



# Digitaler Retrofit

## Datenerfassung an Bestandsanlagen

Prof. Dr. Stefan Berlik

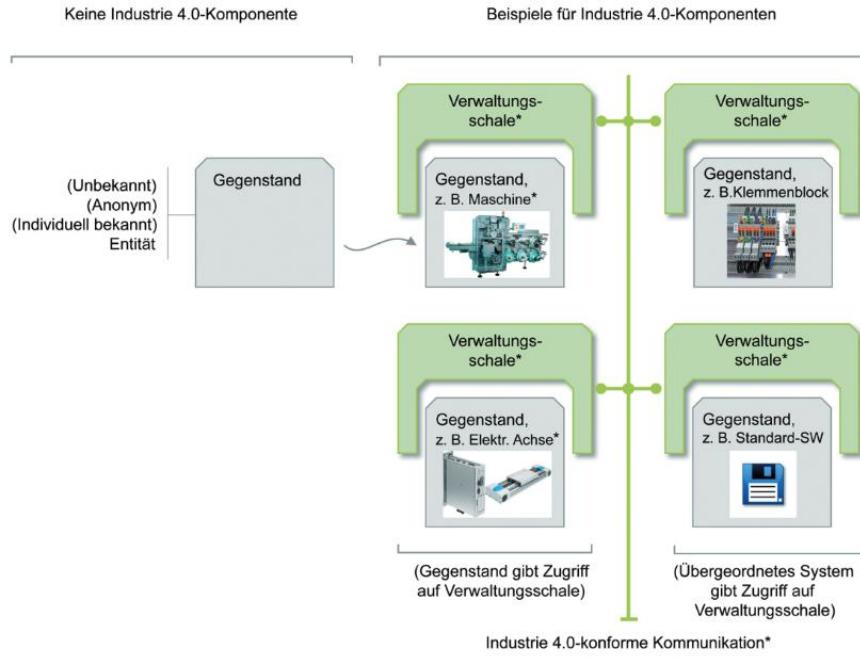
**FH Bielefeld**  
University of  
Applied Sciences

***„Wer neue Antworten will,  
muss neue Fragen stellen.“***

Johann Wolfgang von Goethe



# Referenzarchitekturmodell Industrie 4.0: Verwaltungsschale



- Virtuelle Repräsentation eines zugrundeliegenden Gegenstands; mit
  - Manifest
  - Komponentenmanager
- Mögliche Submodelle für z. B.
  - Identifikation
  - Kommunikation
  - Engineering
  - Konfiguration
  - Safety / Security
  - Lifecycle Status
  - Energy Efficiency
  - Condition Monitoring, ...

