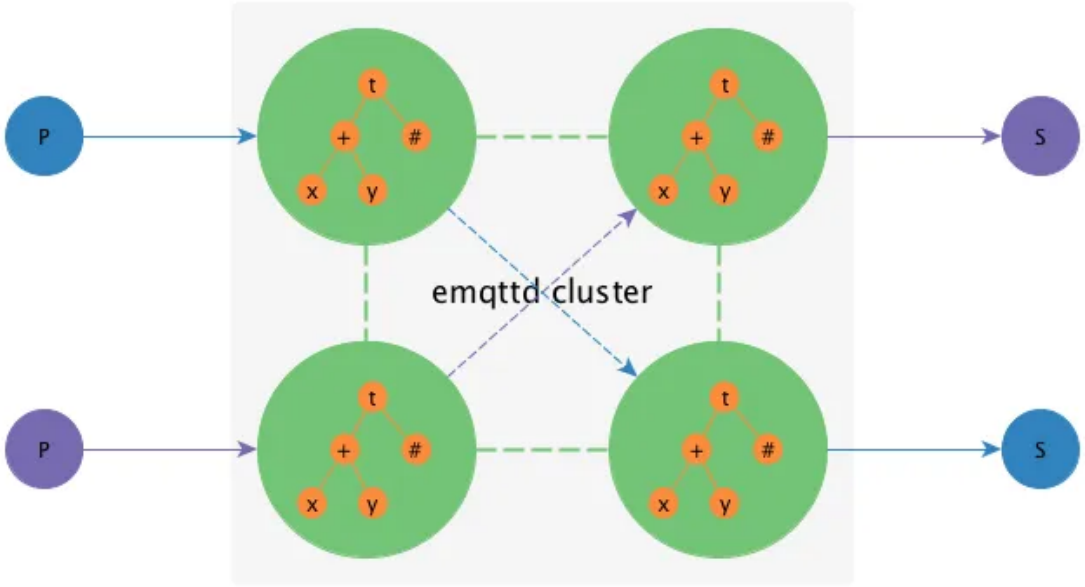
Explicación del comodín de single level:



Escalado MQTT

MQTT me permite gran escalabilidad. Añadir un nuevo Arduino o un suscriptor es muy sencillo dentro de la jerarquía vista

Con escalable me refiero a la capacidad que tiene un sistema para ser ampliado. Los sistemas de sensores en general, particularmente en nuestro caso hablamos del mundillo del Internet de las Cosas, se caracterizan por enviar muchos datos de pequeño tamaño en tiempo real ya que hay muchos sensores transmitiendo simultáneamente y cada breves periodos de tiempo, cuya información necesita ser consumida por otros elementos en tiempo real.



En una Arquitectura basada en Broker es fundamental evitar el SPOF (punto único de fallo).

En el contexto MQTT hay 2 estrategias principales:

- Bridging: hace forward de mensajes a otro bróker MQTT. Es la solución de HiveMQ, Mosquitto, IBM MQ
- Clustering: soportando añadido dinámico de nodos al cluster. Lo usa ActiveMQ, HiveMQ o RabbitMQ.

Cuando un sistema de estas características se empieza a saturar se bloquean las comunicaciones y se pierde la característica de "tiempo real".

Hasta ahora, todos los sistemas que habíamos visto se basaban en un cliente que se comunicaba con un servidor. Si cualquier cliente se intenta comunicar con un servidor que está procesando tanta información que, en ese momento, no es capaz de trabajar con más contenido, el sistema entero fallará, o bien porque se satura y bloquea a nivel global o porque empieza a descartar aquella información que no puede procesar (lo que es inadmisible en muchos caso, imagina una alarma de Riesgo de Explosión en tu cocina porque se ha detectado una fuga de gas...).

Existen varias formas de abordar esta problemática pero, a día de hoy, una de las más empleadas es usar sistemas de colas donde se deja toda la información y el encargado de procesarla va "sacando" de esta cola la información. De esta manera, si ponemos más "encargados de procesamiento" son capaces de vaciar más rápido la cola si viésemos que está se está empezando a llenar y, de cara a los sensores, no sería necesario hacer ningún cambio, ya que ellos siempre envían al mismo sitio.

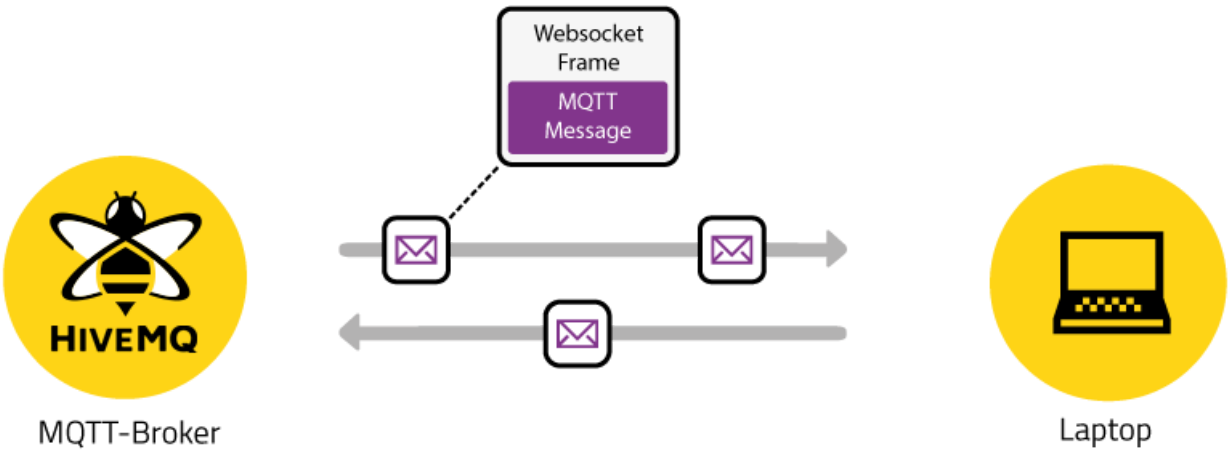
MQTT no hace exactamente lo mismo ya que, para empezar, no hay colas sino "topics" pero la filosofía es muy parecida, permitiendo a grandes sistemas operar con total fluidez y, junto con sus optimizaciones que buscan entre otras cosas reducir consumos y tamaños de trama para poder operar en elementos embebidos, es el motivo por el que es un protocolo tan empleado en comunicaciones M2M.

Además, nos permite una gestión de la seguridad razonablemente sencilla que facilita que nuestros sistemas se comporten de una forma más robusta.

