

MQTT Essentials Part 3: Client, Broker and Connection Establishment



Welcome to the third part of the MQTT Essentials. A blog series about the core features and concepts in the MQTT protocol. **In this post, we'll discuss the role of MQTT client and broker and the parameters and options available, when connecting to a broker.**

In the last post, we explained how the publish/subscribe pattern works and how it is applied in MQTT (<http://www.hivemq.com/mqtt-essentials-part2-publish-subscribe/>). The following is a quick recap of the essence: **Publish/Subscribe decouples a client, which is sending a particular message (called publisher) from another client (or more clients), which is receiving the message (called subscriber).** In order to determine, which message gets to which client, MQTT uses topics. A topic is a hierarchical structured string, which is used for message filtering and routing (More details (<http://www.hivemq.com/mqtt-essentials-part-5-mqtt-topics-best-practices/>)).

The last post was really more academical nature as we examined what publish/subscribe is about and how it can be differentiated from a message queuing approach. **This post will be way more practical and stuffed with basic knowledge about MQTT.** Some topics we discuss are definition of MQTT client & broker, basics of an MQTT connection, the Connect Message with its parameters and the establishing of the connection by the acknowledgement of the broker.

Introduction

As we have seen MQTT decouples publisher and subscriber, so a connection of any client is always with the broker. Before we start diving into the connection details, let's make clear what we mean by client and broker.

Client

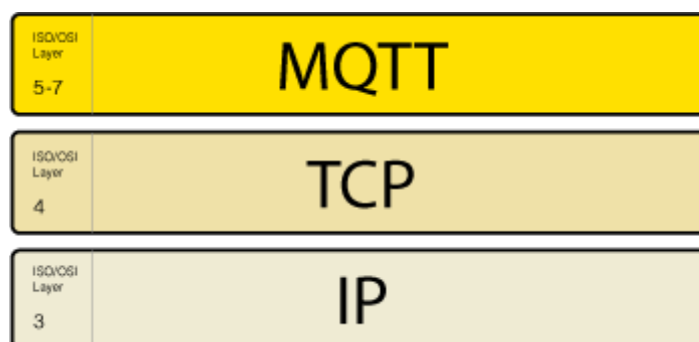
When talking about a client it almost always means an MQTT client. This includes publisher or subscribers, both of them label an MQTT client that is only doing publishing or subscribing. (In general a MQTT client can be both a publisher & subscriber at the same time). **A MQTT client is any device from a micro controller up to a full fledged server, that has a MQTT library running and is connecting to an MQTT broker over any kind of network.** This could be a really small and resource constrained device, that is connected over a wireless network and has a library strapped to the minimum or a typical computer running a graphical MQTT client for testing purposes, basically any device that has a TCP/IP stack and speaks MQTT over it. The client implementation of the MQTT protocol is very straight-forward and really reduced to the essence. That's one aspect, why MQTT is ideally suitable for small devices. **MQTT client libraries are available for a huge variety of programming languages, for example Android, Arduino, C, C++, C#, Go, iOS, Java, JavaScript, .NET.** A complete list can be found on the MQTT wiki (<https://github.com/mqtt/mqtt.github.io/wiki/libraries>).

Broker

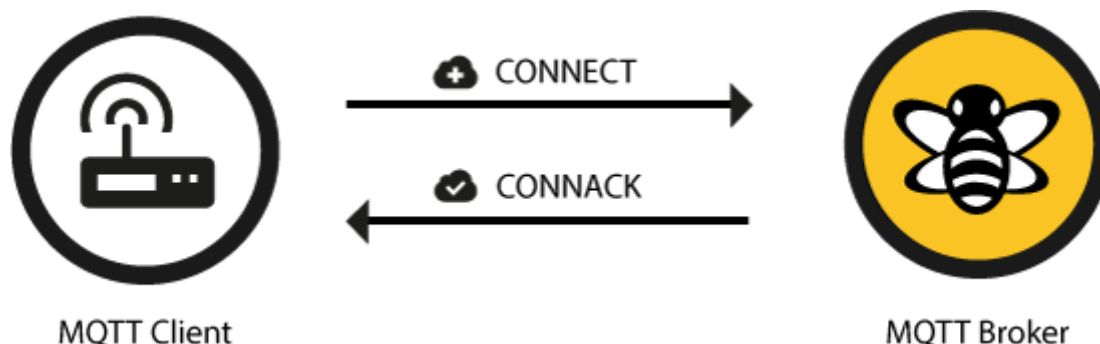
The counterpart to a MQTT client is the MQTT broker, which is the heart of any publish/subscribe protocol. Depending on the concrete implementation, a broker can handle up to thousands of concurrently connected MQTT clients. **The broker is primarily responsible for receiving all messages, filtering them, decide who is interested in it and then sending the message to all subscribed clients.** It also holds the session of all persisted clients including subscriptions and missed messages (More details (<http://www.hivemq.com/mqtt-essentials-part-7-persistent-session-queuing-messages/>)). Another responsibility of the broker is the authentication and authorization of clients. And at most of the times a broker is also extensible, which allows to easily integrate custom authentication, authorization and integration into backend systems. Especially the integration is an important aspect, because often the broker is the component, which is directly exposed on the internet and handles a lot of clients and then passes messages along to downstream analyzing and processing systems. As we described in one of our early blog post (<http://www.hivemq.com/mqtt-sql-database/>) subscribing to all message is not really an option. All in all the broker is the central hub, which every message needs to pass. Therefore **it is important, that it is highly scalable, integratable into backend systems, easy to monitor and of course failure-resistant.** For example HiveMQ solves this challenges by using state-of-the-art event driven network processing, an open plugin system and standard providers for monitoring.

