Mit MQTT einen Roboter steuern [Teil 1]

In den meisten Blogbeiträgen geht es um einfachere Steuerungen oder Regelungen die ein MicroController übernimmt. Doch die IoT-Geräte, Kurzform für Internet of Things, können weitaus mehr. Grundlage dafür bieten Protokolle, welche die Kommunikation zu den einzelnen Geräten sicherstellt, wie das hier genutzte MQTT. Dieser erste Blogbeitrag der Blogserie über MQTT vermittelt die Grundlagen. Am Ende der Blogserie wird mittels MQTT ein kleiner Roboter gesteuert, ähnlich wie mit einer Fernsteuerung.

Voraussetzung

Für diesen Blogbeitrag brauchen Sie nur wenige Bauteile, siehe Tabelle 1.

Pos	Anzahl	Bauteil	Link
1	1	Raspberry Pi	https://www.az-delivery.de/
2	1	Passendes Netzteil	

Tabelle 1: Benötigte Hardware

Denken Sie bei dem Pi daran, dass Sie neben der oben genannten Hardware auch eine MicroSD-Karte brauchen. Hierzu sollten Sie raspbian als Image auf die Karte installieren.

Was ist MQTT?

MQTT ist die Abkürzung für **M**essage **Q**ueuing **T**elemetry **T**ransport und ist auch noch unter dem älteren Namen WebSphere MQTT, kurz WSMQTT, bekannt. Es handelt sich dabei um ein offenes Netzwerkprotokoll für eine Machine-to-Machine-Kommunikation, um z.B. Sensordaten zu übermitteln und/oder Aktoren zu steuern. Hintergrund dieses Protokolls ist, dass auch in sogenannten instabilen Netzen, eine Übertragung von Informationen gewährleistet werden kann. Der Begriff „instabile Netze“ ist mittlerweile ein veralteter Begriff, als Netzwerke noch sehr Störanfällig waren.
MQTT ist, sofern es standardmäßig installiert wird, über den Netzwerkport 1883 erreichbar. Port 8883 ist der gesicherte Netzwerkport, muss aber mittels TLS-Protokoll verschlüsselt werden. Damit Sie MQTT benutzen können, braucht es einen MQTT-Server, den sogenannten Broker, sowie die entsprechenden Endgeräte, welche die Daten senden oder empfangen, auch Clients genannt. Meist findet sich der Broker auf einem Linux-basierten Betriebssystem. Windows-Betriebssysteme sind jedoch ebenfalls möglich. Mittlerweile findet man MQTT in der Automatisierungstechnik und bei IoT-Projekten, da es relativ einfach ist, ein MQTT-Netzwerk mit geringen Ressourcen aufzubauen.
Die aktuelle genutzte Protokollversion 3.1.1, sowie die vorherige Version 3.1, wurden 2010 unter einer freien Lizenz veröffentlicht, welche durch das Standardisierungsgremium OASIS spezifiziert wurde. Die aktuelle Protokollversion ist im Jahre 2010 erschienen und damit, zum Zeitpunkt der Blogveröffentlichung, schon 10 Jahre alt. Die aktuellste Protokollversion 5.0 ist im Jahre 2019 veröffentlicht und zugleich durch das Standardisierungsgremium OASIS spezifiziert worden.

Wie funktioniert MQTT

MQTT nutzt die ereignisgesteuerte Publish/Subscribe-Architektur, zu Deutsch Senden/Empfangen-Architektur. Dabei gibt es keine End-zu-End-Verschlüsselung, wie zum Beispiel bei einer Website-Verbindung, sondern einen Server, den sogenannten Broker, zu welchem sich Sender und

Empfänger, die sogenannten Clients, verbinden. Um diese Verbindungen besser zu verstehen, soll Abbildung 1 dies in vereinfachter Form darstellen.

