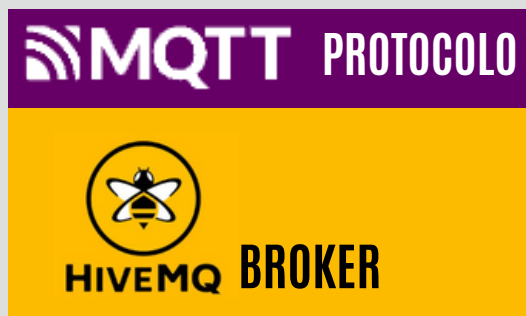


# SISTEMAS DISTRIBUIDOS

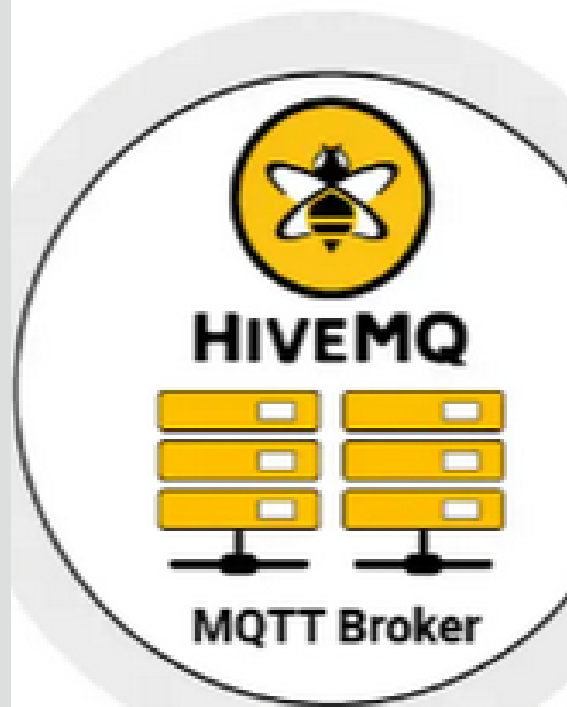
## LICENCIATURA EN INFORMATICA



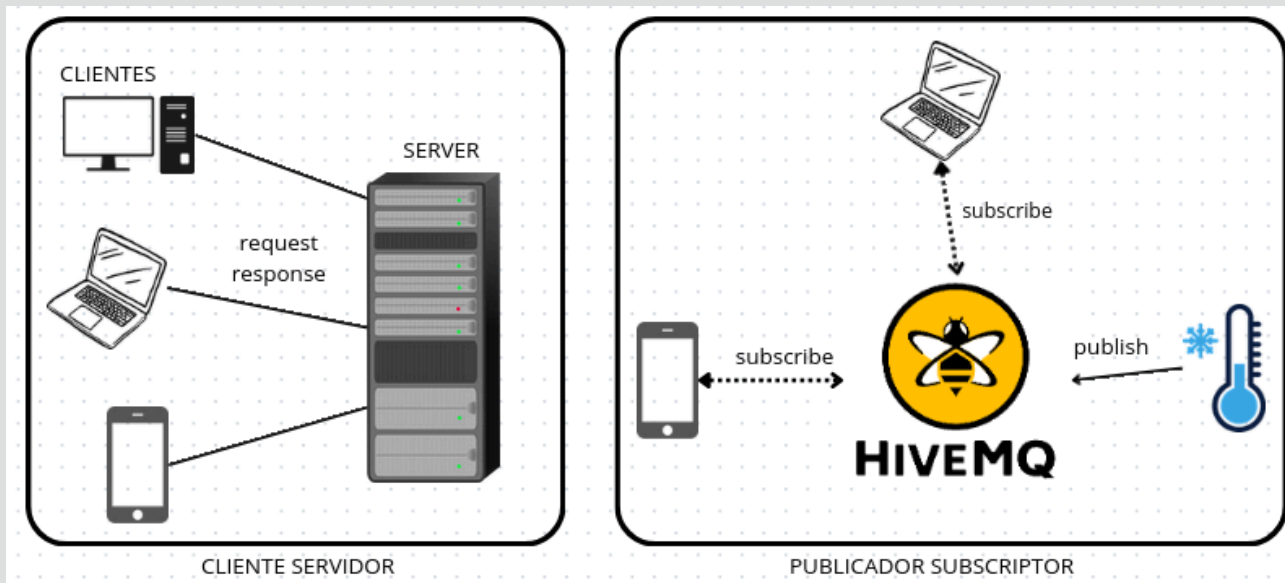
ALUMNO:  
WALDO FUSIMAN

# Contenidos

Introduction	02
Historia	04
Caracteristicas	06
Arquitectura	08
Componentes	10
Temas	13
Calidad de servicio	16
Sesiones	18
Mensajes retenidos	20
Ultima voluntad	21
KeepAlive	22
MQTT5	24
Aplicacion prueba	26
Conclusiones	27
Referencias	28



## SISTEMAS DISTRIBUIDOS



# Introducción

MQTT es un protocolo de mensajería sencillo diseñado originalmente para la comunicación en redes con ancho de banda y recursos computacionales limitados. Desarrollado con simplicidad y escalabilidad en mente, MQTT es particularmente adecuado para aplicaciones de Internet de las cosas (IoT), donde la variedad y cantidad de dispositivos están creciendo exponencialmente.

Es un protocolo de transporte de mensajes del tipo publicación/suscripción.

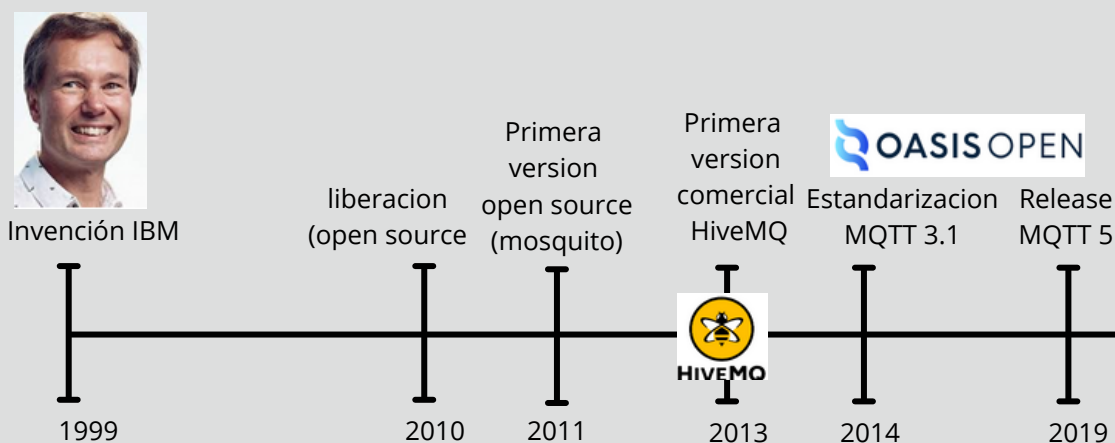
En la comunicación de red tradicional, los clientes y servidores se comunican directamente entre sí. Los clientes solicitan recursos o datos al servidor, luego el servidor procesa y envía una respuesta. Sin embargo, MQTT utiliza el patrón publicación / suscripción para desacoplar el remitente del receptor del mensaje. En lugar de ello, un tercer componente, denominado broker de mensajes, controla la comunicación entre editores y suscriptores. El trabajo del broker consiste en filtrar todos los mensajes entrantes de los editores y distribuirlos correctamente a los suscriptores.

## SISTEMAS DISTRIBUIDOS

MQTT es utilizado principalmente en IOT y aplicaciones M2M (machines to machine). Lo utilizan en la industria de la automoción, telecomunicaciones, energía, seguridad pública, y conectividad, empresas como BMW, Air France, Libertad global, Mercedes Benz, Hitera y Matternet.

