
Dokumentation

für das Projekt Digital HHZ 2.0

Zeitraum: Wintersemester 19/20 &
Sommersemester 20

Projektmitglieder: Emre Kocyigit
Marc Kreidler
Manuel Mock
Isabel Staaden
Victor Veal

Projektbetreuer: Prof. Dr. Christian Decker
Sebastian Kotstein

Kurzfassung

Das Ziel des Digital HHZ 2.0 ist die Bereitstellung einer robusten IoT-Umgebung/Infrastruktur als Enabler für IoT-Projekte, Hackathons und Thesen. Diese Dokumentation gilt als Zusammenfassung aller Tätigkeiten im Projekt und als Übersicht und Anleitung für nachfolgende Projekte. Umgesetzt und in dieser Dokumentation beschrieben sind die Publikation von Daten auf der Digital-HHZ-Webseite, das Plug'n'Play Device und Appliance Management, das Konzept und die Implementierung einer nachhaltigen Datenhaltung, die teilautomatisierte Fehlererkennung und das Testen von Appliances.

Keywords: Dokumentation, Digital HHZ, Digitalisierung, Jahresprojekt, IoT, Appliance

Abbildungsverzeichnis

Abbildung 1 Digital HHZ Architekturbild	9
Abbildung 2 MQTT	12
Abbildung 3 MQTT-Dash für Android	13
Abbildung 4 MQTT.fx.....	14
Abbildung 5 Node-RED-Flow	15
Abbildung 6 Node-RED-Webinterface	15
Abbildung 7 Homie Simulator in Node-RED.....	17
Abbildung 8 Funktion zum Erstellen der Homie Topics	17
Abbildung 9 Autodiscovery der simulierten Tasmota-Lampe	18
Abbildung 10 Ergebnisse DB / DBMS Evaluierung	19
Abbildung 11 TICK-Stack	20
Abbildung 12 Raspberry Pi.....	25
Abbildung 13 M5Stick-C.....	25
Abbildung 14 M5Stick-Programmierungsumgebung UIFlow	26
Abbildung 15 Tasmota-Lampe	27
Abbildung 16 Tasmota Web-Interface	27
Abbildung 17 Dashboard Tabs	29
Abbildung 18 externes Dashboard: Digital HHZ Webseite	29
Abbildung 19: MQTT.fx Broker Status.....	30
Abbildung 20 Dashboard: Monitoring.....	31
Abbildung 21 Monitoring durch Service Discovery.....	31
Abbildung 22 Monitoring M5-Sticks	32
Abbildung 23 Einstellungen Switch-Node	33
Abbildung 24 Discovery Node: Installation	36
Abbildung 25 Discovery Node: Übersicht.....	37
Abbildung 26 Discovery Node: Abhängigkeiten.....	38
Abbildung 27 Discovery Node: Bearbeiten.....	38
Abbildung 28 avahi-browse -ak.....	39
Abbildung 29 Service Discovery Flow.....	40
Abbildung 30 Discovery Node: Switch Node	40
Abbildung 31 Discovery Dashboard (Beispieldarstellung)	41
Abbildung 32 Announce Node: Bearbeiten.....	42
Abbildung 33 Announce Node: Flow	43
Abbildung 34 Tasmota mDNS Service Announcement	43
Abbildung 35 Dashboard Simulation	45
Abbildung 36 Daten Replay	46
Abbildung 37 Dashboard ThingsBoard Devices	47
Abbildung 38 Dashboard ThingsBoard Appliance.....	48

Abbildung 39 ThingsBoard Dashboard	49
Abbildung 40 ThingsBoard Gerät öffnen	50
Abbildung 41 Eigenschaften M5Stick	50
Abbildung 42 M5Stick eingehende Verbindungen konfigurieren.....	51
Abbildung 43 Beziehungen im HHZ	53
Abbildung 44 ER Modell.....	54
Abbildung 45 Telegraf Konfiguration	57
Abbildung 46 Chronograf Explorer	58
Abbildung 47 Chronograf Dashboard.....	59
Abbildung 48 MQTT Datenfluss	65

Tabellenverzeichnis

Tabelle 1 Beispiel der Homie-Topics für eine Tasmota-Lampe.....	18
Tabelle 2 Eigenschaften für M5Stick	51
Tabelle 3 Datenwerte	54
Tabelle 4 Fehlerszenarien	62

Inhaltsverzeichnis

Kurzfassung.....	2
Abbildungsverzeichnis.....	3
Tabellenverzeichnis	5
Inhaltsverzeichnis	6
1 Digital HHZ 2.0.....	8
1.1 Trello Dokumentation und GitHub Repository	8
1.2 Architektur.....	9
1.3 Zugang zum Digital-HHZ-Netzwerk	10
2 Verwendete Softwarekomponenten	12
2.1 MQTT.....	12
2.2 Node-RED	14
2.3 Homie Konvention	16
2.4 InfluxDB	19
2.5 Chronograf.....	20
2.6 Telegraf.....	21
2.7 Kapacitor	21
2.8 Avahi	21
2.9 ThingsBoard	24
3 Verwendete Hardwarekomponenten.....	25
3.1 Raspberry Pi	25
3.2 M5Sticks.....	25
3.3 Tasmota-Lampe.....	27
4 MQTT-Broker Mosquitto.....	28
4.1 Installation	28
4.2 Bridging	28
5 HHZ Dashboard	29
5.1 Digital HHZ.....	29
5.2 Monitoring	29
5.3 Service Discovery	34
5.4 Simulation.....	45

5.5	InfluxDB Chronograf	46
5.6	ThingsBoard	46
5.7	ThingsBoard Devices	47
5.8	ThingsBoard Appliances	48
6	Device Management	49
6.1	Unterscheidung Geräte und Objekte	49
6.2	Anlegen eines Gerätes	49
6.3	Zuordnung physisches Gerät zu Datenobjekt.....	52
6.4	Dashboard.....	52
6.5	ThingsBoard Dokumentation	53
7	Datenmanagement.....	54
7.1	ER-Modell	54
7.2	Wie sind die Daten zu verstehen?	54
7.3	Wie sind die Daten zu verarbeiten?.....	55
7.4	InfluxDB	55
7.5	Telegraf.....	56
7.6	Chronograf.....	58
7.7	Backup und Dropbox Uploader	59
7.8	DigitalHHZ2MonthlyUploader Dropbox developer App	61
7.9	Daten-Replay	61
8	Fehlermanagement im Digital HHZ	62
9	Aufsetzen einer neuen Appliance	63
9.1	Aufsetzen des Raspberry Pis	64
9.2	Node-RED Installation	64
9.3	Erklärung des MQTT Datenflusses	65
9.4	Broker Konfiguration	65
9.5	Erklärung der Topic Struktur.....	66

1 Digital HHZ 2.0

Das Herman-Hollerith-Zentrum (HHZ) soll als Forschungszentrum für Digitale Transformation auch infrastrukturell ein positives Beispiel für Digitalisierung sein. Deshalb sollen smarte Umgebungen geschaffen werden, in denen digitale Dienste, mittels durch Sensoren und Aktoren erzeugter Daten, die Interaktion der Nutzer mit dem Gebäude bzw. der Lern- und Arbeitsumgebung fördern und die Nutzer im Alltag unterstützen.

In Zukunft soll das Digital HHZ mit weiteren digitalen Diensten basierend auf dem *Appliance-Ansatz* erweitert werden. In der Theorie versteht man unter einer Appliance eine Hardwarekomponente, die auf eine Software optimiert wurde. Die Idee ist demzufolge zukünftige digitale Dienste auf einer Hardwarekomponente (z.B. Raspberry Pi) zu implementieren. Appliances am Digital HHZ sind in diesem Sinne „greifbar“ bzw. „zum Anfassen“, weil jeder Raspberry Pi für einen digitalen Dienst steht.

In dieser Dokumentation werden die einzelnen Bestandteile (Software- und Hardwarekomponenten) des Digital HHZs, sowie weitere Technologien und ihre Installation beschrieben.

1.1 Trello Dokumentation und GitHub Repository

Diese Gesamtdokumentation leitet sich größtenteils von unserem online Aufgaben- und Verwaltungsdienst *Trello* ab und verweist auf Ressourcen in unserem *GitHub*.

Die Zielvorgaben des Projekts, sowieso Arbeitspakete und weitere Kommentare zum Projekt wurden auf Trello dokumentiert. Unser Trelloboard kann für die weitere Entwicklung des Digital HHZs verwendet werden.

<https://trello.com/b/O8KbgCGB/digital-hhz-20>

Screenshots, Config-Dateien, Code-Snippets und weitere Ressourcen sind auf unserem GitHub Repository zu finden.

<https://github.com/digitalhhz/DigitalHHZ2>

1.2 Architektur

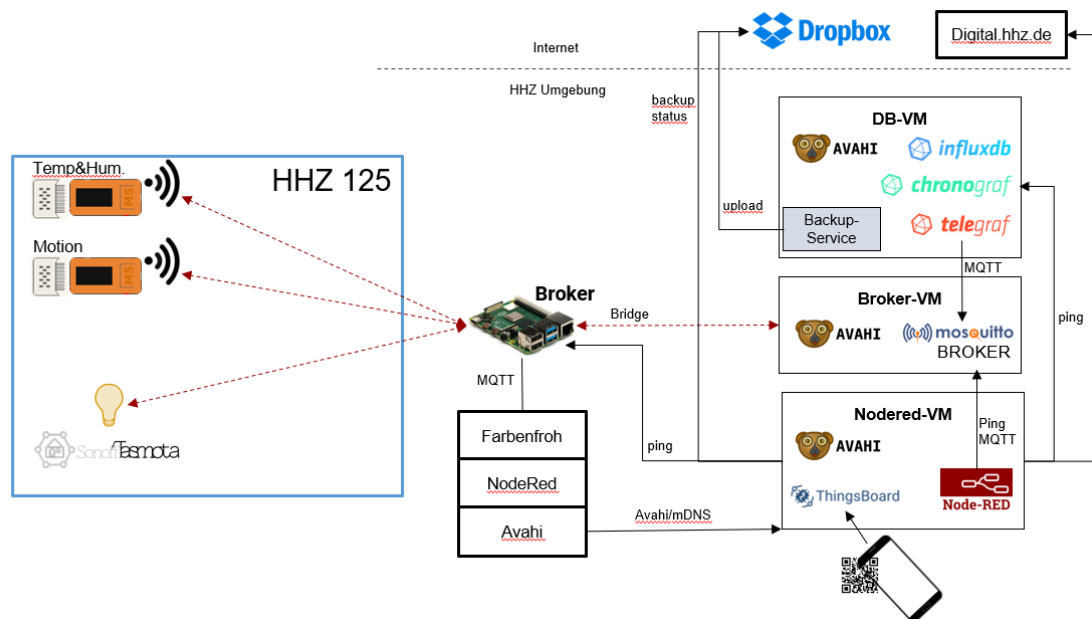


Abbildung 1 Digital HHZ Architekturbild

Abbildung 1 zeigt ein Gesamtbild der verwendeten Technologien sowie die Interaktion der Komponenten untereinander. Die linke Seite stellt die aktuelle Implementierung im Raum 125 dar. Hier sind die Sensoren (M5Sticks) und Aktoren (Tasmota-Lampe) installiert. Die Sensoren und Aktoren kommunizieren über MQTT mit dem lokalen MQTT-Broker, der auf Mosquitto basiert und welcher im Raum 121 auf einem physischen Server (Raspberry Pi) installiert wurde. Die Daten der Sensoren und Aktoren werden auf weitere Mosquitto-Broker gespiegelt (Bridging). Ein zweiter Mosquitto-Broker wird im Smart-Lab auf einer virtuellen Maschine (*Broker-VM*) gehostet. Für den Zugang auf die virtuelle Maschine wird auf das folgende Kapitel (Zugang zum Digital-HHZ-Netzwerk) verwiesen.

Die Digital-HHZ-Webseite (externes Dashboard), welche von außerhalb des Digital-HHZ-Netzwerks (extern) über digital.hhz.de zu erreichen ist, wird auf einem Strato-Server gehostet. Auf dieser Seite werden die aktuell gemessenen Sensorwerte pro Raum aufgelistet und das Projekt kurz vorgestellt. Die Digital-HHZ-Webseite sowie das interne Dashboard wurden mit der Low-Code-Plattform Node-RED entwickelt. Das interne Dashboard wird auf der *Nodered-VM* gehostet. Das interne Dashboard kann nur aufgerufen werden, wenn man im Digital-HHZ-Netzwerk eingeloggt ist.

Auf den Sensoren und Aktoren werden QR-Codes angebracht. Mithilfe dieser können Gerätedaten, sowie deren Zugehörigkeit nachvollzogen werden, indem ein QR-Code-Scanner, bspw. Smartphone, das im Digital-HHZ-Netzwerk eingeloggt ist, diesen QR-

Code scannt. Man wird daraufhin automatisch auf eine gerätespezifische Seite innerhalb des internen Dashboards weitergeleitet und erhält alle notwendigen Informationen über das Gerät selbst und dessen Zugehörigkeit. Die Daten hierfür stammen aus der Device-Management-Software *ThingsBoard*, die ebenfalls auf der *Nodered-VM* installiert wurde. Die Daten werden vom internen Dashboard über die ThingsBoard-API abgerufen.

Auf der *DB-VM* wird der Backup-Service gehostet, welcher aus dem Zeitreihen-Datenbank-Management-System InfluxDB, sowie den weiteren Komponenten des TICK-Stacks (Telegraf, Kapacitor und Chronograf) besteht. Dieser Dienst holt sich über Telegraf von der *Broker-VM* alle Werte und speichert sie in einer Datenbank ab. Zu jedem Monatsanfang wird ein Backup der Datenbank erstellt und automatisch an Dropbox hochgeladen. Backups können jedoch auch manuell erstellt werden. Über Chronograf wurden Dashboards für die Analyse und den csv-Export der Daten erstellt und können intuitiv über das Web-Interface von Chronograf erweitert werden.

Jede VM sowie jeder Raspberry Pi (ob Appliance oder Broker-Pi) muss seine angebotenen Dienste über mDNS/DNS-SD (z.B. Avahi) veröffentlichen. Avahi wird benötigt, um die Geräte/Services im Digital-HHZ zugreifbar zu machen, ohne dass die IP-Adresse bekannt ist.

1.3 Zugang zum Digital-HHZ-Netzwerk

Hinweis: Alle erforderlichen Passwörter sind auf einem separaten Dokument dokumentiert. Hierfür auf die Verantwortlichen des SmartLab zugehen.

Das SmartLab ist die virtuelle Umgebung, auf welcher die drei virtuellen Maschinen in Abbildung 1 gehostet werden. Um auf das SmartLab von außerhalb des Netzwerks zuzugreifen, müssen die folgenden Schritte befolgt werden.

1. Verbinden Sie sich über VPN mit dem Campus-Network. Anleitung: <https://wiki.reutlingen-university.de/pages/viewpage.action?pageId=21201453>
2. Über Remotedesktopverbindung (RDP) auf dem Windows-Terminal-Server 134.103.214.30:3390 verbinden und mit ihren persönlichen Benutzerdaten für den Terminal Server einloggen. Die Benutzerdaten bekommen Sie vom zuständigen Angestellten oder Professor.

IP/Hostname:	134.103.214.30:3390
User:	yourpersonalusername@smartlab.local
Password:	yourpersonalpassword

-
3. Starten Sie Putty (<https://www.putty.org>) oder einen anderen SSH-Client, der auf dem Terminal Server installiert ist, um sich mit der gewünschten VM zu verbinden. Verwenden Sie folgende Verbindungsparameter:

Hostname	IP
broker.digitalhhz.smartlab.local	10.0.103.50
nodored.digitalhhz.smartlab.local	10.0.103.60
database.digitalhhz.smartlab.local	10.0.103.70

Hinweis:

Für die Installation von neuer Software muss der root-User verwendet werden, weil der User digitalhhz kein sudo ausführen kann.

2 Verwendete Softwarekomponenten

Dieses Kapitel stellt die im Projekt verwendeten und evaluierten Softwarekomponenten vor. Dazu gehören Technologien wie MQTT, Node-RED und InfluxDB. Wie diese Komponenten im Digital HHZ 2.0 verwendet werden, wird in späteren Kapiteln beschrieben.

2.1 MQTT

MQTT ist ein Publisher-Subscriber-Protokoll. Zentrales Element ist der MQTT-Broker. Clients können am Broker sogenannte Topics abonnieren (subscribe) und neue Nachrichten in ein Topic veröffentlichen (publishen). Siehe Abbildung 2.

Subscribt ein Client ein Topic, bekommt er automatisch die neu eintreffenden Nachrichten zugestellt. Dabei gibt es 3 Quality-of-Service-Stufen (QoS):

- QoS 0 “At most once” Nachricht wird genau einmal verschickt, keine weitere Sicherungsmechanismen
- QoS 1 “At least once” Empfänger bestätigt den Erhalt, Pakete können mehrmals ankommen
- QoS 2 “Exactly once” Nachricht wird garantiert zugestellt, mehr Traffic

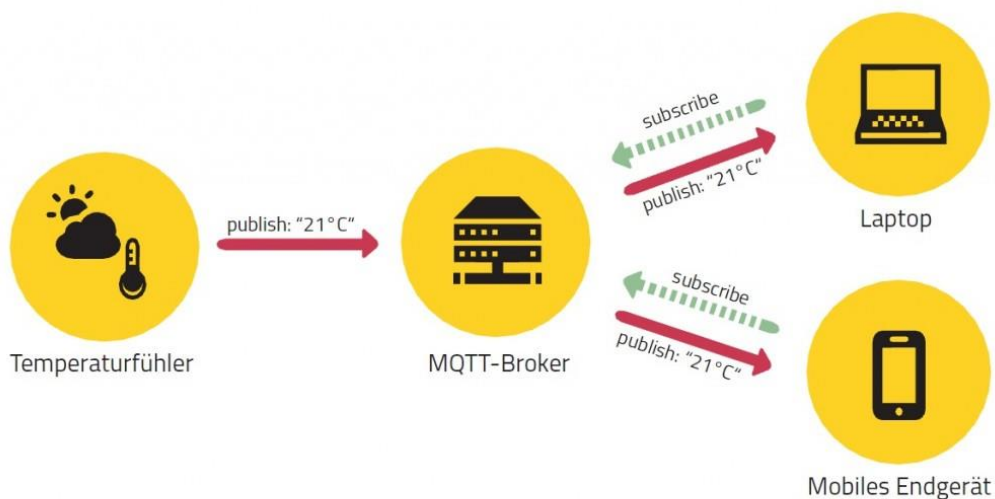


Abbildung 2 MQTT¹

Außerdem ist es möglich eine Nachricht als retained zu verschicken. Das bedeutet, dass die Nachricht in einem Topic auf dem Broker gespeichert wird. Dies ermöglicht, dass

¹ Bildquelle: <https://blog.doubleslash.de/mqtt-fuer-dummies/>, Abgerufen am: 18.09.2020)

z.B. später hinzukommende Teilnehmer die letzte zuvor gesendete Nachricht in einem Topic trotzdem sofort erhalten.

Unter folgendem Link sind umfangreiche weiterführende Informationen zu MQTT zu finden: <https://www.hivemq.com/mqtt-essentials/>.

2.1.1 MQTT-Broker Mosquitto

Für das zentrale Element in der MQTT-Kommunikation verwenden wir den Open-Source MQTT-Broker Eclipse Mosquitto. Mosquitto besitzt eine gute Verbreitung und wird auch von prominenten Projekten im Smart-Home-Bereich verwendet (z.B. Home Assistant, domoticz, FHEM). Das Projekt wird stetig und verlässlich weiterentwickelt. Der Code ist schlank und läuft deshalb auch verlässlich auf Hardware wie dem Raspberry Pi. Neben Open Source MQTT-Brokern gibt es auch kommerzielle Broker wie HiveMQ, den es auch als öffentlich verfügbaren Broker im Internet zum Testen gibt. Wer schnell mal ein MQTT-Gerät oder Programm testen möchte, kann den öffentlichen Broker unter broker.hivemq.com:1883 verwenden und erspart sich so die Installation eines eigenen Brokers.

2.1.2 MQTT-Clients

Zur Nutzung von MQTT in eigenen Applikationen stellt z. B. Eclipse Paho (<http://www.eclipse.org/paho/>) in diversen Sprachen Clients zur Kommunikation mit MQTT-Brokern (z. B. mosquitto, HiveMQ, RabbitMQ, ActiveMQ) zur Verfügung. Es gibt auch Apps für mobile Geräte mit Android oder iOS, die MQTT-Nachrichten senden und empfangen können.

Mit MQTT Dash für Android (<https://play.google.com/store/apps/details?id=net.routix.mqttdash>) wurde z.B. das kleine Dashboard in Abbildung 3 erstellt.