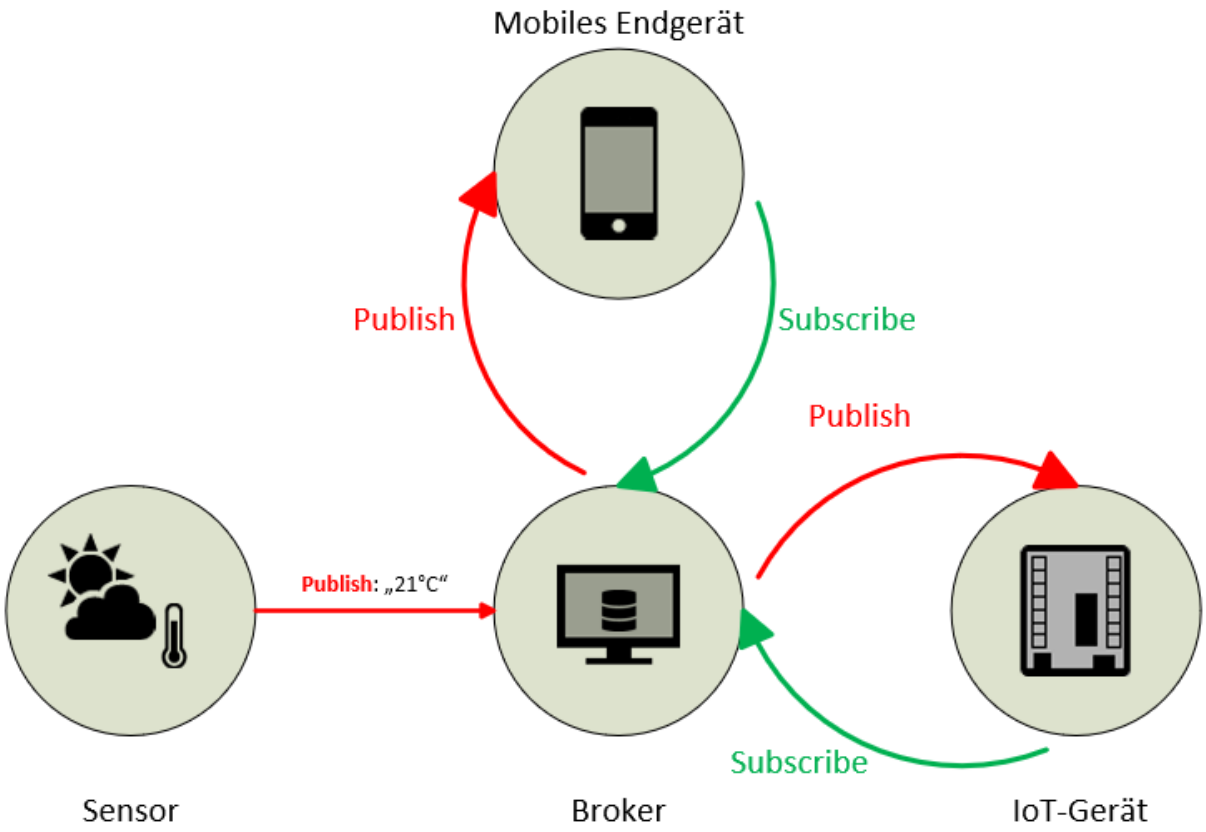

Abbildung 1: Veranschaulichung der MQTT-Architektur

Ein Client, hier im Bild der Sensor, das mobile Endgerät und das IoT-Gerät, kann Daten immer senden und/oder empfangen, der Broker dient lediglich als Informationspuffer, der die Daten an die Clients weiterleitet. Damit der Client eine entsprechende Nachricht vom Broker empfangen kann, muss dieser, beim Verbindungsaufbau oder aber während dem Betrieb, beim Broker die sogenannten Topics abonnieren. Ein Topic können Sie sich wie eine Art URL vorstellen, der den gewünschten Inhalt wiedergibt. Ein Topic für einen Temperatursensor im Wohnzimmer könnte die Messwerte auf *Daheim/Erdgeschoss/Wohnzimmer/Temperatur* veröffentlichen. Um gleich eins klarzustellen, es handelt sich nicht um eine Adresse zu dem Sensor, sondern um die zuletzt übermittelten Sensordaten. Wird der Messwert aktualisiert, prüft der Broker welcher Client diese Information zum Empfang abonniert hat und leitet diese weiter.

