

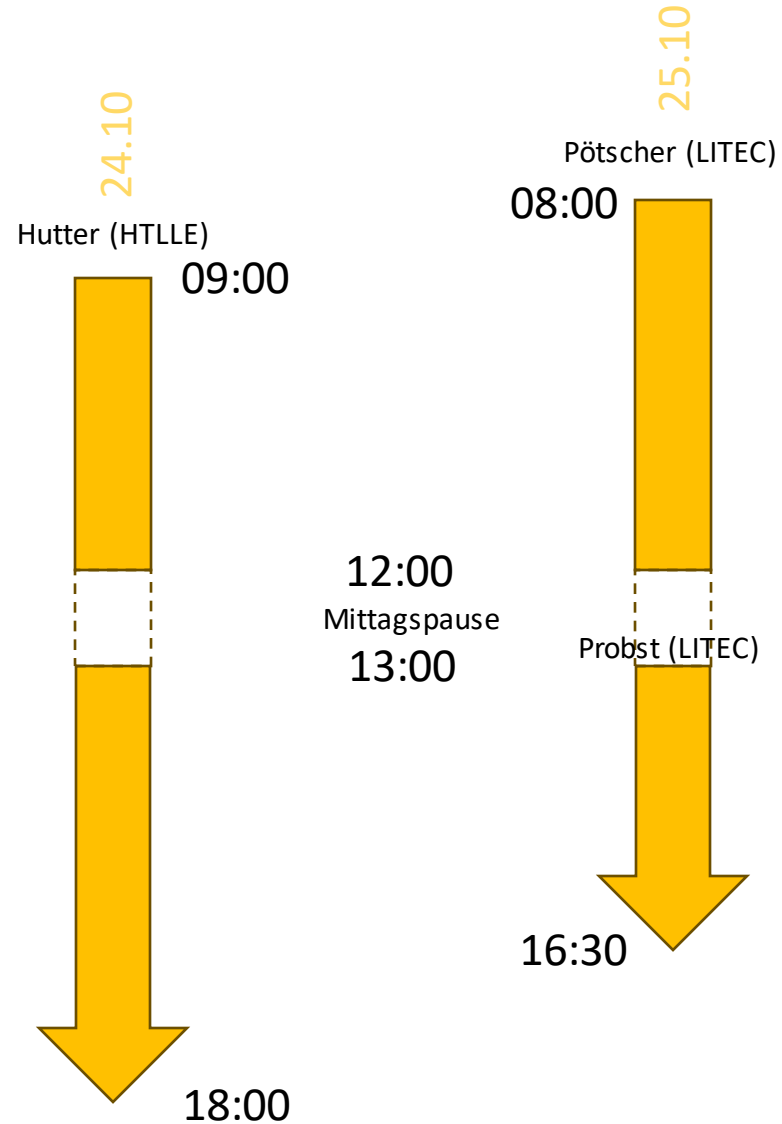
# IoT 4 Coders

mit Thingworks ans Ziel :)

...oder wie die PH es nennen würde: 3Z5B3WCS01

# Agenda

- Eröffnung und organisatorisches
- Raspi aufsetzen
- Anhängen und in Betrieb nehmen
- Daten vom Sense Shield auslesen
- Daten per MQTT versenden
- Daten in TWX holen und auf Things schreiben
- Things um Subscriptions erweitern
- Ein einfaches 2D Mashup machen



- Mashup fertigstellen
- Projekt mit eigener Hardware
- Weitere Vertiefung
- Visualisierung der Daten mit AR/VR
- Klärung offener Punkte

# Schritt 0: Imager herunterladen

- [https://downloads.raspberrypi.org/imager/imager\\_latest.exe](https://downloads.raspberrypi.org/imager/imager_latest.exe)

## Install Raspberry Pi OS using Raspberry Pi Imager

Raspberry Pi Imager is the quick and easy way to install Raspberry Pi OS and other operating systems to a microSD card, ready to use with your Raspberry Pi. [Watch our 45-second video](#) to learn how to install an operating system using Raspberry Pi Imager.

Download and install Raspberry Pi Imager to a computer with an SD card reader. Put the SD card you'll use with your Raspberry Pi into the reader and run Raspberry Pi Imager.

[Download for Windows](#)

[Download for macOS](#)

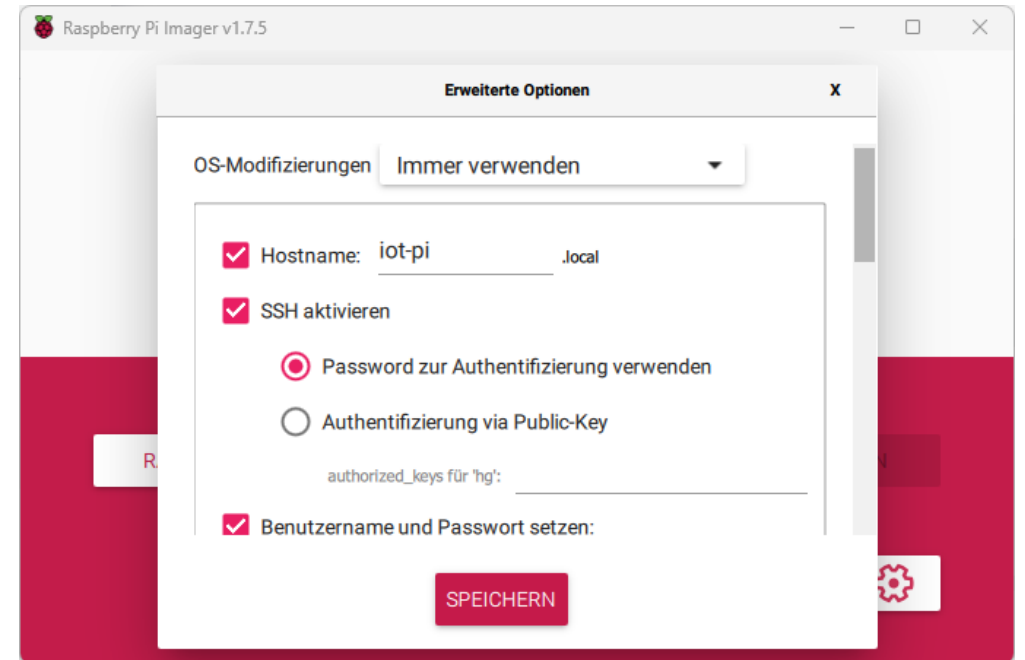
[Download for Ubuntu for x86](#)

To install on **Raspberry Pi OS**, type `sudo apt install rpi-imager` in a Terminal window.



# Vorbereiten der Images

- Hostname: `iot-pi-<kuerzel>`
- SSH erlauben: Ja, mit Passwort
  - Username und Passwort setzen (und merken 😊)
- Wifi Einrichten
  - SSID:
  - Passwort:
  - WIFI Land: AT
- Spracheinstellungen:
  - Zeitzone: Europe/Vienna
  - Tastaturlayout: de



# Schritt 1: Image flashen (32 Bit, Raspi OS)



... und warten  
bis fertig :)

