Allgemeines Wissen

- Grafana
- InfluxDB
- LoRa & TTN
- MQTT
- Telegraf

Grafana

Diese Seite beschreibt den Umgang mit Grafana, einem Tool zur Visualisierung von Daten.

Theorie

Aus der Wikipedia-Seite:

> Grafana ist eine plattformübergreifende Open-Source-Anwendung zur grafischen Darstellung von Daten aus verschiedenen Datenquellen wie z. B. InfluxDB, MySQL, PostgreSQL, Prometheus und Graphite.
>
> Die erfassten Rohdaten lassen sich anschließend in verschiedenen Anzeigeformen ausgeben. Diese können dann zu sogenannten Dashboards zusammengefügt werden. Die Anzeigemöglichkeiten und Datenquellen können zudem mittels Plug-ins erweitert werden.

Zugriff auf InfluxDB Daten

Der Zugriff auf die InfluxDB erfolgt mittels User/Passwort. Es empfiehlt sich einen dedizierten Satz an Credentials zu verwenden, und dem Benutzer nur Leserechte zu gewähren. Sollen mehrere Datenbanken verwendet werden, muss jede als Datenquelle in Grafana angelegt werden. Dabei können auch unterschiedliche Credentials verwendet werden.

Beim Anlegen einer neuen InfluxDB Datenquelle sind folgende Infos anzugeben:

- URL: in dem typischen Docker-Setup ist das http://influxdb:8086
- Auth: Basic auth
- Basic auth details : credentials
- InfluxDB details : nochmal die Credentials

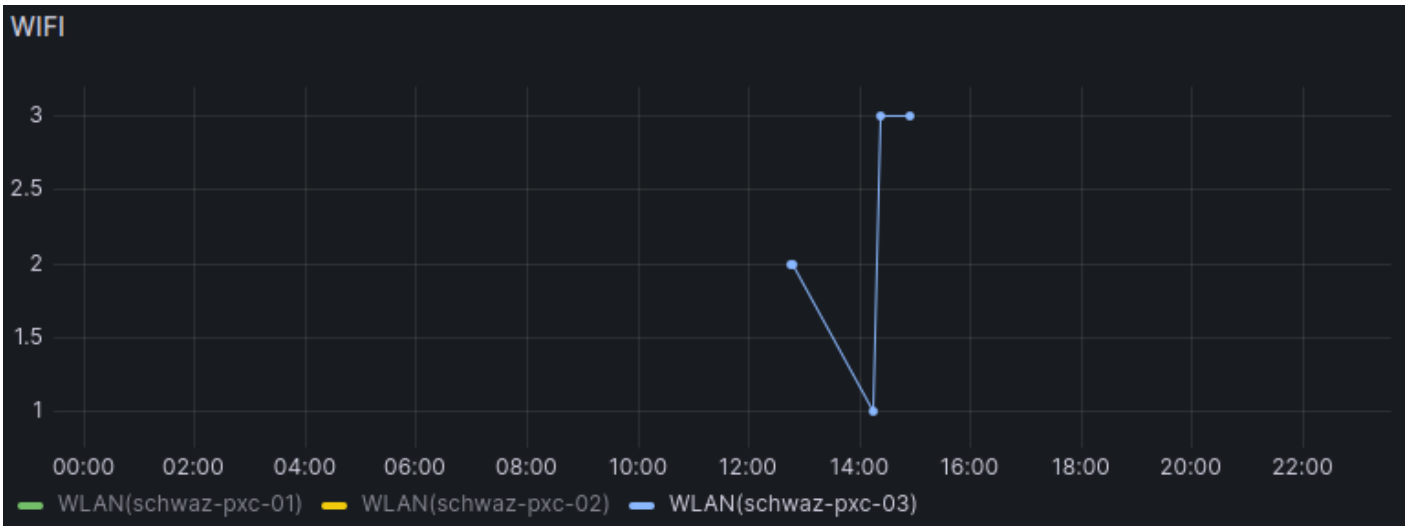
Save & Test sollte eine grüne Bestätigung bringen, dass die Verbindung richtig konfiguriert ist

