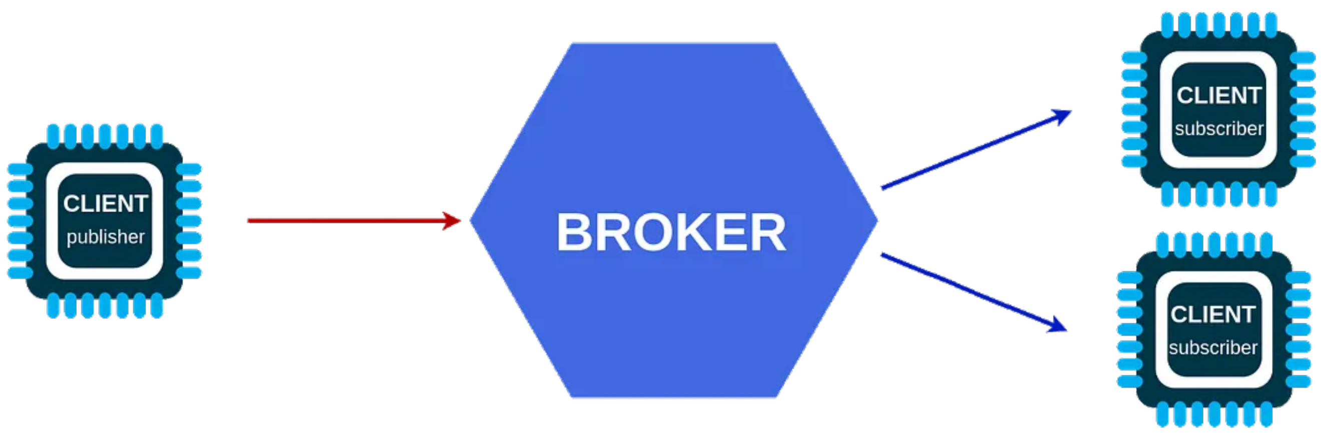
Control System Integrators

A Gentle Introduction to MQTT

Jon Stopple Follow 6 min read · Feb 14, 2024

MQTT might seem intimidating at first. As with anything new, that is totally understandable! This post strives to break down MQTT into the simplest concepts so anyone can understand. We will cover the basics of MQTT: what MQTT is and why you may want to use it or avoid it in your project.

MQTT and IoT seem to go hand in hand. MQTT (Message Queuing Telemetry Transport) is a publish subscribe model for transmitting data and widely used in IoT (Internet of Things). What does this mean? MQTT is a method to push data from a device into the hands of anyone who wants it.



MQTT pub-sub architecture with clients and broker. Modified image used under creative commons license.

Summary

Let’s start with a quick summary to get started with the new terminology.

- MQTT — A communication protocol to transmit small packets of data.
- Clients — Devices like a sensor or computer that sends or receives data.
- Broker — Middleman that acts like a post office and handles messages coming from and going to clients.
- Pub-sub model — Clients publish messages and subscribe to topics. They can do both or only one.
- Publisher — Client that sends data to the broker.
- Subscriber — Client that picks topics or categories of messages to receive from the broker.
- Topic — Subject or context to the message that determines how messages are routed to subscribers.
- Payload — Data being sent. Most commonly a tag, value pairing of data in JSON format.
- QoS (Quality of Service) — Choose how important the message is that it gets delivered once, and only once. QoS 0 has no guarantee of delivery. QoS 1 guarantees at least one delivery. QoS 2 guarantees exactly one delivery.

MQTT works similar to your favorite social media feed. Clients (users) send and receive messages via a broker (social media platform). As a user, you can publish messages to the broker and to route these messages to any subscribers (followers). Users who subscribe to a topic (user) have those messages appear in their feed via the broker. In this way, if a user has no followers, the message is published but no one sees it. It also allows many users to subscribe to a topic and they all see the same message. In general, this is how MQTT works, but now we'll dive into the details.

Clients & Brokers

MQTT operates via clients and a broker. A client can publish messages and subscribe to topics. Devices like sensors, controllers, and computers all act as clients, either publishing data or subscribing to the data channels it cares about. A broker is the hub which all of the messages go through. It receives each message and routes them to the proper subscribers based on the topic. Each client only needs to access the broker to communicate with any device in the network.

Publish Subscribe Model

The publish subscribe model refers to how messages are generated and routed. A published message means the client sends a data packet to the broker. Clients can also subscribe to topics. The broker routes the message to all users who subscribe to the topic.

This model is asynchronous in nature and has the features of a one-to-many relationship. A published message is routed to all subscribers of that topic via the broker. This allows clients to communicate through the broker meaning only the broker networking information is required for the entire system to communicate.