Schreiben erfolgreich

X

**Raspberry Pi OS (32-bit)** wurde auf **SDHC Card** geschrieben

Sie können die SD-Karte nun aus dem Lesegerät entfernen

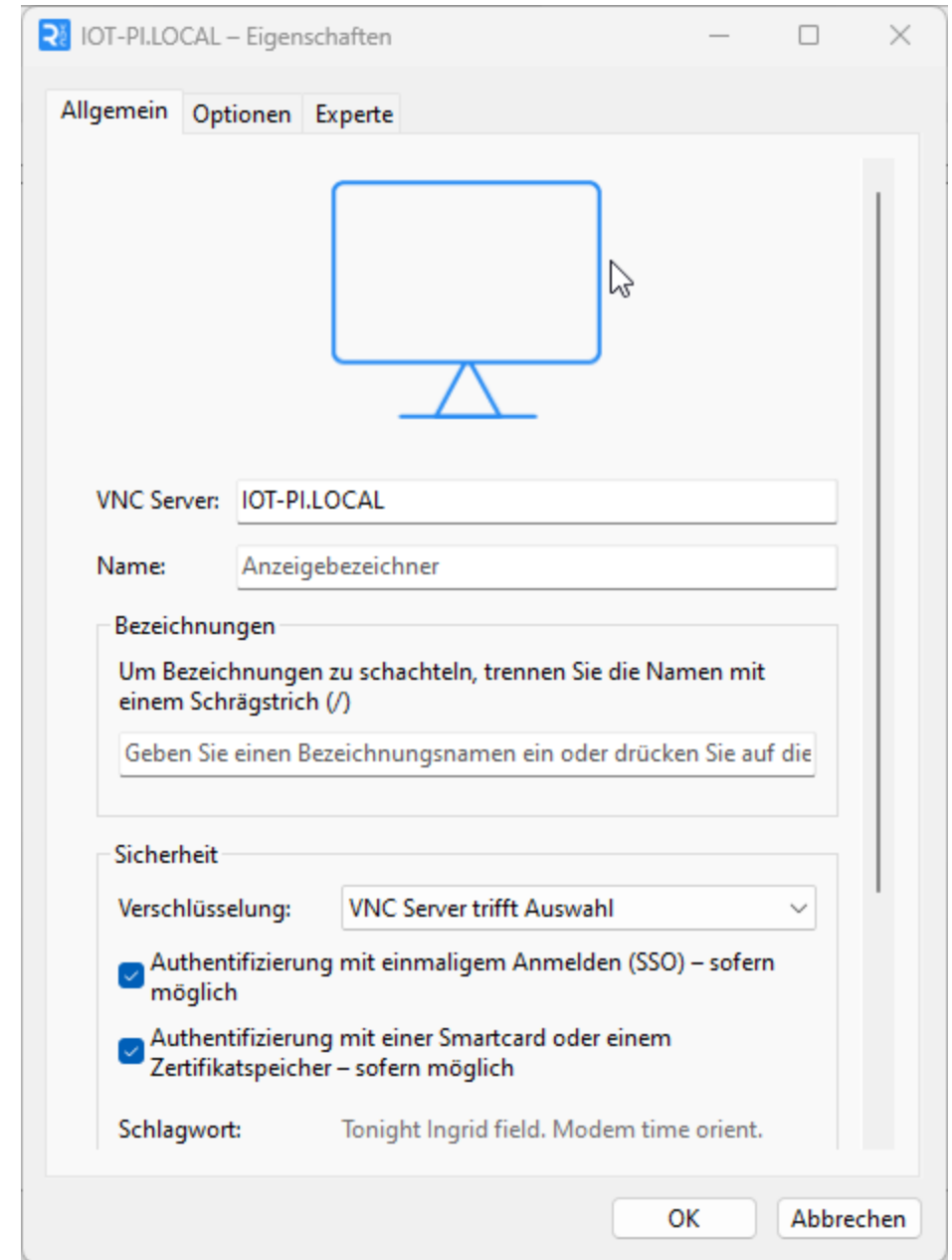
WEITER

# Erster Start:

- Putty herunterladen (oder einen anderen SSH client)
- \$ sudo raspi-config
  - Interface Options
    - VNC
      - Enable
  - System options
    - Boot / Autologin
      - Desktop Autologin
- \$ sudo reboot now

<https://the.earth.li/~sgtatham/putty/latest/w64/putty.exe>

<https://downloads.realvnc.com/download/file/viewer.files/VNC-Viewer-7.6.0-Windows.exe>

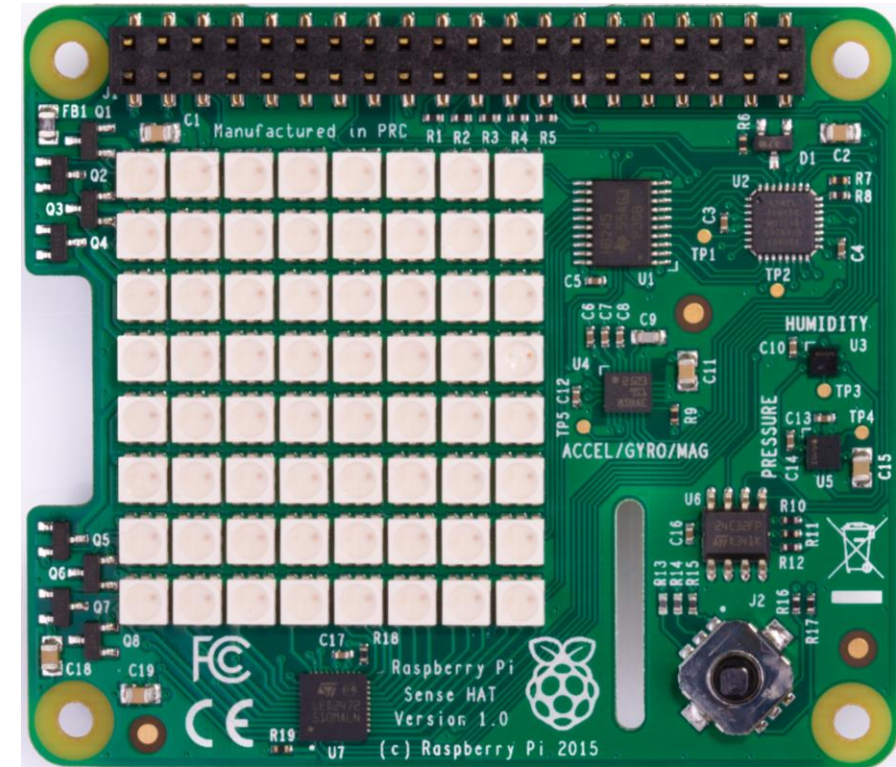


# Packages für Sense Hat installieren

Das Sense-Hat verfügt über ..

- Gyroscope
- Accelerometer
- Magnetometer
- Temperature
- Barometric pressure
- Humidity
- Colour and brightness sensor

```
$ sudo apt update  
$ sudo apt install sense-hat  
$ sudo reboot now
```





# Es ist Zeit für Hello World (in Python)

- Python ist standardmäßig am Pi installiert
- Es gibt eine tolle API für das Sense-hat
- Thonny ist als IDE vorinstalliert
  - Gut für Einsteiger geeignet
  - Mieses Debugging
- VS-Code ist besser
  - `$ sudo apt install code`



The screenshot shows the Thonny IDE window titled "Thonny - <untitled> @ 3:19". The top toolbar contains icons for New (green plus), Load (floppy disk), Save (floppy disk), Run (green play button), Debug (blue bug), Over (green arrow), Into (green arrow), Out (green arrow), Stop (red square), and Zoom (magnifying glass). The main editor area shows a Python script in a file named "<untitled> \*.py":

```
1 from sense_hat import SenseHat
2
3 sense = SenseHat()
4 sense.show_message("Hello world!")
```

Below the editor is a "Shell" window showing the execution output:

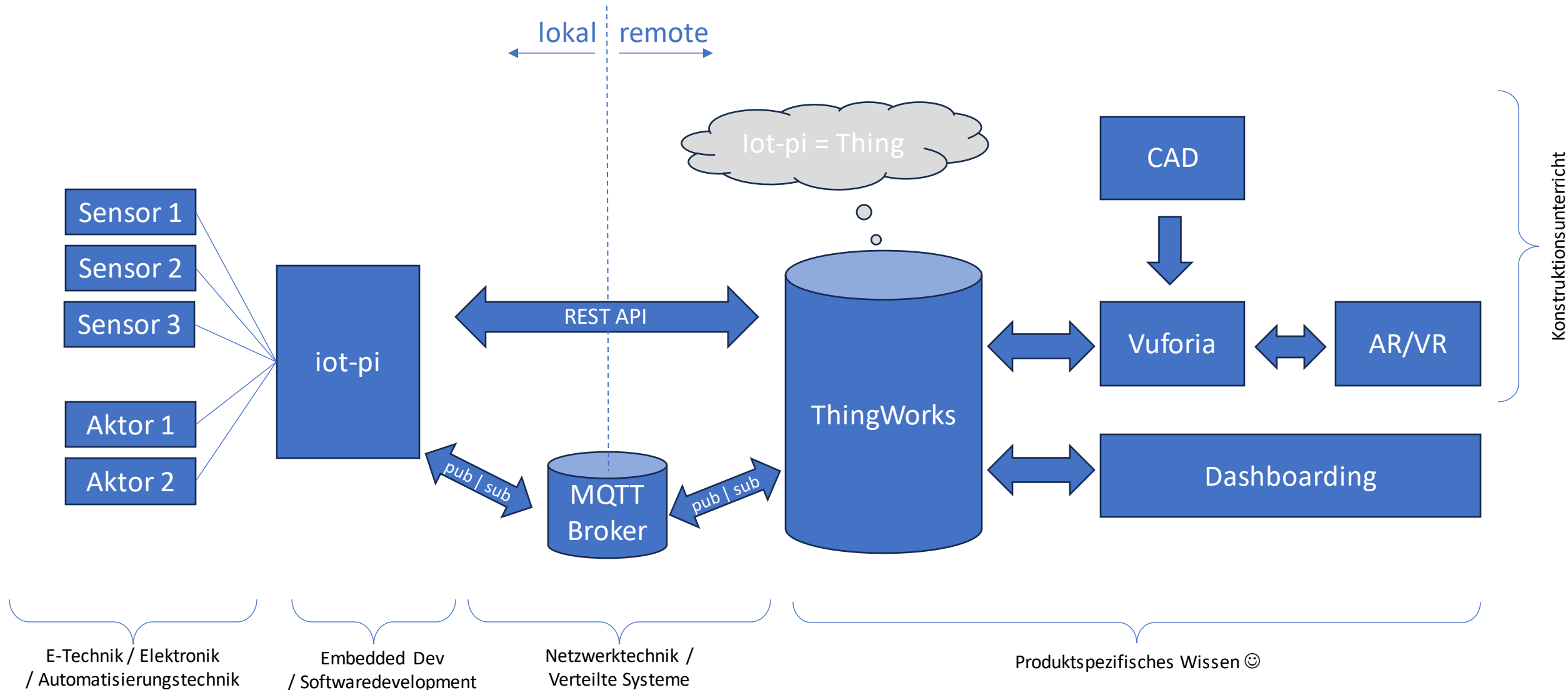
```
>>> %Run -c $EDITOR_CONTENT
WARNING:root:Failed to initialise TCS34725 colour sensor. (sensor not present)
```

The status bar at the bottom right indicates "Local Python 3 • /usr/bin/python3".