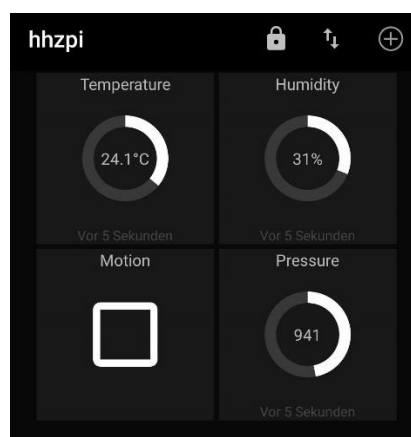


Abbildung 3 MQTT-Dash für Android

2.1.3 MQTT.fx

Auch für das Debugging von MQTT-Kommunikation gibt es Werkzeuge. Neben den einfachen Kommandozeilen-Clients, die mit dem Paket `mosquitto-clients` installiert werden und Python- oder C-Bibliotheken aus dem Paho-Projekt, haben wir häufig das Tool MQTT.fx (<https://mqttfx.jensd.de/>) verwendet, das auf JavaFX basiert und mit einer grafischen Benutzeroberfläche daher kommt. Es bietet vielfältige Möglichkeiten zum Umgang mit MQTT-Nachrichten. Sowohl Publishen als auch Subscriben ist möglich, ebenso kann der Status des MQTT-Brokers überwacht werden, sofern der Broker dies über das Topic `$SYS` unterstützt.

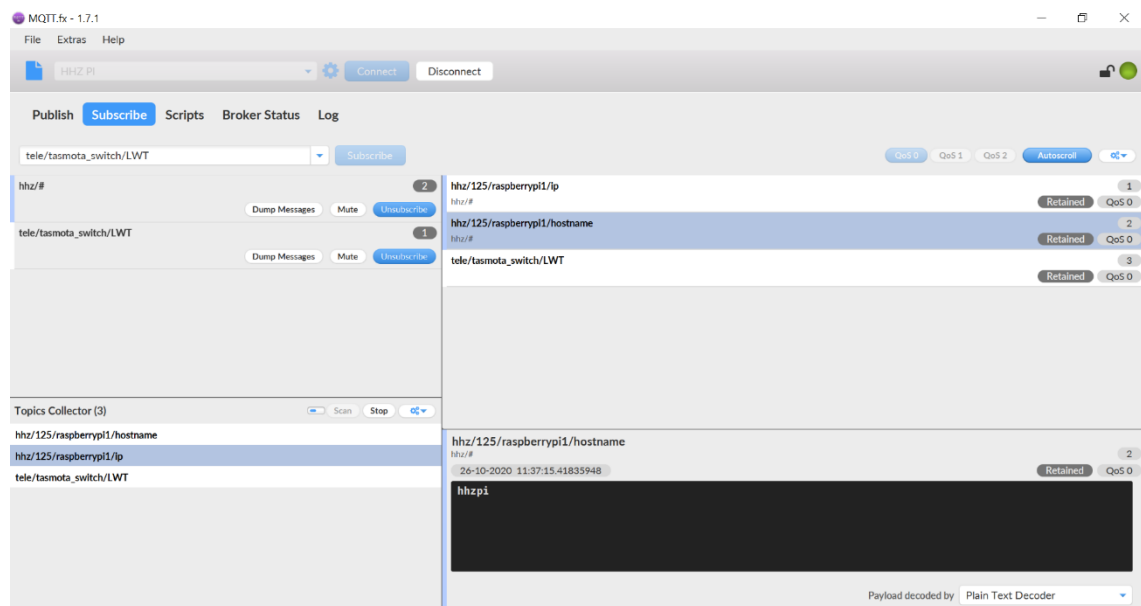


Abbildung 4 MQTT.fx

2.1.4 Mqttwarn

Um ein Monitoring von Services umzusetzen, wurde auch die Lösung Mqttwarn (<https://github.com/jpmens/mqttwarn>) evaluiert. Die Installation und Konfiguration gestalteten sich als sehr fehleranfällig (Compiler-Fehler), unzuverlässig und wenig intuitiv. Da ein Monitoring das zuverlässige Erkennen von Fehlern erlauben soll und nicht selbst die Ursache von vielen Fehlern sein soll, wurde Mqttwarn als Lösung für das Monitoring verworfen. Stattdessen wurde das Monitoring über MQTT und Ping ein Teil des Dashboards.

2.2 Node-RED

Node-RED ist eine datenstromorientierte Low-Code-Entwicklungsumgebung, um IoT-Geräte, APIs und Services zu verknüpfen. Low-Code bedeutet Anwendungen mittels

visueller Elemente statt mittels einer textuellen Programmiersprache modellbasiert zu entwickeln. Dies erlaubt es auch Nicht-Programmierern schnell zu einer Lösung zu kommen, was ideal für den Einsatzzweck Hackathon ist. Erfahrene Programmierer hingegen schätzen die Erweiterbarkeit durch eigenen Code, da es sich auch um eine Open-Source-Software handelt.

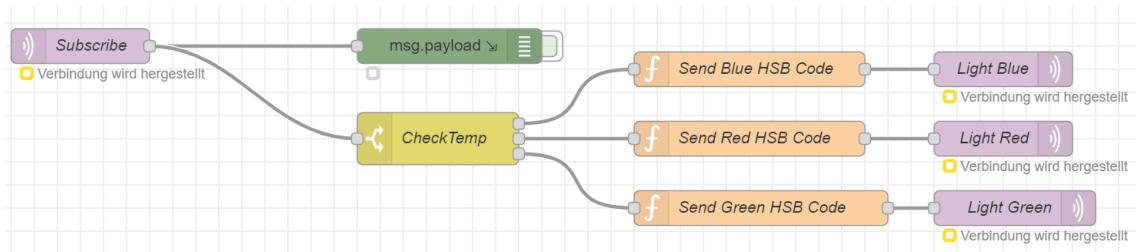


Abbildung 5 Node-RED-Flow

Die Anwendung läuft browser-basiert und ermöglicht es dem Anwender auf eine sehr einfache und benutzerfreundliche Art „Nodes“ miteinander zu verknüpfen und somit „Flows“ (Programme) zu erstellen.

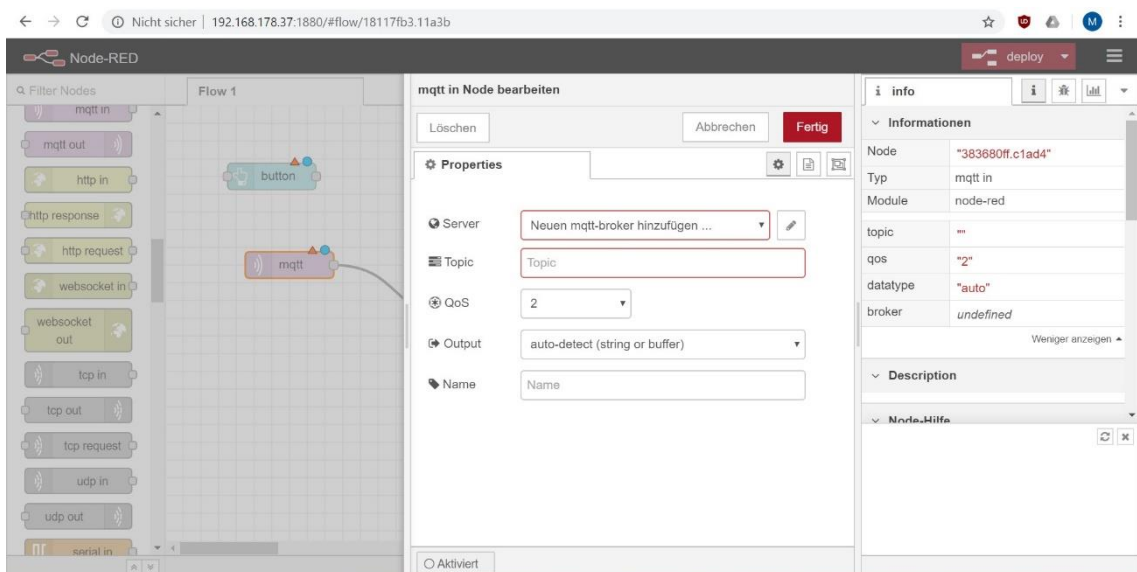


Abbildung 6 Node-RED-Webinterface

Node-RED kann auf verschiedenen Betriebssystemen wie z.B. Windows oder Linux installiert werden. Für unseren Use Case wurde Node-RED auf einem Raspberry Pi eingerichtet. Es lässt sich mit wenigen Schritten schnell installieren. Da Node-RED, wie auch MQTT, bei IBM entwickelt wurde, ist es sehr gut für die Verarbeitung von MQTT-Messages geeignet und verwendet aus MQTT bekannte Paradigmen, wie z.B. Topics. Es gibt eine vorinstallierte Palette an Nodes, aber man kann auch aus Bibliotheken weitere Nodes laden, wie z.B. für die Steuerung von Philips-Hue-Lampen.

2.3 Homie Konvention

2.3.1 Beschreibung

MQTT an sich legt nicht fest, wie der Inhalt der einzelnen Topics aufgebaut ist. Dies kann zum Vorteil sein, weil Prototypen verhältnismäßig schnell umsetzbar sind und auf Entwicklerseite nicht zu viel Vorwissen benötigt wird. Andererseits ist es hierdurch schwierig größere Projekte zu managen. Bei großen Projekten sollte deshalb eine Konvention festgelegt werden, wie die Topics aufgebaut werden. Eine mögliche Konvention ist die *Homie convention*. Diese legt nicht nur fest wie die Topics aufgebaut sein müssen, sondern auch wie Nachrichten verschickt werden (z.B. retained und QoS) oder welche Informationen von einem IoT-Gerät bereitgestellt werden müssen um z.B. Autodiscovery über MQTT zu ermöglichen.

Die komplette Homie convention ist unter folgenden Link zu finden:

<https://homieiot.github.io/>

2.3.2 Umsetzungsmöglichkeit Homie im Digital HHZ

Im Rahmen des Projektes wurde geprüft, inwieweit Homie auch im Digital HHZ umsetzbar ist. Vorteilhaft ist, dass es für Node-RED eine Homie Bibliothek gibt. Dies würde die schnelle Einbindung von Sensoren und Aktoren in eine Appliance innerhalb eines Hackathons stark vereinfachen. Leider lässt sich Homie auf Aktor/Sensorseite nicht so einfach einbinden. Proprietäre Geräte wie Phillips Hue aber auch die Tasmota-Firmware unterstützen (noch) kein Homie. Für Python/MicroPython gibt es zwar Bibliotheken jedoch lassen diese sich auf den M5Sticks leider auch nicht ohne weiteres einbinden.

Innerhalb des Projektes wurde deshalb eine Lösung diskutiert und getestet, die bestimmte Homierelevante Metadaten innerhalb von Node-RED simuliert und bei der von den einzelnen Geräten jeweils nur normale Messwerte etc. „homie-konform“ verschickt werden. In der Praxis hat sich dies als sehr aufwendig herausgestellt, da für ein neues Gerät recht viele Werte händisch in den „Simulator“ eingetragen werden müssten (Abbildung 7 & Abbildung 8). Deshalb wurde der Ansatz für das Digital HHZ erstmal wieder verworfen. Sollten jedoch in Zukunft die M5Sticks und einige der anderen Sensoren/Aktoren Homie nativ unterstützen und es wäre nur ein Teil über den Simulator anzubinden, dann würde die Homie convention wieder von größerem Interesse für das Projekt sein. In Tabelle 1 ist zu sehen, welche Werte beispielsweise von einer Tasmota-Lampe laut Homie zur Verfügung gestellt werden müssten, um ein Autodiscovery (Abbildung 9) zu ermöglichen.

Wir haben uns deshalb für die in Kapitel 0 beschriebene Topicstruktur entschieden

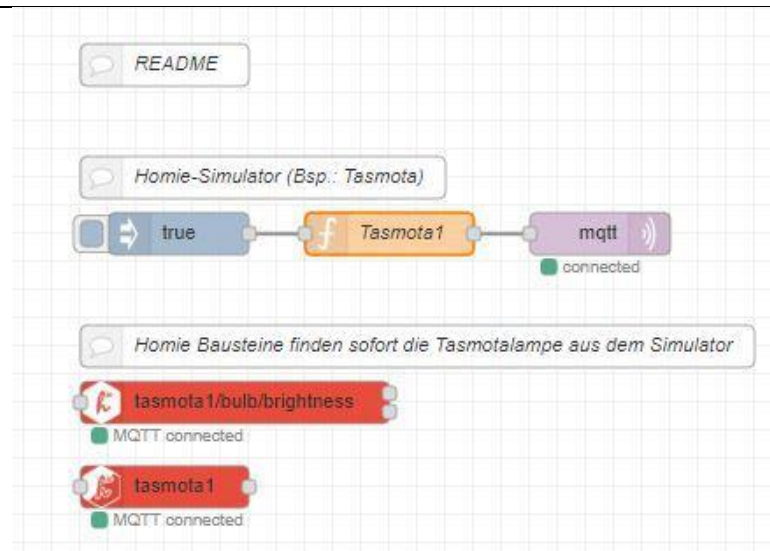


Abbildung 7 Homie Simulator in Node-RED

```

Properties
Name: Tasmota1
Function
1 var s = "/";
2 var deviceID = "tasmota1";
3 //-----Nodes-----
4 var node1 = "bulb";
5 //var node2 = "";
6 //-----Properties-----
7 var prop1 = "power";
8 var prop2 = "brightness";
9 //var prop3 = "property1";
10
11 //-----Device-----
12 var msg1 = { topic: "homie" + s + deviceID + s, "$homie", payload: "custom" };
13 var msg2 = { topic: "homie" + s + deviceID + s, "$name", payload: "Tasmotalampe 1" };
14 var msg3 = { topic: "homie" + s + deviceID + s, "$state", payload: "ready" };
15 var msg4 = { topic: "homie" + s + deviceID + s, "$nodes", payload: "bulb" };
16 var msg5 = { topic: "homie" + s + deviceID + s, "$extensions", payload: "" };
17
18 //-----Node-----
19 var msg6 = { topic: "homie" + s + deviceID + s + node1 + s, "$name", payload: "SH430" };
20 var msg7 = { topic: "homie" + s + deviceID + s + node1 + s, "$type", payload: "Smartbulb" };
21 var msg8 = { topic: "homie" + s + deviceID + s + node1 + s, "$properties", payload: prop1 + ", " + prop2 };
22
23 //-----Property1-----
24 var msg9 = { topic: "homie" + s + deviceID + s + node1 + s + prop1 + s, "$name", payload: "power" };
25 var msg10 = { topic: "homie" + s + deviceID + s + node1 + s + prop1 + s, "$datatype", payload: "boolean" };
26
27 //-----Property2-----
28 var msg11 = { topic: "homie" + s + deviceID + s + node1 + s + prop2 + s, "$name", payload: "brightness" };
29 var msg12 = { topic: "homie" + s + deviceID + s + node1 + s + prop2 + s, "$datatype", payload: "integer" };
30 //+++++optional+++++
31 var msg13 = { topic: "homie" + s + deviceID + s + node1 + s + prop2 + s, "$format", payload: "0:100" };
32 var msg14 = { topic: "homie" + s + deviceID + s + node1 + s + prop2 + s, "$settable", payload: "true" };
33 var msg15 = { topic: "homie" + s + deviceID + s + node1 + s + prop2 + s, "$unit", payload: "%" };
34
35
36 //Returning Msg-objects for MQTT-Topic and MQTT-Payload generation
37 return [[msg1,msg2,msg3,msg4,msg5,msg6,msg7,msg8,msg9,msg10,msg11,msg12,msg13,msg14,msg15]];
38
39
40
41
42

```

Abbildung 8 Funktion zum Erstellen der Homie Topics

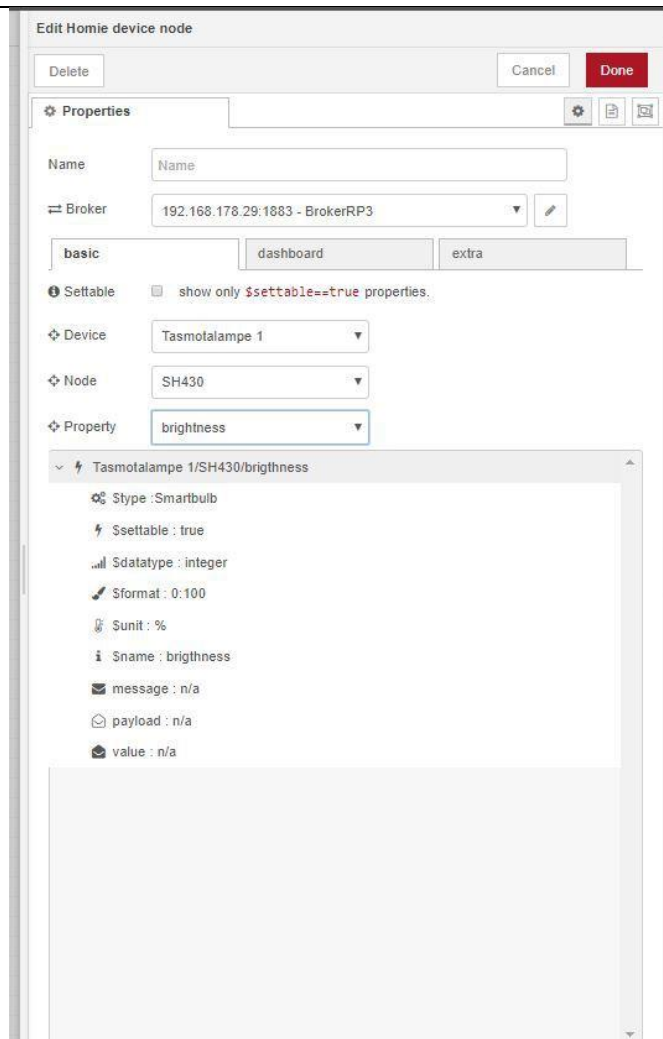


Abbildung 9 Autodiscovery der simulierten Tasmota-Lampe

Tabelle 1 Beispiel der Homie-Topics für eine Tasmota-Lampe

Root-Topic	Device ID	Device Attribute	Payload	Node	Node-Attribute	Payload	Property	Property Attribute	Payload
hhz	1	\$homie	custom						
hhz	1	\$name	Tasmotalampe 1						
hhz	1	\$state	ready						
hhz	1	\$nodes	bulb						
hhz	1	\$extensions	""						
hhz	1			bulb	\$name	SH340			
hhz	1			bulb	\$type	SWISSONE-SH340			
hhz	1			bulb	\$properties	power, brightness, colour			
hhz	1			bulb			power	\$name	Power
hhz	1			bulb			power	\$datatype	boolean
hhz	1			bulb			power	\$settable	true
hhz	1			bulb			brightness	\$name	Brightness
hhz	1			bulb			brightness	\$datatype	integer
hhz	1			bulb			brightness	\$settable	true
hhz	1			bulb			brightness	\$format	0.06944444
hhz	1			bulb			brightness	\$unit	%

2.4 InfluxDB

Um eine nachhaltige Datenhaltung zu gewährleisten und eine nachhaltige Arbeit mit den Daten zu ermöglichen, ist ein Datenbankmanagement (DBMS) notwendig. Hierfür führten wir eine Evaluierung durch. Die folgende Abbildung zeigt die Ergebnisse unserer Evaluierung.

Anforderung	<u>InfluxDB</u>	<u>MemSQL</u>	<u>Riak IoT</u>	<u>OpenTSDB</u>
<i>Zeitreihen</i>	Platz 1 im Ranking	Nicht speziell für Zeitreihen, aber für gut feste zugeordnete Werte geeignet (also Sensorwert zu Uhrzeit)	Platz 17 im Ranking	Platz 7 im Ranking
<i>Big Data/Echtzeit</i>	Für viele, mit Zeitstempel versehene Daten geeignet	Echtzeitanalysen & Pipelines für Big Data	Für schneller Lese- & Schreibvorgänge und analytische Verarbeitung von Streams	Für viele, mit Zeitstempel versehene Daten geeignet
<i>Skalierbarkeit</i>	skalierbar	skalierbar	hochskalierbar	skalierbar
<i>Flexibilität</i>	Zeitfelder in DB sind fest, aber kein Schema	SQL, Schema ist relational	NoSQL, daher Schema flexibel	Zeitfelder in DB sind fest, aber kein Schema
<i>Verfügbarkeit</i>	Open Source Edition läuft nur auf einem <u>Node</u>	Verteilter Speicher	Verteilter Speicher	Verteilter Speicher

Abbildung 10 Ergebnisse DB / DBMS Evaluierung ²³⁴⁵

Unsere Wahl fiel auf InfluxDB, da es die Anforderung von Zeitreihen und Big Data/ Echtzeit am besten erfüllt. InfluxDB ist eine Time-Series Datenbank, welche speziell für Zeitreihendaten entwickelt wurde. InfluxDB kommt in vielen Anwendungsfällen als Datenspeicher zum Einsatz, bei denen große Datenmengen mit Zeitstempel verarbeitet werden, beispielsweise DevOps-Überwachungsdaten, Protokolldaten, Anwendungsmetriken, Daten von IoT-Sensoren oder Echtzeitanalysedaten. Der Datenspeicher stellt auch andere Funktionen bereit, unter anderem Datenaufbewahrungsrichtlinien.⁶

² <https://db-engines.com/en/ranking/time+series+dbms>

³ <https://www.computerweekly.com/de/ratgeber/Uebersicht-und-Vergleich-Datenbankoptionen-fuer-das-Internet-of-Things-IoT>

⁴ <https://docs.memsql.com/v7.0/introduction/documentation-overview/>

⁵ <http://opentsdb.net/docs/build/html/index.html>

⁶ http://support.ptc.com/help/thingworx_hc/thingworx_8_hc/de/index.html#page/ThingWorx/Help/Composer/DataStorage/PersistenceProviders/using_influxdb_as_the_persistence_provider.html

Die weiteren Komponenten des TICK-Stacks ermöglichen das Sammeln (Telegraf), Darstellen & manuelles Analysieren (Chronograf), Verarbeiten & automatisches Analysieren (Kapacitor) von Daten.