Wie Anfangs erwähnt, war MQTT für instabile Verbindungen gedacht, weswegen es drei unterschiedliche Servicequalitäten, im Englischen Quality of Service bzw. QoS, gibt. Bei Stufe 1 oder auch QoS = 0 wird die Nachricht genau einmal gesendet. Eine Bestätigung, wie es bei den anderen QoS der Fall ist, wird dabei komplett ignoriert. Einzig das Veröffentlichen, der sogenannte PUBLISH, ist wichtig.
Bei QoS = 1 spricht man auch von „mindestens einmal geliefert“, was so viel heißt, dass der Sender auf eine Bestätigung vom Empfänger wartet. Diese Empfangsbestätigung nennt sich PUBBACK und ist verpflichtend für die Gegenstelle, andernfalls übersendet der Sender solange, bis eine Empfangsbestätigung kommt. Dies führt im Umkehrschluss dazu, dass eine Nachricht mehrfach an den Empfänger übermittelt wird.
QoS = 2 ist quasi die Kombination aus QoS = 0 und QoS = 1, wobei Sie auch die langsamste Übertragung darstellt. Hier wird die Nachricht exakt nur ein einziges Mal gesendet. Um dies zu

erreichen ist eine zweistufige Empfangsbestätigung nötig. Zunächst sendet (PUBLISH) der Sender seine Nachricht an den Empfänger, welche die Nachricht annimmt und eine Bestätigungsnachricht (PUBREC) sendet, wobei der Inhalt vom Sender als eine Kopie mit angehängt ist. Wird diese Bestätigungsnachricht (PUBREC) vom Sender empfangen, speichert dieser die Information und antwortet ebenfalls mit einer Bestätigungsnachricht (PUBREL). Zuletzt, wenn der Empfänger die PUBREL erhalten hat, schickt dieser die letzte Bestätigungsnachricht (PUBCOMP). Am Ende einer solchen Übertragung von Daten ist sichergestellt, dass die Nachricht auch tatsächlich angekommen ist. Um es verständlicher zu halten, zeigt Abbildung 2 noch einmal schematisch, wie der Nachrichtenverlauf bei allen drei QoS-Varianten aussieht.

