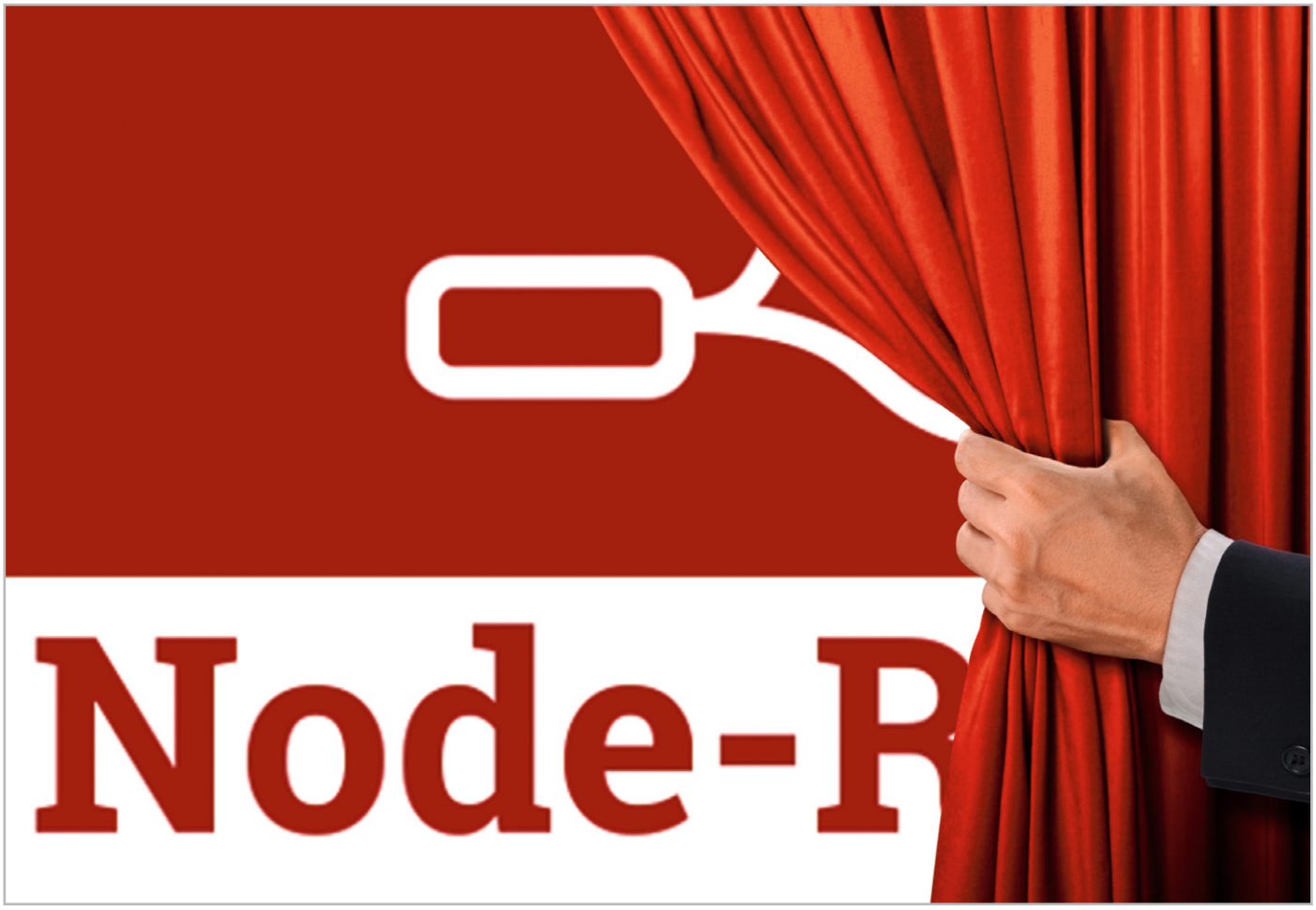
Programmieren (fast) ohne Code

Node-RED: Ein Blick unter die Haube und praktische Anwendungen für Automatisierungen

Teil 3

In den ersten beiden Beiträgen zu Node-RED haben wir uns mit der Installation und den Grundlagen des praktischen Prototyping-Tools vertraut gemacht sowie Möglichkeiten angeschaut, elektronische Bauteile in Node-RED zu integrieren. Im letzten Teil werfen wir einen Blick unter die Haube und schauen uns Beispiele für praktische Anwendungen und Automatisierungen an.



Grundlagen

Wie im ersten Beitrag [1] schon erläutert, ist die Basis von Node-RED Node.js – eine serverseitige, ereignisgesteuerte Plattform zum Betrieb von Netzwerkanwendungen mit einer leichtgewichtigen und effizienten Struktur sowie nicht blockierenden In- und Outputs. Das bedeutet, dass langsame Operationen wie Zugriffe auf das Netzwerk oder Dateisystem andere, schnelle Anweisungen für den Prozessor nicht blockieren. Doch was heißt das nun für die Praxis?

Schauen wir uns dazu noch einmal unseren in JavaScript programmierten einfachen Webserver an:

servertest.js

```
var http = require('http');                          ❶

http.createServer ❷ (function (req, res) {
  res.writeHead(200, {
    'Content-Type': 'text/html'
  });
  res.write('Hallo Welt');
  res.end();

}).listen(3000);                                     ❸
```

❹

Rufen wir das Script per

```
node servertest.js
```

auf, bindet dies zunächst die Bibliothek http ein, die entsprechende Funktionalitäten für das gleichnamige Übertragungsprotokoll bereitstellt ❶.

Dazu gehört zum Beispiel die Methode

```
http.createServer(requestListener);
```
❷

die ein HTTP-Server-Objekt initiiert, das auf dem ausführenden System einen HTTP-Server bereitstellt. Der HTTP-Server kann nun auf einem Port (in unserem Beispiel ist dies der Port 3000) auf Anfragen warten ❸ und auf diese Anfragen mit einem requestListener ❹ antworten.

Der requestListener ist optional und in unserem Fall ist das die Funktion (function) als Argument für die Methode mit den Parametern req und res, die sowohl die Anfrage des Browsers positiv (Status-Code 200 für „OK") beantwortet, als auch die entsprechende Antwort – im vorliegenden Beispiel die Textausgabe (Hallo Welt) – im Browser zurückgibt.

Diese ereignisbasierte Funktionalität (es kommt eine Anfrage von einem Client) ermöglicht das oben erwähnte, nicht blockierende Verhalten, da nur bei tatsächlichen Anfragen das Host-System belastet bzw. der Prozess ausgeführt wird und ansonsten für andere Zwecke zur Verfügung steht.

Die Kommunikation, die im Hintergrund abläuft, kann man sich im Browser anschauen, was sich beispielsweise auch für die Fehlersuche empfiehlt. Dazu kann man in Firefox die Taste F12 drücken (alternativ die Webentwickler-Tools im Hamburger-Menü auswählen). Im Bereich Netzwerkanalyse (Bild 1) sieht man dann den Status-Code 200 1, den man dem Browser übergeben hat, den Content-Type 2, den Text 3 und den Port 4.

Als Content-Type hätte man übrigens auch text/plain wählen können, da wir keine HTML-Tags übergeben.

In Chrome findet man diese Funktion im Menü unter Weitere Tools → Entwicklertools (Bild 2).

Unterbau

Da wir jetzt schon in die Analyse eingetaucht sind, werfen wir noch einen Blick auf Node-RED und den Unterbau, der die ganzen Funktionalitäten ermöglicht.

Unter Linux finden wir – wenn wir es lokal und nicht global installiert haben – Node-RED im Nutzerverzeichnis

```
/home/pi/.node-red
```