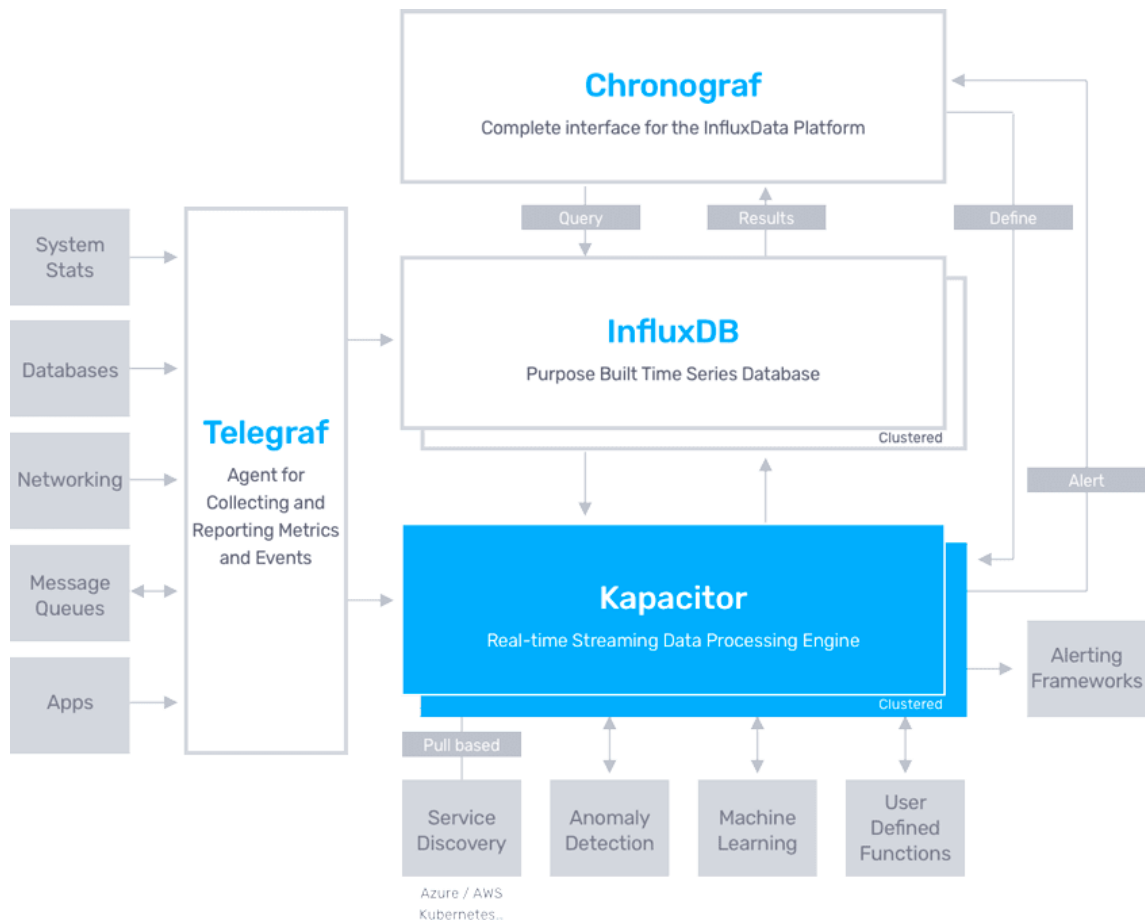


Abbildung 11 TICK-Stack ⁷

2.5 Chronograf

ChronoGraf wurde wie InfluxDB von der InfluxData entwickelt und ist eine Webapplikation, das zur Visualisierung der überwachten Daten angewendet werden kann. Darüber hinaus verfügt die Applikation über Alarmierungs- als auch Automatisierungsfunktionen.⁸

⁷ Bildquelle: <https://www.influxdata.com/time-series-platform/>

⁸ <https://docs.influxdata.com/chronograf/v1.8/>

2.6 Telegraf

Auch Telegraf wurde vom Entwicklerteam InfluxData entwickelt. Telegraf ist ein Serveragent, welcher für das Sammeln und Senden von Metriken und Events von Datenbanken, Systemen und IoT-Sensoren genutzt werden kann. In diesem Projekt wird Telegraf genutzt, um die MQTT-Nachrichten der M5Sticks zu erfassen.

2.7 Kapacitor

Kapacitor ist ebenfalls Teil des TICK-Stacks von InfluxData und wurde von uns vor allem unter dem Aspekt eines Datenreplays zum Simulieren von echten Sensorwerten evaluiert. Dazu haben wir Kapacitor auf dem Datenbankserver installiert und getestet. Die Konfiguration ist nicht für eine einfache Replayfunktion geeignet. Da das meiste in unterschiedlichen Konfigurations- und Skriptdateien im nicht so geläufigen TOML-Format definiert werden muss. Kapacitor ist eher für das Aufzeichnen und Analysieren der Streams geeignet und um Anomalien zu erkennen und zu signalisieren (Alerting). Da das Zusammenspiel von InfluxDB, Telegraf und Kapacitor sehr fehleranfällig ist und das Debugging auch relativ aufwendig ist, haben wir uns gegen die dauerhafte Verwendung von Kapacitor entschieden und stattdessen diese Funktionen über ein leicht verständliches Python-Skript (Kapitel 7.9) und über Funktionen im Node-RED-Dashboard (Kapitel 5.4) realisiert.

2.8 Avahi

2.8.1 Einführung in Service Discovery

Wozu benötigt man ein Service Discovery?

In dynamischen Adhoc-Netzen wie sie z.B. bei einem Hackathon entstehen, sind in einem Netzwerk viele neue Geräte verfügbar, auf deren Dienste (Services) zugegriffen werden soll. In IP-basierten Netzen mit einer dynamischen IP-Konfiguration über einen DHCP-Server ergibt sich das Problem, dass nach der Zuweisung der IP-Adresse diese dem Benutzer nicht bekannt ist. Verfügt das Gerät über ein Display oder andere Ausgabemöglichkeit (Sprachausgabe) kann darüber die IP-Adresse ausgegeben werden. Viele IoT-Geräte wie smarte LED-Lampen, Steckdosen, Lautsprecher, Sensoren oder auch Raspberry Pis ohne angeschlossenes Display besitzen keine Ausgabemöglichkeit und der Zugriff über ein Remote-Terminal wie ssh erfordert die Adressierbarkeit des Gerätes. In solchen Fällen hilft es, wenn die Geräte einen eindeutigen Identifier haben, über den Sie adressiert werden können. Eine Namensauflösung (Name Resolution) erfordert entweder

ein zentrales Namensverzeichnis (DHCP-Server) oder ein Protokoll, dass es Geräten im Netz erlaubt dezentral auf eine Anfrage zu antworten. Da diese Probleme schon lange bestehen, z.B. auch bei der Verwendung von Netzwerkdruckern, gibt es auch schon lange Protokolle, die dafür eine Lösung bieten.

2.8.2 Zeroconf, Bonjour oder Avahi

Zeroconf, Bonjour und Avahi sind Implementierungen eines Protokolls zur Selbstkonfiguration in Adhoc-Netzen auf Basis von IPv4 und IPv6. Es handelt sich dabei um eine Zusammenstellung bestehender Protokolle auf Basis von IP, DNS und NAT, um Services, die in einem IP-Netz bereitgestellt werden, automatisch erkennen zu können, ohne dass manuelle Konfiguration durch den Anwender nötig ist. Das Protokoll wird im RFC6762 (<http://tools.ietf.org/html/rfc6762>) beschrieben.

Diese Protokolle sind auch für den IoT-Bereich interessant, wenn Sensoren und Aktoren sich vernetzen und im lokalen Netzwerk bekannt machen sollen.

Bonjour, ist der Nachfolger von AppleTalk und wird auch als Zeroconf (Zero Configuration Networking) bezeichnet. Obwohl Bonjour von Apple stammt, gibt es das Protokoll nicht nur für Mac OS X, sondern auch für Linux (Avahi) und Windows (Bonjour). Wird auf einem Windowsrechner z.B. iTunes installiert, wird auch der Bonjour-Dienst aktiviert. Inzwischen wurde es von Apple unter der Apache-2.0-Lizenz als Open Source freigegeben.

Avahi ermöglicht die automatische IP-Adress- und Hostname-Vergabe, sowie das Bekanntmachen und Entdecken von Diensten (Service Discovery).

Bei Avahi teilen die Diensteanbieter ihre Dienste von sich aus mit (Annoncierung), so dass sie von anderen Stationen automatisch gefunden werden können. Z.B. teilt der Raspberry Pi auf dem der MQTT-Broker installiert wurde mit, dass unter seiner Adresse ein MQTT-Broker verfügbar ist oder ein Netzwerkdrucker teilt mit, dass unter seinem Hostnamen oder seiner IP-Adresse ein Druckerdienst verfügbar ist. Das Protokoll benötigt im Gegensatz zu DHCP oder DNS keinen zentralen Server. Die Dienste melden sich dynamisch an und ab, ohne dass für den Nutzer Konfigurationsaufwand entsteht.

2.8.3 Allgemeine Funktionsweise

Avahi sendet und empfängt Multicast-DNS-Pakete (mDNS) an die Multicast-Adresse 224.0.0.251 (IPv4) bzw. ff02::fb (IPv6) am Port 5353. Bei Multicast bleiben die Pakete dabei im selben Subnetz, weshalb sich alle Clients im selben Subnetz befinden müssen, damit die Kommunikation zwischen allen Geräten mit Avahi ermöglicht wird.

Um Avahi-Daten über Subnetzgrenzen hinweg übertragen zu können, verwendet man DNS-übliche Unicasts. Dieses Vorgehen wird Wide Area Bonjour (WAB) genannt und erfordert einen DNS-Server.

Ursprünglich war Bonjour ausschließlich für Heimnetzwerke gedacht. Doch mit der Verbreitung von MacBooks, iPhones und iPads verbreitete sich Bonjour auch in größeren Netzwerkumgebungen. Avahi ist auch standardmäßig unter der Raspberry Pi Linux Distribution installiert.

2.8.4 IP-Adress-Vergabe

Wie in der Einführung beschrieben ergibt sich in Adhoc-Netzen das Problem, wie die Geräte an ihre IP-Adressen kommen bzw. wie Geräte ohne das Wissen der IP-Adresse angesprochen werden können. Bei IPv6 gibt es dafür die Autokonfiguration per SLAAC. Damit generiert sich jeder Host eine eigene IPv6-Adresse, die nur link-lokal gültig ist. Bei IPv4 kann Bonjour diese Funktion erfüllen. Auch hier weisen sich die Geräte selbst Adressen zu und Bonjour greift im Falle von Adresskonflikten ein.

Bei IPv6 wird der Adressbereich "fe80" verwendet, bei IPv4-Adressen liegt der Adressbereich für link-lokale bei "169.254.0.0/16".

Hat ein Host eine IPv4-Adresse für sich bestimmt, dann macht er diese im lokalen Netzwerk zusammen mit seiner MAC-Adresse per Broadcast bekannt. Die anderen Netzwerk-Teilnehmer aktualisieren dann ihre ARP-Table (Liste mit benachbarten Netzwerk-Teilnehmern). Über das Address Resolution Protocol (ARP) werden in IP-Netzen die MAC-Adressen mit den IP-Adressen verknüpft.

Wird die IPv4-Adresse von einem anderen Host bereits verwendet, dann muss dieses Gerät seine Adresse verteidigen. Dazu antwortet der Host auf den Broadcast. Um manuelles Eingreifen zu verhindern, überlassen Hosts anderen Hosts eine vergebene IP-Adresse, wenn die diese trotzdem haben wollen.

2.8.5 Namensvergabe

Die Namensvergabe über Multicast DNS (mDNS) ermöglicht die eigenständige Zuweisung von Namen in einem Netzwerk ohne Nutzereingriff und Unicast DNS innerhalb eines lokalen Netzwerks. Dazu weisen sich die Geräte selber Namen zu. Häufig Typenbezeichnungen, benutzerdefinierte Computernamen oder auch Teile aus Benutzernamen. Der dabei ablaufende Prüfprozess ähnelt der Zuweisung von link-lokalen IP-Adressen. Immer dann, wenn ein Konflikt auftritt, wird der Name automatisch abgeändert und erneut geprüft. Bei Avahi wird das zum Beispiel durch das automatische Anhängen eines Bindestrichs und der Ziffer 2 an den Hostnamen erreicht oder in dem eine bereits am Ende

des Hostnamens vorhandene Nummer inkrementiert wird. Wenn z.B. in einem Netz 3 Broker den Hostnamen „hhzbroker“ haben wird der 2. Broker, der aktiviert wird den Namen „hhzbroker-2“ zugewiesen bekommen. Kommt nun der 3. Broker hinzu und verkündet, dass er „hhzbroker“ heißt, wird er zuletzt beim Namen „hhzbroker-3“ landen. Dadurch kann über Avahi auch ein einfaches Redundanzverhalten implementiert werden. Damit man bei den Namen zwischen mDNS und DNS unterscheiden kann, bekommen die Namen ein ".local" als Pseudo-Top-Level-Domain (TLD) angehängt. Das heißt, ".local" ist die TLD in einem lokalen Netzwerk.

Bei der Namensauflösung verwendet mDNS das gleiche Protokoll, wie beim Unicast DNS. Dabei stellt der Client einfach eine DNS-Anfrage an die Multicast-Adresse "224.0.0.251" und bekommt vom betreffenden Host eine Antwort zurück. Dabei muss man berücksichtigen, dass diese Adresse nur im jeweiligen Subnetz erreichbar ist.

2.8.6 Service Discovery

Nicht alle Netzwerk-Teilnehmer sind Clients, die nur auf das Internet zugreifen. Es gibt auch Hosts, die Dienste anbieten. Beispielsweise ein Network Attached Storage (NAS), ein Web-Server oder ein Netzwerk-Drucker. Weil die Dienste nicht vorher wissen, von welchen Clients sie benötigt werden, werden die Bekanntmachungen (Annoncen) an die Multicast-Adressen für IPv4 "224.0.0.251" und IPv6 "ff02::fb" und den Port 5353 geschickt. Das bedeutet, der Service macht die Dienste, die ein Computer anbietet im lokalen Netzwerk bekannt. Auf diese Adressen lauschen alle kompatiblen Geräte auf Bekanntmachungen durch Dienste-Anbieter. Genauso kann jeder Computer in einem Netzwerk seinen Hostnamen bekannt machen. Dies ermöglicht einem Anwender den Dienst bzw. den Rechner ohne Kenntnis dessen IP-Adresse anzusprechen. In Adhoc-Umgebungen, lässt sich so im Netzwerk automatisch nach einem Dienst eines speziellen Typs (z.B. MQTT-Broker) suchen und so diesen ohne manuelle Konfiguration und Kenntnis seines Hostnamens oder seiner IP-Adresse ansprechen. Unter <http://www.dns-sd.org/service-types.html> findet man eine Liste der registrierten Service Types.

2.9 ThingsBoard

ThingsBoard (<https://thingsboard.io/>) ist eine Open-Source IoT Plattform zum Entwickeln, Managen und Skalieren von IoT Projekten. Im Kontext des Digital HHZs wird ThingsBoard genutzt, um das Geräte- und Asset-Management zu ermöglichen. Mithilfe von ThingsBoard können beispielsweise Relationen zwischen Geräten definiert, Daten visualisiert, Datenanalysen umgesetzt, Workflows zur Abrufung von REST APIs und vieles mehr umgesetzt werden.

3 Verwendete Hardwarekomponenten

Dieses Kapitel beschäftigt sich mit der Hardware, die im HHZ aufgebaut ist und die Digitalisierung des HHZs ermöglicht. Das wichtigste Element sind dabei die Raspberry Pis, Sensoren und Aktoren ergänzen die Hardwarekomponenten um verschiedene Funktionen.

3.1 Raspberry Pi

Der Raspberry Pi ist ein Einplatinencomputer, welcher von der Raspberry Pi Foundation entwickelt wurde. Ursprünglich wurde dieser mit der Intention entworfen, um Programmieranfängern den Einstieg in Computerhardware zu ermöglichen und ihnen eine Plattform zum Entwickeln zu bieten.



Abbildung 12 Raspberry Pi

Der Raspberry Pi zeichnet sich durch seine kompakte Größe, seinen geringen Preis und seiner Gesamtheit aus, denn er bietet, trotz seiner Größe, verschiedenste Anschlussmöglichkeiten (Ethernet, HDMI, USB, SD) und unterstützt verschiedene Funkprotokolle (Bluetooth und WiFi). Der Raspberry Pi ist somit fast eine komplette Lösung, jedoch bietet er trotzdem genug Offenheit, um ihn selbst zu erweitern (z.B. Betriebssystem, Treiber, Standards).

Im Digital HHZ wird jede Appliance auf einem eigenen Raspberry Pi installiert. So ist auch der Broker auf einem Raspberry Pi zu finden.

3.2 M5Sticks

Der M5Stick-C ist ein portables, einfach zu bedienendes, preiswertes Open-Source-IoT-Entwicklungsboard, das mit einem ESP32-Mikrocontroller betrieben wird. Der M5StickC wird von der Firma M5Stack entwickelt und vertrieben. Er verfügt über einen internen Akku (80mAh) mit Laderegler. Somit kann der ESP32 auch mobil genutzt werden. Lange Laufzeiten sind nur im Deep Sleep Modus möglich,



Abbildung 13 M5Stick-C

weshalb für einen dauerhaften Einsatz der Anschluss eines USB-Netzteils notwendig ist.

Der M5Stick kann mittels sogenannter Hats um verschiedene Sensoren erweitert werden und wird deswegen im Digital HHZ zur Datenerfassung verschiedener Messwerte verwendet.

Das integrierte Display ermöglicht auch, dass die aktuellen Messwerte oder Meldungen direkt auf dem Gerät angezeigt werden können.

Die Programmierung kann über die für M5Stack-Produkte entwickelte Entwicklungsumgebung UIFlow entweder webbasiert (<https://flow.m5stack.com>) oder lokal z.B. auf einem Windows-Rechner mit Hilfe der Software UIFlow-Desktop erfolgen. Das Schreiben (Flashen) der selbst erstellten Programme erfolgt entweder über USB oder drahtlos über WLAN mit Hilfe eines API-Keys (Over-the-Air). Als Programmiersprachen stehen die Low-Code-Programmierung Blockly (<https://developers.google.com/blockly>) und textuelle Programmierung mittels Python (<https://www.python.org>) zur Verfügung.

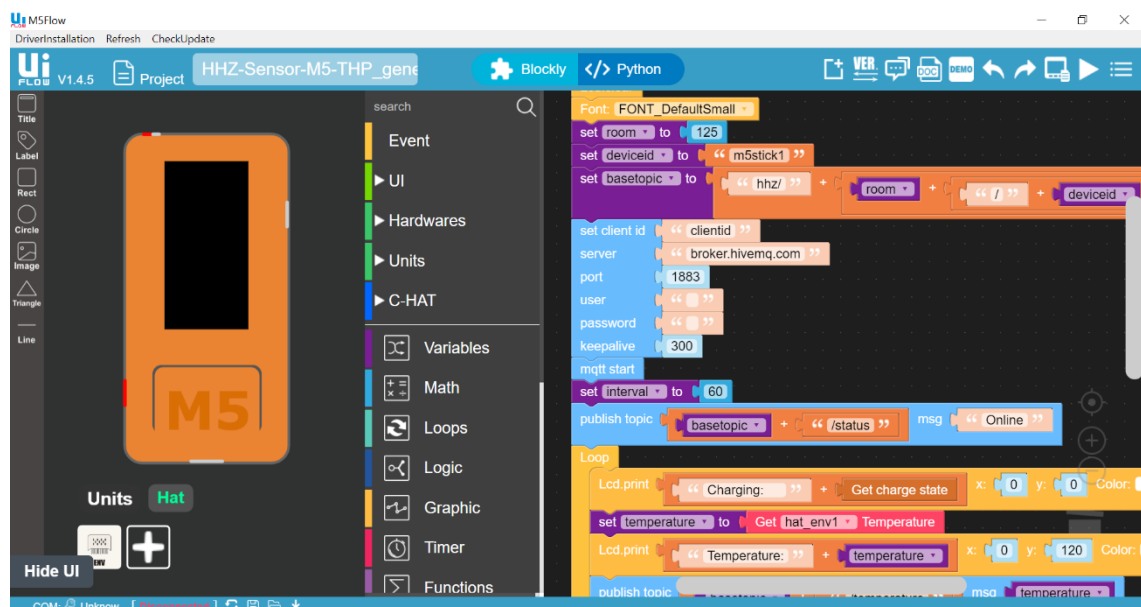


Abbildung 14 M5Stick-Programmierungsumgebung UIFlow

Alternativ kann die Programmierung auch über die Arduino-IDE (<https://www.arduino.cc>) in der Programmiersprache Processing erfolgen. Der M5StickC eignet sich deshalb auch perfekt zum Erlernen der ESP32 Programmierung mit Arduino.