Algunos ejemplos son:

- Casas inteligentes: MQTT se utiliza para conectar varios dispositivos en una casa inteligente, como termostatos, lámparas, cámaras de seguridad, y otros electrodomésticos inteligentes, permitiendo a los usuarios controlar de forma remota mediante una aplicación móvil.
- Automatización industrial: utilizado para conectar máquinas y sensores en fábricas y sectores industriales, para permitir monitoreo en tiempo real de los procesos.
- Agricultura: en agricultura de precisión para monitorear niveles de humedad en suelo, condiciones climáticas y crecimiento de cultivos, para ayudar en la optimización de riego y otras prácticas.
- Atención sanitaria: para conectar dispositivos médicos y sensores, como medidores de glucosa, y monitoreo de frecuencia cardiaca, permitiendo el monitoreo remoto de pacientes.
- Transporte: automóviles conectados y otros sistemas de transporte que permiten el seguimiento y control de vehículos en tiempo real, permitiendo mejorar en seguridad y ayuda a optimizar tráfico.



## Historia

En 1999 Andy Stanford-Clark de IBM y Arien Nipper de Arcom, desarrollaron el protocolo MQTT para IBM para la industria del petróleo y gas, el propósito era minimizar las pérdidas de batería y minimizar de ancho de banda al conectarse y supervisar oleoductos vía satélite.

Originalmente las siglas eran el acrónimo de MQ Telemetry transport, donde MQ hacía referencia a un producto de IBM llamando MQ Series, para soporte de telemetría, actualmente solo es el nombre del protocolo.

Los requisitos que especificaron inicialmente para el protocolo fueron:

- de implementación sencilla.
- Entrega de datos y calidad de servicio
- Liviano y eficiente en el uso de la red.
- Datos agnósticos.
- Que permita sesión continua.

En la actualidad estos objetivos siguen siendo la base de MQTT, sin embargo el enfoque principal del protocolo ha cambiado de los sistemas integrados propietarios al uso abierto de Internet de las cosas (IOT).

IBM utilizó internamente el protocolo durante 10 años y en 2010 lanzó la versión MQTT 3.1 como versión libre de regalías.

En 2011 IBM contribuyó con implementaciones de clientes MQTT al proyecto Paho de la fundación Eclipse, lo que permitió crear un ecosistema solidario, al permitir acceder a los desarrolladores al protocolo y crear sus aplicaciones en base a este. Esto ayudó al crecimiento y adopción del protocolo en la comunidad de desarrolladores.

## SISTEMAS DISTRIBUIDOS

En 2012 HiveMQ creó la primera versión de su software basado en el protocolo MQTT, que fue lanzado en 2013.

En 2013 se envió al organismo de especificación de la Organización para el Avance de Estándares de Información Estructurada (OASIS) para su mantenimiento.

En 2014 OASIS anunció que se haría cargo de la estandarización de MQTT, con el objetivo de convertirlo en un protocolo abierto e independiente de los proveedores.

El proceso de estandarización llevó alrededor de un año y el 29 de octubre de 2014, MQTT se convirtió en un estándar OASIS oficialmente aprobado. MQTT version 3.1.1.

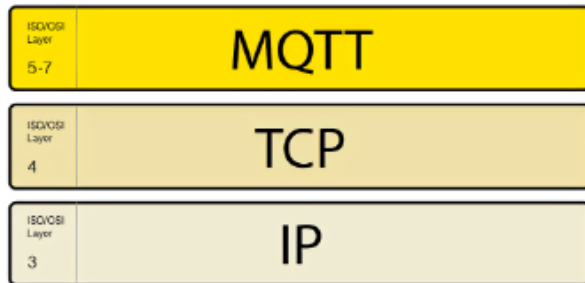
Como organización independiente y neutral, OASIS garantiza que el protocolo se mantenga como un estándar abierto que puede ser implementado por cualquier persona sin tarifas de licencia ni restricciones de propiedad.

Además de ofrecer un foro para que la comunidad se reúna y colabore en las mejoras del protocolo, lo que ha resultado en el desarrollo de MQTT 5, la última versión del protocolo con nuevas características para mejorar la confiabilidad y la escalabilidad.

En marzo de 2019, OASIS ratificó la nueva especificación MQTT 5.

Esta versión introdujo nuevas características a MQTT requeridas para aplicaciones de IoT implementadas en plataformas en la nube y casos que requieren mayor confiabilidad y manejo de errores en la implementación de mensajería de misión crítica, además de mejorar el rendimiento, aumentar la confiabilidad y proporcionar un mayor control sobre comunicación entre clientes y servidores, informe de errores, escalabilidad y soporte mejorado para mensajes sin conexión.

## SISTEMAS DISTRIBUIDOS



Bit	7	6	5	4	3	2	1	0
byte 1	MQTT Control Packet type				Flags specific to each MQTT Control Packet type			
byte 2...	Remaining Length							

## CARACTERISTICAS

Muchos escenarios de IoT tienen conectividad, como redes móviles, donde la comunicación es buena, pero puede presentar interrupciones, por ej. si un auto con conectividad pasa por debajo de un túnel la conexión se pierde frecuentemente, y luego vuelve a conectarse. MQTT está construido para esta clase de escenarios.

### Características:

- Protocolo de mensajes para IoT (Internet de las cosas)
- Ejecuta sobre TCP: utiliza los beneficios del protocolo orientado a la conexión.
- Minimiza sobrecarga: formato de mensaje de longitud variable.
- Es simple de implementar
- Diseñado para comunicaciones confiables sobre canales no confiables.
- Binario: al utilizar un formato binario, el protocolo puede minimizar la cantidad de datos que se deben transmitir y reducir la potencia de procesamiento necesaria para interpretar los mensajes.
- Eficiente: paquetes MQTT pequeños. 2 bytes a 256mb máximo.
- Bidireccionales: mensajes en cualquier dirección.
- Independiente de los datos: no se indica lo que se envía, pueden ser xml, json, o formatos propios, inclusive imágenes y videos por fragmentos.