MQTT Connection

The MQTT protocol is based on top of TCP/IP and both client and broker need to have a TCP/IP stack.



The MQTT connection itself is always between one client and the broker, no client is connected to another client directly. **The connection is initiated through a client sending a CONNECT message to the broker. The broker response with a CONNACK** and a status code. Once the connection is established, the broker will keep it open as long as the client doesn't send a disconnect command or it loses the connection.




MQTT connection through a NAT

It is a common use case that MQTT clients are behind routers, which are using network address translation (NAT) in order to translate from a private network address (like 192.168.x.x, 10.0.x.x) to a public facing one. As already mentioned the MQTT client is doing the first step by sending a CONNECT message. So there is no problem at all with clients behind a NAT, because the broker has a public address and the connection will be kept open to allow sending and receiving message bidirectional after the initial CONNECT.

Client initiates connection with the CONNECT message

So let's look at the MQTT CONNECT (http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html#_Toc398718028) command message. As already mentioned this is sent from the client to the broker to initiate a connection. If the CONNECT message is malformed (according to the MQTT spec) or it takes too long from opening a network socket to sending it, the broker will close the connection. This is a reasonable behavior to avoid that malicious clients can slow down the broker.

A good-natured client will send a connect message with the following content among other things:

MQTT-Packet:	
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>keepAlive</code>	<code>60</code>

Additionally there are other informations included in a CONNECT message, which are more a concern to the implementer of a MQTT library than to the user of a library. If you are interested in the details have a look at the MQTT 3.1.1 specification (<http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html>).

So let's go through all these options one by one:

ClientId

The client identifier (short ClientId) is an **identifier of each MQTT client** connecting to a MQTT broker. As the word identifier already suggests, it should be unique per broker. The broker uses it for identifying the client and the current state of the client. If you don't need a state to be hold by the broker, in MQTT 3.1.1 (current standard) it is also possible to send an empty ClientId, which results in a connection without any state. A condition is that clean session is true, otherwise the connection will be rejected.

Clean Session

The clean session flag indicates the broker, whether the **client wants to establish a persistent session or not**. A persistent session (CleanSession is false) means, that the broker will store all subscriptions for the client and also all missed messages, when subscribing with Quality of Service (QoS) 1 or 2. If clean session is set to true, the broker won't store anything for the client and will also purge all information from a previous persistent session.

Username/Password

MQTT allows to send a **username and password for authenticating the client and also authorization**. However, the password is sent in plaintext, if it isn't encrypted or hashed by implementation or TLS is used underneath. We highly recommend to use username and password together with a secure transport of it. In brokers like HiveMQ it is also possible to authenticate clients with an SSL certificate, so no username and password is needed.

Will Message

The will message is part of the last will and testament feature of MQTT. **It allows to notify other clients, when a client disconnects ungracefully.** A connecting client will provide his will in form of an MQTT message and topic in the CONNECT message. If this clients gets disconnected ungracefully, the broker sends this message on behalf of the client. We will talk about this in detail in an individual post.

KeepAlive

The keep alive is a time interval, the clients commits to by sending regular PING Request messages to the broker. The broker response with PING Response and this mechanism will allow both sides to determine if the other one is still alive and reachable. We'll talk about this in detail in a future post.

That are basically all information that are necessary to connect to a MQTT broker from a MQTT client. Often each individual library will have additional options, which can be configured. They are most likely regarding the specific implementation, for example how should queued message be stored.