3.3 Tasmota-Lampe

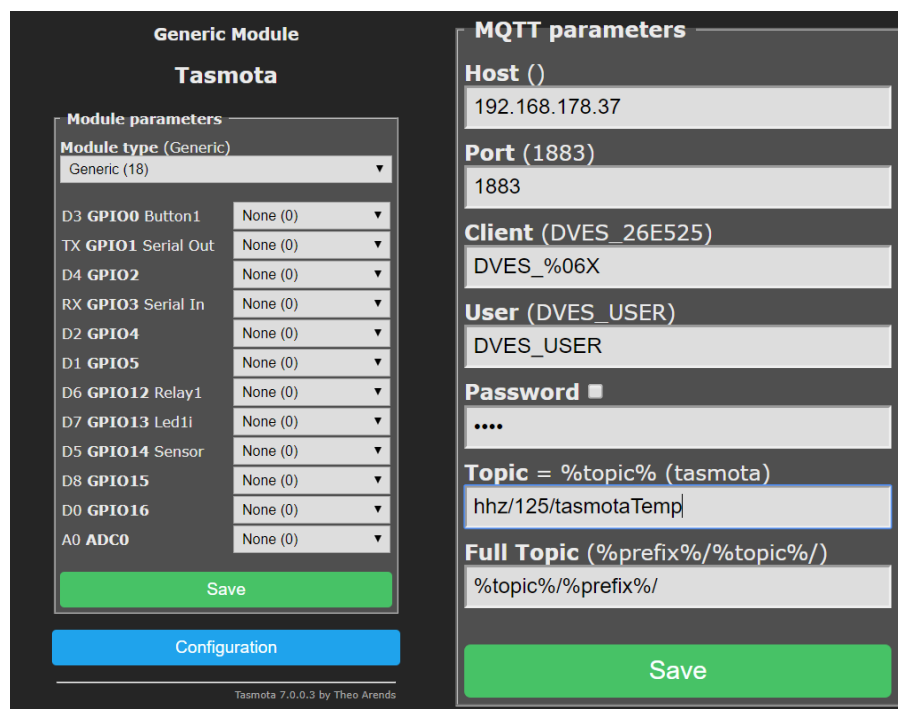
Tasmota (<https://tasmota.github.io/docs/>) ist eine Open-Source-Firmware für ESP8266-basierte Geräte, die von Theo Arends entwickelt und gewartet wird. Alles begann als Sonoff-MQTT-OTA mit einem Commit am 25. Januar 2016. Das Ziel war es, ESP8266-basierte ITEAD-Sonoff-Geräte mit MQTT und Over-the-Air-Firmware auszustatten.



Abbildung 15 Tasmota-Lampe

Tasmota ermöglicht totale lokale Kontrolle mit schneller Einrichtung und Updates durch Steuerung über MQTT, Web UI, HTTP oder seriell. Die Verwendung von Zeitgebern, Regeln oder Skripten kann automatisiert werden und mit Hausautomatisierungslösungen integriert werden. Die Tasmota-Lampe wird im Digital HHZ aufgrund ihrer Erweiterbarkeit und Flexibilität verwendet.

Über das Webinterface der Tasmota-Firmware lässt sich die Tasmota-Geräte bequem konfigurieren. So kann bei den Module-Parametern angegeben werden welche Funktion das Gerät erfüllt und damit festgelegt werden welche Befehle und Daten zur Verfügung stehen und wie sie über MQTT kommuniziert werden.



Generic Module	
Tasmota	
Module parameters	
Module type (Generic)	
Generic (18)	
D3 GPIO0 Button1	None (0)
TX GPIO1 Serial Out	None (0)
D4 GPIO2	None (0)
RX GPIO3 Serial In	None (0)
D2 GPIO4	None (0)
D1 GPIO5	None (0)
D6 GPIO12 Relay1	None (0)
D7 GPIO13 Led1i	None (0)
D5 GPIO14 Sensor	None (0)
D8 GPIO15	None (0)
D0 GPIO16	None (0)
A0 ADC0	None (0)

MQTT parameters	
Host ()	192.168.178.37
Port (1883)	1883
Client (DVES_26E525)	DVES_%06X
User (DVES_USER)	DVES_USER
Password ■
Topic = %topic% (tasmota)	hhz/125/tasmotaTemp
Full Topic (%prefix%/%topic%/)	%topic%/%prefix%/

Abbildung 16 Tasmota Web-Interface

4 MQTT-Broker Mosquitto

4.1 Installation

Um Mosquitto zu installieren muss abhängig vom Betriebssystem das entsprechende Package heruntergeladen und installiert werden. Hierfür wird auf die Mosquitto-Webseite selbst verwiesen, um stets die neuesten Pakete und Installationsanweisungen zu erhalten: <https://mosquitto.org/download/>

4.2 Bridging

Das sogenannte Bridging bei MQTT ermöglicht die Kommunikation zwischen zwei MQTT-Brokern indem einer der Broker sich als Client an den anderen Broker verbindet. Diese Funktion von MQTT wurde genutzt, um alle Sensorwerte von unserer lokalen MQTT Implementierung (gehosted von einem Raspberry Pi) auf eine externe MQTT Implementierung (gehosted auf einer virtuellen Maschine in der SmartLab Umgebung) weiterzuleiten. Durch das Bridging wird auch eine Redundanz des MQTT-Brokers geschaffen d.h. man kann über zwei Broker die Sensordaten abrufen.

Um das Bridging im Broker zu aktivieren müssen folgende Schritte bei nur einem Broker ausgeführt werden. Dabei spielt es keine Rolle auf welchem Broker diese Konfiguration ausgeführt wird.

1) Navigation zum mosquitto Ordner

```
$ cd /etc/mosquitto
```

2) Bearbeitung der mosquitto conf – Datei

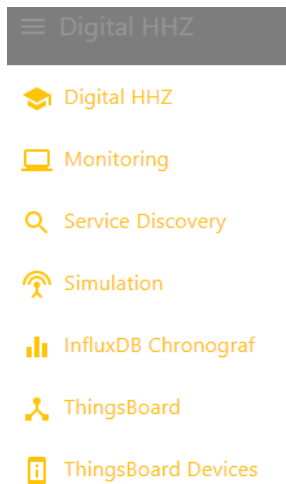
```
$ nano mosquitto.conf
```

3) Einfügen der folgenden Zeilen

```
connection <connection-name>
address <address-of-other-broker>
remote_username <username-for-auth>
remote_password <password-for-auth>
topic # both
```

- *Connection* definiert den Namen der Bridge Verbindung
- *Address* spezifiziert den Port vom Broker
- *Remote_username* wird benötigt falls man einen Usernamen angeben muss
- *Remote_password* wird benötigt falls man ein Password angeben muss
- *Topic* wird angegeben, um die gewünschten Topics zu empfangen oder zu senden. In unserem Fall wollten wir alle Nachrichten der lokalen Implementierung empfangen und alle Nachrichten senden. Aus diesem Grund kann man # schreiben.

5 HHZ Dashboard



Das Digital HHZ Dashboard dient zur Übersicht über alle Funktionen der SmartLab-Umgebung. Es besteht aus verschiedenen Funktionalitäten, die jeweils in einem Menüpunkt auf dem Dashboard dargestellt sind. Die Funktionalitäten und deren Implementierung werden im Folgenden beschrieben.

Abbildung 17 Dashboard Tabs

5.1 Digital HHZ

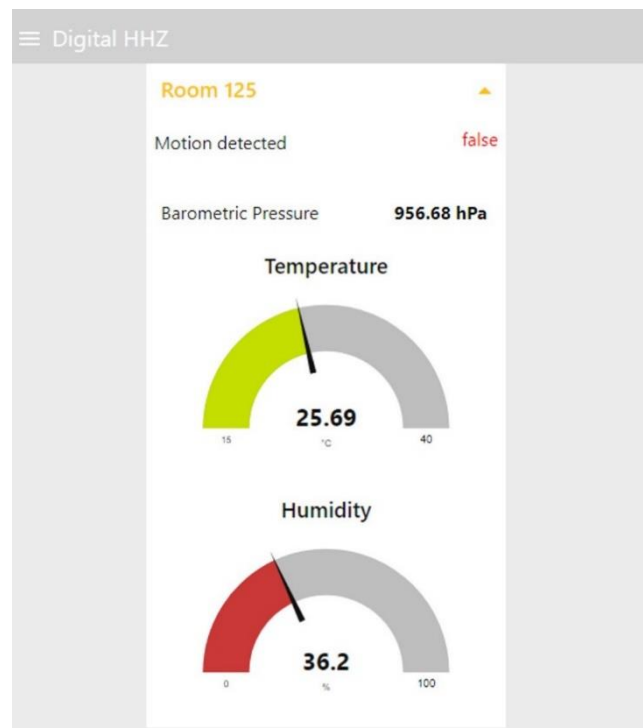


Abbildung 18 externes Dashboard: Digital HHZ Webseite

5.2 Monitoring

Durch den Ausfall des Internets (Glasfaserschaden durch Baggerarbeiten) und von einzelnen Geräten (durch Ausstecken des Netzteils durch Haustechniker) werden der

Projektarbeit wurde sichtbar wie wichtig ein zentrales Monitoring ist. Neben der Evaluation von Ansätzen wie mqttwarn und SNMP haben wir uns für die Implementierung des Monitorings in Form von Node-RED-Dashboard entschieden, da es intuitiv erweiterbar ist und z.B. zukünftig auch ein Alerting über beliebige Webdienste, wie Slack (<https://slack.com>), erlaubt.

5.2.1 Monitoring MQTT-Broker

Für das Monitoring der MQTT-Broker enthält das Dashboard Widgets mit ausgewählten Variablen aus dem Topic \$SYS (vgl. <https://github.com/mqtt/mqtt.github.io/wiki/SYS-Topics>). Wir haben uns dabei an MQTT-Monitoring-Tools (siehe Abbildung 19) orientiert und die für uns interessanten Variablen ausgewählt. Darüber lassen sich Fehler im Netz überwachen, z.B. in dem man die erwartete Zahl an Clients mit der Zahl der aktiven Clients vergleicht.

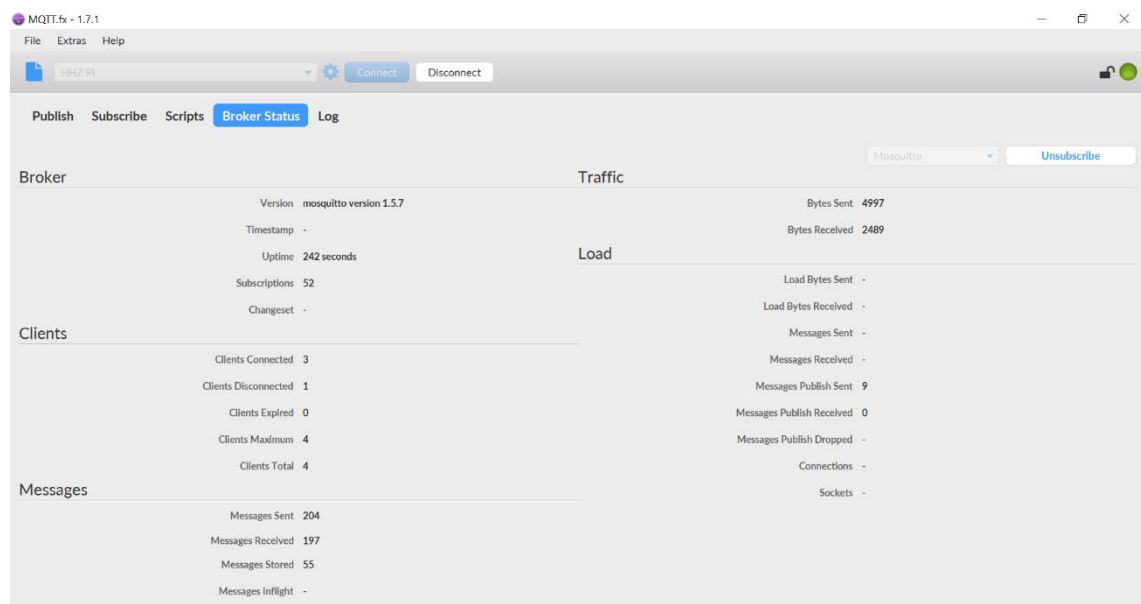


Abbildung 19: MQTT.fx Broker Status

5.2.2 Monitoring der Infrastruktur-Dienste

Im Netzwerkinfrastrukturbereich wird für Monitoring häufig das Protokoll SNMP und verschiedene Netzwerkmanagementtools verwendet. Auf Grund der Einfachheit haben wir uns auch beim Monitoring für MQTT als Kommunikationsprotokoll entschieden, obwohl man das Monitoring eigentlich über einen separaten Kanal und ein anderes Protokoll (diversitäre Redundanz) machen sollte (vgl. <https://www.hivemq.com/blog/why-you-shouldnt-use-sys-topics-for-monitoring>). Als weiteres Protokoll zur Zustandsüberwachung verwenden wir das Internet Control Message Protocol (ICMP) auch bekannt als Ping. Für die Infrastrukturdienste (Database, Broker, Internet) wurden im Block **Server**

LED-Widgets verwendet, die beim Ausbleiben einer Ping-Antwort ihre Farbe von grün auf rot wechseln. Der Gesamtstatus der der 4 Dienste wird oben im Block über die Anzeige Up oder Down angezeigt. Außerdem wird im Block **Latency** die Latenz berechnet und in Form von Tacho-Gauges angezeigt. Im Block **DB-Backup** wird der Zeitstempel des letzten erfolgreich auf Dropbox hochgeladenen Backups angezeigt.

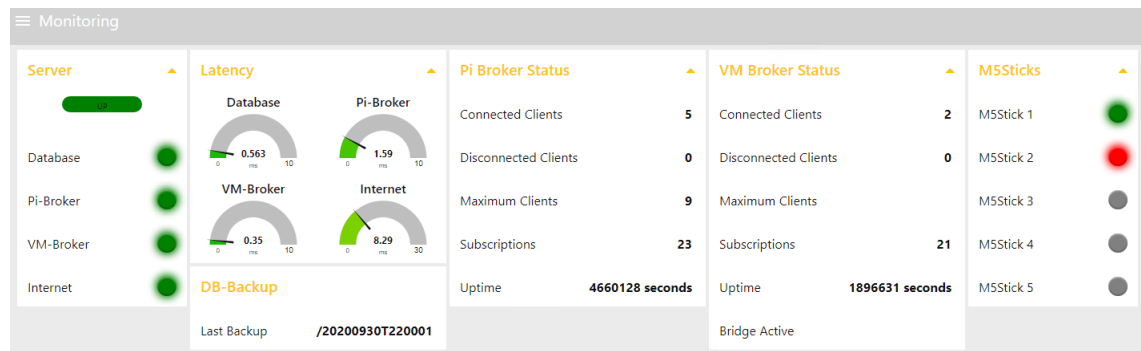


Abbildung 20 Dashboard: Monitoring

Eine weitere Möglichkeit, wie ein Monitoring der Infrastruktur-Dienste erfolgen kann, ist die Verwendung des Service Discovery über Avahi (nähere Details dazu in Kapitel 5.3) In Abbildung 21 wird gezeigt, wie die Dienste wie MQTT Broker, InfluxDB Chronograf und das Node-RED Dashboard erkannt wurden.

Service Discovery		
MQTT-Broker	Node-RED Dashboard	InfluxDB Chronograf
Service Name Mosquitto MQTT Broker	Service Name Digital HHZ Node-RED Dashboard	Service Name Digital HHZ Database
Service Type mqtt	Service Type http	Service Type http
Host broker.local.	Host nodered.local.	Host database.local.
IP Address 10.0.103.70	IP Address 10.0.103.60	IP Address 10.0.103.80
Port 1883	Port 1880	Port 8888

Abbildung 21 Monitoring durch Service Discovery

5.2.3 Monitoring M5Sticks

Im Folgenden wird am Beispiel der M5Stick-Sensoren beschrieben, wie mit Hilfe von MQTT und Node-RED der Status überwacht werden kann.

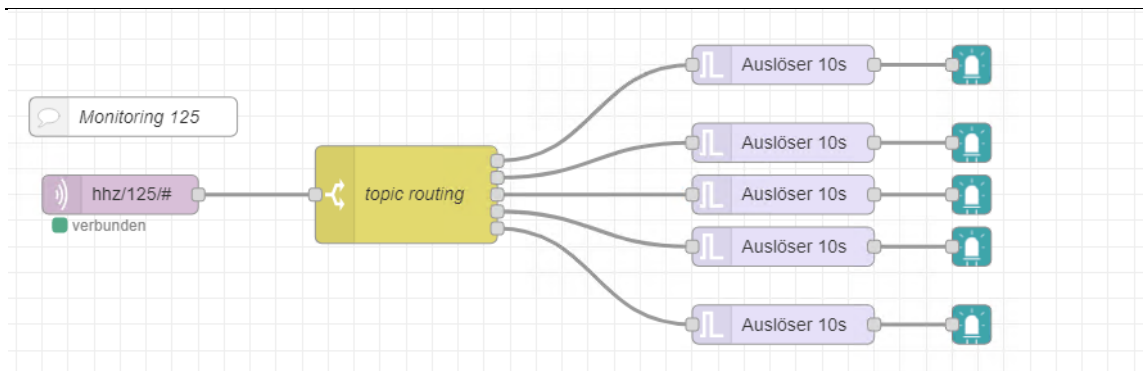


Abbildung 22 Monitoring M5-Sticks

Zusammengefasst, werden zunächst alle Topics abonniert, die mit dem String “hHz/<Raumname>” anfangen. Anschließend werden alle empfangenen Topics nach bestimmten Strings untersucht. Jeder erkannte String repräsentiert einen Sensor aufgrund unserer vorgegebenen Topic Struktur. Für jeden Sensor wird ein Zeitintervall eingestellt. Innerhalb dieses Zeitintervalls wird eine Nachricht von den Sensoren erwartet. Wird eine Nachricht innerhalb dieser Zeit empfangen, so wird “true” gesendet. “false” wird gesendet, wenn keine Nachricht innerhalb der Zeit empfangen wurde. Im letzten Schritt wird der boolean Wert geprüft und die entsprechende Farbe auf dem Dashboard abgebildet (true=grün, false=red).

Im Folgenden werden die Konfigurationen der einzelnen Nodes beschrieben.

Einstellung der Mosquitto-In Node

Diese Node sorgt dafür, dass alle Topics nach dem Schema “hHz/125/” abonniert werden.

- 1) VM oder Pi Broker auswählen
- 2) Credentials angeben

Einstellung des Switch Node

Da jeder M5Stick auf das Topic m5stick<nr1> published, werden hier alle Topics mit dem Schema abgefangen. Pro nr1 (pro M5Stick) wird ein Trigger ausgelöst.

- 1) Beim Feld Eigenschaft “msg.topic” auswählen
- 2) Pro M5Stick eine Regel nach dem Schema in Abbildung 23 hinzufügen.

switch Node bearbeiten

Löschen Abbrechen Fertig

Properties

Name: topic routing

Eigenschaft: msg. topic

☰	enthält	▼ a_z	m5stick1	→ 1	✕
☰	enthält	▼ a_z	m5stick2	→ 2	✕
☰	enthält	▼ a_z	m5stick3	→ 3	✕
☰	enthält	▼ a_z	m5stick4	→ 4	✕
☰	enthält	▼ a_z	m5stick5	→ 5	✕

+ hinzufügen

Alle Regeln überprüfen ▼

☐ Nachrichtenfolgen erneut erstellen

Abbildung 23 Einstellungen Switch-Node

Einstellung der Trigger Nodes

Der Trigger ist für das Intervall zuständig. Werden keine Nachrichten innerhalb der definierten Zeit empfangen, so wird erkannt, dass der M5Stick offline ist.

- 1) Beim Feld Senden den Wert "true" auswählen.
- 2) Beim Feld dann den Wert "warten auf" auswählen und die gewünschte Zeit einstellen.
- 3) Beim Feld dann senden den Wert "false" auswählen.

Einstellung der LED Nodes

Diese Nodes sind für die Darstellung auf dem Dashboard zuständig. Wichtig: sollten neue Geräte dargestellt werden, so muss eine neue Gruppe angelegt werden.

- 1) Gruppe zuweisen (In diesem Fall M5Sticks, weil es bei unseren Geräten um M5Sticks handelt)
- 2) Beim Feld Label pro M5Stick die entsprechende Nummer angeben.

3) Pro msg.payload die entsprechende Farbe zuweisen.

Das beschriebene Vorgehen lässt sich auf alle möglichen Sensoren/Aktoren übertragen, so lange sie über MQTT ein zyklisches regelmäßiges Lebenszeichen senden.

5.3 Service Discovery

Wie bereits in Kapitel 2.8 erwähnt, verwenden wir Avahi für das Service Discovery. In den folgenden Abschnitten wird die Installation und Verwendung von Avahi erläutert.