<https://pythonhosted.org/sense-hat/api/>

<https://code.visualstudio.com/docs/python/python-tutorial>

# Auf ins IoT: Systemübersicht & Grenzen || Fächer



# Use-Case: Transportüberwachung \*



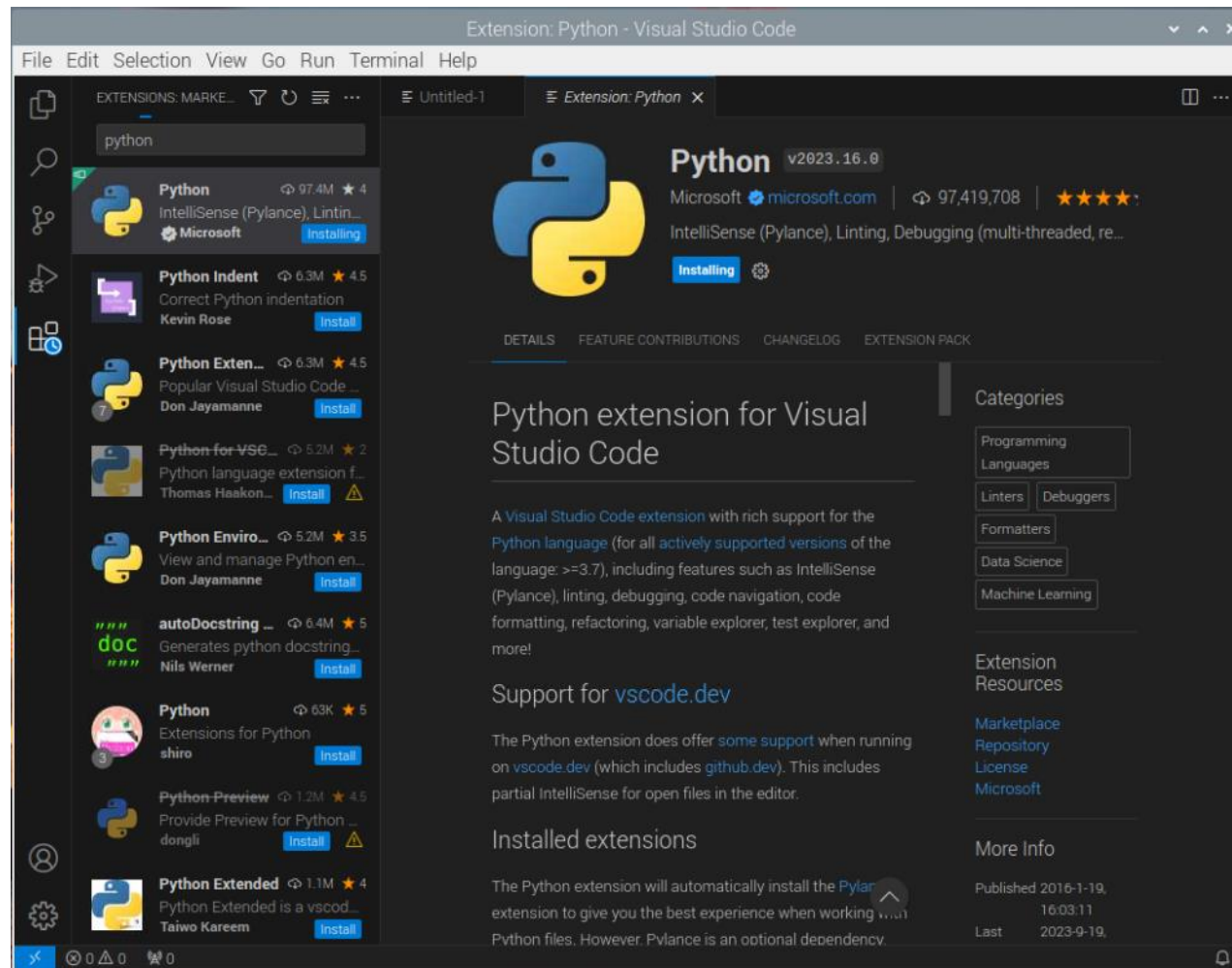
- Güter müssen...
  - Aufrecht transportiert werden => IMU
  - Sind zerbrechlich => IMU
  - Dürfen nicht feucht werden => Hygrometer



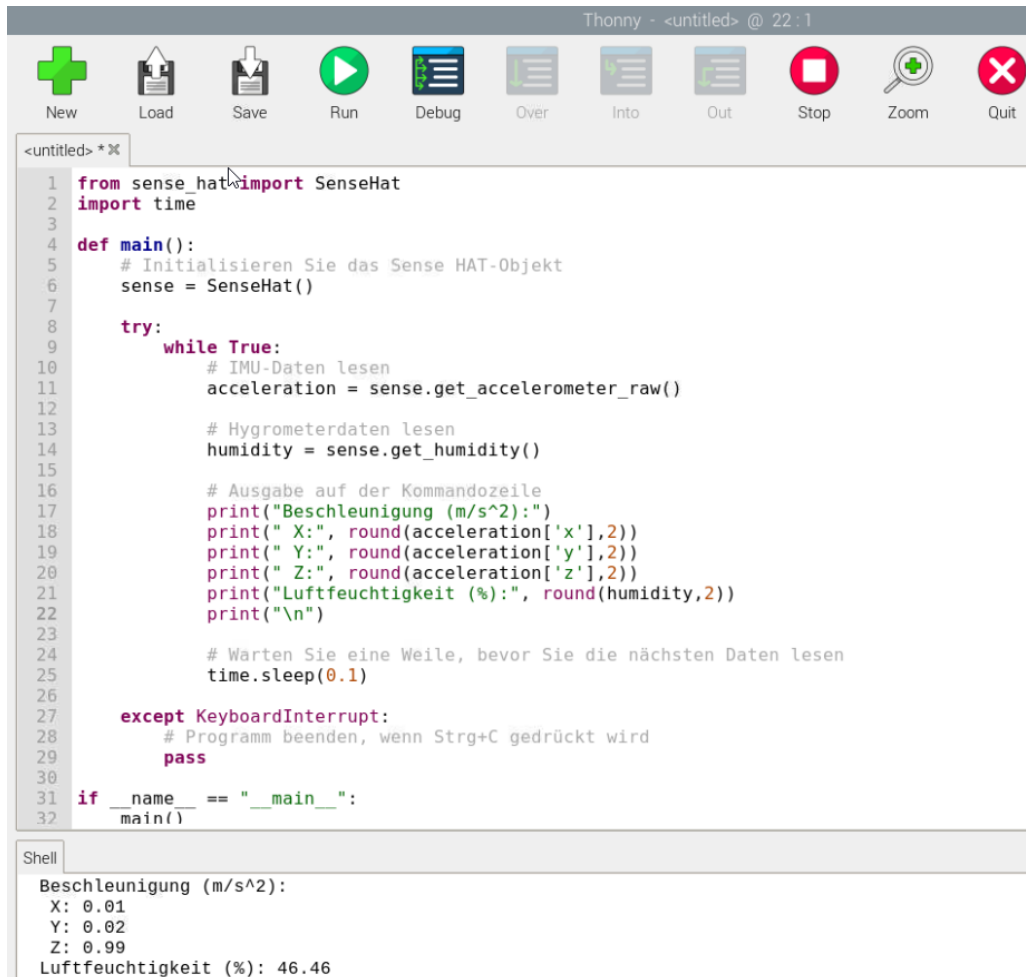
Schritt 1: Entwickeln Sie ein Python Programm das die IMU und das Hygrometer fortlaufend ausliest und die Daten auf der Commandline ausgibt

\* Wir gehen davon aus, dass unser Raspi hier mitreist, immer genügend Akku hat und auch eine stabile Verbindung ins Internet

