Geschichte von MQTT

MQTT steht für **Message Queuing Telemetry Transport** und ist ein Netzwerkprotokoll für die Kommunikation von Machine to Machine (M2M).
Das MQTT-Protokoll wurde 1999 von Dr. Andy Stanford-Clark (IBM) und Arlen Nipper (Arcom, jetzt Cirrus Link) erfunden. Sie benötigten ein Protokoll für minimalen Batterieverlust und minimale Bandbreite, um eine Verbindung mit Ölpipelines über Satellit herzustellen. Allerdings hat sich der primäre Fokus des Protokolls von eingebetteten Systemen zu offenen Internet of Things (IoT) Anwendungsfällen geändert.
Am 29. Oktober 2014 wurde MQTT ein offiziell anerkannter *OASIS-Standard. Im März 2019 ratifizierte die OASIS die neue Spezifikation MQTT 5. Mit dieser neuen MQTT Version wurden neue Funktionen in MQTT eingeführt, die für IoT-Anwendungen erforderlich sind.

Client — Server (Broker) — Prinzip

Die Sender (Publisher) und der Abonnent (Subscriber) treten nie direkt miteinander in Kontakt. Tatsächlich sind sie sich nicht einmal bewusst, dass der andere existiert. Die Verbindung zwischen ihnen wird von einer dritten Komponente, dem MQTT Broker abgewickelt. Die Aufgabe des Brokers ist es, alle eingehenden Nachrichten zu filtern und sie korrekt an die Abonnenten zu verteilen. Der MQTT-Broker steht im Mittelpunkt jedes Publish / Subscribe Protokolls und er kann bis zu Tausende gleichzeitig verbundene MQTT Client verwalten.
Jeder Teilnehmer kann Daten an den Broker senden. Als Beispiel hier ein Temperatursensor. Der sendet die Daten genannt Payload, hier 20 °C an das Topic: „temperatur“. Das Topic ist das Thema (Betreff) für die Daten. Jeder Teilnehmer der das Topic „temperatur“ abonniert hat, bekommt vom Broker die Daten übermittelt.

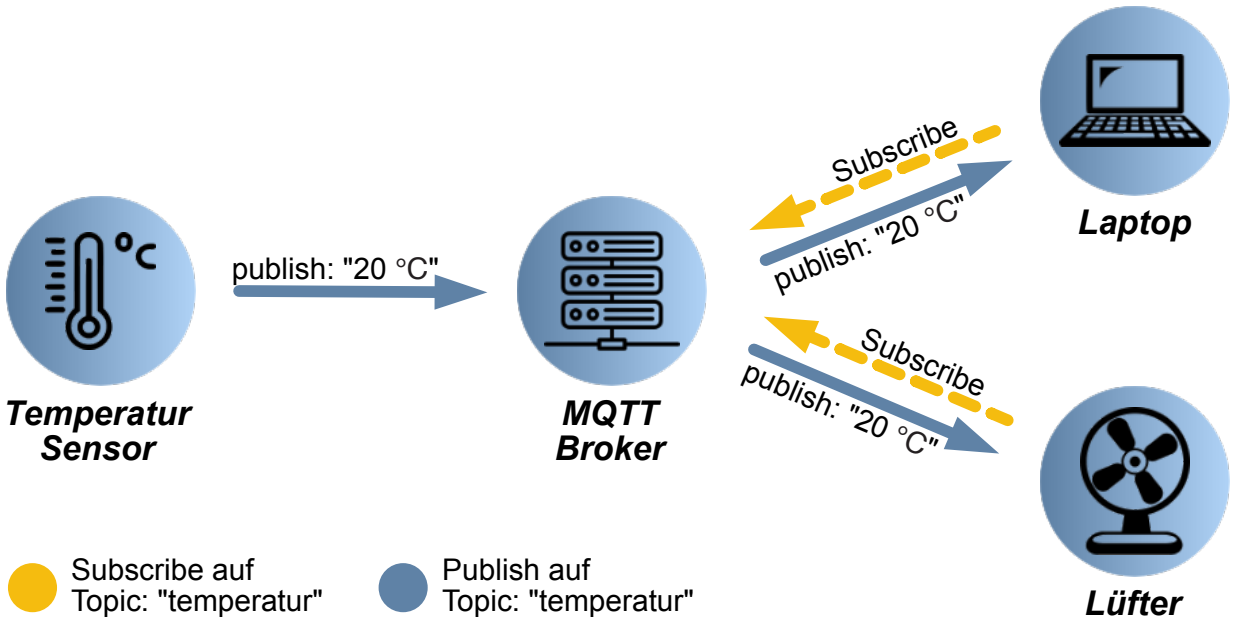


Bild 4.1.1: MQTT Publish / Subscribe Architektur

* Die **Organization for the Advancement of Structured Information Standards (OASIS)** ist eine internationale, nicht-gewinnorientierte Organisation, die sich mit der Weiterentwicklung von E-Business- und Webservice-Standards beschäftigt.