Quelle: Plattform Industrie 4.0

\*Schnittstelle/ Datenformate Industrie 4.0-konform ausgeführt

# I4.0 Verwaltungsschalte per Edge-Device

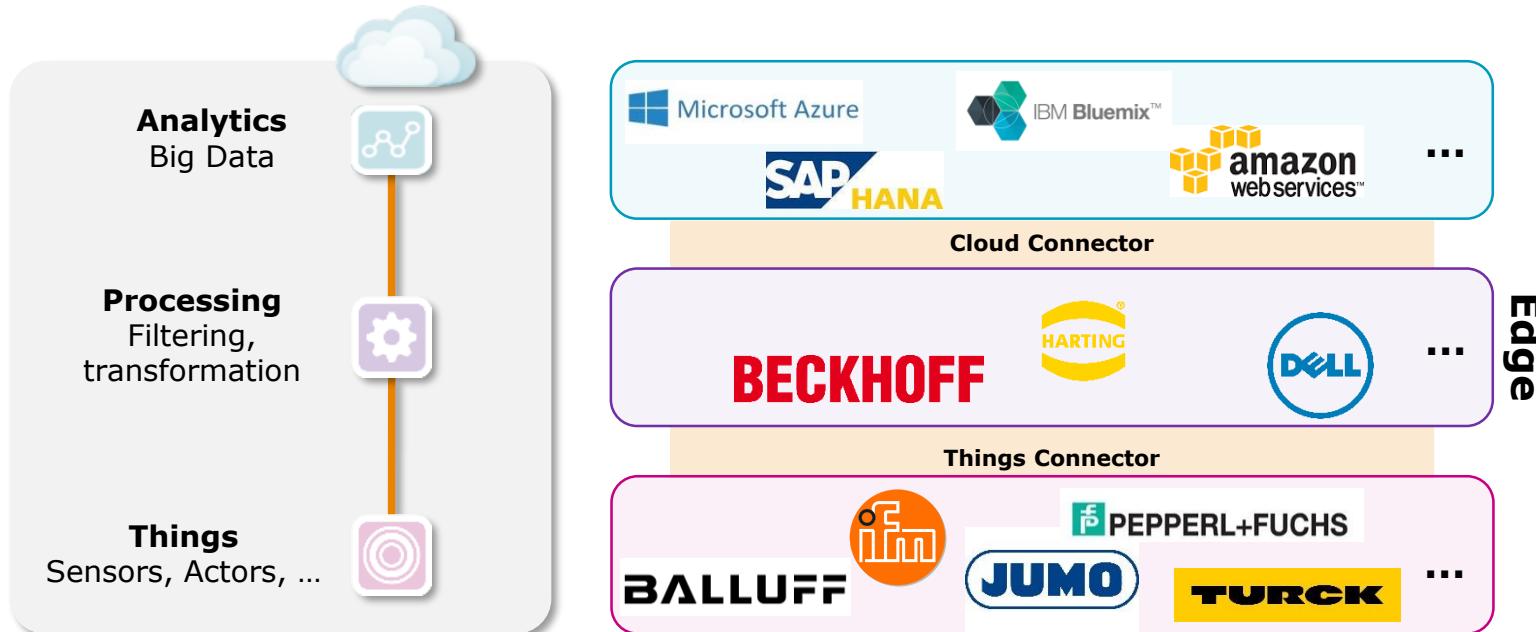


# Edge Devices



[Quelle: Maarten van den Heuvel on Unsplash]

# Industrial Internet of Things



# Beispielhaftes Edge Device: HARTING MICA

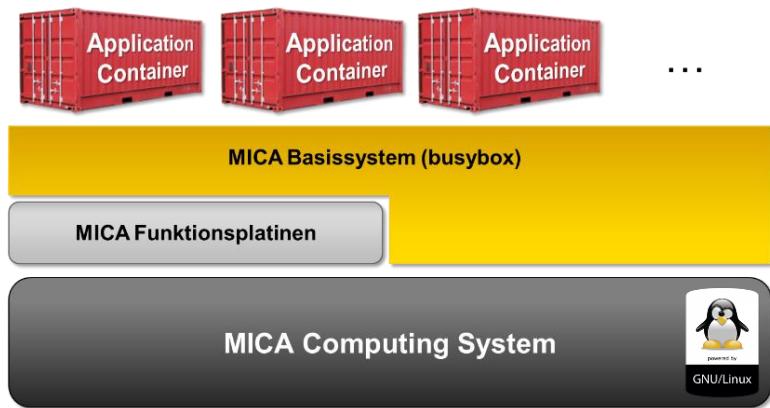
## Technische Daten

- 1,33 GHz Dual Core ARM Prozessor
- 2 GB RAM, 16 GB eMMC Flash Speicher
- 8x GPIO bis 24 V
- Diverse Varianten
  - RFID
  - USB
  - IO-Link
  - ...
- 24 V oder Power over Ethernet (PoE) Stromversorgung
- Robuste Industriesteckverbindungen (M8, M12, TNC...)
- 132 mm x 86 mm x 35 mm
- IP 67