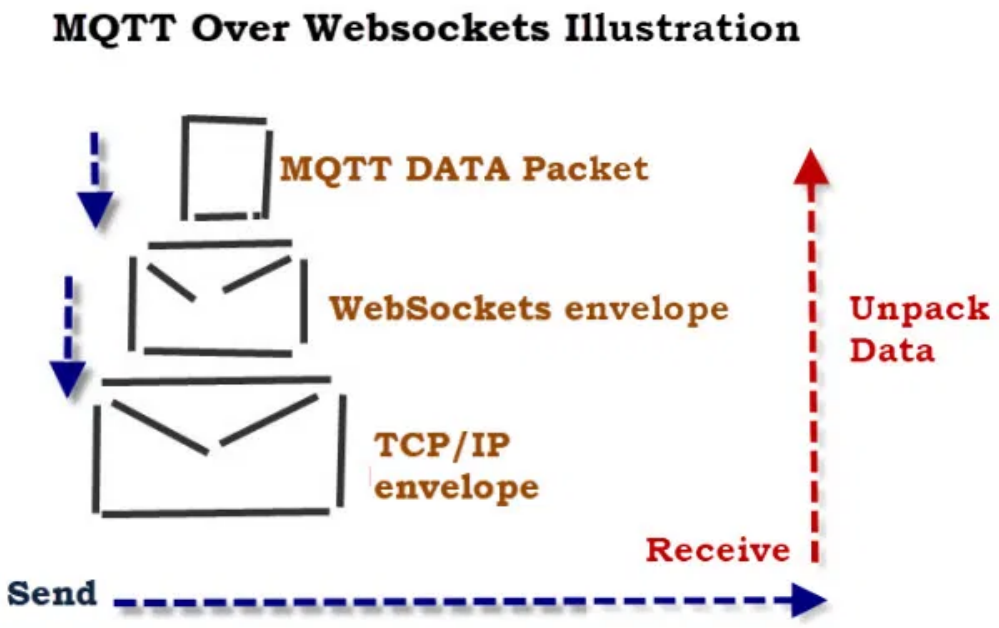
MQTT será el nexo entre hardware (sensor) y todos los elementos típicos del mundo software (servidores, bases de datos, Big Data). En esta capa nos preocupamos de que la información llegue a un sistema que posteriormente se ocupa de distribuirlo entre las demás partes y nos da igual lo que haya a partir de ese momento y su tamaño. Puede que no tengamos nada más que una web de visualización o puede que tengamos un complejo sistema de Machine Learning y Big Data. Puede que seamos un particular enviando un dato de temperatura a un panel de visualización en su Raspberry o puede que seamos una multinacional que controla en tiempo real su producción de amoniaco a nivel global bajando y subiendo la carga de producción en sus diferentes fábricas según los costes de transporte y el consumo de sus diferentes centros de distribución. Nos es lo mismo a este nivel porque nosotros hacemos sólo una cosa y la hacemos bien: enviar datos de un dispositivo hardware a un sistema mucho mayor. Es lo que se llama microservicios que ha popularizado netflix (http://enmilocalfunciona.io/arquitectura-microservicios-1/)

MQTT Sobre Websockets

MQTTT es una arquitectura de publicación/suscripción que se desarrolla principalmente para conectar dispositivos con ancho de banda y potencia limitada a través de redes inalámbricas. Es un protocolo simple y ligero que corre sobre sockets TCP/IP o WebSockets.



MQTT sobre WebSockets puede ser asegurado con SSL. La arquitectura de publicación/suscripción se basa en eventos que permiten enviar mensajes a los dispositivos cliente. El broker MQTT es el punto central de comunicación, y se encarga de despachar todos los mensajes entre los remitentes y los destinatarios legítimos. Un cliente es cualquier dispositivo que se conecta al corredor y puede publicar o suscribirse a temas para acceder a la información.



Un tema contiene la información de enrutamiento para el broker. Cada cliente que quiere enviar mensajes publica sobre un tema determinado, y cada cliente que quiere recibir mensajes se suscribe a un tema determinado. El corredor entrega todos los mensajes con el tema correspondiente a los clientes apropiados.

Sólo necesitará ejecutar MQTTT en sockets web si desea publicar/suscribirse a mensajes directamente desde webapps.

Buena web para estar al día de MQTT: https://www.hivemq.com/blog/

Más información:

- https://www.hivemq.com/blog/mqtt-essentials-special-mqtt-over-websockets

- https://stackoverflow.com/questions/30624897/direct-mqtt-vs-mqtt-over-websocket

Para saber más sobre MQTT leer esta serie de Artículos:

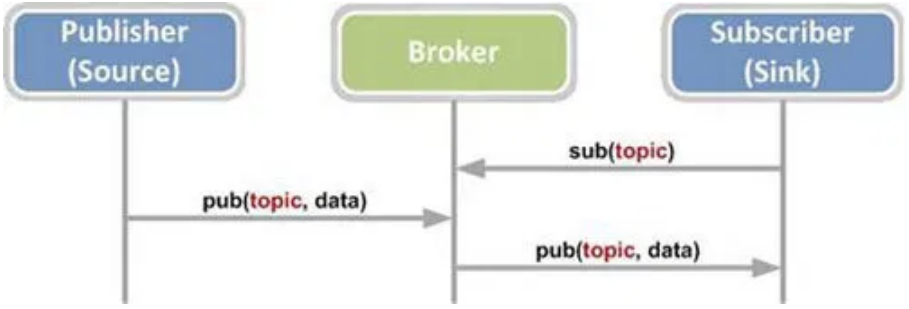
- https://www.hivemq.com/blog/mqtt-essentials-wrap-up
- https://www.hivemq.com/blog/mqtt-essentials-part-1-introducing-mqtt
- https://www.hivemq.com/blog/mqtt-essentials-part2-publish-subscribe
- https://www.hivemq.com/blog/mqtt-essentials-part-3-client-broker-connection-establishment
- https://www.hivemq.com/blog/mqtt-essentials-part-4-mqtt-publish-subscribe-unsubscribe
- https://www.hivemq.com/blog/mqtt-essentials-part-5-mqtt-topics-best-practices
- https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels
- https://www.hivemq.com/blog/mqtt-essentials-part-7-persistent-session-queuing-messages
- https://www.hivemq.com/blog/mqtt-essentials-part-8-retained-messages
- https://www.hivemq.com/blog/mqtt-essentials-part-9-last-will-and-testament
- https://www.hivemq.com/blog/mqtt-essentials-part-10-alive-client-take-over
- https://www.hivemq.com/blog/mqtt-essentials-special-mqtt-over-websockets

Protocolo MQTT

MQTT está diseñado para requerir una sobrecarga mínima del protocolo para cada paquete con el fin de preservar el ancho de banda para los dispositivos embebidos con recursos limitados. Es un marco de trabajo realmente sencillo para la gestión de redes mesh de dispositivos habilitados para TCP.

Los mensajes MQTT se entregan asincrónicamente (“push”) a través de la arquitectura publish subscribe. El protocolo MQTTT funciona intercambiando una serie de paquetes de control MQTT de una manera definida. Cada paquete de control tiene un propósito específico y cada bit del paquete se crea cuidadosamente para reducir los datos transmitidos a través de la red.

Publish and subscribe:

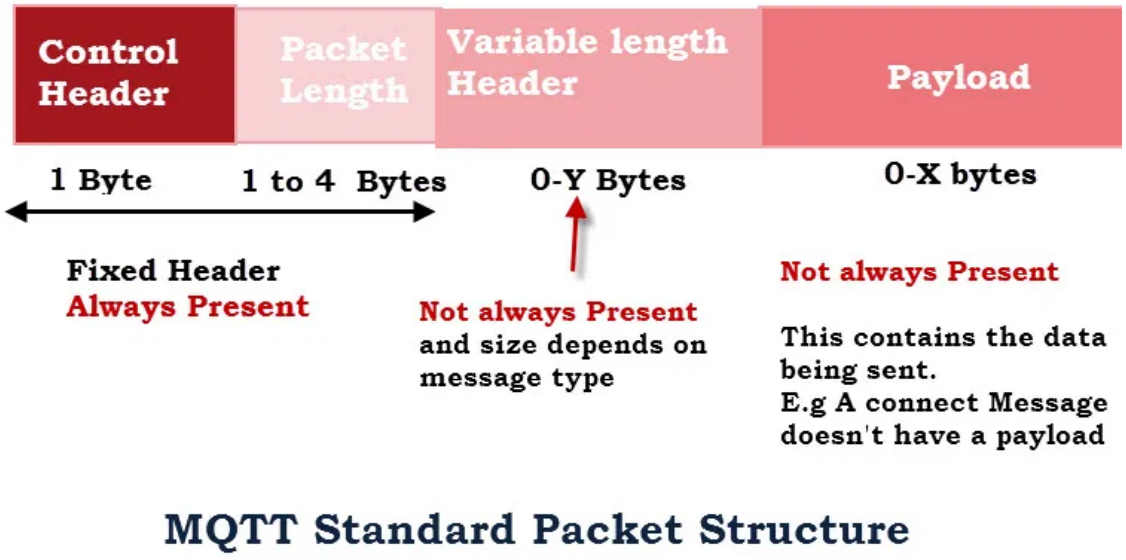


MQTT specificaction: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html

Una sesión MQTT se divide en cuatro etapas: **conexión, autenticación, comunicación y terminación**. Un cliente comienza creando una conexión TCP/IP con el broker utilizando un puerto estándar o un puerto personalizado definido por los operadores del broker. Al crear la conexión, es importante reconocer que el servidor puede continuar una sesión antigua si se le proporciona una identidad de cliente reutilizada.