Der MQTT Broker

Es gibt einige Möglichkeiten einen MQTT-Broker zu benutzen. Man findet viele MQTT-Broker online, einige sind kostenlos und andere können gemietet werden. Ich kenne keinen dieser Broker und möchte daher auch keine Empfehlung aussprechen.
Ich verwende lieber meinen eigenen Homeautomation Server und habe mir dort den MQTT-Broker Mosquitto in Home Assistant installiert. Wie das geht, findet ihr im Kapitel unter HASSIO Installieren im Kapitel Installation von add-ons.
Der Mosquitto MQTT-Broker ist unter der IP-Adresse von Home Assistant erreichbar auf dem Port 1883.

Der MQTT-Client

Ein MQTT-Client kann beides sein, ein Publisher und ein Subsriber, je nachdem ob der Client gerade Nachrichten sendet oder empfängt. Ein MQTT-Client kann ein beliebiges Gerät (von einem Mikrocontroller bis hin zu einem vollwertigen Server) sein, auf dem eine MQTT-Bibliothek läuft und das sich über ein Netzwerk mit einem MQTT-Broker verbindet.

Das MQTT-Topic

Das Topic übersetzt Thema, wird vom Broker zum Filtern von Nachrichten aller Clients verwendet. Das Topic besteht aus einer oder mehreren Themenebenen. Jede Themenebene wird durch einen Schrägstrich (/) getrennt. Bei den Topics wird zwischen Groß- und Kleinschreibung unterschieden. Beispielsweise sind Schlafzimmer/Temperatur und schlafzimmer/temperatur zwei verschiedene Topics.
Welche Struktur ihr für eure Topics wählt, ist euch überlassen und es gibt so gut wie keine Einschränkungen. Nur das $ Zeichen darf nicht am Anfang eines Topics verwendet werden, da es für interne Statistiken vom Broker verwendet wird.
Hier ein paar Grundregeln um eine saubere Topic Struktur zu erstellen.

- Verwende den / nicht am Beginn eines Topics.
Es ist zwar erlaubt aber dadurch wird ein weiterer Topic Level erzeugt der mit einem Zero Zeichen gefüllt wird und nichts bringt.
Beispiel: /haus/obergeschoss/schlafzimmer

- Leerzeichen in den Topics vermeiden
Ein Leerzeichen ist schwer zu lesen und sollte vermieden werden, um Fehler zu vermeiden.

- Kurze Topics verwenden
In jeder Nachricht wird das Topic mitgesendet und um unnötig Ressourcen zu verschwenden, sollten Topics kurz gehalten werden.

- Keine Sonderzeichen verwenden
Weil Zeichen, die nicht ASCII-UTF-8 sind, oft falsch dargestellt werden, ist die Verwendung von Umlauten oder Sonderzeichen nicht zu empfehlen.

Topic Wildcards

Wenn ein Client ein Topic abonniert, kann er genau das Topic einer veröffentlichten Nachricht abonnieren oder er kann Platzhalter verwenden, um mehrere Topics gleichzeitig zu abonnieren. Ein Platzhalter kann nur zum Abonnieren von Topics verwendet werden, nicht zum Versenden einer Nachricht. Es gibt zwei verschiedene Arten von Platzhalter, einstufig und mehrstufig.

Einstufig:
Mit dem + Zeichen kann ein oder mehrere Topic Level ersetzt werden, machen wir hierzu ein Beispiel. Nehmen wir an, dass wir alle Temperaturen im Erdgeschoss ansehen wollen. Die wildcard wird für die verschiedenen Räume im Erdgeschoss verwendet.

Topic: Haus/Erdgeschoss/+/Temperatur

Als Ergebnis haben wir folgende Topic‘s abonniert.

- Haus/Erdgeschoss/**Kueche**/Temperatur
- Haus/Erdgeschoss/**Stauraum**/Temperatur
- Haus/Erdgeschoss/**Terrasse**/Temperatur

Mehrstufig:
Mit dem # Zeichen werden mehrere Topic Level ersetzt und das # Zeichen steht immer an letzter Stelle und ersetzt die nachfolgenden Topic Level, unabhängig wie tief sie verschachtelt sind.
Wird nur das # hashtag Symbol als Topic verwendet, dann werden alle Nachrichten die über MQTT gesendet werden abonniert. Aber machen wir zum besseren Verständnis wieder ein Beispiel.

Topic: # (Die sogenannte Root-Wildcard abonniert alle Topics)

Topic: Haus/Erdgeschoss/#

Als Ergebnis haben wir folgende Topics abonniert.

- Haus/Erdgeschoss/**Kueche/Temperatur**
- Haus/Erdgeschoss/**Kueche/Luftfeuchte**
- Haus/Erdgeschoss/**Stauraum/Temperatur**
- Haus/Erdgeschoss/**Stauraum/Licht**
- Haus/Erdgeschoss/**Terrasse/Temperatur**
- U.s.w.

