edv-buchversand.de

Node-RED

Das umfassende Handbuch

» Hier geht's direkt zum Buch

DIE LESEPROBE

Diese Leseprobe wird Ihnen von www.edv-buchversand.de zur Verfügung gestellt.

Kapitel 3
Das Fundament: die Basics von Node-RED

Panta rhei – alles bewegt sich, alles ist im Fluss. So formulierte es bereits der griechische Philosoph Heraklit. Von Interesse ist aber das Wie und Wohin. Dieses Kapitel führt Sie hinter die Kulissen von Node-RED und stellt Ihnen die Grundlagen vor.

Wenn Sie effizient mit Node-RED arbeiten wollen, müssen Sie das Nachrichtenkonzept und die zentralen Nodes verstehen. Vieles ist dabei sehr intuitiv aufgebaut, und die Theorie dahinter, die ich Ihnen in diesem Kapitel vorstellen möchte, ist nicht schwer. Anhand von einigen Beispielen auf dem Raspberry Pi möchte ich Ihnen aber konkret zeigen, wie das in der Praxis funktioniert. Auch für dieses Kapitel gilt: Ein sequenzielles Abarbeiten der Abschnitte kann schnell ermüdend sein. Verschaffen Sie sich vielleicht zunächst nur einen Überblick, picken Sie sich einzelne Themen heraus und kehren Sie zurück, wenn es später hakt und ausführlichere Informationen nötig werden.

3.1 Das Message-Konzept von Node-RED

In Kapitel 1 haben Sie bereits erfahren, dass Node-RED auf einer Flow-basierten Programmierung beruht, also auf der strukturierten Kommunikation zwischen Datenblöcken. In der Terminologie von Node-RED heißen diese Datenblöcke *Messages* (dt. Nachrichten). Diese Nachrichten fließen von Station zu Station (»von Node zu Node«). Jede Station verändert, ergänzt oder reagiert anderweitig auf die erhaltenen Nachrichten.

Dies erfordert einen einheitlichen Ansatz, also eine übergreifende Struktur der Nachrichten. Node-RED wendet zu diesem Zweck JavaScript-Objekte an.

3.1.1 JSON – das Datenformat für den Datenaustauch

JSON steht für *JavaScript Object Notation*. Es handelt sich um ein Format für den Datenaustausch zwischen einem oder mehreren Systemen, das den modernen Anforderungen der objektorientierten Programmierung in idealer Weise Rechnung trägt.

JSON hat sich in den letzten Jahren derart verbreitet, dass Sie diesem Format kaum mehr entgehen können. Es wird beispielsweise von den Diensten OpenWeatherMap oder Pushbullet eingesetzt, aber auch das Hue-Beleuchtungssystem und viele andere Programme nutzen es.

Obwohl JSON ursprünglich als Untermenge von JavaScript entwickelt wurde, ist es von der Programmiersprache komplett unabhängig. Aktuell gibt es zwar zwei konkurrierende Standards, das spielt für die meisten Anwendungen in der Praxis jedoch keine Rolle. Mehr Informationen finden Sie unter *https://www.json.org/json-de.html*.

Das JSON-Format speichert Daten strukturiert und textbasiert. Es greift den Ansatz der objektorientierten Programmierung auf, indem es die Eigenschaften eines Datenobjekts in einem JSON-Objekt abbildet.

Objektorientierte Programmierung

Objektorientierte Programmierung versucht, Objekte aus der realen Welt (aber natürlich auch ganz oft abstraktere Dinge) auf einem System abzubilden und ihnen eine Identität und Verhaltensweisen zuzuordnen. So ist ein Auto beispielsweise durch seine Attribute (*Eigenschaften*) gekennzeichnet, also durch seine Farbe, seinen Hersteller, seine PS-Zahl etc. Auf dieses Objekt werden dann Operationen (*Methoden*) angewendet. So kann ein Auto geradeaus fahren, hupen oder blinken.

JSON-Objekte kommen ohne umfangreiche Deklarationen oder Beschreibungen aus, da sie immer die gleiche Grundstruktur besitzen. Das minimiert den Overhead für den Austausch der abgebildeten Daten und sorgt dafür, dass sie sowohl maschinenbasiert verarbeitet als auch vom menschlichen Betrachter einfach gelesen werden können. Das Anwendungsgebiet beschränkt sich aber nicht auf den Datenaustausch zwischen Anwendungen, sondern umfasst vielfach auch Parametereinstellungen in Konfigurationsdateien (so auch in Node-RED, z. B. in der Datei *settings.js*, hier ist dann die Dateiendung *.json* bzw. *.js* üblich).

JSON baut auf zwei Strukturen auf:

- **Name-Wert-Paare** – So kann aus `char farbe[] = "blau"` (Programmsicht) `"farbe": "blau"` (JSON) werden.
- **Eine geordnete Liste (Tabelle) von Werten** – So kann aus `char farben [][] = {"blau", "rot", "gelb"}` (Programmsicht) `"farben": ["blau", "rot", "gelb"]` (JSON) werden.

JSON kennt folgende Datentypen:

- Ein *boolescher Wert* (Boolean) hat die Schlüsselwörter `true` und `false`. Als Schlüsselwörter werden sie nicht in Anführungszeichen gesetzt.

  Beispiel: `"anwesend":true`

- Ein *Nullwert* hat das Schlüsselwort `null`.

  Beispiel: `"zweiteAdresse":null`

- Eine *Zahl* kann jede Ziffer sein. Zulässig sind auch ein einleitendes negatives Vorzeichen und ein Dezimalpunkt. Darüber hinaus kann die Zahl durch die Angabe eines Exponenten `e` oder `E` mit einem folgenden Vorzeichen + oder – und einer Folge der Ziffern 0 bis 9 ergänzt werden. Führende Nullen sind nicht erlaubt, einem Dezimalpunkt muss zumindest eine Ziffer folgen:

  `"Temperatur":27.6`

- *Zeichenketten* werden mit doppelten hochgestellten Anführungszeichen eingeleitet und beendet:

  `"Gruss":"Hallo"`

- *Tabellen* beginnen mit `[` und enden mit `]`. Sie enthalten eine durch Kommata getrennte Liste von Werten gleichen oder verschiedenen Typs. Leere Tabellen sind zulässig, führende Kommata sind jedoch nicht erlaubt:

  `"farben": ["blau", "rot", "gelb"]`

- *Objekte* beginnen mit `{` und enden mit `}`. Sie enthalten eine durch Kommata getrennte Liste von Eigenschaften. Eigenschaften wiederum sind Name-Wert-Paare. Leere Objekte sind zulässig, führende Kommata sind jedoch nicht erlaubt:

  `{"Vorname":"Michael", "Nachname":"Mustermann"}`

Einschränkungen ergeben sich daraus, dass das Datenformat nicht zwischen Ganz- und Gleitkommazahlen unterscheidet und dass die Interpretation der Werte der Exponentialdarstellung implementierungsabhängig ist. Auch unterstützt JSON nicht alle von JavaScript unterstützten Datentypen.

Mit der JSON-Notation können Sie Eigenschaften eines Raspberry Pi kurz und eindeutig beschreiben. Die Beschreibung ist aus JSON-Sicht ein Objekt; die Elemente der Beschreibung sind von einer öffnenden geschweiften Klammer `{` und einer schließenden geschweiften Klammer `}` umschlossen.

Das erste Element innerhalb dieser Klammerstruktur beinhaltet den Namen und den Typ des Minicomputers in Form eines Name-Wert-Paars mit einem Zeichenstring als Datentyp (z. B. `"Typ" : "Raspberry PI 3B+"`).

Ein Komma trennt es vom zweiten Element. Dieses beherbergt die Angabe zur Speichergröße in MB ebenfalls in Form eines Name-Wert-Paars mit einer Zahl als Datentyp (z. B. `"SDRAM" : 1024`).

Das nächste durch Komma getrennte Name-Wert-Paar enthält den Takt in GHz als Datentyp Zahl mit einem Dezimalpunkt (z. B. `"Takt" : 1.4`).

Das folgende Element hält Informationen hinsichtlich angeschlossener Geräte an den USB-Ports bereit. Der Elementname ist `"USB"`. Der Wert ist ein Objekt, das wiederum eine Liste von drei Name-Wert-Paaren enthält, z. B.:

```
"USB" : {"USB1" : "USB-Stick", "USB2" : "FP", "USB3" : "Arduino"}
```

Es schließt sich ein Element an, das Informationen zu den *GPIOs* (*General Purpose Input/Output*, d. h. konfigurierbarer digitaler Port eines integrierten Schaltkreises) aufnimmt. Auch hier ist der Wert wieder ein Objekt mit einer Liste von Name-Wert-Paaren (ein Eintrag für jeden GPIO), wobei die Werte hier als Tabelle hinterlegt sind und deshalb in eckigen Klammern stehen, wie etwa:

```
"GPIO" : {"PIN03" : ["GPIO01", "SDA1", "I2C"], "PIN08" : ["GPIO14", "TXD0"] }
```

Ein letztes Name-Wert-Paar soll Auskunft über das Vorhandensein einer Kamera geben, z. B.:

```
"Kamera" : false
```

Die einzelnen Bestandteile des JSON-Objekts sind damit festgelegt. Fix zusammengeschustert, sieht das Objekt, das den Raspberry Pi 3 B+ beschreibt, wie folgt aus:

```
{"Typ" : "Raspberry PI 3B+", "SDRAM" : 1024, "Takt" : 1.4, "USB" : {"USB1" :
"USB-Stick", "USB2" : "FP", "USB3" : "Arduino"}, "GPIO" : {"PIN03" :
["GPIO01", "SDA1", "I2C"], "PIN08" : ["GPIO14", "TXD0"] }, "Kamera" : false}
```

Das ist nicht sonderlich lesbar und augenfreundlich, aber aus der Sicht eines Programms völlig in Ordnung. Eine übersichtlichere Darstellung erreichen Sie mit einem JSON-Formatter wie etwa *https://jsonformatter.org*. Mit ihm können Sie auch selbst erstellte JSON-Strings auf ihre Gültigkeit überprüfen, sie in andere Formate (z. B. XML, CSV) konvertieren und auf den eigenen PC herunterladen (siehe Abbildung 3.1).

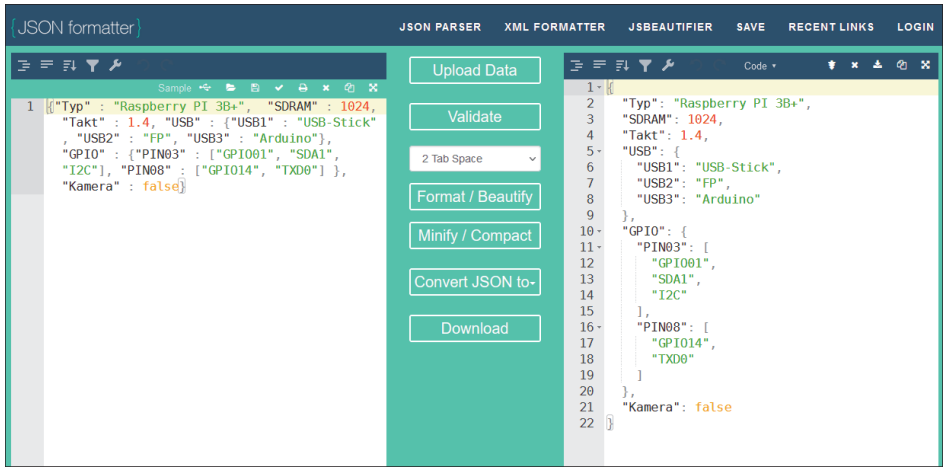


Abbildung 3.1 Der JSON-Formatter

3.1.2 Messages in Node-RED

Das JSON-Objekt für eine Node-RED-Message besteht im einfachsten Fall aus einer Eigenschaft `_msgid` mit einer eindeutigen, die Nachricht kennzeichnenden Nummer und einer Eigenschaft `payload` (dt. Nutzdaten) mit der eigentlichen Nachricht. Sie haben das bereits in einem ersten Ansatz im »Hallo Welt«-Flow des Abschnitt 2.6 kennengelernt.

Sie können im *debug*-Node die Ausgabe in VOLLES NACHRICHTEN-OBJEKT ändern, damit Sie die gesamte Nachricht angezeigt bekommen (siehe Abbildung 3.2).

Abbildung 3.2 Der debug-Node zeigt ein volles Nachrichten-Objekt an.

Erkennbar sind zunächst drei Name-Wert-Paare:

- `_msgid` mit dem Wert `"71ef6333.40b2bc"`
- `payload` mit der Zeichenkette `"Hallo Welt"`
- `topic` mit einem leeren Wert. Der Grund dafür ist, dass diese Eigenschaft im *inject*-Node zwar deklariert, aber nicht gefüllt wurde.

Auf diesem Grundsystem beruht die gesamte Kommunikation in Node-RED. Sie ist jedoch notwendigerweise facettenreicher.

Eine Message kann weitere Eigenschaften haben. Sie legen diese in den Eigenschaften des Nodes und dort in Listeneinträgen fest. Das gilt z. B. für den *inject*-Node, den *change*-Node und viele mehr, wie Abbildung 3.3 zeigt.

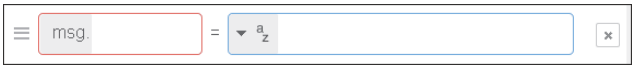


Abbildung 3.3 Message-Eigenschaften werden per Listeneintrag definiert.

Ein mögliches Name-Wert-Paar passend für das Beispiel zum Raspberry Pi wäre also `wertung` und `leistungsfähiger Einplatinenrechner` als Zeichenkette, so wie in Abbildung 3.4.

Abbildung 3.4 Message-Eigenschaft für ein Name-Wert-Paar

Im simpelsten Fall ist der Wert des Name-Wert-Paars eine Zeichenkette oder eine Zahl. Die Optionen für den Wert sind jedoch weitaus vielfältiger.

Ein Klick auf den Pfeil nach unten blättert die Möglichkeiten auf:

- **msg.** – Der Wert ist ein `msg`-Objekt (also `msg.payload` oder `msg.topic`).
- **flow/global.** – Diese Optionen betreffen Kontextvariablen. Näheres dazu finden Sie in Abschnitt 8.1.
- **$^{a}_{z}$ (String)** – kennzeichnet eine Zeichenkette.
- **$^{0}_{9}$ (Number)** – nimmt eine Zahl auf. Das Dezimaltrennzeichen ist ein Punkt.
- **o (Boolean)** – ist ein boolescher Wert, also entweder wahr oder falsch (`true`/`false`).
- **{} (JSON)** – bezieht sich auf ein JSON-Objekt (siehe Abbildung 3.5).

Abbildung 3.5 Message-Eigenschaft für ein »JSON«-Name-Wert-Paar

Sie können das Objekt direkt eingeben. Es ist aber besser, wenn Sie über ▪▪▪ in den JSON-Editor wechseln. Abbildung 3.6 zeigt den Node-RED-JSON-Editor mit dem JSON-Objekt »Raspberry Pi 3B+« aus dem vorherigen Abschnitt.

