Dokumentation der Netzwerkeinrichtung mit LoRaWAN-fähigen Datenloggern an der TU Berlin

Geothermisches Monitoring am HCBC

Stichworte: LoRaWAN, Dragino, TTN, InfluxDB, Telegraf, MQTT, Starlark

Technische Universität Berlin

Fakultät IV – Planen Bauen Umwelt

Fachgebiet Ingenieurgeologie

B.Sc. Moritz Laubert

M.Sc. Felix Schuman

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

ABP	Activation By Personalization
AES	Advanced Encryption Standard
API	Application Programming Interface
BMWi	Bundesministerium für Wirtschaft und Energie
CLI	Command Line Interface
CSV	Comma-Separated Values
DHCP	Dynamic Host Configuration Protocol
EMU	Energy Monitoring Unit
GUI	Graphical User Interface
GWMS	Grundwassermessstelle
HCBC	Hochschulcampus Berlin-Charlottenburg
HTTP	Hypertext Transfer Protocol
JSON	Java Script Object Notation
LoRa	Long Range
LoRaWAN	Long Range Wide Area Network
IoT	Internet of Things
MQTT	Message Queuing Telemetry Transport
OTAA	Over The Air Activation
QoS	Quality of Service
SQL	Structured Query Language
SSL	Secure Sockets Layer
TLS	Transport Layer Security
TTN	The Things Network
TUB	Technische Universität Berlin
UdK	Universität der Künste
UDP	User Datagram Protocol
WAN	Wide Area Network

1. Einleitung

LoRaWAN ist eine Kommunikationstechnologie mit hoher Reichweite und geringem Energieverbrauch, die sich seit 2015 in Entwicklung befindet und erfahrungsgemäß von Ingenieuren oder Wissenschaftlern ohne explizite Vorkenntnisse aus dem Bereich der Informatik implementiert wird. Der vorliegende Bericht behandelt die Einrichtung eines LoRaWAN-Netzwerks am Hochschulcampus Berlin-Charlottenburg (HCBC) zu Zwecken eines geothermischen Monitorings. Das folgende Kapitel erläutert zunächst die grundlegenden Eigenschaften von LoRaWAN-Technologie. Die Dokumentation der Netzwerkeinrichtung am HCBC enthält eine Beschreibung der Arbeitsschritte bzgl. der Konfiguration der Netzwerkkomponenten und kann als Hilfestellung für die Einrichtung eines eigenen LoRaWAN-Netzwerks genutzt zu werden. Einige Komponenten des Netzwerks stellen abgegrenzte Systeme dar, die im Rahmen ihrer Software oder ihres Protokolls eine interne Terminologie besitzen. Für die Einrichtung benötigte Fachbegriffe oder Terminologie werden an geeigneter Stelle erklärt und sind im Glossar zusammengefasst. Im Glossar aufgeführte Begriffe werden im Text *kursiv* ausgeschrieben und ihre englische Bezeichnung meistens beibehalten. Für die Einrichtung verwendete Ressourcen wurden an entsprechenden Stellen als Fußnote verlinkt. Das letzte Kapitel dieses Berichts evaluiert den Einsatz von LoRaWAN-Technologie und gibt Empfehlungen für die Konzeption der Netzwerkstruktur.

2. Grundlagen von LoRaWAN

2.1 LoRaWAN-Technologie

LoRaWAN-Technologie ist ein wichtiger Bestandteil des globalen Trends zu IoT-Anwendungen (**I**nternet **o**f **T**hings), d.h. der kommunikativen Vernetzung physikalischer Objekte mit dem Internet zu Zwecken der Digitalisierung und Automatisierung. Der Begriff '**LoRaWAN**' ist ein Akronym für '**Lo**ng **Ra**nge **W**ide **A**rea **N**etwork' und meint im eigentlichen Sinne das Netzwerkprotokoll für Geräte mit LoRa-Chip. Daneben bezeichnet es die Gesamtheit der Kommunikation LoRaWAN-fähiger Geräte, welche auf zwei Ebenen stattfindet. Auf der physischen Ebene erfolgt die Kommunikation über elektromagnetische Funkwellen im Radio-Spektrum (LoRa). Die dafür genutzten Frequenzbänder sind unlizenziert und frei verfügbar. Der Spektralbereich der für LoRa reservierten Frequenzen ist länderabhängig und beträgt innerhalb der EU 863-870 MHz. Die digitale Kommunikationsebene (WAN) bezieht sich auf den Datenverkehr im Internet, mit welchem die LoRaWAN-fähigen Geräte verbunden werden sollen. Das LoRaWAN-Protokoll wurde 2015 von der Semtech Corporation, einem Halbleiterunternehmen aus den USA, entwickelt und über die von ihr gegründete Organisation LoRa Alliance verwaltet. Die Technologie wird seitdem stetig weiterentwickelt und zunehmend standardisiert. Für LoRaWAN bestehen potenziell vielfältige Anwendungsmöglichkeiten; z.B. im Bereich Smart-Home, Smart-City-Management, Optimierung industrieller und landwirtschaftlicher Prozesse, Tracking gesundheitsbezogener Daten bei vulnerablen Personengruppen und Monitoring physikalisch-chemischer Umweltphänomene.

Für die drahtlose Vernetzung von Geräten mit dem Internet werden die üblichen Verbindungstypen WLAN, Bluetooth und Mobilfunk hinsichtlich ihrer Leistungsfähigkeit grundsätzlich bevorzugt, jedoch sind nicht bei jedem Einsatzzweck die erforderlichen Voraussetzungen für ihre

erfolgreiche Anwendung gegeben. Für IoT-Anwendungen an schwer zugänglichen Standorten ohne Strom- und Internet-Anschluss und bei IoT-Anwendungen mit beweglichen Geräten, die nicht regelmäßig gewartet werden müssen, hätten die genannten Verbindungstypen entweder einen zu hohen Energieverbrauch, oder sind in ihrer Reichweite limitiert. In derartigen Situationen werden an eine geeignete Kommunikationstechnologie die Anforderungen einer stabilen Verbindung, einer hohen Reichweite und vor allem eines geringen Energieverbrauchs gestellt. Diese Fähigkeitslücke wird durch niederfrequente Funktechnologien wie LoRaWAN oder alternativ LPWAN (**L**ow **P**ower **WAN**) geschlossen. LoRaWAN-Technologie besitzt den Vorteil, dass Prinzipien wie Open-Source, Interoperabilität von Geräten unterschiedlicher Hersteller und geteilte Ressourcennutzung verfolgt werden. Die Datenübertragung erfolgt nicht kontinuierlich, sondern in Form von *Payloads* (Datenpakete), die in festgelegten Aktivitätsfenstern gesendet werden. Da ein geringer Energieverbrauch angestrebt wird, ist keine Übertragung von großen Datenvolumen möglich.

2.2 LoRaWAN-Verbindungen

Der Einsatzzweck von LoRaWAN-Technologie ist die Verbindung von Geräten mit dem Internet, was situationsbedingt mithilfe von herkömmlichen Kommunikationstechnologien nicht möglich wäre. Als Schnittstelle LoRaWAN-fähiger Geräte zum Internet dienen sog. Gateways, welche neben einer LoRa-Antenne über einen LAN- und/oder WLAN-Anschluss verfügen und die empfangenen *Payloads* der Endgeräte weiterleiten. LoRaWAN-Verbindungen sind bidirektional, d.h. es können sowohl die Endgeräte Datenpakete über das Internet verschicken (*Uplink Message*) als auch Daten bzw. Anweisungen der Nutzer empfangen (*Downlink Message*). In urbanen Gebieten kann es aufgrund von Interferenzen zu Datenverlust bei der Übertragung kommen. Ergebnissen einer Feldstudie in Philadelphia zufolge liegt der Anteil von empfangenen zu versendeten *Messages* in dicht besiedelten Gebieten bei ca. 90% und in weniger dicht besiedelten bei 95% (Gilson & Grudsky 2017). Versuche der TUB im Innenstadtgebiet von Berlin ergaben einen ähnlichen Anteil von 89% (Brand 2021). Es gibt drei Betriebsweisen für Endgeräte, welche die Verbindungsstabilität auf der einen Seite und die Energieeffizienz auf der anderen Seite beeinflussen. Endgeräte der Klasse A befinden sich dauerhaft im Standby-Modus, außer wenn Daten übertragen werden. Endgeräte der Klasse B besitzen zusätzlich Aktivitätsfenster, in denen sie empfangsbereit sind. Endgeräte der Klasse C sind immer empfangsbereit, der Energieverbrauch aber derart hoch, dass ein Batteriebetrieb kaum möglich ist[1].

Die Reichweite und Intensität eines LoRa-Signals ist primär von Dämpfungsfaktoren in der Umgebung und sekundär von der Topografie des Geländes abhängig. Da elektromagnetische Wellen in Anwesenheit von Materie abhängig gedämpft werden, ist die Landnutzung entscheidend. Dichtbeton und Stahlkonstruktionen können als annähernd undurchdringlich betrachtet werden. Daraus folgt, dass auf natürlichen und landwirtschaftlichen Flächen die Reichweite von LoRaWAN-Verbindungen zwischen 10-40 km liegt, während sie im urbanen Raum auf wenige 100 m sinkt. Modellierungen zur Reichweite des LoRa-Signals durch Fujdiak et. al. (2018) für die Implementierung eines LoRaWAN-Netzwerks an der Universität von Brno, ergaben eine starke Anisotropie der Signal-Ausbreitung, dessen Reichweite richtungsabhängig zwischen <1 km und 34 km variiert. Die Reichweite der Verbindung erhöht sich mit der relativen geodätischen Höhe des Gateways zu den Endgeräten, weshalb Gateways auf topografischen

[1] https://www.thethingsnetwork.org/docs/lorawan/classes

Erhebungen platziert werden sollten. Im urbanen Raum breitet sich das Funksignal vorzugsweise entlang der Straßen bzw. Häuserschluchten aus (vgl. Abbildung 1). Qualitative Tests durch das FG Ingenieurgeologie zur Bestimmung der Reichweite innerhalb des Berliner Stadtgebiets, ergaben auch bei hoch gelegenen Standorten des Gateways eine maximale Reichweite von weniger als 1 km. Die optionale Erweiterung des Datenloggers mit einer externen Verstärkerantenne führte zu einer Erhöhung der Reichweite um ca. 25%.

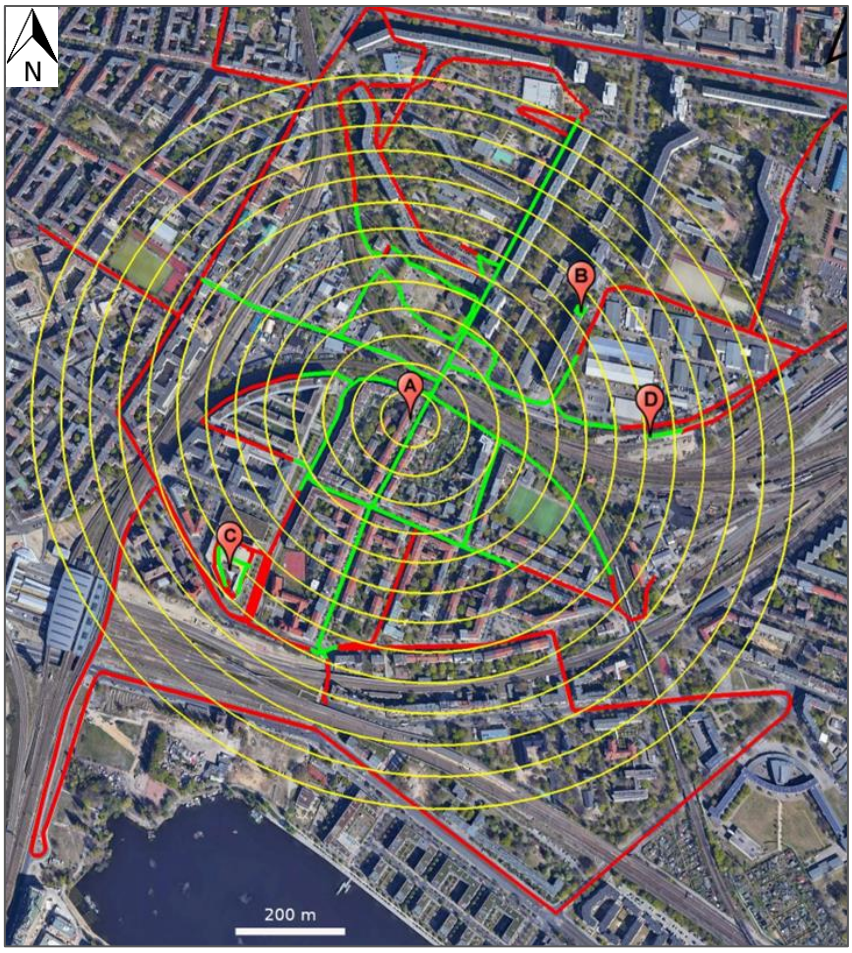

Abbildung 1: Reichweitentest im Gebiet Friedrichshain/Lichtenberg (Brand 2021, Abb.15)

2.3 LoRaWAN-Netzwerke

LoRaWAN-Technologie wurde für den Einsatz im Rahmen eines Netzwerks konzipiert, das üblicherweise aus vier Komponenten besteht; Endgeräte (Sensoren), Gateways, Netzwerkserver und Anwendungsserver. Die Endgeräte werden i.d.R. sternförmig um ein Gateway angeordnet und stehen mit diesem direkt in Verbindung. Eine indirekte Verbindung, sprich zwei oder mehr miteinander gekoppelte Endgeräte, ist theoretisch möglich, verringert aber die effektive Bandbreite. Auch externe Gateways können in ein LoRaWAN-Netzwerk integriert werden, wenn entweder dessen Sicherheitseinstellungen es zulassen nicht authentifizierte *Payloads* weiterzuleiten oder eine Vereinbarung mit den Nutzern vorliegt. Endgeräte und Gateways repräsentieren die physische Ebene eines LoRaWAN-Netzwerks, auf der die Kommunikation über Funkwellen (LoRa) stattfindet.

Die digitale Ebene eines LoRaWAN-Netzwerks umfasst einen Netzwerk- und einen Anwendungsserver. Der Netzwerkserver verwaltet die Kommunikation mit den Geräten und stellt eine Schnittstelle für die drahtlose Interaktion mit den Nutzern dar. Darüber hinaus ist der Netzwerkserver im Falle eines OTAA-Betriebsmodus essenziell für die Datensicherheit im Rahmen des LoRaWAN-Protokolls. Nachdem eine *Payload* vom Gateway empfangen wurde, erfolgt dessen Kommunikation mit dem Netzwerkserver nicht mehr über das LoRaWAN-Protokoll, sondern mittels HTTP- (***H**ypertext **T**ransfer **P**rotocol*) oder MQTT-Protokoll (***M**essage **Q**ueuing **T**elemetry **T**ransport*). Diese *Netzwerk-Protokolle* sind Teil des Internetprotokolls, werden aber auch unabhängig davon zum Datentransfer verwendet. Die Datenbank, in welcher die *Payloads* gespeichert werden, wird im Kontext von LoRaWAN-Netzwerken als Anwendungsserver bezeichnet. Die Kommunikation zwischen Netzwerk- und Anwendungsserver erfolgt meist ebenfalls über das Internetprotokoll. Es ist auch möglich die Funktionen von Netzwerk- und Anwendungsserver parallel von demselben System bereitstellen zu lassen.

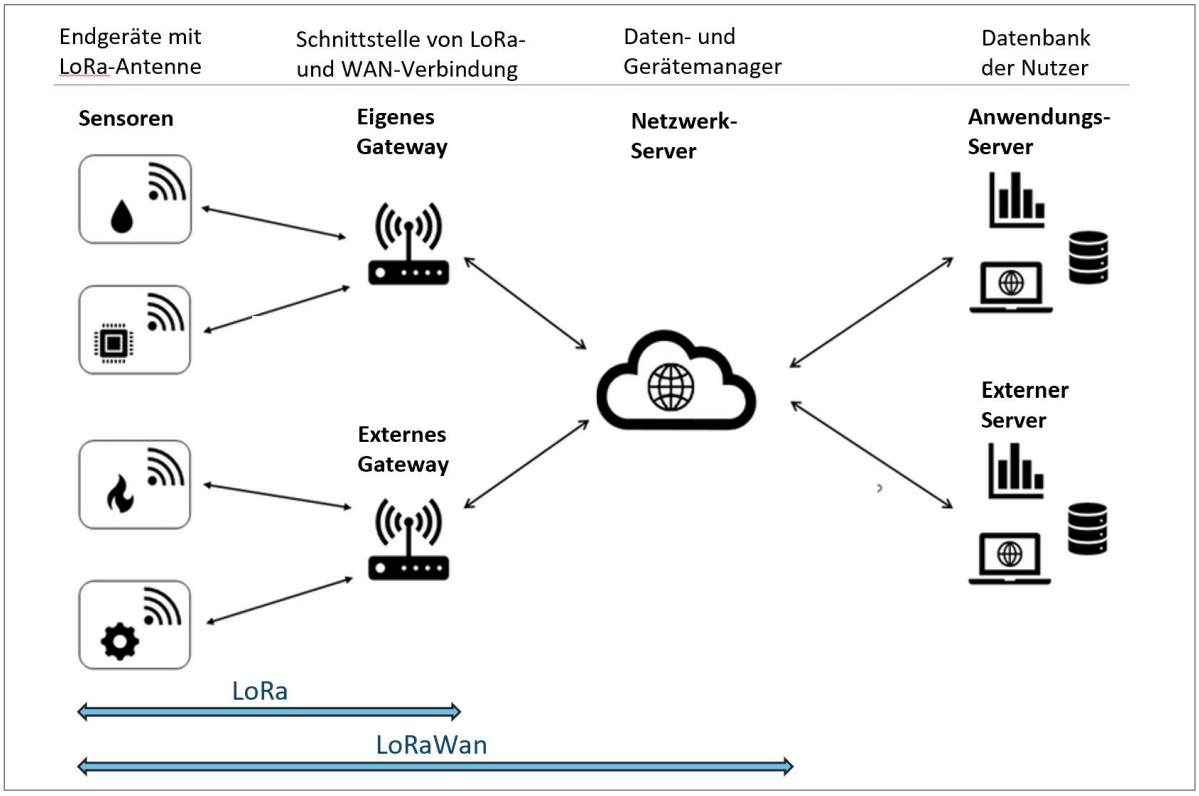

Abbildung 2: Topologie eines LoRaWAN-Netzwerks

2.4 Datensicherheit

LoRaWAN-Verbindungen bedienen sich öffentlich empfangbarer Funktechnologie, weshalb das LoRaWAN-Protokoll vor allem auf einem intelligenten Verschlüsselungskonzept basiert. Es enthält eine mehrstufige Verschlüsselung, um die Datensicherheit der Verbindung über mehrere Schnittstellen des Netzwerks zu gewährleisten (s. Abbildung 3). Für die Verschlüsselung des Datenverkehrs wird der ***A**dvanced **E**ncryption **S**tandard* (AES) genutzt, einem gängigen und nur schwer zu knackenden Verfahren aus der Kryptographie, bei dem derselbe Schlüssel für die Ver- und Entschlüsselung der Daten genutzt wird. Als Masterschlüssel wird vom Administrator ein sog. AppKey (128 Bit) festgelegt, Von diesem werden mit dem NwkS- und AppS-Key zwei weitere 128 Bit Session-Keys abgeleitet. Durch Verwendung von zwei separaten Schlüsseln kann die Speicherung des für die vollständige Entschlüsselung notwendigen

AppKeys in den Anwendungsserver ausgelagert werden, wodurch eine End-to-End Verschlüsselung während des gesamten Datenverkehrs ermöglicht wird[2]. Für den Teil der Datenübertragung im Internet gelten die üblichen Regeln der Netzwerksicherheit, d.h. das Gateway sollte über einen sicheren Internetzugang verfügen. Für die Kommunikation zwischen Netzwerk- und Anwendungsserver wird i.d.R. zusätzlich ein API-Key (128 Bit Schlüssel) generiert, um Datenabfragen zu autorisieren.

