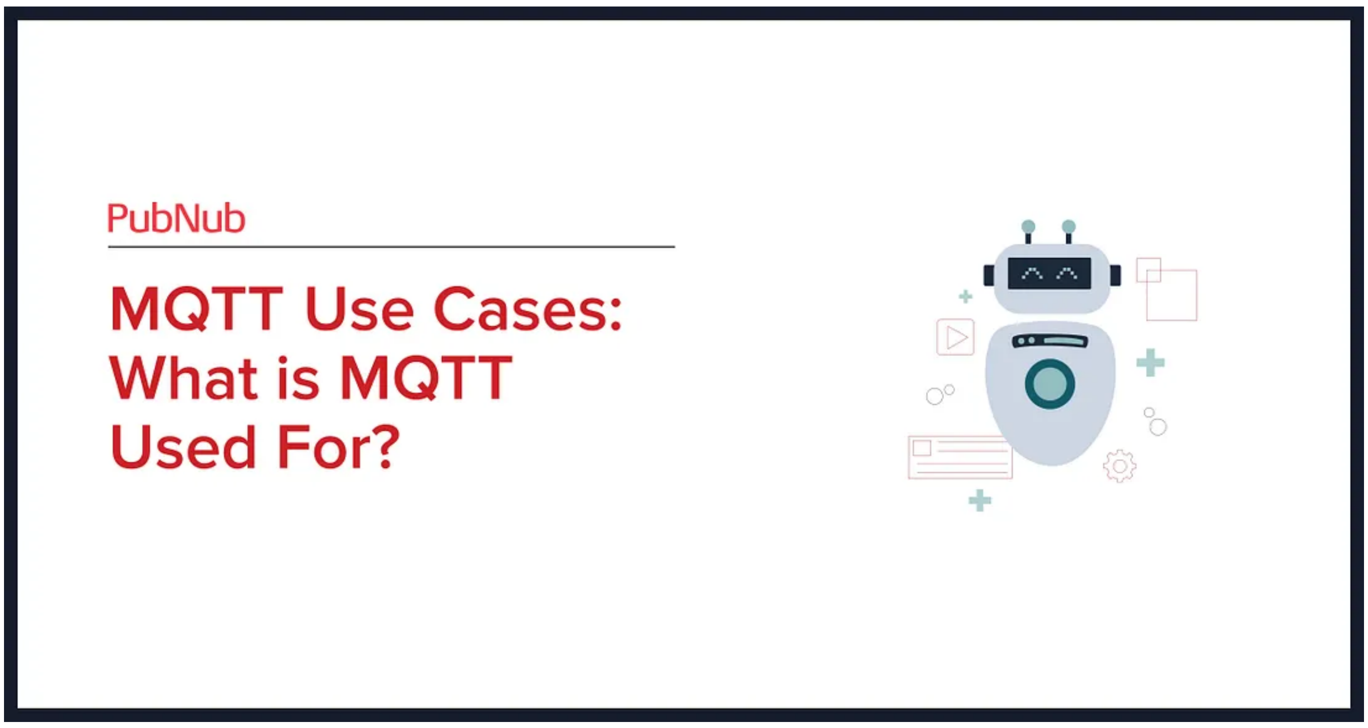
MQTT Use Cases: What is MQTT Used For?

PubNub Follow 11 min read · Feb 7, 2024



What is MQTT used for?

MQTT is a lightweight messaging protocol typically used with IoT devices that are designed to be power efficient and consume minimal bandwidth.

The MQTT protocol runs over TCP/IP and is based on a bi-directional publish-subscribe model with lossless communication, ensuring that MQTT messages are delivered efficiently and without loss.

MQTT was built to be a low-overhead protocol that strongly considered bandwidth and CPU limitations. It was designed with the ability to run in an embedded environment where it would reliably and effectively provide an avenue for communication.

MQTT fundamentally is a publish/subscribe (pub/sub) protocol. It allows clients to connect as a publisher, subscriber, or both to specific topics. You connect to a broker that handles all the message passing, where each client is identified by a unique client ID.

Real-life MQTT Example and Use Case

MQTT is commonly used for IoT applications to connect to and communicate with low-power devices such as sensors, actuators, and home appliances, efficiently managing the payload of data transmitted.