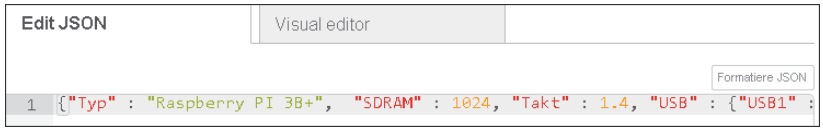


Abbildung 3.6 Der JSON-Editor von Node-RED

Ein Klick auf FORMATIERE JSON erzeugt eine lesefreundliche Darstellung; ein roter Hinweis macht auf Syntaxfehler aufmerksam (siehe Abbildung 3.7).

Abbildung 3.7 Der JSON-Editor zeigt einen Fehler an.

Fahren Sie für nähere Informationen mit dem Mauszeiger auf das Quadrat. Diese Infos sind allerdings nur als grobe Anhaltspunkte zu verstehen, da der eigentliche Fehler oftmals an anderer Stelle liegt – in diesem Beispiel ist es das fehlende Komma in Zeile 3. Beheben Sie etwaige Fehler, denn Node-RED macht Sie bei na-

hezu jeder Aktion auf noch vorhandene Mängel aufmerksam. Das ist auf Dauer störend. Hilfsweise können Sie den Node deaktivieren.

Die Registerkarte VISUAL EDITOR bzw. VISUELLER EDITOR erlaubt die sukzessive Entwicklung eines JSON-Objekts (siehe Abbildung 3.8).

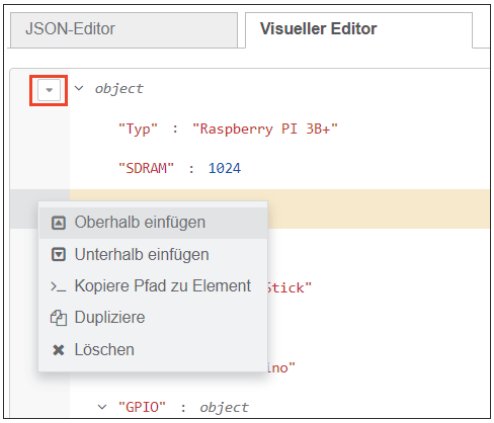


Abbildung 3.8 Der visuelle JSON-Editor von Node-RED

Legen Sie den Fokus der Bearbeitung mit dem Mauszeiger auf eine bestimmte Zeile. Diese ist dann gelb hinterlegt. Durch Klick auf die entsprechenden Felder lassen sich die Name-Wert-Paare des Eintrags unmittelbar ändern. Wenn Sie auf das Dreieck klicken, klappt ein Menü mit weiteren Bearbeitungsmöglichkeiten auf. Dort legen Sie den Typ der Message fest:

- **01 (Buffer)** – Node-RED liest Daten aus einer Datei oder aus dem Netz Byte für Byte in einen Datenpuffer ein. Dies sind Binärdaten, auf die über das Buffer-Objekt zugegriffen wird. Mehr dazu finden Sie in Kapitel 5, »Funktionen programmieren«.
- **O (Timestamp** (dt. Zeitstempel)) gibt die Zeit in Sekunden seit dem 01.01.1970 00:00:00 UTC (*Universal Time Coordinated*) an. Das ist die sogenannte *Unix-Zeit*. JavaScript speichert die Zeit dagegen in Millisekunden. Mehr dazu finden Sie in Kapitel 5 und Kapitel 8, »Daten speichern und archivieren«.
- **J: (Ausdruck)** bezieht sich auf *JSONata* (*http://docs.jsonata.org/*). Dies ist eine leichtgewichtige Abfrage- und Transformationssprache für JSON-Daten – mit ihr wenden Sie einfache Funktionen direkt auf Werte an. Sie können sie beispielsweise nutzen, um Werte anders zu formatieren, auf- und abzurunden oder um Berechnungen durchzuführen.

  JSONata wird ebenfalls in JavaScript implementiert, weist jedoch einige Abweichungen auf. Da Node-RED auf Node.js beruht, können Sie auf JSONata auch in den entsprechenden Nodes zurückgreifen. Entwickeln und testen können Sie Ihren JSONata-Ausdruck am einfachsten mit *https://try.jsonata.org*. Dort finden Sie auch ein Videotutorial.

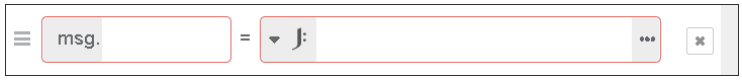


Abbildung 3.9 Message-Eigenschaft des Ausdrucks

JSONata-Ausdrücke können Sie direkt eingeben, komfortabler ist aber der JSONata-Ausdruckseditor, den Sie über ▪ ▪ ▪ erreichen. Abbildung 3.10 zeigt ein Beispiel für die Umwandlung einer Kommazahl aus `msg.payload` in eine Ganzzahl.

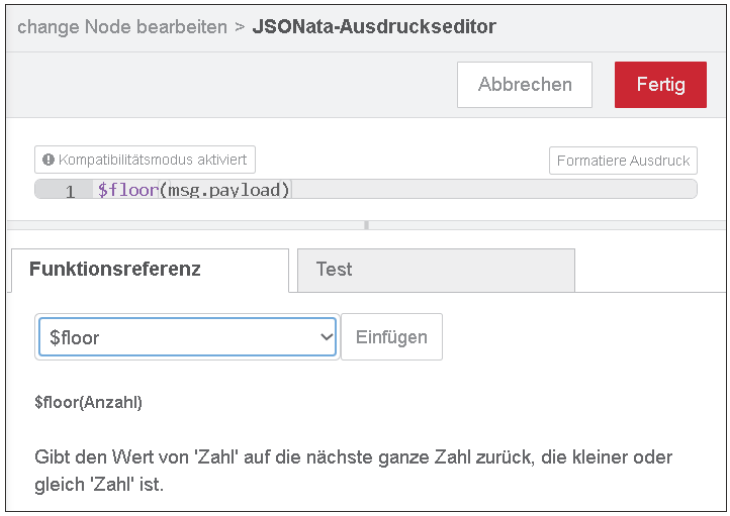


Abbildung 3.10 Der JSONata-Ausdruckseditor von Node-RED

Die Schaltfläche FORMATIERE AUSDRUCK formatiert den Ausdruck lesefreundlich.

Das Auswahlmenü der Registerkarte FUNKTIONSREFERENZ führt zu einer Liste der verfügbaren Funktionen. Die Anzeige der Syntax und eine kurze Beschreibung dienen Ihrer Information.

Nutzen Sie die Registerkarte TEST, um Ihren Ausdruck zu validieren. In Abbildung 3.11 sehen Sie, dass `99.77` in `99` umgewandelt wurde.

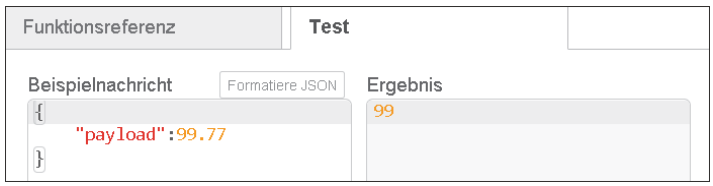


Abbildung 3.11 Der JSONata-Ausdruckseditor, mit dem Sie Ausdrücke testen können

- **$ (env. Variable)** kennzeichnet eine *Umgebungsvariable*. Dies sind konfigurierbare Variablen in Betriebssystemen, die oft Pfade zu bestimmten Programmen oder Daten enthalten oder Informationen und Einstellungen für mehrere Programme

vorhalten. Meist handelt es sich um Zeichenketten. Abbildung 3.12 zeigt ein Beispiel mit der Umgebungsvariablen `$PATH`.

Abbildung 3.12 Message-Eigenschaft einer Umgebungsvariablen

Vordefinierte Umgebungsvariablen

Node-RED bietet selbst einige Umgebungsvariablen an. Dies sind:

- `NR_NODE_ID` – die ID des Nodes
- `NR_NODE_NAME` – der Name des Nodes
- `NR_NODE_PATH` – der Node-Pfad (das Konzept ergibt sich aus den folgenden Umgebungsvariablen)
- `NR_GROUP_ID` – die Gruppen-ID, zu der der Node gehört
- `NR_GROUP_NAME` – der Name der Gruppe
- `NR_FLOW_ID` – die ID des zugehörigen Flows
- `NR_FLOW_NAME` – der Name des zugehörigen Flows

Nachstehender Codeschnipsel zeigt die Verwendung im Rahmen der Programmierung von *function*-Nodes (siehe Kapitel 5).

```
const flowName = env.get("NR_FLOW_NAME")
msg.payload = `Nachricht gesendet von Flow '${flowName}'`
return msg
```

In den folgenden Abschnitten geht es darum, dieses Prinzip aus verschiedenen Blickwinkeln zu betrachten, zu vertiefen und mit Leben zu füllen.

3.2 Die Geschwister inject-Node und debug-Node

Der *inject*-Node und der *debug*-Node sind zwei leistungsfähige Geschwister, die Sie multifunktional einsetzen können. So können Sie mehr als nur einen Flow eröffnen und abschließen.

3.2.1 Der inject-Node

Der *inject-Node* injiziert eine Nachricht entweder manuell (das haben Sie bereits kennengelernt) oder automatisch. Dies können Sie über das Eigenschaftenfenster parametrisieren, das auf den ersten Blick recht aufgeräumt wirkt (siehe Abbildung 3.13). Sein bemerkenswerter Charakter erschließt sich dann aber im Detail.

Abbildung 3.13 Eigenschaften des inject-Nodes

Der *inject*-Node besitzt folgende Eigenschaften.

Message-Objekte

Jedes Message-Objekt hat einen eigenen Eintrag. Ein Klick auf das ×-Icon löscht den Listeneintrag. Die Schaltfläche +HINZUFÜGEN erstellt einen neuen Listeneintrag, den Sie nun mit einem Name-Wert-Paar (siehe Abschnitt 3.1.2) belegen können.

Zur Erinnerung: Jede Message hat eine Message-ID (`msg._msgid`). Das System vergibt diese automatisch beim Erstellen einer neuen Nachricht.

Zeitpunkt

Der *inject*-Node injiziert eine Nachricht üblicherweise manuell. Der Klick auf die Schaltfläche ist dann auch der Zeitpunkt, an dem die Nachricht eingespeist wird.

Aktivieren Sie das Kontrollkästchen EINMAL NACH DER INJEKTION für eine einmalige automatische Injektion. Diese erfolgt bei jedem Neustart eines Flows, also nach jedem Deployment, aber auch nach jedem Systemstart. Deshalb eignet sich diese Funktionalität hervorragend für vorbereitende oder initialisierende Maßnahmen beim Start von Node-RED.

Mit der Zeitangabe stellen Sie die Verzögerung ein, d. h., wie lange das Einspeisen nach dem Deploy oder dem Systemstart erfolgen soll. Ein Anwendungsfall ist, wenn die nachgelagerten Prozessschritte ihrerseits auf die Beendigung vorbereitender Maßnahmen warten müssen.

Die Aktivierung des Kontrollkästchens erkennen Sie auch an der hochgestellten 1 neben dem Flow-Namen im Design-Bereich.

Wiederholungen

Wiederholungen bewirken ebenfalls ein automatisiertes Auslösen des *inject*-Nodes. Wiederholungen sind möglich:

- **als Intervall** – Die Angabe erfolgt in Sekunden, Minuten oder Stunden.
- **zu einem bestimmten Zeitpunkt** – Die Angabe erfolgt über durch einzelne Kontrollkästchen aktivierbare Wochentage in Stunden:Minuten (z. B. 12:00).
- **als Intervall zwischen zwei Uhrzeiten** – Das entspricht einer Kombination aus den beiden vorgenannten Optionen.

Die Aktivierung von Wiederholungen erkennen Sie auch an einem Wiederholungssymbol neben dem Flow-Namen im Design-Bereich.

Die Schaltfläche inject now

Mit der Schaltfläche INJECT NOW senden Sie eine Nachricht aus dem Eigenschaftenfenster des Nodes. So können Sie im Test zeitraubende Deployments vermeiden und das Ergebnis der Änderungen direkt in der Debug-Ausgabe ablesen – eine feine Sache.

3.2.2 Der debug-Node

Den *debug*-Node haben Sie bereits kennengelernt. Als ständiger Begleiter in der Testphase Ihrer Projekte wird er Ihnen wertvolle Dienste leisten (unter EIGENSCHAFTEN, siehe Abbildung 3.14).

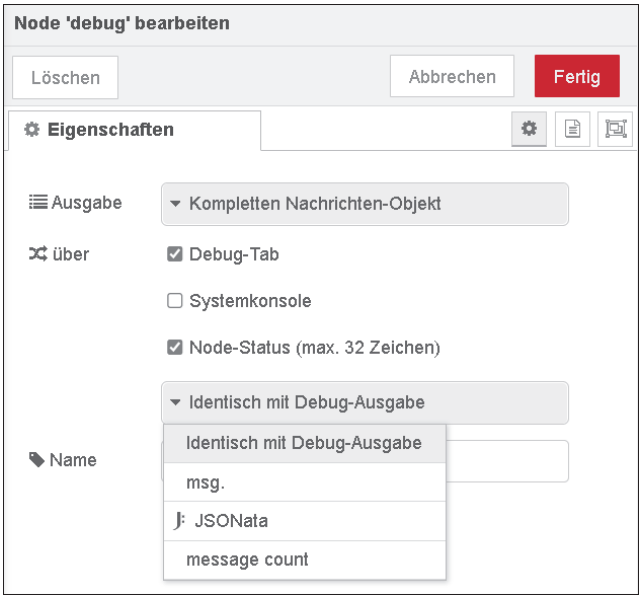


Abbildung 3.14 Eigenschaften eines debug-Nodes

Die Node-Eigenschaften sind sehr überschaubar:

Ausgabe

AUSGABE kann Folgendes sein:

- ein `msg`-Objekt, z. B. `msg.payload` oder `msg.topic`
- das volle Nachrichten-Objekt
- ein Ausdruck – Beispielsweise zeigt `msg.payload * 2` bei einem Wert von `4` in `msg.payload` als Ergebnis 8 im Debug-Fenster an.

über

Hier bestimmen Sie Ziel und Art der Ausgabe, eine Mehrfachauswahl ist zulässig:

- das Debug-Fenster
- die Systemkonsole
- der Node-Status – Hier gibt es vier Möglichkeiten (siehe Abbildung 3.14). Interessant ist insbesondere Option vier. Sie zeigt in der Node-Decoration (siehe Kapitel 2) die Anzahl der empfangenen Nachrichten an (siehe Abbildung 3.15).

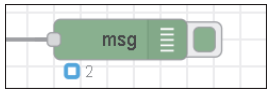


Abbildung 3.15 debug-Node: Anzahl der empfangenen Nachrichten

Richtig interessant machen den *debug*-Node die Möglichkeiten des Debug-Fensters in der rechten Seitenleiste. Abbildung 3.16 zeigt ein Beispiel für eine Debug-Ausgabe.

Abbildung 3.16 Debug-Ausgabe

Die Kopfzeile enthält Informationen zum Zeitpunkt der Ausgabe und die Node-ID des *debug*-Nodes. Da die Kenntnis der Node-ID in aller Regel wenig weiterhilft, erhält der entsprechende Node eine gestrichelte Umrandung, sobald Sie mit dem Mauszeiger den Fokus auf die Meldung legen.