Benutzung der Oberfläche

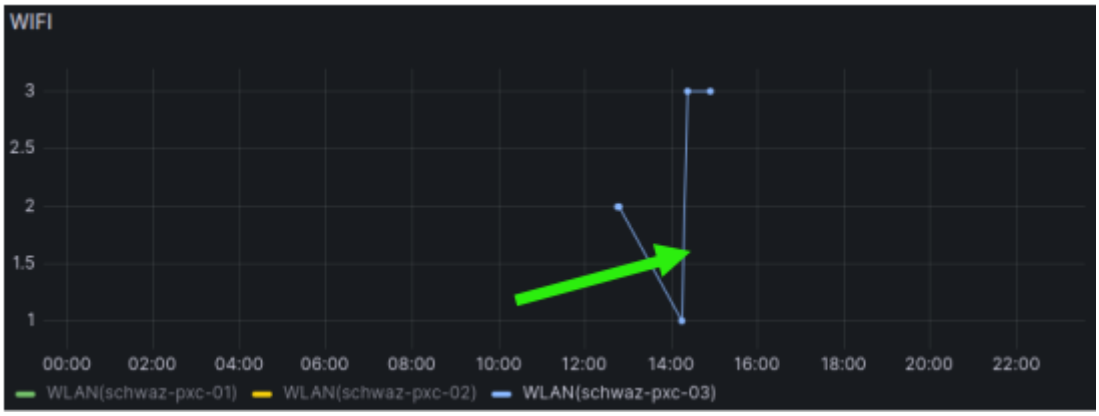
Graphen

Ein Punkt auf dem jeweiligen Graphen entspricht einer übertragenen Messung. Die Kurven der 3 Counter sind farblich unterschiedlich dargestellt.

Man kann die Kurve eines einzelnen Counters anzeigen lassen, wenn man auf dessen Namen in der Legende unter dem Graphen klickt. Dabei wird der Name hervorgehoben, die anderen Namen bleiben grau (im Beispiel ist schwaz-pxc-03 selektiert). Eine Mehrfachselktion ist möglich, man hält CTRL/STRG gedrückt während man die Messreihen anklickt.



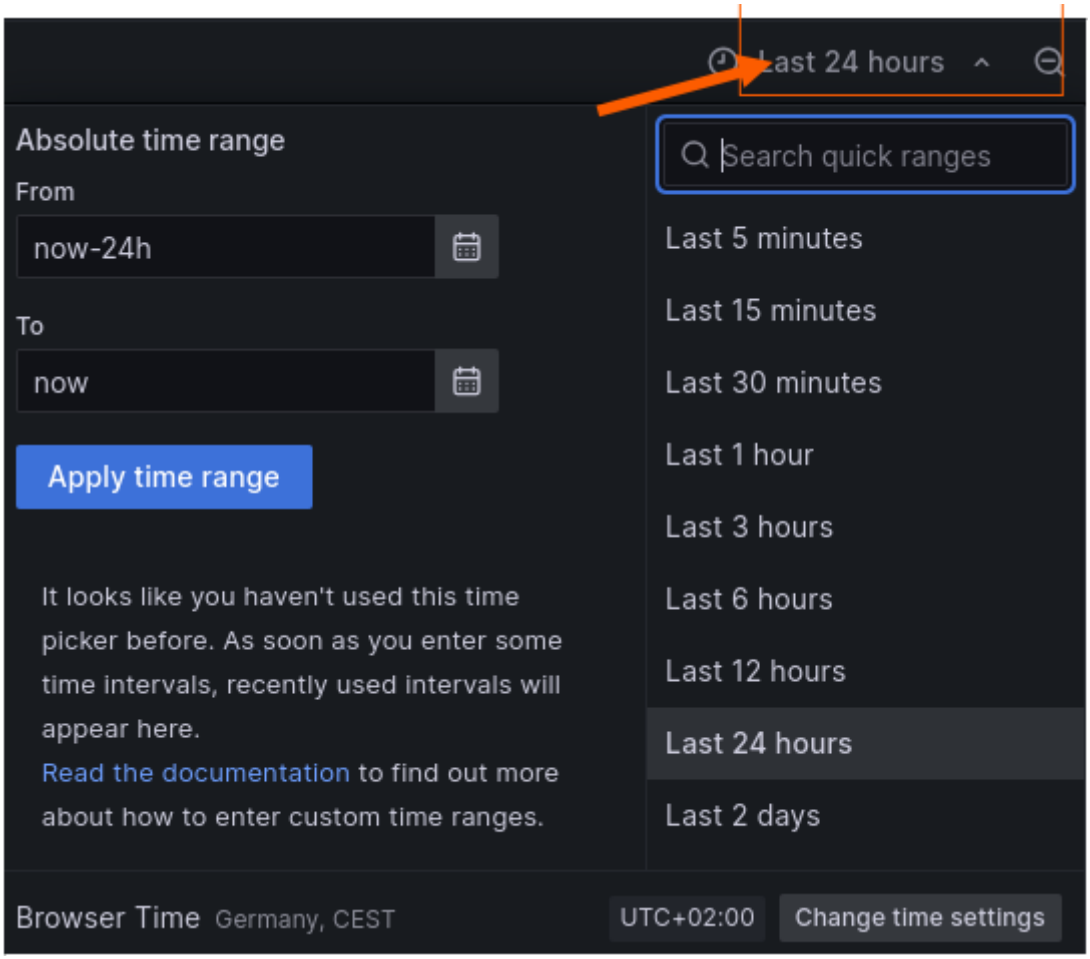
Sind keine Daten empfangen worden, werden die Linien der Graphen einfach mit demselben Wert weitergzogen, bis zum nächsten Datenpunkt (grüner Pfeil).