Retained Messages – zurückgehaltene Nachricht

Verbindet sich ein Client zum ersten Mal mit einem Topic, so bekommt er normalerweise erst eine Nachricht, wenn ein anderer Client eine Nachricht an dieses Topic sendet. Mithilfe von zurückgehaltenen Nachrichten erhalten neu abonnierte Clients sofort nach dem Abonnieren eines Topics eine Statusaktualisierung. Durch die zurückgehaltene Nachricht entfällt die Wartezeit, bis die veröffentlichenden Clients die nächste Aktualisierung senden. Eine Anzeige für die Temperatur würde nach der Verbindung mit einem Topic also so lange keinen Wert anzeigen, bis ein Temperaturfühler einen neuen Wert an das Topic sendet. Die Temperaturanzeige könnte also anstatt keinem Wert zumindest einen älteren Wert anzeigen.
Das Senden einer Retained Message erfolgt mit dem Setzen der Retained-Flag auf true.

Persistent Session – dauerhafte Verbindung

Wenn eine nicht-persistent Session zwischen dem Client und dem Broker abbricht, gehen die Topics die der Client abonniert hat verloren. Bei jeder neuen Verbindung muss der Client sich wieder für alle Topics erneut anmelden und das kostet viel Ressourcen. Um dieses Problem zu vermeiden, kann der Client beim Verbinden zum Broker eine dauerhafte Verbindung anfordern. Dazu wird eine cleanSession-Flag im CONNECT Packet verwendet, wenn sie auf false gesetzt ist, wird eine dauerhafte Verbindung erstellt, bei true nicht. Damit werden alle Informationen, die für den Client relevant sind auf dem Broker gespeichert und über die Client-ID wird diese Sitzung identifiziert. Wenn die Verbindung abbricht und sich der Client wieder neu mit dem Broker verbindet, dann werden ihm alle Nachrichten mit einem QoS 1 und QoS 2 zugestellt die er noch nicht erhalten hat und er wird wieder automatisch mit allen Topics verbunden die er abonniert hat.

Quality of Service Levels

QoS steht für Quality of Service und gibt an, wie sicher eine Nachricht zugestellt werden soll. Es gibt drei QoS Stufen von 0 bis 2, aber schauen wir uns jede einzelne Stufe etwas genauer an.

- QoS 0 - at most once (höchstens einmal)
Der niedrigste QoS Level ist Null und dieser Level gibt keine Garantie für eine erfolgreiche Zustellung. Die Nachricht wird nach dem Senden nicht gespeichert oder noch einmal gesendet. Diese Methode nennt man auch „Fire and Forget“.

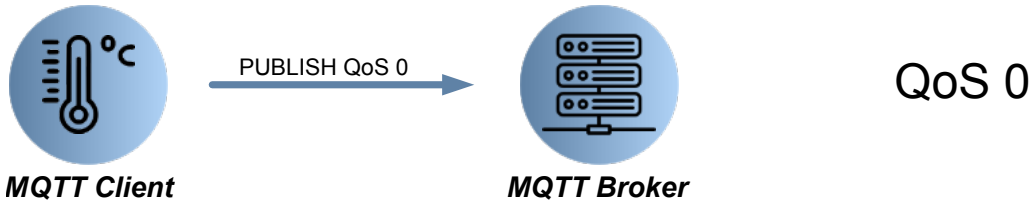


Bild 4.1.2: MQTT QoS 0

- QoS 1 - at least once (mindestens einmal)
Dieser QoS Level 1 garantiert, dass eine Nachricht mindestens einmal beim Empfänger ankommt und ist auch das am meisten benutze QoS. Der Absender verwendet eine Packet-ID beim Versenden jeder Nachricht und speichert die Nachricht, bis er ein PUBACK-Paket vom Empfänger zurückbekommt und damit der Empfang der Nachricht bestätigt wird. Das PUBACK (Publish acknowledgement) enthält die PACKET-ID um Sicherzustellen das die richtige Nachricht empfangen wurde. Wenn der Sender kein PUBACK-Paket erhält, dann sendet er die Nachricht noch einmal und markiert sie als Duplikat (DUP), dadurch ist es möglich, dass die Nachricht mehrfach gesendet wird.