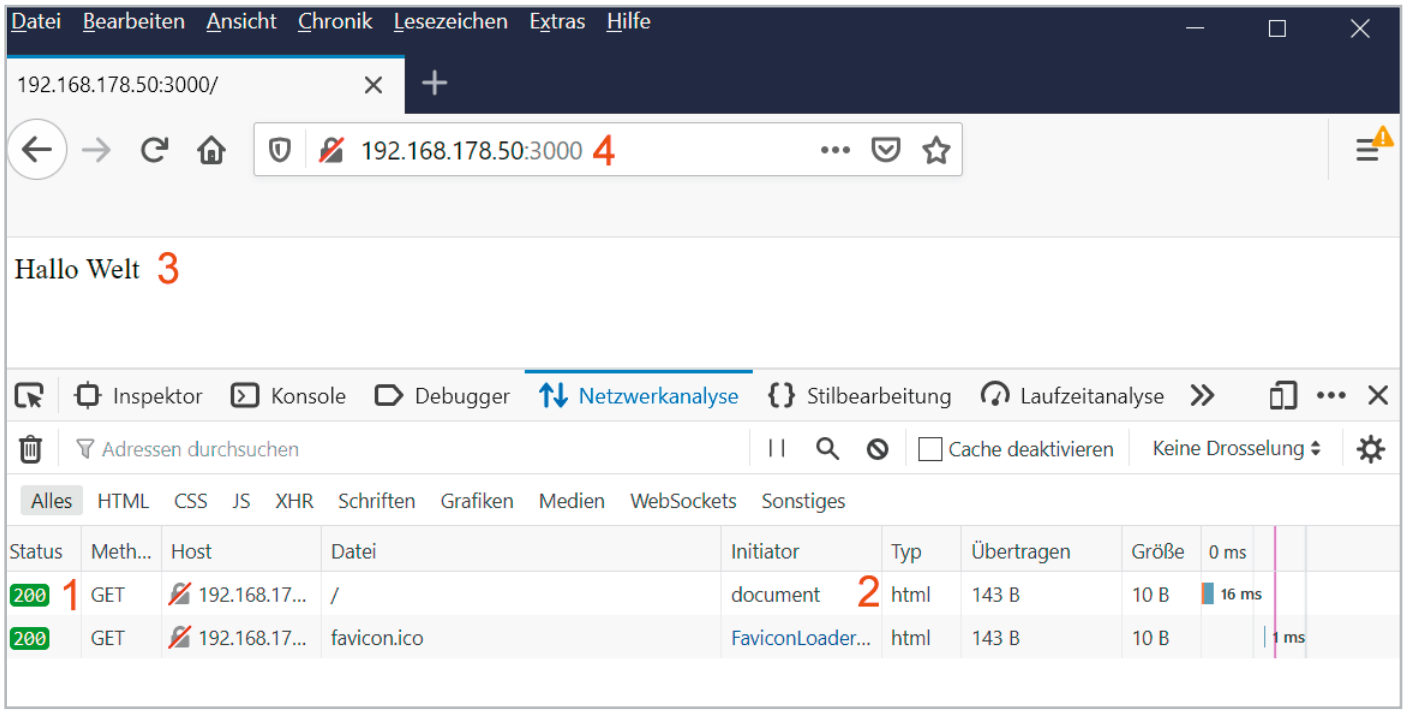


Bild 1: Netzwerkanalyse im Browser Firefox

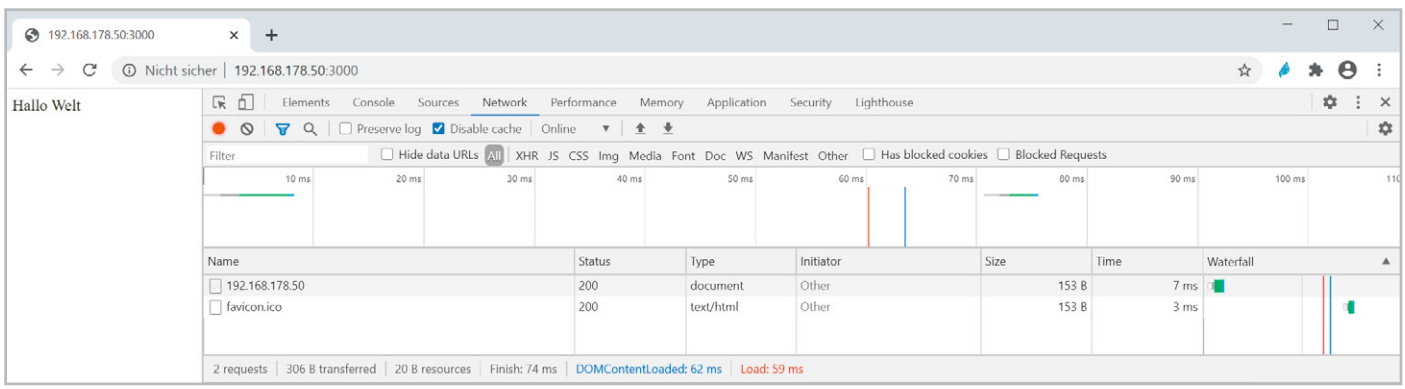


Bild 2: Netzwerkanalyse im Browser Chrome

Tipp: In das Home-Verzeichnis kommt man sehr einfach mit dem Befehl cd ~ (Tilde).

Um auch Verzeichnisse mit einem führenden Punkt anzeigen zu lassen, muss man den Befehl ls mit der Option -a wählen.

Will man wissen, in welchem Verzeichnis man sich aktuell befindet, wählt man das Kommando pwd.

Lässt man sich die Dateistruktur anzeigen, erhält man folgende Ausgabe (Bild 3).

In .config.json befindet sich die Auflistung aller installierten Module mit den Nodes. Lässt man sich die Datei mit

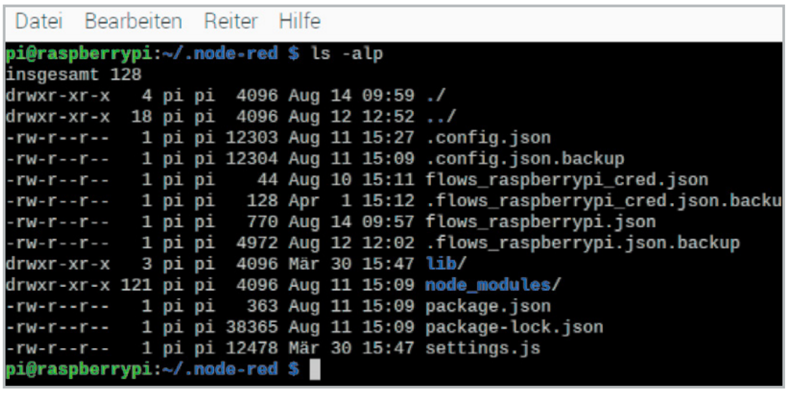
cat .config.json

anzeigen, findet man so zum Beispiel auch die im vorhergehenden Beitrag installierten Pakete mit all ihren Detailinformationen.

Die Datei flows_raspberrypi.json beinhaltet alle zurzeit installierten Flows im JSON-Format.

Die Einstellungen lassen sich in settings.js anpassen. Hier kann man unter anderem den Port für Node-RED verändern, den Editor und die Admin API per User/Passwort schützen oder den Debug-Level anpassen.

Alle im Stammverzeichnis enthaltenen Dateien sollte man zumindest in einem produktiven System häufig per Back-up auf einem anderen System sichern. So kann man sehr schnell die Node-RED-Umgebung nach einem Systemfehler (Beispiel: defekte SD-Card) wiederherstellen.

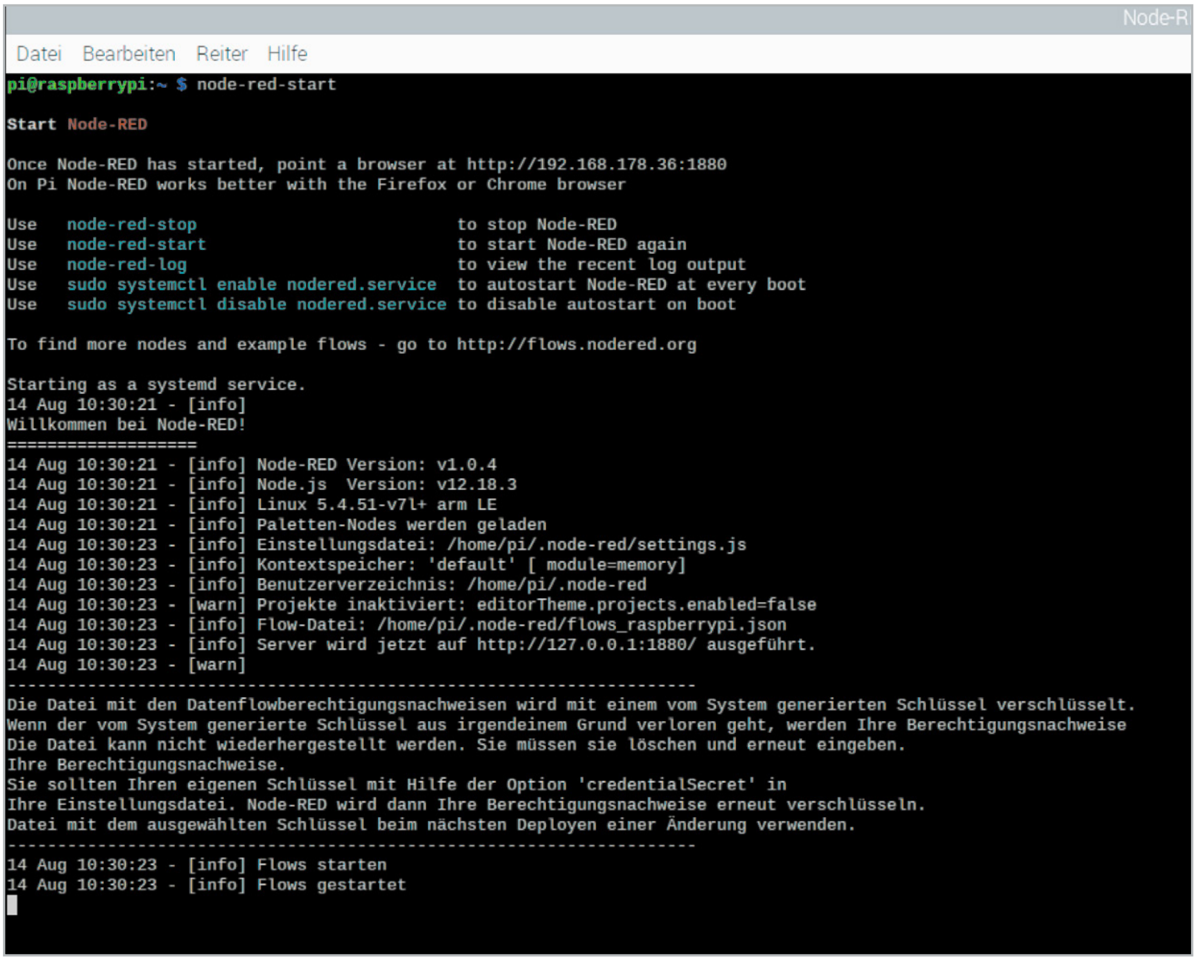


```
Datei  Bearbeiten  Reiter  Hilfe
pi@raspberrypi:~/.node-red $ ls -alp
insgesamt 128
drwxr-xr-x   4 pi pi  4096 Aug 14 09:59 ./
drwxr-xr-x  18 pi pi  4096 Aug 12 12:52 ../
-rw-r--r--   1 pi pi 12303 Aug 11 15:27 .config.json
-rw-r--r--   1 pi pi 12304 Aug 11 15:09 .config.json.backup
-rw-r--r--   1 pi pi    44 Aug 10 15:11 flows_raspberrypi_cred.json
-rw-r--r--   1 pi pi   128 Apr  1 15:12 .flows_raspberrypi_cred.json.backu
-rw-r--r--   1 pi pi   770 Aug 14 09:57 flows_raspberrypi.json
-rw-r--r--   1 pi pi  4972 Aug 12 12:02 .flows_raspberrypi.json.backup
drwxr-xr-x   3 pi pi  4096 Mär 30 15:47 lib/
drwxr-xr-x 121 pi pi  4096 Aug 11 15:09 node_modules/
-rw-r--r--   1 pi pi   363 Aug 11 15:09 package.json
-rw-r--r--   1 pi pi 38365 Aug 11 15:09 package-lock.json
-rw-r--r--   1 pi pi 12478 Mär 30 15:47 settings.js
pi@raspberrypi:~/.node-red $
```