In agriculture, MQTT is used to monitor environmental factors such as soil moisture, temperature, and humidity, allowing farmers to take the appropriate action. In industrial automation, MQTT can monitor the end-to-end manufacturing process, ensuring consistent quality and identifying systemic issues on the production chain. In transportation, MQTT is used to monitor traffic lights, parking meters, and the real-time location of public transportation, all while being mindful of the ecosystem and low bandwidth requirements.

Why use MQTT instead of HTTP?

HTTP and MQTT are two very different communication protocols used for different purposes.

HTTP is a request/response protocol that only delivers data in response to a request by a client. It is designed for large amounts of data and is commonly used for transferring web pages, images, and videos. MQTT is a pub/sub protocol that enables asynchronous message delivery, with multiple clients able to subscribe to a single topic and receive messages from a publisher or publishers. MQTT is optimized for small amounts of data but is more efficient than HTTP in terms of network overhead and device power consumption, making it a suitable choice for applications requiring low bandwidth and minimal power usage, such as those managed by AWS IoT services.

What languages is MQTT written in?

MQTT is just a protocol and can be written in many languages. However, MQTT client libraries — which are used to communicate using the MQTT protocol — are available in multiple languages such as C/C++, Java, Python, and JavaScript — including Node.js, Ruby, Go, PHP, and Swift. An MQTT client library communicates through an MQTT broker, which is also available in several programming languages.

How does MQTT work?

MQTT is a hugely flexible protocol, but there are only two basic entities: the MQTT client and the MQTT broker.

How do MQTT clients work?

The MQTT client can be any endpoint that implements the MQTT protocol. In the case of IoT, the client is the connected device, such as a sensor, monitor, or Arduino board, but MQTT is not restricted to IoT, the client could also be a smartphone or laptop, for example.

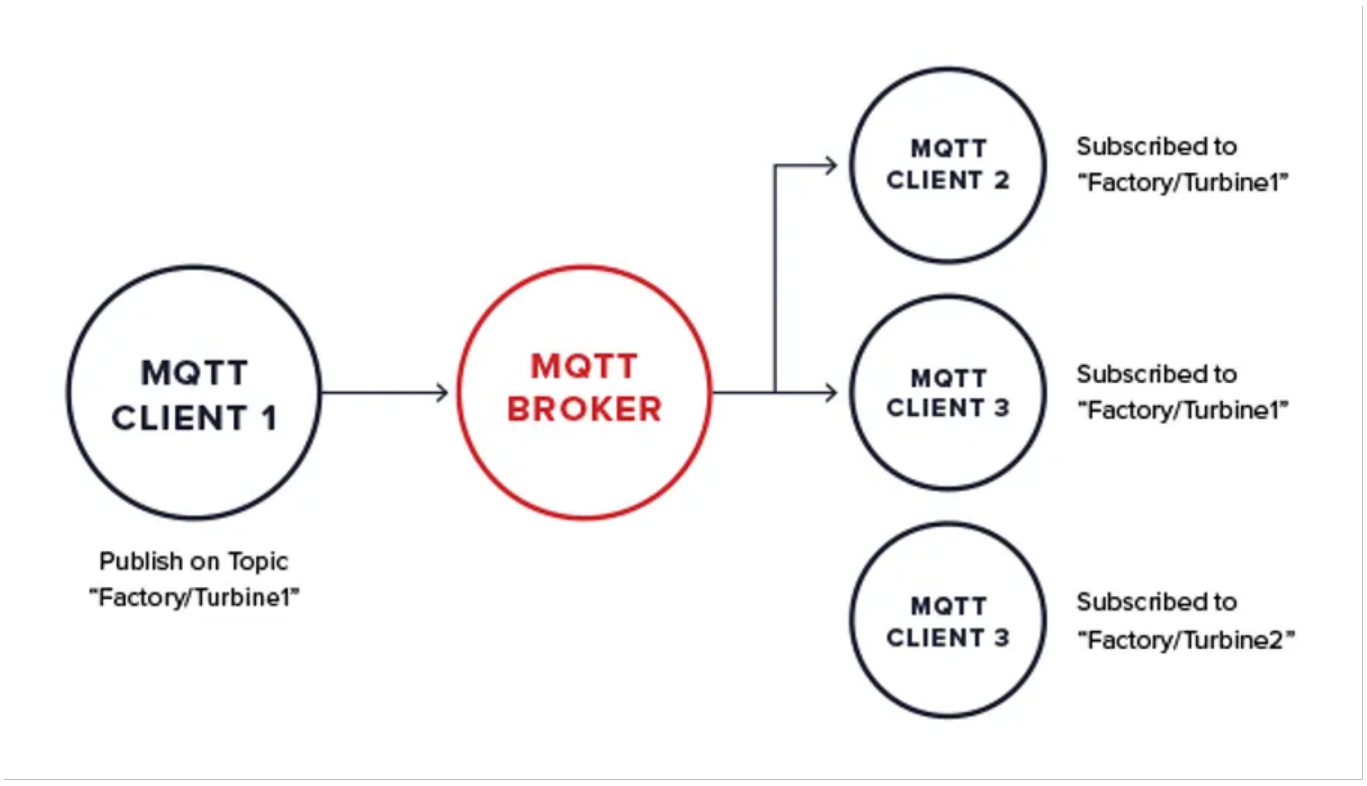
MQTT clients do not communicate directly with each other. Still, all interactions are 'brokered' by a server component referred to as the MQTT Broker, which sits in between clients and handles messages' routing.