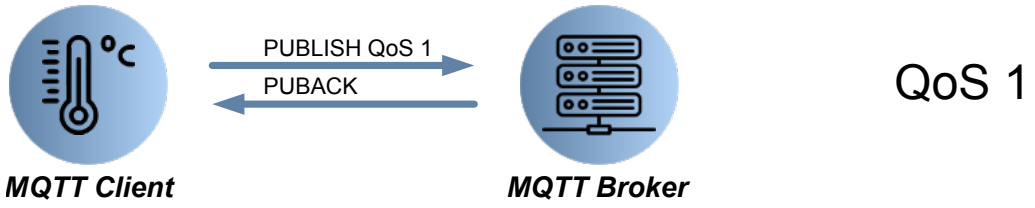


Bild 4.1.3: MQTT QoS 1

- QoS 2 - exactly once (genau einmal)
Der letzte und höchste Level von QoS garantiert, dass jede Nachricht nur einmal beim Empfänger ankommt. Ein Zweistufiger Bestätigungsablauf zwischen dem Sender und dem Empfänger sorgt dafür, das die Garantie eingehalten wird. Durch die zusätzlichen Nachrichten verzögert sich die Zustellung und wertvolle Bandbreite wird beansprucht.
Die Packet-ID der Nachricht wird auch hier mitgesendet vom Absender und die Nachricht wird gespeichert, bis er ein PUBREC-Packet (Publish received) vom Empfänger erhält. Erhält der Sender kein PUBREC-Packet, dass die Packet-ID beinhaltet, dann wird die Nachricht noch einmal gesendet und als Duplikat (DUP) markiert.
Hat der Sender das PUBREC-Packet erhalten, wird die Nachricht gelöscht und ein PUBREL-Packet (Publish release) zum Empfänger gesendet, die wieder die Packet-ID beinhaltet. Den Erhalt bestätigt der Empfänger, indem er ein PUBCOMP-Packet (Publish complete) mit der Packet-ID als Inhalt an den Sender schickt.

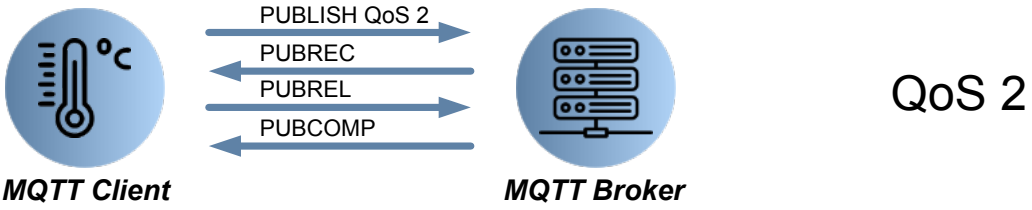


Bild 4.1.4: MQTT QoS 2

Last Will and Testament

Mit der Last Will and Testament (LWT) Funktion kann im Broker beim Ersten verbinden, das Testament vom Client gespeichert werden. Das LWT ist eine normale MQTT Nachricht im CONNECT Packet mit allen Parameter wie Topic, QoS, retained message flag und Payload. Wenn der Client die Verbindung zum Broker verliert, dann sendet der Broker die Last-Will-Nachricht an alle Clients, die das Last-Will-Topics abonniert haben.

MQTT Publish – Nachricht senden

Jeder MQTT-Client kann eine Nachricht senden (publish) der mit dem MQTT-Broker verbunden ist. Jede Nachricht benötigt ein Topic, damit der Broker an alle Abonnenten (subscriber) die Nachricht weiter leiten kann. Der Nachrichteninhalt ist der Payload aber eine Nachricht kann noch mehrere Parameter enthalten wie wir schon gelernt haben. Hier zusammengefasst noch einmal die Parameter für MQTT-Publish.

- Topic Name: Der hierarchisch strukturierte Name des Themas
- Payload: Der Inhalt der Nachricht
- QoS: Quality of Service mit 0, 1 oder 2
- Retain Flag: Gibt an, ob die Nachricht im Broker gespeichert werden soll
- Packet Identifier: Eine eindeutige Bezeichnung um Nachrichten identifizieren zu können
- DUP Flag: Ob die Nachricht ein Duplikat ist und ein weiteres Mal gesendet wird

MQTT Subscribe – Nachricht abonnieren

Um Nachrichten zu einem bestimmten Topic zu erhalten, abonniert der Client beim MQTT-Broker ein Topic auf dem es Nachrichten empfangen möchte.
Dazu sendet der Client an den Broker eine Nachricht mit zwei Attributen.

- Packet Identifier: Eine eindeutige Bezeichnung um Nachrichten identifizieren zu können
- List of Subscriptions: Es können mit einer Nachricht mehrere Topics als Liste abonniert werden und jedem Topic kann ein QoS-Level zugewiesen werden.

Zur Bestätigung das ein Topic abonniert wurde, sendet der Broker eine SUBACK (Subscribe acknowledgement Nachricht) zurück an den Client. Diese Nachricht beinhaltet die Packet-ID und pro Topic das abonniert wurde einen Return Code. Der Return-Code der jedes Topic bestätigt enthält für QoS 0 die Nummer 0, 1 für QoS 1 und 2 für QoS 2 Nachrichten. Wenn der Broker das Abonnieren ablehnt, sendet er als Return-Code 128 zurück.