## SISTEMAS DISTRIBUIDOS

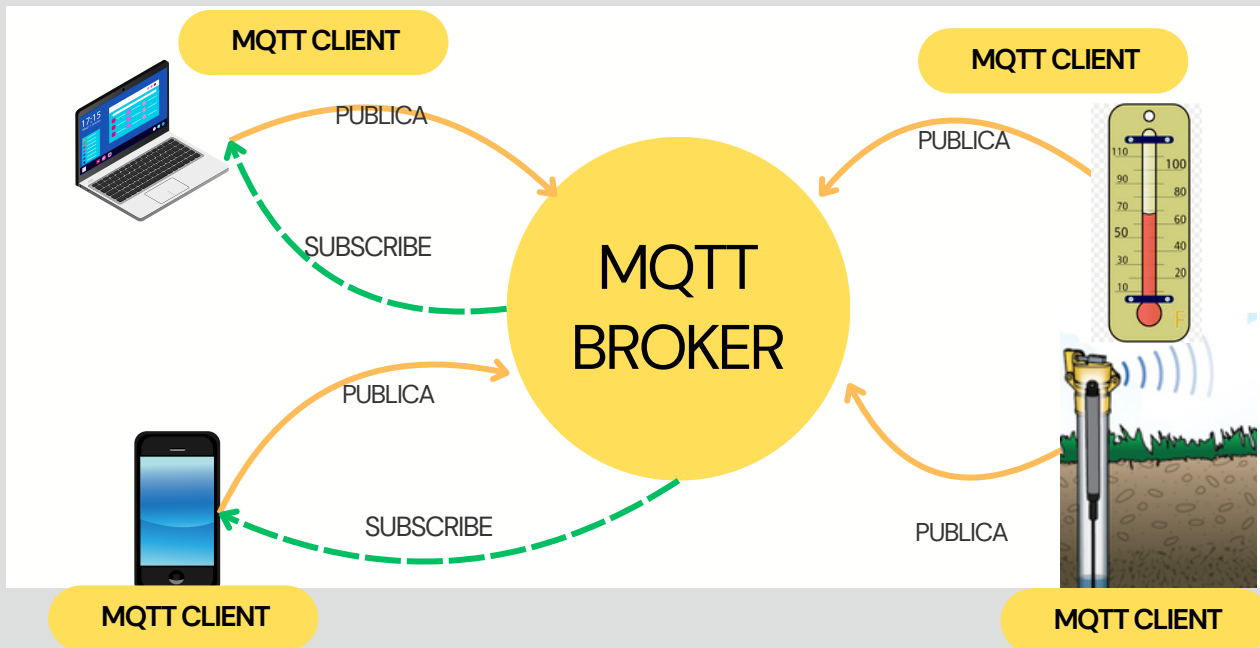
- Escalable: HiveMQ registra clientes con 10 millones de dispositivos a la vez.
- Construido para comunicaciones push: menor latencia, ej. un dispositivo envía datos a un broker en la nube. El broker enviara los datos solo a los dispositivos que necesiten lo necesiten, no a todos.
- Liviano: adecuado para dispositivos con limitaciones: no se necesita computadoras potentes y la memoria puede ser restringida.
- Entrega de mensaje confiables: especifica diferentes niveles de calidad de servicio (QoS) para asegurar entrega de mensajes confiables.
- Buffering de mensajes y reanudación de sesión: soporta sesiones persistentes entre dispositivos y servidores, mejorando la fiabilidad de los mensajes, asegurando que los mensajes son entregados al cliente aun después de desconexiones.

### Sobre TCP

- MQTT requiere TCP/IP
- Utiliza conexiones tcp persistentes
- Mecanismo de latido: para detectar problemas de sockets, servidor o dispositivos que pierden conexión.
- TLS: seguridad a nivel de transporte: conexiones encriptadas.
- Sobre TCP ofrece transferencia confiable, ordenada y con control de errores



## SISTEMAS DISTRIBUIDOS



# ARQUITECTURA

MQTT se ejecuta sobre TCP/IP utilizando el patron publicador/subscriptor. En la arquitectura MQTT, existen dos tipos de sistemas: clientes y brókeres. Un bróker es el servidor con el que se comunican los clientes: recibe comunicaciones de unos y se las envía a otros. Los clientes no se comunican directamente entre sí, sino que se conectan con el bróker. Cada cliente puede ser un publicador, un suscriptor o ambos. MQTT es un protocolo controlado por eventos, donde no hay transmisión de datos periódica o continua. Así se mantiene el volumen de transmisión al mínimo. Un cliente sólo publica cuando hay información para enviar, y un bróker sólo envía información a los suscriptores cuando llegan nuevos datos.

## SISTEMAS DISTRIBUIDOS

### Características del patron publicador/subscriptor

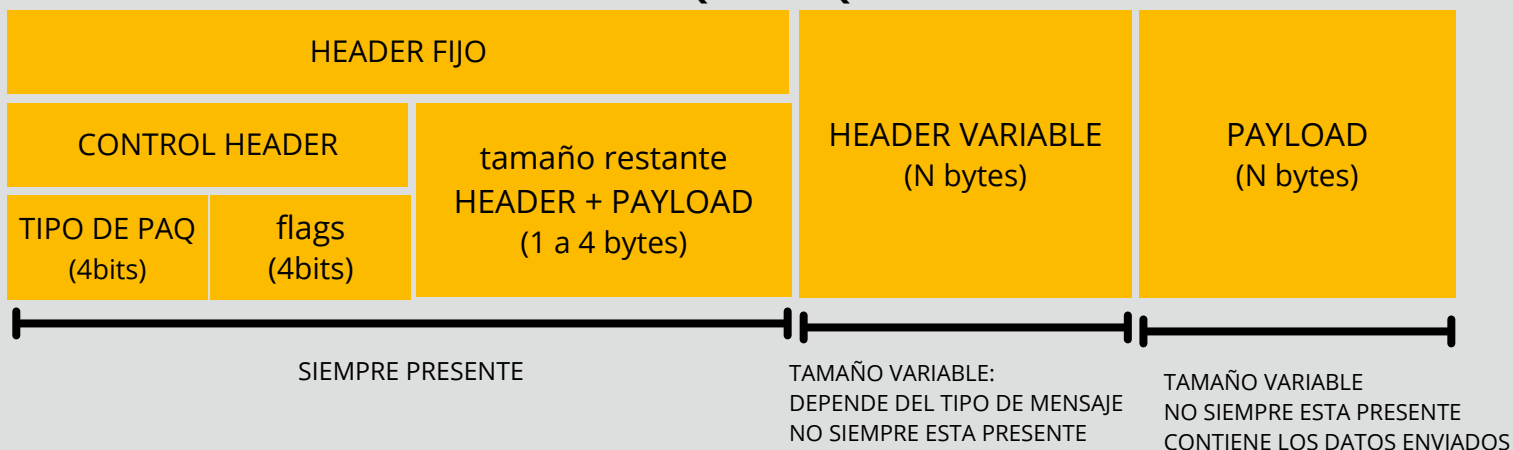
- desacoplamiento espacial: clientes publicadores y subscriptores totalmente independientes físicamente.
- Desacoplamiento en el tiempo (encolado): los datos son entregados a los subscriptores cuando estan en linea.
- Desacoplamiento sincronizado: los cliente publicadores y subscriptores no necesitan esperarse unos a otros.

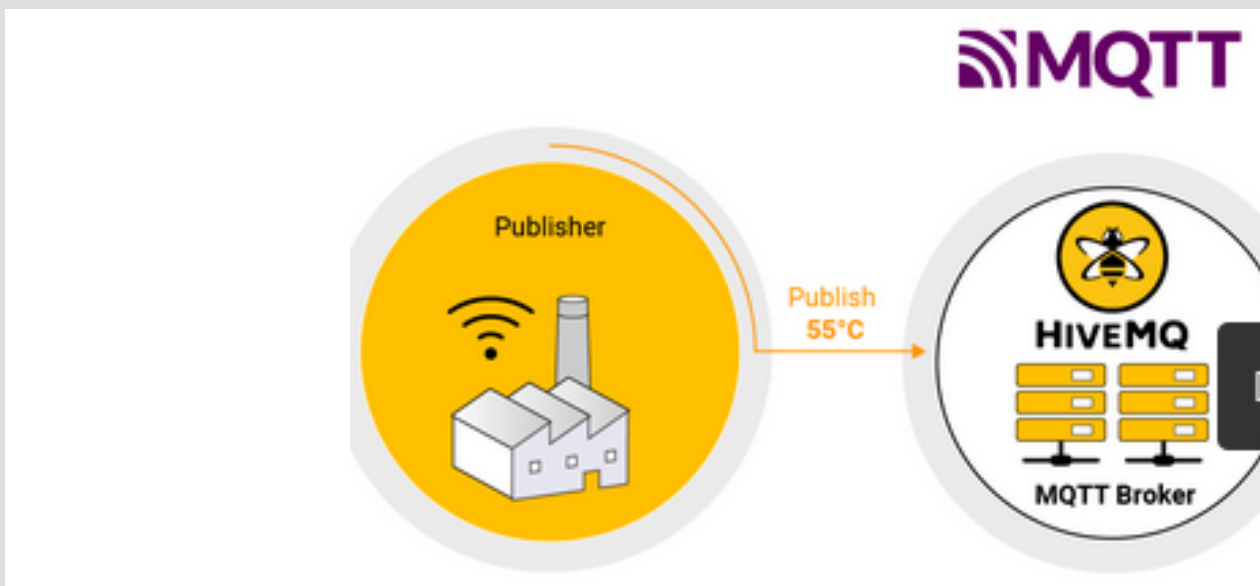
Una ventaja del patron es la escalabilidad. Este patron escala mucho mejor que el patron cliente servidor. Esto es especialmente cierto si se tiene un cluster de brokers (como hiveMQ) que forman un solo broker virtual.