Bild 3: Node-RED Dateistruktur

```
pi@raspberrypi:~/.node-red/node_modules $ ls
ajv               bl                       delayed-stream              firmata-io           ieee754                    mime-db                        node-red-node-arduino
ansi-regex        buffer                   delegates                   forever-agent        inherits                   mime-types                     node-red-node-openweathermap
aproba            caseless                 detect-libc                 form-data            ini                        mimic-response                 node-red-node-pi-gpio
are-we-there-yet  chownr                   ecc-jsbn                    fs-constants         isarray                    minimist                       node-red-node-rbe
asn1              code-point-at            end-of-stream               gauge                is-fullwidth-code-point    mkdirp                         noop-logger
assert-plus       combined-stream          expand-template             getpass              isstream                   mkdirp-classic                 npmlog
asynckit          console-control-strings  extend                      github-from-package  is-typedarray              ms                             number-is-nan
aws4              core-util-is             extsprintf                  har-schema           jsbn                       nan                            oauth-sign
aws-sign2         dashdash                 fast-deep-equal             har-validator        json-schema                napi-build-utils               object-assign
base64-js         debug                    fast-json-stable-stringify  has-unicode          json-schema-traverse       node-abi                       once
bcrypt-pbkdf      decompress-response      file-uri-to-path            http-signature       json-stringify-safe        node-red-contrib-buffer-parser performance-now
bindings          deep-extend              firmata                     i2c-bus              jsprim                     node-red-contrib-i2c           prebuild-install
pi@raspberrypi:~/.node-red/node_modules $
```

Bild 4: Installierte Module



```
Node-R
Datei  Bearbeiten  Reiter  Hilfe
pi@raspberrypi:~ $ node-red-start

Start Node-RED

Once Node-RED has started, point a browser at http://192.168.178.36:1880
On Pi Node-RED works better with the Firefox or Chrome browser

Use   node-red-stop                          to stop Node-RED
Use   node-red-start                         to start Node-RED again
Use   node-red-log                           to view the recent log output
Use   sudo systemctl enable nodered.service  to autostart Node-RED at every boot
Use   sudo systemctl disable nodered.service to disable autostart on boot

To find more nodes and example flows - go to http://flows.nodered.org

Starting as a systemd service.
14 Aug 10:30:21 - [info]
Willkommen bei Node-RED!
===================
14 Aug 10:30:21 - [info] Node-RED Version: v1.0.4
14 Aug 10:30:21 - [info] Node.js  Version: v12.18.3
14 Aug 10:30:21 - [info] Linux 5.4.51-v7l+ arm LE
14 Aug 10:30:21 - [info] Paletten-Nodes werden geladen
14 Aug 10:30:23 - [info] Einstellungsdatei: /home/pi/.node-red/settings.js
14 Aug 10:30:23 - [info] Kontextspeicher: 'default' [ module=memory]
14 Aug 10:30:23 - [info] Benutzerverzeichnis: /home/pi/.node-red
14 Aug 10:30:23 - [warn] Projekte inaktiviert: editorTheme.projects.enabled=false
14 Aug 10:30:23 - [info] Flow-Datei: /home/pi/.node-red/flows_raspberrypi.json
14 Aug 10:30:23 - [info] Server wird jetzt auf http://127.0.0.1:1880/ ausgeführt.
14 Aug 10:30:23 - [warn]
---------------------------------------------------------------------
Die Datei mit den Datenflowberechtigungsnachweisen wird mit einem vom System generierten Schlüssel verschlüsselt.
Wenn der vom System generierte Schlüssel aus irgendeinem Grund verloren geht, werden Ihre Berechtigungsnachweise
Die Datei kann nicht wiederhergestellt werden. Sie müssen sie löschen und erneut eingeben.
Ihre Berechtigungsnachweise.
Sie sollten Ihren eigenen Schlüssel mit Hilfe der Option 'credentialSecret' in
Ihre Einstellungsdatei. Node-RED wird dann Ihre Berechtigungsnachweise erneut verschlüsseln.
Datei mit dem ausgewählten Schlüssel beim nächsten Deployen einer Änderung verwenden.
---------------------------------------------------------------------
14 Aug 10:30:23 - [info] Flows starten
14 Aug 10:30:23 - [info] Flows gestartet
```

Bild 5: Node-RED Logfile

Geht man noch eine Ebene tiefer in das Verzeichnis node_modules, sieht man hier alle installierten Module und auch die in unseren Beiträgen hinzugefügten (Bild 4).

Will man später einmal eigene Module bzw. Nodes schreiben, lohnt sich ein Blick in die einzelnen Verzeichnisse und deren Inhalte. Hier sind alle Bestandteile enthalten, die später für Information und Funktionalität unter Node-RED notwendig sind.

Debugging

Will man Node-RED nach jedem Boot-Vorgang des Raspberry Pi automatisch starten, kann man dies mit

sudo systemctl enable nodered.service

einstellen.

Diese Information findet man unter anderem im Logfile von Node-RED, das beim manuellen Start auf der Kommandozeile automatisch angezeigt wird (Bild 5) und ansonsten mit node-red-log aufgerufen werden kann.

In der Datei settings.js kann man den Debug-Level einstellen und für vertiefendes Debugging noch deutlich erhöhen. Ohnehin lohnt sich ein Blick ins Logfile, da so unter Umständen auch die Ursachen für Fehler einfach erkannt werden können. In Bild 6 sehen wir einen Fehler, der durch einen nicht vorhandenen seriellen Port verursacht wird und zu einem dauernden Neustart von Node-RED führt.

Die Ausgabe vom Debug-Node kann man zusätzlich zu dem Debug-Fenster im Node-RED-Editor auch auf die Konsole (zum Beispiel zur Fehleranalyse) umleiten. Dazu aktiviert man die Einstellung Ziel → Systemkonsole im Konfigurationsfenster des Debug-Node (Bild 7). Weitere Informationen erhält man dazu in der Node-Information. Hier lohnt sich auch ein Blick auf die Möglichkeiten zur Ausgabe des msg-Objektes.

Eine weitere Möglichkeit zur Fehlerbehandlung bietet der Catch-Node, der Fehlermeldungen aller oder bestimmter Nodes in dem ausgewählten Flow abfängt (Bild 8). Diese kann man dann an einen Debug-Node zur Ausgabe im Debug-Fenster oder zur Systemkonsole weiterleiten bzw. zur Weiterverarbeitung des Fehlers an einen separaten Flow senden.

Bis jetzt haben wir immer von Node-RED unter Linux gesprochen. Das Prototyping-Tool lässt sich aber auch auf zahlreichen anderen Plattformen installieren. Dazu lohnt sich ein Blick in die umfangreiche und sehr ausführliche Dokumentation des Projektes unter [2].