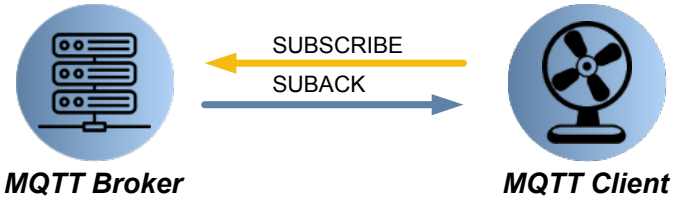


Bild 4.1.5: MQTT Subscribe

MQTT Unsubscribe – Nachricht abbestellen

Wenn Nachrichten nicht mehr benötigt werden, dann können sie auch wieder abbestellt werden. Das funktioniert wie beim Abonnieren von einer Nachricht und der Client sendet an den Broker eine Nachricht mit zwei Attributen.

- Packet Identifier: Eine eindeutige Bezeichnung um Nachrichten identifizieren zu können
- List of Topics: Es können mit einer Nachricht mehrere Topics die man nicht mehr benötigt als Liste gesendet werden.

Zur Bestätigung das ein Topic abbestellt wurde, sendet der Broker eine UNSUBACK (Unsubscribe acknowledgement Nachricht) zurück an den Client und diese Nachricht beinhaltet die Packet-ID.

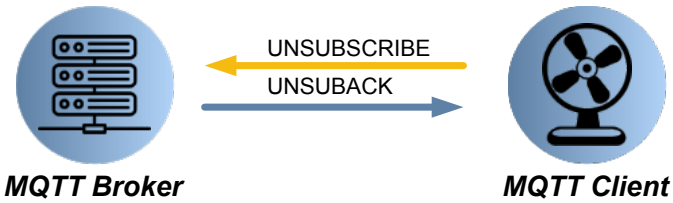


Bild 4.1.6: MQTT Unsubscribe

Erste Verbindung zum Broker - CONNECT Packet

Bei der ersten Verbindung zum Broker sendet der Client ein CONNECT Packet an den Broker. Dieses Packet beinhaltete den Namen vom Client und ob es eine cleanSession oder eine persistent Session sein soll. Optional kann noch ein Username und Passwort gesendet werden und die Parameter für Last Will and Testament (LWT). Der Broker bestätigt mit einem CONNACK Packet die Verbindung. Hier zusammengefasst die Parameter für das CONNECT Packet.

- **ClientID:** Das ist der Name vom Client
- **CleanSession:** Gibt an ob die Verbindung Dauerhaft sein soll
- **Username:** Der Benutzername zum Verbinden mit dem Broker
- **Password:** Das Passwort für die Verbindung mit dem Broker
- **LastWillTopic:** Das Topic an dem der letzte Wille gesendet werden soll
- **LastWillQoS:** QoS für den LWT
- **LastWillMessage:** Die Nachricht für LWT
- **LastWillRetain:** Gibt an ob der LTW dauerhaft beim Broker gespeichert werden soll
- **KeepAlive:** Angabe in Sekunden ohne Kommunikation

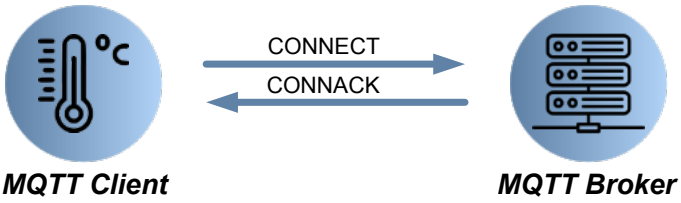


Bild 4.1.7: MQTT Connect Packet

MQTT - Keep Alive

Mit Keep Alive definieren wir eine Zeit in Sekunden in der eine Kommunikation zwischen dem Client und Broker stattfinden muss. Wird in dieser vorgegebenen Zeit keine Nachricht gesendet, dann wird vom Client ein PINGREQ Paket (Ping request) gesendet, dass von Broker mit einem PINGRESP Packet (Ping response) bestätigt wird. Damit können wir Sicherstellen, dass die Verbindung noch steht.
Wird die Keep Alive Zeit um 50 % überschritten ohne das eine Nachricht gesendet wurde, dann trennt der Broker die Verbindung zum Client. Wenn der Client das PINGRESP file vom Broker nicht erhält, dann wird die Verbindung vom Client geschlossen.