Die zweite Zeile zeigt das Nachrichtenthema (siehe auch den Abschnitt zur guten Programmierung in Kapitel 2) und das Datenformat (z. B. Objekt, Zeichenkette, Zahl) als Zeichenkette an.

Die dritte Zeile schließlich bildet den Nachrichteninhalt gemäß den vorgenommenen Debug-Einstellungen ab. Die Inhaltsbestandteile können Sie über die beiden Icons kopieren:

- COPY PATH kopiert den Namen des JSON-Objekts (hier: `payload`).
- COPY VALUE kopiert den Wert.

Üblicherweise erfolgt die Debug-Ausgabe in komprimierter Form. Ist sie jedoch ein JSON-Objekt (siehe Abschnitt 3.1.1 bzw. Zeile 1), lassen sich die Elemente mit den Pfeil-Icons auf- und wieder einklappen (siehe Abbildung 3.17).

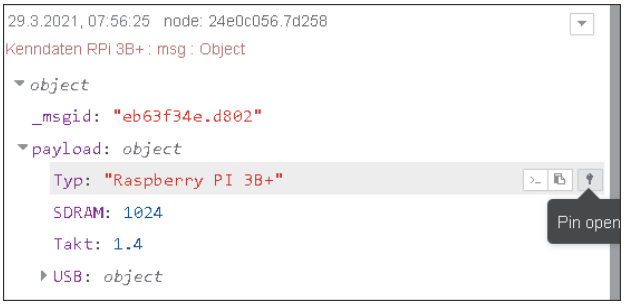


Abbildung 3.17 Die Debug-Ausgabe mit dem JSON-Objekt

Ein weiteres Icon (PIN OPEN) funktioniert wie ein Schalter. Ist er aktiviert, expandiert Node-RED bei jeder weiteren Debug-Ausgabe des entsprechenden *debug*-Nodes automatisch das gesamte Nachrichtenobjekt bis in die tiefsten Verästelungen.

Status eines debug-Nodes

Abschließend noch ein Hinweis zum Status eines *debug*-Nodes: Entfernen oder deaktivieren Sie den *debug*-Node bzw. schalten Sie ihn zumindest mit dem Schalter an der rechten Node-Seite aus, wenn Sie ihn nicht mehr für Testzwecke benötigen.

3.3 Messages manipulieren: Der change-Node und seine Begleiter

Die Verarbeitung von Daten (und nichts anderes macht Node-RED) lebt von Datenänderungen oder Verzweigungen im Prozessablauf. Nodes aus der Gruppe FUNKTION übernehmen diese Aufgabe. Tabelle 3.1 liefert Ihnen einen Überblick.

Wirklich erklärungsbedürftig und in der Praxis wichtig sind nur wenige Nodes. Gleichwohl finden Sie Beispiele zu allen genannten Flows im Download-Material unter *https://www.rheinwerk-verlag.de/5824*.

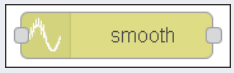
Node	Funktion
	Verzweigung, die eine Nachricht aufgrund einer Bedingung weiterleitet.
	Setzen, Ändern, Löschen oder Verschieben von Eigenschaften einer Nachricht bzw. Kontextvariablen.
	Ordnet einen numerischen Wert einem anderen Bereich zu (vergleichbar mit `map()` in der Arduino-IDE).
	Legt eine Eigenschaft basierend auf einer Vorlage fest.
	Erzeugt eine zufällige Zahl.
	Verschiedene Funktionen, die Sie auf numerische Eingaben anwenden können (Maximum, Minimum, Durchschnitt etc.).
	*rbe* (*Report by Exception*) – gibt eine Nachricht nur bei Wertänderung weiter. Ab Node-RED 2.0.0 heißt dieser Node *filter*.

Tabelle 3.1 Nodes für die »klassische Datenverarbeitung«

3.3.1 Der switch-Node

Der *switch*-Node agiert wie ein Router. Er blockiert regelbasiert Nachrichten oder leitet sie in neue Kanäle. Einen Beispiel-Flow finden Sie in Abbildung 3.18.

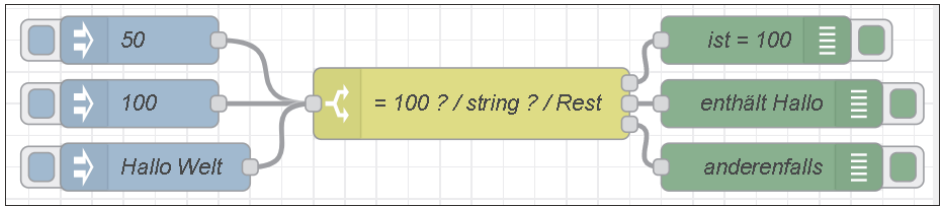


Abbildung 3.18 Der switch-Node reagiert auf die Eingabe.

Die *inject*-Nodes starten den Flow mit der Eingabe der Werte 50, 100 und »Hallo Welt«. Der *switch*-Node routet die Nachricht dann entsprechend dem Wert in unterschiedliche Kanäle (siehe Abbildung 3.19).

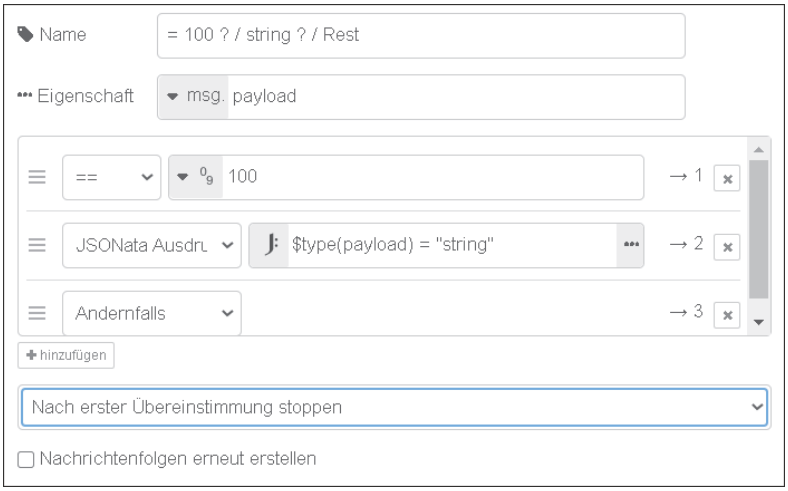


Abbildung 3.19 Die Eigenschaften des switch-Nodes

Das zu überprüfende Feld kann ein Nachrichtenobjekt (hier `msg.payload`), eine Kontextvariable, ein JSONata-Ausdruck oder eine Umgebungsvariable sein.

Die Auswahl des Vergleichsoperators erfolgt über ein Drop-down-Menü. Node-RED unterscheidet vier Regeltypen:

- **Wertregeln** – Die Auswertung erfolgt anhand der konfigurierten Eigenschaft (siehe das Beispiel oben: `== 100` sorgt für einen Treffer, wenn »100« injiziert wird).
- **Sequenzregeln** finden Anwendung bei Nachrichtensequenzen (siehe dazu Abschnitt 3.7 zum *split*-Node).
- **JSONata-Ausdruck** – Die Überprüfung anhand eines JSONata-Ausdrucks erfolgt gegen die gesamte Nachricht. (Im Beispiel oben geht es um den Typ *String*, der getroffen wird, wenn »Hallo Welt« injiziert wird.)
- **Andernfalls-Regel** – Es gibt auch eine Auffangregel, die greift, wenn keine der vorhergehenden Regeln zutrifft. Im Beispiel wird dieser Zweig ausgelöst, wenn 50 injiziert wird.

Regeln fügen Sie mit +HINZUFÜGEN hinzu; das Löschen einzelner Regeln erfolgt über das × in dem jeweiligen Listeneintrag. Jede Regel führt zu einem Ausgangsport des Nodes. Nutzen Sie die Möglichkeit, über die Registerkarte ERSCHEINUNGSBILD bei den Ports nähere Informationen zu hinterlegen.

Die Überprüfung auf Regelübereinstimmung erfolgt sequenziell von der ersten bis zur letzten Regel; jeder Treffer wird nach außen geführt, es sei denn, die Option NACH ERSTER ÜBEREINSTIMMUNG STOPPEN ist gesetzt. Sie verhindert dann alle weiteren Überprüfungen.

Die Behandlung von Nachrichtenfolgen betrifft Nachrichtensequenzen. Per Voreinstellung ändert der *switch*-Node nicht die Eigenschaft `msg.parts` für Nachrichten, die Teil einer Sequenz sind. Wenn Sie das entsprechende Kontrollkästchen aktivieren, können Sie für neue Nachrichtenfolgen eine passende Regel festlegen.

3.3.2 Der change-Node

Der *change*-Node bearbeitet Nachrichten oder Kontextvariablen. Abbildung 3.20 zeigt ein Beispiel, in dem die eingegebene Nachricht umfassend geändert wird.

Abbildung 3.20 Der change-Node verändert Nachrichten.

Der *inject*-Node speist ein JSON-Objekt ein, das ein Auto charakterisiert:

```
{
  "Typ":"Audi",
  "Zustand":"neu",
  "Preis":40000
}
```

Über die Regeln in den Node-Eigenschaften können Sie nun die Nachricht ändern. Dabei enthält jeder Listeneintrag eine Regel (siehe Abbildung 3.21).

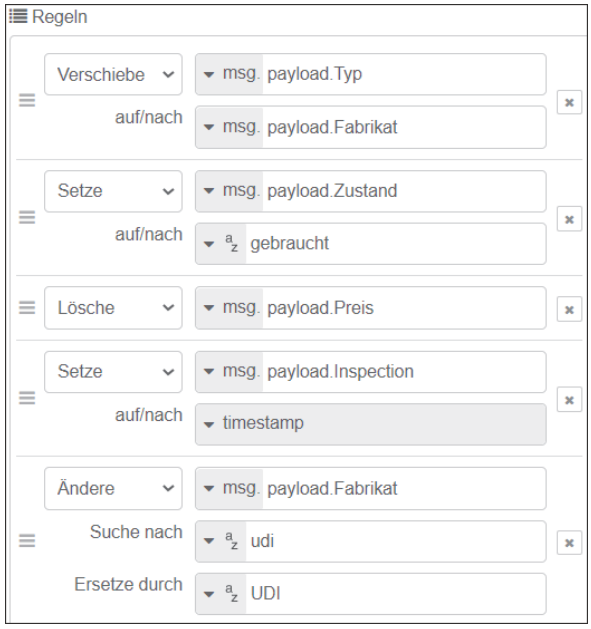


Abbildung 3.21 Die Eigenschaften des change-Nodes

Node-RED bietet vier Änderungsmöglichkeiten an:

- **Setze** – Legt einen neuen Wert fest (im Beispiel: setze Zustand). Der Wert selbst umfasst die gesamte mögliche Wertpalette (siehe Abschnitt 3.1.2).
- **Ändern** – Sucht und ändert Teile einer Nachricht.
- **Bewegen** – Ändert den Namen einer Nachrichteneigenschaft (im Beispiel: `msg.payload.Typ` auf `msg.payload.Fabrikat`).
- **Löschen** – Löscht eine Nachrichteneigenschaft.

Listeneinträge fügen Sie mit +HINZUFÜGEN hinzu; das Löschen eines Eintrags erfolgt über das ×. Abbildung 3.22 zeigt die dazu passende Debug-Ausgabe, nachdem die Werte des Autos geändert wurden.

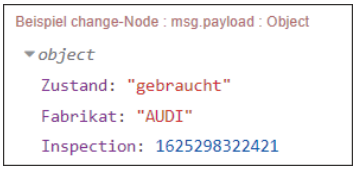


Abbildung 3.22 Die Debug-Ausgabe des change-Nodes

3.4 Der delay-Node

Kernaufgabe des *delay*-Nodes ist es, Nachrichten zu verzögern. Sie können ihn jedoch auch so mit Eigenschaften belegen, dass er den Nachrichtenfluss begrenzt (siehe Abbildung 3.23).

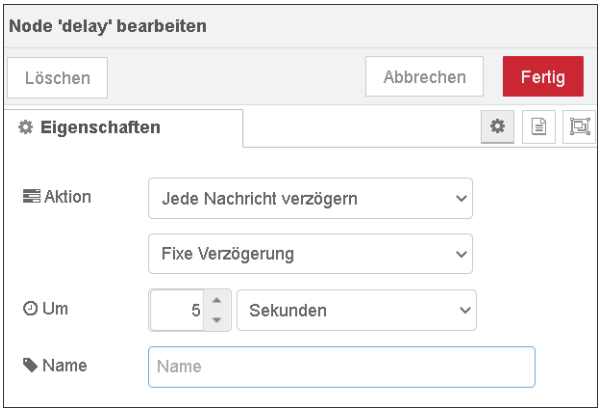


Abbildung 3.23 Die Eigenschaften des delay-Nodes

Die Eigenschaften gliedern sich in drei Gruppen. Näher möchte ich nur auf die Gruppe AKTION eingehen. Sie hat im »Urzustand« zwei Drop-Listen; sie ändert sich aber in Abhängigkeit von der getroffenen Auswahl.

- Jede Nachricht verzögern: Diese Option verzögert jede Nachricht um die in der zweiten Drop-down-Liste gewählte Methode: Die Methode kann sein:
  - eine fixe Verzögerung (um z. B. 45 Minuten)
  - eine zufällige Verzögerung (um z. B. 10 bis 20 Sekunden)
  - eine Verzögerung aus `msg.delay`
- Nachrichtenrate begrenzen: Dies erlaubt es, die Anzahl der weitergeleiteten Nachrichten zu begrenzen:
  - Alle Nachrichten
  - Für jedes msg.topic

  Sie können entscheiden, was mit den Nachrichten passiert, die unter die Begrenzung fallen:
  - Zwischennachricht in eine Nachrichtenschlange einreihen
  - Zwischennachricht löschen
  - Zwischennachricht an einen zweiten Ausgabe-Port leiten

3.5 Dateiformate konvertieren

Node-RED ist in den unterschiedlichsten Dateiformaten zu Hause. Das führt dazu, dass häufig Konvertierungen notwendig sind. Spezielle Nodes aus der Gruppe Parser (dt. »Zusammensetzer«) übernehmen diese Aufgabe (siehe Tabelle 3.2).

Node	Funktion
	Konvertiert bidirektional zwischen einem CSV-formatierten String und einer JavaScript-Objektdarstellung.
	Verwendet einen CSS-Selektor, um Elemente aus einem HTML-Dokument zu extrahieren.
	Konvertiert bidirektional zwischen einem JSON-String und seiner JavaScript-Objektdarstellung.
	Konvertiert bidirektional zwischen einem XML-String und seiner JavaScript-Objektdarstellung.
	Konvertiert bidirektional zwischen einem YAML-formatierten String und seiner JavaScript-Objektdarstellung.

Tabelle 3.2 Nodes für das Konvertieren von Dateiformaten

Das Prinzip ist bei allen Nodes ähnlich, deshalb zeige ich Ihnen an dieser Stelle anhand eines Beispiels nur eine Darstellung des *JSON*-Nodes (siehe Abbildung 3.24).