5.3.1 Avahi-Installation

Auf den aktuellen Versionen der Raspberry Pi Linux Distribution (Raspbian) ist der Avahi-Dienst (Avahi-Daemon) schon installiert und aktiviert. Sollte dies nicht der Fall sein oder wird eine andere Linux Distribution verwendet, kann avahi über apt-get wie folgt installiert werden. Bei uns war dies der Fall auf den Debian-basierten VMs für Database, Node-RED und Broker. Bei der Installation auf den VMs muss man sich mit root einloggen und kann deshalb den sudo-Befehl jeweils weglassen.

```
sudo apt-get update
sudo apt-get install avahi-daemon
```

Ab sofort ist der Host unter seinem Hostnamen z.B. raspberrypi.local zu erreichen. Wenn man den Hostnamen ändern will, dann kann man das mit dem Befehl „hostname“ oder dem Tool „raspi-config“ tun. In den "Advanced Options" von raspi-config gibt es den Menüpunkt "Hostname", mit dem man den Hostnamen des Raspberry Pi ändern kann. Es ist für eine Lernumgebung sinnvoll sprechende und leicht erinnerbare Namen zu verwenden, da diese z.B. auch in Node-RED-Flow verwendet werden können.

5.3.2 Service Discovery mit avahi-browse

Um zu überprüfen, ob Avahi funktioniert, kann man die Avahi-Tools installieren und sich die announcierten Dienste anderer Host im Netzwerk anzeigen lassen.

```
sudo apt-get install avahi-utils
avahi-browse -a
```

Dieses Kommando beendet sich nicht von alleine, das hat den Vorteil, dass neu hinzukommende Services angezeigt werden, sobald sie bekannt gemacht werden. Beenden Sie es mit "Strg + C".

Hier sehen Sie die Ausgabe im Digital HHZ Netz. Jeder Service wird zweimal angezeigt einmal für IPv4 und IPv6. Die Meldung „Invalid service type“ tritt z.B. mit Bosch Smart Home Geräten auf, die das Protokoll scheinbar nicht richtig implementiert haben. Vgl. <https://github.com/lathiat/avahi/issues/212>

```
root@nodered:~# avahi-browse -a
```

```
+ ens160 IPv6 Digital HHZ Node-RED Dashboard      Web Site      local
+ ens160 IPv6 Digital HHZ Database                Web Site      local
+ ens160 IPv4 Digital HHZ Node-RED Dashboard      Web Site      local
+ ens160 IPv4 Digital HHZ Database                Web Site      local
+ ens160 IPv6 Mosquitto MQTT Broker              _mqtt._tcp    local
+ ens160 IPv4 Mosquitto MQTT Broker              _mqtt._tcp    local
avahi_service_browser_new() failed: Invalid service type
```

Will man nur kurz die aktuelle Liste ausgeben möchte, verwendet man stattdessen:

```
avahi-browse -at
```

Will man nur die Domänen anzeigen, geht das mit:

```
avahi-browse -Dt
```

Eine **vollständige Liste** mit aufgelösten IPs erhält man mit:

```
avahi-browse -art
```

Weitere Optionen von **avahi-browse** können wie folgt angezeigt werden:

```
root@nodered:~# avahi-browse --help
avahi-browse [options] <service type>
avahi-browse [options] -a
avahi-browse [options] -D
avahi-browse [options] -b

-h --help          Show this help
-V --version       Show version
-D --browse-domains Browse for browsing domains instead of services
-a --all           Show all services, regardless of the type
-d --domain=DOMAIN The domain to browse in
-v --verbose       Enable verbose mode
-t --terminate     Terminate after dumping a more or less complete list
-c --cache         Terminate after dumping all entries from the cache
-l --ignore-local  Ignore local services
-r --resolve       Resolve services found
-f --no-fail       Don't fail if the daemon is not available
-p --parsable      Output in parsable format
-k --no-db-lookup  Don't lookup service types
-b --dump-db       Dump service type database
```

5.3.3 Service Discovery mit Node-red-node-discovery

Neben dem Service Discovery über avahi-browse wollten wir auch ein hübschere Form des Service Discovery anbieten. Dazu gibt es in der Node-RED-Palette auch schon fertige Nodes für das Service Discovery mit Bonjour/Avahi (<https://flows.node-red.org/node/node-red-node-discovery>). Im folgenden Abschnitt wird beschrieben wie die Nodes installiert und verwendet werden können.

Zunächst muss im Menü der Node-RED-Umgebung der Menüpunkt „Palette verwalten“ ausgewählt werden. Dann öffnet sich der in Abbildung 24 gezeigte Dialog. Um die

richtige Node-Sammlung zu finden, kann im Suchfeld der komplette Name „node-red-node-discovery“ eingegeben werden. Da das Node-RED-Umfeld jedoch sehr agil ist, lohnt es sich eventuell nur nach Discovery zu suchen, um eventuell neuere Implementierungen oder Alternativen zu finden. Durch den Klick auf den Button Installieren werden die Nodes installiert.

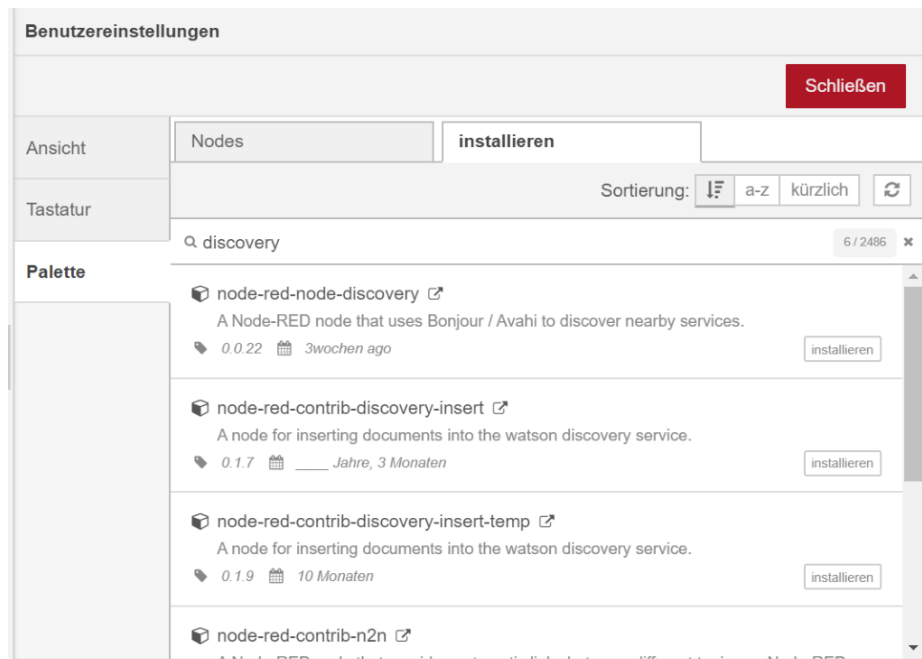


Abbildung 24 Discovery Node: Installation

Wechselt man nach erfolgreicher Installation auf den Tab Nodes, kann man sich anschauen, welche Nodes durch diese Installation neu hinzugekommen sind.

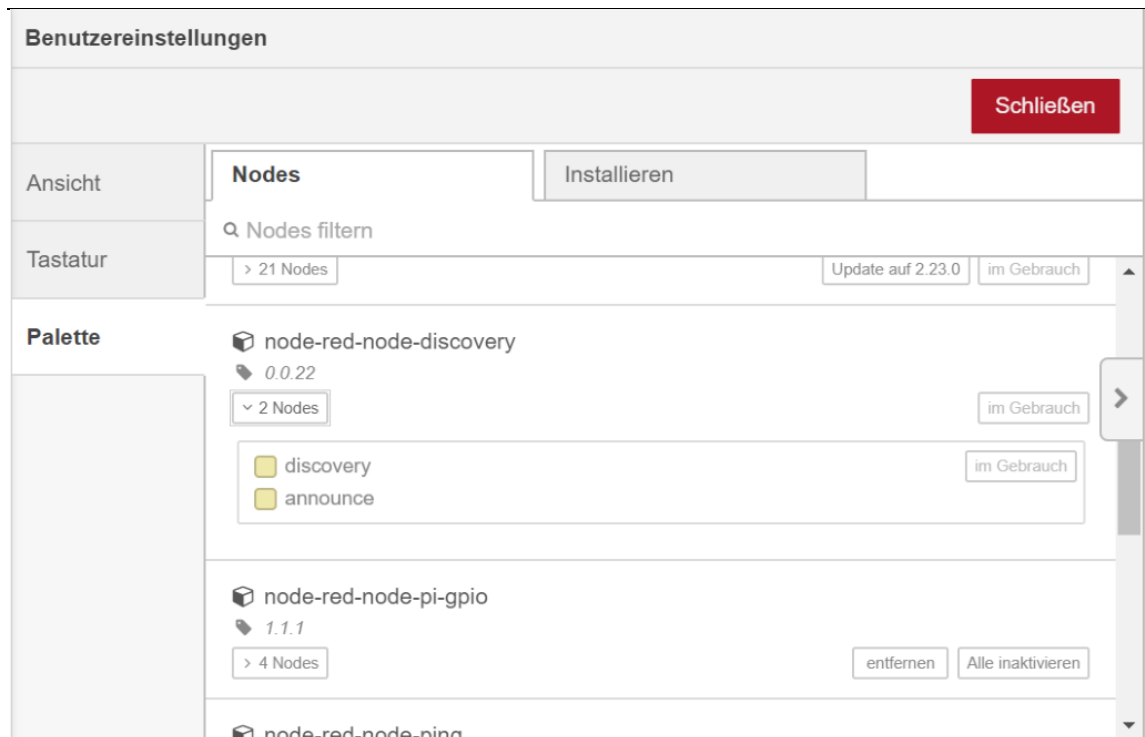


Abbildung 25 Discovery Node: Übersicht

Falls Node-RED einen Fehler bei der Installation meldet, kann dies daran liegen, dass bestimmte Abhängigkeiten zu anderen Linux- oder Node.js-Paketen nicht automatisch aufgelöst werden konnten. Abbildung 26 zeigt wie die benötigten Pakete libavahi-compat-libdnssd-dev und libudev-dev installiert werden. Manchmal kann auch eine manuelle Installation notwendig sein, wenn der User unter dem die Node-RED-Instanz läuft, kein sudo ausführen kann.

```
pi@hzhpi: /etc/avahi
^CGot SIGINT, quitting.
pi@hzhpi: /etc/avahi $ sudo apt-get install libavahi-compat-libdnssd-dev libudev-dev
Paketlisten werden gelesen... Fertig
Abhängigkeitsbaum wird aufgebaut.
Statusinformationen werden eingelesen.... Fertig
libudev-dev ist schon die neueste Version (241-7~deb10u2+rpil).
Die folgenden zusätzlichen Pakete werden installiert:
  libavahi-client-dev libavahi-common-dev libavahi-compat-libdnssd1 libdbus-1-dev
Die folgenden NEUEN Pakete werden installiert:
  libavahi-client-dev libavahi-common-dev libavahi-compat-libdnssd-dev libavahi-compat-libdnssd1 libdbus-1-dev
0 aktualisiert, 5 neu installiert, 0 zu entfernen und 0 nicht aktualisiert.
Es müssen 457 kB an Archiven heruntergeladen werden.
Nach dieser Operation werden 1.326 kB Plattenplatz zusätzlich benutzt.
Möchten Sie fortfahren? [Y/n] j
Holen:1 http://deb.debian.org/debian/raspbian buster/main armhf libavahi-common-dev armhf 0.7-4+b1 [65,1 kB]
Holen:2 http://deb.debian.org/debian/raspbian buster/main armhf libdbus-1-dev armhf 1.12.16-1 [219 kB]
Holen:3 http://deb.debian.org/debian/raspbian buster/main armhf libavahi-client-dev armhf 0.7-4+b1 [64,1 kB]
Holen:4 http://deb.debian.org/debian/raspbian buster/main armhf libavahi-compat-libdnssd1 armhf 0.7-4+b1 [46,4 kB]
Holen:5 http://deb.debian.org/debian/raspbian buster/main armhf libavahi-compat-libdnssd-dev armhf 0.7-4+b1 [61,5
kB]
Es wurden 457 kB in 1 s geholt (526 kB/s).
Vormals nicht ausgewähltes Paket libavahi-common-dev:armhf wird gewählt.
(Lese Datenbank ... 61483 Dateien und Verzeichnisse sind derzeit installiert.)
Vorbereitung zum Entpacken von .../libavahi-common-dev_0.7-4+b1_armhf.deb ...
Entpacken von libavahi-common-dev:armhf (0.7-4+b1) ...
Vormals nicht ausgewähltes Paket libdbus-1-dev:armhf wird gewählt.
Vorbereitung zum Entpacken von .../libdbus-1-dev_1.12.16-1_armhf.deb ...
Entpacken von libdbus-1-dev:armhf (1.12.16-1) ...
Vormals nicht ausgewähltes Paket libavahi-client-dev:armhf wird gewählt.
Vorbereitung zum Entpacken von .../libavahi-client-dev_0.7-4+b1_armhf.deb ...
Entpacken von libavahi-client-dev:armhf (0.7-4+b1) ...
Vormals nicht ausgewähltes Paket libavahi-compat-libdnssd1:armhf wird gewählt.
Vorbereitung zum Entpacken von .../libavahi-compat-libdnssd1_0.7-4+b1_armhf.deb ...
Entpacken von libavahi-compat-libdnssd1:armhf (0.7-4+b1) ...
Vormals nicht ausgewähltes Paket libavahi-compat-libdnssd-dev:armhf wird gewählt.
Vorbereitung zum Entpacken von .../libavahi-compat-libdnssd-dev_0.7-4+b1_armhf.deb ...
Entpacken von libavahi-compat-libdnssd-dev:armhf (0.7-4+b1) ...
libavahi-compat-libdnssd1:armhf (0.7-4+b1) wird eingerichtet ...
libdbus-1-dev:armhf (1.12.16-1) wird eingerichtet ...
libavahi-common-dev:armhf (0.7-4+b1) wird eingerichtet ...
libavahi-client-dev:armhf (0.7-4+b1) wird eingerichtet ...
libavahi-compat-libdnssd-dev:armhf (0.7-4+b1) wird eingerichtet ...
Trigger für libc-bin (2.28-10+rpil) werden verarbeitet ...
pi@hzhpi: /etc/avahi $
```

Abbildung 26 Discovery Node: Abhängigkeiten

Nach der Installation kann es notwendig sein den Node-RED-Service neuzustarten. Auf der Linux-Shell kann dies mit dem Befehl **node-red-restart** ausgeführt werden. Davor sollte sichergestellt werden, dass alle Flows zuvor gespeichert wurden.

Nach der erfolgreichen Installation, sollten die Nodes discovery und announce in der Gruppe der Network-Nodes angezeigt werden und es kann ein Discovery-Node per Drag'n'Drop auf den Flow gezogen werden. Durch einen Doppelklick auf den Node, öffnet sich der Properties-Dialog (Abbildung 27).

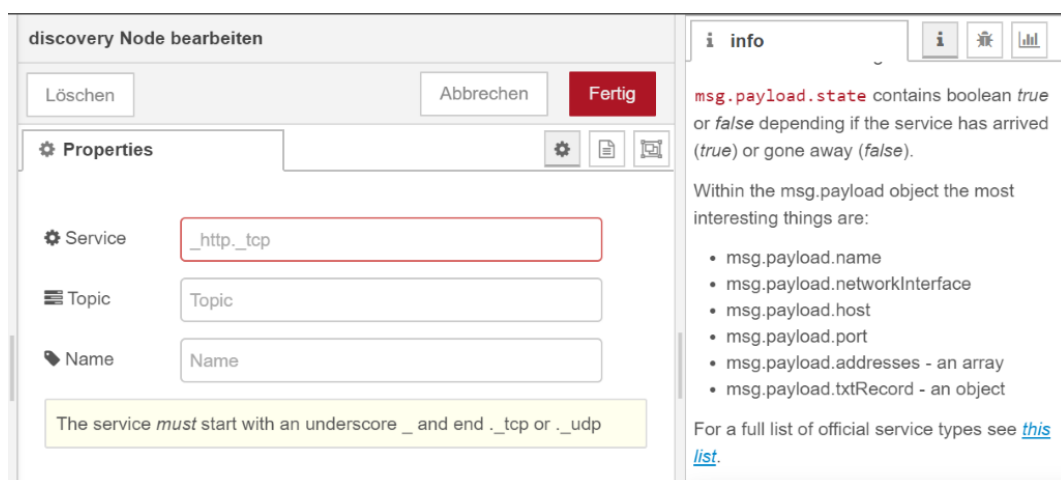


Abbildung 27 Discovery Node: Bearbeiten

Im Feld Service muss nun der Service Name so eingetragen werden, wie er vom Announcing-Service veröffentlicht wird. Dazu kann der Aufruf von `avahi-browse -ak` nützlich sein, da beim Aufruf von `avahi-browse -a` bekannte Services in einen sprechenden Namen aufgelöst werden. Beispielsweise werden Webserver, die von Service Type „`_http._tcp`“ sind, als von `avahi -a` als „Web Site“ angezeigt. Wie bereits erwähnt findet man unter <http://www.dns-sd.org/servicetypes.html> man eine Liste der registrierten Service Types.

```
root@nodered:/etc/avahi/services# avahi-browse -av
Server version: avahi 0.7; Host name: nodered.local
E Ifce Prot Name                                     Type
omain
+ ens160 IPv6 Mosquitto MQTT Broker                  _mqtt._tcp
  local
+ ens160 IPv4 Mosquitto MQTT Broker                  _mqtt._tcp
  local
+ ens160 IPv6 Digital HHZ Node-RED Dashboard         Web Site
  local
+ ens160 IPv6 Digital HHZ Database                   Web Site
  local
+ ens160 IPv4 Digital HHZ Node-RED Dashboard         Web Site
  local
+ ens160 IPv4 Digital HHZ Database                   Web Site
  local
: Cache exhausted
: All for now
^CGot SIGINT, quitting.
root@nodered:/etc/avahi/services# avahi-browse -ak
+ ens160 IPv6 Mosquitto MQTT Broker                  _mqtt._tcp
  local
+ ens160 IPv4 Mosquitto MQTT Broker                  _mqtt._tcp
  local
+ ens160 IPv6 Digital HHZ Node-RED Dashboard         _http._tcp
  local
+ ens160 IPv6 Digital HHZ Database                   _http._tcp
  local
+ ens160 IPv4 Digital HHZ Node-RED Dashboard         _http._tcp
  local
+ ens160 IPv4 Digital HHZ Database                   _http._tcp
  local
```

Abbildung 28 `avahi-browse -ak`

Nachdem der Service Type eingetragen wurde können wie in Abbildung 29 gezeigt, Dashboard-Widget-Nodes zum Flow hinzugefügt werden und mit dem Discovery-Node verknüpft werden. In Abbildung 27 ist in der Hilfe ersichtlich wie sich die `msg.payload` zusammensetzt. So wurde in Abbildung 29 für die Datenfelder Service Name, Service Type, Host, IP Address und Port je ein Textlabel auf dem Dashboard erzeugt, das mit den entsprechenden Werten aus dem Discovery Node verknüpft wurde.

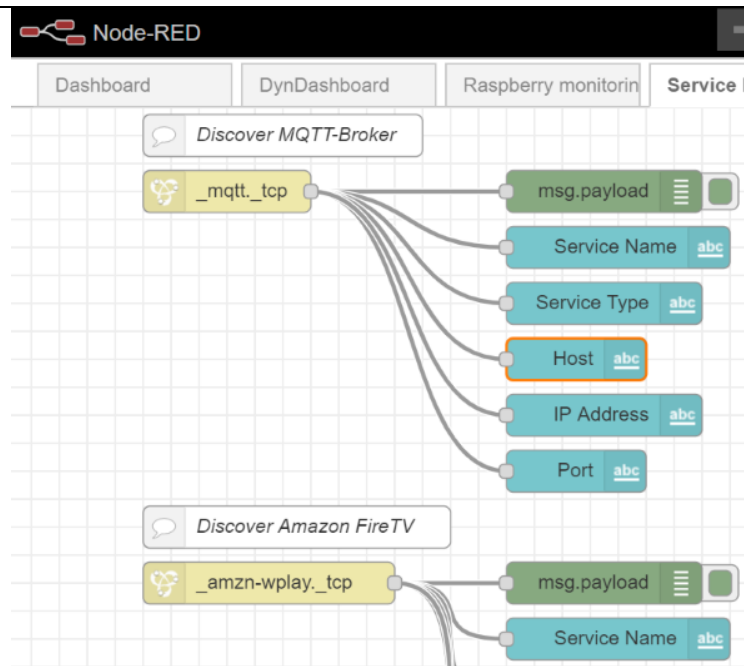


Abbildung 29 Service Discovery Flow

Auch die Verwendung von Switch-Nodes ist möglich, falls es mehrere Dienste vom selben Typ gibt (siehe Abbildung 30).