Una desventaja de este patron es que tiene un solo punto de falla (SPOF) (el broker), si el broker no esta disponible ni los clientes publicadores podarn enviar datos ni los subscriptores podran recibir. Esto se atenúa si se tiene un cluster de brokers ya que se evita un solo punto de falla.

Otra forma en que MQTT minimiza sus transmisiones es con un tamaño de mensaje pequeño y bien definido. Cada mensaje tiene un encabezado fijo de apenas 2 bytes. Se puede utilizar un encabezado opcional, pero eso incrementa el tamaño del mensaje. La carga útil del mensaje está limitada a únicamente 256 MB

### TAMAÑO DE PAQUETE MQTT





## COMPONENTES

En el núcleo de MQTT se encuentran los brokers MQTT y los clientes MQTT.

### Broker MQTT

Es el sistema de back-end que coordina los mensajes entre los diferentes clientes. Las responsabilidades del broker incluyen recibir y filtrar mensajes, identificar a los clientes suscritos a cada mensaje y enviarles los mensajes.

Además también se encarga de otras tareas como: autenticación y autorización de clientes, pasar mensajes a otros sistemas para posterior análisis, y control de mensajes perdidos y sesiones de clientes

Existen gran cantidad de brokers MQTT open source y comerciales, disponibles con sus características, ventajas e inconvenientes:

- Mosquito: broker liviano, desarrollado por la Fundación Eclipse, programado en C y multiplataforma.
- Mosca: broker para NodeJS, desarrollado en javascript.
- HBMQTT: broker escrito en Python.
- EMQTT: desarrollado en Erlang/OTP, para aplicaciones con grandes exigencias de escalabilidad.
- RabbitMQ: broker de mensajería AMQP, que también permite emplear el protocolo MQTT a través de un adaptador.
- ActiveMQ: broker de mensajería JMS (java message script) desarrollado por Apache, también admite protocolo MQTT.
- Moquete: desarrollado por eclipse escrito en java.
- MQTTNet: broker para .NET.
- HiveMQ: broker basado en java

## Cliente MQTT

Es cualquier dispositivo que se comunice mediante MQTT a través de una red puede denominarse dispositivo cliente MQTT, desde un servidor hasta un microcontrolador que implementa una biblioteca MQTT.

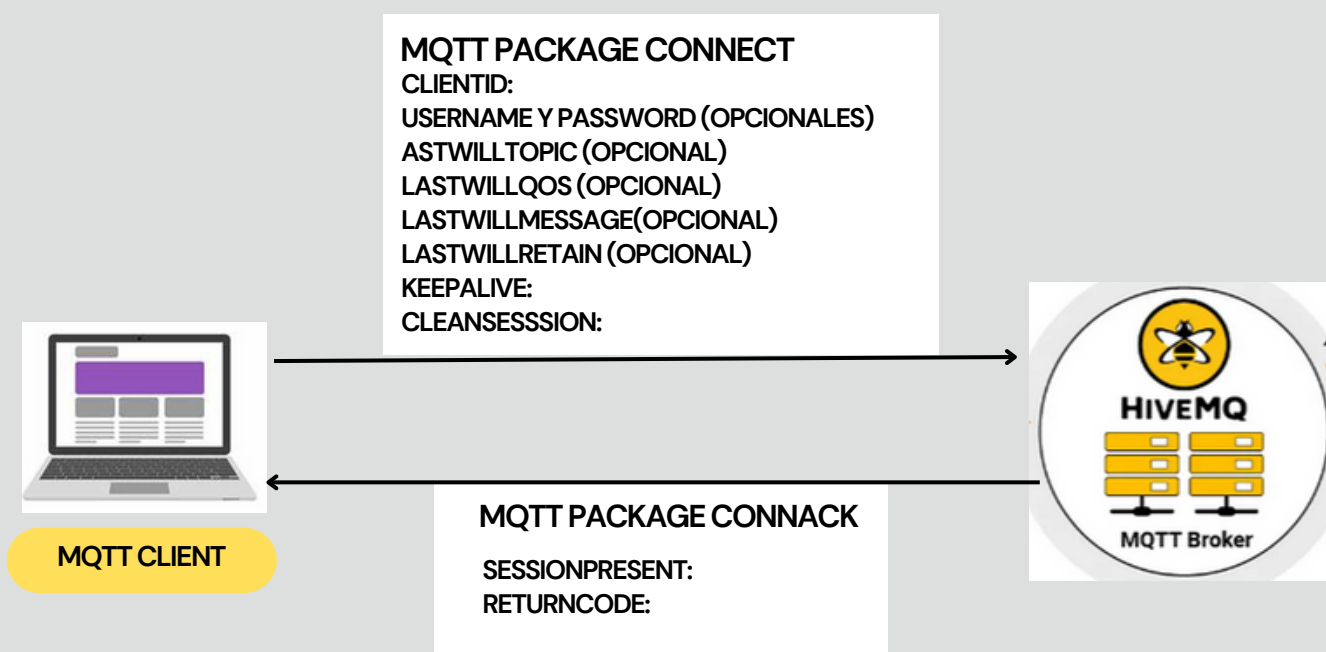
Si el cliente envía mensajes actúa como publicador, si recibe actúa como suscriptor.

Existen bibliotecas de clientes MQTT de código abierto en varios lenguajes de programación, como: el ecosistema elipse paho con librerías en C, java, C#, C++, o python o el ecosistema HiveMQ a con librerías java desarrollada junto a bmw.

## Conexión MQTT

MQTT utiliza conexión TCP/IP para conectarse a los brokers. utilizando este protocolo orientado a la conexión y con detección de errores, garantiza que los paquetes son recibidos en orden.

Los clientes y los brokers comienzan a comunicarse mediante una conexión MQTT. Los clientes inician la conexión al enviar un mensaje CONNECT al broker MQTT. El broker confirma que se ha establecido una conexión al responder con un mensaje CONNACK. Tanto el cliente MQTT como el broker requieren una pila TCP o IP para comunicarse. Los clientes nunca se conectan entre sí, sólo con el broker.