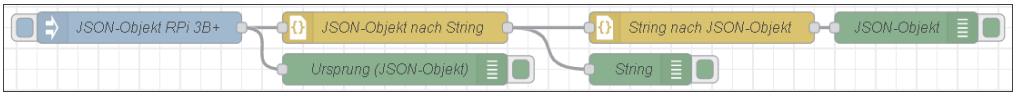


Abbildung 3.24 Ein Beispiel-Flow mit dem JSON-Node

Der *inject*-Node gibt ein JSON-Objekt über `msg.payload` ein. Der erste JSON-Node wandelt dieses in einen String um, und der zweite wandelt die Zeichenkette wieder zurück. Verschiedene *debug*-Nodes protokollieren die Prozessschritte. Werfen Sie einen Blick auf die Node-Eigenschaften aus Abbildung 3.25.

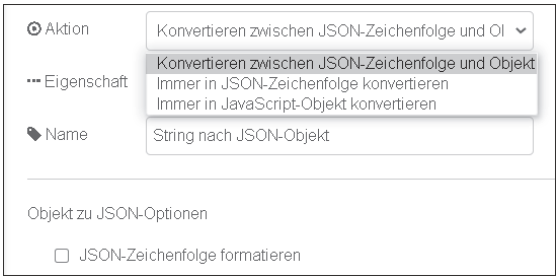


Abbildung 3.25 Die Eigenschaften des JSON-Nodes

Die drei wählbaren Konvertierungsmöglichkeiten sind selbsterklärend. Das Aktivieren des Kontrollkästchens JSON-ZEICHENFOLGE FORMATIEREN bewirkt, dass die Ausgabe Zeilenumbrüche enthält (siehe Abbildung 3.26). Dies ist hilfreich für Debug-Ausgaben in lesbarer Form, aber hinderlich für eventuell folgende Prozessschritte.

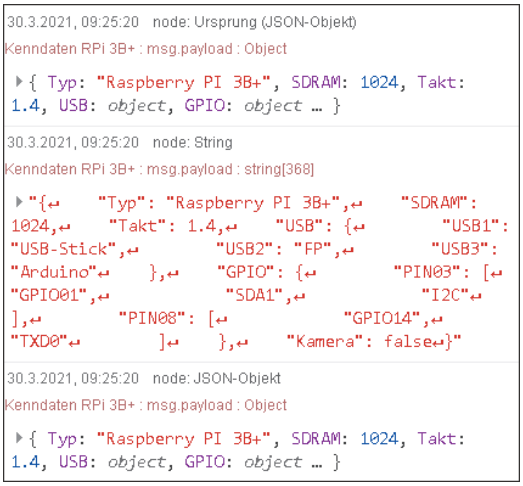


Abbildung 3.26 Debug-Ausgaben der JSON-Nodes

3.6 Auf Prozessereignisse reagieren

Häufig hängen Prozesse von der ordnungsgemäßen und vollständigen Erledigung vorheriger Schritte ab. Mit Node-RED reagieren Sie mit verschiedenen Nodes der Gruppe COMMON (siehe Tabelle 3.3) situationsbezogen auf solche Ereignisse.

Node	Funktion
	Gibt den Statusbericht eines Nodes aus.
	Wartet auf den Abschluss einer Bearbeitung.
	Sorgt für die Fehlerbehandlung.

Tabelle 3.3 Nodes für Prozessereignisse

3.6.1 Der status-Node

Der *status*-Node meldet Statuswechsel von Nodes desselben Node-RED-Editor-Tabs. Er kommt allerdings nur bei Nodes zum Zuge, die auch einen Statuswechsel verzeichnen (also beispielsweise bei *trigger*-Node, *delay*-Node oder *mqtt*-Nodes bei Verbindungsverlust). Abbildung 3.27 zeigt einen Beispiel-Flow.

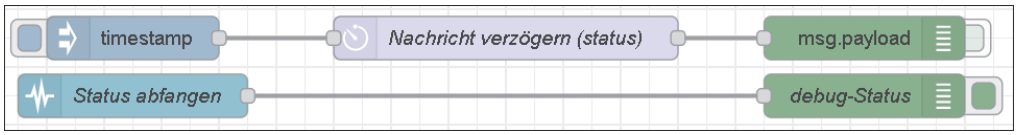


Abbildung 3.27 Der status-Node überwacht den Status.

Der *inject*-Node speist einen Zeitstempel ein, den der *delay*-Node verzögert weitergibt. In den Eigenschaften des *status*-Nodes ist der *delay*-Node als zu überwachender Node selektiert. In diesem Fall wird der *status*-Node zweimal angesprochen: beim Aktivieren der Verzögerung und bei der Rückkehr des *delay*-Nodes in den Ruhezustand (siehe Abbildung 3.28).

Der *status*-Node sendet eine Nachricht mit einem Objekt `msg.status`. Dieses enthält wiederum ein Objekt mit Informationen hinsichtlich des zu überwachenden Nodes.

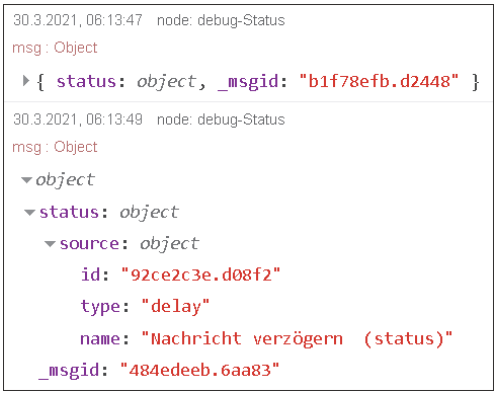


Abbildung 3.28 Die Debug-Ausgabe des status-Nodes

3.6.2 Der complete-Node

Der *complete*-Node reagiert auf die Beendigung eines Nodes. Er greift allerdings nicht bei allen Nodes, so z. B. nicht bei *trigger*- oder *delay*-Nodes. Der wichtigste Anwendungsfall ist die Überprüfung von Nodes ohne einen Ausgangsport wie z. B. dem *email-out*- oder dem *telegram-out*-Node. Für das Beispiel in Abbildung 3.29 genügt an dieser Stelle ein *debug*-Node.

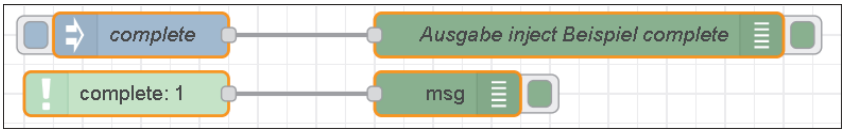


Abbildung 3.29 Der complete-Node reagiert, wenn sich ein anderer Node beendet.

Die Aufgabe des *inject*-Nodes in diesem Beispiel ist lediglich, eine Zeichenkette einzuspeisen. Wählen Sie in den Eigenschaften des *complete*-Nodes durch Aktivieren des Kontrollkästchens den entsprechenden Node aus (siehe Abbildung 3.30).

Abbildung 3.30 Die Eigenschaften des complete-Nodes

Die Debug-Ausgabe für den *complete*-Node zeigt das volle Nachrichtenobjekt an und bestätigt bei einem Inject, dass die Ausgabe des *debug*-Nodes beendet wurde (siehe Abbildung 3.31).

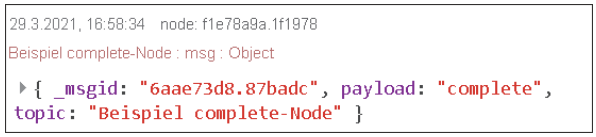


Abbildung 3.31 Die Debug-Ausgabe des complete-Nodes

3.6.3 Der catch-Node

Der *catch*-Node fängt Fehler von Nodes desselben Node-RED-Editor-Tabs ab. Üblicherweise hat ein Fehler bei der Verarbeitung einer Nachricht in einem Node zur Folge, dass der Flow angehalten wird. Mithilfe des *catch*-Nodes ist es möglich, eine eigens dafür vorgesehene Fehlerbehandlung durchzuführen. Die zu überwachenden Nodes lassen sich in den Eigenschaften des Nodes einstellen. Der Beispiel-Flow in Abbildung 3.32 verdeutlicht die Wirkungsweise.

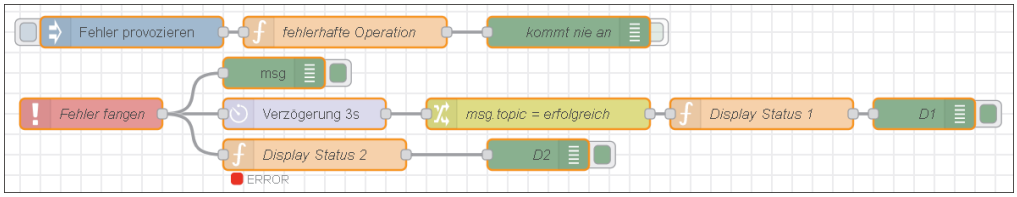


Abbildung 3.32 Beispiel-Flow für den catch-Node

Der *inject*-Node setzt lediglich den Flow in Gang. Der *function*-Node löst mit folgenden Anweisungen einen Fehler aus, weil die Variable `a` nicht definiert ist:

```
var n = 23 + 56 / a;
msg.payload = n;
return msg;
```

Der *catch*-Node fängt den Fehler ab und gibt umfangreiche Fehlerinformationen (siehe Abbildung 3.33) weiter. Sie enthalten alle Angaben, um eine zielgenaue Fehlerbehandlung vornehmen zu können. Neben den ursprünglichen Werten von `msg.payload` und `msg.topic` sind dies insbesondere die Hinweise in `msg.error`.

Der Beispiel-Flow zeigt eine Option der Fehlerbehandlung. Aus Gründen der Übersichtlichkeit gibt es zwei Verarbeitungsstränge: einen mit und einen ohne Fehlerkorrektur.

Der *change*-Node simuliert an dieser Stelle eine erfolgreiche Fehlerkorrektur, indem er `msg.topic` auf `"erfolgreich"` setzt.

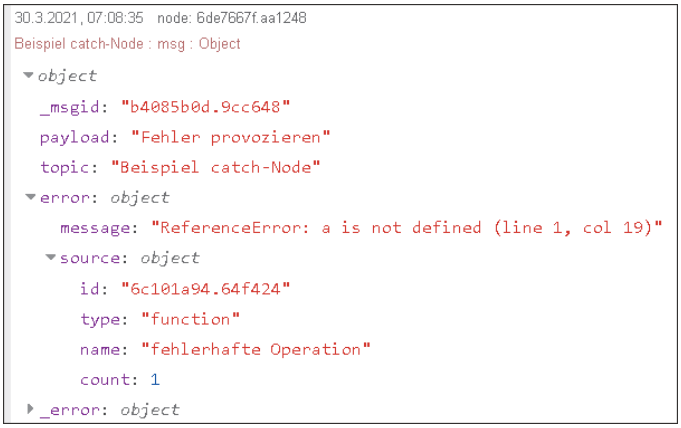


Abbildung 3.33 Fangen Sie Fehler mit dem catch-Node ab.

Der anschließende *function*-Node *Display Status 1* überprüft das Ergebnis und beendet den Flow, sodass der *debug*-Node *D1* nie erreicht wird. Der *function*-Node enthält folgende Anweisungen:

```
// Fehlerbehbung erfolgreich?
if (msg.topic == "erfolgreich") {
  // erfolgreich: Node-Status löschen, return ohne eine Nachricht auszugeben
  node.status({});
  return;
} else {
  // Fehlerbehandlung fehlgeschlagen; Node-Status und msg.topic
  // setzen sowie return Message
  node.status({fill:"red",shape:"dot",text:"ERROR"});
  msg.topic = "Logging flow";
}
return msg;
```

Der zweite Verarbeitungsstrang simuliert eine nicht erfolgreiche Fehlerbehandlung. Der *function*-Node *Display Status 2* ist inhaltlich identisch mit dem *function*-Node *Display Status 1*. Da er unmittelbar an den *catch*-Node anschließt, erhält er keine befreiende Erfolgsmeldung. Das Ergebnis ist dann eine Fehlermeldung unterhalb des *function*-Nodes und eine Weiterleitung der Fehlermessage an den *debug*-Node *D2*.

3.7 Sequenzen (Folgen)

Das war eine Menge Theorie bisher – es ist also Zeit für ein kleines praktisches Beispiel. Hierfür eignet sich das Thema Sequenzen hervorragend. Node-RED begreift

Sequenzen als geordnete Serien von Nachrichten, die auf eine bestimmte Art und Weise miteinander verbandelt sind.

Beispielsweise lässt sich eine Zahlenfolge in einer Nachricht abbilden:

```
payload : "22  77  33"
```

Oder jede Zahl ist Bestandteil einer Nachricht:

```
payload : "22"
payload : "77"
payload : "33"
```

Node-RED verfügt über einige leistungsstarke Nodes aus der Gruppe SEQUENCE, die die Arbeit massiv erleichtern (siehe Tabelle 3.4).

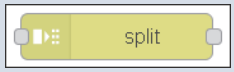
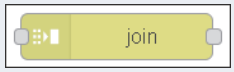
Node	Funktion
	Teilt eine Nachricht in eine Folge von Nachrichten.
	Verbindet Sequenzen von Nachrichten zu einer einzigen Nachricht.
	Sortiert `msg.payload` oder eine andere Sequenz.
	Erstellt Sequenzen von Nachrichten nach verschiedenen Regeln.

Tabelle 3.4 Nodes für die Bearbeitung von Sequenzen

Das folgende Beispiel zeigt das Zusammenspiel der unterschiedlichen Nodes, die Sie bisher in diesem Kapitel kennengelernt haben. Gleichzeitig demonstriert es die Möglichkeiten, Daten auszuwerten, zu verändern und wieder zusammenzuführen. Ausgangspunkt ist eine Übersicht über den Energieverbrauch eines Hauses. Sie enthält Angaben zu den einzelnen Zimmern, zum Jahr und zu dem verbrauchten Strom. Sie möchten nun wissen, wie viel Strom in den einzelnen Zimmern pro Quadratmeter verbraucht wurde. Diese Liste soll eine neue Spalte mit dieser Angabe erhalten.

Legen Sie dazu den Beispiel-Flow aus Abbildung 3.34 an.

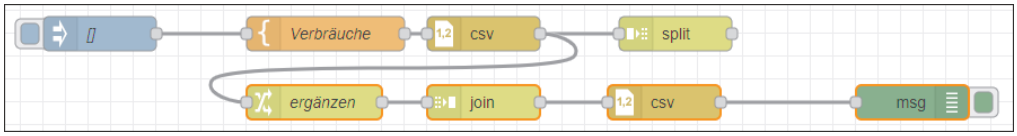


Abbildung 3.34 Der Beispiel-Flow für Sequenzen

Der *inject*-Node hat nur die Aufgabe, den Flow zu starten. Der *template*-Node enthält die Verbrauchsübersicht:

```
Zimmer,Jahr,Verbrauch
Arbeitszimmer,2019,4000
Schlafzimmer,2019,1000
Wohnzimmer,2019,3000
```

Die erste Zeile enthält die Spaltenüberschrift und strukturiert die Werte. Die einzelnen Angaben sind durch ein Kommata getrennt und liegen damit im CSV-Format vor (*Comma Separated Values*).