Navigation im Dashboard

Ändern des Zeitfensters

Per Default werden die Messwerte der letzten 24h dargestellt. Dies kann über den Knopf in der oberen Menüleiste geändert werden (orangefarbener Pfeil).



Beim Klicken öffnet sich ein kleines Fenster, in dem man auf der rechten Seite ein vordefiniertes Zeitfenster auswählen kann (mit dem Mausrad scrollen, um weitere zu sehen, man kann bis zu 2 Jahre auswählen). Auf der rechten Seite kann man das Fenster (von – bis) händisch eingeben, falls eine genauere Kontrolle erwünscht ist.

Dashboard aktualisieren

Per Default aktualisiert sich das Dashboard jede Minute. Das kann in der oberen Menüleiste geändert werden (orangefarbener Knopf). Dabei stehen Aktualisierungsraten von *alle 5 Sekunden* bis *jeden Tag* zur Auswahl.

Eine sofortige Aktualisierung der Daten kann mittels dem Knopf links davon (grüner Pfeil) erzwungen werden.

InfluxDB

"InfluxDB ist eine Datenbank für Zeitreihen" (Wikipedia)

Im Gegensatz zu einer klassischen relationellen Datenbank wie z.B. MySQL, ist InfluxDB dafür optimiert, Daten mit einem Zeitbezug zu speichern, und ist bei der Wiedergabe sehr effizient

- Daten aus einem spezifischen Zeitbereich zu liefern, z.B. “Werte der letzten 2 Tage”
- Daten aus einem Datensatz zu verarbeiten, z.B: “Maximalwerte der letzen 2Tage, über ein 1-stündiges Fenster”
- Datensätze anhand von Tags zu selektieren, z.B. “Werte der letzten 2 Tage der Sensoren sens1 und sens2”

Im Oberlab wird Projektbezogen eine InfluxDB in einer virtuellen Maschine auf dem Oberlab Server betrieben.

InfluxDB ist mittlerweile in der Version 2 verfügbar, die von den Daten und der InfluxQL Sprache sind mit der Version 1 und der Sprache Flux inkompatibel.

Datenstrukturen

In MySQL muß die Datenstruktur beim Erstellen der Tabellen festgelegt werden. Danach werden pro Zeile die verschiedenen Spalten mit Daten befüllt.

Bei InfluxDB werden die Strukturen der Tabellen (nennt sich “bucket”) dynamisch angepasst, d.h. neue “Spalten” werden automatisch erzeugt wenn der Datensatz neue “Spalten” beinhalten.