# Softwarearchitektur

- Nutzt offene Standards
  - Linux, LXC, HTML5, ...
  - Keine Lizenzgebühren  
(GNU General Public License)
- Gesamtes Linux Ökosystem nutzbar
- Breites Spektrum an add-on Paketen
- Freie Wahl der Programmiersprache und Entwicklungsumgebung
- Applikationsentwicklung wird in Containern gekapselt (Virtualisierung auf Betriebssystemebene )
- Einheitliche Weboberfläche



# Flexibilität in der Anwendung



# Benutzerschnittstelle

The image displays two screenshots of a web-based user interface for a HARTING IC MICA device. The left screenshot shows the main 'Home' page with the following icons:

- Python
- MySQL
- GPIO
- MQTT
- NodeRED
- Settings
- Install
- Information

The right screenshot shows the 'GPIO' configuration page, which includes a 'Broker IP' input field with a dropdown menu labeled 'select or add a broker' and a 'Connect' button. Below this, there is a table for mapping pins to various functions:

	Out / In	Off / On	Man / Auto	MQTT Event
Pin 1	<input type="button" value="III"/>	<input type="button" value="II"/>	<input type="button" value="III"/>	<input type="button" value=""/>
Pin 2	<input type="button" value="III"/>	<input type="button" value="II"/>	<input type="button" value="III"/>	<input type="button" value=""/>
Pin 3	<input type="button" value="III"/>	<input type="button" value="II"/>	<input type="button" value="III"/>	<input type="button" value=""/>
Pin 4	<input type="button" value="III"/>	<input type="button" value="II"/>	<input type="button" value="III"/>	<input type="button" value=""/>

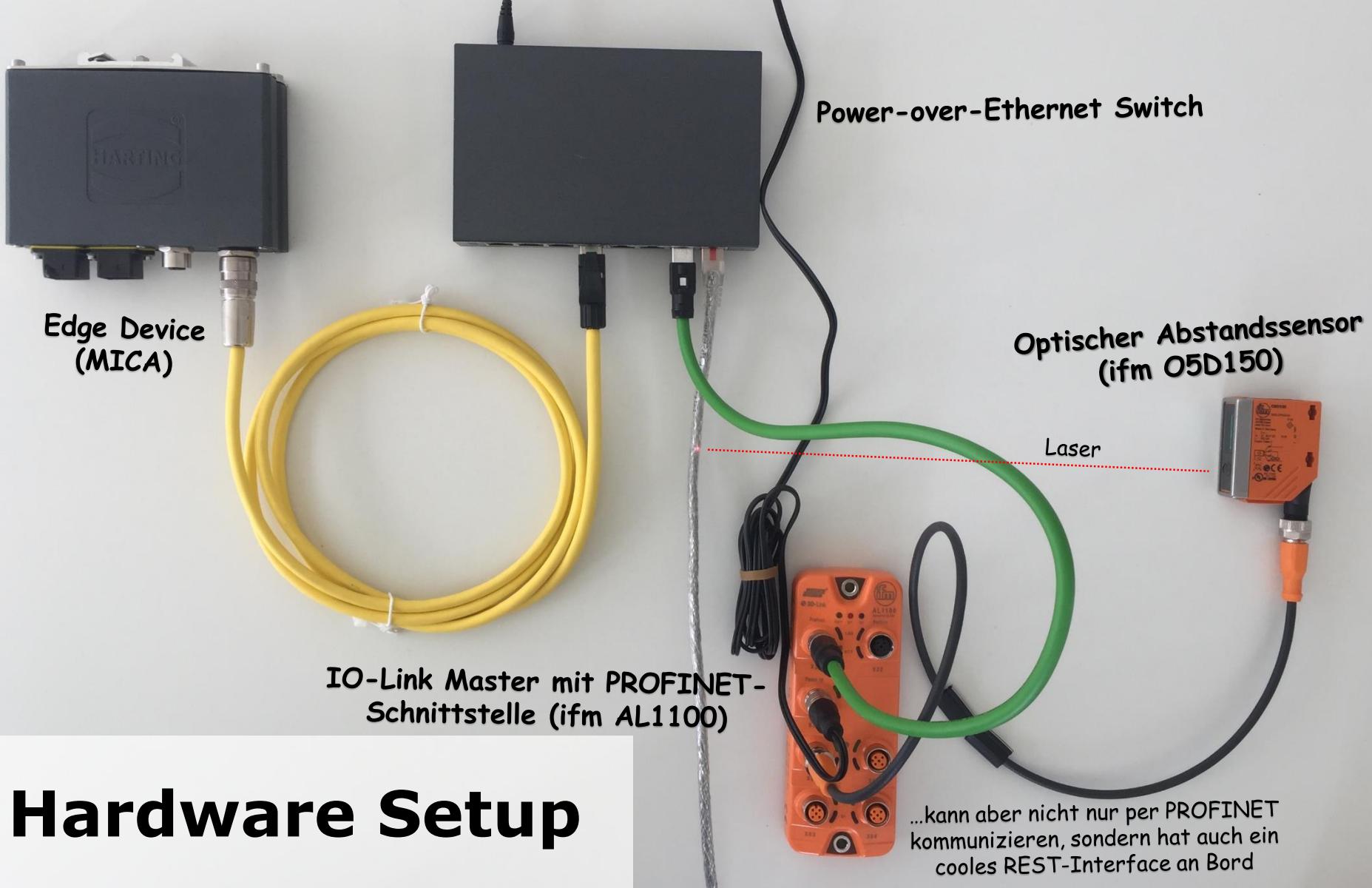
Both screenshots show the URL `https://mica-ece2y/#0_home.html` and `https://mica-ece2y/#1_GPIO` respectively, and the status 'HARTING IC MICA' at the bottom.