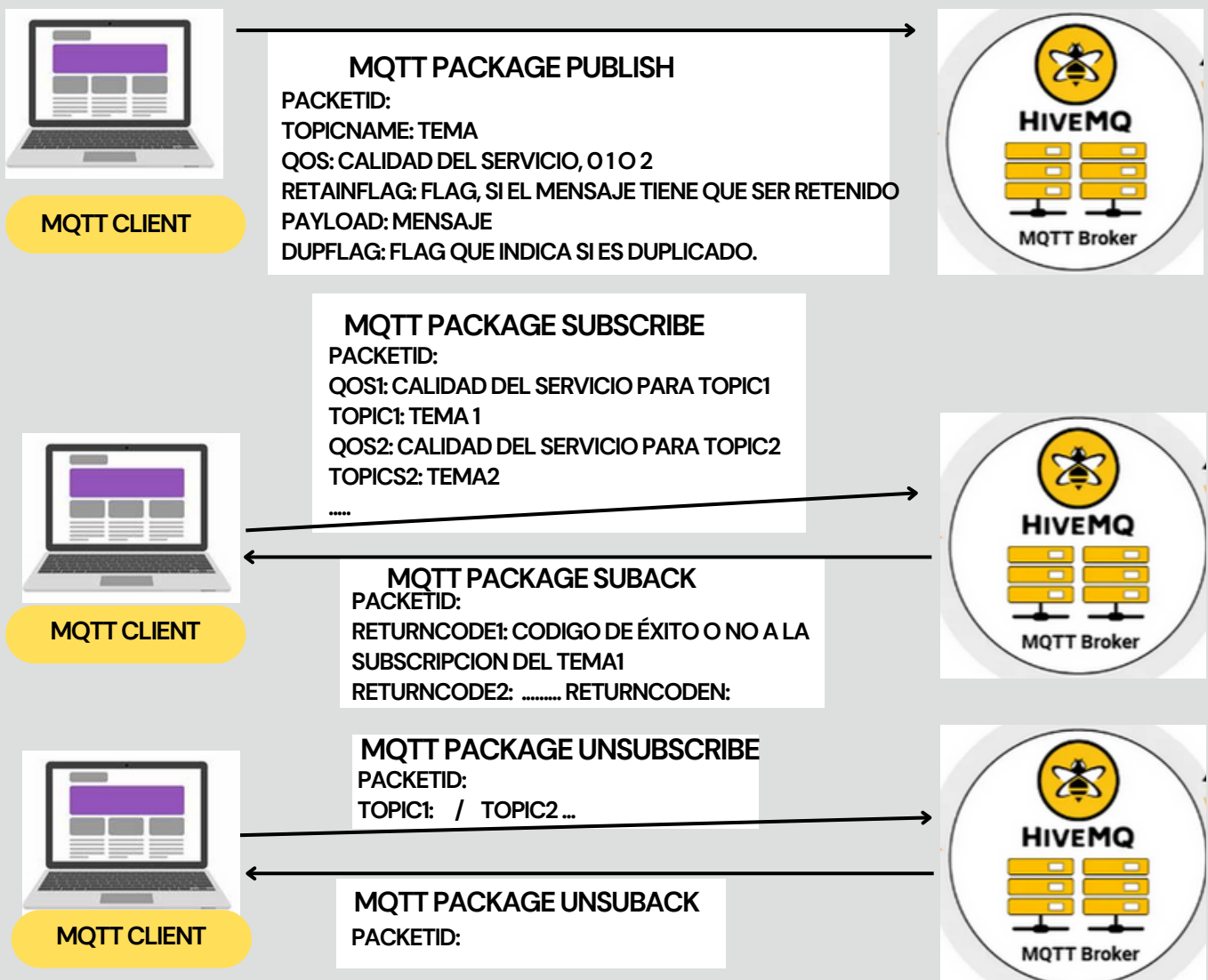


## Funcionamiento de MQTT

- Un cliente MQTT establecer una conexión con el broker MQTT.
- Una vez conectado, el cliente puede publicar mensajes, suscribirse a mensajes específicos o hacer ambas cosas.
- Cuando el agente MQTT recibe un mensaje, lo reenvía a los suscriptores que están interesados.

### Algunos puntos importantes:

- Los clientes no tienen direcciones (como por ej. direcciones de email), y los mensajes no son enviados a los clientes.
- Los mensajes son publicados en el broker con un tema específico.
- El broker realiza el filtrado de mensajes de acuerdo a los temas, y se encarga de distribuirlos a los suscriptores.
- No hay conexión directa entre publicadores y suscriptores.
- Todos los clientes pueden publicar y suscribirse (recibir).
- Mqtt brokers normalmente no almacenan mensajes.





## Temas

El término “tema” se refiere a las palabras claves que utiliza el broker MQTT a fin de filtrar mensajes para los clientes MQTT.

Los mensajes dentro de MQTT se publican como temas

Los temas están organizados jerárquicamente, de forma similar a un directorio de archivos o carpetas en un sistema de archivos, utilizan el carácter de barra (/) como delimitador. Por ejemplo, un sistema doméstico inteligente que funciona en una casa de varios pisos que tiene diferentes dispositivos inteligentes en cada uno de ellos. En ese caso, es posible que el broker MQTT organice temas como:

*hogar/plantabaja/living/luz*

*hogar/primerpiso/cocina/temperatura*

Una jerarquía de temas como *sensores/gas/presión* permite a un suscriptor especificar al broker que sólo le deben enviar datos de clientes que publican al tema presión o, para una vista más amplia, tal vez los datos de todos clientes que publican a cualquier tema de sensor/gas.

Los temas no se crean explícitamente en MQTT. Si el broker recibe datos publicados sobre un tema que actualmente no existe, simplemente se crea dicho tema y los clientes pueden suscribirse al mismo.

Comodines

+ : ej. *hogar/pb/+/temperatura* : indica que en este nivel puede ser cualquier subtema que derive de pb al tenga de subtema a temperatura.

*hogar/pb/living/temperatura* → ok

*hogar/primerpiso/living/temperatura* → X

# : ej. *hogar/pb/#* : puede ser cualquier subtema despues del #

*hogar/pb/living/temperatura* → ok

*hogar/primerpiso/living/temperatura* → X

### Publicación MQTT

Los clientes MQTT publican mensajes que contienen el tema y los datos en formato de bytes. El cliente determina el formato de los datos, como datos de texto, datos binarios, archivos XML o JSON.. Estos datos son específicos de cada implementación, pero pueden ser algo tan simple como una indicación de encendido/apagado o un valor de un determinado sensor, como temperatura, presión, etc.

### Suscripción MQTT

Los clientes MQTT envían un mensaje SUBSCRIBE al broker MQTT para recibir mensajes sobre temas de interés. Se puede suscribir a temas en concreto o mediante comodines, que permiten suscripciones a toda una rama de temas o a parte de ella. Para suscribirse, un cliente envía un paquete SUBSCRIBE y recibe un paquete SUBACK a cambio. Si hay un mensaje retenido para el tema, el nuevo suscriptor también lo recibe.

### Ping

un cliente puede hacer ping al bróker. El suscriptor envía un paquete PINGREQ y, como respuesta, se recibe un paquete PINGRESP. Se pueden utilizar pings para garantizar que la conexión siga funcionando y que la sesión TCP no haya sido cerrada inesperadamente por otro equipo de red, como un router o una puerta de enlace.

### Desconectar

un suscriptor o publicador puede enviar un mensaje de DISCONNECT al bróker. Este mensaje informa al bróker de que ya no necesitará enviar o poner en cola mensajes para un suscriptor y que ya no recibirá datos de un publicador. Este tipo de cierre permite al cliente volver a conectarse utilizando la misma identidad de cliente que en ocasiones anteriores. Cuando un cliente se desconecta sin enviar un mensaje de desconexión, se envía su última voluntad y testamento a los suscriptores.

## Seguridad

El objetivo original del protocolo MQTT era hacer posible la transmisión de datos de una forma más pequeña y eficiente a través de líneas de comunicación costosas y poco fiables. Como tal, la seguridad no fue una de las principales preocupaciones durante el diseño e implementación de MQTT.

Sin embargo, hay algunas opciones de seguridad disponibles a costa de una carga superior en la transmisión de datos y una mayor impronta.