# Python in vs-code hinzufügen



# Zwischenergebnis:

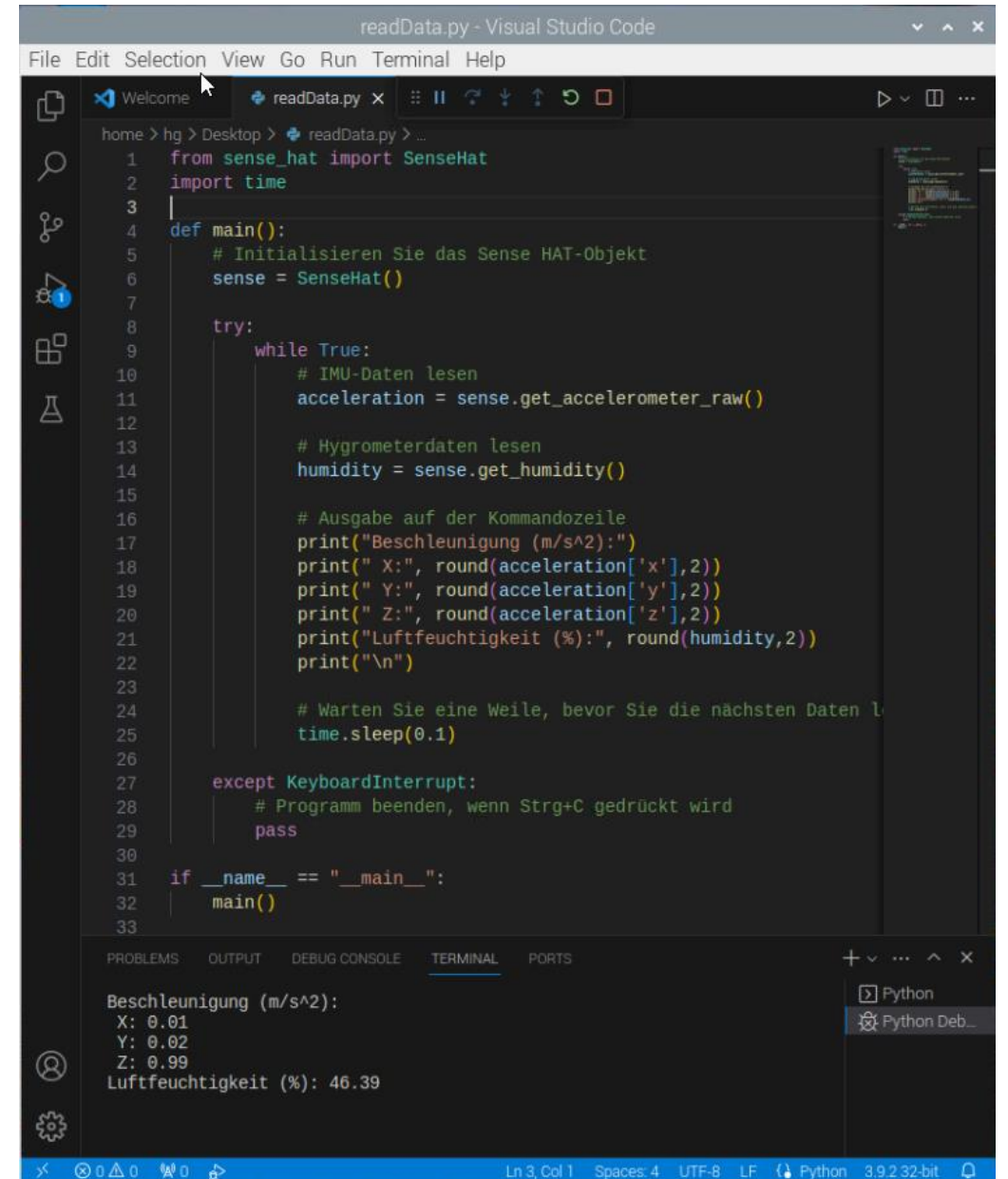


The screenshot shows the Thonny IDE interface. The top toolbar includes icons for New, Load, Save, Run, Debug, Over, Into, Out, Stop, Zoom, and Quit. The main editor displays a Python script named `readData.py`. The script imports `sense_hat` and `time`, initializes a `SenseHat` object, and enters a `while True` loop to read and print acceleration and humidity data. A `KeyboardInterrupt` exception is handled to allow the program to be stopped. The bottom panel shows the shell output, which matches the printed data from the script.

```
1 from sense_hat import SenseHat
2 import time
3
4 def main():
5     # Initialisieren Sie das Sense HAT-Objekt
6     sense = SenseHat()
7
8     try:
9         while True:
10             # IMU-Daten lesen
11             acceleration = sense.get_accelerometer_raw()
12
13             # Hygrometerdaten lesen
14             humidity = sense.get_humidity()
15
16             # Ausgabe auf der Kommandozeile
17             print("Beschleunigung (m/s^2):")
18             print(" X:", round(acceleration['x'],2))
19             print(" Y:", round(acceleration['y'],2))
20             print(" Z:", round(acceleration['z'],2))
21             print("Luftfeuchtigkeit (%):", round(humidity,2))
22             print("\n")
23
24             # Warten Sie eine Weile, bevor Sie die nächsten Daten lesen
25             time.sleep(0.1)
26
27         except KeyboardInterrupt:
28             # Programm beenden, wenn Strg+C gedrückt wird
29             pass
30
31 if __name__ == "__main__":
32     main()
```

Shell

```
Beschleunigung (m/s^2):
X: 0.01
Y: 0.02
Z: 0.99
Luftfeuchtigkeit (%): 46.46
```



The screenshot shows the Visual Studio Code IDE interface. The top toolbar includes icons for File, Edit, Selection, View, Go, Run, Terminal, and Help. The main editor displays the same Python script as the Thonny IDE. The bottom panel shows the terminal output, which matches the printed data from the script.

```
1 from sense_hat import SenseHat
2 import time
3
4 def main():
5     # Initialisieren Sie das Sense HAT-Objekt
6     sense = SenseHat()
7
8     try:
9         while True:
10             # IMU-Daten lesen
11             acceleration = sense.get_accelerometer_raw()
12
13             # Hygrometerdaten lesen
14             humidity = sense.get_humidity()
15
16             # Ausgabe auf der Kommandozeile
17             print("Beschleunigung (m/s^2):")
18             print(" X:", round(acceleration['x'],2))
19             print(" Y:", round(acceleration['y'],2))
20             print(" Z:", round(acceleration['z'],2))
21             print("Luftfeuchtigkeit (%):", round(humidity,2))
22             print("\n")
23
24             # Warten Sie eine Weile, bevor Sie die nächsten Daten lesen
25             time.sleep(0.1)
26
27         except KeyboardInterrupt:
28             # Programm beenden, wenn Strg+C gedrückt wird
29             pass
30
31 if __name__ == "__main__":
32     main()
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
Beschleunigung (m/s^2):
X: 0.01
Y: 0.02
Z: 0.99
Luftfeuchtigkeit (%): 46.39
```

Ln 3, Col 1 Spaces: 4 UTF-8 LF Python 3.9.2 32-bit

# Use-Case: Transportüberwachung

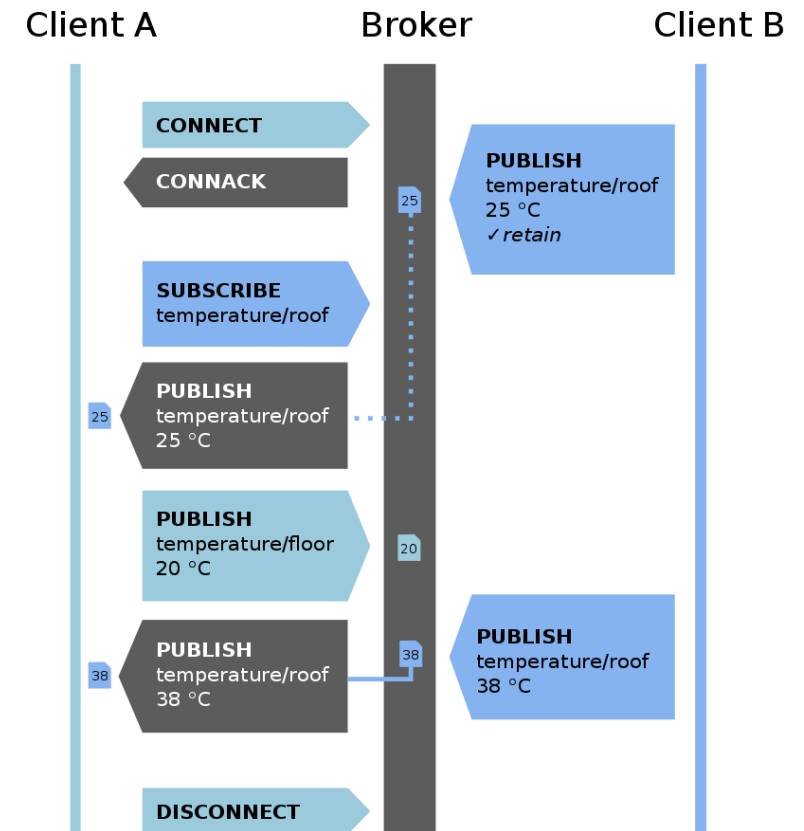
- Jetzt haben wir alles ausgelesen -> Wir müssen die Daten nun irgendwie loswerden



Schritt 2: Entwickeln Sie ein Python Programm das Daten per MQTT an einen Broker sendet

# Schritt 2: Daten übertragen (per MQTT)

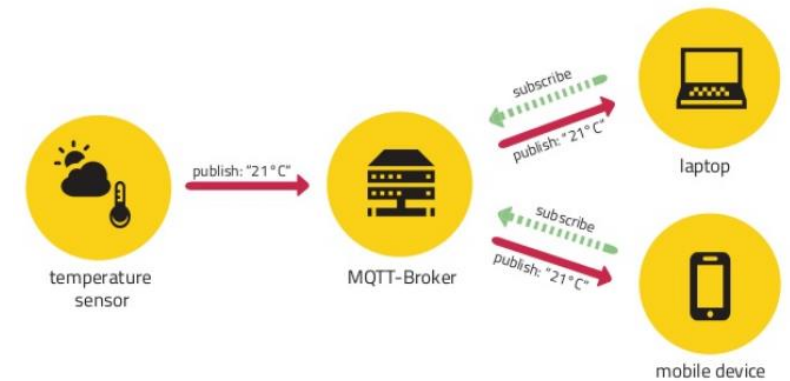
- Publisher: Veröffentlicht Daten (=Sender)
- Subscriber: Konsumiert Daten (=Empfänger)
- Topic: Ein 'Thema' dem die Daten zugeordnet sind (ähnlich einem Chatraum)





# MQTT Broker

- TWX bringt KEINEN eigenen MQTT Broker mit
- TWX spricht MQTT über ein Plugin



- Wir brauchen einen Broker der vom Client und vom Server aus ‚sichtbar‘ ist:
  - Variante 1: Selbstinstallation am Raspi und dann in Internet hängen

```
$ sudo apt-get install mosquitto  
$ sudo apt-get install mosquitto-clients
```

- Variante 2: Einen broker finden und selbigen mitbenutzen
  - z.B: <https://console.hivemq.cloud/>