An sich gibt es keine “Spalten”, sondern 2 verschiedene Datentypen: Werte und Tags

InfluxDB Befehle

Bucket erstellen

```
CREATE DATABASE openhab_db
```

Benutzer anlegen

`CREATE USER openhab WITH PASSWORD 'AnotherSuperbPassword456-'`

`GRANT ALL ON openhab_db TO openhab`

`GRANT READ ON openhab_db TO grafana`

Passwort eines Benutzers ändern

`influx user password -n johndoe -t My5uPErSecR37t0k3n`

`set password for "testuser" = 'testing'`

Backup/restore

`influxd backup -portable -database db <path-to-backup>`

`influxd restore -portable -db db <path-to-backup>`

Daten einspeisen

Siehe auch https://docs.influxdata.com/influxdb/v2/write-data/

Datenstruktur einsehen

`show measurements`

`show series`

`show tag keys`

`show field keys`

`show retention policies`

Daten selektieren

Siehe auch https://docs.influxdata.com/influxdb/v2/query-data/

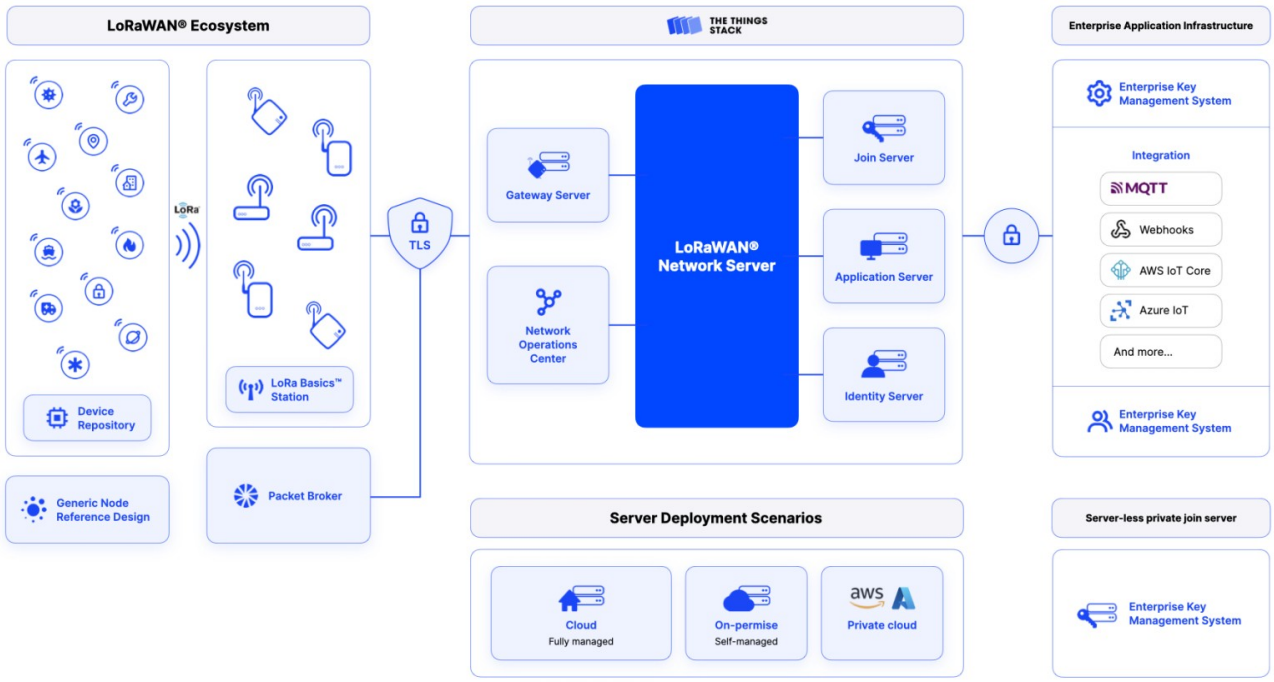
Datenexport zu csv

```
influx -username USER -password PASSWORD -database "oberlab" -execute 'SELECT *
FROM "autogen"."counter" WHERE ("sensor" =~ /^(pxc-0001-oberlab)$/) AND time >= 1669845600000ms and
time <= 17
01295200000ms' -format csv > /etc/influxdb2/test2.csv
```

LoRa & TTN

Diese Seite beschreibt den Einsatz von LoRa und The Things Network (TTN), um Daten von Geräten einzusammeln, um sie anschließend in eine Datenbank abzulegen, und sie final zu visualisieren.