Selbst auf dem (Android-)Smartphone lässt sich Node-RED mit der Termux-App installieren und ausführen [3]. Durch die zahlreichen Sensoren und Informationen, die man aus dem Handy gewinnen kann, lässt sich allein hiermit schon eine umfangreiche Datenmenge zur Weiterverarbeitung nutzen.

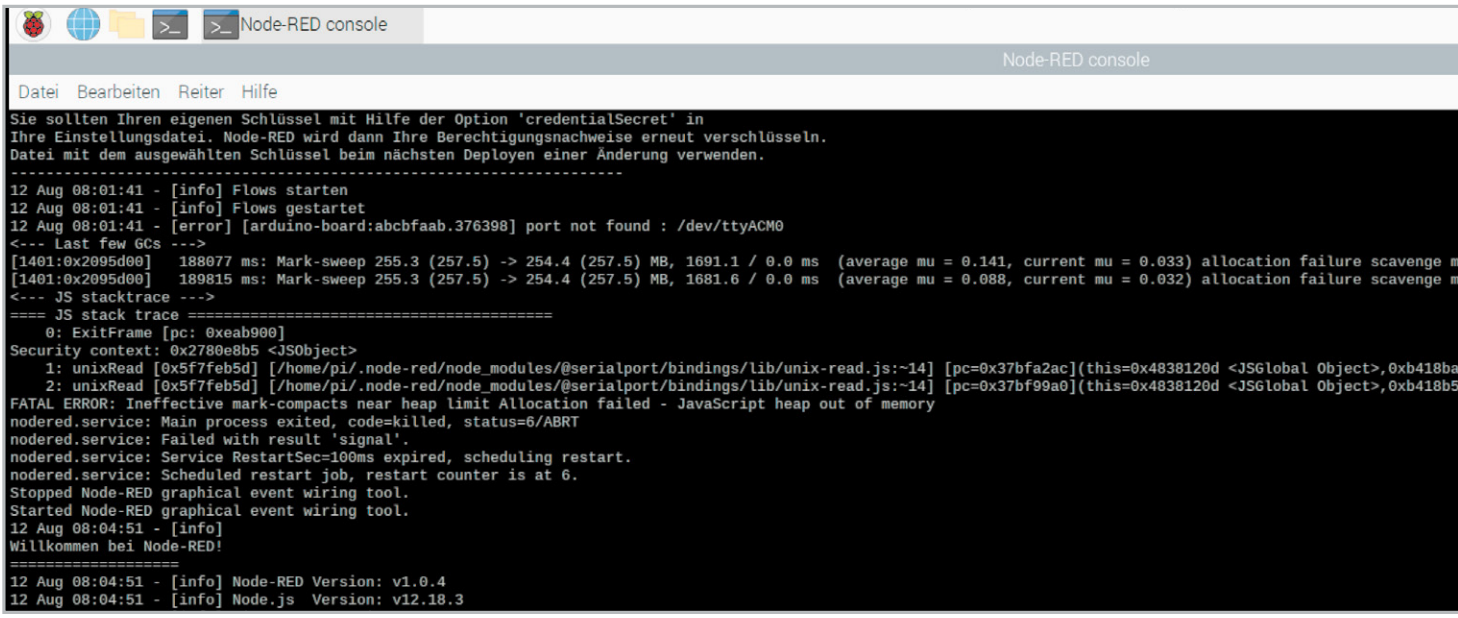


Bild 6: Fehleranalyse mithilfe des Logfiles

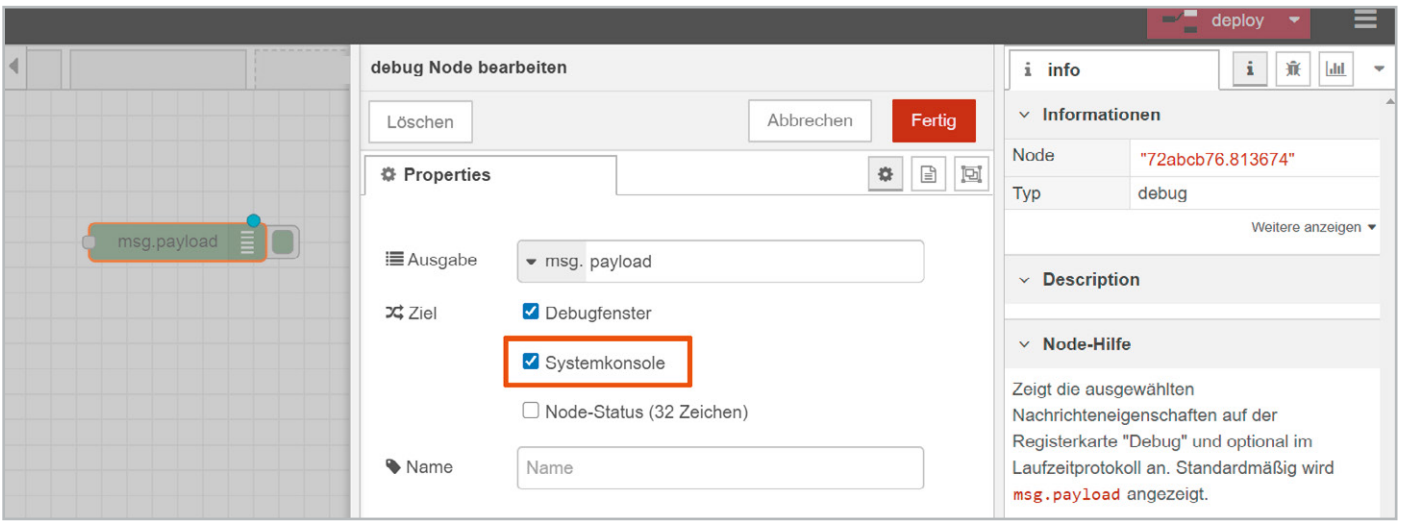


Bild 7: Verschiedene Optionen zur Ausgabe der Debug-Nachrichten

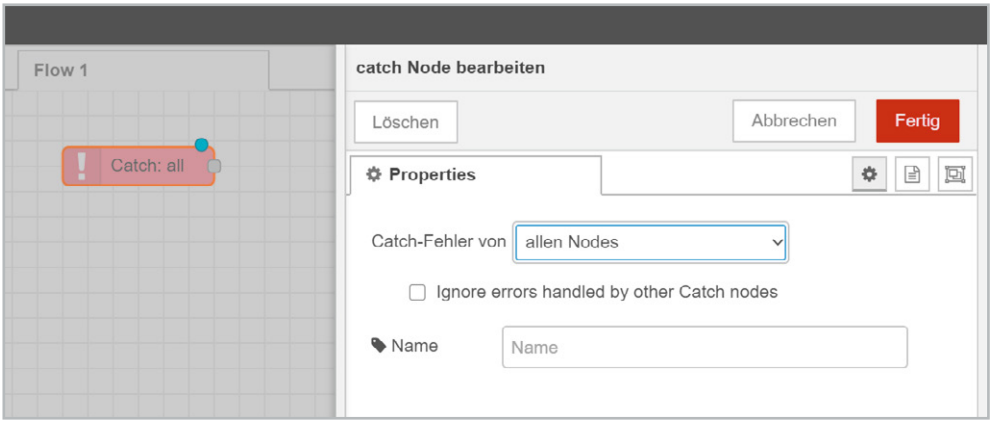


Bild 8: Catch-Node zur Fehlerbehandlung

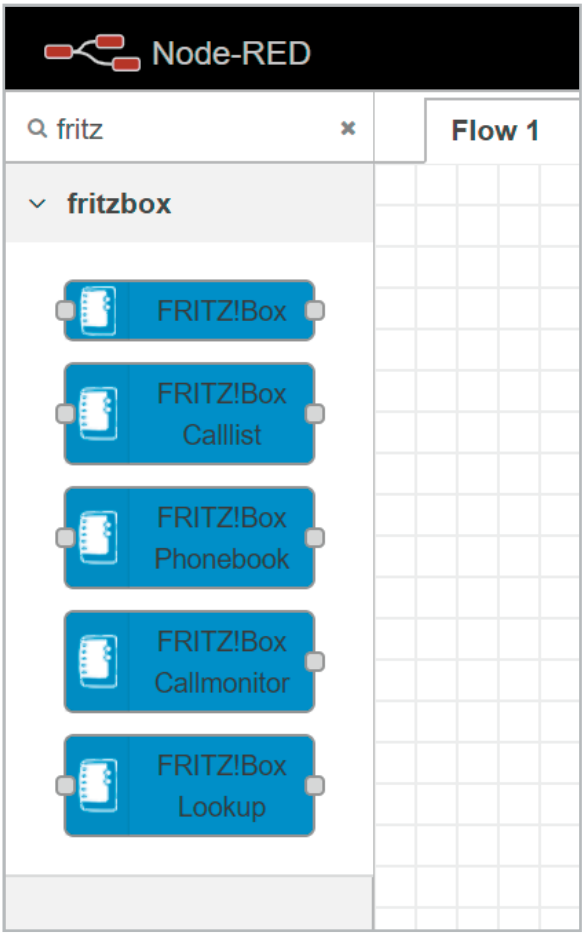


Bild 9: Verfügbare FRITZ!Box-Nodes

Praktische Anwendungen

Nach der Theorie jetzt noch ein paar praktische Anwendungen, die die bei vielen von uns vorhandene Hardware wie die FRITZ!Box verwenden oder auf Dienste wie Twitter zugreifen können. In diesem Zusammenhang werden wir uns auch ein paar Funktionalitäten unter Node-RED anschauen, die zum Teil aber mit JavaScript-Code realisiert werden müssen. Zuletzt werfen wir noch einen Blick auf das schlanke und populäre Übertragungsprotokoll MQTT, das unsere bisher verwendeten seriellen (USB/seriell) und Web-Protokolle (http) auf der Maschine-Maschine-Ebene ergänzt.