# MQTT ausprobieren

1. Broker suchen / erstellen

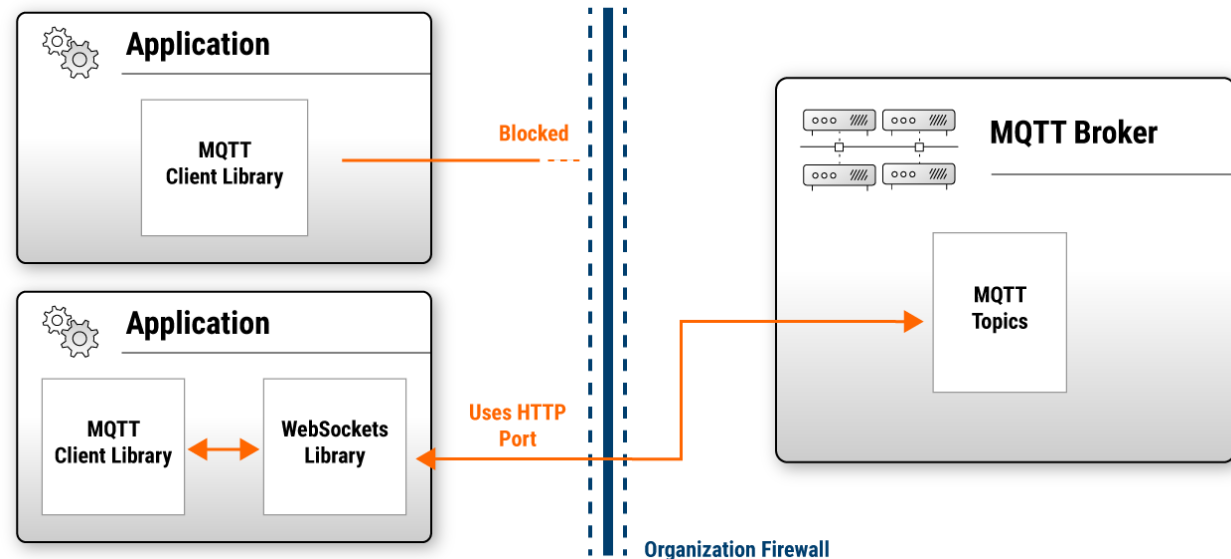
2. Zugangsdaten merken

3. Daten per Python pushen

iot-pi-<kuerzel>/<sensor>/<wert>

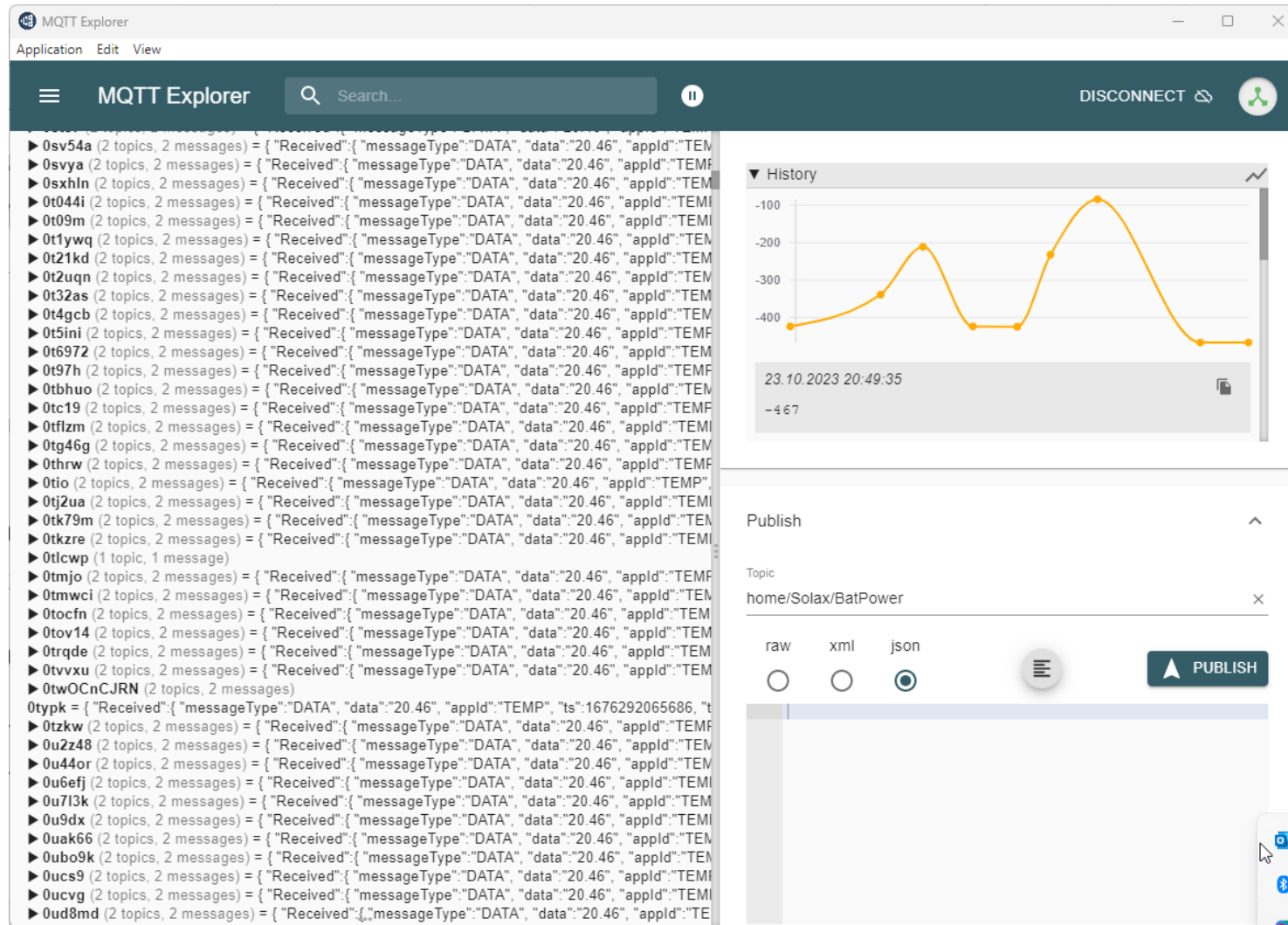
4. Daten wieder ansehen

- <https://www.hivemq.com/demos/websocket-client/>
- <https://github.com/thomasnordquist/MQTT-Explorer/>



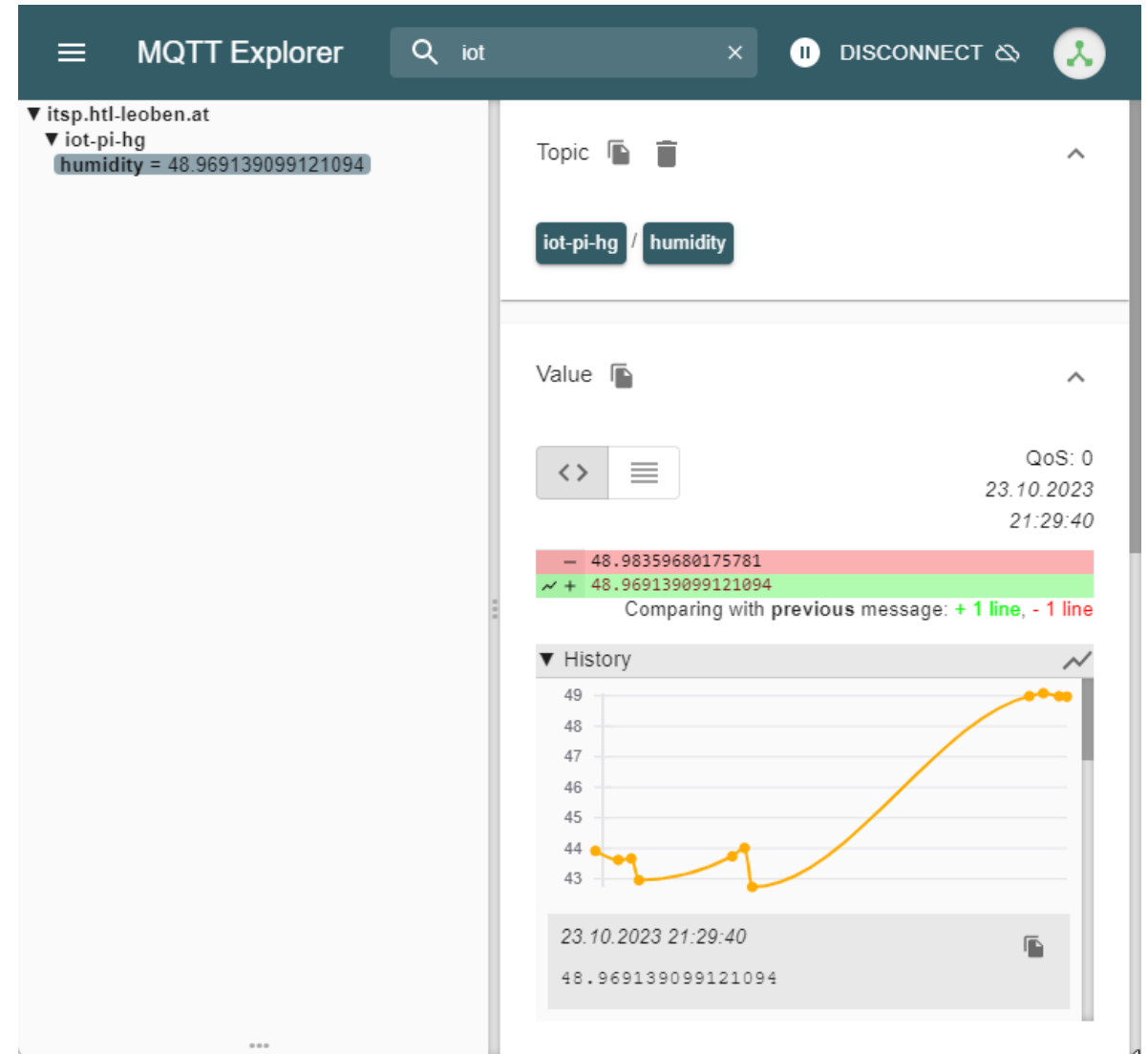
Quelle: [Understanding the Differences between MQTT and WebSockets for IoT \(hivemq.com\)](https://www.hivemq.com/demos/websocket-client/)

