jq - Ein leistungsstarkes Programm zur Auswertung von MQTT-(JSON-) Meldungen

Inhaltsverzeichnis

jq ist ein Programm oder Paket, das in der Lage ist JSON-Strings strukturiert darzustellen. Es verfügt über mächtige Filter und eine Vielzahl an Funktionen, die einzelne Werte aus umfangreichen JSON-Strams sichtbar oder als Variablenwerte weiterverarbeitbar machen. Der Beitrag zeigt die Nutzung insbesondere der Filter und gibt ein Beispiel für die Einbindung in (bash-) Skripte.

MQTT-Meldungen mit jq auswerten

I - Einleitung

Alle Informationen, die ein Tasmota-geflashter Sonoff- (oder alternatives) Gerät ausgibt, sind im JSON-Format aufbereiten. JSON bedeutet JavaScript Object Notation und dient dem Datenaustausch zwischen Geräten. Die Struktur ist dabei recht simpel und dennoch flexibel. Den Kern stellen Objekte dar. Objekte sind an den geschweiften Klammern zu erkennen `{....}`. Diese können Schlüssel-Werte-Paare enthalten `{"POWER":"ON"}`. Objekte können weitere Objekte enthalten `{"StatusNET":{...}}` und/oder einen Array
`{"StatusMEM:{...,"Features":["00000407","0FDAE794","...","..."]}"}`.
Arrays sind Listen von Werten, man beachte die eckigen Klammern als Kennzeichen von Arrays. Innerhalb von Objekten und Arrays sind Elemente, die auf gleicher Ebene stehen, durch Kommata abgetrennt. Schlüssel-Werte Paare sind an der Struktur "Schlüssel":"Wert" zu erkennen, wobei Wert eben auch ein weiteres Objekt sein kann, das einen Array enthält. Das sieht schon manchmal unübersichtlich aus. Doch genau dabei ist jq eine große Hilfe, weil die Darstellung eines JSON-Ausdrucks strukturiert erfolgt und durch Filterung die Anzeige leicht auf (nur) den Wert reduziert werden kann, den man haben will.

In diesem Beitrag soll es um die Auswertung von MQTT-Meldungen gehen, um sie in eignen Skripten weiterzu verarbeiten. Der Autor geht davon aus, daß der Umgang mit MQTT grundsätzlich vertraut ist. Im Artikel MQTT - Eine Einführung sind erste Schritte beschrieben, wie man mit einem MQTTBroker bzw. mit MQTT mit den Geräten kommuniziert.

Bei den hier verwendeten Beispielen sind zwei Terminalfenster bzw. zwei SSH-Verbindungen aufgebaut. In einem Fenster läuft mosquitto_sub als Subscriber, als Empfänger von Nachrichten. In dem anderen Fenster werden mit mosquitto_pub MQTT-Kommandos abgesetzt.

II - Der JSON-Aufbereiter jq

Eine vollständige Dokumentation von jq findet sich hier bzw. in dem Handbuch. Leider gibt es das nur auf Englisch. Das Paket bzw. Programm gibt es native für Linux (und weitere UNIXe). Für Windows und OS X läuft die Installation über spezielle Installer bzw. Umgebungen Chocolate (Windows) oder Homebrew (OS X). Die nachfolgende Darstellung bezieht sich ausschließlich auf die Version für Linux in den beiden Distributionen Ubuntu 18.04 bzw Raspbian Stretch.

jq ist ein Programm, mit dem man eine JSON-String, strukturiert darstellen kan bzw. durch die Nutzung von Filtern genau auf den Wert oder den String reduziert, der für eine Weiterverarbeitung genutzt wird. Die Weiterverarbeitung kann eine Anzeige sein oder ein Wert z. B. ein Sensorwert, der einer Variablen zugewiesen wird.

III - Erste Schritte

III.I - Setting

Ich gehe in den nun folgenden Beispielen von folgender Konstellation aus:

- In einem lokalen Netz läuft ein MQTTBroker
- Die beiden MQTTClients mosquitto_sub und mosquitto_pub sind installiert
- Auf dem Rechner, auf dem mosquitto_sub läuft, ist auch das Paket jq installiert.
- Es werden zwei Terminalfenster geöffnet (oder zwei SSH-Verbindungen). In einem Fenster läuft mosquitto_sub, in dem anderen Fenster werden mit mosquitto_pub MQTT-Statements abgegeben.