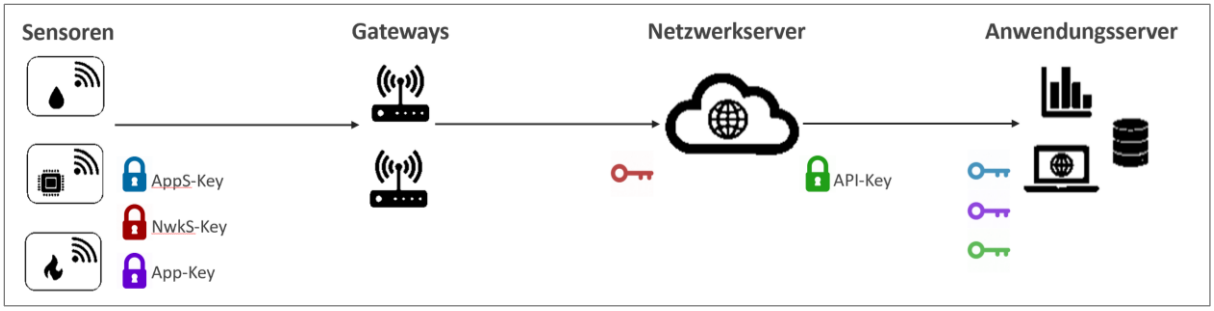

Abbildung 3: Verschlüsselungsebenen innerhalb eines LoRaWAN-Netzwerks

Von der Verschlüsselung abgesehen wird der Datenverkehr einer LoRaWAN-Verbindung über die gegenseitige Authentifizierung von Endgeräten und Netzwerkserver abgesichert. Jedes LoRaWAN-fähige Gerät besitzt eine weltweit einzigartige Geräte-Identifikationsnummer (DevEUI), welche von der LoRa Alliance an die Gerätehersteller in Lizenz verkauft werden. Zusätzlich wird vom Nutzer eine eindeutige Identifikationsnummer der Anwendung (AppEUI) festgelegt. Die Identifikationsnummern werden von Endgeräten und Netzwerkserver genutzt, um sich bei einer Kommunikationsanfrage gegenseitig zu authentifizieren. DevEUI, AppEUI und AppKey sind statische Schlüssel, die dem Netzwerkserver bei der Registrierung von Endgeräten bekannt gemacht werden. Der Prozess der gegenseitigen Authentifizierung beim initialen Aufbau einer LoRaWAN-Verbindung wird als Aktivierung bezeichnet.

Für die Aktivierung sind im LoRaWAN-Protokoll die OTAA- und ABP-Methode zulässig. Bei der OTAA-Methode (**O**ver **T**he **A**ir **A**ctivation) findet regelmäßig eine gegenseitige Authentifizierung von Gerät und Netzwerkserver statt, bei welcher die Session-Keys, d.h. die Verschlüsselung erneuert wird. Im OTAA-Modus betriebene Geräte beginnen nach ihrer Konfiguration in zuvor definierten Zeitabständen Authentifizierungsanfragen (*Join-Request Messages*) über das LoRa-Signal zu versenden, welche die Identifikationsnummern des Geräts enthält. Wenn eine *Join-Request Message* von einem Gateway empfangen und an den Netzwerkserver weitergeleitet wird, versendet der Server in umgekehrter Richtung eine *Join-Accept Message*. In einer *Join-Accept Message* ist die sog. AppNonce enthalten, eine einmalig und zufällig generierte Zeichenfolge, mit welcher in Kombination mit dem statischen AppKey zwei dynamische Session-Keys (NwkSKey und AppSKey) abgeleitet werden (s. Schema in Abbildung 4). Daneben wird dem Endgerät eine dynamische Identifikationsnummer innerhalb des Netzwerks (DevAddr) zugewiesen, und der LoRa-Kanal mitgeteilt, welcher für die *Session* zukünftig genutzt wird. Während der Aktivierung wird von den Endgeräten viel Strom verbraucht, da sie in vergleichsweise kurzen Intervallen *Join-Request Messages* versenden und dauerhaft empfangsbereit für *Join-Accept Messages* sein müssen. Nach Erhalt einer *Join-Accept Message* ist ein Endgerät im OTAA-Modus aktiviert und stoppt die Authentifizierungsanfragen.

[2] https://lora-alliance.org/wp-content/uploads/2020/11/la_whitepaper_security-german_0519.pdf

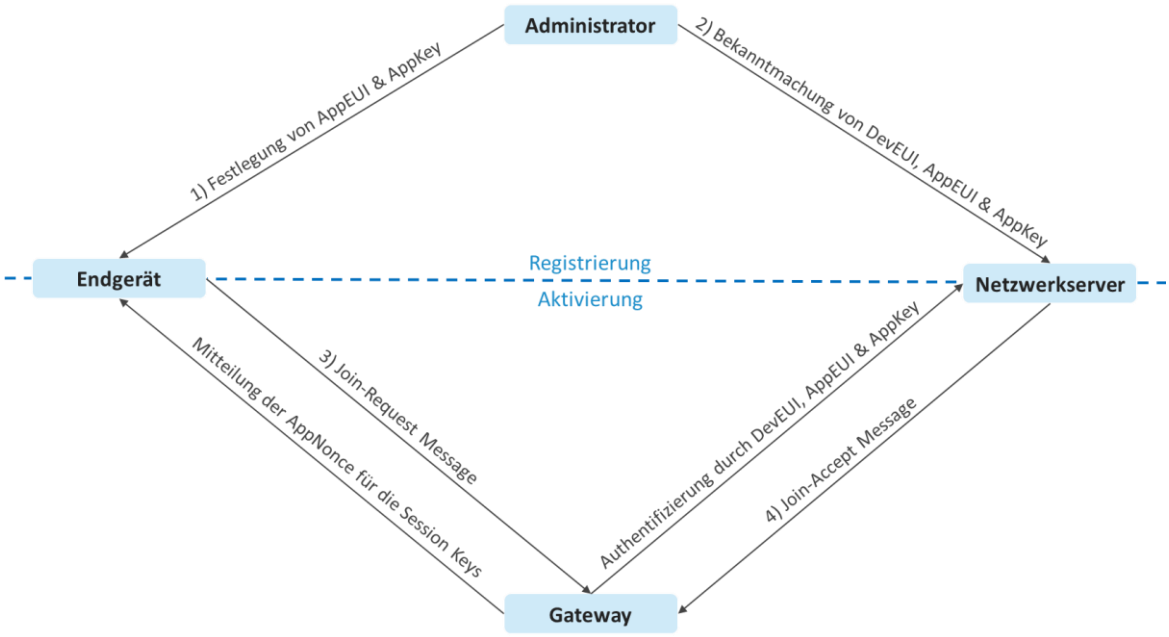

Abbildung 4: Authentifizierungsschema einer LoRaWAN-Verbindung mit der OTTA-Methode

Im Gegensatz zur OTAA-Methode müssen sich Endgeräte bei der Aktivierung mit der ABP-Methode (**A**ctivation **B**y **P**ersonalization) nicht authentifizieren, stattdessen wird der Datenverkehr auf Basis statischer Schlüssel codiert, die im Endgerät gespeichert werden. DevEUI, AppEUI, AppKey, DevAddr, NwkSKey und AppSKey werden während der Konfigurierung des Endgeräts festgelegt. Laut einer vergleichenden Sicherheitsanalyse der beiden Authentifizierungsmethoden von Tomasin et. al. (2017) gilt die ABP- als unsicherer als die OTAA-Methode, lässt sich aber in einfachere Netzwerkstrukturen integrieren, da nicht zwingend ein Netzwerkserver erforderlich ist. Wegen der AES-Verschlüsselung gelten LoRaWAN-Verbindungen zwar als relativ sicher, allerdings handelt es sich bei LoRaWAN um eine sich noch immer in der Entwicklung befindliche Technologie, bei der Sicherheitslücken sukzessive herausgearbeitet werden (Sundaram et. al. 2020). Die Datenübertragung mittels LoRa-Signal gilt gegenüber potenziellen Hackerangriffen naturgemäß als der vulnerabelste Teil einer LoRaWAN-Verbindung. In einer allgemeinen Sicherheitsanalyse erörtern Noura et. al. (2020) mehrere Schwachstellen und bemängeln insbesondere, dass beide Session-Keys (OTAA) und damit die gesamte Verschlüsselung vom AppKey abgeleitet wird. Dieser Master-Key wird im Endgerät gespeichert und kann von Fremden in Erfahrung gebracht werden, wenn das Endgerät nicht vor unbefugtem Zugang geschützt ist. In Bezug auf den Datenschutz wird von Oniga et. al. (2017) empfohlen, nur authentifizierte *Payloads* mittels OTAA-Methode zuzulassen, ein kurzes Intervall für die Generierung der dynamischen Schlüssel einzustellen und die statischen Schlüssel aus Tabelle 1 an einem sicheren Ort zu verwahren.

Tabelle 1: Übersicht der Sicherheitsschlüssel einer LoRaWAN-Verbindung

Schlüssel	Speicher	Funktion	Vergabe durch	Vergabe bei
DevEUI	64 Bit	Eindeutige Identifikation der Endgeräte	Gerätehersteller	Herstellung
AppEUI	64 Bit	Eindeutige Identifikation des Netzwerkservers	Administrator	Registrierung
AppKey	128 Bit	Masterschlüssel zu Erzeugung der Session-Keys	Administrator	Registrierung
DevAddr	32 Bit	Identifikation der Endgeräte im LoRaWAN-Netzwerk	Netzwerkserver	Aktivierung
NwKSKey	128 Bit	Verschlüsselung und Entschlüsselung der Payloads	Netzwerkserver	Aktivierung
AppSKey	128 Bit	Verschlüsselung und Entschlüsselung der Payloads	Netzwerkserver	Aktivierung

3. Das LoRaWAN-Netzwerk am HCBC

3.1 Geothermisches Monitoring

Im Rahmen des vom BMWi geförderten Projekts HCBC zur energetischen Optimierung des Hochschulcampus Berlin-Charlottenburg wird vom Fachgebiet Ingenieurgeologie ein geothermisches Monitoring zum Zwecke der Untersuchung der Beeinflussung des lokalen Temperaturfeldes durchgeführt. Zuvor wurden für das Monitoring u.a. in mehreren Grundwassermessstellen (GWMS) in unmittelbarer Umgebung des Campus die Parameter Druckspiegel, Temperatur und elektrische Leitfähigkeit im monatlichen Intervall händisch gemessen. Durch den Einsatz von LoRaWAN-fähigen Datenloggern während des Monitorings, soll zukünftig zum einen Arbeitszeit durch Automatisierung gespart, und zum anderen zeitlich höher aufgelöste Messdaten gewonnen werden. Für die Pilotphase war angedacht ein LoRaWAN-Netzwerk aus handelsüblichen Komponenten und drei LoRaWAN-fähigen Datenloggern von der UIT GmbH aufzubauen. Das LoRaWAN-Netzwerk wurde in die Energy Monitoring Unit (EMU) eingebunden, einem von der UdK betriebenen Server, der wie im Monitoring Konzept 2019 festgelegt als zentrale Speichereinheit für Messdaten des HCBC-Projekts fungieren soll[3]. Nach der Pilotphase können perspektivisch weitere Messstellen mit LoRaWAN-fähigen Datenloggern bestückt, oder auch andere Geräte in das Netzwerk integriert werden.

3.2 Hardware Komponenten

Die Hardware des Netzwerks von HCBC besteht z.Z. aus drei LoRaWAN-fähigen Datenloggern sowie einem einzelnen eigenen Indoor Gateway vom Hersteller Dragino, welches fensternah in einem hohen Gebäude der TUB platziert wurde. Neben dem eigenen Gateway von HCBC wird ein weiteres Gateway von externen Eigentümern mitbenutzt, um die Empfangsabdeckung zu erhöhen. Dies entspricht im Sinne der LoRaWAN-Technologie dem Prinzip der geteilten Ressourcennutzung. Die Tauchsonden der Datenlogger sind mit Sensoren zur Messung des Druckspiegels bzw. Flurabstands und der Grundwassertemperatur ausgerüstet. Zusätzlich besitzen die Datenlogger interne Sensoren, welche Metadaten zum Betrieb der Datenlogger aufnehmen (Batteriespannung, Gerätetemperatur). Bei zwei Loggern wurde die originäre LoRa-Antenne durch eine verstärkende und ausrichtbare Außenantenne ausgetauscht. Das Messintervall der Logger beträgt 12 Stunden, was bei einem trägen System wie dem Untergrund

[3] https://blogs.tu-berlin.de/hri_hcbc/2022/04/25/elementor-1253/

als ausreichend angesehen wird. Die Datenlogger wurden in unbelegten GWMS in der Umgebung des Campus Berlin-Charlottenburgs verbaut. Zudem ist angedacht, bisherige Sensorsysteme sukzessive durch LoRaWAN-fähige Datenlogger zu ersetzen.

3.3 Netzwerkstruktur

Die digitale Infrastruktur des LoRaWAN-Netzwerks umfasst neben einem Netzwerk- und einem Anwendungsserver auch einen von HCBC konfigurierten Dienst namens Telegraf, der als Schnittstelle zwischen den Server fungiert (s. Abbildung 5). Als Netzwerkserver wird The Things Stack Community Edition verwendet. Dieser Server wird kostenlos von The Thing Network (TTN) zur Verfügung gestellt wird, einem von der LoRa Alliance gegründeten Projekt zur Verbreitung und Standardisierung der LoRaWAN-Technologie. The Things Stack verwaltet die Kommunikation zwischen Datenloggern und Gateways über das LoRaWAN-Protokoll und beinhaltet darüber hinaus eine Cloud, in welcher die *Payloads* zwischengespeichert werden. The Things Stack kommuniziert mit den Gateways über das HTTP-Protokoll, welches aus Gründen der Netzwerksicherheit zusätzlich TLS-verschlüsselt ist. Die *Payloads* im Zwischenspeicher von The Things Stack werden von Telegraf über das MQTT-Protokoll abgerufen, welches speziell auf IoT-Anwendungen ausgelegt ist. Nach dem Abruf der *Payloads* werden diese von Telegraf verarbeitet und an den Anwendungsserver über das *Influx Line Protocol* weitergeleitet. Als Anwendungsserver des LoRaWAN-Netzwerks dient die **E**nergy **M**onitoring **U**nit (EMU), einem Server, der von den Projektbeteiligten vom Fachgebiet Versorgungsplanung und Versorgungstechnik an der UdK eingerichtet und betrieben wird. EMU nutzt das MQTT-Protokoll für die Kommunikation mit externen Anwendungen bzw. den Upload von Daten. Für die Speicherung und den Abruf der Daten ist auf EMU InfluxDB OSS installiert, einer auf Zeitreihendaten spezialisierten Datenbanksoftware.