# MQTT Explorer



# Resultat via „plain“ MQTT

```
1 import paho.mqtt.client as mqtt
2 from sense_hat import SenseHat
3
4 # Datenquelle
5 sense = SenseHat()
6
7 # MQTT-Broker-Verbindungsinformationen
8 broker_address = "itsp.htl-leoben.at"
9 broker_port = 1883
10 mqtt_topic = "iot-pi-hg/humidity"
11 username = "your_username"
12 password = "your_password"
13
14 # MQTT-Client initialisieren
15 client = mqtt.Client()
16 client.username_pw_set(username, password)
17
18 # Verbindung zum MQTT-Broker herstellen
19 client.connect(broker_address, broker_port)
20
21 # Nachricht veröffentlichen
22 message = sense.get_humidity()
23 client.publish(mqtt_topic, message)
24
25 # MQTT-Client sauber beenden
26 client.disconnect()
27
```



# Resultat via Websocket

```
1 import paho.mqtt.client as paho
2
3 from paho import mqtt
4 from sense_hat import SenseHat
5
6 client = paho.Client(client_id="", userdata=None, protocol=paho.MQTTv5)
7 sense = SenseHat()
8
9 # enable TLS for secure connection
10 client.tls_set(tls_version=mqtt.client.ssl.PROTOCOL_TLS)
11
12 # set username and password
13 client.username_pw_set("iottest", "IotTest123")
14
15 # connect to HiveMQ Cloud on port 8883 (default for MQTT)
16 client.connect("dc5a170db2354a828896bc071195f7dd.s2.eu.hivemq.cloud", 8883)
17
18 # a single publish, this can also be done in loops, etc.
19 client.publish("iot-pi-hg/temperature", payload=sense.get_temperature(), qos=1)
20
21 # ciao
22 client.disconnect()
```

Hivemq hat einen integrierten Explorer für die Messages

Message	Topic	QoS	Timestamp
33.1054801940918	iot-pi-hg/temperature	0	1698091327857
32.78321838378906	iot-pi-hg/temperature	0	1698091315098

# Use-Case: Transportüberwachung



- Die Daten sind nun auf einem MQTT Broker für andere (mehr oder weniger) einsehbar.



Schritt 3: Wir würden die Daten gerne nach TWX holen um sie dort ersteinmal anzuzeigen

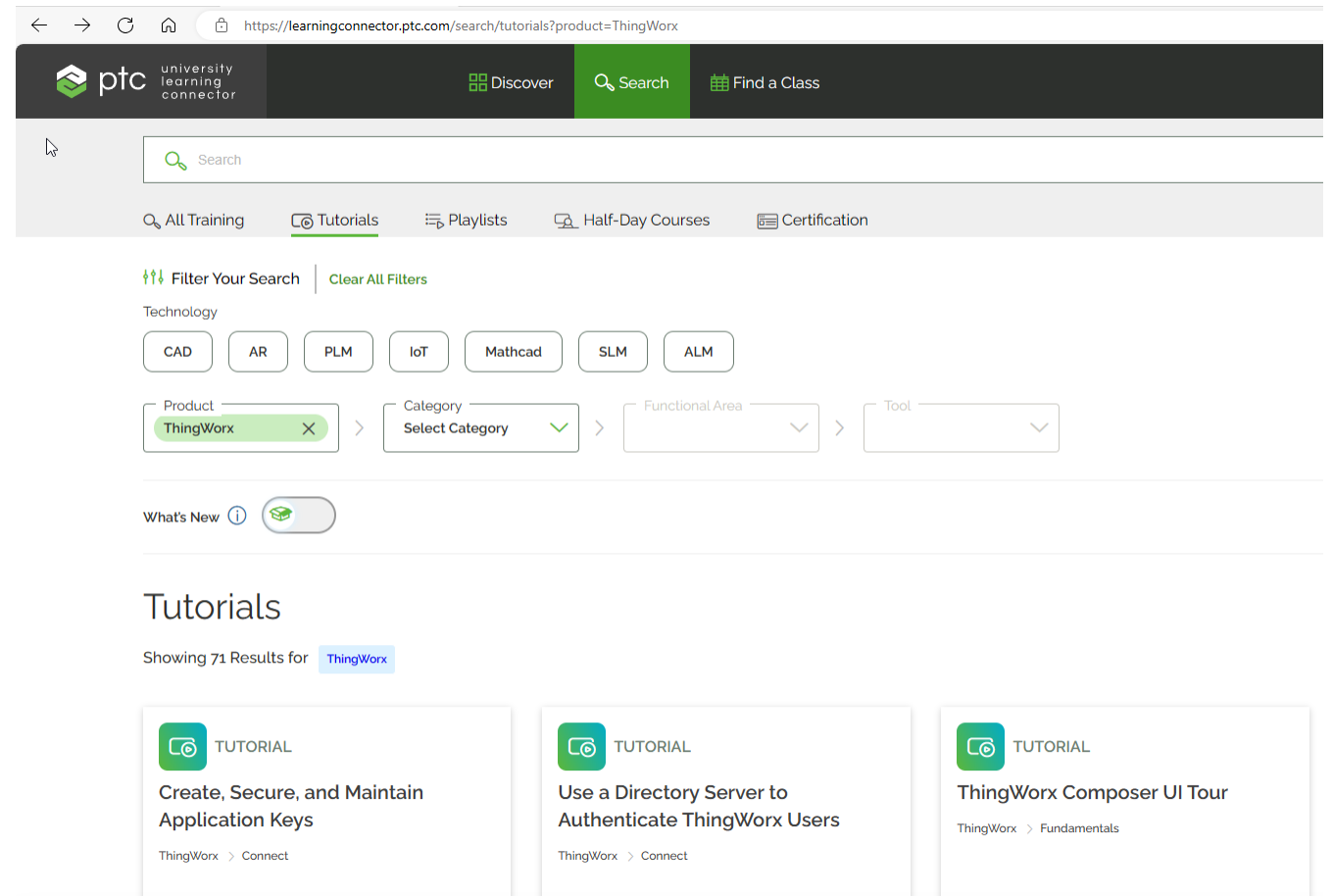
# Unser Zugang zu thingworx®

- HTL Mödling hostet Thingworks Instanzen für HTLs
- Kubernetes basierte Lösung -> Funktioniert im Regelfall gut und einfach
- Eigenes erstellen von „Pods“ (=Instanzen) möglich
  - Isolierte Arbeitsumgebung (z.B. für Diplomarbeiten, Laborgruppen, Klassen)
  - Jede Instanz verfügt immer über die „Grundfunktionen“. Manches muss erst nachinstalliert werden
- Wir verwenden für diese Schulung folgende Instanz:

twx-8cf48	<a href="https://8cf48.twx.htl.schule">https://8cf48.twx.htl.schule</a>	Expires: 06 Jan 2024	Responsible person: Robert Hauss	
11 Jan 2023	Student class: ET_EL_IT_DEMO	Student name:	Initial Password: ●●●●●●●●	E-mail adress: rha[redacted]at
				Running
				Delete

# Hilfe rund um Thingworks

- [https://www.ptc.com/en/support/help/thingworx\\_doc\\_resources](https://www.ptc.com/en/support/help/thingworx_doc_resources)
- <https://info.twx.htl.schule/>
- Diverse Thingworks foren
- PTC University ->
- <https://www.ptc.com/en/ptc-university/product-training-catalogs/thingworx-training>  
(Zugangsdaten werden benötigt)



# Was ist TWX ?

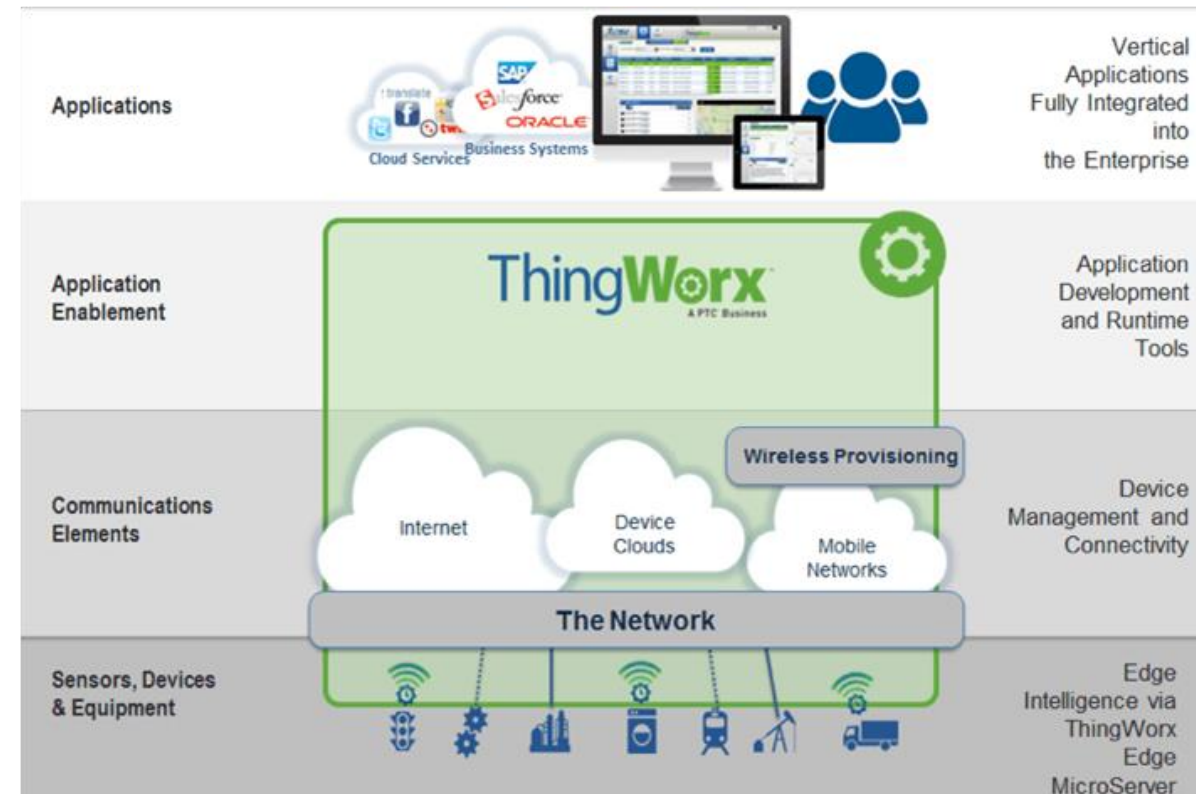
- ThingWorks ist eine leistungsstarke IoT-Plattform (Internet of Things), die von PTC entwickelt wurde. Sie wurde speziell für die Entwicklung und Verwaltung von IoT-Anwendungen und -Lösungen konzipiert.
- ThingWorks bietet Programmierlehrern eine ideale Umgebung, um Schülern die Grundlagen des IoT und der Anwendungsentwicklung näherzubringen.



