MQTT Basics

by Jacob Evans

Prerequisites

This lesson assumes basic knowledge of networking concepts.

Introduction

This article will be covering the MQTT Protocol including:

- What is MQTT?
- Where does it come from?
- Client and Broker in MQTT
- MQTT Protocol Concepts
- MQTT Topics
- MQTT QOS
- Retained messages in MQTT

What is MQTT?

MQTT or Message Queuing Telemetry Transport is a lightweight publish (send a message) subscribe (wait for a message) protocol. It was designed to be used on networks that have low-bandwidth, high-latency and could be unreliable. Since it was designed with these constraints in mind, it's perfect to send data between devices and is very common in the IoT world.

Where does it come from?

MQTT was invented by Dr Andy Stanford-Clark of IBM, and Arlen Nipper of Arcom (now Eurotech), in 1999. They invited the protocol while working on SCADA system for an oil and gas company that needed to deliver real time data. For 10 years IBM used the protocol internally. Then in 2010 they released MQTT 3.1 as free version for public use. Since then many companies have used the protocol including Cybus.
If you're interested in learning more you can click here to read the transcript of a IBM podcast where the creators discuss the history and use of MQTT.

Client and Broker in MQTT

Client

A client is defined as any device from a micro controller to a server as long as it runs a MQTT client library and connects to a MQTT broker over a network. You will find that many small devices that need to connect over the network use MQTT and you will find that there are a huge number of programming languages that support MQTT as well.
Find a list of libraries here.

Broker

The broker is defined as the connector of all clients that publish and receive data. It manages active connections, filtering of messages, receiving messages and then routing messages to the correct clients. Optionally it can also handle the authentication and authorization of clients.
Find a list of brokers here.
For information how they work together continue on to the next section.

MQTT Protocol Concepts

MQTT uses the **publish** (send a message) **subscribe** (wait for a message) pattern. This is an alternative approach to how a normal client (asks for data) server (send back data) pattern works. In the client server pattern a client will connect directly to a component for requesting data and then immediately receive a response back. In the publish subscribe pattern the connection between the components is handled by a third party component called a **broker**.
All messages go through the broker and it handles dispatching the correct message to correct receivers. It does that through the use of a **topic**. A **topic** is a simple string that can have hierarchical levels separated by „/'.

Examples Topics:

- house/toaster/temp
- house/washer/on
- house/fridge/on

The MQTT client can listen for a message by subscribing on a specific topic. It will then receive data when a MQTT client publishes a message on that topic.

Example

```
Client A:                                  Broker
subscribe /house/toaster/temp  ---------->  |
                                            |
                                            |                       Toaster Device:
                                            | <-------- publish /house/toaster/temp
                                            |
                                            |
Client A   < ------ Receive Message ------  |
```

This publish subscribe pattern offers a couple of advantages:

- Publisher and subscriber do not need to know anything about one another because the broker handles the receiving and sending of messages.
- There can be a time difference between the clients that are sending information to one another since the protocol is event based.
- The subscriber and publisher can process the information at different times which means they don't block each other from sending or receiving new information.

Topics in MQTT

Topics are the way that data is organized in MQTT. They are structured in a hierarchy manner using the „/' as a separator. It's very similar to how folders and files are structured in a file system. A few things to remember about topics are that they are case sensitive, must use UTF-8 and and have to have at least 1 character.

Example of basic topics

- /
- house
- house/toaster
- house/toaster/temp

Multiple Subscriptions

A client can also subscribe to multiple topics at once using wildcards. The 2 wildcards in MQTT are:

- # (multi level)
- + (single level)

The level being the level of the topic hierarchy tree that you want to subscribe to.

Example multi level

- Subscribe to house/#
- house/temp
- house/toaster/temp
- house/tv/on

Example single level

Subscribe to house/+/light

- house/frontroom/light
- house/bedroom/light
- house/porch/light

NOT

- house/frontroom/temp
- house/bedroom/lamp

Publishing

An MQTT client can only publish to an individual topic. Meaning you cannot use wildcards.

$SYS Topics

The only topics you will find in a broker after start is the **$SYS** topics. These topics are usually reserved for publishing data about the broker such as the number of current client connections.
If you would like to read more specifics on the requirements of topics see the MQTT specification.

MQTT QOS

QOS (Quality of Service) is defined as the agreement between the **broker** and the client that deals with the guarantee that a message was delivered. MQTT defines 3 levels of QOS.

QOS 0 (At most once)

This is the fastest QOS level. When a message is sent across the network no reply from the broker is sent acknowledging that it received the message. The sending message is then deleted from the client sending queue so no repeat messages will be sent.

QOS 1 (At least once)

This level is the default mode of transfer. The message is guaranteed to be delivered at least once. It will continue to send the message with a DUP flag until an acknowledgment message is sent. This could result in duplicate messages being sent.

QOS 2 (Exactly once)

This is the slowest level as it requires 4 messages. The message is always delivered exactly once.

1. The sender sends the message and waits for a response from the broker.
2. The sender receives the response from the broker. If it doesn't it sends the request again with a DUP flag.
3. The sender sends a message saying that it received the response and awaits acknowledgment.
4. The sender receives acknowledgment and deletes the message. If not it sends another message saying that it received the response with a DUP flag.

Retained messages in MQTT

Normally when a client publishes a message the broker will delete the message after routing to the correct subscribing clients. But what if a client subscribes to a topic after the message is sent? Then it will receive no data until another message is published. This could be desirable in certain situations but in other situations you may want the client to have the last published data. This is the purpose of the **retain flag**. If set to true when a message is sent the broker will cache the message and route it to any new subscribing clients. There is **only 1** retained message per topic and if a new message is published it will replace the retained message.

Summary

MQTT is a lightweight publish subscribe protocol. It is defined with a broker client relationship and organizes its data in hierarchy structures called topics. When publishing messages you can specify a QOS level which will guarantee that a message is sent and specify a retain level for a message so a subscribing client can receive retained data after connecting.

LEARN MORE

- MQTT specification
- Hive MQTT tutorials
- Steve's Guide to MQTT

Cybus is a specialist for secure IIoT Edge software, headquartered in Germany. Cybus Connectware serves smart factories as a universal Edge and DevOps hub. Machine builders and providers of IIoT services use the Cybus Connectware as a software-based gateway. As early as 2017, Cybus published the first secure industrial connector for machine data according to today's DIN SPEC 27070 standard. Industry analyst Gartner named Cybus a worldwide "Cool Vendor". Today, the company counts medium-sized and large companies from numerous industrial sectors such as mechanical engineering, automotive and aviation among its customers.

Cybus GmbH · Osterstraße 124 · 20255 Hamburg · Germany · www.cybus.io · hello@cybus.io · (+49) 40 228 58 68 51