Die gesamte Kette der Datenübertragung wir im folgenden Bild zusammengefasst, Details finden sich in den weiteren Abschnitten



LoRa Datenübertragung

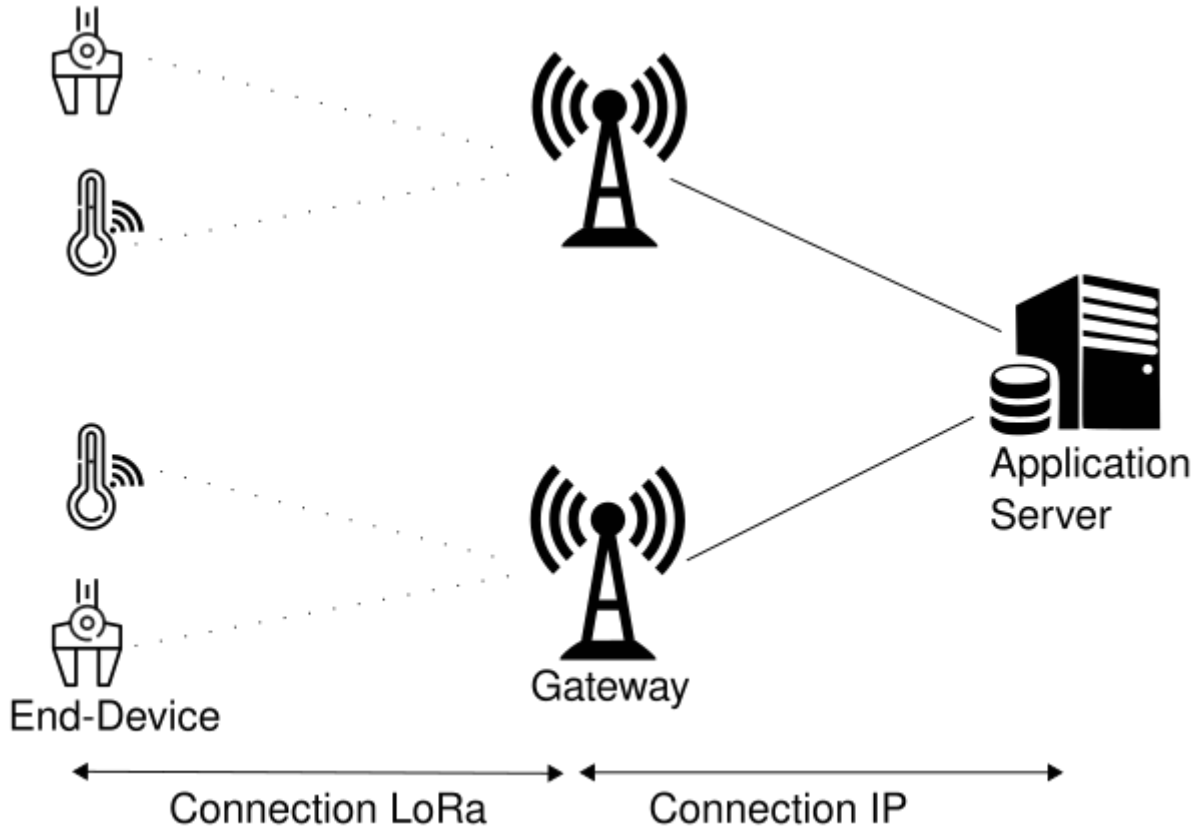
Im IoT Bereich hat sich das Übertragungsprotokoll LoRa (Long Range) etabliert.

Es funkt im freien Frequenzband von 433 bzw 866MHz, und ermöglicht Datenübertragung über Entfernungen von 2km innerorts bis über 40km im Freien.

Es werden allerdings nur wenige Bits übertragen, wodurch sich das Protokoll bestens zum Übermitteln von Status-Informationen, Sensor-Meßwerte... eignen. Das Protokoll ist prinzipiell bidirektional, sodass man damit auch Befehle an die Geräte senden kann.