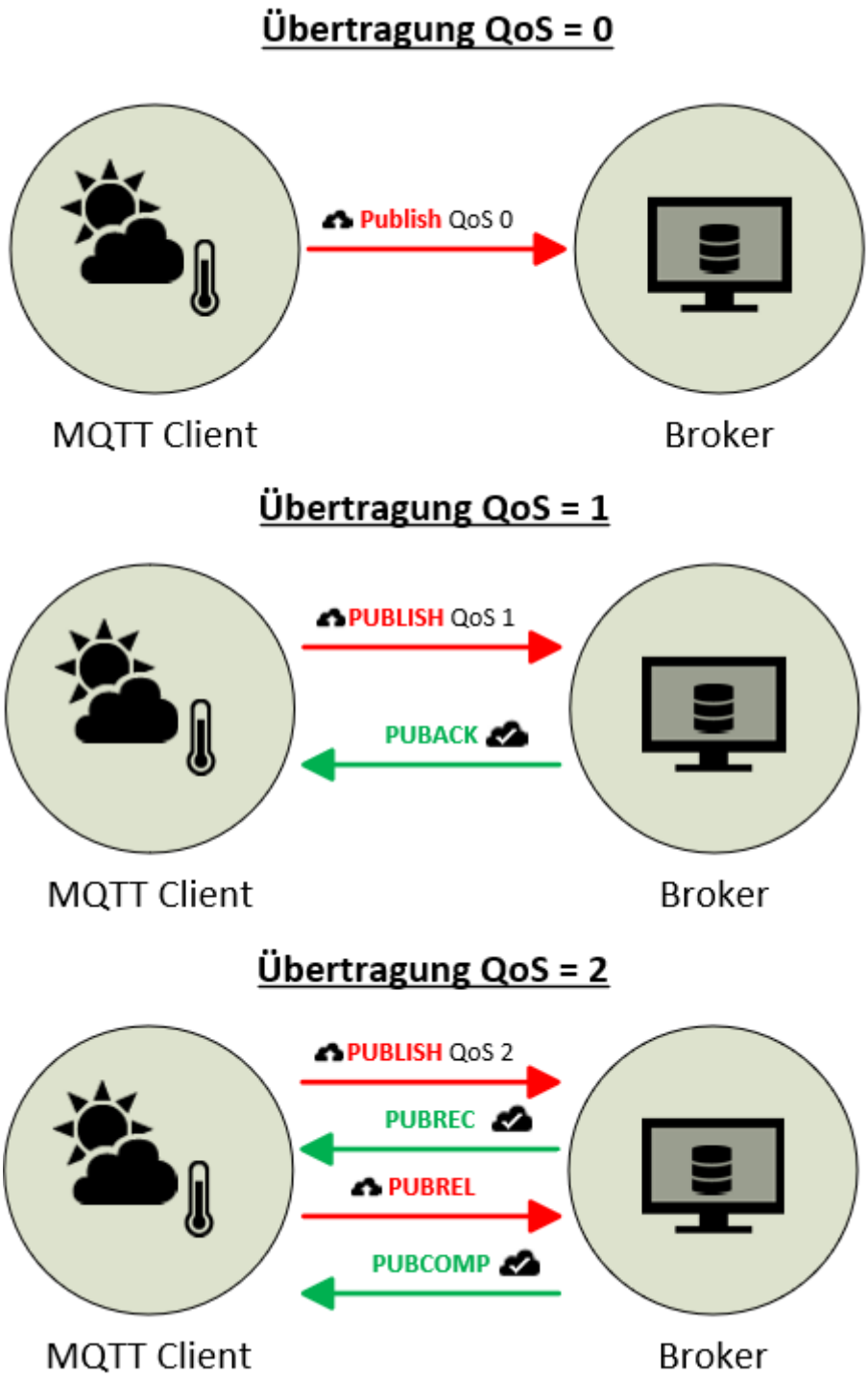

Abbildung 2: MQTT QoS-Übersicht

Nun stellt sich die Frage, wie ein Client nun Sensordaten erfassen kann? Dazu gibt es auch wieder mehrere Varianten, die zum Ziel führen können. In der ersten Variante können Sie, wie schon oben bei der einfachen Sensordatenübertragen gezeigt, den Topic komplett vorgeben. Dazu geben Sie dann einfach den Topic *Daheim/Erdgeschoss/Wohnzimmer/Temperatur* an. Sie erhalten an der Stelle dann aber auch tatsächlich nur genau diesen einen Wert.
Bei der Nutzung von Variante zwei kommt der +-Operator ins Spiel. Durch *Daheim/+/+/Temperatur* erhalten Sie für alle Zimmer auf jeder Etage die Temperaturdaten. Nützlich ich dies, wenn Sie eine solche Struktur für alle Sensoren in ihrem Haus so vorgeben und z.B. gebündelt anzeigen lassen wollen.
Variante drei geht mit dem #-Operator noch einen Schritt weiter. Durch *Daheim/Erdgeschoss/#* erhalten Sie alle verfügbaren Informationen für jedes Zimmer auf der Etage Erdgeschoss. Je nach Informationsquantität kann schon an dieser Stelle einiges zusammenkommen.
Der vierte und letzte Fall ist, sofern Sie Unmengen an Daten übermitteln, die wohl Datenintensivste Topicabonnierung. Mittels /# abonnieren Sie das Wurzelverzeichnis und ihnen wird jede Änderung übermittelt. Gerade diese Variante empfehle ich an dieser Stelle ausdrücklich nicht! Sie sollten sich schon im Vorfeld Gedanken machen, wie Ihre Struktur bzw. die Messdaten, die Sie aufnehmen wollen aussehen vorher festlegen.