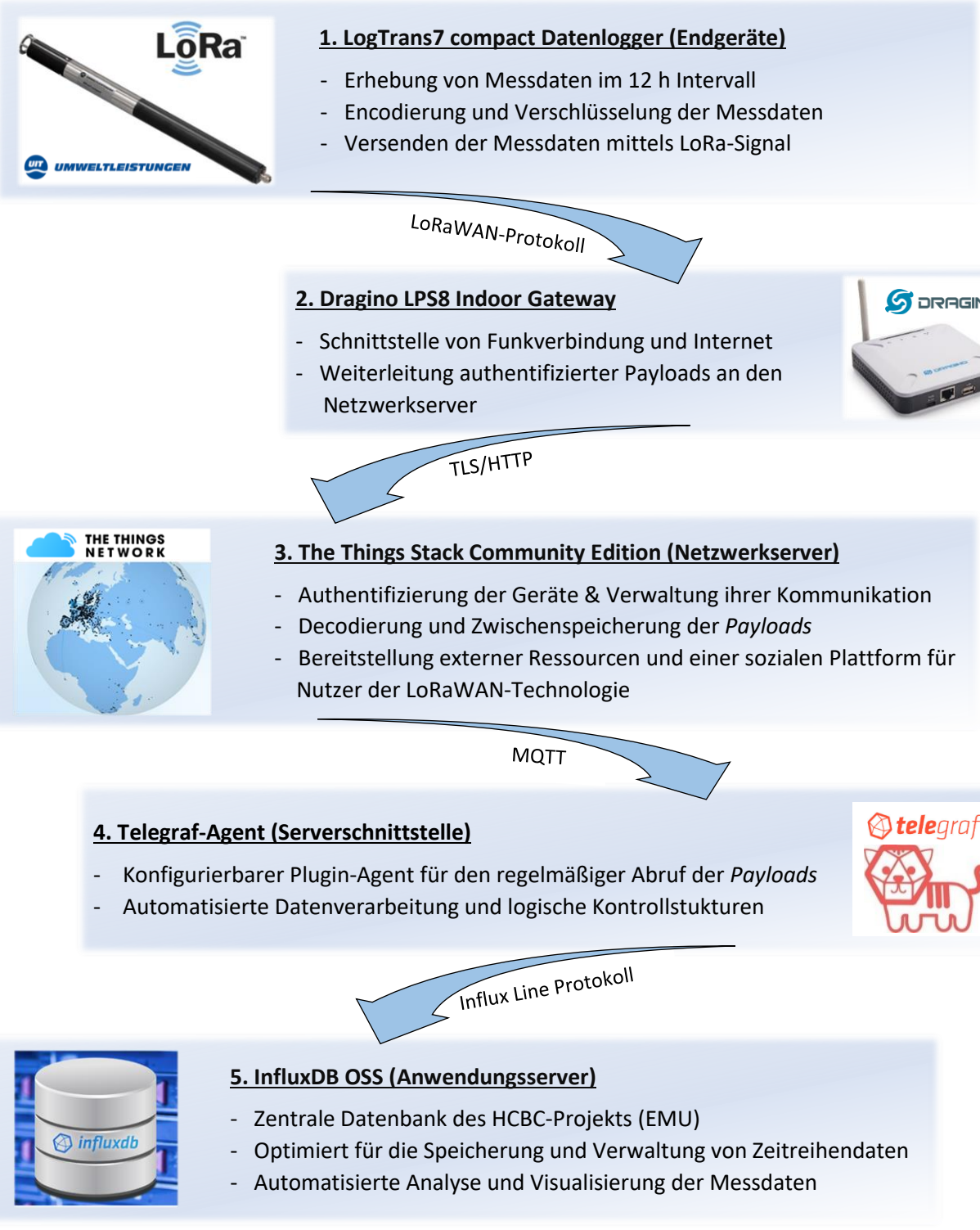

Abbildung 5: Schema der Netzwerkstruktur am HCBC

4. Dokumentation der Netzwerkeinrichtung

4.1 Endgeräte

Bei den im LoRaWAN-Netzwerk eingesetzten Endgeräten handelt es sich um drei Datenlogger vom Modell LogTrans7-IoT-compact von UIT (Umwelt- und Ingenieurtechnik GmbH Dresden). Die Tauchsonden der Logger sind mit Sensoren zur Messung der Temperatur und des Umgebungsdrucks bzw. des hydrostatischen Druckspiegels ausgestattet. Der Energieverbrauch bei laufendem Betrieb beträgt mit dem gewählten Sendeintervall ca. 8 Wh/a, was einem Batteriewechsel entspricht. Die Konfiguration der Logger wurde mit SENSOlog ausgeführt, einer von UIT mitgelieferten Software, die als Schnittstelle für die Interaktion mit ihren Produkten dient. Für die Konfiguration der Datenlogger wurden prinzipiell nicht viele Metadaten benötigt, da diese über ihre einzigartige DevEUI im LoRaWAN-Netzwerk identifiziert werden, und sonstige Metadaten vom Netzwerkserver und von Telegraf hinzugefügt werden (s. Kap. 4.4 und 4.5). Dies entspricht einer klassischen Server-Client Beziehung, was gerade in Hinblick auf den Batteriebetrieb der Logger entscheidend ist. Notwendige Angaben zu den Standortinformationen betrafen die geodätische Messpunkthöhe bzw. Rohroberkante und Einhängtiefe der Tauchsonde, da diese Werte zur Berechnung des Druckspiegels im Softwarekanal der Datenlogger herangezogen werden. Außerdem musste die interne Uhrzeit der Logger eingestellt und vor Abschluss der Konfiguration überprüft werden, da es sonst zu einem abweichenden Sendezeitpunkt und ggfs. zu Datenverlust kommen konnte.

Um die LoRaWAN-Verbindung der LogTrans7 zu konfigurieren, mussten die statischen Schlüssel für die Authentifizierung bei der Konfiguration der Geräte generiert bzw. festgelegt werden (AppEUI und AppKey). Da die Datenlogger im Netzwerkserver zu einer *Application* zusammengefasst werden (s. Kap. 4.3.2), müssen sie eine identische AppEUI besitzen. Die statischen Keys müssen bei der Konfiguration kopiert und an einem sicheren Ort verwahrt werden, da sie zum Registrieren der Geräte im Netzwerkserver benötigt werden. Die Aktivierung der LogTrans7 kann nur mittels OTAA-Methode erfolgen, ein Betrieb im ABP-Modus wird von den Geräten nicht unterstützt. Bei der Konfiguration der Funkverbindung musste zudem das Mess- bzw. Sendeintervall und die mittels LoRa-Signal übertragenen Daten eingestellt werden. Für die Datenübertragung wurden die Messdaten der internen Sensoren und externen Tauchsonden ausgewählt, d.h. die Gerätetemperatur als Prüfwert und die Versorgungsspannung zur Ermittlung des Zeitpunkts für einen Batteriewechsel sowie Grundwassertemperatur und Druckspiegel als eigentliche Monitoringparameter. Da das Datenvolumen einer einzelnen *Payload* nicht ausreicht, um alle Messwerte zu enthalten, werden diese in zwei aufeinander folgenden *Payloads* versendet. Die *Payloads* werden durch eine zusätzliche Integer-Variable unterschieden. Nach vorrübergehender Inaktivität z.B. wegen Wartung der Geräte, muss bei Reaktivierung der LogTrans7 das LoRa-Modul und die Session-Keys zurückgesetzt werden. Das LoRa-Modul wird ausschließlich über die Batterien mit Strom versorgt, auch wenn ein Logger per USB-Kabel angeschlossen ist. Die Funkverbindung kann daher nicht ohne eine ausreichende Batteriespannung konfiguriert werden.

4.2 Gateway

4.2.1 Einrichtung der WAN-Verbindung

Für das LoRaWAN-Netzwerk von HCBC wurde ein Indoor-Gateway Modell LPS8 von der Firma Dragino erworben. Dragino ist ein chinesischer Hersteller für LoRaWAN-fähige Systemkomponenten und IoT-Software, dessen Produkte im Vergleich günstig sind und weite Verbreitung finden. Die Software des LPS8 basiert auf dem Linux-Kernel, welcher durch die gerätespezifische Firmware von Dragino ergänzt wird. Die Software des LPS8 lässt sich bei entsprechenden Kenntnissen mit eigenem Code für Individuallösungen erweitern. Das Gateway kann via LAN oder WLAN mit dem Internet verbunden werden. Die originale LoRa-Antenne kann bei Bedarf ausgetauscht werden. Der Stromanschluss ist mit handelsüblichen Adaptern bis maximal 5 V/1.3 A kompatibel. Die Konfiguration des *LPS8* wird in Web-GUI (***G**raphical **U**ser **I**nterface*) der Firmware ausgeführt, wobei der Zugriff auf die GUI über drei mögliche Wege erfolgen kann:

a) Zugriff via WIFI-Verbindung, bei der das Gateway einen eigenen WIFI-Access-Point nutzt
b) Zugriff via LAN-Verbindung, nachdem das Gateway über ein Ethernet-Kabel mit einem lokalen Netzwerk verbunden wurde
c) Zugriff via Fall-Back-IP, nachdem das Gateway über ein Ethernet-Kabel mit einem PC verbunden wurde

Der Zugriff auf die GUI des Gateways musste via LAN erfolgen, da die Sicherheitsregeln der TUB den Betrieb eines eigenen WLAN-Access-Points verbieten. Bei einer LAN-Verbindung kann über den Webbrowser eines PCs im selben lokalen Netzwerk durch Eingabe der IP-Adresse des Gateways mit dem vom LPS8 genutzten Port auf die GUI zugegriffen werden. Es sei darauf hingewiesen, dass die ältere HTTP- und nicht die HTTPS-Verschlüsselung verwendet wird und das Gateway in seiner Grundkonfiguration nicht den Richtlinien bezüglich der Datensicherheit der TUB entsprach. Bei der Verbindung des LPS8 zu Testzwecken in einem Heimnetzwerk war der Zugriff erst erfolgreich, nachdem in den Einstellungen der Firewall eine Ausnahme deklariert wurde. Ein Zugriff auf das LPS8 innerhalb des lokalen Netzwerks der TUB war daher zunächst nicht möglich, sondern nur via Fall-Back-IP. Ein erfolgreicher Zugriff konnte erst nach weiteren Maßnahmen für die Integration in das lokale Netzwerk der TUB durchgeführt werden.

a) <u>DHCP-Zuweisung einer IP-Adresse</u>
   Dem Gateway wurde anhand seiner MAC-Adresse (physische Hardware-Adresse) mittels DHCP eine valide IP-Adresse zugeteilt. DHCP steht für **D**ynamic **H**ost **C**onfiguration **P**rotocol und ist ein Standard-Protokoll für lokale Netzwerke, in welchem verbundenen Geräten automatisiert eine dynamische IP-Adresse aus validen Intervallen zugewiesen wird.
b) <u>TLS-Verschlüsselung</u>
   TLS ist ein Akronym für **T**ransport **L**ayer **S**ecurity und Nachfolger der früheren SSL-Verschlüsselung. Bei der TLS-Verschlüsselung sorgen der Austausch von Schlüsseln sowie Authentifizierungszertifikate, welche zugelassene Web-Adressen definieren, für eine zusätzliche Sicherheitsschicht bei der Verbindung eines lokalen Netzwerks mit

dem Internet. Nach dem Release eines Updates für die Firmware des LPS8, lässt sich das Gateway statt im ‘LoRaWAN Semtech UDP‘-Modus auch im ‘LoRaWAN Basic Station‘-Modus betreiben. Der neuere Modus ermöglicht eine TLS-Verschlüsselung[4].

c) Spezifische Sicherheitseinstellungen des lokalen Netzwerks
Notwendige Maßnahmen zur erfolgreichen Integration eines Gateways sind letztendlich abhängig von den Sicherheitsanforderungen des jeweiligen lokalen Netzwerks.

4.2.2 Konfiguration des LPS8

Bei der Konfiguration des LPS8 in der Web-GUI wurde aus allgemeinen Gründen der Systemsicherheit zunächst der WLAN-Access-Point ausgeschaltet und ein starkes Gerätepasswort vergeben. In den Systemeinstellungen wurde die lokale Zeitzone geändert, da diese den Zeitstempel im LoRaWAN-Protokoll beeinflusst und eine falsche Zeitzone zu konsekutiven Fehlern bei der Datenverarbeitung führte. Essenziell für den Betrieb eines Gateways ist die Konfiguration des LoRa-Moduls bzw. der Funkverbindung. Es müssen die landesüblichen Frequenzbänder eingestellt werden, welche für LoRaWAN reserviert sind[5]. Die LoRaWAN-Verbindung des LPS8 wird konfiguriert, indem die IP-Adresse des Netzwerkservers und die Gateway-EUI angegeben werden (s. Abbildung 6). Die Gateway EUI ist zur eindeutigen Identifizierung im Netzwerkserver gedacht und erfüllt damit denselben Zweck wie die DevAddr bei den Endgeräten. Sie wird nach der Festlegung verwendet, um das Gateway im Netzwerkserver zu registrieren. Das Gateway benutzt im Standard-Modus das Semtech UDP Protokoll, einer häufig genutzten Version des LoRaWAN-Protokolls. Um der Kommunikation mit dem Netzwerkserver eine zusätzliche TLS-Verschlüsselung hinzuzufügen, musste der aktuellere Basic Station-Modus gewählt werden. Eine Anleitung zur Konfiguration des LPS8 im Basic Station-Modus wird auf der Website von Dragino zur Verfügung gestellt[6].

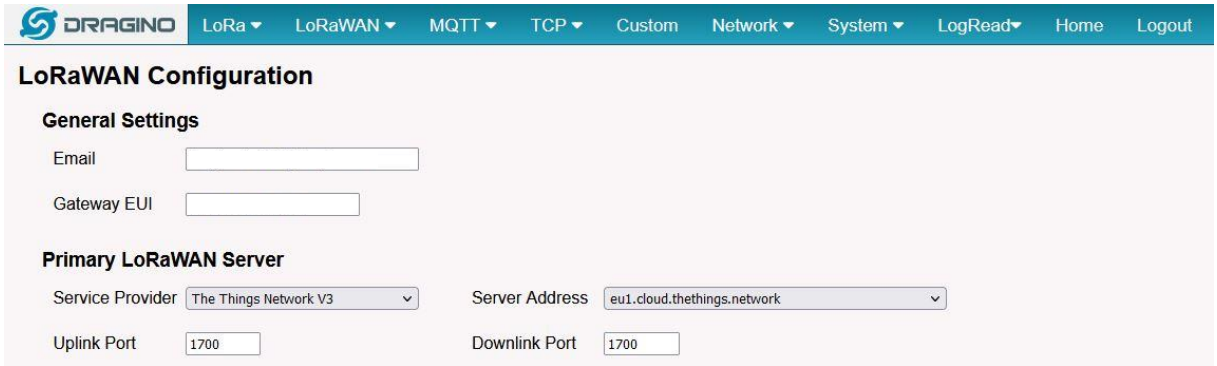

Abbildung 6: Konfiguration der LoRaWAN-Verbindung des Gateways

[4] https://www.thethingsindustries.com/docs/gateways/concepts/lora-basics-station/
[5] https://www.thethingsnetwork.org/docs/lorawan/frequencies-by-country/
[6] http://wiki.dragino.com/xwiki/bin/view/Main/Notes%20for%20TTN/

4.3 Netzwerkserver

4.3.1 The Things Network

Das Projekt The Things Network (TTN) ist nicht nur Anbieter eines kostenlosen Netzwerkservers, sondern auch eng mit den Entwicklern der LoRaWAN-Technologie verflochten. Es wurde von der LoRa Alliance gegründet, einem gemeinnützigen Zusammenschluss von Unternehmen und Forschungseinrichtungen, welche die LoRaWAN- Technologie weiterentwickelt und standardisiert. Zusätzlich verwaltet die LoRa Alliance die einzigartigen Geräte-IDs, welche essenziell für die Datensicherheit von LoRaWAN-Verbindungen sind. Das Projekt wurde von der LoRa Alliance ins Leben gerufen, um die LoRaWAN-Technologie zu verbreiten und Nutzer der Technologie zu einem sozialen Netzwerk zu verbinden. Das soziale Netzwerk ist in sog. Communities, also internationale und regionale Foren z.B. für Berlin organisiert. Das TTN ist für den Austausch von Informationen und gemeinsame Nutzung von Ressourcen (z.B. Gateways) gedacht, stellt den Nutzern aber auch selbst Ressourcen zur Verfügung. Zu den frei verfügbaren Ressourcen gehören Anleitungen, Softwarepakete und Netzwerkserver mit vielfältigen Schnittstellen. Der kostenlose Netzwerkserver von The Things Network nennt sich ‘The Things Stack‘ mit Standorten in Europa, Amerika und Ostasien. In zahlpflichtigen Editionen wird u.a. auch dauerhafter Speicherplatz zur Verfügung gestellt, sodass das TTN gleichzeitig als Anwendungsserver genutzt werden kann.

4.3.2 Integration von Geräten

Ein kostenloser Account bei The Things Network kann auf dessen Website erstellt werden[7]. Von der Startseite eines Accounts gelangt man über den Reiter ‘Console‘ in der Menüleiste zur Verwaltung der Gateways und Endgeräte, wobei letztere im TTN in sog. ‘*Applications*‘ organisiert werden. Die Registrierung von Endgeräten erfolgt im TTN zunächst durch Erstellung einer *Application* mithilfe der AppEUI von einem oder mehreren Geräten. I.d.R. werden mehrere Endgeräte mit denselben Funktionalitäten zu einer *Application* zusammengefasst, z.B. würde ein Ensemble aus drei gleichen Sensoren zur Messung der Luftfeuchte sowie fünf Sensoren von zwei unterschiedlichen Herstellern zur Messung der Raumtemperatur in drei *Applications* gruppiert werden. Bei der Registrierung von Endgeräten in einer *Application* müssen die von ihnen verwendete Version des LoRaWAN-Protokolls und die regionalen Parameter angegeben werden. Diese sind gerätespezifisch, d.h. abhängig von Firmware und LoRa-Chip des jeweiligen Endgeräts. Die Spezifikationen der Endgeräte werden üblicherweise vom Gerätehersteller bereitgestellt oder können auf der Website der LoRa-Alliance in einer Liste mit den Spezifikationen von häufig verbauten LoRa-Chips recherchiert werden[8]. Bei der OTAA-Aktivierung müssen einzelne Endgeräte über Angabe ihrer einzigartigen DevEUI und eines generierbaren Schlüssels (AppKey) registriert werden. Die AppEUI ist für alle Geräte innerhalb einer *Application* dieselbe und wird JoinEUI genannt. Eine alternative Aktivierung mit der ABP-Methode befindet sich in den erweiterten Einstellungen, hierfür müssten zusätzliche Schlüssel erzeugt und den Endgeräten bekannt gemacht werden (s. Abbildung 7). Eine ABP-Aktivierung wird von der Software der LogTrans7 Datenlogger nicht unterstützt. Der Unterschied zwischen OTAA und ABP-Aktivierung ist in Kapitel 2.4 erklärt.