- **Seguridad de red:** si la red en sí puede protegerse, la transmisión de datos inseguros en MQTT es irrelevante. En tal caso, los problemas de seguridad tendrían que producirse desde el interior de la propia red, quizás a través de un actor malicioso u otra forma de penetración en la red.
- **Nombre de usuario y contraseña:** MQTT permite nombres de usuario y contraseñas para que un cliente establezca una conexión con un bróker. Lamentablemente, los nombres de usuario y las contraseñas se transmiten en texto sin cifrar. En 1999, esto era más que suficiente porque interceptar una comunicación por satélite para lo que era esencialmente una lectura de sensor sin importancia habría sido muy difícil. Sin embargo, hoy en día, la interceptación de muchos tipos de comunicaciones de red inalámbrica se ha convertido en algo trivial, lo que hace que dicha autenticación sea casi inútil. Muchos casos de uso requieren un nombre de usuario y una contraseña ya no como protección contra actores maliciosos, sino como una forma de evitar conexiones involuntarias.
- **SSL/TLS:** al ejecutarse sobre TCP/IP, la solución obvia para proteger las transmisiones entre clientes y brókeres es la implementación de SSL/TLS. Lamentablemente, esto añade una sobrecarga sustancial a comunicaciones muy livianas.





## CALIDAD DE SERVICIO (QoS)

El nivel de Calidad de Servicio (QoS) es un acuerdo entre el emisor de un mensaje y el receptor que define la garantía de entrega para un mensaje específico

QoS es una característica clave del protocolo MQTT, que da al cliente el poder de elegir un nivel de servicio que se adapte a su fiabilidad de la red y la lógica de la aplicación. Dado que MQTT gestiona la retransmisión de mensajes y garantiza la entrega (incluso cuando el transporte subyacente no es fiable), QoS facilita enormemente la comunicación en redes poco fiables.

TCP nos da garantías de entrega mientras esté activa la misma conexión TCP

El problema con dispositivos IoT es que se puede perder temporalmente la conexión, las garantías de MQTT duran más tiempo que la conexión TCP real.

por lo tanto, la garantía de entrega de MQTT abarca incluso varias conexiones TCP, de modo que un mensaje puede tener garantías de entrega, incluso si tiene un dispositivo que interrumpe las conexiones con frecuencia.

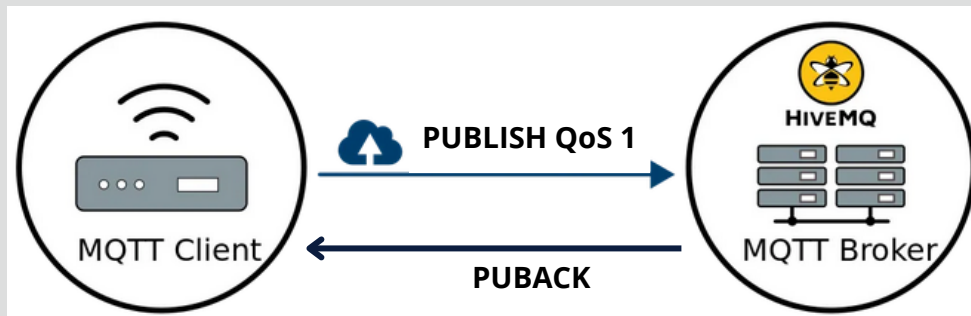
MQTT reconoce 3 niveles de QoS: QoS 0 (Como máximo una entrega), QoS 1 ( Como mínimo una entrega) y QoS 2 (Exactamente una entrega)

### **QoS 0** - como máximo una entrega

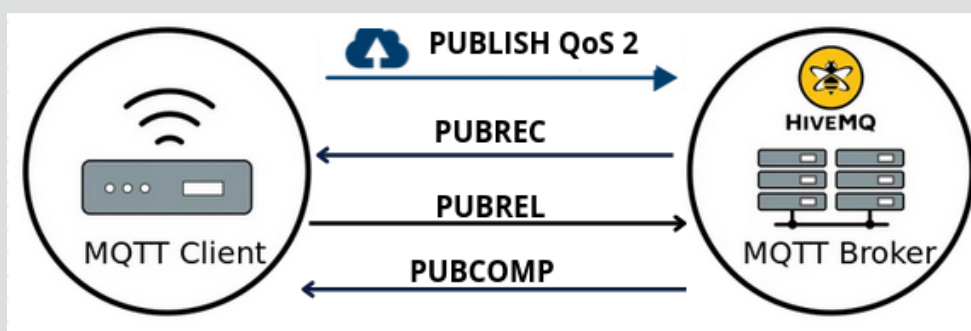
El nivel mínimo de QoS es cero. Este nivel de servicio garantiza una entrega al mejor esfuerzo. No hay garantía de entrega. El destinatario no acusa recibo del mensaje y éste no es almacenado ni retransmitido por el publicador. El nivel 0 de QoS suele denominarse «fire and forget» (publicar y olvidar) y ofrece la misma garantía que el protocolo TCP subyacente.

**QoS 1** - entrega al menos una vez (por defecto)

El nivel 1 de QoS garantiza que un mensaje se entrega al menos una vez al receptor. El emisor almacena el mensaje hasta que recibe un paquete PUBACK del receptor que acusa recibo del mensaje. Es posible que un mensaje se envíe o entregue varias veces.

**QoS 2** - entrega exactamente una vez

QoS 2 es el nivel más alto de servicio en MQTT. Este nivel garantiza que cada mensaje es recibido sólo una vez por los destinatarios previstos. QoS 2 es el nivel de calidad de servicio más seguro y lento. La garantía la proporcionan al menos dos flujos de solicitud/respuesta (un handshake de cuatro partes) entre el emisor y el receptor. El emisor y el receptor utilizan el identificador de paquete del mensaje PUBLISH original para coordinar la entrega del mensaje.



# CONNECT



contains:	Example
<code>clientId</code>	<code>"client-1"</code>
<code>cleanSession</code>	<code>true</code>
<code>username</code> (optional)	<code>"hans"</code>
<code>password</code> (optional)	<code>"letmein"</code>
<code>lastWillTopic</code> (optional)	<code>"/hans/will"</code>
<code>lastWillQos</code> (optional)	<code>2</code>
<code>lastWillMessage</code> (optional)	<code>"unexpected exit"</code>
<code>lastWillRetain</code> (optional)	<code>false</code>

## SESIONES

### Sesiones persistentes

Las sesiones persistentes en MQTT permiten que un cliente mantenga sus suscripciones y estado de mensajes a través de múltiples conexiones.

Cuando un cliente establece una sesión persistente, el broker almacena la información de suscripción del cliente y todos los mensajes no entregados al cliente. Así si el cliente se desconecta y logra reconectarse más tarde puede reanudar sin problemas.

En una sesión no persistente (sesión limpia), si se interrumpe la conexión entre el cliente y el broker, el cliente pierde su conexión, suscripciones y necesita volver a suscribirse al reconectarse.

Esto puede resultar una carga para los clientes con recursos limitados.

Para solucionar este problema, los clientes pueden solicitar una sesión persistente.

Cuando un cliente conecta al broker y envía un paquete CONNECT setea el flag `cleanSession` a `false`, lo que indica al broker que se quiere tener una conexión que el broker recuerde.

El broker es quien tiene que recordar la conexión, y manejar la complejidad, no el cliente.

Todas las librerías clientes MQTT soportan este concepto.

Cuando un cliente con sesión persistente se desconecta, el broker borra toda información del cliente.

En una sesion persistente el broker almacena la siguiente informacion:

- Datos de sesion (clienteid, etc. incluso si no hay subscripciones)
- Todas las subscripciones del cliente, para garatinzar que el cliente no tenga que volver a subscribirse cuando reconecta.
- Mensajes de QoS 1 y 2 sin ack, donde el cliente aun no ha confirmado.
- Cola de mensajes: para los clientes que estan offline, y puedan recibir los datos cuando esten online.