What is MQTT grouping?

All communication in MQTT is grouped into 'topics'. MQTT clients can publish messages to topics and subscribe to receive messages from others. A topic can be any string and is intended to group subjects of common interest, for example, sensor updates would be published to a topic, as would messages in a group chat, depending on the use case.

What is the MQTT broker?

The broker is responsible for managing which clients are subscribed to which topics, receiving messages published on a particular topic, and sending that message to any client subscribed for updates. When the connection between a client and broker is lost, the broker is also responsible for caching the message and delivering it to the client when the connection is re-established.


MQTT CLIENT 1
Publish on Topic "Factory/Turbine1"
MQTT BROKER
MQTT CLIENT 2
Subscribed to "Factory/Turbine1"
MQTT CLIENT 3
Subscribed to "Factory/Turbine1"
MQTT CLIENT 3
Subscribed to "Factory/Turbine2"

What MQTT brokers are available to use?

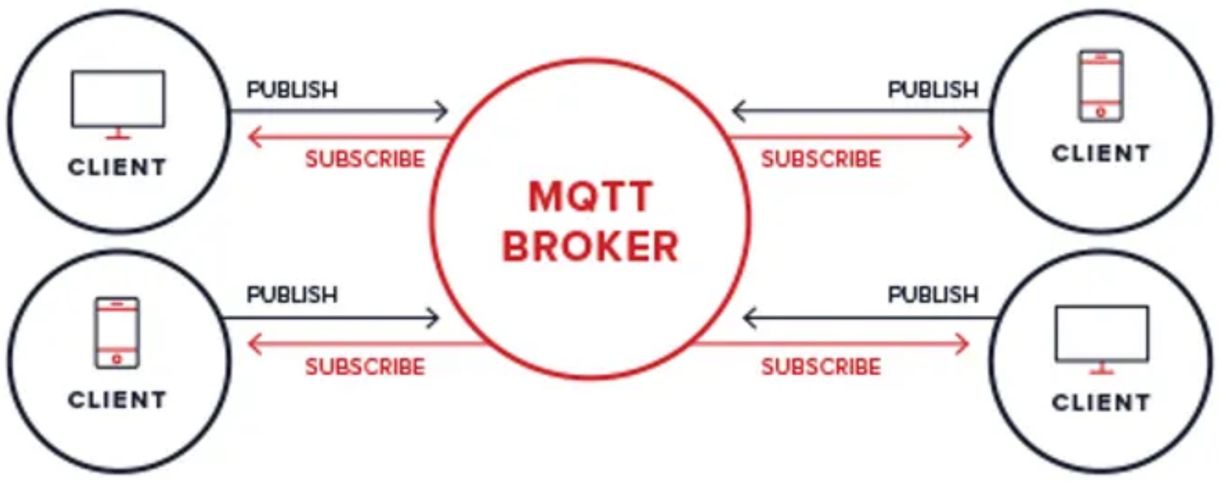
The MQTT broker is fundamental to the MQTT protocol and can be considered the MQTT Server. As previously mentioned, the broker sits between all clients and facilitates communication.