# Hands-On: Sensor auslesen per Node-RED

# Der geniale Plan:

- Auslesen eines Industriesensors per Edge Device und Auswerten der Daten
- Als Sensor nutzen wir einen IO-Link Laser-Abstandssensor
- Zur Programmierung Node-RED



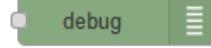
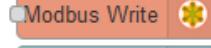
# Node-RED

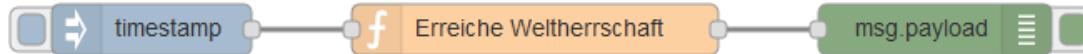
- Node-RED ist ein visueller Editor von IBM um Geräte, APIs und Online-Services zu verdrahten – „*for wiring the Internet of Things*“
- Der Editor
  - ist Browser basiert  
(Aufruf des Editors: Rechnername + Port 1880, z. B. 192.168.1.81:1880)
  - stellt eine breite Palette an Funktionsblöcken bereit um Flows zu modellieren
  - ist durch eigene in JavaScript geschriebene Blöcke erweiterbar
- Dank über 120.000 Modulen im Node package repository ist es ein Leichtes, passende Grundlagen für neue Funktionsblöcke zu finden.
- Node-RED flows werden in JSON gespeichert, was es einfach macht, sie zu importieren oder exportieren und mit anderen zu teilen.

# Node-RED Engine

- Die leichtgewichtige Engine ist in Node.js geschrieben und verfolgt damit einen ereignisgetriebenen, nicht-blockierenden Ansatz.
- Damit ist sie ideal
  - um am Rand des Netzes auf low-cost Hardware wie einem Raspberry Pi zu arbeiten
  - in der Cloud zu laufen

# Grundlegende Idee: Modellierung von Informationsflüssen

- In Node-RED fließen Message-Objekte von einem Knoten zum nächsten
- Messages entstehen in Eingangsknoten:
  - Inject 
  - MQTT 
  - Modbus 
  - ...
- Messages enden in Ausgabeknoten
  - Debug 
  - MQTT 
  - Modbus 
  - GUI-Element 
  - ...
- Während ihres Flusses können sie beliebig modifiziert werden