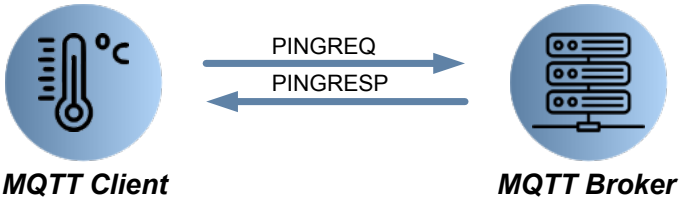


Bild 4.1.8: MQTT Keep Alive

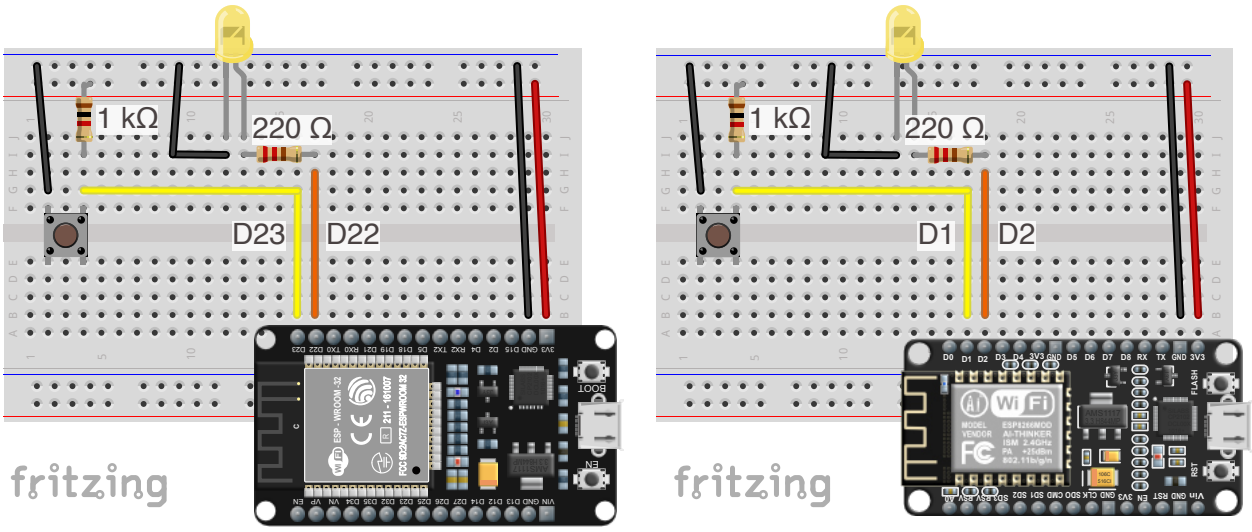
Praktisches MQTT Beispiel

MQTT - Praktisches Beispiel

In diesem Praxisbeispiel haben wir zweimal denselben Aufbau. Einmal mit dem NodeMCU-ESP8266 und einmal mit dem NodeMCU-ESP32. An beide Mikrocontroller sind jeweils eine LED und ein Taster angebracht. Wenn ein Taster gedrückt wird, dann wird die LED über ein MQTT Topic am anderen Mikrocontroller eingeschaltet, solange der Taster gedrückt wird. An dem Taster ist jeweils ein Pull-up Widerstand verbaut mit einem Widerstandswert von 1 kΩ.

Benötigte Teile

* https://amzn.to/3fzAe0M - Widerstand Set
* https://amzn.to/3oUZ14g - 300 Stk. Leuchtdioden Set 3 und 5 mm
* https://amzn.to/34ifkjS - 180 Stück Taktilen Drucktaster Sortiment
* https://amzn.to/2QQ7JSL - Breadboard Steckbrett mit 830 Kontakten oder
* https://amzn.to/35GcPIp - 3 Stk. Breadboard Steckbrett mit 400 Kontakten
* https://amzn.to/2xeKb2V - Set 3 x 40 STK. je 20 cm M2M/ F2M / F2F
* https://amzn.to/2U7Urmn - NodeMCU ESP8266 ESP-12F
* https://amzn.to/3erQVKq - ESP32 Node-MCU Board



MQTT Praxis Beispiel – Teil 1 mit NodeMCU-ESP8266

Starten wir mit dem ESP8266 und gehen den Source Code und die Funktionen Schritt für Schritt durch. Der zweite Teil mit dem ESP32 ist bis auf ein paar Kleinigkeiten derselbe und wir werden nur die Unterschiede behandeln.

Versionshinweis

Arduino IDE V1.8.13

Library: PubSubClient Library by Nick O'Leary V2.8.0
ArduinoOTA by Juraj Andrassy V1.0.5

Boards: ESP8266 by ESP8266 Community V2.7.4
ESP32 by Espressif Systems V1.0.4

Alle Links mit "*" sind Amazon Affiliate Links.