[7] https://www.thethingsnetwork.org/get-started
[8] https://resources.lora-alliance.org/technical-specifications/rp002-1-0-4-regional-parameters

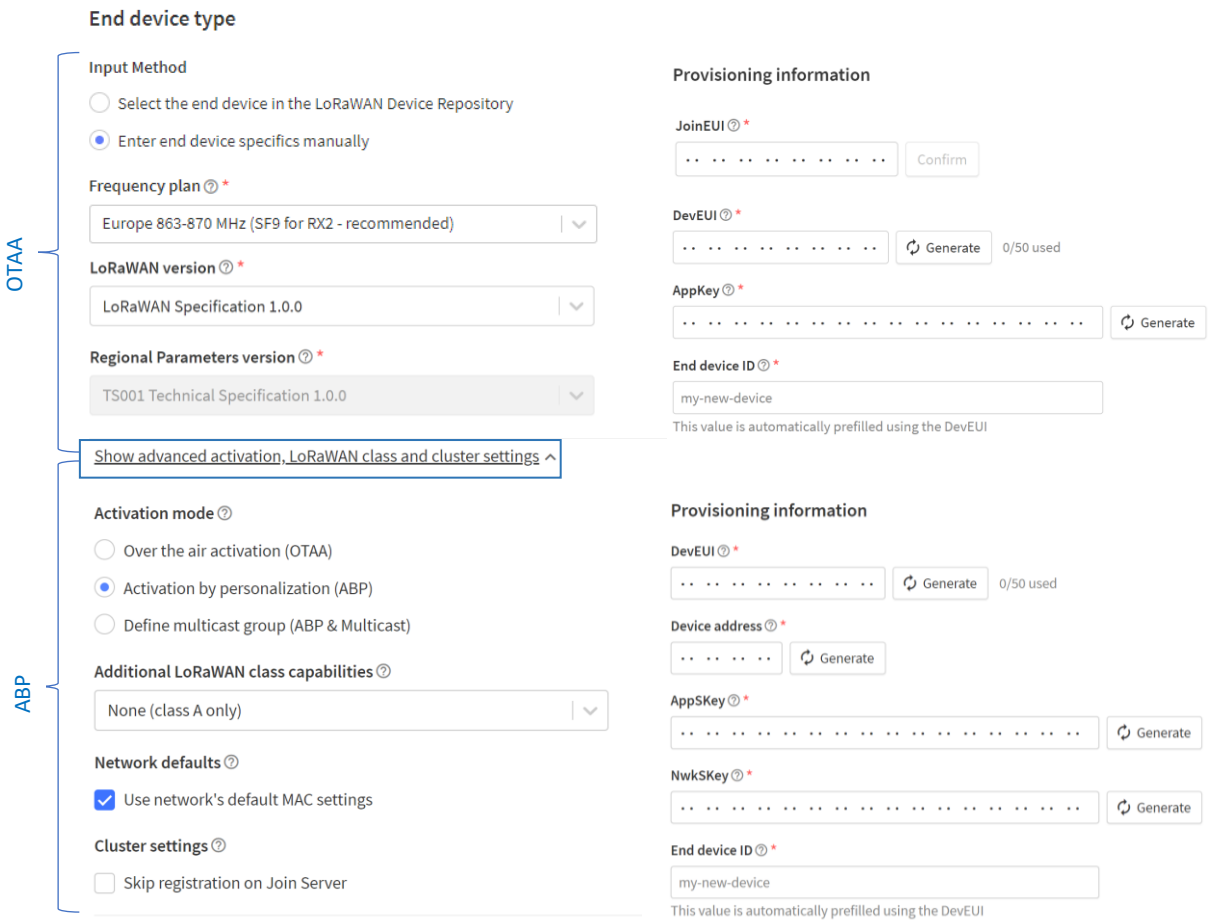

Abbildung 7: Registrierung von Endgeräten im Netzwerkserver The Things Stack

Endgeräte im OTAA-Modus müssen nach der Registrierung noch aktiviert werden, bevor die *Payloads* vom Netzwerkserver empfangen werden können. Die Aktivierung lässt sich als gegenseitige Authentifizierung von Netzwerkserver und Endgeräten beim initialen Aufbau einer LoRaWAN-Verbindung verstehen. Die Aktivierung erfolgt automatisch, wenn bei der Registrierung die korrekten Schlüssel eingegeben wurden, und sich ein Gateway mit funktionstüchtiger Verbindung zum Netzwerkserver in Reichweite des LoRa-Signals befindet. Während des Betriebs der Datenlogger werden vor allem *Uplink-Messages* mit den Messdaten versendet. Allerdings werden bei der Aktivierung sowie bei der Erneuerung der Session-Keys *Up-* und *Downlink Messages* für die gegenseitige Authentifizierung von Netzwerkserver und Endgeräten benötigt (vgl. Abbildung 4). Abgesehen von diesen Situationen sind *Downlink-Messages* eher selten und können eine manuelle Anpassung der Konfiguration der Endgeräte beinhalten, z.B. des Messintervalls. Bei Betrieb von Endgeräten im ABP-Modus werden grundsätzlich nur *Downlink-Messages* benötigt, da der Prozess der gegenseitigen Authentifizierung entfällt. Bei der Registrierung eines Gateways muss die während der Konfiguration festgelegte Gateway-EUI dem TTN bekannt gemacht werden. Falls die Konfiguration des Gateways als zusätzliche Sicherheitsschicht eine TLS-Verschlüsselung enthält, muss bei der Registrierung eine authentifizierte Verbindung gefordert und gültige CUPS- und LNS-Key angegeben werden. Die zusätzlichen Schlüssel werden ähnlich wie im LoRaWAN-Protokoll für eine gegenseitige Authentifizierung nach dem Handshake-Prinzip genutzt. Eine Anleitung für den Betrieb von Endgeräten im LoRaWAN Basic Station-Modus wird auf der Website des TTN bereitgestellt[9].

[9] https://www.thethingsindustries.com/docs/gateways/concepts/lora-basics-station/

4.3.3 Datenverwaltung

Ein über das LoRaWAN-Protokoll versendetes Datenpaket wird als *Payload* bezeichnet. Die *Payloads* sind im TTN nicht unmittelbar abrufbar, sondern erscheinen als sog. *Events* im jeweiligen *Livestream* einer *Application*. Als *Livestream* wird die Cloud bezeichnet, in welcher die Daten zwischengespeichert werden. In einem *Event* ist das Log der Kommunikation des Netzwerkservers mit einem Endgerät enthalten, was sämtliche und größtenteils irrelevante Metadaten inkludiert. Die eigentlichen Messdaten und relevante Metadaten machen nur einen kleinen Abschnitt im Log eines *Events* aus und müssen nach Abruf aus dem *Livestream* gefiltert werden (s. Abbildung 8).

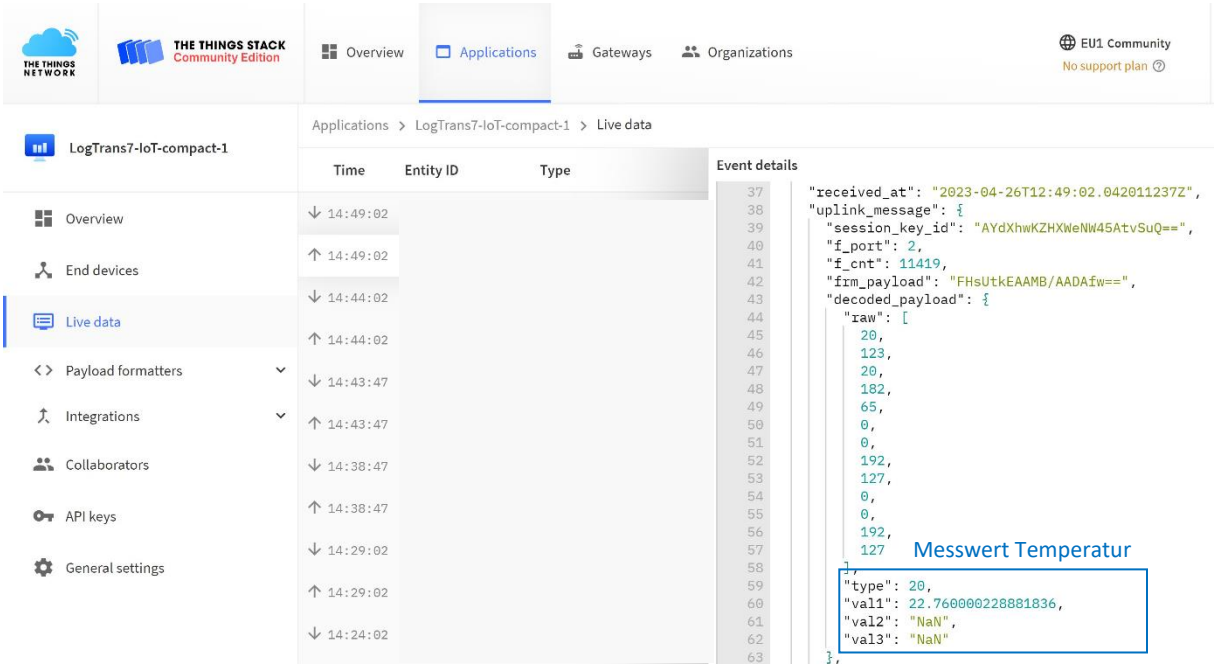

Abbildung 8: Livestream des Netzwerkservers mit Ausschnitt einer empfangenen Payload

Eine *Payload* ist zur Reduzierung des Datenvolumens codiert und nutzt häufig das Binär- oder Hexadezimalsystem für numerische Werte. Die Codierung ist nicht gleichzusetzen mit der Verschlüsselung, welche auf die Datensicherheit abzielt. Die Encodierung der Messdaten erfolgt vor dem Versenden der Payload und ist abhängig von der jeweiligen Firmware des Endgeräts. Eine codierte *Payload* im Rohformat ist für Menschen nicht ohne weiteres verständlich und muss nach dem Datentransfer mittels LoRaWAN-Verbindung wieder decodiert werden. Die Decodierung kann durch Implementierung eines Decoders im Netzwerkserver oder alternativ erst im Anwendungsserver erfolgen. Im TTN kann ein Decoder unter dem Reiter ‘Payload formatters‘ für eine *Application* oder einzelne Endgeräte definiert werden. Der Decoder wird i.d.R. in Javascript verfasst und vom Gerätehersteller zur Verfügung gestellt, da die Codierung in der Firmware enthalten ist.

4.3.4 Integration eines Anwendungsservers

The Things Stack ist in seiner Funktion als Netzwerkserver nicht zur dauerhaften Speicherung der Messdaten gedacht, da die *Events* im *Livestream* einer Retentionszeit unterliegen. In einem kostenlosen Account von The Things Stack werden *Events* erfahrungsgemäß über einen maximalen Zeitraum von 24 h im *Livestream* zwischengespeichert. Die *Events* müssen daher vor

ihrer Löschung abgerufen und an einen Anwendungsserver bzw. Datenbank weitergeleitet werden. Dafür wird in der Menüleiste einer *Application* der Reiter ‘Integrations‘ aufgerufen und die gewünschte Art der Weiterleitung ausgewählt (s. Abbildung 9). Die Weiterleitung kann wie beim LoRaWAN-Netzwerk von HCBC über das MQTT- oder über das als ‘Webhooks‘ bezeichnete HTTP-Protokoll erfolgen. Alternativ kann die Funktion ‘Storage Integration‘ aktiviert werden, bei der ein sog. Packet Forwarder, der sowohl auf den Endgeräten als auch auf dem Anwendungsserver installiert wird, die *Payloads* automatisiert weiterleiten kann, was sich bei Geräten im ABP-Modus anbietet. AWS IoT, Azure IoT und LoRa Cloud sind externe Anbieter für Netzwerkkomponenten von IoT-Anwendungen (u.a. Tochterunternehmen von Amazon und Microsoft), die jeweils eigene Methoden für die Integration ihrer Produkte bereitstellen. The Things Stack verfügt über einen eigenen *Broker* (MQTT-Server), für den die Zugriffsberechtigung zum Abruf der *Events* eingerichtet werden muss. Die Autorisierung der Datenabfrage erfolgt über eine Username-Password Kombination. Die Webadresse des *Brokers* mit den für MQTT reservierten Ports ist voreingestellt. Der Username orientiert sich am Namen der *Application* und als Passwort wird ein API-Key erstellt werden. API-Keys sind nur einmalig sichtbar. Sie sollten nach der Generierung kopiert und an einem sicheren Ort gespeichert werden (ähnlich wie bei *Influx-Token*, vgl. Kap. 4.4.4).

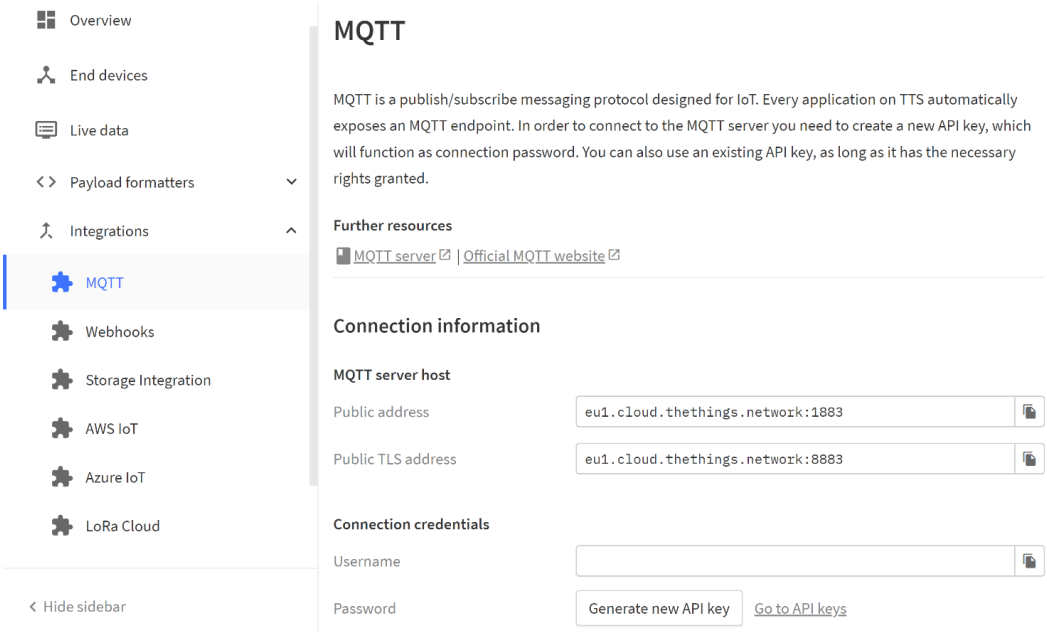

Abbildung 9: Integration eines Anwendungsservers mittels MQTT-Protokoll

4.3.5 MQTT-Protokoll

MQTT ist ein Netzwerkprotokoll, das speziell für IoT-Anwendungen konzipiert ist und in diesem Bereich eine breite Verwendung findet. Das MQTT-Protokoll wird im LoRaWAN-Netzwerk am HCBC genutzt, um die Messdaten vom Netzwerkserver an den Anwendungsserver weiterzuleiten. MQTT stand ursprünglich für ‘**M**essage **Q**ueue **T**elemetry **T**ransport’ und wie im Namen ausgedrückt wird, erfolgt die Kommunikation über den Austausch von kleineren Datenpaketen, die Messages genannt werden (nicht zu verwechseln mit *Up*- und *Downlink-Messages* im LoRaWAN-Protokoll). In einem Netzwerk, welches das MQTT-Protokoll verwendet, trägt ein Server die Bezeichnung *Broker*, und die mit dem *Broker* kommunizierenden

Geräte oder Dienste die Bezeichnung *Clients*. Das MQTT-Protokoll kennt verschiedene Typen von Messages mit spezifischem Einsatzzweck und Datenstruktur. Die meisten Typen sind allerdings nur für die internen Abläufe des Protokolls von Bedeutung. Für den Datenverkehr zwischen Netzwerk- und Anwendungsserver sind zwei Typen entscheidend, sie stellen das Pendant zu *Up*- und *Downlink-Messages* im LoRaWAN-Protokoll dar. Beim *Publishing* (dt. Veröffentlichen) werden Daten von einem *Client* an den *Broker* gesendet, und beim *Subscribing* (dt. Abonnieren) sendet der *Broker* Daten an einen *Client*. In Bezug auf das LoRaWAN-Netzwerk am HCBC fungiert das TTN als *Broker*, da es das MQTT-Protokoll für den Datenabruf verwendet. Der *Client* ist in diesem Sinne Telegraf, ein konfigurierbarer Plugin-Agent, welcher die Messdaten aus dem TTN abruft und dem Anwendungsserver übergibt.

Der Datenverkehr wird im MQTT-Protokoll mithilfe von sog. *Topics* (dt. Themen) strukturiert, welche als Kommunikationskanäle aufgefasst werden können, und für die Zuordnung der Daten notwendig sind (ähnlich wie *Tags* in InfluxDB, vgl. Kap. 4.4.3). *Topics* sind hierarchisch gegliedert und können eine beliebige Anzahl an Sub-*Topics* besitzen, welche über eine Slash-Notation (/) den Basistopics angehängt werden. Dadurch ergeben sich mitunter lange Zeichenketten, welche gleichzeitig als Authentifizierung der *Clients* durch den *Broker* bei Kommunikationsanfragen dienen. Die Benennungsschema der *Topics* kann individuell durch die Netzwerkadministratoren wie im Monitoring-Konzept 2019 von HCBC festgelegt werden[10]. Die korrekte Benennung der *Topics* ist für die Kommunikation mit dem EMU-Server obligatorisch, da sie den zugedachten Speicherort definieren und den Datenverkehr autorisieren. Ein weiteres wichtiges Konzept des MQTT-Protokolls ist die QoS-Policy (***Q**uality **of S**ervice*), welche die Verbindungsstabilität und Energieverbrauch beeinflusst. Das Konzept wird bei Datentransfer mittels verzögerten Austauschs von Datenpaketen ohne dauerhafte Verbindung der Kommunikationsteilnehmer verwendet, wie im LoRaWAN-Protokoll oder MQTT-Protokoll. Es kann eine von drei Einstellungen für die QoS gewählt werden (s. Tabelle 2), ein hoher Wert steigert die Sicherheit, mit welcher ein Datenpaket den Empfänger erreicht, verbraucht allerdings mehr Bandbreite und senkt damit die Energieeffizienz. Im Monitoringkonzept von HCBC wird eine QoS von mindestens 1 (At least once) gefordert. Der MQTT-*Broker* von The Things Stack unterstützt in der kostenlosen Version allerdings nur eine QoS von 0 (At most once), weshalb diese für die Konfiguration von Telegraf genutzt wurde (s. Kap. 4.5.4).

Tabelle 2: Übersicht von QoS-Policys

QoS	**0** (At most once)	**1** (At least once)	**2** (Exactly once)
Policy	Die Nachricht wird höchstens einmal zugestellt. Es wird keine Bestätigung gesendet und die Nachricht wird nicht zwischengespeichert.	Die Nachricht wird mindestens einmal zugestellt. Vom Empfänger wird eine Bestätigung gesendet und die Nachricht wird zwischengespeichert, bis sie erfolgreich zugestellt wurde.	Die Nachricht wird genau einmal zugestellt. Es wird Sequenz aus vier Nachrichten verwendet, um sicherzustellen, dass diese ohne Duplikate zugestellt wird.

[10] https://blogs.tu-berlin.de/hri_hcbc/2020/12/10/konzeption-und-erste-erprobung-des-monitoring-netzwerk/

4.4 Anwendungsserver

4.4.1 Zeitreihendaten

Die Messdaten des geothermischen Monitorings fallen unter die Kategorie der Zeitreihendaten. Im Monitoring-Konzept 2019 wurde entschieden, eine gemeinsame Datenbank für die im Rahmen des HCBC-Projekts erhobenen Messdaten zu nutzen, um im Gegensatz zum bisherigen Betrieb von Parallelstrukturen den technischen und administrativen Aufwand zu verringern, sowie Datenschutz und Systemsicherheit zu erhöhen. Zu diesem Zweck wurde mit InfluxDB eine Datenbank ausgewählt, deren Software speziell auf Zeitreihendaten zugeschnitten ist. Die Grundlage der Speicherstruktur von InfluxDB bilden die charakteristischen Eigenschaften von Zeitreihendaten, welche in Tabelle 3 zusammengefasst sind.