#### Reanudar sesion


Cuando un cliente vuelve a reconectarse enva un paquete CONNECT nuevamente con el mismo clientId. El broker utiliza el flag sessionPresent del paquete ACK para decirle a un cliente si lo recuerda como un cliente, y puede reanudar.

Esta caracaterista esta soportado por muchas librerias clientes mqtt.

#### Cola de mensajes

Este concepto se refiere al mecanismo que utiliza el broker para almacenar mensajes destinados a clientes que, por algún motivo, se encuentran desconectados. Esto permite que, cuando los clientes se reconecten, reciban los mensajes publicados en su ausencia.

El broker encola los mensajes por cliente, solo se guardan los mensajes con QoS 1 y 2, los mensajes QoS 0 nunca son encolados.

PUBLISH		
contains:		Example
packetId (always 0 for qos 0)		4314
topicName		"topic/1"
qos		1
retainFlag		false
payload		"temperature:32.5"
dupFlag		false

## MENSAJES RETENIDOS

Un mensaje retenido en MQTT es el último valor de un tema persistido por el broker. El broker almacena el último mensaje retenido y la QoS correspondiente para ese tema. Cada cliente que se suscribe a un patrón de tema que coincide con el tema del mensaje retenido recibe el mensaje retenido inmediatamente después de suscribirse. El broker sólo almacena un mensaje retenido por tema (el último recibido).

Los mensajes retenidos son útiles cuando se requiere un estado de inicio para garantizar que los clientes obtengan el último valor almacenado inmediatamente después de la suscripción.

### Enviar mensajes retenidos

Desde la perspectiva de un desarrollador, enviar un mensaje retenido es bastante sencillo y directo. Basta con establecer el flag de retención de mensaje (retainFlag) del paquete Publish en true. Los suscriptores reciben el mensaje inmediatamente (incluso si estaban desconectados cuando se publicó el mensaje), esto resuelve el problema del valor de inicio vacío, sólo un mensaje retenido por tema es almacenado por el broker.

CONNECT	
contains:	Example
clientId	"client-1"
cleanSession	true
username (optional)	"hans"
password (optional)	"letmein"
lastWillTopic (optional)	"/hans/will"
lastWillQos (optional)	2
lastWillMessage (optional)	"unexpected exit"
lastWillRetain (optional)	false
keepAlive	60

## ULTIMA VOLUNTAD (LWT)

LWT permite a un cliente definir un mensaje que el broker enviará automáticamente a otros clientes si este se desconecta inesperadamente. Es decir, actúa como un aviso de que el cliente ha perdido la conexión de manera no planificada.

La última voluntad (LWT) es una notificación del broker a los demás clientes cuando un cliente se desconecta.

Cada cliente puede especificar su mensaje de última voluntad cuando se conecta a un broker. El mensaje de última voluntad es un mensaje MQTT normal con un tema (lastWillTopic), un indicador de mensaje retenido (lastWillRetain), una QoS (lastWillQoS) y el mensaje propio (lastWillMessage).

El broker almacena el mensaje hasta que detecta que el cliente se ha desconectado de forma irregular. En respuesta a la desconexión accidental, el agente envía el mensaje de última voluntad a todos los clientes suscritos al tema del mensaje de última voluntad. Si el cliente se desconecta correctamente con un mensaje DISCONNECT correcto, el agente descarta el mensaje LWT almacenado.

El broker envía el mensaje LWT cuando:

- El broker detecta un error de E/S o un fallo de red
- El cliente no se comunica dentro del periodo Keep Alive definido
- El cliente no envía un paquete DISCONNECT antes de cerrar la conexión de red
- El broker cierra la conexión de red debido a un error de protocolo

MQTT-Packet:	
CONNECT	
	
contains:	Example
clientId	"client-1"
cleanSession	true
username (optional)	"hans"
password (optional)	"letmein"
lastWillTopic (optional)	"/hans/will"
lastWillQos (optional)	2
lastWillMessage (optional)	"unexpected exit"
lastWillRetain (optional)	false
keepAlive	60

## MANTENERSE VIVO (KEEPALIVE AND CLIENT TAKEOVER)

MQTT Keep Alive y Client Takeover son dos conceptos para evitar el estado de "conexión TCP medio abierta".

Aunque MQTT es una forma muy fiable y segura de transferir datos, puede ocurrir que la transferencia entre las partes de la comunicación se desincronice. Si un cliente MQTT se le interrumpe la conexión y el broker MQTT no es notificado, todavía está bajo la impresión de una conexión TCP existente, se produce el escenario de conexión TCP semiabierta: un cliente desconectado, el otro cliente no es consciente de ello.

Keep Alive define el tiempo máximo que un broker y un cliente pueden estar sin comunicación entre sí. Cuando se supera este tiempo especificado, el broker cerrará la conexión.

Para evitarlo, y si no envía o recibe paquetes dentro del tiempo, el cliente puede enviar un mensaje PINGREQ dentro del periodo Keep Alive, para indicar que la conexión sigue viva. Cuando el broker recibe el mensaje PINGREQ, responde con un paquete PINGRESP para su confirmación. PINGREQ y PINGRESP no contienen carga útil.

### Client takeover

Si un cliente se desconecta, lo más probable es que intente reconectarse. Por ejemplo: un coche que entra en un túnel intentará reconectarse al broker después de salir del túnel. Si el broker detecta la conexión TCP semiabierto y recibe paquetes con el mismo ClientId, realizará una Client Takeover. Se conectará al «nuevo» cliente utilizando este ClientId, y desconectará al «antiguo» cliente. Este procedimiento se denomina Toma de Control de Cliente.

- El valor de 60 segundos es el valor por defecto de Keep alive.
- Puede ser incrementado hasta 18 horas.
- Puede ser deshabilitado (no es recomendado)





## MQTT 5

### Objetivos de MQTT5

El Comité Técnico (TC) de OASIS buscaba mejorar el rendimiento y la escalabilidad sin inducir una complejidad innecesaria.

Una de las mejoras significativas es la introducción de mecanismos de autenticación mejorados. Estos proporcionan un marco de seguridad más sólido, fundamental en el mundo actual, donde el riesgo de ataques cibernéticos siempre está presente. Los usuarios tienen más opciones para proteger sus dispositivos y datos, incluido el uso de algoritmos de cifrado y técnicas de administración de claves más sofisticados.

Se introdujeron las suscripciones compartidas, una función que permite equilibrar la carga de mensajes entre varias instancias de cliente. Esto garantiza que las personas puedan administrar muchos mensajes de manera eficaz sin sobrecargar a los clientes individuales. Es práctica en escenarios donde muchos dispositivos transmiten datos simultáneamente.

Se introduce el concepto de propiedades del mensaje, que permite incluir metadatos adicionales junto a los mensajes, como marcas de tiempo, información de ubicación o estado del dispositivo.

## Objetivos de MQTT5

El Comité Técnico (TC) de OASIS buscaba mejorar el rendimiento y la escalabilidad sin inducir una complejidad innecesaria.

Una de las mejoras significativas es la introducción de mecanismos de autenticación mejorados. Estos proporcionan un marco de seguridad más sólido, fundamental en el mundo actual, donde el riesgo de ataques cibernéticos siempre está presente. Los usuarios tienen más opciones para proteger sus dispositivos y datos, incluido el uso de algoritmos de cifrado y técnicas de administración de claves más sofisticados.

Se introduce un nuevo tipo de paquete: MQTT PACKAGE AUTH que sirve como herramienta vital para implementar mecanismos de autenticación mas sofisticados.

Se introdujeron las suscripciones compartidas, una función que permite equilibrar la carga de mensajes entre varias instancias de cliente. Esto garantiza que las personas puedan administrar muchos mensajes de manera eficaz sin sobrecargar a los clientes individuales. Es práctica en escenarios donde muchos dispositivos transmiten datos simultáneamente.