III.II - Installation

```
sudo apt install jq
```

Anschließend steht das Programm zur Verfügung. üblicherweise arbeitet das Programm mit Dateien und wird so verwendet:

```
jq [Optionen] . < JSON-Datei.in >JSON-Datei.out
```

Es ist aber auf Dauer lästig erst Dateien zu schreiben und dann diese Dateien wieder einzulesen. Zum Glück gibt es das Pipen. Damit kann man eine Ausgabe, die normalerweise auf einem Bildschirm erscheint, weiterleiten an ein anderes Programm, das diesen Output zunächst weiterverarbeitet und erst dann anzeigt (oder an ein weiteres Programm leitet).

Aber eins nach dem anderen. Schauen wir uns zunächst einmal an, was ein Tasmota-Gerät zurückmeldet, wenn es mit einer anfrage angesprochen wir. In unserem Beispiel hat der MQTTBroker die IP 192.168.178.20 und das Gerät, das angesprochen wird, ist ein TH10 mit Temperatursensor. In einem Fenster abonnieren (subsribe) wir alle (#) Status-Meldungen (stat/) des TH10 (TH10/:

```
mosquitto_sub -h 192.168.178.20 -t stat/TH10/#
```

In dem anderen Fenster publizieren wir eine MQTT-Nachricht an den TH10 und zwar die Ausgabe der Kurzübersicht:

```
mosquitto_pub -h 192.168.178.20 -t cmnd/TH10/status -m ''"
```

Anmerkung: Wenn eine leere Message (Payload) übertragen wird, setzt man hinter den Schalter -m eben eine leere Nachricht mit '' (zwei einzelne Hochkommata).

Das Ergebnis sieht dann so aus:

Code

```
{"Status":{"Module":4,"FriendlyName":["TH10"],"Topic":"TH10","ButtonTopic":"0","Power":0,"Po
```

Nicht sehr übersichtlich, oder?

Nun beenden wir zunächst im mosquitto_sub-Fenster die laufende Anwendung mit `STRG-c` und geben folgendes ein:
`mosquitto_sub -h 192.168.178.20 -t stat/TH10/# | jq .`

Anmerkung: Das Pipe-Symbol | wird mit der Tastenkombination `AltGr-<` erzeugt. Leerstelle und Punkt gehören mit zur Syntax des jq-Befehls.

Jetzt wiederholen wir im mosquitto_pub-Fenster die Eingabe:
`mosquitto_pub -h 192.168.178.20 -t cmnd/TH10/status -m ''"`

Nun sieht die Ausgabe so aus:

Code

```
{


      "TH10"
    ],









  }
}
```

Alles anzeigen

Das ist doch gleich viel übersichtlicher! Aber wird noch besser! Für die folgenden drei Beispiele bleibt der Befehl im mosquitto_pub Fenster immer gleich. Wer es noch nicht weiß, im Terminal kann man die letzten Befehle wiederholen, indem man die Cursortaste "nach oben" benutzt. Das ist sehr praktisch um Befehle während einer Testphase mit wenig Aufwand zu wiederholen.

Im mosquitto_sub-Fenster benutzen wir im Folgenden ein paar Variationen unseres "Abhörbefehls", weil wir jq mit sich ändernden Filtern einsetzen. Also zunächst einmal mosquitto_sub wieder mit `Strg-c` beenden. Cursortaste "nach oben" betätigen. Es erscheint wieder der zuletzt eingegebene Befehl, der aber ergänzt wird:
`mosquitto_sub -h 192.168.178.20 -t stat/TH10/# | jq .Status`

Im mosquitto_sub-Fenster wird nun nur noch der Inhalt des Objektes "Status" wiedergegeben:

Code

```
{

    "TH10"
  ],









}
```

Alles anzeigen

Noch deutlicher wird es nach folgender Änderung:

```
mosquitto_sub -h 192.168.178.20 -t stat/TH10/# | jq .Status.FriendlyName
```

Als Ergebnis haben wir nur noch der Inhalt des Objektes FriendlyName:

Code

```
[
  "TH10"
]
```

III.II.I - Nun sehen wir, dass der Inhalt des Objektes FriendlyName in Wirklichkeit ein Array ist, erkennbar an [...] mit nur einem Eintrag "TH10". Um nur den Inhalt des Arrays zu sehen, genauer den ersten Eintrag des Arrays, geben wir folgende Filterung ein:mosquitto_sub -h 192.168.178.20 -t stat/TH10/# | jq .Status.FriendlyName[0]Nun bekommen wir als Ergebnis nur noch den String des FriendlyName "TH10" geliefert.