Tabelle 3: Charakteristische Eigenschaften von Zeitreihendaten

Zeitliche Abhängigkeit	Jeder Messwert wird einem bestimmten Zeitpunkt zugeordnet. Die Zuordnung erfolgt über einen *Timestamp* (dt. Zeitstempel), welcher das gregorianische Jahr, Tag, Stunden, usw. der angestrebten Präzision bis hin zu Nanosekunden enthält. Da für *Timestamps* eine universelle Gültigkeit angestrebt wird, besitzen diese i.d.R. das Unix-Format. Das Unix-Format bezieht sich auf die UTC-Zeit und ist für Rechner ausgelegt. Es drückt einen Zeitpunkt durch einen numerischen Wert aus, der den vergangenen Sekunden seit dem 1. Januar 1970 um 0.00 Uhr entspricht, was es erlaubt Zeitdifferenzen zu Berechnen.
Zeitlich-logische Abfolge	Zeitreihendaten beziehen sich häufig auf Messreihen, welche durch ihre zeitlich-logische Abfolge miteinander verknüpft sind. Dadurch können sie zu Serien zusammengefasst werden und sekundäre Informationen (Trends, Muster, Korrelationen) abgeleitet werden.
Noise	Entsprechen Zeitreihendaten den Messdaten von Umweltphänomenen, enthalten sie wegen natürlicher Fluktuationen und Messungenauigkeiten üblicherweise einen signifikanten Anteil Noise.
Hoher Speicherbedarf	Zeitreihendaten besitzen die Tendenz große Datensätze zu erzeugen. Wenn z.B. ein Netzwerk aus 20 Sensoren besteht, die jeweils im Intervall von 10 Sekunden 5 Parameter senden, entspricht dies bereits über 300.000 Messwerten pro Jahr. Zeitreihendaten sind meistens aus einer Gleitkommazahlen des Datentyps Float (4 Bytes), einem Zeitstempel (7-13 Bytes) und mindestens einem String (1 Byte pro Zeichen) zusammengesetzt.
Möglichkeit der Aggregation	Zeitreihendaten können über festgelegte Zeitintervalle zusammengefasst werden, was es ermöglicht aus einem hochauflösenden Datensatz einen niedriger auflösenden zu generieren, der aber weniger Noise enthält und weniger Speicherplatz benötigt (*Downsampling*).

4.4.2 InfluxDB

InfluxDB ist eine vom US-amerikanischen Tech-Unternehmen InfluxData entwickelte Datenbanksoftware für die effiziente Speicherung und Verwaltung von Zeitreihendaten. Die Software lässt sich theoretisch ohne jegliche Programmierfähigkeiten nutzen, richtet sich aber hinsichtlich seiner Bedienung an IT-affine Nutzer. InfluxDB wird für Zwecke wie Überwachung, dem Monitoring von Umweltphänomenen und Wirtschaftsentwicklungen sowie Leistungsanalysen von Industrie- und IT-Prozessen eingesetzt, und ist dabei insbesondere für den Einsatz in IoT-

Netzwerken konzipiert. Das Konzept für die effiziente Speicherung und Verwaltung von Zeitreihendaten basiert auf zwei Eigenentwicklungen von InfluxData; dem *Influx-Datenformat* und der Skriptsprache '*Influx Line Protokoll*'. Abgesehen vom geringeren Speicherbedarf, ist die Speicherstruktur von InfluxDB skalierbar, d.h. die Größe der Datenbank kann dynamisch Schrumpfen oder Wachsen, ohne dass die Speicherstruktur angepasst werden muss. Ein weiterer Vorteil von InfluxDB ist die hohe Interoperabilität der Software. Es werden vielfältige Datenquellen und Rohformate akzeptiert und die Software besitzt mehrere Schnittstellen für die Bedienung und das Einladen von Daten. InfluxDB ermöglicht Echtzeitanalysen mit einer Präzision bis hin zu Nanosekunden, darüber hinaus können Datenverarbeitungsprozesse weitestgehend automatisiert werden, z.B. bei der Datenaggregation, dem Design von Grafiken und dem Auslösen von externen Prozessen, die mit InfluxDB gekoppelt wurden.

InfluxDB ist den drei folgend beschriebenen Versionen erhältlich, abhängig davon, ob die kostenlose oder eine zahlpflichtige Variante gewählt wird, und ob der Server von InfluxData Cloud genutzt oder die Software auf einem eigenen Server installiert wird. Für die Einrichtung des LoRaWAN-Netzwerks wurde für Testzwecke zunächst ein kostenloser Account bei InfluxDB Cloud erstellt[11], auf dem EMU-Server wird die zahlpflichtige Version InfluxDB OSS betrieben.

a) <u>InfluxDB Cloud (kostenlos):</u>
   Die kostenlose Variante von InfluxDB Cloud ist eher als Demoversion zu verstehen. Wichtig ist allerdings, dass die Speicherdauer von Daten in den sog. *Buckets* (s. Kap.4.4.4) auf 30 Tage limitiert ist, weshalb diese Version außer für Testzwecke oder zum Verarbeiten von kurzen Messreihen als Datenbank unbrauchbar ist.

b) <u>InfluxDB Cloud (zahlpflichtig):</u>
   Die zahlpflichtige Variante von InfluxDB Cloud besitzt alle Funktionalitäten und ist geeignet, wenn kein eigener Server zur Verfügung steht oder Zeit bei der Einrichtung gespart werden soll.

c) <u>InfluxDB OSS:</u>
   InfluxDB Enterprice kann als Software mit allen Funktionalitäten auf einem eigenen Server installiert werden und bietet die Möglichkeit der Anpassung an die lokale Netzwerkstruktur.

4.4.3 Speicherstruktur

Im Gegensatz zu SQL-basierten Datenbanken, welche eine Speicherstruktur in Form von Tabellen mit Zeilen und Spalten aufweisen (und daher universell für verschiedene Datenkategorien geeignet sind), werden Datentupel in InfluxDB als 'Punkte' aufgefasst, welche entlang der Zeitachse aufgereiht sind und über diese zu Serien zusammengefasst werden. Damit sind sie nahezu losgelöst von übergeordneten Speicherstrukturen, was es erlaubt, sie anders als üblich zu organisieren. Diese Datentupel werden als *Metric* (dt. Messwert) bezeichnet. Der interne Aufbau der *Metrics* entspricht dem sog. *Influx-Datenformat*, welches in Verbindung mit der Datenbanksoftware InfluxDB entwickelt wurde. *Influx* ist jenes Datenformat, in welches die Messdaten der LogTrans7 gebracht werden mussten. Eine *Metric* ist ein Container, der einen oder mehrere Messwerte, die zum selben Zeitpunkt aufgenommen werden und darüber hinaus Metadaten für die Zuordnung der Messwerte beinhaltet. Eine *Metric* bzw. die Datenstruktur des

[11] https://cloud2.influxdata.com/signup

Influx-Datenformat besteht aus vier Elementen, in denen die Daten gespeichert werden; *Measurement, Tags, Field* und *Timestamp*. Die Funktionen der Elemente sind im Folgenden erläutert.

a) *Measurement*

Measurement ist ein frei wählbarer String mit dem Namen der *Metric* und ist gemeinsam mit dem *Timestamp* ihr einziger übergeordnete Indikator. Die Kombination der beiden Elemente ist singulär, d.h. *Metrics* mit demselben *Measurement* und *Timestamp* werden überschrieben. Für die Wahl des Strings für *Measurement* sollte angestrebt werden, *Metrics* mit gemeinsamen *Tags* und *Fields* (s.u.) möglichst gleich zu benennen. Dies hängt unmittelbar mit der Speicherstruktur von InfluxDB zusammen; wenn beispielsweise alle gespeicherten *Metrics* einen verschiedenen String für *Measurement* besäßen, würde dies die *Cardinality* (bzw. Speicherbedarf, s.u.) exorbitant erhöhen. In der Praxis werden *Measurements* meistens in Sinne eines Hashtags für Datenabfragen genutzt.

b) *Tags*

Ein *Tag* (dt. Etikett, Kennzeichnung) ist ein Schlüssel-Wert-Paar, wobei der Wert ebenfalls einen String enthalten muss. *Tags* dienen dazu, *Metrics* zu kategorisieren. I.d.R. werden die Metadaten einer Messung (z.B. Standort und Messgerät) als *Tags* gespeichert. Darüber hinaus bieten sie eine optionale Organisationsstruktur, die nach einem individuellen Schema benannt werden kann. Eine *Metric* muss mindestens einen *Tag* beinhalten, auf der anderen Seite führt das Überladen von *Metrics* mit unzähligen *Tags* zu einer hohen *Cardinality*. Das Filtern von *Metrics* über ihre *Tags*, ist die Methode von InfluxDB, um schnelle und flexible Datenabfragen durchführen zu können. Das Schema der Benennung von *Tags* für im Rahmen des HCBC-Projekts erhobene Messdaten wurde im Monitoring-Konzept 2019 festgelegt.

c) *Fields*

Die *Fields* einer *Metric* enthalten die eigentlichen Messwerte. Sie bestehen aus einer Liste von Schlüssel-Wert-Paaren, wobei der Wert meistens einen numerischen Datentyp (Bool. Integer, Float) enthält. Strings können ebenfalls als *Field* gespeichert werden, wobei es sinnvoller ist, diese als *Tag* zu deklarieren. Am Beispiel der LogTrans7 entsprechen die Parameter ‘battery_voltage‘, ‘device_temperature‘, ‘hydraulic_head‘ und ‘gw_temperature‘ den *Fields* (vgl. Abbildung 10). Die zugehörigen Schlüssel der *Fields* sind ebenfalls dazu gedacht, Messdaten zuzuordnen und werden für Datenabfragen genutzt.

d) *Timestamp*

Jede *Metric* besitzt einen *Timestamp* im Unix-Format mit zuvor definierter Präzision, welche den Zeitpunkt der Messung wiedergibt. Der *Timestamp* ist die zentrale übergeordnete Speicherstruktur, über welche *Metrics* bei Datenabfragen und Analysen durch Definition von Zeiträumen zu Zeitreihen zusammengefasst werden.

Das *Influx-Datenformat* besitzt eine spezielle Syntax, um die vier Elemente voneinander abzugrenzen, wie im Beispiel in Abbildung 10 gezeigt. Die festgelegten Strings (*Measurement* sowie Schlüssel und Werte für *Tags* und *Fields*) dürfen keine Leerzeichen enthalten, solche müssen ggfs. durch einen Unterstrich ersetzt werden.

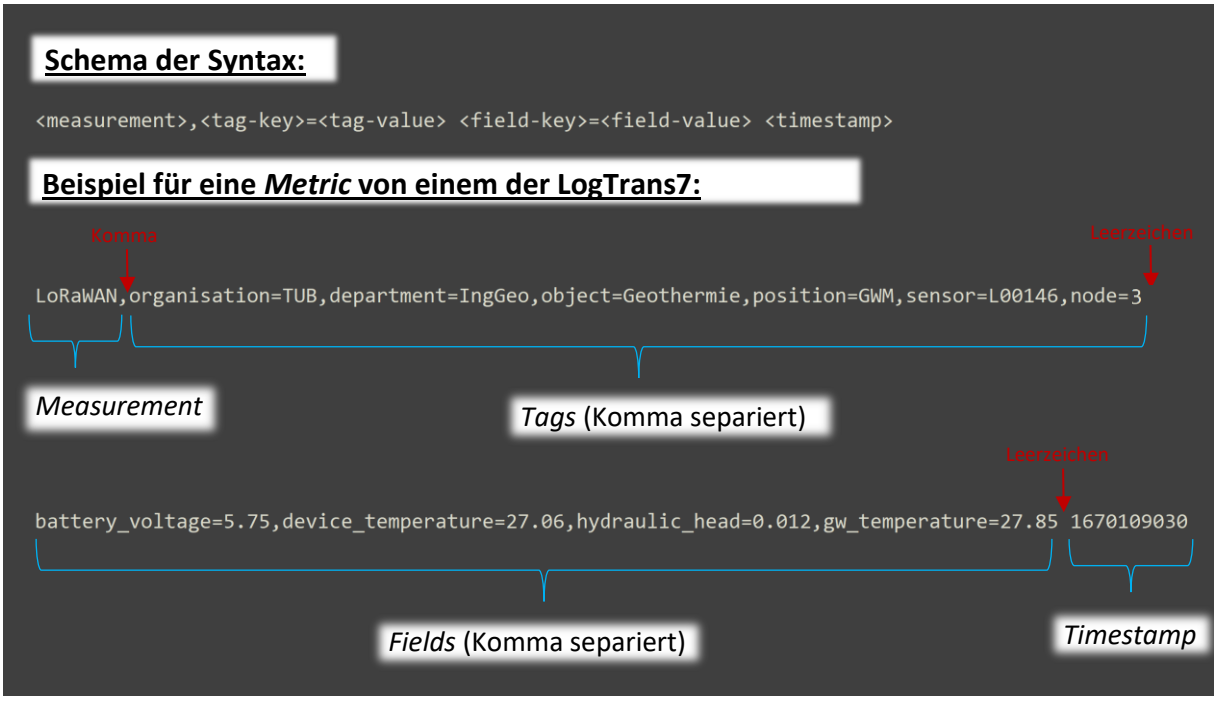

Abbildung 10: Syntax des Influx-Datenformats

Abgesehen von den *Metrics* existiert innerhalb von InfluxDB mit den sog. *Buckets* eine weitere übergeordnete Speicherstruktur. Sie entsprechen am ehesten dem Pendant zu Ordnern in einer SQL-basierten Datenbank. *Buckets* besitzen den primären Zweck, Benutzerrollen und Berechtigung zu verwalten. Sekundär legen sie die Speicherdauer von *Metrics* über die Einstellung der Retentionszeit fest. *Metrics* können nicht gelöscht werden, wenn für den *Bucket* eine unbegrenzte Retentionszeit eingestellt wurde, außer es wird der gesamte *Bucket* vernichtet. Innerhalb der *Buckets* können *Metrics* höchstens von neuen *Metrics* mit demselben *Timestamp* und Namen für *Measurement* überschrieben werden. Eine *Retention-Policy* von 30 Tagen kann z.B. für Testzwecke oder Datenaggregation genutzt werden. Zugriffsberechtigungen werden in InfluxDB über *Influx-Token* realisiert. Ein *Token* ist ein 128-Byte String, welcher als eine Art Passwort zuvor definierte Zugriffsrechte gewährt

Es existieren mehrere Arten von *Token*, welche u.a. über die GUI von InfluxDB erstellt werden können. Sie sind bei der Generierung nur einmal sichtbar und sollten unmittelbar danach kopiert und an einem sicheren Speicherort verwahrt werden. *Token* können für *Member* d.h. Benutzer, *Organizations* d.h. Institutionen oder generell Benutzergruppen, sowie unabhängig von der Benutzerrolle für *Buckets* gelten. Daneben gibt es die sog. API-*Token*, auch All-Access-*Token* genannt. Sie erlauben es auf die API von InfluxDB (programmierbare Schnittstelle für die Kommunikation mit der Datenbank) zuzugreifen. Abhängig von den Berechtigungen des *Members* oder der *Organization*, die den API-*Token* generiert hat, können neue *Buckets* erstellt, existierende *Buckets* gelöscht, und alle enthaltenen Daten abgerufen werden. Abbildung 11 zeigt eine Übersicht der Topologie von Berechtigungen und *Token* in InfluxDB:

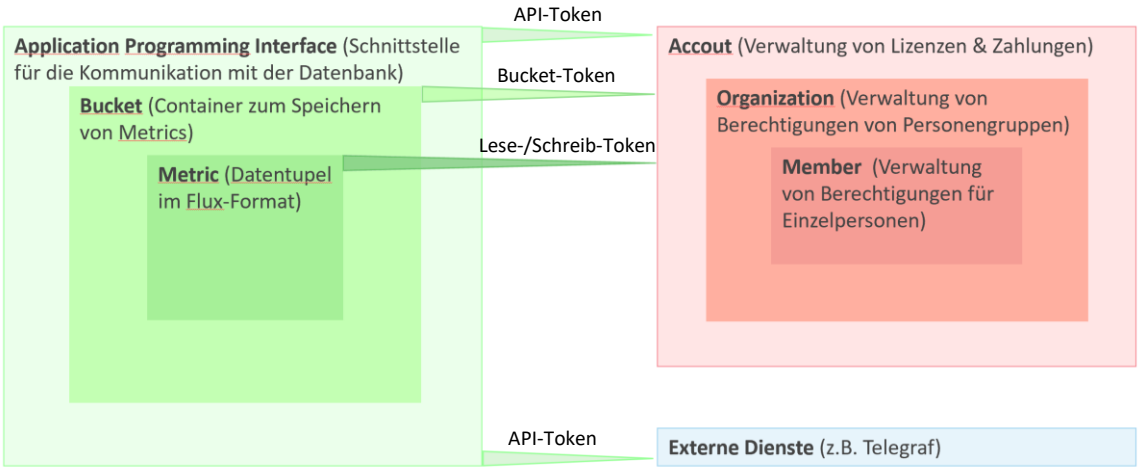

Abbildung 11: Hierarchie der Zugriffsrechte über Influx-Token

4.4.4 Uploads & Datenabfragen

Datenabfragen werden in der GUI von InfluxDB unter dem gleichnamigen Reiter getätigt. Für eine Datenabfrage müssen der *Bucket* (Speicherort und Zugriffsberechtigung), ein Zeitintervall (Serie) und die Bezeichnung für *Measurement* der Datenpunkte (eindeutige Identifikation von *Metrics*) definiert werden. Durch Auswahl von *Tags* und *Fields* können Daten gefiltert werden. Darüber hinaus können weitere Filter und Analysen implementiert werden, allerdings sind die Möglichkeiten für die Interaktion innerhalb der GUI limitiert. Komplexe Datenabfragen können über das *Influx Line Protocol* realisiert werden, der internen Datenbanksprache von InfluxDB. *Influx Line Protocol* ist eine Skriptsprache, die sich an SQL orientiert aber auf die Speicherstruktur von InfluxDB zugeschnitten ist. Die abgefragten Daten können in Form von Grafik und Tabelle ausgegeben werden, wie in Abbildung 12 zu sehen ist. Die Bedienung von InfluxDB ist mitunter nicht intuitiv, weshalb InfluxData kostenlose Onlinekurse zur Verfügung stellt[12], welche in Teilen auch auf Youtube zu finden sind[13]. Daneben gibt es eine schriftliche Dokumentation auf der Website von InfluxData.