Source Code

Link zu Github.com/Edistechlab/

```
/*
Project:  MQTT Test (LED and Switch) with NodeMCU-ESP8266
Author:   Thomas Edlinger for www.edistechlab.com
Date:     Created 17.01.2021
Version:  V1.0
*/

#include <PubSubClient.h>
#include <ArduinoOTA.h>

#define wifi_ssid "Your_SSID"
#define wifi_password "Your_Password"
#define mqtt_server "MQTT_Server_IP"
#define mqtt_user "MQTT_username"
#define mqtt_password "MQTT_PW"
#define ESPHostname "ESP8266_MQTT_Test"
String clientId = "ESP8266-";

#define outTopic "esp8266/outtopic"
#define inTopic "esp8266/intopic"

WiFiClient espClient;
PubSubClient client(espClient);

const int LEDPin = 4;     // der Output Pin wo das LED angehängt ist
const int buttonPin = 5;  // der Input Pin wo der Taster angehängt ist
int buttonState = 0;
int lastButtonState = 0;

void setup() {
  Serial.begin(115200);
  setup_wifi();
  ArduinoOTA.setHostname(ESPHostname);
  // ArduinoOTA.setPassword("admin");
  ArduinoOTA.begin();

  client.setServer(mqtt_server, 1883);
  client.setCallback(callback);
  pinMode(LEDPin, OUTPUT);
  pinMode(buttonPin, INPUT);
}

void loop() {
  if (!client.connected()) {
    reconnect();
  }
  client.loop();
  ArduinoOTA.handle();

  buttonState = digitalRead(buttonPin);
  if (buttonState != lastButtonState) {
    if (buttonState == HIGH) {
      client.publish("esp8266/outtopic", "BTN OFF");
      Serial.print("Changing Button to OFF\n");
      client.publish("esp32/intopic", "OFF");
    } else {
      client.publish("esp8266/outtopic", "BTN ON");
      Serial.print("Changing Button to ON\n");
      client.publish("esp32/intopic", "ON");
    }
    // Kurze Pause zum entprellen
    delay(100);
  }
  lastButtonState = buttonState;
}
```

```
void setup_wifi() {
  delay(10);
  // We start by connecting to a WiFi network
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(wifi_ssid);
  WiFi.begin(wifi_ssid, wifi_password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
}

void callback(char* topic, byte* message, unsigned int length) {
  Serial.print("Message arrived on topic: ");
  Serial.print(topic);
  Serial.print(". Message: ");
  String messageTemp;

  for (int i = 0; i < length; i++) {
    Serial.print((char)message[i]);
    messageTemp += (char)message[i];
  }
  Serial.println();

  if (String(topic) == inTopic) {
    if(messageTemp == "ON"){
      Serial.print("Changing LED to ON\n");
      digitalWrite(LEDPin, HIGH);   //Invertiertes Signal
      client.publish(outTopic, "ON");
      delay(200);
    }
    else if(messageTemp == "OFF"){
      Serial.print("Changing LED to OFF\n");
      digitalWrite(LEDPin, LOW);  //Invertiertes Signal
      client.publish(outTopic, "OFF");
      delay(200);
    }
  }
}

void reconnect() {
  // Loop until we're reconnected
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    // Create a random client ID
    clientId += String(random(0xffff), HEX);
    // Attempt to connect
    if (client.connect(clientId.c_str(), mqtt_user, mqtt_password)) {
      Serial.println("connected");
      // Once connected, publish an announcement...
      client.publish(outTopic, ESPHostname);
      // ... and resubscribe
      client.subscribe(inTopic);
    } else {
        Serial.print("failed, rc=");
        Serial.print(client.state());
        Serial.println(" try again in 5 seconds");
        // Wait 5 seconds before retrying
        delay(5000);
      }
   }
}
```

MQTT Praxis Beispiel - Konzept Übersicht

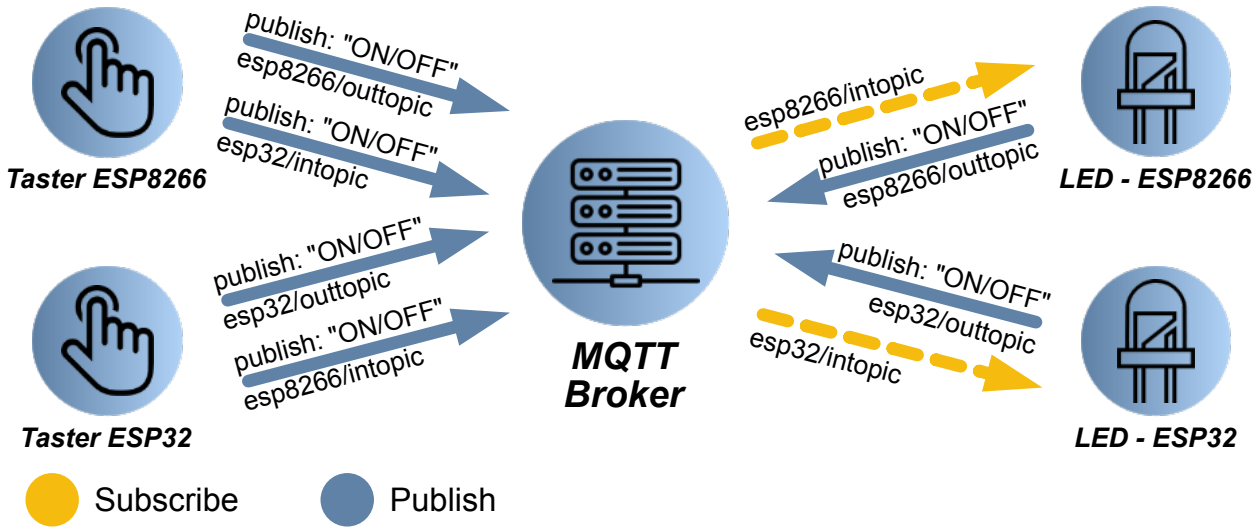


Bild 4.1.10: MQTT Test Aufbau Konzept der Topics

Source Code Erklärung – Teil 1 mit NodeMCU-ESP8266

Für dieses Beispiel benötigen wir zwei Library's. Die PubSub Library wird für die Kommunikation mittels MQTT und die Arduino OTA (Over the air) Library ist optional und sie ermöglicht es uns, zukünftige updates über das Wi-Fi Netzwerk zu machen.