Broker responds with the CONNACK message

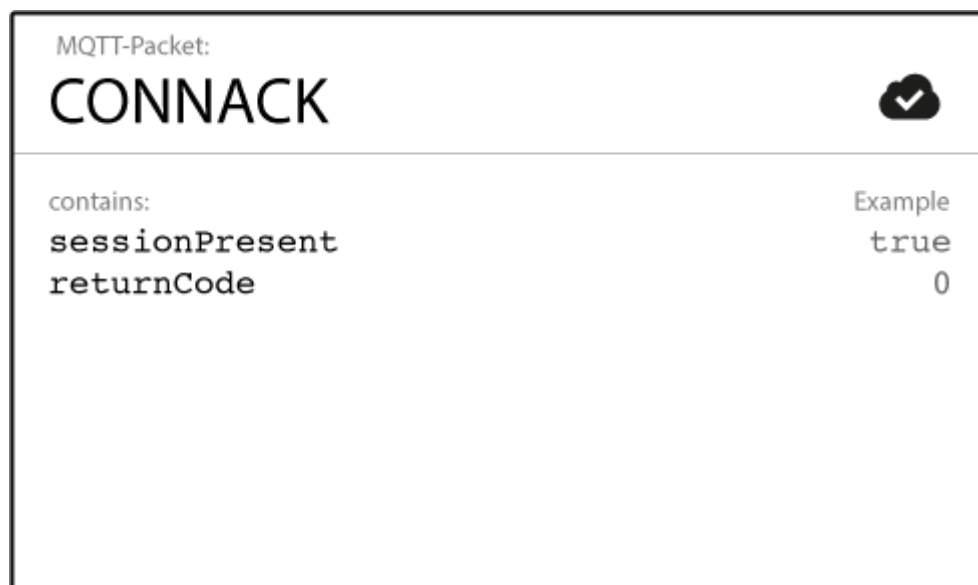
When a broker obtains a CONNECT message, it is obligated to respond with a CONNACK message. The CONNACK contains only two data entries: session present flag, connect return code.

Session Present flag

The **session present flag indicate, whether the broker already has a persistent session of the client from previous interactions.** If a client connects and has set CleanSession to true, this flag is always false, because there is no session available. If the client has set CleanSession to false, the flag is depending on, if there are session information available for the ClientId. If stored session information exist, then the flag is true and otherwise it is false. This flag was added newly in MQTT 3.1.1 and helps the client to determine, whether it has to subscribe to topics or if these are still stored in his session.

Connect acknowledge flag

The second flag in the CONNACK (http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html#_Toc398718033) is the connect acknowledge flag. It signals the client, if the **connection attempt was successful and otherwise what the issue is.**



In the following table you see all return codes at a glance.

Return Code	Return Code Response
0	Connection Accepted
1	Connection Refused, unacceptable protocol version
2	Connection Refused, identifier rejected
3	Connection Refused, Server unavailable
4	Connection Refused, bad user name or password
5	Connection Refused, not authorized

A more detailed explanation of each of these can be found in the MQTT specification (http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html#_Toc398718035).

Loose ends

You maybe ask, how MQTT keeps the connection open, even when there are no messages send? Or how to know when a connection is lost? You have to be patient, but we will devote a whole blog inside of the essentials series to that topic later on.

So that's the end of part three in our MQTT Essentials series. We hope you learned at least one new thing about MQTT and looking forward to the next post about how publishing, subscribing and unsubscribing works in MQTT (<http://www.hivemq.com/mqtt-essentials-part-4-mqtt-publish-subscribe-unsubscribe>).

If you want to be notified as soon as the next part is released, simply sign up for our newsletter below. This brings you fresh content about MQTT and HiveMQ once a week. If you prefer RSS, you can subscribe to our RSS feed here (<http://www.hivemq.com/feed/>).

Subscribe to all upcoming Blog Posts

weekly updates about MQTT and HiveMQ directly to your inbox

Subscribe

13 comments

Pingback: MQTT Essentials Part 4: MQTT Publish, Subscribe & Unsubscribe
(<http://www.hivemq.com/mqtt-essentials-part-4-mqtt-publish-subscribe-unsubscribe/>)

Pingback: MQTT Essentials: Publish & Subscribe (<http://www.hivemq.com/mqtt-essentials-part2-publish-subscribe/>)

Pingback: MQTT Essentials Part 9: Last Will and Testament (<http://www.hivemq.com/mqtt-essentials-part-9-last-will-and-testament/>)

Pingback: MQTT Essentials Part 10: Keep Alive and Client Take-Over (<http://www.hivemq.com/mqtt-essentials-part-10-alive-client-take-over/>)

Pingback: MQTT Security Fundamentals: Authentication with Username and Password (<http://www.hivemq.com/mqtt-security-fundamentals-authentication-username-password/>)



per 28, 2015 at 2:08 am (<http://www.hivemq.com/blog/mqtt-essentials-part-3-client-broker-connection-establishment/#comment-38716>)
Ahmed Al-Haddad says.