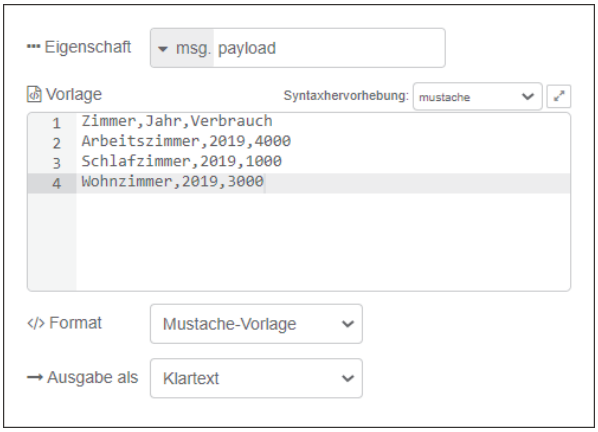


Abbildung 3.35 Die Eigenschaften des template-Nodes

Das Format ist eine *Mustache-Vorlage*. Auf Mustache gehe ich in Kapitel 11, »Dashboards für Fortgeschrittene«, ein.

Der anschließende *csv*-Node überführt die Zeilen von CSV in JSON-Objekte. Üblicherweise erkennt dieser Node eigenständig seine Aufgabe, und daher können in den allermeisten Fällen die Voreinstellungen unverändert bleiben. Da die Eingabe jedoch Spaltenüberschriften enthält, müssen Sie das entsprechende Kontrollkästchen aktivieren (siehe Abbildung 3.36).

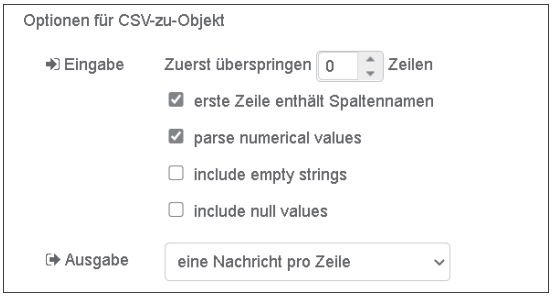


Abbildung 3.36 Aktivieren Sie die Spaltennamen in den Eigenschaften des csv-Nodes.

Abbildung 3.37 zeigt das Ergebnis der Umwandlung (je eine neue Nachricht pro Eintrag, also drei neue Nachrichten) anhand des Listeneintrags »Zimmer«.

Abbildung 3.37 Die Debug-Ausgabe des csv-Nodes

Ab hier trennen sich die Wege. Abzweigung 1 illustriert die Wirkungsweise des *split*-Nodes. Die Eigenschaften des Nodes bleiben an dieser Stelle unverändert. Die Ausgabe des *split*-Nodes besteht wiederum aus drei neuen Nachrichten (`msg.payload` `"Arbeitszimmer"` sowie `msg.payload` `2019` und `msg.payload` `4000`) pro empfangenen Eingang (siehe Abbildung 3.38).

Abbildung 3.38 Die Debug-Ausgabe des split-Nodes

Der *split*-Node begreift die Eingangsnachricht als Sequenz. Jede Message einer Sequenz hat eine Eigenschaft `msg.parts`. `msg.parts` ist ein Objekt mit weiteren Informationen:

- `msg.parts.id` – eine eindeutige ID für alle Elemente der Folge (hier `73597d2c.1e77c4` für alle drei neuen Nachrichten »Arbeitszimmer«, »2019« und »4000« der `msg.payload`)
- `msg.parts.index` – die Position der Nachricht innerhalb der Sequenz, beginnend bei 0
- `msg.parts.count` – die Gesamtzahl der zur Sequenz gehörigen Nachrichten

Der *split*-Node erstellt noch weitere Informationen:

- `msg.parts.type` – die Art der Nachricht (z. B. ob es ein String, Array, Object oder Buffer ist)
- `msg.parts.key` – der Name der Eigenschaft, aus der diese Nachricht erstellt wurde

Die letztgenannten Angaben helfen dem *join*-Node, die Nachrichtenfolge wieder in eine Nachricht zurückzuverwandeln.

Die zweite Abzweigung befasst sich mit der Ergänzung des Verbrauchs je Zimmer. Die Kernarbeit liegt hier bei dem *change*-Node, dessen Eigenschaften Sie in Abbildung 3.39 sehen.

Abbildung 3.39 Der change-Node berechnet den Verbrauch pro Quadratmeter.

Der Node muss zweierlei bewerkstelligen:

- die Spaltenüberschrift um einen neuen Wert `qmVerbrauch` ergänzen, der Ausdruck ist:

  ```
  msg.columns & ",qmVerbrauch"
  ```

- den entsprechenden Verbrauch pro Quadratmeter berechnen und eintragen:

  ```
  /* Flächentabelle  */
  (
    $flaechen := {
      "Arbeitszimmer": 20,
      "Schlafzimmer": 15,
      "Wohnzimmer": 30
  };
  msg.payload.Verbrauch/$lookup($flaechen, msg.payload.Zimmer)
  )
  ```

  Die Variable `flaechen` enthält die Zimmer mit ihrer Fläche in Quadratmeter (qm). Der Verbrauch aus `msg.payload.Verbrauch` wird durch die Quadratmeter und den Wert geteilt, den die JSONata-Funktion `lookup` aus der Liste sucht. Suchkriterium ist `msg.payload.Zimmer`. Das Ergebnis erhält `msg.payload.qmVerbrauch`.

Abbildung 3.40 zeigt die neue Nachricht.

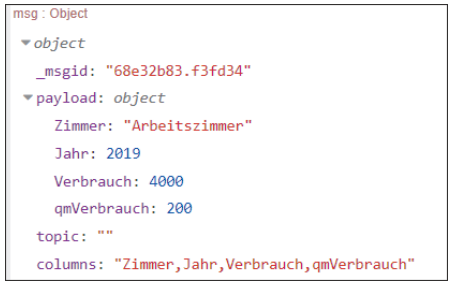


Abbildung 3.40 Die Debug-Ausgabe des change-Nodes zum »qmVerbrauch«

Berechnung des Verbrauchs

Auch – und vor allem – in der Programmierung führen viele Wege nach Rom. Die Verbrauchsberechnung kann auch durch einen *function*-Node erfolgen oder mit einem *change*-Node, in dem die Werte in einer Kontextvariablen hinterlegt sind. Beide Varianten sind in den Beispiel-Flows zum Download enthalten.

Der *join*-Node aus Abbildung 3.41 arbeitet automatisch und setzt die eingehenden Nachrichten wieder zu einer Gesamtnachricht zusammen.

Der abschließende *csv*-Node wandelt alles zurück in das ursprüngliche CSV-Format. Da die Ausgabe Spaltenüberschriften enthalten soll, müssen Sie dies so in den Node-Eigenschaften einstellen, wie Abbildung 3.42 zeigt.

Abbildung 3.41 Die Debug-Ausgabe des join-Nodes zum »qmVerbrauch«

Hinter der schlechten Übersetzung ZEILENNEUERZEILE verbirgt sich eine wichtige Option. Dort stellen Sie ein, wie ein Zeilenumbruch codiert werden soll. Die verschiedenen Betriebssysteme verarbeiten diese Information unterschiedlich, daher müssen Sie die passende Option wählen.

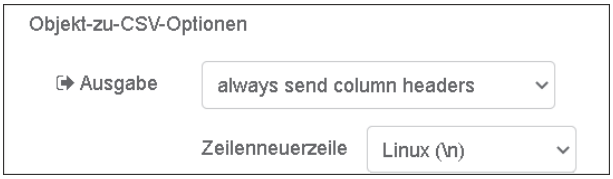


Abbildung 3.42 Die Optionen des csv-Nodes

Abbildung 3.43 zeigt das Ergebnis der Umwandlung.

Abbildung 3.43 Die Rückumwandlung durch den csv-Node

3.8 Fazit

Dieses Kapitel hat Ihnen gezeigt, dass Node-RED schon in der Grundkonfiguration überaus leistungsfähige Nodes bereitstellt. Die Handhabung ist einfach und intuitiv, sodass Sie auch ohne viel Erfahrung schnell einen gelungenen Einstieg finden. Mit ein wenig mehr Aufwand lassen sich auch komplexe Aufgaben schnell und nachvollziehbar umsetzen.

Node (*Dateien auswählen* bzw. *Ordner auswählen*) und einem *change*-Node (zum Sichern der Auswahl in eine *flow*-Kontextvariable).

Die beiden Stränge für das Löschen einer Datei bzw. eines Ordners sind ebenso weitgehend identisch. Sie beginnen mit einem *button*-Node (*Datei löschen* bzw. *Ordner* löschen). Es kommt dann zur Abfrage einer Löschbestätigung über einen *notification*-Node. Abhängig von der Antwort holt sich ein *function*-Node bzw. ein *change*-Node die getroffene Auswahl über die entsprechende Kontextvariable. Das Löschen selbst führt ein *file*-Node für Dateien bzw. ein *exec*-Node (Näheres dazu in Kapitel 9, »Node-RED-Hacks«) für Ordner mit dem Befehl `rmdir` durch.

Abbildung 8.20 zeigt das Dashboard.

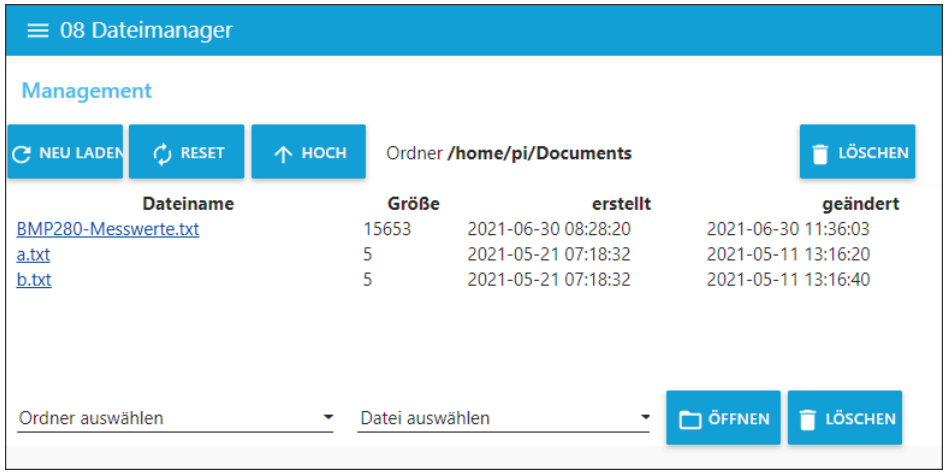


Abbildung 8.20 Der Dateimanager im Dashboard

8.3 Node-RED und InfluxDB

In Abschnitt 8.2.1 haben Sie die sequenzielle Speicherung von Daten kennengelernt – ein Datensatz nach dem anderen wird auf das Speichermedium geschrieben. Das ist zwar einfach, aber vielfach genügt es nicht den modernen Anforderungen an das Datenmanagement: Ein unkontrolliertes Anwachsen des Datenbestands ist eine Facette, komplizierte Methoden der Auswertung der Daten eine andere. Hier kommen Datenbanken ins Spiel.

Eine *Datenbank* ist eine organisierte Sammlung von strukturierten Informationen oder Daten, die üblicherweise elektronisch in einem Computersystem gespeichert sind. Die wesentliche Aufgabe ist, große Datenmengen effizient, widerspruchsfrei und dauerhaft zu speichern. Darüber hinaus soll eine Datenbank die in ihr enthaltenen Informationen in den angeforderten (Teil-)Mengen in unterschiedlichen, be-

darfsgerechten Darstellungsformen für Benutzer und Anwendungsprogramme bereitstellen. Eine Datenbank besteht aus zwei Teilen: aus der *Verwaltungssoftware* (dem sogenannten *Datenbankmanagementsystem*, DMS) und aus der *Datenbasis* (das ist die Datenbank im engeren Sinne).

InfluxDB (*https://www.influxdata.com*) ist ein solches Datenbanksystem, das speziell für Zeitreihen (engl. *Time Series*) entwickelt wurde. Sie können damit sehr gut Messwerte abspeichern, die über einen Zeitraum gesammelt werden.

InfluxDB 2.x

InfluxDB wird aktuell in zwei Hauptsträngen gepflegt: Die ältere Version 1.8 ist noch verfügbar, seit Ende 2020 gibt es darüber hinaus Version 2.x, die sehr rege weiterentwickelt wird – fast im Quartalsrhythmus erscheinen neue Releases. Die neue Version unterscheidet sich grundlegend von InfluxDB 1.x. Manche Neuerungen sind sehr begrüßenswert, wie beispielsweise ein brauchbares Webinterface, aber insgesamt ist die Datenbank wesentlich komplexer und professioneller geworden. Dazu zählt auch, dass zwingend ein 64-Bit-Betriebssystem Voraussetzung ist und dass InfluxDB nunmehr *flux* und nicht mehr *influxQL* als Query Language nutzt.

Weil InfluxDB 1.x noch sehr verbreitet und deutlich eingängiger ist, basieren die folgenden Beispiele auf dieser Version. Ein Umschwenken auf InfluxDB 2.x ist jederzeit möglich, ohne dass Sie die Flows in Ihrem Ablauf ändern müssen.

Sie installieren InfluxDB 2.x auf dem Raspberry Pi folgendermaßen:

1. Ergänzen Sie den InfluxDB-Schlüssel auf Ihrem System.

```
wget -q https://repos.influxdata.com/influxdata-archive_compat.key
echo '393e8779c89ac8d958f81f942f9ad7fb82a25e133faddaf92e15b16e6ac9ce4c
influxdata-archive_compat.key' | sha256sum -c && cat influxdata-archive_
compat.key | gpg --dearmor | sudo tee /etc/apt/trusted.gpg.d/influxdata-
archive_compat.gpg > /dev/null
```

2. Fügen Sie das InfluxDB-Repository den Paketquellen hinzu.

```
echo 'deb [signed-by=/etc/apt/trusted.gpg.d/influxdata-archive_com-
pat.gpg] https://repos.influxdata.com/debian stable main' | sudo tee
/etc/apt/sources.list.d/influxdata.list
```

3. Installieren Sie dann InfluxDB.

```
sudo apt-get update
sudo apt-get install influxdb2
```

Die Nodes des Pakets *node-red-contrib-influxdb* können auch InfluxDB-2.x-Datenbanken bedienen. Änderungen betreffen den *config*-Node (dort müssen Sie die Version und und den Pfad zum Zugangstoken angeben) sowie die Nodes der Palette (dort ändern Sie die Organization und hinterlegen eine flux-Query).

8.3.1 InfluxDB, eine Time Series Database

Was ist eine Time Series Database (TSDB)?

Zunächst sind *Zeitreihen* eine Folge von Datenpunkten (*Messungen*), die Daten über Zeiträume messen und in zeitlicher Reihenfolge abspeichern. Sie haben den Charakter einer Stichprobe. Ein derartiger Datensatz für Wetterdaten könnte folgenden Aufbau haben:

```
Zeitstempel              Temperatur        Luftdruck
1603989302806395143      20,4              1010
1603989861259578154      19,5              1000
1603989888207776368      22,2              1090
```

Die Datensätze einer TSDB weisen drei Gemeinsamkeiten auf:

- Jeder Dateneingang bildet grundsätzlich einen neuen Datenbankeintrag.
- Die Daten kommen normalerweise in zeitlicher Reihenfolge an.
- Die Zeit ist eine Hauptachse. Dabei können die Zeitintervalle regelmäßig oder unregelmäßig sein.