# Messages

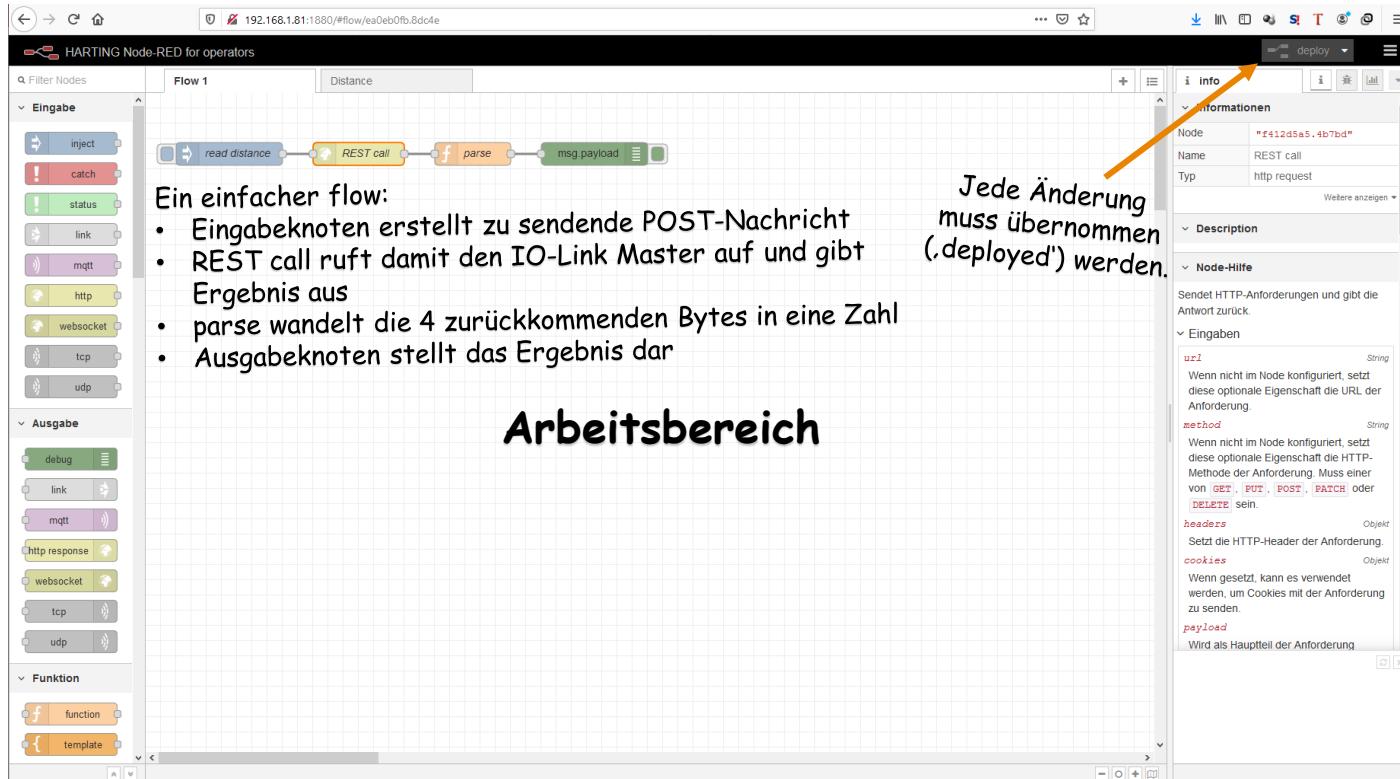
- Messages sind schlichte JSON-Objekte.  
Die des Inject-Knotens sehen beispielsweise wie folgt aus:  
...oder programmatisch (node.js):

```
var msg = {_msgid: "3059b2dd.3f1fae",
            topic: "",
            payload: 1504827168968};
```
- `_msgid` ist eine eindeutige ID und eher für den internen Gebrauch bestimmt
- Die Werte der Standardfelder `payload` und `topic` können beliebig überschrieben werden
- Über `payload` wird typischerweise die auszutauschende Information gesendet, `topic` (das Thema) ist optional und hier z. B. nicht gesetzt
- Neben diesen Standardfeldern können beliebige weitere hinzugefügt werden

```
12.9.2017, 08:41:40 node: ac1ccc76.abb01
msg : Object
  ▼ object
    _msgid: "3059b2dd.3f1fae"
    topic: ""
    payload: 1504827168968
```

# Node-RED > Übersicht

Palette der nutzbaren Knoten.