Many MQTT brokers exist, both open source and proprietary, and one of the most important architectural decisions of any MQTT-based deployment is how to host the broker. Since MQTT is designed to run at a very large scale, any broker architecture will need to scale accordingly. Most corporate deployments will rely on cloud-hosted, proprietary offerings, so they do not have any infrastructure concerns, though self-hosted options are also available.

Sitting in between all communication, the broker can retain messages and keep a secure record of all sessions; clients are isolated from each other, so client insecurities and vulnerabilities can be sandboxed and, depending on your network topology, a broker can reduce the traffic on your network as a whole & allows for more efficient routing.

How to use MQTT and Pub/Sub

Any client that supports the MQTT protocol can subscribe and publish to topics, but all communication goes through the broker, as shown in the example model below.



We have our own MQTT broker, the MQTT bridge, to seamlessly integrate PubNub with your MQTT solution. More information can be found at the end of this article.

MQTT Protocol features

While often considered an IoT solution, MQTT is just the protocol that underpins many well-known IoT products. Many features of MQTT make it particularly well suited to efficient communication between devices:

Easy to get up and running

MQTT has been around for a long time, and there are many robust, reliable, and scalable proprietary and open-source solutions. These pre-existing MQTT components can be used in all projects, from large solutions to passion projects, without modification and regardless of developer experience.

Reliability and configuration

MQTT has the concept of quality of service (QoS), discussed in detail later, which queues and caches messages on the MQTT broker, delivering them to the client when the connection is re-established. This works particularly well for partially connected devices or clients with intermittent connectivity, such as IoT devices.

The client does not have to consider the solution architecture

With MQTT, you publish messages to a topic and receive them when available. As a client, you do not have to worry about establishing or re-establishing a connection or whether your recipient(s) are listening for your messages, it "just works".

Designed for scale

MQTT can cope with whatever scale your solution requires, from startups to global companies. Most famously, Facebook Messenger uses MQTT for its communication.

MQTT session lifecycle

MQTT relies on TCP/IP for connectivity, so follows a similar lifecycle

Connection

The MQTT client initiates a connection to the MQTT broker. Typically, this will be over the standard MQTT ports (1883/883 for secure and insecure connections, respectively)

Authentication

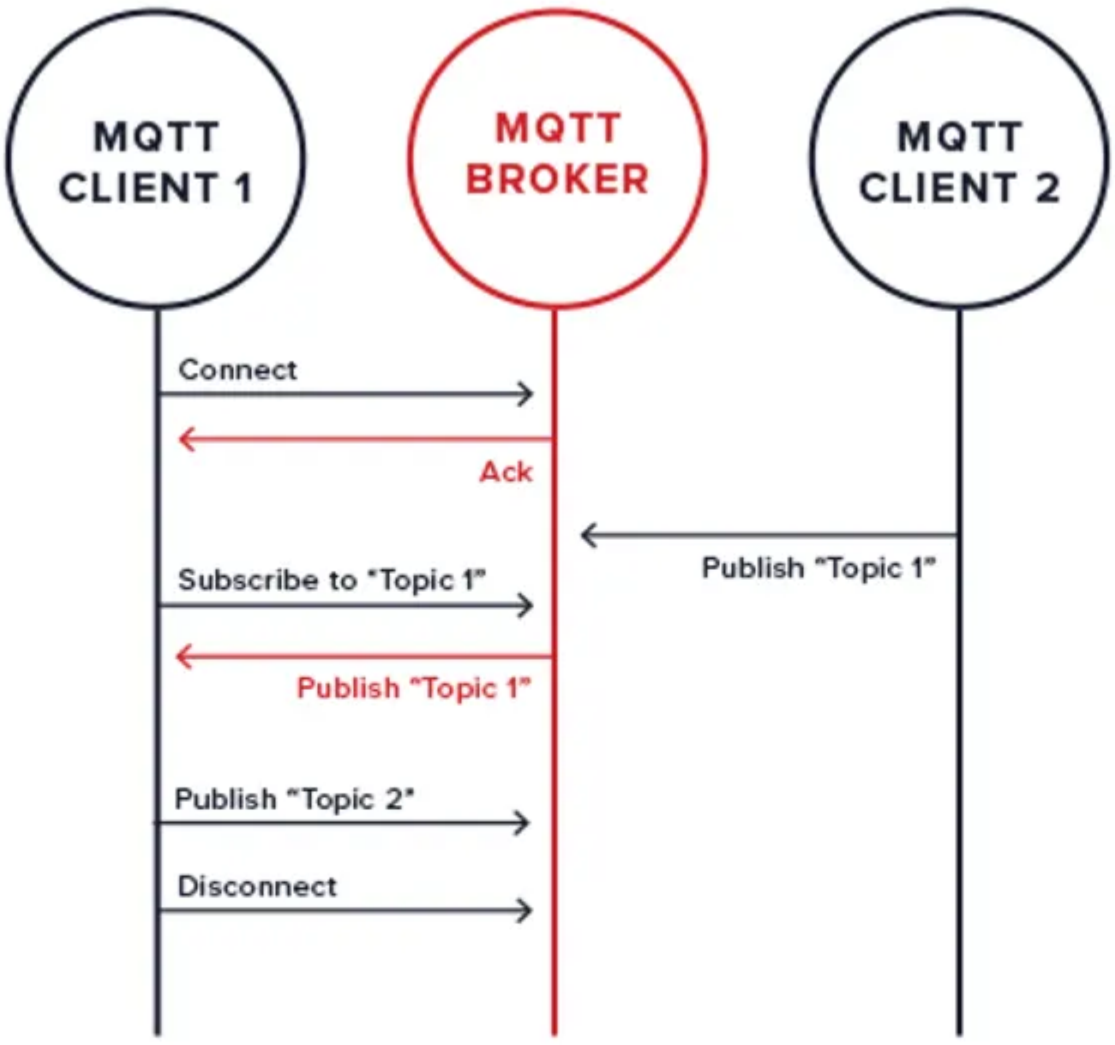
The MQTT protocol does not have any authentication considerations beyond TLS for the underlying TCP/IP connection, which is sufficient for most use cases.