Zuletzt kommt wahrscheinlich noch die Frage auf, was passiert den, wenn ein Sender plötzlich nicht mehr sendet? Dazu gibt es in MQTT die sogenannte Retained Message, die den letzten Willen und das Testament beinhaltet. Damit dies genutzt werden kann, muss der Sender, in unserem Fall ein Sensor, bei der Verbindung zum Broker eben diese Retained Message mitliefern. Darin ist gespeichert, was gesendet werden soll, wenn der Sender nicht mehr erreichbar ist. Pro Topic kann dann ein individueller Inhalt durch den Broker geschrieben werden. Da MQTT schon von Haus aus eine Überwachung mitliefert, brauchen Sie sich um diesen Punkt nicht zu kümmern. Da es zu tief in die Materie geht, kann ich an dieser Stelle aber gerne das Stichwort LWT message mitgeben, womit Sie im Internet genügend Lesestoff finden werden.

Der Raspberry Pi wird zum Broker

Wie in dem Kapitel „Was ist MQTT?“ erwähnt, erfreut sich dieses Protokoll größter Beliebtheit in der IoT-Szene. Sie wollen sicherlich Ihre persönliche Daten oder Daten zur Steuerung ihres Heimes nicht im Internet bei z.B mqtt-dashboard.com/ speichern. Daher werden Sie einen kleinen, lokalen Broker mit dem Raspberry Pi aufbauen. Dazu ist es erst einmal egal, welche Raspberry Pi - Version Sie nehmen und ob dieser schon irgendwelche Aufgaben übernimmt.
Öffnen Sie zunächst einmal das Terminal ihres Raspberry Pis, siehe Abbildung 3 und geben Sie danach die Befehlt aus Code 1 ein.