Jeder neue Datensatz wird an die bestehende Liste »angehängt«. Er markiert in dem Moment die aktuelle Situation; ein Update von Datensätzen erfolgt abgesehen von Fehlerkorrekturen oder einem Löschen nicht.

Auf diesem Ansatz, dass jede einzelne Änderung des Datenspeichers ein neu abgespeicherter Datensatz ist, beruht die Leistungsfähigkeit von Zeitreihendatenbanken. So können Sie auf einfache Weise Veränderungen messen und analysieren. Sie spüren Änderungen in der Vergangenheit präzise auf, überwachen Veränderungen in der Gegenwart oder prognostizieren sogar Entwicklungen in der Zukunft.

Allgemeine Anforderungen an das Datenbankmanagement einer TSDB

Der Ansatz einer TSDB stellt jedoch das Gesamtsystem vor bestimmte Herausforderungen, die das Datenbankmanagementsystem lösen muss. Das Speichern von Daten mit einer hohen Auflösung an Datenpunkten bringt natürlich ein offensichtliches Problem mit sich: Sie erhalten ziemlich schnell viele Daten. Eine Anforderung ist also, dass das DMS automatisiert die angesammelten Daten bereinigt und auf ein verträgliches Maß reduziert. Die InfluxDB realisiert das mit *Retention Policys* (dt. Aufbewahrungsrichtlinien).

Außerdem ist es ein Problem, eine große Datenmenge performant abzufragen. An dieser Stelle greift eine *Indizierung* einzelner Datenfelder.

Und schließlich wollen Sie unnötige Informationen, wie etwa eine Folge von Leerzeichen, Nullen oder nicht belegte Feldinhalte, erst gar nicht speichern. Dies besorgt die

Datenkompression. Sie ist wie das Indexmanagement Teil der *Storage Engine* (wörtlich: Speichermotor).

Detaillierte Informationen speziell zur InfluxDB-Version 1.8 (der Version, die hier zum Einsatz kommt) finden Sie unter:

https://docs.influxdata.com/influxdb/v1.8/concepts/storage_engine/

Datenbankkonzept und -aufbau

Dieser Abschnitt befasst sich mit dem Konzept und den Begriffen rund um die Datenspeicherung in einer InfluxDB. Abbildung 8.21 verdeutlicht schematisch die Struktur einer InfluxDB und gibt Ihnen einen Überblick über die Zusammenhänge.

Das InfluxDB-Dateisystem beherbergt eine oder mehrere Datenbanken. Diese enthalten dann zumindest ein *Measurement* (dt. Messung). Damit ist nicht ein einzelner Wert gemeint, sondern eine zeitliche Abfolge von thematisch gebündelten Messungen. Ein Reiheneintrag weist Feld-Wert-Paare auf (*Tag-Keys* und *Field-Keys*), die mit einem Zeitstempel (*Timestamp*) versehen sind. Die Tag-Key-Felder sind indiziert, Field-Key-Felder nicht – was natürlich auch sinnvoll ist, denn Sie werden ja im Allgemeinen nach den Namen filtern und suchen wollen und nicht nach den Werten. Alle Tag-Keys bilden das *Tag-Set*, alle Field-Keys das *Field-Set*.

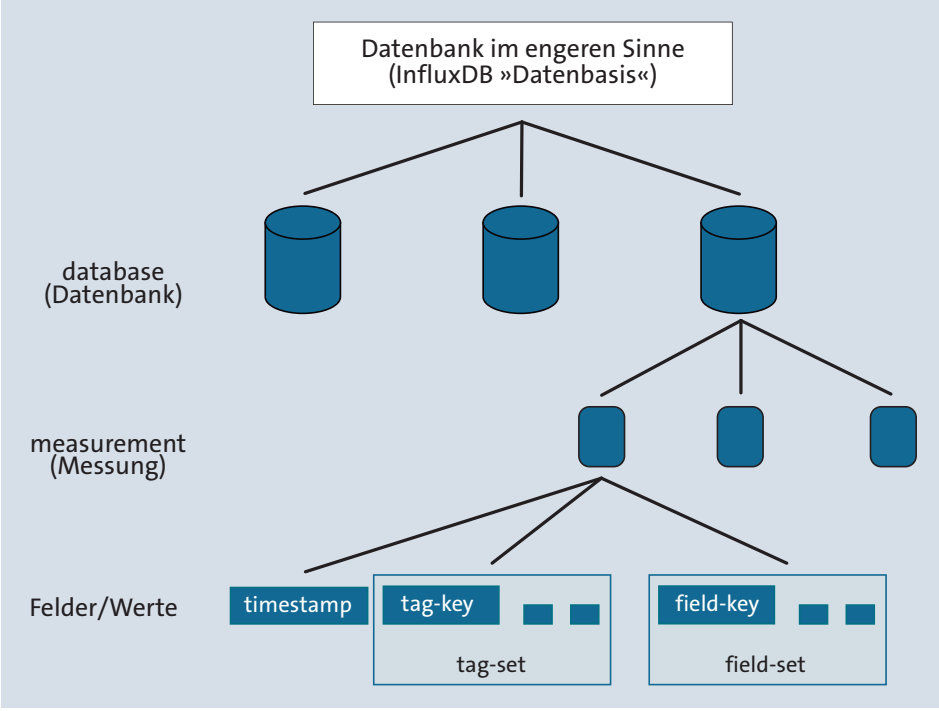


Abbildung 8.21 Speicherstruktur in InfluxDB

8.3.2 InfluxDB installieren

InfluxDB lässt sich auf den gängigen Betriebssystemen installieren. Der folgende Abschnitt zeigt die Installation auf einem Linux-System, genauer gesagt auf einem Raspberry Pi. Sie finden Anleitungen für

- Windows,
- macOS,
- Linux und Docker-Systeme

unter *https://docs.influxdata.com/influxdb/v2.0/install*.

Die aktuelle Version von InfluxDB (Stand: Herbst 2023) ist 3.0. Allerdings ist diese Version bisher nicht als Paket für den Raspberry Pi verfügbar. Befolgen Sie für eine Installation der verfügbaren Version 1.8 die folgenden Schritte.

Schritt 1: Den InfluxDB-Repository-Schlüssel hinzufügen

Fügen Sie im ersten Schritt den InfluxDB-Repository-Schlüssel Ihrem Raspberry Pi hinzu. Durch Hinzufügen des Schlüssels kann der Paketmanager des Raspberry Pi OS das Repository durchsuchen und die Pakete von dort installieren und aktualisieren.

Der Paketmanager `apt` verwendet zur Authentifizierung von Paketen eine Liste von Schlüsseln (Schlüsselbund). Pakete, die mit diesen Schlüsseln authentifiziert werden können, gelten als vertrauenswürdig. Maßgeblich für die Verwaltung der Liste der Schlüssel ist der Befehl `apt-key`.

Geben Sie in einer Konsole folgenden Befehl für das Herunterladen und Hinzufügen des InfluxDB-Schlüssels ein:

```
wget -qO- https://repos.influxdata.com/influxdb.key | \
  sudo apt-key add -
```

Löschen von Keys

Dieser Key bleibt im Schlüsselbund bestehen, auch wenn Sie InfluxDB nicht mehr benötigen und die Pakete mit z. B. `sudo apt remove` deinstallieren. Möchten Sie den Eintrag im Schlüsselbund löschen, besorgen Sie sich zunächst mit `sudo apt-key list` die Key-ID. Das Ergebnis könnte dann beispielsweise folgenden Eintrag haben:

```
pub rsa4096 2015-09-28 [SC]
  05CE 1508 5FC0 9D18 E99E FB22 684A 14CF 2582 E0C5
uid [ unbekannt ] InfluxDB Packaging Service <support@influxdb.com>
sub rsa4096 2015-09-28 [E]
```

Die Key-ID, die Sie für den Befehl `apt-key del` benötigen, besteht aus den letzten acht Zeichen der langen Hex-Zeichenkette (`2582 E0C5`), demnach also `sudo apt-key del 2582E0C5`.

Schritt 2: Das InfluxDB-Repository hinzufügen

Fügen Sie als Nächstes das passende Repository der Liste der Quellen hinzu. In dieser Liste steht, woher die Paketverwaltung neue Programme und Aktualisierungen bekommt. Die Quelle ist abhängig von der Version Ihres Raspberry Pi OS. Für die Raspberry-Pi-OS-Version *Buster* lautete der Aufruf:

```
echo "deb https://repos.influxdata.com/debian buster stable" | \
  sudo tee /etc/apt/sources.list.d/influxdb.list
```

Wenn Sie bereits die nächste Version des Betriebssystems nutzen, die den Namen *Bullseye* trägt, müssen Sie die Zeile entsprechend anpassen.

Schritt 3: Paketliste neu einlesen und InfluxDB installieren

Führen Sie mit `sudo apt update` ein Update der Paketliste durch. Der Befehl liest alle eingetragenen Paketquellen neu ein. Hierbei erfolgt auch eine Prüfung auf die Signatur der Paketlisten.

Installieren Sie InfluxDB nun mit:

```
sudo apt install influxdb
```

Schritt 4: InfluxDB starten

Starten Sie abschließend InfluxDB:

```
sudo systemctl start influxdb
```

Setzen Sie noch den nachstehenden Befehl ab, wenn Sie möchten, dass InfluxDB beim Booten startet:

```
sudo systemctl enable influxdb
```

Die Installation richtet ein paar Ordner und Dateien ein. Dies sind:

- */etc/influxdb/influxdb.conf* – die Konfigurationsdatei von InfluxDB
- */var/lib/influxdb/meta* – Metadaten (User, Datenbanken etc.)
- */var/lib/influxdb/data* – Dateien für die Zeitseriendaten
- */var/lib/influxdb/wal* – temporärer Cache für gerade geschriebene Datenpunkte

Haben Sie ein besonderes Augenmerk auf die Konfigurationsdatei. Neben einer Vielzahl von Einstellungen passen Sie dort auch den Speicherort der anderen Dateien an.

8.3.3 Die ersten Schritte mit InfluxDB

Prinzipiell ist Ihre InfluxDB nach der Installation betriebsbereit. Bedauerlicherweise verfügt InfluxDB nicht über ein GUI, um Anpassungen und dergleichen vorzunehmen. Stattdessen gibt es zwei Kommandozeilentools:

- `influx` ist ein Kommandozeileninterface (CLI) mit verschiedenen Kommandos zur Einrichtung der InfluxDB, mit dem Sie beispielsweise Datenbanken oder Benutzer

verwalten. Sie starten das Tool Influx mit dem Befehl `influx` und verlassen das Tool mit `exit` bzw. `quit`.

- `influxd` stellt eine Verbindung zum InfluxDB-Daemon her und hilft, Prozesse wie das Sichern und Zurückspielen zu organisieren.

Wenn Sie Messwerte verarbeiten wollen, müssen Sie zunächst eine Datenbank anlegen, in der Sie dann Datensätze einfügen und wieder abrufen können.

Schritt 1: Eine Datenbank anlegen

Starten Sie Influx zunächst über die Kommandozeile:

```
influx
```

In InfluxDB können Sie mehrere Datenbanken anlegen und ihnen eindeutige Namen zuweisen. Für eine Datenbank namens *BPM280DB* lautet die Anweisung:

```
CREATE DATABASE BMP280DB
```

Lassen Sie sich alle Datenbanken mit `SHOW DATABASES` anzeigen (siehe Abbildung 8.22); löschen Sie Datenbanken mit `DROP DATABASE <Datenbankname>` (z. B. `DROP DATABASE "BMP280DB"`).

```
pi@raspberrypi:~ $ influx
Connected to http://localhost:8086 version 1.8.6
InfluxDB shell version: 1.8.6
> CREATE DATABASE BMP280DB
> SHOW DATABASES
name: databases
name
----
_internal
BMP280DB
>
```

Abbildung 8.22 Eine Datenbank in InfluxDB anlegen

Schritt 2: Im CLI eine Datenbank zuweisen

Das CLI benötigt die Information, auf welche Datenbank sich Ihre Anweisungen beziehen. Nutzen Sie die Anweisung `USE`, um eine Datenbank bereitzustellen:

```
USE BMP280DB
```

Schritt 3: Daten einfügen

Einzufügende Daten müssen dem Aufbau der InfluxDB entsprechen. Sie bestehen daher aus Measurement, Tag(s), Field(s) und Points. Das Grundformat eines InfluxDB-Datenpunkts ist daher:

```
<measurement>[,<tag-key>=<tag-value>...] <field-key>=<field-value>[,<field2-
key>=<field2-value>...] [unix-nano-timestamp]
```

Tags sind optional. Ein nicht vorhandenes Komma zwischen optionalen Tags und Fields lässt Influx wissen, wann die Auflistung der Fields beginnt. Optional ist auch

der `unix-time-stamp`. Fehlt er, setzt das DMS automatisch die Systemzeit ein. Sie müssen also nur dann einen Wert angeben, wenn Sie auf den Wert eines anderen Zeitstempels Bezug nehmen möchten.

Unix-Timestamp

In Kapitel 5, »Funktionen programmieren«, sind Sie bereits dem Date-Objekt der Unix-Zeit begegnet. Die Entwicklung erfolgte für das Betriebssystem Unix, zwischenzeitlich ist sie als POSIX-Standard festgelegt worden. Die *Unix-Zeit* zählt die vergangenen Sekunden seit dem 01.01.1970, 00:00 Uhr UTC. Dieses Datum trägt auch die Bezeichnung *The Epoch* (dt. »die Epoche«). Das führt bei einer 32-Bit-Speicherung zu einem Problem: Die Anzahl möglicher Sekunden seit dem Start beträgt maximal 2.147.483.647 (2^{31}–1). Dieser Maximalwert ist am 19. Januar 2038 um 03:14:07 Uhr UTC erreicht. Bis dahin müssen alle Systeme, die dieses Zeitsystem nutzen, angepasst werden. Zum Glück ist bis dahin noch ein bisschen Zeit.

Für Node-RED finden Sie ergänzende Informationen in Abschnitt 9.5, »Zeitangaben formatieren«.

Sowohl Tag-Keys als auch deren Werte sind Zeichenketten. Hingegen können die Werte von Field-Keys Zeichenketten (z. B. »eine Zeichenkette«), Integer (z. B. 100i), Fließkommazahlen (z. B. 5 oder 7.8) oder boolesche Werte (z. B. `true` oder `false`) sein.

Das Einfügen von Datensätzen erfolgt mit dem Kommando `INSERT`.

Die nachfolgenden Beispiele beziehen sich auf BMP280-Daten, also auf die Temperatur- und Luftdruckmessung aus Kapitel 4. Das InfluxDB-Measurement soll den Namen `SensorDaten` tragen.