Node-RED ruft FRITZ!Box

Die FRITZ!Box von AVM gehört sicherlich zu den am häufigsten verwendeten Heim-Routern und steht in vielen Haushalten. Doch das Gerät ebnet nicht nur den Weg ins Internet, verteilt das WLAN im Haus und ermöglicht Telefonie, sondern kann auch mit den zahlreichen darauf vorliegenden Daten von Node-RED genutzt werden.

Grundlage dafür bildet das in der FRITZ!Box eingesetzte Protokoll TR-064, ein vom DSL-Forum entwickeltes Protokoll, um DSL-Internetzugangsgeräte aus dem lokalen Netz zu konfigurieren. Auf der Webseite von AVM [4] gibt es weitere Informationen zu TR-064 in Form von zahlreichen PDF-Dokumenten zu den verschiedenen auf der FRITZ!Box verfügbaren Diensten und auch den sonst in der FRITZ!Box eingesetzten Protokollen bzw. Schnittstellen.

Wir können dieses Protokoll für Node-RED nutzen und starten ohne viel Code und Theorie, da es mit node-red-contrib-fritz ein maßgeschneidertes Modul mit sechs Nodes (einer davon ist der FRITZ!Box Konfigurations-Node) gibt. Wir installieren das Paket über die Palettenverwaltung und finden fünf neue Nodes im Editor in der linken Node-Übersicht wieder (Bild 9).

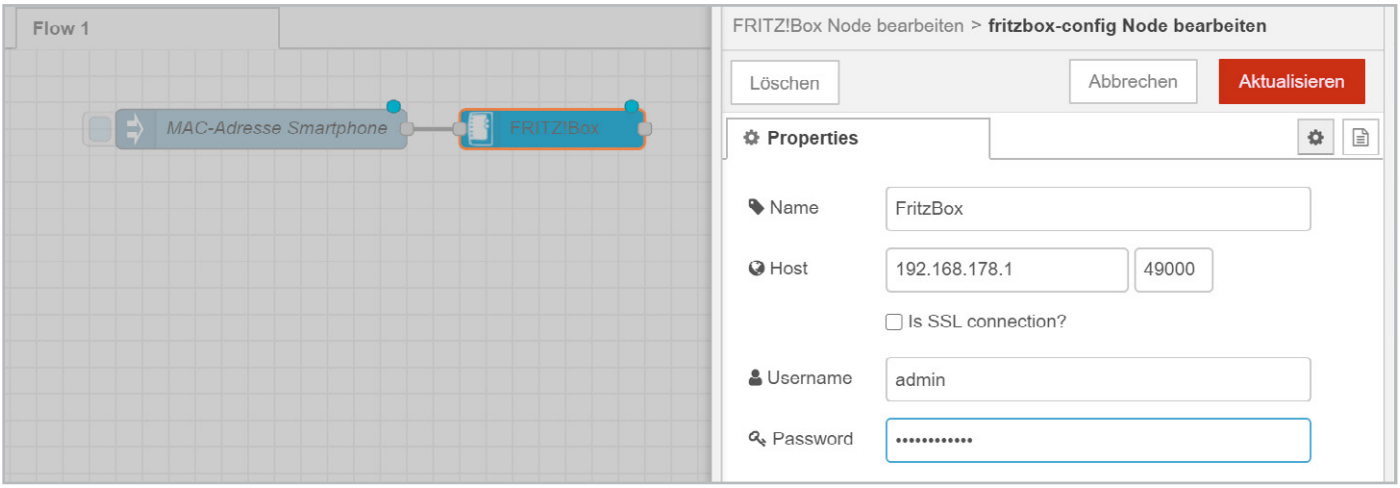


Bild 10: Konfiguration des Zugangs zur FRITZ!Box

Tipp: Mit der Zeit wächst die Anzahl der Nodes stetig an und man muss unter Umständen etwas scrollen und suchen, um die richtigen Nodes zu finden. Mithilfe des Suchfelds (Filter Nodes) im oberen linken Bereich findet man die Nodes schneller.

Da oder nicht da

Fast jeder von uns hat heute sein Smartphone dabei und ist damit im heimischen WLAN eingeloggt. Daher eignet sich diese Tatsache auch wunderbar zur Präsenzerkennung – natürlich aus Datenschutzgründen und zur Wahrung der Privatsphäre nur für das eigene Handy des Node-RED-Entwicklers –, um damit Aktionen auslösen zu können. Mögliche Szenarien sind das An- oder Ausschalten von Licht und Geräten beim Betreten oder Verlassen der Wohnung oder das Aktivieren von Kameras bei Abwesenheit.

Da es sich bei Node-RED um ein Prototyping-Tool handelt, das eigentlich nicht für den produktiven Einsatz vorgesehen ist, geschehen alle im Beitrag vorgestellten Projekte und Automatisierungen auf eigene Gefahr des Nutzers. Mächtige Tools können manchmal auch mächtig danebengehen. Das sollte man auch hier besonders beachten.

Zunächst müssen wir in der FRITZ!Box den Zugriff auf die verfügbaren Dienste per TR-064 aktivieren. Das können wir mit einem angeschlossenen Telefon und der Tastenkombination #96*5 tun bzw. in der erweiterten Ansicht der Administrationsoberfläche der FRITZ!Box unter

Heimnetz → Netzwerk → Netzwerkeinstellungen → weitere Einstellungen

durch die Aktivierung von „Zugriff auf Anwendungen" zulassen. Zusätzlich muss – soweit nicht ohnehin vorhanden – unter System → FRITZ!Box-Benutzer ein Benutzer angelegt werden.

Um unseren Flow zur Präsenzerkennung zu erstellen, ziehen wir zunächst einen Inject-Node in den Editor, bei dem wir unter Nutzdaten die MAC-Adresse unseres Smartphones eingeben.

Nutzdaten {} {"NewMACAddress": "FF:FF:FF:FF:FF:FF"}

Die Adresse kann man, je nach Smartphone, zum Beispiel bei den Einstellungen zum Gerät nachschauen.

Dann ziehen wir einen FRITZ!Box-In-Knoten in den Editor und müssen einen neuen FRITZ!Box-Zugang konfigurieren (Bild 10).

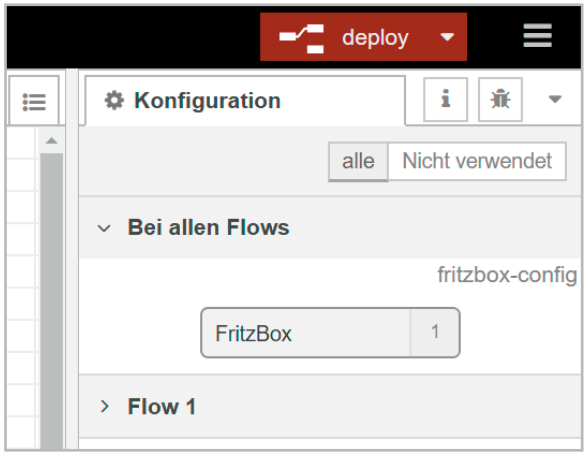


Bild 11: Übersicht über die in Node-RED verwendeten Konfigurations-Nodes

Diese Konfiguration wird in dem nicht im Editor im linken Fenster sichtbaren „fritzbox-config"-Node geschrieben. Alle Konfigurations-Nodes findet man übrigens im rechten Fenster unter Konfigurations-Node (Bild 11).

Dann muss man unter Properties noch die in Bild 12 gezeigten Werte und Einstellungen eintragen.

Danach fügt man einen Debug-Node hinzu, verbindet die Knoten und deployed den Flow. Nun sollten nach dem Auslösen des Inject-Node im Debug-Fenster die Detaildaten für das mit der MAC-Adresse im heimischen WLAN eingeloggten Smartphones erscheinen (Bild 13). Hier ist insbesondere der Schlüssel NewActive interessant, der mit dem Wert 1 das in das WLAN eingeloggte Smartphone signalisiert.

Die Gegenkontrolle – mit aus dem WLAN ausgeloggten Smartphone – sollte unter NewActive eine 0 als Rückgabewert ergeben.