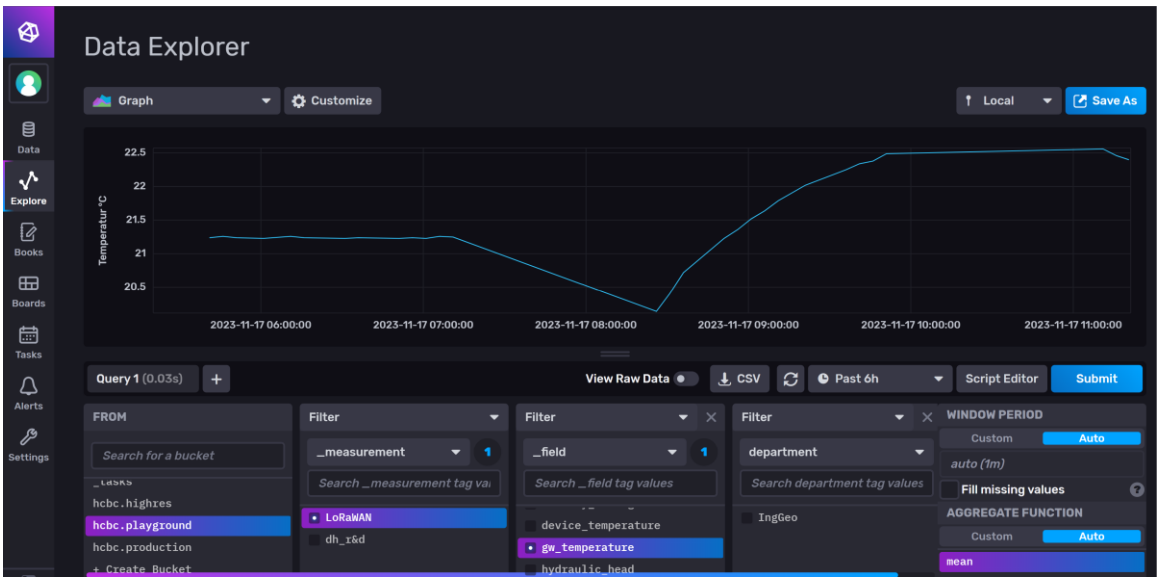

Abbildung 12: Datenabfrage mit Zeitreihe einer Testmessung in InfluxDB

[12] https://university.influxdata.com
[13] https://www.youtube.com/@influxdata8893

Um neue Daten in InfluxDB einzuladen, existieren vier mögliche Methoden, die im Folgenden beschrieben werden.

a) Upload in der GUI

Innerhalb der GUI von InfluxDB können Daten manuell unter dem als Speicherort ausgewählten *Bucket* eingeladen werden. Die Daten müssen sich für einen erfolgreichen Upload bereits im *Influx-Dateiformat* befinden oder eine CSV-notierte Textdatei sein, welche die Daten in der Struktur des *Influx-Datenformats* enthält.

b) Upload über die CLI

InfluxDB bietet eine von InfluxData entwickelte CLI (**C**omand **L**ine **I**nterface/Kommandozeile) als Download an, über welche direkt mit der Datenbank kommuniziert werden kann. Die Anwendung setzt allerdings Kenntnisse der systeminternen Datenbanksprache *Influx Line Protocol* voraus.

c) Upload über einen Client

Es existiert für diverse Skriptsprachen (wie Java, R, Python, usw.) eine frei verfügbare Bibliothek zur Nutzung von InfluxDB. Mit der Bibliothek lässt sich ein *Client* konfigurieren, der mit der Datenbank kommuniziert und über einen *Token* mit entsprechenden Zugriffsrechten Daten abfragen oder schreiben kann. Abbildung 13 zeigt als Beispiel die Konfiguration eines *Clients* mit Python, welcher Messdaten des Monitorings aus einer Exceldatei liest, mit *Tags* versieht und in InfluxDB einlädt. Falls die Daten für den Upload aus dem Internet (z.B. dem TTN) stammen, müsste zusätzlich ein anderer *Client* für den Datenabruf konfiguriert werden, oder als Alternative Telegraf für den Upload genutzt werden.

d) Upload mit Telegraf

Der *Plugin-Agent* Telegraf wurde als Serverschnittstelle konzipiert und ist eine vielseitige und von den InfluxDB-Entwicklern bevorzugte Methode für den Upload von Daten. Mit Telegraf können u.a. Daten automatisiert aus dem Internet abgefragt und an InfluxDB übermittelt werden. Telegraf sollte zwar die sicherste Methode für den Upload darstellen, erfordert aber einen gewissen Konfigurationsaufwand.

```python
# -*- coding: utf-8 -*-

import pandas as pd
import influxdb_client as influx
from influxdb_client.client.write_api import SYNCHRONOUS

# Definition of authentification data to conect with InfluxDB
url = "https://eu-central-1-1.aws.cloud2.influxdata.com"
org = "TUB"
bucket="Test Bucket"
token = 'hmhu6hLMgZX_ifjtD91KjMwWqzva0AuWc2xIP4P4eFwsOsyrU55FNYJWKBGtL2ZwAx7Bfp580ZrdLzLsnoGrbA==''


# Initialization of a client to interact with the API of InfluxDB
client = influx.InfluxDBClient(url=url, token=token, org=org)
write_api = client.write_api(write_options=SYNCHRONOUS)

# Load the data with measurements for hydraulic head, temperature and electric conductivity
# from an excel file into seperate dataframes.
# Respectively the first column with the measurement dates is used as index column
gw = pd.read_excel('Messrunden Zusammenfassung.xlsx', sheet_name='Messstellen GW',
                   header=2, index_col=0, usecols="B:M")
t = pd.read_excel('Messrunden Zusammenfassung.xlsx', skiprows=lambda x: x in [0, 3],
                  sheet_name='Messstellen T°', index_col=0, usecols="B:M")
lf = pd.read_excel('Messrunden Zusammenfassung.xlsx', skiprows=lambda x: x in [0, 3],
                   sheet_name='Messstellen Lf', index_col=0, usecols="B:M")

# Iterator through the header of the dataframe with hydraulic heads containing the names of measurement places
for col in gw.columns :
    # Iterator through a column with a time series of measurements
    for i in range(1,len(gw[col])) :
        # Test if a measurement exists for the queried date and measurement place
        if gw[col].iloc[i] > 0 :
            # Generation of a new metric with the queried measurement and structered after the concept of HCBC
            point = (
                influx.Point("Messrunde")
                 .tag("organisation", "TUB")
                 .tag("department", "IngGeo")
                 .tag("object", "Geothermie")
                 .tag("position", "GW")
                 .tag("sensor", 'manual')
                 .tag('node', 'Messstelle_'+str(col))
                 .field('Flurabstand_m', gw[col].iloc[i])
                 .field('Temperatur_°C', t[col].iloc[i])
                 .field('Leitfähigkeit_mS/cm', lf[col].iloc[i])
                 .time(gw[col].index[i])
                 )
            # Upload the new metric to InfluxDB
            write_api.write(bucket=bucket, org=org, record=point)
            # Helpers for debugging
            print(point)
```

Abbildung 13: Testskript für den Upload von Daten in InfluxDB über einen Python-Client

4.5 Serverschnittstelle

4.5.1 Plugin-Agent Telegraf

Telegraf ist ein ebenfalls von InfluxData entwickelter und frei verfügbarer Plugin-Agent, welcher u.a. aber nicht ausschließlich in Verbindung mit InfluxDB verwendet wird. Ein Agent agiert im Auftrag einer anderen Anwendung oder eines Systems und führt im Hintergrund Aufgaben der Datenverarbeitung durch. Telegraf ist Plugin basiert, was bedeutet, dass die Anwendung aus fertigen Bausteinen besteht, die für den individuellen Zweck ausgesucht und eingeladen werden. Im Kontext des LoRaWAN-Netzwerks von HCBC ist Telegraf die Schnittstelle für die Kommunikation von Netzwerkserver mit Anwendungsserver und hat die Funktion, die Rohdaten der Datenlogger von The Things Stack abzurufen, zu verarbeiten und an den EMU-Server weiterzuleiten. Telegraf wurde wie InfluxDB in ‘Go‘ programmiert, einer aufstrebenden Skriptsprache für Webentwicklung. Go enthält ähnliche Konzepte wie Python, also ein effizienter und intuitiv verständlicher Code, Typsicherheit und Strukturierung von Codeblöcken über obligatorische Einrückungen. Abgesehen davon orientiert sich Go bzgl. der Syntax eher an C++

und muss kompiliert werden. Eine funktionsfähige Telegraf-Anwendung verfügt mindestens über ein Input-Plugin als Datenquelle, ein Output-Plugin als Zielort für die Weiterleitung der Daten, sowie über einen Basiskörper mit der Root-Konfiguration von Telegraf. Neben Input- und Output-Plugins existiert mit den Processor-Plugins eine dritte wichtige Kategorie mit dem Zweck, die Daten vor der Weiterleitung zu filtern und zu verarbeiten. Darüber hinaus gelten Aggregator-Plugins als vierte Kategorie, wobei es sich im Grunde um Processor-Plugins handelt, welche auf die Datenaggregation spezialisiert sind. Mithilfe der Plugins ist Telegraf in der Lage flexibel zahlreiche Datenformate (z.B. JSON, *Inlux* oder CSV) zu verarbeiten und verschiedene Kommunikationsprotokolle (z.B. HTTP oder MQTT) zu nutzen. Zum derzeitigen Zeitpunkt (September 2023) existieren insgesamt 363 frei verfügbare Telegraf-Plugins. Da es sich bei Telegraf um eine Eigenentwicklung von InfluxData handelt, stellt das Unternehmen einen Onlinekurs speziell für die Nutzung von Telegraf bereit[14].

4.5.2 Installation & Start von Telegraf

Anmerkung:
Die folgende Beschreibung bezieht sich auf die Bedienung von Telegraf auf einem PC mit Windows 11-Distribution, was zum Schreiben, Testen und Debuggen des Skripts für die Konfigurationsdatei von Telegraf hilfreich ist. Für den Einsatz von Telegraf im LoRaWAN-Netzwerk muss der *Agent* von einem Server ausgeführt werden, da die Anwendung mit dem Abschalten eines PCs ebenfalls geschlossen wird.

In der Download-Sektion von InfluxData wird die aktuelle Version von Telegraf für verschiedene Betriebssysteme kostenlos zur Verfügung gestellt[15]. Die Bedienung von Telegraf erfolgt über eine CLI (***C**omand **L**ine **I**nterface*) wie z.B. die Kommandozeile oder Windows PowerShell. Eventuelle Probleme beim Ausführen von Telegraf können mit den Sicherheitseinstellungen des Computers bzw. der Firewall zusammenhängen. Bei der Bedienung von Telegraf über die Kommandozeile ausführen zu können, wird zunächst in das Verzeichnis gewechselt, in welchem sich die Datei telegraf.exe befindet.

```
cd “C:\Program Files\InfluxData\telegraf“
```

Falls Telegraf manuell über den Link heruntergeladen wurde, muss die Anwendung noch mit den beiden folgenden Befehlen installiert werden:

```
.\telegraf.exe --service install
Start-Service telegraf
```

Die erfolgreiche Installation kann mit folgendem Befehl überprüft werden:

```
telegraf -version
```

Um Telegraf für den individuellen Einsatzzweck zu konfigurieren, wird eine Textdatei im TOML-Format benötigt. Die Konfigurationsdatei wird unter Angabe eines Strings mit dem Dateipfad, mit dem folgenden Befehl eingebunden. Es können auch mehrere Konfigurationsdateien gleichzeitig ausgeführt werden.

```
telegraf –config “telegraf.conf“
```

[14] https://university.influxdata.com/courses/data-collection-with-telegraf-tutorial
[15] https://portal.influxdata.com/downloads/telegraf

Wenn Telegraf für Testzwecke ausgeführt werden soll, kann über das Keyword '--debug' das Log, also das Protokoll der internen Prozesse von Telegraf offen gelegt werden (s. Abbildung 14). Mit dem Keyword '-test' wird das Output-Plugin ausgeschaltet, d.h. es werden noch keine Daten in InfluxDB gespeichert.

```
telegraf --debug --config 'telegraf.conf' -test
```

```
C:\Windows\System32>cd "C:\Program Files\InfluxData\telegraf
>
C:\Program Files\InfluxData\telegraf>telegraf --config "telegraf_geo.conf"
2023-06-29T10:51:50Z I! Loading config file: telegraf_geo.conf
2023-06-29T10:51:50Z I! Starting Telegraf 1.26.0
2023-06-29T10:51:50Z I! Available plugins: 235 inputs, 9 aggregators, 27 processors, 22 parsers, 57 outputs, 2 secret-st
ores
2023-06-29T10:51:50Z I! Loaded inputs: mqtt_consumer
2023-06-29T10:51:50Z I! Loaded aggregators:
2023-06-29T10:51:50Z I! Loaded processors: starlark
2023-06-29T10:51:50Z I! Loaded secretstores:
2023-06-29T10:51:50Z I! Loaded outputs: influxdb_v2
2023-06-29T10:51:50Z I! Tags enabled: host=Yoda
2023-06-29T10:51:50Z I! [agent] Config: Interval:10s, Quiet:false, Hostname:"Yoda", Flush Interval:10s
2023-06-29T10:51:50Z D! [agent] Initializing plugins
2023-06-29T10:51:50Z D! [agent] Connecting outputs
2023-06-29T10:51:50Z D! [agent] Attempting connection to [outputs.influxdb_v2::mqtt_pusher]
2023-06-29T10:51:50Z D! [agent] Successfully connected to outputs.influxdb_v2::mqtt_pusher
2023-06-29T10:51:50Z D! [agent] Starting service inputs
2023-06-29T10:51:50Z I! [inputs.mqtt_consumer::thing_network_consumer] Connected [tcp://eu1.cloud.thethings.network:1883
]
2023-06-29T10:52:00Z D! [outputs.influxdb_v2::mqtt_pusher] Buffer fullness: 0 / 100 metrics
2023-06-29T10:52:10Z D! [outputs.influxdb_v2::mqtt_pusher] Buffer fullness: 0 / 100 metrics
```

Abbildung 14: Log von Telegraf nach Ausführung in der Kommandozeile

4.5.3 Konfiguration von Telegraf

Die Konfigurationsdatei kann mit einem beliebigen Editor (z.B. Notepad++) geschrieben werden, nur muss zur Identifikation der Datei ein '.conf' dem Dateinamen angehängt werden. Wie bereits erwähnt, besteht eine Konfigurationsdatei aus mindestens einem Input- und Output-Plugin, und kann darüber hinaus Prozessor-Plugins besitzen. Die Anzahl der Plugins innerhalb einer Konfigurationsdatei ist unbegrenzt, eine hohe Anzahl kann aber die Anfälligkeit für Bugs erhöhen. Der Ablauf der Telegraf-Anwendung folgt den Kategorien der Plugins; die Input-Plugins tätigen die Abfrage der Rohdaten, die Prozessor-Plugins verarbeiten sie und Output-Plugins übermitteln die verarbeiteten Daten an eine Datenbank. Die Auswahl der Input und Output- Plugin ist zumeist abhängig von den beteiligten Netzwerkprotokollen und Datenformaten. Die Prozessor-Plugins können etwas freier nach Präferenz oder bestehenden Kenntnissen gewählt werden. In der Konfiguration von HCBC werden die Rohdaten aus dem *Livestream* von The Things Stack durch das MQTT-Consumer Input-Plugin von The Things Stack über das MQTT-Protokoll abgerufen (s. Abbildung 15). Mit dem JSON_v2 Parser wird dabei ein integriertes Sub-Plugin genutzt, welches die Messwerte aus dem Log eines *Events* herausfiltert, und sie vor der Übergabe an das Prozessor-Plugin vom JSON- in das *Influx-Datenformat* umwandelt. Das Starlark Prozessor-Plugin nutzt einen Python-Dialekt, um komplexere logische Abfragen durchzuführen und die Messdaten in die im Monitoring-Konzept festgelegte Speicherstruktur zu bringen. Das InfluxDB_v2 Output-Plugin leitet die Daten an den EMU-Server weiter und legt den Speicherort fest.

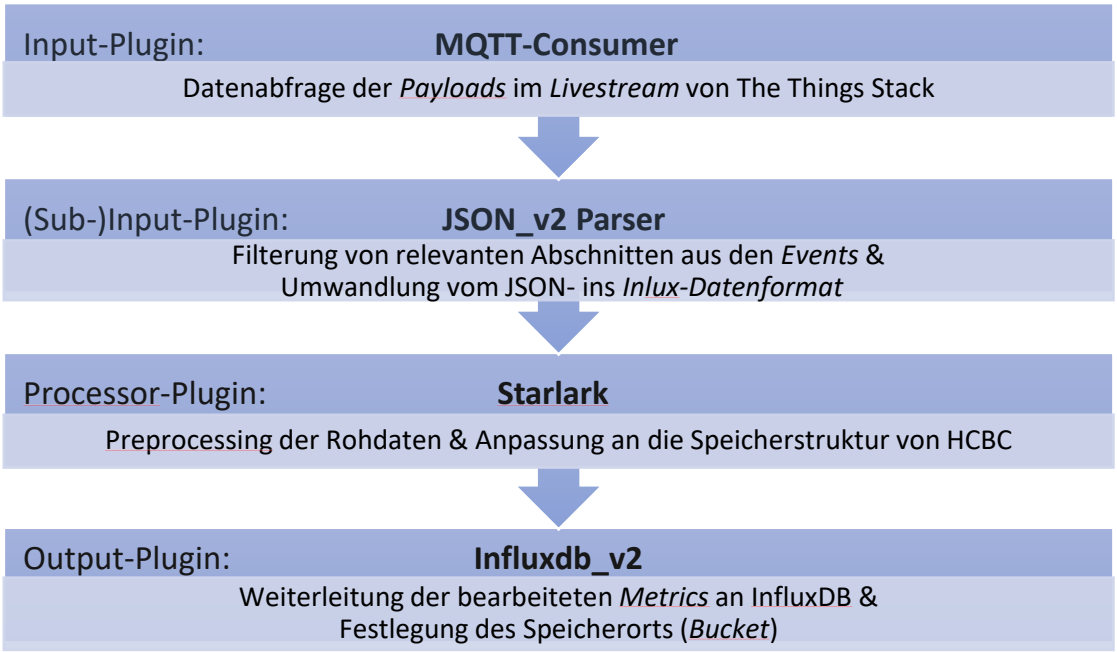

Abbildung 15: Schema der Datenverarbeitung in Telegraf

Das Skript für die Konfigurationsdatei muss das TOML-Format besitzen, in welchem fertig kompilierte Bausteine (Plugins) mit Keywords eingebunden und konfiguriert werden. Einzelne Plugins werden im Skript als Blöcke durch einen Header voneinander abgegrenzt. Der Header enthält zwei eckige Klammern mit dem Namen des Plugins, dem in Punktnotation die Kategorie vorangestellt ist (inputs, outputs, processors). Die Reihenfolge der Plugins im Skript spielt keine Rolle, innerhalb der Plugins gilt jene Hierarchie, welche mittels Einrückungen realisiert wird. Neben den Plugins besitzt eine Konfigurationsdatei immer einen Block mit den Root-Einstellungen. Dieser Block ist mit dem Header '[agent]' betitelt (s. Abbildung 16). Die Root-Einstellungen regeln die zeitliche Aktivität der Plugins und die Menge der übertragenen Daten. Eine Übersicht der einzelnen Plugins ist u.a. auf der Website von InfluxData zu finden[16]. Die API-Referenz der einzelnen Plugins wurde auf Github publiziert[17].