III.II.II - Filtern mit jq

Was jq anzeigt, steuern wir mit der Filterung. Diese Filter sind sehr mächtig. Nachlesen kann man das im jq-Manual, was auch mit guten Beispielen aufwartet. Hier nun ein kurzer Überblick über die in den Beispielen verwendeten Filter:

- Das Zeichen "." `jq .` entspricht der Wurzel des JSON-Ausdrucks, bedeutet also soviel wie "Zeige alles von Anfang an"
- `.Status` bedeutet: Zeige alles von der Wurzel an im Zweig Status
- `.Status.FriendlyName` bedeutet Zeige alles vom Anfang an im Zweig "Status" und den darin enthaltenem Schlüssel (könnte auch ein weiteres Objekt sein) "FriendlyName". Weil FriendlyName einen Array (Python: Liste) beinhaltet, erkennbar an `[ ... ]` greifen wir in nächsten Schritt auf einen Eintrag des Array zu. Ein Array beginnt immer mit dem Eintrag 0:
- `.Status.FriendlyName[0]`

III.II.III - Ausgabeformat

Neben der Filterung, also der Reduzierung der Ausgabe auf genau das, was man sehen möchte, kann man das Format der Ausgabe beeinflussen. Dazu benutzen beenden wir zunächst den mosquitto_sub-Client mit `Strg-c` und wiederholen den letzten Befehl ergänten zusätzlich hinter dem jq den Schalter `-r`:

```
mosquitto_sub -h 192.168.178.20 -t stat/TH10/# | jq -r .Status.FriendlyName[0]
```

Das Ergebnis ist ein "TH10" allerdings ohne die " ". So hat man das nackte Ergebnis, ein Raw-Ergebnis.

Ein weiteres Beispiel: Dazu setzen wir einen neuen Filter und zwar einen, der auf die Filterung der Sensordaten, genauer der Temperatur-Daten abgestimmt ist. Dazu im mosquitto_sub-Fenster die Anwendung mit `Strg-c` beenden und folgenden Filter setzen (wenn ein Temperatur-Feuchtigkeitssensor vom Typ AM2301 angeschlossen ist, sonst den Namens des verwendeten Sensors eingeben):

```
mosquitto_sub -h 192.168.178.20 -t stat/# | jq .StatusSNS.AM2301.Temperature
```

Damit werden nur noch die Daten des Objektes "Temperature" im Objekt "AM2301" im Objekt "StatusSNS" angezeigt.

Im mosquitto_pub-Fenster müssen wir allerdings einen anderen Status abrufen, nämlich den Status, der die Sensordaten enthalten:

```
mosquitto_pub -h 192.168.178.20 -t cmnd/TH10/Status -m '8'</p>
```

Als Ergebnis wird der aktuelle Temperaturwert angezeigt.

III.II.IV - jq-Auswertungen in Skripten

Unten steht ein kleines Beispiel wie ein mit jq gefilterter Wert in einer Variablen a landet. Damit man schnell was sieht, sollte man die Ausgabe von Sensorwerten bei dem Gerät beschleunigen durch die Eingabe von `teleperiod 10` auf der Tasmota-Konsole. Die IP des MQTTBrokers und der MQTT-Name des Gerätes muss natürlich entsprechend den eigenen Verhältnissen angepasst werden. Anders als im Beispiel zuvor werden nun die Sensordaten nicht aus dem abgefragten Status 8 ausgewertet, sondern aus dem tele-Topic SENSOR. Dies ist eine alternative Möglichkeit Sensordaten zu gewinnen.

Bash

```
#!/bin/bash
#MQTT-SERVER;
MQHOST=192.168.178.20
DEVICE="TH10"
#Start_Client#
mosquitto_sub -h $MQHOST -t tele/$DEVICE/SENSOR | while read RAW_DATA
do
a=$(echo $RAW_DATA | jq -r .AM2301.Temperature);
echo $a;
done
```

Den Inhalt der Variablen a ($a) kann man natürlich nicht nur mit [tt]echo[/tt zur Anzeige bringen, sondern der Wert kann in eine Datei geschrieben werden, man kann über einen bestimmten Zeitraum Werte sammeln und Durchschnitte bilden, Trends errechnen, Arrays füllen usw. Wenn man das noch in if...then- oder for x in y-Schleifen einbaut, ist vieles möglich. Insofern ist ein Skript wesentlich flexibler als die Rules, weil ein Skript auf andere Faktoren als "nur" Ereignisse (events) reagieren kann.

Viel Spaß beim Weiterentwickeln!