Los puertos estándar son el 1883 para la comunicación no cifrada y el 8883 para la comunicación cifrada mediante SSL/TLS. Durante el handshake SSL/TLS, el cliente valida el certificado del servidor para autenticar el servidor.

MQTT es llamado un protocolo ligero porque todos sus mensajes tienen una pequeña huella de código. Cada mensaje consta de una cabecera fija, una cabecera variable opcional, una carga útil de mensaje limitada a 256 MB de información y un nivel de calidad de servicio (QoS).



MQTT Standard Packet Structure

MQTTT soporta BLOBS (Binary Large Object) de mensajes de hasta 256 MB de tamaño. El formato del contenido es específico de la aplicación. Las suscripciones de temas se realizan utilizando un par de paquetes SUBSCRIBE/SUBACK. La anulación de la suscripción se realiza de forma similar utilizando un par de paquetes UNSUBSCRIBE/UNSUBACK.

Durante la fase de comunicación, el cliente puede realizar operaciones de **publicación, suscripción, cancelación (unsuscribe) y ping**. La operación de publicación envía un bloque binario de datos, el contenido, a un topic definido por el publisher.

La operación de ping al servidor del broker usando una secuencia de paquetes PINGREQ/PINGRESP, que se usa para saber si está viva la conexión. Esta operación no tiene otra función que la de mantener una conexión en vivo y asegurar que la conexión TCP no ha sido apagada por una pasarela o un router.

Cuando un publicador o suscriptor desea finalizar una sesión MQTT, envía un mensaje DISCONNECT al broker y, a continuación, cierra la conexión. Esto se denomina “graceful shutdown” porque le da al cliente la posibilidad de volver a conectarse fácilmente al proporcionarle su identidad de cliente y reanudar el proceso donde lo dejó.

Si la desconexión ocurre repentinamente sin tiempo para que un publisher envíe un mensaje DISCONNECT, el broker puede enviar a los suscriptores un mensaje del publisher que el broker ha almacenado previamente en caché. El mensaje, que se llama testamento, proporciona a los suscriptores instrucciones sobre qué hacer si el editor muere inesperadamente.

MQTT tiene 14 tipos de mensajes, normalmente un usuario sólo usa los mensajes de **CONNECT, PUBLISH, SUBSCRIBE Y UNSUBSCRIBE**. Si queréis conocer los tipos de mensajes podéis consultarlos en: https://dzone.com/refcardz/getting-started-with-mqtt

Más información: https://internetofthingsagenda.techtarget.com/definition/MQTT-MQ-Telemetry-Transport

Para saber más sobre el protocolo MQTT y aplicaciones:

- Adafruit https://learn.adafruit.com/mqtt-adafruit-io-and-you/overview
- Bald engineer: https://www.baldengineer.com/mqtt-introduction.html
- https://www.baldengineer.com/multiple-mqtt-topics-pubsubclient.html
- https://www.baldengineer.com/mqtt-tutorial.html
- https://thenewstack.io/mqtt-protocol-iot/
- Comparación de protocolos: https://www.slideshare.net/paolopat/mqtt-iot-protocols-comparison
- Presentación MUY BUENA: https://www.slideshare.net/BryanBoyd/mqtt-austin-api
- zigbee and mqtt http://rijware.com/zigbee-and-mqtt/
- mqtt and appinventor http://internetofhomethings.com/homethings/?p=1317

Calidad de Servicio MQTT (QoS)

Al enviar mensajes MQTT existe la posibilidad que los mensajes no lleguen al destinatario.

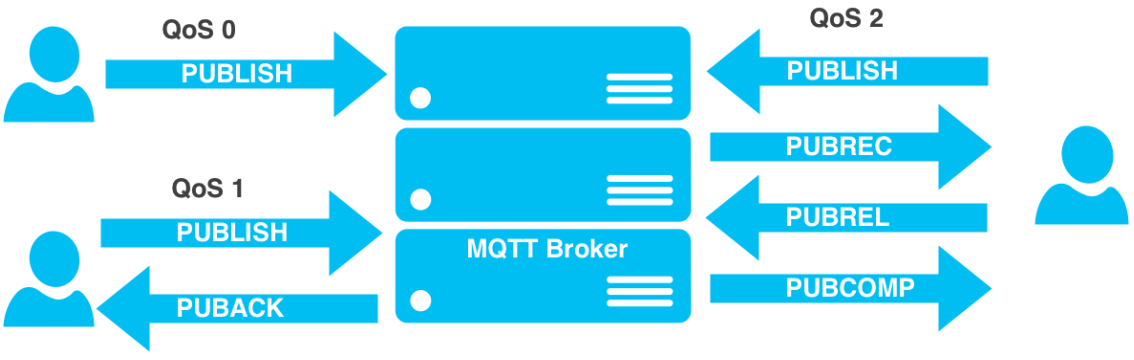
El envío de mensajes sin saber con seguridad que fueron recibidos se llama “QoS 0” (cero).

Es posible que también desee QoS 1, que le permite saber que el mensaje fue recibido. Básicamente, después de cada publicación, el suscriptor dice “OK”. En el lenguaje MQTT se llama “PUBACK” (Reconocimiento de publicación).

También está QoS 2, que no sólo garantiza que su mensaje fue recibido, sino que sólo fue recibido una vez. Esto es un poco más complejo porque necesitas empezar a rastrear los IDs de los paquetes, así que lo dejaremos para más adelante.

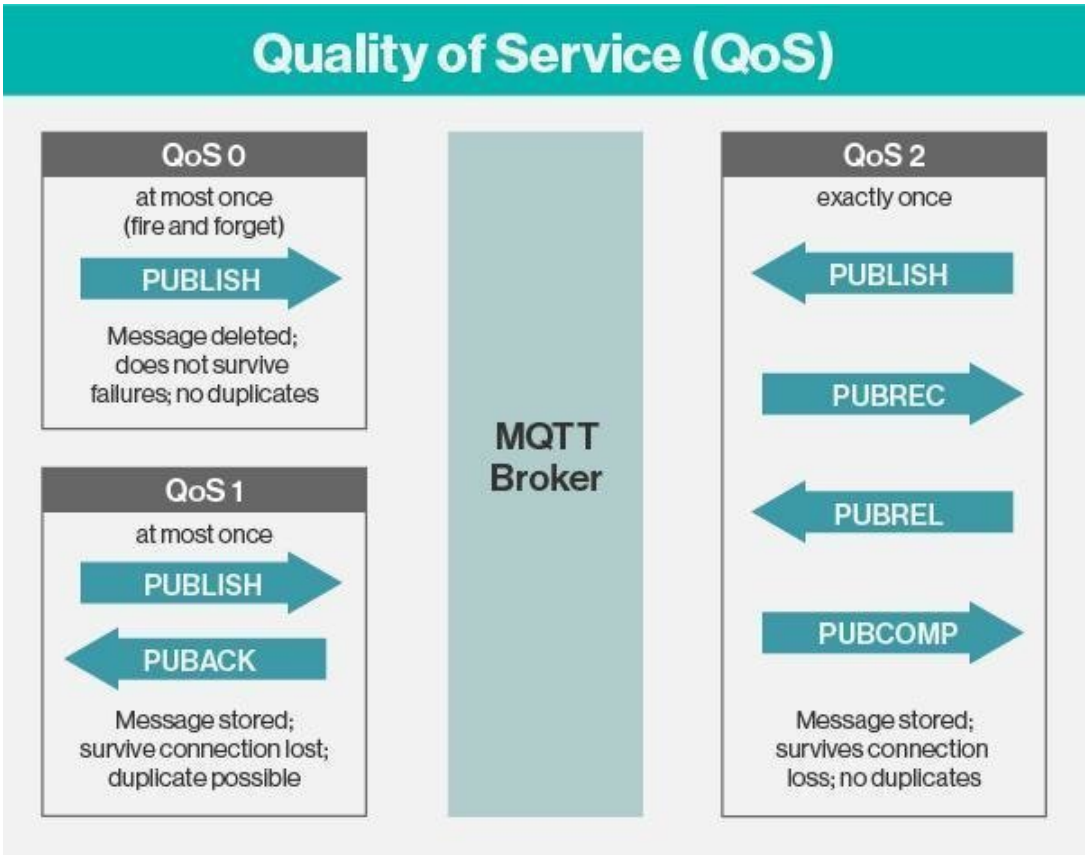
Los niveles de calidad de servicio (QoS) determinan cómo se entrega cada mensaje MQTT y deben especificarse para cada mensaje enviado a través de MQTT. Es importante elegir el valor de QoS adecuado para cada mensaje, ya que este valor determina la forma en que el cliente y el servidor se comunican para entregar el mensaje. Con el uso de MQTT se podrían lograr tres niveles de calidad de servicio para la entrega de mensajes:

- QoS 0 (A lo sumo una vez – at most once) – donde los mensajes se entregan de acuerdo con los mejores esfuerzos del entorno operativo. Puede haber pérdida de mensajes. Confía en la fiabilidad del TCP. No se hacen retransmisiones.
- QoS 1 (Al menos una vez – at least once) – donde se asegura que los mensajes lleguen, pero se pueden producir duplicados. El Receiver recibe el mensaje por lo menos una vez. Si el receiver no confirma la recepción del mensaje o se pierde en el camino el Sender reenvía el mensaje hasta que recibe por lo menos una confirmación. Pueden duplicarse mensajes.
- QoS 2 (Exactamente una vez – exactly once) – donde se asegura que el mensaje llegue exactamente una vez. Eso incrementa la sobrecarga en la comunicación pero es la mejor opción cuando la duplicación de un mensaje no es aceptable.



Existe una regla simple cuando se considera el impacto del rendimiento de la QoS. Es “**Cuanto mayor sea la QoS, menor será el rendimiento**“. MQTT proporciona flexibilidad a los dispositivos de IoT para elegir la calidad de servicio

apropiada que necesitarían para sus requisitos funcionales y ambientales.

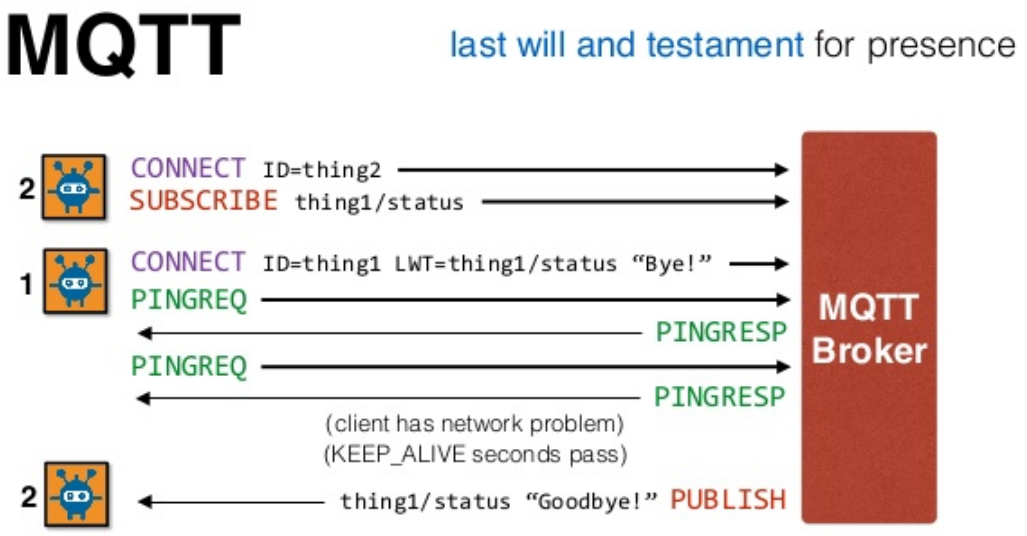


Último deseo y Testamento (MQTT LWT)

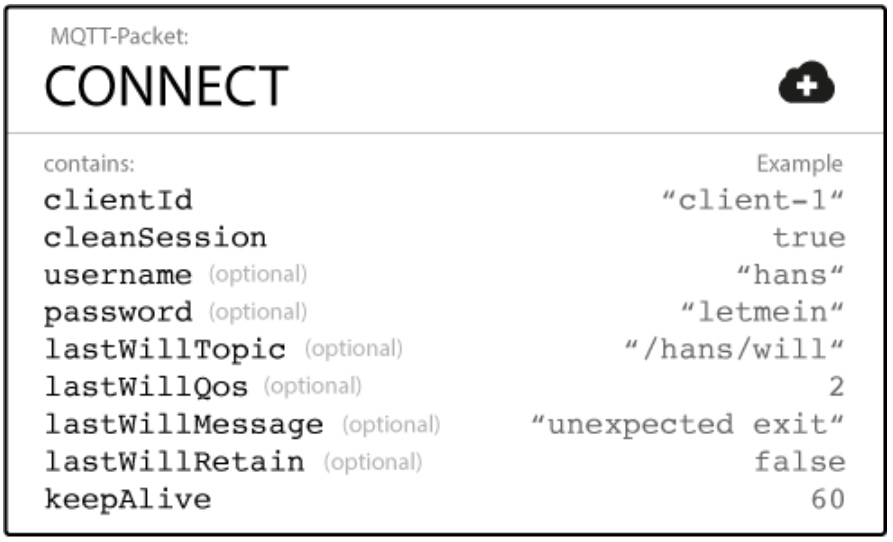
Un cliente puede establecer un mensaje Last Will and Testament (LWT) en el momento en el que conecta conecta con el Broker MQTT. Si el cliente no desconecta correctamente el Broker envía el mensaje LWT.

Cuando un cliente MQTT se conecta al servidor MQTT puede definir un tema y un mensaje que necesita ser publicado automáticamente sobre ese tema cuando se desconecta inesperadamente. Esto también se llama “Ultima voluntad y testamento” (LWT). Cuando el cliente se desconecta inesperadamente, el temporizador keep alive del lado del servidor detecta que el cliente no ha enviado ningún mensaje o el PINGREQ keep alive. Por lo tanto, el servidor publica inmediatamente el mensaje Will en el tema Will especificado por el cliente.

La función LWT puede ser útil en algunos escenarios. Por ejemplo, para un cliente MQTT remoto, esta función se puede utilizar para detectar cuando los dispositivos IoT salen de la red. La función LWT se puede utilizar para crear notificaciones para una aplicación que esté supervisando la actividad del cliente.