Topics

Think of a topic as being an email subject you use to provide context for the rest of the message. Topics determine how a messaged gets routed by the broker to any subscribers. A topic can have many levels separated by a forward slash.

```
field_c/temp/device_a
```

Topics should be thought out and can take many forms depending on the system design. This way, clients can easily choose which topics they should subscribe to. In the example above, the topic sets context to the location (field_c), what measurement is being recorded (temperature), and which device is publishing the data (device_a). This context can be used by subscribers to know how to process the data accordingly.

Topics should be short but specific and detailed. More characters require more bandwidth on the network. At the same time, topics provide important context and distinction to the data being transmitted. Topic differentiation enables smart routing and processing of messages downstream.

Subscribers can choose to include wildcard characters in their topic selection for more generalized subscriptions. The “+” symbol is a single level wildcard character. So, a topic of location/+/deviceA means the client will subscribe to any messages that match the first and third topic levels of location and deviceA as long as any second level topic exists. The “#” symbol is a broad subscription that gets all messages that begin with the specific topic before the wildcard. A common test scenario is subscribing to a topic of #, meaning get all messages published. At scale, wildcard subscriptions should be limited due to increased processing requirements within the broker, but they can be useful for testing and learning.

Payload

So, what does the data look like? The message data, or payload, is typically sent in JSON format, the widely used format of the web, but in reality, a payload can be many formats like plain text, XML, or a byte array. A sample payload might look like this:

```
{ "timestamp": 1707796542671, "temperature": 76.7, "humidity": 21.54 }
```

It is a tag, value pairing of data, with the content enclosed in brackets, each pair separated by a comma, and each tag and value separated by a colon. Each tag is a string enclosed in double quotes and the value can be a string, number, or Boolean.

QoS (Quality of Service)

MQTT clients choose a QoS level with the broker for any message they publish or topic they subscribe to. QoS selections are based on the network and message requirements within the application. QoS levels are 0, 1, and 2, with level 2 requiring the most resources.

QoS 0, or “at most once”, means the message publisher fires the message once to the receiver. If the recipient is not available, the message is lost.

QoS 1, or “at least once”, means the publisher will keep sending a message until the receiver confirms receipt. This guarantees delivery but exposes the possibility of sending the same message twice if the confirmation is not initially received by the publisher.

QoS 2, or “exactly once”, means the publisher sends only one message and the receiver receives only one message. This requires two handshakes. One to confirm the message was received, and another to confirm the confirmation of delivery.

Benefits of MQTT

MQTT is widely used due to its lightweight, simple nature and being inherently scalable.

- Low bandwidth and power requirements.
- Easy to add clients.
- Simple network requirements. Clients only need to connect to the broker. Client to client networking is not required.
- Simple to begin using. There are libraries in most common programming languages to make getting started with MQTT a quick process. Many low code tools (Node-RED) support MQTT as well.
- Asynchronous messaging nature.
- Simple to add security features like encryption and authentication.
- Highly supported environment.

Disadvantages of MQTT

MQTT should not be used in every situation, however.

- High latency. MQTT is not meant for high-speed applications requiring near real-time data transfer.
- Message size limited to 256 MB.
- Lacks standardization. Data architecture must be well documented and known in order to communicate effectively.
- No native security enforcement. Some IoT devices are not able to support SSL/TLS encryption that would prevent data breaches.
- Reliant on broker uptime. If the broker goes down without a backup or redundant system in place, operations will stop.

Other Features

There are many other features of MQTT which we won't get into here. These include persistent sessions, last will and testament, retained messages, keep alive, security considerations, topic tagging, plug-in integration, and message queueing. These features enable even greater functionality to the already powerful MQTT platform.

More Resources

So, how should you get started testing?

- HiveMQ Public Broker
- Mosquitto Cloud or Local Broker
- HiveMQ Cloud or Local Broker
- Node-RED to Publish and Subscribe

The easiest way is to begin testing with the public broker on HiveMQ. After that, I like to use Mosquitto locally as my broker and connect using Node-RED. Node-RED makes it super easy to connect, publish, subscribe, and see what is happening.

If this story was helpful and you wish to show some support, you could:

1. *Earn yourself money with a referral.*
2. *Clap 27 times for this story.*
3. *Leave a comment.*
4. *Follow me for more content like this.*
5. *Connect with me or my company on LinkedIn.*
6. *Share Control System Integrators with someone interested who could benefit from innovative open-source smart manufacturing.*

Thank you for reading and I hope to see you in the next article!