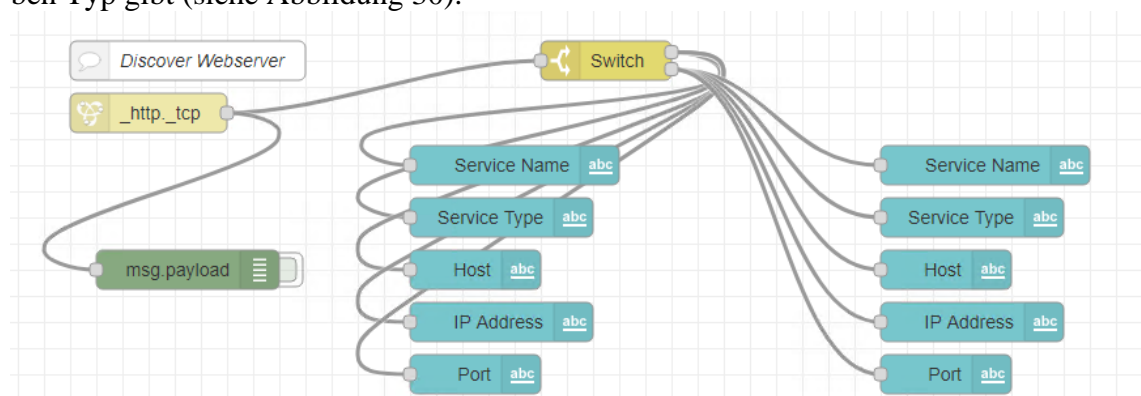


Abbildung 30 Discovery Node: Switch Node

Abbildung 31 zeigt wie ein Service Discovery Dashboard für unterschiedlichste Smart-Home-Geräte aussehen kann. Die Nodes sind im Service Discovery Flow des internen Dashboards alle implementiert. Um es Übersichtlich zu halten, sind die Discovery-Nodes und Dashboard-Widgets für zurzeit nicht im Netz installierte Service Types deaktiviert.

Service Discovery

<div>MQTT-Broker</div> <div>Service NameMosquitto MQTT server on hhzpi</div> <div>Service Typemqtt</div> <div>Hosthhzpi.local.</div> <div>IP Address192.168.178.37</div> <div>Port1883</div>	<div>eWeLink</div> <div>Service NameeWeLink_1000402325</div> <div>Service Typeewelink</div> <div>HosteWeLink_1000402325.local.</div> <div>IP Address192.168.178.25</div> <div>Port8081</div>	<div>Webserver</div> <div>Service NameTasmotaTemp01</div> <div>Service Typehttp</div> <div>HostTasmotaTemp01.local.</div> <div>IP Address192.168.178.40</div> <div>Port80</div>
<div>Amazon FireTV</div>		
<div>Google Chromecast</div>		

Abbildung 31 Discovery Dashboard (Beispieldarstellung)

5.3.4 Service Announcement per avahi service description

Um einen Service zu announce wurden im Digital HHZ zwei Möglichkeiten implementiert. Zum einen die Konfiguration von Avahi über avahi service description files und die Verwendung des Announce-Nodes in Node-RED. In diesem Kapitel wird beschrieben wie ein Avahi service description file erstellt wird.

Avahi veröffentlicht Services deren *.service-Datei sich in /etc/avahi/services befinden. Die Dateien in diesem Ordner müssen durch das user group avahi lesbar sein.

Wenn es noch keine *.service-Datei gibt lässt sich eine solche Datei mit einem beliebigen Editor z.B. nano nach folgendem Muster erstellen.

```
<!-- Put this file named dashboard.service in /etc/avahi/services/ -->
<!DOCTYPE service-group SYSTEM "avahi-service.dtd">
<service-group>
  <name replace-wildcards="yes">Digital HHZ Dashboard on http://%h:1880/ui</name>
  <service>
    <type>_http._tcp</type>
    <port>1880</port>
    <txt-record>info=Node-RED based Dashboard displaying all available sensor values.</txt-record>
  </service>
</service-group>
```

Dazu muss zunächst der Service Type <type> festgelegt werden. Auch hier lohnt sich ein Blick in die Liste der registrierten Service Types <http://www.dns-sd.org/service-types.html> Im Beispiel für das Dashboard verwenden wir „_http._tcp“, da es sich dabei um eine Webseite handelt. Da das DNS-SD-Register 2010 im IANA-Register aufgegangen ist, sollte man auch das IANA-Register (<http://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml>) prüfen oder in der /etc/services Datei nachschauen. Anschließend trägt man die entsprechende TCP/UDP-Port-Nummer unter dem Tag <port> ein. Unter txt-record kann mit dem txt-Key „info=“ eine

Beschreibung des Service erfolgen. Manche Service Types erlauben weitere txt-keys wie z.B. topics=.

Nach dem die Datei erfolgreich erstellt und gespeichert wurde, kann avahi neugestartet werden (sudo service avahi-daemon restart) und sollte dann die neuen Services verkünden.

Als weiteres Beispiel ist hier die Konfiguration für den Mosquitto-Broker abgebildet. Die Wildcard %h im <name>-Tag wird ersetzt durch den Hostnamen des Rechners, wenn das Attribut replace-wildcards auf yes gesetzt ist.

```
<!-- Put this file named mosquitto.service in /etc/avahi/services/ -->
<!DOCTYPE service-group SYSTEM "avahi-service.dtd">
<service-group>
  <name replace-wildcards="yes">Mosquitto MQTT Broker on %h</name>
  <service>
    <type>_mqtt._tcp</type>
    <port>1883</port>
    <txt-record>info=Subscribe to topic hhz/# to discover which topics are published. Ask lab supervisor/professor for user name and password.</txt-record>
  </service>
</service-group>
```

5.3.5 Service Announcement per Node-red-node-discovery: Announce

In Node-RED kann, nachdem der Announce-Node zu einem Appliance-Flow hinzugefügt wurde, analog zum zuvor beschriebenen Vorgehen der Node konfiguriert werden (Abbildung 32) und die Datenfelder entsprechend gefüllt werden.

announce Node bearbeiten

Löschen Abbrechen Fertig

Properties

Service:

Port:

TxtRecord:

Name:

%h in the name will be replaced by the machine hostname.

info

Description

Node-Hilfe

Provides a Bonjour / Avahi / Zeroconf announcement node.

If `msg.payload` is `false` - the announcement is stopped. Any other value starts the announcement process.

The announcement can be customised by the msg if not configured in the edit panel.

- `msg.service` - For a full list of official service types see [this list](#).
- `msg.port` - the tcp or udp port to use.
- `msg.name` - the short description name of the service. If you use %h in the name, it will be replaced by the machine hostname.
- `msg.txtRecord` - a javascript object of name:value pairs.

Abbildung 32 Announce Node: Bearbeiten

Im nächsten Schritt sollte der Announce-Node noch mit einem Trigger-Node verknüpft werden, damit das Announcement in zyklischen Abständen erfolgt. Um das Netz nicht mit zu vielen Announcement-Nachrichten zu fluten, wurde für das Dashboard ein Intervall von 5 Minuten gewählt.

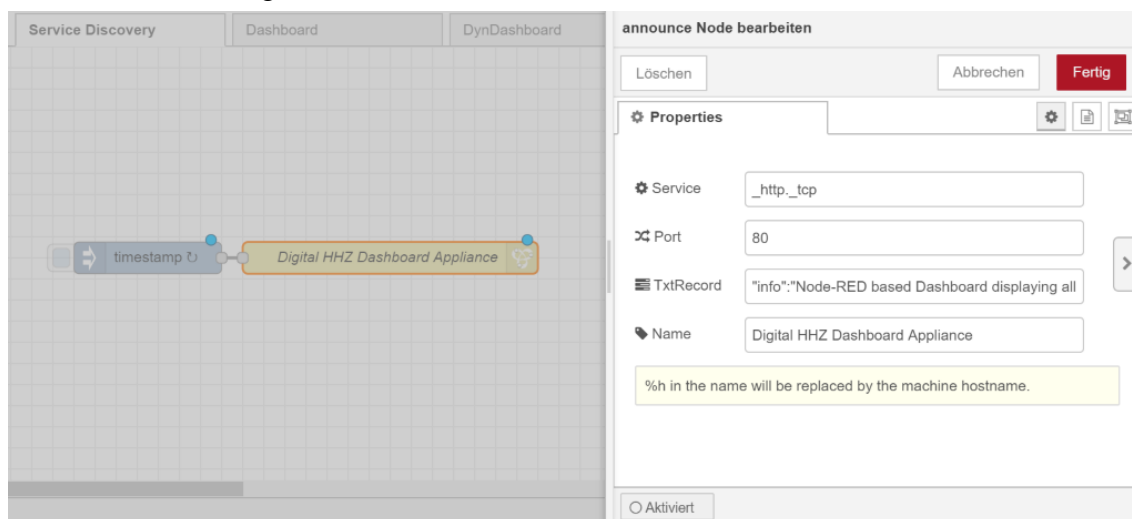


Abbildung 33 Announce Node: Flow

5.3.6 Service Announcement von Tasmota

Trotz anders lautender Foren-Einträge lässt sich auch bei der Tasmota-Firmware mDNS aktivieren. Um mDNS z.B. auf der Tasmota-Lampe zu aktivieren, muss man die SetOption55 auf 1 setzen. Das es unterschiedliche Möglichkeiten (Webinterface, MQTT, ...) gibt, die SetOptions zu setzen, sei an dieser Stelle auf die Dokumentation verwiesen: <https://tasmota.github.io/docs/Commands/>

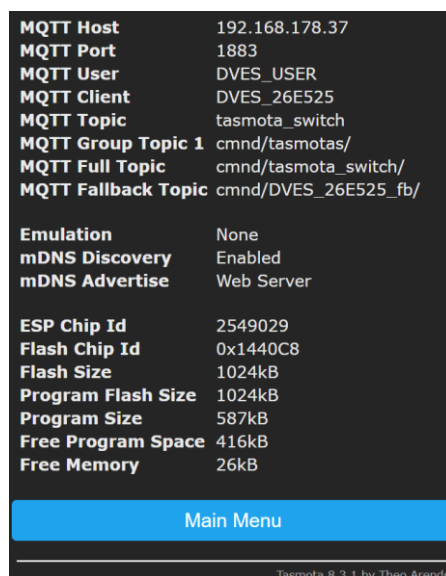


Abbildung 34 Tasmota mDNS Service Announcement

5.3.7 Service Type Discovery

Weitere Service Types können per Reverse Engineering gefunden werden und auf dem Service Discovery Dashboard implementiert werden. Durch schrittweises Einschalten aller verfügbaren Smart-Home-Geräte während auf einem Raspberry Pi ein avahi-brows-a lief, wurden folgenden Service Types entdeckt und als Nodes und Widget im Node-RED-Flow für das Service Discovery implementiert:

- Apple HomeKit (HAP),
- Google Chromecast,
- Amazon FireTV,
- Tasmota,
- Sonoff (eWeLink),
- Ikea Tradfri,
- Philips Hue,
- Raspberry Pi (z.B. ssh),

Amazon Echo (Alexa) hat auch einen Service Type: `_alexa._tcp` (<https://developer.amazon.com/de-DE/docs/alexa/networking/overview.html#dns-sd>)

Aber bei den zwei vorhandenen Amazon Echo-Geräten konnte dieser Service Type weder über avahi-browse noch über Wireshark beobachtet werden. In der Alexa Developer Dokumentation konnte der Grund dafür gefunden werden:

Important: Currently, networking skills are supported only in the United States.

Die vorhandenen Amazon Echos announced jedoch andere Services `ssh._tcp`, `_sftp-ssh._tcp`, `_workstation._tcp`, was aber mit installierten Skills zu tun haben kann. Leider konnte dazu nichts Weiteres in den Dokumentationen der Skills gefunden werden.

5.3.8 Troubleshooting

Wenn ein Host nicht mehr per Computernamen erreichbar ist, kann es helfen, den Avahi-Daemon zu deinstallieren und anschließend wieder zu installieren.

```
sudo apt-get remove avahi-daemon  
sudo apt-get install avahi-daemon
```

Es gibt Smart-Home-Geräte, die mDNS fehlerhaft implementiert haben und zum Absturz von Avahi führen können. Auch fehlerhafte Avahi-Service-Konfigurationsdateien können dazu führen, dass Avahi nicht mehr funktioniert. In diesem Fall lohnt es sich die Konfigurationsdateien in `/etc/avahi/services` mal näher anzuschauen. Beispielsweise führte die Verwendung von Slashes (/) in der Service Description zu Fehlern.

5.4 Simulation

Der Tab *Simulation* kann zur simulierten Wiedergabe von Sensorwerten verwendet. Hierfür können verschiedenste Sensorwerte über eine grafische Oberfläche an den Broker geschickt werden. Hintergrund dieser Funktion ist das Ermöglichen von Tests. Sollten während der Entwicklung einer neuen Appliance Daten benötigt werden, können diese anstatt bspw. über einer Konsoleneingabe oder zusätzlichen Tools, über das interne Dashboard getriggert werden. Der Vorteil ist, dass keine manuelle Authentifizierung am Broker sowie manuelle Topic-Eingabe notwendig ist.

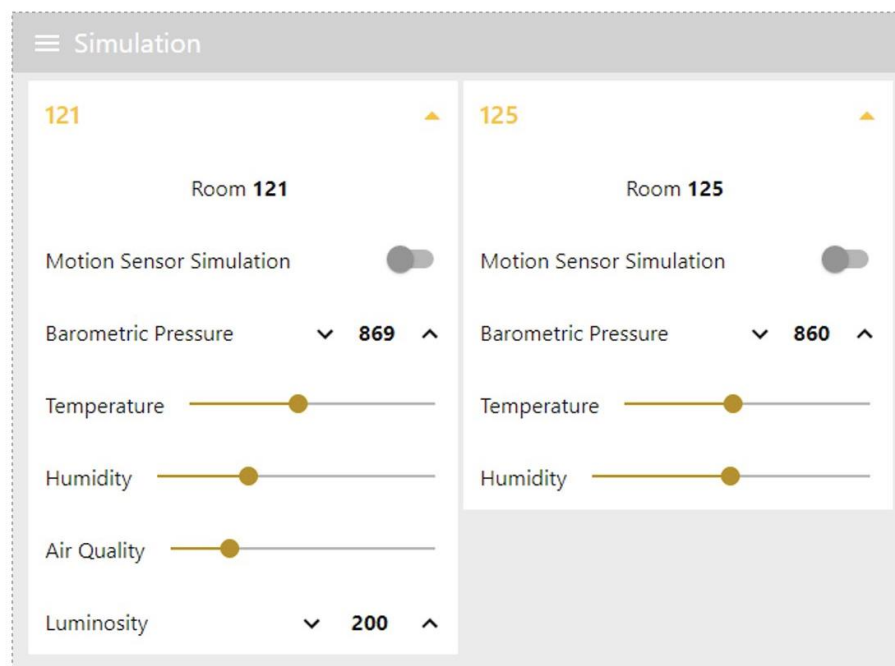


Abbildung 35 Dashboard Simulation

Sollten archivierte Daten wiedergegeben werden, kann unser Pythonskript, das auf dem Github liegt, verwendet werden. Dieses Skript liest die CSV-Datei und schickt sie über ein gewünschtes Topic mit gewünschter Geschwindigkeit an den Mosquitto-Broker.

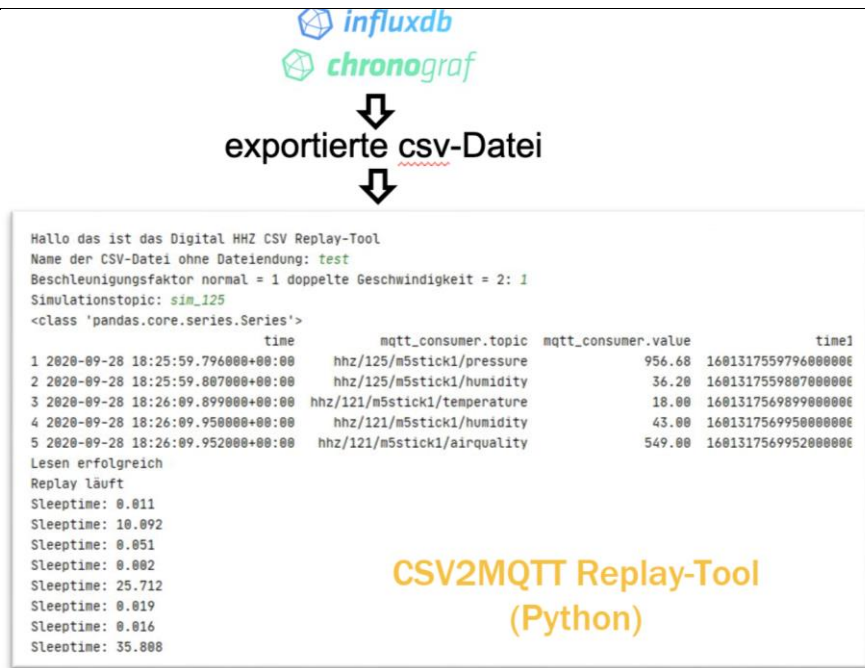


Abbildung 36 Daten Replay

Hinweis:

Für die Verwendung dieses Tools, muss die CSV im gleichen Pfad wie das Skript liegen.

5.5 InfluxDB Chronograf

Klickt man auf diesen Reiter wird man auf Chronograf weitergeleitet. Da sich diese Funktion mit dem Datenmanagement befasst, wird für eine genauere Beschreibung und eine Anleitung zum Aufsetzen des Systems auf das Kapitel 7 verwiesen.

5.6 ThingsBoard

Klickt man auf diesen Reiter wird man auf ThingsBoard weitergeleitet. Da sich diese Funktion mit dem Device Management befasst wird für eine genauere Beschreibung und eine Anleitung zum Aufsetzen des Systems auf das Kapitel 6 verwiesen.

5.7 ThingsBoard Devices

Auf diesen Reiter werden Geräteinformationen angezeigt. Man wird ebenfalls auf diese Seite weitergeleitet, wenn man den QR-Code auf den Geräten scannt. Diese Seite kann auch verwendet werden um QR-Codes für die Devices zu generieren, die dann ausgedruckt werden können und in räumlicher Nähe zum Device platziert werden. Im Access-Token-Feld wird der Access-Token automatisch eingetragen, wenn über die URL <http://nodered.digitalhhz.smartlab.local:1880/device/{accesstoken}> aufgerufen wird. Bei {accesstoken} handelt es sich um einen Platzhalter, der durch den in ThingsBoard konfigurierten Device Access Token ersetzt werden muss. Wir verwenden als Device Access Token die MAC-Adresse. Die Geräteinformationen werden durch einen Node-RED-Flow über die ThingsBoard-API abgerufen. Die Idee dahinter ist, die Möglichkeiten, die ein Gerät bietet einfach abrufen zu können, um damit dann eigene Appliances erstellen zu können. Dazu wird dann das Basic Topic mit der Wildcard # mit einem MQTT-Client wie mqtt.fx abonniert, um zu sehen, welche Topics von diesem Gerät veröffentlicht werden. Die weiteren Felder werden in Kapitel 6 beschrieben.

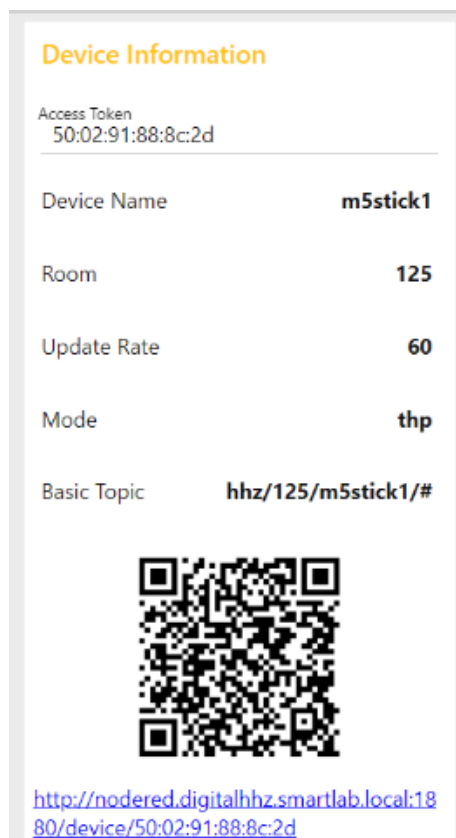


Abbildung 37 Dashboard ThingsBoard Devices

5.8 ThingsBoard Appliances

Auf diesen Reiter werden die Relationen der Geräte angezeigt. Beispielsweise kann man hier auch entnehmen, welche Geräte eine Appliance nutzt. Da diese Funktion ebenfalls ThingsBoard nutzt, wird auf das Kapitel 6 verwiesen.