Als Gegenstelle zum IoT Gerät mit seinem LoRa Modul (z.B. ESP TTGO) dient ein LoRa Gateway, welches typischerweise eine Brücke in's Internet aufbaut (per LAN Kabel, WLAN oder Mobilfunk),

und weiter zu einem Server, der die Daten weiter verarbeitet.



Diese Gateways kosten zwischen 100€ und 800€, weshalb es sinnvoll ist, die Gateways zu teilen. Damit erhöhnt sich die Abdeckung, bzw die Erreichbarkeit der Geräte.

The Things Network

Das Projekt The Things Network (TTN) verwaltet ein solches, weltweites, Netzwerk von Gateways. Man kann das Netzwerk kostenlos verwenden, aber auch eigene Gateways dem Netzwerk hinzufügen, was wir im Oberlab auch tun.

TTN Logo (Wikipedia)

Auch wenn alle Geräte Daten über die Gateways senden können, sind diese mit einem eigenen AES -Schlüssel gesichert, sodass nur der Eigentümer des Geräts die Daten im Klartext sehen kann. Andere Teilnehmer, sowie ebenfalls der Betreiber des Gateways, über den die Datenpakete laufen, sehen nur die verschlüsselte Kommunikation. Die Entschlüsselung erfolgt auf den TTN-Server, wo der Eigentümer die Daten abholen kann.

Es gibt verschiedene Übersichtskarten

- Die offizielle Karte der Gateways von TTN
- TTN Mapper Heatmap mit den Gateways und den Signalraten
- TTN Mapper Radar mit den Paketübetragungen, zeigt die Verbindungen zwischen Geräten und Gateways

Datenpakete

Um die übertragene Datenmenge weiter zu reduzieren, werden die Bits auf das Minimum reduziert, also zB 8 Bit um einen Integer-Zählerwert von 0 bis 255 abzubilden. Der TTN-Server konvertiert diese roh-Bits in eine Datenstruktur, mithilfe einer in Javascript geschriebenen Dekoder-Funktion (es gibt auch eine Enkoder-Funktion wenn man struturierte Daten in Bits wandeln will, um sie zum IoT Gerät zu versenden), zu finden unter “Payload Formatter”.

Die Funktion weist z.B. dem Feld “anzahl_wifi_geräte” den Wert aus dem 5. Byte des Datenpakets zu, bzw dem Feld “batterie_spannung” der Wert aus den 7. und 8. Bytes zu (zuerst nach Ganzzahl gewandelt, dann durch 1000 geteilt - von mV nach V umgerechnet). Die Felder werden unter “decoded_payload” in der JSON Datenstruktur abgelegt, siehe Data Formats Doku:

```
"decoded_payload": {
    "battery": 3754,
    "ble": 8,
    "ble_new": 8,
    "wifi": 1,
    "wifi_new": 0
  },
```


Applikationen

Alle Oberlab Projekte werden in der TTN Console über den Oberlab-Account verwaltet. TTN strukturiert die unterschiedlichen Projekte in “Applications”, z.B. Connecting Peaks. Innerhalb einer Applikation werden Geräte angemeldet - deren Datenströme sind dann im Kontext der Applikation abrufbar. Pro Applikation legt man ebenfalls eine Decoder- und Encoder Funktion an.

In der Console kann man die Live-Daten einsehen, die von TTN empfangen werden. Bei jedem Paket sind sowohl die roh-Daten als auch die dekodierten, strukturierten Daten sichtbar. Damit kann man prüfen, ob der Dekoder richtig funktioniert. Falls am Ende der Verarbeitungskette keine Daten in der Visualisierung (z.B. Grafana) ankommen, mach es Sinn den Datenverkehr live anzusehen, um zu verstehen wo die Kommunikation gestört ist.