Im Block mit den Root-Einstellungen wird die zeitliche Aktivität der Input-Plugins über das Keyword 'intervall' und die der Output-Plugins über das Keyword 'flush_intervall' gesteuert. Mit dem Keyword 'metric_batch_size' wird die maximale Anzahl an *Metrics* festgelegt, die von den Output-Plugins pro Aktivitätszyklus an den Anwendungsserver weitergeleitet bzw. in die Datenbank geschrieben werden. Wenn die maximale Anzahl überschritten wird, werden nicht weitergeleitete *Metrics* zwischengespeichert. Die Größe des Zwischenspeichers wird über das Keyword 'metric_buffer_limit' festgelegt, wobei es generell sinnvoll ist, einen ausreichend großen Speicher zu wählen, um sicherzustellen, dass keine Informationen verloren gehen. Ein Jitter für die Aktivitätsintervalle der Plugins ist dazu gedacht, kleine Fluktuationen in den Datenabfragen einzubauen, um bei annähernd gleichzeitig eintreffenden Daten unterschiedliche Zeitstempel und damit separate *Metrics* zu generieren. Dies kann bei einer Telegraf-Anwendung nützlich sein, die sehr viele Daten verarbeitet, um Informationsverlust durch das Überschreiben von *Metrics* zu verhindern. Mit dem Keyword 'precision' wird die Genauigkeit der

[16] https://docs.influxdata.com/telegraf/v1.27/plugins
[17] https://github.com/influxdata/telegraf/tree/master/plugins

Timestamps für die *Metrics* eingestellt, die von den Output-Plugins weitergeleitet werden. Präzisere *Timestamps* eintreffenden *Metrics* werden gerundet. Bei Verwendung der Konfigurationsdatei zu Testzwecken kann mit den optionalen Keywords ‘debug‘ und ‘quit‘ ein Log der internen Prozesse von Telegraf ausgegeben werden.

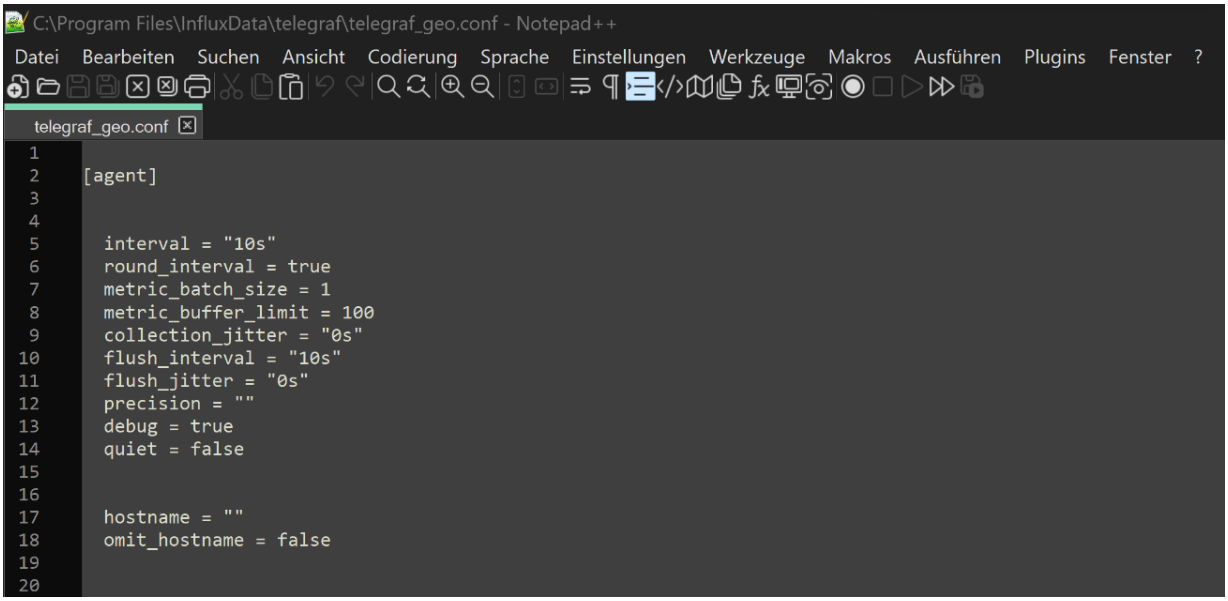

Abbildung 16: Skriptabschnitt mit der Root-Konfiguration des Telegraf-Agents

4.5.4 MQTT-Consumer Input-Plugin

Das MQTT-Consumer Plugin wird dazu verwendet, die *Events* aus dem *Livestream* des TTN abzufragen (s. Kap 4.3.3). Die Keywords des Plugins dienen der Konfiguration der Verbindung zum Netzwerkserver über das MQTT-Protokoll (s. Kap. 4.3.5). Unter dem Keyword ‘server‘ wird die IP-Adresse des *Brokers* angegeben, in diesem Fall die des europäischen Servers von The Things Stack mit dem für MQTT reservierten Port (s. Abbildung 17). Als Passwort für die Autorisierung der Datenabfragen wird der zuvor im TTN generierte API-Key benötigt (s. Kap 4.3.4). Die Angabe der *Topics* richtet sich nach dem spezifischen Namensschema des Netzwerks, was bei HCBC im Monitoring Konzept 2019 festgelegt wurde.

Mit dem Keyword ‘data_format‘ wird das Format der Inputdaten definiert. Die *Events* im *Livestream* des TTN befinden sich im JSON-Datenformat, da die Kommunikation zwischen Gateway und Netzwerkserver über das HTTP-Protokoll abgewickelt wurde. In den MQTT-Consumer als Sub-Plugin integriert, ist der JSON_2 Parser (s.u.), welcher über zusätzliche Methoden zur Datenfilterung verfügt. Für die Nutzung des JSON_2 Parsers wird allerdings die modernere Interpretation ‘JSON_v2‘ vorausgesetzt. Weitere Keywords sind optional und können in der API-Referenz des Plugins recherchiert werden[18]. Wie eine Schnellverbindung zum TTN ohne weitere Filter mittels MQTT-Consumer Plugin eingerichtet wird, wird in einem Videotutorial von einer InfluxDB-Entwicklerin erklärt[19]. In einem Beitrag auf Github wird zudem ein Template bereitgestellt[20].

[18] https://github.com/influxdata/telegraf/tree/master/plugins/inputs/mqtt_consumer
[19] https://www.youtube.com/watch?v=ENZLOOWn3us
[20] https://github.com/influxdata/community-templates/blob/master/thing_network/thing_network.yml

```
21
22  [[inputs.mqtt_consumer]]
23
24    servers = ["tcp://eu1.cloud.thethings.network:1883"]
25
26    username = "name_of_application@ttn"
27    password = "NNSXS.44TNNS3SBZ2G7GESFCM3O3LZDB6K2ZARYLSIGMY.DWTI674LRZ5DZEÜGDXHM3CDHZÖ9V7E"
28
29    topics = ["Test-Topic"]
30    topic_tag = ""
31    qos = 0
32    client_id = ""
33    data_format = "json_v2"
34
```

Abbildung 17: Skriptabschnitt mit der Konfiguration des MQTT-Consumer-Plugins

4.5.5 JSON_v2 Parser

Der JSON_v2 Parser ist ein Sub-Plugin, das dem Processor-Plugin als Filter vorgeschaltet ist. Die Verwendung des MQTT-Consumers ohne Datenfilter führte zu einer unkontrollierten Abfrage aller *Events* im *Livestream*, auch jenen ohne Messdaten wie *Join-Request-/Join-Accept Messages* sowie Fehlermeldungen. Des Weiteren wurden alle in der JSON-Datei eines *Events* enthalten Parameter als eigenständige *Metric* importiert (vgl. Abbildung 8). Der JSON_v2 Parser dient daher dem Zweck die Messdaten und relevante Metadaten zu extrahieren. Diese können zwar auch in einem Processor-Plugin gefiltert und zusammengeführt werden, allerdings führte der Ansatz vermehrt zu Bugs und einer ineffizienten Datenverarbeitung. Die zweite Funktion des JSON_v2 Parsers ist die Transformation der Input-Daten vom JSON- ins *Influx-Datenformat*. JSON ist ein Akronym für **J**ava**S**cript **O**bject **N**otation und ist ein übliches Dateiformat für den Datenverkehr zwischen Webanwendungen im HTTP-Protokoll. Die Datenspeicherung erfolgt nach einem mit Doppelpunkt notierten Schlüssel-Wert Prinzip, wobei Schlüssel immer den Datentyp String besitzen, und Werte diverse Datentypen enthalten können. Die Notation von Strings mit doppelten Anführungszeichen ist anders als bei Go oder Python/Starlark in anderen Abschnitten des Skripts nicht obligatorisch. Als übergeordnete Container kennt das JSON-Datenformat Arrays und Objekte. Bei einem Array zeigt ein Schlüssel auf mehrere Werte, die kommasepariert in eckigen Klammern zusammengefasst werden. Bei einem Objekt zeigt ein Schlüssel auf mehrere Schlüssel-Wert Paare, die mit geschweiften Klammern gruppiert werden. Objekte können beliebig ineinander verschachtelt und die Ebenen mithilfe von Einrückungen strukturiert werden.

In den Root-Einstellungen des JSON_v2 Parsers wurde für alle gefilterten *Metrics* ein einheitlicher String für *Measurement* und dieselben *Timestamps* festgelegt, damit diese zu einer einzelnen *Metric* zusammengefasst werden (s. Abbildung 18). Die Importe einzelner Schlüssel-Wert Paare oder ganzer Objekte werden als Blöcke voneinander abgegrenzt, welche als Importbefehl den Header des JSON_v2 Parsers mit dem in Punktnotation angehängten Zieldatentyp enthalten. Bei Import eines Objekts werden, falls nicht anders deklariert, alle Schlüssel-Wert Paare als *Fields* interpretiert. Weiterführende Informationen zur JSON_v2 Interpretation finden sich auf der Website von InfluxData[21], die API-Referenz ist auf Github verfügbar[22].

[21] https://docs.influxdata.com/telegraf/v1.21/data_formats/input/json_v2/
[22] https://github.com/influxdata/telegraf/tree/master/plugins/parsers/json_v2

```
37  [[inputs.mqtt_consumer.json_v2]]
38
39    measurement_name = "TTN"
40    timestamp_key = "@this.received_at"
41    timestamp_format = "unix"
42
43  [[inputs.mqtt_consumer.json_v2.object]]
44    path = "@this.uplink_message.decoded_payload"
45    excluded_keys = "raw"
46    disable_prepend_keys = true
47
48  [[inputs.mqtt_consumer.json_v2.tag]]
49    path = "@this.end_device_ids.device_id"
50    disable_prepend_keys = true
```

Abbildung 18: Skriptabschnitt mit der Konfiguration des JSON_v2 Parsers

4.5.6 Starlark Processor-Plugin

Nach dem JSON_v2 Parser entsprechen die eintreffenden *Metrics* gefilterten Rohdaten. Vor dem Speichern der Daten in InfluxDB werden im Starlark Processor-Plugin mehrere Operationen zur Datenverarbeitung durchgeführt, die im Folgenden aufgelistet sind.

a) Unterscheidung der beiden *Payloads* mit dem ersten und zweiten Teil einer Messung auf Basis des Wertes für die Variable ‘type‘ (s. Kap. 4.1)

b) Umbenennung der Bezeichnung für die Schlüssel der *Fields* von den internen Variablen der Datenlogger zum tatsächlichen physikalischen Parameter

c) Zuordnung von Metadaten einer *Metric* anhand der einzigartigen DevEUI und Hinzufügen als *Tag* nach der im Monitoring Konzept 2019 festgelegten Speicherstruktur

d) Offenlegung der internen Prozesse von Telegraf, um effizientes Debuggen zu ermöglichen

e) Kontrollstrukturen zur Fehlerbehandlung

Starlark ist sowohl die Bezeichnung des zugehörigen Prozessor-Plugins in Telegraf, als auch der Name einer Skriptsprache, welche einen an Webentwicklung angepassten Dialekt von Python darstellt. Im Gegensatz zu anderen Processor-Plugins ist Starlark nicht Keywords basierend, sondern besitzt einen integrierten Interpreter. Es bestehen einige signifikante Unterschiede zwischen Python und Starlark, z.B. können gängige Packages nicht verwendet werden. Darüber hinaus sind einige weitere Funktionalitäten nicht verfügbar, welche in Tabelle 4 zusammengefasst sind.

Tabelle 4: Unterschiede von Starlark und Python

Funktion/Unterschied	Implementierung mit Python	Implementierung mit Starlark
Packages	Standard-Bibliothek und breite Palette an externen Packages	Starke Limitierung an verfügbaren Packages, Einladen über die Funktion load()
Höhere mathematische Berechnungen	Diverse Python-Packages wie numpy, scipy usw.	Starlark-Package math.star
Konvertierung & Berechnungen mit Zeitinformationen	Diverse Python-Packages wie time, datetime usw.	Starlark-Package time.star

Datenmanagement & Konvertierung von Datenformaten	Verschiedene Packages zur Konvertierung nahezu aller Datenformate	Ausschließlich Konvertierung des JSON-Datenformats mit Starlark-Bibliothek json.star möglich
InfluxDB & *Inlux-Datenformat*	Python-Package influxdb_client	Automatisch enthalten
Objektorientierte Programmierung	Strukturierung über Keyword class	Strukturierung über das *Influx-Datenformat*
Löschen	Löschen von Variablen und Werten über Keyword del	Löschen von *Fields* und *Tags* über die pop()- oder clear()-Methode
Fehlerbehandlung	Unspezifische Fehlerbehandlung über Keywords wie try, except und raise möglich	Nur spezifische Fehlerbehandlung über logische Kontrollstrukturen möglich
Debugging	Automatische Ausgabe in der Konsole	Ausgabe als Log in Verbindung mit Starlark-Package logging.star

Das Starlark-Plugin besitzt bloß zwei Keywords, welche zum Einladen des Starlark-Skripts gedacht sind. Entweder wird das Skript über das Keyword ‘script‘ als externe Datei eingebunden oder das Keyword ‘source‘ genutzt, um es direkt in die Konfigurationsdatei einzubetten (s. Abbildung 19). Unabhängig von den allgemeinen Unterschieden zwischen Python und Starlark, muss das Starlark-Skript für den Einsatz im Processor-Plugin eine definierte Struktur aufweisen. Im Rahmen des Skripts wird immer eine ‘apply(metric)‘-Funktion definiert, die auf alle eintreffenden *Metrics* angewendet wird. Am Ende der Funktion wird über das Keyword ‘return‘ eine oder eine Liste mit mehreren verarbeiteten *Metrics* zurückgegeben, die anschließend das Output-Plugin erreichen. Falls keine *Metric* über ‘return‘ zurückgegeben wird, werden die Eingangsdaten gelöscht, ohne an das Output-Plugin weitergereicht zu werden, was in Kombination mit logischen Abfragen für die Fehlerbehandlung ausgenutzt werden kann. Mit Starlark lassen sich ausschließlich drei Dateiformate verarbeiten; (CSV-notierte) Textdateien sowie Dateien im *Influx*- und JSON-Format. Dateien im JSON-Format können mit der ‘json.star‘-Bibiliothek umgewandelt werden, wie es in einem Youtube-Tutorial eines Mitglieds der Github-Community unter Umgehung des JSON_2 Parsers vorgeführt wird[23]. *Metrics* im *Influx-Datenformat* werden von Starlark automatisch erkannt und bei der Datenverarbeitung wie ein Objekt einer Klasse in Python behandelt. Eine Dokumentation des Plugins ist auf Github zu finden[24].