Paquete:



Ver explicación completa en: https://learn.adafruit.com/mqtt-adafruit-io-and-you/qos-and-wills

Más información:
https://www.ibm.com/developerworks/community/blogs/5things/entry/5_things_to_know_about_mqtt_the_protocol_for_internet_of_things?lang=en

Otros Conceptos MQTT

- Cada mensaje MQTT puede ser enviado como un mensaje con retención (**retained**), en este caso cada nuevo cliente que conecta a un topic recibirá el último mensaje retenido de ese tópico.
- Cuando un cliente conecta con el **Broker** puede solicitar que la sesión sea persistente, en ese caso el **Broker** almacena todas las suscripciones del cliente, todos los mensajes QoS 1 y 2 no procesados o perdidos por el cliente
- Un mensaje MQTT **CONNECT** contiene un valor keepAlive en segundos donde el cliente establece el máximo tiempo de espera entre intercambio de mensajes

Seguridad MQTT

Ya sabemos lo importante que es la seguridad, y más en escenarios IoT en el que comunican objetos entre sí. MQTT confía en tecnologías estándares para esto:

- Autenticación usuario/Password
- Seguridad SSL/TLS

Los puertos estándar son el 1883 para la comunicación no cifrada y el 8883 para la comunicación cifrada mediante SSL/TLS. Durante el handshake SSL/TLS, el cliente valida el certificado del servidor para autenticar el servidor. El cliente también puede proporcionar un certificado de cliente al broker durante el handshake, que el broker puede utilizar para autenticar al cliente. Aunque no forma parte específica de la especificación MQTT, se ha convertido en habitual que los broker admitan la autenticación de clientes con certificados SSL/TLS del lado del cliente.

Dado que el protocolo MQTTT pretende ser un protocolo para dispositivos con recursos limitados y de IoT, el SSL/TLS puede no ser siempre una opción y, en algunos casos, puede no ser deseable. En estos casos, la autenticación se presenta como un nombre de usuario y contraseña de texto claro que el cliente envía al servidor como parte de la secuencia de paquetes CONNECT/CONNNACK. Algunos brokers, especialmente los brokers abiertos publicados en Internet, aceptan clientes anónimos. En tales casos, el nombre de usuario y la contraseña simplemente se dejan en blanco.

Reto MQTT: Seguridad, Interoperabilidad y Autenticación

Debido a que el protocolo MQTT no fue diseñado con la seguridad en mente, el protocolo ha sido tradicionalmente utilizado en redes back-end seguras para propósitos específicos de la aplicación. La estructura temática de MQTT puede fácilmente formar un árbol enorme, y no hay una manera clara de dividir un árbol en dominios lógicos más pequeños que puedan ser federados. Esto dificulta la creación de una red MQTT globalmente escalable porque, a medida que crece el tamaño del árbol temático, aumenta la complejidad.

Otro aspecto negativo de MQTT es su falta de interoperabilidad. Debido a que las cargas útiles de mensajes son binarias, sin información sobre cómo están codificadas (sin metadatos), pueden surgir problemas, especialmente en arquitecturas abiertas en las que se supone que las diferentes aplicaciones de los diferentes fabricantes funcionan a la perfección entre sí.

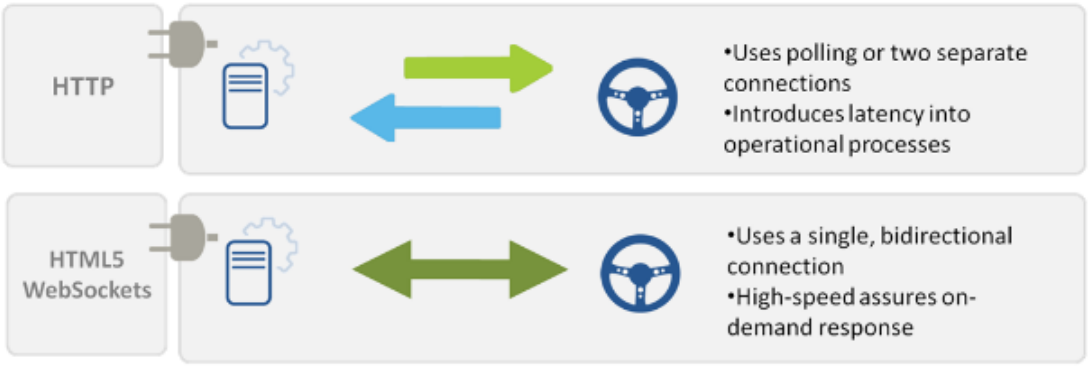
MQTT tiene características de autenticación mínimas incorporadas en el protocolo. El nombre de usuario y las contraseñas se envían en texto claro y cualquier forma de uso seguro de MQTT debe emplear SSL/TLS, que, lamentablemente, no es un protocolo ligero.

Autenticar clientes con certificados del lado del cliente no es un proceso simple, y no hay manera en MQTT, excepto el uso de medios propietarios fuera de banda, para controlar quién posee un topic y quién puede publicar información sobre él. Esto hace que sea muy fácil inyectar mensajes dañinos, ya sea intencionadamente o por error, en la red.

Además, no hay forma de que el receptor del mensaje sepa quién envió el mensaje original a menos que esa información esté contenida en el mensaje real. Las características de seguridad que tienen que ser implementadas sobre MQTT de forma propietaria aumentan la huella de código (footprint) y hacen que las implementaciones sean más difíciles.

MQTT vs HTTP (REST API)

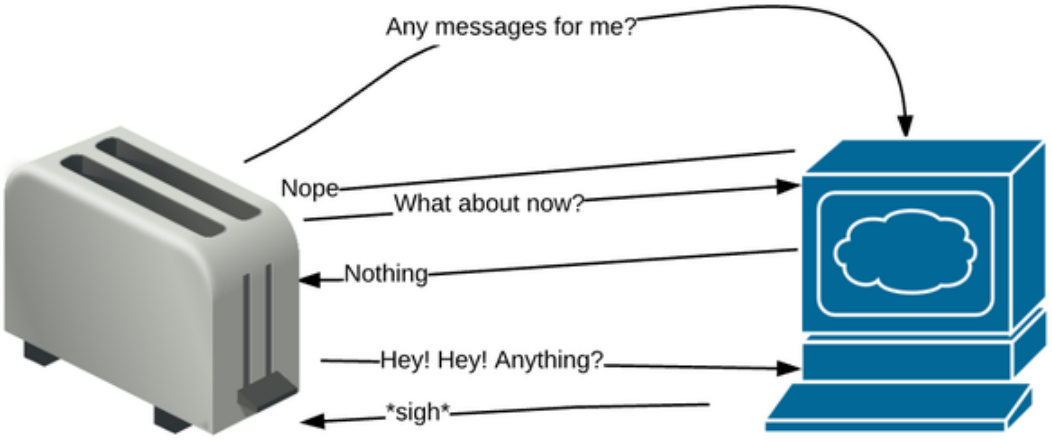
La diferencia entre una REST API y MQTT es que MQTT mantiene una conexión hacia el servicio abierta y puede responder mucho más rápido a los cambios en el feed. La REST API solo conecta al servicio cuando se hace una petición (request) y es más apropiada para proyecto donde el dispositivo permanece en modo sleep (para reducir el consumo) y despierta solo para mandar o recibir datos. (Push vs pull)