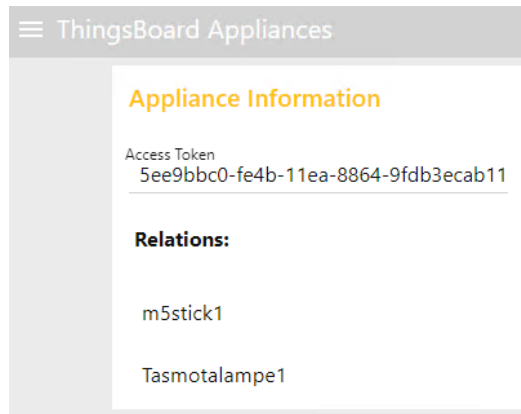


Abbildung 38 Dashboard ThingsBoard Appliance

6 Device Management

ThingsBoard wird im Digital HHZ zum Device Management verwendet. Dabei werden folgende Funktionalitäten genutzt:

- Geräte (engl. „Devices“) und Objekte (engl. „Assets“)
- API zum Abrufen von Geräte-Eigenschaften z.B. zum Konfigurieren der M5Sticks
- API-Aufruf über QR-Code zur Geräteidentifikation
- Dashboard zum Darstellen der Beziehungen zwischen Geräten und Objekten

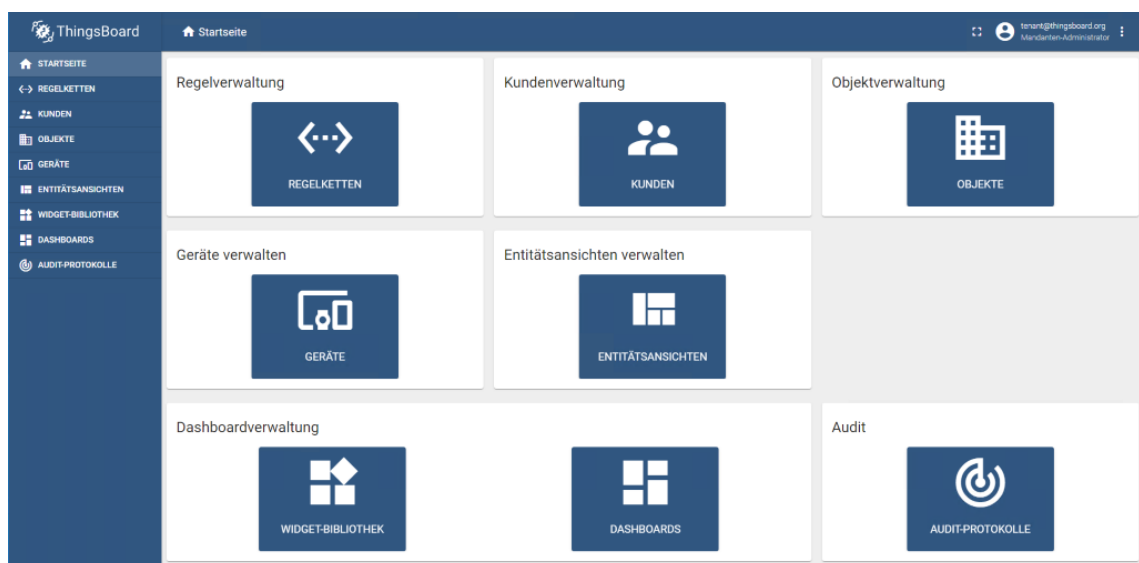


Abbildung 39 ThingsBoard Dashboard

6.1 Unterscheidung Geräte und Objekte

Innerhalb des Digital HHZ werden alle Sensoren und Aktoren zu den Geräten gezählt (z.B. M5Sticks, Tasmota-Lampen usw.). Alle Gebäude, Räume und Appliances werden als logische Objekte angelegt. Zusätzlich gibt es die beiden Objekte „Rooms“ und „Appliances“ diese dienen lediglich der logischen Strukturierung der Darstellung im Dashboard.

6.2 Anlegen eines Gerätes

Nachfolgend wird am Beispiel eines m5Sticks die Erstellung eines Gerätes erklärt.

1. Öffnen der Geräteansicht
2. Über das Plus ein neues Gerät anlegen (z.B. Name:M5stick1, Typ:M5stick)

3. Anschließend das Gerät öffnen (siehe Abbildung 40):

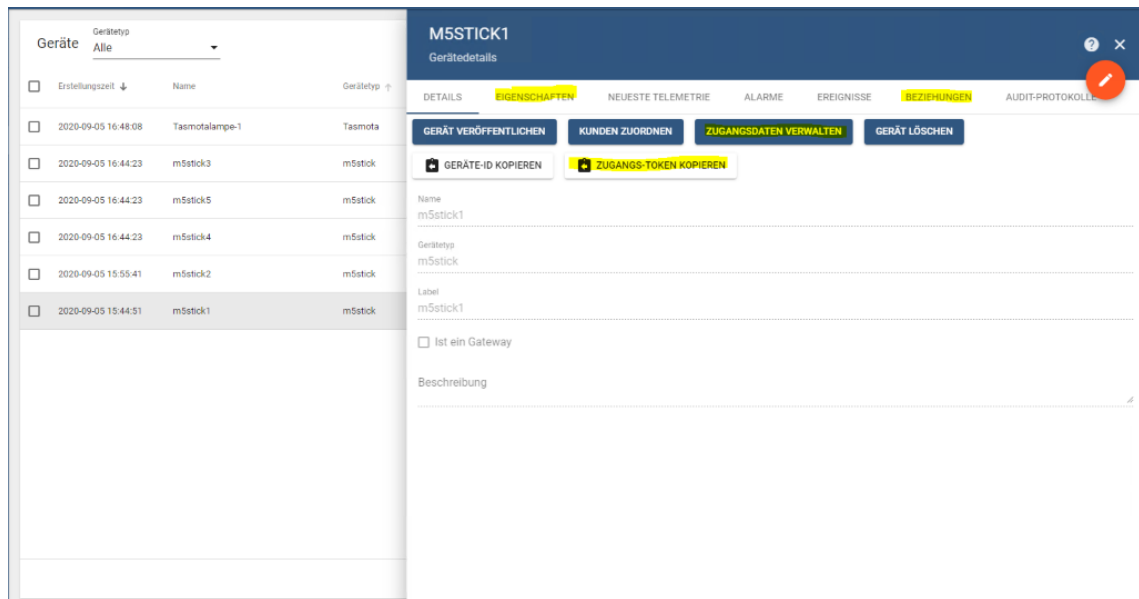


Abbildung 40 ThingsBoard Gerät öffnen

- Über „Zugangsdaten bearbeiten“ die MAC-Adresse des neuen Geräts als Zugangs-Token eintragen
- Bei Eigenschaften können nun unter „gemeinsame Eigenschaften“ (siehe Abbildung 41) beliebige Eigenschaften über das kleine Plus hinzugefügt werden. Der Eintrag als „gemeinsame Eigenschaften“ ist wichtig, damit über eine API später die Eigenschaften abgerufen werden können.

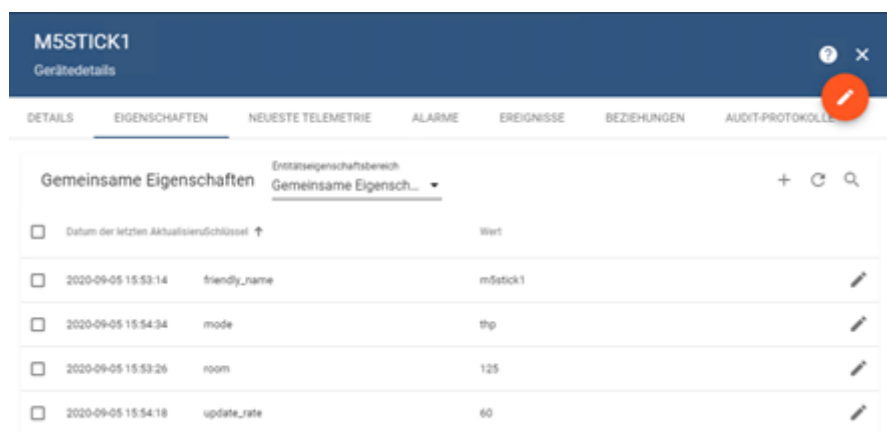


Abbildung 41 Eigenschaften M5Stick

6. Insbesondere bei M5Sticks müssen zwingend folgende Eigenschaften angelegt werden, da dieser beim Restart die Konfiguration/Eigenschaften über die API abrufen (Tabelle 2):

Tabelle 2 Eigenschaften für M5Stick

Schlüssel	Wert
friendly_name	ID für das MQTT-Topic „m5stick1“
mode	„motion“ oder „thp“ (Arbeitsmodus als Motionssensor oder ENV-Sensor)
room	integer z.B. 125
update_rate	integer (Sekunden) z.B. 120

7. Alle Geräte sind logisch gesehen auf der untersten Ebene, deshalb werden unter Beziehungen lediglich „Eingehende Verbindungen“ konfiguriert. Zum einen erfolgt eine Zuordnung zu einem Raum und zum anderen kann so die logische Beziehung zu Appliances hergestellt werden (siehe Abbildung 42)

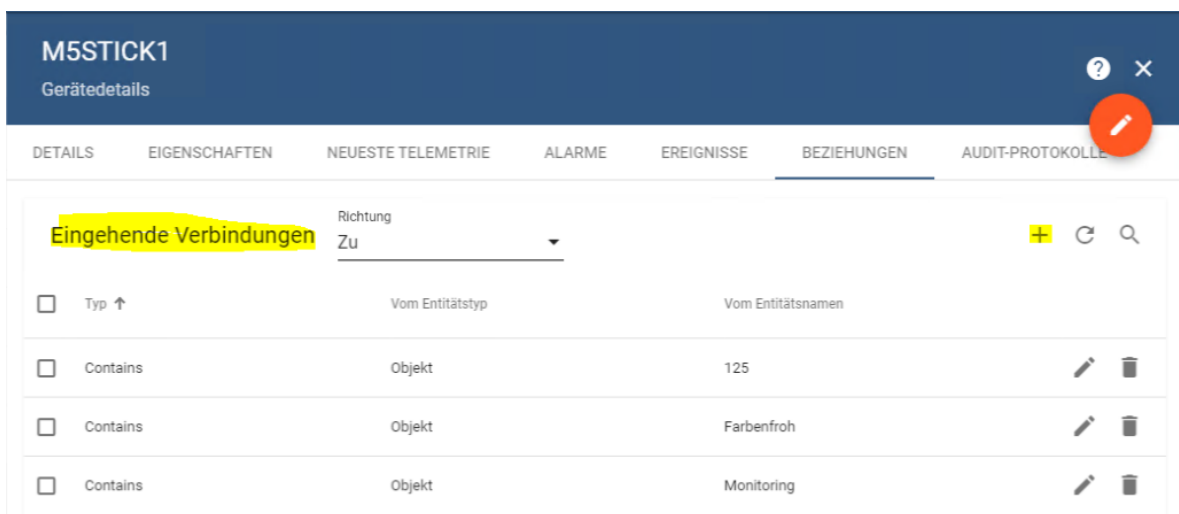


Abbildung 42 M5Stick eingehende Verbindungen konfigurieren

Hinweis: Bei Ausfall eines M5Sticks kann nun einfach ein neuer M5Stick mit der aktuellen M5Stick Software bespielt werden. Anschließend muss in ThingsBoard nur die neue MAC-Adresse als Zugangs-Token eingetragen werden und das System arbeitet normal weiter, ohne dass Anpassungen an den anderen Appliances (DB, Monitoring, etc.) notwendig sind.

6.3 Zuordnung physisches Gerät zu Datenobjekt

An jedem Gerät innerhalb des HHZ ist ein QR-Code angebracht. Dieser enthält eine URL, welche auf eine ThingsBoard-API verweist. In der URL ist die jeweilige MAC-Adresse enthalten. Beim Scannen mit dem Smartphone bekommt man z.B. folgendes zurück.

Bsp. URL:

```
http://nodered.digitalhhz.smartlab.local:8080/api/v1/50:02:91:88:8c:2d/attributes
```

Bsp. Antwort:

```
{
  "client":{
    "secretKey":"value",
    "durationMs":60000
  },
  "shared":{
    "friendly_name":"m5stick1",
    "room":125,
    "update_rate":60,
    "mode":"thp"
  }
}
```

Zu beachten ist, dass das Smartphone mit dem Netzwerk des Digital-HHZ verbunden ist.

Neben dem direktem API-Aufruf lässt sich ein physisches Gerät auch mit Hilfe von Node-Red zuordnen. Die entsprechende URL dazu lautet wie folgt:

- Bei einem Device <http://nodered.digitalhhz.smartlab.local:1880/device/<MAC-Adresse>>
- Bei einer Appliance <http://nodered.digitalhhz.smartlab.local:1880/appliance/<Asset-ID>>

6.4 Dashboard

Über ThingsBoard → Dashboard → Digital HHZ lässt sich eine Übersicht (siehe Abbildung 43) aller Geräte und Devices im HHZ aufrufen (Voraussetzung: die Beziehungen innerhalb der Geräte und Objekte sind richtig gesetzt).

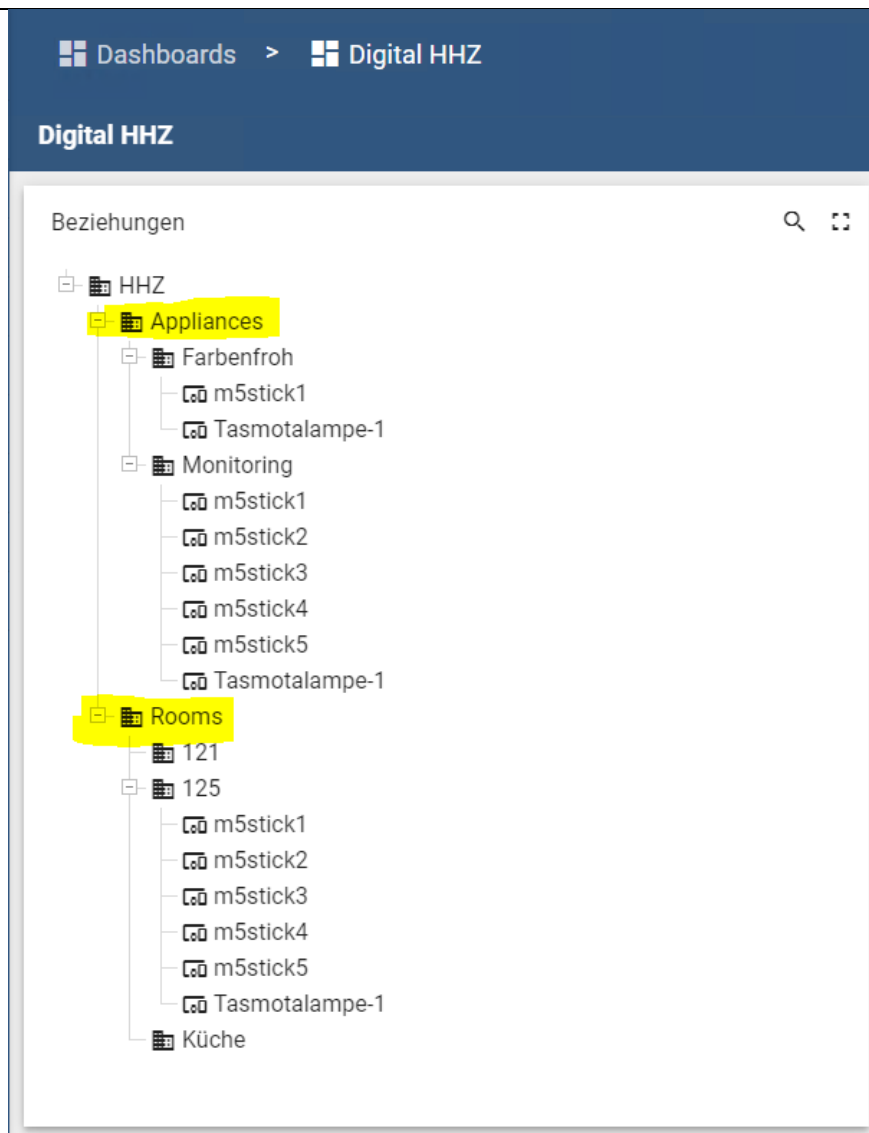


Abbildung 43 Beziehungen im HHZ

Innerhalb der Baumstruktur kann nur einmal logisch und physisch gesucht werden. Unter HHZ → Appliances sind alle eingebundenen Appliances zu finden. Unter der jeweiligen Appliance kann nun eingesehen werden, welche Geräte von ihr genutzt werden. Gleiches gilt für die Zuordnung von Räumen. Somit lassen sich z.B. vor Wartungen an bestimmten Geräten die Folgen für bestimmte Appliances abschätzen/einsehen

6.5 ThingsBoard Dokumentation

Siehe <https://thingsboard.io/docs/guides#AnchorIDGettingStartedGuides> für eine detaillierte Dokumentation zur Nutzung von ThingsBoard.

7 Datenmanagement

Dieses Kapitel beinhaltet das gesamte Wissen über die Datenhaltung mit influxdb, das Datenmanagement mit chronograf, telegraf und der Datensicherung mit influx backup & dropbox uploader.

7.1 ER-Modell

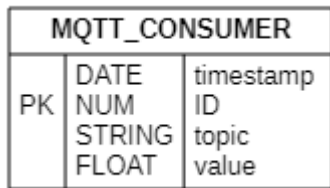


Abbildung 44 ER Modell

7.2 Wie sind die Daten zu verstehen?

Tabelle 3 Datenwerte

Feld	Datentyp	Beschreibung	Beispiel
timestamp	DATE	Influx-db ist eine timeseries-database. Alle inputs werden mit einem Zeitstempel versehen. Format: rfc3339	2019-09-18T12:00:00.000000000Z
(PK) ID	NUM	Eine fortlaufende Nummer dient als Primary Key. Der Zeitstempel könnte theoretisch nicht eindeutig sein, wenn zur selben Zeit Nachrichten gepublisht werden.	4711
topic	STRING	Aktuell gibt es nur den M5Stick-C, allerdings können weitere Sensoren wie der Rehau-Sensor für weitere Appliances implementiert werden. Die M5Sticks bekommen zur eindeutigen Identifikation eine Nummer am Ende.	hhz/125/m5stick3/humidity hhz/125/m5stick3/temperature hhz/125/m5stick3/co2 hhz/125/m5stick4/pressure hhz/125/m5stick1/motion
value	FLOAT	Messwert	856

7.3 Wie sind die Daten zu verarbeiten?

In der Artificial Intelligence Vorlesung können Studierende die Datensätze miteinander vergleichen und versuchen Korrelationen zu finden (z. B. zwischen Lufttemperatur und Luftfeuchtigkeit). Ein Gesamt-Export der Daten ist auf der Database VM durch Eingabe des folgenden commands (am Beispiel TEMPERATURE) möglich:

```
influx -database 'digitalhhz2' -execute 'SELECT * FROM TEMPERATURE' -format csv > TEMPERATURE.csv
```

```
column -s, -t < data3.csv | less -#2 -N -S
```

Alternativ ist ein Zugriff/Export über Chronograf möglich (siehe Kapitel 7.6)

7.4 InfluxDB

7.4.1 Installation

InfluxDB Version 1.8.1.