# Was kann ich damit machen ?

- IoT-Anwendungen erstellen und implementieren.
- Daten von vernetzten Geräten sammeln, analysieren und visualisieren.
- IoT-Protokolle und -Konnektivität verstehen und nutzen.
- Schülern praktische Erfahrungen in der Entwicklung von IoT-Lösungen vermitteln.

Dank seiner benutzerfreundlichen Schnittstellen und umfangreichen Funktionen eignet sich ThingWorks gut für den Einsatz im Unterricht und ermöglicht es Programmierlehrern, Schülern IoT-Konzepte auf praxisnahe Weise beizubringen.



Quelle: Industrial Internet of Things | Thingworx ([modelcamtechnologies.com](http://modelcamtechnologies.com))

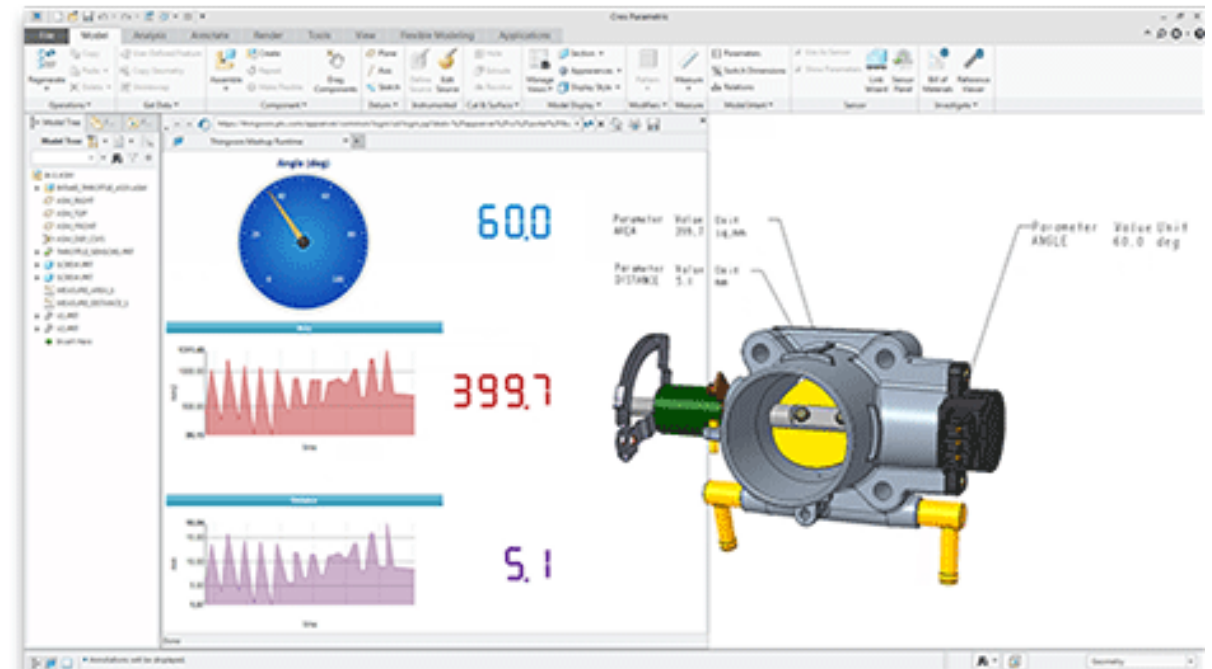
# Warum TWX (und nicht XY\*) ?

Von Programmierern -> Für Programmierer

Niederschwelliger Einstieg

Einfach für Schüler

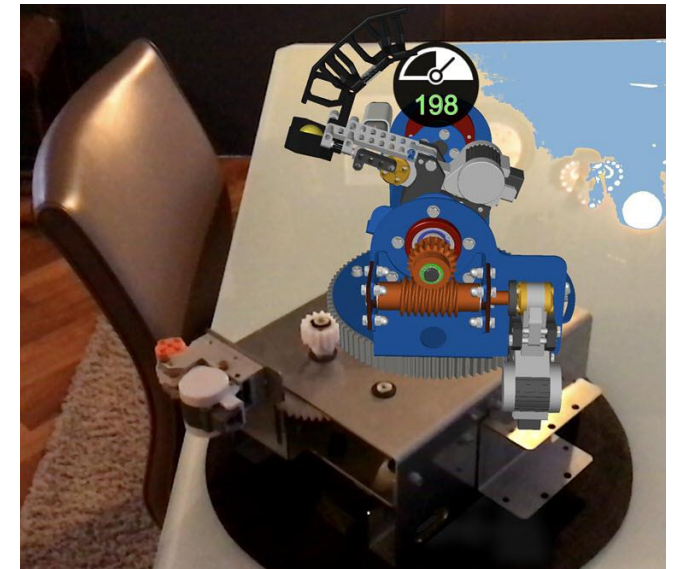
- Things (=Objekte)
- Mashups (=Dashboards)
- Services (=Funktionen)
- Templates (= abstrakte Basisklassen)
- ....



[Quelle: Creo Product Insight to bring a 'new dimension' to the design process - DEVELOP3D](#)

# OK, das kann ich mit "XY" auch ...

- Einbindung verschiedener Schnittstellen (MQTT, REST, ...)
- Industriestandards mittels KepWare
- HomeAutomation via node.red, o.Ä.
- Augmentierung / Visualisierung mittels 3D Endgeräten
  - Handy
  - HoloLens
  - ...
- Einfaches Einbinden von CAD Daten :)
- u.v.m



# Was kommt jetzt ?

- Rundgang durch TWX (GUI)
- Things anlegen
- Eigenschaften anlegen
- Alerts anlegen
- ... Beispiel fertig machen



# Die Thingworks GUI a.k.a. „Composer“

- In unserem Setup gibt es nur einen Benutzer (für alle)
- Projekte helfen Dinge auseinanderzuhalten
  - Projekt anlegen: <Schulkennzahl>\_<Nachname>
- Ein Berechtigungssystem lässt (theoretisch) beliebig komplizierte Berechtigungsstrukturen zu -> Vorsicht: Nicht übertreiben !
- Verfügt über verschiedene Logging Mechanismen



Browse

×

Browse

Things

+ New

View

Edit

Duplicate

Delete

1/2

⌚

	Actions	Name	Description	Date Modified
<input type="checkbox"/>		hivemq-broker		2023-10-23 21:07:56.990
<input type="checkbox"/>		310447_HOEF_SchulungS...		2023-09-29 14:36:04.482
<input type="checkbox"/>		401467_pota_raspi		2023-09-29 13:34:38.692
<input type="checkbox"/>		310_447_STK_Schulung_r...		2023-09-29 13:27:14.083
<input type="checkbox"/>		310447_STK_Schulung_raspi		2023-09-29 13:05:51.585
<input type="checkbox"/>		310477_KRN_raspi		2023-09-29 12:52:24.390
<input type="checkbox"/>		302467_BERM_raspi		2023-09-29 11:51:37.082
<input type="checkbox"/>		302467_SPIR_raspi		2023-09-29 11:51:34.382
<input type="checkbox"/>		304417_STT_raspi		2023-09-29 11:49:21.814
<input type="checkbox"/>		304417_STW_raspiWrN		2023-09-29 11:48:43.489
<input type="checkbox"/>		310447_STK_MQTT_conn...		2023-09-29 11:27:15.283
<input type="checkbox"/>		310447_STK_Schulung_M...		2023-09-29 11:13:18.787
<input type="checkbox"/>		310447_HOEF_SchulungS...		2023-09-29 11:12:48.386
<input type="checkbox"/>		302467_BERM_MQTT_co...		2023-09-29 11:12:35.584
<input type="checkbox"/>		310447_STK_Schulung_Th...		2023-09-29 11:12:19.487
<input type="checkbox"/>		310477_KRN_MQTT_conn...		2023-09-29 11:12:14.885

# Things

... entsprechen Objekten in einer OO- Programmiersprache

- Haben Eigenschaften (Properties)
- Sind stark typisiert (u.U etwas schräge Datentypen)
- Verfügen über EventListener (Services)
- Können Funktionen für andere Dinge zur Verfügung stellen (Serices)
- Haben auch Spezialfunktionen wie z.B Alerts um Grenzwertverletzungen einfach handeln zu können
- Können von „Templates“ erben (Template ≈ abstr. Basisklasse)



# Use-Case: Transportüberwachung

- Die Rohdaten sind nun in TWX – wir möchten diese nun weiterverarbeiten



Schritt 4: Schreiben Sie eine Subscription welche erkennt ob sich ein Objekt gerade bewegt oder nicht



# Thingworks + Javascript =

Alle clientside customizations sind in Javascript zu schreiben (Services, Subscriptions, ...)