Diese 0 oder 1 beim Schlüssel NewActive kann man nun für eine Präsenzerkennung – die sich natürlich nur auf das im WLAN eingeloggte oder nicht eingeloggte Smartphone bezieht – zum Beispiel in einem

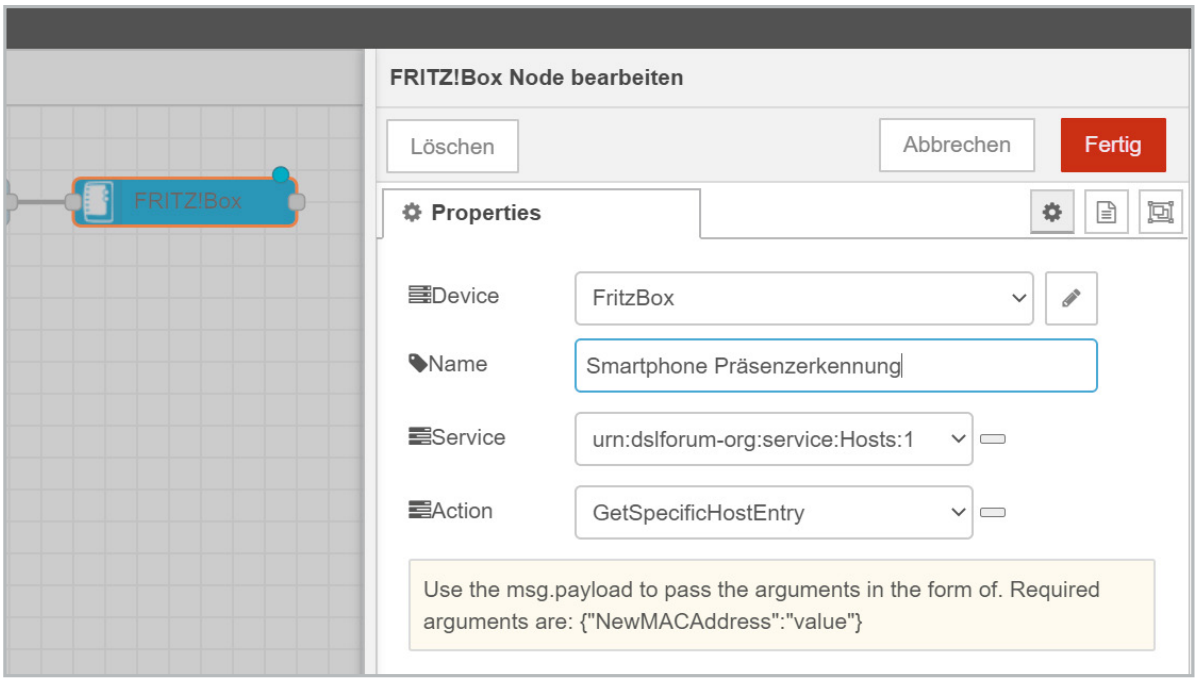


Bild 12: Eintragen der Werte und Einstellungen in Properties

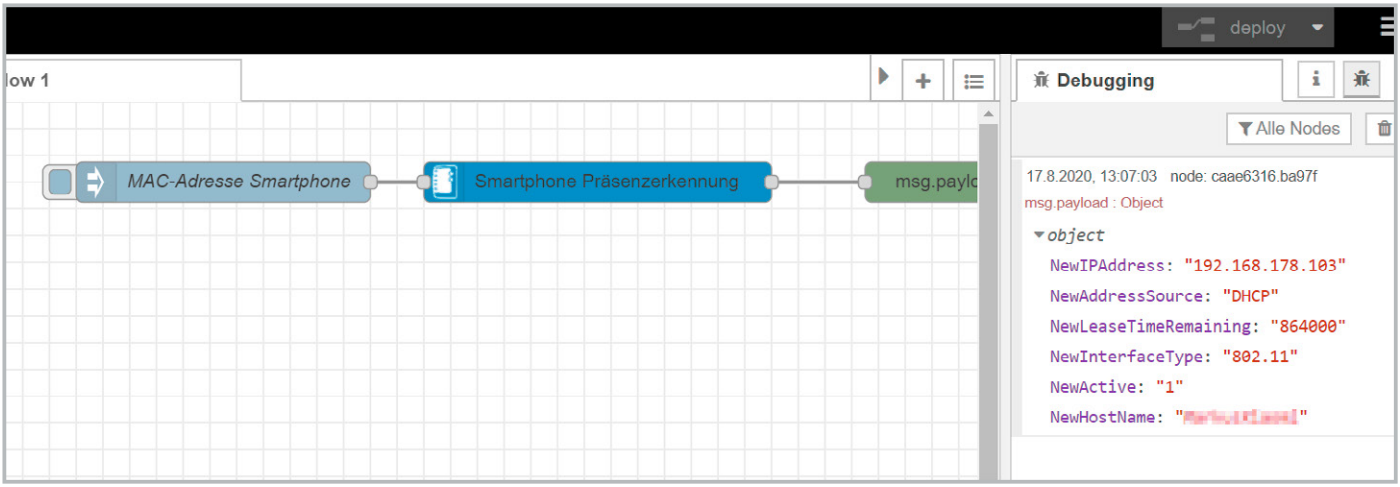


Bild 13: Rückgabe der Detaildaten bei Anfrage mit MAC-Adresse

Function-Node nutzen. Wie man Key/Value-Pairs in einem Function-Node ausliest, haben wir bereits im zweiten Teil unserer Node-RED-Einführung gezeigt [5]. In diesem Fall müsste man

msg.payload.NewActive

abfragen und weiterverarbeiten.

Tipp: Um Werte wie die Rückgabe des NewActive-Schlüssels in einem Flow oder sogar global zugänglich zu machen, kann man beispielsweise einen Change-Node nutzen und statt msg. unter Festlegen global. wählen und diesen Wert als msg.payload festlegen. Dann ist diese Variable global verfügbar.

Beispiel Change-Node:
Festlegen global.FritzboxNewActiveSmartphone
bis msg.payload

Mit einer Abfrage in einem Function-Node wie:

```
var smartphonePresence = global.get('Fritzbox
NewActiveSmartphone');
```

kann dann auf diese Variable (in unserem Fall 0/1) zugegriffen werden.

Zwitschern aus Node-RED

Auch für Social-Media-Dienste wie Facebook, Twitter oder Instagram gibt es bei Node-RED entsprechende Module. Viele über das Internet ansprechbare Dienste können per API (Application Programming Interface = Anwendungs-Programmierschnittstelle) angesprochen werden. Damit werden von dem Anbieter, also beispielsweise von Twitter, einige der Funktionen (Beispiel: Absenden eines Tweets) nach außen gekapselt zugänglich gemacht. So kann der Nutzer in einer vordefinierten Form auf den Dienst eines Anbieters zugreifen, muss aber bei Änderung oder Einstellen der API-Funktion den entsprechenden Teil jeweils anpassen. Die Dokumentation zur Twitter-API findet man unter [6].

In Teil 1 unserer Einführung [1] haben wir uns in dem Beispiel zum Abruf von Wetterdaten vom Anbieter Openweathermap.org diese API-Funktionalität bereits zunutze gemacht, ohne näher auf die Hintergründe einzugehen. Am Beispiel Twitter wollen wir nun ein wenig hinter die Kulissen schauen.

Wir installieren dazu das Modul node-red-node-twitter und erhalten die beiden Nodes „twitter in" und „twitter out". Als Konfigurations-Node wird zusätzlich twitter-credentials installiert.

Damit wir auf die Dienste von Twitter zugreifen können, müssen wir neben einem Benutzerkonto (Twitter ID @TwitterName) noch eine Anwendung unter https://developer.twitter.com/en/apps konfigurieren. In dem Konfigurations-Node twitter-credentials, der im Hintergrund beim erstmaligen Aufruf eines „twitter in"- oder „twitter out"-Node gefüllt werden muss, werden die notwendigen Daten wie Twitter ID, API key, API secret key, Access token und Access token secret eingegeben.

Die Werte für Access token und Access token secret sollte man sich übrigens gleich an einem sicheren Ort notieren, da diese nach dem ersten Generieren auch auf der eigenen Entwicklerseite verborgen werden.

Benutzen wir nun einen Inject-Node mit einem String, können wir diesen mit einem „twitter out"-Node über unser Twitter-Konto versenden und den Vorgang natürlich wieder mit allen Möglichkeiten von Node-RED automatisieren. Zwar ist im Fehlerfall die Menge der möglichen Tweets über die API auf 300 in drei Stunden begrenzt. Aber auch das ist schon eine große Menge und man sollte genau prüfen, was man tut, damit der erzeugte Flow in Node-RED nicht in einer Flut von Tweets endet.

Schauen wir nun noch einmal unter die Haube von Node-RED und sehen uns die Implementierung in einem Modul mit den einzelnen Nodes genauer an. Dazu gehen wir zu

home/pi/.node-red/node_modules/node-red-node-twitter

und rufen die Datei

27-twitter.js

per cat-Kommando auf der Konsole oder mit einem Editor wie nano bzw. einem Desktop-Editor auf.