1. Installation und Start über [influx](#)
2. Konfiguration der influxdb.conf Datei

```
[http]
Enabled = true
Bind-address = „8086“
https-enabled = false
https-certificate = "/etc/ssl/influxdb.pem"
```

Notiz:

- influxDB V 2.0. beta gibt es nicht für 32 bit systeme. Auf den meisten RasPis laufen 32 bit, deshalb wird hier V.18. installiert
- eine grafische Oberfläche, die über localhost:8086 erreicht werden kann gibt es laut der Dokumentation nicht. Man kann aber für Reporting-Funktionen Grafana oder Chronograph installieren

7.4.2 Login

1. **Login VPN:** VPN-Verbindung □ vpn.reutlingen-university.de
2. **Remote Desktop Connection:** 134.103.214.30:3390 , <user>@smartlab.local
3. **Putty:** 10.0.103.80 oder database.digitalhhz.smartlab.local
4. **Login as:** siehe Password Datei
5. **Start InfluxDB:** Befehl “influx” eingeben, dann startet der Service
6. **DB connecten:** „USE digitalhhz2“
7. **SQL-Befehle:**
 - a. **Show measurements**
 - b. **Select....**

7.5 Telegraf

7.5.1 Installation

1. Installationhhz/
2. Testen

```
sudo systemctl start telegraf
sudo systemctl enable telegraf
sudo systemctl status telegraf
```

3. Anpassung telegraf.conf file
 - a. Pfad:

```
etc/telegraf/telegraf.conf
```

- b. MQTT

```
[[inputs.mqtt_consumer]]
servers = ["tcp://192.168.2.127:1883"]
topics = [
    "telegraf/host01/cpu",
    "telegraf/+/mem",
    "sensors/hhz/125/m5stick2",
    "hhz/125/m5stick2",
]
data_format = "value"
data_type = "integer"
```

- c. InfluxDB Connection

```
[[outputs.influxdb]]
urls = ["http://localhost:8086"]
database = "digitalhhz2"
```

4. Schritt 2 erneut ausführen → Falls Status failed, stimmt was in der config Datei nicht
5. Testen ob Nachrichten ankommen
 - a. Über Kommandozeile

```
mosquitto_sub -d -t hhz/125/m5stick2
mosquitto_pub -d -t hhz/125/m5stick2 -m "1"
```

- b. Per M5Stick

6. Ausgabe der Daten in der Kommandozeile (Auf dem anderen Raspi)
 - a. influx
 - b. USE digitalhhz2
 - c. SELECT * from mqtt_consumer

Befehle, um telegraf connection zu überprüfen:

<https://github.com/influxdata/telegraf>

🔗 Generate a telegraf config file:

test

9/11

^ v x

```
telegraf config > telegraf.conf
```

🔗 Generate config with only cpu input & influxdb output plugins defined:

```
telegraf --section-filter agent:inputs:outputs --input-filter cpu --output-filter influxdb config
```

🔗 Run a single telegraf collection, outputting metrics to stdout:

```
telegraf --config telegraf.conf --test
```

🔗 Run telegraf with all plugins defined in config file:

```
telegraf --config telegraf.conf
```

🔗 Run telegraf, enabling the cpu & memory input, and influxdb output plugins:

```
telegraf --config telegraf.conf --input-filter cpu:mem --output-filter influxdb
```

Abbildung 45 Telegraf Konfiguration

Notiz:

- Die MQTT server adresse ist nicht localhost, da der Broker auf einem anderen Raspi läuft als die DB
- Es wird der Mosquitto-Broker verwendet mit dem Standardport 1883

```
servers = ["tcp://10.0.103.70:1883"]
#
# ## Topics that will be subscribed to.
topics = [
  "telegraf/host01/cpu",
  "telegraf+/mem",
  "sensors/hhz/125/*",
  "hhz/125/*",
  "hhz/125/m5stick1/motion",
  "hhz/125/m5stick2/motion",
  "hhz/125/m5stick3/motion",
  "hhz/125/m5stick4/motion",
  "hhz/125/m5stick5/motion",
  "hhz/125/m5stick6/temperature",
  "hhz/125/m5stick7/humidity",
]
```

7.6 Chronograf

7.6.1 Login

1. **Login VPN:** VPN-Verbindung → vpn.reutlingen-university.de
2. **Remote Desktop Connection:** 134.103.214.30:3390 , <user>@smartlab.local
3. **Putty:** 10.0.103.80 oder database.digitalhhz.smartlab.local
4. **Login as:** root oder digitalhhz, Password: (siehe Passwort-Datei)
5. **Start Chronograf:** sudo systemctl start chronograf (falls Service nicht bereits läuft→normalerweise sollte der Service beim Hochfahren der VM starten)
6. **Browser:** <http://database.digitalhhz.smartlab.local:8888> (ACHTUNG: nicht localhost:8888)

7.6.2 Explore: Auswahl von Sensordaten und Export

Unter dem Reiter Explore kann auf die Datenbanktabelle zugegriffen und Daten angezeigt, exportiert und zum Dashboard hinzugefügt werden (siehe Abbildung 46)

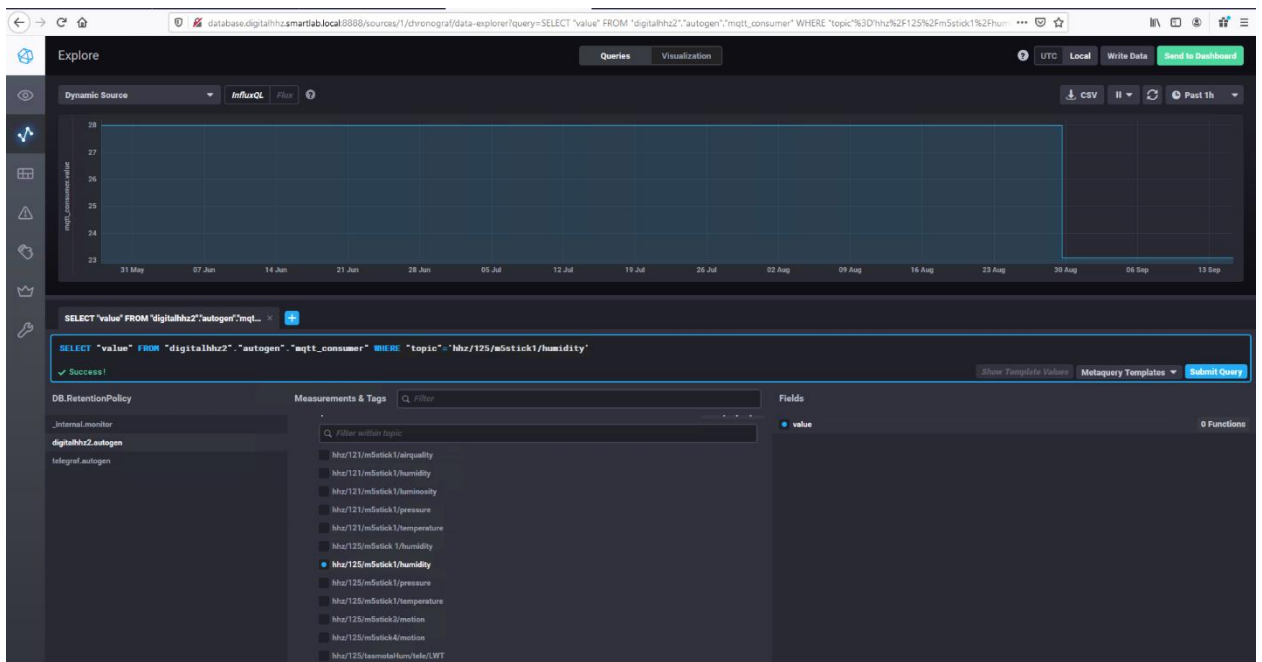


Abbildung 46 Chronograf Explorer

7.6.3 Dashboard:

Im Dashboard werden in Echtzeit Daten, die über Telegraf kommen und in die DB geschrieben werden, dargestellt.

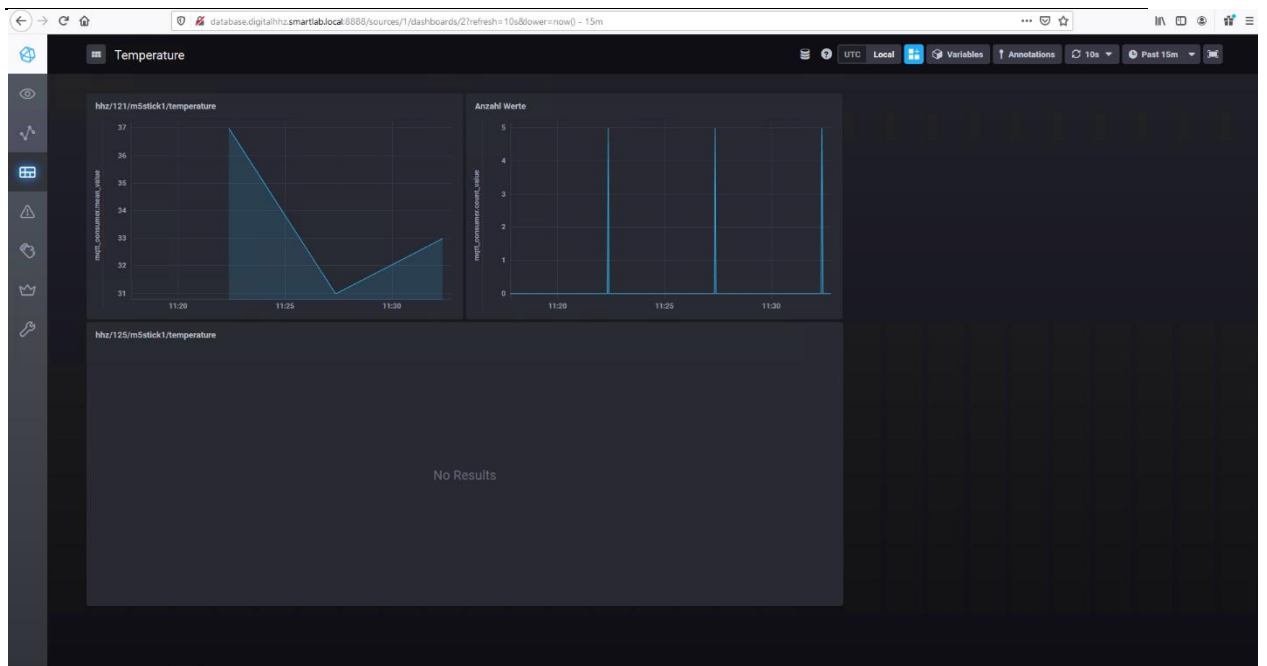


Abbildung 47 Chronograf Dashboard

7.7 Backup und Dropbox Uploader

Hinsichtlich des Ziels der Archivierung der Sensordaten, muss monatlich automatisch ein DB-Backup erstellt werden. Die Daten sollen per Zieldefinition nicht inkrementell, sondern komplett extrahiert werden.

InfluxDB ermöglicht ein Backup der Daten mittels Kommandozeilen-Befehl, was im Folgenden als Beispiel aufgezeigt wird:

```
$ influxd backup -portable -database mydatabase -host <remote-node-IP>:8088 /tmp/mysnapshot
```

Für nähere Information siehe Dokumentation von Influx: https://docs.influxdata.com/influxdb/v1.8/administration/backup_and_restore/

Das Backup wird automatisch mittels Crontab ausgeführt, der monatlich immer am ersten Tag läuft.

7.7.1 Automatischer Backup der Daten in der Dropbox

Mit folgender Anleitung kann nachvollzogen werden, wie der Dropbox Uploader aufgesetzt wurde und wie dieser funktioniert: <https://www.addictivetips.com/ubuntu-linux-tips/use-dropbox-from-the-linux-command-line/>

Der Befehl für den Upload wurde in ein crontab geschrieben, das immer am Monatsanfang läuft→

Um den Upload auf Dropbox mit dem Dropbox Uploader durchzuführen, muss zuerst die Datei nach */home/Dropbox-Uploader/* verschoben werden. Anschließend muss der

Upload mit folgendem Befehl ausgeführt werden. Es sei beachtet, dass der Zielpfad angegeben werden muss.

```
./dropbox_uploader.sh upload testfile.jpg /dropbox/whatever/folder/you/want
```

7.7.2 Manuelles Backup

1. Login auf database VM (Anleitung siehe oben)
2. Backup Befehl in cmd-line:

```
influxd backup -portable -database digitalhhz2 -host localhost:8888 /tmp/mysnapshot
```

3. Backup sollte erfolgreich in das angegebene Ziel (/tmp/mysnapshot) erstellt worden sein
4. Backup besteht aus mehreren Fragmenten + meta + manifest file
5. Mit WinSCP können die Daten auf einen lokalen Rechner gezogen werden

7.7.3 Crontab einsehen und ändern

<https://www.stetic.com/developer/cronjob-linux-tutorial-und-crontab-syntax/>

1. Login auf database VM
2. Eingabe cmd: crontab -e
3. Zeile des jeweiligen CronJobs ändern und mit strg+x speichern und schließen
4. In var/syslogs können die Meldungen ausgelesen werden

Der Crontab läuft immer am ersten jeden Monats:

```
0 0 1 * * /root/dropboxbackup.sh >> /var/log/cronbackup.log
```

7.7.4 Shell Script

Der Shell-Script zum automatischen Backup wird immer am ersten des Monats durchgeführt.

```
#!/usr/bin/bash
```

```
influxd backup -portable -database digitalhhz2 -host localhost:8088
/tmp/influxbackup
if [[ $? == 0 ]]; then
/root/Dropbox-Uploader/dropbox_uploader.sh upload /tmp/influxbackup/*
/
if [[ $? == 0 ]]; then
rm -r /tmp/influxbackup/
fi
fi
```

7.8 DigitalHHZ2MonthlyUploader Dropbox developer App

7.8.1 Dropbox-Zugangsdaten

Email: digitalhhz@gmail.com

Password: (siehe Passwort-Datei)

Der folgenden Command lädt alle Dateien innerhalb des angegeben Ordners hoch.

```
dropbox-Uploader# ./dropbox_uploader.sh upload /tmp/influxbackup/* /
```

7.9 Daten-Replay

Zum Replay historischer Daten aus einer CSV-Datei wurde im Rahmen des Projektes ein kleines Python-Script geschrieben (MQTT_replay.py). Der Benutzer kann den Dateinamen, die Replaygeschwindigkeit und einen Simulationsraum für das Topic vorgeben. Wichtig ist, dass die CSV-Datei im selben Ordner liegt, wie das Python-Skript.

Hier ist eine beispielhafte Ausgabe des Scripts zu sehen:

```
Hallo das ist das Digital HHZ CSV Replay-Tool
Name der CSV-Datei ohne Dateiendung: test
Beschleunigungsfaktor normal = 1 doppelte Geschwindigkeit = 2: 1
Simulationstopic: sim_125
<class 'pandas.core.series.Series'>
      time      mqtt_consumer.topic mqtt_consumer.value
time1
1 2020-09-28 18:25:59.796000+00:00  hhz/125/m5stick1/pressure      956.68
1601317559796000000
2 2020-09-28 18:25:59.807000+00:00  hhz/125/m5stick1/humidity      36.20
1601317559807000000
3 2020-09-28 18:26:09.899000+00:00  hhz/121/m5stick1/temperature      18.00
1601317569899000000
4 2020-09-28 18:26:09.950000+00:00  hhz/121/m5stick1/humidity      43.00
1601317569950000000
5 2020-09-28 18:26:09.952000+00:00  hhz/121/m5stick1/airquality      549.00
1601317569952000000
Lesen erfolgreich
Replay läuft
Sleeptime: 0.011
Sleeptime: 10.092
Sleeptime: 0.051
Sleeptime: 0.002
Sleeptime: 25.712
Sleeptime: 0.019
Sleeptime: 0.016
```

8 Fehlermanagement im Digital HHZ

In der folgenden Tabelle werden mögliche Fehlerszenarien aufgelistet und wie diese erkannt werden.

Tabelle 4 Fehlerszenarien

Fehler	Lösung	Status	Beschreibung
Absturz MQTT-Broker	Redundanter MQTT-Broker	2	Stürzt der lokale Broker ab, erhält der andere Broker keine Nachrichten mehr
Absturz Sensor	Node-RED Dashboard	2	Sobald kein Sensorwert innerhalb eines Intervalls beim Dashboard empfangen wird, wird der Sensor als offline angezeigt.
Internetverbindungsprobleme	Node-RED Dashboard	1	Aktuell wird auf dem Dashboard die Internetlatenz angezeigt.
Stromausfall	-	0	Für den Fall eines Stromausfalls gibt es noch keine Lösung, um die Infrastruktur erhalten zu lassen.
Absturz Appliance	Avahi	1	Der Absturz einer Appliance wird mithilfe von Avahi erkannt und auf dem Dashboard angezeigt.
Absturz Dashboard	-	0	Das Dashboard wird auf dem Smartlab gehostet.
Hacking: ungewollter Client	<ul style="list-style-type: none">▪ MQTT-Broker Credentials▪ Dashboard zeigt verbundene Clients an	2	Der MQTT-Broker ist mit Credentials geschützt. Darüber hinaus wird die Anzahl der Clients auf dem Dashboard angezeigt.
Absturz der Datenbank	<ul style="list-style-type: none">▪ Automatisches Backup mit Crontab	2	Backups werden automatisch am Anfang des Monats auf Dropbox geladen. Diese können jedoch auch manuell getriggert werden.
Verlust von Sensorwerten	<ul style="list-style-type: none">▪ Automatisches Backup mit Crontab	2	Die Sensorwerte können über das Backup nochmal neu eingespielt werden. Hierfür kann das Python-Skript genutzt werden.

0 = Keine Lösung 1 = In Bearbeitung 2 = Lösung implementiert

9 Aufsetzen einer neuen Appliance

Dieses Kapitel beschreibt die notwendigen Schritte, um eine neue Appliance in der HHZ Smart Lab Infrastruktur aufzusetzen. Die Infrastruktur ist darauf ausgelegt, neue Services für das smarte HHZ als eine Appliance zu erstellen. Das vereinfacht die Skalierbarkeit der Infrastruktur und vereinheitlicht das Aufsetzen eines neuen Service, auch ohne tiefe Kenntnisse über die Infrastruktur. Der Raspberry Pi muss so installiert und konfiguriert werden, dass der Service in die Infrastruktur integriert ist und mit den anderen Appliances kommunizieren kann. Auch die Sensoren und Aktoren müssen konfiguriert werden bzw. bestimmten Strukturen folgen.

Dem entsprechend sind drei grundlegende Schritte notwendig:

1. Einrichten und Konfigurieren von Raspbian auf einem Raspberry Pi
2. Installation einer Node-RED-Instanz auf dem Raspberry
3. Hinzufügen des MQTT-Brokers innerhalb der Node-RED-Instanz mit den Zugangsdaten des Digital HHZ-Brokers

Nun können Sensordaten vom Broker subscribed und je nach Anwendung auch Aktoren aus Node-RED mit entsprechenden Nodes angesprochen werden. Hierbei ist die generelle Topicstruktur des Digital-HHZ's zu beachten.

Um eine Appliance dauerhaft in das Digital HHZ einzubinden sind folgende weitere Schritte zwingend notwendig:

4. Konfigurieren von Avahi nach oben beschriebener Anleitung
5. Anlegen der Appliance innerhalb von ThingsBoard als Objekt (ggf. Beziehung zu anderen Räumen und mindestens dem HHZ-Objekt und dem Appliance-Objekt herstellen)
6. Raspberry-Pi der Appliance mit einem QR-Code versehen der nach folgendem Schema aufgebaut ist und die Asset-ID aus ThingsBoard enthält → <http://nodered.digitalhhz.smartlab.local:1880/appliance/<Asset-ID>>

In den nachfolgenden Kapiteln werden einzelne Schritte noch einmal genauer erläutert

9.1 Aufsetzen des Raspberry Pis

Als erster Schritt muss das Betriebssystem „Raspbian“ auf den Pi aufgespielt werden. Das Image dazu kann hier <http://www.raspberrypi.org/downloads> heruntergeladen werden. Nach Entpacken der ZIP-Datei muss das Image auf die SD-Karte gespielt werden. Dieser Schritt unterscheidet sich je nach Betriebssystem des verwendeten Laptops (Google-Suche).

Sobald Raspbian auf der SD-Karte liegt, kann diese in den Raspberry Pi gesteckt werden. Der Raspberry Pi muss nun an den Strom, an einen Bildschirm, an eine Tastatur und zur vereinfachten Verwendung an eine Maus angeschlossen werden. Sobald der Raspberry Pi Strom bekommt, startet er.

Der Hostname vom Raspberry Pi ist standardmäßig „raspberrypi“ und bei mehreren Raspberry Pis im Netzwerk kann der richtige Pi schlecht gefunden werden. Deswegen muss der Hostname geändert und entsprechend der Appliance benannt werden. Um das zu tun, kann auf der Oberfläche des Raspberry Pi das Menü ausgewählt werden, dann Preferences und dann Raspberry Pi Configuration. Dort kann der Hostname geändert werden und der Raspberry Pi anschließend neu gebootet werden, damit die Änderung in Kraft tritt.

Weitere Möglichkeiten zum Ändern des Hostnames können hier gefunden werden: <https://www.tomshardware.com/how-to/raspberry-pi-change-hostname>

9.2 Node-RED Installation

Als nächster Schritt muss Node-RED auf dem Raspberry Pi installiert werden. Dazu muss ein Terminal geöffnet werden und folgender Befehl ausgeführt werden:

```
curl -sL https://raw.githubusercontent.com/node-red/linux-installers/master/deb/update-nodejs-and-nodered
```

Wenn eine Fehlermeldung kommt, muss vorher folgender Befehl ausgeführt werden:

```
sudo apt install build-essential git
```

Eine komplette Anleitung ist hier zu finden: <https://nodered.org/docs/getting-started/raspberrypi>

Um auf Node-RED zuzugreifen, muss im Browser folgendes eingegeben werden:

IP:1880

Die IP kann durch den Befehl `hostname -I` im Terminal herausgefunden werden.

9.3 Erklärung des MQTT Datenflusses

Folgendes Bild soll den MQTT Datenfluss veranschaulichen:



Abbildung 48 MQTT Datenfluss

Es gibt einen zentralen MQTT-Broker im HHZ, der auf einem Raspberry Pi läuft. Alle Appliances, Sensoren und Aktoren kommunizieren über diesen Broker. Wenn ein Sensor Werte misst, published er diese an den MQTT-Broker. Die Appliances, also die Node-RED Instanzen, subscriben sich auf ihre relevanten Topics, so bekommen sie die Sensordaten. Die Appliance hat nun einen Node-RED Flow implementiert, das bedeutet, ein Befehl ist definiert, der bei bestimmten Sensorwerten ausgeführt werden soll. Diesen Befehl published die Appliance an den MQTT-Broker. Die Aktoren wiederum subscriben sich am Broker für dieses Topic, das den Befehl enthält. Damit ist der Ablauf von der Messung der Sensorwerte bis zu der Ausführung der Aktoren beschrieben.

9.4 Broker Konfiguration

Der Aktor der Appliance muss mit dem MQTT-Broker verbunden werden. Der Aktor kann zum Beispiel eine Tasmota-Lampe sein. Es muss auf die Konfigurationsoberfläche des Aktors zugegriffen werden und nach einer Einstellung gesucht werden, die es ermöglicht, MQTT zu konfigurieren.

Die Zugangsdaten zum Broker können aus der Passwortdatei entnommen werden.

9.5 Erklärung der Topic Struktur

Alle Sensoren und Aktoren müssen der folgenden Topicstruktur entsprechen:

```
hhz/<room>/<deviceId>/<measuredValue>
```

Bsp.:

```
hhz/125/m5stick/temperature
```

```
hhz/125/m5stick/motion
```

Dabei ist wichtig, dass die Groß- und Kleinschreibung beachtet wird.