[23] https://www.youtube.com/watch?v=rC5uLbA2ZZU
[24] https://github.com/influxdata/telegraf/tree/master/plugins/processors/starlark

```
57      [[processors.starlark]]
58        # Additional Filter
59        namepass = ["TTN"]
60
61        # Actual start of the starlark-script
62        source = '''
63      load("logging.star", "log")
64
65      def apply(metric):
66          # preventive error handling
67          log.debug("input-data: {}".format(metric))
68          if metric.name == "LoRaWAN" :
69              return None
70          if not "type" in metric.fields:
71              return None
72
73          # Remove previous non-sense tags
74          metric.tags.clear()
75          # Add global tags to the measurement
76          metric.tags["organisation"] = "TUB"
77          metric.tags["department"] = "IngGeo"
78          metric.tags["object"] = "GWMS"
79          metric.tags["node"] = "TTN"
80
81
82          # Add individual tags for logger-id and well-nr
83          if metric.fields["device_id"] == "DevEUI_1":
84              metric.tags["sensor"] = "L00146"
85              metric.tags["position"] = "Messstelle_1"
86          elif metric.fields["device_id"] == "DevEUI_2":
87              metric.tags["sensor"] = "L00145"
88              metric.tags["position"] = "Messstelle_2"
89          elif metric.fields["device_id"] == "DevEUI_3":
90              metric.tags["sensor"] = "L00144"
91              metric.tags["position"] = "Messstelle_3"
92          else:
93              metric.fields["error"] = "unknown_device"
94
95          # The origin of the measurement is split in 2 payloads.
96          # Part 1 and 2 are queried by variable <type>.
97          # The values of the payload get assigned to the correct fields
98          # by mapping a scheme for type 19 and type 20 on them.
99
100         scheme19 = {
101         "val1": "battery_voltage",
102         "val2": "device_temperature",
103         "val3": "hydraulic_head"  }
104         scheme20 = {
105         "val1": "gw_temperature"  }
106
107         if metric.fields["type"] == 19.0 :
108             for k, v in metric.fields.items():
109                 if k in scheme19:
110                     metric.fields[scheme19[k]] = v
111                 metric.fields.pop(k)
112         elif metric.fields["type"] == 20.0 :
113             for k, v in metric.fields.items():
114                 if k in scheme20:
115                     metric.fields[scheme20[k]] = v
116                 metric.fields.pop(k)
117         else :
118             metric.fields.clear()
119             metric.fields["error"] = "unknown_type_of_message"
120
121         # Passing on the processed payload to the output-plugin
122         metric.name = "LoRaWAN"
123         log.debug("output-data: {}".format(metric))
124         return metric
125     '''
```

Abbildung 19: Skriptabschnitt mit der Konfiguration des Starlark Processor-Plugins

4.5.7 InfluxDB_v2 Output-Plugin

Das InfluxDB_v2 Output-Plugin kann als Template in der GUI von InfluxDB heruntergeladen bzw. kopiert werden. Die notwendigen Keywords für eine erfolgreiche Verbindung mit InfluxDB beinhalten Angaben für die Authentifizierung der Telegraf-Anwendung. Dazu gehört die vollständige Webadresse eines Servers auf dem InfluxDB installiert ist, wie dem Server von InfluxDB Cloud oder dem EMU-Server bei HCBC. Daneben muss ein *Bucket* als Speicherort für die verarbeiteten *Metrics* und ein API-*Token* mit entsprechender Berechtigung angegeben werden. API-*Token* stellen eine sensible Information dar, weshalb empfohlen wird diese als Systemvariablen abzuspeichern. Systemvariablen gelten als einer der sichersten Speicherplätze eines Rechners, auch kann es bei der Verwaltung vieler *Token* und ihrer mehrfachen Verwendung hilfreich sein, diese mit Variablennamen für die Zuordnung zu versehen.

5. Fazit

5.1 Evaluation der Netzwerkeinrichtung

Das 2015 entwickelte LoRaWAN-Protokoll ist eine vergleichsweise junge Technologie, die im Rahmen des geothermischen Monitorings erstmals durch das FG Ingenieurgeologie erprobt wurde. Sekundärer Zweck der Einrichtung eines LoRaWAN-Netzwerks am HCBC war es daher, Expertise auf dem Gebiet der LoRaWAN-Technologie zu sammeln und ihr Einsatzpotenzial für zukünftige Projekte zu bewerten. Das vorliegende Kapitel enthält eine Diskussion von Schwierigkeiten bei der Netzwerkeinrichtung sowie eine Rezension der LoRaWAN-Technologie und der verwendeten Netzwerkkomponenten. Die Netzwerkeinrichtung erfolgte durch Mitarbeiter der TUB aus dem Fachgebiet Ingenieurgeologie. Obwohl handelsübliche Komponenten eingesetzt wurden, erwies sich die Einrichtung des LoRaWAN-Netzwerks langwieriger als erwartet. Während der Pilotphase kam es zu zahlreichen Bugs bei Integration einzelner Komponenten, sodass eine aufwendige Einarbeitung in ihre Funktionsweise und Bedienung notwendig war, was insbesondere InfluxDB und Telegraf betrifft. Während sich der wissenschaftliche Diskurs zur LoRaWAN-Technologie zumeist auf eine theoretische Analyse des LoRaWAN-Protokolls fokussiert, ließen sich Informationen für ihre praktische Anwendung stattdessen in Internetforen und den Websites der Entwickler entnehmen (graue Literatur). Detailprobleme wurden mittels Try&Error-Methode gelöst, wobei Hindernisse vor allem aus einem Ensemble von kleineren Bugs und Wissenslücken aus dem Bereich der Webentwicklung bestanden.

Als unzuverlässigste Komponente des Netzwerks hatte sich während der Pilotphase das Gateway LPS8 von Dragino erwiesen. Nach einem Update im Januar 2023 kam es zu einem Totalausfall des Gateways, der erst mit einem Boot der Firmware behoben werden konnte. Vor dem Update bestand keine Möglichkeit der TLS-Verschlüsselung, sodass das LPS8 nicht die allgemeinen Sicherheitsanforderungen der TUB erfüllte und zunächst nur unvollständig in das lokale Netzwerk integrierbar war.

Die maximale Reichweite der LoRa-Verbindung stellte sich in der dicht bebauten Umgebung des Campus Berlin-Charlottenburgs mit wenigen 100 m als vergleichsweise gering heraus, obwohl das Gateway in einem hohen Gebäude platziert wurde.

Die Verwendung des kostenlosen Netzwerkservers von The Things Network wird als sinnvoll erachtet, da dieser Vorteile wie eine hohe Systeminteroperabilität und gemeinsame Ressourcennutzung gewährt. Zudem wird durch die enge Verknüpfung des TTN mit der LoRa Alliance sichergestellt, dass das Netzwerk die aktuellen Standards der LoRaWAN-Technologie verwendet.

Die Verwendung von InfluxDB als Anwendungsserver wurde im Monitoring Konzept von 2019 festgelegt, um den Wandel vom Parallelbetrieb zur Nutzung einer zentralen Datenbank für das HCBC-Projekt einzuleiten. InfluxDB wurde von einem renommierten Tech-Unternehmen für den Einsatz bei IoT-Anwendungen mit einer hohen Anzahl unterschiedlicher Geräte und großen Datenmengen entwickelt und wartet mit einer breiten Palette an Funktionalitäten auf. In einem kleineren Netzwerk hingegen konnte die Software ihr Potenzial nicht entfalten. Während des geothermischen Monitorings ist angedacht, eine überschaubare Menge von Datenloggern einzusetzen, bei denen im Hinblick auf den Untergrund als träges System kein

Bedarf nach einem hochfrequenten Messintervall besteht. In Bezug auf den Einsatz im LoRaWAN-Netzwerk kann die von InfluxDB bereitgestellte Infrastruktur nicht ausgenutzt werden. Auch ist zu bemängeln, dass für die vollumfängliche Nutzung der Software tiefergehende Kenntnisse aus dem Bereich der Informatik benötigt werden, sodass die Abstriche in der Benutzerfreundlichkeit überwiegen.

Abschließend lässt sich festhalten, dass LoRaWAN zum gegenwärtigen Zeitpunkt eine Technologie ist, die sich noch merklich in der Entwicklung befindet. Dies spiegelt sich in der Anfälligkeit für Bugs sowie häufigen Updates und Release neuer Versionen der Netzwerkkomponenten wider. In früheren Versionen des LoRaWAN-Protokolls wurden zudem potenzielle Sicherheitslücken nachgewiesen (Noura et. al. 2020). Trotzdem besitzt LoRaWAN in Bezug auf Verbindungsstabilität und Interoperabilität der Systemkomponenten entscheidende Vorteile für den Einsatz in einem automatisierten Umweltmonitoring. Auch ist anzunehmen, dass LoRaWAN-Technologie in absehbarer Zeit zunehmend dokumentiert und standardisiert sein wird.

5.2 Schlussfolgerungen für die Konzeption der Netzwerkstruktur

Im vorliegenden Kapitel werden auf Basis der Erfahrungen mit der Netzwerkeinrichtung am HCBC Schlussfolgerungen für die künftigen Einsatzbereiche von LoRaWAN-Technologie durch das Fachgebiet Ingenieurgeologie gezogen, und Empfehlungen für die Konzeption der Netzwerkstruktur zu Zwecken wie Umweltmonitoring abgeleitet. Die Erprobung der LoRaWAN-Technologie lieferte als wichtigste Erkenntnis für künftige Einsatzbereiche, dass es sich nicht um eine Plug&Play-Lösung für die automatisierte Erhebung von Messdaten im Sinne einer IoT-Anwendung handelt. Die Datensicherheit von LoRaWAN-Verbindungen beruht auf der gegenseitigen Authentifizierung mit einem Netzwerkserver und der Verwendung einer dynamischen Verschlüsselung, was den Einsatz von LoRaWAN-fähigen Geräten im Rahmen eines Netzwerks erforderlich macht. Die Einrichtung eines Netzwerks kann mitunter aufwendig sein, weshalb dieses für einen langfristigen Betrieb wie z.B. ein Monitoring ausgelegt sein sollte. Nach der Netzwerkeinrichtung ist die Integration gleichartiger Geräte jedoch ohne größeren Aufwand möglich. Die digitale Infrastruktur des LoRaWAN-Netzwerks am HCBC ist derzeitig nicht einmal annähernd ausgelastet und könnte theoretisch hunderte Geräte verwalten. Der limitierende Faktor bei der Vergrößerung des Netzwerks stellt die Empfangsabdeckung durch die Gateways bzw. ihrer potenziellen Standorte dar.

Da die Ausbreitung des LoRa-Signals in urbanen Gebieten anisotrop und stark limitiert ist, sollte die Empfangsabdeckung vor der Konzeption des Netzwerks experimentell bestimmt oder ggfs. modelliert werden. Erst nachdem anhand der effektiven Reichweite potenzielle Standorte für Gateways und Endgeräte ermittelt wurden, sollte ein Konzept für die Netzwerkstruktur entworfen und geeignete Komponenten beschafft werden. Wegen anfänglicher Bugs und nachgewiesener Sicherheitslücken (Noura et. al. 2020 & Sundaram et. al. 2020) wird empfohlen auf neuere Geräte zu setzen, welche die aktuelle Version des LoRaWAN-Protokolls verwenden. Vor der Beschaffung kann vorhandene Infrastruktur durch externe Gateways geprüft werden, z.B. auf einer Übersichtskarte mit öffentlich sichtbaren Gateways von The Things Network[25]. Allerdings ist die Verwendung von eigenen Geräten der Netzwerksicherheit und -stabilität zuträglich. Die Einrichtung eines LoRaWAN-Netzwerks muss individuell auf die verwendeten

[25] https://ttnmapper.org

Komponenten und ihren zugedachten Einsatzzweck abgestimmt werden und lässt sich nicht nach einer vorgefertigten Anleitung durchführen. Anstatt das Netzwerk wie bei HCBC aus Komponenten unterschiedlicher Anbieter zusammenzusetzen, könnte es sich in Hinsicht auf eine unkomplizierte Netzwerkeinrichtung als vorteilhaft erweisen, mit dem TTN kompatible Netzwerk-Bundles zu erwerben, welche z.B. von Amazon oder Microsoft angeboten werden. Der Betrieb von Endgeräten im ABP-Modus könnte das Problem ebenfalls lösen, würde aber zulasten der Datensicherheit gehen. Bei der Implementierung von großen und komplexen LoRaWAN-Netzwerken könnte es ggfs. wirtschaftlicher sein, die Einrichtung einem Dienstleister mit entsprechender Expertise zu überlassen.

6. Quellenverzeichnis

Brand, S. (2021): IoT Deployment von LoRaWAN-Sensoren im urbanen Umfeld zur Messung von Grundwassertemperaturen und -ständen, Eine Analyse zu Vor- und Nachteilen, Hindernissen und Umsetzbarkeit; Abschlussarbeit B.Sc., Technische Universität Berlin, Fachgebiet Ingenieurgeologie

Fujdiak, R. et. al. (2018): Simulated Coverage Estimation of Single Gateway LoRaWAN Network; 25th International Conference on Systems, Signals and Image Processing (IWSSIP), pp. 1-4, Maribor, Slovenia

Gilson, R & Grudsky, M. (2017): LoRaWAN capacity trial in dense urban environments; Semtech Corporation & MachineQ

Mekki, K. et. al. (2019): A comparative study of LPWAN technologies for large-scale IoT deployment; ICT Express Vol. 5 Issue 1, pp. 1-7

Noura, H. et. al. (2020): LoRaWAN security survey: Issues, threats and possible mitigation techniques; Internet of Things Vol. 12, Elsevier B.V.

Oniga, B., et. al. (2017): Analysis, design and implementation of secure LoRaWAN sensor networks; *13th IEEE International Conference on Intelligent Computer Communication and Processing (ICCP)*, pp. 421-428, Cluj-Napoca, Romania

Sundaram, J. P. S. et. al. (2020): A Survey on LoRa Networking: Research Problems, Current Solutions, and Open Issues; *IEEE Communications Surveys & Tutorials* Vol. 22, pp. 371-388

Tomasin, S., Zulian, S., Vangelista, L. (2017): Security Analysis of LoRaWAN Join Procedure for Internet of Things Networks; *IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*, pp. 1-6, San Francisco, USA

7. Glossar

Activation By Personalization (ABP)	Betriebsmodus von Endgeräten in welchem die LoRaWAN-Verbindung statisch verschlüsselt wird. Es findet keine gegenseitige Authentifizierung mit dem Netzwerkserver statt
Advanced Encryption Standard (AES)	Blockchiffre bzw. Verschlüsselungsmethode, die in der Kryptographie weit verbreitet ist
Application	Gruppierung gleichartiger Endgeräte im Netzwerkserver des TTN
Application Programming Interface (API)	Programmierbare Benutzerschnittstelle einer Anwendung oder Software
Broker	Bezeichnung des Servers, der in einem MQTT-Netzwerk die Kommunikation verwaltet und Ressourcen zur Verfügung stellt
Bucket	Übergeordnete Speicherstruktur in InfluxDB, um Zugriffsrechte zu verwalten und die Speicherdauer von Daten festzulegen
Cardinality	Konzept für die effiziente Nutzung der Speicherstruktur von InfluxDB, bei der eine möglichst niedrige Kardinalität angestrebt wird. Das Konzept beschreibt das Verhältnis von der Anzahl separater Datentupel zum belegten Speicherplatz in der Datenbank
Client	Bezeichnung eines IoT-Geräts oder -Anwendung, das mit einem *Broker* über das MQTT-Protokoll kommuniziert
Command Line Interface (CLI)	Programmierbare Benutzerschnittstelle eines Rechners
Downsampling	Verringerung des Speicherbedarfs eines Datensatzes durch Mittelwertbildung o.ä.
Events	Kommunikations-Log einer LoRaWAN-Verbindung im Netzwerkserver des TTN, welche die empfangenen *Payloads* enthalten
Field	Element in der Datenstruktur des *Influx-Datenformats*, das der Speicherung von numerischen Werten dient
Graphical User Interface (GUI)	Graphische Benutzerschnittstelle eines Rechners, Anwendung oder Software
Hypertext Transfer Protocol (HTTP)	*Netzwerkprotokoll*, das hauptsächlich aber nicht ausschließlich für Datenübertragungen im Internet verwendet wird
Influx-Datenformat	Dateiformat und Konzept zur Strukturierung von Zeitreihendaten in InfluxDB

Influx Line Protocol	Interne Skriptsprache der Datenbanksoftware InfluxDB. Das *Influx Line Protocol* orientiert sich an SQL, ist aber an die Verwaltung von Zeitreihendaten angepasst
Internet of Things (IoT)	Bezeichnung für die kommunikative Vernetzung physikalischer Objekte mit dem Internet zu Zwecken der Digitalisierung
Join-Request- / Join-Accept-Message	Spezielle Nachrichtentypen des LoRaWAN-Protokolls, mit welchen Kommunikationsanfragen getätigt werden. Sie werden bei der gegenseitigen Authentifizierung mittels OTTA-Aktivierung ausgetauscht
Livestream	Zwischenspeicher für *Events* bzw. *Payloads* im Netzwerkserver des TTN
Long Range (LoRa)	Funkverbindung im niedrigen Frequenzbereich (Radio), die Teil einer LoRaWAN-Verbindung ist
Measurement	Element und eindeutiger Indikator eines Datentupels im *Influx-Datenformat*
Member	Bezeichnung eines Benutzeraccounts bei InfluxDB
Message Queuing Telemetry Transport	*Netzwerkprotokoll*, das speziell für IoT-Anwendungen geeignet und Teil des Internetprotokolls ist
Metric	Bezeichnung eines Datentupels im *Influx-Datenformat*
Netzwerk-Protokoll	Menge von Regeln, die Syntax, Semantik und Synchronisation bei einer Datenübertragung innerhalb eines Netzwerks definieren. Obligatorisch für die korrekte Interpretation der übertragenen Informationen zwischen Kommunikationspartnern
Organization	Gruppierung mehrerer *Member* bzw. Benutzeraccounts in InfluxDB
Over The Air Activation (OTAA)	Betriebsmodus von Endgeräten, in welchem die LoRaWAN-Verbindung dynamisch verschlüsselt wird. Die LoRaWAN-Verbindung wird mit einer gegenseitigen Authentifizierung von Endgerät und Netzwerkserver durch *Join-Request- / Join-Accept-Message* initiiert
Payload	Kleinere Datenpakete mit Nutzdaten (Daten ohne Steuer- und Protokollinformation) die im Rahmen des LoRaWAN-Protokolls ausgetauscht werden
Publishing	Spezieller Nachrichtentyp des MQTT-Protokolls, mit denen *Clients* Daten an einen *Broker* versenden
Quality of Service (QoS)	Konzept für die Übertragunssicherheit bei Datentransfer mittels verzögertem Austausch von Datenpaketen

Secure Sockets Layer (SSL)	Zusätzliche Sicherheitsschicht für Datenübertragungen im Internet und die vorherige Version der TLS-Verschlüsselung
Subscribing	Spezieller Nachrichtentyp des MQTT-Protokolls, mit dem ein *Broker* Daten an einen *Client* versendet
Tag	Element in der Datenstruktur des *Influx-Datenformats*, das der Speicherung von Metadaten dient
Timestamp	Ein Zeitstempel ist allgemein ein exakter Zeitpunkt, der durch einen numerischen Wert ausgedrückt wird. In Bezug auf InfluxDB wird ein Element eines Datentupels im *Influx-Datenformat* als *Timestamp* bezeichnet
Token	Passwort, welches zuvor definierte Zugriffsrechte in InfluxDB gewährt
Topic	Hierarchisch gegliederte Kommunikationskanäle zur Strukturierung des MQTT-Protokolls
Transport Layer Security (TLS)	Verschlüsselungsprotokoll, dass eine zusätzliche Sicherheitsschicht für Datenübertragungen im Internet darstellt
Uplink-/ Downlink-Message	Spezieller Nachrichtentyp des LoRaWAN-Protokolls, mit denen *Payloads* von Endgeräten an den Netzwerkserver (Uplink) und umgekehrt (Downlink) versendet werden