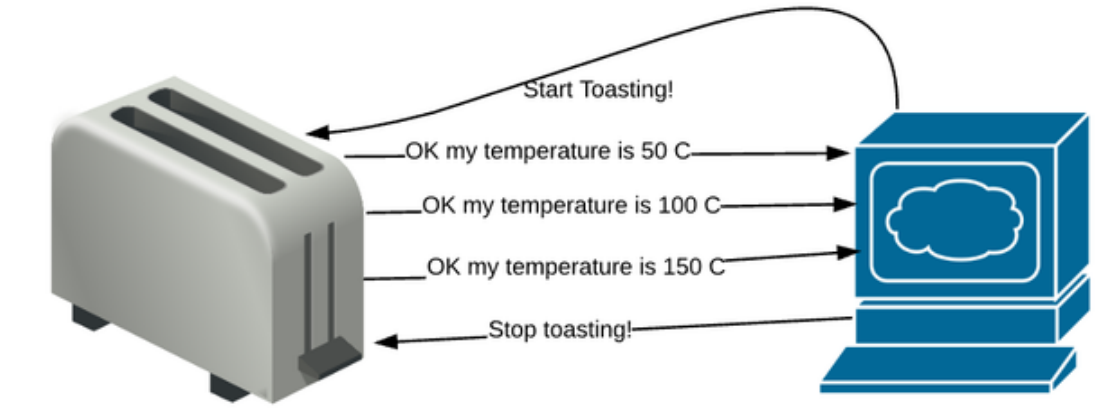
Push = websocket

Pull = REST API

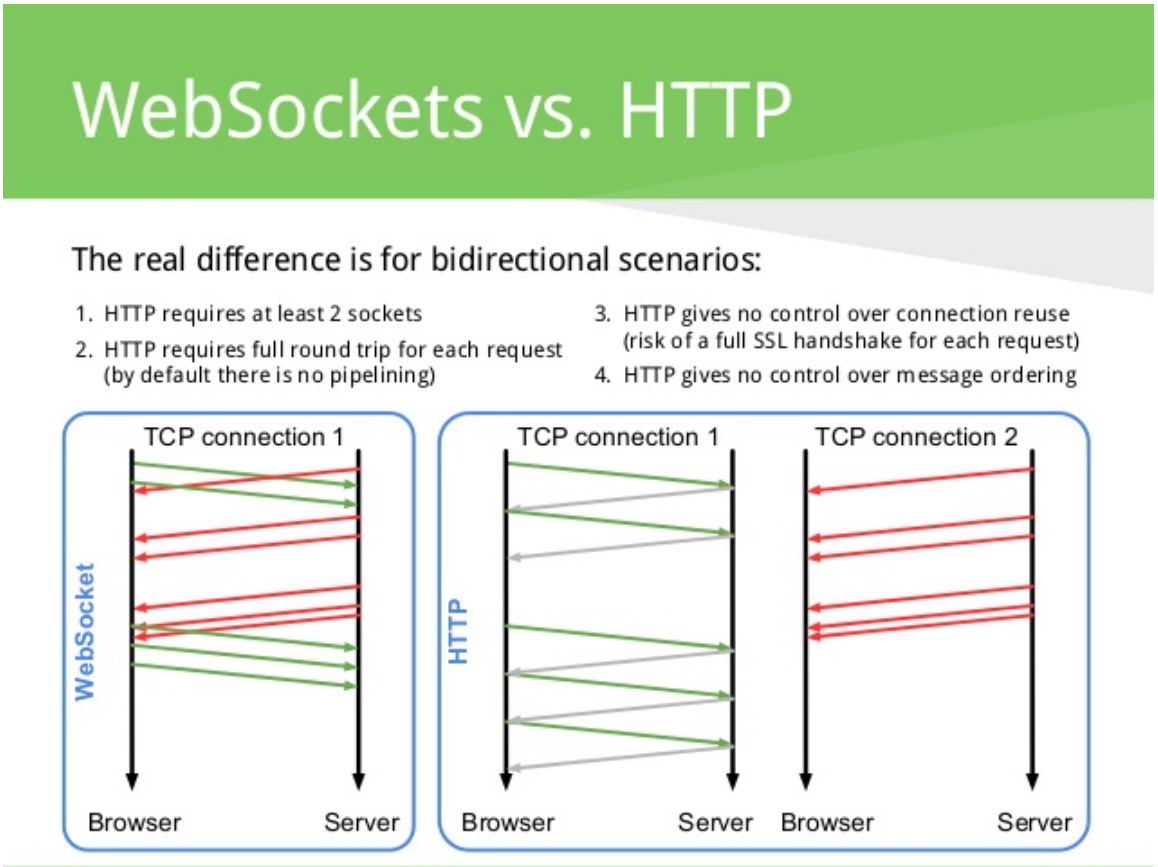
HTTP no tiene estado (stateless), por lo que debe tener una conexión por cada transferencia de datos: una conexión cada vez que desee escribir datos, una conexión para la lectura. HTTP es ideal para grandes cantidades de datos, como los que se utilizan para sitios web, y se puede utilizar para conexiones IoT. Pero no es ligero y no es muy rápido. Otro problema con HTTP es que es sólo pull.



MQTTT es un gran protocolo. Es extremadamente sencillo y ligero. La conexión a un servidor sólo tarda unos 80 bytes. Usted permanece conectado todo el tiempo, cada dato 'publicación' (datos push de un dispositivo a otro) y cada dato 'suscripción' (datos push de un servidor a otro) es de unos 20 bytes. Ambos ocurren casi instantáneamente.



MQTT puede funcionar sobre cualquier tipo de red, ya sea una red mesh, TCP/IP, Bluetooth, etc.



Si se usa MQTT usando Bluetooth, XBee, Bluetooth LE, LoRA u otro protocolo y dispositivo no conectado a Internet, ¡necesitarás una pasarela!

Ejemplo de gateway: https://learn.adafruit.com/datalogging-hat-with-flora-ble/

Clientes MQTT

Existen muchos clientes y librerías para MQTT, puesto que se trata de un protocolo libre sencillo de implementar.

Una aplicación de cliente MQTT se encarga de recopilar información del dispositivo de telemetría, conectar con el servidor y publicar la información en el servidor. También puede suscribirse a temas, recibir publicaciones y controlar el dispositivo de telemetría.

Cliente online: http://www.hivemq.com/demos/websocket-client/

Los mejores clientes MQTT: https://www.hivemq.com/blog/seven-best-mqtt-client-tools

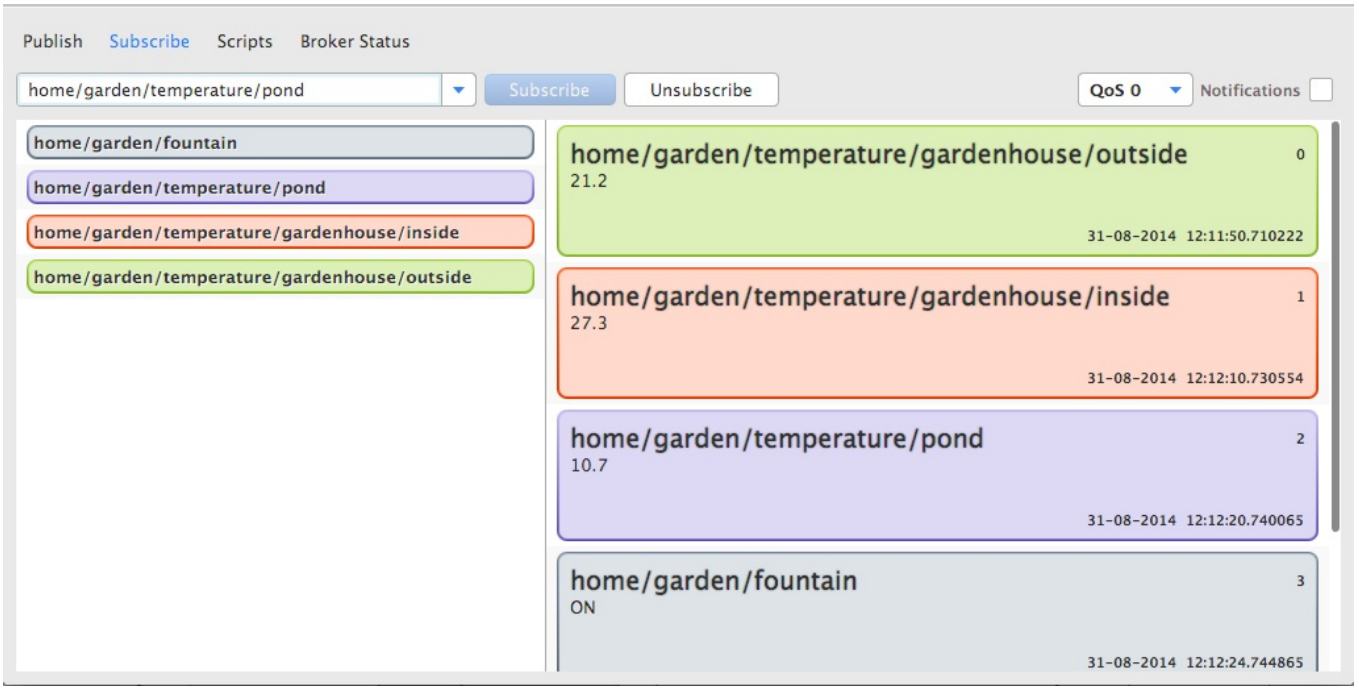
Tres herramientas MQTT y como simular MQTT: https://dzone.com/articles/top-3-online-tools-to-simulate-an-mqtt-client