Die Beispielwerte werden zunächst »von Hand« eingetragen. In der Praxis wird es natürlich eher der Fall sein, dass die Werte automatisch in die Datenbank geschrieben werden. Dies ist Gegenstand des nächsten Abschnitts. Da Probleme mit unsauberen Datensätzen aber erfahrungsgemäß in vielen Projekten für Ärger sorgen, sollten Sie sich einmal genau ansehen, wie die Daten gespeichert werden:

- Fügen Sie einen Datensatz mit zwei Feldern (`Temperatur`, `Luftdruck`) ein. Alle Felder sind Field-Keys. Den Zeitstempel stellt das System bereit, das Messdatum ist also der Augenblick, in dem der Befehl abgeschickt wird.

  ```
  INSERT SensorDaten Temperatur=20.4,Luftdruck=1010
  ```

- Wenn Sie einen eigenen Zeitstempel angeben wollen, müssen Sie die Unix-Notation verwenden:

  ```
  INSERT SensorDaten Temperatur=20.4,Luftdruck=1010 1604332334517000000
  ```

  Die Auflösung (*Precision*) des Zeitstempels erfolgt per Default gemäß dem Standard der RFC3339 in Nanosekunden, die seit dem 01.01.1970 vergangen sind. Der

Wert 1604332334517000000 entspricht 2020-11-02T15:52:14.517000000, also dem 2. November 2020 um kurz vor 16 Uhr. Sie können dies beim Start von `influx` mit dem Parameter `-precision` ändern (z. B. `influx -precision ms` für Millisekunden). Die neue Einstellung gilt sowohl für die Ein- wie auch für die Ausgabe von Datensätzen.

- Fügen Sie einen Datensatz mit drei Feldern (`SensorID`, `Temperatur`, `Luftdruck`) ein. Das Feld `SensorID` ist ein Tag-Key, die Felder `Temperatur` und `Luftdruck` sind Field-Keys. Achten Sie auf die Kommata, da geht häufig etwas schief! Den Zeitstempel stellt das System bereit.

  ```
  INSERT SensorDaten,SensorID=1 Temperatur=20.4,Luftdruck=1010
  ```

  Sobald Tag-Keys definiert sind, sind Field-Keys mit alphanumerischem Inhalt nicht mehr möglich; sie müssen dann als `tag.key` definiert sein. Erlaubt sind nur noch numerische und boolesche Werte.

Das DMS der InfluxDB legt Messungen, Tag-Keys und Field-Keys anders als bei SQL automatisch an, wenn sie nicht existieren. Dies ist einerseits angenehm, andererseits ist damit aber der große Nachteil verbunden, dass syntaktische oder namentliche Ungenauigkeiten schnell zu unerwünschten Datenmodellen führen.

Lassen Sie sich in Zweifelsfällen die Art der Keys anzeigen:

- Tag-Keys – `SHOW Tag Keys`
- Field-Keys – `SHOW Field Keys`

Das Management der Messungen erfolgt mit:

- Messungen anzeigen – `SHOW measurements`
- Messung löschen mit z. B. – `DROP measurement SensorDaten`

Schritt 4: Datenbank auswerten

Für Datenbankabfragen stehen zwei Tools bereit:

- Influx Query Language (*InfluxQL*)
- *Flux*

Flux ist eine funktionale Skriptsprache zur Abfrage, Analyse und Verarbeitung von Daten in Zeitreihen. Es nutzt die Funktionalität von InfluxQL und des *TICK*-Skripts. Das ist ein ganzer Stapel aus den Influx-Softwarekomponenten *Telegraf*, *InfluxDB*, *Chronograf* und *Kapacitor* (zusammen eben als TICK abgekürzt), der diese Komponenten zu einem System mit einer einheitlichen Syntax kombiniert.

Flux bildet somit eine Alternative zu InfluxQL. Das zugrunde liegende funktionale Sprachmuster ist äußerst leistungsfähig und flexibel. Es überwindet viele der Einschränkungen von InfluxQL und ermöglicht Ihnen das Arbeiten mit dem Kombinieren von Daten, den sogenannten *Joins*, das Sortieren nach Tags, das Arbeiten mit

mehreren Datenquellen, die String-Manipulation und die Datenformung sowie vieles andere mehr. Wenn Sie professionell mit InfluxDB arbeiten wollen und große Datenmengen damit verarbeitet werden müssen, ist Flux wohl die bessere Alternative. Mit Flux steigt die Komplexität jedoch deutlich an, und es gibt sehr viel mehr Komponenten, in die Sie sich einarbeiten müssen und die gewartet werden wollen. Für kleinere Projekte sollten Sie es daher möglichst einfach halten und die Datenabfrage mit InfluxQL vorziehen.

Die Grundform der SELECT-Anweisung

Datenbankabfragen erfolgen mit der `SELECT`-Anweisung. InfluxDB folgt dabei der allgemein verbreiteten SQL-Syntax:

```
SELECT <field_key>[,<field_key>,<tag_key>] FROM <measurement_name>,
[,<measurement_name>]
```

Dabei können unterschiedliche Formate zur Anwendung kommen. So ergibt ein * als Auswahlkriterium eine Liste aller Einträge, also etwa `SELECT * FROM SensorDaten` wie in Abbildung 8.23.

```
> use BMP280DB
Using database BMP280DB
> INSERT SensorDaten Temperatur=20.4,Luftdruck=1010
> INSERT SensorDaten Temperatur=21.0,Luftdruck=990
> INSERT SensorDaten Temperatur=22.2,Luftdruck=1022
> select * from SensorDaten
name: SensorDaten
time                Luftdruck Temperatur
----                --------- ----------
1622007442091079444 1010      20.4
1622007468009649793 990       21
1622007495147041734 1022      22.2
>
```

Abbildung 8.23 Eine SELECT-Anweisung in InfluxDB

Auch lassen sich einfache mathematische Operationen in die `SELECT`-Klausel einfügen (`SELECT "Luftdruck"*1000,"Temperatur" FROM SensorDaten`). Das Ergebnis ist dann eine Liste wie in Abbildung 8.23, nur mit Luftdruckwerten in Pascal anstelle von Hektopascal. Weitere Möglichkeiten sind Funktionen (z. B. eine Durchschnittsberechnung mit der Funktion `MEAN`), Feldformatumformungen (das sogenannte *Casten*) und der Einsatz von regulären Ausdrücken.

Die SELECT-Anweisung mit der WHERE-Klausel

Selbstverständlich ist es auch möglich, mit einer `WHERE`-Klausel Datensätze zu selektieren. Die `WHERE`-Klausel ist optional. Die entsprechende Syntax ergänzt die Syntaxgrundform der `SELECT`-Anweisung.

```
SELECT_Klausel FROM_Klausel WHERE <Bedingung> [(AND|OR) <Bedingung> [...]]
```

So gibt folgende Anweisung nur noch drei Einträge aus (siehe Abbildung 8.24):

```
SELECT * FROM SensorDaten WHERE "Temperatur" > 20.5
```

```
> SELECT * FROM SensorDaten WHERE "Temperatur" > 20.5
name: SensorDaten
time                Luftdruck Temperatur
----                --------- ----------
1622007468009649793 990       21
1622007495147041734 1022      22.2
>
```

Abbildung 8.24 SELECT-Anweisung mit WHERE-Klausel in InfluxDB

Genau wie die `SELECT`-Klausel kann auch die `WHERE`-Klausel mathematische Ausdrücke enthalten (z. B. `WHERE "Temperatur"*2 > 44`, das Ergebnis hat nur noch einen Eintrag).

InfluxQL verfügt über eine Fülle von Funktionen und Schlüsselwörtern einschließlich der Unterstützung von regulären Ausdrücken im *Go-Stil*, mit denen Sie Ihre Daten sehr detailliert filtern können. Schauen Sie sich dazu diese Übersicht an:

https://docs.influxdata.com/influxdb/v1.8/query_language/spec

Benutzer einrichten und Datenbank absichern

Die Einrichtung von Benutzern ist spätestens dann wichtig, wenn der Zugang zur Datenbank mit einem Passwort geschützt werden muss. Rufen Sie also mit `influx` das CLI-Werkzeug auf und legen Sie mit `CREATE USER` einen Benutzer an, der über alle Rechte verfügt:

```
CREATE USER admin WITH PASSWORD 'raspberry' WITH ALL PRIVILEGES
```

Ein Anzeigen aller Nutzer erfolgt mit `show users`, das Löschen mit `drop user admin`.

Der Parameter `WITH ALL PRIVILEGES` weist dem Benutzer umfassende Rechte zu. Sie können natürlich mehreren Benutzern diese weitgehenden Privilegien zubilligen. Eingeschränkte Zugriffsrechte erteilen Sie, indem Sie einen Benutzer ohne diesen Parameter einrichten:

```
CREATE USER <Nutzername> WITH PASSWORD 'raspberry'
```

In einem zweiten Schritt teilen Sie der InfluxDB mit, welche Rechte dieser Nutzer hat (z. B. das Recht für einen lesenden Zugriff):

```
GRANT READ ON <datenbankname> TO <Nutzernamen>
```

`<datenbankname>` ist dabei die Datenbank, für die Rechte eingeräumt werden sollen. Als Rechte sind Lesen (`READ`), Schreiben (`WRITE`) oder beides (`ALL`) möglich.

Mit `REVOKE` entziehen Sie Berechtigungen, mit `SHOW GRANTS FOR <user>` sehen Sie die einem Nutzer erteilten Berechtigungen.

Mit einer Änderung der Konfigurationsdatei setzen Sie die Benutzerauthentifikation in Kraft. Suchen Sie den Bereich für das `http-setup` und setzen Sie:

```
auth-enabled = true
```

Nach einem Restart ist der Zugangsschutz aktiviert. Das Übermitteln der Zugangsberechtigungen hängt dann vom jeweiligen Anwenderprogramm ab:

- Im CLI gibt es mehrere Möglichkeiten, z. B. direkt beim Start:

  ```
  influx -username admin -password raspberry
  ```

- In Node-RED ist ein Konfigurations-Node (siehe Abbildung 8.28) dafür zuständig.

8.3.4 Mit Node-RED Daten in InfluxDB speichern

Selbstverständlich ist es nicht sinnvoll, Messwerte per Hand in eine Datenbank einzutragen. Das soll an dieser Stelle Node-RED übernehmen.

Für die Anbindung einer InfluxDB ist das Modul *node-red-contrib-influxdb* ideal. Es installiert in der Gruppe SPEICHER drei neue Nodes nebst einem Konfigurations-Node.

Node-RED unterstützt diese Form der Datenspeicherung mit insgesamt drei Nodes (siehe Tabelle 8.2). Hinzu kommt ein Konfigurations-Node.

Node	Funktion
	Einfache Query-Abfragen gegen eine InfluxDB.
	Speichert Daten in eine InfluxDB.
	Ermöglicht das Schreiben mehrerer Datenpunkte zu mehreren Measurements.

Tabelle 8.2 InfluxDB-Nodes

Der Flow für das Speichern von BMP280-Daten in eine InfluxDB ist nicht sonderlich komplex. Sie sehen ihn in Abbildung 8.25.

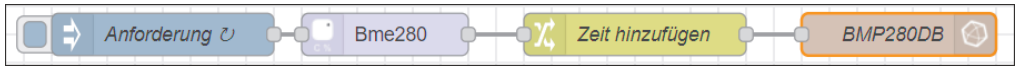


Abbildung 8.25 Flow für die Datenspeicherung in InfluxDB

Die ersten zwei Nodes sind hinsichtlich ihrer Aufgaben und ihrer Eigenschaften identisch mit dem Flow aus Abschnitt 8.2.1. Der *change*-Node erfährt die beiden Änderungen aus Abbildung 8.26.

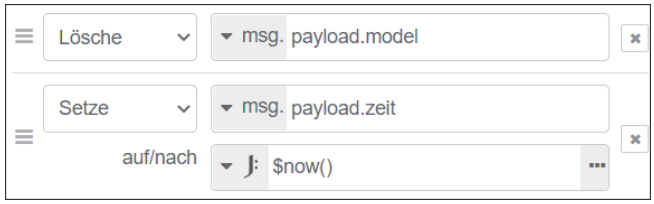


Abbildung 8.26 Der change-Node für die Datenspeicherung

Der Grund für die Änderung ist, dass in der gewählten Umgebung die Vergabe des Zeitstempels nicht durch InfluxDB erfolgen soll. Stattdessen soll für Testzwecke das Datum in lesbarer Form gespeichert werden.

Neu ist der *influxdb-out*-Node, dessen Eigenschaften Sie in Abbildung 8.27 sehen.

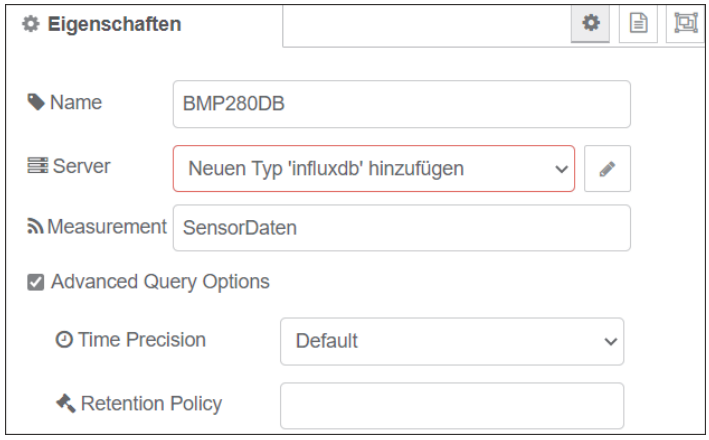


Abbildung 8.27 Eigenschaften des influxdb-out-Nodes

Neben dem NAMEN hat der Node die folgenden maßgeblichen Eigenschaften:

- SERVER – Dies ist der InfluxDB-Datenbankserver. Dessen Eigenschaften bestimmen Sie in einem Konfigurations-Node.
- MEASUREMENT ist die Bezeichnung der Messung. Sie wird angelegt, wenn noch keine Messung dieses Namens in der Datenbank existiert.
- Wenn Sie die Option ADVANCED QUERY OPTIONS aktivieren, erscheinen zwei zusätzliche Einstellungsmöglichkeiten:
  - TIME PRECISION bezieht sich auf die Auflösung des Zeitstempels.
  - RETENTION POLICY legt fest, wie mit alten Einträgen umgegangen wird. Darauf gehe ich im nächsten Abschnitt genauer ein.

Abbildung 8.28 zeigt die Eigenschaften des InfluxDB-Konfigurations-Nodes.