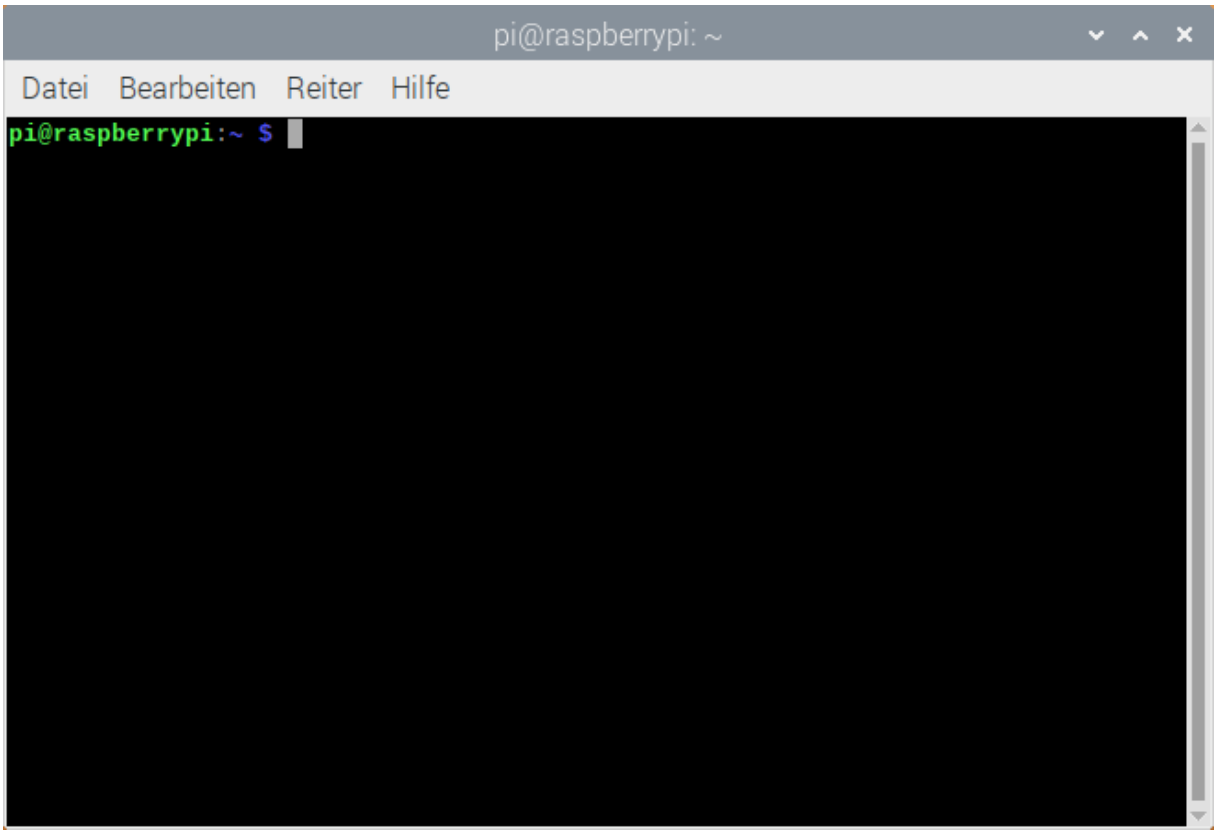

Abbildung 3: Öffne Terminal im Pi

```
sudo apt upgrade
sudo apt dist-upgrade
```

Code 1: Raspbian updaten

Damit aktualisieren Sie erst einmal den Raspberry Pi und alle installierten Pakete. Danach geben Sie den Befehl aus Code 2 ein. Damit laden Sie sich den beliebten mosquitto-broker und die Clientanwendung auf ihren Raspberry Pi. Bestätigen Sie an der entsprechenden Stelle die Installation im Terminal.

```
sudo apt install mosquitto mosquitto-clients
```

Code 2: Mosquitto-Server und -Client installieren

Damit der Broker auch nach einem Neustart direkt mitstartet nutzen Sie den Befehl aus Code 3.

```
sudo systemctl enable mosquitto.service
```

Code 3: Service von mosquitto bei Neustart vom Pi mit starten

Da der deamon auf dem Pi bisher ggf. noch nicht läuft, starten Sie diesen mit dem Befehl aus Code 4.

```
sudo mosquitto -d
```

Code 4: mosquitto starten

Ab diesem Punkt haben Sie einen lokalen, unverschlüsselten MQTT-Broker in ihrem Heimnetzwerk eingerichtet. Dieser ist, sofern Sie keine Weiterleitung von Ports auf den Raspberry Pi über ihren Router eingerichtet haben, im Internet nicht erreichbar. Für den privaten Zweck soll es aber erst einmal reichen. Damit auch der Broker vom Pi nach einem Neustarts erreichbar bleibt, sollten Sie dem Raspberry Pi eine fest IP-Adresse über den Router vergeben.

Daten an den Broker senden und empfangen

Nun wird es Zeit die Theorie in die Praxis umzusetzen und Nachrichten an den Broker zu senden und auch Nachrichten vom Broker zu empfangen. Dazu werden die Terminalbefehle „mosquitto_sub“ zum subscriben und „mosquitto_pub“ zum publishen von Nachrichten benötigt.
Den Anfang wird der Terminalbefehl „mosquitto_sub“ machen und die wichtigsten Befehle werden nachfolgend erläutern. Wollen Sie über die hier gezeigten Befehle hinausgehen, empfehle ich an dieser Stelle das intern mitgelieferte Linux-Handbuch mit dem Kommando „man mosquitto_sub“, „man mosquitto_pub“ oder „man mosquitto“.
Um umfangreich zu sehen, was der mosquitto-Subscriber im Hintergrund abarbeitet und auch um alle Nachrichten zu empfangen, nutzen Sie den Befehl aus Code 5.

```
mosquitto_sub -d -q 2 -t /#
```

Code 5: Mosquitto-Subscriber auf Wurzelknoten lauschen lassen

Der hier gezeigte Code 5 nutzt dabei die Optionen aus Tabelle 2.

Option	Erläuterung
-d	Aktiviert den Debug-Modus und zeigt alle nützlichen Informationen an.
-q	Spezifiziert welche QoS der Subscriber mit dem Broker kommunizieren soll. In Code 5 wird QoS 2 genutzt.
-t	Gibt an, welches Topic vom Subscriber abonniert werden soll. In Code 5 abonnieren der Subscriber alle Topics.

Tabelle 2: Optionen des mosquitto-Subscriber

Haben Sie den Befehl aus Code 5 eingegeben, wird die Ausgabe zunächst die erfolgreiche Verbindung zu dem Broker anzeigen, siehe Abbildung 4 Markierung 1.