Herramientas MQTT: https://www.hivemq.com/blog/overview-of-mqtt-client-tools

MQTT.fx

Uno de los clientes MQTT más populares para instalar en ordenador es MQTT.fx hecho en java y basado en Eclipse Paho http://www.eclipse.org/paho/

Está disponible para Windows, Linux y MAC



Web: https://mqttfx.jensd.de/

Descarga: http://www.jensd.de/apps/mqttfx/1.7.1/

Referencias: http://mqttfx.jensd.de/index.php/references

Más información:

- https://www.hivemq.com/blog/mqtt-toolbox-mqtt-fx
- https://learn.adafruit.com/desktop-mqtt-client-for-adafruit-io?view=all
- https://blogs.sap.com/2017/11/14/using-mqtt.fx-as-the-mqtt-simulation-tool-to-post-mqtt-messages-to-iot-service-4.0/

MQTT-Spy

MQTT-spy es una utilidad de código abierto destinada a ayudarle a monitorear la actividad sobre temas de MQTTT. Ha sido diseñado para tratar con grandes volúmenes de mensajes, así como con publicaciones ocasionales.

mqtt-spy es probablemente una de las utilidades de código abierto más avanzadas para publicar y monitorear actividades sobre temas de MQTT. Está dirigido a dos grupos de usuarios:

- Innovadores que necesitan una herramienta para crear prototipos de IO o proyectos de integración
- Usuarios avanzados que necesitan una utilidad avanzada para sus entornos de trabajo

Web: https://kamilfb.github.io/mqtt-spy/

Wiki: https://github.com/eclipse/paho.mqtt-spy/wiki

Descarga: https://github.com/eclipse/paho.mqtt-spy/wiki/Downloads

Getting Started: https://github.com/eclipse/paho.mqtt-spy/wiki/GettingStarted

Ver mqtt-spy como aplicación para probar un mosquitto: https://github.com/kamilfb/mqtt-spy/wiki/Overview

Resumen: https://github.com/kamilfb/mqtt-spy/wiki/Overview

Más información:

- https://www.hivemq.com/blog/mqtt-toolbox-mqtt-spy
- Advanced: https://www.hivemq.com/blog/mqtt-toolbox-mqtt-spy-advanced
- Daemon: https://www.hivemq.com/blog/mqtt-toolbox-mqtt-spy-daemon
- Versiones: https://github.com/eclipse/paho.mqtt-spy/releases

MQTT Explorer

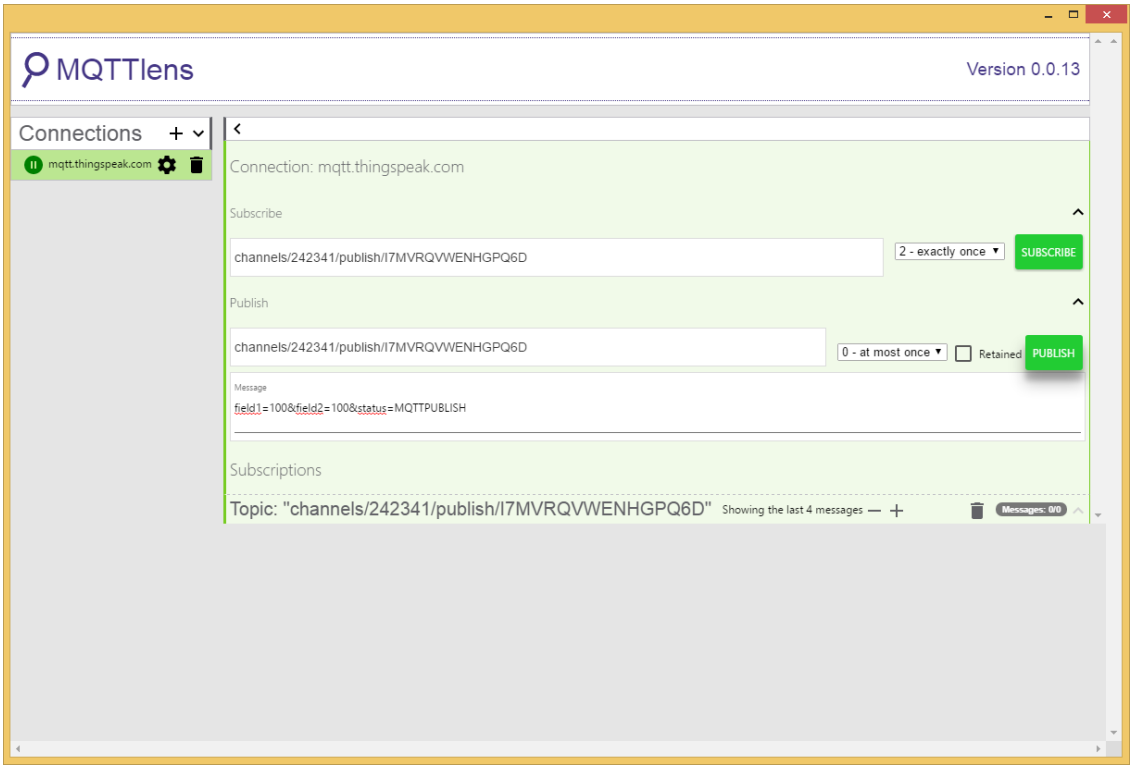
Web http://mqtt-explorer.com/

MQTT Lens

MQTT se puede instalar fácilmente a través de Google Chrome App Store. La herramienta tiene una interfaz bastante limpia y soporta todas las opciones de conexión disponibles desde la especificación MQTT, excepto las sesiones persistentes. Acepta conexiones a más de un broker al mismo tiempo y los colorea de manera diferente para facilitar su asociación.

La interfaz para suscribirse, publicar y ver todos los mensajes recibidos es simple y fácil de entender. Lamentablemente no hay posibilidad de publicar mensajes retenidos. Pero aunque esta aplicación se instala a través de Chrome, se ejecuta como una aplicación independiente.

Usando MQTT Lens:



Uso de MQTT lens: http://www.hivemq.com/blog/mqtt-toolbox-mqtt-lens

Descarga: https://chrome.google.com/webstore/detail/mqttlens/hemojaaeigabkbcookmlgmdigohjobjm

Clientes MQTT para móvil

Eclipse Paho Android Service: https://www.eclipse.org/paho/clients/android/

Algunos clientes Android (por orden de descargas):

- https://play.google.com/store/apps/details?id=net.routix.mqttdash
- https://play.google.com/store/apps/details?id=at.tripwire.mqtt.client
- https://play.google.com/store/apps/details?id=com.thn.iotmqttdashboard
- https://play.google.com/store/apps/details?id=snr.lab.iotmqttpanel.prod
- https://play.google.com/store/apps/details?id=in.dc297.mqttclpro&hl=es

IOS:

- https://itunes.apple.com/us/app/mqtt-buddy/id1252991874?mt=8
- http://1j2.com/ihometouch/

Clientes MQTT en Dispositivos embebidos

MQTT se puede usar desde diversos dispositivos como cliente mediante el uso de librerías:

- Arduino
- Python
- Clientes MQTT
- Raspberry Pi
- Autómatas (ver PLC de Unitronics y otros)
- Otros sistemas embebidos

MQTT y Arduino

MQTT ha surgido como un protocolo de mensajería estándar para la IoT. Se puede utilizar en redes TCP/IP y es muy ligero. La norma sigue un modelo de publicación-suscripción (“pub/sub”).