Danach definieren wir unsere Wi-Fi und MQTT Zugangsdaten, gefolgt von einem MQTT Hostnamen und einer Client-ID zur Identifikation vom ESP.

Als Topic verwenden wir einmal das „esp8266/outtopic“ mit dem wir Daten vom Mikrocontroller zum MQTT Broker senden und das „esp8266/intopic“ mit dem wir Daten vom Broker empfangen können.

Auf dem GPIO Pin 5 definieren wir die LED und auf GPIO 4 den Taster. Dann benötigen wir noch zwei Variablen für den Status vom Taster, mehr dazu etwas später.

Im Setup wird die Baut-Rate zum Serial Monitor auf 115200 Baut gesetzt und die Funktion setup-wifi aufgerufen die sich mit dem Wi-Fi Netzwerk verbindet. Danach wird OTA aktiviert, wer dafür ein Passwort setzen will, kann das hier machen.
In den nächsten Zeilen wird die Verbindung zum MQTT Server aufgebaut und die LEDPin als Output und der buttonPin als Input definiert.

In der Loop Funktion wird geprüft, ob wir mit dem MQTT Server verbunden sind, wenn nicht wird mit der Funktion reconnect noch einmal versucht sich mit dem MQTT Broker zu verbinden.

Ob der Taster gedrückt ist, überprüfen wir in den nächsten Zeilen. In die Variable lastButtonState wird der aktuelle ButtonState geschrieben. Nur wenn sich der Status ändert, also von ON auf OFF oder umgekehrt, dann startet die Funktion. Wenn der Taster nicht gedrückt ist, der buttonState HIGH ist (Invertiertes Signal), dann senden wir über MQTT den Taster Status OFF auf das ESP32/intopic Topic. Wird der Taster gedrückt, dann senden wir den MQTT Status vom Taster ON. Um den Taster zu entprellen wird eine kurze Pause von 100 Millisekunden gemacht. Mit void setup_wifi() wird der ESP mit dem Wi-Fi Netz verbunden.

In der callback Funktion wird der Eingang von Nachrichten über MQTT überprüft. Wir schreiben in den Serial Monitor, dass wir eine Nachricht erhalten haben und lesen sie mit der for Schleife aus. Wenn das Payload (die Nachricht) ON ist, dann wird die LED eingeschaltet, wenn das Payload OFF ist, wird die LED ausgeschaltet.

Die void reconnect() Funktion verbindet sich mit dem MQTT-Broker und Abonniert mit subscribe(inTopic) das Topic das wir auslesen wollen.

Source Code Link zu Github.com/Edistechlab/

```
#include <PubSubClient.h>
#include <ArduinoOTA.h>
#include <WiFi.h>  // ESP32 only

#define wifi_ssid "Your_SSID"
.......
.......
#define ESPHostname "ESP32_MQTT_Test"
String clientId = "ESP32-";

#define outTopic "esp32/outtopic"
#define inTopic "esp32/intopic"

WiFiClient espClient;
PubSubClient client(espClient);

const int LEDPin = 22;    // der Output Pin wo das LED angehängt ist
const int buttonPin = 23; // der Input Pin wo der Taster angehängt ist
.......
.......
buttonState = digitalRead(buttonPin);
  if (buttonState != lastButtonState) {
    if (buttonState == HIGH) {
      client.publish("esp32/outtopic", "BTN OFF");
      Serial.print("Changing Button to OFF\n");
      client.publish("esp8266/intopic", "OFF");
    } else {
      client.publish("esp32/outtopic", "BTN ON");
      Serial.print("Changing Button to ON\n");
      client.publish("esp8266/intopic", "ON");
    }
    // Kurze Pause zum entprellen
    delay(100);
  }
  lastButtonState = buttonState;
}
```

MQTT Praxis Beispiel – Teil 2 mit NodeMCU-ESP32

Der Source Code unterscheidet sich nur durch ein paar Kleinigkeiten beim ESP32. Um mit MQTT kommunizieren zu können, benötigt der ESP32 die WiFi.h Library, diese binden wir am Beginn vom Sketch ein. Die beiden Topics ändern sich und die GPIO Pins an dem der Taster und die LED angeschlossen sind.
Wenn der Taster gedrückt wird, dann senden der ESP32 auf dem Topic esp8266/intopic den ON Befehl um damit die LED am ESP8266 einzuschalten.