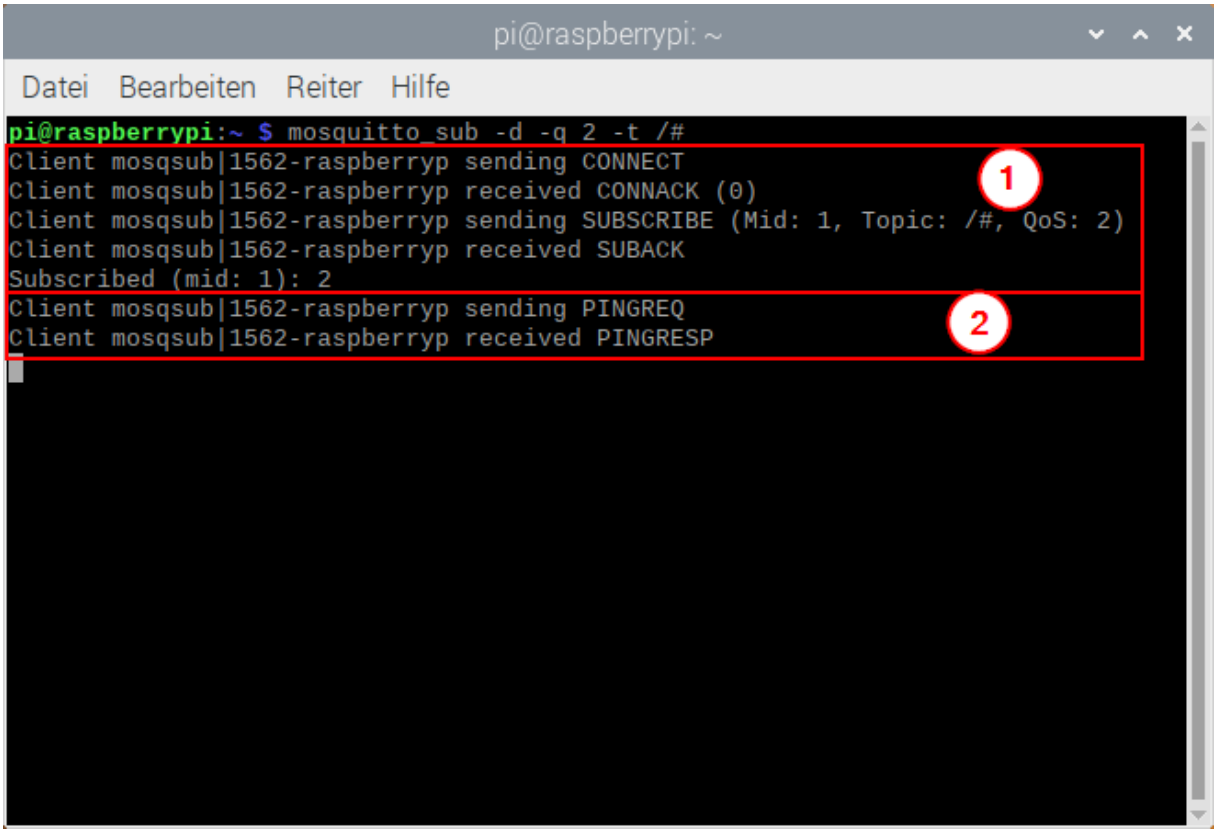

Abbildung 4: Terminalausgabe von moquitto_sub

In Markierung 1 von Abbildung 4 sieht man, dass eine Verbindung aufgebaut wird und die Gegenstelle, der Broker, die Verbindung akzeptiert hat. Direkt danach übermittelt der Subscriber welches Topic abonniert werden soll, hier „/#“, und welcher QoS angefordert wurde, hier „QoS: 2“. Zuletzt wird die Subscribe-Anfrage mittels SUBACK akzeptiert. Die Verbindung zum Broker ist hergestellt und das gewünschte Topic ist abonniert.
Markierung 2 von Abbildung 4 wird nach einer Zeit bei Ihnen im Terminal erscheinen. Dies ist, wie in „Wie funktioniert MQTT“ beschrieben, die Abfrage, ob der Subscriber noch online ist. Der Broker fragt nach, ob der Subscriber noch online (Request) ist, welcher einer entsprechenden Antwort (Request) an den Broker sendet.