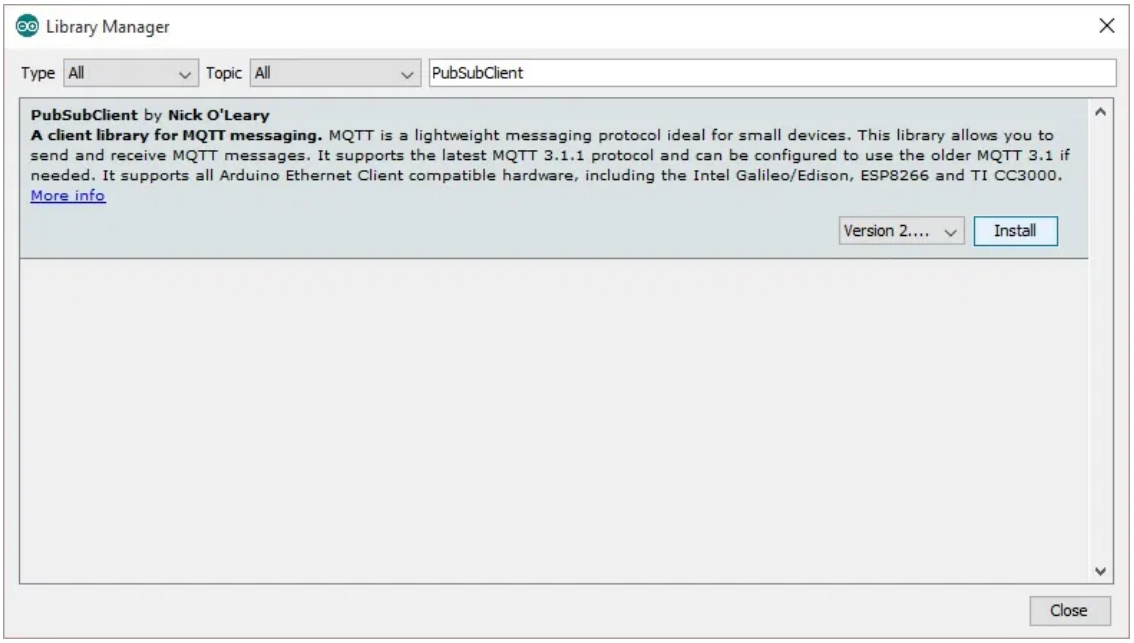
Como habrás imaginado, para conseguir una comunicación MQTT, emplearemos una librería. Existen muchas disponibles gracias a la gran (tanto en tamaño como en calidad) comunidad que existe alrededor de Arduino.

Una de las librerías más conocidas y la más estable y flexible es Arduino Client for MQTT http://pubsubclient.knolleary.net/ que nos provee de un sencillo cliente que nos permite tanto subscribirnos como publicar contenido usando MQTT. Internamente, usa la API de Arduino Ethernet Client lo que lo hace compatible con un gran número de shields y placas.

Web: https://pubsubclient.knolleary.net/

Repositorio: https://github.com/knolleary/pubsubclient

Esta librería está disponible en el gestor de librerías.



Documentación: https://pubsubclient.knolleary.net/api.html

Hardware compatible:

- Arduino Ethernet
- Arduino Ethernet Shield
- Arduino YUN – use the included YunClient in place of EthernetClient, and be sure to do a Bridge.begin() first
- Arduino WiFi Shield – if you want to send packets greater than 90 bytes with this shield, enable the MQTT_MAX_TRANSFER_SIZE option in PubSubClient.h.
- Sparkfun WiFly Shield – when used with this library
- Intel Galileo/Edison
- ESP8266
- ESP32

Getting started con esa librería: https://ricveal.com/blog/arduino-mqtt/

Más info de esta librería: https://www.hivemq.com/blog/mqtt-client-library-encyclopedia-arduino-pubsubclient/

Tutorial con esta librería MQTT, Arduino y bluemix:: https://www.ibm.com/developerworks/ssa/cloud/library/cl-bluemix-arduino-iot2/index.html

Otras Librerías MQTT para Arduino

A la hora de elegir una librería MQTT, debemos comprobar que funciona con el dispositivo HW Arduino y el HW de comunicación.

Otras librerías MQTT para Arduino:

- https://github.com/i-n-g-o/esp-mqtt-arduino
- https://github.com/adafruit/Adafruit_MQTT_Library

- https://github.com/256dpi/arduino-mqtt (Pendiente de probar)

Más información de uso librería Adafruit:

- Ejemplos de uso de esas librerías: https://github.com/SensorsIot/MQTT-Examples
- Buen video que explica todo: https://www.youtube.com/watch?v=9G-nMGcELG8
- Broker gratuito de Adafruit https://io.adafruit.com/aprendiendoarduino/dashboards
- Tutorial: https://learn.adafruit.com/mqtt-adafruit-io-and-you/intro-to-adafruit-mqtt

Seguridad MQTT con ESP 8266: https://io.adafruit.com/blog/security/2016/07/05/adafruit-io-security-esp8266/

Esta entrada se publicó en Arduino, Curso IoT Open Source, Librerías, Librerías Arduino, Mosquitto, MQTT, Protocolos, Websockets y está etiquetada con Arduino, Arquitectura MQTT, CoAP, Curso IoT Open Source, Escalado MQTT, HiveMQ, Librerías Arduino, Mosquitto, MQTT, MQTT Client, MQTT LWT, MQTT QoS, MQTT Topics, MQTT wildcards, MQTT.fx, Protocolos, Protocolos IoT, PubSubClient, Seguridad MQTT, Websockets en 19 noviembre, 2018 [https://aprendiendoarduino.wordpress.com/2018/11/19/mqtt/] .

---

7 pensamientos en “MQTT”

Pingback: Saber más de IoT... | Aprendiendo Arduino

Naguissa
9 octubre, 2019 en 5:38 pm

Hola,

Genial este artículo sobre MQTT, muy completo. Sabía lo que era pero normalmente lo he usado con Arduino y en mi cabeza lo tenía limitado a ello.

Quería comentar que, debido a mis proyectos, he montado un servidor MQTT en un servidor de Internet. Y ya que estaba, lo he puesto a disposición pública. Para que no haya problemas de acceso todos los usuarios tienen que autenticarse y, además, el topic siempre empieza con una key de su usuario.

Pero creo que es super práctico para hacer pruebas o para montar proyectos sin necesidad de tener un servidor (o una Raspberry) en casa dedicado a ello.

La información está disponible en: https://www.foroelectro.net/arduino/es/mqtt-doc

Saludos

jecrespom Autor de la entrada
10 octubre, 2019 en 8:40 am

Muchas gracias por la información, lo probaré...

Naguissa
10 octubre, 2019 en 2:23 pm

Si deseas una prueba rápida publiqué la que hice yo con un ESP8266 y el IDE Arduino:
https://www.foroelectro.net/servicios-de-la-web-f28/aservidor-mqtt-v3-1-disponible-t155.html#p833

Con las herramientas del artículo puedes hacer las pruebas; funciona genial.

Pingback: Saber más de IoT 2019... | Aprendiendo Arduino

Pingback: Demo IoT | Aprendiendo Arduino

Pingback: Saber Más Node-RED Developer | Aprendiendo Arduino

Este sitio usa Akismet para reducir el spam. Aprende cómo se procesan los datos de tus comentarios .