Serverside customizations sind in Java geschrieben (Plugins etc.)

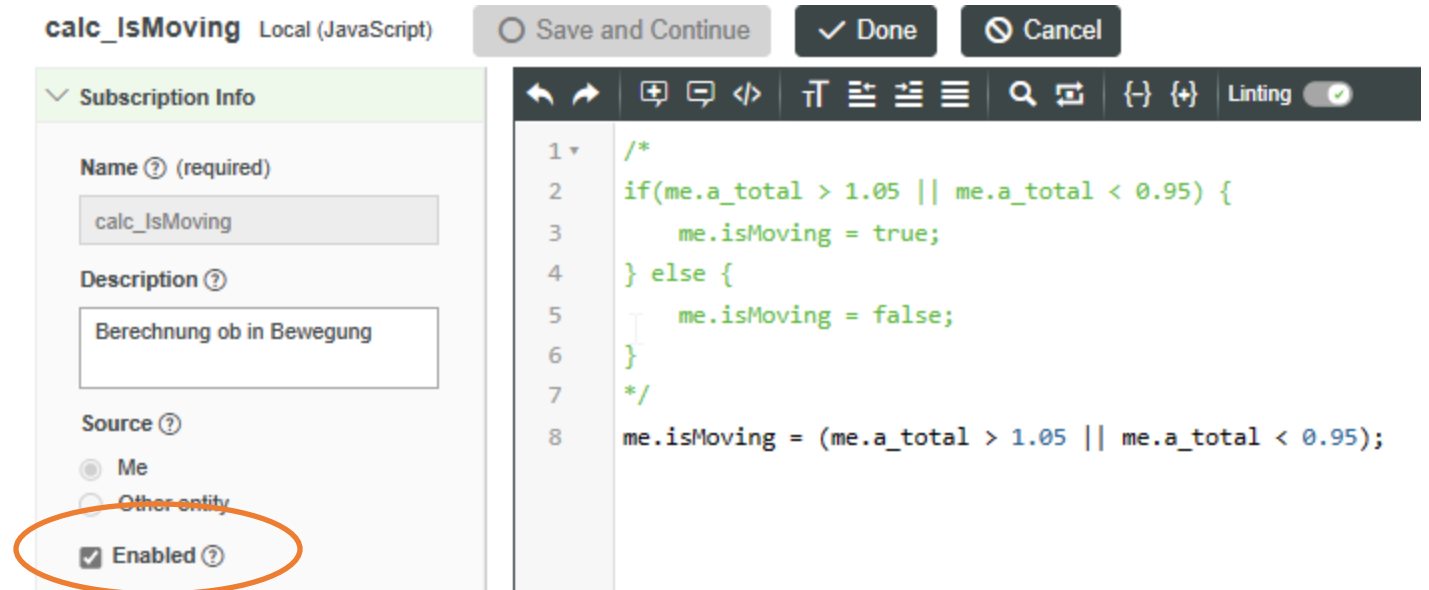
MashUps (=Dashboards) sind unter der Haube HTML, CSS+ Javascript

```
> typeof NaN
< "number"
> 9999999999999999
< 10000000000000000
> 0.5+0.1==0.6
< true
> 0.1+0.2==0.3
< false
> Math.max()
< -Infinity
> Math.min()
< Infinity
> []+[]
< ""
> []+{}
< "[object Object]"
> {}+[]
< 0
> true+true+true===3
< true
> true-true
< 0
> true==1
< true
> true===1
< false
> (!+[]+[]+![]).length
< 9
> 9+"1"
< "91"
> 91-"1"
< 90
> []==0
< true
> []+[]
< ""
> []+{}
< "[object Object]"
> {}+[]
< 0
> true+true+true===3
< true
> true-true
< 0
```



# Subscriptions

- ermöglichen es uns auf Änderungen der Rohdaten zu reagieren
- werden von TWX selbstständig aufgerufen
- Können daraufhin selbst
  - wieder Daten im eigenen Thing ändern
  - Können aber auch andere Mechanismen auf anderen Things auslösen



# Use-Case: Transportüberwachung



- Wir haben nun Daten in TWX und eine boolsche Variable welche auf true gesetzt wird wenn sich unser Paket bewegt.



Schritt 5: Holen Sie diese Daten mit dem Raspi wieder per HTTP ab und färben Sie das Display ROT wenn sich das Ding mit über 3g Bewegt hat

# Daten aus Thingworks verwenden

- Daten können auf verschiedene Arten aus TWX geholt werden
  - MQTT (Property -> publish)
  - Mail (eigenes Plugin erforderlich)
  - Per HTTP Request (API Key erforderlich)

```
$ curl --silent -H "Accept: application/json" -H "appkey: 08e5752d-80e5-4efd-aa27-5feb70606f0d" https://8cf48.twx.htl.schule/Thingworx/Things/MeinThing/Properties/i1skljfhsadoifh  
{  
  "dataShape": {  
    "fieldDefinitions": {  
      "i1": {  
        "name": "i1",  
        "description": "Strangstrom i1  
skljfhsadoifh",  
        "baseType": "NUMBER",  
        "ordinal": 2,  
        "aspects": {  
          "minimumValue": 0.0,  
          "isPersistent": true,  
          "dataChangeType": "VALUE",  
          "units": "mA",  
          "maximumValue": 350.0,  
          "cacheTime": 0.0  
        }  
      }  
    },  
    "rows": [ { "i1": 65.0 } ]  
  }  
}
```

# Daten mit der requests Bibliothek holen

```
1 import requests
2
3 # Thingworx Einstellungen
4 THINGWORX_URL = 'https://8cf48.twx.htl.schule/Thingworx/Things/MeinThing/Properties/i1'
5 HEADERS = {
6     'Accept': 'application/json',
7     'appkey': '08e5752d-80e5-4efd-aa27-5feb70606f0d'
8 }
9
10 response = requests.get(THINGWORX_URL, headers=HEADERS)
11
12 # Überprüfen Sie die Antwort
13 if response.status_code == 200:
14     data = response.json()
15     i1_value = data['rows'][0]['i1']
16     print(f"i1 Wert: {i1_value}")
17 else:
18     print(f"Fehler beim Abrufen von Daten: {response.text}")
```

Diagram labels and annotations:

- TwX Instanz**: Points to the domain part of the URL (`twx.htl.schule`).
- Thing name**: Points to the `MeinThing` part of the URL.
- Property Name**: Points to the `i1` part of the URL.
- API Key**: Points to the `appkey` value in the headers.
- Property Name**: Points to the `i1` key in the JSON response structure.

# Use-Case: Transportüberwachung



- Die Transportüberwachung funktioniert nun. Wir hätten gerne für unseren Kunden eine Anzeige der Daten (= 2D Mashup)



Schritt 6: Erstellen Sie eine Weboberfläche welche die aktuellen Daten anzeigt

# 2D Daten Repräsentation

- ... nennt sich in TWX „MashUp“ und erzeugt HTML Seiten
- Ist mit einem (manchmal etwas gewöhnungsbedürftigen) Editor ohne HTML Kenntnisse machbar
- Kann Daten aus 0-N Things anzeigen
- Kann wiederum Events auslösen (auf die Dinge dann reagieren können) -> GUI



# Lösung MashUp

The screenshot displays the Thingworx MashUp editor interface. The top bar shows the project name '302467\_BERM\_Mashup' and navigation buttons like 'View Mashup', 'Save', 'Cancel', and 'More'. Below this is a toolbar with icons for General Information, Design, Custom CSS, Mobile Settings, Permissions, Change History, and View Relationships. The main workspace is divided into several sections:

- Widgets:** A list of available widgets including Bar Chart, Breadcrumb, Bubble Chart, Button, Button Bar, Checkbox, Chip Based Data Filter, Collection, and Custom Mashup. The 'Standard (57)' category is selected.
- Properties:** A panel on the left showing the properties of the selected widget 'leddisplay-12'. It includes fields for Id, Type, DisplayName, Description, CustomClass, and Data. The 'Data' field is set to 'leddisplay-12'.
- Design:** The central workspace showing the MashUp layout. It contains three digital display widgets labeled 'Acceleration X', 'Acceleration Y', and 'Acceleration Z', each showing the value '0.000'. Below these is a widget labeled 'IsMoving' with a green bug icon.
- Bindings:** A section at the bottom showing the data bindings for the MashUp. It includes a 'Things\_302467\_BERM\_raspi' data source, a 'GetPropertyValues' function, and a binding to the 'leddisplay-12' widget.
- Data Properties:** A panel on the right showing the data properties for the selected widget. It includes a 'Filter' dropdown and a 'GetPropertyValue' function.

The bottom right corner of the interface features a floating toolbar with icons for various applications and a cursor icon.



# Zusammenfassung vom Tag 1:

Wir haben ....

- Einen Raspi von Null weg aufgesetzt
- Daten von Sensoren mit Python ausgelesen
- Selbige Daten mittels MQTT an einen Broker versendet
- Gelernt was TWX ist und wie man Daten dort verwaltet
- Ein MashUp gebaut welches diese Daten anzeigt

# Ausblick auf Tag 2:

Andreas Pötscher wird am Vormittag ...

- Denjenigen helfen wo es noch nicht so toll geklappt hat
- E.v.t bei den Dashboards weitermachen
- Das bestehende Beispiel erweitern
- Denjenigen helfen die eigene (andere) Hardware mit haben

Andreas Probst wird am Nachmittag ...

- Einen Einblick geben wie man 3D Visualisierungen mit TWX macht
- Offene Fragen besprechen