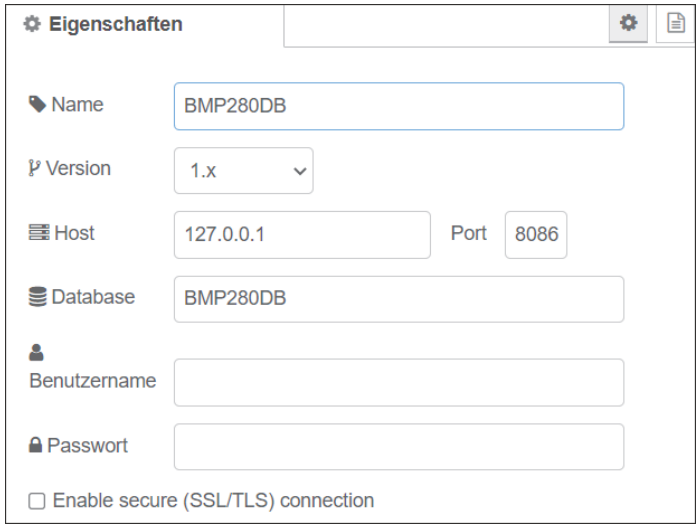


Abbildung 8.28 Eigenschaften des InfluxDB-Konfigurations-Nodes

Die Eigenschaften sind:

- NAME des InfluxDB-Datenbankservers.
- VERSION der InfluxDB-Software. Auf dem Raspberry Pi ist derzeit Version 1.8 aktuell, auf anderen Systemen kann es auch die Version 2.0 sein.
- Der HOST ist die IP-Adresse mit der Angabe des Ports des Geräts, auf dem der Datenbankserver gehostet ist. Die Angabe 127.0.0.1 ist identisch mit *localhost* und besagt, dass Node-RED und InfluxDB auf demselben Rechner laufen. InfluxDB lauscht standardmäßig an Port 8086. Wenn Sie InfluxDB auf einem anderen Rechner in Ihrem Netzwerk installieren, müssen Sie diesen Eintrag anpassen.
- DATABASE ist der Name der Datenbank.
- Die Felder BENUTZERNAME und PASSWORT bieten Ihnen die Möglichkeit, eine Authentifikation für den Zugriff auf InfluxDB zu hinterlegen. Betroffen sind die InfluxDB-Konfigurationsdatei und über das CLI einzurichtende Berechtigungen.
- Bei der Option ENABLE SECURE (SSL/TLS) CONNCECTION sind nur Angaben nötig, wenn der InfluxDB-Server mit SSL/TLS abgesichert ist.

Kontrollieren Sie mit der `SELECT`-Anweisung den Dateneingang in Ihrer InfluxDB.

8.3.5 Mit Node-RED Daten aus der InfluxDB auslesen

Das Auslesen von Datenbankeinträgen erfolgt mit dem *influxdb-in*-Node, dessen Flow Sie in Abbildung 8.29 sehen.

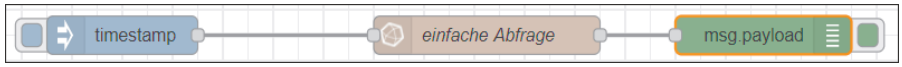


Abbildung 8.29 Die InfluxDB auslesen

Die Eigenschaften des *influxdb-in*-Nodes sehen Sie in Abbildung 8.30.

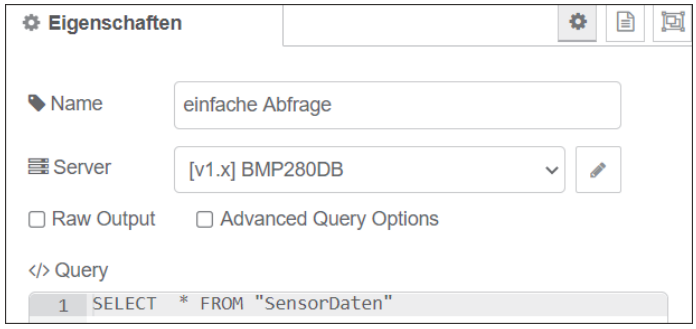


Abbildung 8.30 Eigenschaften des influxdb-in-Nodes

Hinsichtlich seiner Eigenschaften gilt:

- NAME ist die Node-Bezeichnung.
- Bei SERVER geben Sie den Host der InfluxDB an.
- Wichtig ist die Option RAW OUTPUT:

  Üblicherweise erfolgt die Ausgabe in `msg.payload` als eine Objekttabelle mit allen gefundenen Datenpunkten, so wie in Abbildung 8.31.

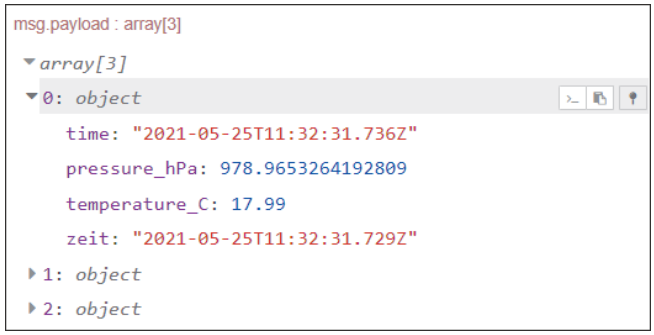


Abbildung 8.31 Die formatierte Debug-Ausgabe

  Ein aktiviertes Kontrollkästchen bewirkt eine *rohe*, d. h. unformatierte Ausgabe wie in Abbildung 8.32.

- Wenn Sie die Option ADVANCED QUERY OPTIONS aktivieren, haben Sie zusätzliche Optionen zur Time Precision und zur Retention Policy.

Abbildung 8.32 Die Debug-Ausgabe von »rohen« Daten

- QUERY nimmt die Suchanfrage auf, wie Sie sie auch im CLI `influxd` verwenden würden. Daher ist es vorteilhaft, wenn Sie die `SELECT`-Klausel vor ihrer Übernahme in Node-RED zunächst im CLI testen. Wenn dort die gewünschten Ergebnisse zurückgeliefert werden, bauen Sie die Abfrage in Node-RED ein.

Diese Grundversion der Datengewinnung führt in der Praxis aller Voraussicht nach zu Schwierigkeiten, da es zu viele Datenpunkte gibt. Sie sollten also einen Filter vorsehen, mit dem Sie die Daten vorbereiten. Dazu können Sie unterschiedliche SQL-Ausdrücke verwenden:

- **SELECT mit WHERE-Klausel**
  Setzen Sie die `WHERE`-Klausel ein, um beispielsweise den Zeitraum einzugrenzen (`SELECT * FROM "SensorDaten" WHERE timestamp > 1621719686598`). In einem solchen Fall kann es hilfreich sein, die Zeitangabe als String in der Datenbank abzuspeichern.
- **SELECT mit LIMIT-Klausel**
  Die `LIMIT`-Klausel begrenzt die Rückgabe auf eine bestimmte Anzahl der zuerst gefundenen Einträge. `SELECT * FROM "SensorDaten" LIMIT 4` gibt nur die ersten vier Treffer aus.

- **Komplexe Datenbankabfragen**
  Formulieren Sie eine komplexere Datenbankabfrage, zum Beispiel:

  ```
  SELECT mean("temperature_C") AS "mean_Temperatur" FROM "SensorDaten" WHERE
  time > now() - 12h GROUP BY time(30m) FILL(null)
  ```

  Diese Anweisung selektiert mit der Anweisung `mean("temperature_C")` den Durchschnitt der Temperaturmessungen und benennt den Field-Key `temperature_C` in `mean_Temperatur` für die Ausgabe um. Die Selektion ist auf einen Zeitraum begrenzt, der vom aktuellen Zeitpunkt 12 Stunden in die Vergangenheit reicht. Die Bildung des Durchschnitts erfolgt für jeweils eine Gruppe, die eine Zeitspanne von 30 Minuten umfasst.

Abbildung 8.33 zeigt einen Beispiel-Flow für die Datenabfrage:

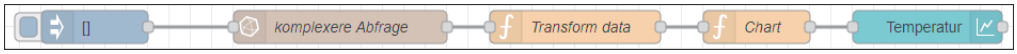


Abbildung 8.33 Eine komplexe Datenabfrage aus InfluxDB

Das Feld `Query` *des influxdb-in*-Nodes erhält nun die genannte Abfrage.

Der erste *function*-Node bereitet `msg.data` aus der empfangenen `msg.payload` auf:

```
let data = [];
for (let element of msg.payload) {
    data.push({"x":element.time, "y":element.mean_Temperatur});
}
msg.data = data;
return msg;
```

Ein zweiter *function*-Node stellt ergänzende Informationen bereit, damit alles ein bisschen übersichtlicher wird:

```
msg.topic = "Data set 1";
series = [];
data = [];
labels = [];
series.push("Field1");
labels.push("Field2");
data.push(msg.data);
msg.payload = [{"series":series, "data":data,"labels": labels}];
return msg;
```

Das Dashboard aus Abbildung 8.34 zeigt dann ein Liniendiagramm mit den über 30 Minuten gemittelten Temperaturen.

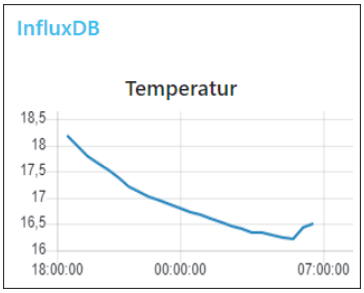


Abbildung 8.34 Die Dashboard-Ausgabe der gemittelten Temperaturen

Eine Abfrage auf einen aggregierten Datenbestand

Mit einfachen Mitteln lässt sich der Datenbestand zusammenfassen. Das Mittel der Wahl sind *Continuous Querys* (CQ).

CQs sind InfluxQL-Abfragen, die automatisch und regelmäßig für Echtzeitdaten ausgeführt werden und Abfrageergebnisse in einer bestimmten Messung speichern. Die Syntax ist:

```
CREATE CONTINUOUS QUERY <cq_name> ON <database_name>
BEGIN
  <cq_query>
END
```

`cq_query` benötigt eine Funktion (eine `SELECT`-Klausel), eine `INTO`-Klausel und eine `GROUP-BY-time()`-Klausel:

Dieses Beispiel erstellt eine neue regelmäßige Abfrage `cq_SensorDaten`, die alle 30 Minuten den Temperaturdurchschnitt in eine neue Messung `DurchschnittTemperaturen` spielt. Sie erfasst jeweils alle Daten mit Zeitstempeln, die zwischen dem letzten Lauf und dem aktuellen Aufruf liegen:

```
CREATE CONTINUOUS QUERY "cq_SensorDaten" ON "BMP280DB" BEGIN SELECT
  mean("temperature_C") INTO DurchschnittTemperaturen FROM "SensorDaten"
  GROUP BY time(5m) END
```

Die Ausgabe sehen Sie in Abbildung 8.35.

Das Management der Continuous Querys erfolgt mit einigen Konsolenbefehlen:

- Continuous Querys anzeigen – `show continuous queries`

  Die Anweisung ist penibel: Sie zeigt die komplette Bezeichnung des Measurements an:

  `"<DBName>"."<retention_policy>"."<measurement>"`

- Continuous Querys löschen – `drop continuous query <Query-Name> on <Datenbankname>`

```
> CREATE CONTINUOUS QUERY "cq_SensorDaten" ON "BMP280DB" BEGIN SELECT mean("temperature_C"
) INTO DurchschnittTemperaturen FROM "SensorDaten" GROUP BY time(5m) END
> show continuous queries

name: _internal
name query
---- -----

name: BMP280DB
name           query
----           -----
cq_SensorDaten CREATE CONTINUOUS QUERY cq_SensorDaten ON BMP280DB BEGIN SELECT mean(temper
ature_C) INTO BMP280DB.autogen.DurchschnittTemperaturen FROM BMP280DB.autogen.SensorDaten
GROUP BY time(5m) END
>
```

Abbildung 8.35 Eine kontinuierliche Abfrage in InfluxDB

8.3.6 Die InfluxDB sauber halten

Im Laufe der Zeit sammeln sich unzählige Datensätze an. Um nicht unnötig Speicherplatz zu belegen und um den Überblick zu behalten, sollten Sie alte Datensätze löschen. Hier bieten sich verschiedene Verfahren an.

Zeitserien manuell löschen

Sie können Zeitserien manuell löschen. Unter Zeitserien sind hier Datenpunkte zu verstehen, die einer Messreihe zugeordnet sind. InfluxDB greift dabei zwei unterschiedliche Ansätze auf:

- **DROP SERIES**

  `DROP SERIES` löscht alle Punkte aus einer Datenreihe in einer Datenbank und löscht die Reihe aus dem Index. Die Syntax ist:

  `DROP SERIES FROM <Messung_name[,Messung_name]> WHERE <tag_key>='<tag_value>'`

  Fehlt die `FROM`-Klausel, bearbeitet der Befehl alle Messreihen der gesamten InfluxDB! Dort sollten Sie also unbedingt eine Einschränkung eingeben.

  Sie löschen z. B. alle Zeitserien der Messreihe `SensorDaten` mit `DROP SERIES FROM SensorDaten`. Einschränkungen mit der `WHERE`-Klausel sind möglich.

- **DELETE**

  Die Anweisung `DELETE` löscht alle Punkte aus einer Reihe in einer Datenbank. Im Gegensatz zur `DROP`-Anweisung wird die Serie nicht aus dem Index entfernt. Sie erlaubt aber im Gegensatz zur `DROP SERIES`-Anweisung den Zeitstempel als Auswahlkriterium.

Wenn Sie Messungen automatisiert löschen möchten, können Sie dies elegant mit Retention Policys und Shards erledigen.

Retention Policys und Shards

Retention Policys (dt. Aufbewahrungsrichtlinien) und *Shards* (dt. Scherben) sind eine feine Möglichkeit, den InfluxDB-Datenbestand im Zaum zu halten.

- **Retention Policys**
  Aufbewahrungsrichtlinien bieten im Hinblick auf einen »ballastfreien« Datenbestand eine einfache und effektive Möglichkeit, alte und überholte Daten zu verwerfen. Daten erhalten somit eine Art »Ablaufdatum«. Der automatische Löschvorgang setzt ein, sobald dieses Datum erreicht ist. InfluxDB löscht hier aber nicht nur jeweils einen Datenpunkt, sondern es entfernt eine ganze Shard Group.
- **Shard Group**
  Eine *Shard Group* ist ein Container für Shards, die wiederum die tatsächlichen Zeitreihendaten enthalten. Jede Shard Group hat eine entsprechende Aufbewahrungsrichtlinie, und alle Shards innerhalb einer einzelnen Shard Group halten sich an dieselbe Aufbewahrungsrichtlinie. Zusätzlich hat jede Shard Group eine Shard-Gruppendauer; sie bestimmt das Zeitfenster (das Zeitintervall) einer Shard Group. Das Zeitintervall kann bei der Konfiguration der Aufbewahrungsrichtlinie angegeben werden. Wenn nichts angegeben ist, beträgt die Standarddauer der Shard Group standardmäßig sieben Tage.
- **Shards**
  Das schiere Volumen an Zeitreihendaten einer typischen Zeitreihendatenbank mit diversen Speichermodellen und Abfragen erfordert einen alternativen Ansatz zur Kategorisierung dieser Daten: *Shards*. Shards sind ideale Container für Zeitreihendaten. Das Zusammenfassen der Daten in Shards (das sogenannte *Sharding*) in InfluxDB ermöglicht einen hoch skalierbaren Ansatz zur Steigerung des Durchsatzes und der Gesamtleistung. Es entfaltet seine Wirkung speziell bei rasch wachsenden Datenmengen in Zeitreihendatenbanken.

Wenn Sie in InfluxDB eine Datenbank erstellen, wird automatisch eine Standardaufbewahrungsrichtlinie für diese Datenbank angelegt. Diese Richtlinie trägt den Namen »autogen«. Die Standardrichtlinie ist unendlich, d. h., es erfolgt kein Löschen. Sie können diesen Wert aber bereits bei der Einrichtung mitgeben:

```
CREATE DATABASE <database_name> DURATION <duration> WITH REPLICATION <n>
  SHARD DURATION <duration> NAME <name>
```

Die Dauer der Shard Groups hängt von der Dauer der Aufbewahrungsrichtlinie ab, sofern keine anderen Einstellungen getroffen werden (siehe Tabelle 8.3).