Publish / Subscribe

Once connected, clients can publish or subscribe to topics to send and receive messages.

Disconnect

Disconnection can be initiated by either the client or broker and will cause the MQTT session to end after any in-progress requests have been processed.



MQTT Quality of Service (QoS)

Quality of Service allows a solution designer to specify how reliable an MQTT connection should be, in general, the more reliable the connection, the greater the potential memory overhead for retries and message retention.

QoS -1

The message is sent only once and the client and broker take no additional steps to acknowledge delivery. This is also known as “fire and forget”.

This only applies to messages sent by a client. Once messages are received at the broker, they are considered QoS 0

QoS 0

The sender and receiver engage in a handshake to ensure only one copy of the message is received. This assures delivery and avoids multiple copies of the message being sent.

QoS 1

The message is re-sent multiple times until its receipt is acknowledged by the recipient. This works best for time-sensitive messages but can lead to duplicate messages being received.

MQTT use cases and IoT

MQTT is a lightweight and power-efficient protocol that over the years has been used in the kind of solutions where those are key considerations.

IoT is the most popular example where MQTT is a great fit. IoT devices are typically battery-powered and can be very expensive to maintain, so lasting a long time between battery swaps is preferred. Whilst in the home environment it is not difficult to replace a battery now and then, in an industrial environment, with thousands of sensors, spread across a wide area and often in very difficult-to-reach locations their maintenance can get costly very quickly.

MQTT IoT use cases include fire detectors, theft tracking, location monitoring, sensors, engine status, etc. MQTT also has very low data overheads so, in those environments where data is expensive, or where thousands of devices are deployed so bandwidth is a concern, MQTT is ideal.

PubNub and MQTT work great together for IoT. Please see our IoT demo and IoT tutorial to learn more about how PubNub and MQTT can deliver a reliable, scalable, and efficient IoT solution.

MQTT is also very popular in real-time communication, the best-known example being Facebook Messenger. Why would Facebook choose to use MQTT? Because it would not drain your phone’s battery, would not use excessive data, runs on secure protocols, allows scaling, easily facilitates group chats, and distributes all data through a central MQTT broker backbone infrastructure.

Does PubNub support MQTT use cases?

Absolutely! PubNub provides an MQTT bridge offering very low latency which avoids having to deploy a custom MQTT broker and provides simple integration with other PubNub functionality such as Functions and data streaming.