Geräte

Geräte werden über ihre DevEUI identifiziert - eine *eindeutige* 16-stellige Hex-Zahl die man selber vergibt wenn man die Geräte einrichtet. Sie melden sich beim TTN an, indem sie die JoinEUI verwenden, ebenfalls eine 16-stellige Hex-Zahl die man pro Applikation angibt. Diese IDs müssen im Gerät bzw in der Firmware einprogrammiert werden. Passen sie nicht, ist keine Anmeldung im TTN-Netzwerk möglich. Es empfiehlt sich daher, die verschiedenen IDs im Projekt zu dokumentieren, eine nachträgliche Änderung geht nicht, man muss dazu das Gerät bei TTN löschen und wieder einrichten! Rechner zum erstellen von EUIs

Die End device ID ist der Name vom Gerät, wie er später in den Datenstrukturen sichtbar sein wird (z.B. *miesbach-pxc-01*). Die ID kann ebenfalls nur beim Einrichten gesetzt werden, später nicht mehr. Der Feld *Name* wird nur für die Auflistung in der TTN Console verwendet, in den Datenstrukturen taucht er nicht auf. Mehr Details in der TTN Doku zum Anmelden von Geräten.

Zugriff auf die Daten

Die empfangenen Daten können dann bei TTN über verschiedene Integrationen abgeholt werden, z.B.

- deren MQTT Server (TTN Doku zu MQTT)
- Node Red
- IFTTT
- Webhooks (z.B. Cayenne)

Bei den Oberlab-Projekten werden die TTN-Daten von deren MQTT Server auf den Oberlab MQTT Server (mqtt.oberlab.de) gespiegelt, und zur weiteren Verarbeitung zur Verfügung gestellt. z.B. können die MQTT Daten mittels Telegraf in eine InfluxDB Datenbank geschrieben werden, bevor sie mit Grafana visuell in Dashboards dargestellt werden.

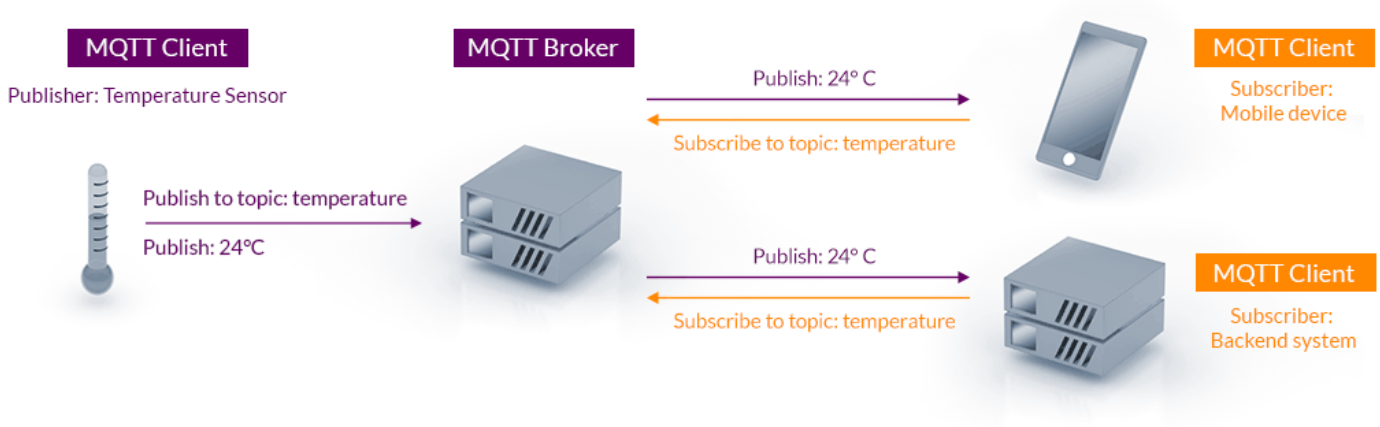
Zur Integration mit den Integrationen sind API Keys notwendig, sie werden pro Applikation erstellt. Idealerweise sollten die Keys nicht mehrfach verwendet werden, sondern es sollte ein Key pro Applikation/Person/… erstellt werden. Keys können verschiedene Zugriffsrechte auf die Daten haben, je weniger Rechte desto sicherer!