Se introduce el concepto de propiedades del mensaje, que permite incluir metadatos adicionales junto a los mensajes, como marcas de tiempo, información de ubicación o estado del dispositivo.



## APLICACION DE PRUEBA

se implementa una aplicación multicontenedor para ejemplificar la comunicaciones entre dos clientes, a través de un broker:

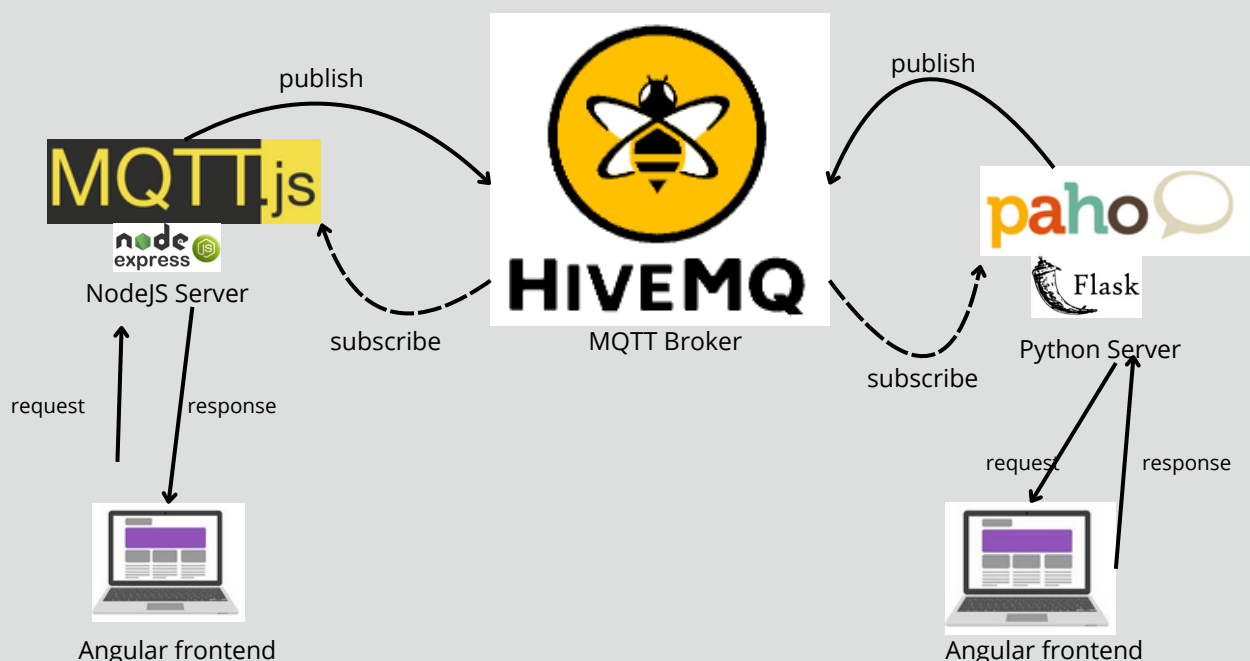
Cliente1: desarrollado en nodejs express con la librería mqtt cliente mqttjs.

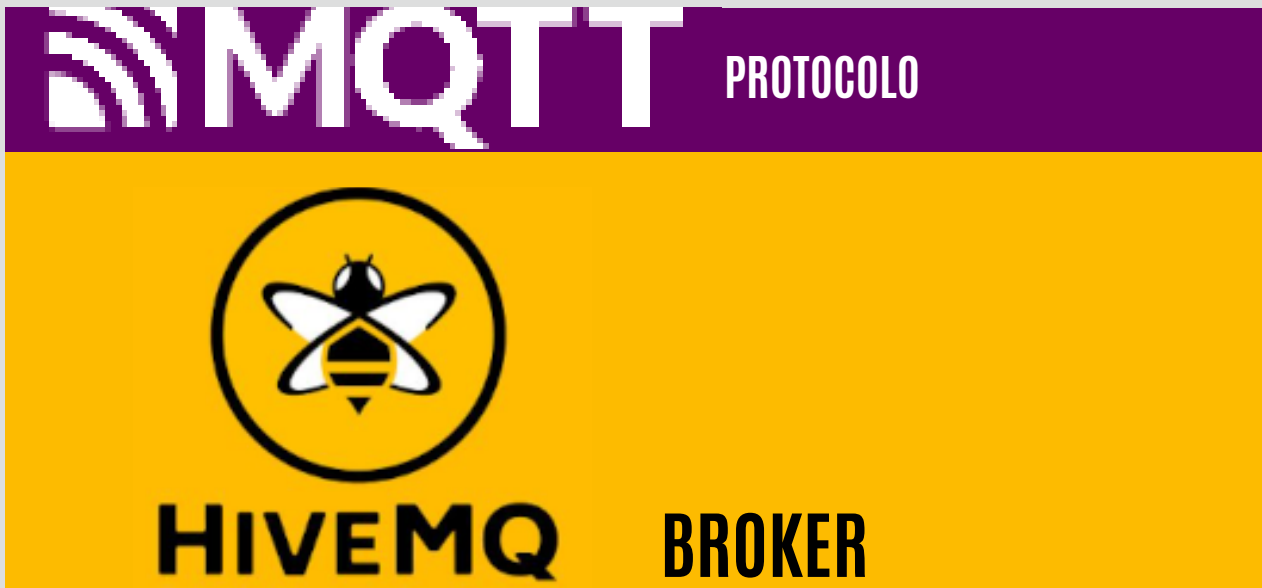
Cliente2: desarrollado en python flask con librería mqtt cliente paho.

Broker HiveMQ, con el la version 3.1 del protocolo.

Con los clientes mqtt se interactura mediante una aplicación Cliente/Servidor en Angular.

Repositorio: <https://github.com/wfusiman/hivemq>





## CONCLUSIONES

El protocolo MQTT, y la implementación HiveMQ destacan por su sencillez y facilidad de uso.

El broker HiveMQ ofrece distribuciones open source (Community), comerciales (Enterprise) y Cloud. Requieren poca configuración inicial para ser operativos.

Las librerías clientes, con unos pocos pasos se puede desarrollar aplicaciones clientes, además de que existen muchas librerías disponibles en todos los lenguajes.

Si bien el protocolo está pensado para entornos de IoT, puede ser utilizado para comunicar sistemas distribuidos más complejos.

Una utilidad puede ser conseguir sincronización, y tolerancia a fallas (mediante control de estados).

Aunque no sería apropiado para el intercambio de datos de gran volumen, ni multimedia, dada la simplicidad, tamaño y limitación de los paquetes MQTT.



PROTOCOLO

**HIVEMQ BROKER**

## REFERENCIAS

- <https://aws.amazon.com/es/what-is/mqtt/>
- <https://www.paessler.com/es/it-explained/mqtt>
- <https://www.hivemq.com/blog/mqtt-brokers-beginners-guide/>
- <https://www.hivemq.com/blog/mqtt-packets-comprehensive-guide/>
- <https://www.hivemq.com/blog/mqtt-essentials-part-4-mqtt-publish-subscribe-unsubscribe/>
- <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-7-persistent-session-queuing-messages/>
- <https://www.hivemq.com/blog/mqtt5-essentials-part6-user-properties/>
- <https://docs.hivemq.com/hivemq/latest/user-guide/index.html>
- [https://moodle.uneg.edu.ve/pluginfile.php/114008/mod\\_page/content/6/Sistema-Distribuidos-esp.pdf](https://moodle.uneg.edu.ve/pluginfile.php/114008/mod_page/content/6/Sistema-Distribuidos-esp.pdf)