Since PubNub is designed around the publish-subscribe model, it maps very nicely to MQTT:

- Publish / Subscribe: These concepts are identical for both PubNub and MQTT
- PubNub channels: The equivalent in MQTT is the 'topic'.
- UUID: Both PubNub and MQTT have the concept of a unique identifier assigned to each client.

Wildcards: Both PubNub and MQTT support wildcards when subscribing to channels/topics. See the PubNub docs for more detail.

How to leverage MQTT use cases

At the heart of using MQTT as a communication avenue is the topic. It's a remarkably simple idea not unique to MQTT; however, the MQTT protocol leverages the power of this quite nicely. A topic implicitly accomplishes several tasks, most importantly ensuring that a message is delivered to the correct listeners. MQTT treats a topic as a file path. When thinking of a topic as a simple communication filter, the path application can become very powerful.

You could be interested in a particular higher level of the path or the leaf element. Without explicitly saying so, MQTT filters messages based on where you subscribe in the tree path. It's a simple idea, that can be used very effectively.

MQTT technical specifications to meet your use case

Let's dig a little deeper into the technical aspects of MQTT. First, the protocol runs on top of the TCP/IP networking stack. When clients connect and publish/subscribe, MQTT has different message types that help with the handshaking of that process. The MQTT header is two bytes and first byte is constant.

In the first byte, you specify the type of message being sent as well as the QoS level, retain, and DUP (duplication) flags. The second byte is the remaining length field. There is more information you can glean from the MQTT specification if you are interested.

How to use PubNub and MQTT

With PubNub now supporting MQTT over our real-time, global Data Stream Network, we wanted to give an overview of the protocol and why you might or might not use it from our perspective.

You may already have deployed MQTT-based devices. Or you may be considering a new IoT deployment. Either way, the question arises, "When should I use MQTT, and when should I use PubNub?" And when does it make sense to use both? If you are considering robust, bi-directional communication options for your MQTT-based devices, PubNub provides enterprise-grade security, scalability and reliability, and many value-added features too, like functions-as-a-service, presence detection, and message storage and playback. And all this power is all available with a simple publish-subscribe command.

Like the proverbial peanut butter and chocolate, PubNub and MQTT together can elegantly solve many embedded device use cases. You can leverage PubNub with MQTT to satisfy use cases like low-latency, efficient anomaly detection. For example, say you have MQTT-based sensors on industrial, power-generating turbines. You can leverage PubNub — including the "Functions-as-a-Service" Functions — to react to and transmit sensor data in real time, so an operations team can identify turbine problems before data even touches a centralized data center. You can also leverage PubNub with MQTT-based home alarms to review signals across alarms and identify if there is a bad alarm that is sending false positives. And across use cases, you can leverage PubNub's integrations to multiple machine learning services to detect anomalies.

If MQTT is not part of your current infrastructure, you might consider leveraging the PubNub Data Stream Network directly along with PubNub's numerous IoT SDKs for all of your Internet of Things connectivity. Or the PubNub Network with MQTT can be a great choice for low-powered devices.

To utilize the PubNub Network with MQTT-enabled devices, you need to subscribe your devices to PubNub at mqtt.pubnub.com. Our PubNub Improves MQTT Support & IoT Capabilities guide walks you through the subscription steps in more detail.

Additional MQTT Resources

- MQTT Protocol
- Paho Project (MQTT Client)

How can PubNub help you?

This article was originally published on PubNub.com

Our platform helps developers build, deliver, and manage real-time interactivity for web apps, mobile apps, and IoT devices.

The foundation of our platform is the industry's largest and most scalable real-time edge messaging network. With over 15 points-of-presence worldwide supporting 800 million monthly active users, and 99.999% reliability, you'll never have to worry about outages, concurrency limits, or any latency issues caused by traffic spikes.

Experience PubNub

Check out Live Tour to understand the essential concepts behind every PubNub-powered app in less than 5 minutes

Get Setup

Sign up for a PubNub account for immediate access to PubNub keys for free

Get Started

The PubNub docs will get you up and running, regardless of your use case or SDK

Written by PubNub

2.8K followers · 2.8K following

Follow

APIs for building secure realtime apps, with a global programmable network to power them.