MQTT

Diese Seite beschreibt den Einsatz von MQTT um Datenpakete zwischen Applikationen zu vermitteln

MQTT Theorie

Wikipedia zu MQTT: ein offenes Netzwerkprotokoll für Machine-to-Machine-Kommunikation (M2M), das die Übertragung von Telemetriedaten in Form von Nachrichten zwischen Geräten ermöglicht, trotz hoher Verzögerungen oder beschränkter Netzwerke. Entsprechende Geräte reichen von Sensoren und Aktoren, Mobiltelefonen, Eingebetteten Systemen in Fahrzeugen oder Laptops bis zu voll entwickelten Rechnern.



In der MQTT Architektur werden Daten von einem Client zu einem Server (*Broker*) geschickt (*published*). Dieser puffert die Pakete für eine gewisse Zeit.

Pakete werden einem Thema (*topic*) zugewiesen. Topics sind hierarchisch angeordnet (z.B. projekte/connecting_peaks/v2023/miesbach-pxc-01 oder home_automation/lab_gmund/sensoren/tür01), womit man eine gewisse Struktur erstellen kann.

Andere Clients melden sich beim Broker an, abonnieren (*subscribe*) dedizierte Themen oder horchen allem zu. Sobald ein Datenpaket zu einem abonnierten Thema verfügbar ist, sendet ihn der Broker dem Client (Push-Verfahren).

Dank des Zwischenpufferns können zeitversetzt abgeholt werden, bzw können verzögert zugestellt werden falls die Verbindung instabil ist.

Die Kommunikation ist ggf bidirektional, d.h. ein Sensor published ein Paket, welches empfangen wird, und eine Reaktion auslöst, die als weiteres Paket über MQTT published wird, entweder zum Sensor zurück, oder zu einem weiteren Aktuator.

MQTT und TTN

The Things Network bildet die Brücke zwischen IoT Geräte, die per LoRa funken, und Applikationen. TTN stellt einen MQTT Server zur Verfügung, der die Datenpakete weiterleitet.

Die Topics folgen dem Schema: v3/{application id}@{tenant id}/devices/{device id}, wobei

- *application id* ist z.B.oberlab-counter-sandbox, also der Name der Applikation wie der Benutzer sie vergeben hat
- *tenant id* ist “ttn”
- *device id* ist nicht die DeviceEID, sondern der DeviceName

Clients

- Grafische Oberfläche mqtt-explorer für Windows, Mac, Linux
- Kommandozeile unter Linux : z.B. mosquitto-client Paket, welches die Befehle mosquitto_sub und mosquitto_pub bereitstellt.
- Telegraf: Client, um die Datenpakete an eine Influx Datenbank weiterzuleiten

Ober-MQTT Server

Auf dem OberServer im Lab läuft ein zentraler MQTT Server (mit der Mosquitto Software). Er ist unter mqtt.oberlab.de erreichbar. Der Zugang ist allerdings nur mit Login/Passwort möglich. Es sollte pro Client ein eigener Satz an Credentials erstellt werden!

Er verwaltet nciht nur eigene Daten, sondern meldet sich als Client bei anderen Server als Client an (z.B. bei TTN), sammelt und konzentriert die Datenpakete. Somit reicht es, sich mit dem OberMTT Server zu verbinden, um auf alle MQTT-Daten zuzugreifen.

Daten von TTN zu OberMQTT spiegeln

Die Daten der Sandbox Applikation werden zu unseren Server gespiegelt, dabei wird der Topic zu connecting_peaks/sandbox geändert

Der passende Eintrag wird in `/docker/conf/mosquitto.conf` vorgenommen:

```
connection ttn_mqtt_01
address eu1.cloud.thethings.network:1883
remote_password <API Key>
remote_username <API Key User>
local_password <password>
local_username <username>
notifications true
# topic pattern [[[ out | in | both ] qos-level] local-prefix remote-prefix]
# subscribes to the remote topic $SYS/broker/clients/total and republishes the messages received to the local
topic test/mosquitto/org/clients/total
# topic clients/total in 0 test/mosquitto/org/ $SYS/broker/
topic # in 0 connecting_peaks/sandbox/ v3/oberlab-counter-sandbox@ttn/
```

Credentials

Die lokalen Credentials werden in `/docker/conf/mosquitto.conf` abgelegt

Telegraf

TBD