Der nächste Schritt, nachdem wir nun ein Subscriber haben, ist einen Publisher zu nutzen, um Daten an den Broker zu senden. Die erste Nachricht soll natürlich „Hello world“ sein. Diesen kleinen Satz sendet der Publisher an das Topic /test/testpub.
Der Befehl dafür zeigt Code 6.

```
mosquitto_pub -d -q 2 -t /test/testpub -m “Hello world“
```

Code 6: mosquitto_pub "Hello world"

Die genutzten Optionen für Code 6 sind in Tabelle 3 aufgelistet.

Option	Erläuterung
-d	Aktiviert den Debug-Modus und zeigt alle nützlichen Informationen an.
-q	Spezifiziert welche QoS der Publisher mit dem Broker kommunizieren soll. In Code 6 wird QoS 2 genutzt.
-t	Gibt an, an welches Topic die Daten gesendet werden sollen, hier /test/testpub
-m	Sende nachfolgende Nachricht, hier „Hello world“

Tabelle 3: Optionen des mosquitto-Publisher

Schauen wir uns einmal an, was bei der Übermittelung von Daten, siehe Abbildung 5 Markierung 1, an den Broker gesendet werden. Wie schon zuvor wurde QoS 2 in den Optionen gewählt, weswegen Sie auch den entsprechenden Kommunikationsverlauf sehen.

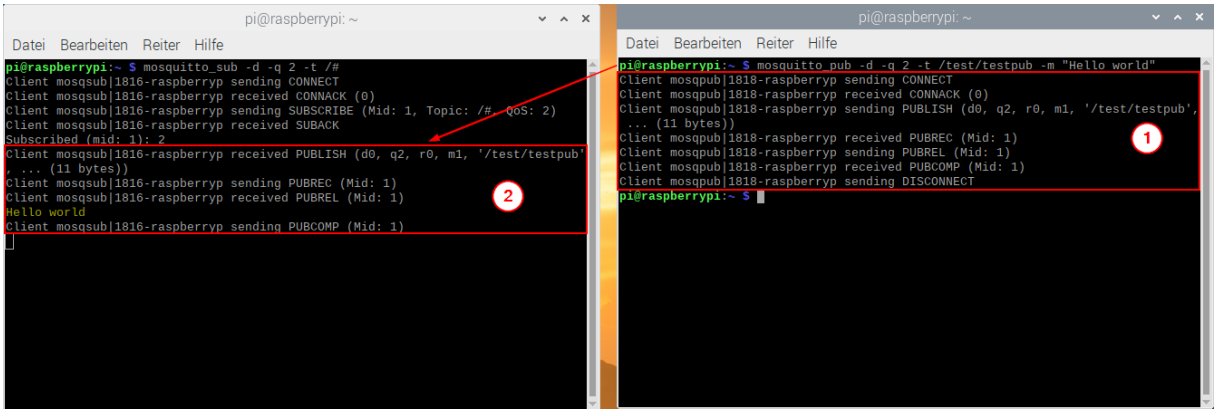

Abbildung 5: Terminalausgabe von moquitto_pub und mosquitto_sub

Direkt danach schickt der Broker die neue Nachricht an den Subscriber, siehe Abbildung 5 Markierung 2, da dieser alle Topics abonniert hat. An der oberen Umrandung von Markierung 2 von Abbildung 5 sehen Sie, dass neue Daten im Topic /test/testpub empfangen wurden. Der QoS 2 – Handshake wird direkt dahinter angezeigt, wobei die Nachricht „Hello world“ empfangen wurde.

Damit haben Sie die Grundlagen von MQTT und die technischen Hintergründe zu MQTT gelernt. Spielen Sie ruhig einmal mit den Befehlen und senden Sie Daten an den Broker und öffnen Sie ggf. weitere Terminals, die andere Topics abonnieren.
Im nächsten Teil der Blogserie werden Sie einen Arduino bzw. ESP32 mit dem MQTT verbinden und Daten senden und die Gegenstelle entsprechend darauf reagieren.

Dieses und weitere Projekte finden sich auf GitHub unter https://github.com/M3taKn1ght/Blog-Repo