Hier finden wir recht weit am Ende des Codes den Teil, der für die Versendung von Tweets „twitter out"-Node ❶) über die Twitter API zuständig ist. Zunächst werden die entsprechenden Authentifizierungsdaten ❷ festgelegt:

```
function TwitterOutNode(n) {
```
❶

```
  RED.nodes.createNode(this,n);
  this.topic = n.topic;
  this.twitter = n.twitter;
  this.twitterConfig = RED.nodes.getNode(this.twitter);
  var credentials = RED.nodes.getCredentials(this.twitter);
  var node = this;
  node.status({});

  if (this.twitterConfig.oauth) {
    var twit = new Ntwitter({
      consumer_key: credentials.consumer_key,
      consumer_secret: credentials.consumer_secret,
      access_token_key: credentials.access_token,
      access_token_secret: credentials.access_token_secret
    });
```
❷

Etwas weiter unten im Code, der zunächst die msg.payload auf eine Direktnachricht prüft, finden wir dann die Funktionalität und auch die API-URL aus der Twitter-API-Dokumentation [7], die das Versenden eines Tweets beschreibt wieder:

```
...
node.twitterConfig.post("https://upload.twitter.com/1.1/media/upload.json"...
...
```

Node-RED ist also auch hinter den Kulissen kein Hexenwerk und mit ein bisschen JavaScript und Verständnis der Funktionen und Abläufe kann man vorhandene Nodes verstehen und irgendwann vielleicht sogar eigene Module mit Nodes programmieren.

Maschine an Maschine

Zuletzt wollen wir uns in dieser Sammlung nützlicher Informationen zur Funktionalität von Node-RED ein Protokoll anschauen, das immer beliebter wird und besonders für die Maschine-zu-Maschine-Kommunikation genutzt wird. Die Rede ist von MQTT.

Das „Message Queuing Telemetry Transport"-Protokoll (MQTT) ist ein offenes Netzwerkprotokoll, das die Übertragung von Daten in Form von Nachrichten zwischen Geräten ermöglicht und dies trotz (hoher) Latenzen oder eingeschränkt arbeitender Netzwerke. Besonders in einfachen Geräten für das Internet der Dinge (IoT = Internet of Things) wie SoCs (System-on-Chip) oder Mikrocontroller, eingebettete Systeme, aber auch in der Industrie wird das Protokoll für die Kommunikation zwischen Maschinen verwendet.

Ein sogenannter Broker ist dabei die (einzelne) Verbindungsstelle für die angeschlossenen Clients, die ihre Daten an diesen Broker übermitteln oder von dort abrufen können. Der Vorteil dieses Systems ist, dass die Clients sehr schlanke Systeme sein können (SoCs, Mikrocontroller), da die Datensammlung und die Verarbeitung auf dem Broker stattfinden, der in der Regel deutlich mehr Rechenperformance aufweist.

Dem System liegt ein sogenanntes Publish-Subscribe-Modell zugrunde. Dabei können Clients (wie Mikrocontroller) Daten an den Broker senden (publish) und dieser kann diese an Empfänger (Mikrocontroller, Server etc.) senden, die sich für den Empfang dieser Daten angemeldet (subscribe) haben. Dabei müssen die Teilnehmer des Datenverkehrs keine Informationen über die sendende Gegenstelle (IP-Adresse ...) haben, da die gesamte Kommunikation über den Broker läuft.

Die Daten werden an einen sogenannten Topic als Daten-Endpunkt gesendet bzw. von dort abgerufen, dem ein Schrägstrich (/, slash) als Trennzeichen für die Hierarchie dient. Ein mögliches Topic wäre demnach z. B.:

haus/wohnzimmer/temperatur

Die verschiedenen Topic-Level haben unter anderem den Vorteil, dass man unter einem Topic-Level (z. B. /wohnzimmer/) alle darunter vorhandenen Datenlieferanten auf einmal abfragen kann.

Mit drei verschiedenen QoS-Stufen (Quality of Service)

0 – höchstens einmal
1 – mindestens einmal
2 – genau einmal

kann man zudem die Qualität des Nachrichtentransports definieren. Daneben gibt es noch weitere Einstellungen für das Speichern bzw. Zurückhalten von Daten (retain) sowie für gesicherte Übertragungen.

Dies soll nur ein sehr knapper Blick hinter die Kulissen von MQTT sein – das Internet bietet auch hier wieder eine Fülle von Informationen zu Protokolldetails, Funktionen und den verschiedenen Brokern.

MQTT mit Mosquitto auf dem Raspberry Pi

Ein bekannter Open-Source-Broker für den Raspberry Pi ist Mosquitto von der Eclipse Foundation [8]. Wir können ihn auf unserem Raspberry Pi mit

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

installieren und mit

```
sudo systemctl enable mosquitto
```

(nach jedem Bootvorgang) starten.

In Node-RED benötigen wir für die Kommunikation mit dem MQTT-Broker – in unserem Fall Mosquitto – das Modul mit den Knoten mqtt in und mqtt out, das standardmäßig mit Node-RED installiert wird.

Tipp: Die Standardmodule von Node-RED wie mqtt sind nicht im Home-Verzeichnis bei den Node-RED-Modulen zu finden, sondern liegen im Verzeichnis

```
/usr/lib/node_modules/node-red/node_modules/
```

Für einen Flow mit einer MQTT-Funktionalität nutzen wir in unserem Beispiel einen „mqtt out"-Node, den wir in den Editor ziehen (Bild 14).

Wir geben hier die IP-Adresse an, unter der wir zuvor den Mosquitto-Broker installiert haben. Natürlich ist es möglich, im Netzwerk verschiedene MQTT-Broker zu nutzen, auf die auch verschiedene MQTT-Nodes zugreifen können – auch mehrere auf den gleichen Topic auf einem Broker, da dies durch die Publish-Subscribe-Funktionalität ermöglicht wird.

Danach müssen wir noch den „mqtt out"-Node konfigurieren. Wir geben als QoS-Level 0 (höchstens einmal) an und definieren als Topic

```
fritzbox/presence.
```

Unter diesem Topic an diesem Broker in diesem Fall mit der IP 192.168.178.36 kann nun jeder autorisierte Teilnehmer die veröffentlichten (publish) Daten abfragen (subscribe) (Bild 15).

Wir fügen dem Flow noch einen Inject-Node hinzu, der die Kommunikation mit der FRITZ!Box aus unserem obigen Beispiel zur Präsenzerkennung simuliert. Als Payload fügen wir eine 1 mit dem Datentyp „number" ein.

Wir verbinden die Nodes, deployen den Flow und aktivieren den Inject-Node. Im Debug-Fenster beobachten wir: nichts.

Klar, denn es fehlt ja noch die Gegenstelle (Client, subscribe), die wir mit einem „mqtt in"-Node simulie-