## Arbeitsbereich

Jede Änderung  
muss übernommen  
(.deployed') werden.

- Ausgabebereich
- Infos zu den Knoten
  - Infos zum Debuggen
  - Infos zur Visualisierung



# Node-RED > Auslesen des Sensors

The screenshot shows the Node-RED interface with a flow named "Flow 1". The flow consists of an "inject" node, a "read distance" node, a "REST call" node, a "parse" function node, and a "msg payload" debug node. A large orange arrow points from the "parse" node to the "function Node bearbeiten" panel on the right. This panel displays the code for the "parse" function:

```
1 json = JSON.parse(msg.payload)
2 rawInt = parseInt(json.data.value, 16)
3 distance = rawInt >> 4
4
5 msg.payload = distance
6 return msg;
```

Another orange arrow points from the "msg payload" debug node in the flow to the "Debugging" panel on the right, which shows the output: "19.2.2020, 20:48:29 node: 685122b8:2b927c msg.payload : number 36".

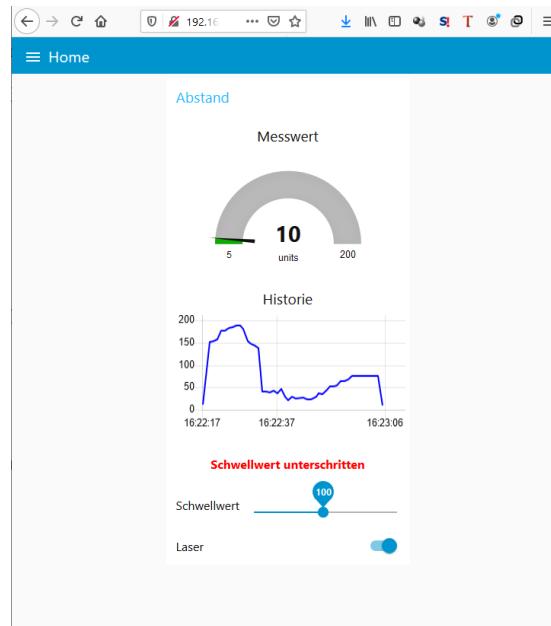
**Zur Berechnung der Distanz müssen wir im parse Knoten ganze 4 Zeilen programmieren:**

- Die Nutzlast der eingehenden Nachricht als JSON parsen, ...
- die 4 enthaltenen Bytes als Integerzahl interpretieren ...
- und davon nur die oberen 12 Bits auswerten (siehe ifm Datenblatt zum Sensor), das ...
- als neue Nutzlast in die ausgehende Nachricht schreiben.

**Et voilà:**  
Die Entfernung des Netzwerkkabels vom Sensor beträgt 36 cm

# Node-RED > Einfaches Dashboard

- Mit ein paar weiteren Knoten können schnell einfache Dashboards erstellt werden:



# Kritik

# Anregungen

# Fragen



**Prof. Dr. Stefan Berlik**  
Big Data Analytics

**Fachhochschule Bielefeld**  
**University of Applied Sciences**  
Fachbereich Ingenieurwissenschaften und Mathematik  
Campus Gütersloh

+49.521.106-70129  
[stefan.berlik@fh-bielefeld.de](mailto:stefan.berlik@fh-bielefeld.de)

[Bildquellen: soweit nicht anders angegeben bei den Herstellern bzw. pixabay]