Wow you guys. Awesome work seriously!!!

Reply (<http://www.hivemq.com/blog/mqtt-essentials-part-3-client-broker-connection-establishment?replyto=38716#respond>)



per 17, 2015 at 11:38 am (<http://www.hivemq.com/blog/mqtt-essentials-part-3-client-broker-connection-establishment/#comment-40941>)
C- says.

Can the following sentence be improved?

"Once the connection is initiated MQTT will keep it open as long as the client doesn't send a disconnect command or it loses the connection (More details on how this is recognized is in a later post)."

C-:

Reply (<http://www.hivemq.com/blog/mqtt-essentials-part-3-client-broker-connection-establishment?replyto=40941#respond>)



per 20, 2015 at 7:17 pm (<http://www.hivemq.com/blog/mqtt-essentials-part-3-client-broker-connection-establishment/#comment-41154>)
The HiveMQ Team (<http://www.hivemq.com>) says.

Hi,

thanks for the feedback! We fixed that sentence and it should be easier to read now.

Dominik from the HiveMQ Team



Sundeeep Saitija says:
June 24, 2015 at 9:21 am (<http://www.hivemq.com/blog/mqtt-essentials-part-3-client-broker-connection-establishment/#comment-41265>)

Great Link for a quick go thru the MQTT essential.. Kudos,, Thanks

Reply (<http://www.hivemq.com/blog/mqtt-essentials-part-3-client-broker-connection-establishment?replytocom=41265#respond>)



VJ_time (<http://www.facebook.com>) says:
July 18, 2016 at 2:49 pm (<http://www.hivemq.com/blog/mqtt-essentials-part-3-client-broker-connection-establishment/#comment-47509>)

very good work! guys. thank you!

Reply (<http://www.hivemq.com/blog/mqtt-essentials-part-3-client-broker-connection-establishment?replytocom=47509#respond>)



VJ_time says:
July 18, 2016 at 2:49 pm (<http://www.hivemq.com/blog/mqtt-essentials-part-3-client-broker-connection-establishment/#comment-47511>)

great work! Guys! thank you!

Reply (<http://www.hivemq.com/blog/mqtt-essentials-part-3-client-broker-connection-establishment?replytocom=47511#respond>)



sumee says:
2016 at 7:25 pm (<http://www.hivemq.com/blog/mqtt-essentials-part-3-client-broker-connection-establishment/#comment-54983>)

Thanks for the information:-)

Reply (<http://www.hivemq.com/blog/mqtt-essentials-part-3-client-broker-connection-establishment?replytocom=54983#respond>)



Vincent says:
2016 at 2:34 am (<http://www.hivemq.com/blog/mqtt-essentials-part-3-client-broker-connection-establishment/#comment-55672>)

Useful info for new learners.Thanks for the materials.

Reply (<http://www.hivemq.com/blog/mqtt-essentials-part-3-client-broker-connection-establishment?replytocom=55672#respond>)

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

Website

Post Comment

« MQTT Essentials Part 2: Publish & Subscribe (<http://www.hivemq.com/blog/mqtt-essentials-part2-publish-subscribe>)

» MQTT Essentials Part 4: MQTT Publish, Subscribe & Unsubscribe (<http://www.hivemq.com/blog/mqtt-essentials-part-4-mqtt-publish-subscribe-unsubscribe>)

Sign up for our newsletter to keep up with the latest news about HiveMQ and MQTT in general ([Privacy Policy \(/privacy-policy/\)](#))

email address

Subscribe

© 2016 dc-square GmbH (<http://www.dc-square.de/>)



(<https://www.facebook.com/HiveMQ>)



(<https://www.linkedin.com/company/hivemq>)



(<https://www.twitter.com/HiveMQ>)



(<https://www.github.com/HiveMQ>)



(<https://www.facebook.com/HiveMQ>)



(<https://plus.google.com/+Hivemq/>)



(<http://www.hivemq.com/feed/>)

[Imprint \(/imprint/\)](/imprint/) [Privacy Policy \(/privacy-policy/\)](/privacy-policy/);