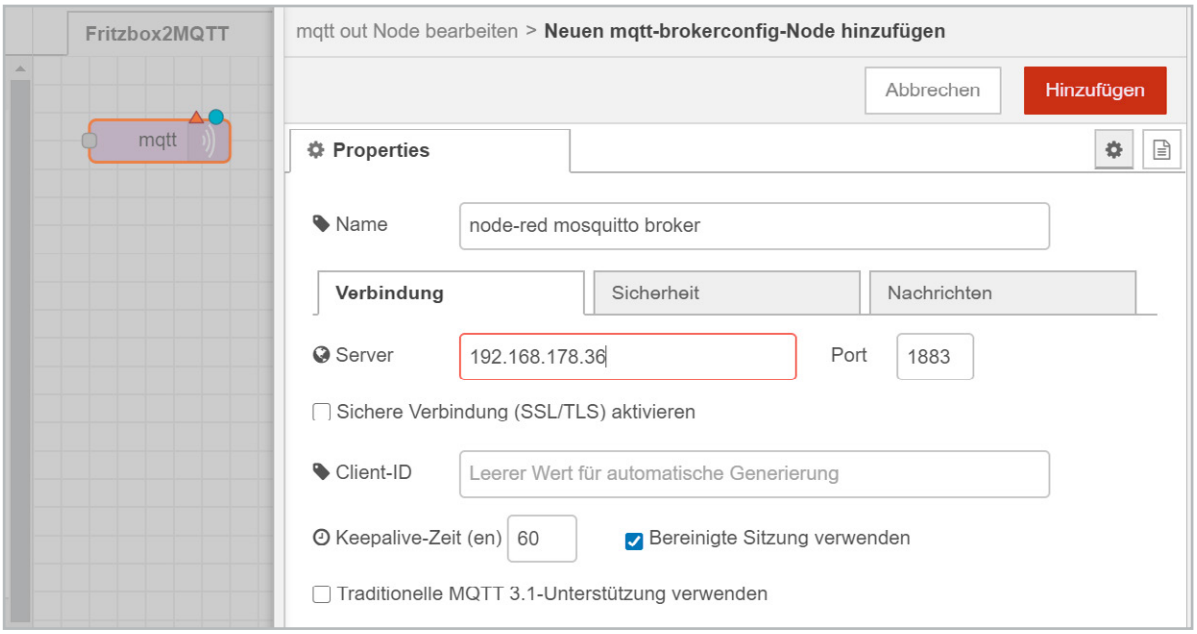


Bild 14: Konfiguration des MQTT-Brokers

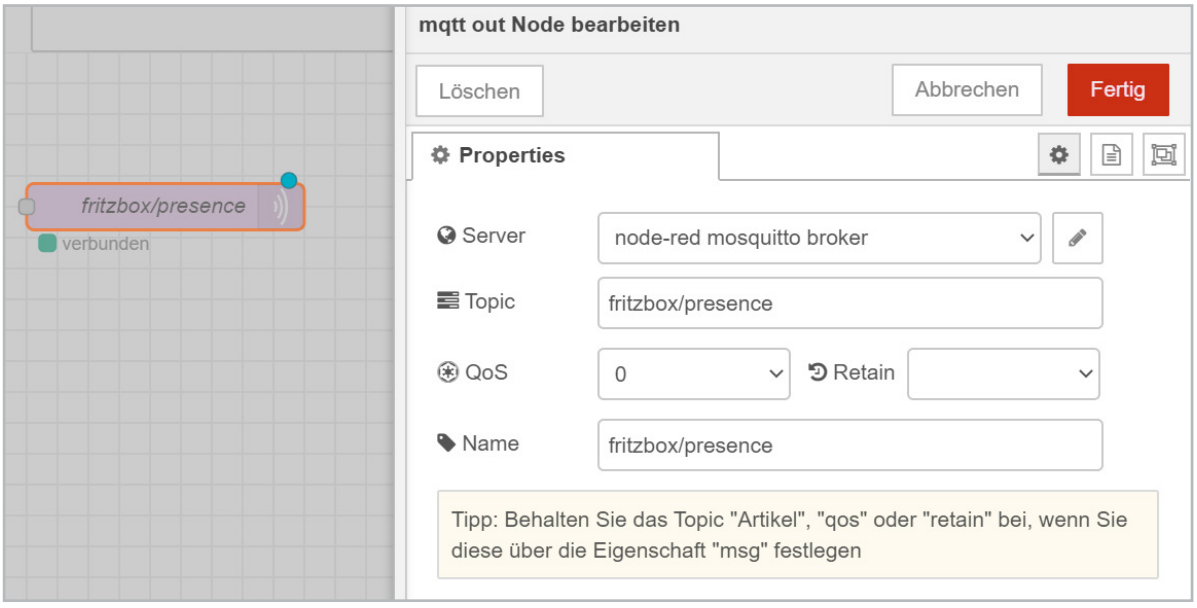


Bild 15: Festlegen des Topics und des QoS-Levels

ren. Wir ziehen diesen Node in den Editor, konfigurieren den Server, Topic und QoS wie in dem „mqtt out"-Node und schließen noch einen Debug-Node an. Wir verbinden die Nodes und deployen den Flow.

Nun sollte beim Aktivieren des Inject-Nodes im Debug-Fenster eine 1 erscheinen und uns die erfolgreiche – wenn auch nur simulierte – Verbindung anzeigen (Bild 16).

Wir können uns diesen Datenverkehr auch außerhalb von Node-RED anschauen. Dazu öffnen wir ein Terminal und nutzen den Mosquitto-Client zum Abonnieren von Daten (mosquitto_sub) mit dem passenden Topic (-t fritzbox/presence), dem Host (-h 192.168.178.36) und wollen beim Veröffentlichen von neuen Daten (Präsenzerkennung) diese höchstens einmal (QoS=0, -q 0) erhalten.

Mit dem Befehl

```
mosquitto_sub -h 192.168.178.36 -t fritzbox/presence -q 0
```

aktivieren wir das Abonnieren (subscribe) der Daten.

Schicken wir nun in Node-RED per Inject-Node wieder eine 1 unter dem o. a. Topic an den Broker, sollte dies auch auf der Konsole erscheinen (Bild 17).

Fazit

Die Möglichkeiten von Node-RED, vor allem als Plattform für schnelles Prototyping, sind in den letzten Jahren enorm gewachsen. Mittlerweile gibt es mehr als 2500 Module, viele Beispiel-Flows und eine immer größer werdende Community.

Aber nicht nur für das Prototyping eignet sich das Software-Tool, sondern es ist auch ein Spezialist für das Verbinden von Welten. Was als Hardware-Sensor-Wert per Mikrocontroller und WLAN auf der Plattform landet, kann von hier auf die verschiedensten Arten weitertransportiert werden. Das Ganze kann mit einer großen Anzahl von (öffentlich zugänglichen) Datenquellen verknüpft und beispielsweise per http, MQTT oder seriell an die unterschiedlichsten Geräte weiterversendet werden. Zudem ist die Plattform eine Open-Source-Plattform und man kann eigene Software hinzufügen oder vorhandene modifizieren.

In unseren drei Teilen zur Einführung in Node-RED haben wir nur an der Oberfläche der Möglichkeiten gekratzt. Wir wünschen viel Spaß beim Entdecken und Deployen! ELV

Weitere Infos:

[1] Programmieren (fast) ohne Code – Node-RED als universelles Prototyping-Tool, Teil 1: Artikel-Nr. 251410
[2] Node-RED-Dokumentation: https://nodered.org/docs/
[3] Node-RED auf dem Android-Smartphone: https://wiki.termux.com/wiki/Termux:API
[4] Protokolle/Schnittstellen auf der FRITZ!Box: https://avm.de/service/schnittstellen/
[5] Programmieren (fast) ohne Code – Einbinden von Raspberry Pi, ESP32, Arduino und Elektronik-Bauteilen in Node-RED, Teil 2: Artikel-Nr. 251516
[6] Twitter-API-Dokumentation: https://developer.twitter.com/en/docs/twitter-api
[7] Twitter-Status-Update: https://developer.twitter.com/en/docs/twitter-api/v1/tweets/post-and-engage/api-reference/post-statuses-update
[8] Mosquitto (Eclipse): https://mosquitto.org/

Alle Links finden Sie auch online unter: de.elv.com/elvjournal-links

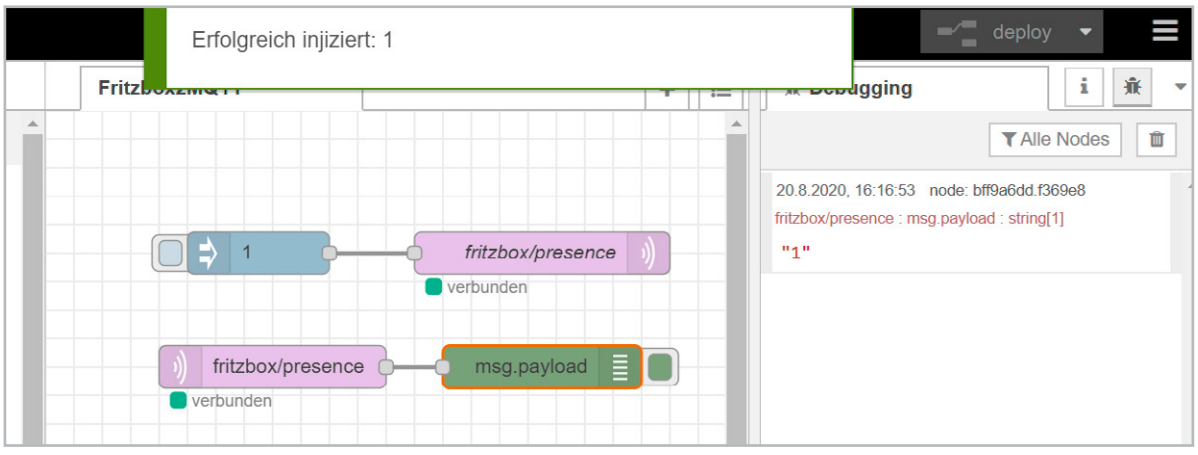


Bild 16: „mqtt out"- und „mqtt in"-Node mit einer simulierten MQTT-Datenkommunikation

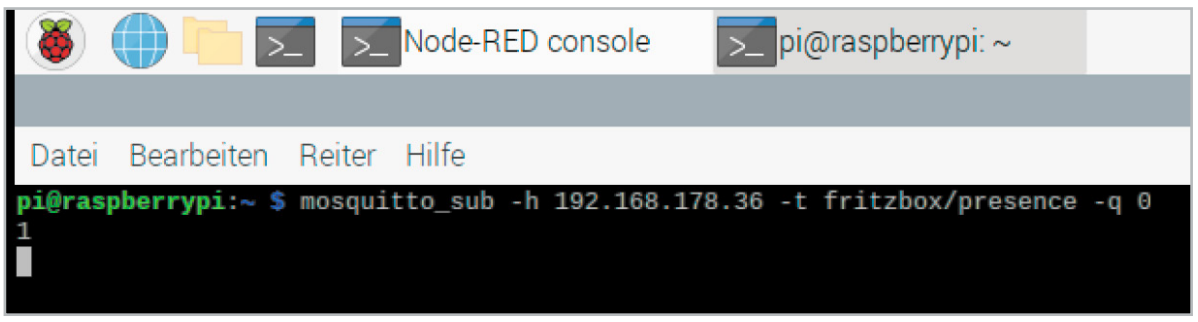


Bild 17: Empfang von Daten per Mosquitto-Client auf der Konsole